Redux

Create a store

Create a store object by calling the createStore method. For that import the createStore from the redux package. Open the index/main js file and import the createStore method .

Import {createStore} from 'redux';

Declare the store object as a constant. It ensures you will not be modifying the store variable again and recreating it.

Const store = createStore();

You also need to create reducers. You can split the main reducer code into multiple child reducers and combine them using combineReducers method. Create a file in reducers folder with the name reducer-users.js and add the following code.

```
export default function (){
    return [
        {
            id:1,
            firstname:"Sonu",
            lastname:"Sathyadas",
            age:32,
            description:"Sonu is working as Tech Lead in 
Synergetics",
            
thumbnail:'https://synstorage.blob.core.windows.net/images/sonu.jpg'
        },
        {
            id:2,
            firstname:"Aditya",
            lastname:"Sonu",
            age:2,
            description:"Aditya is playing in floor",
            
thumbnail:'https://synstorage.blob.core.windows.net/images/aditya.jp
g'
        },
        {
            id:3,
            firstname:"Aswathy",
            lastname:"Sonu",
            age:27,
            description:"Achu is working very hard to manage Adi",
thumbnail:'https://synstorage.blob.core.windows.net/images/achu.jpg'
        }
    ];
}
```

You can create more reducers in your application and combine them into a common reducer method. Create a new file called reducers.js in reducer directory and add the following code.

```
import {combineReducers } from 'redux';

import userReducer from './reducer-users';

const allReducers=combineReducers({
    users:userReducer
});

export default allReducers;
```

The name **users** is used to refer the userReducer object and it is used all over the program to access the reducer.

Now you need to provide access to the store object to the entire application. For that you can use the Provider class from the ‘react-redux’ package. Create a App component in the components folder and this will act as the root component.

```
import React from 'react';

const App = () =>(
    <div>
        <h2>Username List:</h2>
        <hr/>
        <h2>User details:</h2>
    </div>
);

export default App;
```

In your main.js/index.js file add the following code. You can use the Provider class as an DOM element to encapsulate the App component. You can pass the store object to the provider element. It makes available to the entire application.

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

import allReducers from './reducers/reducers';
import App from './components/App';

const store = createStore(allReducers);

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

Now you need to create component/container that takes the data from the store and bind to the DOM. For that create a folder called containers in src. Create a file called user-list.jsx in side it add the following code in that.

```
import React, {Component} from 'react';
import {bindActionCreators} from 'redux';
import {connect} from 'react-redux';

class UserList extends Component
{
    render(){
        return (
            <ul>
                <li>One</li>
                <li>Two</li>
                <li>Three</li>
            </ul>
        );
    }
}

function mapStateToProps(state, ){
    return{
        users:state.users
    }
}

export default connect(mapStateToProps)( UserList);
```

The method mapStateToProps(state) is used to map the state data to the props object of the component. For our user-list component we need to map the users list to the users property of the props.

Later use the connect function to invoke the method and pass the data to the component.

Update the user-list.jsx to list all the users firstname and lastname in the page. CreateListItems function returns the list of users.

```
import React, {Component} from 'react';
import {bindActionCreators} from 'redux';
import {connect} from 'react-redux';

class UserList extends Component
{
    createListItems(){
        return this.props.users.map(item=>{
            return (
                <li key={item.id}>{item.firstname} {item.lastname}</li>
            )
        })
    }

    render(){
        return (
            <ul>
                {this.createListItems()}
            </ul>
        );
    }
}

function mapStateToProps(state, ){
    return{
        users:state.users
    }
}

export default connect(mapStateToProps)( UserList);
```

Now we need to create some actions. When you click on the user name, you need to show the details of the user in the below section. For that we can use action. An action contains a type attribute and payload. Payload defines what data need to be passed when calling that action. The first attribute name in action must be “type” but payload (data) name can be anything.

Create a new folder called “actions” and create a file called index.js. It defines an action creator function which return the action object.

```

//create an action creator
export const selectUser=(user)=>{
    console.log("You clicked on user " + user.firstname);
    //return an action with type(what action) and data (on which data)
    return {
        type:"USER_SELECTED",
        payload:user
    }
}
```

Now in the User-list.jsx file you can create a new method to bind the actions to properties. The method name can be matchDispatchToProps(dispatch). This will bind the actiongenerator to the props.

Later update the connect() method to include the new method as the second parameter.

```
import React, {Component} from 'react';
import {bindActionCreators} from 'redux';
import {connect} from 'react-redux';

import {selectUser} from '../actions/index';

class UserList extends Component
{
    createListItems(){
        return this.props.users.map(item=> <li key={item.id}
onClick={()=>this.props.onSelectUser(item)}>{item.firstname} {item.lastname}</li>);
    }

    render(){
        return (
            <ul>
                {this.createListItems()}
            </ul>
        );
    }
}

function mapStateToProps(state, ){
    return{
        users:state.users
    }
}

function matchDispatchToProps(dispatch){
    let actionMap={
        onSelectUser:selectUser
    };
    return bindActionCreators(actionMap, dispatch);
}

export default connect(mapStateToProps, matchDispatchToProps)( UserList);
```

Now you need to define the reducer to execute the action sent by the user selection. When you click the user name it send an action with type “USER_SELECTED”. So you need to create a reducer method to return the user detail. Add a new reducer called “reducer-activeuser.js” in reducers folder.

```

export default function(state=null, action){
    switch(action.type){
        case "USER_SELECTED":
            return action.payload;
            break;
    }
    return state;
}
```

Add the reducer to the reducers file using combineReducers method

```
import {combineReducers } from 'redux';

import UsersReducer from './reducer-users';
import ActiveUserReducer from './reducer-activeuser';

const allReducers=combineReducers({
    users:UsersReducer,
    activeUser:ActiveUserReducer
});

export default allReducers;
```

Now create a new container to display the details of selected user. Create a component in containers folder with the name "user-detail.jsx". Add the following code to it.

```
import React, { Component } from 'react';
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';

class UserDetail extends Component {
    render() {
        if (this.props.user) {
            return (
                <div>
                    <img src={this.props.user.thumbnail} />
                    <h2>Age: {this.props.user.firstname}</h2>
                    <h3>Description: {this.props.user.description}</h3>
                </div>
            );
        }
        else{
            return (<div>Select a user</div>)
        }
    }
}

function mapStateToProps(state) {
    return {
        user: state.activeUser
    };
}

export default connect(mapStateToProps)(UserDetail);
```

We are adding the if condition in the render method, because when the app loads we are not yet selected any user. So the default value of "user" property will be null. Because we are setting in the activeusers reducer state as null. It shows an error in browser something like “cannot read thumbnail of null”. To avoid that check if user object available or not. If not return a default message.