Redux with create-react-app tool

You can create a react application using a built-in tool available on npm. It is create-react-app. Open the command prompt and install create-react-app tool globally using the following command.

npm install -g create-react-app

Once the installation is done, you can use the create-react-app command to create a new project.

Create-react-app myapp

It creates a pre-configured project. Open the project folder in command prompt and you can run it using the npm start command.

Once the project is opened in VS Code you can add the redux libraries required for the project. Install “react”, “react-redux”, “redux-thunk” packages using npm.

npm install –save react react-redux redux-thunk

Redux thunk: Redux Thunk middleware allows you to write action creators that return a function instead of an action. The thunk can be used to delay the dispatch of an action, or to dispatch only if a certain condition is met. The inner function receives the store methods dispatch and getState as parameters.

You also need to install the redux dev tools extension using the following command

npm install –save redux-devtools-extension

To implement React CRUD operation application we need to follow the steps:

1) Create store
2) Setup middleware
3) Create providers

To start with, open the index.js file and create store object by using the following code. Don’t forget to import the createStore method from ‘redux’ package.

```
Import {createStore} from ‘redux’;

const store=createStore(

        rootReducer

);
```

The root reducer is a file which contains the code for combining all reducers. Next, we need to import the applyMiddleware method from the redux package. Also, import the composeWithDevTools from the ‘redux-devtools-extensions’ package.

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

import {composeWithDevTools} from 'redux-devtools-extension';
```

Also import the thunk middleware from the redux-thunk package

```
Import thunk from ‘redux-thunk’;
```

Now, you need to update the createStore method to apply thunk middleware to it.

```
    const store=createStore(
          rootReducer,
          applyMiddleware(thunk)
    );
```

This extension will help you to see the redux state values in chrome browser inspect element section.

Create the rootReducer in the folder ‘reducers’.

```
import {combineReducers} from 'redux';
import games from './games';
export default combineReducers({
    games
})
```

The Root reducer combines all child reducers, here we have a child reducer which returns the list of all games. Create a file called games-reducer.js in the reducers folder and add the following code to it.

```
export default function games(state=[], action=[]){
    switch(action.type){
        default:
            return state;
    }
}
```

Finally import the provider from react-redux and wrap the root component in the Provider.

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

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

The complete code for index.js looks like:

```
import React from 'react';
import ReactDOM from 'react-dom';
import { createStore, applyMiddleware } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import { Provider } from 'react-redux';

import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import rootReducer from './reducers/rootReducer';
import thunk from 'redux-thunk';

const store = createStore(
    rootReducer,
    composeWithDevTools(
        applyMiddleware(thunk)
    )
);
ReactDOM.render(
    <Provider store={store}>
        <App />
    </Provider>, document.getElementById('root'));
registerServiceWorker();
```

Configuring React Router for the application

To configure the react router in to your application, you need to install the react-router package to your application.

npm install –save react-router-dom

Open the index.js and import the BrowserRouter from the react-router package.

Import {BrowserRouter} from 'react-router-dom';

Add the BrowserRouter element to wrap the Provider element. Add the Provider inside the BrowserRouter

```
ReactDOM.render(

    <BrowserRouter>

        <Provider store={store}>
```

```
            <App />
        </Provider>
    </BrowserRouter>, document.getElementById('root'));
```

Now, we can define some links in the App.js file to navigate to various pages. Open the App.js file and add some hyperlinks to it.

```
import React, { Component } from 'react';
import { NavLink, Route, Switch } from 'react-router-dom';
import './App.css';
import Home from './components/home';
import About from './components/about';
import GamesPage from './components/games-page';
class App extends Component {
    render() {
        return (
            <div className="App">
                <hr />
                <NavLink to="/">Home</NavLink> |
                <NavLink to="about">About</NavLink> |
                <NavLink to="games">Games</NavLink>
                <hr />
                <Switch>
                    <Route exact path="/" component={Home} />
                    <Route path="/about" component={About} />
                    <Route path="/games" component={GamesPage} />

                </Switch>
            </div>
        );
```

```
    }
}
export default App;
```

Create the components for ‘GamesPage’, ‘Home’ and ‘About’.

Now Open the GamesPage component and write the code to connect to the redux. You can use download the prop-types node package to define the propTypes of the component.

npm install –save prop-types

Add the following code to the GamesPage.jsx

