

React With Redux Certification Training

COURSE OUTLINE

MODULE 10

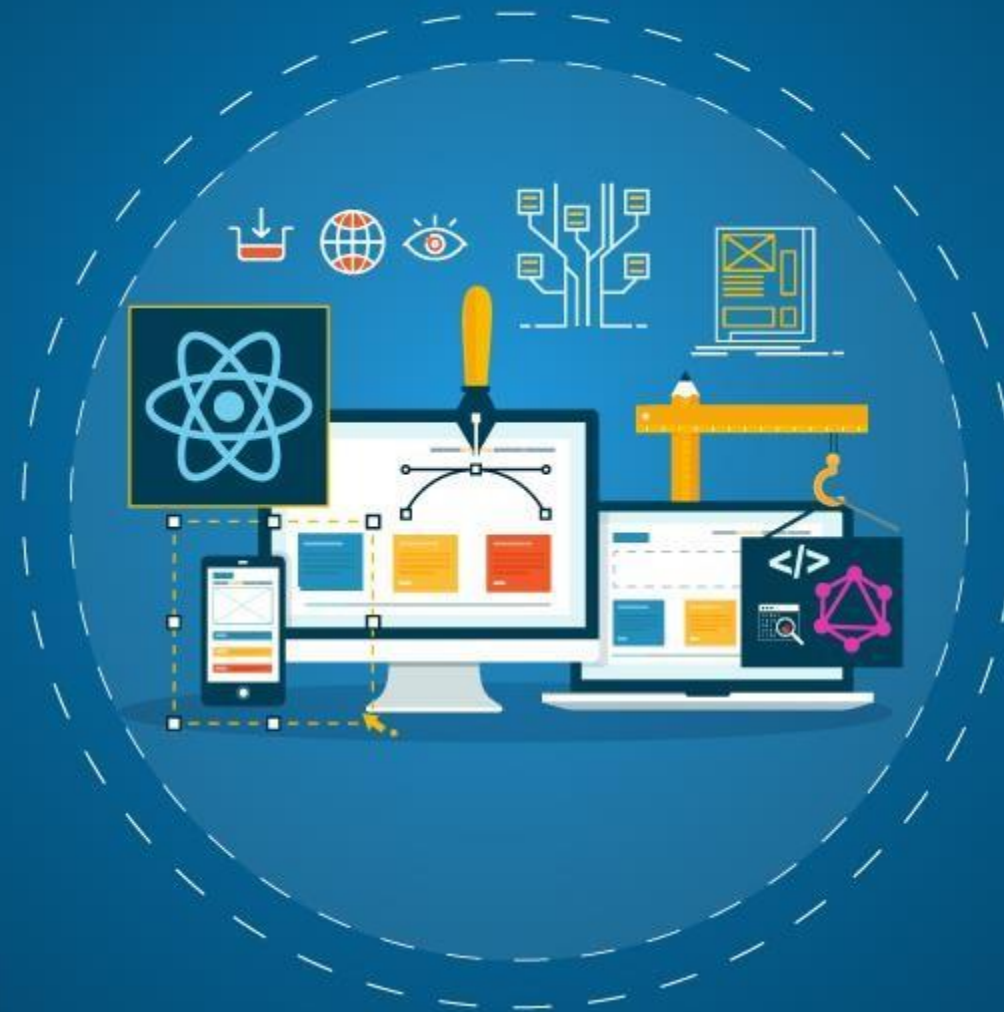
1. Introduction to Web Development and React

2. Components and Styling the Application Layout

3. Handling Navigation with Routes

4. React State Management using Redux

5. Asynchronous Programming with Saga Middleware



6. React Hooks

7. Fetching Data using GraphQL

8. React Application Testing and Deployment

9. Introduction to React Native

10. Building React Native Applications with APIs

Topics

Following are the topics covered in this module:

- Native modules
- Native Navigation libraries
- Integration of Redux with React Native
- React Native and Redux major components
- Redux Thunk middleware
- NPM libraries
- Shopping cart application using React Native and Redux
- Integration of Redux actions, store and reducers In React Native application

Objectives

After completion of this module you should be able to:

- Use of React Native Modules
- Navigate using Navigation libraries
- Handle Async Actions using Thunk middleware
- Recognise the NPM libraries to build React Native and Redux
- Build an application using React Native and Redux



Native Modules

Native Modules

A native module is a set of JavaScript functions that are implemented natively for each platform (in our case is iOS and Android).



It is used in cases where native capabilities are needed, when react native lacks a corresponding module, or when the native performance is to be better.



React Native Navigation Libraries



React Native Navigation Libraries

The majorly used Navigation libraries are: *React Navigation* and *React Native Navigation*

*React
Navigation*

- It is written in **JavaScript** and does not directly use the native navigation APIs on iOS and Android
- It **recreates** some subset of those APIs
- It allows for **integration** of third-party JS plugins, maximum customization, and easier debugging, without having prior knowledge of Objective-C, Swift, Java, Kotlin, etc.

*React Native
Navigation*

React Native Navigation Libraries

The majorly used Navigation libraries are: *React Navigation* and *React Native Navigation*

*React
Navigation*

*React Native
Navigation*

React Native Navigation differs slightly, it directly uses native navigation APIs on iOS and Android, which allows for a more native look and feel.



It is easy to display static
data using React Native
mobile application.



But, in industry we don't practise with static data, customers requirement vary, to meet these requirements we have to build the application that displays bulk amount of data.

How should I manage my customer requirements without compromising my application performance?





Under such circumstances
make use of ***API's*** to integrate
bulky data. To consume the API
and for better management
the application make use of
React Redux.

