React With Redux Certification Training

COURSE OUTLINE
MODULE 04

Topics

Following are the topics covered in this module:

- Need of Redux
- What is Redux?
- Redux Architecture
- Redux Action
- Redux Reducers
- Redux Store
- Principles of Redux
- Pros of Redux
- NPM libraries required to work with Redux
- More about react-redux package
- Building a food list application using React and Redux
- Building News application using React and Redux where data is received by an API

Objectives

After completion of this module you should be able to:

- Analyse why should we use redux with react
- Explain redux architecture
- Deploy actions
- Implement reducer functions
- Integrate store in your application
- Understand principles of redux
- Install redux and configure required NPM libararies
- Build a news application using react and redux methodologies

Why Do We Need Redux?

Problems With React

React applications are made up of ***nested components*** and each component has its own ***state***

User actions (such as click of buttons), leads to ***transition*** of state from the old state to a new state

As ***an application grows***, it becomes ***hard to determine*** the ***overall state of the application*** and cumbersome to track the upcoming ***updates***

Data flows unidirectionally only from ***a parent component to child component***, so sharing data among ***sibling components*** becomes ***difficult***

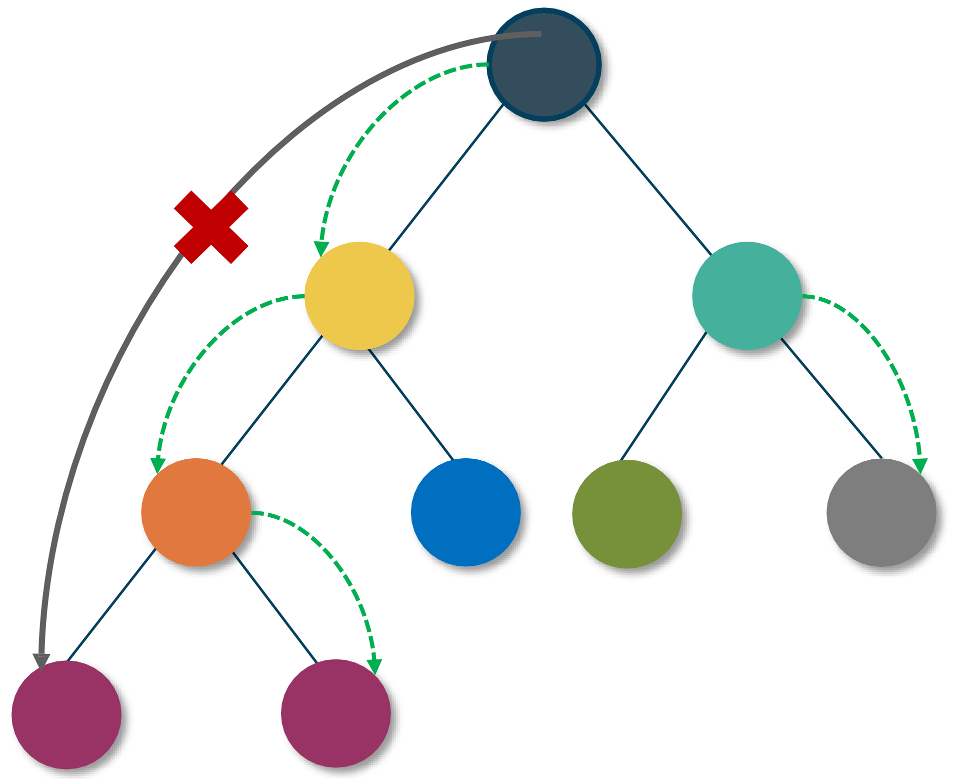


Fig: Unidirectional flow of data from parent to child component

Problems Solved Using Redux

- With Redux, all the component states of an application can be collected in one place called ***store***
- Components will ***dispatch state changes*** to the store
- On the other hand, components that need to be aware of state changes can ***subscribe*** to the store
- In this way, ***Redux*** supports the ***Direct communication between*** the components

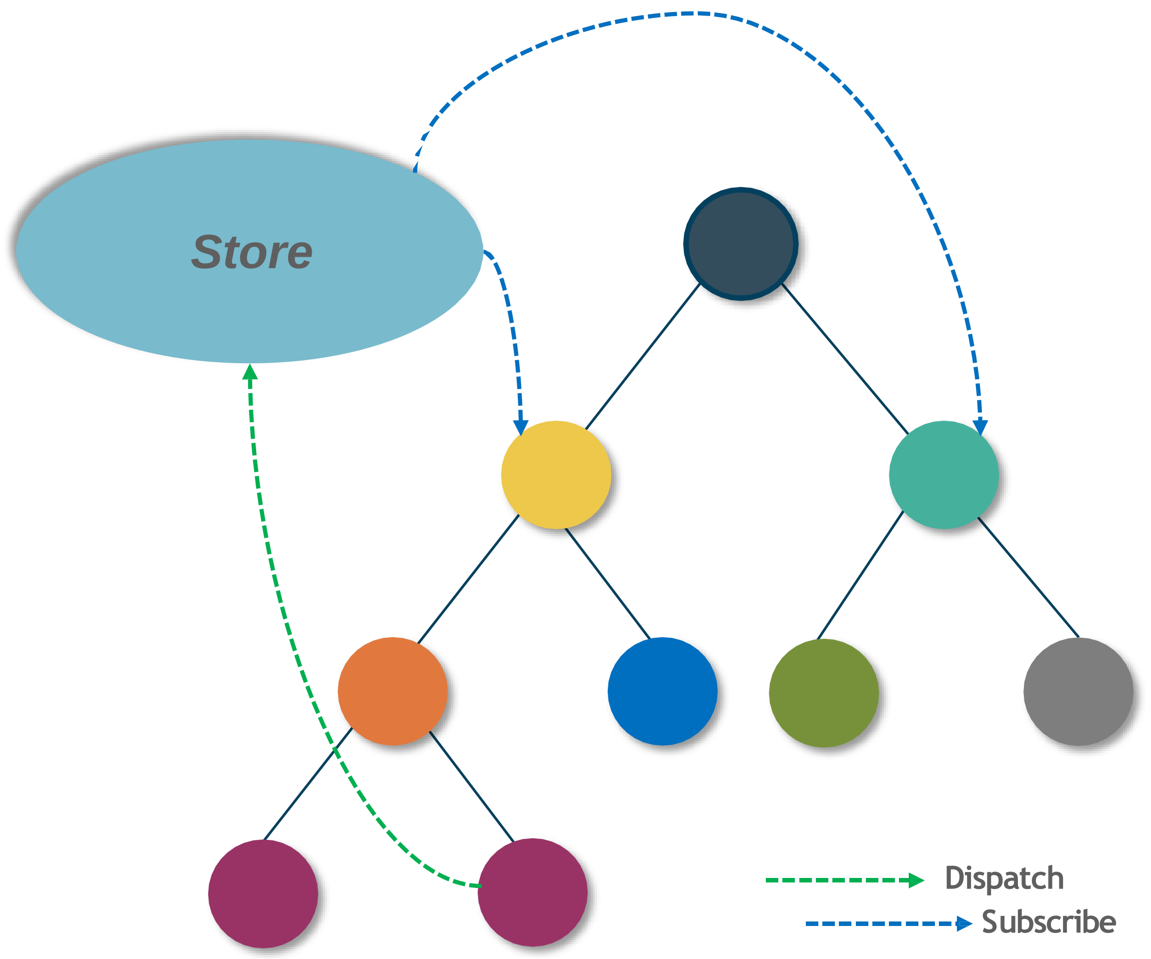


Fig: Unidirectional flow of data between components via Store

Redux

What Is Redux?

Redux is a predictable state container for JavaScript apps.

Redux has 4 parts: ***Action, Reducer, Store and View***

Action describes the ***changes in state*** of an application

Reducer carry out the ***state transition*** depending upon the actions

Store holds ***state*** of an application

View represents state changes and application in the browser

It is commonly used as ***state management library***, as it makes ***state mutations*** predictable by imposing certain ***restriction*** on how and when updates should happen

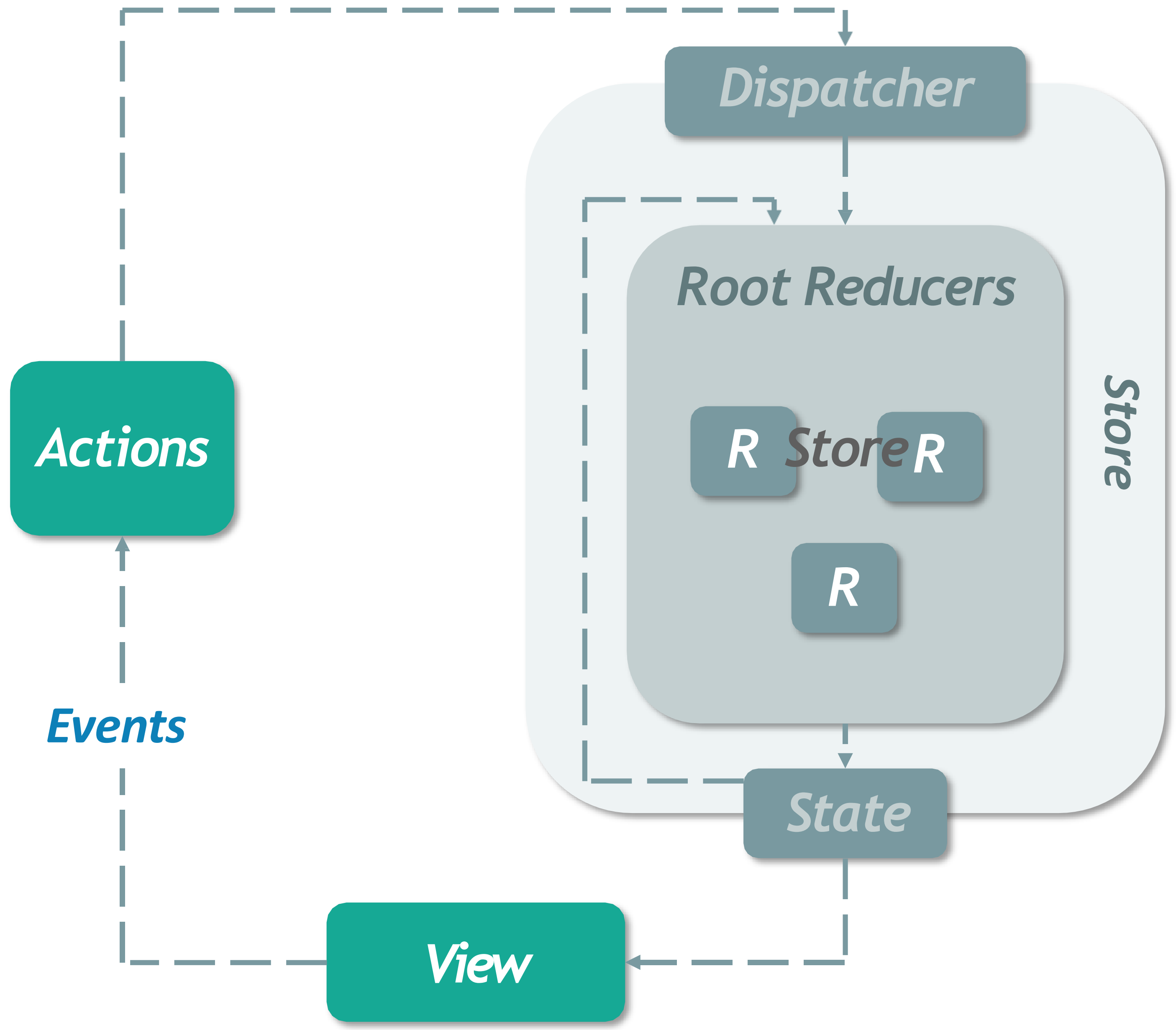
Redux

Redux Architecture

Redux Architecture: View

Redux features a ***unidirectional data flow*** and enforces a single store, where the state of the whole application is maintained.