```
import React, {Component} from 'react';
import {connect} from 'react-redux';
import PropTypes from 'prop-types';

class GamesPage extends Component{
    render(){
        return(
            <div>
                <h2>Games Page</h2>
            </div>
        );
    }
}
GamesPage.propTypes={
    games:PropTypes.array.isRequired
}
function mapStateToProps(state){
    return{
        games:state.games
```

```
    }
}
export default connect(mapStateToProps)(GamesPage);
```

We can create a child component to list all the games in the GamesPage. Create a new file in the components folder and name it as game-list.jsx

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import PropTypes from 'prop-types';


export default function GamesList({ games }){
    const emptyMessage=(
        <p>There is no games in your collection</p>
    );
    const gamesList=(
        <p>Games list</p>
    );
    return (
        <div>{games.length===0 ? emptyMessage : gamesList} </div>
    );
}
GamesList.propTypes = {
    games: PropTypes.array.isRequired
}
```

Connecting to back-end server using Ajax to fetch games data

To implement backend server connectivity using AJAX we need to create a backend server first. For that create a folder in the project root folder with the name “backend”. Add a file called server.js.

Now install the express package and mongodb package

npm install –save mongodb express

Also add the following packages to transpile ES6 code

npm install –save-dev nodemon babel-cli babel-preset-es2015

Because, we are using babel, create a .babelrc file in the backend folder. Add the following code.

```
{
    "presets": [
        "es2015"
    ]
}
```

Open the server.js file and add the following code.

```
import express from 'express';
import mongodb from 'mongodb';

const app = express();
const dbUrl = 'mongodb://localhost:27017';

mongodb.MongoClient.connect(dbUrl, function (err, client) {
    const db=client.db("GamesDb");
    //apis
    app.get("/api/games", (req, resp) => {
        db.collection('games')
            .find({})
            .toArray((err, games) => {
                resp.json({ games });
            });
    })
    if (!err) {
        app.listen(8080, () => console.log('Server is running on
localhost:8080'));
    } else {
        console.log("Could not start server, error in connecting
MongoDB")
    }
})
```

Now, when you run the server application it runs on port number 8080. But our react application is running on port number 3000. So we can add a code in the package.json file of the react application to redirect all requests to port 8080. Open the package.json file of the react application (not backend server package.json) and add the following configuration at the end.

```
  "scripts": {
```

```
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  },
  "proxy":"http://localhost:8080/"
}
```

Create a new folder called “actions” in the react application and add an index.js file to it. Add a method ‘fetchGames” to call the API to get the list of games.

```
export function fetchGames(){
    return dispatch =>{
        fetch("/api/games")
        .then(resp=>resp.json())
        .then(data=>dispatch(setGames(data.games)));
    };
}
```

The fetch function reads the data and convert to json and dispatch to the action called setGames. So we also need to create an action creator method called setGames().

```
export const SET_GAMES='SET_GAMES';

export function setGames(games){
    return{
        type:SET_GAMES,
        games
    }
}
export function fetchGames(){
    return dispatch =>{
        fetch("/api/games")
        .then(resp=>resp.json())
        .then(data=>dispatch(setGames(data.games)));
    };
}
```

Open the games-page.jsx and add the componentDidMount() lifecycle method to it. Also you need to update the render method to call the Games-list component to list all games.

```
class GamesPage extends Component {
```

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

    render() {
        return (
            <div>
                <h2>Games Page</h2>
                <GamesList games={this.props.games} />
            </div>
        );
    }
}
```

Add Bootstrap to React App

Copy the CDN urls of bootstrap and jquery to the index page.

Adding forms to Redux

You can now add a new link to move to a form where we can add new games details. Update the App.js with the following code.

