Lesson 05 Demo 04

Create a React Application Demonstrating Retrieval of Data

Objective: To develop a React application that demonstrates retrieval of data asynchronously

Tools required: Node Terminal, React app, and Visual Studio Code

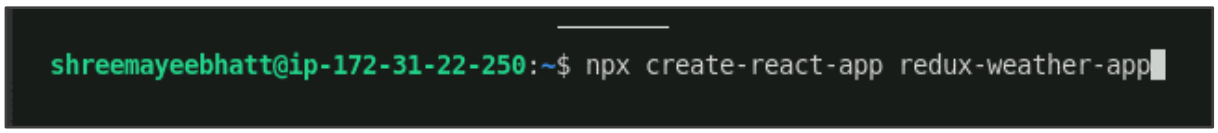
Prerequisites: Knowledge of creating a React app and understanding of the folder structure

Steps to be followed:

1. Create a new **React** app
2. Create a new file called **reducers.js**
3. Create a new file called **types.js**
4. Create a new file called **actions.js**
5. Create a new file called **Weather.js**
6. Create a new file called **WeatherForm.js**
7. Open the existing file **App.js** in the **src** folder
8. Create a new file called **index.js**
9. Create a new file called .**env**
10. Update the **axios.get** URL in **actions.js**
11. Run the app and view it in the browser

Step 1: Create a new React app

1.1 Open your terminal and run the **npx create-react-app redux-weather-app** command



Note: this command will create a new React app with the name redux-weather-app

1.2 Move to the **redux-weather-app** directory by running the **cd redux-weather-app** command in the terminal

1.3 Install the necessary dependencies by running the command **npm install redux react-redux redux-thunk axios**

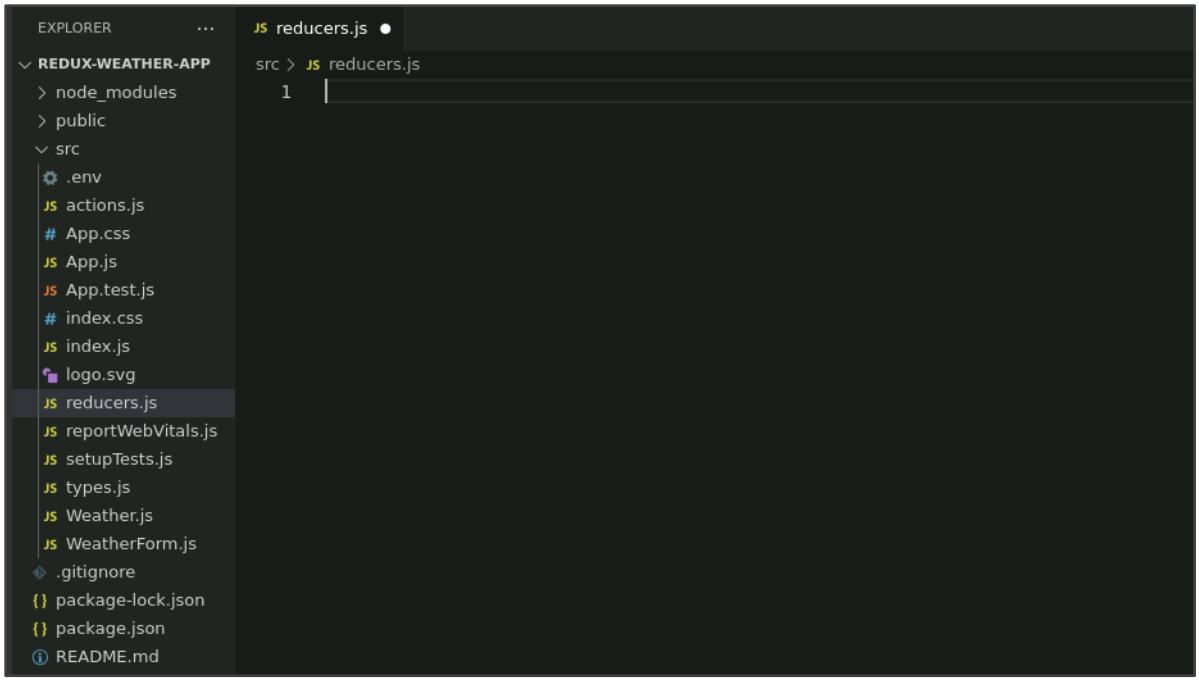
```
shreemayeebhatt@ip-172-31-22-250:~$ cd redux-weather-app/
shreemayeebhatt@ip-172-31-22-250:~/redux-weather-app$ npm install redux react-redux redux-thunk axios
```

