

Lesson 05 Demo 03

Creating a React Application Using Combine Reducer

Objective: To develop a React application that demonstrates the use of combined reducers

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 AddTodo.js
4. Import the rootReducer from reducers.js into the index.js file
5. Run the application 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 combined-reducers-todo** command

Note: This command will create a new **React** app with the name **combined-reducers-todo**

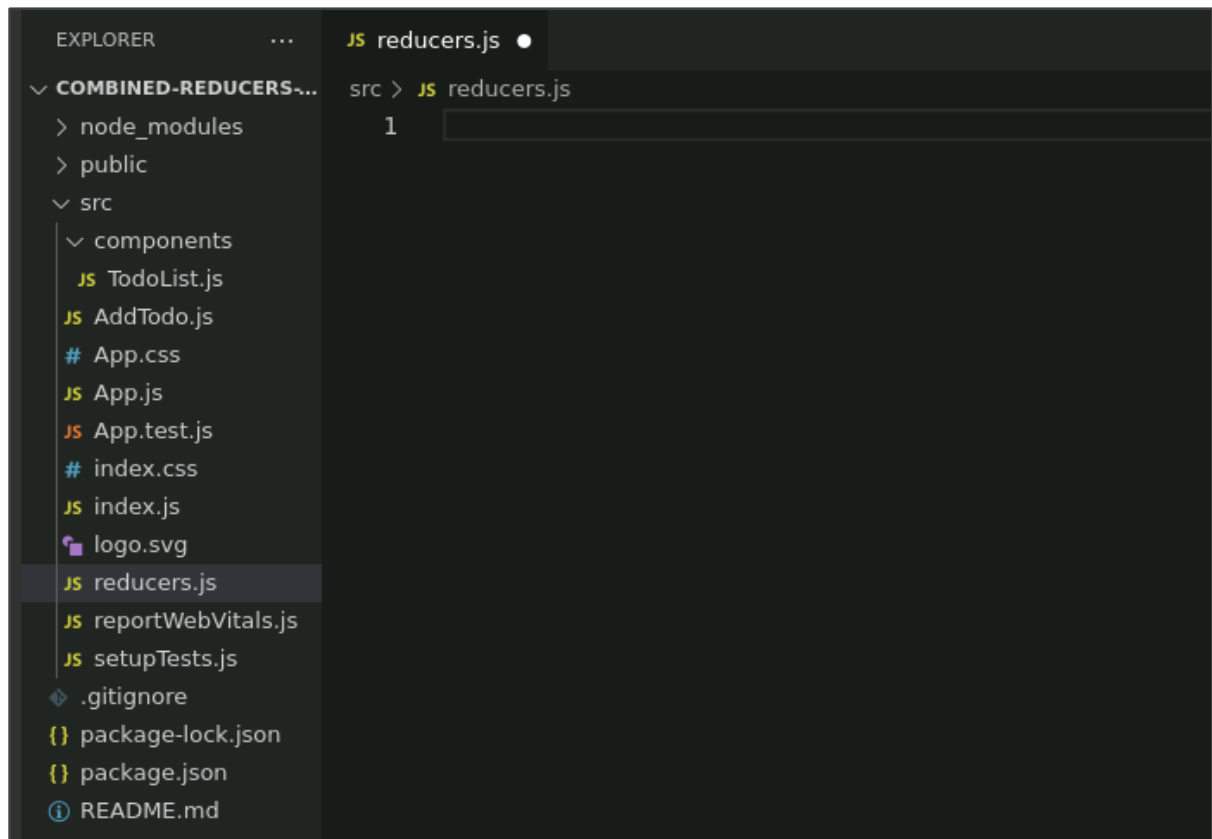
```
shreemayeebhatt@ip-172-31-22-250:~$ npx create-react-app combined-reducers-todo
```

