**React Project Setup From Scratch**

1) Create new repository using command “**mkdir ui-setup**”.

2) Enter in your project repository and setup node configuration using command “**npm init -y**”.

3)Once inside install webpack, webpack-cli, and the webpack-dev-server using command "**npm i webpack webpack-cli webpack-dev-server –save-dev**".

4) To run webpack easily from an NPM script, open up package.json and configure a "dev" script '**"dev": "webpack --mode development**"'.

5) Make source repository using command “**mkdir src**”.

6) Run index.js file inside src folder and write some code using command “**echo 'console.log("Hello webpack!")' > src/index.js**”.

**Configuring webpack :-**

7) To configure webpack through a file create a webpack.config.js in the root folder using command “**touch webpack.config.js**”.

8) In webpack.config.js we can change how webpack behaves by adding or altering:

entry point

output

loaders

plugins

code splitting

9) To change the entry point path we can do:

const path = require("path");

module.exports = {

entry: { index: path.resolve(\_\_dirname, "source", "index.js") }

};

10) Now webpack will look in source/index.js for the first file to load. To change instead the output of our bundle we can do:

const path = require("path");

module.exports = {

output: {

path: path.resolve(\_\_dirname, "build")

}

};

**Working with HTML :-**

11) A web application without an HTML page is almost useless. To work with HTML in webpack we need to install a plugin, html-webpack-plugin using command "**npm i html-webpack-plugin --save-dev**".

12) Once the plugin is installed we can configure it:

const HtmlWebpackPlugin = require("html-webpack-plugin");

const path = require("path");

module.exports = {

plugins: [

new HtmlWebpackPlugin({

template: path.resolve(\_\_dirname, "src", "index.html")

})

]

}

13) create a simple HTML file in src/index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Webpack tutorial</title>

</head>

<body>

</body>

</html>

**Webpack's development server:-**

14) Install the package now using command "**npm i webpack-dev-server –save-dev**".

15) To configure webpack-dev-server, open up package.json and add a "start" script '**"start": "webpack serve –open"**'.

16) add below code to remove source map warning in webpack.conig.js:

devtool: "source-map"

**Working with webpack's loaders:-**

17) To work with CSS in webpack we need to install at least two loaders:

* **css-loader** for loading CSS files with import
* **style-loader** for loading the stylesheet in the DOM

18) Install above loaders using command “**npm i css-loader style-loader –save-dev**”.

19) Then configure them in webpack.config.js:

module: {

rules: [

{

test: /\.css$/,

use: ["style-loader", "css-loader"]

}

]

},

20) To test CSS in webpack create a simple stylesheet in src/style.css:

h1 {

color: orange;

}

21) Also, add an HTML element to our HTML template in src/index.html:

<h1>Hello webpack!</h1>

22) Finally, load the CSS in src/index.js:

import "./style.css";

**Working with modern JavaScript:-**

23) webpack doesn't know on its own how to transform JavaScript code. This task is outsourced to a third-party loader, specifically babel-loader, with babel.

Before moving forward we need to install a bunch of packages:

* babel core, the actual engine
* babel preset env for compiling modern Javascript down to ES5
* babel loader for webpack

24) Import above module using command “**npm i @babel/core babel-loader @babel/preset-env –save-dev**”.

25) Then configure babel by creating a new file, babel.config.json. Here we configure babel to use preset-env:

{

"presets": [

"@babel/preset-env"

]

}

26) Finally, configure webpack to use the loader for transforming JavaScript files

{

test: /\.js$/,

exclude: /node\_modules/,

use: ["babel-loader"]

}

27) To use React components with webpack, alongside with babel loader you should also install the babel preset for React using command "npm i @babel/preset-react –save-dev".

28) Once done, configure babel to use the React preset in babel.config.json:

{

"presets": ["@babel/preset-env", "@babel/preset-react"]

}

29) Finally can install React using command "npm i react react-dom –save".

30) Update index.js file with code:

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import "./style.css";

ReactDOM.render(

<App />,

document.getElementById('root'),

);

31) add code “<div id = "root"></div>” in index.html.