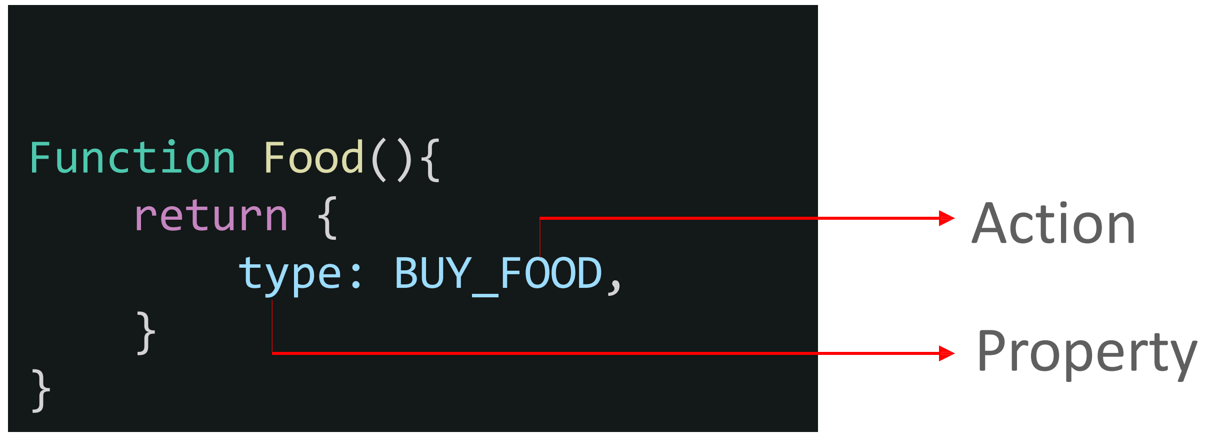
- ***View*** represents ***user interface*** of an application
- When user ***clicks certain button/option*** in an application, ***events*** are generated
- These events carry the ***information(user activities)*** to be performed
- These events further lead to ***actions***



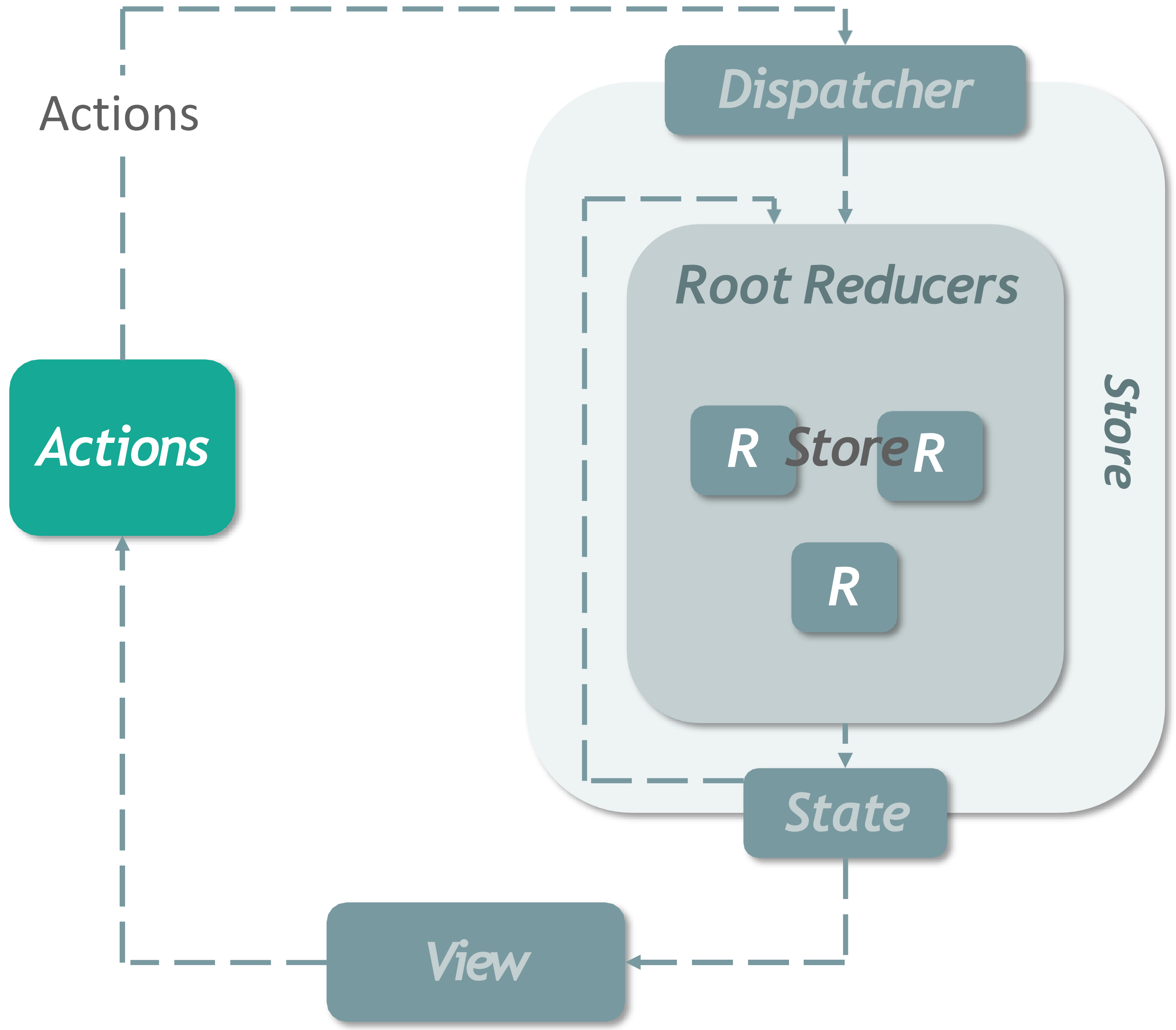
Redux Architecture: Actions

- ***Actions*** are the plain JavaScript objects
- They have a '***type***' property that indicates the action to be performed
- The 'type' property is defined as ***string constants***
- These actions get ***dispatched*** to ***reducer***

Syntax: Action



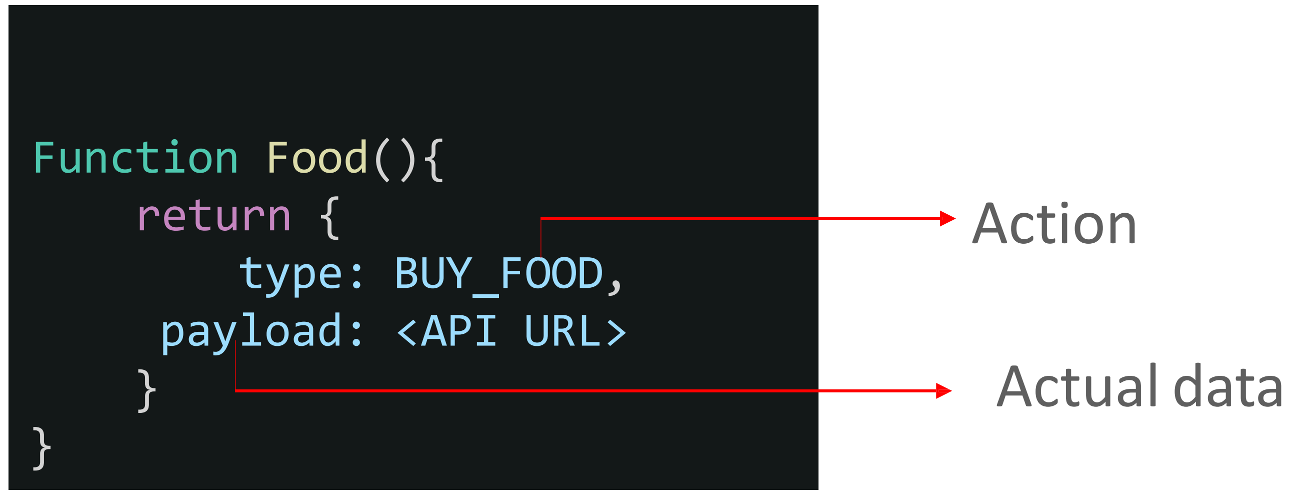
```
Function Food(){
    return {
        type: BUY_FOOD,
    }
}
```



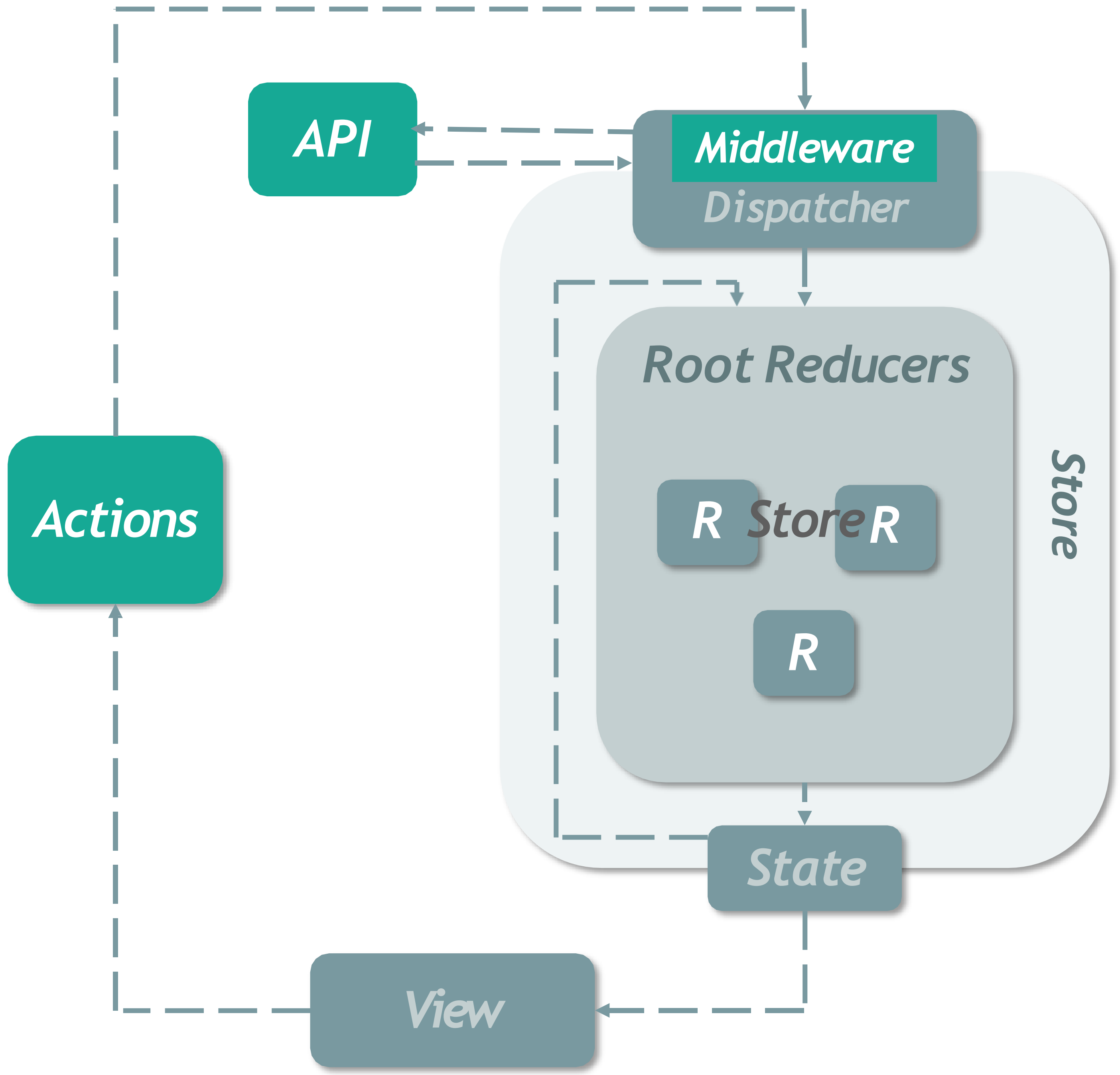
Redux Architecture: Actions (contd.)

- In case of loading data from an ***external API***, ***middleware*** process an Action
- ***Middleware*** is a code, placed between the framework receiving a request and the framework generating a response
- ***Middleware communicates*** with the ***API***, ***collects data*** from API and later ***dispatches*** it to ***reducer***

Syntax: Action

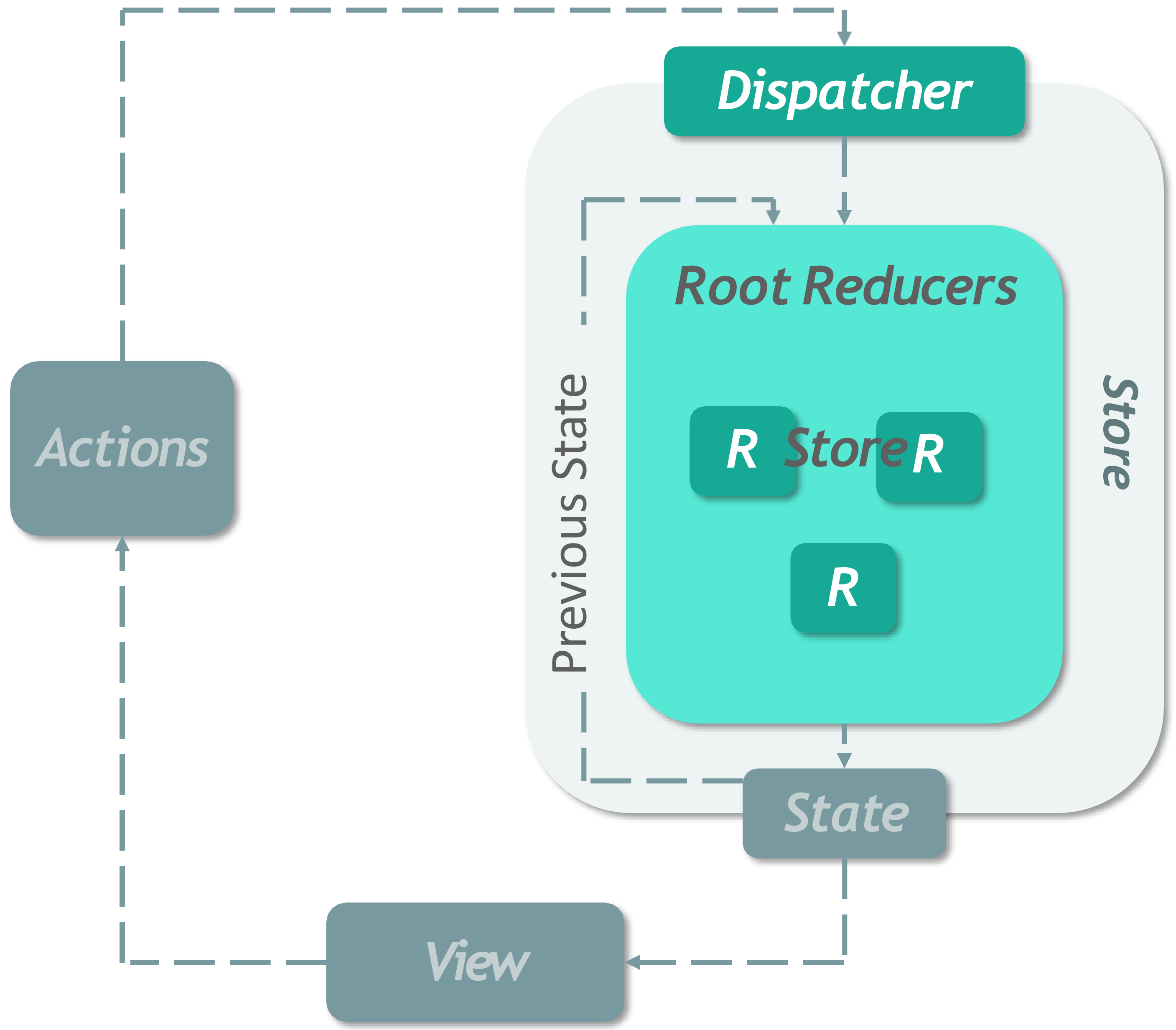


```
Function Food(){
    return {
        type: BUY_FOOD,
       payload: <API URL>
    }
}
```