Step 2: Create a new file called reducers.js

2.1 Open **Visual Studio Code**

2.2 Navigate to the project folder

2.3 In the **src** directory, create a new file named **reducers.js**



2.4 Import the necessary action types from **./types**

```
import { FETCH_WEATHER_REQUEST, FETCH_WEATHER_SUCCESS, FETCH_WEATHER_FAILURE } from './types';
```

2.5 Define the **initial state** for the weather data

```
const initialState = {
loading: false,
weatherData: null,
error: null,
};
```

2.6 Create the weather **reducer** function that handles the state updates based on the **dispatched** actions

```
const weatherReducer = (state = initialState, action) => {
switch (action.type) {
case FETCH_WEATHER_REQUEST:
return {
...state,
loading: true,
error: null,
};
case FETCH_WEATHER_SUCCESS:
return {
...state,
loading: false,
weatherData: action.payload,
};
case FETCH_WEATHER_FAILURE:
return {
...state,
loading: false,
error: action.payload,
};
default:
return state;
}
};
```

2.7 Export the **weatherReducer**

```
export default weatherReducer;
```

```
import { FETCH_WEATHER_REQUEST, FETCH_WEATHER_SUCCESS,
FETCH_WEATHER_FAILURE } from './types';
const initialState = {
loading: false,
weatherData: null,
error: null,

};
const weatherReducer = (state = initialState, action) => {
switch (action.type) {
case FETCH_WEATHER_REQUEST:
return {
...state,
loading: true,
error: null,
};
case FETCH_WEATHER_SUCCESS:
return {
...state,
loading: false,
weatherData: action.payload,
};
case FETCH_WEATHER_FAILURE:
return {
...state,
loading: false,
error: action.payload,
};
default:
return state;
}
```