32) create App.js file with code:

import React from 'react';

export default class App extends React.Component {

render() {

return (

<p>Welcome to react world.</p>

);

}

}

**Run app in production mode:-**

33) just add script in packages.json script ‘**"build": "webpack --mode production"**’.

**Work with performance:-**

1) Code splitting refers to an optimization technique aiming at:

* avoid big bundles
* avoid dependencies duplication

2) Code splitting with optimization.splitChunks just Add following line in webpack.config.js:

optimization: {

splitChunks: { chunks: "all" }

},

**Work with eslint:-**

1) Install the following linter packages:

* eslint is the core JavaScript linter.
* eslint-loader tells webpack that you want to use eslint in our build
* babel-eslint provides linting for valid ES6 code
* eslint-plugin-react extends ESLint rules to cover React

2) To install above packages run command “npm install --save-dev eslint eslint-loader babel-eslint eslint-plugin-react”.

3) Next create an ESLint file to configure our settings using command "touch .eslintrc".

4) Place the following in .eslintrc file you just created:

5) add ‘**"eslint": "eslint ."**’ code into scripts in package.json file.

6) create file .eslintignore and add node\_module and build folder in it like:

/node\_modules

/build

7) Replace .eslintrc rules section with:

"rules": {

"space-in-parens": [ 0, "always" ],

"template-curly-spacing": [ 2, "always" ],

"array-bracket-spacing": [ 2, "always" ],

"object-curly-spacing": [ 2, "always" ],

"computed-property-spacing": [ 2, "always" ],

"no-multiple-empty-lines": [ 2, { "max": 1, "maxEOF": 0, "maxBOF": 0 } ],

"quotes": [ 1, "single", "avoid-escape" ],

"no-use-before-define": [ 2, { "functions": false } ],

"semi": [0, "never"],

"prefer-const": 1,

"react/prefer-es6-class": 0,

"react/jsx-filename-extension": 0,

"react/jsx-curly-spacing": [ 2, "always" ],

"react/jsx-indent": [ 2, 4 ],

"react/prop-types": [ 1 ],

"react/no-array-index-key": [ 1 ],

"class-methods-use-this": [ 0 ],

"no-undef": [ 0 ],

"no-case-declarations": [ 1 ],

"no-return-assign": [ 1 ],

"no-param-reassign": [ 1 ],

"no-shadow": [ 1 ],

"camelcase": [ 1 ],

"no-underscore-dangle" : [0, "always"],

}

8) to hide version waring of eslint just add below code in .eslintrc file:

settings: {

react: {

version: 'detect',

}

}

**Work with React Test with Karma, jasmine and enzyme:-**

1) Install karma and its dependencies using followings commands:

a) npm install karma –save-dev

b) npm install karma-jasmine karma-chrome-launcher jasmine-core –save-dev

c) npm install karma-webpack karma-sourcemap-loader --save-dev

2) Now test karma is working properly using command “**./node\_modules/karma/bin/karma start**”

you can test effects after every changes using commands(optional) “**./node\_modules/karma/bin/karma start --single-run**”

3) Genrate your configuration file using command "**./node\_modules/karma/bin/karma init**".

4) Now let’s modified your karma.conf.js using following steps:

a) Include require modules in the top of file:

require('webpack');

const path = require('path');

b) Update files section and add following in it’s array:

{ pattern: 'spec/javascripts/spec-bundle.js', watched: false },

5) Create folders and files mention in files pattern using command “mkdir spec && mkdir spec/javascripts && touch spec/javascripts/spec-bundle.js”.

