## React Key Concepts

### The Birth of React.js

Previously we have **jQuery** which allows developer easily interact with the DOM across all the browsers.

Then the library like **backbone.js** came up because our JavaScript files gets bigger and bigger it helps to organize this JavaScript files.

**SPA**

Because it becomes easier and easier to work with DOM and also **Ajax** came, we have the birth of **SPA.** Basically we focus less on HTML and lot more on JavaScript; we only load the application code once instead of making new request to the server where it return new document instead now our application acted more like desktop application where we stay on the same page the entire time and the js file simply changes or update the html file/DOM to display new things

In 2010 **Angular.js** which created by google really became a standard way of building the application. For that we have better container where things like controller, views and models and this idea of MVC where each JavaScript files is divided into different things it did.

Since the App got bigger and bigger we have more complexity, using Angular.js is very tough to find the bug and the data is flowing everywhere. In 2013 Facebook releases **react** and the solution is really good and 2014 angular.js completely rewrite to **angular**

### React Concepts:-

* **Don’t touch the DOM, I will do it.**
* **Building website like lego blocks. (Reusable component).**
* **Unidirectional Data Flow.**
* **UI, The rest is up to you.**

#### Declarative vs Imperative

* + **Imperative**

Many existing framework and library before react where directly manipulating the DOM on every page and this way of programming we referred as **Imperative** as we directly change the individual parts of the app in response to various user events. **Problem** is it becomes difficult to see the relationship between events and the edge cases.

* + **Declarative**

Instead of Imperative react came up with concept called **declarative**, as we know DOM manipulation leads to performance issue, it takes a long time. So react say just declare the how the app looks like (give me the state), I will do everything like find the best way to use the DOM. i.e. it contains one JavaScript object that describe how the app should look like

This resulted in:-

1. Less complexity
2. Better code quality
3. Faster Developer time

#### Component Architecture

React is all about where we have multiple components surrounded one another and we can also reuse them in other places/inside another component.

Component is nothing but a JavaScript function that receive some input/props and return an html but from a JavaScript.

|  |
| --- |
|  |

#### One Way Data Flow

Means if any state changes it will from top to bottom it can never go from bottom to top (means it can never go up).

One way data flow is a restriction because of this it easier to understand how the app works.

|  |
| --- |
|  |

#### UI Library

React care about component, virtual DOM and I am only going to work with the view, everything else we need can use other module, library mix and match to the need.

React 360:-<https://facebook.github.io/react-360/>

React blessed: - <https://github.com/Yomguithereal/react-blessed>

React Desktop: - <https://reactdesktop.js.org/>

#### How to be a great react Developer

|  |
| --- |
|  |

## React Basics

### Creating the react app

<https://create-react-app.dev/docs/getting-started/>

* + Old way

npm install –g create-react-app

This is the package which creates a startup project that contains webpack, babel, lint and debugging.

Create-react-app <project name>

* + New ways is

First uninstalled the create-react-app: **npm uninstall -g create-react-app**

Then type the below command with project name

**npx create-react-app <app-name>**

<https://create-react-app.dev/docs/getting-started/>

### Understanding project structure and files

* + **In Package.json**
* **react-scripts library** it helps us not to worry about webpack and babel.
* **react library** helps to write html like syntax inside a JavaScript file.
* **react-dom library** helps to interact with the DOM.

|  |
| --- |
| {  "name": "monsters-rolodex",  "version": "0.1.0",  "private": true,  "dependencies": {  "@testing-library/jest-dom": "^4.2.4",  "@testing-library/react": "^9.5.0",  "@testing-library/user-event": "^7.2.1",  "react": "^16.13.0",  "react-dom": "^16.13.0",  "react-scripts": "3.4.0"  },  "scripts": {  "start": "react-scripts start",  "build": "react-scripts build",  "test": "react-scripts test",  "eject": "react-scripts eject"  },  "eslintConfig": {  "extends": "react-app"  },  "browserslist": {  "production": [  ">0.2%",  "not dead",  "not op\_mini all"  ],  "development": [  "last 1 chrome version",  "last 1 firefox version",  "last 1 safari version"  ]  }  } |

* In **source folder** where we write all the react code/files for the application
* In **public folder** where the browser needs the older version of JavaScript and html to understand and the also the build files will be present.
* **npm start** to start the project.
* **npm run build** to build the file for the production.
* **npm run eject** it will show all the webpack , test config files. But don’t it as the files changing/updating is doing by react team member.

## React.js

Before React, jQuery was their which leads to inconsistency are totally sequentially. React can be used in mobile devices; virtual reality apps, desktop apps anywhere have a view react can use it.

React divide everything in small component, which at the end combines to create the website.

And Components can be reusable.

|  |
| --- |
|  |

### Data flow in react is top-bottom only

|  |
| --- |
|  |

### Virtual DOM

As we know always need to minimize the amount of DOM manipulation, because it reduces the performance of the web and also increases bug.

With React it creates the virtual DOM. Virtual DOM is just a JavaScript object and this object just describes the current state of the website.

We just give the object to react and the react bot will automatically make changes to the DOM in the most optimum way possible.

### Eco-System

React has great ecosystem. It is probably one of the biggest ecosystems in JavaScript

|  |
| --- |
|  |

### create-react-app

npm install –g create-react-app

This is the package which creates a startup project that contains webpack, babel, lint and debugging.

### Creating the new project

Create-react-app <project name>

New ways is

npx create-react-app my-app

<https://create-react-app.dev/docs/getting-started/>

### Components

Component can be creating using function as well with class. Reason why we use class because react has given ability to write more functionality verses the function return html

* **Function way**

**In functional component we can’t have state, lifecycle hooks only return html code**

Step 1:- Create a js file

1. It needs to import React and component.
2. Create function which return the html
3. Export the function.

Adding CSS file to the component.

1. Create a new css file and import it in the component

|  |
| --- |
| import React from 'react';  import logo from './logo.svg';  import './App.css';  function App() {  return (  <div className="App">  <header className="App-header">  <img src={logo} className="App-logo" alt="logo" />  <p>  Edit <code>src/App.js</code> and save to reload.  </p>  <a  className="App-link"  href="https://reactjs.org"  target="\_blank"  rel="noopener noreferrer"  >  Learn React  </a>  </header>  </div>  );  }  export default App; |

Step 2:- How to use in other file.

All remember in React application we will have React package as well as ReactDOM to connect react to the DOM

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import App from './Hello';  import \* as serviceWorker from './serviceWorker';  ReactDOM.render(<App />, document.getElementById('root')); |

* **Class way**

**In Class component we can have state, lifecycle hooks**

Step 1:- Create a js file

1. It needs to import React and component.
2. Create class which extends Component.
3. Export the class.

Adding CSS file to the component.

1. Create a new css file and import it in the component

Hello.js

|  |
| --- |
| import React, {Component} from 'react';  import './App.css';  class App extends Component {  render() {  return (  <div>  <p>Welcome to React</p>  <h1>Hello World</h1>  </div>  );  }  }  export default App; |

Step 2:- How to use in other file.

All remember in React application we will have React package as well as ReactDOM to connect react to the DOM

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import App from './App';  import \* as serviceWorker from './serviceWorker';  ReactDOM.render(<App />, document.getElementById('root')); |

### Tachyons

npm install tachyons

Tachyons allows us to have similar to bootstrap, predefined class names that we just add to components to make things look really nice.

<https://tachyons.io/>

After install just import in the component “import tachyons” basically in index.js

Why className because this html code is called as JSX. It allows writing Html syntax in JavaScript.