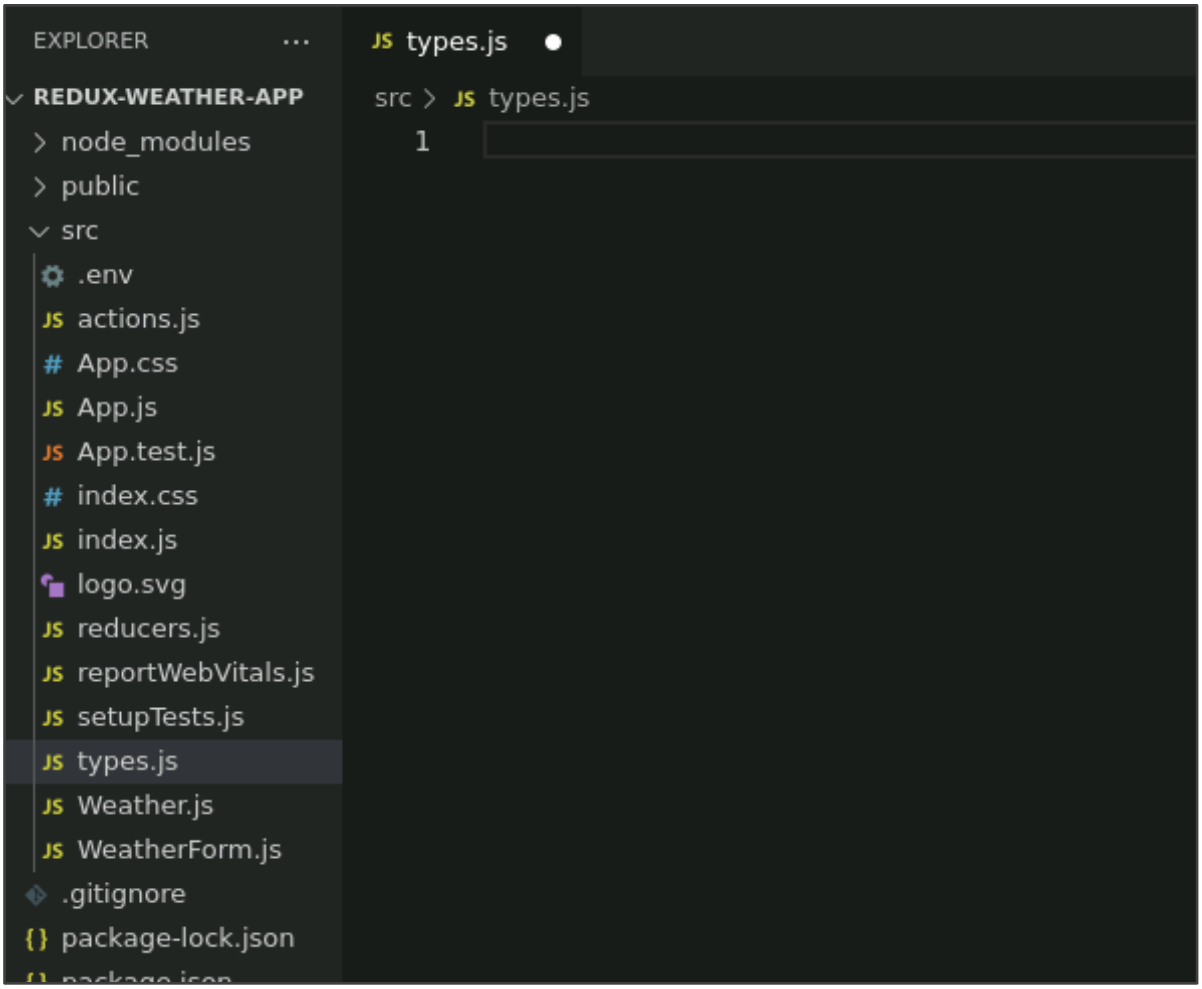
```
};
export default weatherReducer;
```

```
import { FETCH_WEATHER_REQUEST, FETCH_WEATHER_SUCCESS, FETCH_WEATHER_FAILURE } from './types';

const initialState = {
loading: false,
weatherData: null,
error: null,
};

const weatherReducer = (state = initialState, action) => {
switch (action.type) {
case FETCH_WEATHER_REQUEST:
return {
...state,
loading: true,
error: null,
};
case FETCH_WEATHER_SUCCESS:
return {
...state,
loading: false,
weatherData: action.payload,
};
case FETCH_WEATHER_FAILURE:
return {
...state,
loading: false,
error: action.payload,
};
default:
return state;
}
};

export default weatherReducer;
```

This will contain the **reducer** function that will manage the state of the weather data

