React Bible

Important Libraries

Create React App:
Basic Folder Structure and scaffolding https://github.com/facebook/create-react-app

Styling, Template, Responsiveness:

1. Reactstrap https://reactstrap.github.io/
   Easy to use React Bootstrap 4 components
   Bootstrap 4 over Bootstrap 3 has many advantages with lot of new classes that helps development of styling easier.
   Always use Bootstrap react component based library instead of including cdn, because cdn has jquery and jquery uses its own way of manipulating DOM, this brings conflict on using with react.
2. Styled Components https://www.styled-components.com/

State Maintain:
Redux with Redux-thunk middleware

API / Ajax call:
Axios https://www.npmjs.com/package/axios

Routing in react:
React router dom 4 https://reacttraining.com/react-router/core

Unit test cases:

1. Jest https://jestjs.io/
2. Enzyme (assertion) https://airbnb.io/enzyme/

Linting:
Eslint with airbnb config https://www.npmjs.com/package/eslint-config-airbnb

Chrome Dev tools:
React dev tool
Redux dev tool

IDE / Editor: VS code https://code.visualstudio.com/

Infinite Scroll : React waypoint
Form: Redux Form, Formik

Important React Concepts

These following concepts are **most essential** to learn react.

- React component State & Lifecycle
- Handling Events
- Conditional Rendering
- Reconciliation
- Higher Order Components
- Commonly used Lifecycle Methods
    - ➔ ***render()***
    - ➔ ***componentDidMount()***
    - ➔ ***componentDidUpdate()***
    - ➔ ***componentWillUnmount()***
- Component Lifecycle Diagram

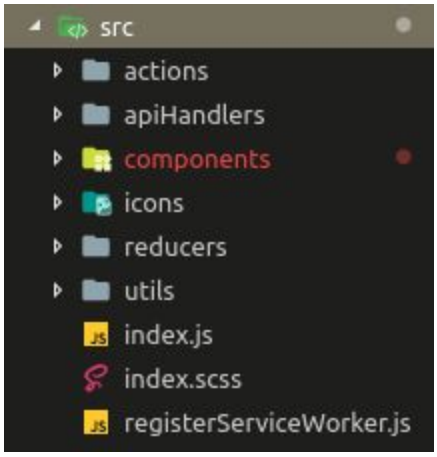
Checkout this for list of blogs for reference on these concepts.

Procedures & practices

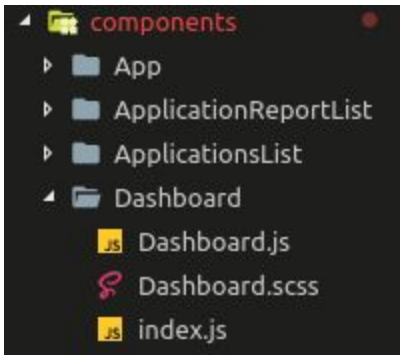
Folder Structure

- In react, every UI thinking should be in terms of components.
  Building components gives multiple advantages like reusability etc.

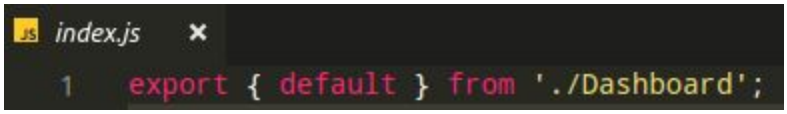
- **src/** folder should have following folder structure



- Each component should be written in a folder with corresponding component name,
  Each component can contain css/scss file with corresponding name.



- Component folder should contain one **index.js** file that exports main default component



- Simple React Component eg:

Dashboard.js

```
You, a few seconds ago | 3 authors (You and others)
1  import React, { Component } from 'react';
2  import './Dashboard.scss';
3
   You, a few seconds ago | 2 authors (You and others)
4  export class Dashboard extends Component {
5    render() {
6      return (
7        <div>
8          Hello world
9        </div>
10     );
11   }
12 }
13
14 export default Dashboard;
```

Browser Support:

- To add different browser support , we should install and import these packages in ***src/index.js*** file

```
import 'core-js/es6/map';
import 'core-js/es6/set';
import 'core-js/fn/array/find';
import 'core-js/fn/array/includes';
import 'core-js/fn/number/is-nan';
import 'raf/polyfill';
```

API calls

- All API calls should go through actions,
- A common Wrapper can be written that can take parameters and makes API call and returns promise.

