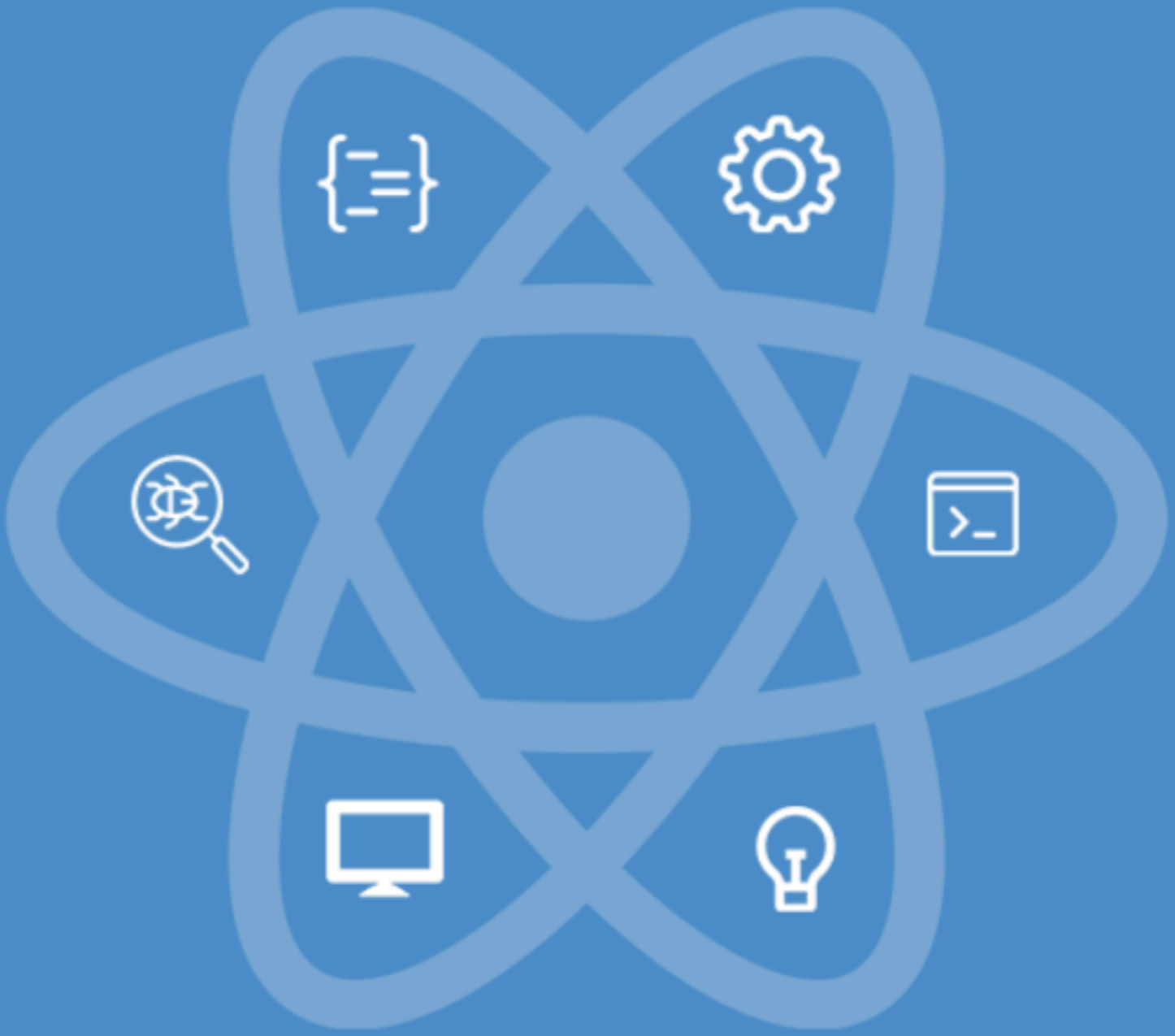
filestack

React JS
Full Tutorial



Introduction To React

An integral part of developing websites in today's digital landscape is creating user interfaces that are both efficient and creative. This is where React comes in. Whether you're an experienced developer or just getting started, React has emerged as a formidable tool for building modern, scalable online applications.

Companies like Facebook, Airbnb, and Netflix rely on React's flexibility and performance. But what makes React so unique? It's all about the components: reusable, modular pieces of code that make complex UI creation easier.

In this ebook, we'll review everything in React, from the fundamentals of components and JSX to using current technologies like Tailwind CSS to create stunning layouts. Get ready to learn the framework for transforming the web development environment.

React JS Full Course E-book Content

React is optimized to render and update only the components that require it as data changes, making it ideal for dynamic, data-driven single-page apps. Instead of reloading the entire page, React intelligently refreshes only the essential sections, resulting in a smooth and quick user experience.

This method is at the heart of how React works, giving developers a strong tool for managing frequent updates without losing performance.

When the data changes, this website renders and updates the appropriate components effectively: https://reactjs.org/

Getting Started With React

The simplest method to get started with React is through your terminal, with a program called "**npx**." This helpful tool lets you quickly create and launch a React app with minimal setup.

However, before you can utilize npx, you must first install Node.js, the foundation for controlling the React environment. Below is a general guide to installing Node.js on your PC, while the processes may differ slightly based on your operating system.

Installing Node.js on Windows:

1) Download Node.js:
    a) Visit the official Node.js website at `https://nodejs.org/`.
    b) Download the recommended version for your operating system (Windows Installer MSI).
2) Run the Installer:
    a) Run the downloaded MSI installer.
    b) Follow the prompts in the Node.js Setup Wizard. You can use the default settings.
3) Check Installation:
    a) Open a command prompt or PowerShell.
    b) Type the following commands to check if Node.js and npm (Node Package Manager) are installed:

```
node -v
```

```
npm -v
```

Installing Node.js on macOS:

1. Using Homebrew:
    a. Open a terminal.
    b. Install Homebrew if you haven't already by running:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```


```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```


Install Node.js using Homebrew:

```
brew install node
```

Check Installation:

    c. Open a terminal.
    d. Type the following commands to check if Node.js and npm are installed:

```
node -v
```

```
npm -v
```

Setting Path Variables (Environment Variables):

For Windows:

The Node.js installer typically adds Node.js to your system PATH automatically. If it doesn't, you might need to add it manually.

- Find the path where Node.js is installed (e.g., C:\Program Files\nodejs).
- Add this path to the system environment variable PATH.

For macOS/Linux:

The installation process usually adds Node.js to the system PATH.

You can verify by running:

```
echo $PATH
```

The directory containing the Node.js executable (e.g., /usr/local/bin) should be included in the output.

To persistently set PATH variables, you can add the following line to your shell profile file (e.g., ~/.bashrc or ~/.zshrc):

```
export PATH="/path/to/nodejs/bin:$PATH"
```

Replace /path/to/nodejs/bin with the actual path to the directory containing the Node.js executable.

After making these changes, close and reopen your terminal to apply the updates. Once the setup is complete, you can use Node.js and npm from the command line. Run node -v and npm -v in your terminal or command prompt to verify the installations.

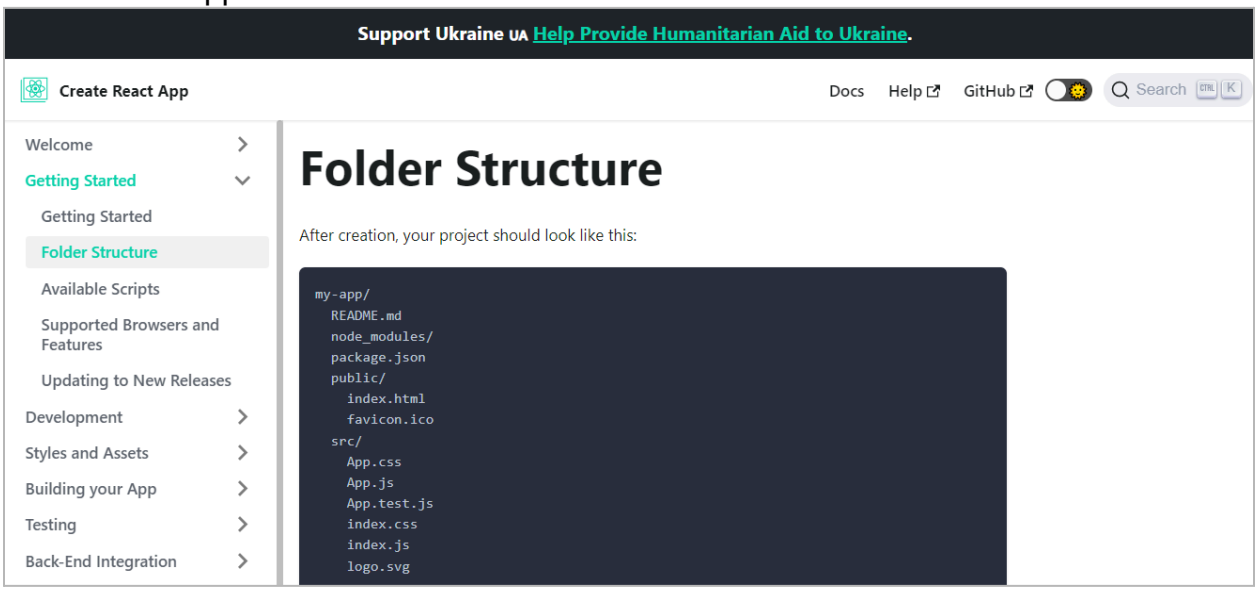
Next, create a directory for your project. You can name it anything you like; in this example, we'll use "filestackapp." Use the following command in the Command Prompt to create the directory.

```
mkdir filestackapp
```

All your files will be stored in the "filestackapp" folder. To create the React app, use the following command with npx:

```
npx create-React-app hello
```

Creating a new React app will take some time, during which a folder structure will be set up. For more details about the folder structure, refer to the documentation at create-react-app.dev.



Within the newly generated folder structure is a source directory called src. This is where your project code will reside as you progress through the next steps. You can also organize your code by creating subdirectories within src.

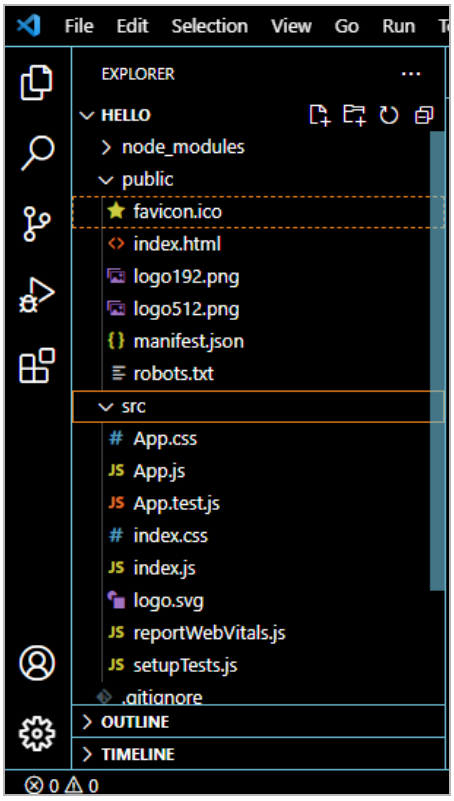
Webpack, a strong open-source module bundler, examines all files in this directory to ensure speedy rebuilds. When you return to the terminal, you will notice a number of useful commands presented. Next, navigate the hello directory and launch Visual Studio Code from the terminal.

Use the command below:

```
code hello
```

This will open a directory in the Visual Studio Code.

Now, let’s look at our project here:



The index.html file is the main webpage that will display the actual results. This page contains the metadata for your website. Inside the index.html is a highlighted line, and all the content will be placed within this line.

```
<div id="root"></div>
```

The index.html file doesn't require any changes except for modifying the metadata. The main coding happens in the index.js file in the src directory. This file is essential because it contains the JavaScript code that drives your application.

In index.html, there is a div element named "root." The index.js file's job is to find and access this div using its ID, also called "root." Its primary function is to insert the application into this "root" element.

Another important file in the src directory is app.js. This file handles the rendering of the application on the webpage. The index.html file renders the app.js file, making it crucial to maintain this structure.

This setup is important because your entire application will be built inside app.js. We'll explore these files in more detail in the sections below.

Index.js.

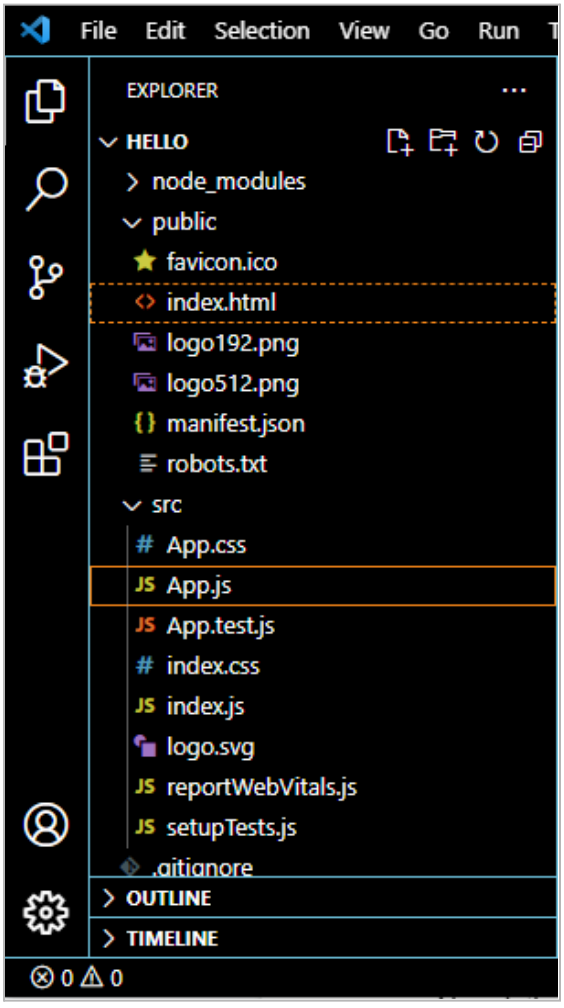
In traditional Node applications, the entry point is usually a main file. However, in React, index.js serves as the entry point, defining what to render and where to render it.

App.js

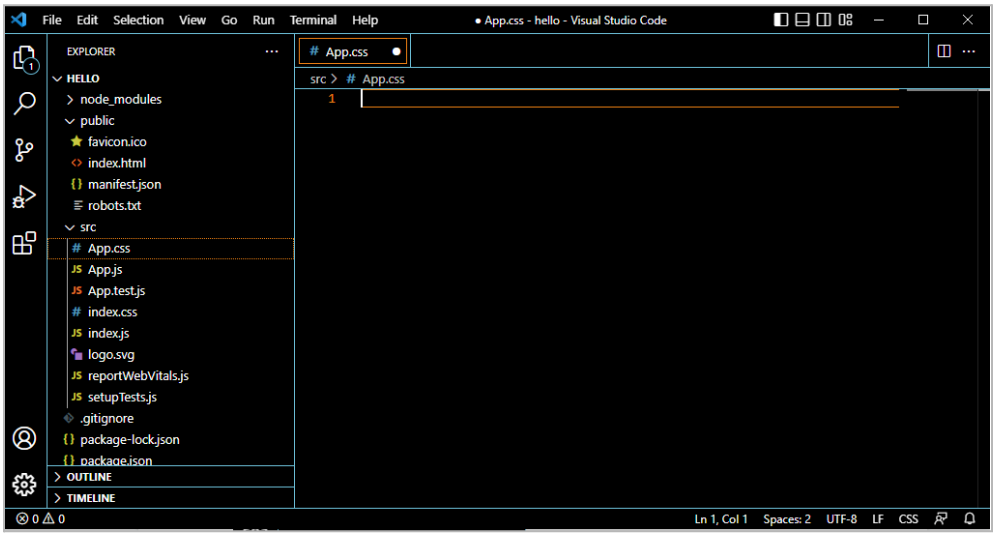
App.js contains the root component of the React application. In React, every component and its hierarchy start from this root. The <App /> component is the most crucial part at the top of the React hierarchy, which makes it feel like everything begins with App.js.

Meanwhile, index.js serves as the app's entry point and is the first file to be executed. In this file, we render the App.js component.

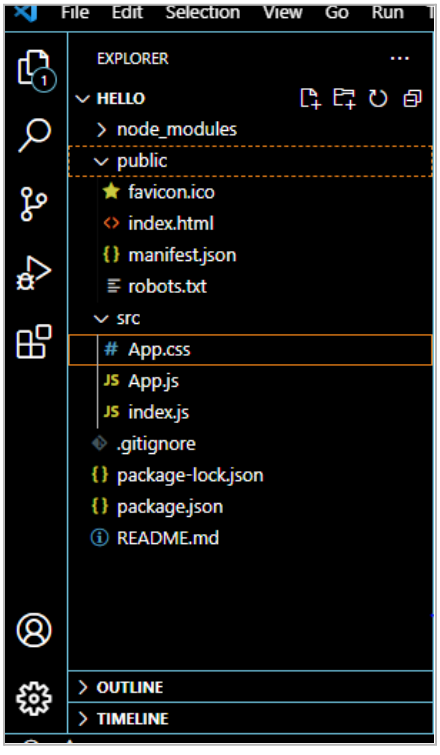
Now, let's clean up our project to establish a clear structure. **We will review the following files:**



In the public folder, start by removing the files logo192.png and logo512.png. Next, clear the code in the App.css file in the src folder.



Next, we will remove the following files: app.test.js, index.css, logo.svg, setupTests.js, and reportWebVitals.js. **After these changes, your file structure should look like this:**



Navigate to **terminal>new** terminal and then enter the following command:

```
npm start
```

Here, we will get the following errors:

```
Compiled with problems:

ERROR in ./src/App.js 4:0-30

Module not found: Error: Can't resolve './logo.svg' in 'C:\Users\adullah\filestack\hello\src'


ERROR in ./src/index.js 6:0-21

Module not found: Error: Can't resolve './index.css' in 'C:\Users\adullah\filestack\hello\src'


ERROR in ./src/index.js 8:0-48

Module not found: Error: Can't resolve './reportWebVitals' in 'C:\Users\adullah\filestack\hello\src'
```

These errors occur because the files we deleted are still being referenced in the code. Review your code, locate the lines causing errors, and remove or update them to fix this. The problematic file paths will be highlighted in red. For instance, the first error might look like this:

```
ERROR in ./src/App.js 4:0-30
```

Next, open app.js and locate the line that references the deleted file. **Remove the first line as shown below:**

```
import logo from './logo.svg';
```

After that, remove the other two lines in app.js that are causing errors. Once you've made these changes, return to the terminal to check the results.

```
Compiled with problems:

ERROR in ./src/App.js 4:0-30

Module not found: Error: Can't resolve './logo.svg' in 'C:\Users\adullah\filestack\hello\src'


ERROR in ./src/index.js 6:0-21

Module not found: Error: Can't resolve './index.css' in 'C:\Users\adullah\filestack\hello\src'
```

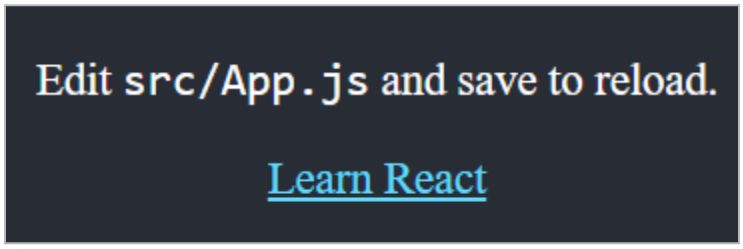
You can also see these errors displayed just below the code. **In App.js, remove the following line to resolve the issue:**

```
<img src={logo} className= "App-logo" alt="logo"/>
```

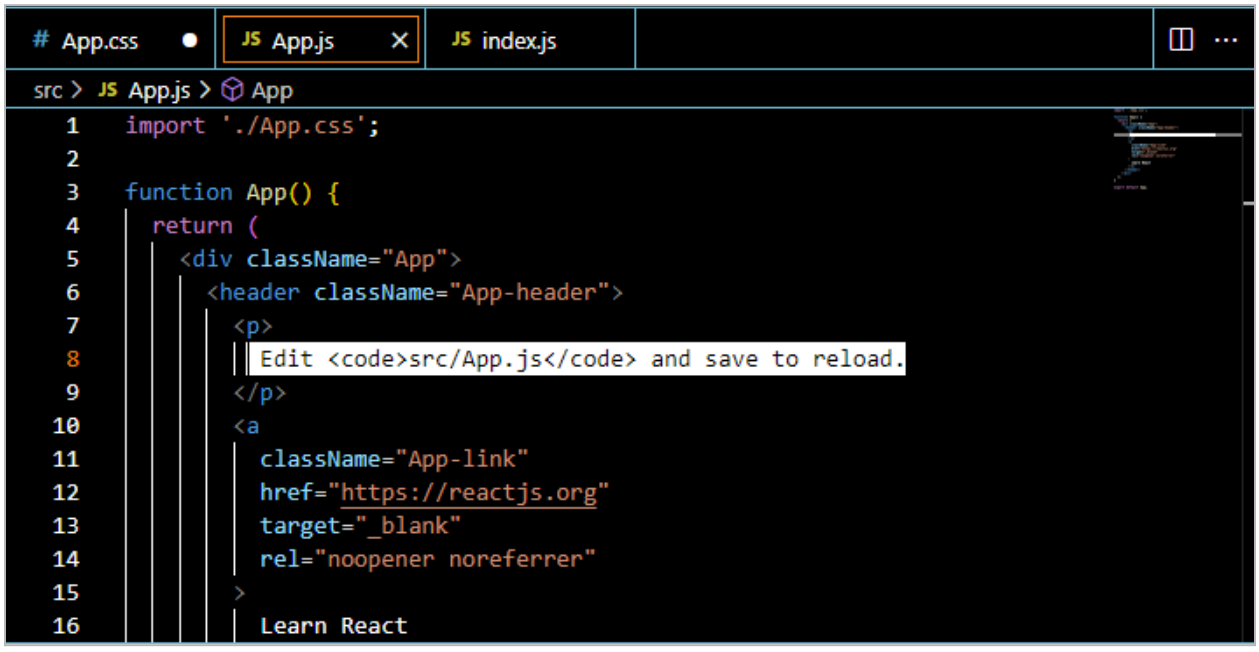
Finally, remove the following line from index.js to complete the cleanup:.

```
import './index.css';
```

After removing the specified lines of code from those files, you should see the errors resolved in your terminal or console.



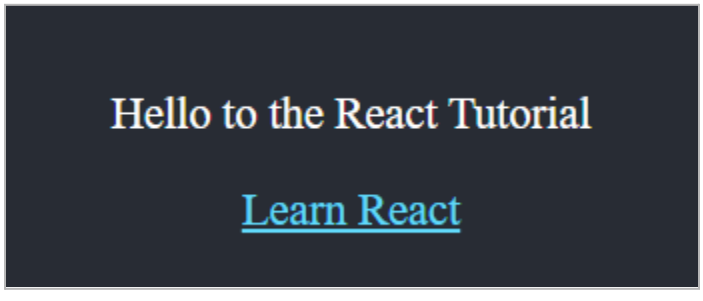
The next step is to open App.js and edit the following line of code:



Suppose we write anything in this line.

```
"Hello to the React Tutorial"
```

We will get the results on the server below:

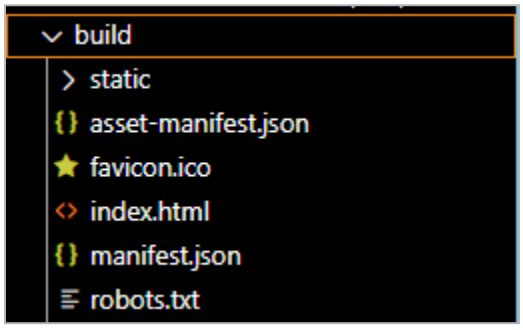


Create React App Files

It's important to note that the application is developed locally on localhost using port 3000. To deploy the application on the internet, you need to execute a specific command, **as outlined below:**

```
npm run build.
```

As a result, a new build directory will be created containing the following files:

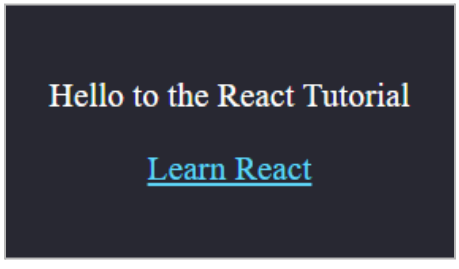


It's important to note that there are two ways to view the server results:

```
npm run build
npm start
```

The npm build command is used to create a production-ready build of your application, which optimizes and bundles the code for deployment. On the other hand, npm start is used to start the development server and run your application locally, allowing you to test and debug your code during development.

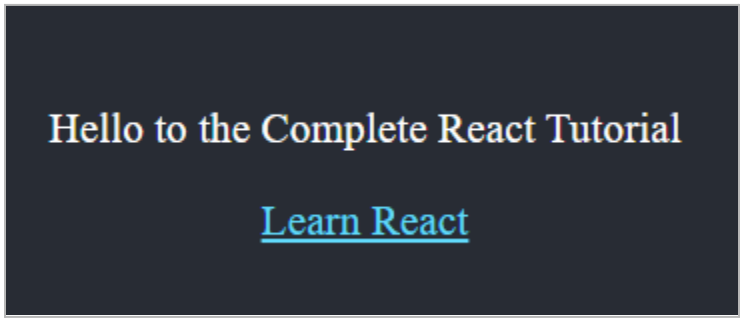
Once you run either of these commands in the terminal, the application results will appear in your browser. After pressing Enter, it will automatically open in a new tab.



Now, if we make any changes to the code, the updates will instantly appear on the screen. You can see the modified line highlighted on the screen.

```
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <p>
          Hello to the Complete React Tutorial
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
```

After writing the code, save it. **The results will be compiled below:**



One of the best features of npm start is that it shows changes immediately as you update the code.

The package.json file in the source code lists all the dependencies required for the application. Our application relies on externally authored code; importing these packages is essential. To install the necessary packages, execute the following command:

```
npm install.
```

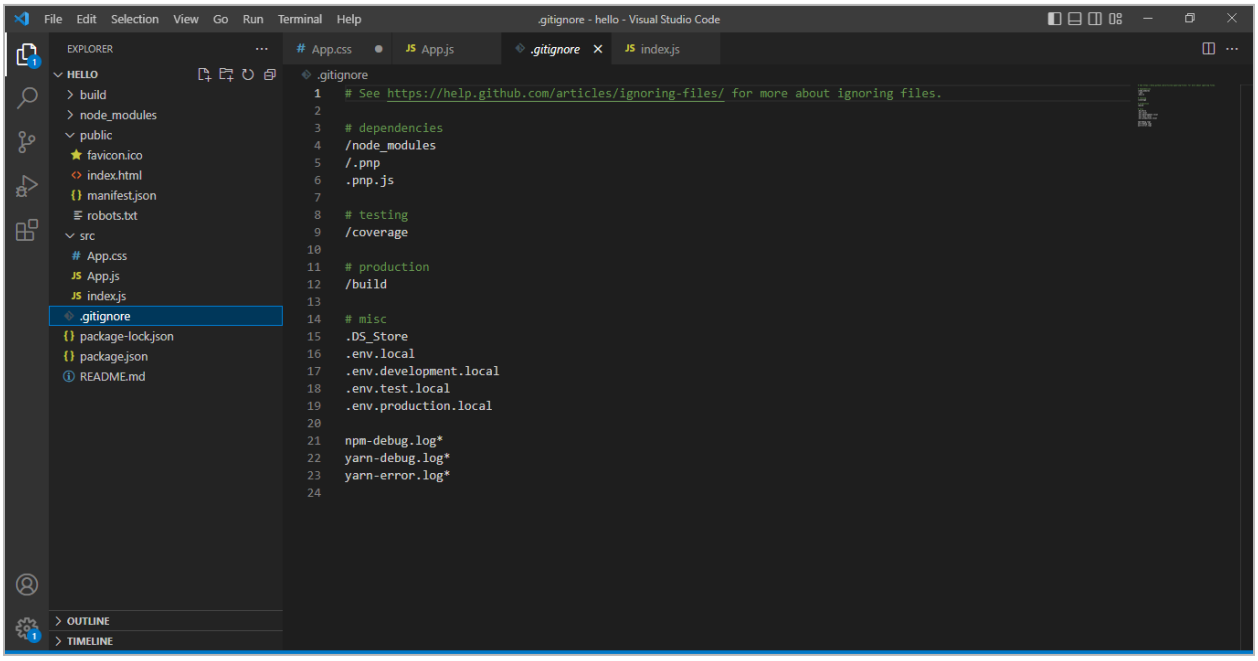
It's important to note that the need for packages arises mainly when building something that depends on externally developed libraries.

After installing the dependencies, check the package-lock.json file. This file specifies the exact versions of the packages used in your React application. Essentially, it provides guidelines for maintaining the application's consistency. If the package-lock.json file is accidentally deleted, you can regenerate it by running the npm install command.

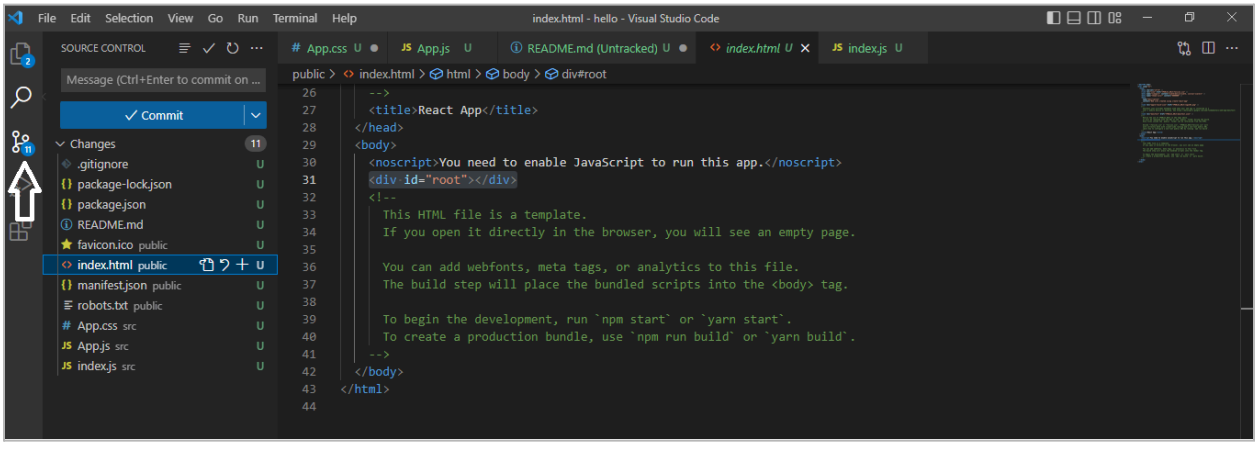
Similarly, if the node_modules folder is deleted by mistake, you can regenerate it by running the same command.

Running the npm install command reinstalls dependencies and regenerates missing files like node_modules. There's also a file called .gitignore, which defines which files and directories should be excluded when building our project.

The .gitignore file keeps certain files and directories out of version control, preventing unnecessary or sensitive files from being tracked in a Git repository. This helps streamline collaboration, reduces the size of the repository, and keeps the focus on the critical source code during development.



The Source Control tab provides details about the changes made in your project. **You can access it here:**



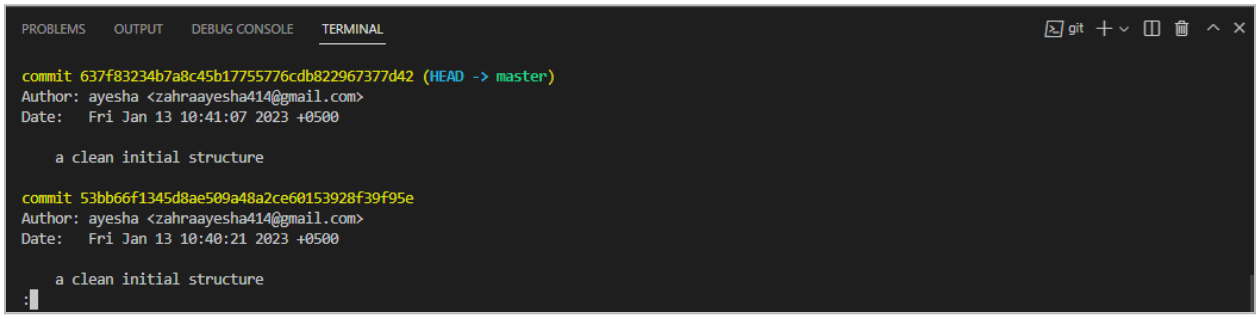
You can always discard your changes if they are being tracked in the project. You may notice a capital "U" in front of each file, indicating that no changes have been tracked since the beginning. This happens because Git wasn't installed on your computer.

You should install Git to maintain a proper commit history when working on a project. To create a clean initial structure, write a commit message in the text box above the Commit button, then click the commit symbol.

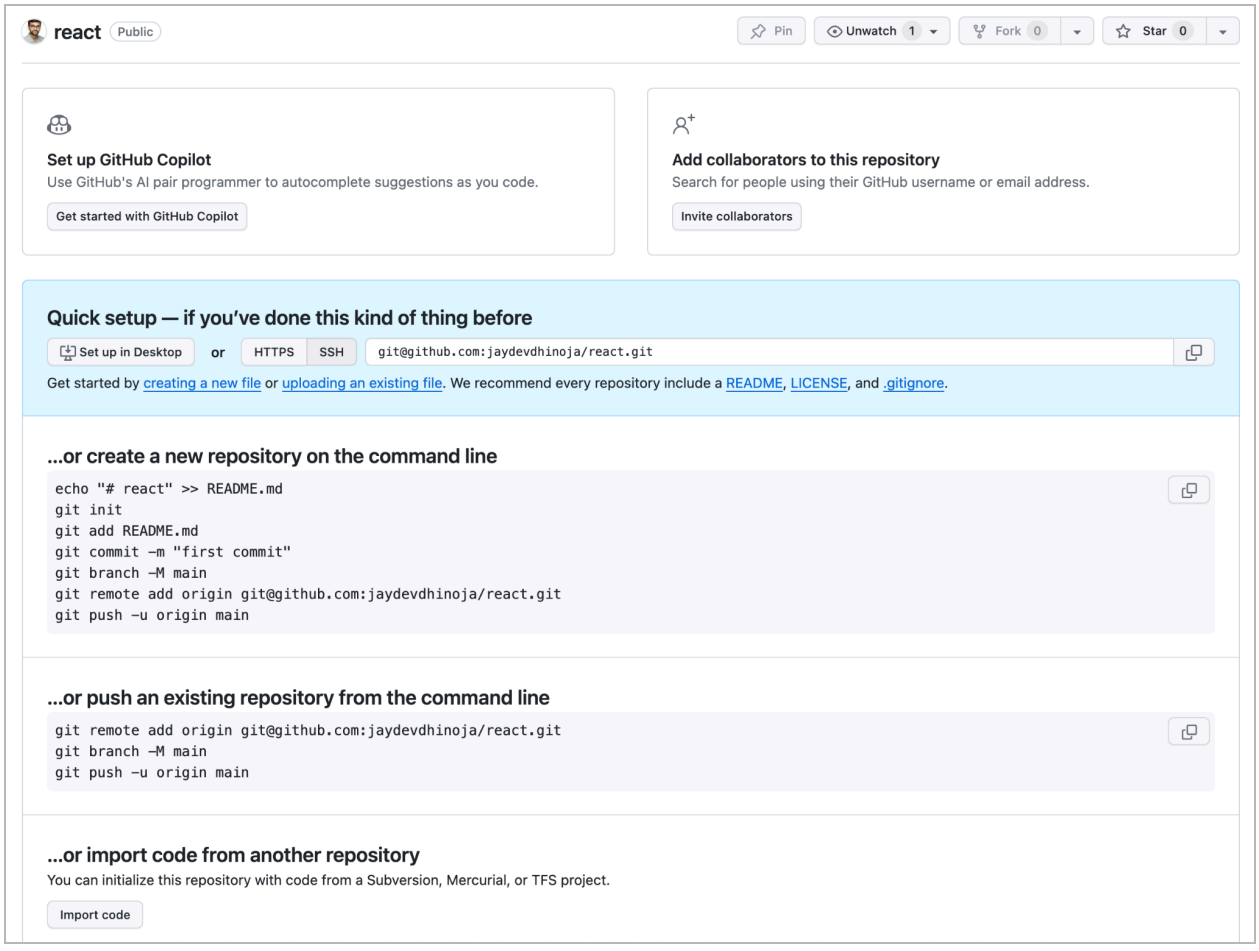
Ensure that your Git username and email are already configured. If they aren't, you must run the configuration commands in the Command Prompt.

```
Git config --global user.name "your username here."
git config --global user.email your email here."
```

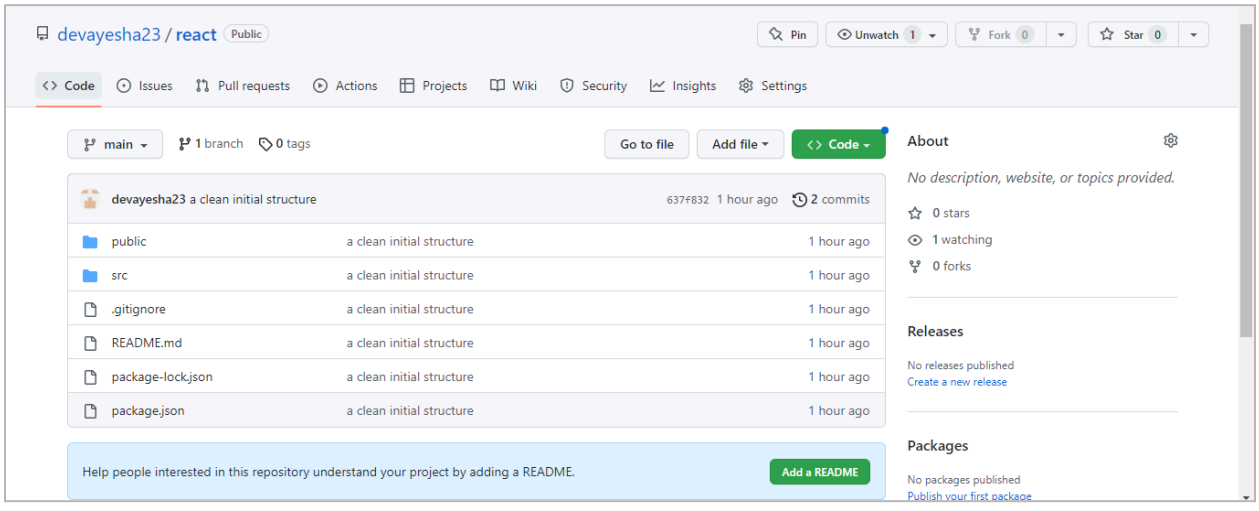
After setting up "a clean initial structure," the untracked changes in Source Control will be resolved. Next, enter git log in your terminal to view your Git history, and it will display the commit labeled "a clean initial structure."



The next step is to create a GitHub repository named "React." After creating the repository, copy the code lines provided under the heading "push an existing repository from the command line."



We pasted those code lines into a new terminal in Visual Studio Code. After executing the commands, we refreshed our repository on GitHub. **The results are displayed below:**



Make sure to run the command npm start in the new terminal before pasting the copied lines of code.

1. Our First Component

React is structured entirely with components, which are reusable sections of code. These components can be easily added to your application, similar to how you would use reusable functions. To make the process simpler and more understandable, we will create our first component.

Start by creating a folder named components inside your src directory. Then, create a new file within this folder named Employee.js. Next, open both App.js and Employee.js simultaneously. You will need to write the following code in Employee.js:

```
function Employee() {
    return <h3> here is an employee</h3>
}

export default Employee;
```

The next step is to open the App.js file. First, we need to import our component. **Use the following line of code to import the Employee component:**

```
import Employee from './components/Employee'
```

Next, we will add our component in the App.js file where we previously had the line “Hello to the React tutorial.” Replace that line with <Employee /> instead of <Employee></Employee>. The concise syntax <Employee /> is preferred over <Employee></Employee> to enhance readability and simplify the code.

This approach can be used wherever the <Employee /> component is utilized. **After running the code, you will see the results in your browser as shown below:**

We have successfully developed our first component in React.

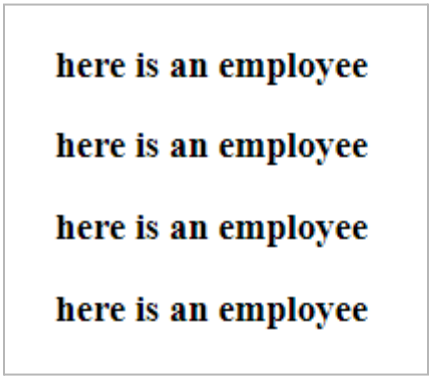
Now, let’s explore an example of conditional rendering inside our component. In the App.js file, the return statement typically returns a single element.

We can add a variable to this; different outputs will be displayed based on its value. If the variable's logic is true (in this example, it's hardcoded to true), it will show one set of results; if false, it will display something else.

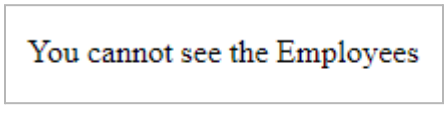
A variable is a named storage location in computer memory that holds a value, allowing data to be manipulated and referenced in a program. **For instance, let's set the variable const showEmployees to true as shown below:**

```
import './App.css';
import Employee from './components/Employee'

function App() {
  console.log('we are about to list the employees');
  const showEmployees = true;
  return (
    <div className="App">
      {showEmployees ? (
        <>
        <Employee />
        <Employee />
        <Employee />
        <Employee />
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
    </div>
  );
}

export default App;
```

It will give us the following output:



When we set **const** showEmployees to false, we get the following output.



You can also implement console logs inside this component to track its behavior. **The complete code is shown below:**

```
function App() {
  console.log('we are about to list the employees');
  const showEmployees = false;
  return (
    <div className="App">
      {console.log('inside the return')}

      {showEmployees ? (
        <>
        <Employee />
        <Employee />
        <Employee />
        <Employee />
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
    </div>
  );
}

export default App;
```

React components encapsulate specific functionalities within your application, allowing them to perform particular actions. These components are self-contained and can be used throughout your entire application. In essence, React applications are constructed by nesting components within each other.

2. Props in React Component

Firstly, it's important to understand that the Employee component is a template for other components. We will use this template as a guide moving forward. However, it's crucial to note what each component specifically does.

```
function Employee() {
    return <h3> here is an employee</h3>
}

export default Employee;
```

Now, let's look at an example of how to use props. **We will implement props in our**

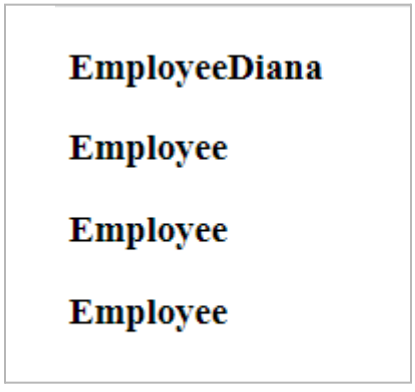
component template as shown below:

```
function Employee(props) {
    return <h3>Employee{props.name}</h3>
}

export default Employee;
```

Next, we'll move to the App.js file and modify its structure. **Specifically, we'll include the Employee component with a name attribute like this:**

```
<Employee
name="Diana" />
```

This works similarly to a function with parameters but uses React props, properties passed to components to enable dynamic content display. **After saving your changes, the results will be compiled and displayed below:**



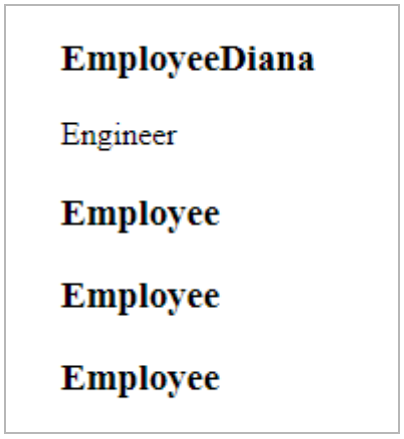
Props are simple to use in our React components when done this way. **To edit the component template and return different values using props, like the employee's position, we may add the following code to it:**

```
function Employee(props) {
    return (
        <>
            <h3>Employee{props.name}</h3>
             <p>{props.role}</p>
         </>
    );
}

export default Employee;
```

Then, we will make the following changes in App.js file:

```
<Employee name="Diana" role=
"Engineer"/>
```

The final output will be shown below:



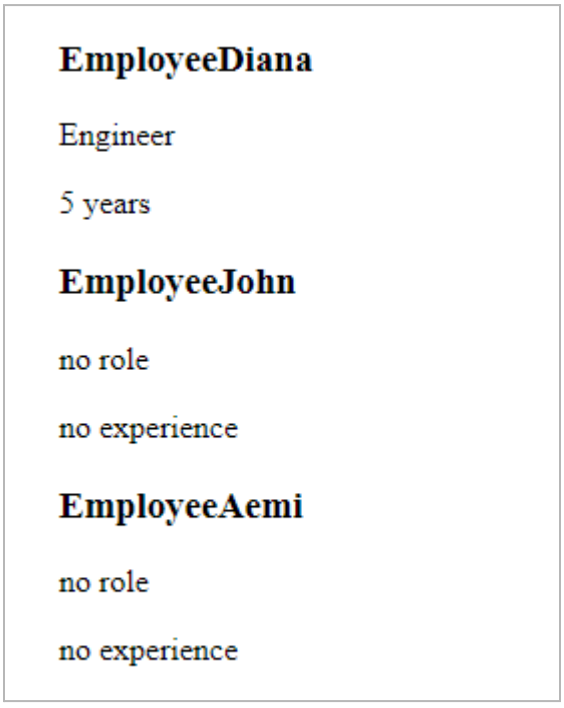
Similarly, we can add as many properties as we need to our React components using props. Let's modify other employee entries and use props to check if they have a role assigned. **The changes to the component template are as follows:**

```
function Employee(props) {
   return (
      <>
         <h3>Employee{props.name}</h3>
          <p>{props.role ? props.role : "no role"}</p>
          <p>{props.experience ? props.experience:"no experience"}</p>
       </>
   );
}

export default Employee;
```

The changes in App.js file are:

```
<Employee name= "Diana" role= "Engineer" experience= "5 years"/>
<Employee name= "John"/>
<Employee name= "Aemi"/>
```

The results we get from these changes are:



Now let's explore React component syntax. Learning how to write components might be hard for beginners, especially because curly brackets and quotations are not always used consistently. **Here's another syntax that we can use to include properties in our components:**

```
{props.role ? <p>{props.role}</p> : <p>No role</p>}
```

The results will be displayed similarly to the previous syntax.

3. Introduction to useState Hook

The state is the second most important React notion after props. It's crucial to realize that prop values can only be altered in the parent component; they are not intended to be altered within the child component.

If you try to modify the props inside a child component, an error will occur. On the other side, the State allows us to modify values directly within a component, notably the parent.

A React application's state is intimately related to its user interface and aids in tracking values. The user interface immediately adjusts without requiring a page refresh when the status changes.

Let's say we wish to use our code to add a value. The following code will be entered when App.js is opened:

```
import './App.css';
import Employee from './components/Employee'

function App() {
  const showEmployees = true;
  return (
    <div className="App">

      {showEmployees ? (
        <>
        <input type= 'text' onChange={() =>{ // onChange event gets
triggered every time the input field is edited
          console.log('e.target.value') // Logs the input value in the browser
console
        }}/>

        <Employee name= "Diana" role= "Engineer" experience= "5 years"/>
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
```

```
    </div>
  );
}

export default App;
```

This code allows us to input a value in the form of text. **The output is displayed as shown below:**

Now, let's modify the code structure to introduce a new value in our output.

```
import './App.css';
import Employee from './components/Employee'
```

```
function App() {
  let role='GIS Analyst';
  const showEmployees = true;
  return (
    <div className="App">
      {console.log('inside the return')}

      {showEmployees ? (
        <>
        <input type= 'text' onChange={(e) =>{
          console.log(e.target.value);
          role = e.target.value;
        }}/>

        <Employee name= "Diana" role= "Engineer" experience= "5 years"/>
        <Employee name= "John" role= {role}/>
        <Employee name= "Aemi"/>
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
    </div>
  );
}

export default App;
```

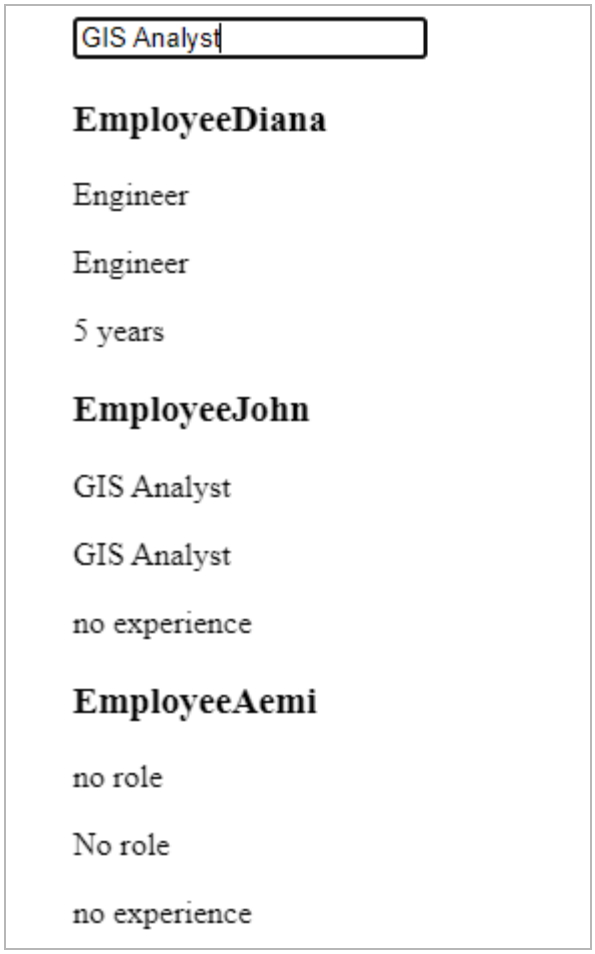
It will give us the following output:

But, we want to update the value “GIS Analyst” when typing into the text bar showing at the top in output. Therefore, we need to import useState which will help us to update the value. **Here is our final code structure to update the value:**

```
import './App.css';
import Employee from './components/Employee'
import { useState } from 'React'

function App() {
  const [role, setRole] = useState('GIS Analyst');
  const showEmployees = true;
  return (
    <div className="App">
      {console.log('inside the return')}
```

```
    {showEmployees ? (
      <>
      <input type= 'text' onChange={(e) =>{
        console.log(e.target.value);
        setRole(e.target.value);
      }}/>

      <Employee name= "Diana" role= "Engineer" experience= "5 years"/>
      <Employee name= "John" role= {role}/>
      <Employee name= "Aemi"/>
      </>
     ) : (
      <p>You cannot see the Employees</p>
     )}
   </div>
  );
}

export default App;
```

The output is shown bellow:



Never forget that setting a value for a variable must never be done directly. Rather, we must use a set state function to assign a value. To ensure correct state management in a React application, we used the variable "role" and the "setRole" function in the previous code snippet to assign and update its value.

Ignoring a setter function may make it more difficult to update the state dynamically, impacting how the website renders because a responsive user interface depends on precise and timely adjustment of such state variables.

Take note that one example of a hook is an estate. Several hooks are included with React to add functionality to our code. React hooks all begin with "use," such as useState. With hooks, we can utilize several React functionalities without creating classes.

4. Installation of Tailwind CSS for React

CSS allows us to design our web applications and insert beautiful colors and styles into them. Learning Tailwind is much similar to learning plain CSS. Tailwind allows us installation with Create React App. Continuing our above project, we have to install Tailwind using the following code:

```
npm install -D tailwindcss
```

```
npx tailwindcss init
```

You can also copy these codes from the Tailwind website where all the guidance is already given. The next step is to copy the code given below and paste it into the tailwind.config.js file.

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

We have to save the above code and then add the following code into App.css file.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

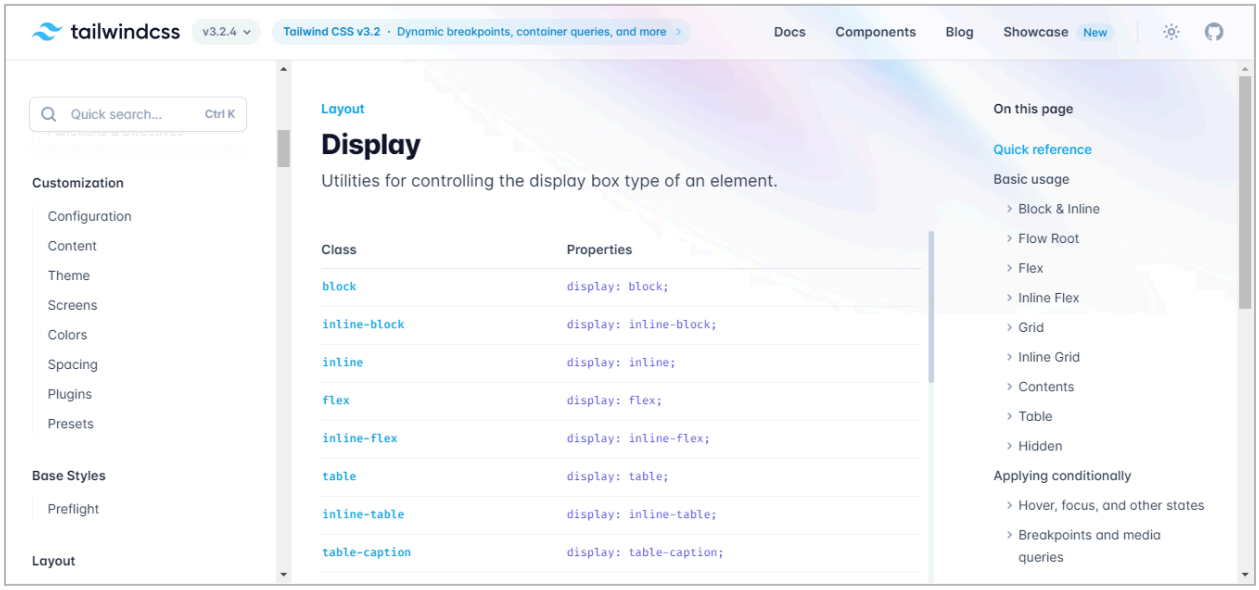
The next step is to App.js and implement the following changes to our code:

```
function App() {
  const [role, setRole] = useState('GIS Analyst');
  const showEmployees = true;
```

```
  return (
    <div className="App bg-red-300">
      {console.log('inside the return')}

      {showEmployees ? (
        <>
        <input type= 'text' onChange={(e) =>{
          console.log(e.target.value);
          setRole(e.target.value);
        }}/>

        <Employee name= "Diana" role= "Engineer" experience= "5 years"/>
        <Employee name= "John" role= {role}/>
        <Employee name= "Aemi"/>
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
    </div>
  );
}

export default App;
```

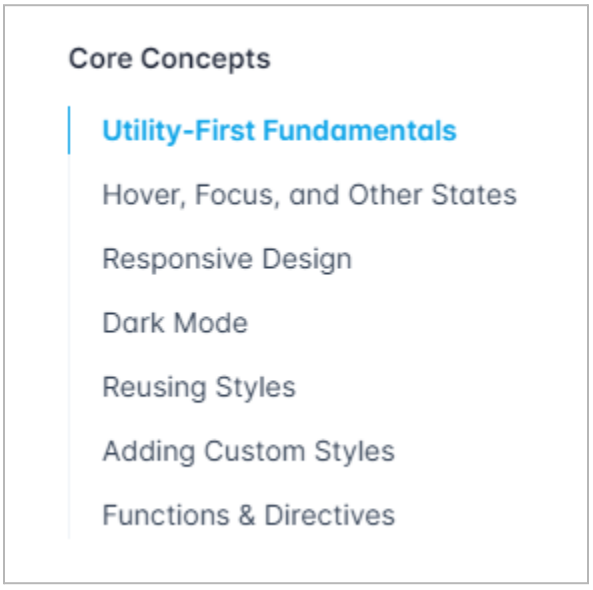
Then, we open the terminal and run the command npm run start. **The output will be shown as bellow:**

For our ease, we will rename the file App.css to index.css. In the above example, we added red background colour into our code as bg-red-300. However, there are more colours as well as styling options available on the Tailwind website. **One example is shown below:**



5. Styling React with Tailwind CSS Classes

We will style our React application with the help of Tailwind CSS classes. Moreover, we have to try some examples on the Tailwind website. For this purpose, we have to visit the Tailwind website and check the documentation. We have to click on Utility-First Fundamentals under the heading Core Concepts.



Here, we will find different examples. **We copied the code given below:**

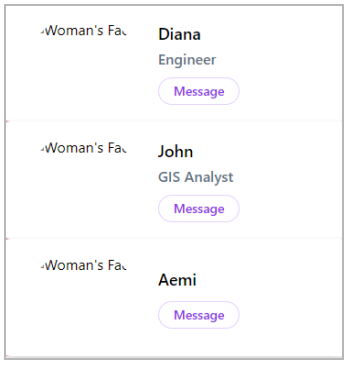
```
<div className="py-8 px-8 max-w-sm mx-auto bg-white rounded-xl shadow-lg space-y-2 sm:py-4 sm:flex sm:items-center sm:space-y-0 sm:space-x-6">

 <img className="block mx-auto h-24 rounded-full sm:mx-0 sm:shrink-0" src="/img/erin-lindford.jpg" alt="Woman's Face">

 <div className="text-center space-y-2 sm:text-left">

   <div className="space-y-0.5">

     <p className="text-lg text-black font-semibold">
       Erin Lindford

     </p>

     <p className="text-slate-500 font-medium">
       Product Engineer

     </p>

   </div>

   <button className="px-4 py-1 text-sm text-purple-600 font-semibold rounded-full border border-purple-200 hover:text-white
```

```
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">Message</button>
  </div>
</div>
```

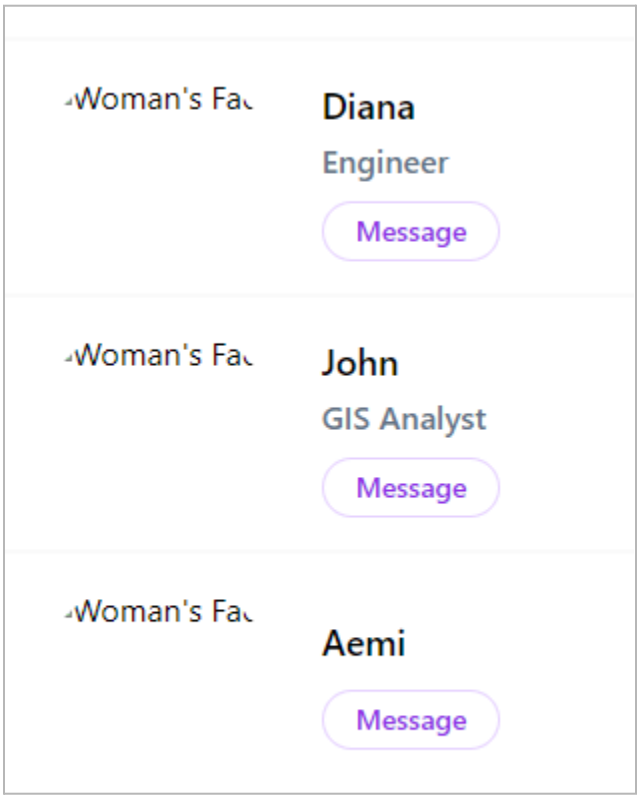
The next step is to add this code into Employee.js file after making some adjustments. **We can see it below:**

```
function Employee(props) {
    return (
        <div className="py-8 px-8 max-w-sm mx-auto bg-white rounded-xl
shadow-lg space-y-2 sm:py-4 sm:flex sm:items-center sm:space-y-0
sm:space-x-6">
  <img className="block mx-auto h-24 rounded-full sm:mx-0
sm:shrink-0" src="/img/erin-lindford.jpg" alt="Woman's Face"/>
  <div className="text-center space-y-2 sm:text-left">
    <div className="space-y-0.5">
      <p className="text-lg text-black font-semibold">{props.name}</p>
      <p className="text-slate-500 font-medium">{props.role}
      </p>
    </div>
    <button className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">Message</button>
  </div>
</div>
    );
}

export default Employee;
```

After we write the code, save it and then run the code. **We will get the following output.**



We must remove the red background colour from the App.js file to make it look much better. **It will show us something like this:**



Let's take a look at the parameters that make up these icons. **Here we got the line of code from the Employee.js file.**

```
<div className="py-8 px-8 max-w-sm mx-auto bg-white rounded-xl shadow-lg space-y-2 sm:py-4 sm:flex sm:items-center sm:space-y-0 sm:space-x-6">
```

The padding is displayed at the x and y axes by Py-8 and PX-8. Using the Tailwind website's padding settings, we can also modify their values.

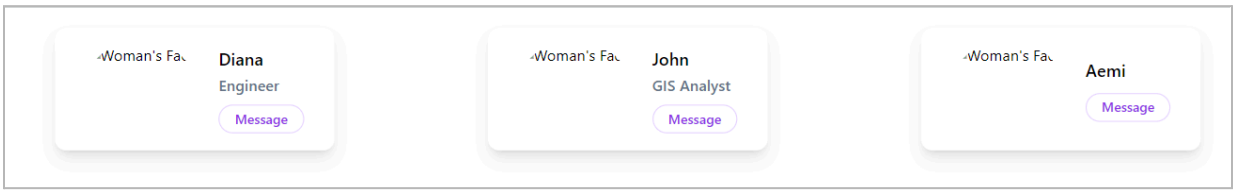
Next, we have the background color, margin, max-width, and edges styled as rounded-x1. All settings are available on the Tailwind website for modification based on your requirements.

Now let’s add a few more changes to our code in App.js. We have the below code with the new addition as “flex.”

```
import './index.css';
import Employee from './components/Employee'
import { useState } from 'react'

function App() {
  const [role, setRole] = useState('GIS Analyst');
  const showEmployees = true;
  return (
    <div className="App">
      {console.log('inside the return')}

      {showEmployees ? (
        <>
        <input type= 'text' onChange={(e) =>{
          console.log(e.target.value);
          setRole(e.target.value);
        }}/>
        <div className= "flex flex-wrap">
        <Employee name= "Diana" role= "Engineer" experience= "5 years"/>
        <Employee name= "John" role= {role}/>
        <Employee name= "Aemi"/>
        </div>
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
```

```
    </div>
  );
}

export default App;
```

It will give us the following output:



The Tailwind website offers a wide range of flex and flex-wrap alternatives as well. Let's now attempt to add an image to every employee. To our code, we will append a property image.

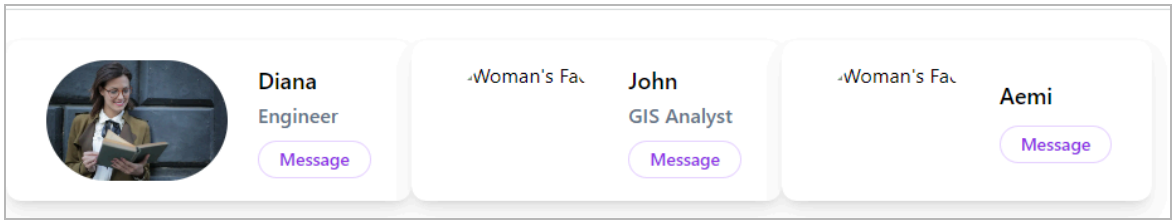
Additionally, we need to provide it with our image's URL. Assume for the moment that we obtained the URL of an image from pexel.com and added it to our img property as follows:

```
<div className= "flex flex-wrap">
        <Employee name= "Diana" role= "Engineer" experience= "5 years"
img='https://images.pexels.com/photos/3772711/pexels-photo-3772711.jp
eg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1'/>
        <Employee name= "John" role= {role}/>
        <Employee name= "Aemi"/>
        </div>
```

Then, we will make the changes to src in our component's code.