Step 3: Create a new file called types.js

3.1 In the **src** directory, create a new file named **types.js**



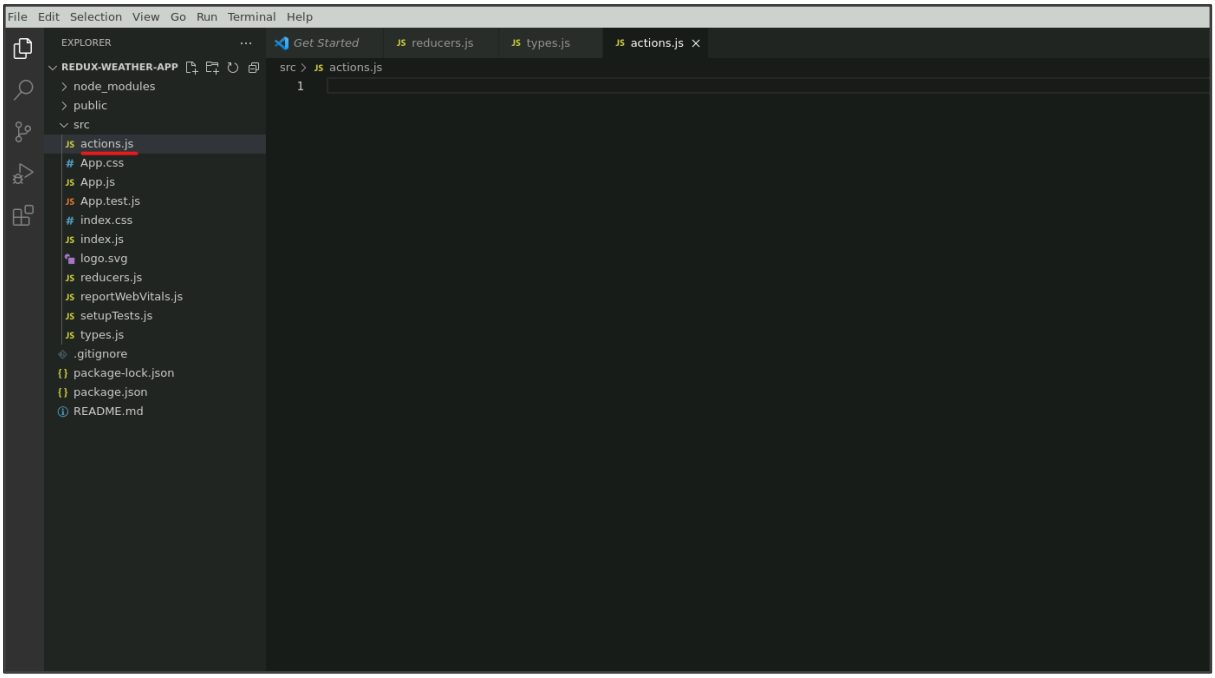
3.2 Define the action types as constants that will be used in the **reducer** and **actions**

```
types.js > ...
export const FETCH_WEATHER_REQUEST = 'FETCH_WEATHER_REQUEST';
export const FETCH_WEATHER_SUCCESS = 'FETCH_WEATHER_SUCCESS';
export const FETCH_WEATHER_FAILURE = 'FETCH_WEATHER_FAILURE';
```

This will contain the action types that we will use in our **reducer**

Step 4: Create a new file called actions.js

4.1 In the **src** directory, create a new file named **actions.js**



4.2 Import **Axios** for making HTTP requests, the action types from **./types**, and the necessary dependencies for **async** actions

```
import axios from 'axios';
import { FETCH_WEATHER_REQUEST, FETCH_WEATHER_SUCCESS, FETCH_WEATHER_FAILURE } from './types';
```

4.3 Define the **fetchWeather** action creator that makes an asynchronous **API** call to fetch weather data based on the city

4.4 Dispatch the corresponding actions based on the success or failure of the **API** request

```
export const fetchWeather = city => {
return async dispatch => {
dispatch({ type: FETCH_WEATHER_REQUEST });

try {
const response = await axios.get(`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=YOUR_API_KEY_HERE`);
dispatch({ type: FETCH_WEATHER_SUCCESS, payload: response.data });
} catch (error) {
dispatch({ type: FETCH_WEATHER_FAILURE, payload: error.message });
}
};
};
```