Project Setup

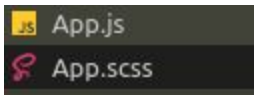
Scaffolding using create-react-app

- Create new project from [create-react-app]
```
➜  Private-Dir create-react-app hello-world

Creating a new React app in /home/qwinix/Documents/Private-Dir/hello-world.

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

SCSS setup

- Run "***npm eject***"
```
➜  hello-world git:(master) npm run eject

> hello-world@0.1.0 eject /home/qwinix/Documents/Private-Dir/hello-world
> react-scripts eject

? Are you sure you want to eject? This action is permanent. Yes
Ejecting...
```
  Ejecting the application gives multiple advantages, implementing Sass, setting up different environments, custom webpack config etc.

- Run ***npm i sass-loader node-sass --save***
```
➜  hello-world git:(master) ✗ npm install sass-loader node-sass --save
```

- Rename **App.css** to **App.scss** and in **App.js** too
```
import './App.scss';      You, a m
```


Eslint setup

- Eslint Airbnb has good and friendly eslint rules.
  Install following packages as dev dependencies.
  eslint
  eslint-config-airbnb
  eslint-plugin-import
  eslint-plugin-jsx-a11y
  eslint-plugin-react

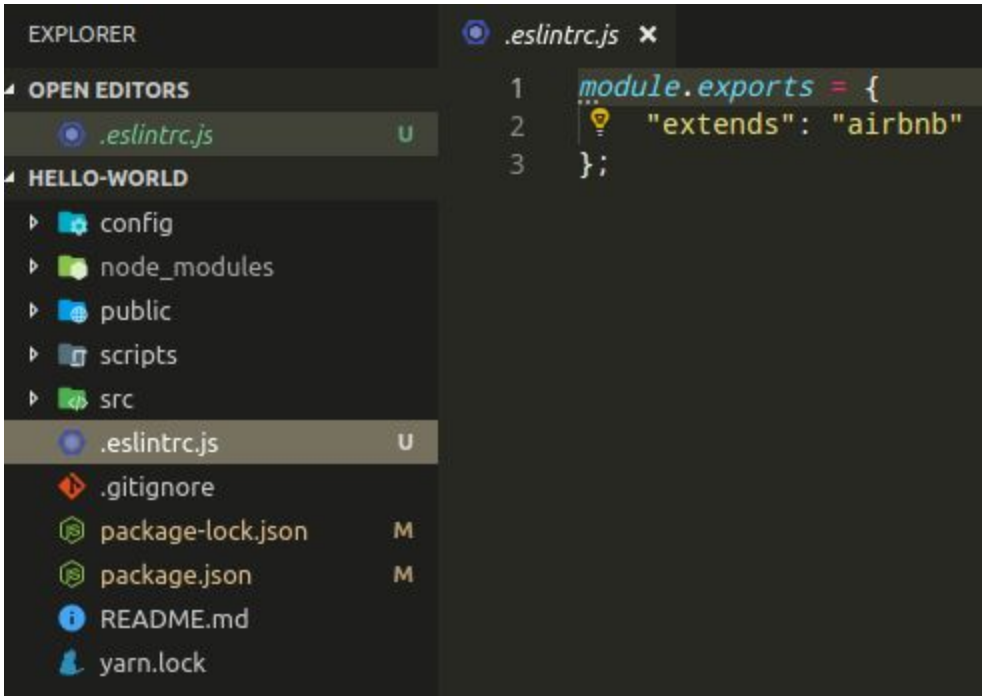
  npm i eslint eslint-config-airbnb eslint-plugin-import eslint-plugin-jsx-a11y eslint-plugin-react --save-dev

```
hello-world git:(master) npm i eslint eslint-config-airbnb eslint-plugin-import eslint-plugin-jsx-a11y eslint-plugin-react --save-dev
```

- Run ***./node_modules/.bin/eslint --init*** , and answer the questions by referring following image.