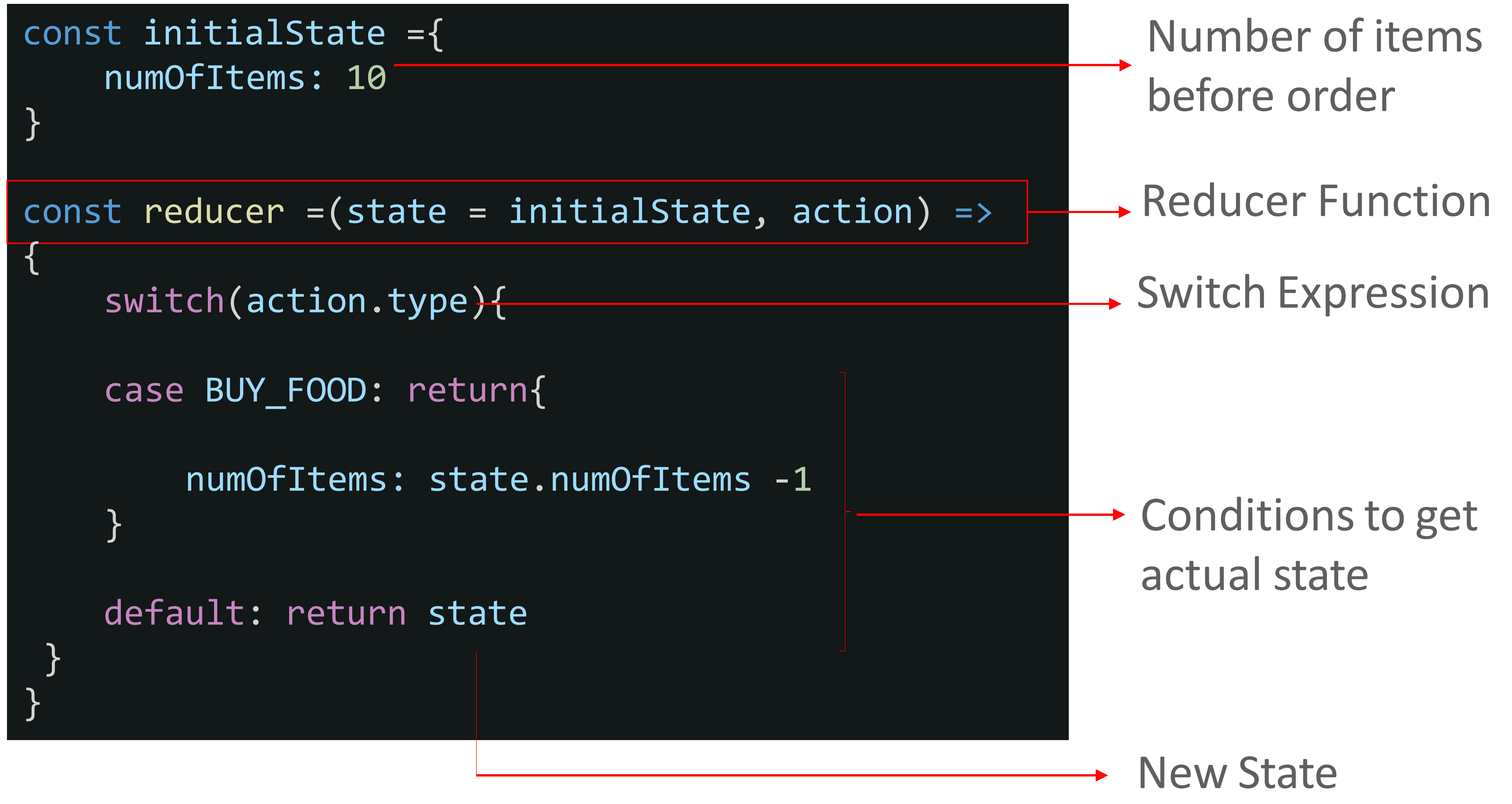
Redux Architecture: Reducers

- ***Reducer*** is a function that takes an Action and the current application State and it returns a new State of application ***({previousState, action => newState })***
- It specifies how the application ***state changes*** in response to the action
- The ***root reducer function*** is then called with the ***current state*** and a ***dispatched action***
- That ***root reducer*** may ***delegate*** the work to other ***smaller reducer*** functions, it then returns a ***new state***

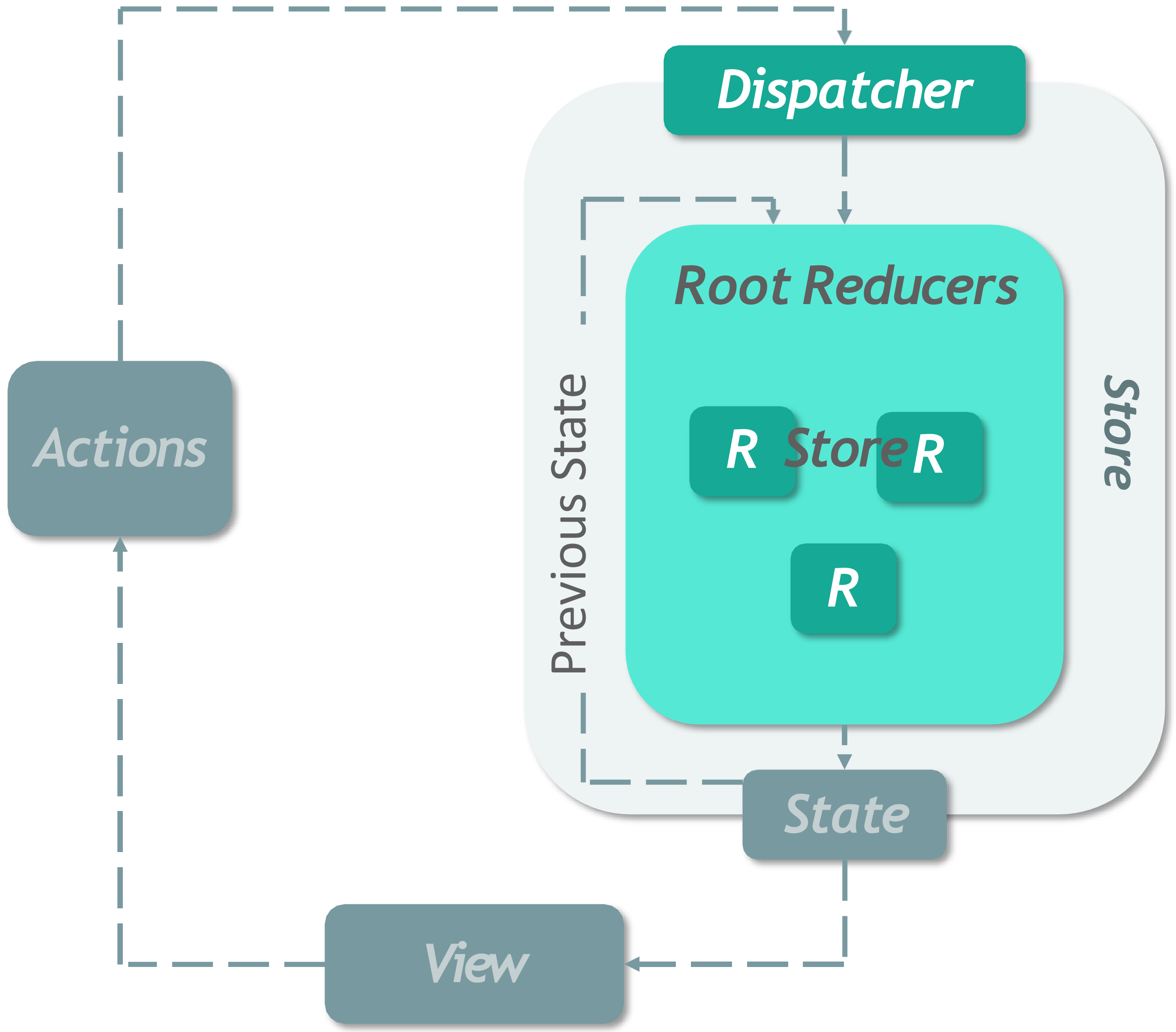


Redux Architecture: Reducers Syntax

Syntax: Reducers

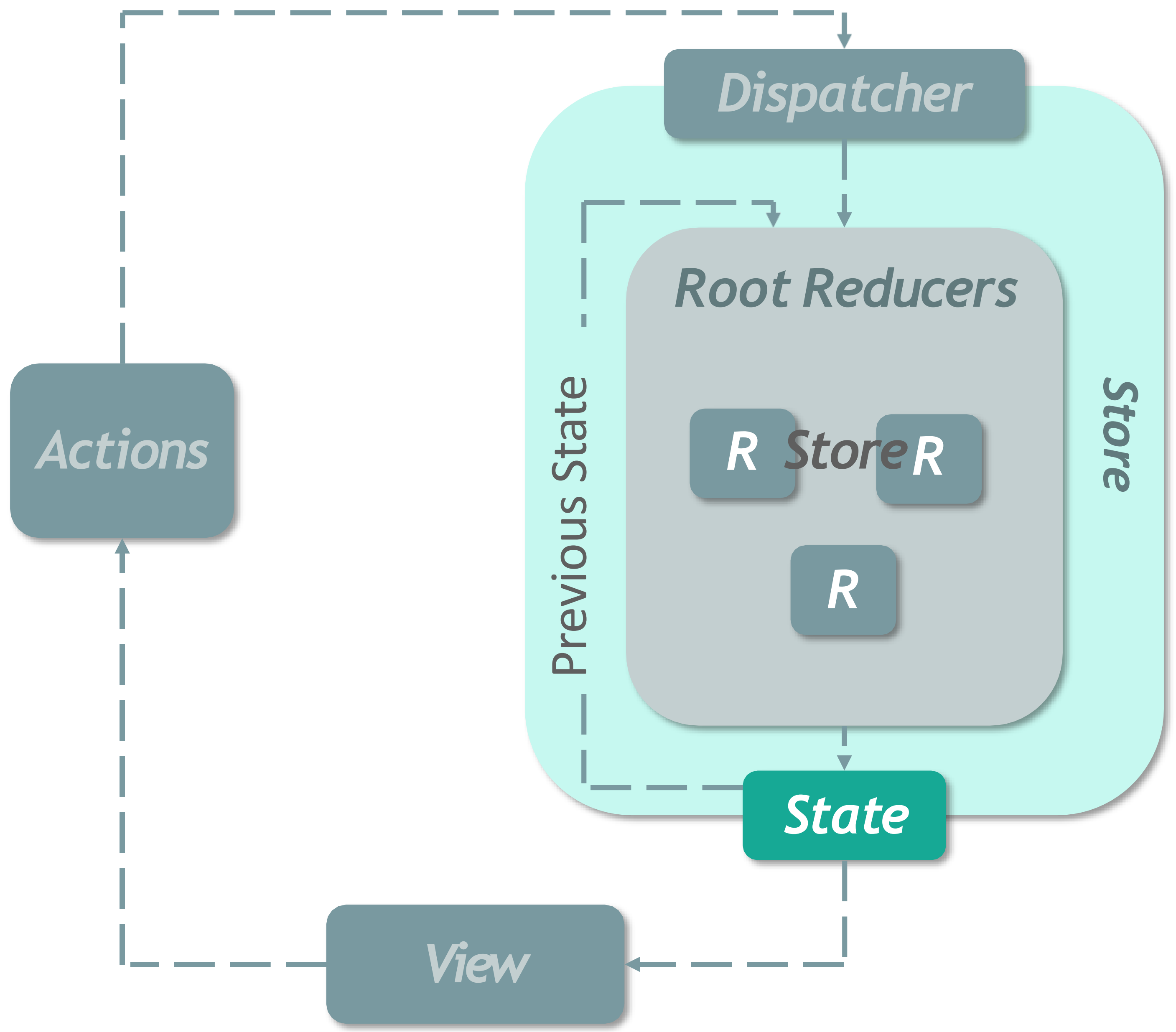


```
const initialState ={
    numOfItems: 10
}

const reducer =(state = initialState, action) =>
{
    switch(action.type){

    case BUY_FOOD: return{

        numOfItems: state.numOfItems -1
    }

    default: return state
 }
}
```



Redux Architecture: Store

- The ***Store*** in Redux is the object that brings *actions* and *reducers* together
- The store ***holds*** the application *state data* and ***handles*** all state updates
- The store handles state updates by passing the ***current state*** and an ***action*** through a ***reducer***
- The store has a ***dispatch method*** that takes *actions* as an argument. When an action is dispatched through the store, the action is sent through the Reducers and the state is updated



Redux Architecture: Store Methods

A ***store*** has three important methods as given below:

- getState()
- dispatch()
- subscribe()

It helps you ***retrieve*** the ***current State*** of your Redux Store.