```
src={props.img}
```

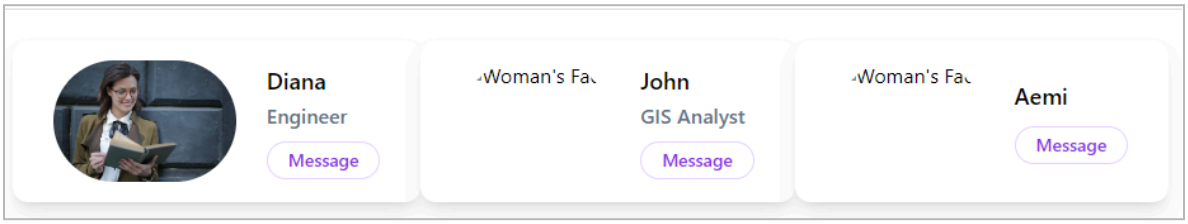
The output is shown in the image given below:



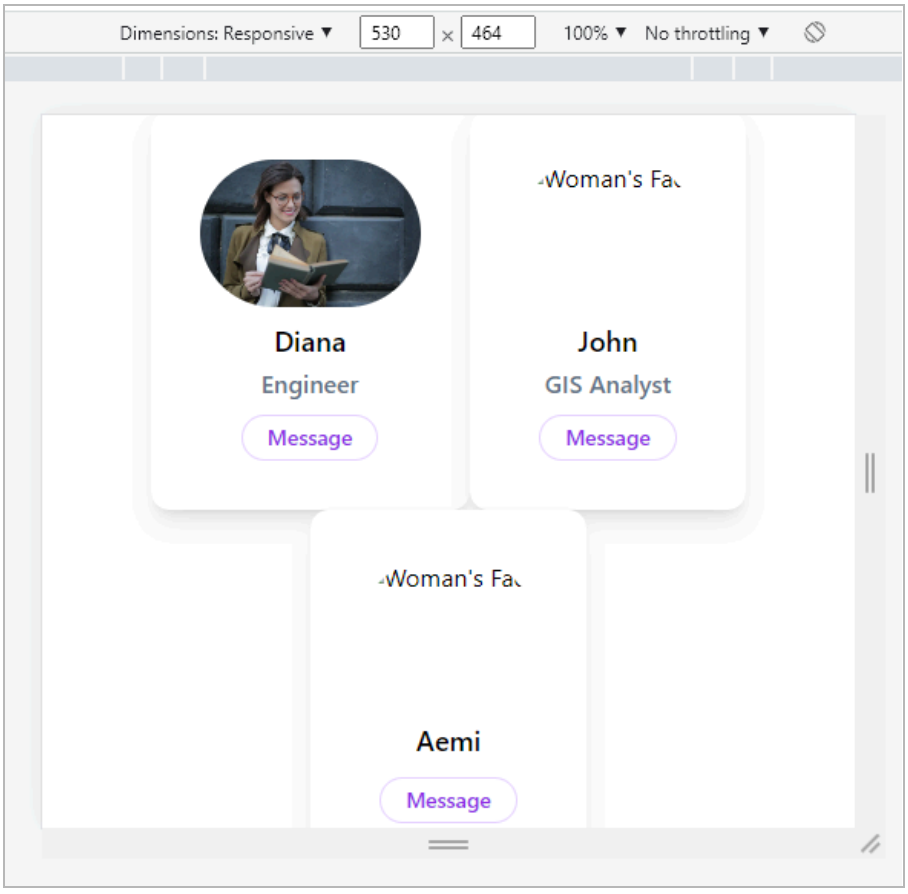
Since we applied an image only to one employee in our code, the other employees won't display any images in the output. **We added justify-centre into our App.js file, as you can see below:**

```
<div className= "flex flex-wrap justify-center">
```

It adjusts our icons into the center. **The output we get is given below:**



We must go to the console log to see how our program appears on a mobile device. The top bar will contain a mobile icon. Clicking it takes us to our application's mobile view. **In the preceding example, the mobile view is:**



We can quickly scroll up and down through the personnel listed above.

Mapping Through State Array in React

In this section, we'll learn how to work with numerous data bits within a single state variable.

Simply put, we'll learn how to operate with arrays in React. Currently, we have employee components with slightly different data. Our goal is to display one of these components as a loop.

Our initial step is to transfer all data to the single variable on top. Step two involves replacing the presentation of those components with a loop based on the map.

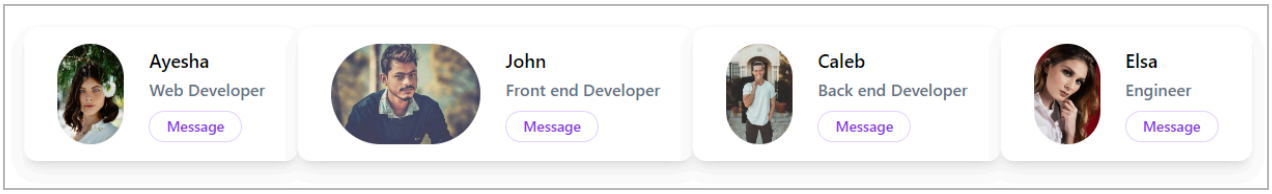
You can notice the addition of a loop into our code from the App.js file as follows:

```
import './index.css';
import Employee from './components/Employee'
import { useState } from 'react'
```

```
function App() {
  const [role, setRole] = useState('GIS Analyst');
  const [employees, setEmployees] = useState(
    [
      {name: 'Ayesha',
      role: 'Web Developer',
      img:
'https://images.pexels.com/photos/3586798/pexels-photo-3586798.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    {name: 'John',
      role: 'Front end Developer',
      img:
'https://images.pexels.com/photos/694438/pexels-photo-694438.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    {name: 'Caleb',
      role: 'Back end Developer',
      img:
'https://images.pexels.com/photos/775358/pexels-photo-775358.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    {name: 'Elsa',
      role: 'Engineer',
      img:
'https://images.pexels.com/photos/3610877/pexels-photo-3610877.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    ]);
  const showEmployees = true;
  return (
    <div className="App">
      {console.log('inside the return')}
```

```
      {showEmployees ? (
        <>
        <input type= 'text' onChange={(e) =>{
          console.log(e.target.value);
          setRole(e.target.value);
        }}/>
        <div className= "flex flex-wrap justify-center">
       {employees.map((employee) => {
        console.log(employee);
        return (
          <Employee
          name= {employee.name}
          role= {employee.role}
          img= {employee.img}
          />
        );
       })}
        </div>
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
    </div>
  );
}

export default App;
```

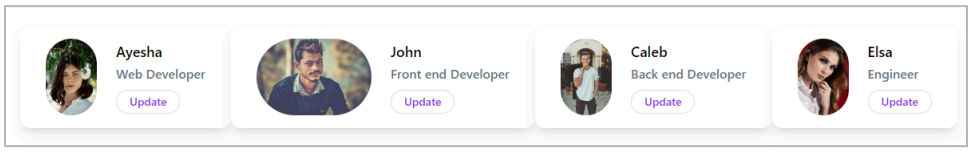
We have made other adjustments, which you can see in the employees.map above.
Our tweaks to the code resulted in the following:



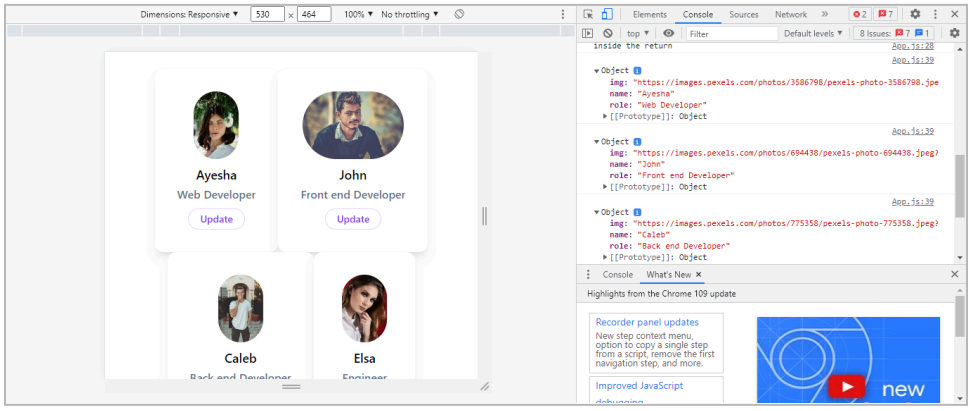
The nicest thing is that we can scale it up to hundreds of thousands of employees without adding new code. **The above graphic shows the ability to send a message. It can also be modified from the Employee.js file as follows:**

```
<button className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">
        Update</button>
```

The output is as follows:



You must be aware that React maintains track of all data on the page. We may check it in the developer console of our website. **Our project's tracking details are as follows:**



When we open our developer console now, we get a warning that each child in the parent component should have a unique ID. We can manually issue them IDs, but this is time-consuming. **To assign a unique identifier to each element, we will need to run the command below:**

```
npm install uuid
```

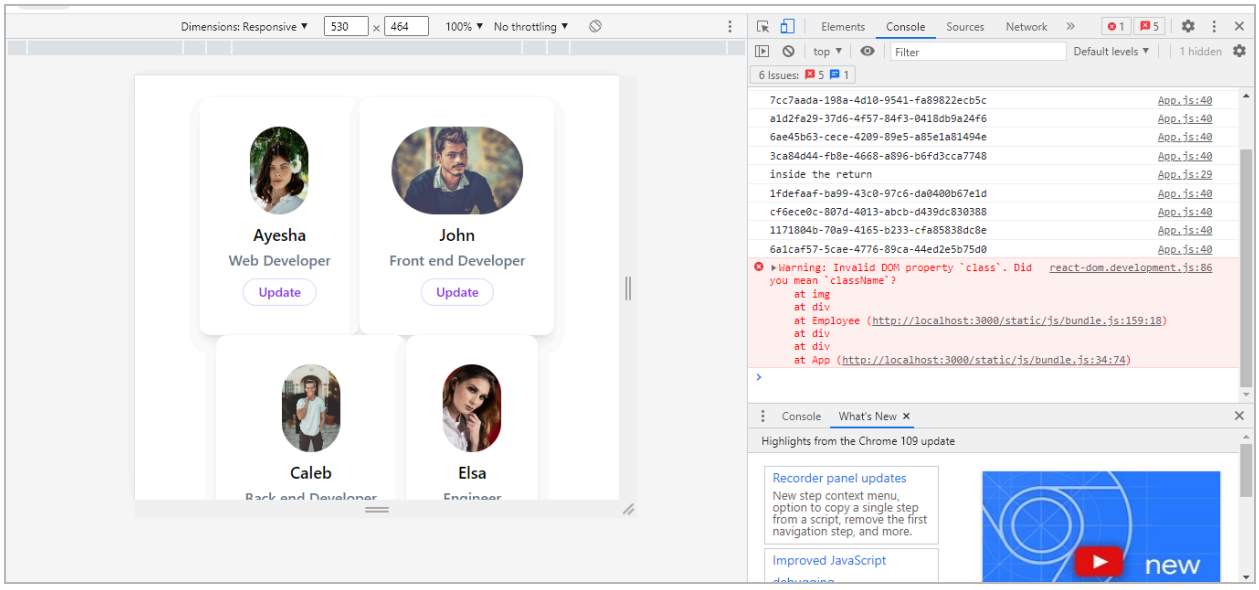
It will install a unique identifier package. **Then, we have to make a few more changes in our code, as seen below:**

```
import './index.css';
import Employee from './components/Employee';
import { useState } from 'react';
import {v4 as uuidv4} from 'uuid';

function App() {
  const [role, setRole] = useState('GIS Analyst');
  const [employees, setEmployees] = useState(
    [
      {name: 'Ayesha',
      role: 'Web Developer',
      img:
'https://images.pexels.com/photos/3586798/pexels-photo-3586798.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    {name: 'John',
      role: 'Front end Developer',
      img:
'https://images.pexels.com/photos/694438/pexels-photo-694438.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    {name: 'Caleb',
      role: 'Back end Developer',
      img:
'https://images.pexels.com/photos/775358/pexels-photo-775358.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    {name: 'Elsa',
```

```
      role: 'Engineer',
      img:
'https://images.pexels.com/photos/3610877/pexels-photo-3610877.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    ]);
  const showEmployees = true;
  return (
    <div className="App">
      {console.log('inside the return')}

      {showEmployees ? (
        <>
        <input type= 'text' onChange={(e) =>{
          console.log(e.target.value);
          setRole(e.target.value);
        }}/>
        <div className= "flex flex-wrap justify-center">
       {employees.map((employee) => {
        console.log(uuidv4());
        return (
          <Employee
          key= {uuidv4()}
          name= {employee.name}
          role= {employee.role}
          img= {employee.img}
          />
        );
       })}
        </div>
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
```

```
    </div>
  );
}

export default App;
```

Here, we no longer get the warning that each element has a unique key identifier.



Create a Popup Modal with React Bootstrap

In this section. We will learn how to make a modal popup window. This modal window appears at the top of our webpage. We can either interact with or close the window. **Let's get to the React Bootstrap.**

https://react-bootstrap.github.io/getting-started/

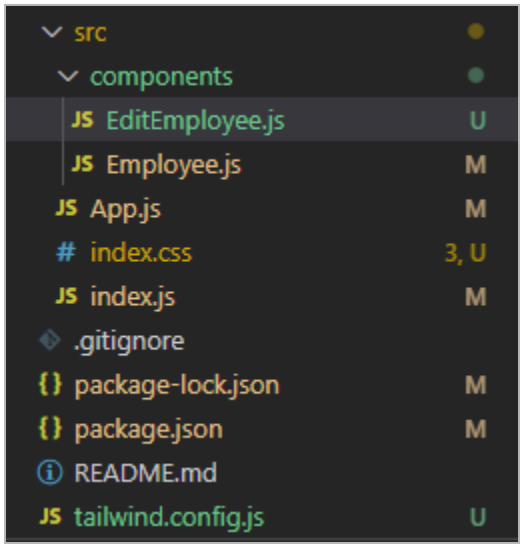
This page contains the React Bootstrap installation tutorial. **First, we will copy and paste the following command into the terminal.**

```
npm install react-bootstrap bootstrap
```

This will be added to the package.json file. We must copy and paste the CSS code into the index.js file. **The code is provided below:**

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Following that, we need to develop a new component, as seen in the figure below. The component is called EditEmployee.js.



Then, we'll go to the React Bootstrap documentation. Here, we will look for a modal and find a variety of alternatives. In this project, we choose the Static Backdrop Modal from the manual. We'll copy the code and put it into our newly created component.

```
import React, { useState } from 'react';
import Button from 'react-bootstrap/Button';
import Modal from 'react-bootstrap/Modal';

function Example() {
  const [show, setShow] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (
    <>
      <Button variant="primary" onClick={handleShow}>
        Launch static backdrop modal
```

```
      </Button>

      <Modal
        show={show}
        onHide={handleClose}
        backdrop="static"
        keyboard={false}
      >
        <Modal.Header closeButton>
          <Modal.Title>Modal title</Modal.Title>
        </Modal.Header>
        <Modal.Body>
          I will not close if you click outside me. Don't even try to press
          escape key.
        </Modal.Body>
        <Modal.Footer>
          <Button variant="secondary" onClick={handleClose}>
            Close
          </Button>
          <Button variant="primary">Understood</Button>
        </Modal.Footer>
      </Modal>
    </>
  );
}
```

However, we must make a few changes to the preceding code, such as altering the function name and adding the export statement. **The modified code is provided below**:

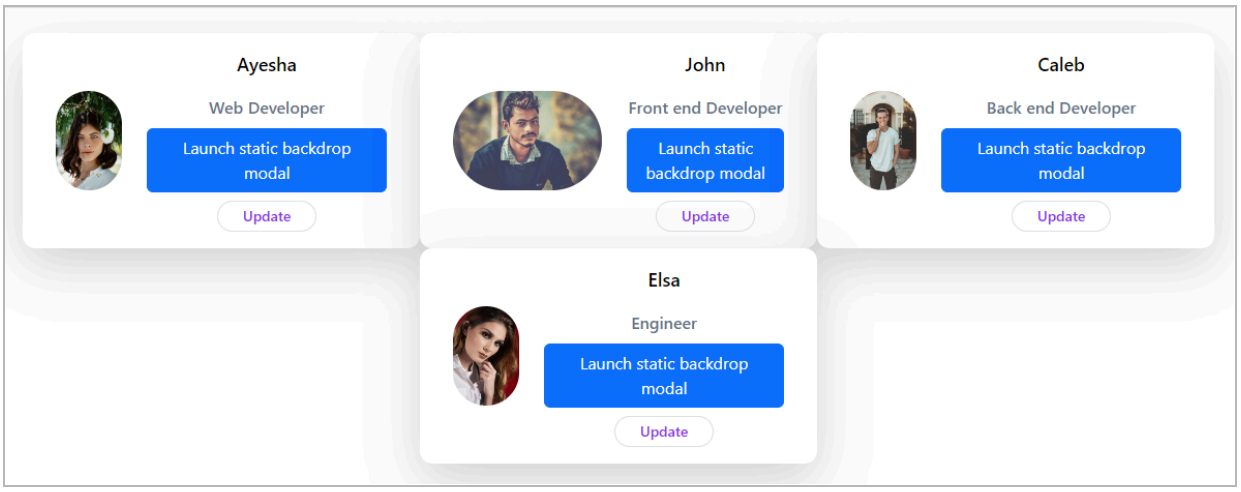
```
import React, { useState } from 'react';
import Button from 'react-bootstrap/Button';
import Modal from 'react-bootstrap/Modal';

function EditEmployee() {
```

```
  const [show, setShow] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (
    <>
      <Button variant="primary" onClick={handleShow}>
        Launch static backdrop modal
      </Button>

      <Modal
        show={show}
        onHide={handleClose}
        backdrop="static"
        keyboard={false}
      >
        <Modal.Header closeButton>
          <Modal.Title>Modal title</Modal.Title>
        </Modal.Header>
        <Modal.Body>
          I will not close if you click outside me. Don't even try to press
          escape key.
        </Modal.Body>
        <Modal.Footer>
          <Button variant="secondary" onClick={handleClose}>
            Close
          </Button>
          <Button variant="primary">Understood</Button>
        </Modal.Footer>
      </Modal>
    </>
  );
}

export default EditEmployee;
```

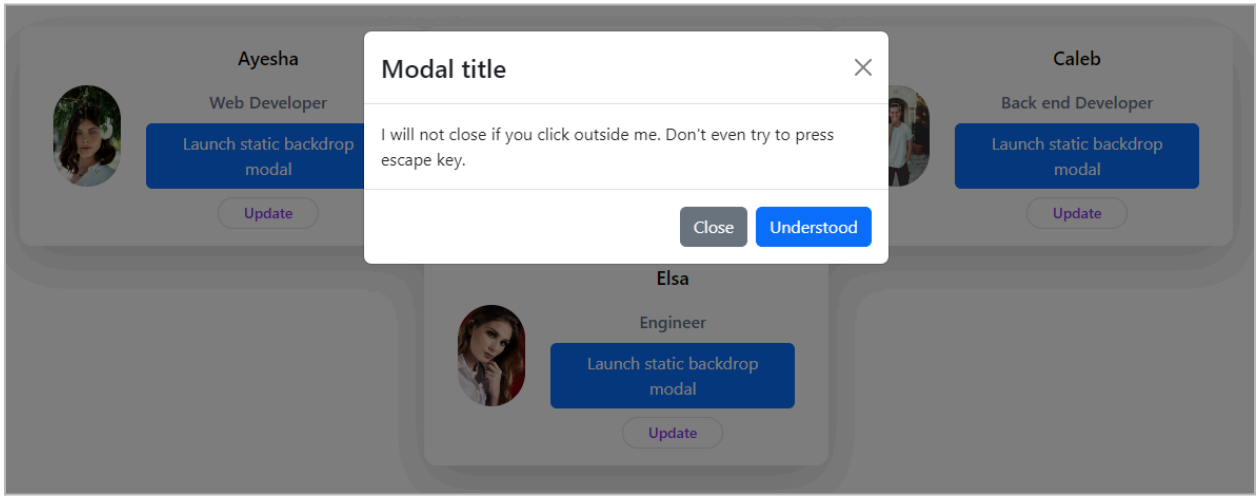
The next step is to add the <EditEmployee/> button to our code in the Employee.js file.

```
import EditEmployee from "./EditEmployee";

function Employee(props) {
    return (
        <div className="py-8 px-8 max-w-sm bg-white rounded-xl
shadow-lg space-y-2 sm:py-4 sm:flex sm:items-center sm:space-y-0
sm:space-x-6">
  <img className="block mx-auto h-24 rounded-full sm:mx-0
sm:shrink-0"
  src={props.img}
  alt="Woman's Face"/>
  <div className="text-center space-y-2 sm:text-left">
    <div className="space-y-0.5">
      <p className="text-lg text-black font-semibold">{props.name}</p>
      <p className="text-slate-500 font-medium">{props.role}
      </p>
    </div>
    <EditEmployee/>
    <button className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">
        Update</button>
  </div>
</div>
    );
}

export default Employee;
```

This is the output that we get after running our above codes.



Here, we have two buttons. One is the update button, while the other is the "Launch static backdrop modal" button. **The Update button does not operate, while the other button displays the following window:**



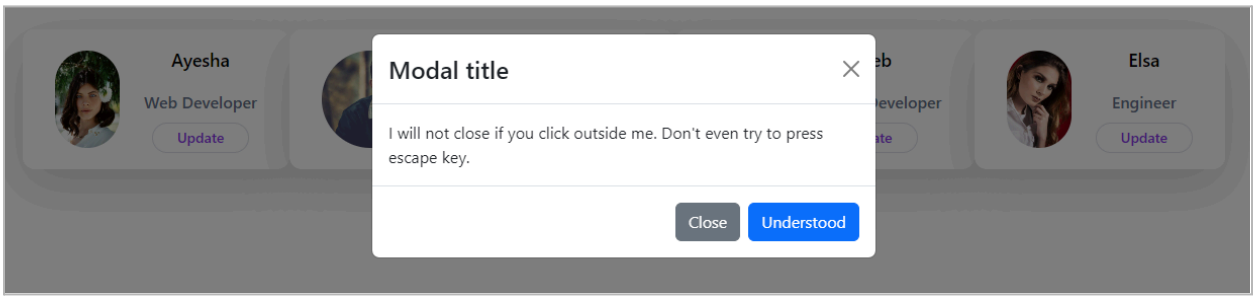
In the next step, we will take the button from the Employee.js file and paste it to our new component EditEmployee bellow:

```
import React, { useState } from 'react';
import Button from 'react-bootstrap/Button';
import Modal from 'react-bootstrap/Modal';

function EditEmployee() {
  const [show, setShow] = useState(false);
```

```
  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (
    <>
    <button onClick={handleShow}
    className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">
        Update
        </button>

      <Modal
        show={show}
        onHide={handleClose}
        backdrop="static"
        keyboard={false}
      >
        <Modal.Header closeButton>
          <Modal.Title>Modal title</Modal.Title>
        </Modal.Header>
        <Modal.Body>
          I will not close if you click outside me. Don't even try to press
          escape key.
        </Modal.Body>
        <Modal.Footer>
          <Button variant="secondary" onClick={handleClose}>
            Close
          </Button>
          <Button variant="primary">Understood</Button>
        </Modal.Footer>
      </Modal>
    </>
```

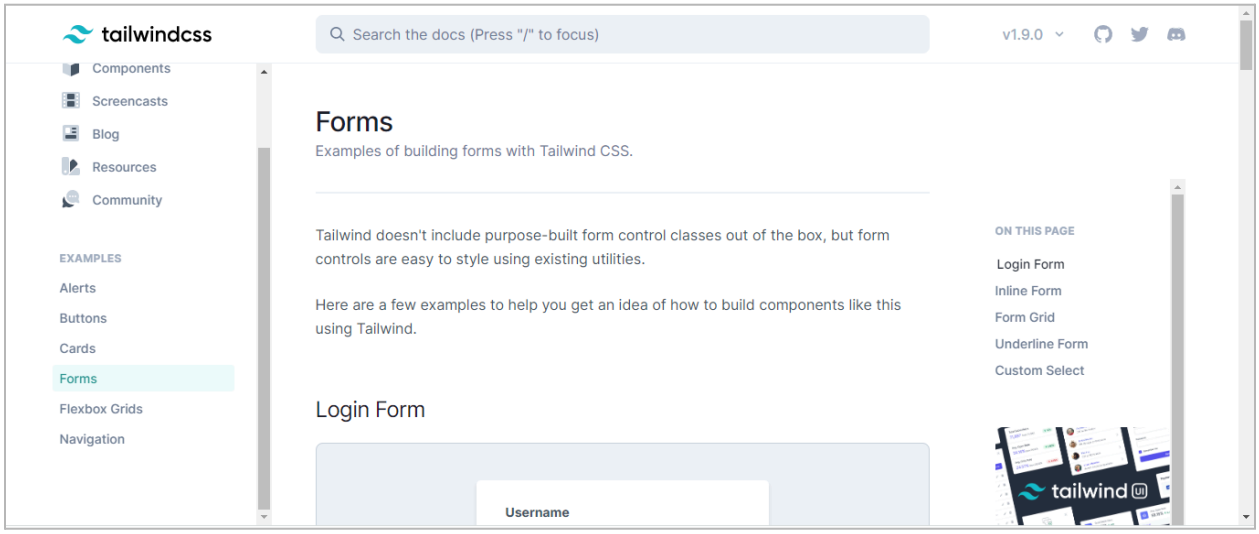
```
  );
}

export default EditEmployee;
```

We also made some more adjustments to the above code. **It produces the following output. When we click the update button, we see the modal window.**



Create and Style HTML Forms

This section will teach us to create a form inside the above modal window. We will take some code from the Tailwind website.



We took a piece of code from this page and adjusted it according to our component. **The updated code is as follows:**

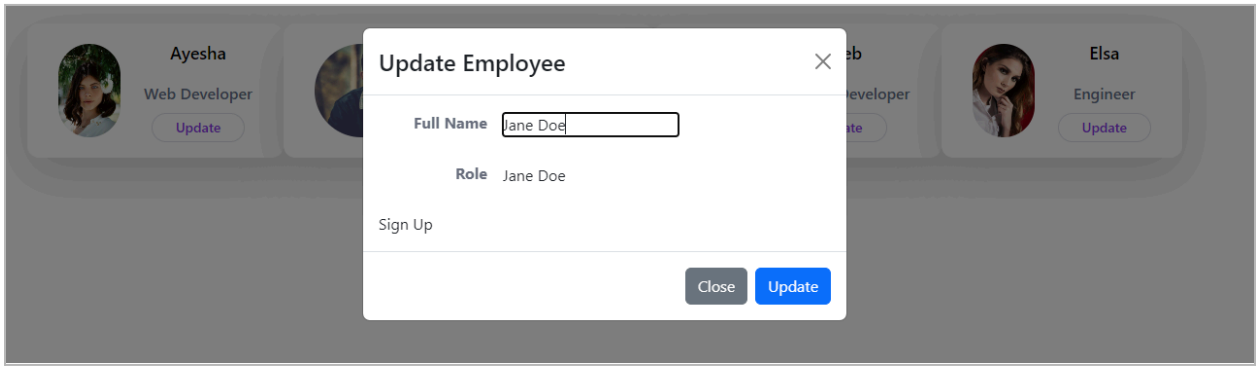
```
import React, { useState } from 'react';
import Button from 'react-bootstrap/Button';
```

```
import Modal from 'react-bootstrap/Modal';

function EditEmployee() {
  const [show, setShow] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (
    <>
    <button onClick={handleShow}
    className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">
        Update
        </button>

      <Modal
        show={show}
        onHide={handleClose}
        backdrop="static"
        keyboard={false}
      >
        <Modal.Header closeButton>
          <Modal.Title>Update Employee</Modal.Title>
        </Modal.Header>
        <Modal.Body>
        <form className="w-full max-w-sm">
  <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="name">
        Full Name
```

```
      </label>
    </div>
    <div classNameName="md:w-2/3">
      <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500" id="name"
type="text" value="Jane Doe"/>
    </div>
              </div>
              <div className="md:flex md:items-center mb-6">
    <div classNameName="md:w-1/3">
      <label classNameName="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="role">
        Role
      </label>
    </div>
    <div classNameName="md:w-2/3">
      <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500" id="role"
type="text" value="Jane Doe"/>
    </div>
              </div>
              <div classNameName="md:flex md:items-center">
              <div classNameName="md:w-1/3"></div>
              <div classNameName="md:w-2/3">
              <button classNameName="shadow bg-purple-500
hover:bg-purple-400 focus:shadow-outline focus:outline-none text-white
font-bold py-2 px-4 rounded" type="button">
                  Sign Up
              </button>
              </div>
            </div>
            </form>
```

```
      </Modal.Body>
      <Modal.Footer>
        <Button variant="secondary" onClick={handleClose}>
          Close
        </Button>
        <Button variant="primary">Update</Button>
      </Modal.Footer>
    </Modal>
  </>
 );
}

export default EditEmployee;
```

It gives us the following output:

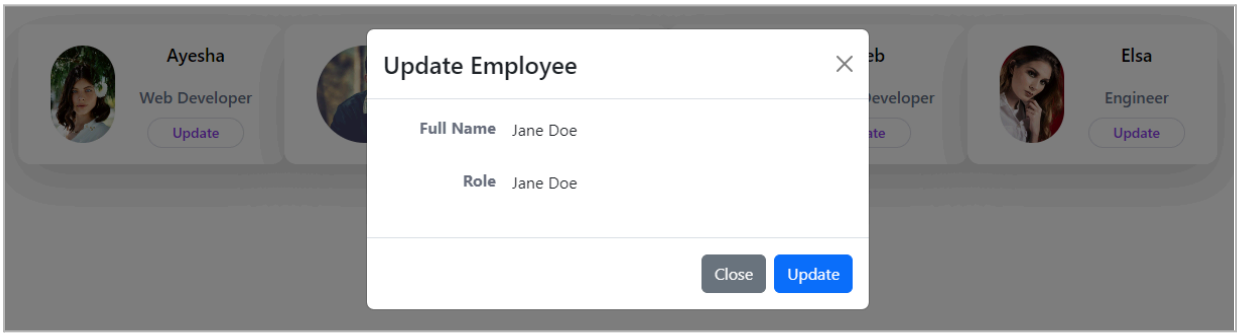


Our modal window now displays an editable form. However, we must activate the Update and Sign Up buttons right now. If we wish to use the Update button, we must rearrange our form. First, we'll remove the following code from the EditEmployee component.

```
</div>
          </div>
          <div classNameName="md:flex md:items-center">
          <div classNameName="md:w-1/3"></div>
          <div classNameName="md:w-2/3">
          <button classNameName="shadow bg-purple-500
```

```
hover:bg-purple-400 focus:shadow-outline focus:outline-none text-white
font-bold py-2 px-4 rounded" type="button">
                Sign Up
              </button>
              </div>
            </div>
```

It looks much better now as we can see below:



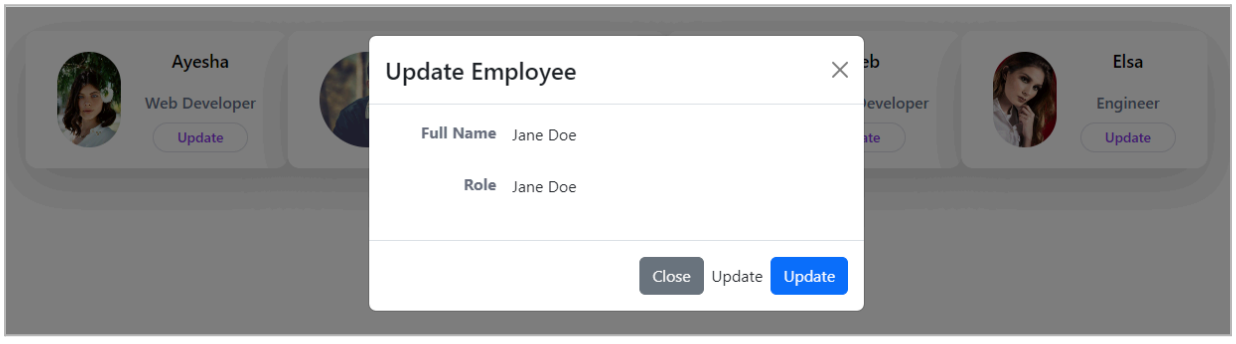
The next step is to update the function body with an ID and a new button. **You may find those changes in the code provided below.**

```
function EditEmployee() {
  const [show, setShow] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (
    <>
    <button onClick={handleShow}
    className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">
        Update
        </button>
```

```
    <Modal
      show={show}
      onHide={handleClose}
      backdrop="static"
      keyboard={false}
    >
      <Modal.Header closeButton>
        <Modal.Title>Update Employee</Modal.Title>
      </Modal.Header>
      <Modal.Body>
      <form id = 'editmodal'className="w-full max-w-sm">
  <div className="md:flex md:items-center mb-6">
   <div className="md:w-1/3">
    <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="name">
      Full Name
    </label>
   </div>
   <div classNameName="md:w-2/3">
    <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500" id="name"
type="text" value="Jane Doe"/>
   </div>
            </div>
            <div className="md:flex md:items-center mb-6">
   <div className="md:w-1/3">
    <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="role">
      Role
    </label>
   </div>
   <div classNameName="md:w-2/3">
```

```
      <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500" id="role"
type="text"
      value="Jane Doe"/>
      </div>
      </div>
           </form>
        </Modal.Body>
        <Modal.Footer>
          <Button variant="secondary" onClick={handleClose}>
            Close
          </Button>
          <button form= "editmodal">Update</button>
          <Button variant="primary">Update</Button>
        </Modal.Footer>
      </Modal>
    </>
  );
}

export default EditEmployee;
```

The above code changes give us a new update button on our web page.



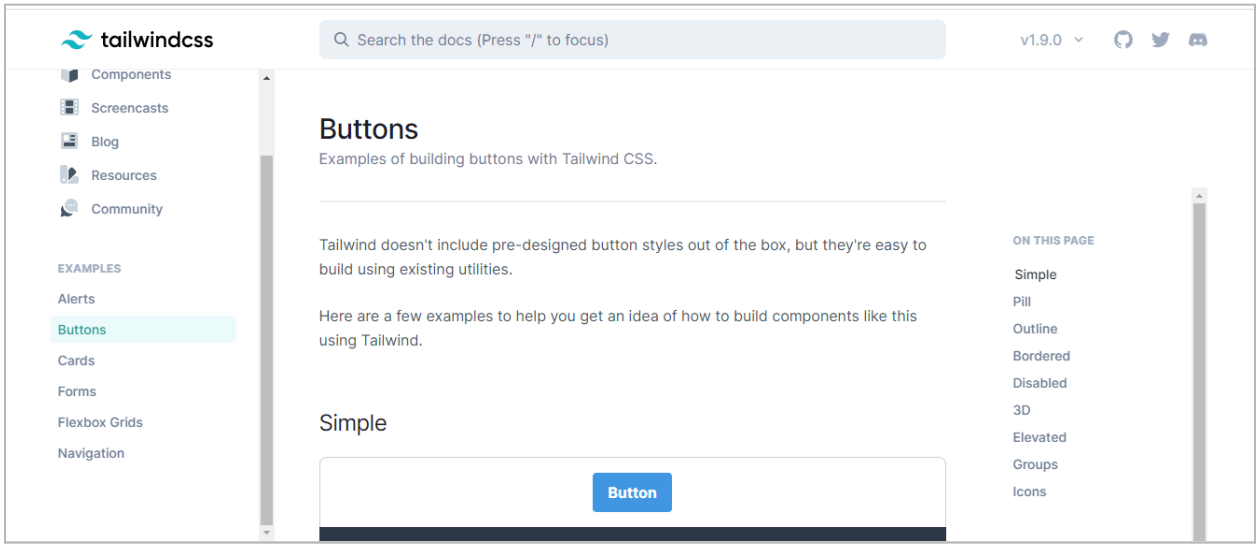
However, the outcome does not match our expectations. As a result, we will add a class to our button to make it more visually appealing. Therefore, we eliminated the following button from our code in EditEmployee.js:

```
<Button
variant="primary">Update</Button>
```

Then, we have to add a class in the following line of code:

```
<button form=
"editmodal">Update</button>
```

For this purpose, we will navigate to the Tailwind collections of buttons.

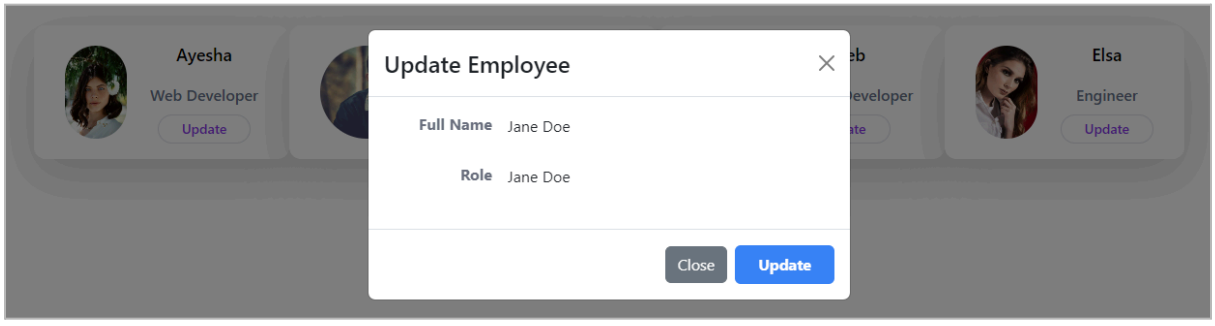


In the first button, named “Simple,” we will copy its classes. **You can see the classes in the image given below:**

After copying them, we will paste them into our button code in EditEmployee.js as follows:

```
<button className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded" form= "editmodal">Update</button>
```

Now, it is giving us the following output:



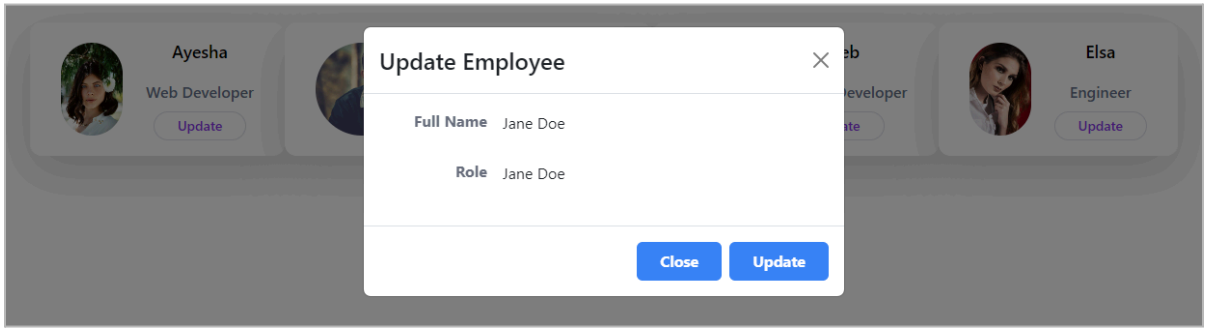
We will create a few additional arrangements if we want to maintain consistency with both the Close and Update buttons. **We eliminated the following Close button from our code:**

```
<Button variant="secondary"
onClick={handleClose}>
          Close
        </Button>
```

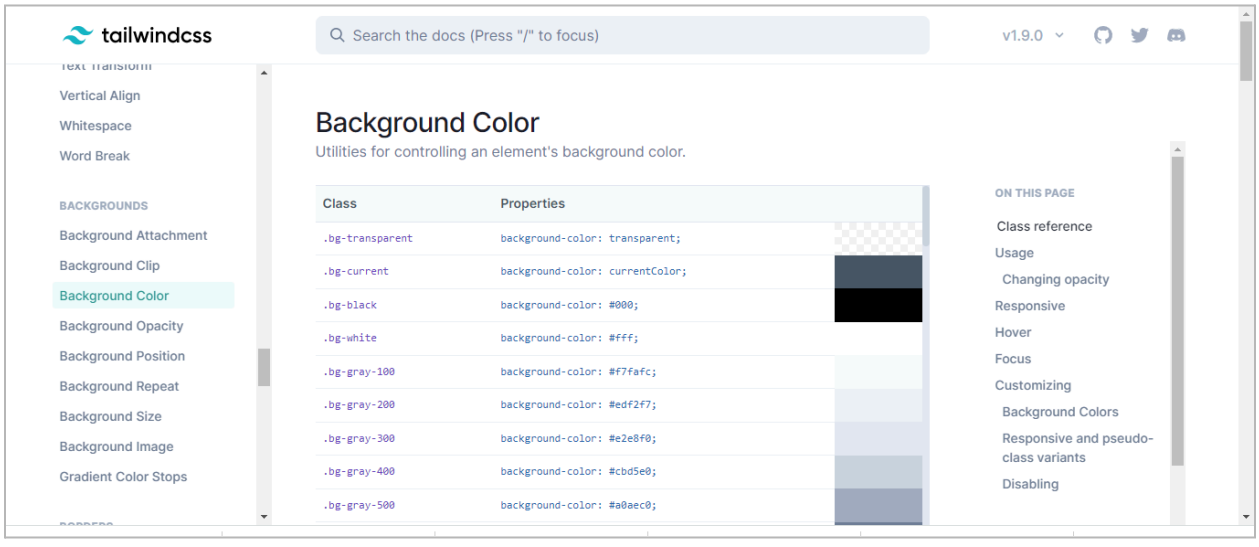
Then, we added the following Close button into our code:

```
 <button className="bg-blue-500 hover:bg-blue-700 text-white font-bold
py-2 px-4 rounded"
        onClick={handleClose}>Close</button>
```

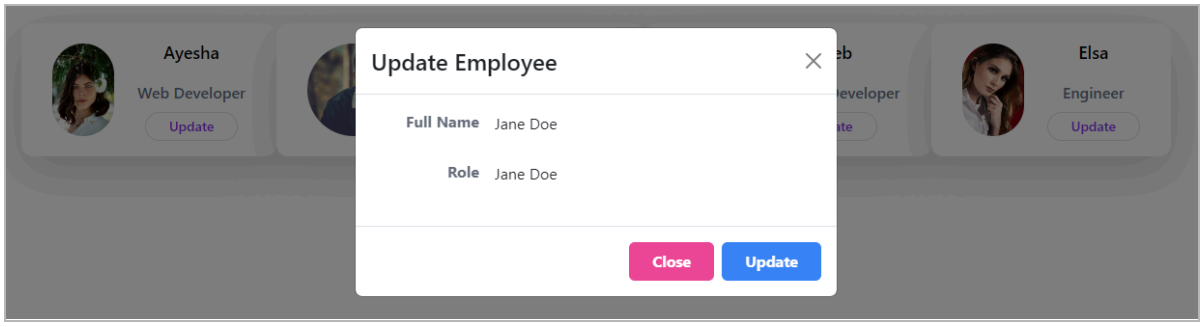
Now, we can see that the things are consistent in our output:



Let's change the color of our Close button. We'll go to Tailwind and search for the background in the search box. **Here, we can see various options:**

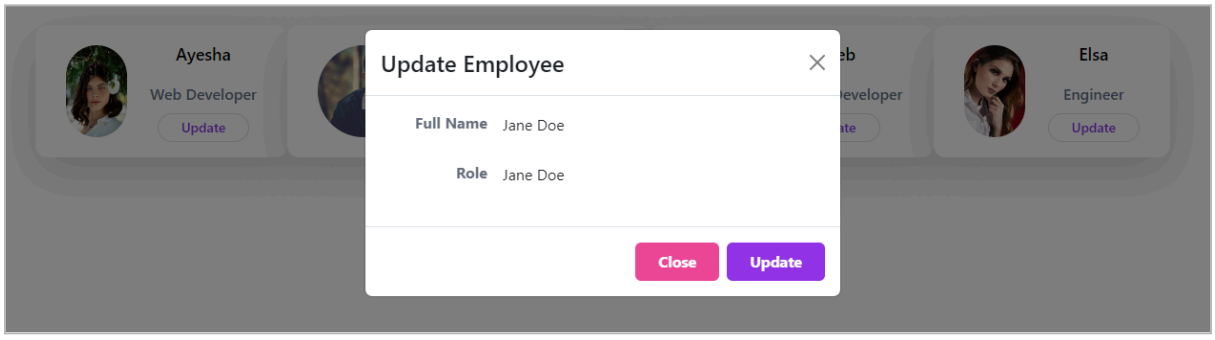


So, we copied the bg-pink-500 and added it to our class. **We get the output as follows:**

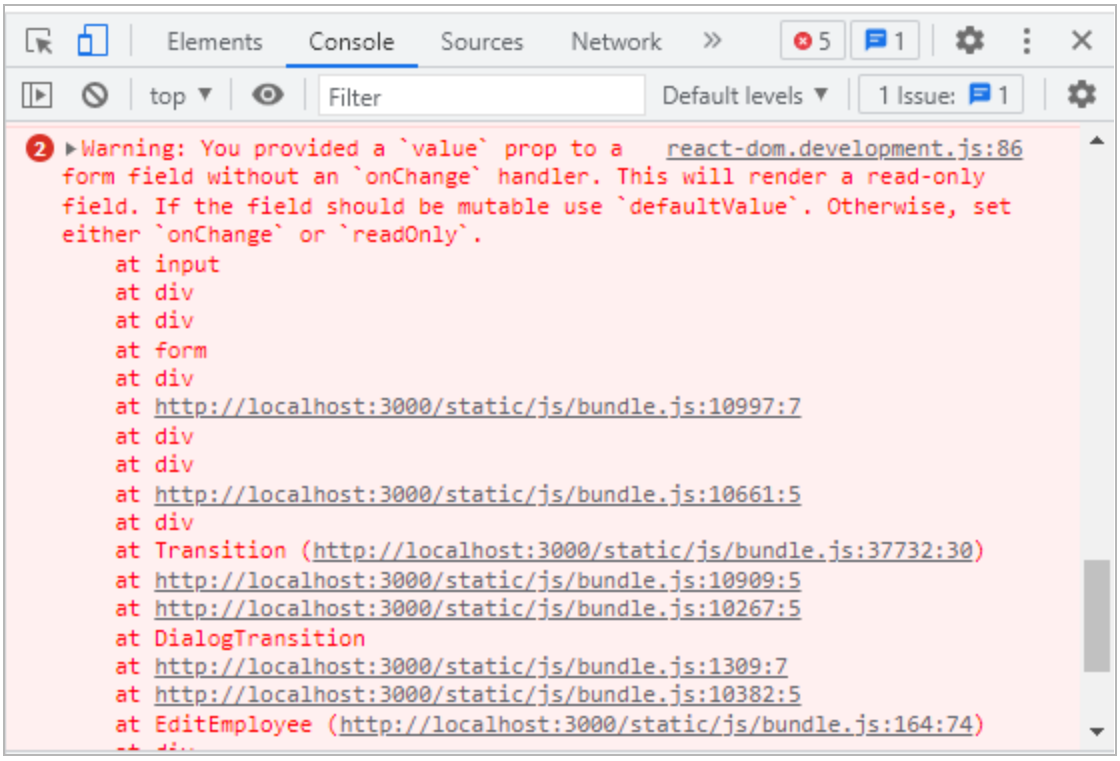


Now we can discuss the hue of Hover. It would be preferable to make the Update button the same hue as Hover, which is bg-purple-700. We made the following improvements to our Update button.

```
<button className="bg-purple-600 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"
        form= "editmodal">Update</button>
```



Even though everything appears to have improved significantly. But there is something we must concentrate on. The following problem appears on our web page when we access the developer console.

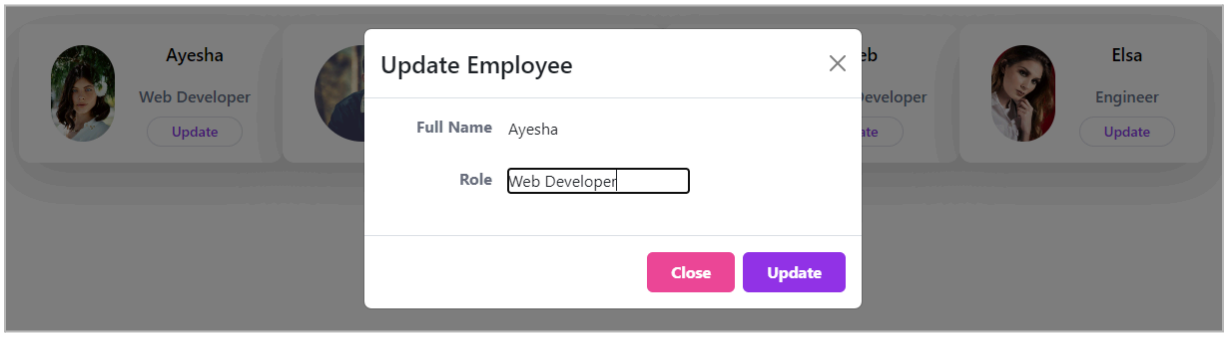


In the next section, we will learn how to fix this error and improve our code.

Profile Form Data in Modal

In this section, we'll look at how to create an editable form in React and establish default values that appear when the form is loaded. But before we start, let's address the previous section's issue, which was a warning about the form's value.

To resolve this issue, open the EditEmployee component and replace the value with defaultValue. This change will correct the problem and allow you to alter the form fields as intended. The image below shows the ultimate product.



Even though the form is now editable, we can still not adjust its value. To make the value updatable, we must delete the DefaultValue from the EditEmployee component and add props. **The following changes were made to the code in EditEmployee.js:**

```
function EditEmployee(props) {
  const [show, setShow] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (
    <>
    <button onClick={handleShow}
    className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">
        Update
        </button>

      <Modal
        show={show}
        onHide={handleClose}
        backdrop="static"
        keyboard={false}
      >
        <Modal.Header closeButton>
          <Modal.Title>Update Employee</Modal.Title>
        </Modal.Header>
        <Modal.Body>
        <form id = 'editmodal'className="w-full max-w-sm">
  <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="name">
        Full Name
      </label>
```

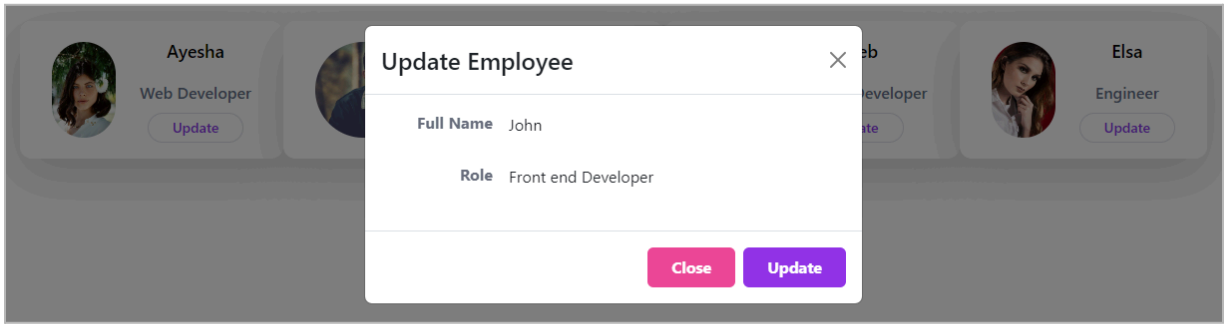
```
    </div>
    <div classNameName="md:w-2/3">
      <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200
      rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
      id="name" type="text"
      value={props.name}
      />
    </div>
              </div>
              <div classNameName="md:flex md:items-center mb-6">
    <div classNameName="md:w-1/3">
      <label classNameName="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="role">
        Role
      </label>
    </div>
    <div classNameName="md:w-2/3">
      <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500" id="role"
type="text"
      value={props.role}
      />
      </div>
      </div>
           </form>
        </Modal.Body>
        <Modal.Footer>
          <button classNameName="bg-pink-500 hover:bg-pink-700 text-white
font-bold py-2 px-4 rounded"
          onClick={handleClose}>Close</button>
           <button classNameName="bg-purple-600 hover:bg-purple-700 text-white
```

```
font-bold py-2 px-4 rounded"
          form= "editmodal">Update</button>
        </Modal.Footer>
      </Modal>
    </>
  );
}

export default EditEmployee;
```

Then, we must add the following code line to our code in the Employee.js file.

```
<EditEmployee name={props.name}
role={props.role}/>
```

After running the code, we can see that the name and role are automatically updated on our web page:



The next step is to allow the edit modal to keep the form's content state. Let us replace the values that way while maintaining their state.

We made adjustments to the EditEmployee.js file. **The modified code is provided below.**

```
function EditEmployee(props) {
  const [name, setName] = useState(props.name);
  const [role, setRole] = useState(props.role);
  const [show, setShow] = useState(false);
```

```
  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (
    <>
    <button onClick={handleShow}
    className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">
        Update
        </button>

      <Modal
        show={show}
        onHide={handleClose}
        backdrop="static"
        keyboard={false}
      >
        <Modal.Header closeButton>
          <Modal.Title>Update Employee</Modal.Title>
        </Modal.Header>
        <Modal.Body>
        <form id = 'editmodal'className="w-full max-w-sm">
  <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="name">
        Full Name
      </label>
    </div>
    <div classNameName="md:w-2/3">
      <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200
```

```
      rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
      id="name" type="text"
      value={name}
      onChange={(e)=>{setName(e.target.value)}}
      />
    </div>
              </div>
              <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="role">
        Role
      </label>
    </div>
    <div classNameName="md:w-2/3">
      <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500" id="role"
type="text"
      value={role}
      onChange={(e)=>{setRole(e.target.value)}}
      />
      </div>
      </div>
           </form>
        </Modal.Body>
        <Modal.Footer>
          <button className="bg-pink-500 hover:bg-pink-700 text-white
font-bold py-2 px-4 rounded"
          onClick={handleClose}>Close</button>
           <button className="bg-purple-600 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"
          form= "editmodal">Update</button>
```

```
      </Modal.Footer>
    </Modal>
   </>
  );
}

export default EditEmployee;
```

Using the onChange attribute for the variables' names and roles, these fields become editable as soon as the page loads. However, it is critical to appropriately update these values after making modifications, which we shall discuss in the following section.

Update Parent Component State in Child Component

In this phase, we must update the following data from the App.js file with the data from the EditEmployee.js file.

```
function App() {
  const [role, setRole] = useState('GIS Analyst');
  const [employees, setEmployees] = useState(
    [
      {name: 'Ayesha',
      role: 'Web Developer',
      img: 'https://images.pexels.com/photos/3586798/pexels-photo-3586798.jpeg?auto=compress&cs=tinysrgb&w=1260
    },
    {name: 'John',
      role: 'Front end Developer',
      img: 'https://images.pexels.com/photos/694438/pexels-photo-694438.jpeg?auto=compress&cs=tinysrgb&w=1260&h=
    },
    {name: 'Caleb',
      role: 'Back end Developer',
```

It's vital to notice that App.js is the parent component of both Employee.js and EditEmployee.js. We'll add a callback method in the parent to allow communication between these components.

This function will be invoked anytime changes are required in the parent component. To get started, we add an ID and create the callback function in the App.js file.

The changed code is given below:

```
function App() {
  const [role, setRole] = useState('GIS Analyst');
  const [employees, setEmployees] = useState(
    [
      { id: 1,
```

```
        name: 'Ayesha',
      role: 'Web Developer',
      img:
'https://images.pexels.com/photos/3586798/pexels-photo-3586798.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    { id: 2,
      name: 'John',
      role: 'Front end Developer',
      img:
'https://images.pexels.com/photos/694438/pexels-photo-694438.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    {
      id: 3,
      name: 'Caleb',
      role: 'Back end Developer',
      img:
'https://images.pexels.com/photos/775358/pexels-photo-775358.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    { id: 4,
      name: 'Elsa',
      role: 'Engineer',
      img:
'https://images.pexels.com/photos/3610877/pexels-photo-3610877.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },
    ]);

    function updateEmployee(id, newName, newRole){
      console.log('updateEmployee inside of app.js');
    }

  const showEmployees = true;
  return (
    <div className="App">
      {
      console.log('inside the return') }
```

```
      {showEmployees ? (
        <>
        <input type= 'text' onChange={(e) =>{
          console.log(e.target.value);
          setRole(e.target.value);
        }}/>
        <div className= "flex flex-wrap justify-center">
       {employees.map((employee) => {
        return (
          <Employee
          key= {employee.id}
          id= {employee.id}
          name= {employee.name}
          role= {employee.role}
          img= {employee.img}
          updateEmployee={updateEmployee}
          />
        );
       })}
        </div>
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
    </div>
  );
}

export default App;
```

Then, we will add ID into our Employee.js file:

```
<EditEmployee id={props.id}
    name={props.name}
    role={props.role}
```

```
updateEmployee={props.updateEmployee}/>
```

The next step is to make big changes to our function in the EditEmployee.js file.

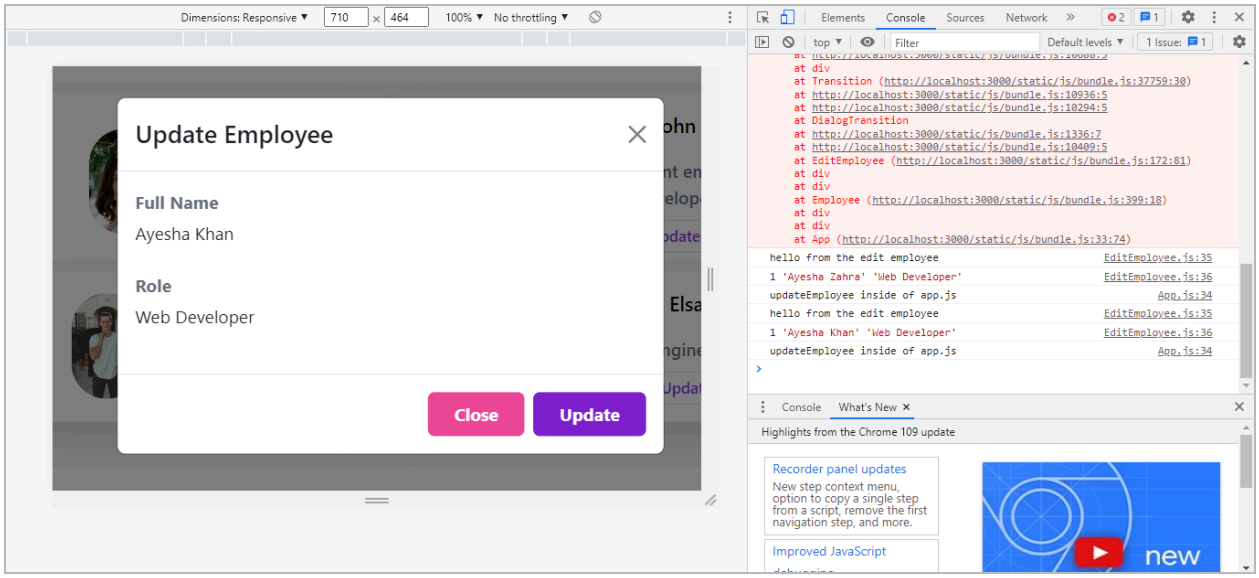
```
import React, { useState } from 'react';

import Modal from '../modal/Modal';
```

```
function EditEmployee(props) {
  const [name, setName] = useState(props.name);
  const [role, setRole] = useState(props.role);
  const [show, setShow] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (

    <>
    <button onClick={handleShow}
    className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">
        Update
        </button>

      <Modal
        show={show}
        onHide={handleClose}
        backdrop="static"
```

```
      keyboard={false}
    >
      <Modal.Header closeButton>
        <Modal.Title>Update Employee</Modal.Title>
      </Modal.Header>
      <Modal.Body>
      <form
      onSubmit={(e)=>{
        e.preventDefault();
        console.log('hello from the edit employee');
        console.log(props.id, name, role);
        props.updateEmployee(props.id, name, role);

      }}

      id = 'editmodal'className="w-full max-w-sm">
  <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="name">
        Full Name
      </label>
    </div>
    <div classNameName="md:w-2/3">
      <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200
      rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
      id="name" type="text"
      value={name}
      onChange={(e)=>{setName(e.target.value)}}
      />
    </div>
            </div>
```

```
              <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="role">
        Role
      </label>
    </div>
    <div classNameName="md:w-2/3">
      <input classNameName="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500" id="role"
type="text"
      value={role}
      onChange={(e)=>{setRole(e.target.value)}}
      />
      </div>
      </div>
           </form>
        </Modal.Body>
        <Modal.Footer>
          <button className="bg-pink-500 hover:bg-pink-700 text-white
font-bold py-2 px-4 rounded"
          onClick={handleClose}>Close</button>
           <button
           className="bg-purple-600 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"
          form= "editmodal">Update</button>
        </Modal.Footer>
      </Modal>
    </>
  );
}

export default EditEmployee;
```

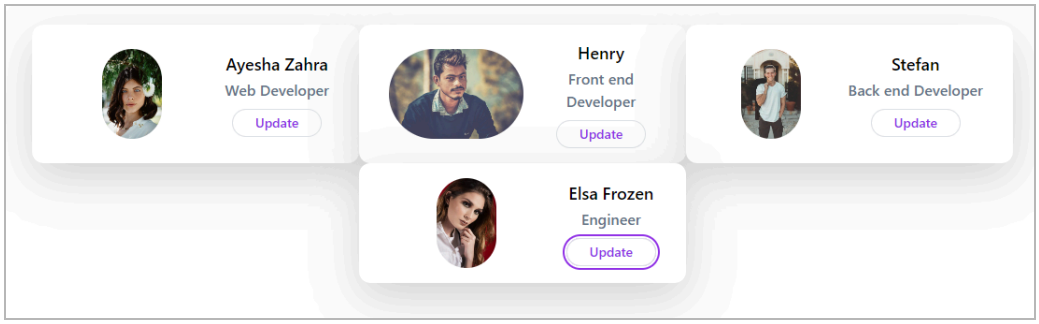
After running the code, we can see the modifications to our web page in the developer console. We altered the name "Ayesha" into "Ayesha Zahra" and, subsequently, "Ayesha Khan" bellow:



We made a few more changes to make it updatable on the web page and not just the developer console. In the App.js file, we made the following changes to our function updateEmployee

```
function updateEmployee(id, newName, newRole){
    const updatedEmployees = employees.map((employee)=>{
      if (id == employee.id){

      return {...employee, name: newName, role: newRole};
      }
      return employee;
    });
    setEmployees(updatedEmployees);
  }
```

The updated names are shown on the web page now:

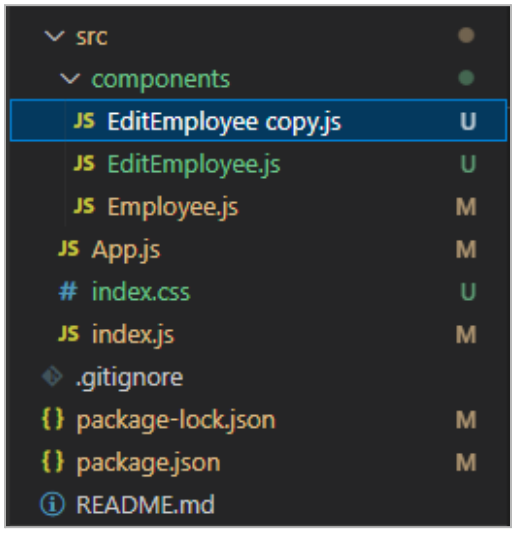


Here we are doing one more thing. When we change the name or role, the form does not close but remains on the screen. To close it automatically right after we click on the Update button, we add a property handleClose(), as seen below.

```
onSubmit={(e)=>{
        handleClose();
        e.preventDefault();
        console.log('hello from the edit employee');
        console.log(props.id, name, role);
        props.updateEmployee(props.id, name, role);
```

How to Push to State Array?

This section will teach how to add a new employee to our web page. First of all, we will copy and paste our EditEmployee.js file. **You can see it in the screenshot given below:**



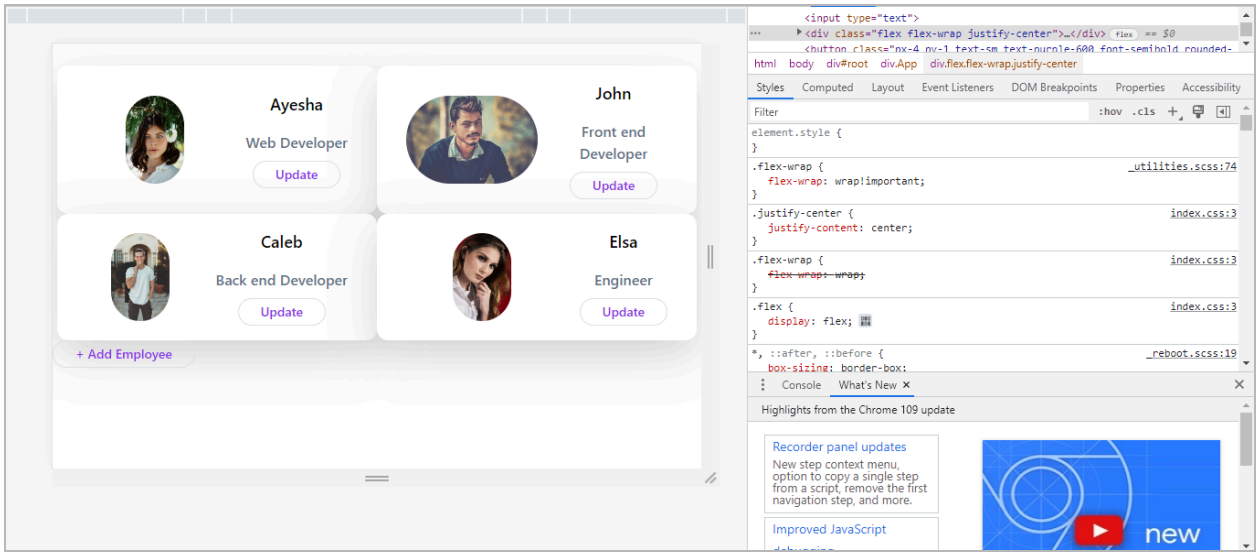
The following step is to rename the new file "AddEmployee.js." Then, we must alter the new file's EditEmployee function to AddEmployee. We also need to modify the method at the bottom of the code, as seen below.

export default AddEmployee;

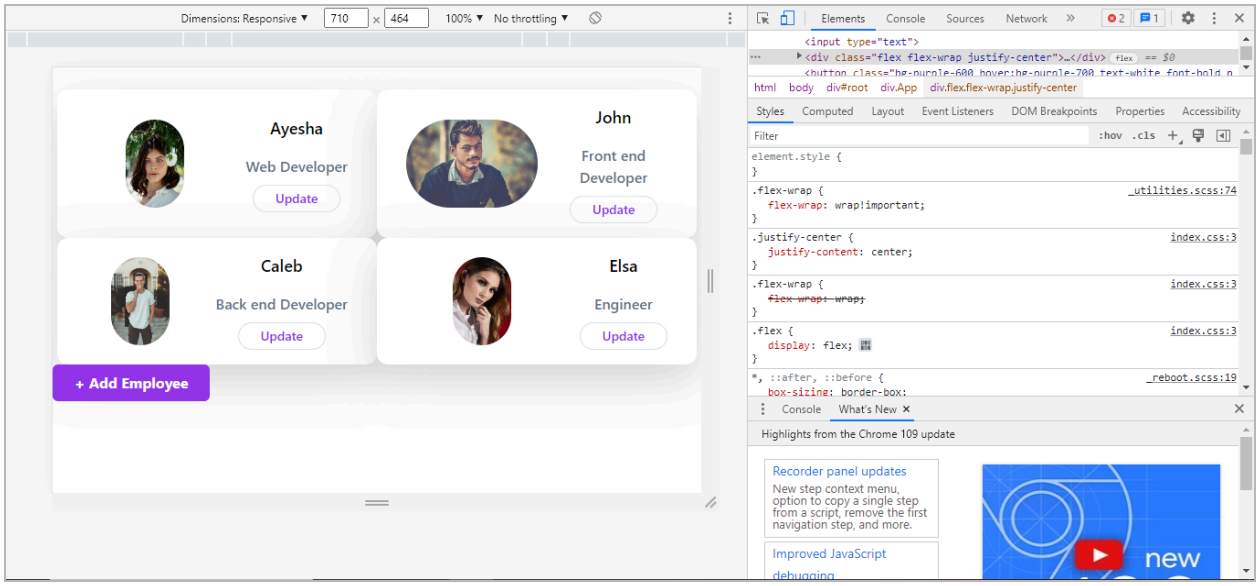
The main difference between the EditEmployee and AddEmployee components is that the values in EditEmployee were filled. On the other hand, we will add our values in the AddEmployee. We made the following changes to our AddEmployee component.