```
→ hello-world git:(master) ✗ ./node_modules/.bin/eslint --init
? How would you like to configure ESLint? Use a popular style guide
? Which style guide do you want to follow? Airbnb (https://github.com/airbnb/javascript)
? Do you use React? Yes
? What format do you want your config file to be in? JavaScript
Checking peerDependencies of eslint-config-airbnb@latest
The config that you've selected requires the following dependencies:

eslint-config-airbnb@latest eslint@^4.19.1 || ^5.3.0 eslint-plugin-import@^2.14.0 eslint
? Would you like to install them now with npm? Yes
```

- This provides ***eslintrc.js*** file in your project directory.



- Update this file with code below

```
module.exports = {
  extends: 'airbnb',
  env: {
    browser: true,
    jest: true,
  },
  rules: {
    indent: [ 'error', 2, { MemberExpression: 0 }, ],
    'react/jsx-filename-extension': [1, {
      extensions: ['.js', '.jsx'],
    }],
    'react/prefer-stateless-function': [{
      ignorePureComponents: true,
    }],
    'react/jsx-one-expression-per-line': false,
    'react/prop-types': [
      'enabled',
    ],
  },
};
```

- Create new file with name ***.eslintignore*** with following code

node_modules
build
src/registerServiceWorker.js
/build/**
/coverage/**
/docs/**
/jsdoc/**
/templates/**
/tmp/**
test.js
/config
/scripts
!.eslintrc.js

- Add this in ***"scripts"*** **section of** ***package.json***
  "eslint": "./node_modules/.bin/eslint .",
  "eslint-fix": "./node_modules/.bin/eslint . --fix"

```
"scripts": {
  "start": "node scripts/start.js",
  "build": "node scripts/build.js",
  "test": "node scripts/test.js",
  "eslint": "./node_modules/.bin/eslint .",
  "eslint-fix": "./node_modules/.bin/eslint . --fix"
},
```

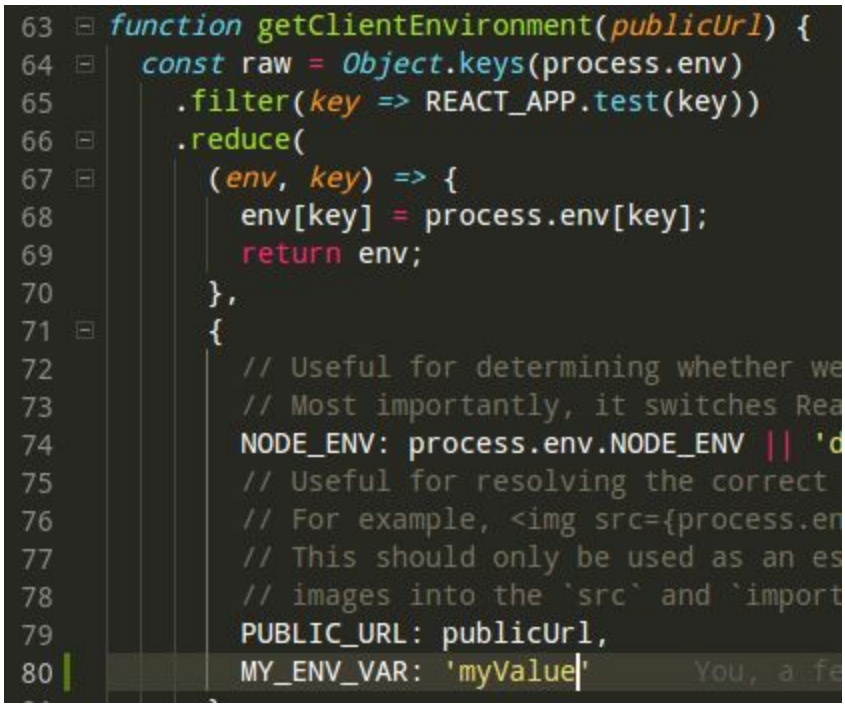
- **NOTE:** This eslint setup does not compile your source code if there are any linting errors. This can be disabled by commenting out ***eslintFormatter*** options in both ***webpack.config.dev.js*** and ***webpack.config.prod.js .***
  Always restart application when config files are changed, to take effect.

- (Optional) install ***eslint*** and ***editorconfig*** plugins in your editor (vscode)
  Create ***.editorconfig*** file in your project root path with following code.
  This helps in removing trailing spaces and fixing up project indentation on saving files