And always return one element otherwise it will give error.

|  |
| --- |
| import React, {Component} from 'react';  import './Hello.css';  class Hello extends Component {  render() {  return (  <div className="f1 tc">  <p>Welcome to React</p>  <h1>Hello World</h1>  </div>  );  }  } |

### Props

It allows send data from parent to child using string interpolation.

Allows remember all JavaScript code should put inside {} in JSX code

**In Index.js (parent)**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import Hello from './Hello';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  ReactDOM.render(<Hello greeting={"Hello" + "How are you"}/>, document.getElementById('root')); |

Now this greeting can be used in child component in Hello

**In Hello.js (Child)**

|  |
| --- |
| import React, {Component} from 'react';  import './Hello.css';  class Hello extends Component {  render() {  return (  <div className="f1 tc">  <p>Welcome to React</p>  <h1>{ this.props.greeting}</h1>  </div>  );  }  } |

How to convert the component class to function?

|  |
| --- |
| import React, {Component} from 'react';  import './Hello.css';  const Hello =(props) =>{  return (  <div className="f1 tc">  <p>Welcome to React</p>  <h1>{props.greeting}</h1>  </div>  );  }  export default Hello; |

### Export without default

Export without default can have multiple exports.

|  |
| --- |
| export const robots = [  {  id: 1,  name: 'Leanne Graham',  username: 'Bret',  email: 'Sincere@april.biz'  },  {  id: 2,  name: 'Ervin Howell',  username: 'Antonette',  email: 'Shanna@melissa.tv'  }  ] |

While importing it other file it should be in {} so that we can pass multiple export

import {robots} from './robots';

### How to use template string in the component

**In child component**

|  |
| --- |
| import React from 'react';  const Card = (props) => {  return (  <div className="tc bg-light-green dib br3 pad3 ma2 grow bw2 shadow-5">  <img alt="robots" src={`https://robohash.org/${props.id}?200x200` }/>  <div>  <h2>{props.name}</h2>  <p>{props.email}</p>  </div>  </div>    );  }  export default Card; |

**In Parent component How we are passing to child component**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import Card from './Card';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import {robots} from './robots';  ReactDOM.render(  <div>  <Card id={robots[0].id} name={robots[0].name} email={robots[0].email}/>  <Card id={robots[1].id} name={robots[1].name} email={robots[1].email}/>  <Card id={robots[2].id} name={robots[2].name} email={robots[2].email}/>  <Card id={robots[3].id} name={robots[3].name} email={robots[3].email}/>  </div>  , document.getElementById('root')); |

### How to use Destructuring we can do something nice

|  |
| --- |
| import React from 'react';  const Card = (props) => {  const {name, email,id} = props;  return (  <div className="tc bg-light-green dib br3 pad3 ma2 grow bw2 shadow-5">  <img alt="robots" src={`https://robohash.org/${id}?200x200` }/>  <div>  <h2>{name}</h2>  <p>{email}</p>  </div>  </div>    );  }  export default Card; |

Another way is to

|  |
| --- |
| import React from 'react';  const Card = ({name, email,id}) => {  return (  <div className="tc bg-light-green dib br3 pad3 ma2 grow bw2 shadow-5">  <img alt="robots" src={`https://robohash.org/${id}?200x200` }/>  <div>  <h2>{name}</h2>  <p>{email}</p>  </div>  </div>    );  }  export default Card; |

Imp:- As of their was lot of copy and paste in parent component , as we know in One Way Data Flow we always had a parent such as app component that has children.

Using this knowledge we can have one big component that have different children, so that we make component small and reusable.

### How to loop list using map

|  |
| --- |
| import React from 'react';  import Card from './Card';  const Cardlist = ({robots}) => {  const CardComponent = robots.map((user,i) => {  return <Card id={robots[i].id} name={robots[i].name} email={robots[i].email}/>  })  return(  <div>  {CardComponent}  </div>  );  }  export default Cardlist; |

And also remember when we are doing loop we should give some unique key

|  |
| --- |
| import React from 'react';  import Card from './Card';  const Cardlist = ({robots}) => {  const CardComponent = robots.map((user,i) => {  return (  <Card  key={i}  id={robots[i].id}  name={robots[i].name}  email={robots[i].email}  />  );  })  return(  <div>  {CardComponent}  </div>  );  }  export default Cardlist; |

### State

How to communicate child component with another child of different node?

At first child will communicate with parent then parent will communicate with the child with another node.

**State** is simply a description of APP or Simply an Object. An object that describes our application. It can be changed and affect our app. **basically it changes because of user interaction**

**Props** are simply thing that come out of state. It’s never changes and it just input we get and never modify them. **Things we are passing to other component**

**State turns into props when it passed down to other child component.**

**For State declaration we need declare inside constructor and also here we declare inside a class as constructor is required. These state properties can change and affect the App. And usually state is present in the parent component so that it can passes state to different components.**

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from './SearchBox';  import Cardlist from './Cardlist';  import {robots} from './robots';  class App extends Component {  constructor(){  super();  this.state = {  robots: robots,  searchfield: ''  }  }  render(){  return (  <div className="tc">  <h1>Robo Friends</h1>  <SearchBox />  <Cardlist robots = {robots}/>  </div>  )  }  }  export default App; |

### How child communicate to parent

**Using Event we are communicate with parent and using props parent send data to child**

Here every time we type something the onChange will call the searchChange function

|  |
| --- |
| import React from 'react';  const SearchBox = (searchChange) => {  return (  <div className="pa2">  <input  className="pa3 ba b-green bg-lightest-blue"  type="search"  placeholder="Search Robots"  onChange = {searchChange}  />  </div>  );  }  export default SearchBox; |

SearchChange is declare in parent component

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from './SearchBox';  import Cardlist from './Cardlist';  import {robots} from './robots';    class App extends Component {  constructor(){  super();  this.state = {  robots: robots,  searchfield: ''  }  }  onSearchChange(event){  console.log(event.target.value);  }  render(){  return (  <div className="tc">  <h1>Robo Friends</h1>  <SearchBox searchChange = {this.onSearchChange} />  <Cardlist robots = {this.state.robots}/>  </div>  )  }  }  export default App; |

### How to update in React (setState)

**setState() is an asynchronous function call**

<https://reactjs.org/docs/react-component.html#setstate>

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from './SearchBox';  import Cardlist from './Cardlist';  import {robots} from './robots';    class App extends Component {  constructor(){  super();  this.state = {  robots: robots,  searchfield: ''  }  }  onSearchChange =(event) => {  this.setState({searchfield: event.target.value})  const filteredRobots = this.state.robots.filter(robot => {  return robot.name.toLowerCase().includes(this.state.searchfield.toLowerCase);  });  console.log(filteredRobots);  }  render(){  return (  <div className="tc">  <h1>Robo Friends</h1>  <SearchBox searchChange = {this.onSearchChange} />  <Cardlist robots = {this.state.robots}/>  </div>  )  }  }  export default App; |

**As setState() is an asynchronous function so if you want to see the right after setting, we have to do it inside of the second argument provide in setState() function.**

|  |
| --- |
| render() {  return (  <div className="App">  <input type="search" placeholder="Search Monster"  onChange={e => this.setState({ searchField: e.target.value }, () => {  console.log(this.state.searchField);  })} />  <CardList monsters={this.state.monsters} />  </div>  );  }  } |

**Whenever we create a function inside a class component always need to create with arrow function, otherwise it will give error**

|  |
| --- |
| onSearchChange =(event) => {  this.setState({searchfield: event.target.value});  } |

**If State or props is updating in the setState.**

**Anytime if you want to update state and the props or state we need to use with in the update, then we use the syntax of function instead of object and instead of this. State directly we called using previous state.**

|  |
| --- |
| class App extends Component {  constructor() {  super();  this.state = {  meaningOfLife: 47  }  }  handleClick = () => {  this.setState({meaningOfLife : this.state.meaningOfLife + 1},  () => console.log(this.state.meaningOfLife)  )  } |

**Instead**

|  |
| --- |
| class App extends Component {  constructor() {  super();  this.state = {  meaningOfLife: 47  }  }  handleClick = () => {  this.setState((prevState, prevProps) => {  return { meaningOfLife: prevState.meaningOfLife + 1 }  },  () => console.log(this.state.meaningOfLife)  )  } |

### Events

React events does not manually touch the DOM instead if any event is changed it improve to react via react robot, because we are using JSX react has something called synthetic Event basically a fake event that it pretend to be a DOM event.