- 1.2 Move to the **combined-reducers-todo** directory by running the **cd combined-reducers-todo** command in the terminal
- 1.3 Install the necessary dependencies by running the **npm install redux react-redux** command

```
shreemayeebhatt@ip-172-31-22-250:~$ cd combined-reducers-todo/  
shreemayeebhatt@ip-172-31-22-250:~/combined-reducers-todo$ npm install redux react-redux
```

Step 2: Create a new file called reducers.js

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



2.2 Import the **combineReducers** function from the **redux** package

2.3 Define a **reducer** function for the **todoList** state

2.4 Use the **combineReducers** function to combine the reducers into a **rootReducer**

2.5 Export the **rootReducer**

```
import { combineReducers } from 'redux';
const todoListReducer = (state = [], action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return [...state, action.payload];
    case 'REMOVE_TODO':
```

```
return state.filter(todo => todo.id !== action.payload);  
default:
```

```
return state;  
}
```

```
};
```

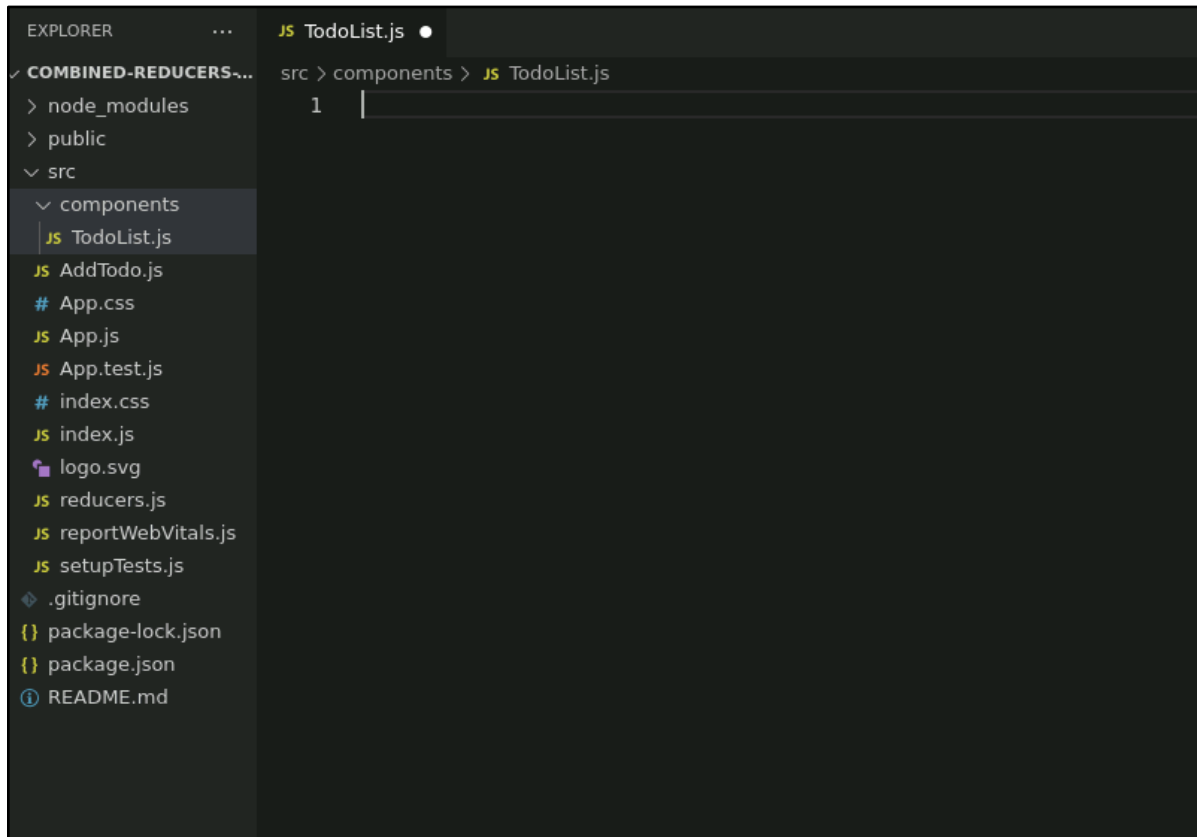
```
const rootReducer = combineReducers({  
  todoList: todoListReducer,  
});
```

```
export default rootReducer;
```

```
> JS reducers.js > ...  
1  import { combineReducers } from 'redux';  
2  
3  const todoListReducer = (state = [], action) => {  
4    switch (action.type) {  
5      case 'ADD_TODO':  
6        return [...state, action.payload];  
7      case 'REMOVE_TODO':  
8        return state.filter(todo => todo.id !== action.payload);  
9      default:  
10       return state;  
11    }  
12  };  
13  
14  const rootReducer = combineReducers({  
15    todoList: todoListReducer,  
16  });  
17  
18  export default rootReducer;  
19  |
```