```
root = true
 [*]
charset = utf-8
end_of_line = lf
indent_size = 2
indent_style = space
max_line_length = 80
trim_trailing_whitespace = true
 [*.md]
max_line_length = 0
trim_trailing_whitespace = false
 [COMMIT_EDITMSG]
max_line_length = 0
```

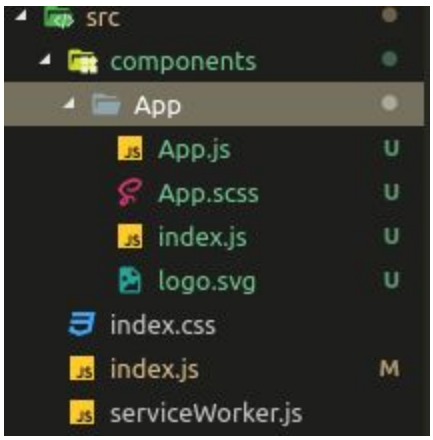
Environment variables setup

- The ***config/env.js*** can have environment variables setup, this can be used throughout the application as ***process.env.MY_ENV_VAR***



React Router Dom setup

- Have following folder structure setup. Move ***App.js, App.scss*** to ***components/App.*** Change things accordingly in index.js with respect to imported path.



- Run ***npm i react-router-dom --save ,*** add the following code in ***src/index.js***

```
import { Route, BrowserRouter } from 'react-router-dom';
const app = (
 <BrowserRouter basename={process.env.PUBLIC_URL}>
   <Route component={App} />
 </BrowserRouter>
);
ReactDOM.render(app, document.getElementById('root'));
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import { Route, BrowserRouter } from 'react-router-dom';
import App from './components/App';
import * as serviceWorker from './serviceWorker';

const app = (
  <BrowserRouter basename={process.env.PUBLIC_URL}>
    <Route component={App} />
  </BrowserRouter>
);

ReactDOM.render(app, document.getElementById('root'));
```

- The ***src/components/App/App.js*** should contain all routes

```
import React, { Component } from 'react';
import { Route, Switch } from 'react-router-dom';
import './App.scss';
import Home from '../Home';

class App extends Component {
  render() {
    return (
      <Switch>
        <Route exact path="/" component={Home} />
      </Switch>
    );
  }
}

export default App;
```

- The 404 component can be implemented like this, any path not mentioned will render this component.

```
<Switch>
  <Route exact path="/" component={Home} />
  <Route component={FourOhFourComponent} />
</Switch>
```

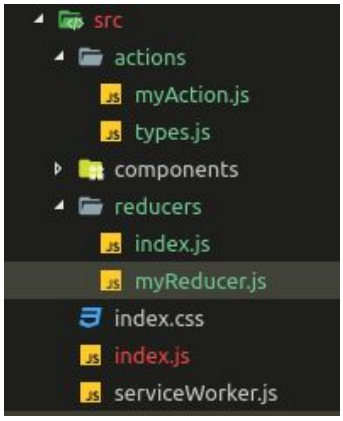
- The official documentation and tutorial
  https://reacttraining.com/react-router/core/guides/philosophy

React Redux setup

- Run ***npm i redux react-redux redux-thunk --save***

```
→ hello-world git:(master) npm i redux react-redux redux-thunk --save
```

- Create ***actions*** and ***reducers*** directories under ***src/*** and have following folder structure



- Next step is to create store and make it available for whole application.
  Add following to ***src/index.js***

```
import { Provider } from 'react-redux';
import { createStore, applyMiddleware, compose } from 'redux';
import ReduxThunk from 'redux-thunk';
import reducers from './reducers';
const composeEnhancers =
window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
const store = createStore(reducers, {},
composeEnhancers(applyMiddleware(ReduxThunk)));
const app = (
 <Provider store={store}>
  <BrowserRouter basename={process.env.PUBLIC_URL}>
   <Route component={App} />
  </BrowserRouter>
 </Provider>
);
```