And also synthetic event will be camel case like: **onClick , onChange etc.**

<https://reactjs.org/docs/events.html#keyboard-events>

### Life Cycle Hooks

Methods come with react, and they life cycle hooks because it automatically trigger on app when loaded in browser. We can just put the methods into a class component they will automatically get triggered.

<http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>

|  |
| --- |
|  |

1. Mounting :-

When refresh the website the app component gets mounted into the document.getElementById(‘root’)

ReactDOM.render(<App />, document.getElementById('root'));

If we check the index.html file the webpage is nothing but a div with id of root.

<div id="root"></div>

When we a mount a component we are replacing the above div adding our app component. Mounting is the start of the app.

Mounting has four function defined, and also order is same as defined below:-

* + [constructor()](https://reactjs.org/docs/react-component.html#constructor)
  + [static getDerivedStateFromProps()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
  + [render()](https://reactjs.org/docs/react-component.html#render)
  + [componentDidMount()](https://reactjs.org/docs/react-component.html#componentdidmount)

1. Updating

Whenever a component changes. It has below function defined:

* + [static getDerivedStateFromProps()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
  + [shouldComponentUpdate()](https://reactjs.org/docs/react-component.html#shouldcomponentupdate)
  + [render()](https://reactjs.org/docs/react-component.html#render)
  + [getSnapshotBeforeUpdate()](https://reactjs.org/docs/react-component.html#getsnapshotbeforeupdate)
  + [componentDidUpdate()](https://reactjs.org/docs/react-component.html#componentdidupdate)

1. UnMounting

Whenever a component is removed from a DOM or page.

* + [componentWillUnmount()](https://reactjs.org/docs/react-component.html#componentwillunmount)

### Fetching from JSON placeholder (fetch)

Which we will discuss more briefly in next section. And also fetch is part of window object it comes with browser, it is tool to make request to server.

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from './SearchBox';  import Cardlist from './Cardlist';  import './App.css';    class App extends Component {  constructor(){  super();  this.state = {  robots: [],  searchfield: ''  }  }  componentDidMount(){  fetch('https://jsonplaceholder.typicode.com/users')  .then(response =>{  return response.json();  })  .then(users =>{  this.setState({robots: users});  })    }  onSearchChange =(event) => {  this.setState({searchfield: event.target.value});  }  render(){    const filteredRobots = this.state.robots.filter(robot => {  return robot.name.toLowerCase().includes(this.state.searchfield.toLowerCase());  });  return (  <div className="tc">  <h1 className="f1">Robo Friends</h1>  {console.log(filteredRobots)}  <SearchBox searchChange = {this.onSearchChange} />  <Cardlist robots = {filteredRobots}/>  </div>  )  }  }  export default App; |

### Children

React can use to create component for functionality also like scrolling

As of now we see only self closing custom component. But we didn’t see component that wraps another component, then how we are going talk with the wrap component. There come the children, every props has children define.

|  |
| --- |
| class App extends Component {  constructor(){  super();  this.state = {  robots: [],  searchfield: ''  }  }  componentDidMount(){  fetch('https://jsonplaceholder.typicode.com/users')  .then(response => response.json())  .then(users => this.setState({robots: users}));    }  onSearchChange =(event) => {  this.setState({searchfield: event.target.value});  }  render(){    const filteredRobots = this.state.robots.filter(robot => {  return robot.name.toLowerCase().includes(this.state.searchfield.toLowerCase());  });  if(this.state.robots.length === 0){  return <h1>Loading</h1>  }  else {  return (  <div className="tc">  <h1 className="f1">Robo Friends</h1>  {console.log(filteredRobots)}  <SearchBox searchChange = {this.onSearchChange} />  <Scroll>  <Cardlist robots = {filteredRobots}/>  </Scroll>  </div>  );  }  }  }  export default App; |

In scroll component

|  |
| --- |
| import React from 'react';  const Scroll = (props) => {  return (  <div style={{overflowY:'scroll', border:'1px solid black',height:'500px'}}>  {props.children}  </div>  )  };  export default Scroll; |

### Inline style in the custom component inside JSX

Use {{}}

|  |
| --- |
| import React from 'react';  const Scroll = (props) => {  return (  <div style={{overflowY:'scroll', border:'1px solid black',height:'500px'}}>  {props.children}  </div>  )  };  export default Scroll; |

### Pure component vs smart component

Component without state is termed as pure component.

Smart component is the component where state, life cycle hooks present that has the class syntax.

### Folder (Components and Containers)

All pure components will be present in components folder

All smart components will be present in containers folder.

### Keeping the react projects up to date

Always remember to update the package.json file with the respective version.

npm audit fix :- it will try to audit and fix the some version issue

npm audit:- it will list out all the vulnerabilities

npm audit fix --force – it will update all the package with latest version

### Error Boundary

Error Boundary we need react version 16 or higher. Using **componentDidCatch()** predefined life cycle hooks method.

Error Boundary when we put our code in production, when we don’t have the details error in development mode.

**Create the component for Error to catch**

componentDidCatch :- Act as a try catch block in JavaScript

|  |
| --- |
| import React, { Component } from 'react';  class ErrorBoundary extends Component{  constructor(props){  super(props);  this.state = {  hasError: false  }  }  componentDidCatch(error,info){  this.setState({hasError:true});  }    render(){  if(this.state.hasError){  return <h1>Error Occured.Please contact support team</h1>  }  return this.props.children  }  }  export default ErrorBoundary; |

**How to use the ErrorBoundary component in the other component to catch error**

**In app.js**

Here below if any error in cardlist component it catches error and display error message

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from '../Components/SearchBox';  import Cardlist from '../Components/Cardlist';  import Scroll from '../Components/Scroll';  import './App.css';  import ErrorBoundary from '../Components/ErrrorBoundary';  class App extends Component {  constructor(){  super();  this.state = {  robots: [],  searchfield: ''  }  }  componentDidMount(){  fetch('https://jsonplaceholder.typicode.com/users')  .then(response => response.json())  .then(users => this.setState({robots: users}));    }  onSearchChange =(event) => {  this.setState({searchfield: event.target.value});  }  render(){  const {robots,searchfield} = this.state;  const filteredRobots = robots.filter(robot => {  return robot.name.toLowerCase().includes(searchfield.toLowerCase());  });  return !robots.length ?  <h1>Loading</h1> :  (  <div className="tc">  <h1 className="f1">Robo Friends</h1>  {console.log(filteredRobots)}  <SearchBox searchChange = {this.onSearchChange} />  <Scroll>  <ErrorBoundary>  <Cardlist robots = {filteredRobots}/>  </ErrorBoundary>  </Scroll>  </div>  );  }  }  export default App; |

