**Introduction**

I shall be talking about the tools that belong to something referred as the React ecosystem. It consists of number of very helpful tools such as Redux, Thunk and Reselect. It is a good idea to revise react basics, command-line operations and few command specific to Node.js. Software required for this project is a recent version of NPM. To install the most recent version of NPM type *npm install -g npm* in the command line and later check the version of it by typing *npm -v*.

I shall be using exercise file which are saved on my desktop. After opening up the terminal inside the required directory, I need to run *npm install* and hit Enter. After that, all I must do is run *npm run dev* for application to run successfully. Later on, during the course of my project, if I need to run server locally. After *npm install* command in that directory, I simply need to run *npm run start* and hit Enter. And the server should start up correctly.

My project will take one step further developers who have spent most of their time developing with Vanilla React and now aiming to manage a full scale React application.

**Reasons for using the React ecosystem**

This project is bit of learning curve for some developers. With Vanilla React/Basic React, when it comes to how I load, manipulate, and store that data I am displaying. The result is that all logic ends up getting crammed straight into components themselves. The major React Ecosystem tools are Redux (the purpose of this add-on is to help me manage the state of my React Application in an effective and bug-free way), Thunk (allows us to do is separate out the so-called side effects of my application), Reselect (its purpose is to abstract away the detail of how data is stored in the state) and styled components (gives us a nice way styling our components).

**Building a basic React project**

My aim is to dig a little deeper into the inner working of a React project then incorporate each of the ecosystem tools and concepts one by one in my project. The main things that I need is an index.htm file (this file will be sent to the client and what React will render my app into), modern JavaScript syntax (ES6), webpack (use things like CSS modules to style my app), root component (container that holds the rest of my application components) and finally react-hot-loader (allows me to see the changes in app without having to refresh every time).

I am starting my project by creating a directory (React Ecosystems) that will hold my React project and run *npm init -y* inside this directory to create *package.json* file. Once I have done that, I shall create two directories (public and src) inside of my root directory. Next thing that I am going to do is create index.html file inside the public directory.

I am going to support my React code with ES6 syntax. In order to do all this, I shall be running *npm install –save-dev @babel/core @babel/cli @babel/preset-env @babel/preset-react* then hit Enter. The next thing I need to do is create a *.babelrc* file to tell Babel transpiler what presets/plugins to use to transpile my code. To build and server my project, I need to create three files inside of my source folder. The first file is going to be index.js (this contains the code that inserts in my react app in index.html page of the project), App.js and App.css for code and styling respectively. And finally, run *npm install react react-dom* as dependencies of the project.

Once I can setup a simple React app, I am interested in building webpack to build my app. I shall open a terminal window inside the react-ecosystems directory and *type npm install –save-dev webpack webpack-cli webpack-dev-server style-loader css-loader and babel-loader* and hit enter. Then I add *webpack.config.js* to the root directory.

After getting React app up and running, my problem was to make a change to my code in app.js component and then seeing the changes without physically refreshing my browser. I solved this problem by running *npm install –save-dev react-hot-loader.* Next, I want to open my *package.json* file and create two more scripts “*build*” and “*dev*”.

Earlier, I talked about building up a working React app from scratch. I am building a basic to-do app. Its purpose is to keep track of things that I want to get done. My plan is to build the very basics of this app and later I add more and more functionality to it by using ecosystem pieces.

First component in my project is the *TodoList* component. For each *todo*, I am going to display a *TodoListItem* component. Next task it to implement my *TodoListItem* component to display the text of the individual todos. I am also displaying two buttons (‘Mark as Completed’ and ‘Remove’) inside their own div for styling purpose. The next and final component is *TodoForm* component. Here, I create an input box and keep track of the value in the input box.

At this stage, I have completed *TodoList*, *TodoListItem* and *TodoFrom* components. First thing, I must do is open app.js component and replace ‘Hello’ with to-do list that I created. I also add reference in imports for other files.

**Why do I use Redux?**

I am looking at my tool called Redux. I want to take forward the idea of global state and solve the problems in state management by adding some strict rules along with organization with it. That is what Redux does. One of the core concepts related to Redux is that I have one central global state called the store. All my components can access the store (the big json object). The store basically contains anything that I load from the server. There are two other pieces of Redux flow (Redux actions and reducers). The key point at this stage is that my components only allow to make changes to the state by triggering the predefined actions and only changes allowed are the corresponding changes that I specify in my reducers.

Now, I talk about adding Redux to my project. I install Redux by command *npm install redux and react-redux* then hit Enter. Next thing I am going to do is create *store.js* file to put logic for setting up my Redux store. The next step is to open my *index.js* file and wrap my whole app inside the *Redux provider*.

**Creating REMOVE\_TODO AND CREATE\_TODO Redux actions**

Once I have basic setup to add Redux in my project. I am adding actions and reducers to keep track of my project state in the *actions.js* file. Now I am defining a reducer this will keep track of the state of the *todos* in my project according to my component trigger actions. So, I am creating a new file *reducers.js* in *todos* directory and referring it in *store.js* file as well. Here I use a switch block to make the *todos* reducer respond correctly to my *CREATE\_TODO* and *REMOVE\_TODO* actions.

The next step is to give the components to the Redux store by using higher order *connect* function. I shall start with *NewTodoForm* creating *mapStateToProps* and *mapDispatchToProps* as parameters of connect. Later, adding some functionality to the *onClick* method of button to keep track of my input box. Similarly, I connect my *TodoList.js* and *TodoListItem.js* to the Redux store. Here, I am going to take care of remove button and then making the completed button work later during the project. At this stage, I test my application to see if it’s working correctly.