```
/* eslint-disable no-underscore-dangle */
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import { Route, BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
import { createStore, applyMiddleware, compose } from 'redux';
import ReduxThunk from 'redux-thunk';
import reducers from './reducers';
import App from './components/App';
import * as serviceWorker from './serviceWorker';


const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
const store = createStore(reducers, {}, composeEnhancers(applyMiddleware(ReduxThunk)));
const app = (
  <Provider store={store}>
    <BrowserRouter basename={process.env.PUBLIC_URL}>
      <Route component={App} />
    </BrowserRouter>
  </Provider>
);

ReactDOM.render(app, document.getElementById('root'));
```

- The ***actions/types.js*** should contain all constants that are saved in ***store***
- Now the ***reducers/index.js*** should have ***combineReducers*** that combines all reducers.

index.js

```
1  import { combineReducers } from 'redux';
2  import myReducer from './myReducer';
3
4  export default combineReducers({
5    myReducer,
6  });
```

Components actions and reducers eg:

- Source code for following eg https://github.com/s8sachin/react-hello-world

- In this example let's have an input tag and on clicking button, it should send the value to redux store and display it.
- In your ***actions/myAction.js***

```
myAction.js
1  import { INPUT_VALUE } from './types';
2
3  export const submitValueAction = value => (
4    dispatch => dispatch({ type: INPUT_VALUE, payload: value })
5  );
```

This action should dispatch an object with ***type*** & ***payload.***
The payload value is dispatched to ***reducer***.

- Add this in ***reducers/myReducer.js***

```
myReducer.js
1  import { INPUT_VALUE } from '../actions/types';
2
3  const INITIAL_STATE = {
4    inputValueFromReducer: '',
5  };
6
7  export default (state = INITIAL_STATE, action) => {
8    switch (action.type) {
9    case INPUT_VALUE:
10     return { ...state, inputValueFromReducer: action.payload };
11   default:
12     return state;
13   }
14 };
```

The reducer has ***INITIAL_STATE*** defined where, and it checks for state change on ***dispatch*** from action.
However now we dispatched type ***INPUT_VALUE*** , the previous state is maintained the same through ***{ ...state }*** and the new value ***inputValueFromReducer*** is changed to ***action.payload***, the payload value dispatched from ***action.***

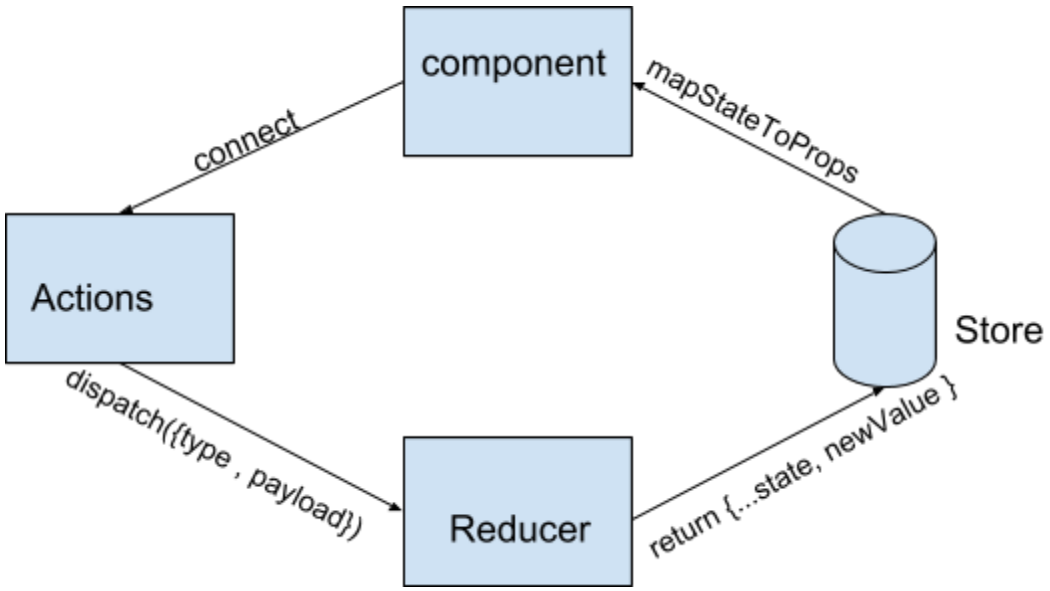
- Now it's the component that talks to action

Home.js

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


class Home extends Component {
  constructor(props) {
    super(props);
    this.state = { inputVal: '' };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(e) {
    this.setState({ inputVal: e.target.value });
  }

  handleSubmit() {
    const { inputVal } = this.state;
    this.props.submitValueAction(inputVal);
  }

  render() {
    const { inputVal } = this.state;
    const { inputValueFromReducer } = this.props;
    return (
      <div>
        <input value={inputVal} onChange={this.handleChange} /><br />
        <button type="button" onClick={this.handleSubmit}>Submit</button> <br />
        <span>Your value : {inputValueFromReducer}</span>
      </div>
    );
  }
}

const mapStateToProps = (state) => {
  const { inputValueFromReducer } = state.myReducer;
  return { inputValueFromReducer };
};

export default connect(mapStateToProps, { submitValueAction })(Home);
```