### How to fetch data from backend

Here ComponentDidMount() is the life cycle hooks in React

|  |
| --- |
| componentDidMount() {  fetch('http://localhost:3008')  .then(response => response.json())  .then(data => console.log(data));  } |

### Submitting a post request to backend with Request headers and request body

|  |
| --- |
| onSubmitSignIn = () => {  fetch('http://localhost:3008/signin',{  method:'post',  headers: {'Content-Type':'application/json'},  body: JSON.stringify({  email: this.state.signInEmail,  password: this.state.signInPassword  })  })  .then(response => response.json())  .then(data =>{  if(data === 'success'){  this.props.onRouteChange('home')  }  });  } |

### Deploying our React App

Via GitHub pages

<https://create-react-app.dev/docs/deployment/#github-pages>

### Setting up SASS in React Project

Sass is the most mature, stable, and powerful professional grade CSS extension language in the world.

Great things of Create-react-app is it actually comes with the configuration for SASS. Basically it configure webpack and babel to convert the SASS file into CSS because the browser can only CSS files

Step 1:- Add node sass to the project

npm install node-sass

### Routing

In SPA application we request the page and all the navigation happens in the JS because we no need to talk to the server any more. The JavaScript files or React library allow us to manipulate the DOM, so the server is mainly focusing on what data to send instead of focusing on what view or html need to send.

React don’t come with any prebuild routing, unlike Angular which is framework with prebuild routing.

In React we can user library or implements on our own. Here we are going to use **React-Router**

#### React Router DOM

<https://reacttraining.com/react-router/web/guides/quick-start>

Step 1 : - Install the react router DOM

npm install react-router-dom

Step 2:- Import **BrowerRouter** from react-router-dom and then place the component between BrowserRouter

|  |
| --- |
| import React from "react";  import ReactDOM from "react-dom";  import { BrowserRouter } from "react-router-dom";  import "./index.css";  import App from "./App";  ReactDOM.render(  <BrowserRouter>  <App />  </BrowserRouter>,  document.getElementById("root")  ); |

Step 3:- Import the **Router** from react-route-dom in the component where we want to map the route.

Here exact is if the path matches then only it should give, bydefault it is true.

|  |
| --- |
| import React from "react";  import { Route } from "react-router-dom";  import "./App.css";  import HomePage from "./pages/homepage/homepage.component";  function App() {  return (  <div>  <Route exact path="/home" component={HomePage} />  </div>  );  }  export default App; |

Another component called **switch** which is used to return the first match in a route

|  |
| --- |
| import React from "react";  import { Switch, Route } from "react-router-dom";  import "./App.css";  import HomePage from "./pages/homepage/homepage.component";  function App() {  return (  <div>  <Switch>  <Route exact path="/" component={HomePage} />  <Route exact path="/Hello" component={Hello} />  </Switch>  </div>  );  }  export default App; |

#### Link

Link component is the special component that react-router-dom gives us to dynamically pass in.

Basically it is used not to load the page, as if you use href it will load the page. As react is used for SPA.

|  |
| --- |
|  |

#### History Props

Other way of doing navigation instead of link, but it gives us more dynamic access basically functionality we can control when we want this.

|  |
| --- |
|  |

#### Location Props

It basically tells us where we are currently in the page. Basically in point we need the pathname

#### Nested Route

For nested route we use the match props. Below code explain everything in the routing

|  |
| --- |
| import React from 'react';  import { Route, Link } from 'react-router-dom';  import './App.css';  const HomePage = props => {  console.log(props);  return (  <div>  <button onClick={() => props.history.push('/topics')}>Topics </button>  <h1>HOME PAGE</h1>  </div>  );  };  const TopicsList = props => {  return (  <div>  <h1>TOPIC LIST PAGE</h1>  <Link to={`${props.match.url}/13`}>TO TOPIC 13</Link>  <Link to={`${props.match.url}/17`}>TO TOPIC 17</Link>  <Link to={`${props.match.url}/21`}>TO TOPIC 21</Link>  </div>  );  };  const TopicDetail = props => {  return (  <div>  <h1>TOPIC DETAIL PAGE: {props.match.params.topicId}</h1>  </div>  );  };  function App() {  return (  <div>  <Route exact path='/' component={HomePage} />  <Route exact path='/blog/asdqw/topics' component={TopicsList} />  <Route path='/blog/asdqw/topics/:topicId' component={TopicDetail} />  <Route exact path='/blog/topics' component={TopicsList} />  <Route path='/blog/topics/:topicId' component={TopicDetail} />  </div>  );  }  export default App; |

#### WithRouter Component

Basically for getting the value of history and match for the nested component we user WithRouter

It’s a higher order component and higher order component is essentially a function that a component as an argument and returns us a modified component

|  |
| --- |
| import React from "react";  import { withRouter } from "react-router-dom";  import "./menu-item.component.scss";  const MenuItem = ({ title, imageUrl, size, history, linkUrl, match }) => (  <div  className={`${size} menu-item`}  onClick={() => history.push(`${match.url}${linkUrl}`)}  >  <div  className="background-image"  style={{ backgroundImage: `url(${imageUrl})` }}  />  <div className="content">  <h1 className="title">{title.toUpperCase()}</h1>  <span className="subtitle">SHOP NOW</span>  </div>  </div>  );  export default withRouter(MenuItem); |

#### Redirect and render to

Redirect is basically used to stop redirect to some page after login by checking any object

|  |
| --- |
| import React from "react";  import { Switch, Route, Redirect } from "react-router-dom";  import { connect } from "react-redux";  import "./App.css";  import HomePage from "./pages/homepage/homepage.component";  import ShopPage from "./pages/shop/shop.component";  import SignInAndSignOut from "./pages/sign-in-and-sign-out/sign-in-and-sign-out.component";  import Header from "./components/header/header.component";  import { auth, createUserProfileDocument } from "./firebase/firebase.util";  import { setCurrentUser } from "./redux/user/user.actions";  class App extends React.Component {  unSubscribeFromAuth = null;  componentDidMount() {  const { setCurrentUser } = this.props;  this.unSubscribeFromAuth = auth.onAuthStateChanged(async userAuth => {  console.log(userAuth);  if (userAuth) {  const userRef = await createUserProfileDocument(userAuth);  userRef.onSnapshot(snapshot => {  setCurrentUser({  currentUser: {  id: snapshot.id,  ...snapshot.data()  }  });  console.log(this.state);  });  }  setCurrentUser(userAuth);  });  }  componentWillUnmount() {  this.unSubscribeFromAuth();  }  render() {  return (  <div>  <Header />  <Switch>  <Route exact path="/" component={HomePage} />  <Route exact path="/Shop" component={ShopPage} />  <Route  exact  path="/SignIn"  // Here basically it check currentUser value if present to it will redirect to current page it will never went to signinout page  // render is a Javascript that determines what component to return  render={() =>  this.props.currentUser ? (  <Redirect to="/" />  ) : (  <SignInAndSignOut />  )  }  />  </Switch>  </div>  );  }  }  const mapStateToProps = ({ user }) => ({  currentUser: user.currentUser  });  const mapDispatchToProps = dispatch => ({  setCurrentUser: user => dispatch(setCurrentUser(user))  });  export default connect(  mapStateToProps,  mapDispatchToProps  )(App); |