2.6 Create a new file called **TodoList.js**. This will be your presentational component that will display the **list of todos**

2.7 In the **src/components** directory, create the **TodoList.js** file



2.8 Create a functional component named **TodoList** that receives props for **todos** and **removeTodo**

2.9 Render a list of **todos**, displaying the text of each **todo** and a button to remove it

2.10 Export the **TodoList** component

```
import React from 'react';
```

```
function TodoList({ todos, removeTodo }) {  
  return (  
    <ul>
```

```
    {todos.map(todo => (
```

```
<li key={todo.id}>
```

```
{todo.text}
```

```
<button onClick={() => removeTodo(todo.id)}>X</button>
```

```
</li>
```

```
))}
```

```
</ul>
```

```
);
```

```
}
```

```
export default TodoList;
```

```
src > components > JS TodoList.js > ...
```

```
1 import React from 'react';
```

```
2
```

```
3 function TodoList({ todos, removeTodo }) {
```

```
4 return (
```

```
5 <ul>
```

```
6 {todos.map(todo => (
```

```
7 <li key={todo.id}>
```

```
8 {todo.text}
```

```
9 <button onClick={() => removeTodo(todo.id)}>X</button>
```

```
10 </li>
```

```
11 ))}
```

```
12 </ul>
```

```
13 );
```

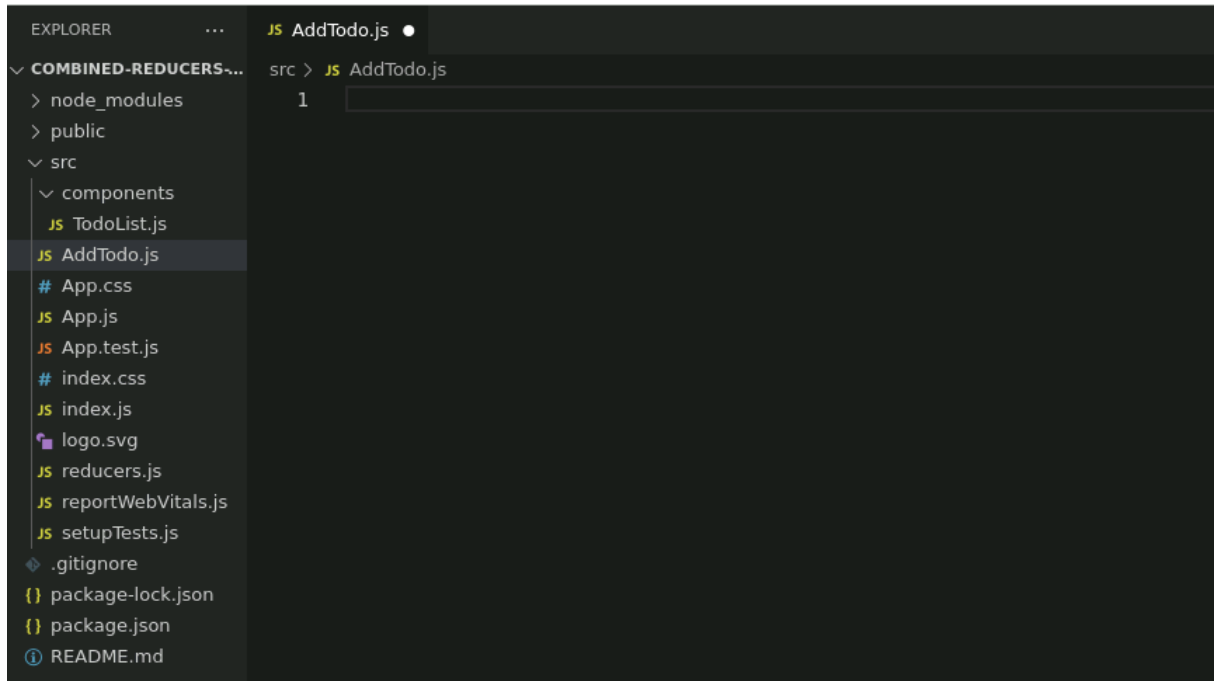
```
14 }
```

```
15
```

```
16 export default TodoList;
```