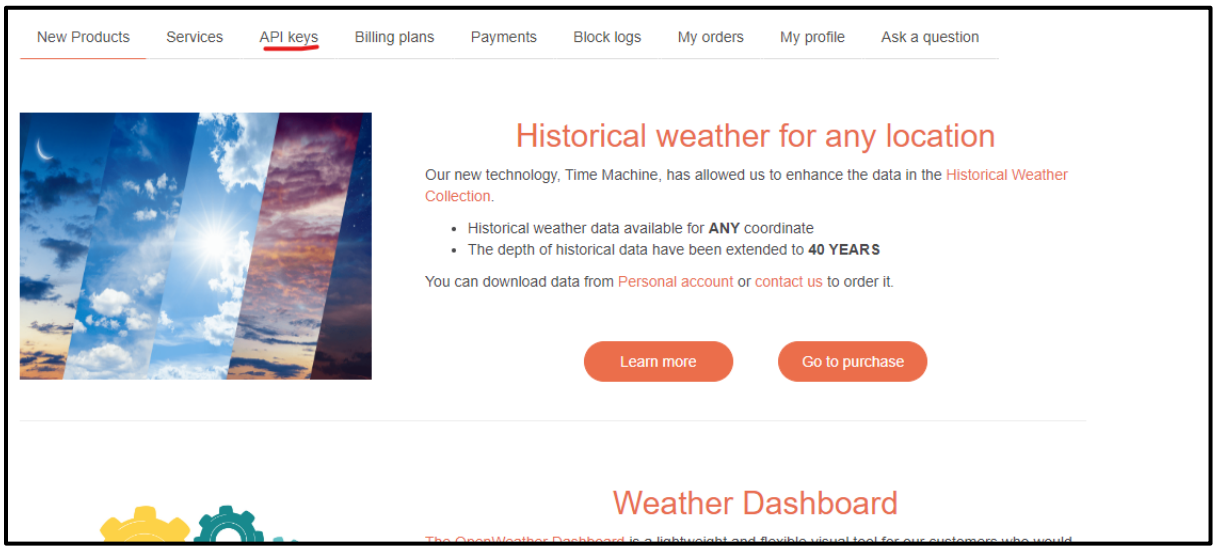
```
import axios from 'axios';

import { FETCH_WEATHER_REQUEST, FETCH_WEATHER_SUCCESS,
FETCH_WEATHER_FAILURE } from './types';
export const fetchWeather = city => {
return async dispatch => {

dispatch({ type: FETCH_WEATHER_REQUEST });
try {
const response = await
axios.get(`https://api.openweathermap.org/data/2.5/weather?q=${city}&
appid=YOUR_API_KEY_HERE`);
dispatch({ type: FETCH_WEATHER_SUCCESS, payload: response.data });
} catch (error) {
dispatch({ type: FETCH_WEATHER_FAILURE, payload: error.message });
}

};
};
```

Note:

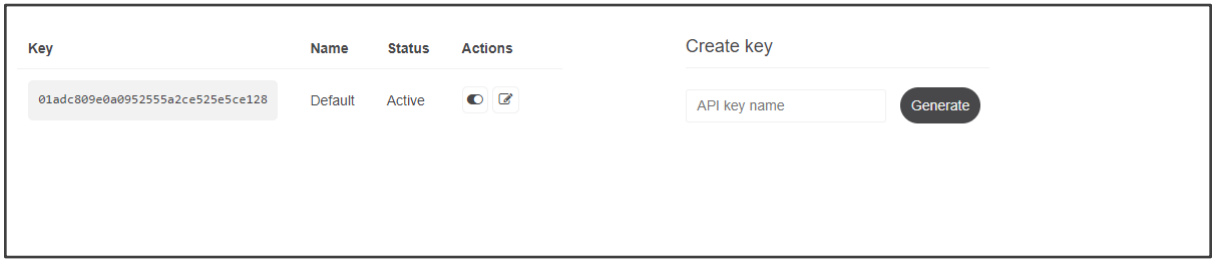
To get the **API** Key:

1. Go to **https://api.openweathermap.org** and sign up

2. Go to **API** keys on the home page



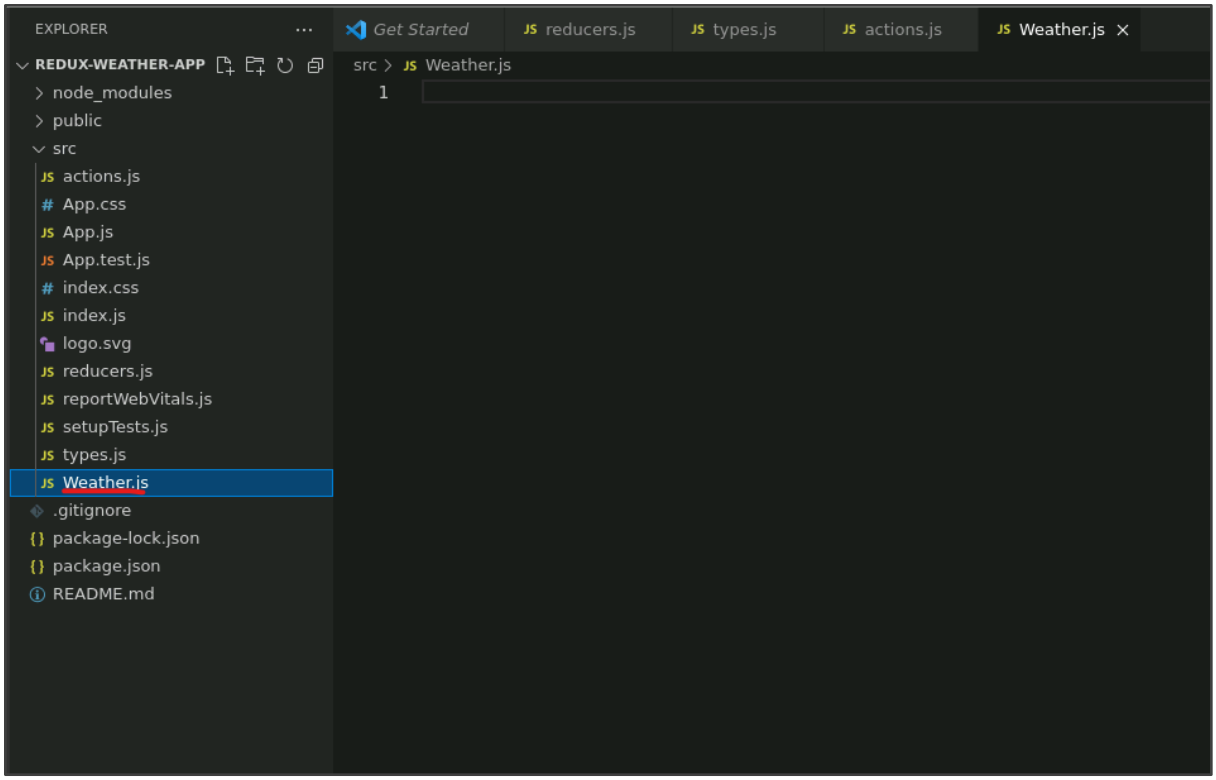
3. Copy the key and paste it in a code



This will contain the action creators that will dispatch the actions to the reducer

Step 5: Create a new file called Weather.js

5.1 In the **src** directory, create a new file called **Weather.js**



5.2 Create the functional component **Weather** that receives the loading, **weatherData**, and **error props**

5.3 Render different elements based on the loading and error states, displaying the weather data if available

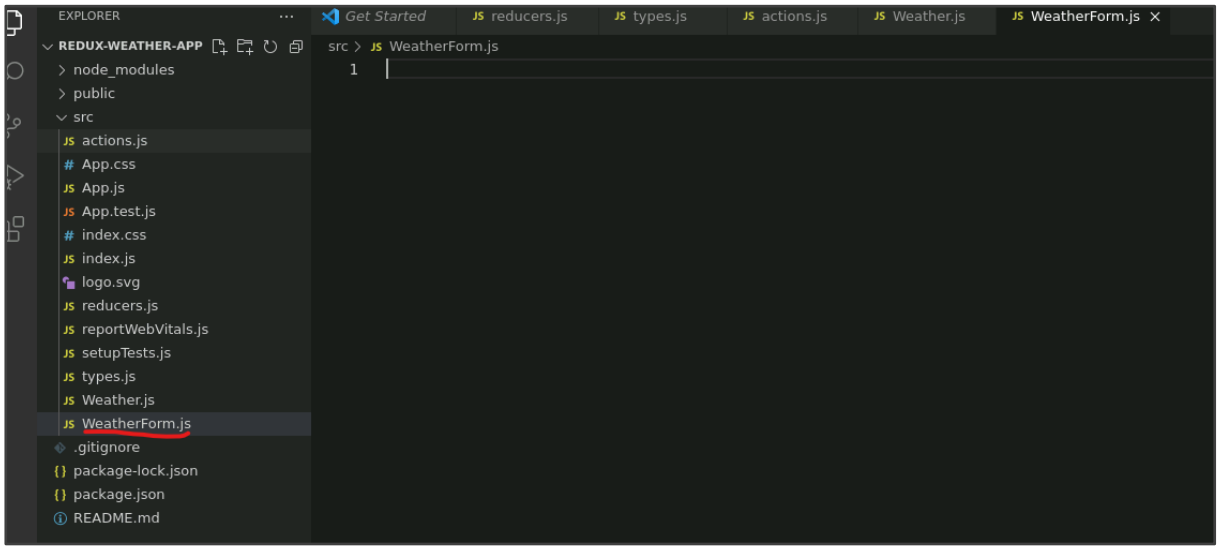
```
import React from 'react';
function Weather({ loading, weatherData, error }) {
if (loading) {
return <div>Loading...</div>;
}

if (error) {
return <div>{error}</div>;
```

```
}

if (!weatherData) {
return null;
}

const { name, main } = weatherData;

return (
<div>
<h2>{name}</h2>
<p>Current temperature: {main.temp}°F</p>
<p>Feels like: {main.feels_like}°F</p>
<p>Humidity: {main.humidity}%</p>
<p>Pressure: {main.pressure} hPa</p>
</div>
);
}

export default Weather;
```

```
import React from 'react';

function Weather({ loading, weatherData, error }) {
if (loading) {
return <div>Loading...</div>;
}

if (error) {
return <div>{error}</div>;
}

if (!weatherData) {
return null;
}

const { name, main } = weatherData;

return (
<div>
<h2>{name}</h2>
<p>Current temperature: {main.temp}°F</p>
<p>Feels like: {main.feels_like}°F</p>
<p>Humidity: {main.humidity}%</p>
<p>Pressure: {main.pressure} hPa</p>
</div>
);
}

export default Weather;
```