### How to import SVG file as a component?

|  |
| --- |
| import React from "react";  import { Link } from "react-router-dom";  import { ReactComponet as Logo } from "../../assests/crown.svg";  import "./header.styles.scss";  const Header = () => (  <div>  <Link className="logo-container" to="/">  <Logo className="logo"></Logo>  </Link>  </div>  );  export default Header; |

### Form in React

Here event.preventDefault() is used to stop the default submit of page in the form

|  |
| --- |
| import React from "react";  import "./sign-in.styles.scss";  class SignIn extends React.Component {  constructor() {  super();  this.state = {  email: "",  password: ""  };  }  handleSubmit = event => {  event.preventDefault();  this.setState({ email: "", password: "" });  };  // this is used for every input to set the value  handleChange = event => {  const { name, value } = event.target;  this.setState({ [name]: value });  };  render() {  return (  <div className="sign-in">  <h2>I already have a account</h2>  <span>Sign in with email and password</span>  <form onSubmit={this.handleSubmit}>  <input  name="email"  type="email"  value={this.state.email}  onChange={this.handleChange}  required  ></input>  <label>Email</label>  <input  name="password"  type="password"  value={this.state.password}  onChange={this.handleChange}  required  ></input>  <label>Password</label>  <input type="submit" value="Submit Form"></input>  </form>  </div>  );  }  }  export default SignIn; |

## Firebase

Firebase is the alternative way to handle backend code in a minimize way. We can implement backend without having to do Backend development.

### Adding Firebase to the react project

<https://www.npmjs.com/package/firebase>

**npm install firebase**

## Redux

### State Management

State description how the app should look like. Child component update the parent component using event.

Think State as memory, an app needs to remember things in order to work otherwise we will have simple html webpages.

|  |
| --- |
|  |

When the app is grows bigger and bigger it very tough to manage the state. So Redux solve this problem for us.

In redux we keep the state in a store, the store is simply the state but in one massive object. The one simple object that describe how our app should look and all we do it just pass down the state to whichever component needs it as props

|  |
| --- |
|  |

### Why Redux?

React is very good in view side but not necessary in managing state.

But using Redux solves the problem. And Redux is very good in the below points:-

1. Good for managing large state.
2. Useful for sharing data between containers.
3. Predictable state management using the 3 principles :-
   * + Single Source of truth.
     + State is read only.
     + Change using pure functions

|  |
| --- |
|  |

**With Redux all the actions goes to one reducer, then the reducer goes to store and then the store determines how the app should look like.**

**Redux uses the architectural pattern of Flux pattern, which is unidirectional data flow.**

**Store is nothing but the state**

|  |
| --- |
|  |

Before that we have something called MVC. The problem is MVC we have controller that update the different piece of model and model can change the view and that view can trigger the another change and so on and so forth.

|  |
| --- |
|  |

**Redux at the end of the data is same as this.State. With Redux we can replace all this.state from react and happens all in redux library.**

|  |
| --- |
|  |

But one caveat, we would technically have redux with the state but also still keep a little bit of react state in a component. Redux does not completely replace this.state

|  |
| --- |
|  |

|  |
| --- |
|  |

### Installing Redux

<https://react-redux.js.org/introduction/quick-start>

npm install redux

We need another package which we tell react to use redux, because redux can be used in other libraries.

npm install react-redux

Here only connection will be done with containers or the smart components and with redux store and also component will component with the store and viceversa.

|  |
| --- |
|  |

### Redux Actions and Reducers

|  |
| --- |
|  |

|  |
| --- |
|  |

**Creating an action**

**actions.js**

Below action is going to take text which is what the user input and it’s going to return an object type and payload. Payload is the common name that is used in redux, payload is we are sending whatever data is needed to go on to the Reducer.

|  |
| --- |
| import {CHANGE\_SEARCH\_FIELD} from './constants.js';  export const setSearchField = (text) => ({  type: CHANGE\_SEARCH\_FIELD,  payload: text  }) |

**Create a constants.js file for all the type declaration instead of hardcode.**

|  |
| --- |
| export const CHANGE\_SEARCH\_FIELD = 'CHANGE\_SEARCH\_FIELD'; |

**Creating the Reducer**

Reducer is a big function just read the action and spits out the state.

**reducer.js**

Below it is simply saying we receive an action for example CHANGE\_SEARCH\_FIELD, if that the case then return the new state with action.payload

|  |
| --- |
| import {CHANGE\_SEARCH\_FIELD} from './constants.js';  const initialState = {  searchField: ''  }  // Here state = intitialState is the new ES6 syntax to make a parameter default  export const searchRobots = (state=initialState, action={}) => {  switch(action.type){  case: CHANGE\_SEARCH\_FIELD:  return Object.assign({}, state, {searchField: action.payload });  default:  return state;  }  } |

**Instead of using Object.assign we can use the spread operator (…)**

|  |
| --- |
| export const searchRobots = (state=initialState, action={}) => {  switch(action.type){  case: CHANGE\_SEARCH\_FIELD:  return { ... state, {searchField: action.payload }}  }  } |

**How to combine two reducer in a root-reducer?**

|  |
| --- |
| import { combineReducers } from "redux";  import userReducer from "./user/user.reducer";  import cartReducer from "./cart/cart.reducer";  export default combineReducers({  user: userReducer,  cart: cartReducer  }); |

### Redux Store and Provider

Let connect redux to react application.

For that import provider and connect from ‘react-redux’.

**In this section we will discuss about provider and in next section about connect.**

import {provider, connect} from 'react-redux';

**Let create the store**

Store is the source of all i.e. it’s a big object that describes the state of our app, so that react can render it and make changes and display to the user.

**Import the createStore from redux and create the store which will get the reducer**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import {provider, connect} from 'react-redux';  import {createStore} from 'redux';  import './index.css';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import App from './Containers/App';  import { searchRobots } from './reducers';  const store = createStore(searchRobots)  ReactDOM.render(<App />, document.getElementById('root')); |

**How to pass the store to the component tree?**

We don’t want to pass the store all the way down to the component tree over and over again. React-Redux has given a component (provider). Now the provider component is going to take care the passing down the store to all the components in the component tree from the app.

Means we can dispatch actions to that store or we can actually pull value from the store and pass into the component.

|  |
| --- |
| ReactDOM.render(  <Provider store="store">  <App/>  </Provider>, document.getElementById('root')); |

**We can also create store in the separate file**

|  |
| --- |
| import { createStore, applyMiddleware } from "redux";  import logger from "redux-logger";  import rootReducer from "./root-reducer";  // this the middleware which execute whenever any actions fires  const middlewares = [logger];  const store = createStore(rootReducer, applyMiddleware(...middlewares));  export default store; |

In index.js

|  |
| --- |
| import React from "react";  import ReactDOM from "react-dom";  import { BrowserRouter } from "react-router-dom";  import { Provider } from "react-redux";  import store from "./redux/store";  import "./index.css";  import App from "./App";  ReactDOM.render(  <Provider store={store}>  <BrowserRouter>  <App />  </BrowserRouter>  </Provider>,  document.getElementById("root")  ); |

### Redux Connect

In containers component only we will do the redux connect. And connect is higher order function means function that returns another function.