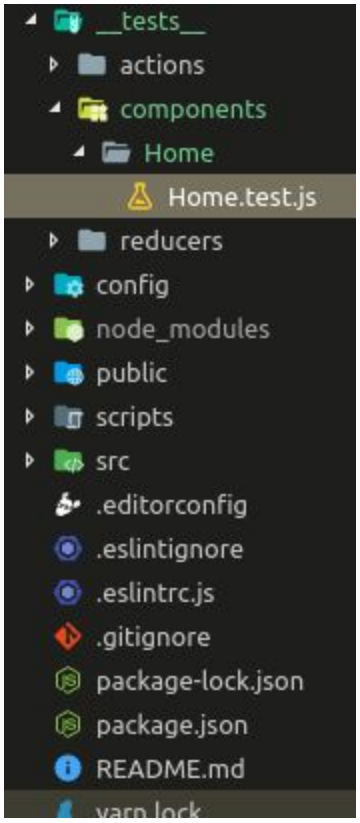
- This Home component has an input tag and button.
  The Component should use ***connect*** from ***react-redux*** to connect the component ***Home*** to the redux store.
  From above example, ***connect*** uses ***mapStateToProps***, this ***mapStateToProps*** defines the value that needs to be taken from the redux store. In this case, we are taking ***inputValueFromReducer***, this is the value that is being updated from reducer from previous step.
  Then connect ***submitValueAction*** action. This action is called ***onClick*** of submit button.
  Observe ***handleSubmit()*** function, here we are calling the ***submitValueAction*** and passing ***this.state.inputVal*** that currently holds value from input tag.
  Finally the value form redux store is obtained through ***this.props.inputValueFromReducer*** and can be used to display.

Basic Redux Flow



Jest & Enzyme Setup

- Add the ***__tests__*** folder should replicate test cases for ***src/*** directory.
  NOTE: The ***__tests__*** directory is outside ***src/*** is only in case of ***ejected*** application, If in case of ***non-ejected*** application, ***__tests__*** should be under ***src/ directory.***



- Do ***npm i enzyme enzyme-adapter-react-16 redux-mock-store enzyme-to-json***

```
hello-world git:(master) ✗ npm i enzyme enzyme-adapter-react-16 redux-mock-store enzyme-to-json
```

- In your ***package.json*** file change this under ***"jest"***

```
"testMatch": [
  "<rootDir>/__tests__/**/*.{js,jsx,ts,tsx}",
  "<rootDir>/?(*.)(spec|test).{js,jsx,ts,tsx}"
],
```

```
"collectCoverageFrom": [
  "!*/node_modules/**",
  "!<rootDir>/coverage/**",
  "!<rootDir>/config",
  "!*/public/**",
  "!*/scripts/**",
  "!*/src/registerServiceWorker.js",
  "**/*.{js,jsx}"
],
```

```
"coveragePathIgnorePatterns": [
  "/node_modules/",
  "<rootDir>/build/",
  "<rootDir>/scripts/",
  "<rootDir>/config/",
  "<rootDir>/public/",
  "<rootDir>/coverage/",
  "<rootDir>/.eslintrc.js",
  "<rootDir>/src/registerServiceWorker.js",
  "<rootDir>/src/index.js"
],
```

- Writing your component test cases.