6) Update spec-budle.js file with following code:

const context = require.context('./', true, /\.spec\.js$/);

context.keys().forEach(key => describe(`[${key}]`, () => context(key)));

7) update preprocessors section in karma.conf.js file with following code:

'spec/javascripts/\*\*/\*.js': ['webpack', 'sourcemap'],

8) update webpack in karma.conf.js file with following code:

webpack: {

devtool: 'inline-source-map',

module: {

rules: [

{

test: /\.js$/,

exclude: /(node\_modules)/,

loader: 'babel-loader',

},

],

},

resolve: {

alias: {

shop: path.resolve(\_\_dirname, 'src/javascripts'),

},

},

},

9) Setup enzyme before create your tests using following steps:

a) Install enzyme and its adopter using command “**npm i --save-dev enzyme enzyme-adapter-react-16**”

b) create enzyme config file using command “**mkdir spec/javascripts/helpers && touch spec/javascripts/helpers/enzyme-config.js**”.

c) Insert following code in enzyme-config.js file:

import { configure } from 'enzyme';

import Adapter from 'enzyme-adapter-react-16';

configure({ adapter: new Adapter() });

10) Let’s create a new file named App.spec.js to test App.js file. Now update file with following code:

import { shallow } from 'enzyme';

import React from 'react';

import App from 'shop/App';

import './helpers/enzyme-config';

describe( '<App /> ', () => {

const wrapper = shallow( <App /> );

const p = wrapper.find( 'p' );

it( 'renders a <p>', () => {

expect( p.exists() ).toBe( true );

} );

it( 'p had expected a text', () => {

expect( p.text() ).toContain( 'Welcome to react world.' );

} );

} );

11) finally modified test scripts with karma stating using bellow code:

"test": "karma start --single-run",

"karma": "karma",

12) Now you can run your test using commands “**npm test**” or “**npm run test**”.

**Implement redux in react app:-**

**Redux is a predictable state container for the JavaScript apps.**

This means Redux allows us to structure the app in such a manner that the data can be managed consistently and handled easily within the application.

1) Install redux and its dependencies using followings commands: “npm install redux react-redux redux-thunk jquery prop-types”.

2) Now create some repositories for following purpose:

* **containers**:- containers are the primary components and the entry point from which we call child or generic components.
* **components:-** The component is the basic building block of any application. Thus the components created in this application are designed generically so that they can be re-usable throughout the application.
* **selectors**:- A selector is simply a function that accepts Redux state as an argument and returns data that is derived from that state.
* **actions:-** actions are payloads of information that send data from your application to your store.
* **reducers:-** actions describe the fact that something happened but don’t specify how the application’s state changes in response. That is the job of reducers.
* **thunks:-** 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.
* **apigateway:-** use an API service to call endpoint.
* **store:-** A store holds the whole state tree of your application. The only way to change the state inside it is to dispatch an action on it.

3) For test api I have used following tutorial:

https://scotch.io/tutorials/json-server-as-a-fake-rest-api-in-frontend-development

4) Run server using command “./node\_modules/json-server/lib/cli/bin.js --watch db.json”.

5) Next up create a new file, src/javascripts/store/index.js and initialize the store using code:

import \* as redux from 'redux';

import thunk from 'redux-thunk';

import reducer from './../reducers/rootReducer';

import API from './../apigateway/';

export default function createStore() {

return redux.createStore(

reducer,

redux.compose(

redux.applyMiddleware(thunk.withExtraArgument(API)),

// window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ && window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_()

),

);

}

6) create a new file, src/javascripts/apigateway/index.js for api with code:

export default {};

7) create a new file, src/javascripts/reducers/rootReducer.js with code:

import { combineReducers } from 'redux';

import { todoList } from './todos';

export default combineReducers({

todos,

});

8) create a new file, src/javascripts/reducers/todos.js with code:

import { TODO\_LIST } from './../actions/todos';

export function todoList(state = [], action) {

switch (action.type) {

case TODO\_LIST:

return action.value;

default:

return state;

}

}

9) create a new file, src/javascripts/actions/todos.js with code:

export const TODO\_LIST = 'TODO\_LIST';

export function setTodoList(value) {

return { type: TODO\_LIST, value };

}

10) create a new file, src/javascripts/selectors/index.js with code:

export function getTodoList( state ) {

return state.todoList;

}

11) Now create a container in containers folder named "todoContainer.js" with code:

import { setTodoList } from './../thunks/';

import Todo from './../components/todos/';

import {

getTodoList

} from '../selectors/';

export function mapStateToProps(state) {

return {

callSign: getTodoList(state)

};

}

export function mapDispatchToProps(dispatch) {

return {

onSetTodoList: () => dispatch(setTodoList())