**Step 1:- Import the action in the smart component app.js**

|  |
| --- |
| import { setSearchField } from '../actions' |

**Step 2:- Import the connect method from react-redux.**

import {connect} from 'react-redux'

**Step 3:- Use the connect method in very bottom where we do export default. And connect is higher order function means function that returns another function.**

**And connect two parameters** (mapStateToProps, mapDispatchToProps)

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

Here basically we connected the app component and said to subscribe to any state changes in the redux store and the App component knows that there is a redux store somewhere and any time there is changes to it, it might be interested to it.

Now we need to tell it what we are interested in i.e. what state should I listen and what action/dispatch should it listen

**Step 4:- Define the parameter mapStateToProps**

**This function is going to receive a state and it’s going to returns an object.**

**The object is basically searchField from reducer with the state of the searchRobots, which is going to be use as a prop by the app component. Because in index.js we can have created the store with searchRobots reducer.**

|  |
| --- |
| const mapStateToProps = state => {  return {  searchfield: state.searchRobots.searchfield  }  } |

**Step 5:- Define the parameter mapDispatchToProps**

**It’s is going to receive a dispatch, dispatch (basically what triggers the action). To dispatch into the reducer, so this dispatch can now be used to send action**

|  |
| --- |
| const mapDispatchToProps = (dispatch) => {  return {  onSearchChange: (event) => dispatch(setSearchField(event.target.value))  }  } |

**mapStateToProps :- Is used to what piece of state app need to listen to and send down as props.Basically it is getting value from reducer/**

**mapDispatchToProps:- What props app need to listen to that are actions, that needs to get dispatch.Basically it send actions to reducers to update the state**

|  |
| --- |
| import React from 'react'  import {connect} from 'react-redux'  import CardList from '../Components/CardList';  import SearchBox from '../Components/SearchBox';  import Scroll from '../Components/Scroll';  import ErrorBoundary from '../Components/ErrorBoundary';  import { setSearchField } from '../actions'  const mapStateToProps = state => {  return {  searchField: state.searchField  }  }  const mapDispatchToProps = (dispatch) => {  return {  onSearchChange: (event) => dispatch(setSearchField(event.target.value))  }  }  class App extends React.Component {  constructor() {  super();  this.state = {  robots: []  }  }  componentDidMount() {  fetch('https://jsonplaceholder.typicode.com/users')  .then(response => response.json())  .then(users => this.setState({robots:users}));  }  render() {  const {robots} = this.state;  const {searchField,onSearchChange} = this.props;  const filteredRobots = robots.filter(robot => {  return robot.name.toLowerCase().includes(searchField.toLowerCase())  })  return !robots.length ?  <h1>Loading..</h1> :  (  <div className="tc">  <h1 className="f2">RoboFriends</h1>  <SearchBox onSearchChange = {onSearchChange} />  <Scroll>  <ErrorBoundary>  <CardList robots={filteredRobots}/>  </ErrorBoundary>  </Scroll>  </div>  );  }  }  //subscribe to any state changes in the redux store  export default connect(mapStateToProps, mapDispatchToProps)(App); |

**How to get two props from different reducers**

|  |
| --- |
| import React from "react";  import { connect } from "react-redux";  import { Link } from "react-router-dom";  import { auth } from "../../firebase/firebase.util";  import CartIcon from "../cart-icon/cart-icon.component";  import CartDropdown from "../cart-dropdown/cart-dropdown.component";  import { ReactComponent as Logo } from "../../assests/crown.svg";  import "./header.styles.scss";  const Header = ({ currentUser, hidden }) => (  <div className="header">  <Link className="logo-container" to="/">  <Logo className="logo" />  </Link>  <div className="options">  <Link className="option" to="/shop">  SHOP  </Link>  <Link className="option" to="/shop">  CONTACT  </Link>  {currentUser ? (  <div className="option" onClick={() => auth.signOut()}>  SIGN OUT  </div>  ) : (  <Link className="option" to="/signin">  SIGN IN  </Link>  )}  <CartIcon />  </div>  {hidden ? null : <CartDropdown />}  </div>  );  // user and cart reducer  const mapStateToProps = ({ user: { currentUser }, cart: { hidden } }) => ({  currentUser,  hidden  });  export default connect(mapStateToProps)(Header); |

#### Dispatch Action Shorthand

As we know connect actually passes dispatch into the component as a prop if we don’t supply a second argument in connect().

Basically no need to write mapDispatchToProps() method. Directly we can dispatch as highlighted below.

|  |
| --- |
| import React from "react";  import { withRouter } from "react-router-dom";  import { connect } from "react-redux";  import { createStructuredSelector } from "reselect";  import CustomButton from "../custom-button/custom-button.component";  import CartItem from "../cart-item/cart-item.component";  import { selectCartItems } from "../../redux/cart/cart.selectors";  import { toggleCartHidden } from "../../redux/cart/cart.actions";  import "./cart-dropdown.styles.scss";  const CartDropdown = ({ cartItems, history, dispatch }) => (  <div className="cart-dropdown">  <div className="cart-items">  {cartItems.length ? (  cartItems.map(cartItem => (  <CartItem key={cartItem.id} item={cartItem} />  ))  ) : (  <span className="empty-message">Your cart is empty</span>  )}  </div>  <CustomButton  onClick={() => {  history.push("/checkout");  dispatch(toggleCartHidden());  }}  >  GO TO CHECKOUT  </CustomButton>  </div>  );  const mapStateToProps = createStructuredSelector({  cartItems: selectCartItems  });  export default withRouter(connect(mapStateToProps)(CartDropdown)); |

### Selectors in Redux

Selectors are the way to get the slice of the state. Here whenever any reducer update mapstatetoprops is actually called every single time

|  |
| --- |
| import React from "react";  import { connect } from "react-redux";  import { toggleCartHidden } from "../../redux/cart/cart.actions";  import { ReactComponent as ShoppingIcon } from "../../assests/shopping-bag.svg";  import "./cart-icon.styles.scss";  const CartIcon = ({ toggleCartHidden, itemCount }) => (  <div className="cart-icon" onClick={toggleCartHidden}>  <ShoppingIcon className="shopping-icon" />  <span className="item-count">{itemCount}</span>  </div>  );  const mapDispatchToProps = dispatch => ({  toggleCartHidden: () => dispatch(toggleCartHidden())  });  //Here basically we can cartItems from the cart reducer instead of full cart state  const mapStateToProps = ({ cart: { cartItems } }) => ({  itemCount: cartItems.reduce(  (accumulatedQuantity, cartItem) => accumulatedQuantity + cartItem.quantity,  0  )  });  export default connect(  mapStateToProps,  mapDispatchToProps  )(CartIcon); |

Reducer of cart

|  |
| --- |
| import { CartActionTypes } from "./cart.types";  import { addItemToCart } from "./cart.utils";  const INITIAL\_STATE = {  hidden: true,  cartItems: []  };  const cartReducer = (state = INITIAL\_STATE, action) => {  switch (action.type) {  case CartActionTypes.TOGGLE\_CART\_HIDDEN:  return {  ...state,  hidden: !state.hidden  };  case CartActionTypes.ADD\_ITEM:  return {  ...state,  cartItems: addItemToCart(state.cartItems, action.payload)  };  default:  return state;  }  };  export default cartReducer; |

#### Memorization in Redux (using reselect)

Memorization is basically caching of the selectors value, we can achieve this memorization using the library called reselect.

Reselect rewrites the selectors in such away, so that it knows that if the property which is pulling from the state and using are the same (means the value has not change and the output of the selector is not different), then it will not pass into the component which will pass the old value and the react component will know not to re-render.