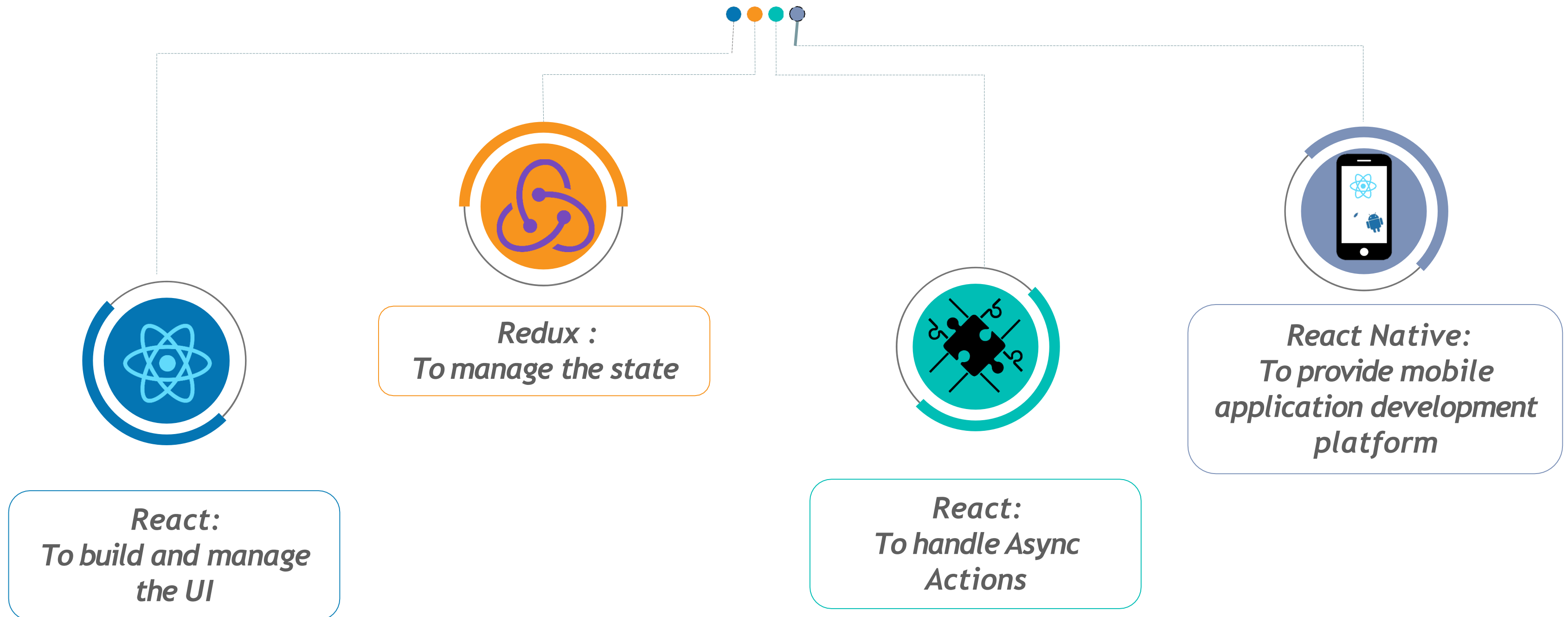


Integrating React Redux With React Native



React Native And Redux

Requirement to perform the integration React Native and Redux are:





Middleware: Redux Thunk



Redux Thunk



Redux Thunk is a middleware that lets you call action creators that return a function instead of an action object



This function receives the **store's dispatch method**, which is then used to dispatch **regular synchronous actions** inside the body of the function once the asynchronous operations have completed



The most common use-case for Redux Thunk is for **communicating asynchronously** with an **external API** to retrieve or save data



Redux Thunk makes it easy to dispatch actions that follow the lifecycle of a request to an **external API**.



You can install it in react applications using: `npm install --save redux-thunk`

NPM

NPM Libraries

Major NPM libraries being used are:

react-redux

Install it using: npm i react-redux

redux-thunk

Install it using: npm i redux-thunk

expo-cli

Install it using: npm i expo-cli



Demo: Shopping Cart Application Using Redux And React Native



We will continue the
integration of the Redux within
the shopping cart application
built in last module.



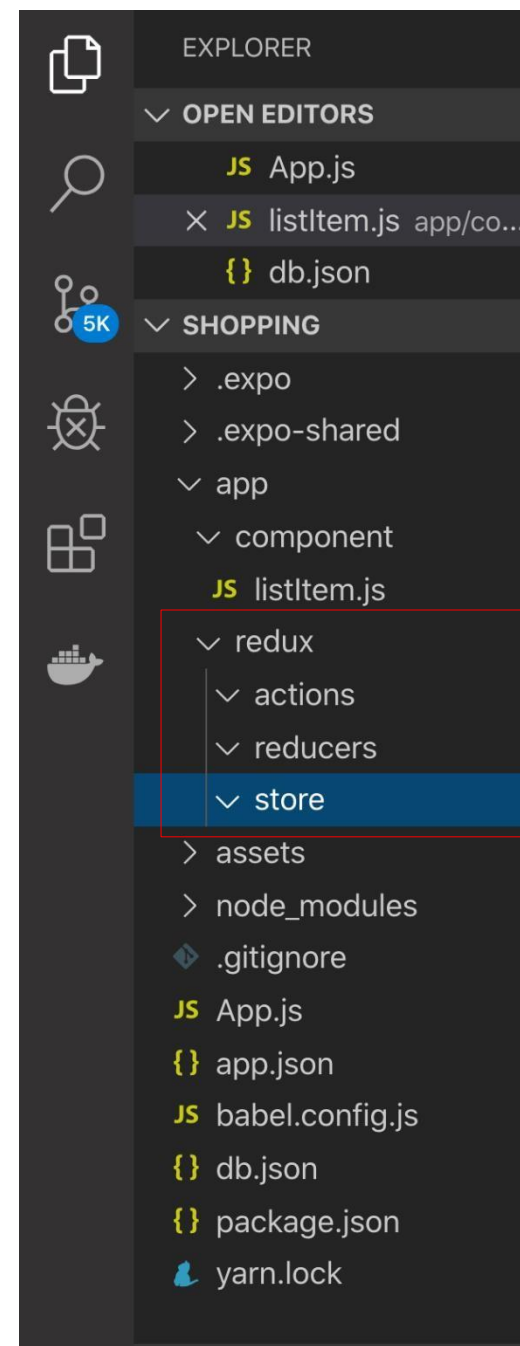
Demo: Redux Setup

Install *Redux and Thunk* middleware to integrate Redux in React Native application.

```
Avyaans-MacBook-Pro:shopping avi$ npm i redux react-redux redux-thunk
```