The syntax for getState is as follows : ***store.getState()***

Redux Architecture: Store Methods (contd.)

A ***store*** has three important methods as given below:

getState()

dispatch()

subscribe()

It allows you to ***dispatch*** an action to ***change*** the state in your application.

The syntax for dispatch is as follows:
store.dispatch({type:'ITEMS_REQUEST'})

Redux Architecture: Store Methods (contd.)

A ***store*** has three important methods as given below:

getState() dispatch() subscribe()

It helps you register a callback that Redux Store will call when an Action has been dispatched. As soon as the Redux state has been ***updated***, the ***View*** will ***re-render*** automatically.

The syntax for subscribe is as follows: ***store.subscribe(()=>{ console.log(store.getState());})***

Note: subscribe() function always ***returns*** a function for ***unsubscribing*** the listener. To unsubscribe the listener, we can use the below syntax:

const unsubscribe = store.unsubscribe(()=>{console.log(store.getState());});

unsubscribe();

Redux Architecture: Store Methods (contd.)

```
const store = createStore(reducer)

console.log('Initialstate', store.getState())

const unsubscribe =store.subscribe(()=> console.log ('updated state',
 store.getState()))

store.dispatch (Food())
store.dispatch (Food())
store.dispatch (Food())

unsubscribe()
```

Reducer contains the initial state

Displays the current state using getState()

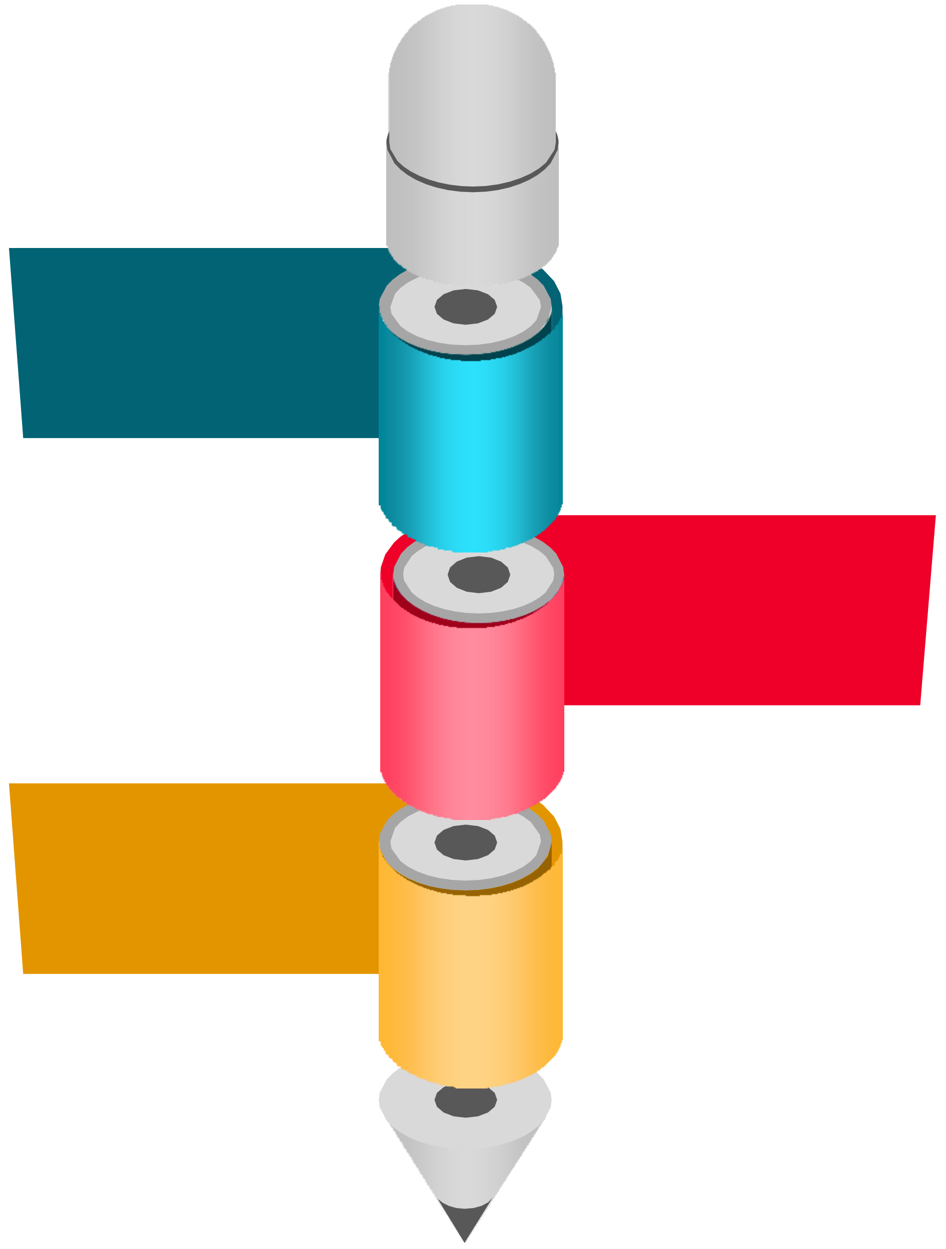
Allows the app to access the state from store using subscribe()

Dispatches the action

Principles Of Redux

Principles Of Redux

Redux follows three fundamental principles:



Single Source Of Truth

State is read-only

Changes are made with pure functions

Principles Of Redux: Single Source Of Truth

Single Source Of Truth

State is read-only

Changes are made with pure functions

The state of your whole application is stored in an object tree within a single store.

Explanation of Principle

Redux uses ***store*** for storing all the application state at one place. Components state is stored in the store and they receive updates from the store itself.

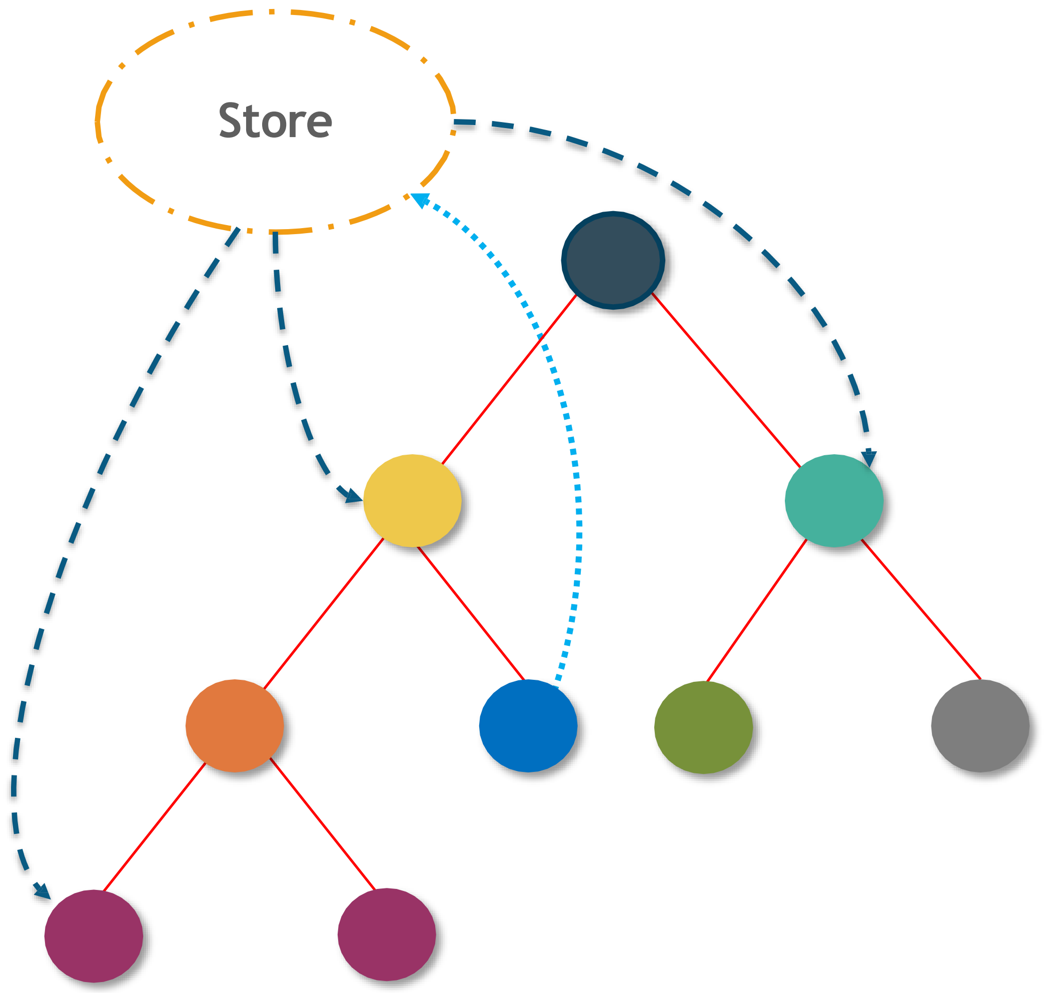


Fig: Flow of data within the sibling components via store

Principles Of Redux: State Is Read-only

The only way to ***change*** the state is to ***emit*** an ***action***, an object describing what happened.

Single Source Of Truth

State is read-only

Changes are made with pure functions

Explanation of Principle

You can change the state only by ***triggering*** an ***action,*** which is an object describing what happened.

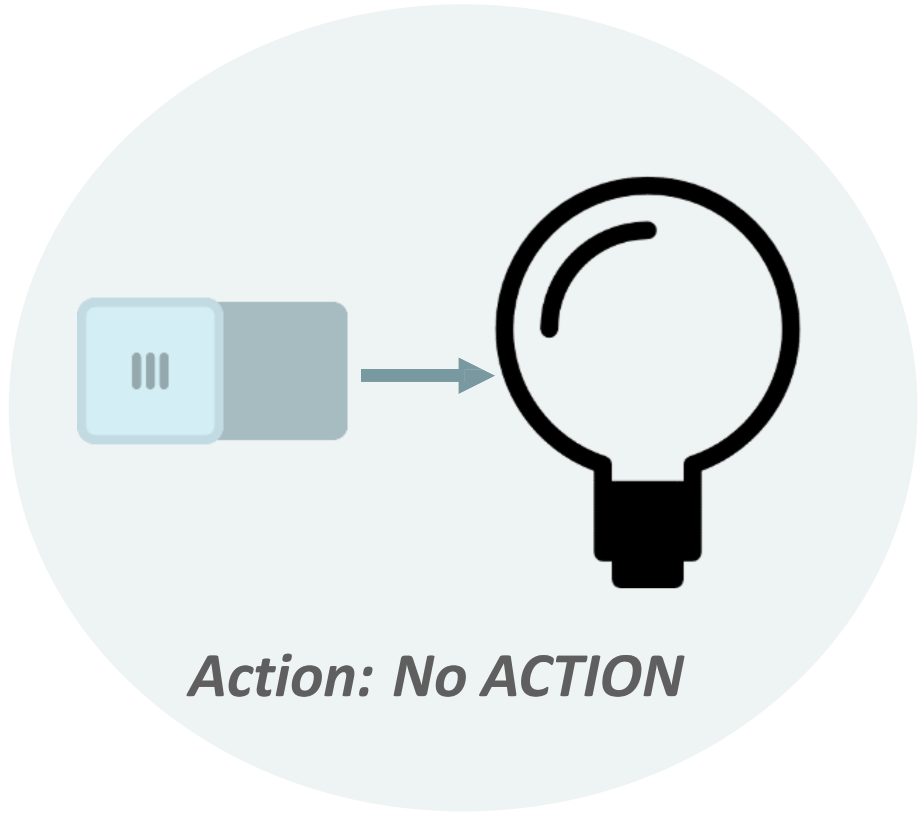


Fig: Previous State

Principles Of Redux: State Is Read-only (contd.)

The only way to ***change*** the state is to ***emit*** an ***action***, an object describing what happened.