  Home.test.js basic test case is to test whether component loads without crashing

Home.test.js

```
import React from 'react';
import { shallow, mount, configure } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';
import ReduxThunk from 'redux-thunk';
import configureStore from 'redux-mock-store';
import toJson from 'enzyme-to-json';
import Home from '../../../src/components/Home';

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

const middlewares = [ReduxThunk];
const mockStore = configureStore(middlewares);

describe('Home', () => {
  it('should load without crashing', () => {
    const store = mockStore({ myReducer: { inputValueFromReducer: '' } });
    const wrapper = mount(<Home store={store} />);
    expect(toJson(wrapper)).toMatchSnapshot();
  });
});
```

- Sample test case for action is as follows

myAction.test.js

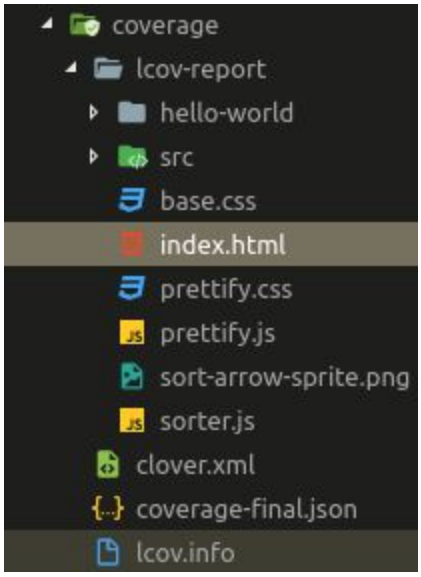
```
import { configure } from 'enzyme';
import configureStore from 'redux-mock-store';
import ReduxThunk from 'redux-thunk';
import Adapter from 'enzyme-adapter-react-16';
import { submitValueAction } from '../../src/actions/myAction';

const middlewares = [ReduxThunk];
const mockStore = configureStore(middlewares);
configure({ adapter: new Adapter() });

let store;
beforeEach(() => { // Runs before each test in the suite
  store = mockStore();
  store.clearActions();
});

describe('submitValueAction', () => {
  it('dispatch sample action', () => {
    store.dispatch(submitValueAction('testingValue'));
    const expected = [{ payload: 'testingValue', type: 'inputValue' }];
    expect(store.getActions()).toEqual(expected);
  });
});
```

- Sample reducer test case

myReducer.js

```
import myReducer from '../../src/reducers/myReducer';
import { INPUT_VALUE } from '../../src/actions/types';

describe('Reducers', () => {
  it('initial state is correct', () => {
    const action = { type: INPUT_VALUE, payload: 'dummyVal' };
    const INITIAL_STATE = {
      inputValueFromReducer: 'dummyVal',
    };
    expect(myReducer(undefined, action)).toEqual(INITIAL_STATE);
  });
});
```

- Run ***npm run test-coverage*** to get coverage
  This should generate ***coverages/*** directory.



Important references of best practices:

React Official Documentation: https://reactjs.org/docs/hello-world.html
React Official Tutorial: https://reactjs.org/tutorial/tutorial.html

Open Dota web: https://github.com/odota/web
Sound redux: https://github.com/andrewngu/sound-redux

Similar project structure and coding standards followed:
https://github.com/s8sachin/react-hello-world
https://github.com/s8sachin/grabbd-demo
https://github.com/s8sachin/dota2-react
https://github.com/s8sachin/unsplash-react

Blogs:
Best Practices
Render Props
Intro to Webpack
Folder structure
Mini patterns

Cool React based projects:
https://react.rocks/
https://www.opendota.com/
https://soundredux.io/#/