```
import React, { Component } from 'react';
import { NavLink, Route, Switch } from 'react-router-dom';

import Home from './components/home';
import About from './components/about';
import GamesPage from './components/games-page';
import GameForm from './components/game-form';

class App extends Component {
    render() {
        return (
            <div className="container-fluid">
                <hr />
                <NavLink to="/">Home</NavLink> |
                <NavLink to="/about">About</NavLink> |
                <NavLink to="/games">Games</NavLink> |
                <NavLink to="/games/new">New Game </NavLink>
                <hr />
                <Switch>
                    <Route exact path="/" component={Home} />
                    <Route path="/about" component={About} />
```

```
                    <Route exact path="/games" component={GamesPage}
/>
                    <Route exact path="/games/new"
component={GameForm}/>
                </Switch>
            </div>
        );
    }
}

export default App;
```

To complete the task we need to create a form. Add a new jsx file to the components folder with the name ‘game-form.jsx’. Add the following code to it.

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

class GameForm extends Component {
    constructor() {
        super();
        this.state = {
            title: "",
            cover: "",
            errors: {}
        };
        this.handleChange = this.handleChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
    }

    handleChange(e) {
        let errors=Object.assign({}, this.state.errors);
        delete errors[e.target.name];

        this.setState({
            [e.target.name]: e.target.value,
            errors
        })
    }

    handleSubmit(e) {
```

```
        e.preventDefault();
        let errors = {};
        if (this.state.title === '') {
            errors.title = "Cannot be empty";
        }
        if (this.state.cover === '') {
            errors.cover = "Cannt be empty";
        }
        this.setState({ errors });
    }

    render() {
        return (
            <div className="row">
                <div className="col-md-6 col-md-offset-3">
                    <form className="form"
onSubmit={this.handleSubmit}>
                        <div className="form-group">
                            <label className="control-
label">Title</label>
                            <input type="text" className="form-
control"
                                name="title" value={this.state.title}
onChange={this.handleChange} />
                            {
                                this.state.errors.title && <div
className="text-danger">{this.state.errors.title}</div>
                            }
                        </div>
                        <div className="form-group">
                            <label className="control-
label">Cover</label>
                            <input type="cover" className="form-
control"
                                name="cover" value={this.state.cover}
onChange={this.handleChange} />
                            {
                                this.state.errors.cover && <div
className="text-danger">{this.state.errors.cover}</div>
                            }
                        </div>
                        <div className="form-group">
```

```
                            {this.state.cover !== "" && <img
classNameName="thumbnail" src={this.state.cover} />}
                        </div>
                        <div classNameName="form-group">
                            <button type="submit" classNameName="btn
btn-success"> Add </button>
                        </div>
                    </form>
                </div>
            </div>
        );
    }
}

export default connect()(GameForm);
```

Update the server.js file with the following code to handle the NotFound error

```
app.use((req,res)=>{
        res.status(404)
        .json({
            errors:{
                global:"Something went wrong, try after some time"
            }
        })
    })
```

Now, we need to update the actions.js file to send a save the form data. Add the following code at the end of actions.js file

```
export function saveGame(game) {
    return dispatch => {
        return fetch('/api/games', {
            method: 'post',
            body: JSON.stringify(game),
            headers: {
                "Content-Type": "application/json"
            }
        })
        .then(handleResponse);
```

```
    }
}

// handling response
function  handleResponse(response){
    if(response.ok){
        return response.json();
    }else{
        let error=new Error(response.statusText);
        error.response=response;
        throw error;
    }
}
```

Now, we need to update the handleSubmit method to do the validation of controls and call the API using saveGame method

```
handleSubmit(e) {
        e.preventDefault();
        let errors = {};
        if (this.state.title === '') {
            errors.title = "Cannot be empty";
        }
        if (this.state.cover === '') {
            errors.cover = "Cannt be empty";
        }
        this.setState({ errors });
        //check form is valid
        const isValid = Object.keys(errors).length === 0;
        if (isValid) {
            this.setState({ loading: true });
            const { title, cover } = this.state;
            this.props.saveGame({ title, cover })
                .then(
                    (data) => { console.log(data) },
                    (err) => err.response.json()
                        .then(({ errors }) => {
                            this.setState({ errors, loading:
false });
                        })
                );
        }
```

```
    }
```

The error callback add the global error message to the errors property of state object. Show the error message in the view by adding the following code inside the form tag.

```
   {
    !!this.state.errors.global && <div className="alert alert-
danger">
    <p>{this.state.errors.global}</p>
    </div>
   }
```

Handle the POST request in server

Install the body-parser middleware and configure it.

npm install –save body-parser