Step 3: Create a new file called AddTodo.js

3.1 In the **src/components** directory, create a new file named **AddTodo.js**



Note: This will be our presentational component that will display the form to add a new **todo**

3.2 Import **React** and **useState**

3.3 Create a functional component called **AddTodo** that receives a property for **addTodo**

3.4 Manage the input text state using the **useState** hook

3.5 Handle form submission by adding a new **todo** to the list

3.6 Export the **AddTodo** component

```
import React, { useState } from 'react';  
function AddTodo({ addTodo }) {  
  const [text, setText] = useState("");
```

```

const handleSubmit = e => {
  e.preventDefault();
  addToDo({
    id: Date.now(),
    text,
  });
  setText("");
};

return (
  <form onSubmit={handleSubmit}>
    <input type="text" value={text} onChange={e =>
      setText(e.target.value)} />
    <button type="submit">Add Todo</button>
  </form>
);
}

export default AddTodo;

```

```

import React, { useState } from 'react';

function AddTodo({ addToDo }) {
  const [text, setText] = useState('');

  const handleSubmit = e => {
    e.preventDefault();
    addToDo({
      id: Date.now(),
      text,
    });
    setText('');
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={text} onChange={e => setText(e.target.value)} />
      <button type="submit">Add Todo</button>
    </form>
  );
}

export default AddTodo;

```

Note: This will be our container component that will connect the **TodoList** and **AddTodo** components to the Redux store.

3.7 In the **src** directory, open the existing file named **App.js**

3.8 Import **React**, the **useSelector** and **useDispatch** hooks from **react-redux**, and the **TodoList** and **AddTodo** components

3.9 Create the **App** functional component

3.10 Use the **useSelector** hook to select the **TodoList** state from the Redux store

3.11 Use the **useDispatch** hook to get the dispatch function

3.12 Define the **addTodo** and **removeTodo** functions that dispatch the corresponding actions to the Redux store

3.13 Render the **AddTodo** and **TodoList** components, passing the required props

3.14 Export the **App** component

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import TodoList from './components/TodoList';
```

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

```
function App() {
  const todoList = useSelector(state => state.todoList);
  const dispatch = useDispatch();
```

```
  const addTodo = todo => {
    dispatch({ type: 'ADD_TODO', payload: todo });
  };
```

```
  const removeTodo = id => {
    dispatch({ type: 'REMOVE_TODO', payload: id });
```



```

};

return (
  <div>
    <AddTodo addTodo={addTodo} />
    <TodoList todos={todoList} removeTodo={removeTodo} />