```
function AddEmployee(props) {
  const [name, setName] = useState('');
  const [role, setRole] = useState('');
  const [show, setShow] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (

    <>
    <button onClick={handleShow}
    className="px-4 py-1 text-sm text-purple-600 font-semibold
rounded-full border border-purple-200 hover:text-white
hover:bg-purple-600 hover:border-transparent focus:outline-none
focus:ring-2 focus:ring-purple-600 focus:ring-offset-2">
        + Add Employee
        </button>
```

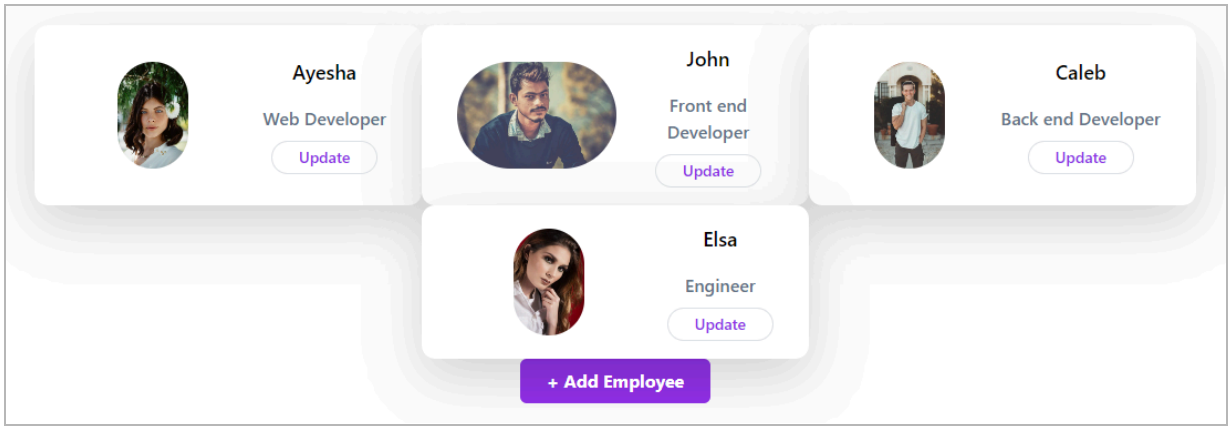
Then, we added the AddEmployee component to our App.js file. We can see an “Add Employee” button on our web page.



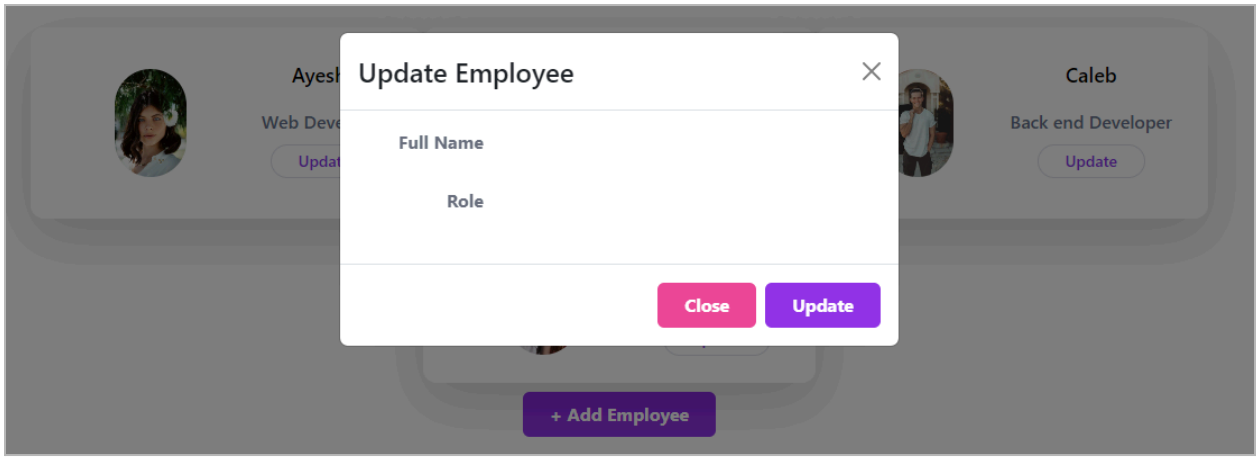
But the button is not how we want it to appear. So, we will replace the button's styling. We can see that the styling has now changed.



However, we can see that the button is on the screen's left side. To incorporate it into the button, we will make minor changes by adding block mx-auto to its style. **Now, we can see it in the center of the screen.**



When we click on this newly created button to add an employee, it shows Update Employee.



To make it “Add Employee”, we must change a code line in our AddEmployee components.

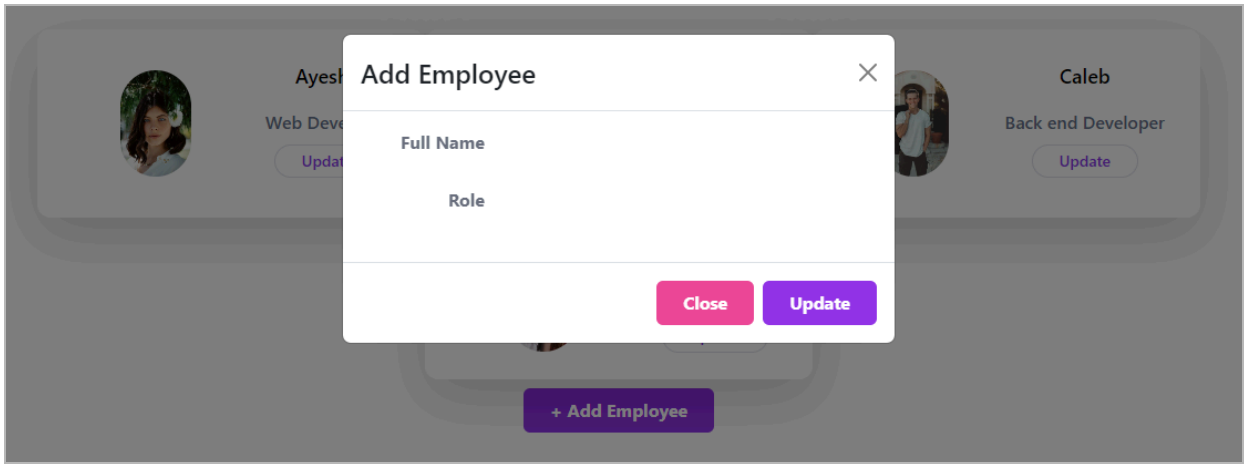
We made the following changes:

```
<Modal.Title>Update Employee</Modal.Title>
```

to

```
<Modal.Title>Add Employee</Modal.Title>
```

Now, we can see that the dialogue box is updated.



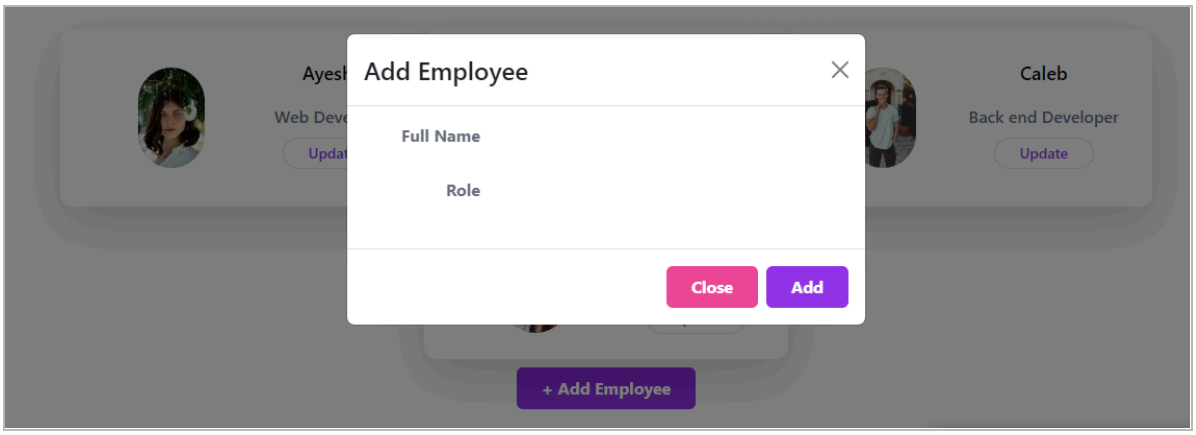
We also need to change the Update purple button in this dialogue box. **Therefore, we made the following changes.**

```
<button
        className="bg-purple-600 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"
        form= "editmodal">
          Update
        </button>
```

to

```
<button
        className="bg-purple-600 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"
        form= "editmodal">
         Add
        </button>
```

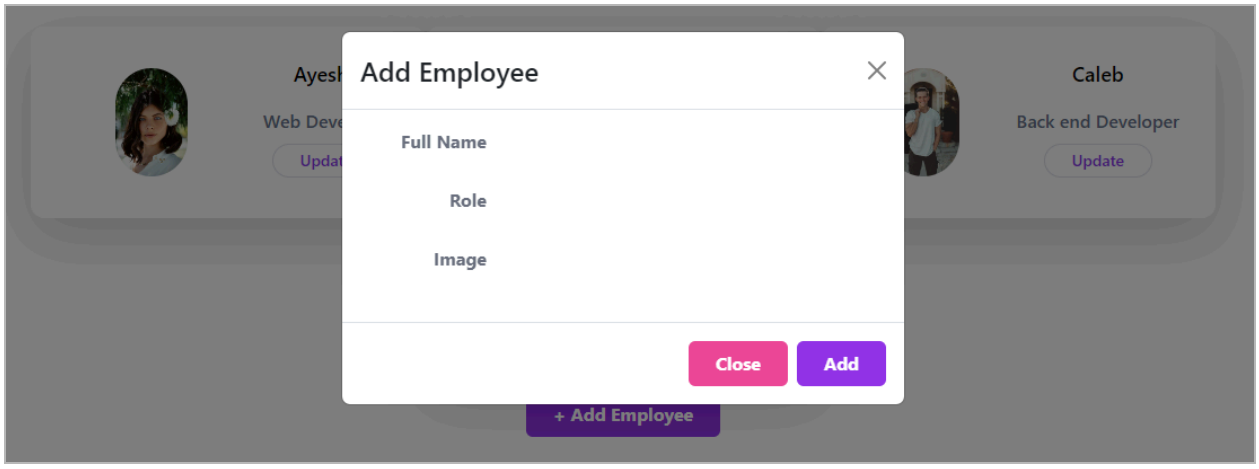
The final output is shown below:



The next step is to include an image input in our chat box. As a result, we will add the following code to the AddEmployee component.

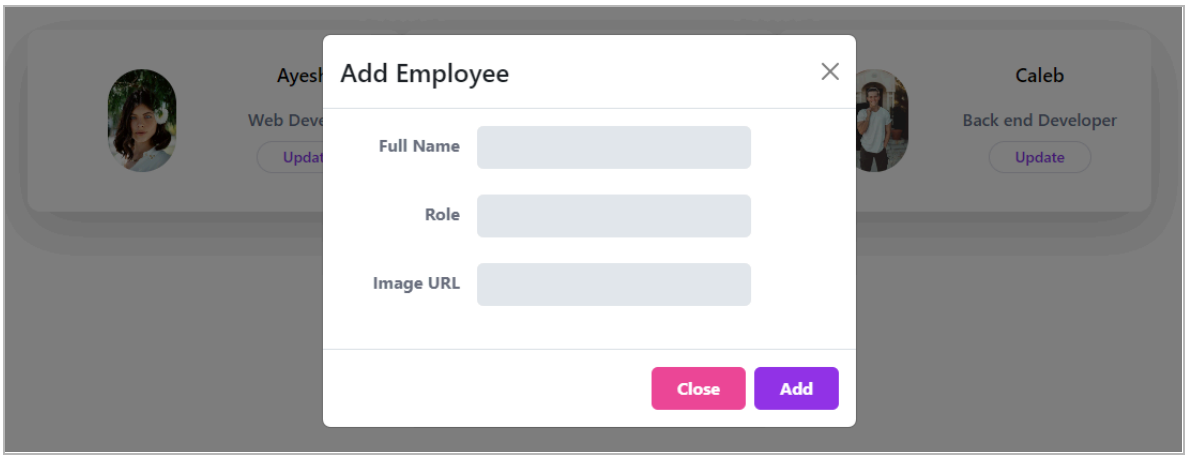
```
<div className="md:flex md:items-center mb-6">
   <div className="md:w-1/3">
    <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="img">
       Image
    </label>
   </div>
   <div className="md:w-2/3">
    <input className="bg-gray-200 appearance-none border-2
    border-gray-200 rounded w-full py-2 px-4 text-gray-700
    leading-tight focus:outline-none focus:bg-white
focus:border-purple-500" id="img" type="text"
    value={img}
    onChange={(e)=>{setImg(e.target.value)}}
    />
    </div>
    </div>
```

The output is shown as follows:



We can also make it an image URL. **Therefore, we will change the code to the following:**

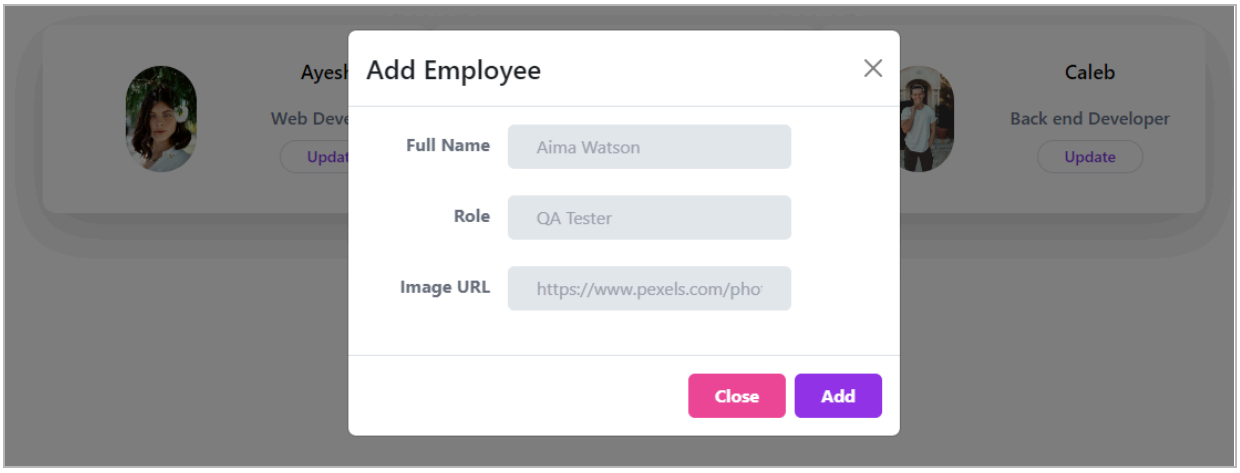
```
<div className="md:flex md:items-center mb-6">
   <div className="md:w-1/3">
     <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="img">
       Image URL
     </label>
   </div>
   <div className="md:w-2/3">
     <input className="bg-gray-200 appearance-none border-2
     border-gray-200 rounded w-full py-2 px-4 text-gray-700
     leading-tight focus:outline-none focus:bg-white
focus:border-purple-500" id="img" type="text"
     value={role}
     onChange={(e)=>{setRole(e.target.value)}}
     />
     </div>
     </div>
```



Let's give our new employees some value by employing a placeholder. The code for the Image URL is provided below. Repeat the process for the Full Name and Role data in the dialog box. As shown below, we must include it in the "id" part.

```
</div>
   <div className="md:w-2/3">
     <input className="bg-gray-200 appearance-none border-2
     border-gray-200 rounded w-full py-2 px-4 text-gray-700
     leading-tight focus:outline-none focus:bg-white
focus:border-purple-500"
     id="img"
     placeholder=
"https://www.pexels.com/photo/woman-wearing-hat-3310695/"
     type="text"
     value={role}
     onChange={(e)=>{setRole(e.target.value)}}
     />
     </div>
     </div>
```

The output is shown below:



The final code of the above output will look like the below code:

```
<div className="md:flex md:items-center mb-6">
   <div className="md:w-1/3">
     <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="name">
       Full Name
     </label>
   </div>
   <div className="md:w-2/3">
     <input className="bg-gray-200 appearance-none border-2
border-gray-200
     rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
     id="name"
     placeholder="Aima Watson"
     type="text"
     value={name}
     onChange={(e)=>{setName(e.target.value)}} // onChange event gets
triggered when the input field is edited
     />
   </div>
            </div>
            <div className="md:flex md:items-center mb-6">
```

```
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="role">
        Role
      </label>
    </div>
    <div className="md:w-2/3">
      <input className="bg-gray-200 appearance-none border-2
border-gray-200 rounded
      w-full py-2 px-4 text-gray-700 leading-tight focus:outline-none
focus:bg-white
      focus:border-purple-500"
      id="role"
      placeholder="QA Tester"
      type="text"
      value={role}
      onChange={(e)=>{setRole(e.target.value)}}
      />
      </div>
      </div>
      <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="img">
        Image URL

      </label>
    </div>
    <div className="md:w-2/3">
      <input className="bg-gray-200 appearance-none border-2
      border-gray-200 rounded w-full py-2 px-4 text-gray-700
      leading-tight focus:outline-none focus:bg-white
focus:border-purple-500"
      id="img"
```

```
    placeholder=
"https://www.pexels.com/photo/woman-wearing-hat-3310695/"
    type="text"
    value={role}
    onChange={(e)=>{setRole(e.target.value)}}
    />
    </div>
    </div>
```

We added the information manually for now, but we need to make it editable and updatable via the dialogue box. To modify the web page, we must create a new function in our App.js file, as we did earlier. **Let's start.**

First of all, we added a new function in App.js. **The code is given as follows:**

```
function newEmployee(name, role, img) {
    const newEmployee = {
      id: uuidv4(),
      name: name,
      role: role,
      img: img,
    };
    setEmployees( [...employees, newEmployee]

    )
  }
```

Then, we changed the AddEmployee button in the same component; App.js.

```
<AddEmployee newEmployee = {newEmployee}/>
```

The next step is to modify the AddEmployee component. We moved the handleclose from top to bottom. We then made a few more adjustments to the code. **The completed code is shown below.**

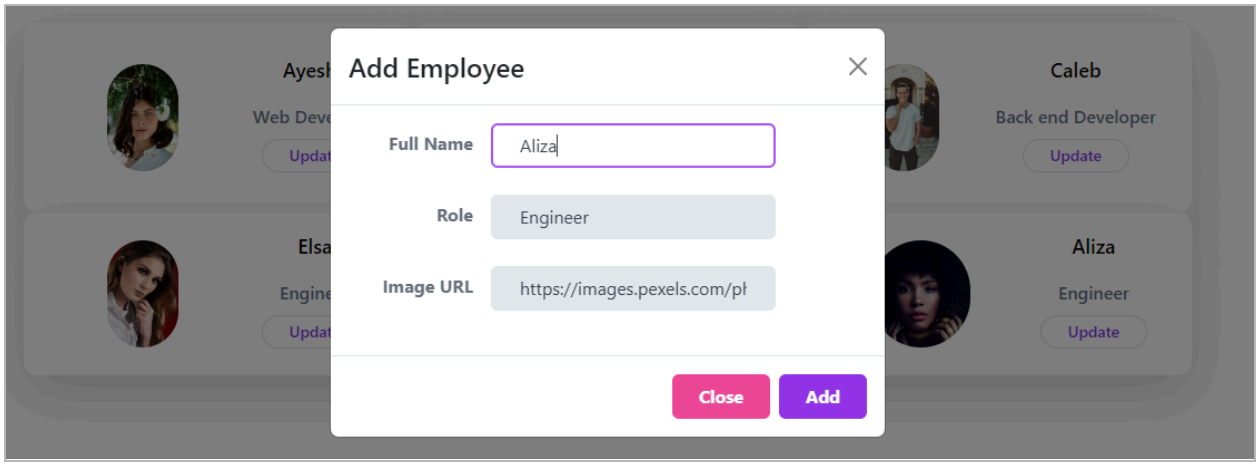
```
import React, { useState } from 'react';
import Button from 'react-bootstrap/Button';
import { propTypes } from 'react-bootstrap/esm/Image';
import Modal from 'react-bootstrap/Modal';

function AddEmployee(props) {
  const [name, setName] = useState('');
  const [role, setRole] = useState('');
  const [img, setImg] = useState('');
  const [show, setShow] = useState(false);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (

    <>
    <button onClick={handleShow}
    className="block mx-auto m-2 bg-purple-600 hover:bg-purple-700
text-white font-bold py-2 px-4 rounded">
        + Add Employee
        </button>

      <Modal
        show={show}
        onHide={handleClose}
        backdrop="static"
        keyboard={false}
      >
        <Modal.Header closeButton>
```

```
          <Modal.Title>Add Employee</Modal.Title>
        </Modal.Header>
        <Modal.Body>
        <form
        onSubmit={(e)=>{
          e.preventDefault();
          props.newEmployee(name, role, img);

        }}

        id = 'editmodal'className="w-full max-w-sm">
  <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="name">
        Full Name
      </label>
    </div>
    <div className="md:w-2/3">
      <input className="bg-gray-200 appearance-none border-2
border-gray-200
      rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
      id="name"
      placeholder='Aima'
      type="text"
      value={name}
      onChange={(e)=>{setName(e.target.value)}}
      />
    </div>
              </div>
              <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
```

```
md:mb-0 pr-4" for="role">
        Role
      </label>
    </div>
    <div className="md:w-2/3">
      <input className="bg-gray-200 appearance-none border-2
border-gray-200 rounded
      w-full py-2 px-4 text-gray-700 leading-tight focus:outline-none
focus:bg-white
      focus:border-purple-500"
      id="role"
      placeholder="QA Tester"
      type="text"
      value={role}
      onChange={(e)=>{setRole(e.target.value)}}
      />
      </div>
      </div>
      <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="img">
        Image URL

      </label>
    </div>
    <div className="md:w-2/3">
      <input className="bg-gray-200 appearance-none border-2
      border-gray-200 rounded w-full py-2 px-4 text-gray-700
      leading-tight focus:outline-none focus:bg-white
focus:border-purple-500"
      id="img"
      placeholder=
"https://www.pexels.com/photo/woman-wearing-hat-3310695/"
```

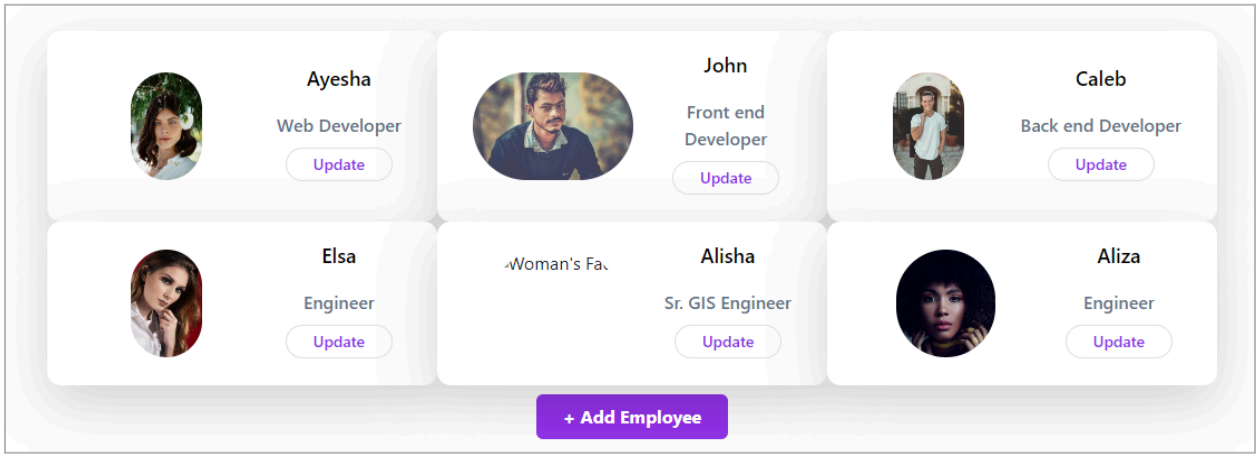
```
      type="text"
      value={img}
      onChange={(e)=>{setImg(e.target.value)}}
      />
      </div>
      </div>
           </form>
        </Modal.Body>
        <Modal.Footer>
          <button className="bg-pink-500 hover:bg-pink-700 text-white
font-bold py-2 px-4 rounded"
          onClick={handleClose}>Close</button>
           <button
           className="bg-purple-600 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"
           onClick={
            handleClose
          }
          form= "editmodal">
            Add
          </button>
        </Modal.Footer>
      </Modal>
    </>
  );
}

export default AddEmployee;
```

After running the code, we can easily add an employee to our list.

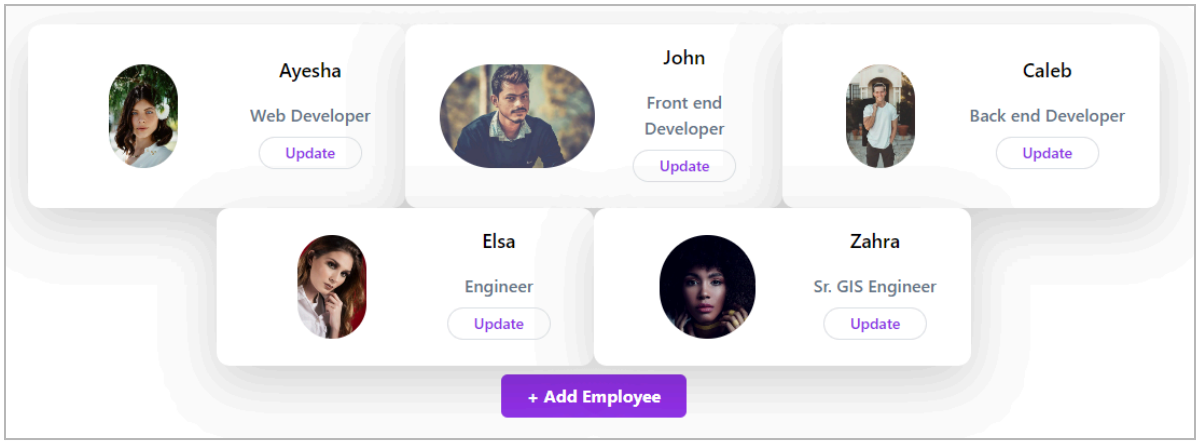


After inputting our names and roles, we must exit the dialogue box. Then, we must copy the picture address, paste it into the picture URL field, and click Add.

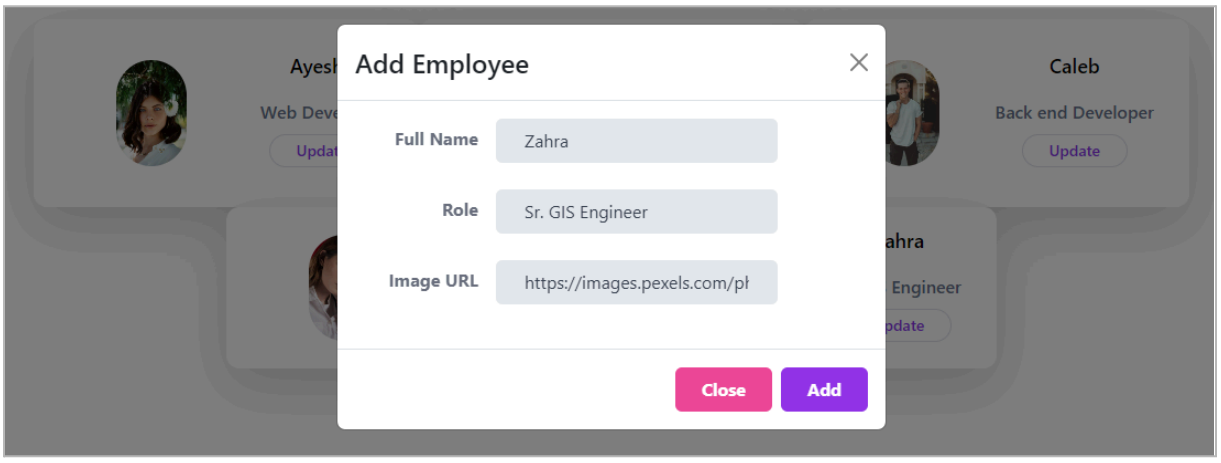
It will include our staff on the web page. However, if we enter all values simultaneously, namely name, role, and image, the image will not load. You can check the output shown below:



Here is our final output:



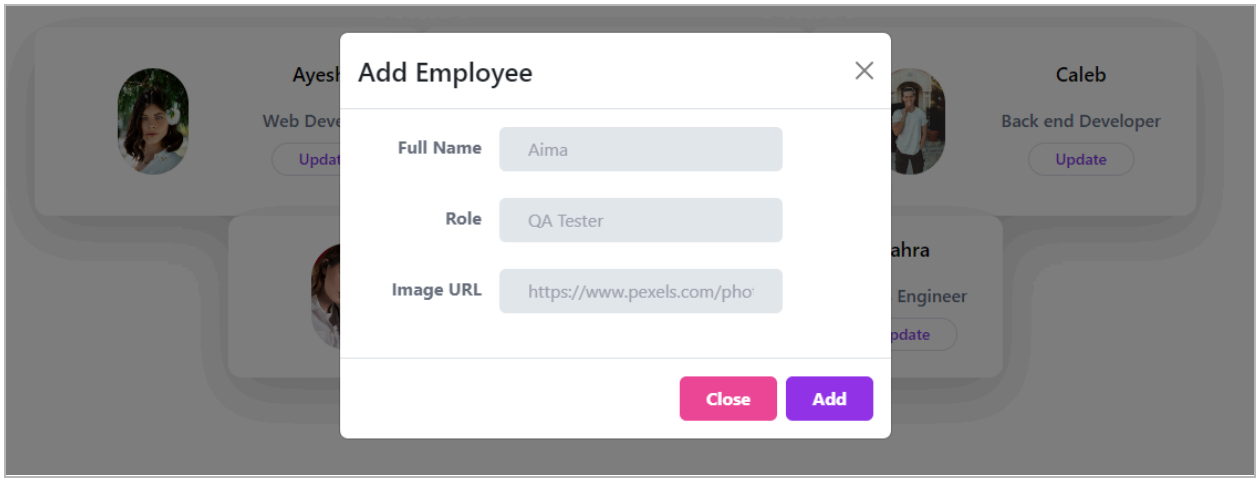
When we click on Add Employee, it shows us the values of our previously added employee. **You can see the output below:**



We must make the following changes to our AddEmployee component to set the default values. **You need to add it under the form submission part as we did here:**

```
<form
        onSubmit={(e)=>{
          e.preventDefault();
          setName('');
          setRole('');
          setImg('');
          props.newEmployee(name, role, img);

        }}
```

Here, we can see the changes:



Pass a Component to Props

In this part, we'll introduce a new method for sending data from the parent component to the child component: props. We can transfer data between components more effectively by passing them as properties.

The following are the main changes we made to the App.js component to implement this approach:

```
import './index.css';
import Employee from './components/Employee';
import { useState } from 'react';
import {v4 as uuidv4} from 'uuid';
import AddEmployee from './components/AddEmployee';
import EditEmployee from './components/EditEmployee';

function App() {
  const [role, setRole] = useState('GIS Analyst');
  const [employees, setEmployees] = useState(
    [
      { id: 1,
        name: 'Ayesha',
      role: 'Web Developer',
      img:
'https://images.pexels.com/photos/3586798/pexels-photo-3586798.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
```

```
  },
  { id: 2,
    name: 'John',
    role: 'Front end Developer',
    img:
'https://images.pexels.com/photos/694438/pexels-photo-694438.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
  },
  {
    id: 3,
    name: 'Caleb',
    role: 'Back end Developer',
    img:
'https://images.pexels.com/photos/775358/pexels-photo-775358.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
  },
  { id: 4,
    name: 'Elsa',
    role: 'Engineer',
    img:
'https://images.pexels.com/photos/3610877/pexels-photo-3610877.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
  },
  ]);

  function updateEmployee(id, newName, newRole){
    const updatedEmployees = employees.map((employee)=>{
      if (id === employee.id){

      return {...employee, name: newName, role: newRole};
      }
      return employee;
    });
    setEmployees(updatedEmployees);
```

```
  }

  function newEmployee(name, role, img) {
    const newEmployee = {
      id: uuidv4(),
      name: name,
      role: role,
      img: img,
    };
    setEmployees( [...employees, newEmployee]

    )
  }

const showEmployees = true;
return (
  <div className="App">
    {
    console.log('inside the return') }

    {showEmployees ? (
      <>
      <input type= 'text' onChange={(e) =>{
        setRole(e.target.value);
      }}/>
      <div
      className= "flex flex-wrap justify-center">
     {employees.map((employee) => {
      const editEmployee = (
      <EditEmployee id={employee.id}
      name={employee.name}
      role={employee.role}
      updateEmployee={employee.updateEmployee}/>
     );
```

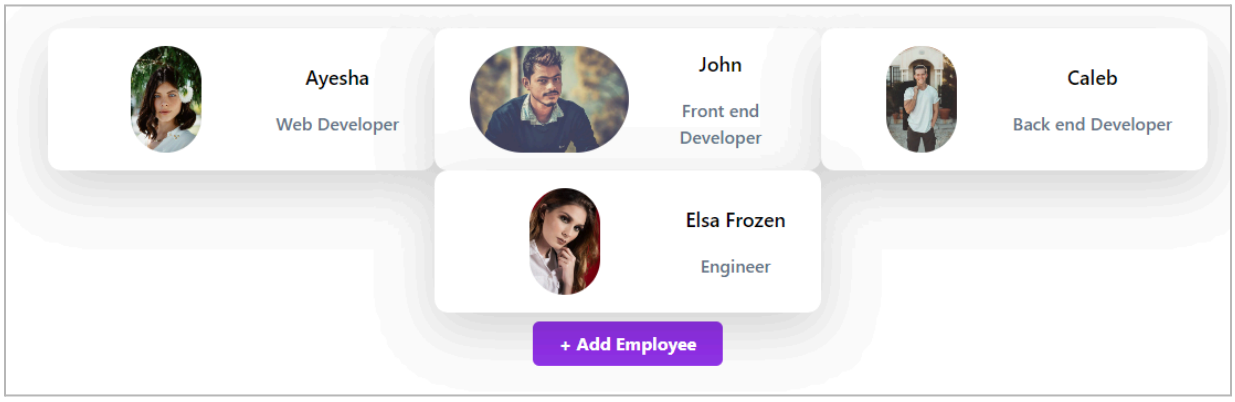
```
        return (
          <Employee
          key= {employee.id}
          id= {employee.id}
          name= {employee.name}
          role= {employee.role}
          img= {employee.img}
          editEmployee={editEmployee}
          />
        );
       })}
        </div>
        <AddEmployee newEmployee = {newEmployee}/>
        </>
       ) : (
        <p>You cannot see the Employees</p>
       )}
    </div>
  );
}

export default App;
```

Then, we also made changes to the function in the Employee.js component as follows:

```
function Employee(props) {
    return (
        <div className="min-w-[350px] max-w-[350px] py-8 px-8
max-w-sm bg-white rounded-xl shadow-lg space-y-2 sm:py-4 sm:flex
sm:items-center sm:space-y-0 sm:space-x-6">
  <img className="block mx-auto h-24 rounded-full sm:mx-0
sm:shrink-0"
  src={props.img}
```

```
 alt="Woman's Face"/>
 <div className="text-center space-y-2 sm:text-left">
  <div className="space-y-0.5">
   <p className="text-lg text-black font-semibold">{props.name}</p>
   <p className="text-slate-500 font-medium">{props.role}
   </p>
  </div>
  {props.EditEmployee}

 </div>
</div>
   );
}

export default Employee;
```

Here's how we build an application by combining several components. As you'll observe below, the result appears fairly similar to what we got with the previous method.

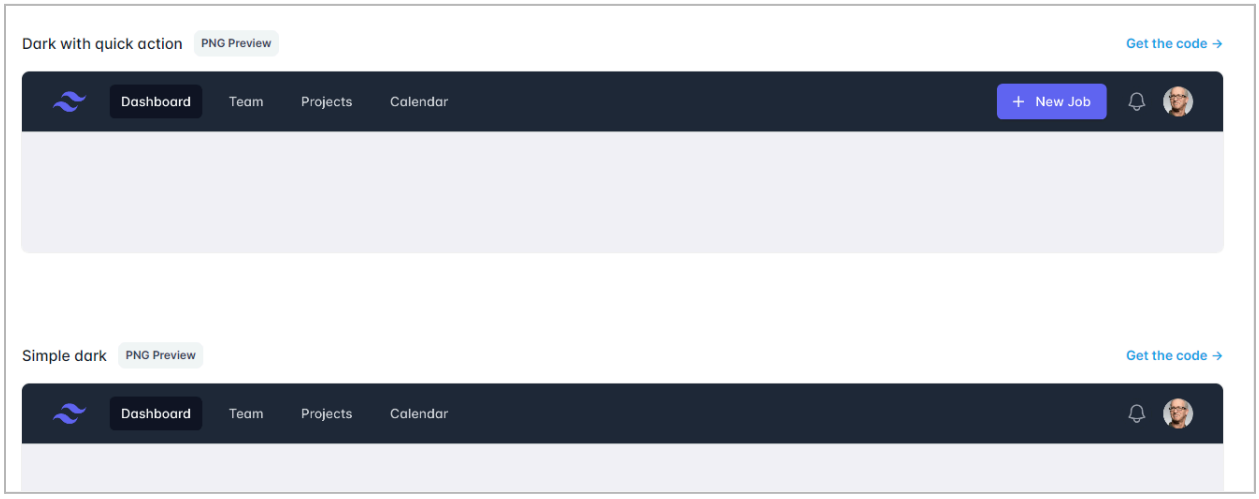


Create a Navbar with Tailwind CSS

We will create a banner for our web page in this section. First of all, we will navigate to the Tailwind navbar.The Navbar is a navigation bar, which you can access through this link:

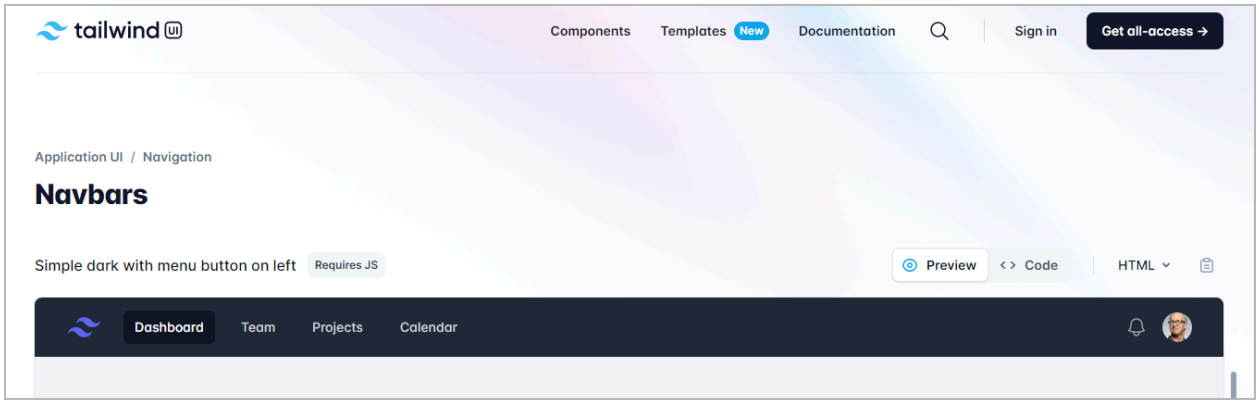
https://tailwindui.com/components/application-ui/navigation/navbars.

It's worth noting that Tailwind's official Navbar component is one of its premium features. However, in this section, we'll look at alternate ways to design a custom Tailwind-based Navbar without breaking the rules or engaging in unethical behavior. The first step is to select the Navbar design that best meets your demands.



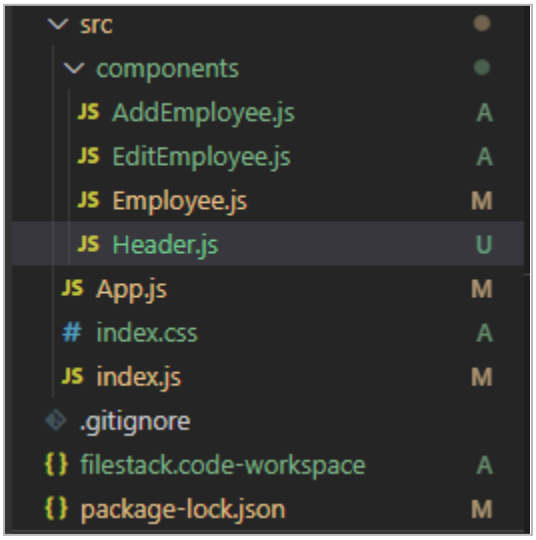
To proceed, click the 'Get the code' button in the upper right corner of each Navbar. However, keep in mind that some of these features are charged.

We've picked a free Navbar for this book, which may be at the top of the list. You can preview this Navbar and get the code by clicking the button at the top of the first bar, which confirms it's free to use.



Another great feature of Tailwind Navbars is their compatibility with React and Vue. **You can see the options in the image given below:**

In this situation, we will select React and proceed. The first step is to commit our modifications to Visual Studio Code, which will keep everything current. Following that, we will create a new component called Header.js in the components directory.



The next step is to copy the code from the Tailwind Navbar in React and paste it into Visual Studio Code:

You must change the following line in the React code taken from the Tailwind website:

```
export default function Example()
```

to

```
export default function
Header()
```

The next step is to add our Header.js in App.js. W**e will import it using the following command in App.js.**

```
import Header from
'./components/Header';
```

Then, we have to add <Header/> in the following code section of the App.js.

```
 return (
  <div className="App">
   <Header/>
```

After we run the code, the following command is shown on the screen:

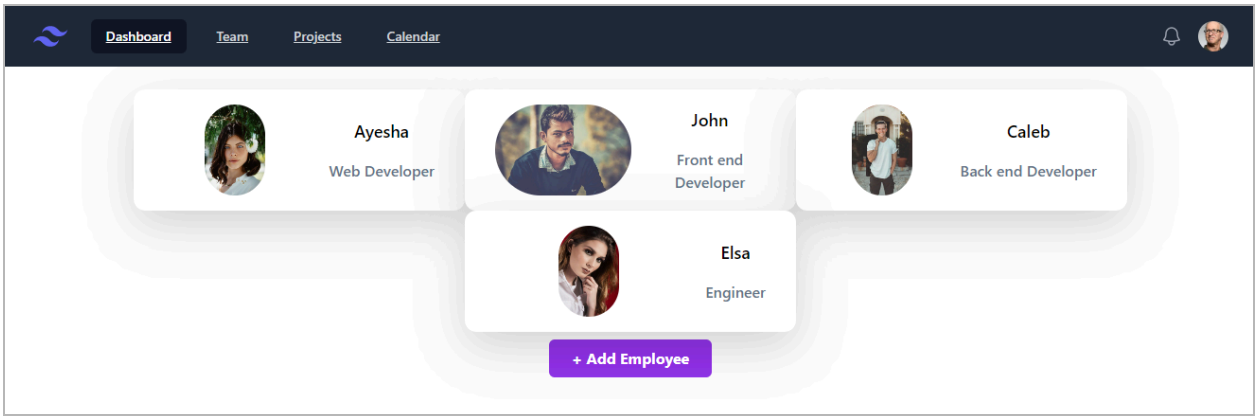
```
Compiled with problems:                                                    X

ERROR in ./src/components/Header.js 5:0-65
Module not found: Error: Can't resolve '@headlessui/react' in 'C:\Users\adullah\filestack\hello\src\components'

ERROR in ./src/components/Header.js 6:0-77
Module not found: Error: Can't resolve '@heroicons/react/24/outline' in 'C:\Users\adullah\filestack\hello\src\components'
```

We will copy @headlessui/react from the above error and then enter the following command in Visual Studio Code:

```
npm install  @headlessui/react
```

Then, we will do the same with @heroicons/react

After installing these commands, we can see that our web page is working with the banner now.

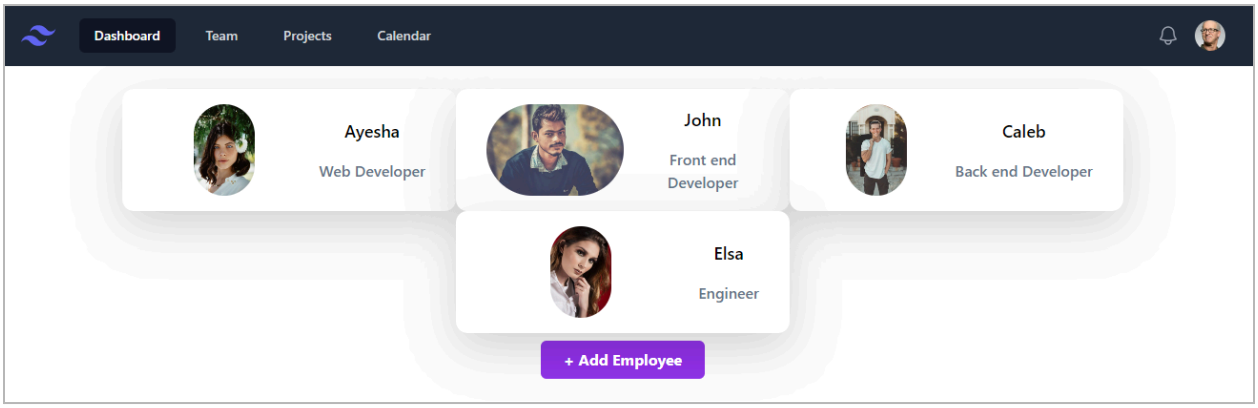


Pages and props.children

We can see in the above image that an underline is present with options in the Navbar. **We will make the following changes to our Header.js component to remove it.**

```
'no-underline bg-gray-900 text-white' :
                    'no-underline text-gray-300 hover:bg-gray-700
hover:text-white',
                    'px-3 py-2 rounded-md text-sm font-medium'
```

Now, we can see that the line is removed successfully.



Since we don't want to add any logo or profile picture in the top right corner, we will remove some code sections. **The sections that we must remove are given below:**

```
 <div className="flex flex-shrink-0 items-center"> </div>

<img
                    className="block h-8 w-auto lg:hidden"
```

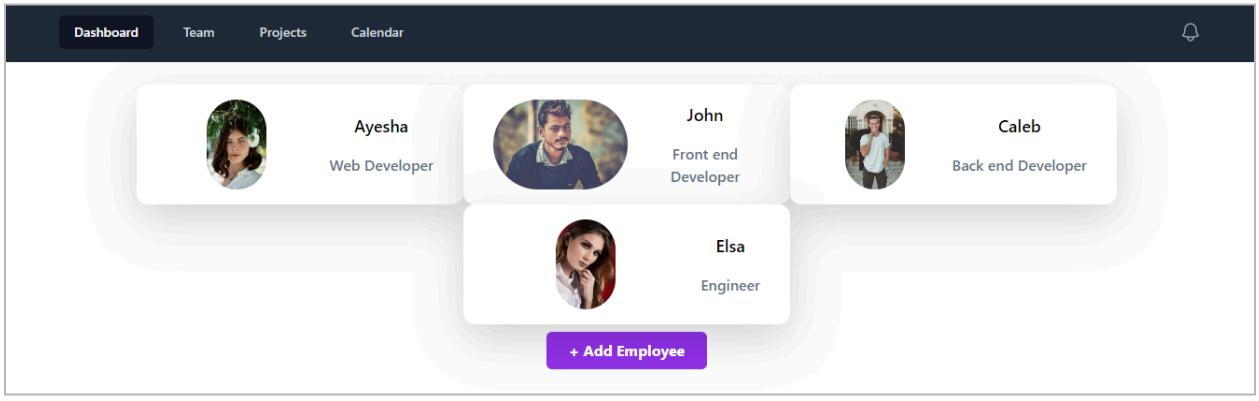
```
src="https://tailwindui.com/img/logos/mark.svg?color=indigo&shade=500"
                alt="Your Company"
              />
              <img
                className="hidden h-8 w-auto lg:block"

src="https://tailwindui.com/img/logos/mark.svg?color=indigo&shade=500"
                alt="Your Company"
              />
```

We should also remove the following code snippet:

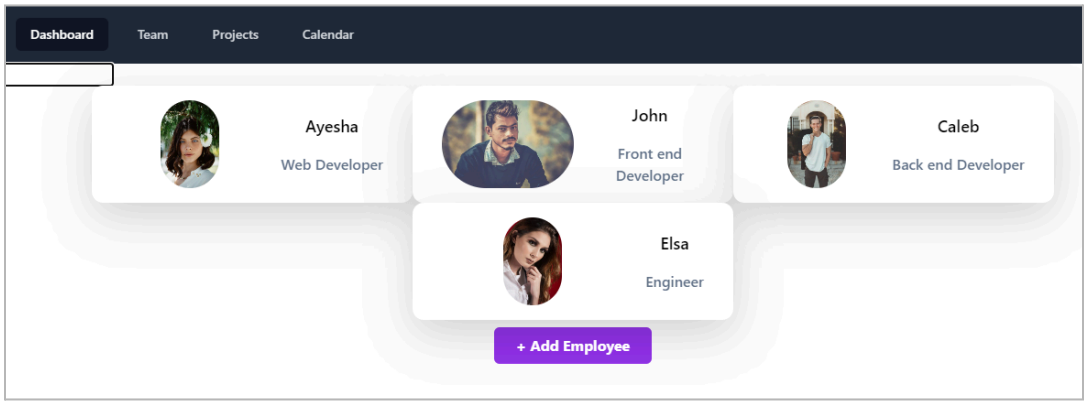
```
<img
                  className="h-8 w-8 rounded-full"

src="https://images.unsplash.com/photo-1472099645785-5658abf4ff4e?ixlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=facearea&facepad=2&w=256&h=256&q=80"
                  alt=""
                />
```

Note that these changes are made in the Header.js component.

We can see that the images are removed from our web page navigational bar:



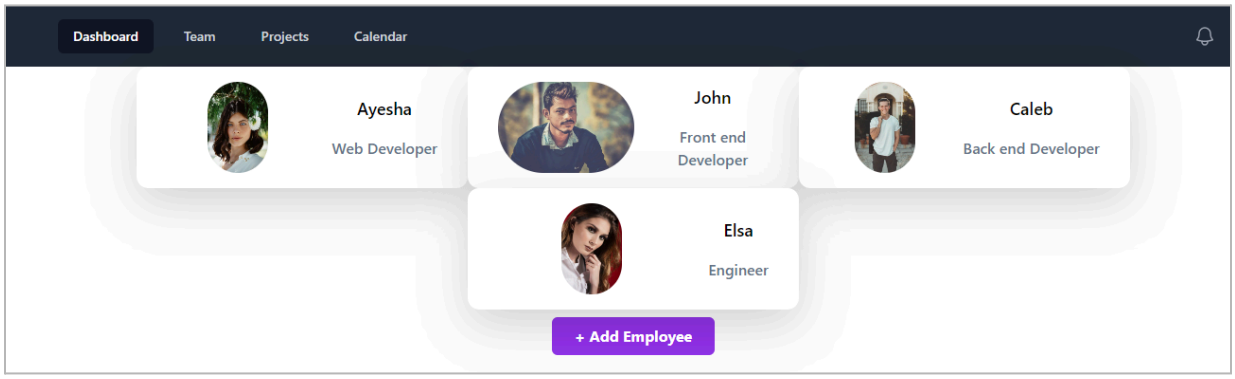
There is also an input bar at the top that we also need to remove:



We have to remove this bar from the App.js component. **You must search for input and then remove the following code section:**

```
<input type= 'text' onChange={(e) =>{
        setRole(e.target.value);
      }}/>
```

Now, we can see that there is no input bar at the top:



The next step is to reduce the size of the navigational bar. It is set to h-16, and we will make it h-14.

```
<div className="relative flex h-14 items-center justify-between">
```

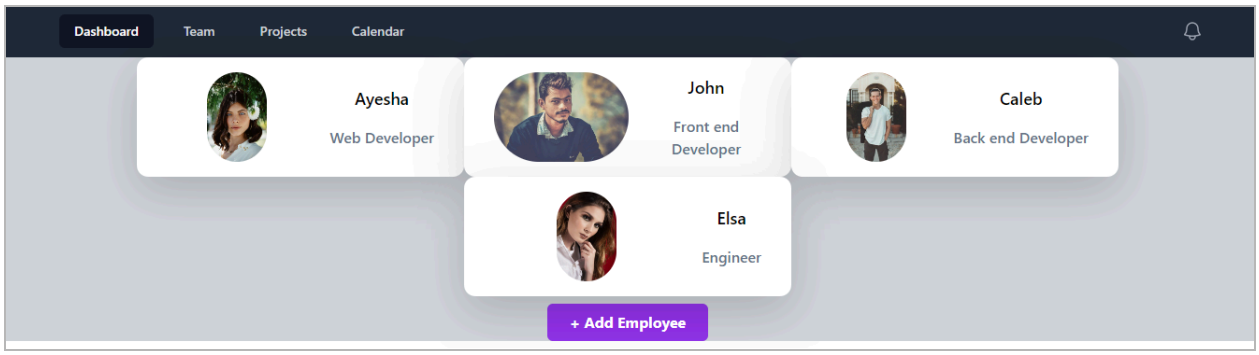
Note that the above change is performed in the Header.js component.

Now, we must change the background color of our application from the App.js

component.

```
<div className="App
bg-gray-300">
```

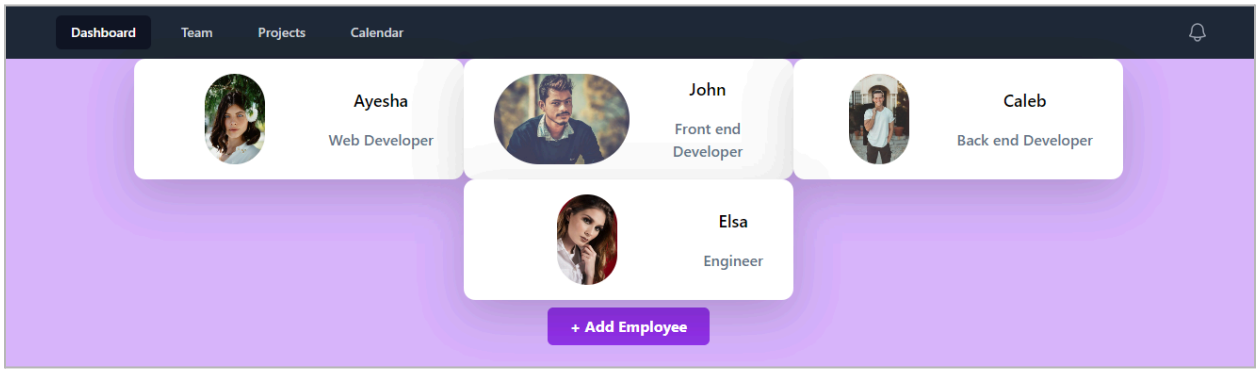
Half the web app page is filled with gray.



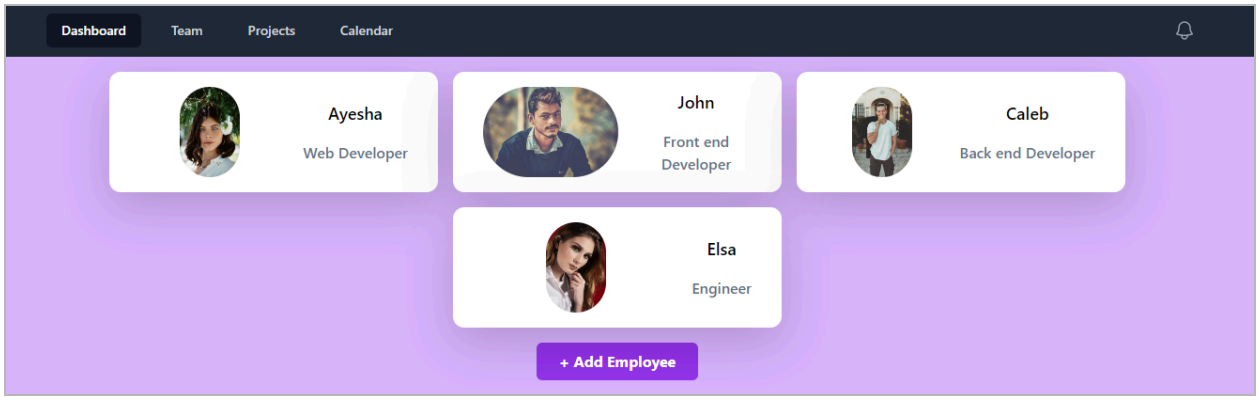
To do it, we will add min-h-screen into the above code:

```
<div className="App bg-purple-300
min-h-screen">
```

Note that we also changed the color from grey to purple. **Now, the final view is:**



To make our application look better, we added margins. **The final look is as follows:**



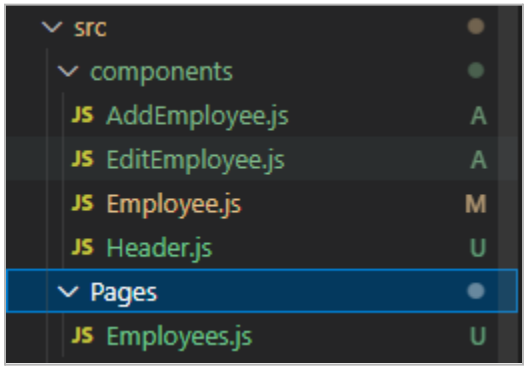
We made these changes in the Employee.js component.

```
<div className="min-w-[350px] max-w-[350px] py-8 px-8 max-w-sm bg-white rounded-xl
      shadow-lg space-y-2 sm:py-4 sm:flex sm:items-center sm:space-y-0 sm:space-x-6 my-2 mx-2">
```

Pages and Props.children

The primary goal of this section is to manage the many pages of our application while also learning how to navigate and route in React.

We begin by creating a new folder called pages and inserting a file called Employees.js. The next step is to copy the code from App.js, make the necessary changes, and paste it into Employees.js.



The changed code is as follows:

```
import '../index.css';
```

```
import Employee from '../components/Employee';
import { useState } from 'react';
import {v4 as uuidv4} from 'uuid';
import AddEmployee from '../components/AddEmployee';
import EditEmployee from '../components/EditEmployee';
import Header from '../components/Header';
function Employees() {
  const [employees, setEmployees] = useState(
    [
      { id: 1,
        name: 'Ayesha',
      role: 'Web Developer',
      img:
'https://images.pexels.com/photos/3586798/pexels-photo-3586798.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },


    { id: 2,
      name: 'John',
      role: 'Front end Developer',
      img:
'https://images.pexels.com/photos/694438/pexels-photo-694438.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
    },


    {
      id: 3,
      name: 'Caleb',
      role: 'Back end Developer',
      img:
'https://images.pexels.com/photos/775358/pexels-photo-775358.jpeg?aut
o=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
```

```
  },


  { id: 4,
    name: 'Elsa',
    role: 'Engineer',
    img:
'https://images.pexels.com/photos/3610877/pexels-photo-3610877.jpeg?
auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1',
  },
  ]);

  function updateEmployee(id, newName, newRole){
    const updatedEmployees = employees.map((employee)=>{
      if (id == employee.id){

      return {...employee, name: newName, role: newRole};
      }
      return employee;
    });
    setEmployees(updatedEmployees);
  }

  function newEmployee(name, role, img) {
    const newEmployee = {
      id: uuidv4(),
      name: name,
      role: role,
      img: img,
    };
    setEmployees( [...employees, newEmployee]

    )
  }
```

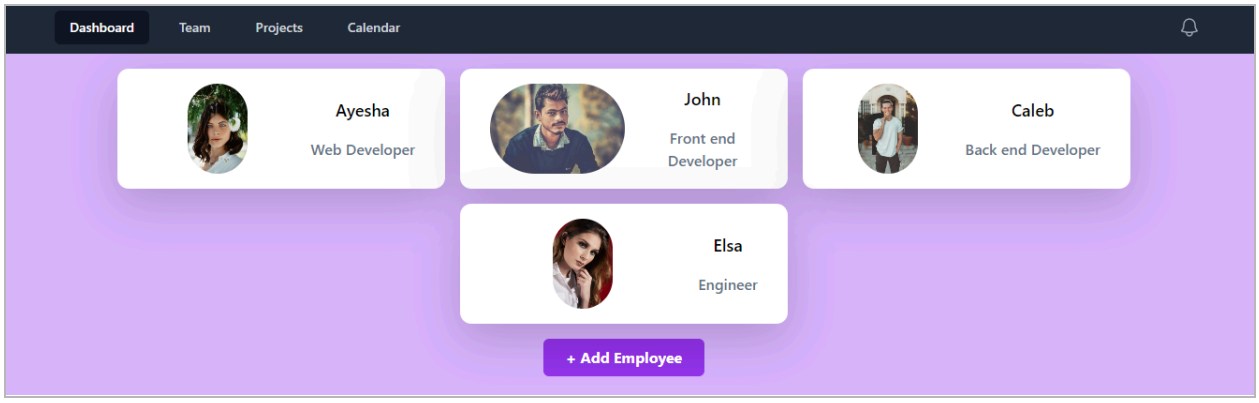
```
const showEmployees = true;
return (
  <div className="App bg-purple-300 min-h-screen">
    <Header/>
    {
    console.log('inside the return') }

    {showEmployees ? (
      <>

      <div
      className= "flex flex-wrap justify-center my-2">
     {employees.map((employee) => {
      const editEmployee = (
      <EditEmployee id={employee.id}
      name={employee.name}
      role={employee.role}
      updateEmployee={employee.updateEmployee}/>
     );
      return (
        <Employee
        key= {employee.id}
        id= {employee.id}
        name= {employee.name}
        role= {employee.role}
        img= {employee.img}
        editEmployee={editEmployee}
        />
      );
     })}
      </div>
      <AddEmployee newEmployee = {newEmployee}/>
      </>
```

```
      ) : (
       <p>You cannot see the Employees</p>
      )}
    </div>
  );
}

export default Employees;
```

The next step is to remove most of the code from App.js. We should remove all of the jsx(HTML) returned by the component and only keep our Employee component. **Now, it should look like this:**

```
import './index.css';
import Employee from './components/Employee';
import { useState } from 'react';
import {v4 as uuidv4} from 'uuid';
import AddEmployee from './components/AddEmployee';
import EditEmployee from './components/EditEmployee';
import Header from './components/Header';
function App() {


    return <Employees/>;
}

export default App;
```

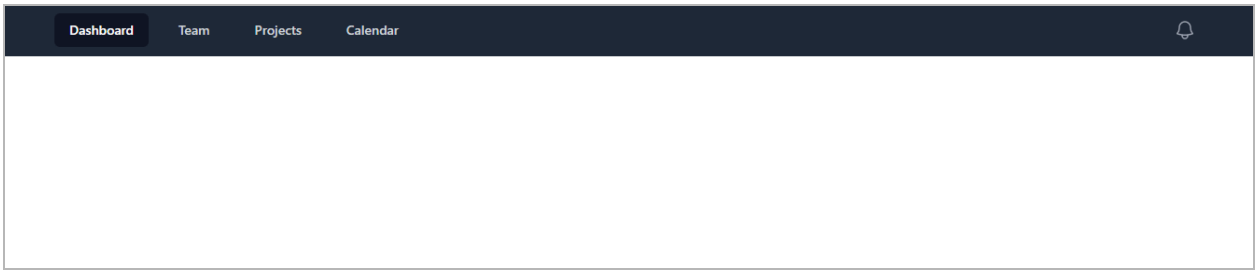
Now, returning to our application, it looks the same. **However, our code is more organized now.**



The next part is to learn how to add the header for every page. Simply put, we will surround our pages with headers. **Therefore, we will remove <Header/> from Employees.js and add it to the App.js as under:**

```
import './index.css';
import Employee from './components/Employee';
import { useState } from 'react';
import {v4 as uuidv4} from 'uuid';
import AddEmployee from './components/AddEmployee';
import EditEmployee from './components/EditEmployee';
import Header from './components/Header';
import Employees from './Pages/Employees';
function App() {
    return (
     <Header>
     <Employees/>
     </Header>
    );
}

export default App;
```

When we open the web page, we don't see anything; it should show the dashboard page. But there is a way to fix it.



To sort out this problem, we added props in the Header.js component. **You can see the example as follows:**

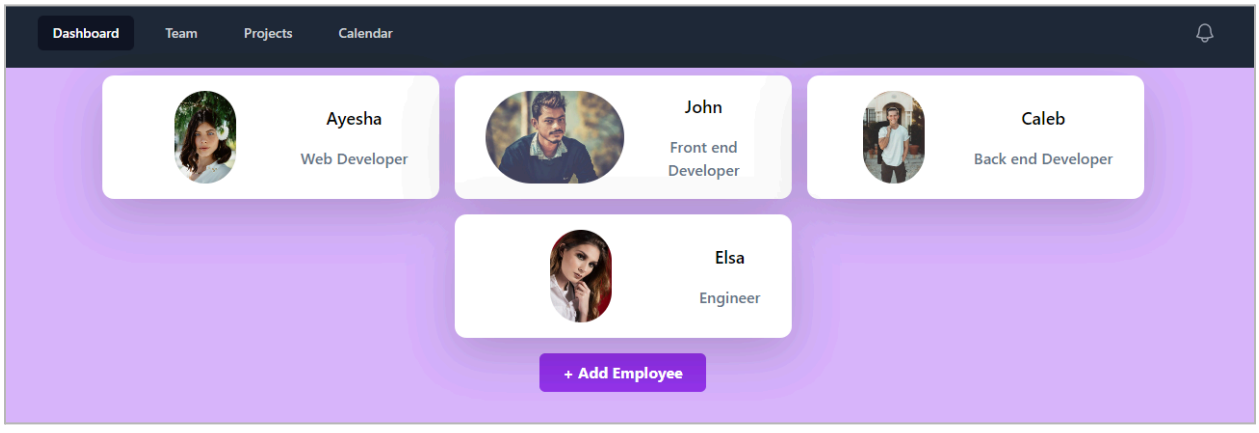
```
export default function Header({ …props
})
```

Then, we also added something at the end of the Header.js component.

```
</Disclosure.Panel>
        {props.children}
      </>
    )}
  </Disclosure>
 )
}
```

We added {props.children} to our previous code here.

We can see that our page looks the same:



Routing with React Router

We will begin the section by installing a router inside our code. **For this purpose, we will run the following command:**

```
npm install react-router-dom
```

This will install three things in our app. **Therefore, we will use the following command in the component App.js.**

```
import {BrowserRouter, Routes, Route} from
'react-router-dom';
```

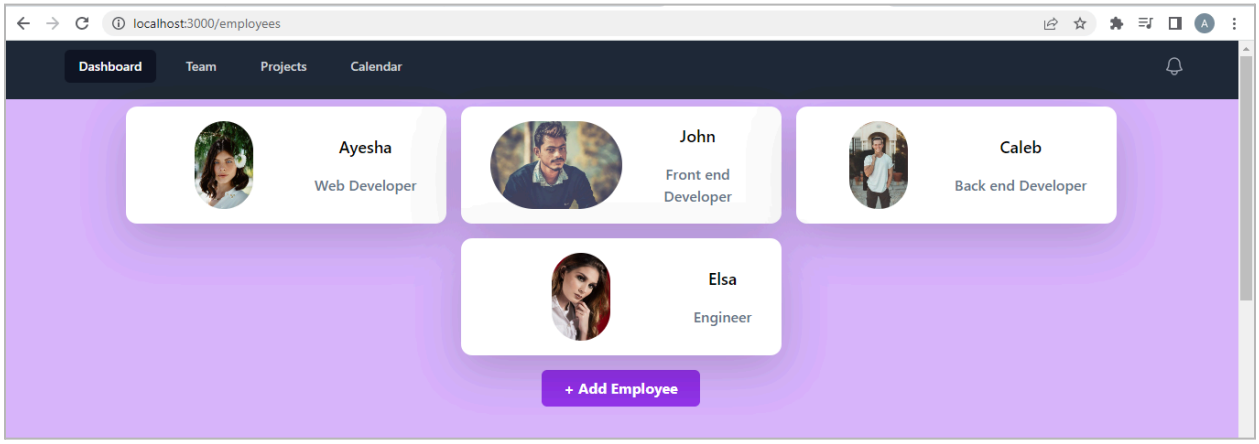
We also made a few more changes to our function in App.js. **You can see the changes as follows:**

```
import './index.css';
import Employee from './components/Employee';
import { useState } from 'react';
import {v4 as uuidv4} from 'uuid';
import AddEmployee from './components/AddEmployee';
import EditEmployee from './components/EditEmployee';
import Header from './components/Header';
import Employees from './Pages/Employees';
import {BrowserRouter, Routes, Route} from 'react-router-dom';


function App() {
    return (
     <Header>
       <BrowserRouter>
       <Routes>
         <Route path= "/employees"element={<Employees/>}/>
       </Routes>
       </BrowserRouter>
```

```
      <Employees/>
      </Header>
    );
}

export default App;
```

At this point, we must be able to visit the employees' page through the following URL: http://localhost:3000/employees

We can see that it is working now:



Let's make another page here with the name Customer.js.

Add the following code to your new component:

```
export default function Customers() {
   return <h1>Hello there!</h1>;
}
```

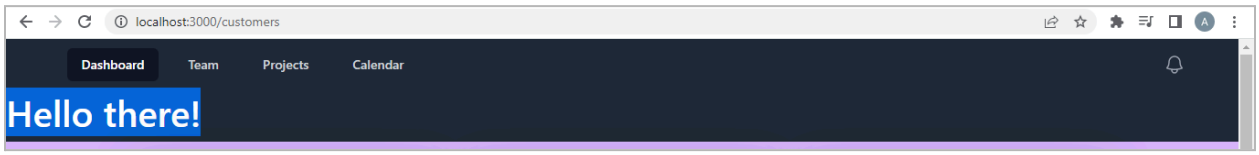
Then, import this component into the App.js using the following line:

```
import Customers from './pages/Customers';
```

Then, we must create a new route for our customers' page as follows:

```
<Route path=
"/customers"element={<Customers/>}/>
```

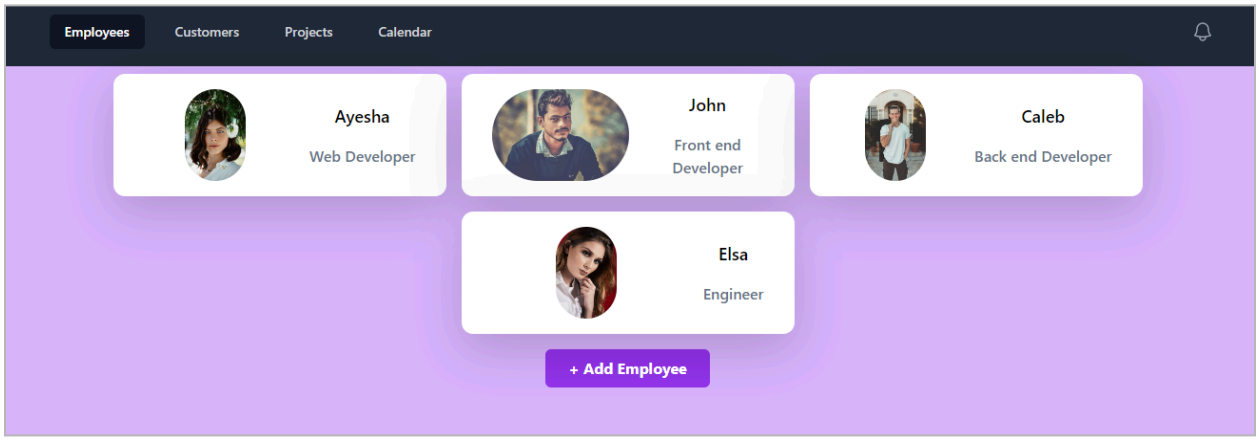
This must be added to the App.js function under the route for employees in the previous example. Now, let's check out our new page at localhost:3000. As you can see, we are getting a fresh blank page with the text "Hello there".



Let's set up the navigation so that the Dashboard tab directs us to the Employees page while the Team tab takes us to the Customers page. **We've incorporated this functionality into the Header.js code as shown below:**

```
const navigation = [
  { name: 'Employees', href: '/Employees', current: true },
  { name: 'Customers', href: '/Customers', current: false },
  { name: 'Projects', href: '#', current: false },
  { name: 'Calendar', href: '#', current: false },
]
```

You can see the above changes on our web page now.



Create an Active Page Link in the Navbar

First, we'll add a footer to our website. For this purpose, we'll go to Header.js. We'll add the footer attribute to the end of the code.

```
</Disclosure.Panel>
        {props.children}
        <footer>Example</footer>
      </>
    )}
  </Disclosure>
```

Then, we changed the following code from:

```
const navigation = [
  { name: 'Employees', href: '/Employees', current: true },
  { name: 'Customers', href: '/Customers', current: false },
  { name: 'Projects', href: '#', current: false },
  { name: 'Calendar', href: '#', current: false },
]
```

to:

```
const navigation = [
  { name: 'Employees', href: '/Employees'},
  { name: 'Customers', href: '/Customers'},
  { name: 'Projects', href: '/other'},
  { name: 'Calendar', href: '/other2'},
]
```

The next step is to change the anchor tag to the NavLink tag. **You can see the example below:**

```
<NavLink
                    key={item.name}
                    href={item.href}
                    className={classNames(
                      item.current
```

```
                    ? 'no-underline bg-gray-900 text-white' :
                    'no-underline text-gray-300 hover:bg-gray-700
hover:text-white',
                    'px-3 py-2 rounded-md text-sm font-medium'
                  )}
                  aria-current={item.current ? 'page' : undefined}
                >
                  {item.name}
                </NavLink>
```

After the above changes, we will import the NavLink from react-router-dom into the Header.js component.

```
import { NavLink } from 'react-router-dom';
```

Then, we made the following changes to our code inside the NavLink tags.