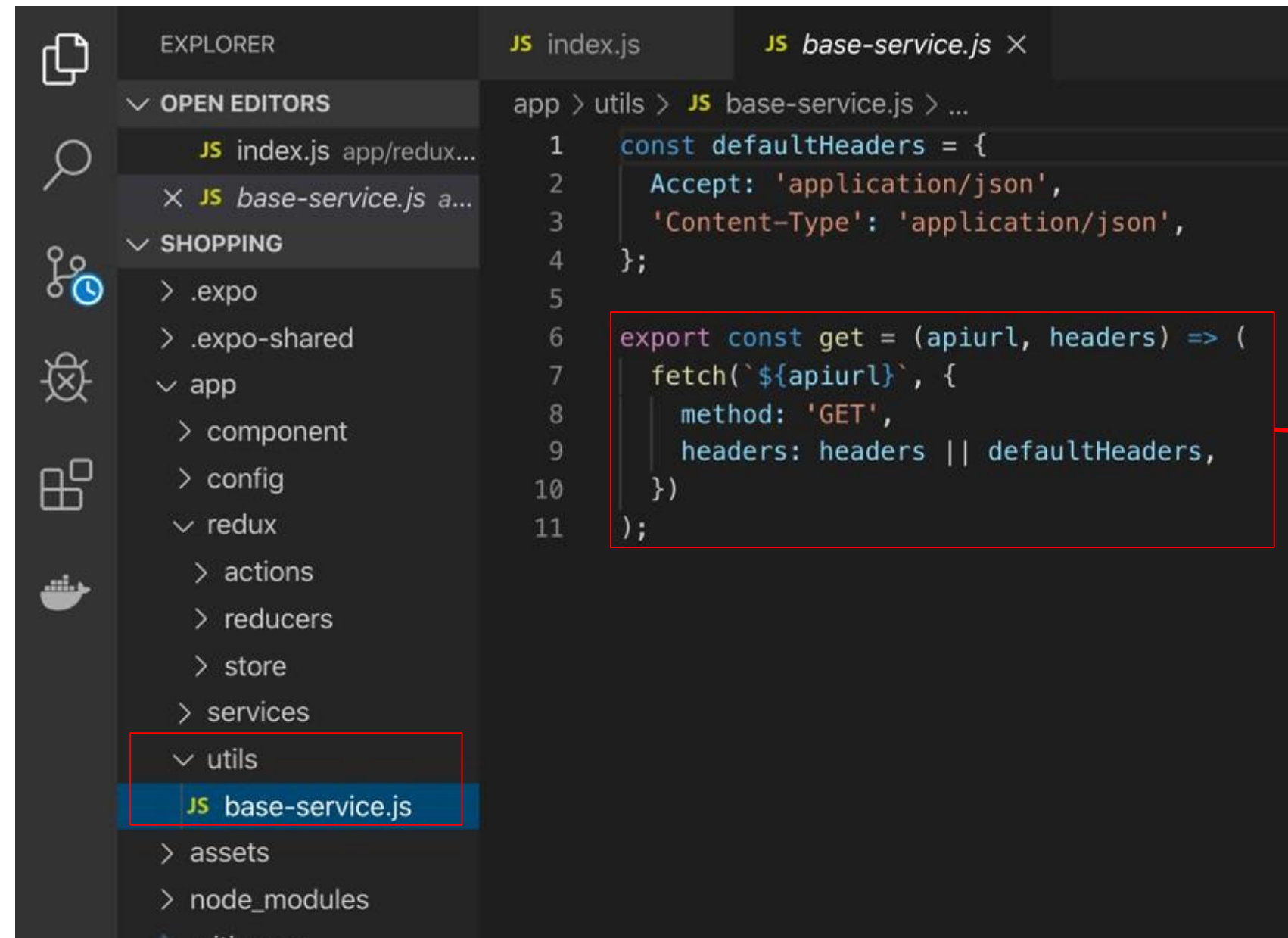
Demo: Add Redux Folder Structure

Create the Redux folder structure as shown below:



Demo: Fetch()

Create a folder *utils* and add *base-service.js* in it. In this file define *fetch()* method to make an API call.



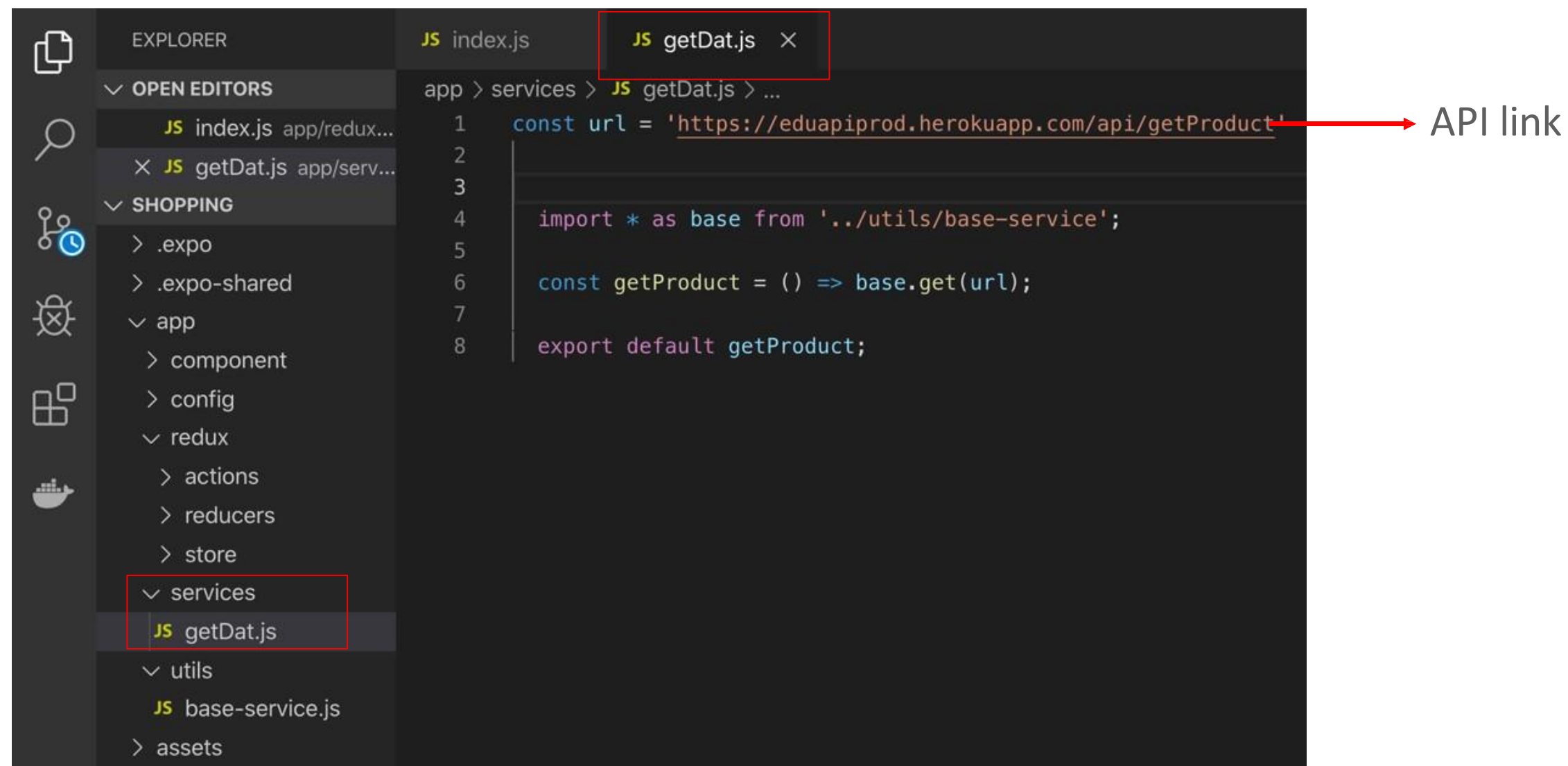
```
EXPLORER
  OPEN EDITORS
    JS index.js app/redux...
    JS base-service.js a...
  SHOPPING
    .expo
    .expo-shared
    app
      component
      config
      redux
        actions
        reducers
        store
        services
      utils
        JS base-service.js
      assets
      node_modules

JS index.js
JS base-service.js X
app > utils > JS base-service.js > ...
1  const defaultHeaders = {
2    Accept: 'application/json',
3    'Content-Type': 'application/json',
4  };
5
6  export const get = (apiurl, headers) => (
7    fetch(`${apiurl}`, {
8      method: 'GET',
9      headers: headers || defaultHeaders,
10    })
11  );
```

Function to make an API call

Demo: Define API Call

Create a services folder and add getDat.js file. In this file make a call to the API.