Single Source Of Truth

State is read-only

Changes are made with pure functions

Explanation of Principle

You can change the state only by ***triggering*** an ***action,*** which is an object describing what happened

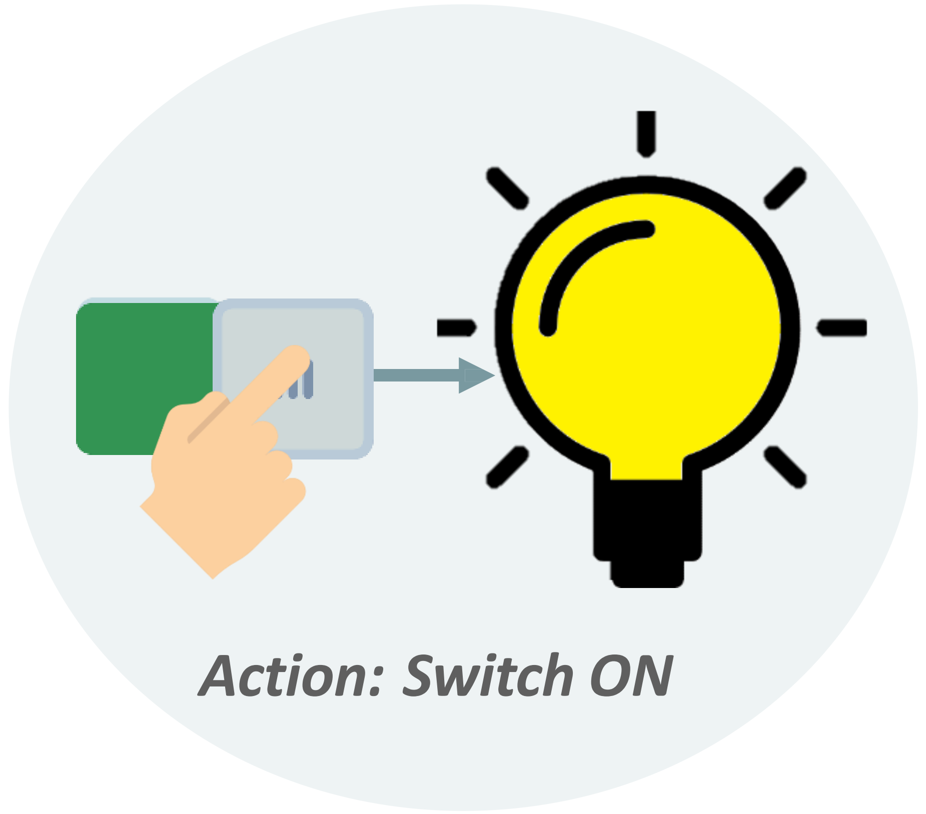


Fig: Current State due to imposed Action

Principles Of Redux: Changes Using Pure Functions

To specify how the state tree is ***transformed*** by ***actions***, you write pure reducers.

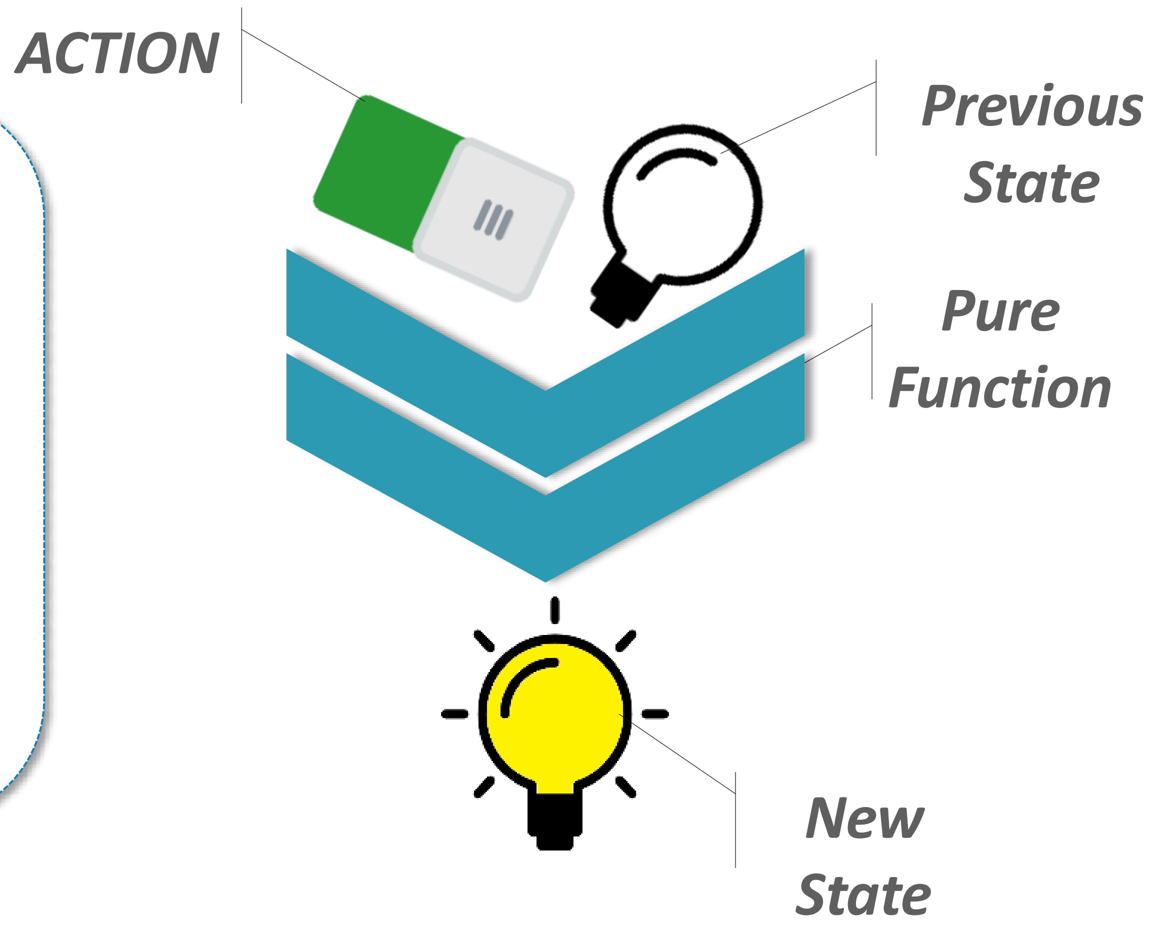
Single Source Of Truth

State is read-only

Changes are made with pure functions

Explanation of Principle

Pure functions called as ***Reducers*** are used to indicate, how the *state* has been transformed by the *action*.



Advantages Of Redux

Advantages Of Redux

Feasible data sharing between components: Due to central store, any component can access any state from the store, there is no need of passing props back and forth

Maintainability: Redux is strict about how code should be organized, due to this anyone with knowledge of Redux finds easy to understand the structure of any Redux application

Ease of testing: Testing will be easy as UI and data management are separated

Easy debugging: By logging actions and state, it is easy to understand coding errors, network errors and other errors that might occur in production environment

Faster access to components: History of state is maintained, this helps in implementing features like undo very easily

NPM

NPM

NPM packages required to work with Redux are::

react

It is a ***user interface*** library.

Installation: ***npm i react***

redux

It is a ***state management*** library

Installation: ***npm i redux***

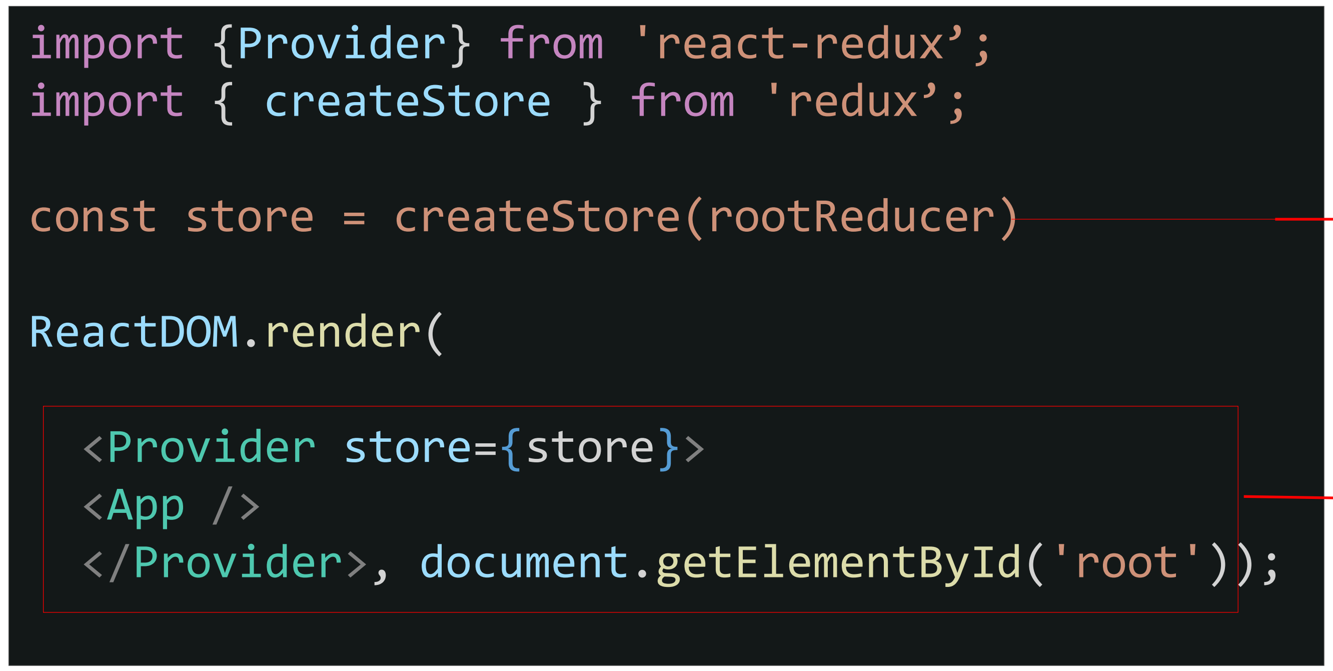
react- redux

- It is used to bind the two libraries
- It lets ***React application components*** to read data from a ***Redux store*** and ***dispatch actions*** to the store to update data
- Installation: ***npm i react-redux***

More About react-redux Library

The commonly used features of this library are:

<Provider/>: ***react-redux*** provides ***Provider*** to allow components of the application to take data from the store



```
import {Provider} from 'react-redux';
import { createStore } from 'redux';

const store = createStore(rootReducer)

ReactDOM.render(

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

Defines Store

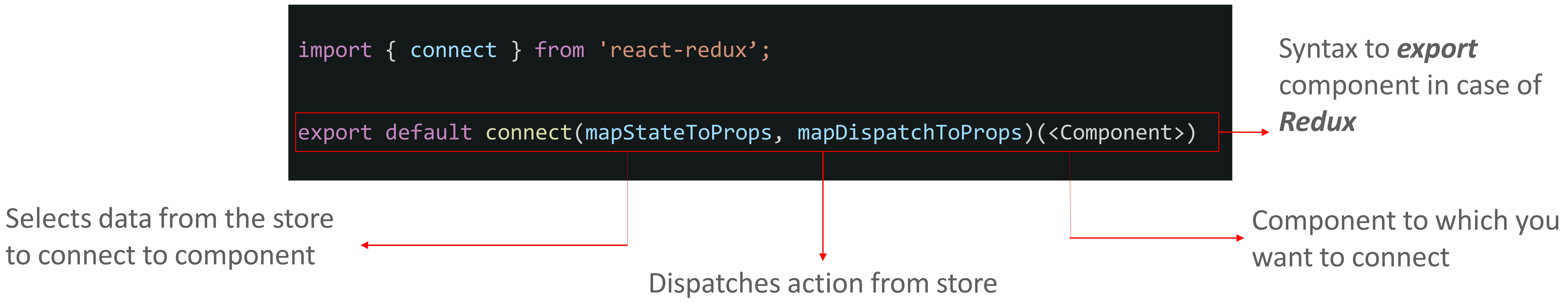
Syntax of defining Provider

Since react components are linked to each other, most applications will render a <**Provider/**> at the top level, with the entire ***App*** component inside it.

More About react-redux Library (contd.)

The commonly used features of this library are:

connect(): ***react-redux*** provides a connect function to connect your component to the store.