```
<NavLink
                  key={item.name}
                  href={item.href}
                { /* className={
                    classNames(
                    item.current
                    ? 'no-underline bg-gray-900 text-white' :
                    'no-underline text-gray-300 hover:bg-gray-700
hover:text-white',
                    'px-3 py-2 rounded-md text-sm font-medium'
                    )}*/}
                    className = {({isActive}) => {
                      console.log(item.href + '' + isActive)
                    }}

                >
                  {item.name}
```

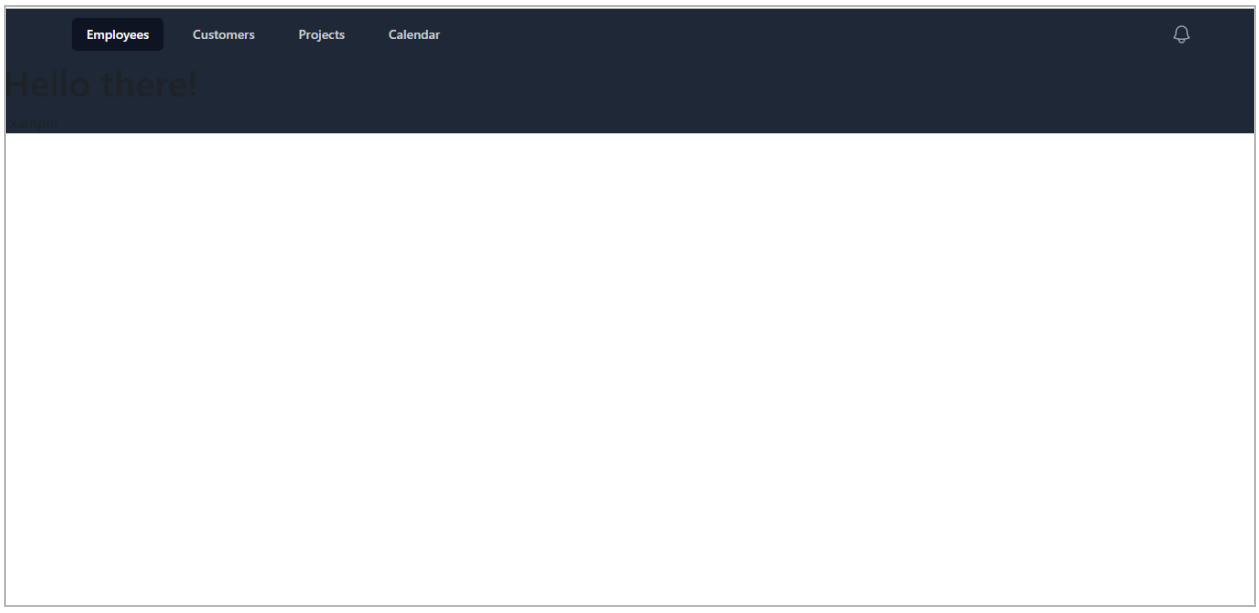
```
                </NavLink>
```

The next step is to change the App function in the App.js file.

```
function App() {
    return (

      <BrowserRouter>
       <Header>
      <Routes>
        <Route path= "/employees"element={<Employees/>}/>
        <Route path= "/customers"element={<Customers/>}/>
      </Routes>
      </BrowserRouter>
      </Header>
    );
}
```

When we view the webpage, we will see the main content for the Employees tab. However, the other tabs will display a blank page. If you check the developer console, you'll notice that the Employees tab is true, but the other tabs are false. This happens because only the Employees tab displays material, while the rest are empty.

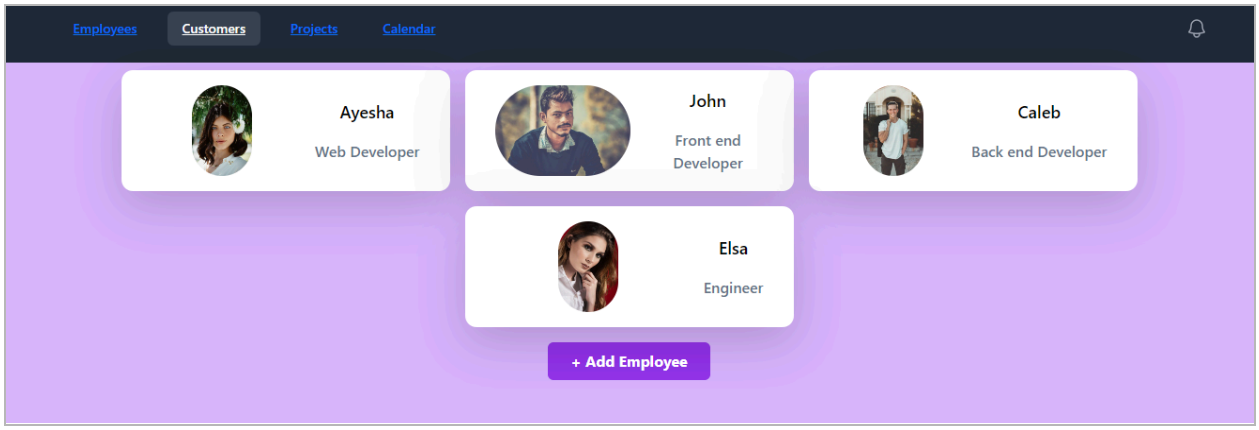
The image below shows that, although it is on the Customers page, it is inactive.



We made a few more changes to our code inside the NavLink tags to activate the opened tab.

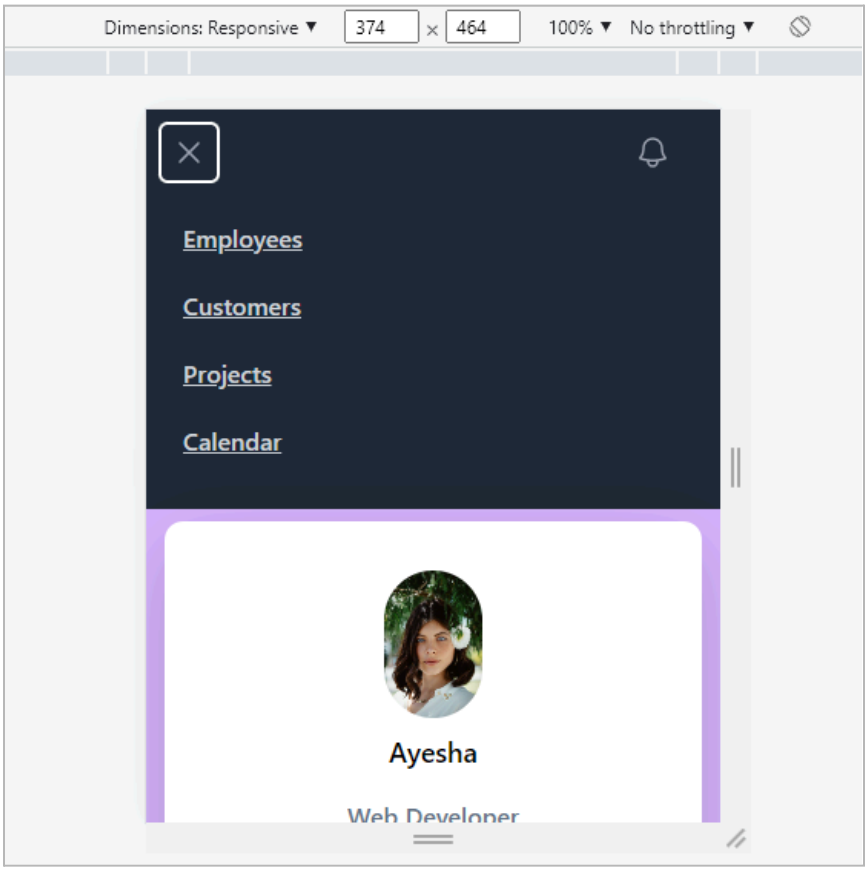
```
<NavLink
                    key={item.name}
                    to={item.href}
                      className = {({isActive}) => {
                        return (
                          'px-3 py-2 rounded-md text-sm font-medium
no-underline' +
                          (!isActive ?
                            'text-gray-300 hover:bg-gray-700 hover:text-white'
                            : 'bg-gray-900 text-white')
                          );
                      }}

                  >
                    {item.name}
                  </NavLink>
```

Now, you can see the changes in our active tabs.



Finishing up Our Header

First, we must verify that our header is clean and clear. You must have noticed one thing: the mobile version of our program. As shown in the screenshot below, the mobile version lacks features found on laptop screens.



Like the other version, the mobile version does not show clear buttons. As a result, we will also apply those styles to the mobile version. **First, we deleted the following line from our code:**

```
{/* Profile dropdown */}
            <Menu as="div" className="relative ml-3"> </Menu>
```

The next step is to remove the following section from the Header.js component.

```
<Disclosure.Button
              key={item.name}
              as="a"
              href={item.href}
              className={classNames(
                item.current ? 'bg-gray-900 text-white' : 'text-gray-300
hover:bg-gray-700 hover:text-white',
                'block px-3 py-2 rounded-md text-base font-medium'
              )}
              aria-current={item.current ? 'page' : undefined}
            > </Disclosure.Button>
```

Now, copy the following code and put it where you removed the above section:

Copy the styling properties from the Disclosure button and add it to the NavLink. **The final NavLink should look like this now:**

```
<NavLink
key={item.name}
to ={item.href}
  className = {({isActive}) => {
    return (
      'block px-3 py-2 rounded-md text-base font-medium no-underline' +
      (!isActive
        ? 'text-gray-300 hover:bg-gray-700 hover:text-white no-underline'
        : 'bg-gray-900 text-white no-underline')
      );
  }} >
```

```
{item.name}
</NavLink>
```

Remove the following section from the Header.js code:

```
 <div>
                <Menu.Button className="flex rounded-full bg-gray-800
text-sm focus:outline-none focus:ring-2 focus:ring-white
focus:ring-offset-2 focus:ring-offset-gray-800">
                  <span className="sr-only">Open user menu</span>

                </Menu.Button>
              </div>
              <Transition
                as={Fragment}
                enter="transition ease-out duration-100"
                enterFrom="transform opacity-0 scale-95"
                enterTo="transform opacity-100 scale-100"
                leave="transition ease-in duration-75"
                leaveFrom="transform opacity-100 scale-100"
                leaveTo="transform opacity-0 scale-95"
              >
                <Menu.Items className="absolute right-0 z-10 mt-2 w-48
origin-top-right rounded-md bg-white py-1 shadow-lg ring-1 ring-black
ring-opacity-5 focus:outline-none">
                  <Menu.Item>
                    {({ active }) => (
                      <a
                        href="#"
                        className={classNames(active ? 'bg-gray-100' : '',
'block px-4 py-2 text-sm text-gray-700')}
                      >
                        Your Profile
                      </a>
```

```
                )}
              </Menu.Item>
              <Menu.Item>
                {({ active }) => (
                  <a
                    href="#"
                    className={classNames(active ? 'bg-gray-100' : '',
'block px-4 py-2 text-sm text-gray-700')}
                  >
                    Settings
                  </a>
                )}
              </Menu.Item>
              <Menu.Item>
                {({ active }) => (
                  <a
                    href="#"
                    className={classNames(active ? 'bg-gray-100' : '',
'block px-4 py-2 text-sm text-gray-700')}
                  >
                    Sign out
                  </a>
                )}
              </Menu.Item>
            </Menu.Items>
          </Transition>
        </div>
      </div>
    </div>
```

The final code should look like below:

```
import { Fragment } from 'react'
import { Disclosure, Menu, Transition } from '@headlessui/react'
```

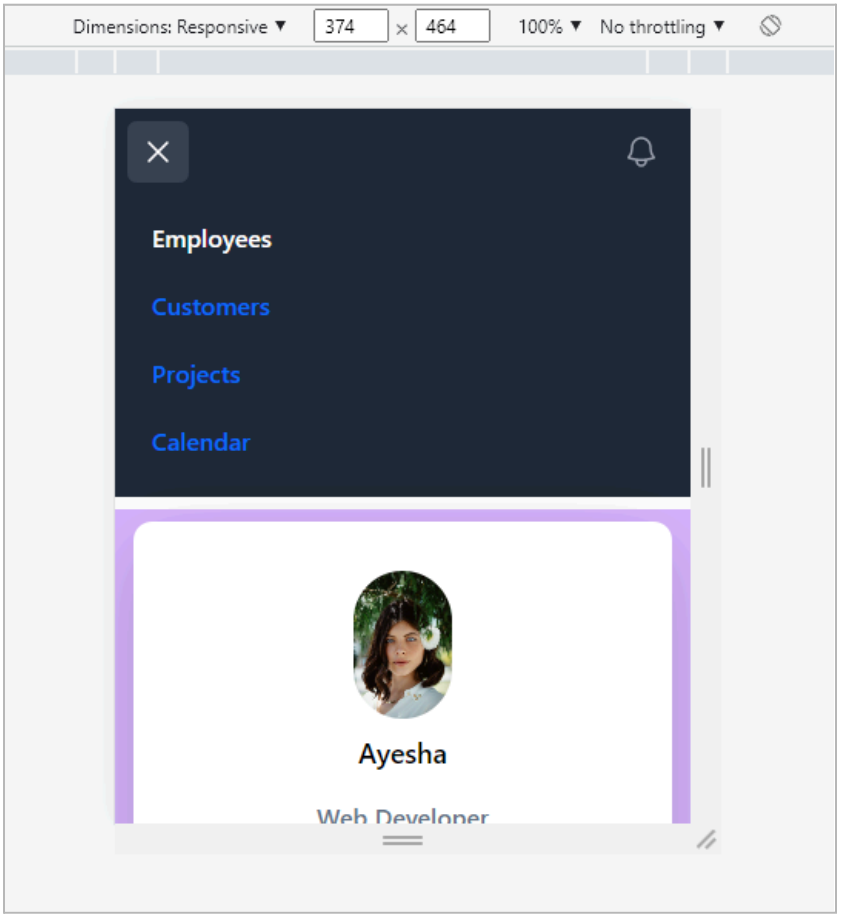
```
import { Bars3Icon, BellIcon, XMarkIcon } from
'@heroicons/react/24/outline'
import { NavLink } from 'react-router-dom'

const navigation = [
  { name: 'Employees', href: '/Employees'},
  { name: 'Customers', href: '/Customers'},
  { name: 'Projects', href: '/other'},
  { name: 'Calendar', href: '/other2'},
]

function classNames(...classes) {
  return classes.filter(Boolean).join(' ')
}

export default function Header(props) {
  return (
    <>
    <Disclosure as="nav" className="bg-gray-800">
      {({ open }) => (
        <>
          <div className="mx-auto max-w-7xl px-2 sm:px-6 lg:px-8">
            <div className="relative flex h-14 items-center justify-between">
              <div className="absolute inset-y-0 left-0 flex items-center
sm:hidden">
                {/* Mobile menu button*/}
                <Disclosure.Button className="inline-flex items-center
justify-center rounded-md p-2 text-gray-400 hover:bg-gray-700
hover:text-white focus:outline-none focus:ring-2 focus:ring-inset
focus:ring-white">
                  <span className="sr-only">Open main menu</span>
                  {open ? (
                    <XMarkIcon className="block h-6 w-6" aria-hidden="true"
/>
```

```
              ) : (
                <Bars3Icon className="block h-6 w-6" aria-hidden="true" />
              )}
            </Disclosure.Button>
          </div>
          <div className="flex flex-1 items-center justify-center
sm:items-stretch sm:justify-start">

            <div className="hidden sm:ml-6 sm:block">
              <div className="flex space-x-4">
              {  /* className={
                      classNames(
                      item.current
                      ? 'no-underline'
                      : 'no-underline'
                      ,

                      )}*/}
                {navigation.map((item) => (
                  <NavLink
                    key={item.name}
                    to ={item.href}
                      className = {({isActive}) => {
                        return (
                          'px-3 py-2 rounded-md text-sm font-medium
no-underline' +
                          (!isActive
                            ? 'text-gray-300 hover:bg-gray-700 hover:text-white
no-underline'
                            : 'bg-gray-900 text-white no-underline')
                          );
                      }}

                  >
```

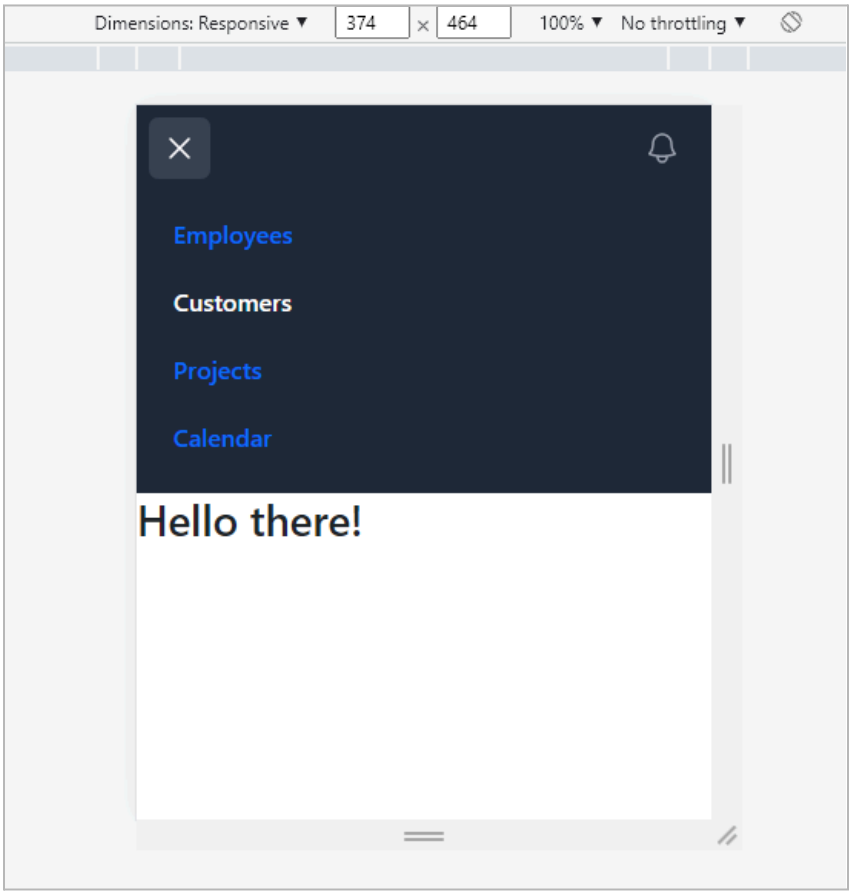
```
                    {item.name}
                  </NavLink>
                ))}
              </div>
            </div>
          </div>
          <div className="absolute inset-y-0 right-0 flex items-center pr-2
sm:static sm:inset-auto sm:ml-6 sm:pr-0">
            <button
              type="button"
              className="rounded-full bg-gray-800 p-1 text-gray-400
hover:text-white focus:outline-none focus:ring-2 focus:ring-white
focus:ring-offset-2 focus:ring-offset-gray-800"
            >
              <span className="sr-only">View notifications</span>
              <BellIcon className="h-6 w-6" aria-hidden="true" />
            </button>
</div>
</div>
</div>
        <Disclosure.Panel className="sm:hidden">
          <div className="space-y-1 px-2 pt-2 pb-3">
            {navigation.map((item) => (

<NavLink
key={item.name}
to ={item.href}
 className = {({isActive}) => {
   return (
    'block px-3 py-2 rounded-md text-base font-medium no-underline' +
    (!isActive
      ? 'text-gray-300 hover:bg-gray-700 hover:text-white no-underline'
      : 'bg-gray-900 text-white no-underline')
    );
```

```
  }} >
                {item.name}
                </NavLink>
              ))}
            </div>
          </Disclosure.Panel>
        </>
      )}
    </Disclosure>
    {props.children}
    </>
  );
}
```

You can see that the buttons in the mobile version are similar to the other version:



Now, we can see there is no padding on the Customers page.



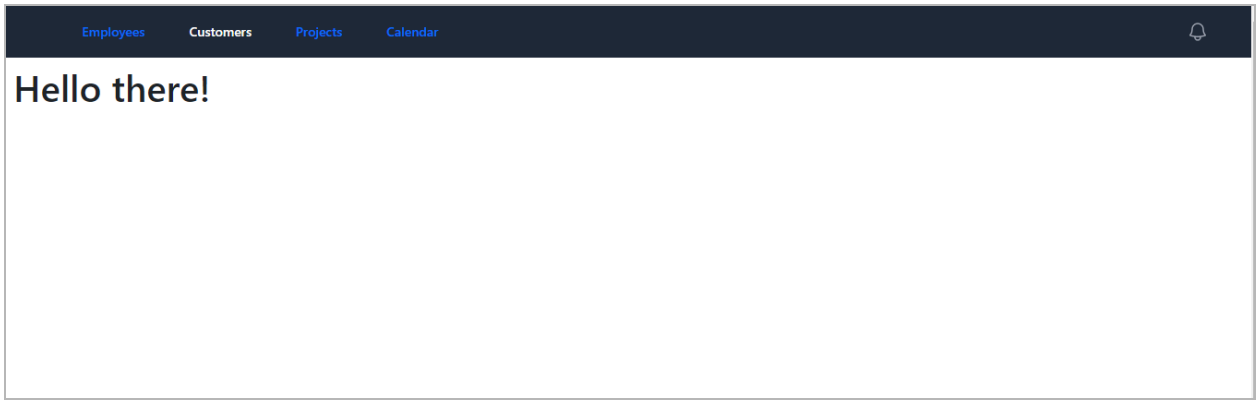
To style it, we will remove the following line of code from the Employees.js component and place it with {props.children} in the Header.js.

```
bg-purple min-screen
```

Changes made in the Header.js are given below:

```
<div className=" bg-purple min-screen px-2
py-2">{props.children}</div>
```

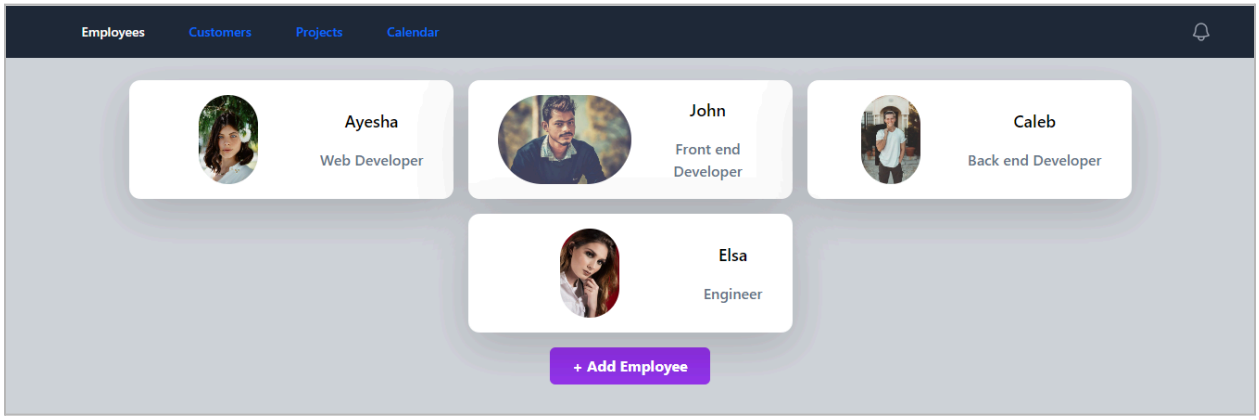
Now, you can see that there is padding on the Customers page.



We added a few more styling to the page.

```
<div className= "bg-gray-300">
   <div className="bg-gray-300 min-h-screen px-2 py-2">{props.children}</div>
   </div>
```

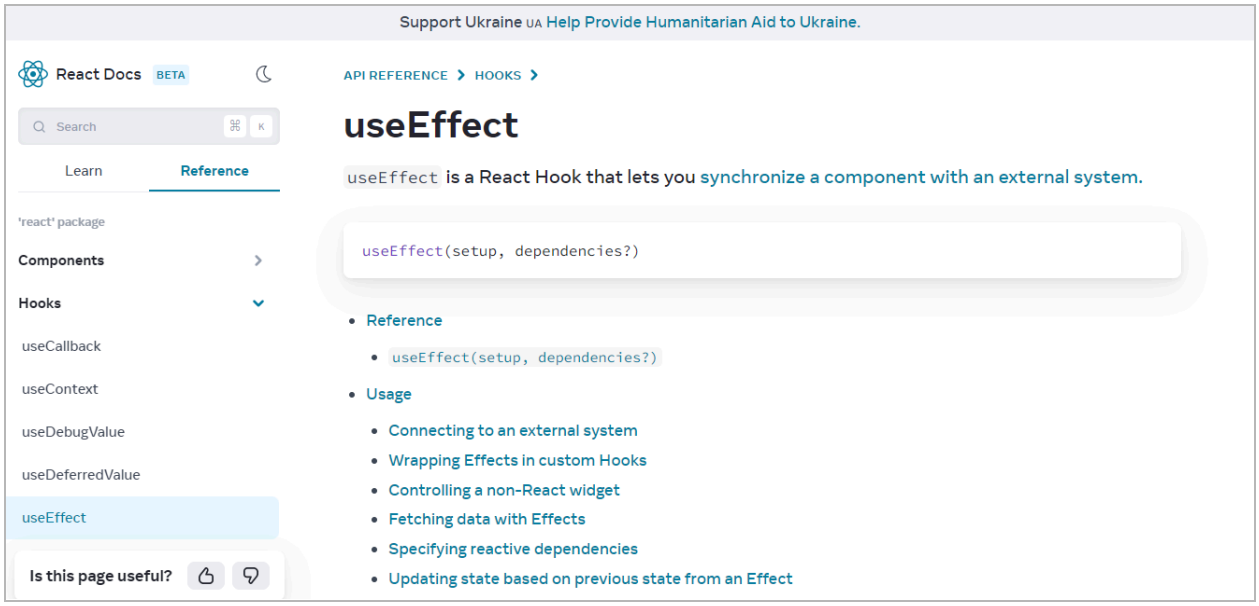
Now, it's looking like this:



Commit all the above changes to the Visual Studio Code. **Let's move to the next section.**

Intro to use Effect Hook

You can learn about the useEffect hook from the following link:
https://beta.reactjs.org/reference/react/useEffect



To use it, we are creating a new app inside our project. Under the component, we have created another component Dictionary.js.

Add the following code to the Dictionary.js component.

```
import { useState } from 'react';
export default function Dictionary()
{
  const [word, setWord] = useState('');
  return (
   <>
   <input type='text' onChange={(e) => {

       setWord(e.target.value);

   }} />
   <h1>Let's get the definition for the {word}</h1>
   </>
  );
}
```

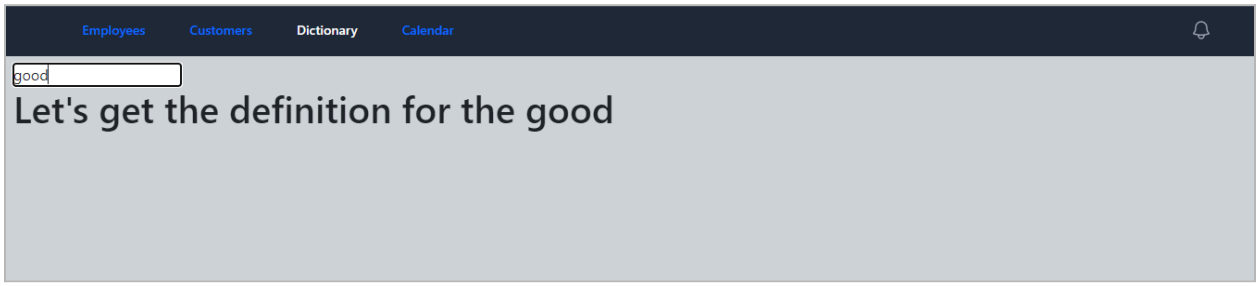
Then, add the route for the new component in the App.js as under:

```
<Route path= "/dictionary"element={<Dictionary/>}/>
```

The next step is to link to the Header.js through the following changes:

```
const navigation = [
  { name: 'Employees', href: '/Employees'},
  { name: 'Customers', href: '/Customers'},
  { name: 'Dictionary', href: '/dictionary'},
  { name: 'Calendar', href: '/other2'},
]
```

You can see that our dictionary is working now:

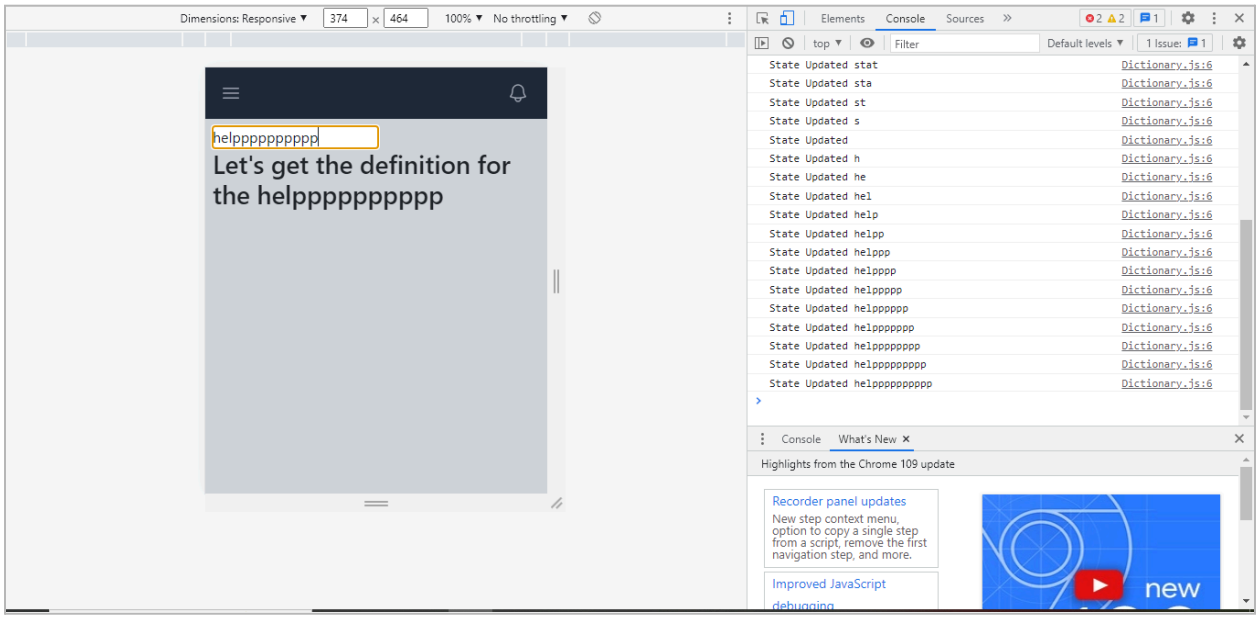


Then, we must add useEffect below:

```
import { useState, useEffect } from 'react';
export default function Dictionary()
{
   const [word, setWord] = useState();
   useEffect(() => {
    console.log('State Updated', word)
   });
   return (
    <>
    <input type="text"
    onChange={(e) => {
```

```
        setWord(e.target.value);

    }} />
    <h1>Let's get the definition for the {word}</h1>
    </>
  );
}
```

When you add any word, it will keep updating in the console log due to useEffect.



When you refresh the console, you will see only two updates.

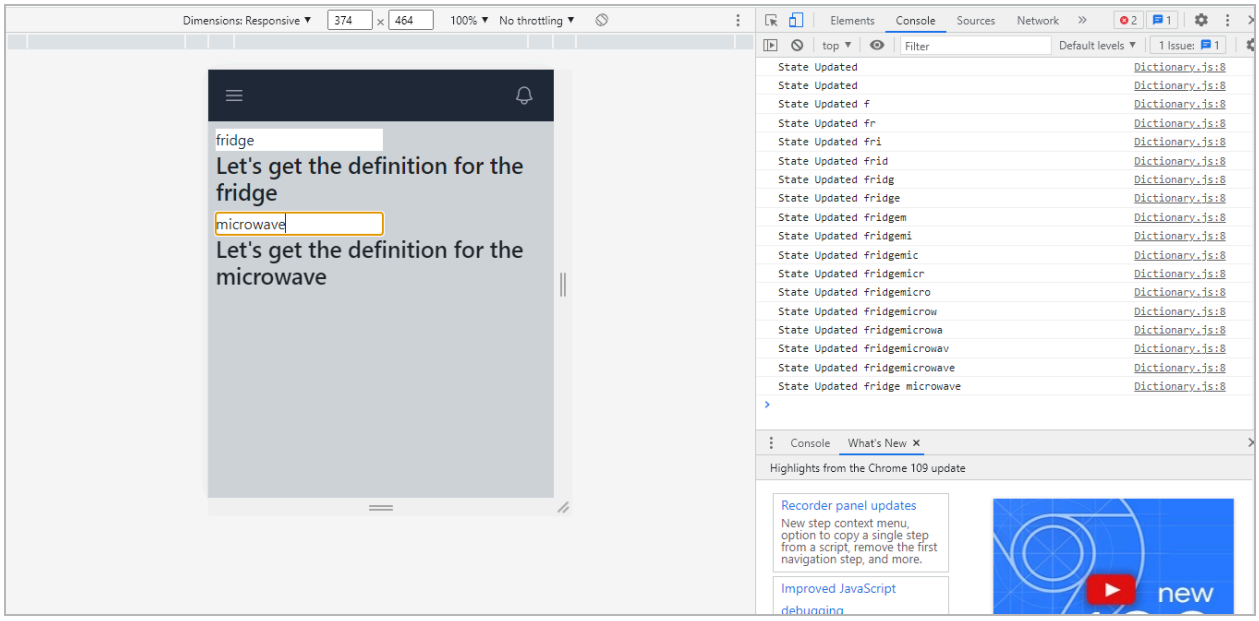
This is due to the strict mode in the index.js component. You can learn more about strict mode from there: https://reactjs.org/docs/strict-mode.html

Use Effect Dependency Array Explained

We made a few changes to our code in Dictionary.js as follows:

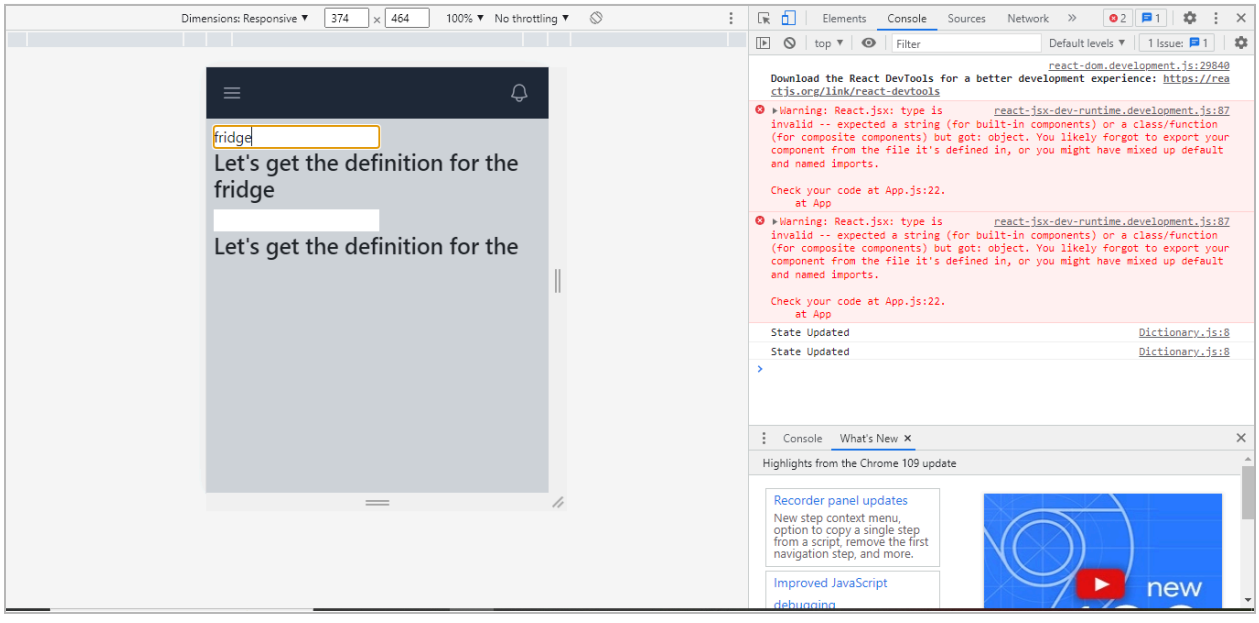
```
import { useState, useEffect } from 'react';
export default function Dictionary()
{
   const [word, setWord] = useState('');
   const [word2, setWord2] = useState('');

   useEffect(() => {
    console.log('State Updated', word + ' ' + word2)
   });
   return (
    <>
    <input type="text"
    onChange={(e) => {

        setWord(e.target.value);

    }} />
    <h2>Let's get the definition for the {word}</h2>
    <input type="text"
    onChange={(e) => {

        setWord2(e.target.value);

    }} />
    <h2>Let's get the definition for the {word2}</h2>
    </>
   );
}
```

We did not add any dependency array, and this is how it appears:



Now, when we add an empty dependency array in the above code, the console will show no changes.

```
console.log('State Updated', word + ' ' + word2)
  }, []);
```



You can use the useEffect hook several times in your React code, each with different requirements. In this scenario, we focus on certain data, and the hook is only activated

when the associated state variables change.

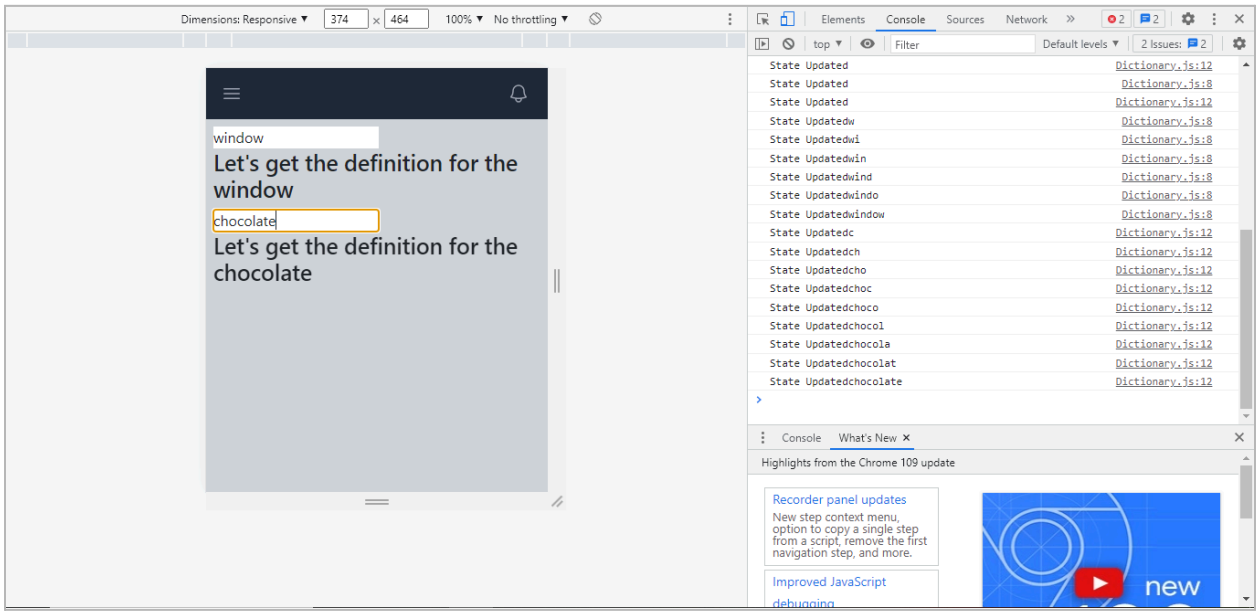
This ensures that the effect is activated in response to changes in the supplied data, keeping your component's behavior dynamic and efficient.

We made the changes to our code as follows:

```
import { useState, useEffect } from 'react';
export default function Dictionary()
{
  const [word, setWord] = useState('');
  const [word2, setWord2] = useState('');

  useEffect(() => {
   console.log('State Updated' + word)
  }, [word]);

  useEffect(() => {
   console.log('State Updated' + word2)
  }, [word2]);


  return (
   <>
   <input type="text"
   onChange={(e) => {

      setWord(e.target.value);

   }} />
   <h2>Let's get the definition for the {word}</h2>
   <input type="text"
   onChange={(e) => {

      setWord2(e.target.value);
```

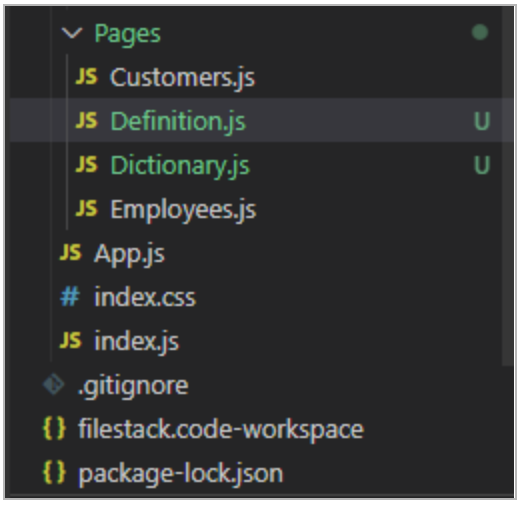
```
    }} />
    <h2>Let's get the definition for the {word2}</h2>
    </>
   );
}
```

Now, you can see two different functions being executed.



Fetch an API to Display on the Page

In this part, we'll utilize the useEffect hook to collect data from an API. We first transferred the Dictionary.js component to the pages directory and generated a new page titled Definition.js.



Then, we added the following code to our new page:

```
import {useEffect} from 'react';
export default function Definition() {

useEffect(() => {
    console.log('page loaded');
}, []);

    return <p>Here is a definition</p>;
};
```

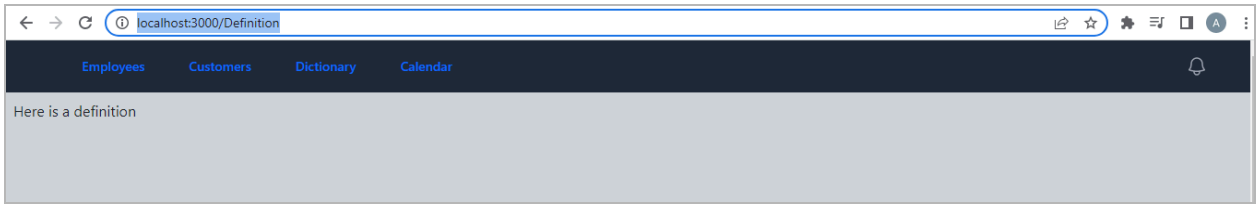
We imported it to the App.js component.

```
import Definition from './Pages/Definition';
```

Then, we added a route in the App.js.

```
<Route path= "/definition" element={<Definition/>}/>
```

Now, we can access it through http://localhost:3000/Definition and get the following results:

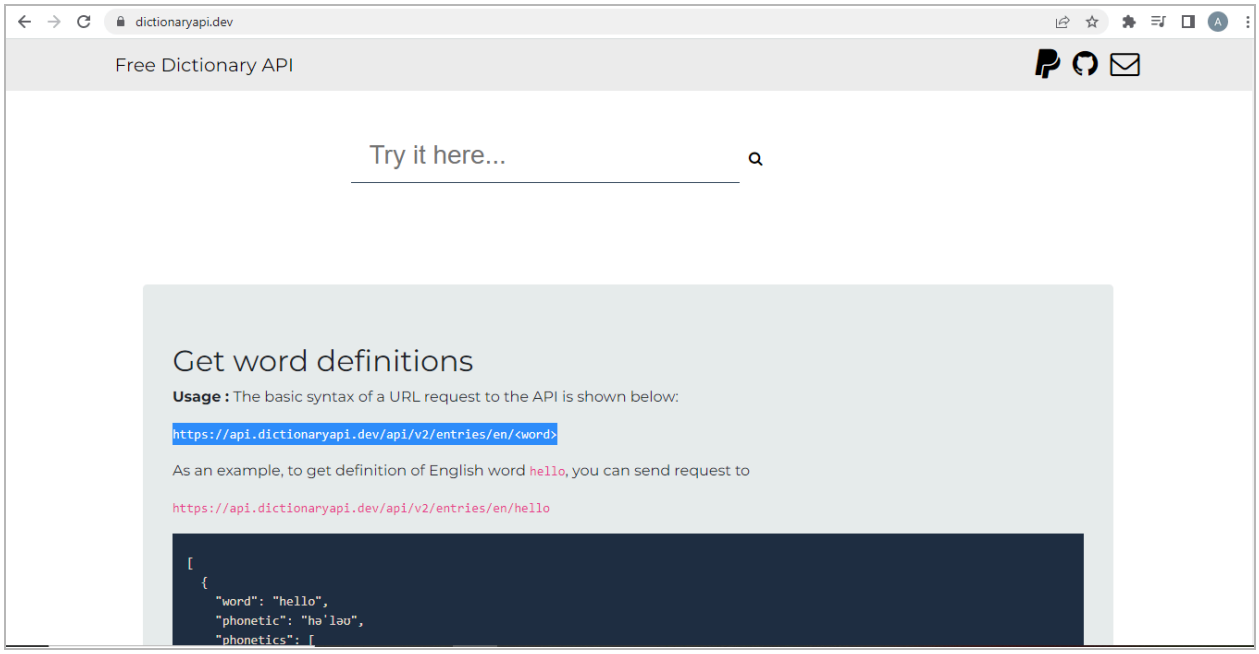


Now, we will use a free API inside our code. **We made the following changes to our code in the Definition.js.**

```
import {useState, useEffect} from 'react';
export default function Definition() {

    const[word, setWord] = useState();

useEffect(() => {
    console.log('page loaded');
}, []);

    return <p>Here is a definition</p>;
};
```

We will use Words API by RapidAPI. You can learn about it here: https://rapidapi.com/dpventures/api/wordsapi

Note that it is important to use the API key at the backend to ensure the secure use of your API key. We took the API key from a free dictionary API:



You can access it from this URL: https://dictionaryapi.dev/

But before we use it, we also need a Fetch API. You can access it from this URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

We copied the below code from the above URL:

```
fetch('http://example.com/movies.json')
  .then((response) => response.json())
  .then((data) => console.log(data));
```

We added the above code to the Definition.js as follows:

```
import {useState, useEffect} from 'react';
export default function Definition() {

    const[word, setWord] = useState();

useEffect(() => {
    fetch('http://example.com/movies.json')
  .then((response) => response.json())
  .then((data) => console.log(data));
```

```
}, []);

    return <p>Here is a definition</p>;
};
```

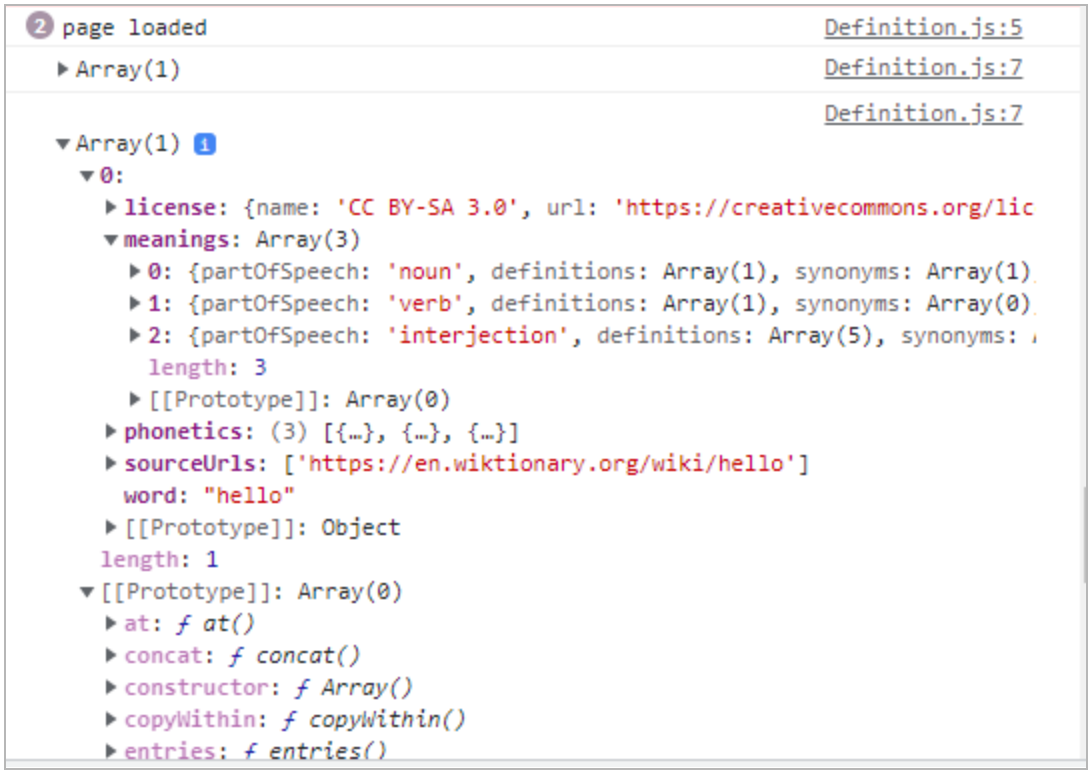
Then, we copied the following URL from the free dictionary API and added it to the above code.

https://api.dictionaryapi.dev/api/v2/entries/en/hello

The final code is as follows:

```
import {useState, useEffect} from 'react';
export default function Definition() {

    const[word, setWord] = useState();

useEffect(() => {
    fetch('https://api.dictionaryapi.dev/api/v2/entries/en/hello')
  .then((response) => response.json())
  .then((data) => console.log(data));

}, []);

    return <p>Here is a definition</p>;
};
```

When we open the console, we see the details about the word “hello”.



We made a few changes to our code as follows:

```
import {useState, useEffect} from 'react';
export default function Definition() {

    const[word, setWord] = useState();

useEffect(() => {
    fetch('https://api.dictionaryapi.dev/api/v2/entries/en/hello')
  .then((response) => response.json())
  .then((data) => console.log(data [0].meanings));

}, []);

    return <p>Here is a definition</p>;
};
```

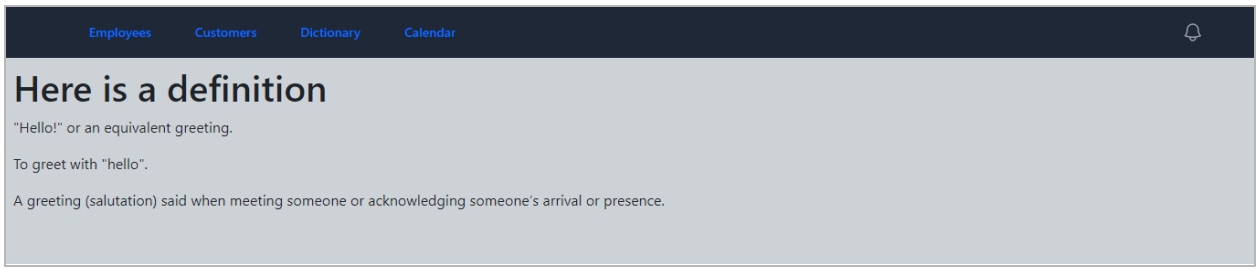
We can see the response to the above changes as follows:

```
                                                          Definition.js:7
▼(3) [{…}, {…}, {…}] i
  ▶0: {partOfSpeech: 'noun', definitions: Array(1), synonyms: Array(1), an
  ▶1: {partOfSpeech: 'verb', definitions: Array(1), synonyms: Array(0), an
  ▶2: {partOfSpeech: 'interjection', definitions: Array(5), synonyms: Arra
    length: 3
  ▶[[Prototype]]: Array(0)
```

We made some changes to our code in the Definition.js as follows:

```
import {useState, useEffect} from 'react';
export default function Definition() {

    const[word, setWord] = useState();

useEffect(() => {
    fetch('https://api.dictionaryapi.dev/api/v2/entries/en/hello')
  .then((response) => response.json())
  .then((data) => {
  setWord(data[0].meanings);
  console.log(data [0].meanings);
});

}, []);

    return (
    <>
    <h1>Here is a definition</h1>
    {word.map((meaning) => {
        return <p>{meaning.definitions[0].definition}</p>;
    })}
    </>
    );
}
```

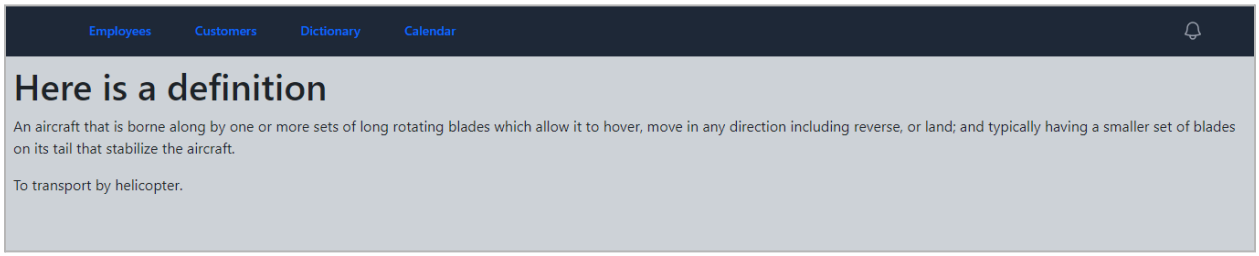
Now, you can see some information on the page:



Now, we change the word hello to helicopter in the link given inside the above code:

```
https://api.dictionaryapi.dev/api/v2/entries/en/helicopter
```

Here is our web page's response:

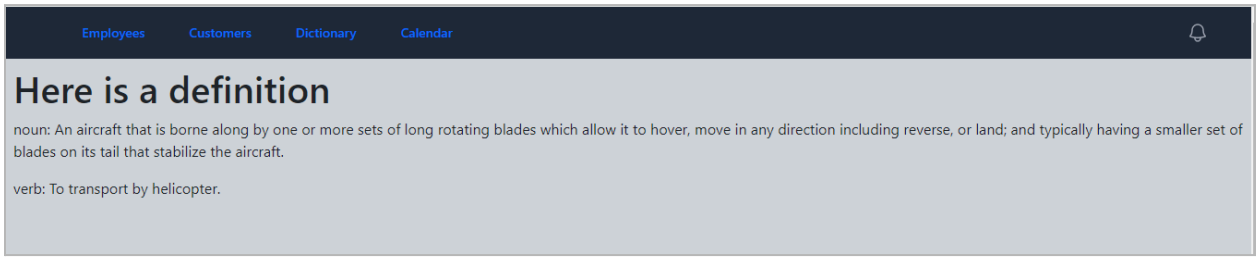


Let's add part of the speech to the information in our definitions. **We made a few more changes to our code as follows:**

```
import { useState, useEffect } from 'react';

export default function Definition() {
  const [word, setWord] = useState([]);

  useEffect(() => {
    fetch('https://api.dictionaryapi.dev/api/v2/entries/en/helicopter')
      .then((response) => response.json())
      .then((data) => {
        setWord(data[0].meanings);
        console.log(data[0].meanings);
      });
  }, []);

  return (
```

```
    <>
      <h1>Here is a definition</h1>
      {word.map((meaning, index) => (
        <p key={index}>
          {meaning.partOfSpeech}: {meaning.definitions[0].definition}
        </p>
      ))}
    </>
  );
}
```

The output is shown below:



This is how you display data from an API.

URL Parameters in Router

This section aims to write a word in the URL and get its definition. **One example is here:**

http://localhost:3000/definition/cucumber

When we want to work with the URLs, we visit the router section of our code. **This section is given in the App.js as under:**

We made the following changes to our code in the Definition.js.

```
import { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';

export default function Definition() {
  const [word, setWord] = useState([]);
  let { search } = useParams();

  useEffect(() => {
    fetch('https://api.dictionaryapi.dev/api/v2/entries/en/' + search )
      .then((response) => response.json())
      .then((data) => {
        setWord(data[0].meanings);
        console.log(data[0].meanings);
      });
  }, []);

  return (
    <>
      <h1>Here is a definition</h1>
      {word.map((meaning, index) => (
        <p key={index}>
          {meaning.partOfSpeech}: {meaning.definitions[0].definition}
```

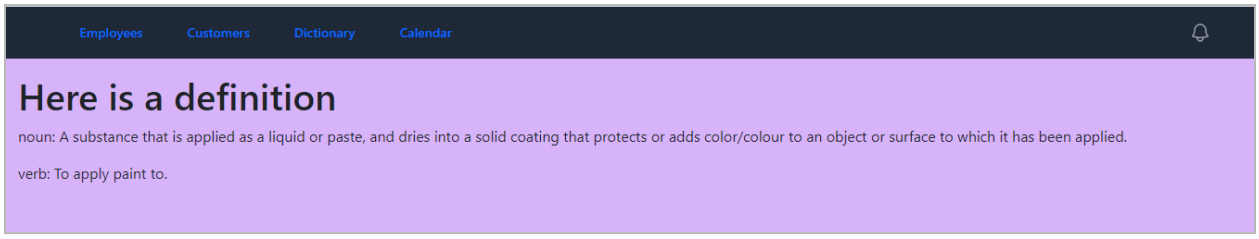
```
        </p>
      ))}
    </>
  );
}
```

We also added the following value in the App.js component.

```
<Route path= "/definition/:search "element={<Definition/>}/>
```

Now, when we enter a word in the URL, it gives us the results below:

http://localhost:3000/definition/paint



Redirect with use Navigate Hook

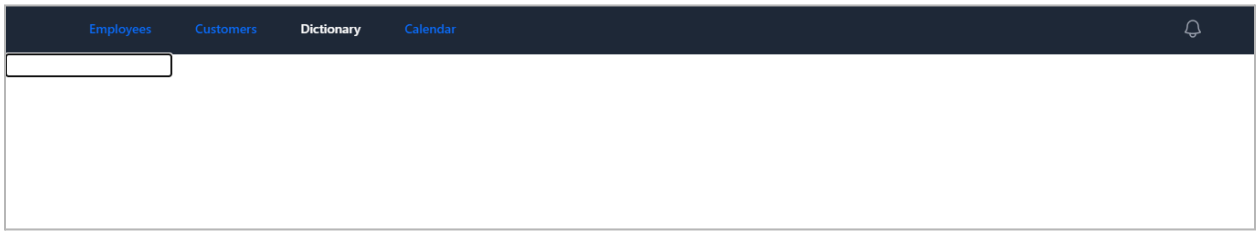
In this section, we'll design a search bar to make it easier for people to find things. **We removed the line below from the App.js component.**

```
<Route path= "/definition" element={<Definition/>}/>
```

Then, we removed the next part from the Dictionary.js component.

```
<h2>Let's get the definition for the {word}</h2>
   <input type="text"
   onChange={(e) => {

       setWord2(e.target.value);

   }} />
   <h2>Let's get the definition for the {word2}</h2>
```

Now, our page appears as below:



Then, we will also remove the following part:

```
const [word2, setWord2] = useState('');
```

```
useEffect(() => {
 console.log('State Updated' + word)
}, [word]);

useEffect(() => {
 console.log('State Updated' + word2)
}, [word2]);
```

Our code should look like this now:

```
import { useState, useEffect } from 'react';
export default function Dictionary()
{
  const [word, setWord] = useState('');

  return (
   <div className="bg-purple-300 min-h-screen px-3 py-3">
   <>
   <input type="text"
   onChange={(e) => {

       setWord(e.target.value);
```

```
    }} />

    </>
    </div>
   );
}
```

The output appears as follows:

Then run the following command:

```
npm install react-router-dom
```

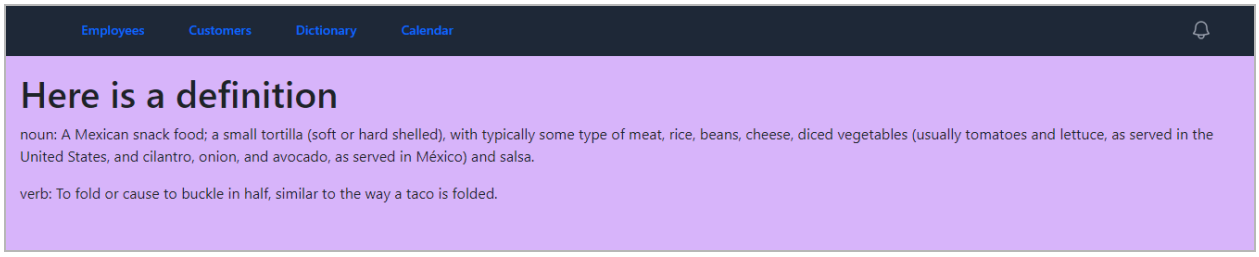
The next step is to add the following updated code to our Dictionary.js component.

```
import { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
export default function Dictionary()
{
   const [word, setWord] = useState('');
   const navigate = useNavigate();

   return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
    <>
    <input type="text"
    onChange={(e) => {

        setWord(e.target.value);

    }} />
```

```
    <button onClick ={() => {
        navigate('/definition/tacos');
    }}
    >
        Search </button>

    </>
    </div>
   );
}
```

It will give us the following output:

We get the following output When we click the “Search” button:



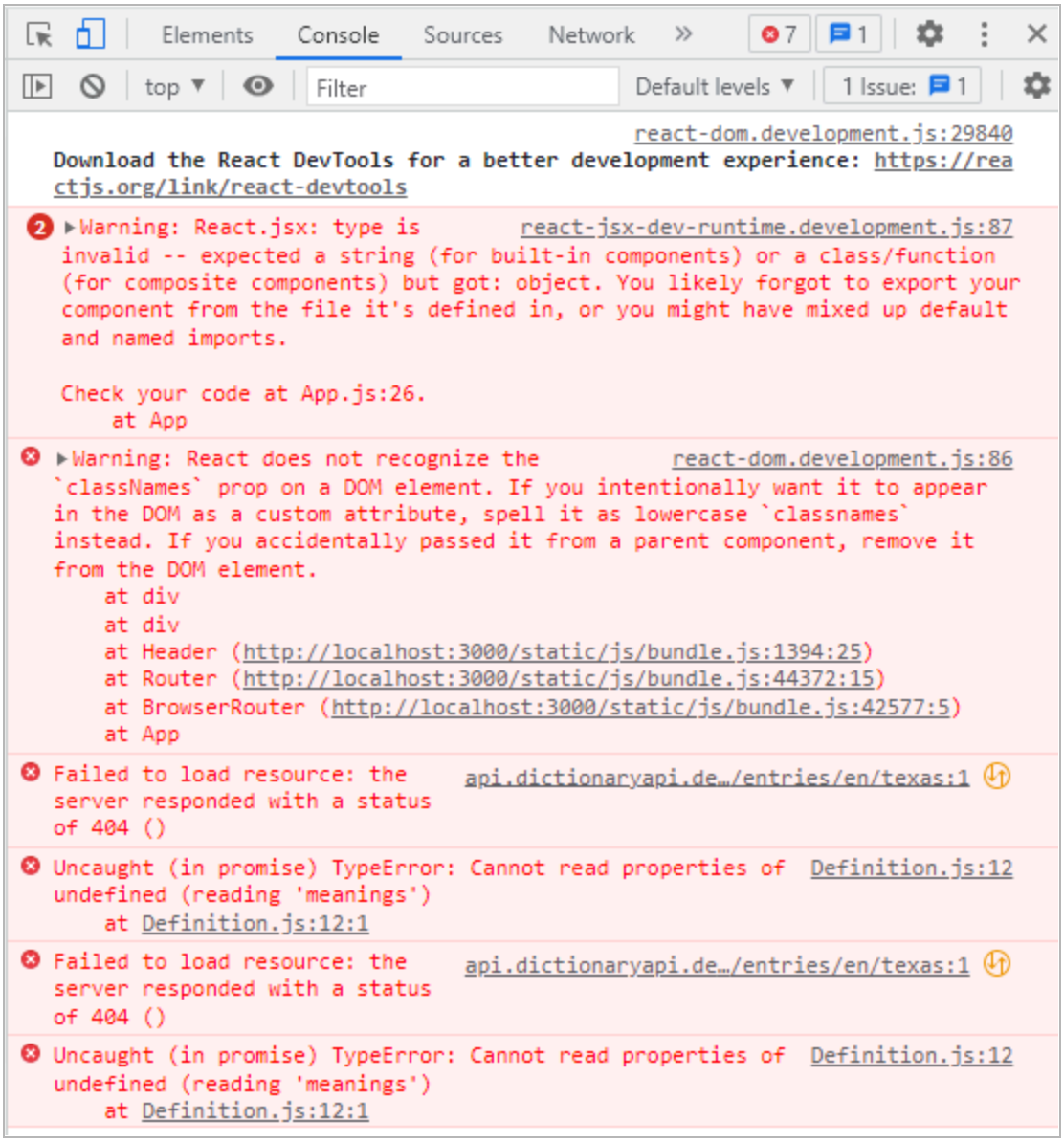
We obtain the results for the words we entered into our code. But what if we want to enter any word into the Search bar and see the results? This is quite simple. Y**ou only need to make the following changes to onClick in your code.**

```
<button onClick ={() => {
        navigate('/definition/' + word);
    }}
    >
        Search </button>
```

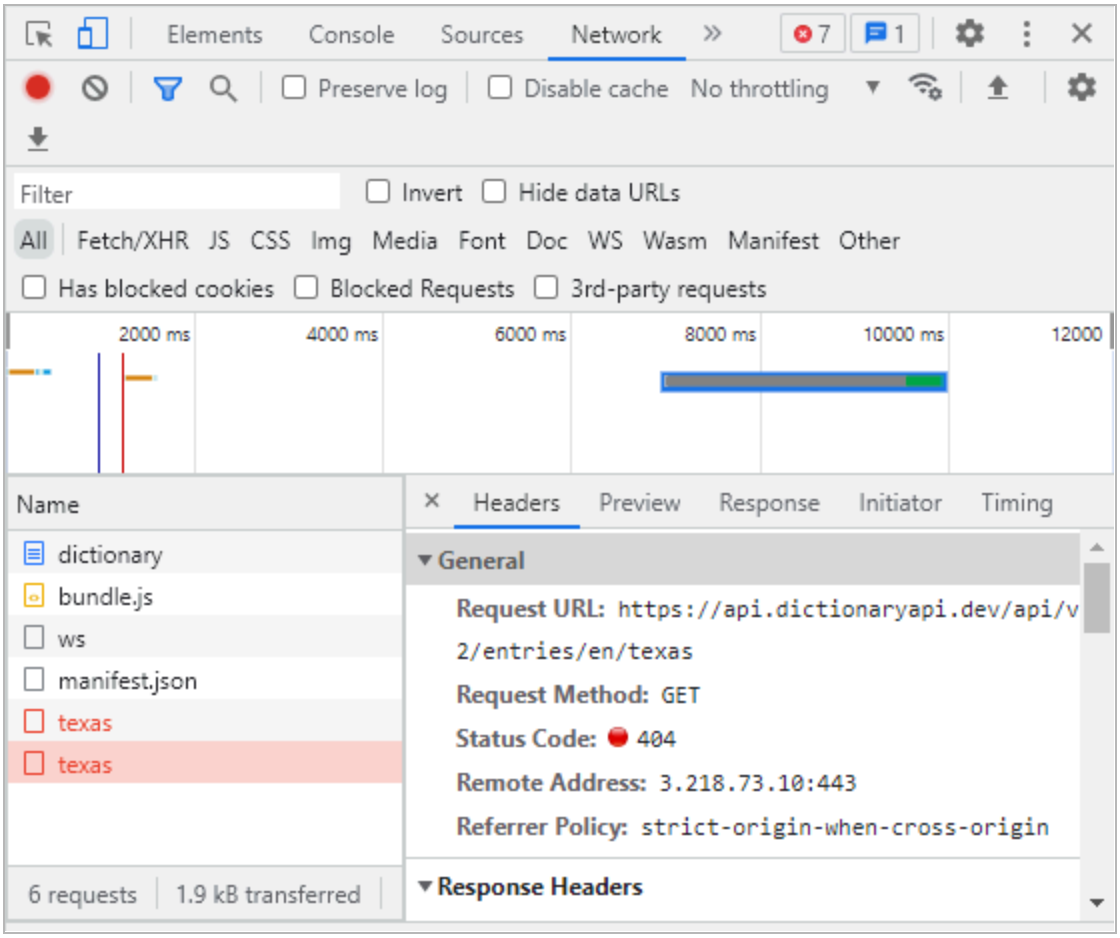
Now, we can enter the word in the search bar and get the results as above.

We have noticed one more thing here. **When we enter a word that is not found in the words map, it gives us a blank page like the one below:**

This is the error that we get in the console. It shows that the word is undefined.



The error occurs due to 404, which means that the resource wasn't found.



Now, let us discuss about history. When we hit return, we should be taken to the definition page, not the dictionary page. Here's how we can accomplish it.

We will make the following code changes in the Dictionary.js component:

```
<button onClick ={() => {
      navigate('/definition/' + word, {replace : true });
  }}
  >
      Search </button>
```

Now, we come to the definition page when we hit back. Create a 404 (Not Found) Page

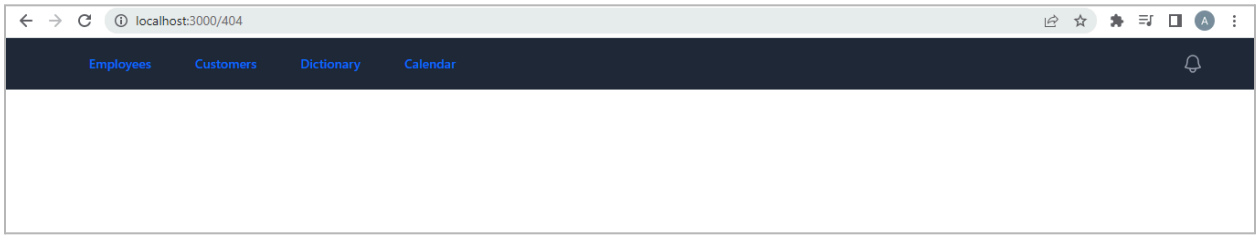
First of all, we made the following changes to the Definition.js component.