**Step 1:- Install the reselect package**

npm install reselect

**Step 2:- Create a new file cart.selectors.js so that we can reuse the selectors in some other component**

|  |
| --- |
| import { createSelector } from "reselect";  // here basically we are pulling only the cart state from the state  const selectCart = state => state.cart;  // selecting the only the CartItems from the cart state  export const selectCartItems = createSelector(  [selectCart],  cart => cart.CartItems  );  //Now from the above selectCartItems we are getting the counts using the reduce  export const selectCartItemsCount = createSelector(  [selectCartItems],  cartItems =>  cartItems.reduce(  (accumulatedQuantity, cartItem) =>  accumulatedQuantity + cartItem.quantity,  0  )  ); |

**Step 3:- Import the selector in the component**

|  |
| --- |
| import React from "react";  import { connect } from "react-redux";  import { toggleCartHidden } from "../../redux/cart/cart.actions";  import { selectCartItemsCount } from "../../redux/cart/cart.selectors";  import { ReactComponent as ShoppingIcon } from "../../assests/shopping-bag.svg";  import "./cart-icon.styles.scss";  const CartIcon = ({ toggleCartHidden, itemCount }) => (  <div className="cart-icon" onClick={toggleCartHidden}>  <ShoppingIcon className="shopping-icon" />  <span className="item-count">{itemCount}</span>  </div>  );  const mapDispatchToProps = dispatch => ({  toggleCartHidden: () => dispatch(toggleCartHidden())  });  /\*const mapStateToProps = ({ cart: { cartItems } }) => ({  itemCount: cartItems.reduce(  (accumulatedQuantity, cartItem) => accumulatedQuantity + cartItem.quantity,  0  )  });\*/  //Above code now change as below by using selectors from cart.selectors.js  // Also we are passing the state to the selectors which will check, that it needs cart state from the state  const mapStateToProps = state => ({  itemCount: selectCartItemsCount(state)  });  export default connect(  mapStateToProps,  mapDispatchToProps  )(CartIcon); |

**createStructuredSelector**

**How to combine two selectors in same component (using createStructuredSelector)**

**Even though we may use one selector in a component but better to use createStructuredSelector, so the in future we can add multiple selector**

|  |
| --- |
| import React from "react";  import { connect } from "react-redux";  import { createStructuredSelector } from "reselect";  import { Link } from "react-router-dom";  import { auth } from "../../firebase/firebase.util";  import CartIcon from "../cart-icon/cart-icon.component";  import CartDropdown from "../cart-dropdown/cart-dropdown.component";  import { selectCurrentUser } from "../../redux/user/user.selectors";  import { selectCartHidden } from "../../redux/cart/cart.selectors";  import { ReactComponent as Logo } from "../../assests/crown.svg";  import "./header.styles.scss";  const Header = ({ currentUser, hidden }) => (  <div className="header">  <Link className="logo-container" to="/">  <Logo className="logo" />  </Link>  <div className="options">  <Link className="option" to="/shop">  SHOP  </Link>  <Link className="option" to="/shop">  CONTACT  </Link>  {currentUser ? (  <div className="option" onClick={() => auth.signOut()}>  SIGN OUT  </div>  ) : (  <Link className="option" to="/signin">  SIGN IN  </Link>  )}  <CartIcon />  </div>  {hidden ? null : <CartDropdown />}  </div>  );  /\*  const mapStateToProps = state => ({  currentUser: selectCurrentUser(state),  hidden: selectCartHidden  });\*/  //using createStructuredSelector the above comment code can also use  const mapStateToProps = createStructuredSelector({  currentUser: selectCurrentUser,  hidden: selectCartHidden  });  export default connect(mapStateToProps)(Header); |

### Redux Middleware

It simply listens for action; depending on the action Middleware can modify the action or trigger another action.

|  |
| --- |
|  |

### Redux logger

It is middleware which we can use for logging purpose in console.

npm install redux-logger

|  |  |
| --- | --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import {Provider} from 'react-redux';  import {createStore, applyMiddleware } from 'redux';  import {createLogger} from 'redux-logger';  import './index.css';  import App from './Containers/App';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import { searchRobots } from './reducers';  const logger = createLogger();  const store = createStore(searchRobots, applyMiddleware(logger));  ReactDOM.render(  <Provider store={store}>  <App />  </Provider>, document.getElementById('root'));  // If you want your app to work offline and load faster, you can change  // unregister() to register() below. Note this comes with some pitfalls.  // Learn more about service workers: https://bit.ly/CRA-PWA  serviceWorker.unregister();  **How to use logger only for development environment?**   |  | | --- | | import { createStore, applyMiddleware } from "redux";  //it allows the browser to cache or store  import { persistStore } from "redux-persist";  import logger from "redux-logger";  import rootReducer from "./root-reducer";  const middlewares = [];  if (process.env.NODE\_ENV === "development") {  middlewares.push(logger);  }  export const store = createStore(rootReducer, applyMiddleware(...middlewares));  //Creating the new persistor version of store  export const persistor = persistStore(store);  export default { store, persistor }; |  Redux Persist <https://www.npmjs.com/package/redux-persist>  Redux persist is a library allowing to save the redux store in the local storage of your browser. |

**Step 1:- Install the library**

npm install redux-persist

**Step 2:- Import the persistStore from redux-persist in store.js**

|  |
| --- |
| import { createStore, applyMiddleware } from "redux";  //it allows the browser to cache or store  import { persistStore } from "redux-persist";  import logger from "redux-logger";  import rootReducer from "./root-reducer";  const middlewares = [logger];  export const store = createStore(rootReducer, applyMiddleware(...middlewares));  //Creating the new persistor version of store  export const persistor = persistStore(store);  export default { store, persistor }; |

**Step 3:- Import persistReducer from redux-persist and storage from redux-persist/lib/storage in root-reducer.js**

|  |
| --- |
| import { combineReducers } from "redux";  import { persistReducer } from "redux-persist";  //we will get here the local storage object on the window object  // means it is saying that default I want localstorage for storage  import storage from "redux-persist/lib/storage";  import userReducer from "./user/user.reducer";  import cartReducer from "./cart/cart.reducer";  const persistConfig = {  key: "root",  storage,  // define the reducer storage you want to store in localstorage  whitelist: ["cart"]  };  const rootReducer = ombineReducers({  user: userReducer,  cart: cartReducer  });  export default persistReducer(persistConfig, rootReducer); |

**Step 4:- Import PersistGate from redux-persist In index.js so that need to bring the actual persisto**

|  |
| --- |
| import React from "react";  import ReactDOM from "react-dom";  import { BrowserRouter } from "react-router-dom";  import { Provider } from "react-redux";  import { PersistGate } from "redux-persist/integration/react";  import { store, persistor } from "./redux/store";  import "./index.css";  import App from "./App";  ReactDOM.render(  <Provider store={store}>  <BrowserRouter>  <PersistGate persistor={persistor}>  <App />  </PersistGate>  </BrowserRouter>  </Provider>,  document.getElementById("root")  ); |

### Redux Async Actions (redux thunk)

Using middle ware we can do this. In Redux we have the middle ware redux thunk to handle asynchronous actions like Ajax calls.