```
import { connect } from 'react-redux’;

export default connect(mapStateToProps, mapDispatchToProps)(<Component>)
```

Syntax to ***export*** component in case of ***Redux***
Selects data from the store to connect to component
Dispatches action from store
Component to which you want to connect

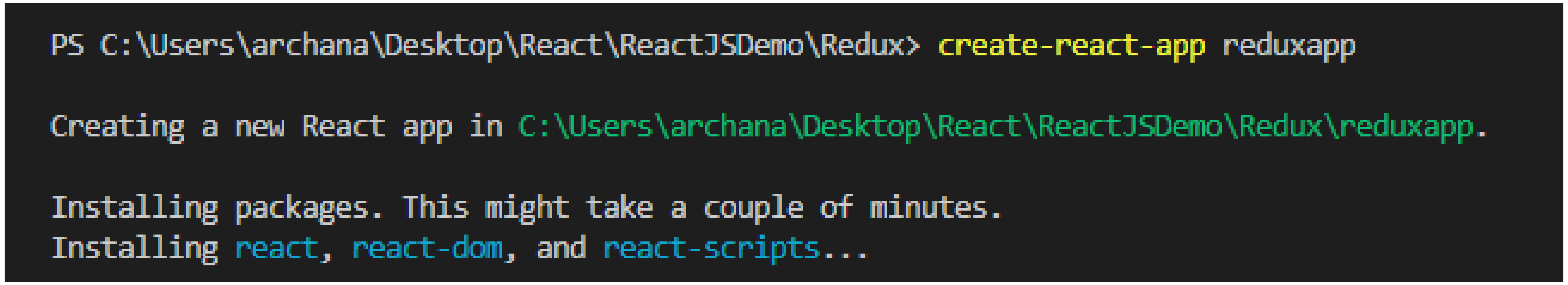
The <**Provider** /> makes the **Redux** store available to any nested components that have been wrapped in the connect() function.

Now we should create a React-Redux application using all the concepts discussed till now.

Demo 1: React Application Using Redux

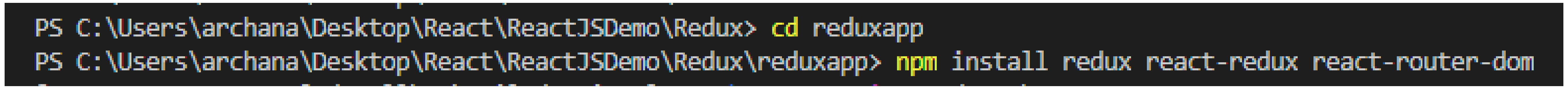
Demo: Installation Of Required Packages

Start building the application using the command: ***create-react-app <application_name>***



```
PS C:\Users\archana\Desktop\React\ReactJSDemo\Redux> create-react-app reduxapp

Creating a new React app in C:\Users\archana\Desktop\React\ReactJSDemo\Redux\reduxapp.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...
```

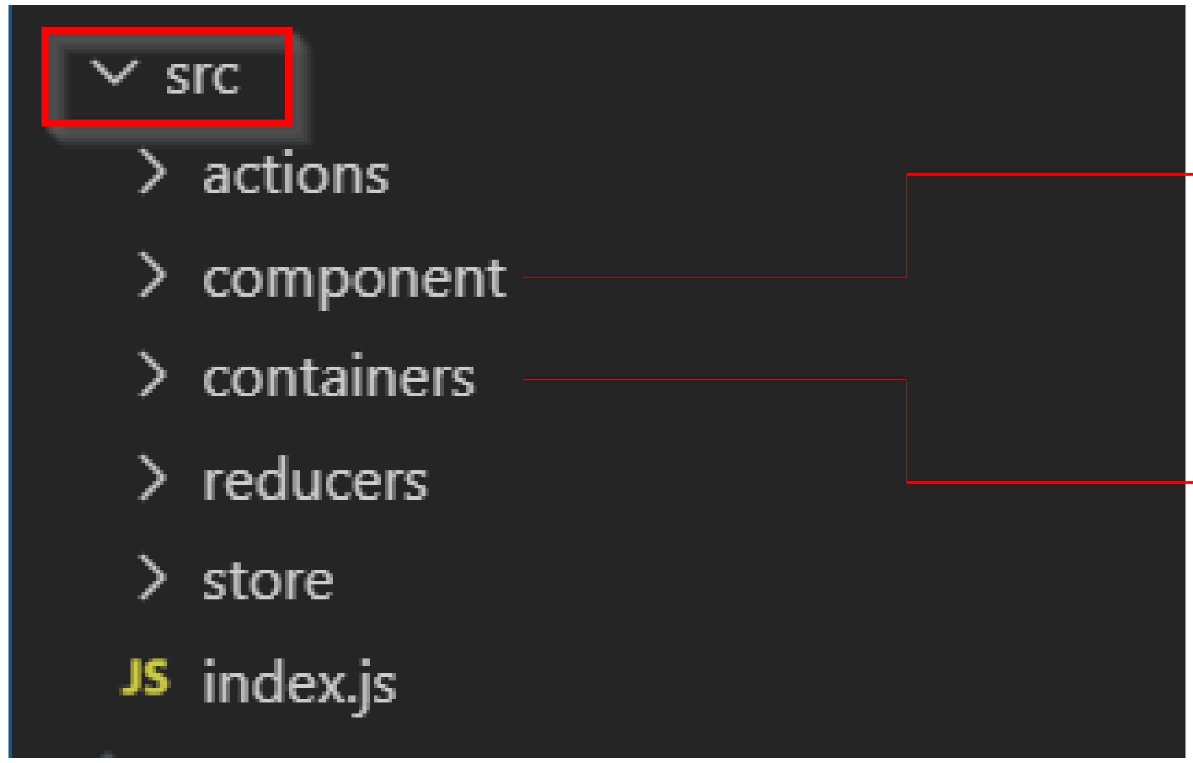
Navigate to your application folder using: ***cd <application_name>***
Install the required packages: ***redux, react-redux, react-router-dom***



```
PS C:\Users\archana\Desktop\React\ReactJSDemo\Redux> cd reduxapp
PS C:\Users\archana\Desktop\React\ReactJSDemo\Redux\reduxapp> npm install redux react-redux react-router-dom
```

Demo: Folder Structure

Open the ***source folder*** and remove all the files. Create a new folder structure as mentioned below:



```
src
  actions
  component
  containers
  reducers
  store
  index.js
```

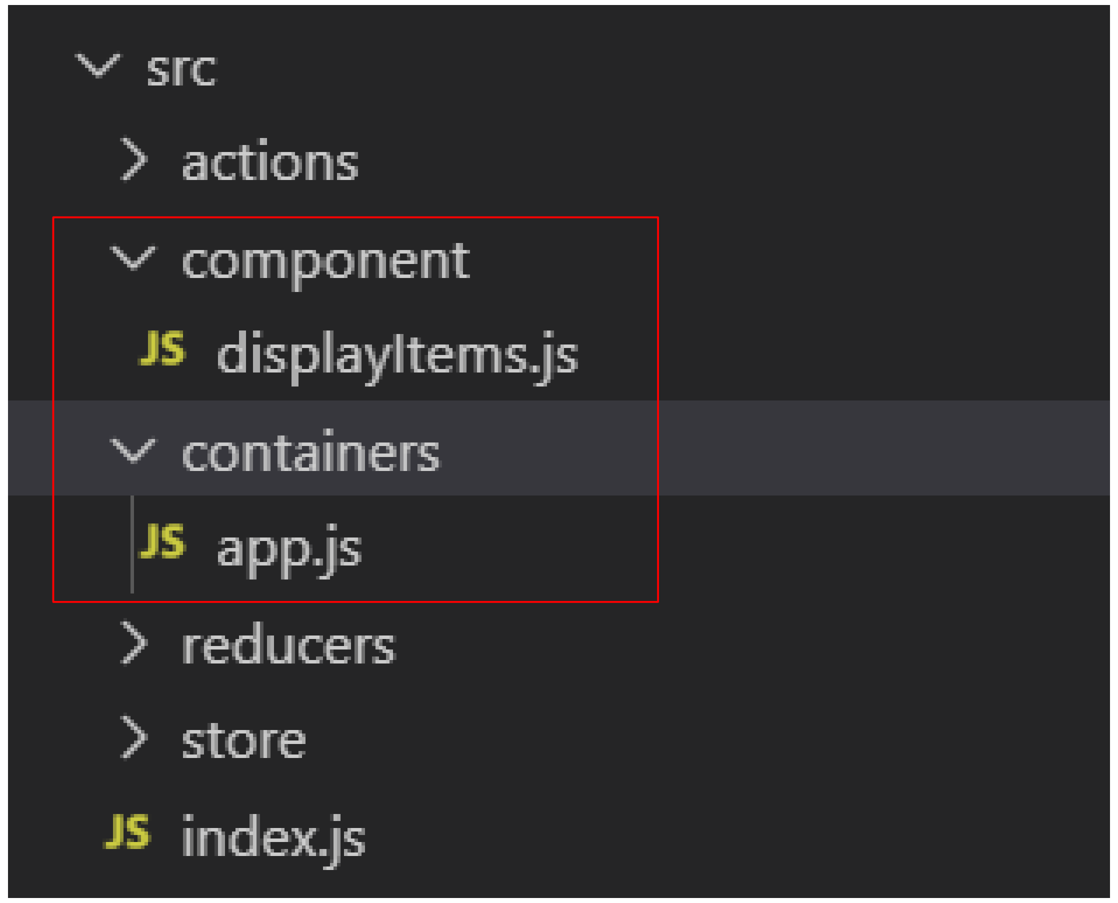
Manages the display part of the application

Manages the communication with Redux

The other folders like actions, reducers, store will have an index.js file which acts as an entry point to the respective folders

Demo: Folder Structure (contd.)

Create two files app.js and displayItem.js and add them to containers and component folder, respectively.



```
src
  actions
  component
    displayItems.js
  containers
    app.js
  reducers
  store
  index.js
```

Demo: Index.js

Add the ***paths*** of created folders to ***Index.js*** and import ***Provider*** to connect store to the view section of your application.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-
redux'; import App from
'./containers/app';  import store from
'./store';

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

Demo: Store

To the index.js of ***store*** folder add the below snippet. As per the architecture, a ***reducer*** is present inside the ***store*** so we will ***import*** it in the store folder.

```
import { createStore, applyMiddleware } from 'redux';

import reducers from '../reducers';


let store = createStore(reducers,
applyMiddleware())
export default store;
```

Creates a Redux store to hold the complete state of the application

It is used to support Asynchronous actions

Connects reducer and middleware to Redux store

Demo: Actions

To the index.js of ***actions*** folder add the below snippet.

```
export function foodItems(){
    return{
        type:'FOOD_ITEMS',
        payload:
        [
         { id:1, name: ‘Donuts’},
         { id:2, name: ‘Ice-cream’},
         { id:3, name: ‘Choclates’}
        ]
    }
}
```

$\rightarrow$ Action type (used to trigger the action)

$\rightarrow$ Application data to be displayed

Demo: Reducers - food_Reducer.js

In the reducer folder create a ***food_Reducer.js*** file. In it write a ***reducer function*** to ***receive*** the ***action***.

```
export default function(state=null,
action)
  {
    switch(action.type){
     case 'FOOD_ITEMS':
         return action.payload
    default:
         return state
  }
}
```

Passes ***initial state and action*** to the function

Action type to be processed

Sends data present in payload to ***state***

Final state after processing the action

Demo: Reducers - Index.js

Add an ***index.js file*** to the reducers folder and ***import*** all the application reducers in it. Collect these reducers in a single object - ***rootReducers*** using ***combineReducers***.

```
import {combineReducers} from 'redux'
import food from './food_Reducers'

const rootReducer = combineReducers (

 {
   food
 }

)

export default rootReducer;
```

The **combineReducers** is a function that turns an object whose values **are** different reducing functions into a single reducing function, which **can** be passed to ***createStore***

File holding the reducer function

Demo: App.js

Open app.js file present in containers folder, paste the below code in it to establish the communication with Redux.

```
import React, {Component} from 'react';
import {connect} from 'react-redux';
import * as actions from  '../actions'

class App extends Component {
componentDidMount(){this.props.foodItems();}

render(){
 return(
 this.props.finalState
)
}}

function mapStateToProps(state){
    return({
        finalState: state.food
    })
}
export default connect(mapStateToProps, actions)(App);
```

As only one action is present

Calls action

Function defined to receive the state from reducers

Props to which received state is assigned

Demo: displayItems.js

In ***component*** folder open displayItems.js file, paste the below code in it to bind the data and send the food items to be displayed on screen.