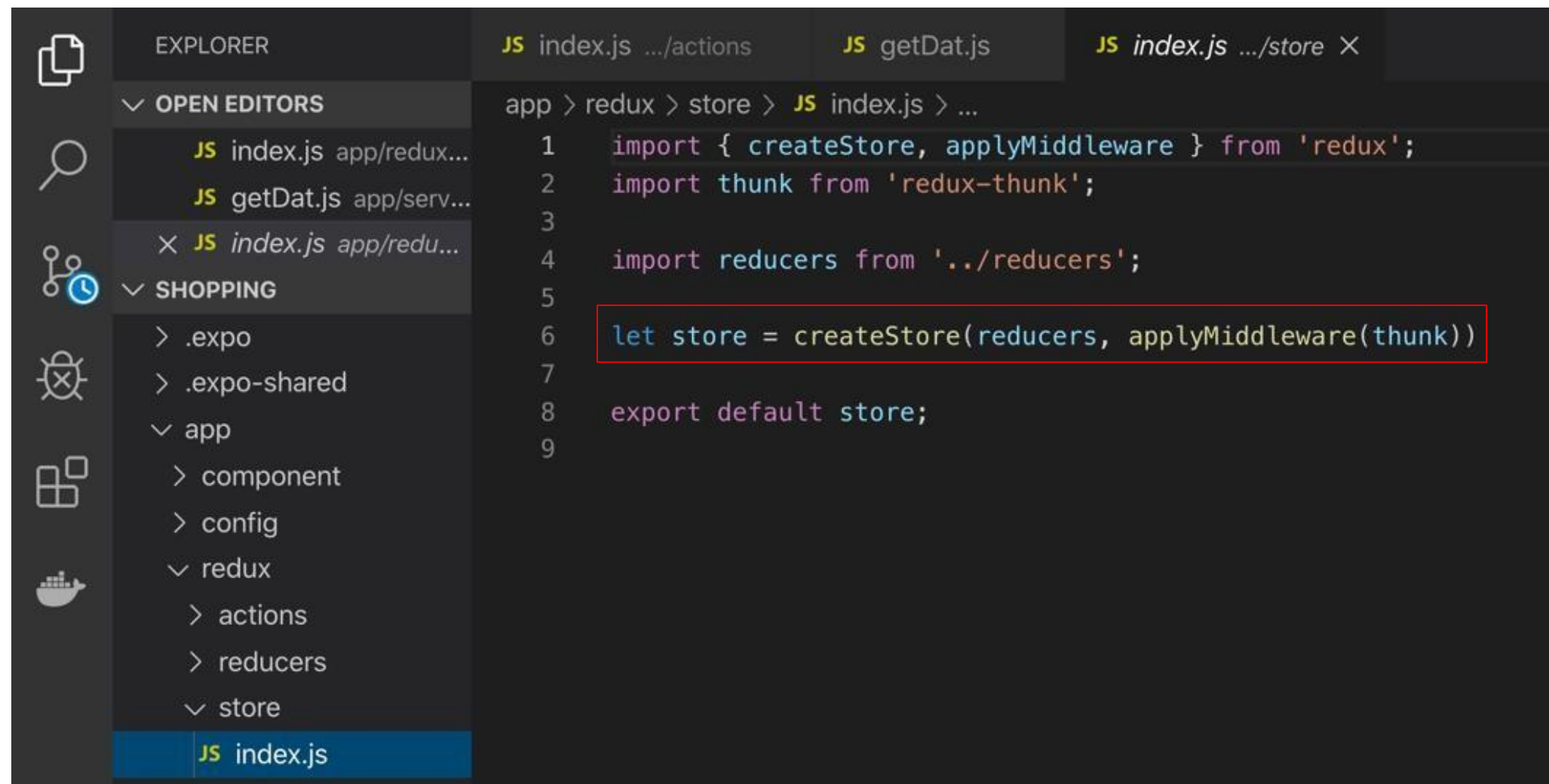


```
1 const url = 'https://eduapiprod.herokuapp.com/api/getProduct'
2
3
4 import * as base from '../utils/base-service';
5
6 const getProduct = () => base.get(url);
7
8 export default getProduct;
```

API link

Demo: Store

Create a index.js file in store, add the below snippet to the store section. Create the *store* and connect *Thunk* middleware along with *reducers* to it.