```
import { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
```

```
export default function Definition() {
  const [meanings, setMeanings] = useState([]);
  const { search } = useParams();
  const navigate = useNavigate();

  useEffect(() => {
    fetch(`https://api.dictionaryapi.dev/api/v2/entries/en/${search}`)
      .then((response) => {
        if (response.status === 404) {
          navigate('/404');
        } else {
          return response.json();
        }
      })
      .then((data) => {
        setMeanings(data[0]?.meanings);
      })
      .catch((error) => {
        console.error(error);
      });
  }, [search, navigate]);

  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
      {meanings.length > 0 ? (
        <>
          <h1>Definition:</h1>
          {meanings.map((meaning, index) => (
            <p key={index}>
              {meaning.partOfSpeech}: {meaning.definitions[0].definition}
            </p>
          ))}
        </>
```

```
      ) : (
        <p>No definition found for "{search}"</p>
      )}
    </div>
  );
}
```

It gives us a 404 page when no definition is found for our word as under:



Now, we will make a 404 component. We created a new component, NotFound.js, under the folder components. **Then, we added the following code to it:**

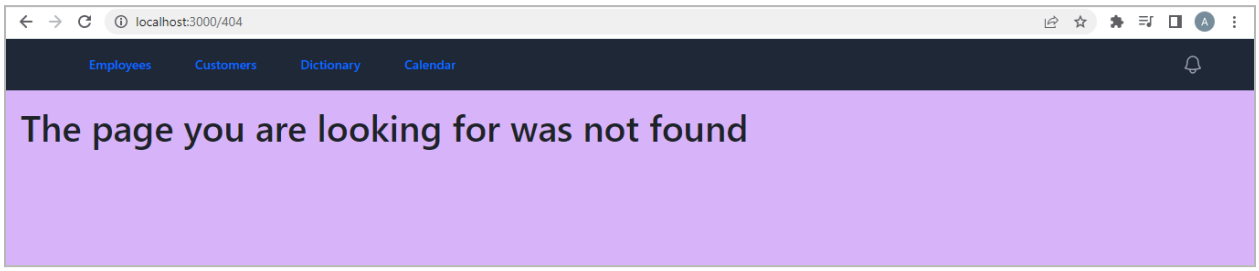
```
export default function NotFound() {
return (
<div className="bg-purple-300 min-h-screen px-3 py-3">
<h1>The page you are looking for was not found</h1>
</div>
);
}
```

The next step was to import it into the App.js and create a route. **You can see the changes below code:**

```
import './index.css';
import Employee from './components/Employee';
import { useState } from 'react';
import { v4 as uuidv4 } from 'uuid';
import AddEmployee from './components/AddEmployee';
import EditEmployee from './components/EditEmployee';
import Header from './components/Header';
import Employees from './Pages/Employees';
```

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Customers from './Pages/Customers';
import Dictionary from './Pages/Dictionary';
import Definition from './Pages/Definition';
import NotFound from './components/NotFound';

function App() {
  return (

    <BrowserRouter>
      <Header />
      <Routes>
        <Route path="/employees" element={<Employees />} />
        <Route path= "/dictionary" element={<Dictionary/>}/>
        <Route path= "/definition" element={<Definition/>}/>
        <Route path= "/404" element={<NotFound/>}/>
        <Route path= "/definition/:search"
        element={<Definition/>}
        />
        <Route path="/customers" element={<Customers />} />
      </Routes>
    </BrowserRouter>
  );
}
```

The output is shown as follows:



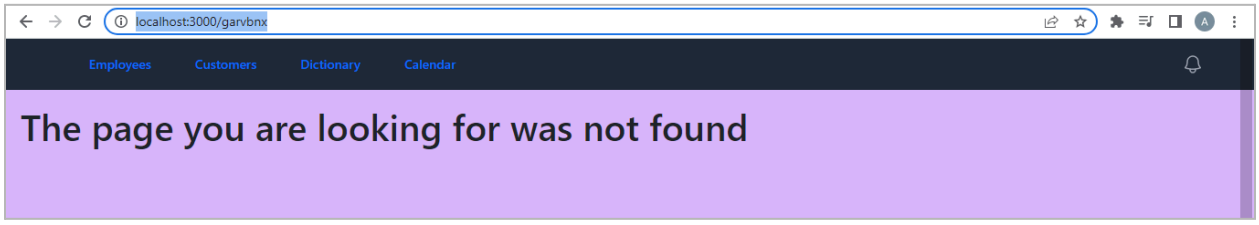
If we enter anything wrong, it gives us a blank page. But we want to appear as above.
To make it happen, we will enter the following route:

```
<Route path= "*"element={<NotFound/>}/>
```

Now, if we enter some random URL such as the one given below:

```
http://localhost:3000/garvbnx
```

It gives us the following output:



We make a few more changes to our code as follows:

```
import { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import NotFoundComponent from '../components/NotFound';

export default function Definition() {
  const [meanings, setMeanings] = useState([]);
  const [notFound, setNotFound] = useState(false);
  const { search } = useParams();
  const navigate = useNavigate();

  useEffect(() => {
    fetch(`https://api.dictionaryapi.dev/api/v2/entries/en/${search}`)
      .then((response) => {
        if (response.status === 404) {
          setNotFound(true);
        } else {
          return response.json();
        }
      })
      .then((data) => {
        if (data && data.length > 0) {
          setMeanings(data[0].meanings);
        } else {
          setNotFound(true);
```

```
      }
    })
    .catch((error) => {
      console.error(error);
    });
  }, [search, navigate]);

  if (notFound === true) {
    return <NotFoundComponent />;
  }

  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
      {meanings.length > 0 && (
        <>
          <h1>Definition:</h1>
          {meanings.map((meaning, index) => (
            <p key={index}>
              {meaning.partOfSpeech}: {meaning.definitions[0].definition}
            </p>
          ))}
        </>
      )}
    </div>
  );
}
```

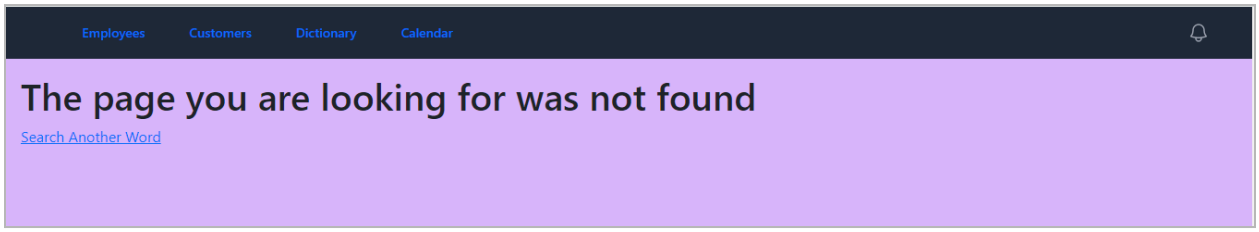
When we enter a term that the dictionary does not recognize, we will get a 404 error page. In addition, we have updated the component so that if a term is not found, it will offer the option to search for another word.

```
import { useState, useEffect } from 'react';
import { useParams, useNavigate, Link } from 'react-router-dom';
import NotFoundComponent from '../components/NotFound';

export default function Definition() {
  const [meanings, setMeanings] = useState([]);
  const [notFound, setNotFound] = useState(false);
```

```
const { search } = useParams();
const navigate = useNavigate();

useEffect(() => {
  fetch(`https://api.dictionaryapi.dev/api/v2/entries/en/${search}`)
    .then((response) => {
      if (response.status === 404) {
        setNotFound(true);
      } else {
        return response.json();
      }
    })
    .then((data) => {
      if (data && data.length > 0) {
        setMeanings(data[0].meanings);
      } else {
        setNotFound(true);
      }
    })
    .catch((error) => {
      console.error(error);
    });
}, [search, navigate]);

if (notFound === true) {
  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
    <>
      <NotFoundComponent/>
      <Link to='/dictionary'>Search Another Word</Link>
    </>
    </div>
  );
}

return (
  <div className="bg-purple-300 min-h-screen px-3 py-3">
    {meanings.length > 0 && (
      <>
```

```
        <h1>Definition:</h1>
        {meanings.map((meaning, index) => (
          <p key={index}>
            {meaning.partOfSpeech}: {meaning.definitions[0].definition}
          </p>
        ))}
      </>
    )}
  </div>
 );
}
```

Here is the output:



Fetch Response Status Codes and Errors

You can get some information about status codes here:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Now, we implemented some errors and changes in our code in the Definition.js.

```
import { useState, useEffect } from 'react';
import { useParams, useNavigate, Link } from 'react-router-dom';
import NotFoundComponent from '../components/NotFound';

export default function Definition() {
  const [meanings, setMeanings] = useState([]);
  const [notFound, setNotFound] = useState(false);
  const { search } = useParams();
  const navigate = useNavigate();

  useEffect(() => {
```

```
  const url = 'http://httpstat.us/401';
  //const url =  `https://api.dictionaryapi.dev/api/v2/entries/en/${search}`;
  fetch(url)
    .then((response) => {
      if (response.status === 404) {
        setNotFound(true);
      }
      else if (response.status === 401){
        navigate('/login')
      }
        else if (response.status === 500){
          //setServerError(true)
        }
        return response.json();
    })
    .then((data) => {
      if (data && data.length > 0) {
        setMeanings(data[0].meanings);
      } else {
        setNotFound(true);
      }
    })
    .catch((error) => {
      console.error(error);
    });
}, [search, navigate]);

if (notFound === true) {
  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
    <>
      <NotFoundComponent/>
      <Link to='/dictionary'>Search Another Word</Link>
    </>
```

```
      </div>
    );
  }

  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
      {meanings.length > 0 && (
        <>
          <h1>Definition:</h1>
          {meanings.map((meaning, index) => (
            <p key={index}>
              {meaning.partOfSpeech}: {meaning.definitions[0].definition}
            </p>
          ))}
        </>
      )}
    </div>
  );
}
```

If we enter the 401 error into the URL in our code, it directs us to the login page. It means that it is working fine.

We made more changes to our web page to give an error page when you enter the wrong URL.

```
import { useState, useEffect } from 'react';
import { useParams, useNavigate, Link } from 'react-router-dom';
import NotFoundComponent from '../components/NotFound';

export default function Definition() {
```

```
const [meanings, setMeanings] = useState([]);
const [notFound, setNotFound] = useState(false);
const [error, setError] = useState(false);


const { search } = useParams();
const navigate = useNavigate();

useEffect(() => {
  const url = 'http://httpstat.us/501';
  //const url =  `https://api.dictionaryapi.dev/api/v2/entries/en/${search}`;
  fetch(url)
    .then((response) => {
      if (response.status === 404) {
        setNotFound(true);
      }
      else if (response.status === 401){
        navigate('/login')
      }
        else if (response.status === 500){
          //setServerError(true)
        }

        if (!response.ok){
          setError(true);
          throw new Error('Something went wrong');
        }
        return response.json();
    })
    .then((data) => {
      if (data && data.length > 0) {
        setMeanings(data[0].meanings);
      } else {
        setNotFound(true);
```

```
      }
    })
    .catch((error) => {
      console.error(error);
    });
}, [search, navigate]);

if (notFound === true) {
  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
    <>
      <NotFoundComponent/>
      <Link to='/dictionary'>Search Another Word</Link>
    </>
    </div>
  );
}

if (error === true) {
  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
    <>
      <p>Something went wrong, try again? </p>
      <Link to='/dictionary'>Search Another Word</Link>
    </>
    </div>
  );
}


return (
  <div className="bg-purple-300 min-h-screen px-3 py-3">
    {meanings.length > 0 && (
      <>
```

```
        <h1>Definition:</h1>
        {meanings.map((meaning, index) => (
          <p key={index}>
            {meaning.partOfSpeech}: {meaning.definitions[0].definition}
          </p>
        ))}
      </>
    )}
  </div>
 );
}
```

The output is as under:

These are the basics of dealing with errors and status codes. Let's move to the next section.

Build and Style a Search Component

In this section, we'll make our app more user-friendly. To begin, we will allow the enter key to retrieve search results. Then, we'll customize our page to make it more visually appealing. **Here's the completed code.**

```
import { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
export default function Dictionary()
{
  const [word, setWord] = useState('');
  const navigate = useNavigate();

  return (
   <div className="bg-purple-300 min-h-screen px-3 py-3">
      <form className="flex justify-center space-x-2 max-w-[300px]"
```

```
        onSubmit={()=>{

        navigate('/definition/' + word);

        }}
        >

    <input className="shrink min-w-0 px-2 rounded py-1"
    placeholder='Type Here'
    type="text"
    onChange={(e) => {

        setWord(e.target.value);

    }} />

    <button className="bg-purple-600 hover:bg-purple-700 text-white
font-bold py-1 px-2 rounded">Search</button>
    </form>
    </div>
   );
}
```

The above code helps us in getting the search results after pressing enter. **You can see the page's styling in the image below:**

Then, we created a new component named DefinitionSearch.js. We copied the whole code from Dictionary.js and placed it into the new component.

Here is the code from Dictionary.js:

```
import DefinitionSearch from "../components/DefinitionSearch";
```

```
export default function Dictionary()
{
  return <DefinitionSearch/>;
}
```

On the other hand, here is the code from the DefinitionSearch.js

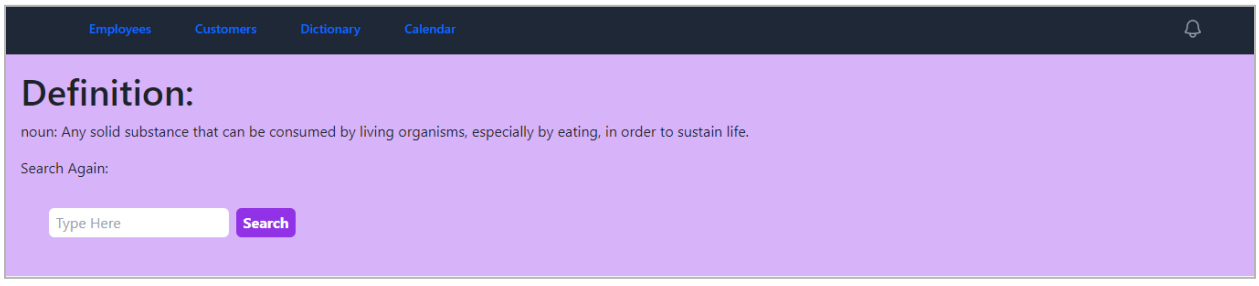
```
import { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
export default function DefinitionSearch(){

   const [word, setWord] = useState('');
  const navigate = useNavigate();

  return (
   <div className="bg-purple-300 min-h-screen px-3 py-3">
      <form className= "flex justify-center space-x-2 max-w-[300px]"

      onSubmit={()=>{

      navigate('/definition/' + word);

      }}
      >

   <input className="shrink min-w-0 px-2 rounded py-1"
   placeholder='Type Here'
   type="text"
   onChange={(e) => {

      setWord(e.target.value);

   }} />

   <button className="bg-purple-600 hover:bg-purple-700 text-white
font-bold py-1 px-2 rounded">Search</button>
   </form>
   </div>
```

```
  );
}
```

Then, we made the following changes to our code in the Definition.js.

```
return (
  <div className="bg-purple-300 min-h-screen px-3 py-3">
   {meanings.length > 0 && (
    <>
     <h1>Definition:</h1>
     {meanings.map((meaning, index) => (
      <p key={index}>
       {meaning.partOfSpeech}: {meaning.definitions[0].definition}
      </p>
     ))}
     <p>Search Again:</p>
     <DefinitionSearch/>

    </>
   )}
  </div>
 );
}
```

Here is how our page appears when we search for any word.



Now, you have to run the following commands one by one in the terminal:

```
git add .

git commit -m "style the dictionary"
```

```
git push
```

Setup up a Django Backend (Full Stack App)

The backend stores information in the database, allowing your application to maintain state and remember data. **You can learn more about Django from the below link:**

https://www.djangoproject.com/.

Then, you need to download Python from the link given below:

https://www.python.org/downloads/.

The next step is to set the Python environment on your Mac OS or Windows system. **After you set up the environment, run the following code in the command prompt to create a directory:**

```
mkdir backend
```

Then, run the following commands:

```
py -m venv .venv

dir

venv\Scripts\activate.
```

After activating the virtual environment, you must run the following command to install Django.

```
py -m pip install django
```

Then, run this command:

```
django-admin

django-admin allows you to create a new Django project structure with the necessary files and directories using the command django-admin startproject projectname as listed below:
```

The next command to run is:

```
django-admin startproject customers .
```

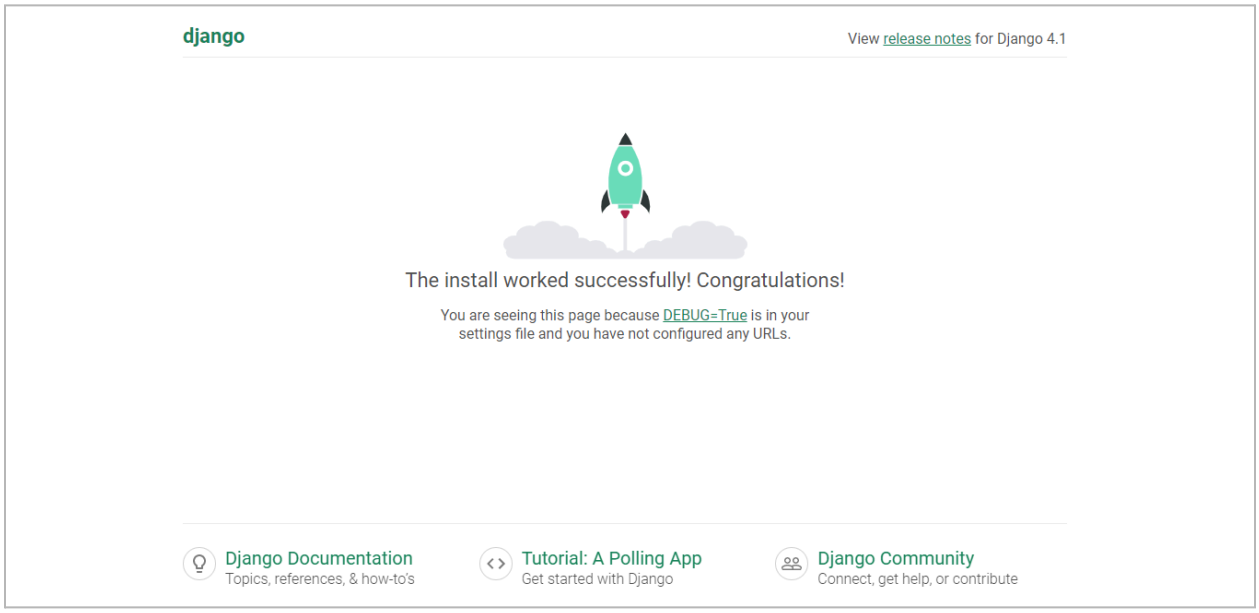
Then, run this command:

```
py manage.py runserver.

This is a convenient way to run your Django project locally for development and testing, providing a simple server setup with live reloading and debugging capabilities
```

Now, it gives us the following URL:

http://127.0.0.1:8000/

When you visit it, we get something as follows:



Create a REST API Backend

Run code . in the command prompt to create a REST API backend. Then, open the Visual Studio Code.

In Visual Studio Code, you must run the following commands:

```
git init
```

Then, create a git repository by clicking on the new file in the Visual Studio Code.

Get a template for gitignore from this link:

https://www.toptal.com/developers/gitignore/api/django

Paste the template in the gitignore file. Save it and then commit all the changes.

Push this code to GitHub by creating a new repository. Then, paste the following lines in the terminal:

```
git remote add origin
https://github.com/devayesha23/react-backend-django.git
git branch -M main
git push -u origin main
```

Then, run the following command in the terminal:

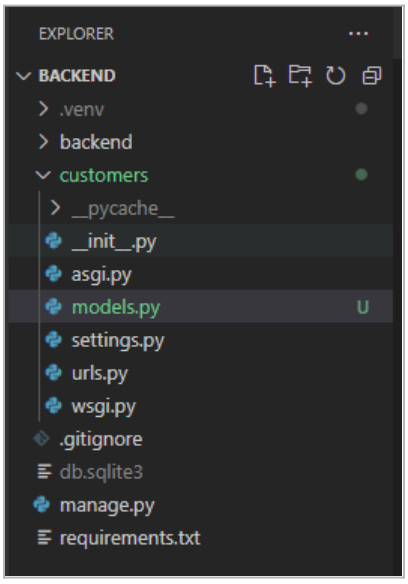
```
git push origin
```

The next step is to activate the virtual environment in the Visual Studio Code. **Now, run the following commands in the terminal:**

```
py -version

pip -version

pip install djangorestframework

python.exe -m pip install --upgrade pip

pip freeze

pip freeze > requirements.txt

pip install -r requirements.txt

git add .

git commit -m add “requirements.txt”

git push
```

```
py manage.py migrate
```

Please remember that the commands must be executed in a specific order. Next, we'll create models for our application. Create a new file titled models.py in the customer's directory.



Add the following code to your newly made file:

```
from django.db import models
class Customer(models.Model):
    name=models.CharField(max_length=200)
    industry=models.CharField(max_length=100)
```

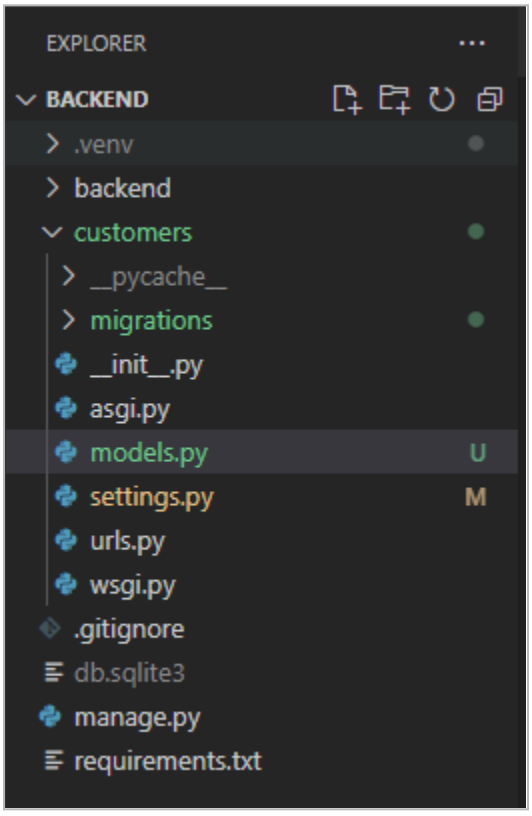
The next step is to open the settings.py file and add your app name under the installed apps section. **In this case, we will add customers as follows:**

```
INSTALLED_APPS = [
    'customers',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Save it and run the following command:

```
py manage.py makemigrations customers
```

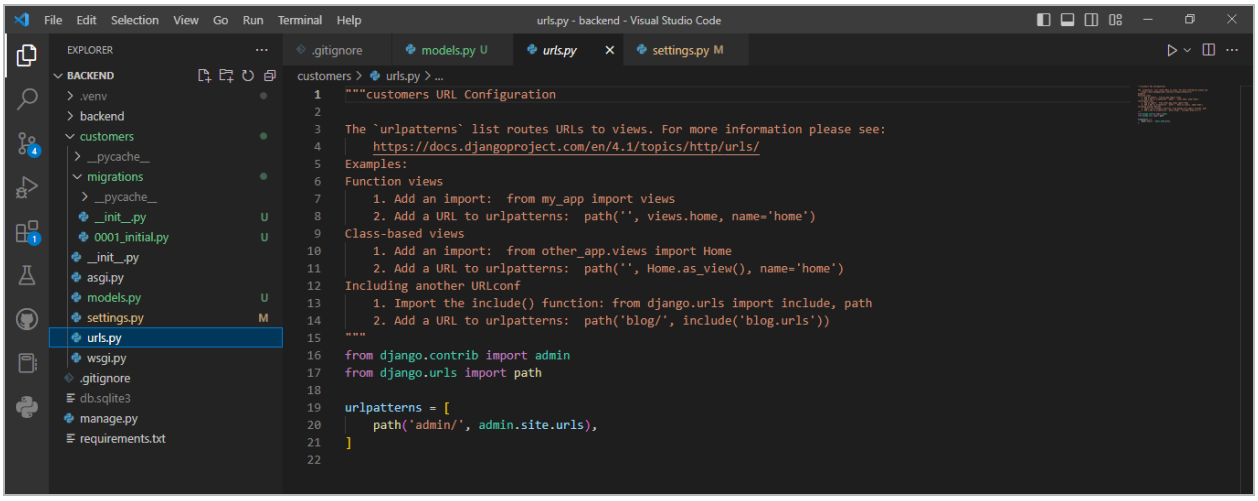
Here, you can see a new file named Migrations:



Then, run the following command:

```
py manage.py migrate
```

Look at urls.py, which is extremely similar to React's routing file. It allows us to enter routes for our pages.



Let's add a route in this file. **We made the following changes to the code in the urls.py file.**

```
from django.contrib import admin
from django.urls import path
from customers import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/customers/', views.customers, name='customers')
]
```

The next step is to create a file views.py in the customers' folder.

Then, add the following code to the newly made file.

```
from customers.models import Customer
from django.http import JsonResponse
from customers.serializers import CustomerSerializer


def customers(request):
   data = Customer.objects.all()
   serializer = CustomerSerializer(data, many=True)
   return JsonResponse({'customers': serializer.data})
```

The next step is to create a file with the name “serializers.py”

Again, we need to add “rest_framework” inside the settings.py file.

```
INSTALLED_APPS = [
    'rest_framework',
    'customers',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

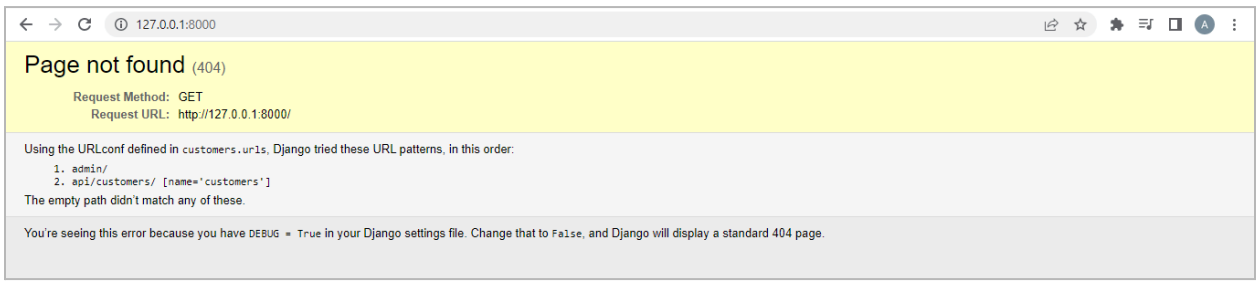
Here is the code that we have to add inside the serializers.py file.

```
from rest_framework import serializers
from customers.models import Customer

class CustomerSerializer(serializers.ModelSerializer):
    class Meta:
        model = Customer
        fields = '__all__'
```

You can start the development server by running the given command:

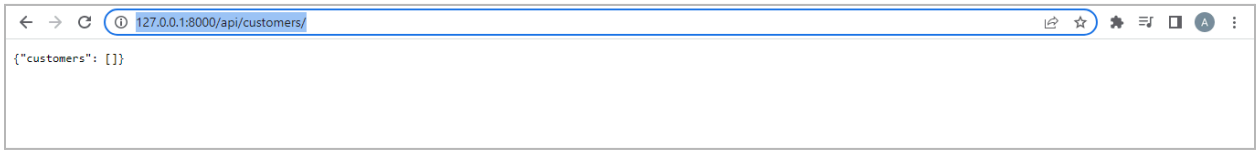
```
py manage.py runserver
```

This is the page that we get now:



Let's navigate to the URL: http://127.0.0.1:8000/api/customers/

If it gives us the following page, it means that it is working fine:

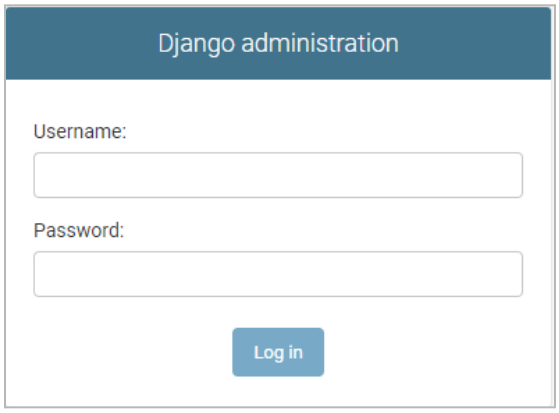


Save all the changes and commit them. **Let's move to the next section.**

Consume Backend API

In this section, we'll learn how to enter our data into the database. Then, we'll learn how to get that data from our React application. The final output image showed only one consumer on the screen. We'll add more customers there. The simplest approach to achieve this is to create a new file.

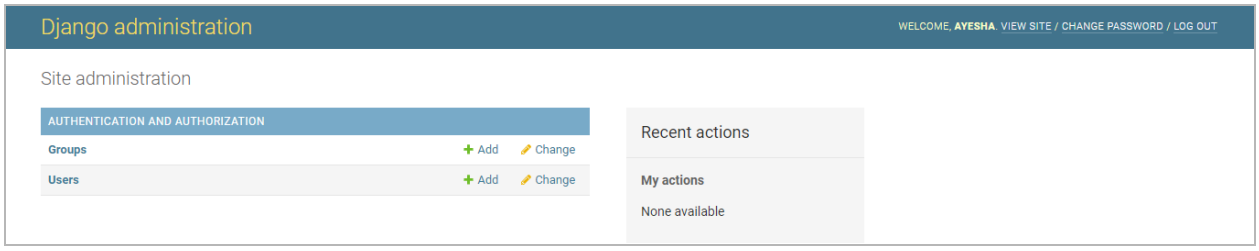
We added a new file called admin.py in the customers' subdirectory. When you access the URL "http://127.0.0.1:8000/admin/," you will see the following page. We'll venture into this page to see what it delivers us.



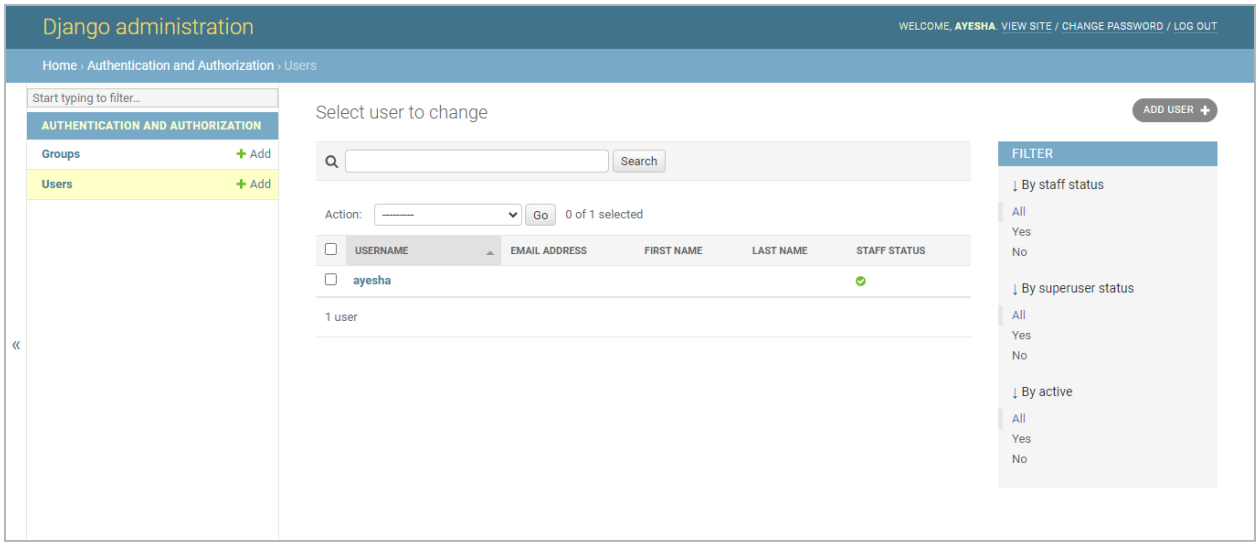
Run the following command in the terminal:

```
py manage.py
createsuperuser
```

It will ask us to enter our username, password, and email address. We must provide the relevant information and restart the server. It will display the above login page, but we will enter our login information now. **This will lead us to the next page:**



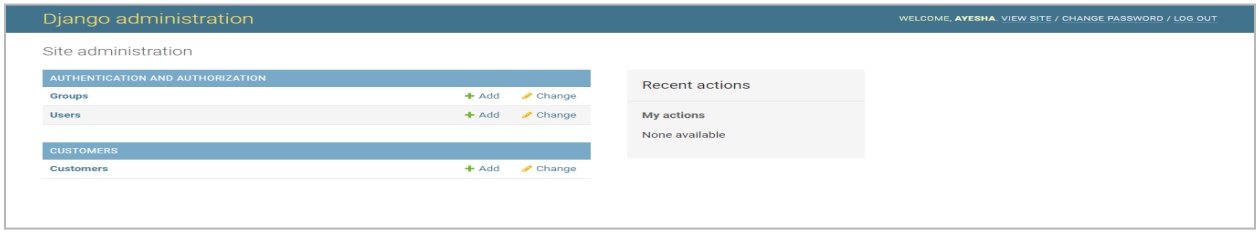
Here, when we click on the Users tab, we get the details for our user as follows:



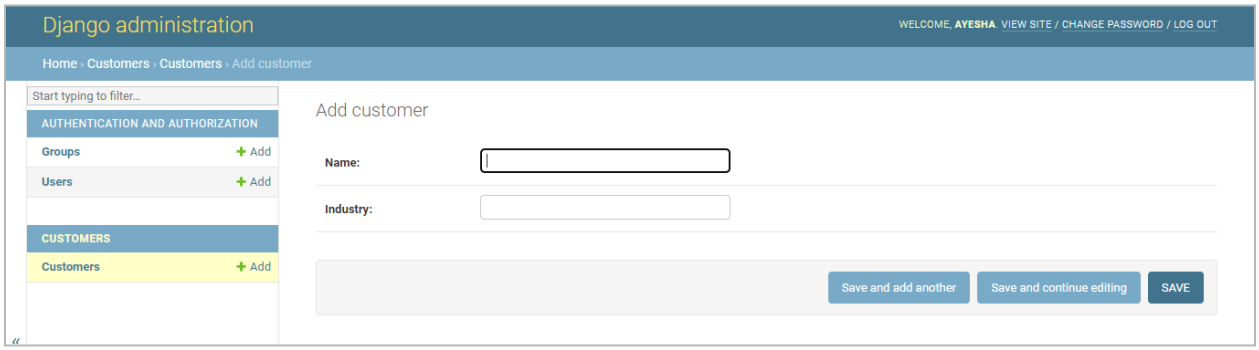
The next step is to create a new model. Add the following code to the admin.py file.

```
from django.contrib import admin
from customers.models import Customer

admin.site.register(Customer)
```

Refresh the site administration page, and it will show us a new model, "Customer," as follows:

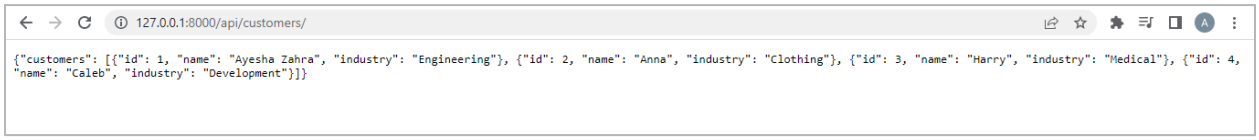


It also allows us to add customers through the following form.



Let's suppose we add a customer here and then visit: http://127.0.0.1:8000/api/customers/ . **It shows us the following output now:**

Let's add a few more customers here, and our page will look like the below:



The next step is to connect the Customers model to the React application we created previously. **Enter your React application's Customers.js file and add the following code to do so.**

```
import { useEffect, useState } from 'react';
export default function Customers(){

    const[customers, setCustomers] = useState;

    useEffect(() => {
        console.log('Fetching...');
        fetch('http://localhost:8000/api/customers/').then((response)=>
response.json()).then((data)=>{
        console.log(data);
        setCustomers(data);

        });
```

```
    });
    return <p>Hello there</p>;
}
```

When we run the above code, it will give us an error like below:

```
Access to fetch at 'http://localhost:8000/api/customers/' from origin 'http://localhost:3000' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.
```

Let's fix the error. By default, our backend denies all requests from the origin. **To resolve this issue, we will install the following package:**

You must execute the following command in the backend terminal:

```
pip install django-cors-headers.
```

Then, run the following command:

```
pip freeze > requirements.txt.
```

Make the given changes to the settings.py file.

```
INSTALLED_APPS = [
    'corsheaders',
    'rest_framework',
    'customers',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Then, add a few changes to the below section in the settings.py file:

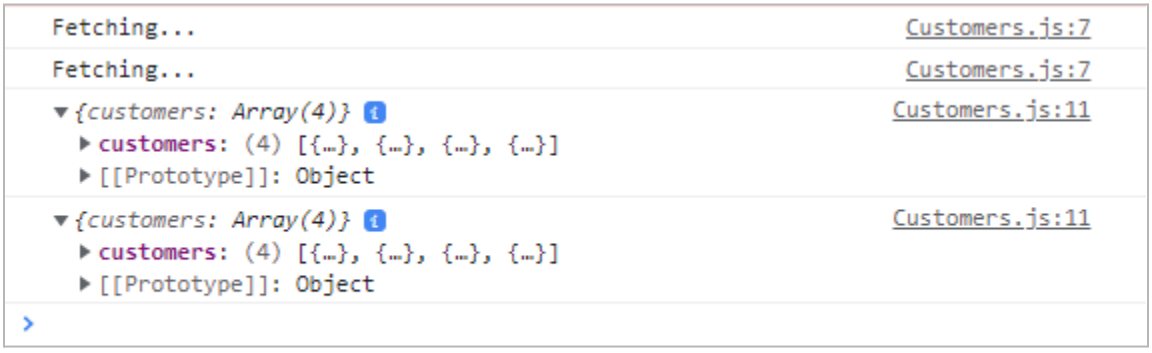
```
MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Putting together a list of permitted origins is the next stage. Put the following command in the settings.py file directly beneath the MIDDLEWARE section.

```
CORS_ALLOWED_ORIGINS =
['http://localhost:3000']
```

From the backend, you can restart the server.

The output appears in the development console when we restart the development server.

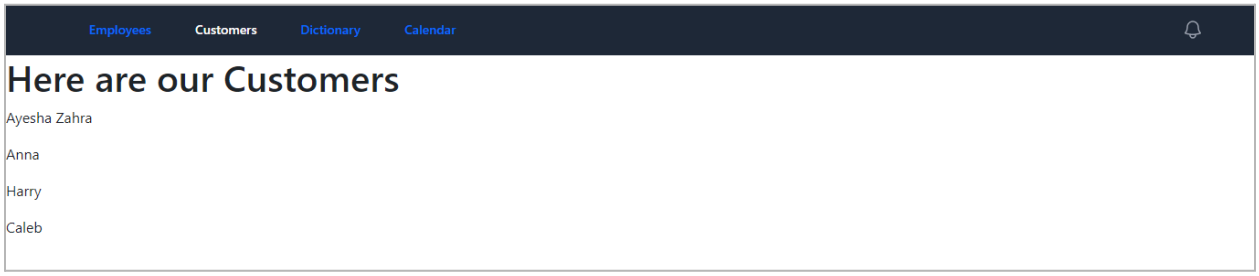


We wish to display the aforementioned facts on our webpage. We have to request our backend for this purpose. **We added This code to the React app's Customers.js file.**

```
import { useEffect, useState } from 'react';
```

```
export default function Customers() {
  const [customers, setCustomers] = useState([]);

  useEffect(() => {
    console.log('Fetching...');
    fetch('http://localhost:8000/api/customers/')
      .then((response) => response.json())
      .then((data) => {
        console.log(data);
        setCustomers(data.customers);
      });
  }, []);

  return (
  <>

  <h1>Here are our Customers</h1>
  {customers ?
  customers.map((customer) => {
    return<p>{customer.name}</p>;
  }) : null }

  </>
  );
}
```

We can now observe that the clients we added to the Django database are displayed as follows on the web page of our React app:



Create a Details by ID API

Our main goal in this section is to get the individual customer details through the URL

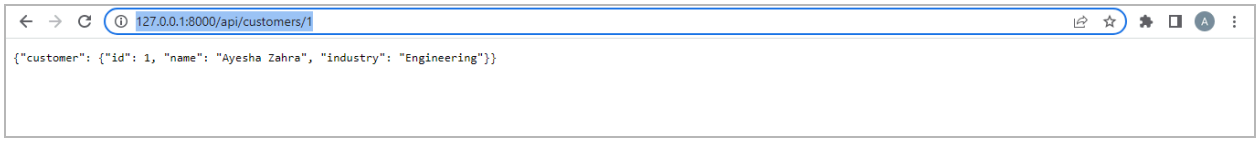
on the web page. **For this purpose, we changed the views.py file as follows:**

```
from customers.models import Customer
from django.http import JsonResponse
from customers.serializers import CustomerSerializer
from django.http import Http404

def customers(request):
   data = Customer.objects.all()
   serializer = CustomerSerializer(data, many=True)
   return JsonResponse({'customers': serializer.data})

def customer(request, id):

   try:
      data = Customer.objects.get(pk=id)

      serializer = CustomerSerializer(data)
      return JsonResponse({'customer': serializer.data})

   except Customer.DoesNotExist:

      raise Http404("Customer with the provided ID does not exist.")
```

The urls.py file needs to be modified as follows in the following steps:

```
from django.contrib import admin
from django.urls import path
from customers import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/customers/', views.customers, name='customer'),
    path('api/customer/<int:id>', views.customer, name ='customer')
]
```

Now, when we go to the following URL: http://127.0.0.1:8000/api/customers/1, we receive the following information for each customer:

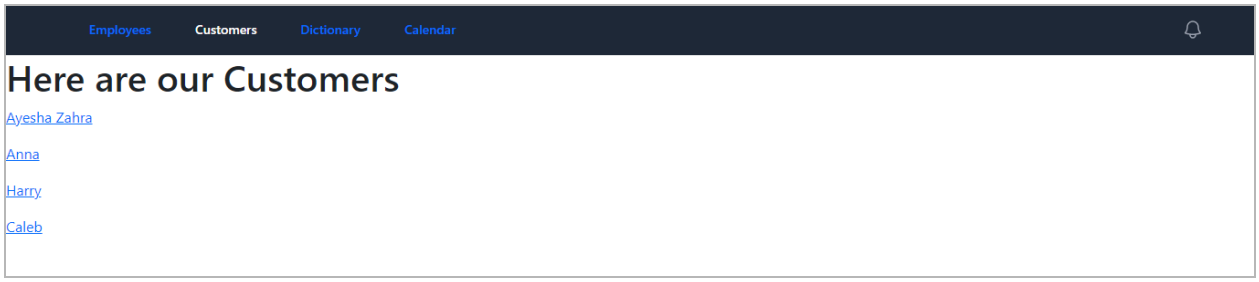


The next step is to obtain the same response in the React application. We modified the Customers.js file in the React app to achieve this.

```
import { useEffect, useState } from 'react';
import {Link} from 'react-router-dom';

export default function Customers() {
  const [customers, setCustomers] = useState([]);

  useEffect(() => {
    console.log('Fetching...');
    fetch('http://localhost:8000/api/customers/')
      .then((response) => response.json())
      .then((data) => {
        console.log(data);
        setCustomers(data.customers);
      });
  }, []);

  return (
  <>

  <h1>Here are our Customers</h1>
  {customers ?
  customers.map((customer) => {
    return (
    <p>
    <Link to = {"/customers/" + customer.id }>{customer.name}</Link>
    </p>
  );


  }) : null }

  </>
```

```
  );
}
```

The links are clickable now, and here is the output:



Clicking on the first customer's name produces the output seen below. We will fill this page in the next phase.

Create a Details Page

As we previously observed, their details are not displayed when we click on a customer. Now, let's get that resolved. First, in our React project, we'll make a new page called Customer.js under the pages directory.

To get started, take these actions. To the Customer.js component, add the following code:

```
import { useParams, Link } from 'react-router-dom';
import { useEffect, useState } from 'react';

export default function Customer(){
    const {id} = useParams();
    const [customer, setCustomer] = useState([]);
    useEffect(() => {
        console.log('useEffect');
        const url = 'http://localhost:8000/api/customers/' + id;
        fetch(url)
        .then((response) => {
            return response.json();
```

```
        })
        .then((data) => {
            setCustomer(data.customer);
        });
    }, []);
    return(
        <>
        { customer ? <div>
            <p>{customer.id}</p>
            <p>{customer.name}</p>
            <p>{customer.industry}</p>
            </div> : null }
            <Link to = "/customers">Go Back</Link>
            </>

    );
}
```

Then, add the route for the above component in the App.js file.

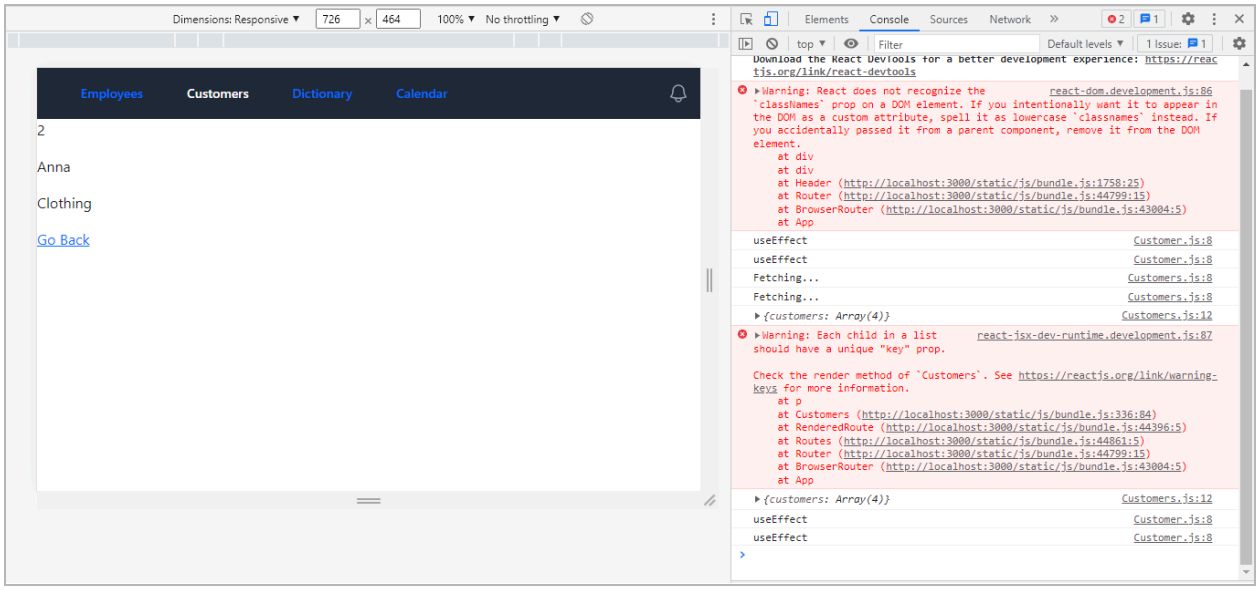
```
<Route path="/customers/:id" element={<Customer />} />
```

The next step is to import it within the App.js as follows:

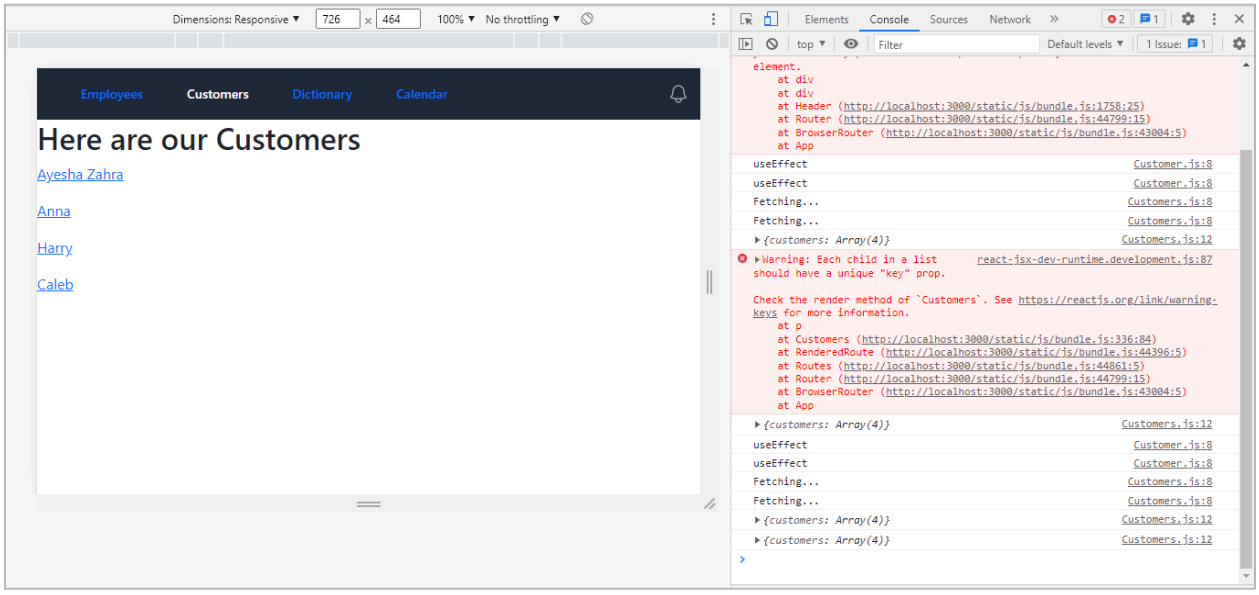
```
import Customer from './Pages/Customer';
```

Save the code, and let's visit the following URL: http://localhost:3000/customers/2

It will bring us to the following page:



We can see that the client with ID number 2 has been successfully shown on the page. In addition, you can go back and view the details of any other customer.



Let’s move to the next section, where we will learn to deal with errors.

Return 404 From Backend API (Django)

In this section, we'll see how to show a suitable error page when something goes wrong during a fetch operation. The user should receive an appropriate error message if a URL is improperly input.

To do this, we made the following changes to our React app's Customer.js file:

```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';

export default function Customer(){
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    useEffect(() => {
        console.log('useEffect');
        const url = 'http://localhost:8000/api/customers/' + id;
        fetch(url)
        .then((response) => {

            if (response.status === 404)

            navigate('/404');
            return response.json();
        })
        .then((data) => {
            setCustomer(data.customer);
        });
    }, []);
    return(
        <>
        { customer ? <div>
            <p>{customer.id}</p>
            <p>{customer.name}</p>
            <p>{customer.industry}</p>
            </div> : null }
            <Link to = "/customers">Go Back</Link>
            </>

    );
}
```

Then, we will make the following changes to views.py on our backend.

```
from customers.models import Customer
from django.http import JsonResponse, Http404
from customers.serializers import CustomerSerializer


def customers(request):
    data = Customer.objects.all()
    serializer = CustomerSerializer(data, many=True)
    return JsonResponse({'customers': serializer.data})

def customer(request, id):
    try:
        data = Customer.objects.get(pk=id)
    except Customer.DoesNotExist:
        raise Http404('Customer does not exist')
    serializer = CustomerSerializer(data)
    return JsonResponse({'customer': serializer.data})
```

Now, if we enter some random URL like http://localhost:3000/customers/pizza, **it will show us the following page:**

The next step is to restructure our URLs to avoid duplication while routing the application. **To accomplish this, we first created a file called shared.js in the src directory and included the following code:**

```
export const baseUrl = 'http://localhost:8000/';
```

Then, we make a few changes to the Customer.js component as follows:

```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';
import { baseUrl } from '../shared';
```

```
export default function Customer(){
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    useEffect(() => {
        console.log('useEffect');
        const url = baseUrl + 'api/customers/' + id;
        fetch(url)
        .then((response) => {

            if (response.status === 404)

            navigate('/404');
            return response.json();
        })
        .then((data) => {
            setCustomer(data.customer);
        });
    }, []);
    return(
        <>
        { customer ? <div>
            <p>{customer.id}</p>
            <p>{customer.name}</p>
            <p>{customer.industry}</p>
            </div> : null }
            <Link to = "/customers">Go Back</Link>
            </>

    );
}
```

We also made some changes to the Customers.js component.

```
import { useEffect, useState } from 'react';
import {Link} from 'react-router-dom';
import { baseUrl } from '../shared';
```

```
export default function Customers() {
  const [customers, setCustomers] = useState([]);

  useEffect(() => {
    const url = baseUrl + 'api/customers/'
    fetch(url)
      .then((response) => response.json())
      .then((data) => {
        setCustomers(data.customers);
      });
  }, []);

  return (
  <>

  <h1>Here are our Customers</h1>
  {customers ?
  customers.map((customer) => {
    return (
    <p>
    <Link to = {"/customers/" + customer.id }>{customer.name}</Link>
    </p>
  );


  }) : null }

  </>
  );
}
```

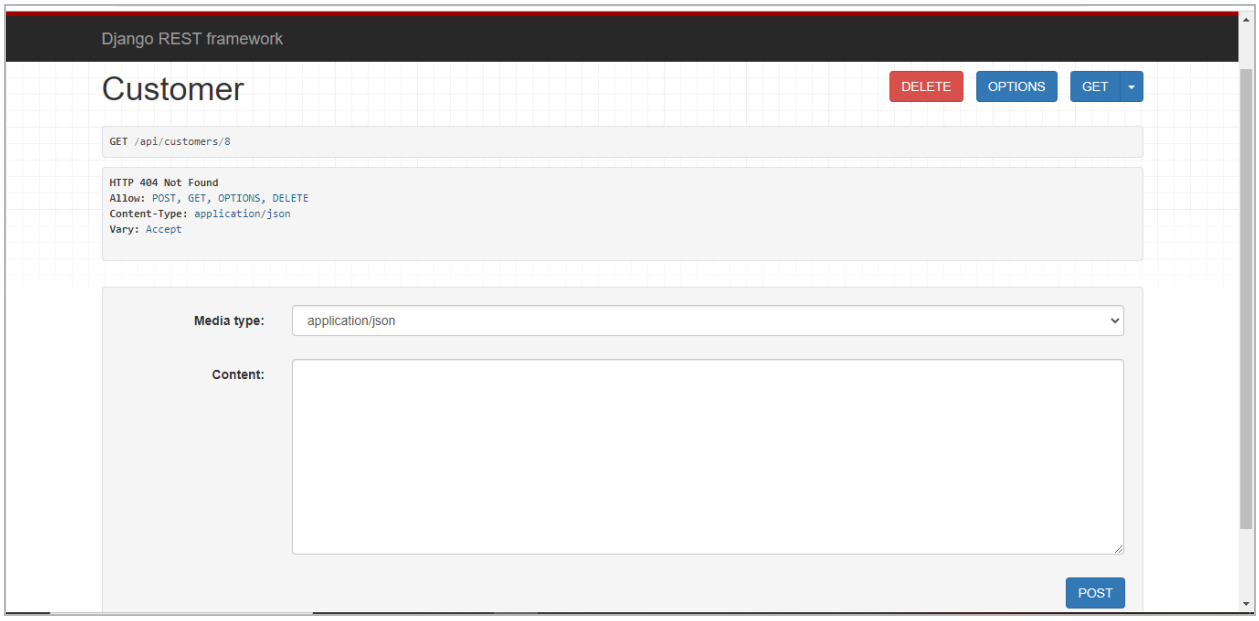
We can see that our app is still working fine after rearranging the URLs.

Code a Full CRUD API (Create, Read, Update, Delete)

First, we committed to implementing our previous modifications in the front and back end. Then, we made some adjustments to the views.py file. These improvements will enable us to support GET, POST, and DELETE methods.

```
from customers.models import Customer
from django.http import JsonResponse, Http404
from customers.serializers import CustomerSerializer
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status

def customers(request):
   data = Customer.objects.all()
   serializer = CustomerSerializer(data, many=True)
   return JsonResponse({'customers': serializer.data})

@api_view(['GET', 'POST', 'DELETE'])

def customer(request, id):
   try:
      data = Customer.objects.get(pk=id)
   except Customer.DoesNotExist:
      return Response(status=status.HTTP_404_NOT_FOUND)
   serializer = CustomerSerializer(data)
   return Response({'customer': serializer.data})
```

This is what we get when we visit the URL: http://127.0.0.1:8000/api/customers/8

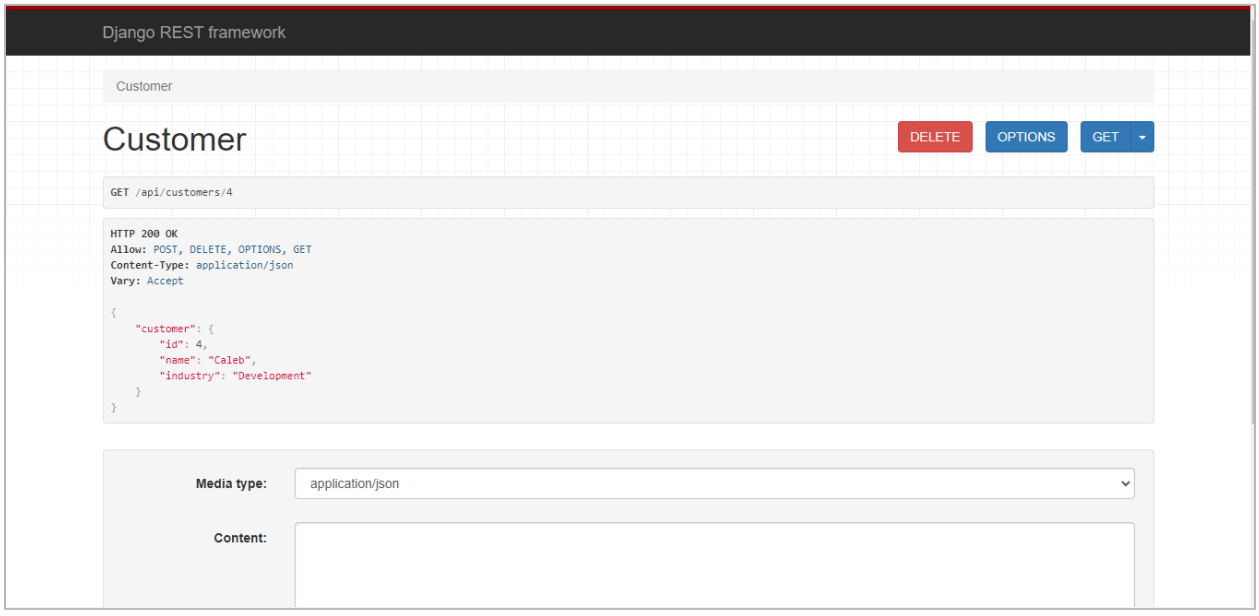


First, we committed to executing our previous changes on both the front and back ends. We then made some changes to the views.py file. These modifications will allow us to support methods like GET, POST, and DELETE.

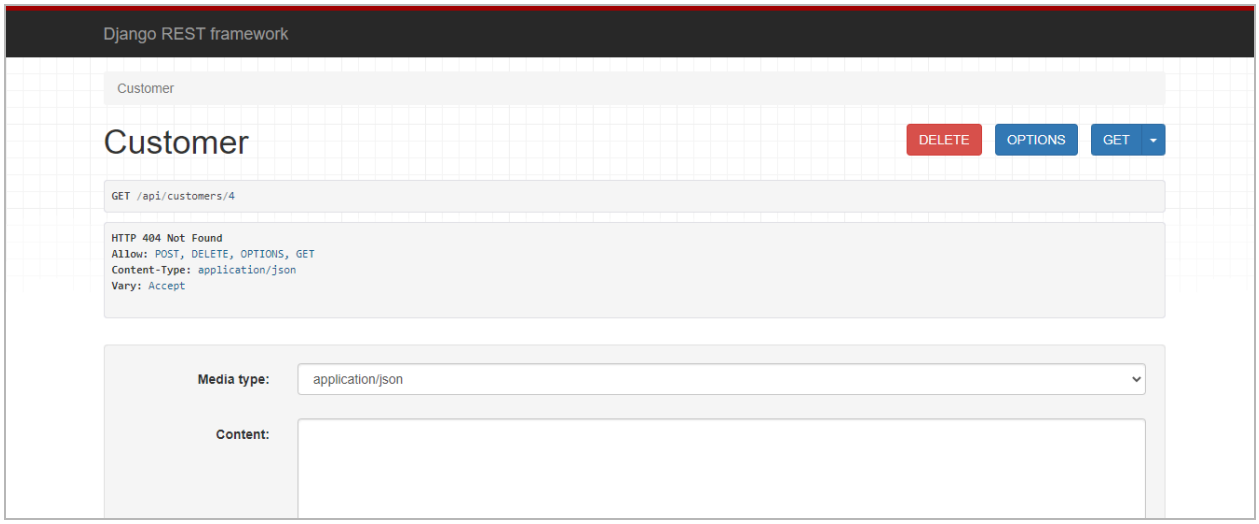
```
from customers.models import Customer
from django.http import JsonResponse, Http404
from customers.serializers import CustomerSerializer
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status

@api_view(['GET', 'POST'])

def customers(request):

  data = Customer.objects.all()

  serializer = CustomerSerializer(data, many=True)

  return Response({'customers': serializer.data})

@api_view(['GET', 'POST', 'DELETE'])

def customer(request, id):
```

```
try:
    data = Customer.objects.get(pk=id)
except Customer.DoesNotExist:
    return Response(status=status.HTTP_404_NOT_FOUND)

if request.method == 'GET':
    serializer = CustomerSerializer(data)
    return Response({'customer': serializer.data})
elif request.method == 'DELETE':
    data.delete()
    return Response(status=status.HTTP_204_NO_CONTENT)
```

Now, when we visit http://127.0.0.1:8000/api/customers/4 , it shows us the following details.



We can delete it now and then revisiting the same URL gives us the following output:

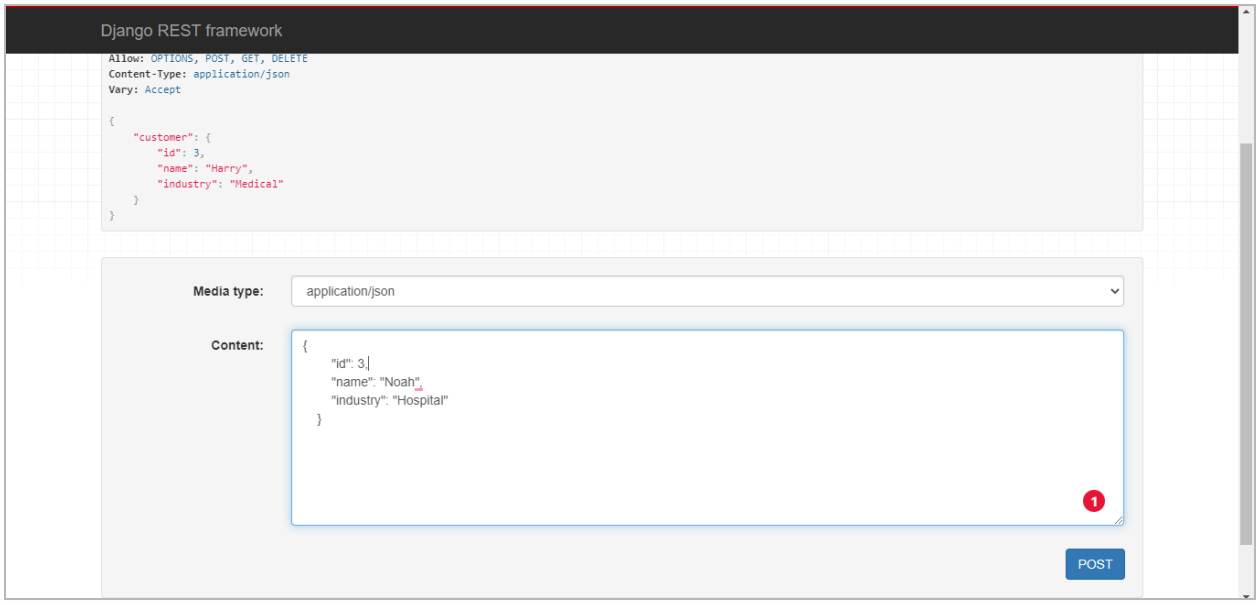


The next step is to discuss POST, allowing us to alter data for a specific client on the list. We made the following adjustments at the end to get the code to work.

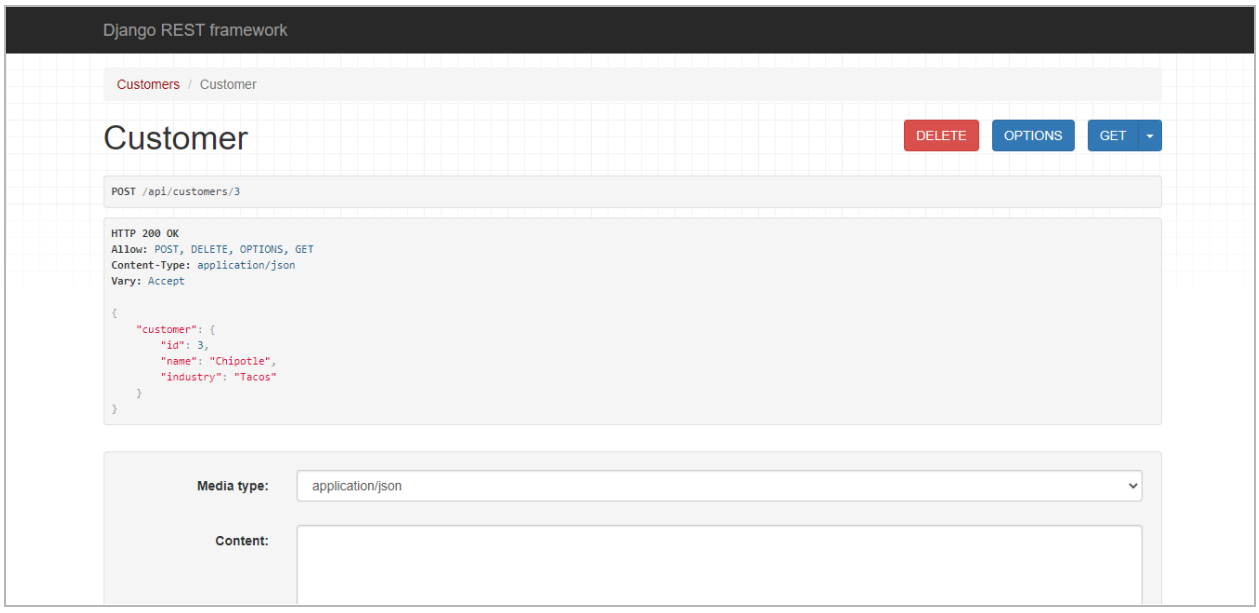
```
from customers.models import Customer
from django.http import JsonResponse, Http404
from customers.serializers import CustomerSerializer
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status

@api_view(['GET', 'POST'])
def customers(request):
   data = Customer.objects.all()
   serializer = CustomerSerializer(data, many=True)
   return Response({'customers': serializer.data})

@api_view(['GET', 'POST', 'DELETE'])

def customer(request, id):
   try:
      data = Customer.objects.get(pk=id)
   except Customer.DoesNotExist:
      return Response(status=status.HTTP_404_NOT_FOUND)

   if request.method == 'GET':
      serializer = CustomerSerializer(data)
```

```
    return Response({'customer': serializer.data})
  elif request.method == 'DELETE':
    data.delete()
    return Response(status=status.HTTP_204_NO_CONTENT)
  elif request.method == 'POST':
    serializer = CustomerSerializer(data, data=request.data)
    if serializer.is_valid():
      serializer.save()
    return Response({'customer': serializer.data})
  return Response (serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
```

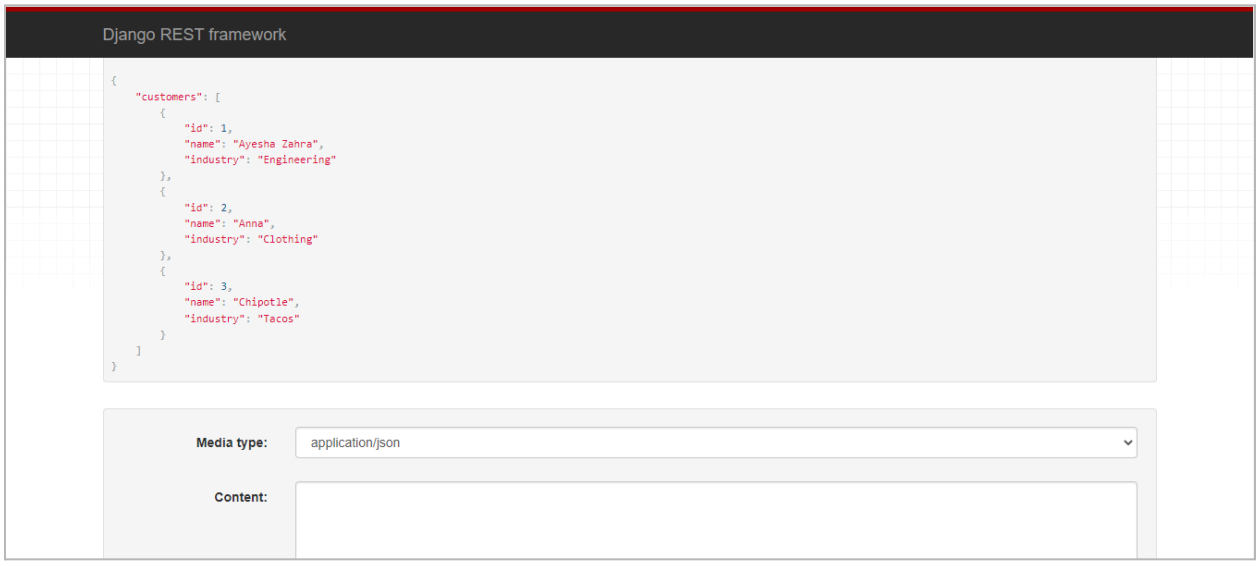
Now if we click on POST after adding some information like below:



It will give us the modified information as follows:



You will be directed to a new website when you click the red "Customers" button in the top bar. This page also provides a form for entering new information. However, the form is not yet functional. We'll have to edit the code in views.py on the backend to remedy this.