```
import React from 'react';

const DisplayItems = (props) => {

    const List =({datalist}) => {
        if(datalist){
            return datalist.map((data) => {
                return(
                    <div key={data.id}>
                        {data.name}
                    </div>
                )
})}}

return(
        <div>
        {List(props)}
        </div>
        )
}
export default DisplayItems;
```

Function defined to bind the data using map operator

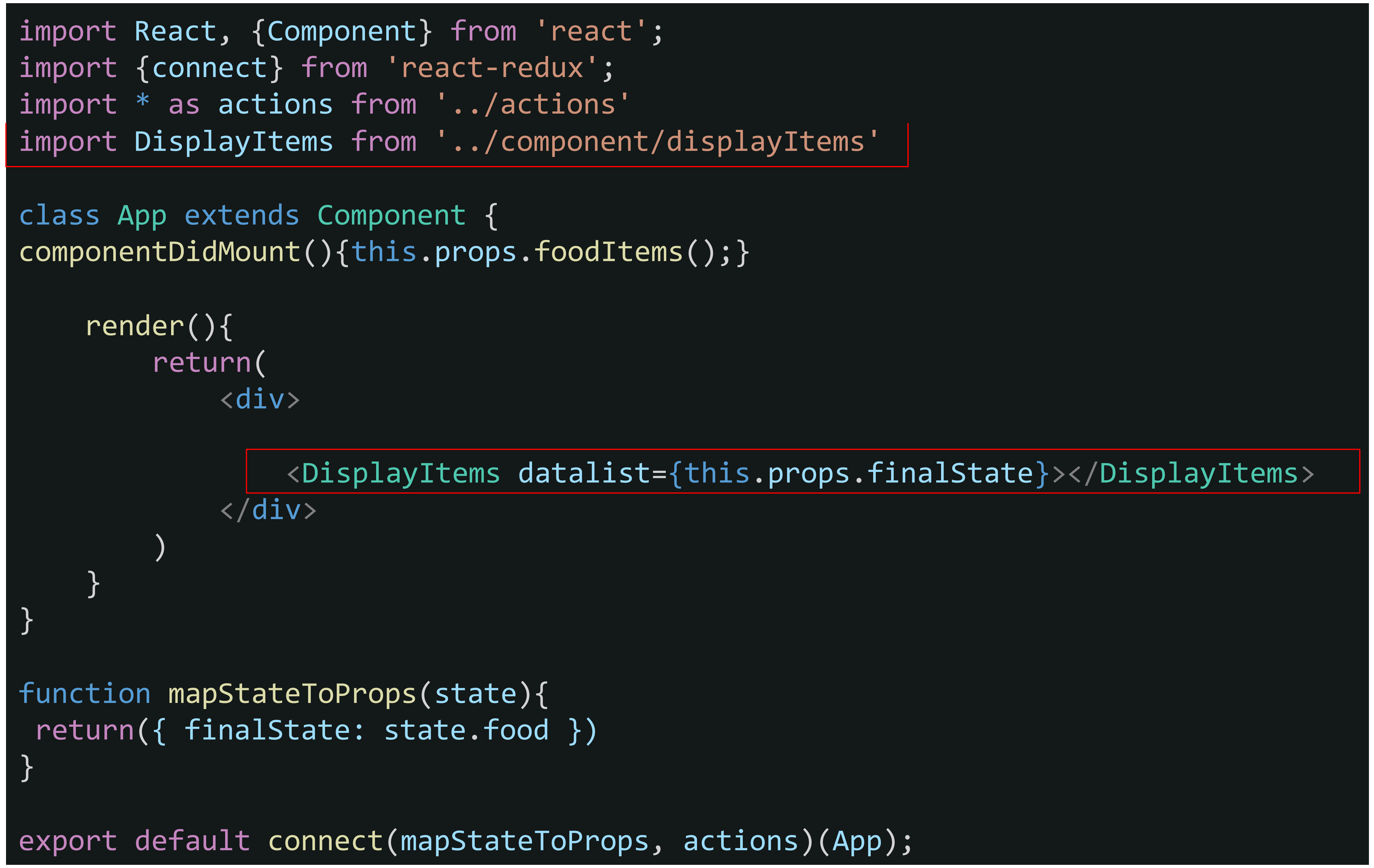
Props that will hold the data

Binding the data using map operator

Returns the props holding the data

Demo: App.js

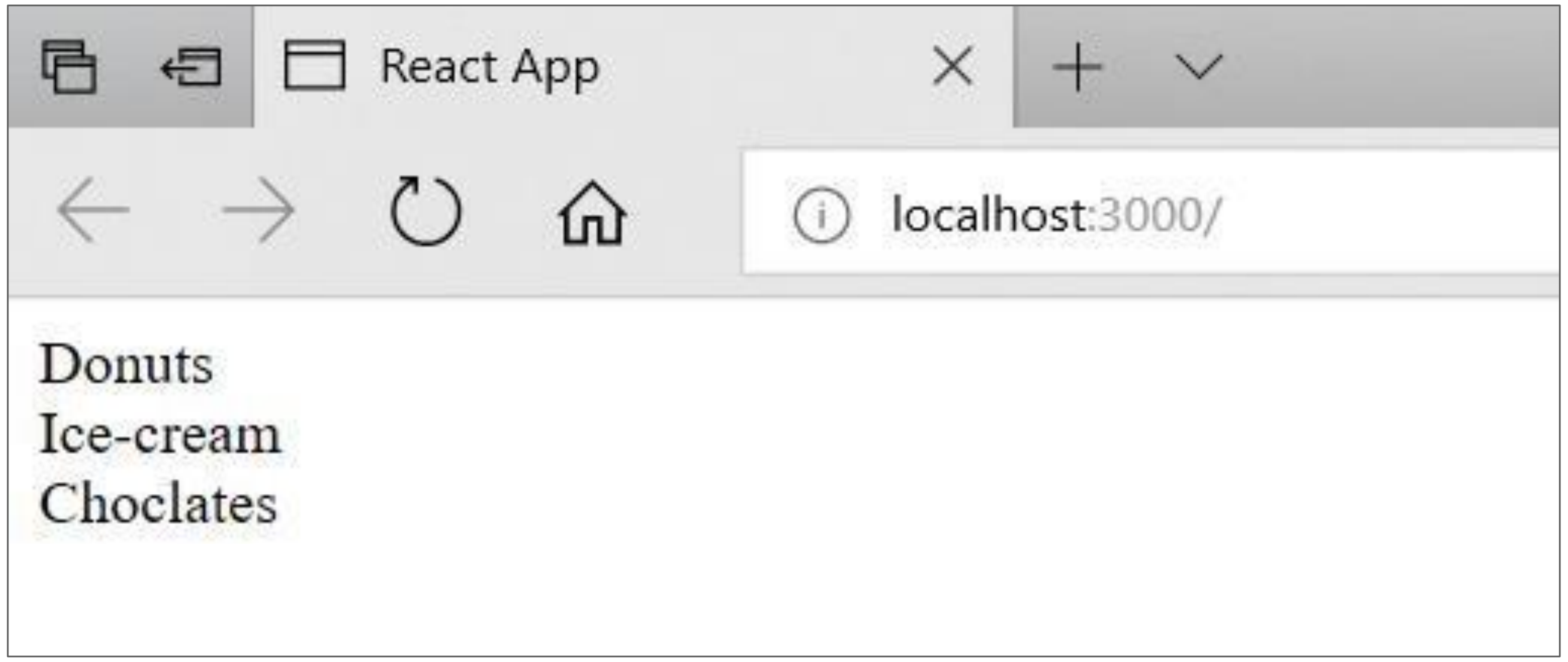
Import the ***DisplayItems*** function in App.js and add the ***props:datalist*** in render function



```
import React, {Component} from 'react';
import {connect} from 'react-redux';
import * as actions from '../actions'
import DisplayItems from '../component/displayItems'

class App extends Component {
componentDidMount(){this.props.foodItems();}

    render(){
        return(
            <div>

                <DisplayItems datalist={this.props.finalState}></DisplayItems>
            </div>
        )
    }
}

function mapStateToProps(state){
 return({ finalState: state.food })
}

export default connect(mapStateToProps, actions)(App);
```

Demo: Output

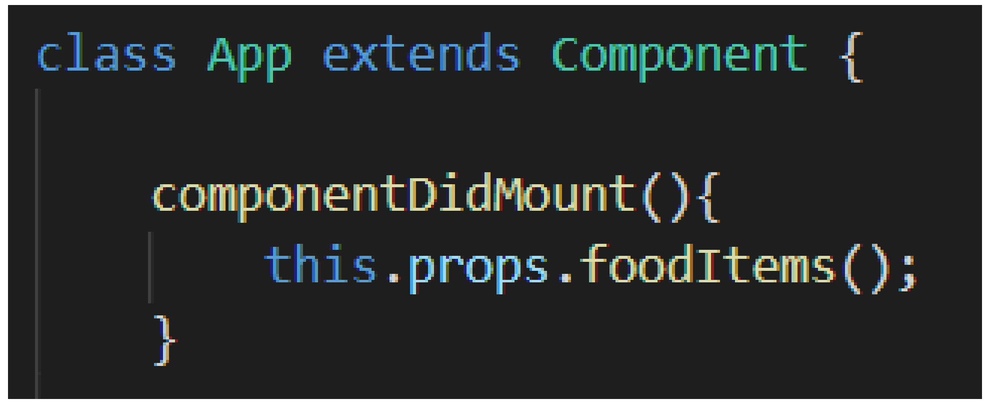
Run the application using: ***npm start*** and check the output at ***localhost:3000***



```
Donuts
Ice-cream
Choclates
```

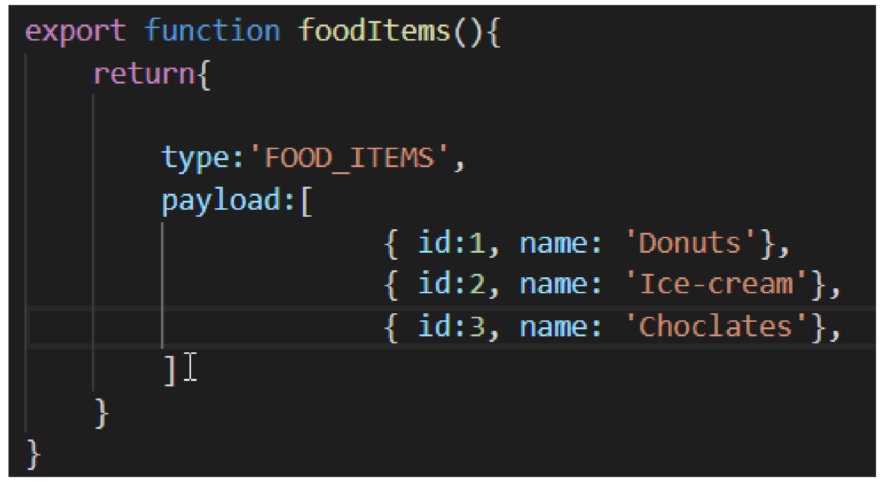
Summary Of Application Using React-Redux

1. App.js (view section) Calls action



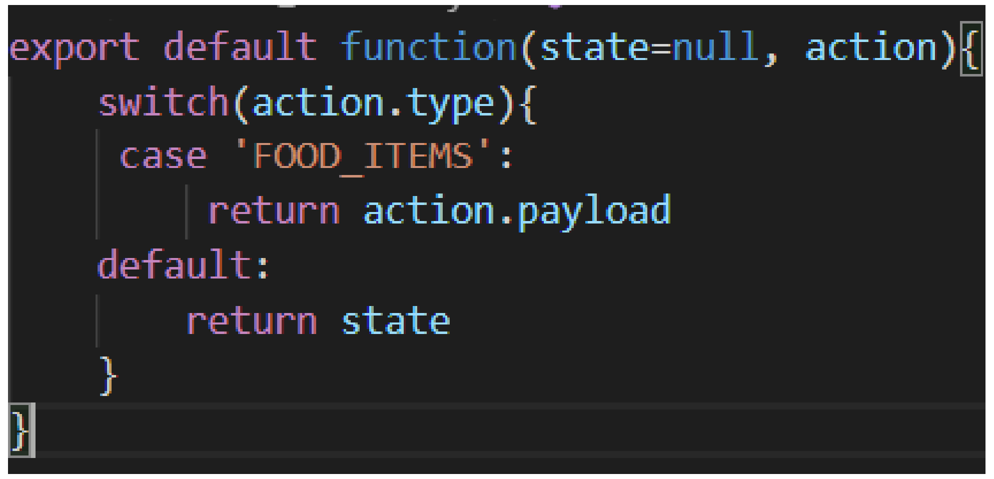
```
class App extends Component {

    componentDidMount(){
        this.props.foodItems();
    }
```

2.Action sends type and payload



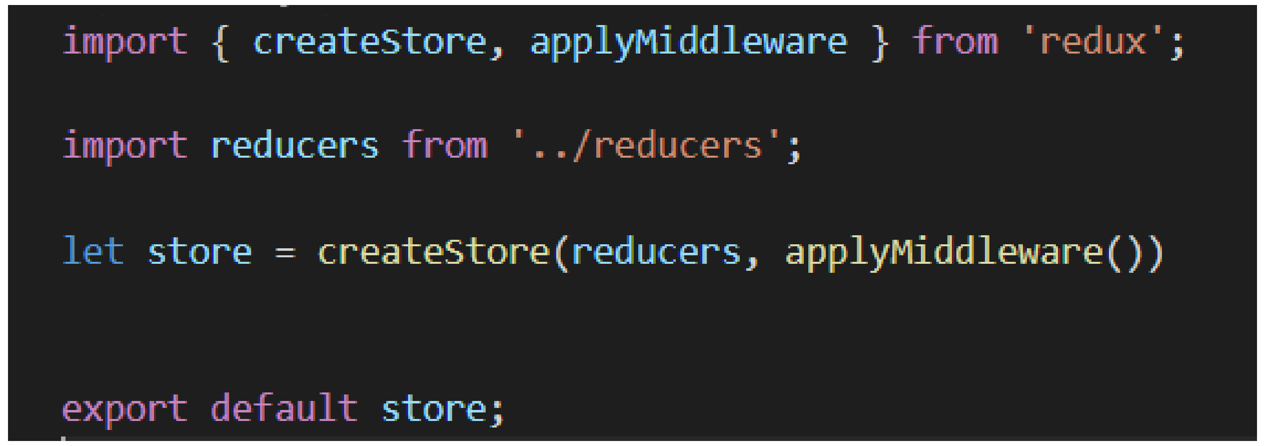
```
export function foodItems(){
    return{

        type:'FOOD_ITEMS',
        payload:[
                        { id:1, name: 'Donuts'},
                        { id:2, name: 'Ice-cream'},
                        { id:3, name: 'Choclates'},
        ]
    }
}
```

3.Reducer accepts the action type and payload