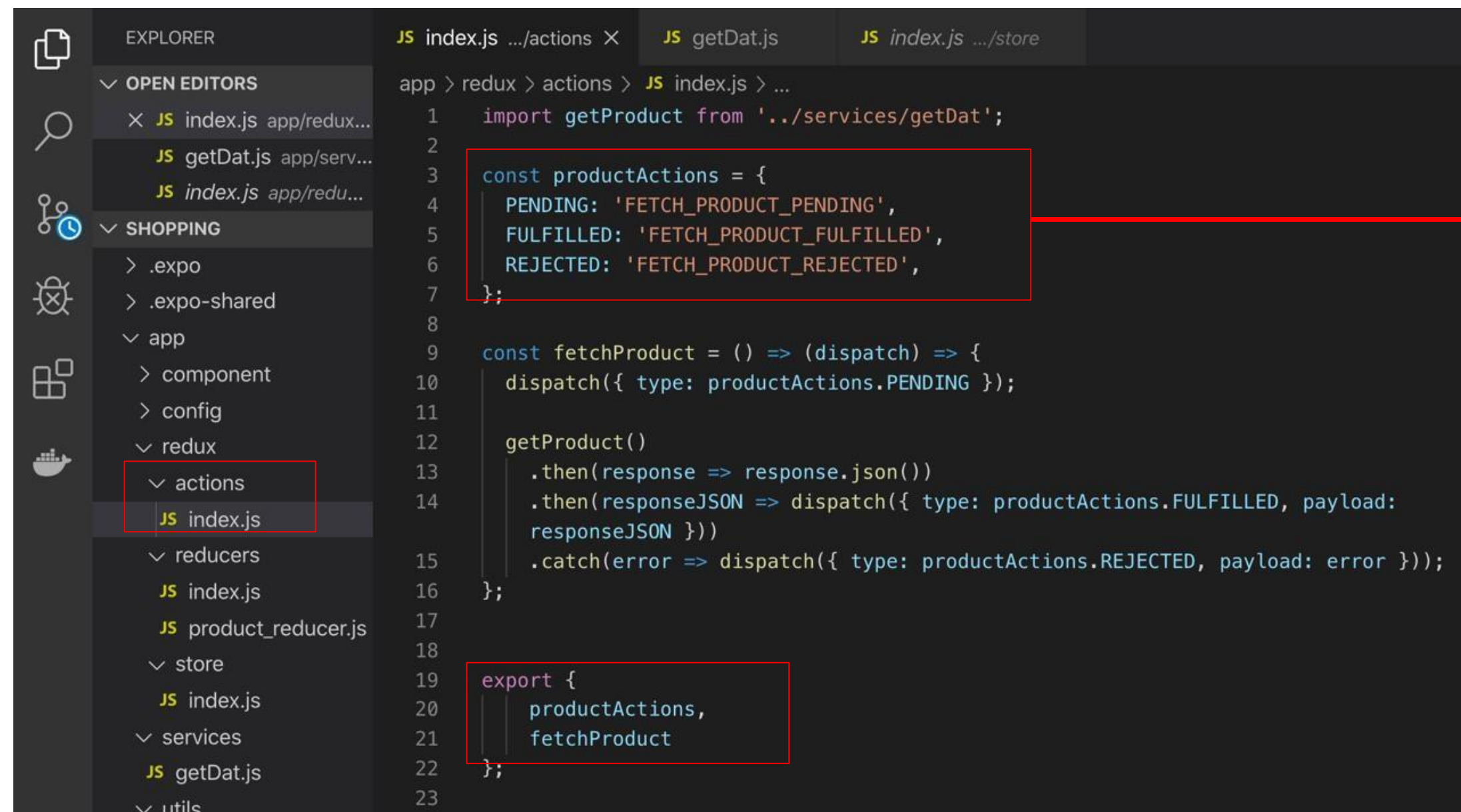
The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure: 'app' > 'redux' > 'store'. The 'store' folder is expanded, and 'index.js' is selected. The main editor area shows the content of 'index.js' with the following code:

```
1 import { createStore, applyMiddleware } from 'redux';
2 import thunk from 'redux-thunk';
3
4 import reducers from '../reducers';
5
6 let store = createStore(reducers, applyMiddleware(thunk))
7
8 export default store;
9
```

The line `let store = createStore(reducers, applyMiddleware(thunk))` is highlighted with a red rectangular box.

Demo: Actions

This time in actions we are not going to call the API. API calling is already defined in service folder. So from actions export only type and payload.



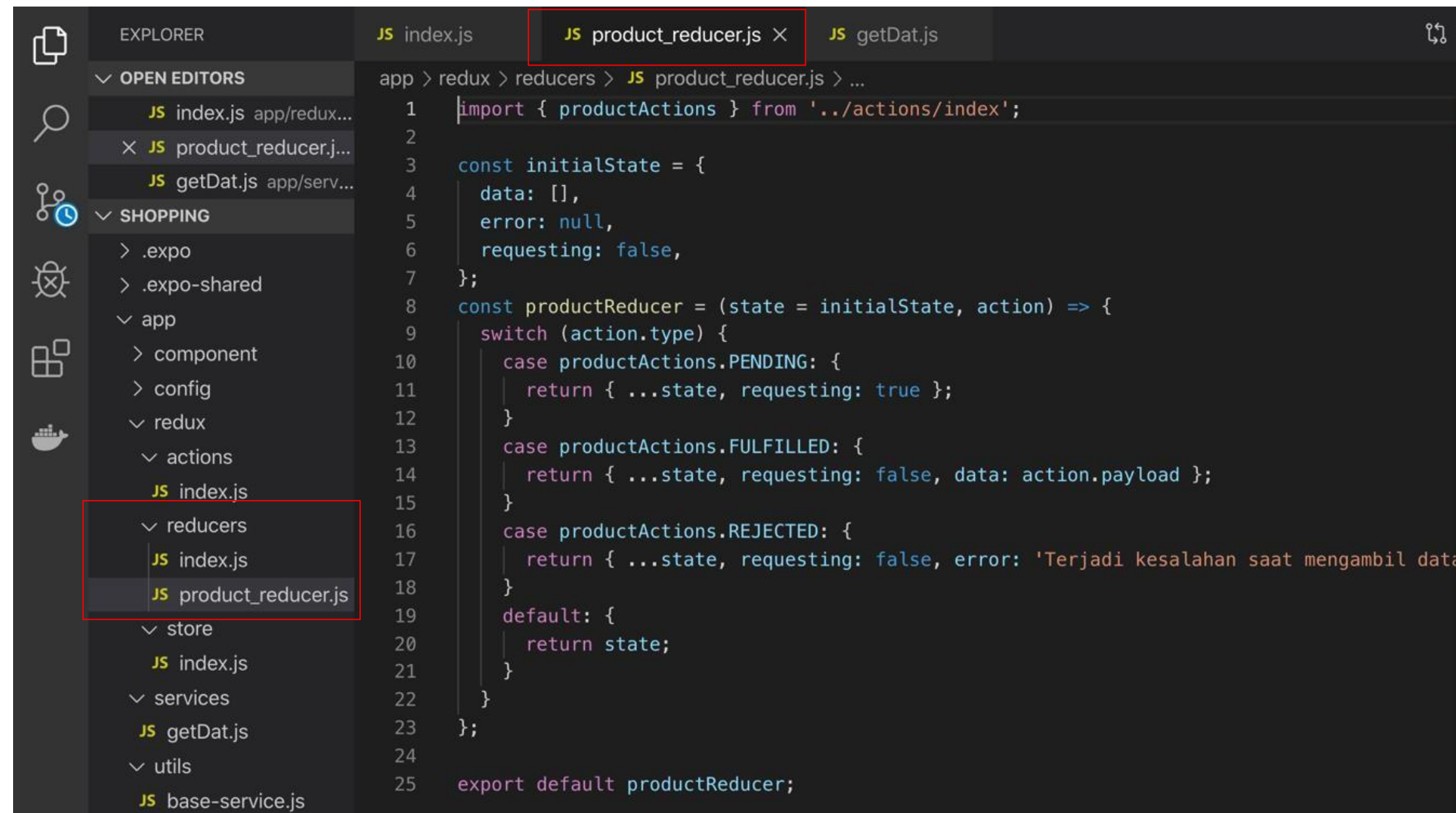
The screenshot shows a VS Code editor with the Explorer sidebar on the left and the Editor pane on the right. In the Explorer, the 'actions' folder is expanded, and 'index.js' is selected. The Editor pane shows the code for 'index.js' in the 'actions' directory. The code includes an import for 'getProduct' from '../services/getDat', a constant 'productActions' with three state types (PENDING, FULFILLED, REJECTED), a 'fetchProduct' function that dispatches these actions, and an export statement for 'productActions' and 'fetchProduct'. Two red boxes highlight the 'productActions' constant and the export statement. A red arrow points from the 'productActions' box to the text 'Action creators' on the right.

```
1 import getProduct from '../services/getDat';
2
3 const productActions = {
4   PENDING: 'FETCH_PRODUCT_PENDING',
5   FULFILLED: 'FETCH_PRODUCT_FULFILLED',
6   REJECTED: 'FETCH_PRODUCT_REJECTED',
7 };
8
9 const fetchProduct = () => (dispatch) => {
10   dispatch({ type: productActions.PENDING });
11
12   getProduct()
13     .then(response => response.json())
14     .then(responseJSON => dispatch({ type: productActions.FULFILLED, payload: responseJSON }))
15     .catch(error => dispatch({ type: productActions.REJECTED, payload: error }));
16 };
17
18
19 export {
20   productActions,
21   fetchProduct
22 };
23
```

Action creators

Demo: Reducers

Create the reducers folder structure as shown below. In product_reducers.js file add the below snippet.