Step 6: Create a new file called WeatherForm.js

6.1 In the **src** directory, create a new file named **WeatherForm.js**



6.2 Create the functional component **WeatherForm** that receives the **onSubmit** props

6.3 Use the **useState** hook to manage the input value for the city

6.4 Handle form submission by calling the **onSubmit** function with the city value

```
import React, { useState } from 'react';
function WeatherForm({ onSubmit })
{
const [city, setCity] = useState('');
const handleSubmit = e => {
e.preventDefault();
onSubmit(city);
};
return (

<form onSubmit={handleSubmit}>
<input type="text" value={city} onChange={e => setCity(e.target.value)} />
<button type="submit">Get Weather</button>

</form>
```

```
);
}
export default WeatherForm;
```

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

function WeatherForm({ onSubmit }) {
const [city, setCity] = useState('');

const handleSubmit = e => {
e.preventDefault();
onSubmit(city);
};

return (
<form onSubmit={handleSubmit}>
<input type="text" value={city} onChange={e => setCity(e.target.value)} />
<button type="submit">Get Weather</button>
</form>
);
}

export default WeatherForm;
```

This will be our presentational component that will display the form to enter a city and submit it to fetch the weather data

Step 7: Open the existing file App.js in the src folder

7.1 In the **src** directory, create a new file named **App.js**

7.2 Import the necessary dependencies, including the **Weather**, **WeatherForm components**, and the **fetchWeather action creator**

7.3 Create the app's functional component that makes use of the **useSelector** and **useDispatch** hooks to access the Redux store's and actions' dispatch mechanisms

7.4 Render the **WeatherForm** and **Weather** components, passing the necessary props

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { fetchWeather } from './actions';
import Weather from './Weather';
import WeatherForm from './WeatherForm';

function App() {

const weatherData = useSelector(state => state.weatherData);
const loading = useSelector(state => state.loading);
const error = useSelector(state => state.error);
const dispatch = useDispatch();
const handleSubmit = city => {
dispatch(fetchWeather(city));
};

return (
<div>
<h1>Weather App</h1>

<WeatherForm onSubmit={handleSubmit} />
<Weather loading={loading} weatherData={weatherData} error={error} />
</div>
);
}
export default App;
```

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { fetchWeather } from './actions';
import Weather from './Weather';
import WeatherForm from './WeatherForm';

function App() {
const weatherData = useSelector(state => state.weatherData);
const loading = useSelector(state => state.loading);
const error = useSelector(state => state.error);
const dispatch = useDispatch();

const handleSubmit = city => {
dispatch(fetchWeather(city));
};

return (
<div>
<h1>Weather App</h1>
<WeatherForm onSubmit={handleSubmit} />
<Weather loading={loading} weatherData={weatherData} error={error} />
</div>
);
}

export default App;
```

Step 8: Create a new file called index.js

8.1 In the **src** directory, open the **index.js** file

8.2 Import the necessary dependencies, including the **Provider** component**,** the **createStore**, **applyMiddleware** functions, and the **App** component

8.3 Create the Redux store using the **createStore** function, passing the **weatherReducer** and **applyMiddleware(thunk)** as arguments

8.4 Wrap the **App** component with the **Provider** component, passing the Redux store as a props

8.5 Use the **ReactDOM.render** function to render the wrapped App component to the root element in the HTML document