```
export default function(state=null, action){
    switch(action.type){
     case 'FOOD_ITEMS':
         return action.payload
    default:
        return state
    }
}
```

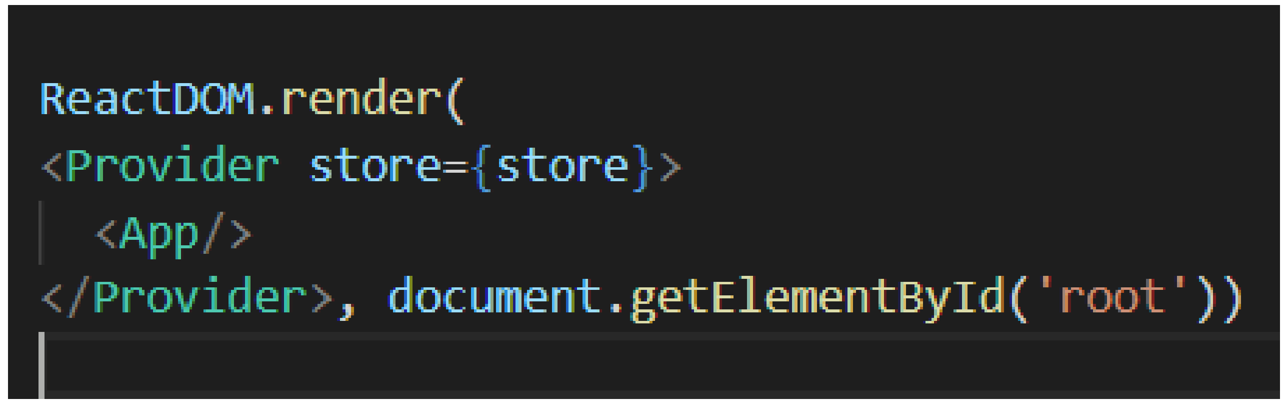
4.Reducer is connected to store



```
import { createStore, applyMiddleware } from 'redux';

import reducers from '../reducers';

let store = createStore(reducers, applyMiddleware())


export default store;
```

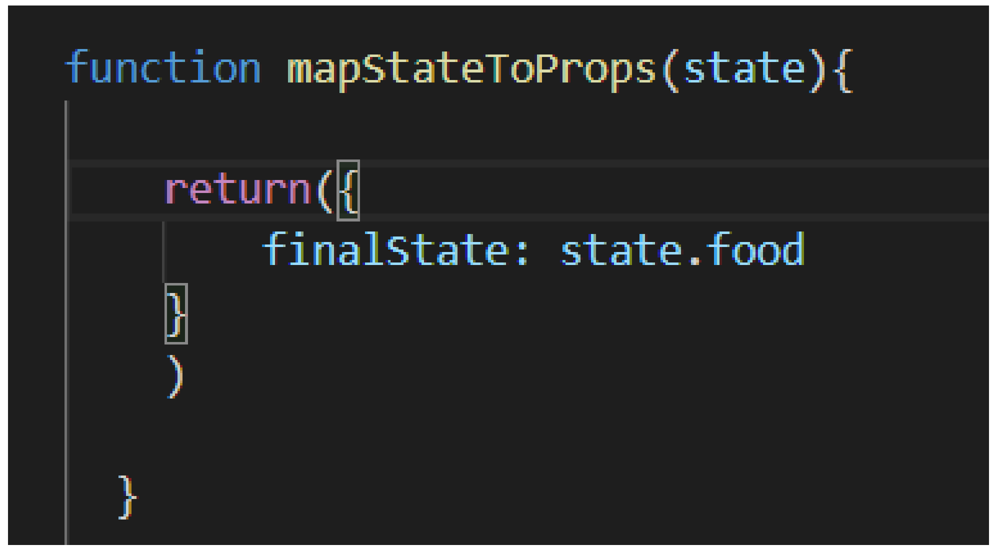
Summary Of Application Using React-Redux

5. Store is connected to view



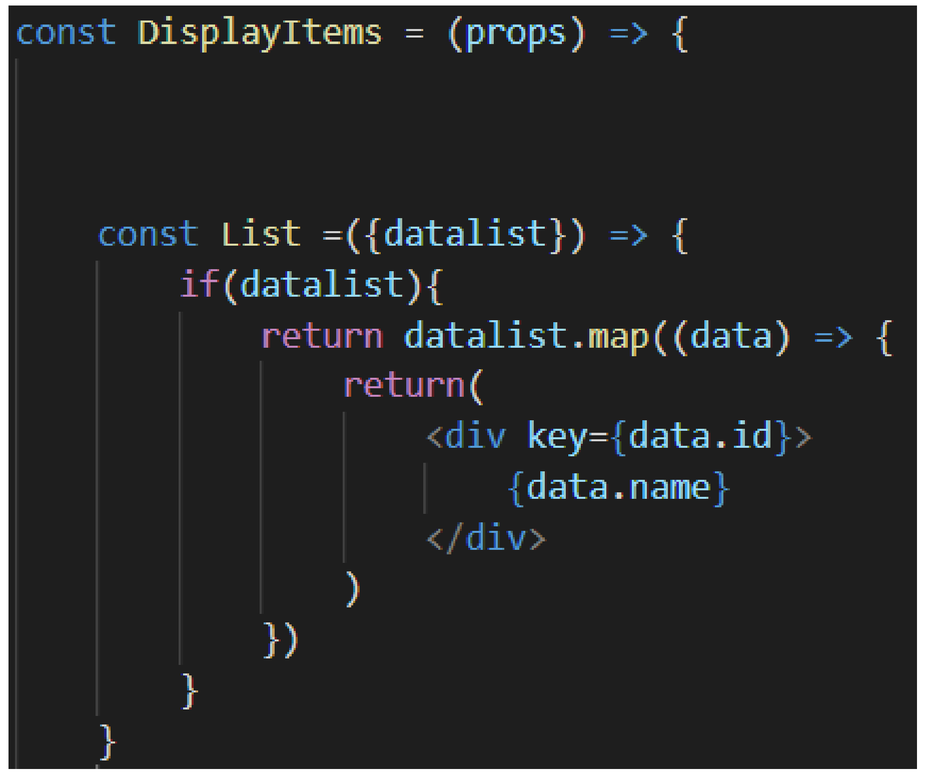
```
ReactDOM.render(
<Provider store={store}>
  <App/>
</Provider>, document.getElementById('root'))
```

6.In view section mapStateToProps calls store, gets state from reducers and maps it to props

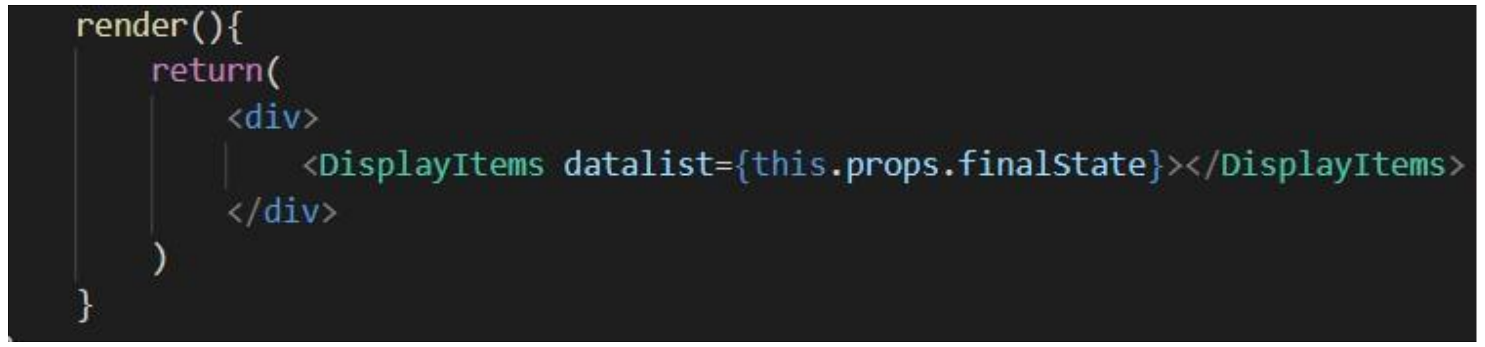


```
function mapStateToProps(state){

    return({
        finalState: state.food
    })
    )

  }
```

7. View performs the data binding and finally props is passed to render function and displayed on screen



```
const DisplayItems = (props) => {



    const List =({datalist}) => {
        if(datalist){
            return datalist.map((data) => {
                return(
                    <div key={data.id}>
                        {data.name}
                    </div>
                )
            })
        }
    }
```

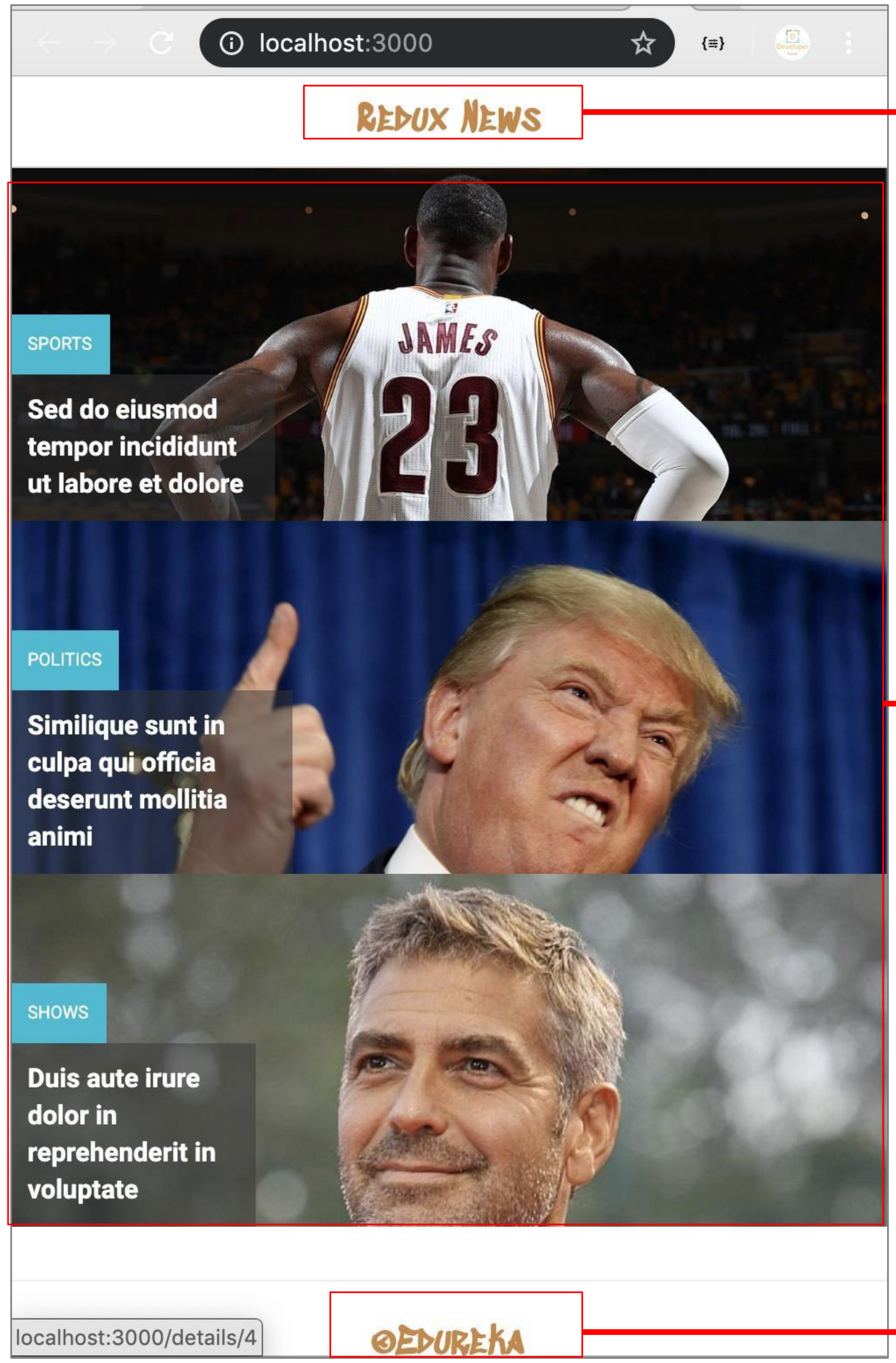


```
render(){
    return(
        <div>
            <DisplayItems datalist={this.props.finalState}></DisplayItems>
        </div>
    )
}
```

Demo 2: To Build A News Application Using React And Redux

Output: News Application Using React And Redux

Here we will build an application using ***React and Redux***, where data is received by an ***API*** and displayed on screen.


localhost:3000
REDUX NEWS
SPORTS
Sed do eiusmod tempor incididunt ut labore et dolore
POLITICS
Similique sunt in culpa qui officia deserunt mollitia animi
SHOWS
Duis aute irure dolor in reprehenderit in voluptate
localhost:3000/details/4
©EDUREKA

Header

News data rendered by an API

Footer

Questions

Comments
Ratings
Suggestions
FEEDBACK
Survey
Likes
Ideas