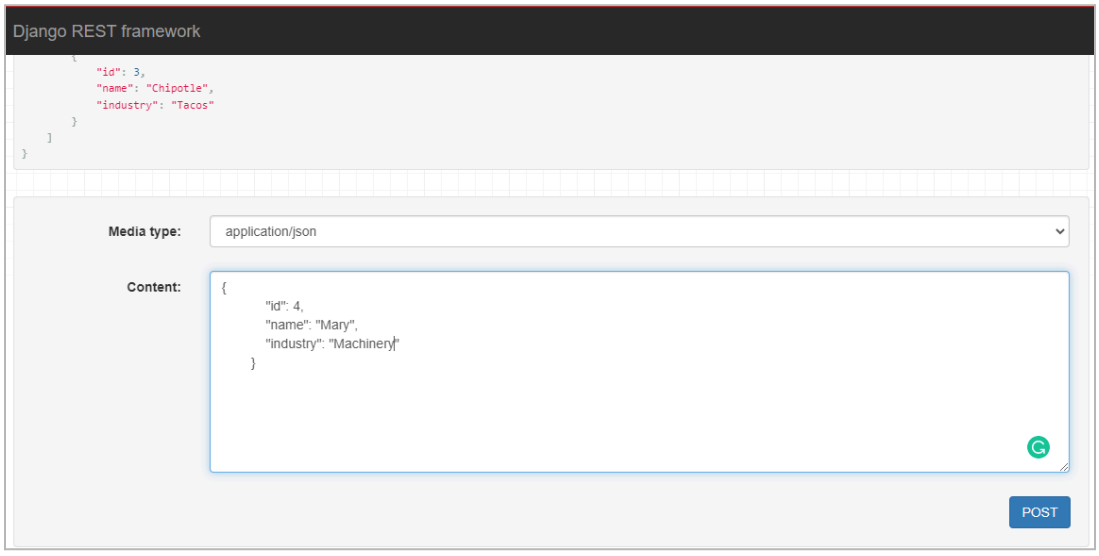


Let's make some changes to our code as follows:

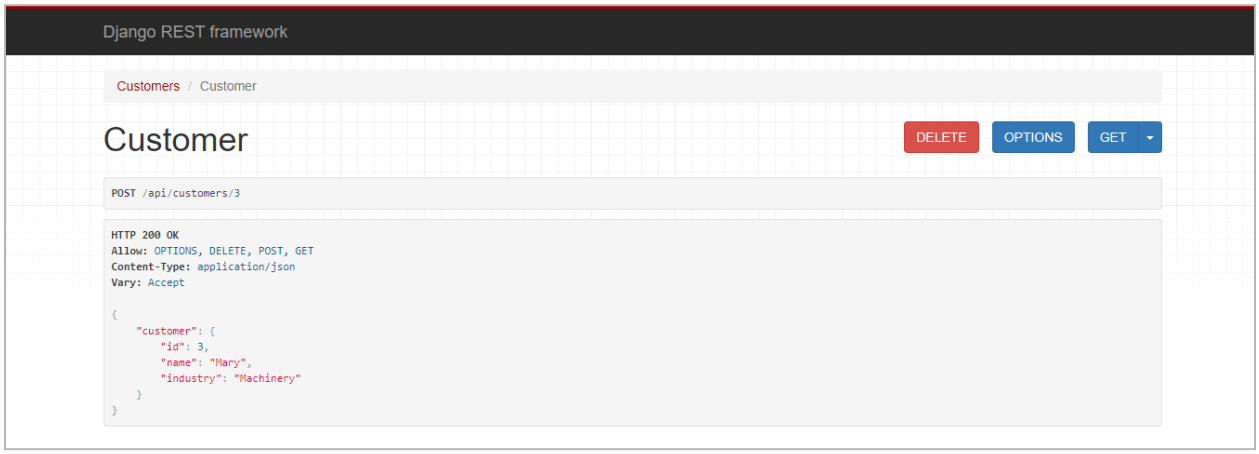
```
from customers.models import Customer
from django.http import JsonResponse, Http404
from customers.serializers import CustomerSerializer
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status

@api_view(['GET', 'POST'])
def customers(request):
  if request.method == 'GET':
    data = Customer.objects.all()
    serializer = CustomerSerializer(data, many=True)
    return Response({'customers': serializer.data})

  elif request.method == 'POST':
    serializer = CustomerSerializer(data=request.data)

if serializer.is_valid():
    serializer.save()
    return Response({'customer': serializer.data},
status=status.HTTP_201_CREATED)
  return Response (serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

@api_view(['GET', 'POST', 'DELETE'])

def customer(request, id):
  try:
     data = Customer.objects.get(pk=id)
  except Customer.DoesNotExist:
    return Response(status=status.HTTP_404_NOT_FOUND)

  if request.method == 'GET':
    serializer = CustomerSerializer(data)
    return Response({'customer': serializer.data})
  elif request.method == 'DELETE':
    data.delete()
```

```
        return Response(status=status.HTTP_204_NO_CONTENT)
    elif request.method == 'POST':
        serializer = CustomerSerializer(data, data=request.data)
        if serializer.is_valid():
            serializer.save()
        return Response({'customer': serializer.data})
    return Response (serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
```

Then, we will fill out the form and click on POST.



We can see the output as follows:



DELETE Request with Fetch

When we visit the Customers page in our React app, we receive the following error in the developer console.

```
Warning: Each child in a list should have a unique "key" prop.
```

The error was removed by making some changes at the bottom of the code in the Customers.js file.

```
import { useEffect, useState } from 'react';
import {Link} from 'react-router-dom';
import { baseUrl } from '../shared';

export default function Customers() {
  const [customers, setCustomers] = useState([]);

  useEffect(() => {
    const url = baseUrl + 'api/customers/'
    fetch(url)
      .then((response) => response.json())
      .then((data) => {
        setCustomers(data.customers);
      });
  }, []);

  return (
  <>

  <h1>Here are our Customers</h1>
  <ul>
  {customers ?
  customers.map((customer) => {
    return (
    <li key={customer.id}>
    <Link to = {"/customers/" + customer.id }>{customer.name}</Link>
    </li>
  );
```

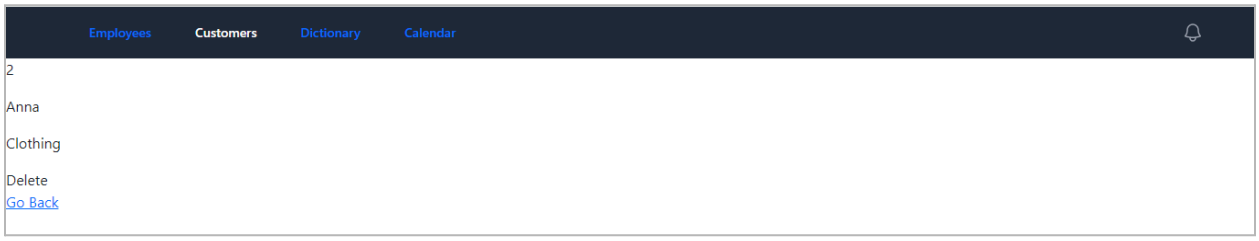
```
  }) : null }
  </ul>

  </>
  );
}
```

The next step is to add a delete button to remove a customer directly from the React app. **Below are the changes we made to the code in Customers.js:**

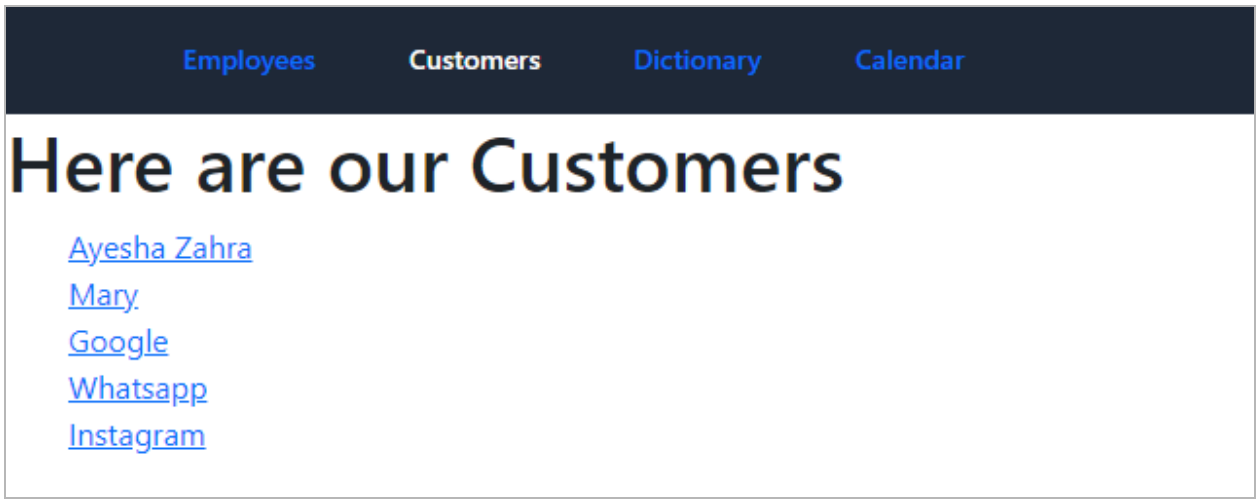
```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';
import { baseUrl } from '../shared';

export default function Customer(){
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    useEffect(() => {
        console.log('useEffect');
        const url = baseUrl + 'api/customers/' + id;
        fetch(url)
        .then((response) => {

            if (response.status === 404)

            navigate('/404');
            return response.json();
        })
        .then((data) => {
            setCustomer(data.customer);
        });
    }, []);
    return(
        <>
        { customer ? <div>
            <p>{customer.id}</p>
```

```
            <p>{customer.name}</p>
            <p>{customer.industry}</p>
            </div> : null }
<button
    onClick={(e) => {
        const url = baseUrl + 'api/customers/' + id;
        fetch (url, { method: 'DELETE'}).then((response) => {
            if(!response.ok) {
                throw new Error('Something went wrong')
            }
           navigate('/customers');
        })
        .catch((e) => {console.log(e)});
    }}
    > Delete
</button>
<br/>
            <Link to = "/customers">Go Back</Link>
            </>

    );
}
```

Here is the output:

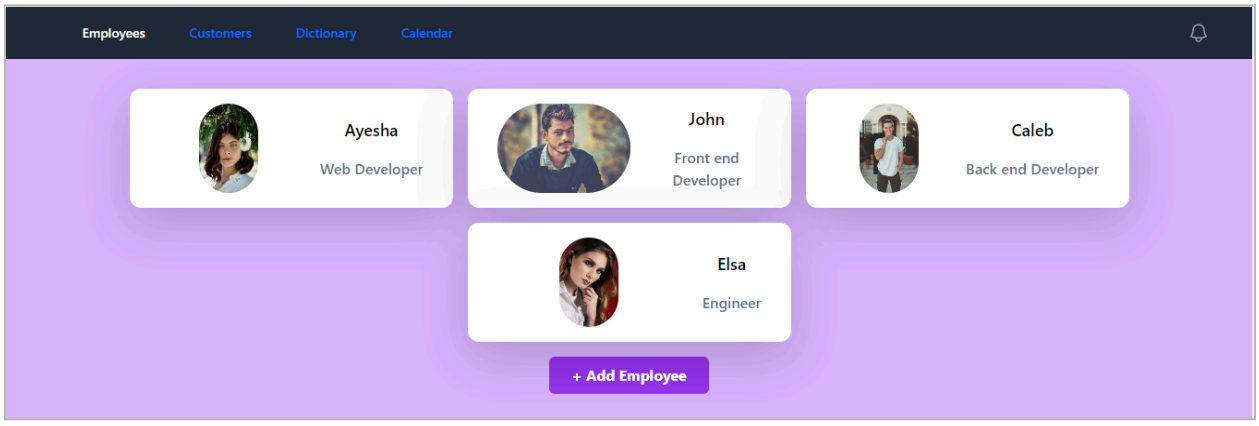


The delete button needs some design, but it still works properly. Assuming we delete the above customer, we can see that it is no longer on our customer list.



Popup Modal to Add Data (POST)

We see an Add Employee button when we open the Employees tab in our React app. We'll enable this button in the Customers tab right now.



To implement this, we'll start by duplicating the AddEmployee.js component, renaming it to AddCustomer.js, and then making the necessary adjustments to the duplicated component.

First, we made some key changes to the AddCustomer.js component. Add the following code to the new component:

```
import React, { useState } from 'react';
import Button from 'react-bootstrap/Button';
import Modal from 'react-bootstrap/Modal';

export default function AddCustomer(props) {
  const [name, setName] = useState('');
```

```
 const [industry, setIndustry] = useState('');
 const [show, setShow] = useState(false);

 const handleClose = () => setShow(false);
 const handleShow = () => setShow(true);

 return (

   <>
   <button onClick={handleShow}
   className="block mx-auto m-2 bg-purple-600 hover:bg-purple-700
text-white font-bold py-2 px-4 rounded">
       + Add Customer
       </button>

     <Modal
       show={show}
       onHide={handleClose}
       backdrop="static"
       keyboard={false}
     >
       <Modal.Header closeButton>
         <Modal.Title>Add Customer</Modal.Title>
       </Modal.Header>
       <Modal.Body>
       <form
       onSubmit={(e)=>{
         e.preventDefault();
         setName('');
         setIndustry('');
         props.newCustomer(name, industry);

       }}

       id = 'editmodal'className="w-full max-w-sm">
 <div className="md:flex md:items-center mb-6">
   <div className="md:w-1/3">
     <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="name">
       Full Name
```

```
      </label>
    </div>
    <div className="md:w-2/3">
      <input className="bg-gray-200 appearance-none border-2
border-gray-200
      rounded w-full py-2 px-4 text-gray-700 leading-tight focus:outline-none
focus:bg-white focus:border-purple-500"
      id="name"
      placeholder='Name Here'
      type="text"
      value={name}
      onChange={(e)=>{setName(e.target.value)}}
      />
    </div>
              </div>
              <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="industry">
        Industry
      </label>
    </div>
    <div className="md:w-2/3">
      <input className="bg-gray-200 appearance-none border-2
border-gray-200 rounded
      w-full py-2 px-4 text-gray-700 leading-tight focus:outline-none
focus:bg-white
      focus:border-purple-500"
      id="Industry"
      placeholder="Industry"
      type="text"
      value={industry}
      onChange={(e)=>{setIndustry(e.target.value)}}
      />
      </div>
      </div>
      <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
    </div>
    </div>
```

```
          </form>
        </Modal.Body>
        <Modal.Footer>
          <button className="bg-pink-500 hover:bg-pink-700 text-white
font-bold py-2 px-4 rounded"
          onClick={handleClose}>Close</button>
           <button
           className="bg-purple-600 hover:bg-purple-700 text-white font-bold
py-2 px-4 rounded"
           onClick={
            handleClose
          }
          form= "editmodal">
            Add
          </button>
        </Modal.Footer>
      </Modal>
    </>
  );
}
```

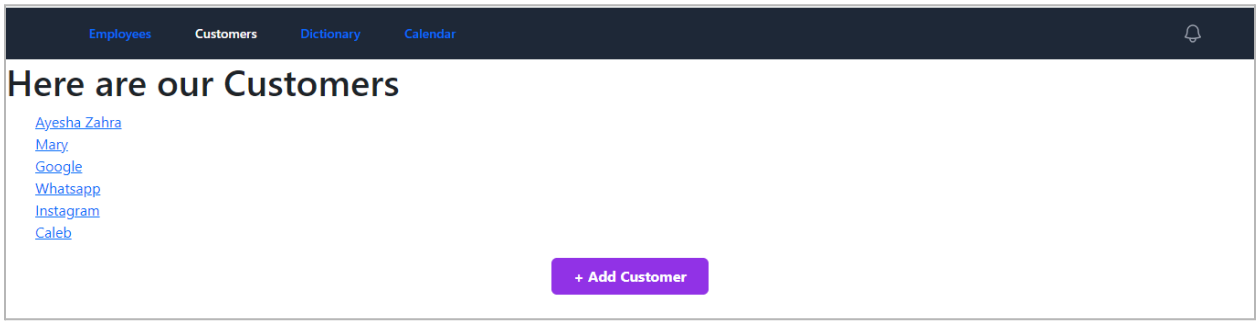
The next step is to make some changes to your code in the Customers.js component.

```
import { useEffect, useState } from 'react';
import {Link} from 'react-router-dom';
import AddCustomer from '../components/AddCustomer';
import { baseUrl } from '../shared';

export default function Customers() {
  const [customers, setCustomers] = useState([]);

  useEffect(() => {
    const url = baseUrl + 'api/customers/'
    fetch(url)
      .then((response) => response.json())
      .then((data) => {
        setCustomers(data.customers);
      });
  }, []);
```

```
function newCustomer(name, industry) {
  const data = {name: name, industry: industry};
  const url = baseUrl + 'api/customers/';
  fetch(
    url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
    }
  ).then((response) => {
    if(!response.ok)
    throw new Error('Something went wrong');
    return response.json();

  } ).then(()=> {

  }).catch((e)=>{
    console.log(e);
  });
}
return (
<>

<h1>Here are our Customers</h1>
<ul>
{customers ?
customers.map((customer) => {
  return (
  <li key={customer.id}>
  <Link to = {"/customers/" + customer.id }>{customer.name}</Link>
  </li>
);


}) : null }
</ul>
```

```
  <AddCustomer newCustomer = {newCustomer}/>

  </>
  );
}
```

Now you can see that we have a button to add customers using our web application. Furthermore, it works perfectly.



This section's major purpose has been achieved. **Let's proceed to the following phase, where we'll create even more incredible application improvements.**

Close the modal on POST Success (and Add Results to State)

In this section, we'll learn how to automatically close the modal after successfully clicking "Add" to add a customer. We've made some changes to the Customers.js component, as shown below.

Note that we've added a toggleShow function in the Customers.js and AddCustomer.js components.

```
import { useEffect, useState } from 'react';
import {Link} from 'react-router-dom';
import AddCustomer from '../components/AddCustomer';
import { baseUrl } from '../shared';

export default function Customers() {
  const [customers, setCustomers] = useState([]);
  const [show, setShow] = useState(false);
  function toggleShow() {
    setShow(!show)
```

```
}

useEffect(() => {
  const url = baseUrl + 'api/customers/'
  fetch(url)
    .then((response) => response.json())
    .then((data) => {
      setCustomers(data.customers);
    });
}, []);

function newCustomer(name, industry) {
  const data = {name: name, industry: industry};
  const url = baseUrl + 'api/customers/';
  fetch(
    url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
    }
  ).then((response) => {
    if(!response.ok)
    throw new Error('Something went wrong');
    return response.json();

  } ).then((data)=> {
      toggleShow();
  }).catch((e)=>{
    console.log(e);
  });
}
return (
<>

<h1>Here are our Customers</h1>
<ul>
{customers ?
```

```
  customers.map((customer) => {
    return (
    <li key={customer.id}>
    <Link to = {"/customers/" + customer.id }>{customer.name}</Link>
    </li>
  );


  }) : null }
  </ul>
  <AddCustomer newCustomer = {newCustomer} show={show}
toggleShow={toggleShow}/>

  </>
  );
}
```

Then, here are the changes that we made to the AddCustomer.js component:

```
import React, { useState } from 'react';
import Button from 'react-bootstrap/Button';
import { propTypes } from 'react-bootstrap/esm/Image';
import Modal from 'react-bootstrap/Modal';

export default function AddCustomer(props) {
  const [name, setName] = useState('');
  const [industry, setIndustry] = useState('');
  const [show, setShow] = useState(props.show);

  const handleClose = () => setShow(false);
  const handleShow = () => setShow(true);

  return (

    <>
    <button onClick={props.toggleShow}
    className="block mx-auto m-2 bg-purple-600 hover:bg-purple-700
text-white font-bold py-2 px-4 rounded">
```

```
        + Add Customer
        </button>

      <Modal
        show={props.show}
        onHide={handleClose}
        backdrop="static"
        keyboard={false}
      >
        <Modal.Header closeButton>
          <Modal.Title>Add Customer</Modal.Title>
        </Modal.Header>
        <Modal.Body>
        <form
        onSubmit={(e)=>{
          e.preventDefault();
          setName('');
          setIndustry('');
          props.newCustomer(name, industry);

        }}

        id = 'editmodal'className="w-full max-w-sm">
  <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="name">
        Full Name
      </label>
    </div>
    <div className="md:w-2/3">
      <input className="bg-gray-200 appearance-none border-2
border-gray-200
      rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
      id="name"
      placeholder='Name Here'
      type="text"
      value={name}
```

```
      onChange={(e)=>{setName(e.target.value)}}
      />
    </div>
              </div>
              <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
      <label className="block text-gray-500 font-bold md:text-right mb-1
md:mb-0 pr-4" for="industry">
        Industry
      </label>
    </div>
    <div className="md:w-2/3">
      <input className="bg-gray-200 appearance-none border-2
border-gray-200 rounded
      w-full py-2 px-4 text-gray-700 leading-tight focus:outline-none
focus:bg-white
      focus:border-purple-500"
      id="Industry"
      placeholder="Industry"
      type="text"
      value={industry}
      onChange={(e)=>{setIndustry(e.target.value)}}
      />
      </div>
      </div>
      <div className="md:flex md:items-center mb-6">
    <div className="md:w-1/3">
    </div>
    </div>
          </form>
        </Modal.Body>
        <Modal.Footer>
          <button className="bg-pink-500 hover:bg-pink-700 text-white
font-bold py-2 px-4 rounded"
          onClick={props.toggleShow}>
            Close
            </button>
          <button
          className="bg-purple-600 hover:bg-purple-700 text-white
```

```
font-bold py-2 px-4 rounded"
          form= "editmodal">
            Add
          </button>
        </Modal.Footer>
      </Modal>
    </>
  );
}
```

After making the modifications mentioned above, the modal should now close automatically. However, one more issue: when a new customer is added, it does not appear on the screen instantly. Instead, we must refresh the page to view the update.

In the next steps, we will address this issue by changing the Customers.js code.

```
function newCustomer(name, industry) {
    const data = {name: name, industry: industry};
    const url = baseUrl + 'api/customers/';
    fetch(
      url, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
      }
    ).then((response) => {
      if(!response.ok)
      throw new Error('Something went wrong');
      return response.json();

    } ).then((data)=> {
        toggleShow();
        console.log(data);
        setCustomers([...customers, data.customer]);
    }).catch((e)=>{
      console.log(e);
    });
```

```
}
```

Now, there will be no need to refresh the page to let the new customer appear on the web page.

Dynamic Edit Form to Update API Data

Currently, we can only add and remove customers from the list. In this section, we'll learn how to update our customer list. For this reason, we made the following adjustments to the Customer.js component.

```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';
import { baseUrl } from '../shared';

export default function Customer(){
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    useEffect(() => {
        console.log('useEffect');
        const url = baseUrl + 'api/customers/' + id;
        fetch(url)
        .then((response) => {

            if (response.status === 404)

            navigate('/404');
            return response.json();
        })
        .then((data) => {
            setCustomer(data.customer);
        });
    }, []);
    return(
        <div className = "bg-purple-300 min-h-screen py-2">
        <>
        { customer ? <div>
            <input className="m-2 block px-2"
```

```
            type='text' value={customer.id}/>

            <input className="m-2 block px-2"
            type='text' value={customer.name}/>

            <input className="m-2 block px-2"
            type='text' value={customer.industry}/>
            </div> : null }
<button
   onClick={(e) => {
      const url = baseUrl + 'api/customers/' + id;
      fetch (url, { method: 'DELETE'}).then((response) => {
         if(!response.ok) {
            throw new Error('Something went wrong')
         }
        navigate('/customers');
      })
      .catch((e) => {console.log(e)});
   }}
   > Delete
</button>
<br/>
            <Link to = "/customers">Go Back</Link>
            </>
            </div>

   );

}
```

Now, we can see that our customer's data is updatable now:

The next step is to enable the Save and Cancel button for our updates in the form. Here is the code that enables both buttons. However, the Save button needs some changes

to change the data from the backend.

```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';
import { baseUrl } from '../shared';

export default function Customer(){
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    const [tempCustomer, setTempCustomer] = useState();
    const [changed, setChanged] = useState(false);

    useEffect(() => {
        console.log('customer', customer);
        console.log('tempcustomer', tempCustomer);
        console.log('changed');
    });


    useEffect(() => {
        console.log('useEffect');
        const url = baseUrl + 'api/customers/' + id;
        fetch(url)
        .then((response) => {

            if (response.status === 404)

            navigate('/404');
            return response.json();
        })
        .then((data) => {
            setCustomer(data.customer);
            setTempCustomer(data.customer);
        });
    }, []);
    return(
        <div className = "bg-purple-300 min-h-screen py-2">
        <>
```

```
        { customer ? <div>
            <input className="m-2 block px-2"
            type='text' value={tempCustomer.id}/>

            <input className="m-2 block px-2"
            type='text' value={tempCustomer.name}
            onChange={(e)=>{
                setChanged(true);
                setTempCustomer({
                    ...tempCustomer, name: (e.target.value)
                })
            }}
            />

            <input className="m-2 block px-2"
            type='text' value={tempCustomer.industry}
            onChange={(e)=>{
                setChanged(true);
                setTempCustomer({
                    ...tempCustomer, industry: (e.target.value)
                })
            }}
            />
            {changed ? <>
            <button
            className='block'
            onClick={(e)=>{
                setTempCustomer({...customer});
            }}>Cancel</button>
            <button>Save</button>
            </> : null}
            </div> : null }
<button
    onClick={(e) => {
        const url = baseUrl + 'api/customers/' + id;
        fetch (url, { method: 'DELETE'}).then((response) => {
            if(!response.ok) {
                throw new Error('Something went wrong')
            }
```

```
            navigate('/customers');
          })
          .catch((e) => {console.log(e)});
    }}
    > Delete
</button>
<br/>
            <Link to = "/customers">Go Back</Link>
            </>
            </div>

    );

}
```

To enable the save button, we added new function "**updateCustomer**" to the Customer.js file as under:

```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';
import { baseUrl } from '../shared';

export default function Customer(){
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    const [tempCustomer, setTempCustomer] = useState();
    const [changed, setChanged] = useState(false);

    useEffect(() => {
        console.log('customer', customer);
```

```
    console.log('tempcustomer', tempCustomer);
    console.log('changed');
});


useEffect(() => {
    console.log('useEffect');
    const url = baseUrl + 'api/customers/' + id;
    fetch(url)
    .then((response) => {

        if (response.status === 404)

        navigate('/404');
        return response.json();
    })
    .then((data) => {
        setCustomer(data.customer);
        setTempCustomer(data.customer);
    });
}, []);

function updateCustomer() {
    const url = baseUrl + 'api/customers/' + id;
    fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(tempCustomer)

    }).then((response) => {
        return response.json();
    }).then((data) => {
        setChanged(false);
        console.log(data);
    }).catch();
}
```

```
  return(
    <div className = "bg-purple-300 min-h-screen py-2">
    <>
    { customer ? <div>
      <input className="m-2 block px-2"
      type='text' value={tempCustomer.id}/>

      <input className="m-2 block px-2"
      type='text' value={tempCustomer.name}
      onChange={(e)=>{
        setChanged(true);
        setTempCustomer({
          ...tempCustomer, name: (e.target.value)
        })
      }}
      />

      <input className="m-2 block px-2"
      type='text' value={tempCustomer.industry}
      onChange={(e)=>{
        setChanged(true);
        setTempCustomer({
          ...tempCustomer, industry: (e.target.value)
        })
      }}
      />
      {changed ? <>
      <button
      className='block'
      onClick={(e)=>{
        setTempCustomer({...customer});
        setChanged(false);
      }}>Cancel</button>

      <button onClick={updateCustomer}>
        Save</button>
      </> : null}
      </div> : null }
<button
```

```
    onClick={(e) => {
        const url = baseUrl + 'api/customers/' + id;
        fetch (url, { method: 'DELETE'}).then((response) => {
            if(!response.ok) {
                throw new Error('Something went wrong')
            }
           navigate('/customers');
        })
        .catch((e) => {console.log(e)});
    }}
    > Delete
</button>
<br/>
            <Link to = "/customers">Go Back</Link>
            </>
            </div>

    );

}
```

Comparing State Objects

In this section, we'll implement a comparison between the new data and the old data by making some updates to our code. **We've added a new function called compareCustomers, as shown below:**

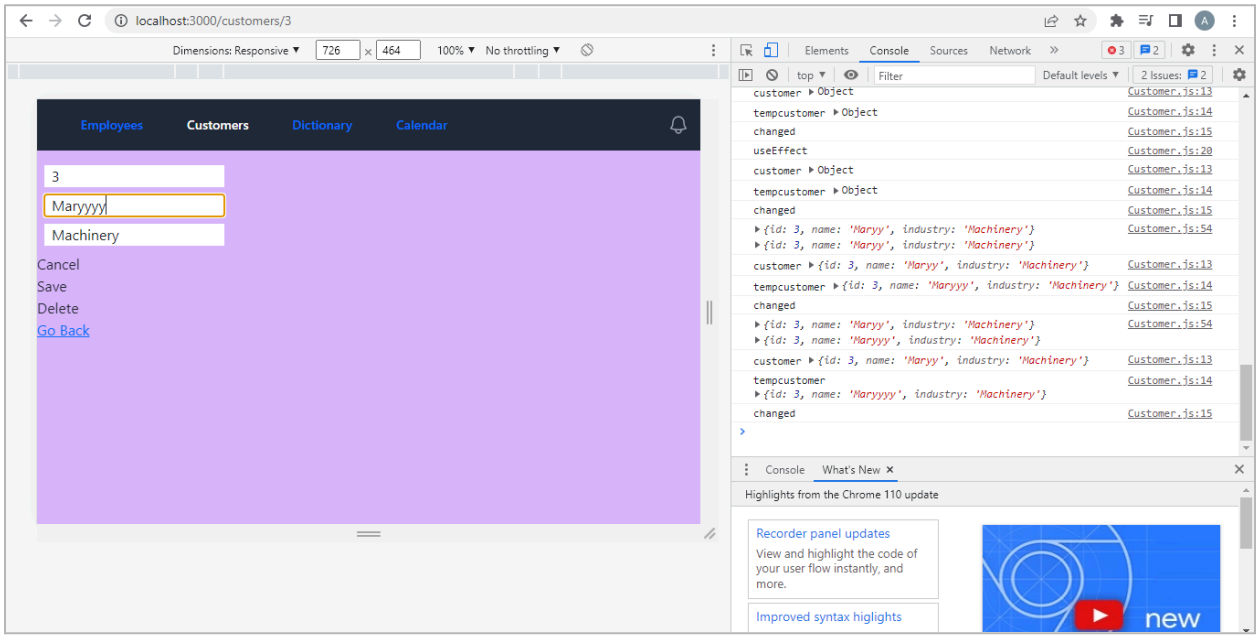
```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';
import { baseUrl } from '../shared';

export default function Customer(){
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    const [tempCustomer, setTempCustomer] = useState();
    const [changed, setChanged] = useState(false);
```

```
useEffect(() => {
    console.log('customer', customer);
    console.log('tempcustomer', tempCustomer);
    console.log('changed');
});


useEffect(() => {
    console.log('useEffect');
    const url = baseUrl + 'api/customers/' + id;
    fetch(url)
    .then((response) => {

        if (response.status === 404)

        navigate('/404');
        return response.json();
    })
    .then((data) => {
        setCustomer(data.customer);
        setTempCustomer(data.customer);
    });
}, []);

function updateCustomer() {
    const url = baseUrl + 'api/customers/' + id;
    fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(tempCustomer)

    }).then((response) => {
        return response.json();
    }).then((data) => {
        setChanged(false);
        console.log(data);
    }).catch();
```

```
}

function compareCustomers(){
    console.log(customer, tempCustomer);
    let equal = true;
    if(customer.name !== tempCustomer.name)
    {
        equal = false;
    }
    if(customer.industry !== tempCustomer.industry){
        equal = false;
    }
    if(equal){
        setChanged(false);
    }
}

return(
    <div className = "bg-purple-300 min-h-screen py-2">
    <>
    { customer ? <div>
        <input className="m-2 block px-2"
        type='text' value={tempCustomer.id}/>

        <input className="m-2 block px-2"
        type='text' value={tempCustomer.name}
        onChange={(e)=>{
            setChanged(true);
            setTempCustomer({
                ...tempCustomer, name: (e.target.value)
            })
            compareCustomers();
        }}
        />

        <input className="m-2 block px-2"
        type='text' value={tempCustomer.industry}
        onChange={(e)=>{
            setChanged(true);
```

```
                setTempCustomer({
                    ...tempCustomer, industry: (e.target.value)
                })
                compareCustomers();
            }}
            />
            {changed ? <>
            <button
            className='block'
            onClick={(e)=>{
                setTempCustomer({...customer});
                setChanged(false);
            }}>Cancel</button>

            <button onClick={updateCustomer}>
                Save</button>
            </> : null}
            </div> : null }
<button
    onClick={(e) => {
        const url = baseUrl + 'api/customers/' + id;
        fetch (url, { method: 'DELETE'}).then((response) => {
            if(!response.ok) {
                throw new Error('Something went wrong')
            }
           navigate('/customers');
        })
        .catch((e) => {console.log(e)});
    }}
    > Delete
</button>
<br/>
            <Link to = "/customers">Go Back</Link>
            </>
            </div>

    );

}
```

You can see in the image that the comparison occurs in the developer console after the above changes.



The next step is to make adjustments that will cause the save and cancel buttons to disappear when you undo the changes. Alternatively, if you make no modifications, these buttons will not appear. **Here is the code that you need to add to the Customer.js file.**

```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';
import { baseUrl } from '../shared';

export default function Customer(){
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    const [tempCustomer, setTempCustomer] = useState();
    const [changed, setChanged] = useState(false);

    useEffect(() => {
        if(!customer) return;
        if(!tempCustomer) return;
        console.log(customer, tempCustomer);
        let equal = true;
```

```
    if(customer.name !== tempCustomer.name) equal = false;
    if(customer.industry !== tempCustomer.industry) equal = false;
    if(equal) setChanged(false);

});


useEffect(() => {
    console.log('useEffect');
    const url = baseUrl + 'api/customers/' + id;
    fetch(url)
    .then((response) => {

        if (response.status === 404)

        navigate('/404');
        return response.json();
    })
    .then((data) => {
        setCustomer(data.customer);
        setTempCustomer(data.customer);
    });
}, []);

function updateCustomer() {
    const url = baseUrl + 'api/customers/' + id;
    fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(tempCustomer)

    }).then((response) => {
        return response.json();
    }).then((data) => {
        setChanged(false);
        console.log(data);
    }).catch();
```

```
}

return(
    <div className = "bg-purple-300 min-h-screen py-2">
    <>
    { customer ? <div>
        <input className="m-2 block px-2"
        type='text' value={tempCustomer.id}/>

        <input className="m-2 block px-2"
        type='text' value={tempCustomer.name}
        onChange={(e)=>{
            setChanged(true);
            setTempCustomer({
                ...tempCustomer, name: (e.target.value)
            })

        }}
        />

        <input className="m-2 block px-2"
        type='text' value={tempCustomer.industry}
        onChange={(e)=>{
            setChanged(true);
            setTempCustomer({
                ...tempCustomer, industry: (e.target.value)
            })

        }}
        />
        {changed ? <>
        <button
        className='block'
        onClick={(e)=>{
            setTempCustomer({...customer});
            setChanged(false);
        }}>Cancel</button>

        <button onClick={updateCustomer}>
```

```
            Save</button>
          </> : null}
          </div> : null }
<button
   onClick={(e) => {
      const url = baseUrl + 'api/customers/' + id;
      fetch (url, { method: 'DELETE'}).then((response) => {
         if(!response.ok) {
            throw new Error('Something went wrong')
         }
        navigate('/customers');
      })
      .catch((e) => {console.log(e)});
   }}
   > Delete
</button>
<br/>
          <Link to = "/customers">Go Back</Link>
          </>
          </div>

   );

}
```

Display Form Errors on Page

We should see an error when we remove all of the letters from the customer's name in the input form and click save. For this reason, we made additional adjustments to the Customer.js code. **Here are the changes:**

```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';
import { baseUrl } from '../shared';

export default function Customer(){
   const {id} = useParams();
   const navigate = useNavigate();
```

```
const [customer, setCustomer] = useState({});
const [tempCustomer, setTempCustomer] = useState({});
const [changed, setChanged] = useState(false);
const [error, setError] = useState('');

useEffect(() => {
    if(!customer) return;
    if(!tempCustomer) return;
    let equal = true;
    if(customer.name !== tempCustomer.name) equal = false;
    if(customer.industry !== tempCustomer.industry) equal = false;
    if(equal) setChanged(false);

});


useEffect(() => {
    const url = baseUrl + 'api/customers/' + id;
    fetch(url)
    .then((response) => {

        if (response.status === 404) {
            navigate('/404');
```

```
        }
        return response.json();
    })
    .then((data) => {
        setCustomer(data.customer);
        setTempCustomer(data.customer);
    });
}, []);

function updateCustomer() {
    e.preventDefault();
    const url = baseUrl + 'api/customers/' + id;
    fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(tempCustomer)

    }).then((response) => {
        console.log('response', response);
        if(!response.ok) throw new Error('Something went wrong')
        return response.json();

    }).then((data) => {
        setChanged(false);
        console.log(data);
        setError(undefined);
    }).catch((e) => {
        console.log('e', e);
        setError(e.message);
    });
}

return(
    <div className = "bg-purple-300 min-h-screen py-2">
    <>
    { customer ? <div>
```

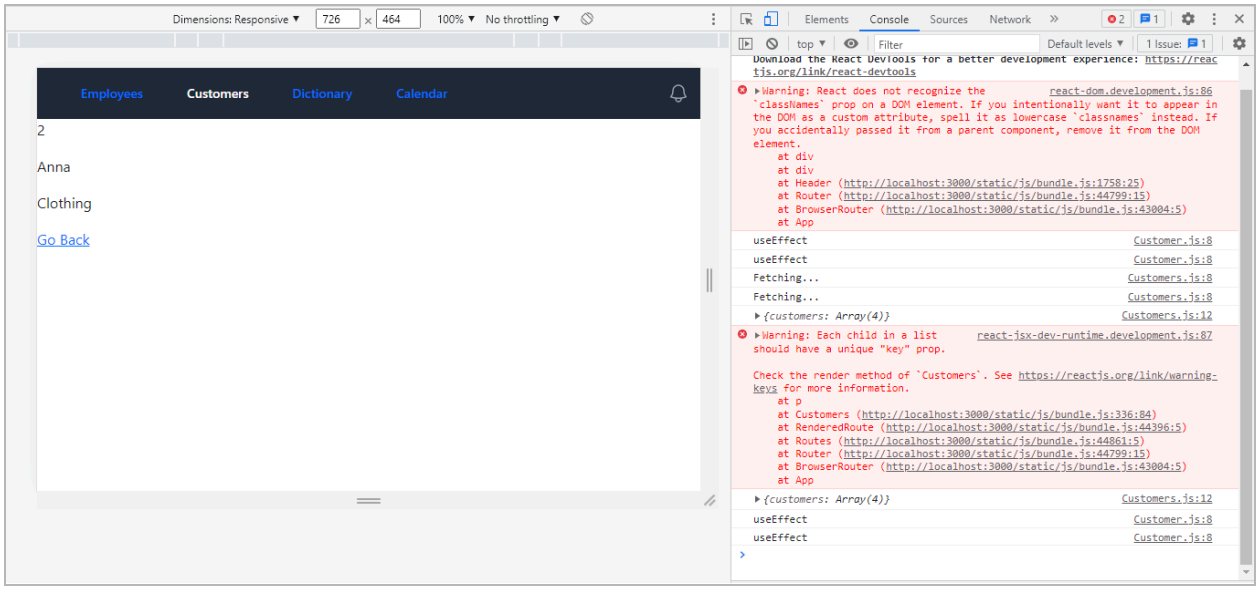
```
            <input className="m-2 block px-2"
            type='text' value={tempCustomer.id}/>

            <input className="m-2 block px-2"
            type='text' value={tempCustomer.name}
            onChange={(e)=>{
                setChanged(true);
                setTempCustomer({
                    ...tempCustomer, name: (e.target.value)
                })

            }}
            />

            <input className="m-2 block px-2"
            type='text' value={tempCustomer.industry}
            onChange={(e)=>{
                setChanged(true);
                setTempCustomer({
                    ...tempCustomer, industry: (e.target.value)
                })

            }}
            />
            {changed ? <>
            <button
            className='block'
            onClick={(e)=>{
                setTempCustomer({...customer});
                setChanged(false);
            }}>Cancel</button>

            <button onClick={updateCustomer}>
                Save</button>
            </> : null}
<button
    onClick={(e) => {
        const url = baseUrl + 'api/customers/' + id;
        fetch (url, { method: 'DELETE'}).then((response) => {
```

```
            if(!response.ok) {
                throw new Error('Something went wrong');
            }
            navigate('/customers');
        })
        .catch((e) => {console.log(e)});
    }}
    > Delete
</button>
{error ? <p>{error}</p> : null}
</div> : null }
<br/>
            <Link to = "/customers/">Go Back</Link>
            </>
            </div>

    );

}
```

It gives us a response as follows:

Tailwind CSS Form and Button Styling

When we click on our customer, it looks like a rough screen with no styling. **You can see one such example below:**



To make it look better, we have done some styling for the Customer.js file as follows:

```
import { useParams, Link, useNavigate } from 'react-router-dom';
import { useEffect, useState } from 'react';
import { baseUrl } from '../shared';

export default function Customer(){
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    const [tempCustomer, setTempCustomer] = useState();
    const [changed, setChanged] = useState(false);
    const [error, setError] = useState();

    useEffect(() => {
        if(!customer) return;
        if(!tempCustomer) return;
        let equal = true;
        if(customer.name !== tempCustomer.name) equal = false;
        if(customer.industry !== tempCustomer.industry) equal = false;
        if(equal) setChanged(false);

    });
```

```
useEffect(() => {
    const url = baseUrl + 'api/customers/' + id;
    fetch(url)
    .then((response) => {

        if (response.status === 404)

        navigate('/404');

        if(!response.ok) {
            throw new Error('Something went wrong, try again');
    }
        return response.json();
    })
    .then((data) => {
        setCustomer(data.customer);
        setTempCustomer(data.customer);
    }).catch((e) => {
        setError(e.message);
    })
}, []);

function updateCustomer(e) {
    e.preventDefault();
    const url = baseUrl + 'api/customers/' + id;
    fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(tempCustomer)

    }).then((response) => {
        if(!response.ok) throw new Error('Something went wrong')
        return response.json();

    }).then((data) => {
```

```
            setChanged(false);
            setError(undefined);
        }).catch((e) => {
            setError(e.message);
        });
    }

    return(
        <div className = "bg-purple-300 min-h-screen py-2">
        <>

        { customer ?

        <div className="p-3">

            <form className="w-full max-w-sm"
                id="customer" onSubmit={updateCustomer}>
                    <div className="md:flex md:items-center mb-6">
                    <div className="md:w-1/4">
                <label for="name">Name</label>
                </div>
                <div className="md:w-3/4">
            <input    id="name"
           className="bg-gray-200 appearance-none border-2
border-gray-200
          rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"

            type='text'
            value={tempCustomer.name}
            onChange={(e)=>{
                setChanged(true);
                setTempCustomer({
                    ...tempCustomer, name: (e.target.value)
                });

            }}
            />
            </div>
```

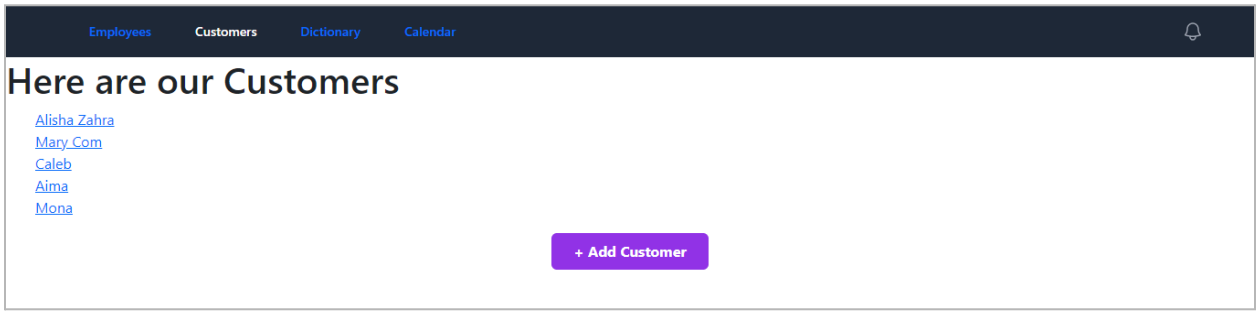
```
            </div>

            <div className="md:flex md:items-center mb-6">
            <div className="md:w-1/4">
            <label for="industry">industry</label>
            </div>
            <div className="md:w-3/4">
            <input
            id="industry"
            className="bg-gray-200 appearance-none border-2
border-gray-200
     rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"

            type='text' value={tempCustomer.industry}
            onChange={(e)=>{
               setChanged(true);
               setTempCustomer({
                  ...tempCustomer, industry: (e.target.value)
               })

            }}
            />
            </div>
            </div>
            </form>
            {changed ? <>
               <div className="mb-2">
            <button
            className="bg-purple-500 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded mr-2"
            onClick={(e)=>{
               setTempCustomer({...customer});
               setChanged(false);
            }}>Cancel</button>
            <button
            className="bg-purple-500 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"
```

```
            form="customer">
                Save</button>
                </div>
            </> : null}
            <div>
<button className="bg-purple-500 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"
    onClick={(e) => {
        const url = baseUrl + 'api/customers/' + id;
        fetch (url, { method: 'DELETE'}).then((response) => {
            if(!response.ok) {
                throw new Error('Something went wrong');
            }
            setError(undefined);
           navigate('/customers');
        })
        .catch((e) => {
            setError(e.message)
        });
    }}
    > Delete
</button>
</div>
</div> : null }
{error ? <p>{error}</p> : null}
<br/>
<div className="p-3">
            <Link to = "/customers/">

                <button className=" no-underline bg-purple-500
hover:bg-purple-700 text-white font-bold py-2 px-4 rounded"
            >

                    ← Go Back

                </button>
                </Link>
            </div>
            </>
```

```
        </div>


    );
}
```

Now, when we click on our customer, it will look like this:

The next step is to style the page so that we see all our customers.



We only changed the Link tag at the bottom of our code for the above page. **Here is the changed part:**

Note that these changes were made to the Customers.js component, not the Customer.js component.

```
<Link to = {"/customers/" + customer.id }>
    <button className=" no-underline bg-purple-500
hover:bg-purple-700 text-white font-bold py-2 px-4 rounded"
  >
    {customer.name}
    </button>
```

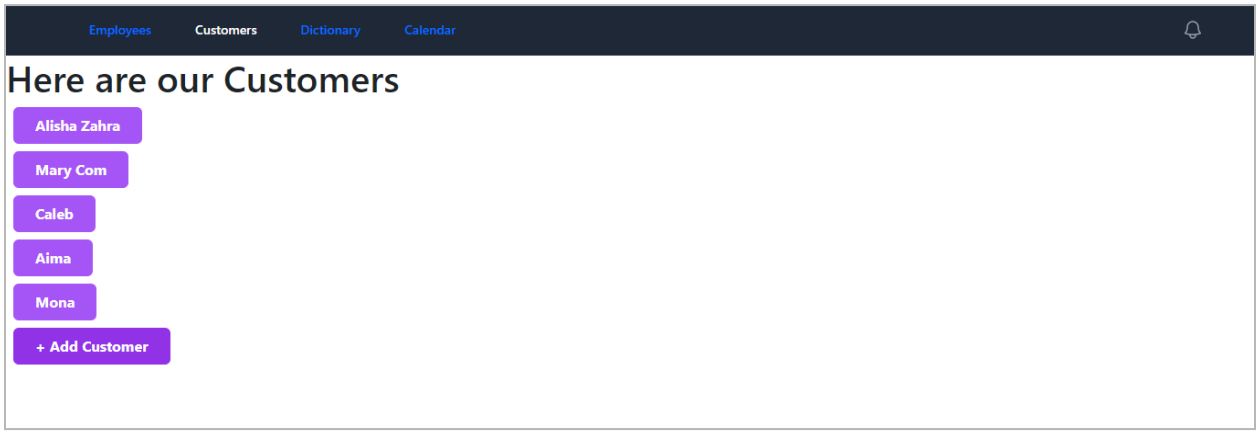
```
    </Link>
```

Then, we opened the AddCustomer.js component and made some minor changes to the following part:

```
<button onClick={props.toggleShow}
   className=" block m-2 bg-purple-600 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded">
       + Add Customer
       </button>
```

We only removed the mx-auto property from the className in the above code.

Our page looks like below now:



Intro to JWTs and Authentication (JSON Web Tokens)

JSON Web Tokens (JWTs) provide a secure and efficient way to exchange information between parties using a compact and URL-safe JSON object.

JWTs are commonly used in web applications for authentication and data transmission due to their simplicity and effectiveness. A JWT comprises three key parts: the header, the payload, and the signature.

The header contains metadata about the token, such as the token type and the algorithm used for signing. The payload holds the transmitted data, like user information or authorization details.

The signature is a hashed combination of the header, payload, and a secret key, ensuring the token has not been tampered with.

JWTs are particularly useful for authentication and authorization in web applications. When a user logs in, the server generates a JWT containing user information and sends it to the client.

The client can then use this JWT in future requests, allowing the server to verify the user's identity and grant access to protected resources.

Now, let's get started with JWTs in our backend. First, we opened the backend and ran the following command within the virtual environment.

```
pip install djangorestframework-simplejwt
```

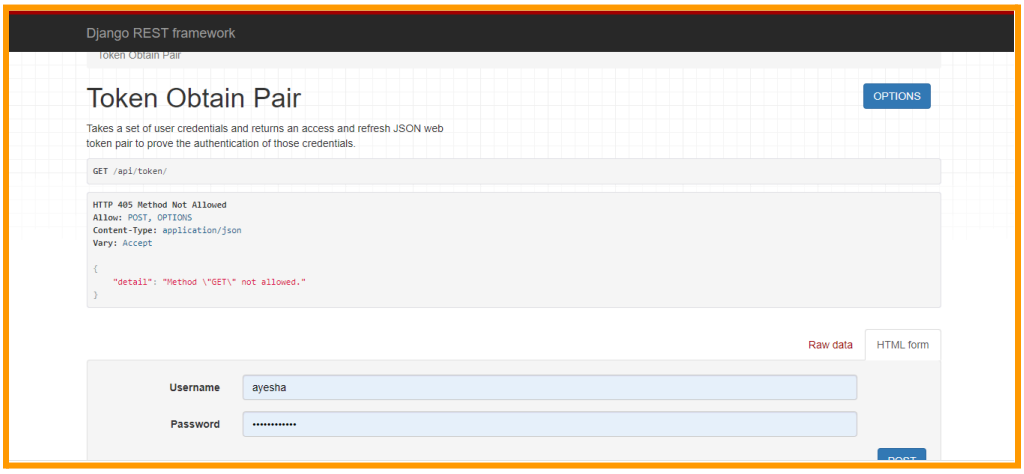
The next step is to open the settings.py file in the backend and add the following under the MIDDLEWARE part.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES':
('rest_framework_simplejwt.authentication.JWTAuthentication',)
}
```

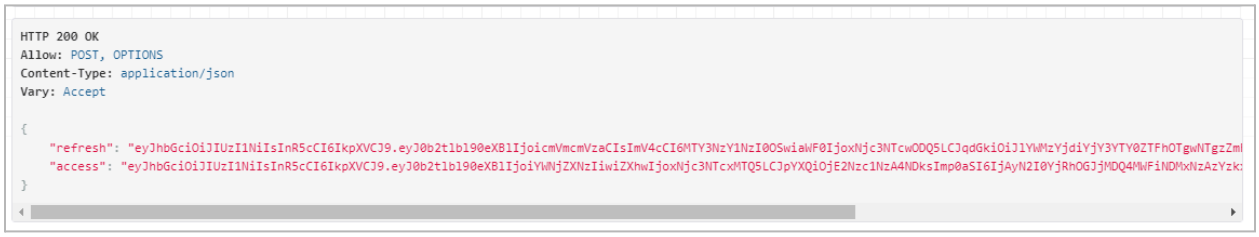
The next step is to make the following changes to the urls.py file:

```
from django.contrib import admin
from django.urls import path
from customers import views
from rest_framework_simplejwt.views import TokenObtainPairView,
TokenRefreshView

urlpatterns = [
    path('api/token/',
TokenObtainPairView.as_view(),name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(),
name='token_refresh'),
    path('admin/', admin.site.urls),
    path('api/customers/', views.customers, name='customers'),
    path('api/customers/<int:id>/', views.customer, name='customer'),
]
```

Now when we visit http://127.0.0.1:8000/api/token/, **we get the following response:**



We must log in above to get tokens as follows:



We can create more users by running the following command:

```
py manage.py createsuperuser
```

The next step is to add the following changes to the views.py file as under:

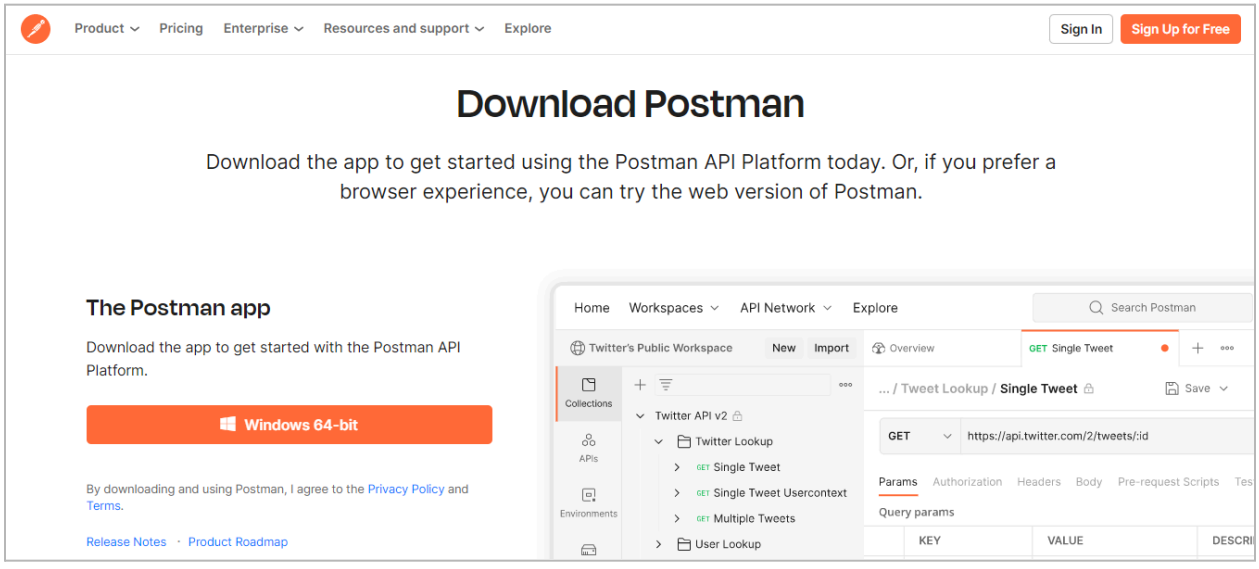
```
from customers.models import Customer
from django.http import JsonResponse, Http404
from customers.serializers import CustomerSerializer
from rest_framework.decorators import api_view, permission_classes
from rest_framework.response import Response
from rest_framework import status
from rest_framework.permissions import IsAuthenticated
```

```
@api_view(['GET', 'POST'])
@permission_classes([IsAuthenticated])
def customers(request):
  if request.method == 'GET':
    data = Customer.objects.all()
    serializer = CustomerSerializer(data, many=True)
    return Response({'customers': serializer.data})

  elif request.method == 'POST':
    serializer = CustomerSerializer(data=request.data)
  if serializer.is_valid():
    serializer.save()
    return Response({'customer': serializer.data},
status=status.HTTP_201_CREATED)
  return Response (serializer.errors,
status=status.HTTP_400_BAD_REQUEST)


@api_view(['GET', 'POST', 'DELETE'])
@permission_classes([IsAuthenticated])

def customer(request, id):
  try:
     data = Customer.objects.get(pk=id)
  except Customer.DoesNotExist:
    return Response(status=status.HTTP_404_NOT_FOUND)

  if request.method == 'GET':
    serializer = CustomerSerializer(data)
    return Response({'customer': serializer.data})
  elif request.method == 'DELETE':
    data.delete()
    return Response(status=status.HTTP_204_NO_CONTENT)
```

```
  elif request.method == 'POST':
    serializer = CustomerSerializer(data, data=request.data)
    if serializer.is_valid():
      serializer.save()
    return Response({'customer': serializer.data})
  return Response (serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
```

It gives us the following output:

To resolve the above issue, we must download postman from [here](#).

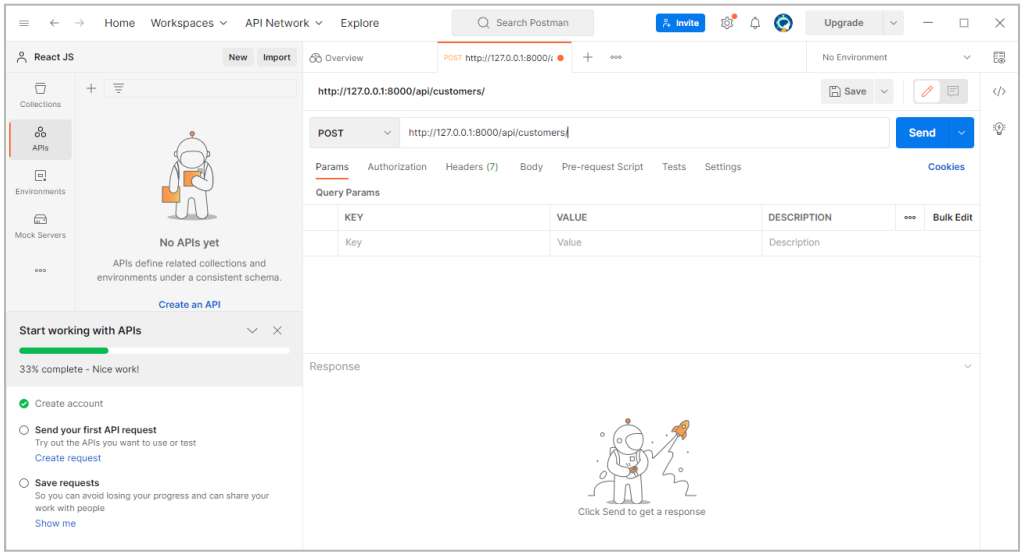


Postman is a widely used collaboration platform that simplifies API development, enabling developers to design, test, and document APIs more efficiently. With its intuitive desktop interface, Postman makes interacting with APIs a breeze. Here's what you can do with the Postman app:

- Create and send HTTP requests using GET, POST, PUT, and DELETE.
- Inspect and analyze API responses in detail.
- Organize API requests into collections for better management.
- Automate API testing with custom test scripts.
- Collaborate seamlessly with team members on API development projects.

Postman is essential for streamlining API workflows and improving collaboration across teams.

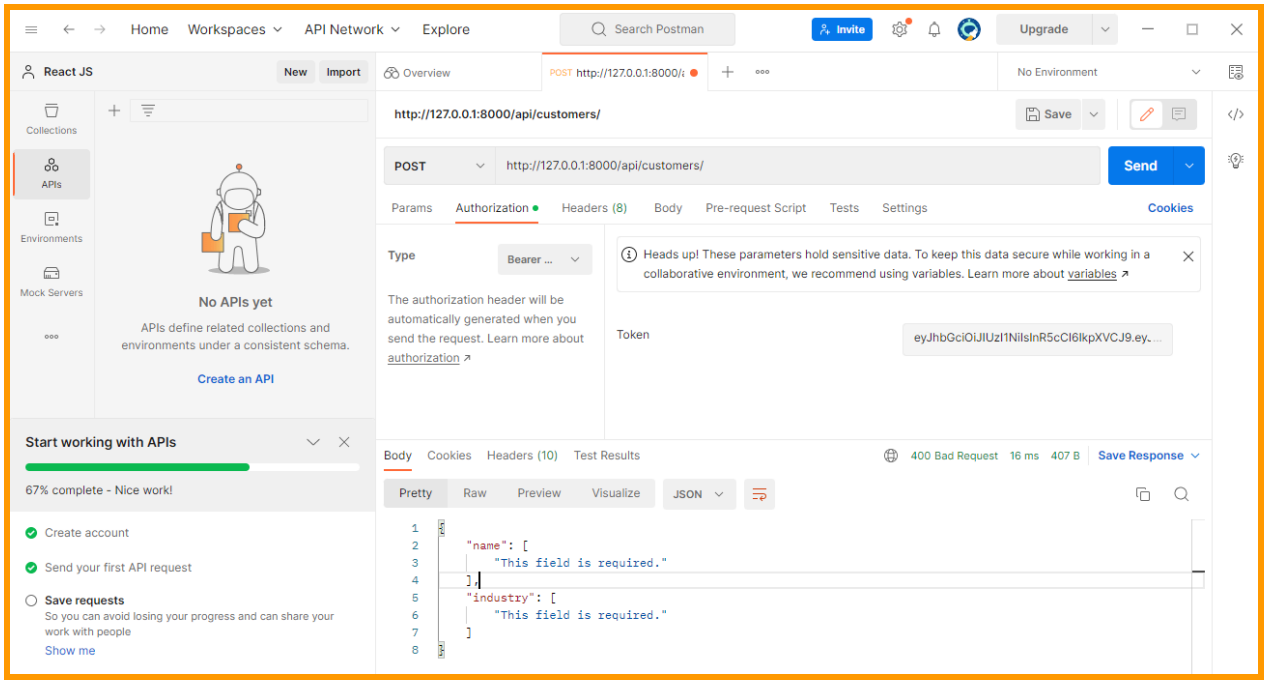
The workspace at the Postman app should look like this:



Now, we enter the URL as shown in the image above:http://127.0.0.1:8000/api/customers/

Then, navigate to the Authorizations>Bearer Token and paste the access token there:

It will show us the response at the bottom as under:



Create a Login Page

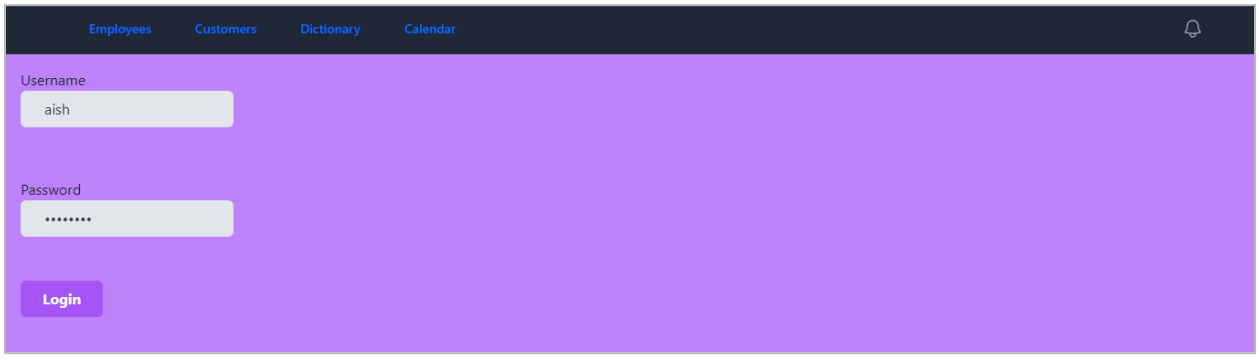
First, we created a Login.js component. Next, we set up a route and imported it into App.js. After that, we added the following code to the new Login.js component:

```
import { useState } from 'react';
import { baseUrl } from '../shared';
export default function Login() {

   const[username, setUsername] = useState(‘’);
   const[password, setPassword] = useState(‘’);

   function login(e){
      e.preventDefault();
      const url = baseUrl + 'api/token/';
      fetch(url, {
         method: 'POST',
         headers: {
            'Content-Type': 'application/json',
         },
         body: JSON.stringify({
            username: username,
```

```
            password: password,
          }),
      }).then((data)=>{
          console.log(data);
      })
  }

 return (
   <form className=" bg-purple-400 min-h-screen"
            id="customer"
            onSubmit={login}>
                <div className="md:flex md:items-center mb-6">
                <div className=" p-3 md:w-1/4">
                  <div>
            <label for="username">Username</label>
            </div>
            <div className="md:w-3/4">
          <input    id="username"
         className=" bg-gray-200 appearance-none border-2
border-gray-200
         rounded w-30 py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"

          type='text'
          value={username}
          onChange={(e)=>{
             setUsername(e.target.value);
          }}
          />
          </div>
          </div>
          </div>

          <div className="md:flex md:items-center mb-6">
          <div className="p-3 md:w-1/4">
          <div>
          <label for="password">Password</label>
          </div>
          <div className="md:w-3/4">
```

```
            <input
            id="password"
            className="bg-gray-200 appearance-none border-2
border-gray-200
     rounded w-30 py-2 px-4 text-gray-700 leading-tight focus:outline-none
focus:bg-white focus:border-purple-500"

            type='password' value={password}
            onChange={(e)=>{

               setPassword(e.target.value);
            }}
            />
            </div>
            <button className="mt-5 bg-purple-500 hover:bg-purple-700
text-white font-bold py-2 px-4 rounded">Login</button>
            </div>
            </div>


            </form>
)
       }
```

Here, we see that our login page is ready.



localStorage and Bearer Auth Tokens

First, we introduced a local storage property into our code inside the Login.js component. You can see the changes made and their output as follows:

```
import { useState } from 'react';
import { baseUrl } from '../shared';
export default function Login() {

   const[username, setUsername] = useState();
   const[password, setPassword] = useState();

   function login(e){
      e.preventDefault();
      const url = baseUrl + 'api/token/';
      fetch(url, {
         method: 'POST',
         headers: {
            'Content-Type': 'application/json',
         },
         body: JSON.stringify({
            username: username,
            password: password,
         }),
      }).then((response)=>{
         return response.json();

      }).then((data)=>{
         localStorage.setItem('access', data.access);
         localStorage.setItem('refresh', data.refresh);
         console.log(localStorage);
      })
   }

 return (
   <form className=" bg-purple-400 min-h-screen"
            id="customer"
            onSubmit={login}>
                 <div className="md:flex md:items-center mb-6">
                 <div className=" p-3 md:w-1/4">
                   <div>
            <label for="username">Username</label>
            </div>
```

```
            <div className="md:w-3/4">
          <input    id="username"
         className=" bg-gray-200 appearance-none border-2
border-gray-200
         rounded w-30 py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"

          type='text'
          value={username}
          onChange={(e)=>{
             setUsername(e.target.value);
          }}
          />
          </div>
          </div>
          </div>

          <div className="md:flex md:items-center mb-6">
          <div className="p-3 md:w-1/4">
          <div>
          <label for="password">Password</label>
          </div>
          <div className="md:w-3/4">
          <input
          id="password"
          className="bg-gray-200 appearance-none border-2
border-gray-200
     rounded w-30 py-2 px-4 text-gray-700 leading-tight focus:outline-none
focus:bg-white focus:border-purple-500"

          type='password' value={password}
          onChange={(e)=>{

             setPassword(e.target.value);
          }}
          />
          </div>
          <button className="mt-5 bg-purple-500 hover:bg-purple-700
text-white font-bold py-2 px-4 rounded">Login</button>
```

```
        </div>
        </div>


        </form>
)
    }
```

This is the output that we get through the above implementations:

We can use the above access as proof that we have access to the API.

useLocation and useNavigate State (Redirect to Previous Page on Login)

This section will introduce a new hook, useLocation, and code update. **First, we modified the Customers.js component, making small adjustments to the initial fetch section as shown below:**

```
if (response.status === 401) {
  navigate('/login', {
    state: {
      previousUrl: '/customers',
    },
  });
}
```

We then made some adjustments to the Login.js page, as follows:

```
import { useState, useEffect } from 'react';
import { baseUrl } from '../shared';
import { useLocation } from 'react-router-dom';
export default function Login() {

    const[username, setUsername] = useState();
    const[password, setPassword] = useState();
    const location = useLocation();

    useEffect(()=>{
        console.log(location);
    })

    function login(e){
        e.preventDefault();
        const url = baseUrl + 'api/token/';
        fetch(url, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({
                username: username,
                password: password,
            }),
        }).then((response)=>{
            return response.json();

        }).then((data)=>{
            localStorage.setItem('access', data.access);
            localStorage.setItem('refresh', data.refresh);
            console.log(localStorage);
        })
    }
```

```
return (
   <form className=" bg-purple-400 min-h-screen"
              id="customer"
              onSubmit={login}>
                   <div className="md:flex md:items-center mb-6">
                   <div className=" p-3 md:w-1/4">
                     <div>
              <label for="username">Username</label>
              </div>
              <div className="md:w-3/4">
           <input    id="username"
          className=" bg-gray-200 appearance-none border-2
border-gray-200
          rounded w-30 py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"

           type='text'
           value={username}
           onChange={(e)=>{
              setUsername(e.target.value);
           }}
           />
           </div>
           </div>
           </div>

           <div className="md:flex md:items-center mb-6">
           <div className="p-3 md:w-1/4">
           <div>
           <label for="password">Password</label>
           </div>
           <div className="md:w-3/4">
           <input
```

```
          id="password"
          className="bg-gray-200 appearance-none border-2
border-gray-200
     rounded w-30 py-2 px-4 text-gray-700 leading-tight focus:outline-none
focus:bg-white focus:border-purple-500"

          type='password' value={password}
          onChange={(e)=>{

             setPassword(e.target.value);
          }}
          />
          </div>
          <button className="mt-5 bg-purple-500 hover:bg-purple-700
text-white font-bold py-2 px-4 rounded">Login</button>
          </div>
          </div>


          </form>
)
      }
```

It directs us to the login page and gives us the following output:

We must replicate this behavior on all pages that redirect to the login page. Let's make the necessary changes. **Search for the pages that include redirection to the login page, and insert the following code using navigate:**

```
{
        navigate('/login',  {
          state: {
            previousUrl: location.pathname,
          }})
      }
```

useContext Hook Introduction

The useContext hook in React enables components to consume the context provided by a Provider component quickly. This allows you to pass data or functions down to any child component in the subtree without manually passing props at each level.

First, we'll define a context, assign a value, and then access that value from another component. **So far, it has produced a positive response:**

Then, we added a login button to our application. We changed several files to accomplish this. You can look at them here.

Changes Made in App.js:

```
import './index.css';
import { createContext, useState } from 'react';
import Header from './components/Header';
import Employees from './Pages/Employees';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Customers from './Pages/Customers';
import Dictionary from './Pages/Dictionary';
import Definition from './Pages/Definition';
import NotFound from './components/NotFound';
import Customer from './Pages/Customer';
import Login from './Pages/Login';

export const LoginContext = createContext();
```

```
function App() {
const [LoggedIn, setLoggedIn] = useState(true);
  return (
    <LoginContext.Provider value={[LoggedIn, setLoggedIn]}>
    <BrowserRouter>
      <Header />
      <Routes>
        <Route path="/employees" element={<Employees />} />
        <Route path= "/dictionary"element={<Dictionary/>}/>
        <Route path= "/definition"element={<Definition/>}/>
        <Route path= "/404"element={<NotFound/>}/>
        <Route path= "*"element={<NotFound/>}/>
        <Route path= "/dictionary/:search"
        element={<Definition/>}
        />
        <Route path="/customers" element={<Customers />} />
        <Route path= "/login"element={<Login/>}/>
        <Route path="/customers/:id" element={<Customer />} />
      </Routes>
    </BrowserRouter>
    </LoginContext.Provider>
  );
}

export default App;
```

Changes Made In the Customers.js:

```
import { useEffect, useState, useContext } from 'react';
import { Link, useNavigate, useLocation } from 'react-router-dom';
import AddCustomer from '../components/AddCustomer';
import { baseUrl } from '../shared';
import { LoginContext } from '../App';
```

```
export default function Customers() {
  const [loggedIn, setLoggedIn] = useContext(LoginContext);
  const [customers, setCustomers] = useState([]);
  const [show, setShow] = useState(false);

  const location = useLocation();
  const navigate = useNavigate();

  function toggleShow() {
    setShow(!show);
  }

  useEffect(() => {
    const url = baseUrl + 'api/customers/';
    fetch(url, {
      headers: {
        'Content-Type': 'application/json',
        Authorization: 'Bearer ' + localStorage.getItem('access'),
      },
    })
      .then((response) => {
        if (response.status === 401) {
          setLoggedIn(false);
          navigate('/login', {
            state: {
              previousUrl: location.pathname,
            },
          });
        }
        return response.json();
      })
      .then((data) => {
        setCustomers(data.customers);
      })
```

```
    .catch((error) => console.log(error));
}, [navigate, location.pathname]);

function newCustomer(name, industry) {
  const data = { name: name, industry: industry };
  const url = baseUrl + 'api/customers/';
  fetch(url, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer ' + localStorage.getItem('access'),
    },
    body: JSON.stringify(data),
  })
    .then((response) => {
      if (!response.ok) {
        throw new Error('Something went wrong');
      }
      return response.json();
    })
    .then((data) => {
      toggleShow();
      setCustomers([...customers, data.customer]);
    })
    .catch((error) => console.log(error));
}

return (
  <>
    <h1>Here are our Customers</h1>
    {customers.map((customer) => (
      <div className="m-2" key={customer.id}>
        <Link to={`/customers/${customer.id}`}>
          <button className="no-underline bg-purple-500
```

```
hover:bg-purple-700 text-white font-bold py-2 px-4 rounded">
              {customer.name}
            </button>
          </Link>
        </div>
      ))}
      <AddCustomer newCustomer={newCustomer} show={show}
toggleShow={toggleShow} />
    </>
  );
}
```

Changes Made In the Customer.js:

```
import { useParams, Link, useNavigate, useLocation} from
'react-router-dom';
import { useContext, useEffect, useState } from 'react';
import { baseUrl } from '../shared';
import NotFound from '../components/NotFound';
import { LoginContext } from '../App';

export default function Customer(){
    const [loggedIn, setLoggedIn] = useContext(LoginContext);
    const {id} = useParams();
    const navigate = useNavigate();
    const [customer, setCustomer] = useState();
    const [tempCustomer, setTempCustomer] = useState();
    const [changed, setChanged] = useState(false);
    const [error, setError] = useState();
    const [notFound, setNotFound] = useState();
    const location = useLocation();

    useEffect(() => {
        if(!customer || !tempCustomer) {
          return;
```

```
    }
    let equal = true;
    if(customer.name !== tempCustomer.name) {
      equal = false;
    }
    if(customer.industry !== tempCustomer.industry) {
      equal = false;
    }
    if(equal) {
      setChanged(false);
    }

});


useEffect(() => {
    const url = baseUrl + 'api/customers/' + id;
    fetch(url, {
        headers: {
          'Content-Type': 'application/json',
          Authorization: 'Bearer ' + localStorage.getItem('access'),
        },
    }

        )
    .then((response) => {

        if (response.status === 404) {

        navigate('/404');
        setNotFound(true);
            } else if (response.status === 401) {
                setLoggedIn(false);
                navigate('/login', {
```

```
                    state: {
                      previousUrl: location.pathname,
                    },
                  });
              }

          if(!response.ok) {
              throw new Error('Something went wrong, try again');
      }
          return response.json();
      })
      .then((data) => {
          setCustomer(data.customer);
          setTempCustomer(data.customer);
      }).catch((e) => {
          setError(e.message);
      })
  }, []);

  function updateCustomer(e) {
      e.preventDefault();
      const url = baseUrl + 'api/customers/' + id;
      fetch(url, {
          method: 'POST',
          headers: {
              'Content-Type': 'application/json',
              Authorization: 'Bearer' + localStorage.getItem('access'),
            },
          body: JSON.stringify(tempCustomer)

      }).then((response) => {
          if (response.status === 401) {
              setLoggedIn(false);
              navigate('/login', {
```

```
                state: {
                  previousUrl: location.pathname,
                },
              });
          }
          if(!response.ok) throw new Error('Something went wrong')
          return response.json();

      }).then((data) => {
          setChanged(false);
          setError(undefined);
      }).catch((e) => {
          setError(e.message);
      });
  }

  return(
      <div className = "bg-purple-300 min-h-screen py-2">
      <>

      { customer ?