It’s a middle ware that provides a get state and dispatch functions that are passed on you are able to handle something called side effects like Ajax calls. There are other options also but redux thunk is best and simplest to start off with

npm install --save redux-thunk

**step 1:- Import the middleware and apply the middleware**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import {Provider} from 'react-redux';  import {createStore, applyMiddleware } from 'redux';  import {createLogger} from 'redux-logger';  import thunkMiddleware from 'redux-thunk';  import './index.css';  import App from './Containers/App';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import { searchRobots } from './reducers';  const logger = createLogger();  const store = createStore(searchRobots, applyMiddleware(thunkMiddleware,logger));  ReactDOM.render(  <Provider store={store}>  <App />  </Provider>, document.getElementById('root'));  // If you want your app to work offline and load faster, you can change  // unregister() to register() below. Note this comes with some pitfalls.  // Learn more about service workers: https://bit.ly/CRA-PWA  serviceWorker.unregister(); |

**Step 2:- Implement constants, actions, reducers, provider and the connect for the async action.**

**Step a: - Create the constants**

|  |
| --- |
| export const REQUEST\_ROBOTS\_PENDING = 'REQUEST\_ROBOTS\_PENDING';  export const REQUEST\_ROBOTS\_SUCCESS = 'REQUEST\_ROBOTS\_SUCCESS';  export const REQUEST\_ROBOTS\_FAILED = 'REQUEST\_ROBOTS\_FAILED'; |

**Step b: - Create the actions using the above constants and implement the functionality of asynchronous operation**.

|  |
| --- |
| import {  CHANGE\_SEARCHFIELD,  REQUEST\_ROBOTS\_SUCCESS,  REQUEST\_ROBOTS\_PENDING,  REQUEST\_ROBOTS\_FAILED  } from './constants'  export const setSearchField = (text) => ({  type: CHANGE\_SEARCHFIELD,  payload: text  })  export const requestRobots = (dispatch) => {  dispatch({type: REQUEST\_ROBOTS\_PENDING});  fetch('https://jsonplaceholder.typicode.com/users')  .then(response => response.json())  .then(data => dispatch({type: REQUEST\_ROBOTS\_SUCCESS, payload: data}))  .catch(error => dispatch({type: REQUEST\_ROBOTS\_FAILED, payload: error}))  } |

**Step c: - Create the reducers using the above constants**

|  |
| --- |
| import {  CHANGE\_SEARCHFIELD,  REQUEST\_ROBOTS\_SUCCESS,  REQUEST\_ROBOTS\_PENDING,  REQUEST\_ROBOTS\_FAILED  } from './constants'  const initialStateSearch = {  searchField: ''  }  export const searchRobots = (state = initialStateSearch, action={}) => {  switch(action.type) {  case CHANGE\_SEARCHFIELD:  return Object.assign({}, state ,{searchField:action.payload});  default:  return state;  }  }  const initialStateRobots = {  isPending: false,  robots: [],  error: ''  }  export const requestRobots = (state = initialStateRobots, action={}) => {  switch(action.type) {  case REQUEST\_ROBOTS\_PENDING:  return Object.assign({}, state, {isPending: true});  case REQUEST\_ROBOTS\_SUCCESS:  return Object.assign({}, state, {robots: action.payload, isPending: false});  case REQUEST\_ROBOTS\_FAILED:  return Object.assign({}, state, { error: action.payload, isPending:false});  default:  return state;  }  } |

**Step d: Import the reducer in index.js and also add both the reducer (as we have two reducer now) in the store using combinedReducers (which combines all the reducer into a root).**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import {Provider} from 'react-redux';  import {createStore, applyMiddleware, combineReducers } from 'redux';  import {createLogger} from 'redux-logger';  import thunkMiddleware from 'redux-thunk';  import './index.css';  import App from './Containers/App';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import { searchRobots, requestRobots } from './reducers';  const logger = createLogger();  const rootReducer = combineReducers({searchRobots,requestRobots});  const store = createStore(rootReducer, applyMiddleware(thunkMiddleware,logger));  ReactDOM.render(  <Provider store={store}>  <App />  </Provider>, document.getElementById('root'));  // If you want your app to work offline and load faster, you can change  // unregister() to register() below. Note this comes with some pitfalls.  // Learn more about service workers: https://bit.ly/CRA-PWA  serviceWorker.unregister(); |

**Step e: Implement in container instead of directly use the state, use it from the redux.**

|  |
| --- |
| import React from 'react'  import {connect} from 'react-redux'  import CardList from '../Components/CardList';  import SearchBox from '../Components/SearchBox';  import Scroll from '../Components/Scroll';  import ErrorBoundary from '../Components/ErrorBoundary';  import { setSearchField, requestRobots } from '../actions'  const mapStateToProps = state => {  return {  searchField: state.searchRobots.searchField,  robots: state.requestRobots.robots,  isPending: state.requestRobots.isPending,  error: state.requestRobots.error  }  }  const mapDispatchToProps = (dispatch) => {  return {  onSearchChange: (event) => dispatch(setSearchField(event.target.value)),  onRequestRobots: () => dispatch(requestRobots())  }  }  class App extends React.Component {  componentDidMount() {  this.props.onRequestRobots();  }  render() {  const {searchField,onSearchChange, robots, isPending} = this.props;  const filteredRobots = robots.filter(robot => {  return robot.name.toLowerCase().includes(searchField.toLowerCase())  })  return isPending ?  <h1>Loading..</h1> :  (  <div className="tc">  <h1 className="f2">RoboFriends</h1>  <SearchBox onSearchChange = {onSearchChange} />  <Scroll>  <ErrorBoundary>  <CardList robots={filteredRobots}/>  </ErrorBoundary>  </Scroll>  </div>  );  }  }  //subscribe to any state changes in the redux store  export default connect(mapStateToProps, mapDispatchToProps)(App); Stripe The new standard for online payment. |

<https://stripe.com/>

**How to create the stripe checkout component?**

**Step 1:- Install the package react-stripe-checkout**

<https://www.npmjs.com/package/react-stripe-checkout>

npm install react-stripe-checkout

**Step 2:- Import the package StripeCheckout in the component and also use the publishable key from the stripe website**

|  |
| --- |
| import React from "react";  import StripeCheckout from "react-stripe-checkout";  const StripCheckoutButton = ({ price }) => {  const priceForStripe = price \* 100;  const publisableKey = "pk\_test\_ojFa0DNhBiHKRIpUBr4pmPrK00Uh5Bau20";  // As we are test so just log, we are not accessing the backend  const onToken = token => {  console.log(toke);  alert("Payment successfull");  };  return (  <StripeCheckout  label="Pay Now"  // name of the application you can give  name="CJ Private Ltd."  billingAddress  shippingAddress  image="https://sendeyo.com/up/d/f3eb2117da"  description={`Your total is &#8377;${price}`}  amount={priceForStripe}  panelLabel="Pay Now"  token={onToken}  stripeKey={publisableKey}  />  );  };  export default StripCheckoutButton; |

<https://stripe.com/docs/testing#cards>

|  |
| --- |
|  |

## Resources

### UI Library

React 360:-<https://facebook.github.io/react-360/>

React blessed: - <https://github.com/Yomguithereal/react-blessed>

React Desktop: - <https://reactdesktop.js.org/>

### Web Developer Monthly News

<https://zerotomastery.io/blog/?tag=WDM>

### Yarn

It is the replacement of npm created by facebook

<https://classic.yarnpkg.com/en/docs/install#windows-stable>

### IE11 issue

<https://medium.com/@matwankarmalay/create-react-app-ie11-script1002-syntax-error-how-to-get-rid-of-it-d70000c53ddf>

### SASS

<https://sass-lang.com/>

### Whimsical :- for visual workspace

<https://whimsical.com/>