```
import React from 'react';

import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import weatherReducer from './reducers';
import App from './App';
const store = createStore(weatherReducer, applyMiddleware(thunk));
ReactDOM.render(
<Provider store={store}>
    <App />
</Provider>,
document.getElementById('root')
);
```

This will be the entry point of our application

Step 9: Create a new file called .env

9.1 In the **src** directory, create a new file named **.env**

9.2 Set the API key from the **OpenWeatherMap** API as an environment variable in the **.env file** - **REACT_APP_API_KEY=YOUR_API_KEY_HERE**

Step 10: Update the axios.get URL in actions.js

10.1 In the **actions.js** file, update the axios.get URL to include the API key from the **.env** file

const response = await axios.get(`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${process.env.REACT_APP_API_KEY}`);

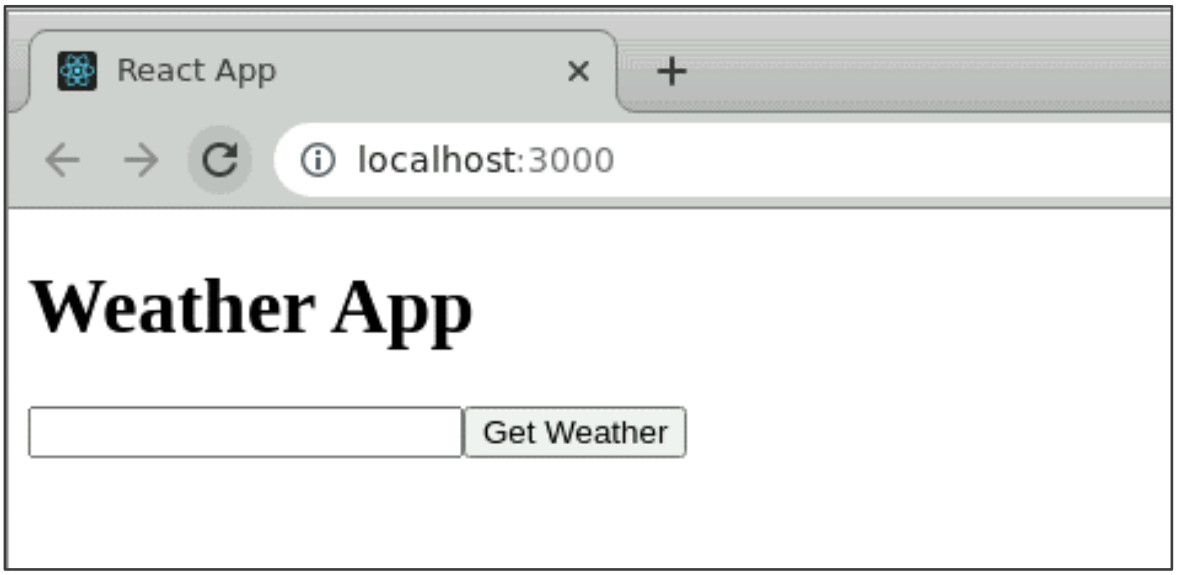
Step 11: Run the app and view it in the browser

11.1 In the terminal, navigate to the project directory

11.2 Run the command **npm start** to start the development server

11.3 Open your browser and navigate to **http://localhost:3000**

The app should be running, and you should see a simple weather app where you can enter a city and get the current weather data displayed



In conclusion, we successfully built a React application that demonstrates data retrieval using Redux. The application utilizes Redux for state management, Redux Thunk for asynchronous actions, and Axios for making HTTP requests. Users can enter a city to fetch and display the current weather data. The application follows best practices by separating concerns into different files and components, enhancing maintainability and reusability.