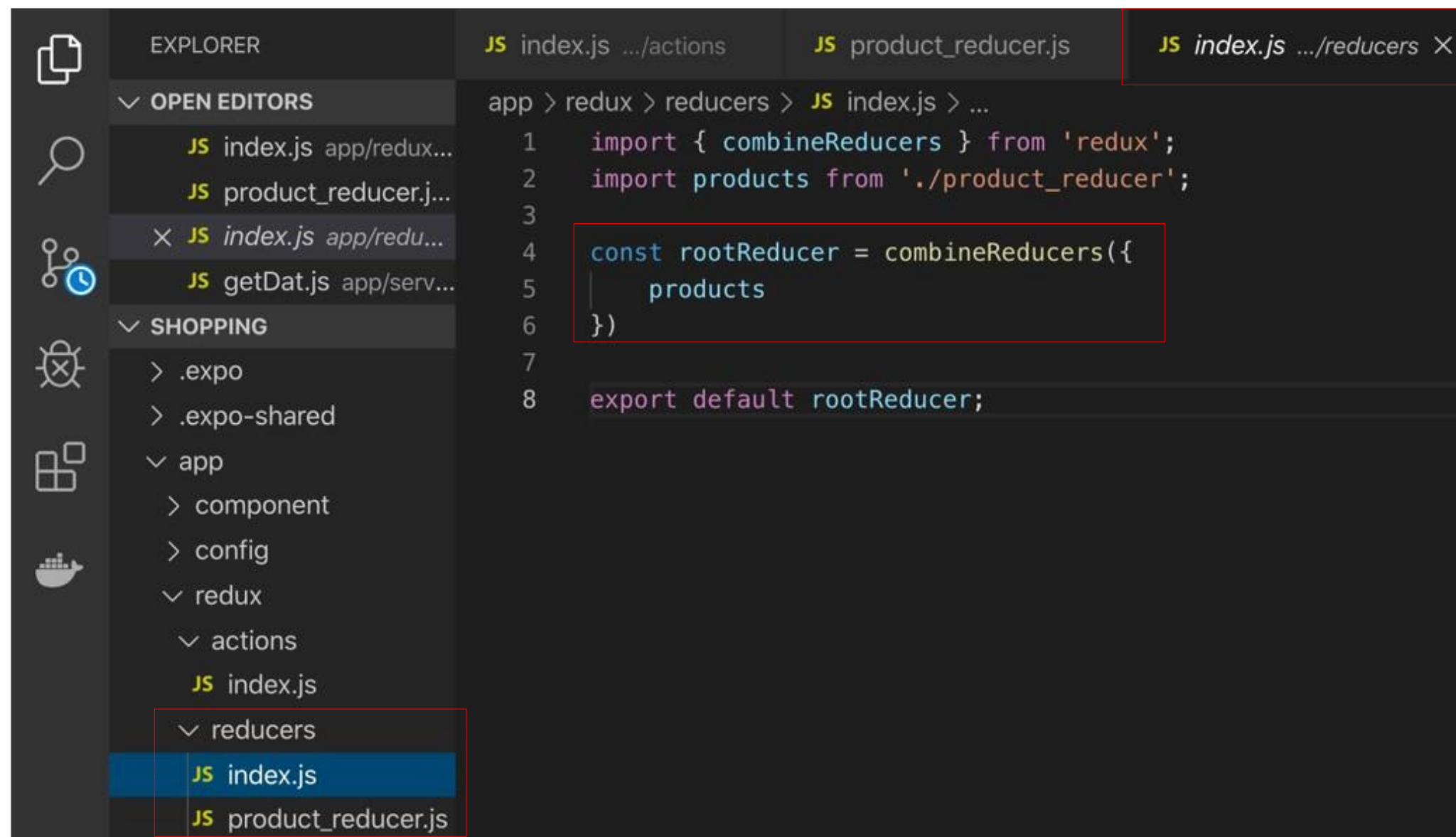


The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor view on the right. The Explorer sidebar shows the project structure, with the 'reducers' folder highlighted. The Editor view shows the 'product_reducer.js' file, which contains the following code:

```
1 import { productActions } from '../actions/index';
2
3 const initialState = {
4   data: [],
5   error: null,
6   requesting: false,
7 };
8 const productReducer = (state = initialState, action) => {
9   switch (action.type) {
10     case productActions.PENDING: {
11       return { ...state, requesting: true };
12     }
13     case productActions.FULFILLED: {
14       return { ...state, requesting: false, data: action.payload };
15     }
16     case productActions.REJECTED: {
17       return { ...state, requesting: false, error: 'Terjadi kesalahan saat mengambil data' };
18     }
19     default: {
20       return state;
21     }
22   }
23 };
24
25 export default productReducer;
```


Demo: Reducers (contd.)

Add all the created reducers to root reducer.