      <div className="p-3">

          <form className="w-full max-w-sm"
              id="customer" onSubmit={updateCustomer}>
                  <div className="md:flex md:items-center mb-6">
                  <div className="md:w-1/4">
              <label for="name">Name</label>
              </div>
              <div className="md:w-3/4">
          <input    id="name"
         className="bg-gray-200 appearance-none border-2
border-gray-200
```

```
          rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"

          type='text'
          value={tempCustomer.name}
          onChange={(e)=>{
             setChanged(true);
             setTempCustomer({
                ...tempCustomer, name: (e.target.value)
             });

          }}
          />
          </div>
          </div>

          <div className="md:flex md:items-center mb-6">
          <div className="md:w-1/4">
          <label for="industry">industry</label>
          </div>
          <div className="md:w-3/4">
          <input
          id="industry"
          className="bg-gray-200 appearance-none border-2
border-gray-200
     rounded w-full py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"

          type='text' value={tempCustomer.industry}
          onChange={(e)=>{
             setChanged(true);
             setTempCustomer({
                ...tempCustomer, industry: (e.target.value)
             })
```

```
            }}
            />
            </div>
            </div>
            </form>
            {changed ? <>
               <div className="mb-2">
            <button
            className="bg-purple-500 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded mr-2"
            onClick={(e)=>{
               setTempCustomer({...customer});
               setChanged(false);
            }}>Cancel</button>
            <button
            className="bg-purple-500 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"

            form="customer">
               Save</button>
               </div>
            </> : null}
            <div>
<button className="bg-purple-500 hover:bg-purple-700 text-white
font-bold py-2 px-4 rounded"
   onClick={(e) => {
      const url = baseUrl + 'api/customers/' + id;
      fetch (url, { method: 'DELETE',

      headers: {
         'Content-Type': 'application/json',
         Authorization:
         'Bearer' +
```

```
            localStorage.getItem('access'),
          },

    }).then((response) => {

          if (response.status === 401) {
              setLoggedIn(false);
              navigate('/login', {
                  state: {
                    previousUrl: location.pathname,
                  },
              });
          }


          if(!response.ok) {
              throw new Error('Something went wrong');
          }
          setError(undefined);
          navigate('/customers');
        })
        .catch((e) => {
          setError(e.message)
        });
    }}
    > Delete
</button>
</div>
</div> : null }
{error ? <p>{error}</p> : null}
<br/>
<div className="p-3">
          <Link to = "/customers/">
```

```
            <button className=" no-underline bg-purple-500
hover:bg-purple-700 text-white font-bold py-2 px-4 rounded"
          >

              ← Go Back

            </button>
            </Link>
          </div>
          </>
          </div>


    );

}
```

Changes Made In the Login.js:

```
import { useState, useEffect, useContext } from 'react';
import { baseUrl } from '../shared';
import { useLocation, useNavigate } from 'react-router-dom';
import { LoginContext } from '../App';

export default function Login() {
  const [loggedIn, setLoggedIn] = useContext(LoginContext);
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const location = useLocation();
  const navigate = useNavigate();

  function login(e) {
    e.preventDefault();
    const url = baseUrl + 'api/token/';
    fetch(url, {
```

```
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        username: username,
        password: password,
      }),
    })
      .then((response) => {
        return response.json();
      })
      .then((data) => {
        localStorage.setItem('access', data.access);
        localStorage.setItem('refresh', data.refresh);
        setLoggedIn(true);
        navigate(location?.state?.previousUrl ? location.state.previousUrl :
'/customers');
      });
  }

  return (
    <form
      className="bg-purple-400 min-h-screen"
      id="customer"
      onSubmit={login}
    >
      <div className="md:flex md:items-center mb-6">
        <div className="p-3 md:w-1/4">
          <label htmlFor="username">Username</label>
        </div>
        <div className="md:w-3/4">
          <input
            id="username"
```

```
          className="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-30 py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
          type="text"
          value={username}
          onChange={(e) => {
            setUsername(e.target.value);
          }}
        />
      </div>
    </div>

    <div className="md:flex md:items-center mb-6">
      <div className="p-3 md:w-1/4">
        <label htmlFor="password">Password</label>
      </div>
      <div className="md:w-3/4">
        <input
          id="password"
          className="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-30 py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
          type="password"
          value={password}
          onChange={(e) => {
            setPassword(e.target.value);
          }}
        />
      </div>
    </div>

    <button className="mt-5 bg-purple-500 hover:bg-purple-700
text-white font-bold py-2 px-4 rounded">
      Login
```

```
      </button>
    </form>
  );
}
```

Changes Made In the Header.js:

```
import { useContext, useEffect } from 'react';
import { Disclosure, Menu, Transition } from '@headlessui/react';
import { Bars3Icon, BellIcon, XMarkIcon } from
'@heroicons/react/24/outline';
import { NavLink } from 'react-router-dom';
import { LoginContext } from '../App';


const navigation = [
  { name: 'Employees', href: '/Employees'},
  { name: 'Customers', href: '/Customers'},
  { name: 'Dictionary', href: '/dictionary'},
]

function classNames(...classes) {
  return classes.filter(Boolean).join(' ')
}

export default function Header(props) {
  const [LoggedIn, setLoggedIn] = useContext(LoginContext);

  return (
    <>
    <Disclosure as="nav" className="bg-gray-800">
      {({ open }) => (
        <>
          <div className="mx-auto max-w-7xl px-2 sm:px-6 lg:px-8">
            <div className="relative flex h-14 items-center justify-between">
```

```
            <div className="absolute inset-y-0 left-0 flex items-center
sm:hidden">
              {/* Mobile menu button*/}
              <Disclosure.Button className="inline-flex items-center
justify-center rounded-md p-2 text-gray-400 hover:bg-gray-700
hover:text-white focus:outline-none focus:ring-2 focus:ring-inset
focus:ring-white">
                <span className="sr-only">Open main menu</span>
                {open ? (
                  <XMarkIcon className="block h-6 w-6" aria-hidden="true"
/>
                ) : (
                  <Bars3Icon className="block h-6 w-6" aria-hidden="true" />
                )}
              </Disclosure.Button>
            </div>
            <div className="flex flex-1 items-center justify-center
sm:items-stretch sm:justify-start">

              <div className="hidden sm:ml-6 sm:block">
                <div className="flex space-x-4">
                  {navigation.map((item) => (
                    <NavLink
                      key={item.name}
                      to ={item.href}
                        className = {({isActive}) => {
                          return (
                            'px-3 py-2 rounded-md text-sm font-medium
no-underline' +
                            (!isActive
                              ? 'text-gray-300 hover:bg-gray-700 hover:text-white
no-underline'
                              : 'bg-gray-900 text-white no-underline')
                            );
```

```
                    }}

                  >
                    {item.name}
                  </NavLink>
                ))}
                 <NavLink

                    to ={LoggedIn ? '/logout' : '/login'}
                    className= 'px-3 py-2 rounded-md text-sm font-medium
no-underline text-gray-300 hover:bg-gray-700 hover:text-white
no-underline'
                  >
                    {LoggedIn ? 'Logout' : 'Login'}

                  </NavLink>
                </div>
              </div>
            </div>
            <div className="absolute inset-y-0 right-0 flex items-center pr-2
sm:static sm:inset-auto sm:ml-6 sm:pr-0">
              <button
                type="button"
                className="rounded-full bg-gray-800 p-1 text-gray-400
hover:text-white focus:outline-none focus:ring-2 focus:ring-white
focus:ring-offset-2 focus:ring-offset-gray-800"
              >
                <span className="sr-only">View notifications</span>
                <BellIcon className="h-6 w-6" aria-hidden="true" />
              </button>
</div>
</div>
</div>
        <Disclosure.Panel className="sm:hidden">
```

```
          <div className="space-y-1 px-2 pt-2 pb-3">
            {navigation.map((item) => (

<NavLink
key={item.name}
to ={item.href}
 className = {({isActive}) => {
   return (
    'block px-3 py-2 rounded-md text-base font-medium no-underline' +
    (!isActive
      ? 'text-gray-300 hover:bg-gray-700 hover:text-white no-underline'
      : 'bg-gray-900 text-white no-underline')
    );
 }} >
                {item.name}
                </NavLink>
            ))}
            <NavLink

                  to ={LoggedIn ? '/logout' : '/login'}
                  className= 'block px-3 py-2 rounded-md text-base
font-medium no-underline text-gray-300 hover:bg-gray-700
hover:text-white no-underline'
                >
                  {LoggedIn ? 'Logout' : 'Login'}

                </NavLink>
          </div>
        </Disclosure.Panel>
      </>
    )}
   </Disclosure>
   <div className= "bg-gray-300">
   <div classNames="bg-purple-300 min-h-screen px-2 py-2">
```

```
      {props.children}
      </div>
      </div>
    </>
  );
}
```

We still need to do some styling. **Here is what our output looks like:**

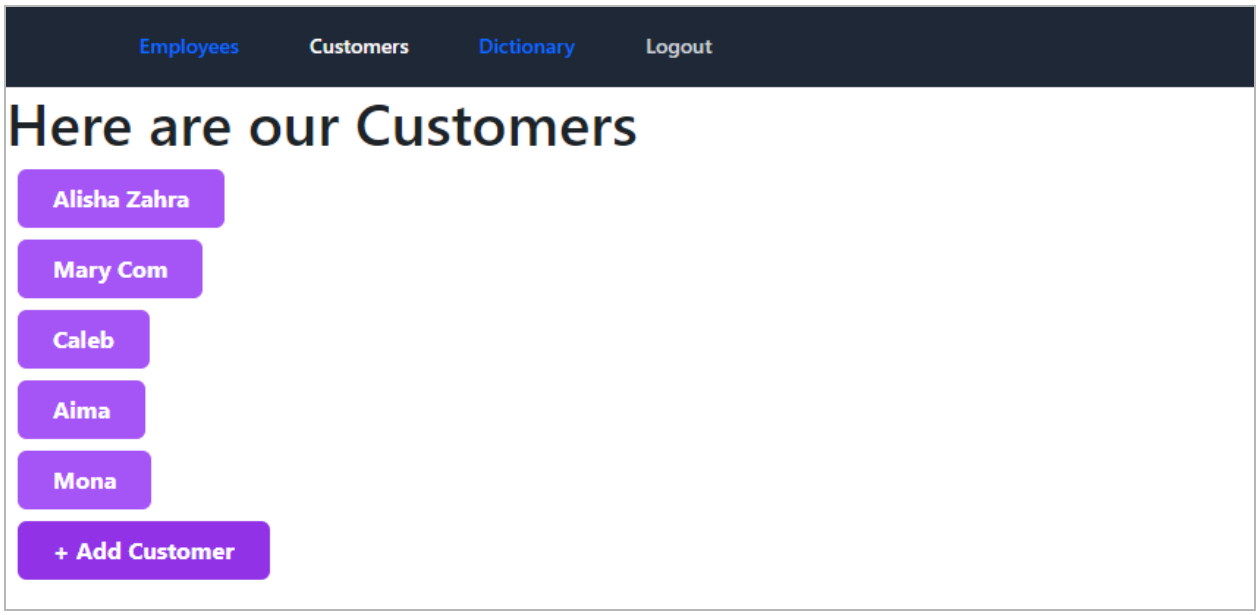
Create a Logout Button

In the previous section, we encountered an issue where visiting the Employees page redirected us to the logout page. **We'll address this now. First, we made the following changes to the App.js file:**

```
import './index.css';
import { createContext, useState } from 'react';
import Header from './components/Header';
import Employees from './Pages/Employees';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Customers from './Pages/Customers';
import Dictionary from './Pages/Dictionary';
import Definition from './Pages/Definition';
import NotFound from './components/NotFound';
import Customer from './Pages/Customer';
import Login from './Pages/Login';

export const LoginContext = createContext();
```

```
function App() {
const [loggedIn, setLoggedIn] = useState(localStorage.access ? true :
false);
  return (
    <LoginContext.Provider value={[loggedIn, setLoggedIn]}>
    <BrowserRouter>
      <Header />
      <Routes>
        <Route path="/employees" element={<Employees />} />
        <Route path= "/dictionary"element={<Dictionary/>}/>
        <Route path= "/definition"element={<Definition/>}/>
        <Route path= "/404"element={<NotFound/>}/>
        <Route path= "*"element={<NotFound/>}/>
        <Route path= "/dictionary/:search"
        element={<Definition/>}
        />
        <Route path="/customers" element={<Customers />} />
        <Route path= "/login"element={<Login/>}/>
        <Route path="/customers/:id" element={<Customer />} />
      </Routes>
    </BrowserRouter>
    </LoginContext.Provider>
  );
}

export default App;
```

Now, when we click Login, we are given a Logout button. **You can see it below:**



Then, we clear the local storage in the developer console through the following command:

```
localstorage.clear()
```

Now, when we visit other pages, a Login button will appear. However, if the local storage isn't cleared manually, it will continue to show a Logout button. We'll implement a function that automatically clears the local storage to resolve this. **Let's get started.**

Here's the code to clear the local storage. **Be sure to add this code in the Header.js file:**

```
import { useContext, useEffect } from 'react'
import { Disclosure, Menu, Transition } from '@headlessui/react'
import { Bars3Icon, BellIcon, XMarkIcon } from
'@heroicons/react/24/outline'
import { NavLink } from 'react-router-dom'
import { LoginContext } from '../App'


const navigation = [
  { name: 'Employees', href: '/Employees'},
  { name: 'Customers', href: '/Customers'},
```

```
  { name: 'Dictionary', href: '/dictionary'},
]

function classNames(...classes) {
  return classes.filter(Boolean).join(' ')
}

export default function Header(props) {
  const [LoggedIn, setLoggedIn] = useContext(LoginContext);

  return (
    <>
    <Disclosure as="nav" className="bg-gray-800">
      {({ open }) => (
        <>
          <div className="mx-auto max-w-7xl px-2 sm:px-6 lg:px-8">
            <div className="relative flex h-14 items-center justify-between">
              <div className="absolute inset-y-0 left-0 flex items-center
sm:hidden">
                {/* Mobile menu button*/}
                <Disclosure.Button className="inline-flex items-center
justify-center rounded-md p-2 text-gray-400 hover:bg-gray-700
hover:text-white focus:outline-none focus:ring-2 focus:ring-inset
focus:ring-white">
                  <span className="sr-only">Open main menu</span>
                  {open ? (
                    <XMarkIcon className="block h-6 w-6" aria-hidden="true"
/>
                  ) : (
                    <Bars3Icon className="block h-6 w-6" aria-hidden="true" />
                  )}
                </Disclosure.Button>
              </div>
              <div className="flex flex-1 items-center justify-center
```

```
sm:items-stretch sm:justify-start">

            <div className="hidden sm:ml-6 sm:block">
              <div className="flex space-x-4">
              {  /* className={
                    classNames(
                    item.current
                    ? 'no-underline'
                    : 'no-underline'
                    ,

                    )}*/}
                {navigation.map((item) => (
                  <NavLink
                    key={item.name}
                    to ={item.href}
                      className = {({isActive}) => {
                        return (
                          'px-3 py-2 rounded-md text-sm font-medium
no-underline' +
                          (!isActive
                            ? 'text-gray-300 hover:bg-gray-700 hover:text-white
no-underline'
                            : 'bg-gray-900 text-white no-underline')
                          );
                      }}

                    >
                      {item.name}
                    </NavLink>
                  ))}
                { LoggedIn ?
                  <NavLink
```

```
                    to ={'/login'}
                    onClick={() => {
                      console.log('logging out...');
                      setLoggedIn(false);
                      localStorage.clear()
                    }}
                    className= 'px-3 py-2 rounded-md text-sm font-medium
no-underline text-gray-300 hover:bg-gray-700 hover:text-white
no-underline'

                  >
                    Logout

                  </NavLink> :

                  <NavLink

                    to ={'/login'}
                    className= 'px-3 py-2 rounded-md text-sm font-medium
no-underline text-gray-300 hover:bg-gray-700 hover:text-white
no-underline'

                  >
                    Login

                  </NavLink> }


                </div>
              </div>
            </div>
            <div className="absolute inset-y-0 right-0 flex items-center pr-2
sm:static sm:inset-auto sm:ml-6 sm:pr-0">
              <button
```

```
              type="button"
              className="rounded-full bg-gray-800 p-1 text-gray-400
hover:text-white focus:outline-none focus:ring-2 focus:ring-white
focus:ring-offset-2 focus:ring-offset-gray-800"
            >
              <span className="sr-only">View notifications</span>
              <BellIcon className="h-6 w-6" aria-hidden="true" />
            </button>
</div>
</div>
</div>
        <Disclosure.Panel className="sm:hidden">
          <div className="space-y-1 px-2 pt-2 pb-3">
            {navigation.map((item) => (

<NavLink
key={item.name}
to ={item.href}
 className = {({isActive}) => {
   return (
     'block px-3 py-2 rounded-md text-base font-medium no-underline' +
     (!isActive
       ? 'text-gray-300 hover:bg-gray-700 hover:text-white no-underline'
       : 'bg-gray-900 text-white no-underline')
     );
 }} >
              {item.name}
              </NavLink>
            ))}
            { LoggedIn ?
            <NavLink

                to ={'/logout'}
                className= 'block px-3 py-2 rounded-md text-base
```

```
font-medium no-underline text-gray-300 hover:bg-gray-700
hover:text-white no-underline'
                >
                 Logout

                </NavLink>
                :
                <NavLink

                  to ={'/login'}
                  className= 'block px-3 py-2 rounded-md text-base
font-medium no-underline text-gray-300 hover:bg-gray-700
hover:text-white no-underline'
                >
                 Login

                </NavLink> }
            </div>
          </Disclosure.Panel>
        </>
      )}
    </Disclosure>
    <div className= "bg-gray-300">
    <div classNames="bg-purple-300 min-h-screen px-2 py-2">
      {props.children}
      </div>
      </div>
    </>
  );
}
```

To keep it working fine, we made a few more changes to our codes in the Header.js file and the App.js file.

Changed made to the Header.js:

```
import { useContext, useEffect } from 'react'
import { Disclosure, Menu, Transition } from '@headlessui/react'
import { Bars3Icon, BellIcon, XMarkIcon } from
'@heroicons/react/24/outline'
import { NavLink } from 'react-router-dom'
import { LoginContext } from '../App'

const navigation = [
  { name: 'Employees', href: '/Employees'},
  { name: 'Customers', href: '/Customers'},
  { name: 'Dictionary', href: '/dictionary'},
]

function classNames(...classes) {
  return classes.filter(Boolean).join(' ')
}

export default function Header(props) {
  const [LoggedIn, setLoggedIn] = useContext(LoginContext);

  return (
    <>
    <Disclosure as="nav" className="bg-gray-800">
      {({ open }) => (
        <>
          <div className="mx-auto max-w-7xl px-2 sm:px-6 lg:px-8">
            <div className="relative flex h-14 items-center justify-between">
              <div className="absolute inset-y-0 left-0 flex items-center
sm:hidden">
                {/* Mobile menu button*/}
                <Disclosure.Button className="inline-flex items-center
justify-center rounded-md p-2 text-gray-400 hover:bg-gray-700
hover:text-white focus:outline-none focus:ring-2 focus:ring-inset
focus:ring-white">
```

```
                <span className="sr-only">Open main menu</span>
                {open ? (
                  <XMarkIcon className="block h-6 w-6" aria-hidden="true"
/>
                ) : (
                  <Bars3Icon className="block h-6 w-6" aria-hidden="true" />
                )}
              </Disclosure.Button>
            </div>
            <div className="flex flex-1 items-center justify-center
sm:items-stretch sm:justify-start">

              <div className="hidden sm:ml-6 sm:block">
                <div className="flex space-x-4">
                {  /* className={
                        classNames(
                        item.current
                        ? 'no-underline'
                        : 'no-underline'
                        ,

                        )}*/}
                  {navigation.map((item) => (
                    <NavLink
                      key={item.name}
                      to ={item.href}
                        className = {({isActive}) => {
                          return (
                            'px-3 py-2 rounded-md text-sm font-medium
no-underline' +
                            (!isActive
                              ? 'text-gray-300 hover:bg-gray-700 hover:text-white
no-underline'
                              : 'bg-gray-900 text-white no-underline')
```

```
                    );
                  }}

                >
                  {item.name}
                </NavLink>
              ))}
              { LoggedIn ?
                <NavLink

                  to ={'/login'}
                  onClick={() => {
                    console.log('logging out...');
                    setLoggedIn(false);
                    localStorage.clear()
                  }}
                  className= 'px-3 py-2 rounded-md text-sm font-medium
no-underline text-gray-300 hover:bg-gray-700 hover:text-white
no-underline'

                >
                  Logout

                </NavLink> :

                <NavLink

                  to ={'/login'}
                  className= 'px-3 py-2 rounded-md text-sm font-medium
no-underline text-gray-300 hover:bg-gray-700 hover:text-white
no-underline'

                >
                  Login
```

```
                  </NavLink> }


                </div>
              </div>
            </div>
            <div className="absolute inset-y-0 right-0 flex items-center pr-2
sm:static sm:inset-auto sm:ml-6 sm:pr-0">
              <button
                type="button"
                className="rounded-full bg-gray-800 p-1 text-gray-400
hover:text-white focus:outline-none focus:ring-2 focus:ring-white
focus:ring-offset-2 focus:ring-offset-gray-800"
              >
                <span className="sr-only">View notifications</span>
                <BellIcon className="h-6 w-6" aria-hidden="true" />
              </button>
</div>
</div>
</div>
        <Disclosure.Panel className="sm:hidden">
          <div className="space-y-1 px-2 pt-2 pb-3">
            {navigation.map((item) => (

<NavLink
key={item.name}
to ={item.href}
  className = {({isActive}) => {
    return (
      'block px-3 py-2 rounded-md text-base font-medium no-underline' +
      (!isActive
        ? 'text-gray-300 hover:bg-gray-700 hover:text-white no-underline'
        : 'bg-gray-900 text-white no-underline')
```

```
    );
 }} >
                {item.name}
                </NavLink>
            ))}

            { LoggedIn ?
            <NavLink

                to ={'/logout'}
                className= 'block px-3 py-2 rounded-md text-base
font-medium no-underline text-gray-300 hover:bg-gray-700
hover:text-white no-underline'
                >
                 Logout

                </NavLink>
                :
                <NavLink

                to ={'/login'}
                className= 'block px-3 py-2 rounded-md text-base
font-medium no-underline text-gray-300 hover:bg-gray-700
hover:text-white no-underline'
                >
                 Login

                </NavLink> }
          </div>
        </Disclosure.Panel>
      </>
    )}
  </Disclosure>
  <div className= "bg-gray-300">
```

```
    <div classNames="bg-purple-300 min-h-screen px-2 py-2">
      {props.children}
      </div>
      </div>
    </>
  );
}
```

Changes Made To The App.js:

```
import './index.css';
import { createContext, useState } from 'react';
import Header from './components/Header';
import Employees from './Pages/Employees';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Customers from './Pages/Customers';
import Dictionary from './Pages/Dictionary';
import Definition from './Pages/Definition';
import NotFound from './components/NotFound';
import Customer from './Pages/Customer';
import Login from './Pages/Login';

export const LoginContext = createContext();

function App() {
const [LoggedIn, setLoggedIn] = useState(localStorage.access ? true :
false);

function changeLoggedIn(value){
  setLoggedIn(value);
  if(value === false) {
    localStorage.clear();
  }
}
```

```
  return (
    <LoginContext.Provider value={[LoggedIn, changeLoggedIn]}>
    <BrowserRouter>
      <Header />
      <Routes>
        <Route path="/employees" element={<Employees />} />
        <Route path= "/dictionary" element={<Dictionary/>}/>
        <Route path= "/definition" element={<Definition/>}/>
        <Route path= "/404"element={<NotFound/>}/>
        <Route path= "*"element={<NotFound/>}/>
        <Route path= "/dictionary/:search"
        element={<Definition/>}
        />
        <Route path="/customers" element={<Customers />} />
        <Route path= "/login"element={<Login/>}/>
        <Route path="/customers/:id" element={<Customer />} />
      </Routes>
    </BrowserRouter>
    </LoginContext.Provider>
  );
}

export default App;
```

Auth Refresh Tokens

In this section, we'll explore how refresh tokens assist with logging in and out, especially since refresh and access tokens expire relatively quickly.

We'll also address this issue by making the necessary updates. **To do so, we'll use part of a video as a reference to update our backend, which can be found at the following link:**

https://django-rest-framework-simplejwt.readthedocs.io/en/latest/settings.html

First, we must add the following code to the settings.py file:

```
SIMPLE_JWT = {
    "ROTATE_REFRESH_TOKENS": True,
}
```

Then, we made the following changes to the settings.py file in the backend.

```
ALLOWED_HOSTS = []

from datetime import timedelta

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=16),
    "ROTATE_REFRESH_TOKENS": True,
}
```

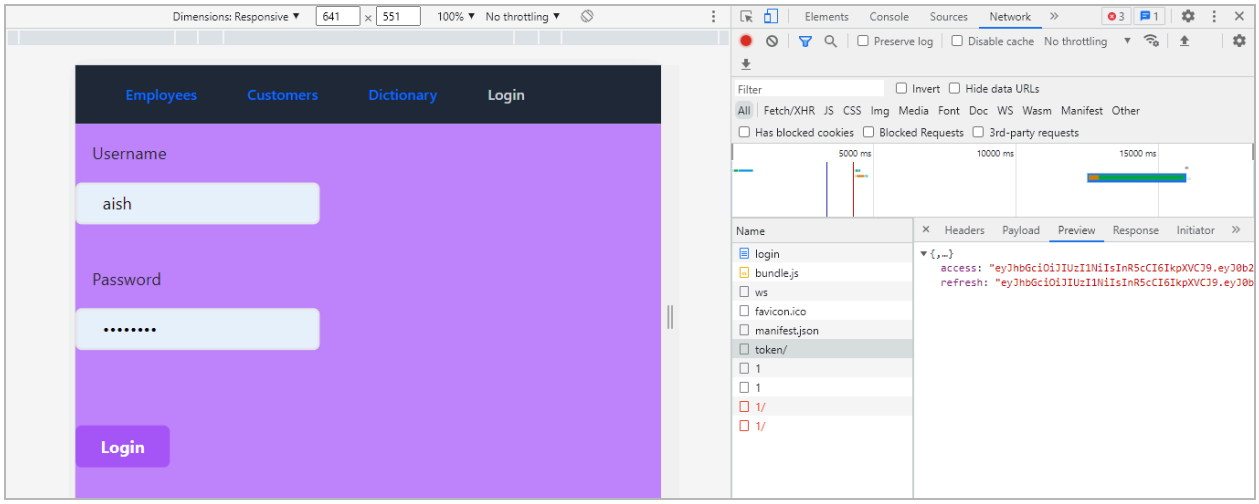
The next step is to enable access and refresh tokens on the front end. **Here is the code to achieve it:**

```
import './index.css';
import { createContext, useState, useEffect } from 'react';
import Header from './components/Header';
import Employees from './Pages/Employees';
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Customers from './Pages/Customers';
import Dictionary from './Pages/Dictionary';
import Definition from './Pages/Definition';
import NotFound from './components/NotFound';
import Customer from './Pages/Customer';
import Login from './Pages/Login';
import { baseUrl } from './shared';

export const LoginContext = createContext();

function App() {
```

```
  useEffect(() => {
    function refreshTokens(){
      if(localStorage.refresh){
        const url = baseUrl + 'api/token/refresh/';
        fetch(url , {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },

            body: JSON.stringify({
              refresh: localStorage.refresh,
            }),

          }).then((response) => {
          return response.json();
        }).then((data) => {
          localStorage.access = data.access;
          localStorage.refresh = data.refresh;
          setLoggedIn(true);
        });
      }
    }

    const minute = 1000 * 60;
    refreshTokens();
    setInterval(refreshTokens, minute * 3);
  }, []);
const [LoggedIn, setLoggedIn] = useState(localStorage.access ? true :
false);

function changeLoggedIn(value){
  setLoggedIn(value);
  if(value === false) {
```

```
    localStorage.clear();
  }
}

  return (
    <LoginContext.Provider value={[LoggedIn, changeLoggedIn]}>
    <BrowserRouter>
      <Header />
      <Routes>
        <Route path="/employees" element={<Employees />} />
        <Route path= "/dictionary"element={<Dictionary/>}/>
        <Route path= "/definition"element={<Definition/>}/>
        <Route path= "/404"element={<NotFound/>}/>
        <Route path= "*"element={<NotFound/>}/>
        <Route path= "/dictionary/:search"
        element={<Definition/>}
        />
        <Route path="/customers" element={<Customers />} />
        <Route path= "/login"element={<Login/>}/>
        <Route path="/customers/:id" element={<Customer />} />
      </Routes>
    </BrowserRouter>
    </LoginContext.Provider>
  );
}

export default App;
```

We see in the console log that it is working now:



The purpose of the above changes was to increase the time of the token so that the user doesn't have to log in and out again and again. Let's move to the next section.

User Register Form and API

In this section, we will learn how to implement a feature that allows users to register themselves on the website. **First, we included the following path in the urls.py file:**

```
path('api/register', views.register, name='register'),
```

Then, we made some changes to our code in the serializers.py file.

```
from rest_framework import serializers
from customers.models import Customer
from django.contrib.auth.models import User

class CustomerSerializer(serializers.ModelSerializer):
    class Meta:
        model = Customer
        fields = '__all__'
class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = '__all__'
    def create(self, validated_data):
        user = User.objects.create(
            username = validated_data['username'],
```

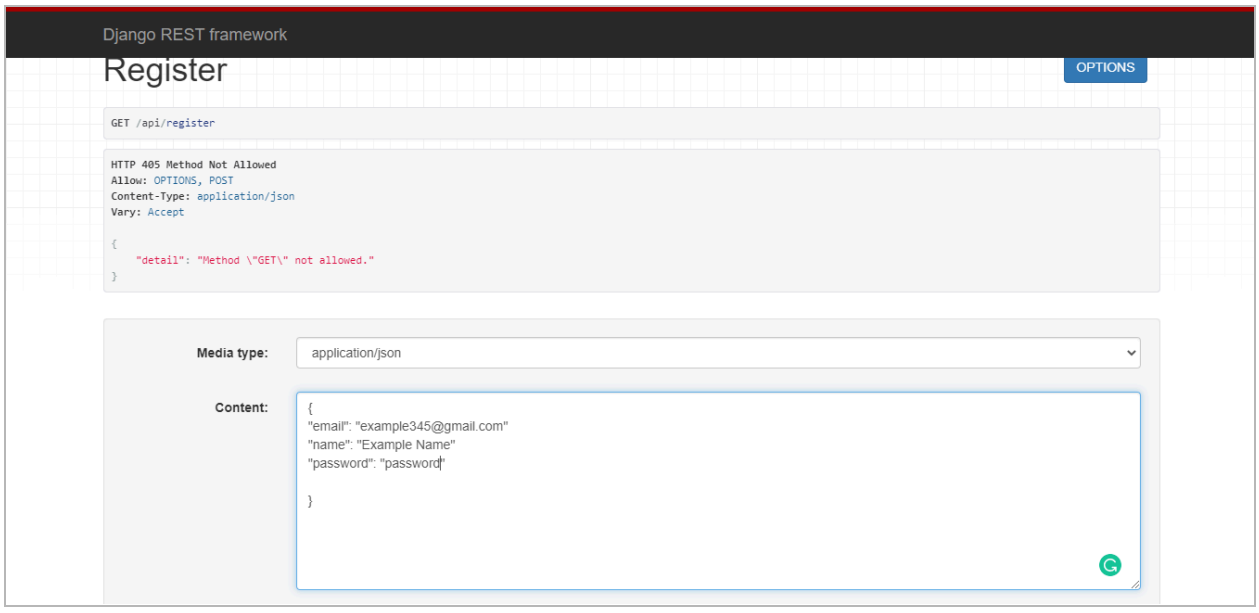
```
            email = validated_data['email']
        )

        user.set_password(validated_data['password'])
        user.save()
        return user
```

The next changes were made to the views.py file as follows:

```
from customers.models import Customer
from django.http import JsonResponse, Http404
from customers.serializers import CustomerSerializer, UserSerializer
from rest_framework.decorators import api_view, permission_classes
from rest_framework.response import Response
from rest_framework import status
from rest_framework.permissions import IsAuthenticated

@api_view(['GET', 'POST'])
@permission_classes([IsAuthenticated])
def customers(request):
   if request.method == 'GET':
      data = Customer.objects.all()
      serializer = CustomerSerializer(data, many=True)
      return Response({'customers': serializer.data})

   elif request.method == 'POST':
      serializer = CustomerSerializer(data=request.data)
   if serializer.is_valid():
      serializer.save()
      return Response({'customer': serializer.data},
status=status.HTTP_201_CREATED)
   return Response (serializer.errors,
status=status.HTTP_400_BAD_REQUEST)


@api_view(['GET', 'POST', 'DELETE'])
@permission_classes([IsAuthenticated])
```

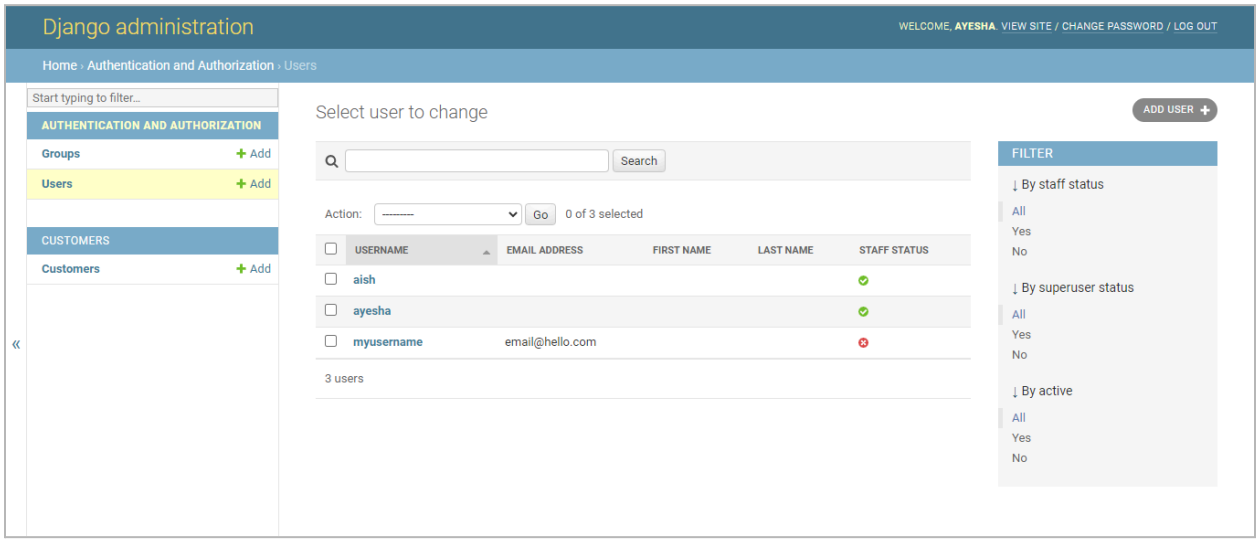
```
def customer(request, id):
   try:
       data = Customer.objects.get(pk=id)
   except Customer.DoesNotExist:
      return Response(status=status.HTTP_404_NOT_FOUND)

   if request.method == 'GET':
      serializer = CustomerSerializer(data)
      return Response({'customer': serializer.data})
   elif request.method == 'DELETE':
      data.delete()
      return Response(status=status.HTTP_204_NO_CONTENT)
   elif request.method == 'POST':
      serializer = CustomerSerializer(data, data=request.data)
      if serializer.is_valid():
         serializer.save()
      return Response({'customer': serializer.data})
   return Response (serializer.errors,
status=status.HTTP_400_BAD_REQUEST)


@api_view(['POST'])
def register(request):
      serializer = UserSerializer()
      serializer = UserSerializer(data=request.data)
      if serializer.is_valid():
         serializer.save()
         return Response(status=status.HTTP_201_CREATED)
```

The next step is to visit http://127.0.0.1:8000/api/register, **and add some info in the text editor as under:**



After clicking on the POST, you will see a response HTTP 201 Created.

The next step is to visit http://127.0.0.1:8000/admin/, where you'll find the user listed below.



We successfully registered a user at the backend. But now we need to make some changes at the front end. First, we duplicated the Login.js file on the front end and inserted the following code. Take note that we renamed the duplicate file Register.js.

```
import { useState, useEffect, useContext } from 'react';
import { baseUrl } from '../shared';
import { useLocation, useNavigate } from 'react-router-dom';
```

```
import { LoginContext } from '../App';

export default function Register() {
  const [LoggedIn, setLoggedIn] = useContext(LoginContext);
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [email, setEmail] = useState('');

  const location = useLocation();
  const navigate = useNavigate();

  useEffect(() => {
    localStorage.clear();
    setLoggedIn(false);
  }, [])

  function register(e) {
    e.preventDefault();
    const url = baseUrl + 'api/register/';
    fetch(url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        email: email,
        username: username,
        password: password,
      }),
    })
      .then((response) => {
        return response.json();
      })
      .then((data) => {
        localStorage.setItem('access', data.access);
        localStorage.setItem('refresh', data.refresh);
        setLoggedIn(true);
        navigate(location?.state?.previousUrl ? location.state.previousUrl :
'/customers');
```

```
    });
  }

  return (
    <form
      className="bg-purple-400 min-h-screen"
      id="customer"
      onSubmit={register}
    >

<div className="md:flex md:items-center mb-6">
        <div className="p-3 md:w-1/4">
          <label htmlFor="email">Email</label>
        </div>
        <div className="md:w-3/4">
          <input
            id="email"
            className="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-30 py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
            type="email"
            value={email}
            onChange={(e) => {
              setEmail(e.target.value);
            }}
          />
        </div>
      </div>

      <div className="md:flex md:items-center mb-6">
        <div className="p-3 md:w-1/4">
          <label htmlFor="username">Username</label>
        </div>
        <div className="md:w-3/4">
          <input
            id="username"
            className="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-30 py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
```

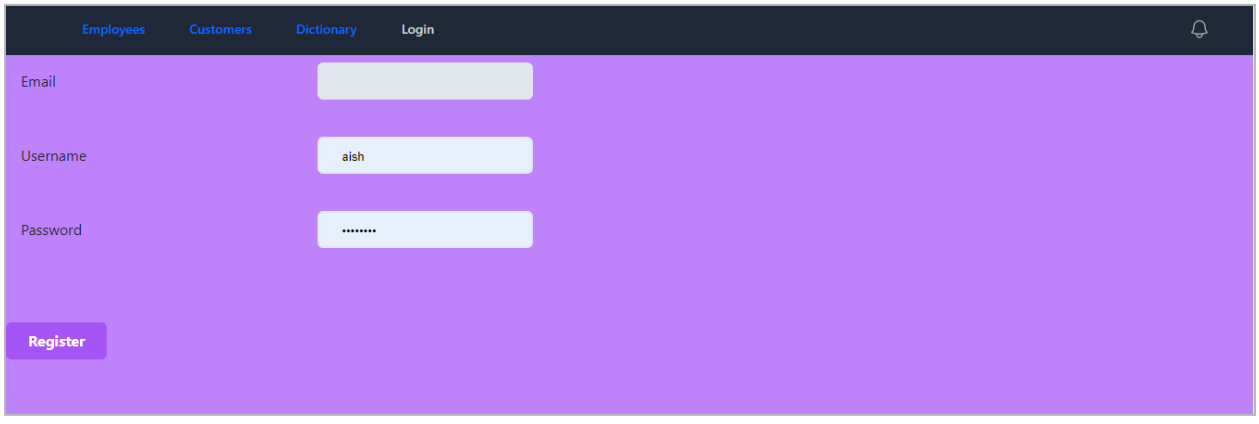
```
            type="text"
            value={username}
            onChange={(e) => {
              setUsername(e.target.value);
            }}
          />
        </div>
      </div>

      <div className="md:flex md:items-center mb-6">
        <div className="p-3 md:w-1/4">
          <label htmlFor="password">Password</label>
        </div>
        <div className="md:w-3/4">
          <input
            id="password"
            className="bg-gray-200 appearance-none border-2
border-gray-200 rounded w-30 py-2 px-4 text-gray-700 leading-tight
focus:outline-none focus:bg-white focus:border-purple-500"
            type="password"
            value={password}
            onChange={(e) => {
              setPassword(e.target.value);
            }}
          />
        </div>
      </div>

      <button className="mt-5 bg-purple-500 hover:bg-purple-700
text-white font-bold py-2 px-4 rounded">
        Register
      </button>
    </form>
  );
}
```

The next step is to add the following changes to the views.py file.

```
from customers.models import Customer
from django.http import JsonResponse, Http404
from customers.serializers import CustomerSerializer, UserSerializer
from rest_framework.decorators import api_view, permission_classes
from rest_framework.response import Response
from rest_framework import status
from rest_framework.permissions import IsAuthenticated
from rest_framework_simplejwt.tokens import RefreshToken

@api_view(['GET', 'POST'])
@permission_classes([IsAuthenticated])
def customers(request):
    if request.method == 'GET':
        data = Customer.objects.all()
        serializer = CustomerSerializer(data, many=True)
        return Response({'customers': serializer.data})
    elif request.method == 'POST':
        serializer = CustomerSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({'customer': serializer.data},
status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)


@api_view(['GET', 'POST', 'DELETE'])
@permission_classes([IsAuthenticated])
def customer(request, id):
    try:
        data = Customer.objects.get(pk=id)
    except Customer.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)

    if request.method == 'GET':
        serializer = CustomerSerializer(data)
        return Response({'customer': serializer.data})
    elif request.method == 'DELETE':
```

```
        data.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
    elif request.method == 'POST':
        serializer = CustomerSerializer(data, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({'customer': serializer.data})
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)


@api_view(['POST'])
def register(request):
    serializer = UserSerializer(data=request.data)
    if serializer.is_valid():
        user = serializer.save()
        refresh = RefreshToken.for_user(user)
        tokens = {
            'refresh': str(refresh),
            'access': str(refresh.access_token)
        }
        return Response(tokens, status=status.HTTP_201_CREATED)
    return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
```

Then, we also added a route for the Register page inside the App.js and imported it here. Now, if we visit http://localhost:3000/register, **it shows us the following page:**



We successfully created a register page. **Let's move to the next section now.**

Create a Custom Hook (useFetch)

Hooks are a powerful way to introduce functionality into our React app. The useFetch hook, for example, is a custom hook that simplifies fetching data from an API endpoint by managing the loading, error, and success states for you.

This hook takes a URL as a parameter and returns an object with three properties: data, loading, and error.

The useEffect hook handles the API request, updating the state based on the response. The fetch function performs the request and parses the response as JSON.

If successful, the data state is updated with the parsed JSON, and the loading state is false. In case of an error, the error state is updated with the error object, and loading is also set to false.

In short, this custom hook helps consolidate all the fetching logic in one place. Let's start integrating this hook into our code.

First, we created a new hooks folder inside the src directory. Then, we added a new file named UseFetch.js. **Now, let's add the following code to the useFetch file:**

```
import { useState, useEffect } from 'react';
export default function useFetch(url) {

    const[data, setData] = useState(null);
    useEffect(() => {
        fetch(url).then((response)=>{
            return response.json();
        }).then((data)=>{
            setData(data);
        })
    }, []);
    return data;
}
```

The next step is to add the relevant changes to the Definition.js file.

```
import { useState, useEffect } from 'react';
import { useParams, useNavigate, Link, useLocation, useFetcher } from
'react-router-dom';
import DefinitionSearch from '../components/DefinitionSearch';
```

```
import NotFoundComponent from '../components/NotFound';
import useFetch from '../hooks/UseFetch';

export default function Definition() {
  /*const [meanings, setMeanings] = useState([]);*/
  const [notFound, setNotFound] = useState(false);
  const [error, setError] = useState(false);


  const { search } = useParams();
  const navigate = useNavigate();
  const location = useLocation();
  const meanings = useFetch(
    'https://api.dictionaryapi.dev/api/v2/entries/en/' + search
    );

  useEffect(() => {
    console.log(meanings);
  });

  if (notFound === true) {
    return (
      <div className="bg-purple-300 min-h-screen px-3 py-3">
      <>
        <NotFoundComponent/>
        <Link to='/dictionary'>Search Another Word</Link>
      </>
      </div>
    );
  }


  if (error === true) {
    return (
      <div className="bg-purple-300 min-h-screen px-3 py-3">
      <>
        <p>Something went wrong, try again? </p>
        <Link to='/dictionary'>Search Another Word</Link>
      </>
```

```
    </div>
  );
 }

 return <p>Work In Progress</p>;
 return (
  <div className="bg-purple-300 min-h-screen px-3 py-3">
   {meanings.length > 0 && (
    <>
     <h1>Definition:</h1>
     {meanings.map((meaning, index) => (
      <p key={index}>
       {meaning.partOfSpeech}: {meaning.definitions[0].definition}
      </p>
     ))}
     <p>Search Again:</p>
     <DefinitionSearch/>

    </>
   )}
  </div>
 );
}
```

When we open the definition page, the following output indicates everything works fine.

We are getting a good response right now, but we want to show our dictionary.
Therefore, we rearranged our code and made a few changes to it as follows:

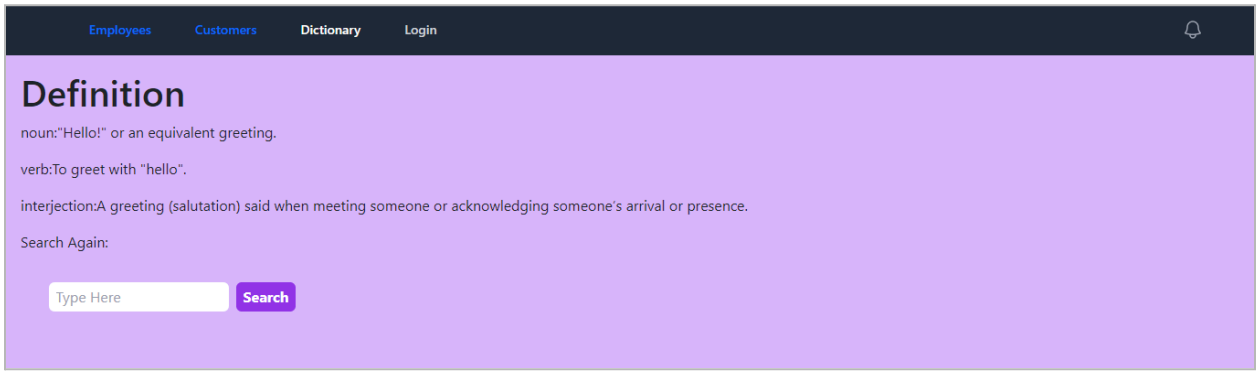
```
import { useState, useEffect } from 'react';
import { useParams, useNavigate, Link, useLocation } from
'react-router-dom';
import DefinitionSearch from '../components/DefinitionSearch';
import NotFoundComponent from '../components/NotFound';
```

```
import useFetch from '../hooks/UseFetch';
import { v4 as uuidv4 } from 'uuid';

export default function Definition() {
  /*const [meanings, setMeanings] = useState([]);*/
  const [notFound, setNotFound] = useState(false);
  const [error, setError] = useState(false);


  const { search } = useParams();
  const navigate = useNavigate();
  const location = useLocation();
  const word = useFetch(
    'https://api.dictionaryapi.dev/api/v2/entries/en/' + search
    );

  useEffect(() => {
    console.log(word);
  });

  /* useEffect(() => {
    //const url = 'http://httpstat.us/500';
    const url =  `https://api.dictionaryapi.dev/api/v2/entries/en/${search}`;
    fetch(url)
      .then((response) => {
        if (response.status === 404) {
          setNotFound(true);
        }
        else if (response.status === 401){

        }
          else if (response.status === 500){
            setError(true)
          }

          if (!response.ok){
            setError(true);
            throw new Error('Something went wrong');
          }
```

```
        return response.json();
      })
      .then((data) => {
        if (data && data.length > 0) {
          setMeanings(data[0].meanings);
        } else {
          setNotFound(true);
        }
      })
      .catch((error) => {
        console.error(error);
      });
  }, [search, navigate]); */

  if (notFound === true) {
    return (
      <div className="bg-purple-300 min-h-screen px-3 py-3">
      <>
        <NotFoundComponent/>
        <Link to='/dictionary'>Search Another Word</Link>
      </>
      </div>
    );
  }


  if (error === true) {
    return (
      <div className="bg-purple-300 min-h-screen px-3 py-3">
      <>
        <p>Something went wrong, try again? </p>
        <Link to='/dictionary'>Search Another Word</Link>
      </>
      </div>
    );
  }


  return (
```

```
    <div className="bg-purple-300 min-h-screen px-3 py-3">

    <>
        {word?.[0].meanings ? (
          <>
            <h1>Definition</h1>
            {word[0].meanings.map((meaning) => {
              return (
                <p key = {uuidv4()}>
                {meaning.partOfSpeech + ':'}
                {meaning.definitions[0].definition}
                </p>
                );
                })}
                <p>Search Again:</p>
            <DefinitionSearch/>
        </>
      ) : null }
    </>
    </div>
  );
}
```

We can see that our definition page is working perfectly fine now.



The next step is to build the error capability inside our custom hook. **First, we made a few changes to the useFetch.js file under:**

```
import { useState, useEffect } from 'react';
export default function useFetch(url) {
```

```
    const[data, setData] = useState();
    const[errorStatus, setErrorStatus] = useState();
    useEffect(() => {
        fetch('').then((response)=>{
            if(!response.ok){
                throw response.status();
            }
            return response.json();
        }).then((data)=>{
            setData(data);
        })
        .catch((e)=>{
            setErrorStatus(e);
        });
    }, []);
    return data;
}
```

Then, we made changes to the Definition.js page, as you can see below:

```
import { useState, useEffect } from 'react';
import { useParams, useNavigate, Link, useLocation, useFetcher } from
'react-router-dom';
import DefinitionSearch from '../components/DefinitionSearch';
import NotFoundComponent from '../components/NotFound';
import useFetch from '../hooks/UseFetch';
import { v4 as uuidv4 } from 'uuid';

export default function Definition() {
  /*const [meanings, setMeanings] = useState([]);*/
  //const [notFound, setNotFound] = useState(false);
  //const [error, setError] = useState(false);


  const { search } = useParams();
```

```
const navigate = useNavigate();
const location = useLocation();
const [word, errorStatus] = useFetch(
  'https://api.dictionaryapi.dev/api/v2/entries/en/' + search
  );

if (errorStatus === 404) {
  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
    <>
      <NotFoundComponent/>
      <Link to='/dictionary'>Search Another Word</Link>
    </>
    </div>
  );
}


if (errorStatus) {
  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
    <>
      <p>Something went wrong, try again? </p>
      <Link to='/dictionary'>Search Another Word</Link>
    </>
    </div>
  );
}


return (
  <div className="bg-purple-300 min-h-screen px-3 py-3">

  <>
        {word?.[0].meanings ? (
          <>
            <h1>Definition</h1>
            {word[0].meanings.map((meaning) => {
              return (
```

```
                    <p key = {uuidv4()}>
                    {meaning.partOfSpeech + ':'}
                    {meaning.definitions[0].definition}
                    </p>
                    );
                    })}
                    <p>Search Again:</p>
                <DefinitionSearch/>
            </>
          ) : null }
        </>
        </div>
      );
}
```

Destructuring Explained (Custom Hook Parameters and Return Data)

In this section, we will learn how custom hook parameters return data. Let's make some changes to our files.

Changes Made to the UseFetch.js:

```
import { useState, useEffect } from 'react';
import { useNavigate, useLocation } from 'react-router-dom';
export default function useFetch(url, {method, headers, body}) {

    const[data, setData] = useState();
    const[errorStatus, setErrorStatus] = useState();

    const navigate = useNavigate();
    const location = useLocation();
    useEffect(() => {
        fetch(url, {
            method,
            headers,
            body,
        }).then((response)=>{
```

```
            if(response.status === 401){
                navigate('/login', {
                    state: {
                      previousUrl: location.pathname,
                    },
                  });
            }
            if(!response.ok){
                throw response.status();
            }
            return response.json();
        }).then((data)=>{
            setData(data);
        })
        .catch((e)=>{
            setErrorStatus(e);
        });
    }, []);
    return { data, setData, errorStatus};
}
```

Changes Made to the Definition.js:

```
import { useState, useEffect } from 'react';
import { useParams, useNavigate, Link, useLocation, useFetcher } from
'react-router-dom';
import DefinitionSearch from '../components/DefinitionSearch';
import NotFoundComponent from '../components/NotFound';
import useFetch from '../hooks/UseFetch';
import { v4 as uuidv4 } from 'uuid';

export default function Definition() {
  /*const [meanings, setMeanings] = useState([]);*/
  //const [notFound, setNotFound] = useState(false);
  //const [error, setError] = useState(false);


  const { search } = useParams();
```

```
const navigate = useNavigate();
const location = useLocation();
const [word, errorStatus] = useFetch(
  'https://api.dictionaryapi.dev/api/v2/entries/en/' + search
  );

if (errorStatus === 404) {
  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
    <>
      <NotFoundComponent/>
      <Link to='/dictionary'>Search Another Word</Link>
    </>
    </div>
  );
}


if (errorStatus) {
  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">
    <>
      <p>Something went wrong, try again? </p>
      <Link to='/dictionary'>Search Another Word</Link>
    </>
    </div>
  );
}


return (
  <div className="bg-purple-300 min-h-screen px-3 py-3">

  <>
        {word?.[0].meanings ? (
          <>
            <h1>Definition</h1>
            {word[0].meanings.map((meaning) => {
              return (
```

```
                    <p key = {uuidv4()}>
                    {meaning.partOfSpeech + ':'}
                    {meaning.definitions[0].definition}
                    </p>
                    );
                    })}
                    <p>Search Again:</p>
                <DefinitionSearch/>
            </>
          ) : null }
        </>
        </div>
      );
}
```

Changes Made to theCustomers.js:

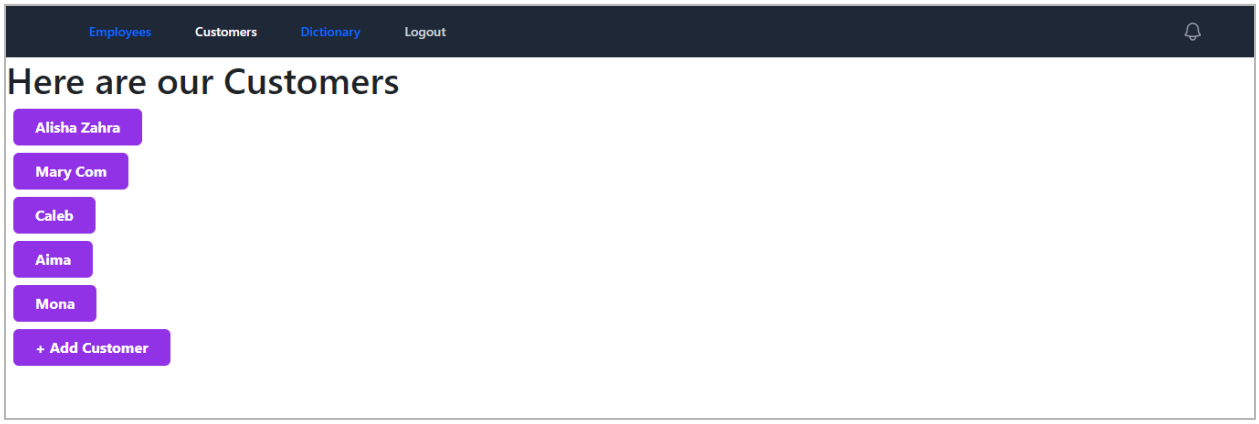
```
import { useEffect, useState, useContext } from 'react';
import { Link, useNavigate, useLocation } from 'react-router-dom';
import AddCustomer from '../components/AddCustomer';
import { baseUrl } from '../shared';
import { LoginContext } from '../App';
import useFetch from '../hooks/UseFetch';

export default function Customers() {
  const [LoggedIn, setLoggedIn] = useContext(LoginContext);
  //const [customers, setCustomers] = useState([]);
  const [show, setShow] = useState(false);

  const location = useLocation();
  const navigate = useNavigate();

  const url = baseUrl + 'api/customers/';
  const { data: {customers} = {}, errorStatus} = useFetch(url, { method:
'GET',
  headers: {
  'Content-Type': 'application/json',
  Authorization: 'Bearer ' + localStorage.getItem('access'),
```

```
},
} );

useEffect(()=>{
  console.log(customers);
});

  function toggleShow() {
    setShow(!show);
  }

  /*

  useEffect(() => {
    const url = baseUrl + 'api/customers/';
    fetch(url, {
      headers: {
        'Content-Type': 'application/json',
        Authorization: 'Bearer ' + localStorage.getItem('access'),
      },
    })
      .then((response) => {
        if (response.status === 401) {
          setLoggedIn(false);
          navigate('/login', {
            state: {
              previousUrl: location.pathname,
            },
          });
        }
        return response.json();
      })
      .then((data) => {
        setCustomers(data.customers);
      })
      .catch((error) => console.log(error));
  }, [navigate, location.pathname]); */

  function newCustomer(name, industry) {
```

```
  /*const data = { name: name, industry: industry };
  const url = baseUrl + 'api/customers/';
  fetch(url, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      Authorization: 'Bearer ' + localStorage.getItem('access'),
    },
    body: JSON.stringify(data),
  })
    .then((response) => {
      if (!response.ok) {
        throw new Error('Something went wrong');
      }
      return response.json();
    })
    .then((data) => {
      toggleShow();
      setCustomers([...customers, data.customer]);
    })
    .catch((error) => console.log(error)); */
}



 return (
   <>
    <h1>Here are our Customers</h1>
    {customers ?
    customers.map((customer) => {
      return(
      <div className="m-2" key={customer.id}>
        <Link to={`/customers/${customer.id}`}>
        <button className="no-underline bg-purple-600
hover:bg-purple-700 text-white font-bold py-2 px-4 rounded">
                                {customer.name}
                            </button>
                        </Link>
                    </div>
```

```
                    );
                })
                : null}

            <AddCustomer
                newCustomer={newCustomer}
                show={show}
                toggleShow={toggleShow}
            />
        </>
    );
}
```

It helps us get the customer's data on our screen.



However, the Add customer functionality still doesn't work here. Let's talk about it in the next few sections.

Default Values and Nested Data with Restructuring

In this section, we will perform the data restructuring. You can find out how our Definition.js file has been restructured.

```
import { useState, useEffect } from 'react';
import { useParams, useNavigate, Link, useLocation, useFetcher } from
'react-router-dom';
import DefinitionSearch from '../components/DefinitionSearch';
import NotFoundComponent from '../components/NotFound';
```

```
import useFetch from '../hooks/UseFetch';
import { v4 as uuidv4 } from 'uuid';

export default function Definition() {
  /*const [meanings, setMeanings] = useState([]);*/
  //const [notFound, setNotFound] = useState(false);
  //const [error, setError] = useState(false);


  const { search } = useParams();
  const navigate = useNavigate();
  const location = useLocation();
  const {data: [{meanings : word}] = [{}],
    errorStatus} = useFetch(
    'https://api.dictionaryapi.dev/api/v2/entries/en/' + search,
    );

  if (errorStatus === 404) {
    return (
      <div className="bg-purple-300 min-h-screen px-3 py-3">
      <>
        <NotFoundComponent/>
        <Link to='/dictionary'>Search Another Word</Link>
      </>
      </div>
    );
  }


  if (errorStatus) {
    return (
      <div className="bg-purple-300 min-h-screen px-3 py-3">
      <>
        <p>Something went wrong, try again? </p>
        <Link to='/dictionary'>Search Another Word</Link>
      </>
      </div>
    );
  }
```

```
  return (
    <div className="bg-purple-300 min-h-screen px-3 py-3">

    <>
        {word ? (
          <>
            <h1>Definition</h1>
            {word.map((meaning) => {
                return (
                    <p key = {uuidv4()}>
                    {meaning.partOfSpeech + ':'}
                    {meaning.definitions[0].definition}
                    </p>
                    );
                    })}
                    <p>Search Again:</p>
              <DefinitionSearch/>
        </>
      ) : null }
    </>
    </div>
  );
}
```

The next structuring was performed in the UseFetch.js file.

```
import { useState, useEffect } from 'react';
import { useNavigate, useLocation } from 'react-router-dom';
export default function useFetch(url, {method, headers, body} = {}) {

    const[data, setData] = useState();
    const[errorStatus, setErrorStatus] = useState();

    const navigate = useNavigate();
    const location = useLocation();
    useEffect(() => {
```

```
        fetch(url, {
            method: method,
            headers: headers,
            body: body,
        }).then((response)=>{
            if(response.status === 401){
                navigate('/login', {
                    state: {
                      previousUrl: location.pathname,
                    },
                  });
            }
            if(!response.ok){
                throw response.status();
            }
            return response.json();
        }).then((data)=>{
            setData(data);
        })
        .catch((e)=>{
            setErrorStatus(e);
        });
    }, []);
    return { data, setData, errorStatus};
}
```

You can see that the website is still working fine after the code has been structured.

Custom Hook on Button Click (onClick POST with useFetch)

This section will enable the Add Customer functionality on our customers' page. First,

we changed the UseFetch.js file by creating the **appendData** function.