  </div>

);
}

export default App;

```

```

import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import TodoList from './components/TodoList';
import AddTodo from './AddTodo';

function App() {
  const todoList = useSelector(state => state.todoList);
  const dispatch = useDispatch();

  const addTodo = todo => {
    dispatch({ type: 'ADD_TODO', payload: todo });
  };

  const removeTodo = id => {
    dispatch({ type: 'REMOVE_TODO', payload: id });
  };

  return (
    <div>
      <AddTodo addTodo={addTodo} />
      <TodoList todos={todoList} removeTodo={removeTodo} />
    </div>
  );
}

export default App;

```

Step 4: Import the rootReducer from reducers.js in the index.js file

Note: This will create the **Redux store** using the **createStore** function from **redux**, and will pass the store to the **Provider** component from **react-redux**, and wrap the **App** component

- 4.1 In the **src** directory, open the existing **index.js** file
- 4.2 Import **React**, **ReactDOM**, the **Provider** component from **react-redux**, the **createStore** function from **redux**, the **rootReducer** from **reducers.js**, and the **App** component
- 4.3 Create the **Redux** store using the **createStore** function and pass the **rootReducer** to it
- 4.4 Wrap the **App** component with the **Provider** component, passing the **Redux store** as a prop
- 4.5 Use the **ReactDOM.render** function to render the wrapped **App** component to the DOM

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { createStore } from 'redux';
import rootReducer from './reducers';
import App from './App';
```

```
const store = createStore(rootReducer);
```

```
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

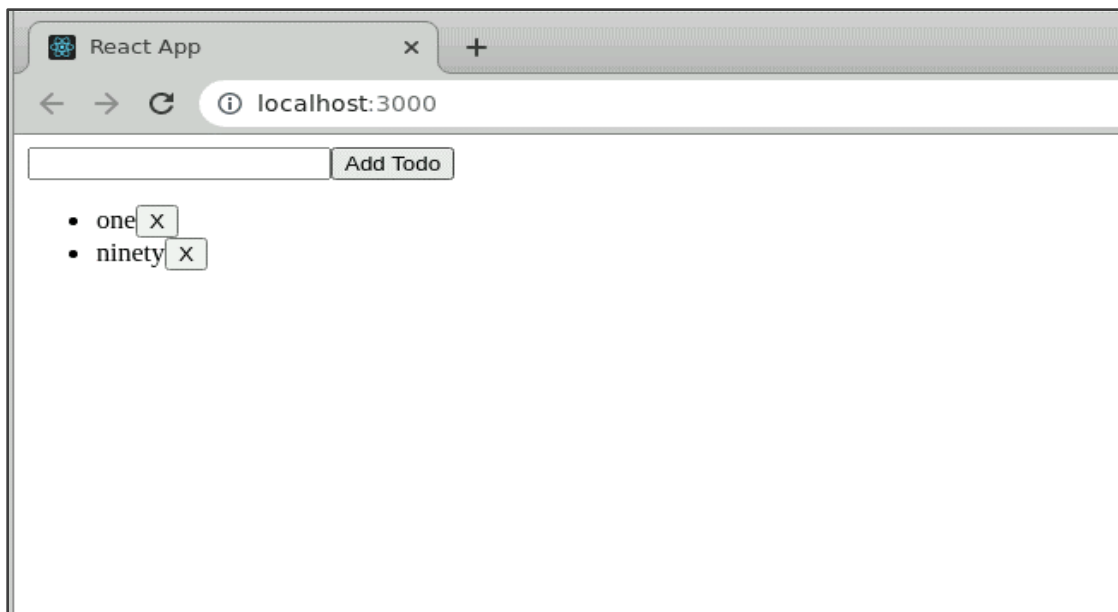
```
index.js / ...  
  
import React from 'react';  
import ReactDOM from 'react-dom';  
import { Provider } from 'react-redux';  
import { createStore } from 'redux';  
import rootReducer from './reducers';  
import App from './App';  
  
const store = createStore(rootReducer);  
  
ReactDOM.render(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

Step 5: Run the app and view it in the browser

5.1 In the terminal, navigate to the project directory

5.2 Run the command **npm start** to start the app

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



The above code demonstrates a systematic process: initiating a React app, configuring reducers in reducers.js to manage state, creating components for displaying and adding todos, connecting components to the Redux store in App.js, and concluding by importing the root reducer in the index for streamlined integration.

With this, you have successfully created a React application using combined reducers to manage state with Redux.