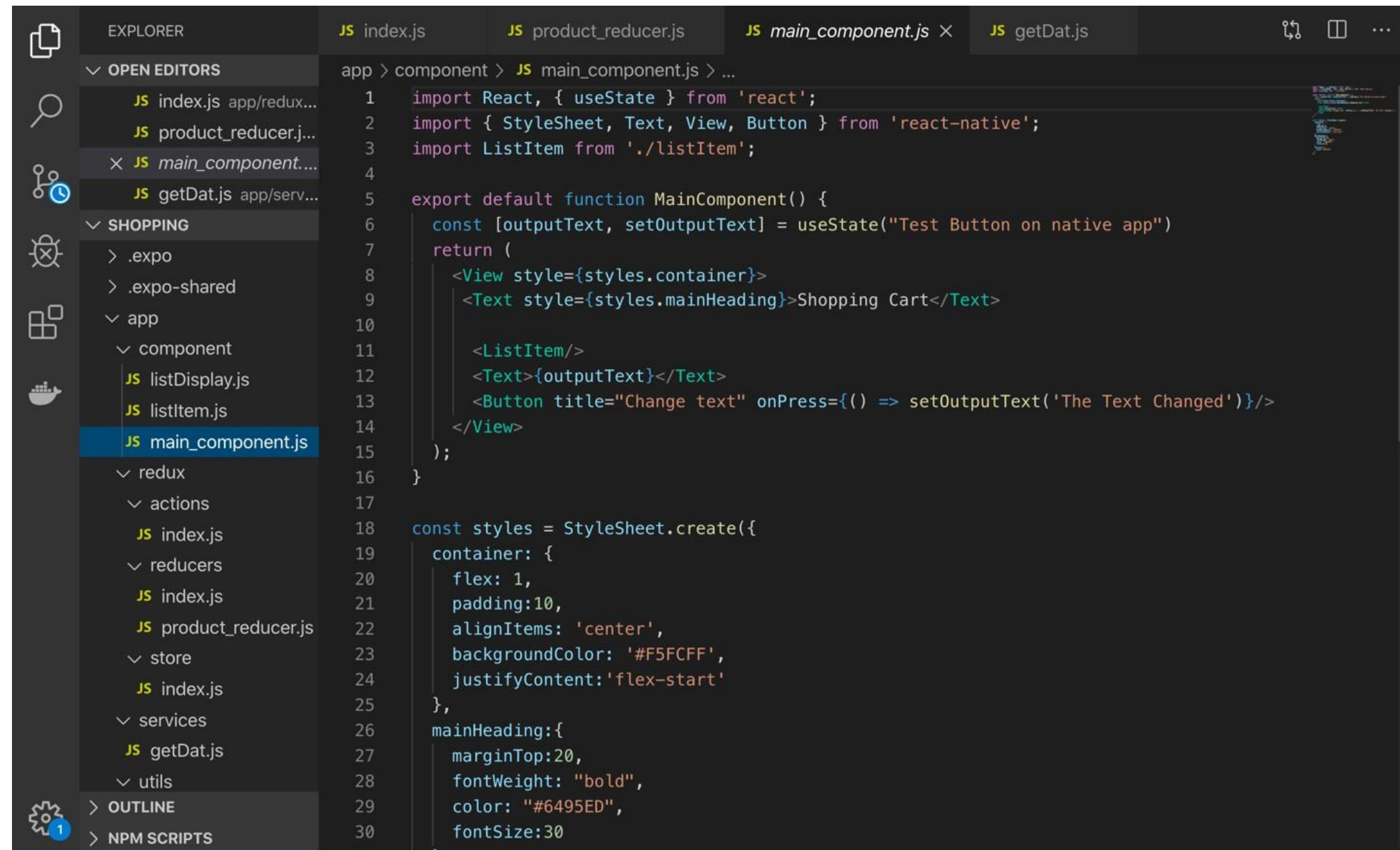


The screenshot shows the Visual Studio Code interface with the Explorer, Search, and Run and Debug views on the left. The Explorer view shows the project structure with the 'reducers' folder expanded, and 'index.js' and 'product_reducer.js' selected. The Search view shows the same files. The Run and Debug view shows the code for 'index.js' in the 'reducers' folder, which is highlighted with a red box. The code in the editor is as follows:

```
app > redux > reducers > JS index.js > ...
1  import { combineReducers } from 'redux';
2  import products from './product_reducer';
3
4  const rootReducer = combineReducers({
5    |   products
6  })
7
8  export default rootReducer;
```

Demo: main_component.js

In component folder create a component main_component.js to build main page of the application.