};

}

export default connect(mapStateToProps, mapDispatchToProps)(Todo);

12) Now create a file in src/javascripts/thunks.js/index.js" with code:

import {

setTodoList,

} from './../actions/todos';

export function setTodoList() {

return ( dispatch, getState, api ) => {

return api

.getTodoList()

.then( ( res ) => {

dispatch( setTodoList( res ) );

return res;

} );

};

}

13) Now update in src/javascripts/apigateway/index.js" with code:

import $ from 'jquery';

export function ajaxErrorHandler( { responseJSON = {}, statusText } ) {

const message = responseJSON.message || statusText;

return Promise.reject( Error( message ) );

}

function getJSON( url ) {

return $.getJSON( url ).catch( ajaxErrorHandler );

}

const getTodos = () => getJSON( 'http://localhost:3000/todos' );

export default {

getTodos,

};

14) Now create a file in src/javascripts/components/todos/index.js" with code:

import React, { Component } from 'react';

import PropTypes from 'prop-types';

class Todos extends Component {

componentDidMount() {

this.props.onSetTodoList();

}

render() {

console.log(this.props.todos);

return (

<div className="App">

welcome to todos.

</div>

);

}

}

export default Todos;

Todos.propTypes = {

todos: PropTypes.array.isRequired,

onSetTodoList: PropTypes.func

}

15) Update Route.js file with follwings:

a) import container using code "import TodoContainer from './containers/todoContainer';".

b) add container on route using code "<Route exact path="/todos" component = { TodoContainer } />".

16) Replace src/javascripts/index.js" with code to register store:

**Todo.js**

import React from 'react';

import ReactDOM from 'react-dom';

import { Provider } from 'react-redux';

import createStore from './store/index';

import App from './App';

import './../assets/styles/style.css';

const store = createStore();

ReactDOM.render(

<Provider store={ store }>

<App />

</Provider>,

document.getElementById( 'root' )

);

Now Test your project will be run and you can show api data in console as well.

**Implement storybook in react app:-**

1) install the storybook package into our project using command “ **npx -p @storybook/cli sb init** ”

2) install bootsrap, react-bootsrap module.

3) some file will be add in src/storybook with dummy component.

4) To start storybook using command “**npm run storybook**”.

5) Replace code of main.js and try to create your own storybook.

module.exports = {

"stories": [

"../src/javascripts/\*\*/\*.stories.@(js|jsx|ts|tsx)"

],

"addons": [

"@storybook/addon-links",

"@storybook/addon-essentials"

]

}

6) Create storybook like:

import React, { Component } from 'react';

import PropTypes from 'prop-types';

class SingleTodo extends Component {

render() {

return (

<div className="post">

<h2 className="post\_title">{this.props.todo.title}</h2>

<p className="post\_message">{this.props.todo.description}</p>

<div className="control-buttons">

<button

className="edit btn btn-success"

onClick = { () => this.props.editTodo( this.props.todo ) }

>Edit</button>

<button

className="delete btn btn-danger"

onClick = { () => this.props.deleteTodo( this.props.todo.id ) }

>Delete</button>

</div>

</div>

);

}

}

export default SingleTodo;

SingleTodo.propTypes = {

deleteTodo: PropTypes.func,

editTodo: PropTypes.func,

todo: PropTypes.object.isRequired

}

**Todo.stories.js**

import React from 'react';

import { storiesOf } from '@storybook/react';

import { Container, Row, Col } from 'react-bootstrap';

import { action } from '@storybook/addon-actions';

import 'bootstrap/dist/css/bootstrap.min.css';

import './../../../assets/styles/style.css';

import Todo from './Todo';

const todo = { id: 1, title: 'title', description: 'desc' };

const actions = {

deleteTodo: action('deleteTodo'),

editTodo: action('editTodo'),

};

storiesOf('<Todo>', module)

.addDecorator(story => (

<Container>

<Row className="show-grid">

<Col md={12}>

{story()}

</Col>

</Row>

</Container>

))

.add('render todo detail data', () => <Todo todo={todo} {...actions} />);

please contact us and take full code from