```
import { useState, useEffect } from 'react';
import { useNavigate, useLocation } from 'react-router-dom';
export default function useFetch(url, {method, headers, body} = {}) {

   const[data, setData] = useState();
   const[errorStatus, setErrorStatus] = useState();

   const navigate = useNavigate();
   const location = useLocation();

   function request() {
   fetch(url, {
         method: method,
         headers: headers,
         body: body,
      }).then((response)=>{
         if(response.status === 401){
            navigate('/login', {
               state: {
                 previousUrl: location.pathname,
               },
             });
         }
         if(!response.ok){
            throw response.status();
         }
         return response.json();
      }).then((data)=>{
         setData(data);
      })
      .catch((e)=>{
         setErrorStatus(e);
      });
```

```
}

function appendData(newData) {
    fetch(url, {
        method: 'POST',
        headers: headers,
        body: JSON.stringify(newData),
    }).then((response)=>{
        if(response.status === 401){
            navigate('/login', {
                state: {
                  previousUrl: location.pathname,
                },
              });
            }
              if(!response.ok){
                throw response.status;
              }
              return response.json()
            }).then((d) => {
                const submitted = d?.[0];
                console.log('in the then...', submitted);
                console.log(data);
                const newState = {...data};
                Object.values(newState)[0].push(submitted);
                setData(newState);

            }).catch((e) => {
                console.log(e);
                setErrorStatus(e);
            })

}
```

```
    return {request, appendData, data, errorStatus};
}
```

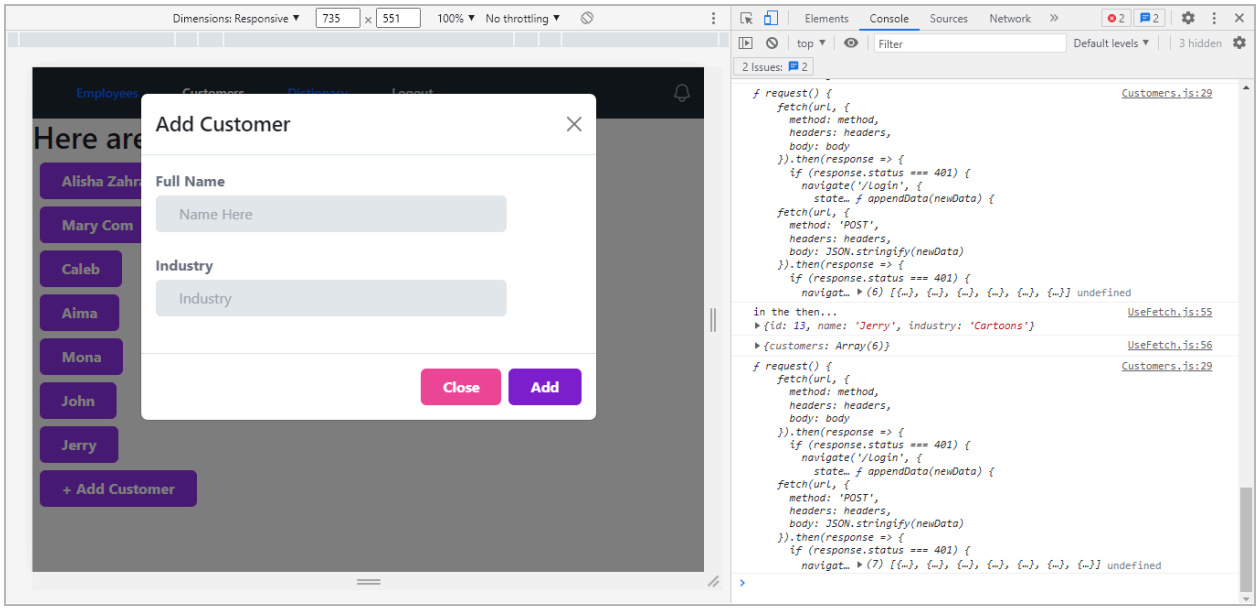
We also changed the Customers.js file so that our new function works fine.

```
import { useEffect, useState, useContext } from 'react';
import { Link, useNavigate, useLocation } from 'react-router-dom';
import AddCustomer from '../components/AddCustomer';
import { baseUrl } from '../shared';
import { LoginContext } from '../App';
import useFetch from '../hooks/UseFetch';

export default function Customers() {
  const [LoggedIn, setLoggedIn] = useContext(LoginContext);
  //const [customers, setCustomers] = useState([]);
  const [show, setShow] = useState(false);

  const location = useLocation();
  const navigate = useNavigate();

  const url = baseUrl + 'api/customers/';
  const {request, appendData, data: {customers} = {}, errorStatus} =
useFetch(url, { method: 'GET',
  headers: {
  'Content-Type': 'application/json',
  Authorization: 'Bearer ' + localStorage.getItem('access'),
},
} );

useEffect(() => {
  request();
}, [])

useEffect(()=>{
  console.log(request, appendData, customers, errorStatus);
```

```
}, []);

  function toggleShow() {
    setShow(!show);
  }

  /*

  useEffect(() => {
    const url = baseUrl + 'api/customers/';
    fetch(url, {
      headers: {
        'Content-Type': 'application/json',
        Authorization: 'Bearer ' + localStorage.getItem('access'),
      },
    })
      .then((response) => {
        if (response.status === 401) {
          setLoggedIn(false);
          navigate('/login', {
            state: {
              previousUrl: location.pathname,
            },
          });
        }
        return response.json();
      })
      .then((data) => {
        setCustomers(data.customers);
      })
      .catch((error) => console.log(error));
  }, [navigate, location.pathname]); */

  function newCustomer(name, industry) {
```

```
    appendData({name: name, industry: industry});
  }



  return (
    <>
      <h1>Here are our Customers</h1>
      {customers ?
      customers.map((customer) => {
        return(
        <div className="m-2" key={customer.id}>
          <Link to={`/customers/${customer.id}`}>
          <button className="no-underline bg-purple-600
hover:bg-purple-700 text-white font-bold py-2 px-4 rounded">
                                {customer.name}
                              </button>
                            </Link>
                          </div>
                      );
                    })
                  : null}

            <AddCustomer
              newCustomer={newCustomer}
              show={show}
              toggleShow={toggleShow}
            />
        </>
    );
}
```

Let's check our site. It should be working fine now.



The next step is to add the following code to the Customers.js file. It will automatically close the popup window when you click the Add button.

```
if(!errorStatus) {
    toggleShow();
  }
```

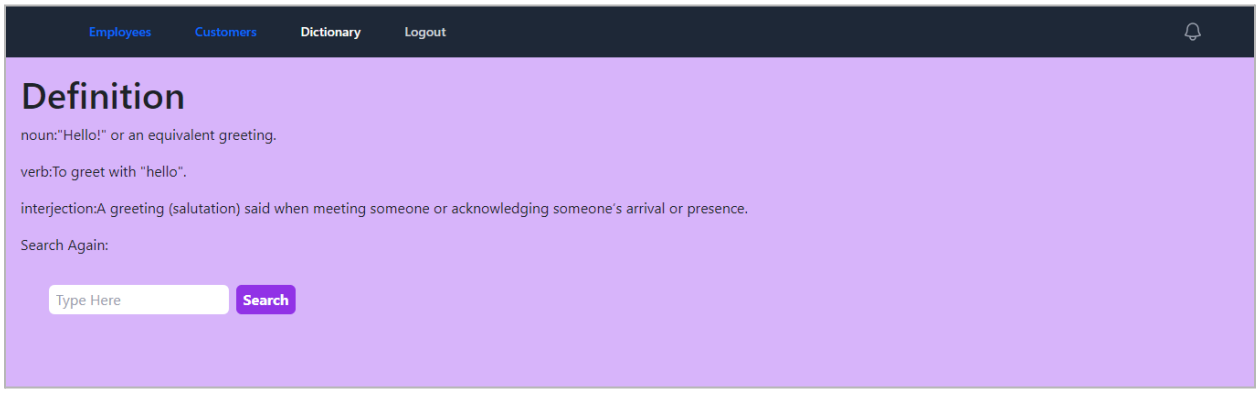
After all the changes, the definition search has stopped working. It shows us a blank page when we click on the search button.

For this purpose, we added the following changes to our code inside the Definition.js file.

```
const {request, data: [{meanings : word}] = [{}],
  errorStatus} = useFetch(
  'https://api.dictionaryapi.dev/api/v2/entries/en/' + search,
  );
```

```
useEffect(() => {
  request();
})
```

We see that our web page is working fine now.



TypeScript and Axios Intro

You can learn about adding typescript here:

https://create-react-app.dev/docs/adding-typescript/

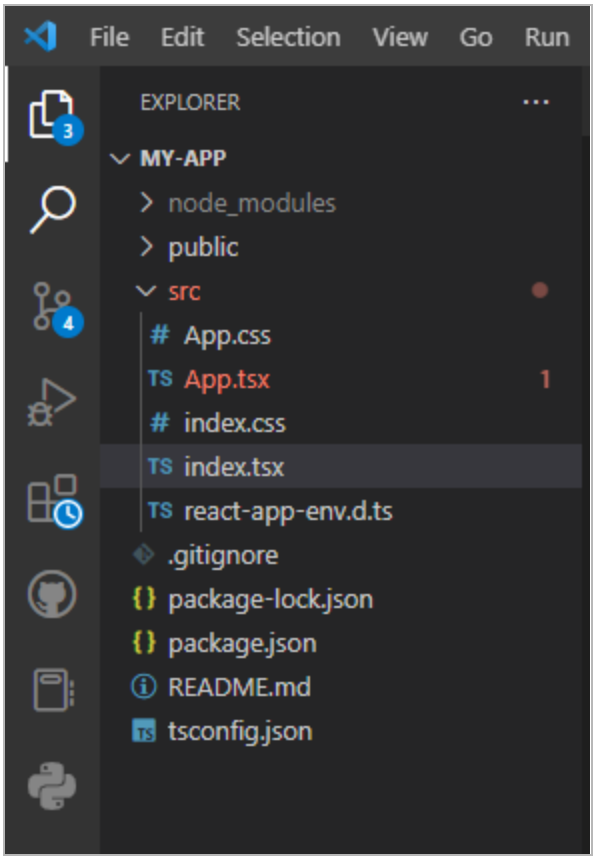
First, open the Command Prompt and run the following command there:

```
npx create-react-app my-app --template typescript
```

Then, run the following code to open the new app inside the Visual Studio Code.

```
code my-app
```

You must clean some files once it is opened in the Visual Studio Code. **Your file structure should look like the below:**



The code inside the App.tsx should look like below:

```
import React from 'react';
import './App.css';

function App() {
  return (
    <div className="App">
     Hello
    </div>
  );
}

export default App;
```

Note that you must clear the App.css and index.css files. **Now, we can see our web page as follows:**

The next step is implementing the Axios. You can learn about it from this link:
https://axios-http.com/

You can install it by using the following command:

```
npm install axios
```

The next step is to import it inside the App.tsx file as under:

```
import axios from 'axios';
```

Note that we are making this app to get information relevant to currencies. **We will take some data from the following page:**

https://www.coingecko.com/

We will use the APIs from the above page. **For this purpose, we can visit the URL:**

https://www.coingecko.com/en/api/documentation

You must copy the URL from the documentation page and add it to the App.tsx file.

'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=market_cap_desc&per_page=100&page=1&sparkline=false'

The code inside the App.tsx should look like below:

```
import React from 'react';
import './App.css';
import axios from 'axios';

function App() {
  const url =
```

```
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&
order=market_cap_desc&per_page=100&page=1&sparkline=false';
  );
  return (
    <div className="App">
     Hello
    </div>
}

export default App;
```

After running the above code, we get the following output inside the console.

The next step is to show this data on our web page. We made some changes to our code inside the App.tsx file.

```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';

export type Crypto = {

  ath: number;

  atl: number;

  current_price: number;

  id: string;

  name: string;

  symbol: string;

  high_24h: number;

  low_24h: number;

};

function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >();
    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
        setCryptos(response.data);
      })
    }, []);
        return <div className="App">
     {cryptos ? cryptos.map((crypto) => {
```

```
      return <p>{crypto.name + ' ' + crypto.current_price}</p>
    }) : null}
   </div>;
}

export default App;
```

You can see that the results are now displayed on our web page.

The next step is to make more changes in the App.tsx file.

```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';

export type Crypto = {
  ath
  :
  number;
  atl
  :
  number;
```

```
  current_price
  :
  number;
  id
  :
  string;
  name
  :
  string;
  symbol
  :
  string;
  high_24h :
  number;
  low_24h :
  number;
}

function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
        setCryptos(response.data);
      })
    }, []);
        return <div className="App">
     {cryptos ? cryptos.map((crypto) => {
      return <p>{crypto.name + ' ' + crypto.current_price}</p>
     }) : null }
    </div>;
}
```

```
export default App;
```

Save the above code and run the following commands in the terminal:

```
git add .
```

```
git commit -m "initial api request with axios and typescript"
```

The next step is navigating to your GitHub and creating a new repository with ts-Axios. Here, you have to save all the code from the Visual Studio Code.

TypeScript Components

In this section, we will learn to create components using TypeScript. **You can also learn about it from the link given below:**

https://react-typescript-cheatsheet.netlify.app/docs/basic/getting-started/function_components

First, we'll create a components folder inside the src directory. Then, we'll build our first component, CryptoSummary.tsx.

Next, we'll remove the following line of code from App.tsx and move it to CryptoSummary.tsx to keep our structure organized.

```
return <p>{crypto.name + ' ' +
crypto.current_price}</p>
```

You must add another return statement to the App.tsx because the previous one was removed. The following statement must replace the previous statement. It will fix the issues in App.tsx.

```
return <CryptoSummary crypto = {crypto}/>;
```

Now, the code inside the CryptoSummary.tsx should look like this:

```
import { Crypto } from "../App";
```

```
export type AppProps = {
    crypto : Crypto
}

export default function CryptoSummary({crypto} : AppProps):
JSX.Element {

    return <p>{crypto.name + ' ' + crypto.current_price}</p>

}
```

The next step is to create a new folder within the source. You must name the new folder types. Create a new file, Cryptos.tsx, within the new folder.

Once you've completed the preceding steps, remove the following code from the App.tsx. Then add it to Crytos.tsx.

```
export type Crypto = {
  ath
  :
  number;
  atl
  :
  number;
  current_price
  :
  number;
  id
  :
  string;
  name
  :
  string;
  symbol
  :
  string;
  high_24h :
  number;
```

```
  low_24h :
  number;
}
```

The next step is to import Crypto.tsx inside the App.tsx and CryptoSummary.tsx using the following code.

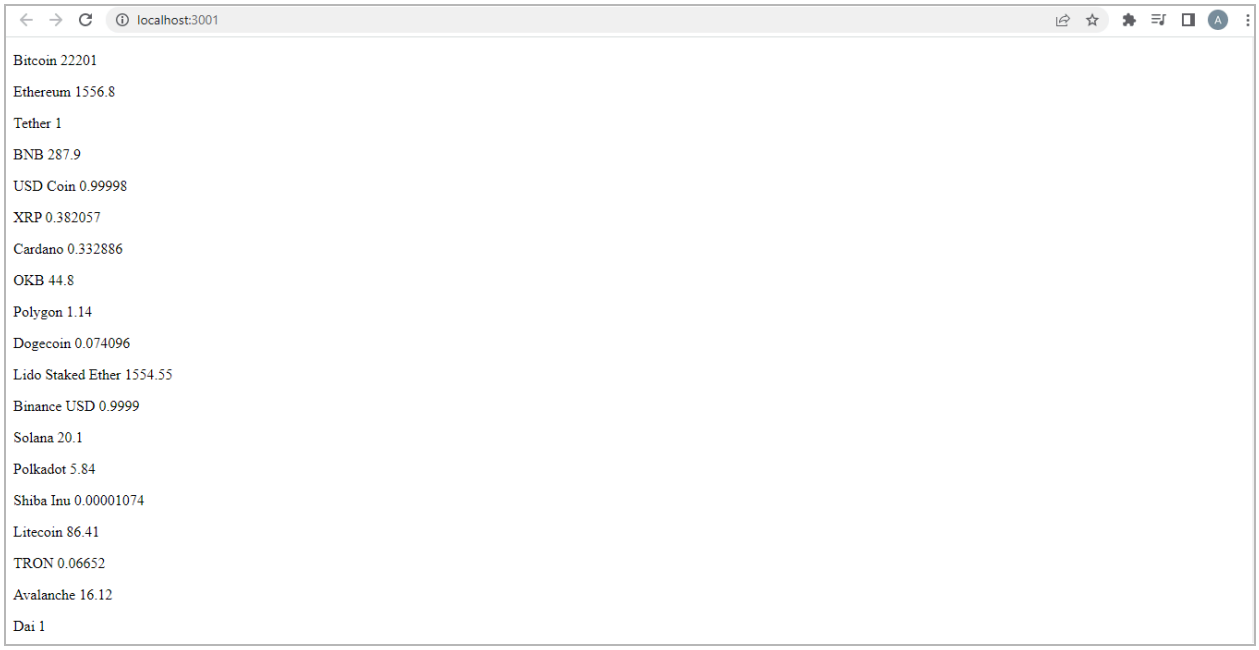
```
import { Crypto } from './types/Types';
```

Since we need to make a few more adjustments, we'll rename Crypto.tsx to Types.tsx. Similarly, we'll update all relevant imports to reflect this new name. Next, we'll move Types.tsx to the src directory and remove the previously created types folder.

The final code inside Types.tsx should look like the following:

```
import { Crypto } from '../Types';

export type AppProps = {
    crypto : Crypto;
};

export default function CryptoSummary({crypto} : AppProps):
JSX.Element {

    return <p>{crypto.name + ' ' + crypto.current_price}</p>;

}
```

Now, our web page should look perfectly fine. **But we have to create a drop-down for our list instead of a view like in the image below:**



Generate Drop-Down List from the API

We will use an API to create a drop-down list that will dynamically show the data within it. **To accomplish this, we will edit the return statement within App.tsx, as shown below:**

```
return <div className="App">
        <select>
    {cryptos ? cryptos.map((crypto) => {
     return <option>{crypto.name}</option>;
    }) : null }
    </select>
   </div>;
```

A dropdown like this will appear on our web page:

However, we need to make a few more changes to our code. It will allow us to display certain values when we click on any currency in the dropdown list.

Here's the modified App.tsx code:

```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';
import CryptoSummary from './Components/CryptoSummary';
import { Crypto } from './Types';


function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
  const [selected, setSelected] = useState<Crypto | null >();
    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
        setCryptos(response.data);
      })
    }, []);
        return (
        <>
        <div className="App">
          <select onChange={(e)=>{
            const c = cryptos?.find((x) => x.id === e.target.value);
            setSelected(c);
          } }>
     {cryptos ? cryptos.map((crypto) => {
      return <option key={crypto.id}
value={crypto.id}>{crypto.name}</option>;
     }) : null }
     </select>
    </div>
    {selected ? <CryptoSummary crypto={selected} /> : null}
    </>
    );

}

export default App;
```

Next, we must create a default option that displays “Choose an option” instead of a currency. **To implement this, we made the following changes to App.tsx:**

```
return (
      <>
      <div className="App">
        <select onChange={(e)=>{
          const c = cryptos?.find((x) => x.id === e.target.value);
          setSelected(c);
        } }

        defaultValue = "default"
        >

  <option value = 'default'>Choose an Option </option>


  {cryptos ? cryptos.map((crypto) => {
   return <option key={crypto.id} value={crypto.id}>{crypto.name}</option>;
  }) : null }

  </select>
</div>
{selected ? <CryptoSummary crypto={selected} /> : null}
</>
);
```

You can see that it is appearing on our web page now:

In the next section, we will implement chart.js to show the values as charts on our web page.

Crypto Price Chart with Chart.js

You can learn about chart.js from the below link:

https://www.chartjs.org/

Moreover, you can also learn about the implementation of React with chart.js from the below link:

https://react-chartjs-2.js.org/

Let's begin our implementation. **First, run the following command in the terminal:**

```
npm install chart.js react-chartjs-2
```

Note that we will implement a line chart inside our app. **You can check it out at the following link:**

https://react-chartjs-2.js.org/examples/line-chart

Get the following part from the above link and add it to your code:

```
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';
import { Line } from 'react-chartjs-2';

ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
```

```
  Legend
);
```

The next step is to add the following two code lines into the App.tsx.

```
const [data, setData] = useState();
const [options, setOptions] = useState();
```

You must know that chart.js also comes with TypeScript. **You can learn about it from the link given below:**

https://react-chartjs-2.js.org/faq/typescript/

You have to add the following import into your code:

```
import type { ChartData, ChartOptions } from 'chart.js';
```

We will also make some more changes to our code. It must look like this:

```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';
import CryptoSummary from './Components/CryptoSummary';
import { Crypto } from './Types';
import type { ChartData, ChartOptions } from 'chart.js';

import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';
import { Line } from 'react-chartjs-2';
```

```
ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend
);


function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
  const [selected, setSelected] = useState<Crypto | null >();
  const [data, setData] = useState<ChartData<'line'>>();
  const [options, setOptions] = useState<ChartOptions<'line'>>({
    responsive: true,
  plugins: {
    legend: {
      position: 'top' as const,
    },
    title: {
      display: true,
      text: 'Chart.js Line Chart',
    },
  },
  });
    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
        setCryptos(response.data);
      })
    }, []);
        return (
        <>
        <div className="App">
```

```
          <select onChange={(e)=>{
            const c = cryptos?.find((x) => x.id === e.target.value);
            setSelected(c);
            axios.get('url').then((response) => { });
            console.log('Getting Crypto Prices...');
          } }

          defaultValue = "default"
          >
      {cryptos ? cryptos.map((crypto) => {
       return <option key={crypto.id}
value={crypto.id}>{crypto.name}</option>;
      }) : null }
      <option value = 'default'>Choose an Option </option>
      </select>
     </div>
     {selected ? <CryptoSummary crypto={selected} /> : null}
    {data ? <Line options={options} data={data}/> : null }
     </>
     );

}

export default App;
```

The next step is to have an API request from the below link:

https://www.coingecko.com/en/api/documentation

We got the following API URL from the above link.

https://api.coingecko.com/api/v3/coins/bitcoin/market_chart?vs_currency=usd&days=30&interval=daily

We implemented the above URL inside our code and made a few more adjustments to make it work fine.

Here is our final code:

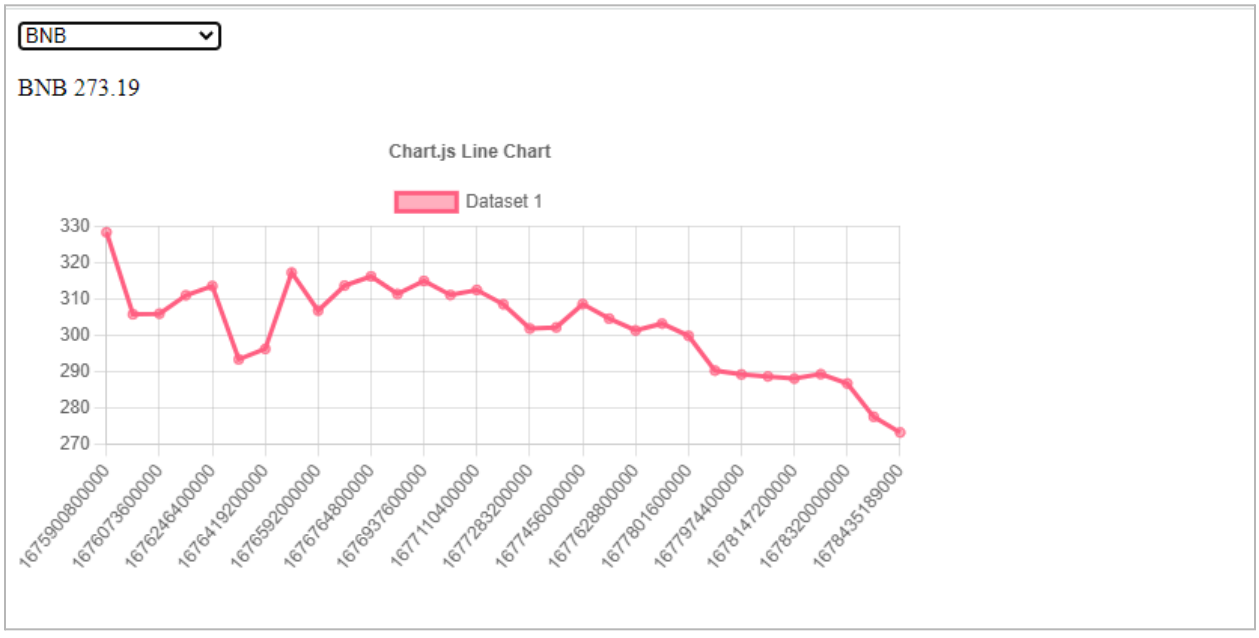
```
import './App.css';
import { useEffect, useState } from 'react';
```

```
import axios from 'axios';
import CryptoSummary from './Components/CryptoSummary';
import { Crypto } from './Types';
import type { ChartData, ChartOptions } from 'chart.js';

import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';
import { Line } from 'react-chartjs-2';

ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend
);


function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
  const [selected, setSelected] = useState<Crypto | null >();
  const [data, setData] = useState<ChartData<'line'>>();
  const [options, setOptions] = useState<ChartOptions<'line'>>({
    responsive: true,
  plugins: {
    legend: {
      position: 'top' as const,
    },
    title: {
```

```
      display: true,
      text: 'Chart.js Line Chart',
    },
  },
  });
    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
        setCryptos(response.data);
      })
    }, []);
        return (
        <>
        <div className="App">
          <select onChange={(e)=>{
            const c = cryptos?.find((x) => x.id === e.target.value);
            setSelected(c);
            axios.get(

`https://api.coingecko.com/api/v3/coins/${c?.id}/market_chart?vs_currenc
y=usd&days=30&interval=daily`
              ).then((response) => {
                console.log(response.data);
                setData(
                  {
                    labels: response.data.prices.map((price: number[]) => {return
price[0]}),
                    datasets: [
                      {
                        label: 'Dataset 1',
                        data: response.data.prices.map((price: number[]) =>
{return price[1]}),
                        borderColor: 'rgb(255, 99, 132)',
                        backgroundColor: 'rgba(255, 99, 132, 0.5)',
                      },
                    ],
                  });
```

```
            });
          } }

          defaultValue = "default"
          >
     {cryptos ? cryptos.map((crypto) => {
      return <option key={crypto.id}
value={crypto.id}>{crypto.name}</option>;
     }) : null }
     <option value = 'default'>Choose an Option </option>
     </select>
    </div>
    {selected ? <CryptoSummary crypto={selected} /> : null}
   {data ? <div style= {{width:600}}>
    <Line options={options} data={data}/></div> : null }
    </>
    );

}

export default App;
```

Here is how our web page looks now:



So far, things are looking considerably better. However, we want to make a few additional improvements to improve the appearance of the timestamps. We'll be using the moment node package for this.

npm install moment

After you install it, you have to import it using the following command:

```
import moment from 'moment';
```

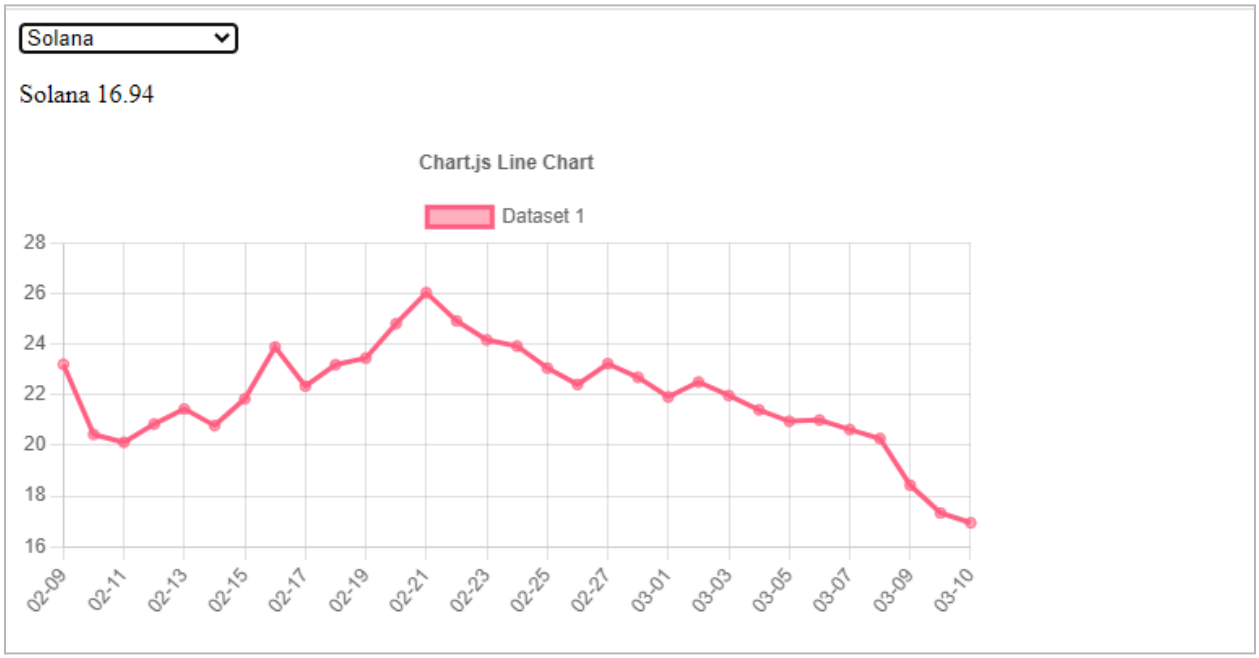
We will format the timestamps that appear inside our chart. **So, here is the code:**

```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';
import CryptoSummary from './Components/CryptoSummary';
import { Crypto } from './Types';
import type { ChartData, ChartOptions } from 'chart.js';
import moment from 'moment';

import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';
import { Line } from 'react-chartjs-2';

ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
```

```
  Legend
);


function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
  const [selected, setSelected] = useState<Crypto | null >();
  const [data, setData] = useState<ChartData<'line'>>();
  const [options, setOptions] = useState<ChartOptions<'line'>>({
    responsive: true,
  plugins: {
    legend: {
      position: 'top' as const,
    },
    title: {
      display: true,
      text: 'Chart.js Line Chart',
    },
  },
  });
    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
        setCryptos(response.data);
      })
    }, []);
        return (
        <>
        <div className="App">
          <select onChange={(e)=>{
            const c = cryptos?.find((x) => x.id === e.target.value);
            setSelected(c);
            axios.get(

`https://api.coingecko.com/api/v3/coins/${c?.id}/market_chart?vs_currenc
y=usd&days=30&interval=daily`
              ).then((response) => {
```

```
            console.log(response.data);
            setData(
              {
                labels: response.data.prices.map((price: number[]) => {

                  return moment.unix(price[0] / 1000).format('MM-DD');
                }
                ),
                datasets: [
                  {
                    label: 'Dataset 1',
                    data: response.data.prices.map((price: number[]) =>
{return price[1]}),
                    borderColor: 'rgb(255, 99, 132)',
                    backgroundColor: 'rgba(255, 99, 132, 0.5)',
                  },
                ],
              });
          });
        } }

        defaultValue = "default"
        >
    {cryptos ? cryptos.map((crypto) => {
     return <option key={crypto.id}
value={crypto.id}>{crypto.name}</option>;
    }) : null }
    <option value = 'default'>Choose an Option </option>
    </select>
   </div>
   {selected ? <CryptoSummary crypto={selected} /> : null}
  {data ? <div style= {{width:600}}>
   <Line options={options} data={data}/></div> : null }
   </>
   );

}

export default App;
```

Now, we can see a better view of timestamps.



Dynamic Chart with Multiple Drop-Downs (Chart.js)

In this section, we'll create a drop-down menu to select the time range for displaying currency data. We've added options for 30 days, 7 days, and 1 day. Additionally, for the 1-day range, we included hourly timestamps to provide more detailed data.

Here's the code:

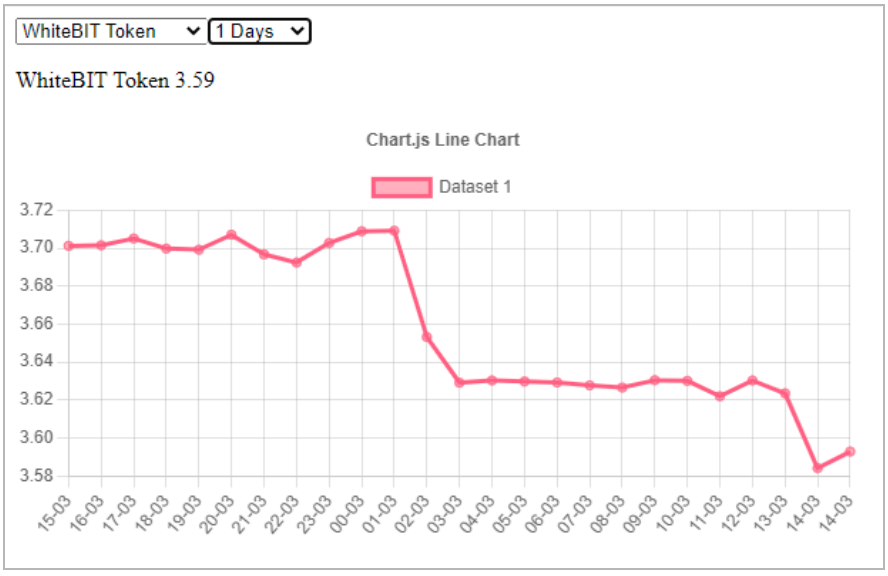
```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';
import CryptoSummary from './Components/CryptoSummary';
import { Crypto } from './Types';
import type { ChartData, ChartOptions } from 'chart.js';
import moment from 'moment';

import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
```

```
  Title,
  Tooltip,
  Legend,
} from 'chart.js';
import { Line } from 'react-chartjs-2';

ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend
);


function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
  const [selected, setSelected] = useState<Crypto | null >();
  const [range, setRange] = useState<number>(30);
  const [data, setData] = useState<ChartData<'line'>>();
  const [options, setOptions] = useState<ChartOptions<'line'>>({
    responsive: true,
  plugins: {
    legend: {
      position: 'top' as const,
    },
    title: {
      display: true,
      text: 'Chart.js Line Chart',
    },
  },
  });
    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
```

```
        setCryptos(response.data);
      })
    }, []);

    useEffect(() => {
      axios.get(

`https://api.coingecko.com/api/v3/coins/${selected?.id}/market_chart?vs_
currency=usd&days=${range}&${
          range === 1 ? 'interval=hourly' : "interval=daily"}`
        ).then((response) => {
          console.log(response.data);
          setData(
            {
              labels: response.data.prices.map((price: number[]) => {

                return moment.unix(price[0] / 1000).format(range === 1 ?
'HH-MM' : 'MM-DD');
              }
              ),
              datasets: [
                {
                  label: 'Dataset 1',
                  data: response.data.prices.map((price: number[]) => {return
price[1]}),
                  borderColor: 'rgb(255, 99, 132)',
                  backgroundColor: 'rgba(255, 99, 132, 0.5)',
                },
              ],
            });
        });

    }, [selected, range]);


        return (
        <>
        <div className="App">
          <select onChange={(e)=>{
```

```
          const c = cryptos?.find((x) => x.id === e.target.value);
          setSelected(c);
        } }

        defaultValue = "default"
        >
    {cryptos ? cryptos.map((crypto) => {
     return <option key={crypto.id}
value={crypto.id}>{crypto.name}</option>;
    }) : null }
    <option value = 'default'>Choose an Option </option>
    </select>
    <select onChange={(e) => {

     setRange(parseInt(e.target.value));

    }}>
    <option value={29}>30 Days</option>
    <option value={6}>7 Days</option>
    <option value={1}>1 Days</option>
    </select>
   </div>
   {selected ? <CryptoSummary crypto={selected} /> : null}
  {data ? <div style= {{width:600}}>
   <Line options={options} data={data}/></div> : null }
   </>
   );

}

export default App;
```

Here is the output of the above code:



We will do some more formatting to display something like:

"Bitcoin Price Over Last 30 Days" at the top of the chart. **Here is the code to implement these changes:**

```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';
import CryptoSummary from './Components/CryptoSummary';
import { Crypto } from './Types';
import type { ChartData, ChartOptions } from 'chart.js';
import moment from 'moment';

import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';
import { Line } from 'react-chartjs-2';
```

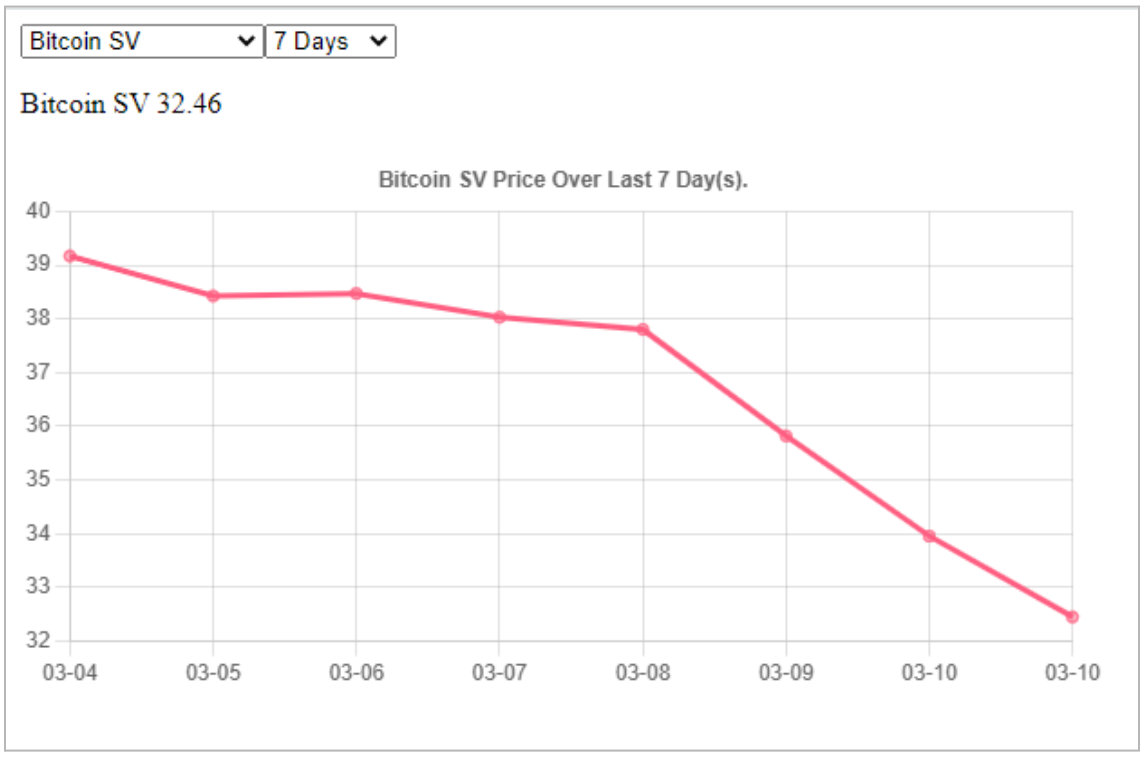
```
ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend
);


function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
  const [selected, setSelected] = useState<Crypto | null >();
  const [range, setRange] = useState<number>(30);
  const [data, setData] = useState<ChartData<'line'>>();
  const [options, setOptions] = useState<ChartOptions<'line'>>({
    responsive: true,
  plugins: {
    legend: {
      position: 'top' as const,
    },
    title: {
      display: true,
      text: 'Chart.js Line Chart',
    },
  },
  });
    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
        setCryptos(response.data);
      })
    }, []);

    useEffect(() => {
      if(!selected) return;
```

```
    axios.get(

`https://api.coingecko.com/api/v3/coins/${selected?.id}/market_chart?vs_
currency=usd&days=${range}&${
        range === 1 ? 'interval=hourly' : "interval=daily"}`
      ).then((response) => {
        console.log(response.data);
        setData(
          {
            labels: response.data.prices.map((price: number[]) => {

              return moment.unix(price[0] / 1000).format(range === 1 ?
'HH-MM' : 'MM-DD');
            }
            ),
            datasets: [
              {
                label: 'Dataset 1',
                data: response.data.prices.map((price: number[]) => {return
price[1]}),
                borderColor: 'rgb(255, 99, 132)',
                backgroundColor: 'rgba(255, 99, 132, 0.5)',
              },
            ],
          });

          setOptions({
            responsive: true,
            plugins: {
              legend: {
                display: false,
              },
              title: {
                display: true,
                text: `${selected?.name} Price Over Last ` + range + (range
=== 1
                ? ` Day.`
                : ` Day(s).`),
              },
```

```
            },
          })
        });

    }, [selected, range]);


        return (
        <>
        <div className="App">
          <select onChange={(e)=>{
            const c = cryptos?.find((x) => x.id === e.target.value);
            setSelected(c);
          } }

          defaultValue = "default"
          >
     {cryptos ? cryptos.map((crypto) => {
      return <option key={crypto.id}
value={crypto.id}>{crypto.name}</option>;
     }) : null }
     <option value = 'default'>Choose an Option </option>
     </select>
     <select onChange={(e) => {

      setRange(parseInt(e.target.value));

     }}>
     <option value={30}>30 Days</option>
     <option value={7}>7 Days</option>
     <option value={1}>1 Days</option>
     </select>
    </div>
    {selected ? <CryptoSummary crypto={selected} /> : null}
   {data ? <div style= {{width:600}}>
    <Line options={options} data={data}/></div> : null }
    </>
    );
```

```
}

export default App;
```

Here is the output of the above changes:



Calculate Crypto Values

In this step, we'll set up our web page to calculate cryptocurrency values dynamically. In addition, we may use a pie chart to illustrate the data. First, we'll make changes to list the currencies individually as they are selected from the drop-down menu.

Here's the code and its output:

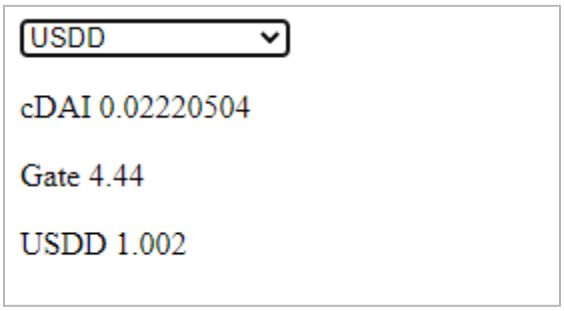
```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';
import CryptoSummary from './Components/CryptoSummary';
import { Crypto } from './Types';
import type { ChartData, ChartOptions } from 'chart.js';
import moment from 'moment';
```

```
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';
import { Line } from 'react-chartjs-2';

ChartJS.register(
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend
);


function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
  const [selected, setSelected] = useState<Crypto[] >([]);
  const [range, setRange] = useState<number>(30);
  /*
  const [data, setData] = useState<ChartData<'line'>>();
  const [options, setOptions] = useState<ChartOptions<'line'>>({
    responsive: true,
  plugins: {
    legend: {
      position: 'top' as const,
    },
    title: {
      display: true,
      text: 'Chart.js Line Chart',
```

```
  },
 },
 });
 */
   useEffect(() => {
     const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
     axios.get(url).then((response) => {
       setCryptos(response.data);
     })
   }, []);

 /*  useEffect(() => {
     if(!selected) return;
     axios.get(

`https://api.coingecko.com/api/v3/coins/${selected?.id}/market_chart?vs_
currency=usd&days=${range}&${
         range === 1 ? 'interval=hourly' : "interval=daily"}`
       ).then((response) => {
         console.log(response.data);
         setData(
           {
             labels: response.data.prices.map((price: number[]) => {

               return moment.unix(price[0] / 1000).format(range === 1 ?
'HH-MM' : 'MM-DD');
             }
             ),
             datasets: [
               {
                 label: 'Dataset 1',
                 data: response.data.prices.map((price: number[]) => {return
price[1]}),
                 borderColor: 'rgb(255, 99, 132)',
                 backgroundColor: 'rgba(255, 99, 132, 0.5)',
               },
             ],
```

```
        });

        setOptions({
          responsive: true,
          plugins: {
            legend: {
              display: false,
            },
            title: {
              display: true,
              text: `${selected?.name} Price Over Last ` + range + (range
=== 1
              ? ` Day.`
              : ` Day(s).`),
            },
          },
        })
      });

   }, [selected, range]); */


      return (
      <>
      <div className="App">
        <select onChange={(e)=>{
          const c = cryptos?.find((x) => x.id === e.target.value) as Crypto;
          setSelected([...selected, c]);
        } }

        defaultValue = "default"
        >
    {cryptos ? cryptos.map((crypto) => {
     return <option key={crypto.id}
value={crypto.id}>{crypto.name}</option>;
    }) : null }
    <option value = 'default'>Choose an Option </option>
    </select>
```

```
    </div>

     {selected.map((s) => {return <CryptoSummary crypto = {s}/>})}


    {/*selected ? <CryptoSummary crypto={selected} /> : null*/}
   {/*data ? <div style= {{width:600}}>
    <Line options={options} data={data}/></div> : null */ }
    </>
    );

}

export default App;
```

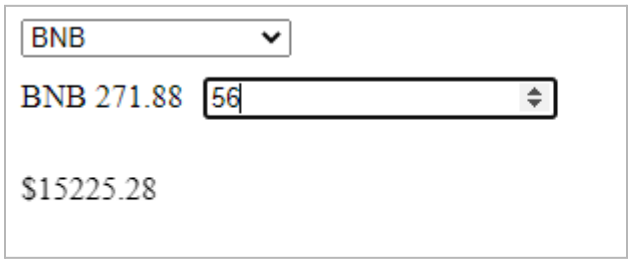


Next, we made changes to the CryptoSummary.tsx as follows:

```
import { useState, useEffect } from 'react';
import { Crypto } from '../Types';

export type AppProps = {
    crypto : Crypto;
};

export default function CryptoSummary({crypto} : AppProps):
JSX.Element {

    useEffect(()=>{
        console.log(crypto.name, amount, crypto.current_price *
```

```
parseFloat(amount));
    });

    const[amount, setAmount] = useState<string>('0');

    return (
    <div>

    <span>{crypto.name + ' ' + crypto.current_price}</span>
    <input
    type= "number"
    style={{margin: 10}} value={amount} onChange={(e) => {
        setAmount(e.target.value)
    }}></input>

    <p>${(crypto.current_price *
parseFloat(amount)).toLocaleString(undefined, {minimumFractionDigits:
2,
        maximumFractionDigits: 2}
        )}</p>

    </div>
    );

}
```

It returns the following output, indicating that we can calculate the values of several currencies in dollars.



Aggregate Data with a map and reduce

First, we will remove the following code from the App.tsx file:

```
  const [range, setRange] =
useState<number>(30);
```

The next step is to change the CryptoSummary.tsx file code. **You must modify the string to a number, as shown below:**

```
const[amount, setAmount] = useState<string>('0');
```

to

```
 const[amount, setAmount] =
useState<number>(0);
```

Next, we'll remove parseFloat from our code and make additional adjustments. **The final code inside CryptoSummary.tsx should look like this:**

```
import { useState, useEffect } from 'react';
import { Crypto } from '../Types';

export type AppProps = {
    crypto : Crypto;
    updateOwned: (crypto: Crypto, amount: number) => void;
};

export default function CryptoSummary({crypto, updateOwned}:
AppProps): JSX.Element {

    useEffect(()=>{
        console.log(crypto.name, amount, crypto.current_price * amount);
    });

    const[amount, setAmount] = useState<number>(0);

    return (
    <div>
```

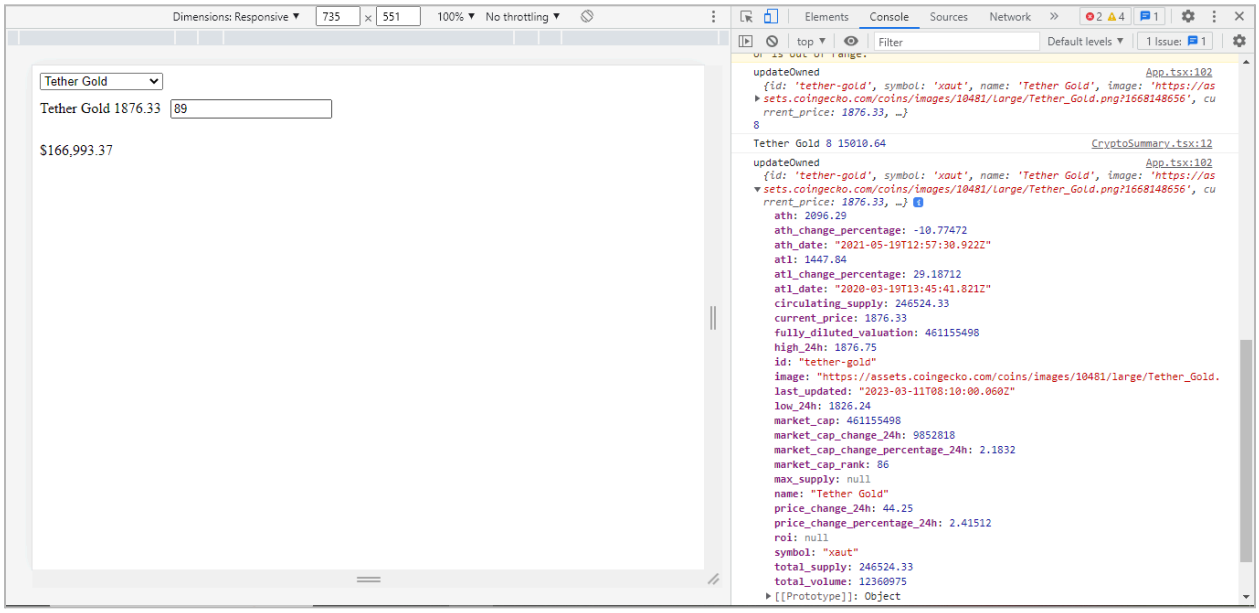
```
    <span>{crypto.name + ' ' + crypto.current_price}</span>
    <input
    type= "number"
    style={{margin: 10}} value={amount} onChange={(e) => {
        setAmount(parseFloat(e.target.value));
        updateOwned(crypto, parseFloat(e.target.value))
    }}></input>

    <p>${(crypto.current_price * amount).toLocaleString(undefined,
{minimumFractionDigits: 2,
        maximumFractionDigits: 2}
        )}</p>

    </div>
    );

}
```

The next step is implementing our function updateOwned on the App.tsx.

```
function updateOwned(crypto: Crypto, amount: number): void{

     console.log('updateOwned', crypto, amount);

   }
       return (
       <>
       <div className="App">
         <select onChange={(e)=>{
           const c = cryptos?.find((x) => x.id === e.target.value) as Crypto;
           setSelected([...selected, c]);
         } }

         defaultValue = "default"
         >
    {cryptos ? cryptos.map((crypto) => {
     return <option key={crypto.id}
```

```
value={crypto.id}>{crypto.name}</option>;
     }) : null }
     <option value = 'default'>Choose an Option </option>
     </select>

    </div>

     {selected.map((s) => {return <CryptoSummary crypto = {s}
updateOwned={updateOwned}/>;
    })}


    {/*selected ? <CryptoSummary crypto={selected} /> : null*/}
   {/*data ? <div style= {{width:600}}>
    <Line options={options} data={data}/></div> : null */ }
    </>
    );

}

export default App;
```

Here is how we get output inside the developer console for the above changes:



The next step was to show the selected values more appropriately. For this purpose, we

changed the function inside the App.tsx file. **This was accomplished by incorporating a temporary object beneath:**

```
  useEffect(() => {
    console.log('SELECTED:', selected);
  }, [selected])


  function updateOwned(crypto: Crypto, amount: number): void{
    console.log('updateOwned', crypto, amount);
    let temp = [...selected];
    let tempObj = temp.find(()=> crypto.id === crypto.id)
    if(tempObj) {
     tempObj.owned = amount;
     setSelected(temp);
    }
  }
      return (
      <>
      <div className="App">
        <select onChange={(e)=>{
          const c = cryptos?.find((x) => x.id === e.target.value) as Crypto;
          setSelected([...selected, c]);
        } }

        defaultValue = "default"
        >
    {cryptos ? cryptos.map((crypto) => {
     return <option key={crypto.id}
value={crypto.id}>{crypto.name}</option>;
    }) : null }
    <option value = 'default'>Choose an Option </option>
    </select>

  </div>

    {selected.map((s) => {return <CryptoSummary crypto = {s}
updateOwned={updateOwned}/>;
  })}
```

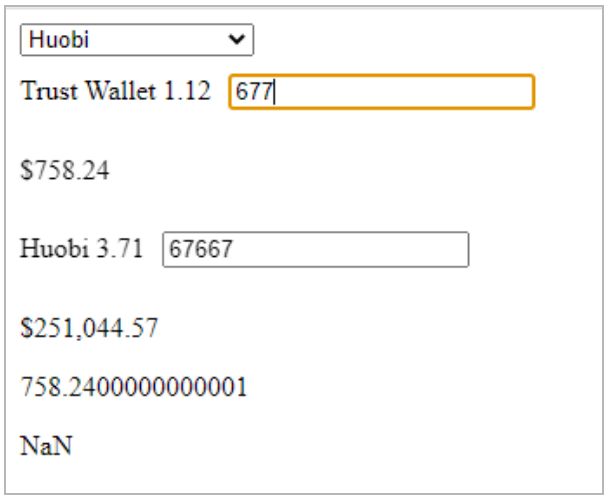
```
   {/*selected ? <CryptoSummary crypto={selected} /> : null*/}
  {/*data ? <div style= {{width:600}}>
   <Line options={options} data={data}/></div> : null */ }
   </>
   );

}

export default App;
```

Here is how we get the output now:

The next step is to display the above output on our web page. **We added the following map option in the App.tsx file:**

```
{selected ? selected.map((s) => {
     return<p>{s.current_price * s.owned}</p>;
   }) : null}
```

We can see the selected values on the web page:



However, we still have to add a new function, “reduce”, to help us add values to our web page.

Here is the final code version:

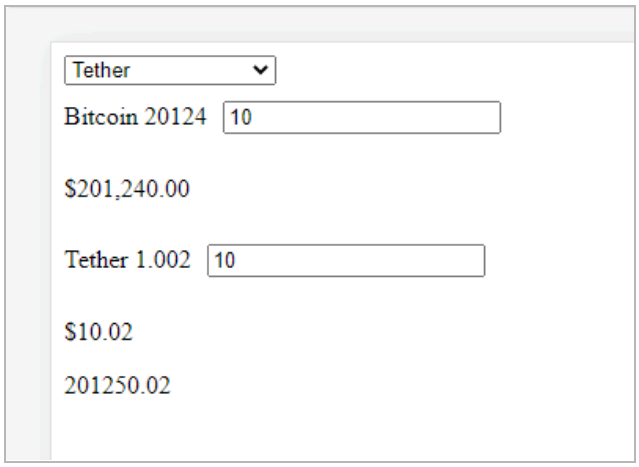
```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';
import CryptoSummary from './Components/CryptoSummary';
import { Crypto } from './Types';
import type { ChartData, ChartOptions } from 'chart.js';
import moment from 'moment';

import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
} from 'chart.js';
import { Line } from 'react-chartjs-2';

ChartJS.register(
  CategoryScale,
```

```
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend
);


function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
  const [selected, setSelected] = useState<Crypto[] >([]);

  /*
  const [data, setData] = useState<ChartData<'line'>>();
  const [options, setOptions] = useState<ChartOptions<'line'>>({
    responsive: true,
  plugins: {
    legend: {
      position: 'top' as const,
    },
    title: {
      display: true,
      text: 'Chart.js Line Chart',
    },
  },
  });
  */
    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
        setCryptos(response.data);
      })
    }, []);

  /*  useEffect(() => {
      if(!selected) return;
```

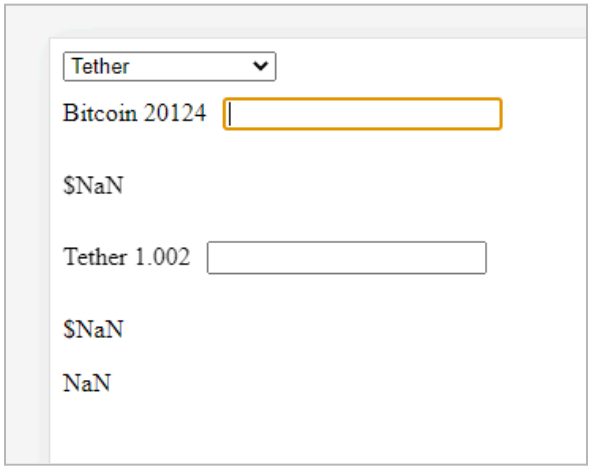
```
    axios.get(

`https://api.coingecko.com/api/v3/coins/${selected?.id}/market_chart?vs_
currency=usd&days=${range}&${
        range === 1 ? 'interval=hourly' : "interval=daily"}`
      ).then((response) => {
        console.log(response.data);
        setData(
         {
          labels: response.data.prices.map((price: number[]) => {

            return moment.unix(price[0] / 1000).format(range === 1 ?
'HH-MM' : 'MM-DD');
          }
          ),
          datasets: [
           {
            label: 'Dataset 1',
            data: response.data.prices.map((price: number[]) => {return
price[1]}),
            borderColor: 'rgb(255, 99, 132)',
            backgroundColor: 'rgba(255, 99, 132, 0.5)',
           },
          ],
         });

        setOptions({
          responsive: true,
          plugins: {
           legend: {
            display: false,
           },
           title: {
            display: true,
            text: `${selected?.name} Price Over Last ` + range + (range
=== 1
             ? ` Day.`
             : ` Day(s).`),
           },
```

```
          },
        })
      });

  }, [selected, range]); */

  useEffect(() => {
    console.log('SELECTED:', selected);
  }, [selected]);


  function updateOwned(crypto: Crypto, amount: number): void{
    console.log('updateOwned', crypto, amount);
    let temp = [...selected];
    let tempObj = temp.find((c)=> c.id === c.id);
    if(tempObj) {
     tempObj.owned = amount;
     setSelected(temp);
    }
  }
      return (
      <>
      <div className="App">
        <select onChange={(e)=>{
          const c = cryptos?.find((x) => x.id === e.target.value) as Crypto;
          setSelected([...selected, c]);
        } }

        defaultValue = "default"
        >
   {cryptos ? cryptos.map((crypto) => {
    return <option key={crypto.id}
value={crypto.id}>{crypto.name}</option>;
    }) : null }
    <option value = 'default'>Choose an Option </option>
    </select>

  </div>
```

```
    {selected.map((s) => {return <CryptoSummary crypto = {s}
updateOwned={updateOwned}/>;
    })}


    {/*selected ? <CryptoSummary crypto={selected} /> : null*/}
   {/*data ? <div style= {{width:600}}>
    <Line options={options} data={data}/></div> : null */ }

    {selected
    ?
    selected
    .map((s) => {
      return s.current_price * s.owned;
    })
    .reduce((prev, current)=>{
      return prev + current;
    }, 0)
    : null}

    </>

    );

}

export default App;
```



However, there is still one problem. When we don't enter any number, it shows NaN which seems weird. **Here is an example:**



We will make a few changes to the App.tsx and CryptoSummary.tsx files to resolve the issue.

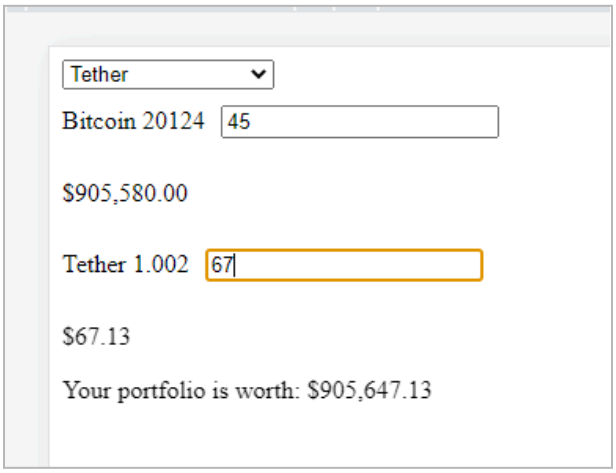
Here are the changes made in the CryptoSummary.tsx file:

```
<p>

  { isNaN(amount)
  ? '$0.00'
  : '$' +


 (crypto.current_price * amount).toLocaleString(undefined,
{minimumFractionDigits: 2,
```

```
        maximumFractionDigits: 2}
        )}

        </p>
```

Then, the changes made in the App.tsx file are:

```
{selected
    ? 'Your portfolio is worth: $' +
    selected
    .map((s) => {

      if(isNaN(s.owned)) {
        return 0;
      }
      return s.current_price * s.owned;
    })
    .reduce((prev, current)=>{
      return prev + current;
    }, 0).toLocaleString(undefined, {
      minimumFractionDigits: 2,
      maximumFractionDigits: 2,
    })
    : null}
```



The next section will display our currencies using a pie chart.

Pie Chart with Chart.js (react-charts-2)

As discussed, we will use the pie chart to display our data. **You can learn about it there:**

https://react-chartjs-2.js.org/examples/pie-chart/

We will implement the React pie chart in our app through the App.tsx file. **Here is the code:**

```
import './App.css';
import { useEffect, useState } from 'react';
import axios from 'axios';
import CryptoSummary from './Components/CryptoSummary';
import { Crypto } from './Types';
import type { ChartData, ChartOptions } from 'chart.js';
import moment from 'moment';

import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
  Title,
  Tooltip,
  Legend,
  ArcElement,
} from 'chart.js';
import { Pie } from 'react-chartjs-2';

ChartJS.register(

  ArcElement,
  Tooltip,
  Legend,

);
```

```
function App() {
  const [cryptos, setCryptos] = useState<Crypto[] | null >(null);
  const [selected, setSelected] = useState<Crypto[] >([]);
  const [data, setData] = useState<ChartData<'pie'>>();

    useEffect(() => {
      const url =
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd&order=
market_cap_desc&per_page=100&page=1&sparkline=false';
      axios.get(url).then((response) => {
        setCryptos(response.data);
      })
    }, []);

  /*  useEffect(() => {
      if(!selected) return;
      axios.get(

`https://api.coingecko.com/api/v3/coins/${selected?.id}/market_chart?vs_
currency=usd&days=${range}&${
          range === 1 ? 'interval=hourly' : "interval=daily"}`
        ).then((response) => {
          console.log(response.data);
          setData(
            {
              labels: response.data.prices.map((price: number[]) => {

                return moment.unix(price[0] / 1000).format(range === 1 ?
'HH-MM' : 'MM-DD');
              }
              ),
              datasets: [
                {
                  label: 'Dataset 1',
                  data: response.data.prices.map((price: number[]) => {return
price[1]}),
                  borderColor: 'rgb(255, 99, 132)',
```

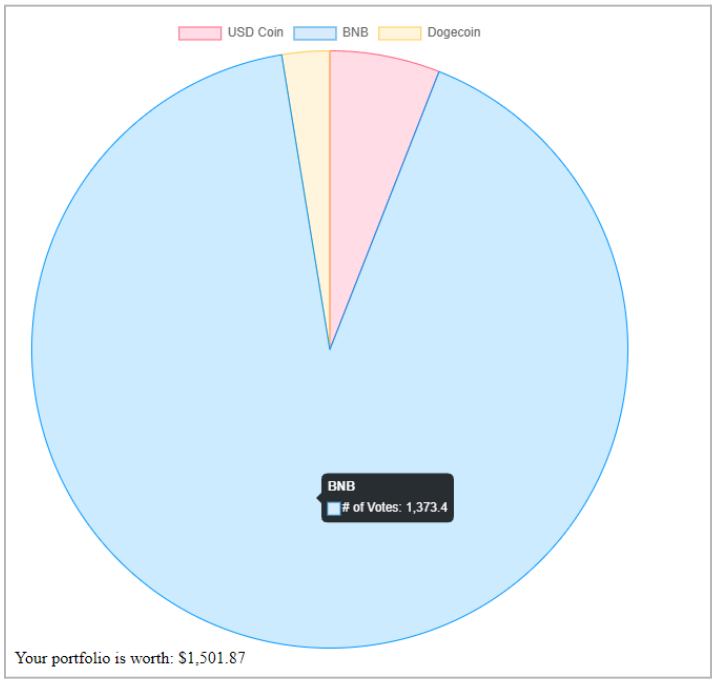
```
            backgroundColor: 'rgba(255, 99, 132, 0.5)',
          },
        ],
      });

      setOptions({
        responsive: true,
        plugins: {
          legend: {
            display: false,
          },
          title: {
            display: true,
            text: `${selected?.name} Price Over Last ` + range + (range
=== 1
            ? ` Day.`
            : ` Day(s).`),
          },
        },
      })
    });

  }, [selected, range]); */

  useEffect(() => {
    console.log('SELECTED:', selected);
    if(selected.length === 0) return;
    setData( {

 labels: selected.map((s) => s.name),
 datasets: [
  {
    label: '# of Votes',
    data: selected.map((s) => s.owned * s.current_price),
    backgroundColor: [
      'rgba(255, 99, 132, 0.2)',
      'rgba(54, 162, 235, 0.2)',
      'rgba(255, 206, 86, 0.2)',
      'rgba(75, 192, 192, 0.2)',
```

```
        'rgba(153, 102, 255, 0.2)',
        'rgba(255, 159, 64, 0.2)',
      ],
      borderColor: [
        'rgba(255, 99, 132, 1)',
        'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)',
        'rgba(75, 192, 192, 1)',
        'rgba(153, 102, 255, 1)',
        'rgba(255, 159, 64, 1)',
      ],
      borderWidth: 1,
    },
  ],
})
    }, [selected]);


    function updateOwned(crypto: Crypto, amount: number): void{
      console.log('updateOwned', crypto, amount);
      let temp = [...selected];
      let tempObj = temp.find((c)=> c.id === crypto.id);
      if(tempObj) {
       tempObj.owned = amount;
       setSelected(temp);
      }
    }
        return (
        <>
        <div className="App">
          <select onChange={(e)=>{
            const c = cryptos?.find((x) => x.id === e.target.value) as Crypto;
            setSelected([...selected, c]);
          } }

          defaultValue = "default"
          >
     {cryptos ? cryptos.map((crypto) => {
      return <option key={crypto.id}
```

```
value={crypto.id}>{crypto.name}</option>;
     }) : null }
     <option value = 'default'>Choose an Option </option>
     </select>

    </div>

     {selected.map((s) => {return <CryptoSummary crypto = {s}
updateOwned={updateOwned}/>;
    })}


    {/*selected ? <CryptoSummary crypto={selected} /> : null*/}
   {data ? <div style= {{width:600}}>
    <Pie data={data}/></div> : null}

    {selected
    ? 'Your portfolio is worth: $' +
    selected
    .map((s) => {

      if(isNaN(s.owned)) {
        return 0;
      }
      return s.current_price * s.owned;
    })
    .reduce((prev, current)=>{
      return prev + current;
    }, 0).toLocaleString(undefined, {
      minimumFractionDigits: 2,
      maximumFractionDigits: 2,
    })
    : null}

    </>

    );

}
```

```
export default App;
```

You can see the final results as under:



Key Takeaways

- Master the fundamentals of React for modern web development.
- Effortlessly style React components using Tailwind CSS.
- Implement secure user authentication with JSON Web Tokens (JWTs).
- Perform CRUD operations by connecting your app to a backend API.
- Visualize data interactively using Chart.js or FusionCharts.
- Create reusable logic with custom hooks for efficient code management.
- Handle errors gracefully and provide feedback to users.
- Optimize performance with techniques like code splitting and lazy loading.
- Deploy your React app to platforms like Heroku or Netlify.

React is a powerful tool because it allows developers to build websites more quickly and efficiently by breaking the UI into smaller, reusable components.

This makes the code cleaner and more maintainable and allows for faster updates, resulting in a more responsive user experience. Its popularity stems from how it simplifies web development, making it more efficient and less complex.

Further Reading

Official Documentation: Keep abreast on all things React! For the latest recent news and updates on React development, visit https://reactjs.org/. React's essential ideas, sophisticated features, and recommended practices are all covered in detail and up to date in the official documentation available at reactjs.org.

React Blog: Direct from the React team, the React blog (https://reactjs.org/blog/) provides information about new features, upgrades, and best practices.

React Podcast: Discussions with React developers can be found on the React Podcast (https://reactpodcast.com/).

Reactiflux: Join Reactiflux (https://www.reactiflux.com is a Discord community where React developers can ask questions, share knowledge, and stay updated on the latest trends in React development.

React Newsletter: Weekly updates on React news, tutorials, and resources are provided by The React Newsletter (https://reactnewsletter.com/).

React Courses and Tutorials: Platforms like Udemy, Pluralsight, and Frontend Masters offer courses and tutorials covering advanced React topics, providing hands-on learning and practical insights.

React Conferences and Meetups: Attending React conferences and local meetups offers valuable insights into new patterns, advanced techniques, and opportunities to network within the React community.

Project Ideas

👉To-Do List App

Create a task list app. Tasks can be added, changed, and removed. This teaches you how to update the content displayed on the screen and manage information.

👉Weather App

Make an application that displays the weather. You will discover how to retrieve meteorological data from the internet and display it on your application.

👉E-commerce Website

Construct a basic internet store. Products can be viewed, added to carts, and purchased by users. You will learn how to handle customer interactions and store management from this.

👉Blog Platform

Provide a space for people to create and publish blog entries. This aids in your comprehension of creating, revising, and displaying blog entries on a website.

👉Portfolio Website

Make your website to showcase your talents and work. This shows you how to create a visually appealing and functional website.