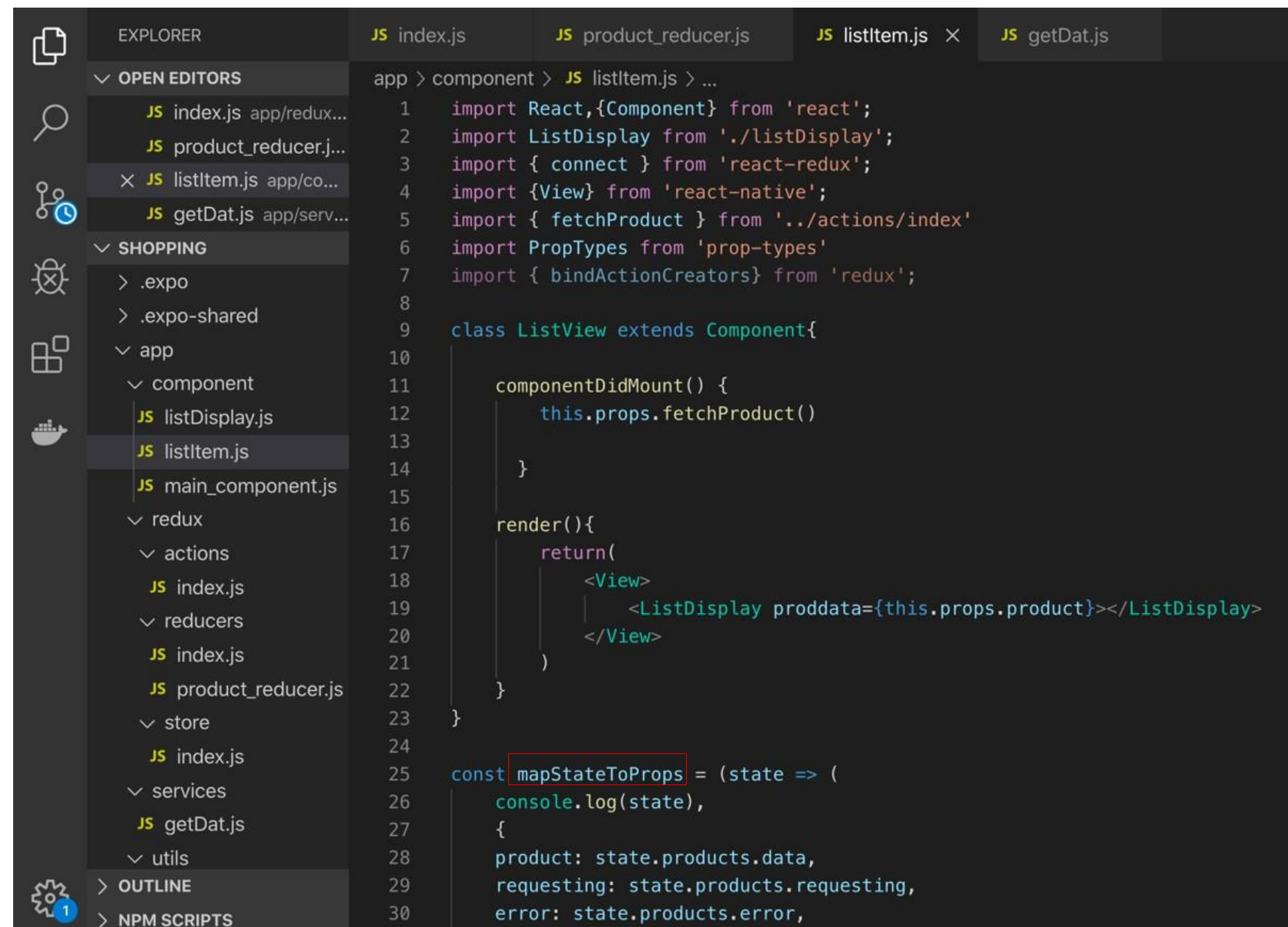


The screenshot shows the Visual Studio Code editor interface. On the left, the Explorer sidebar displays the project structure. The 'component' folder is expanded, showing files like listDisplay.js, listItem.js, and main_component.js, which is currently selected. The main editor area shows the code for main_component.js. The code imports React and components from 'react' and 'react-native', and a ListItem component from './listItem'. It defines a MainComponent function using useState to manage a text state. The UI includes a heading 'Shopping Cart', a list item, a text display, and a button to change the text. A StyleSheet is also defined for styling.

```
1 import React, { useState } from 'react';
2 import { StyleSheet, Text, View, Button } from 'react-native';
3 import ListItem from './listItem';
4
5 export default function MainComponent() {
6   const [outputText, setOutputText] = useState("Test Button on native app")
7   return (
8     <View style={styles.container}>
9       <Text style={styles.mainHeading}>Shopping Cart</Text>
10
11       <ListItem/>
12       <Text>{outputText}</Text>
13       <Button title="Change text" onPress={() => setOutputText('The Text Changed')}/>
14     </View>
15   );
16 }
17
18 const styles = StyleSheet.create({
19   container: {
20     flex: 1,
21     padding:10,
22     alignItems: 'center',
23     backgroundColor: '#F5FCFF',
24     justifyContent: 'flex-start'
25   },
26   mainHeading:{
27     marginTop:20,
28     fontWeight: "bold",
29     color: "#6495ED",
30     fontSize:30
```

Demo: ItemList.js

We need one list view component in which we will receive the data through action, call the prop to dispatch the action.
Receive the data using *MapDispatchToProps()* method.



```
app > component > JS listItem.js > ...
1  import React,{Component} from 'react';
2  import ListDisplay from './listDisplay';
3  import { connect } from 'react-redux';
4  import {View} from 'react-native';
5  import { fetchProduct } from '../actions/index'
6  import PropTypes from 'prop-types'
7  import { bindActionCreators} from 'redux';
8
9  class ListView extends Component{
10
11      componentDidMount() {
12          this.props.fetchProduct()
13      }
14
15
16      render(){
17          return(
18              <View>
19                  <ListDisplay proddata={this.props.product}></ListDisplay>
20              </View>
21          )
22      }
23  }
24
25  const mapStateToProps = (state => (
26      console.log(state),
27      {
28          product: state.products.data,
29          requesting: state.products.requesting,
30          error: state.products.error,
```

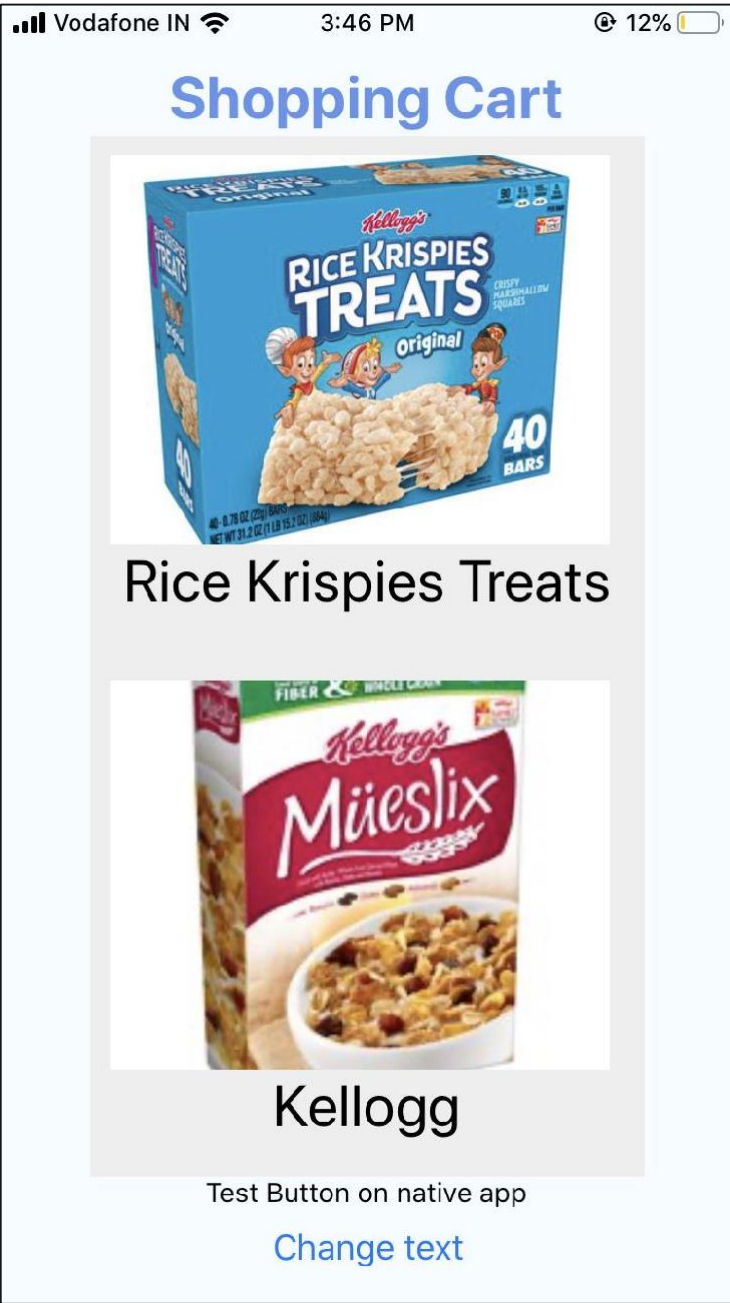

Demo: ItemList.js (contd.)

We need one list view component in which we will receive the data through action, call the prop to dispatch the action.
Receive the data using *MapDispatchToProps()* method.

```
23 }
24
25 const mapStateToProps = (state => (
26   console.log(state),
27   {
28     product: state.products.data,
29     requesting: state.products.requesting,
30     error: state.products.error,
31   }));
32
33 const mapDispatchToProps = {
34   fetchProduct
35 };
36
37
38 ListView.propTypes = {
39
40   fetchProduct: PropTypes.func.isRequired
41 };
42
43
44 export default connect(mapStateToProps, mapDispatchToProps)(ListView);
```


Demo: Output Of React Native and Redux Application

Check the output by starting the application using: *npm start*





Questions



FEEDBACK

