**REACT.JS**

**What is React?**

* React is a JavaScript library used for building interfaces.
* Created by Facebook.
* Used by some big platforms such as Netflix and Instagram.
* Used to create JavaScript-driven dynamic web apps.

**What is Redux?**

* Redux is a layer on top of React that helps with state management of our app.
* State Management means:-
  + Management of data in app.
  + Management of UI state of the app. (Pop up open or close)

**What we’ll learn**

* React Basics: -
  + Components, events, templates, props & forms.
* React Router
  + Routes, Route parameters, redirects
* Redux
  + Stores, actions & reducers.
* Small projects in this playlist.
* In next playlist, complete playlist on a project with React, Redux and firebase.
* Lesson can be found on repository “The course files for the React & Redux Complete tutorial playlist on The Net Ninja YouTube channel.

**# 2. HOW REACT WORKS**

Basic overview of how react works

* Components are the heart and soul of React. We create components in react for different parts of our application such as navbar, search box, footer. Then its react job to take these components and inject them in the DOM (Document Object Model) so that we can see them in the web page.
* React takes all these components and create a JavaScript representation of DOM which is called Virtual DOM.
* Then this Virtual DOM is rendered to the browser to create Actual DOM based on it. That’s how we see all these components.
* Now whenever we make any change in any one of the component, React update the DOM inside the browser so that user can see that change.
* This process is very fast because of Virtual DOM.
* For Example: -
  + Suppose we made some change in search box or in state of search box then react will create a new Virtual DOM and compare it with the old Virtual DOM and find exactly where new update to be done in the DOM. It only update that part instead of whole part.

**How the components are made and how they looks like**

* Components generally looks like HTML templates but it’s actually JSX (JavaScript XML).
* They can contain ‘State’ (data or UI state).
* They can also contain JavaScript for functionality.

A sample components look like: -

class App extends React.Component {

    state = {

        name: 'Ryu',

        age: 30

    }

    render() {

        return (

            <div className="app-content">

                <h1>Hello World</h1>

                <p>My name is {this.state.name} and I am {this.state.age} </p>

            </div>

        )

    }

}

**REACT SETUP**

* There are couple of ways to setup React and start using it.
  + We can use a package called “create react app” to create a full react package where react control the flow of whole project by with a web pack setup.
  + We can use **React CDN** to quickly get up and running with react. This is good when you only want to use react to create widgets or to control small sections of web page. It’s also good for beginners.
* Go to <https://reactjs.org/docs/cdn-links.html>
* Copy two scripts from the top of the page.
* Create a simple index.html file which is basically nothing more than a simple html file.
* Now below the meta tag and above the title tag, paste the above two scripts.

**<script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></script>**

**<script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>**

* First script is the core react library.
* Second script is for react-DOM library. It is used to inject components to the DOM.
* So we want both of these scripts to be loaded into our index.html file.

**FIRST REACT COMPONENT**

* We use components to take control over different things or elements inside our website such as navbar, search bar, contact form kind of things.
* We create our first React component to take control over this element.
* <div id="app"></div>

How Do We Create A React Component: -

* There are few different ways of creating components but for now we will discuss JavaScript classes (A class based component)
* Everything will be written inside the <script> tags.
* class App extends React.Component { // statements }
* We have access to this (**React.Component**) react object because we have loaded react with the help of script.
* Now we are using **React.Component** to inherit all the base functionality of a component into this class.
* This class will represent our component.
* These class based component must have atleast one function that is **render( )**.
* This render function is responsible for rendering our template to whatever element we want to render to.
* In this function, we are going to return a value.
* This value is going to be our JSX template.
* JSX is way for us to write HTML code inside JavaScript.
* Now we need to take care of two things while writing JSX
  + First one, there should be only one root element.
  + Second one, we don’t use word class instead of that we use word classname, because class is a reserved keyword in JavaScript.

Now we have written our component.

* class App extends React.Component {
* render() {
* return (
* <div className="app-content">
* <h1>Hey, ninjas</h1>
* </div>
* )
* }
* }

But for now it’s not doing anything. Because we don’t have anything that will connect our component and that html element.

* That’s where **react dom** comes into the picture.
* Now, the way by which we will take this component and render this to dom is: -
* ReactDOM.render(<App />, document.getElementById('app'));
* ReactDOM we have access to this object because we have used the script to load react dom in our project.
* Inside ReactDOM.render() will take two parameters:-
  + First one, Which class or which component we want to render to the DOM. We don’t write it directly, we write it as a tag. (In above case it is **App**).
  + Second one, where we want to render it in the DOM. (select that element where you want to place your component).
* One last thing is, JSX is not supported in the browsers. So we need to take this code and transpile (source-to-source compilers) this into something which is supported.
* To do this, we will use tool called babel.
  + We will go to this link <https://babeljs.io/setup#installation>
  + And copy and paste this script with react and react dom script.

<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

* + Make assign a type to script tag that is *type=”text/babel”*.
* <script type="text/babel">

That is our first working component in react.

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

    <script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></script>

    <script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>

    <title>Document</title>

</head>

<body>

    <div id="app">Hello</div>

    <script type="text/babel">

        class App extends React.Component {

            render() {

                return (

                    <div className="app-content">

                        <h1>Hey, ninjas</h1>

                    </div>

                )

            }

        }

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

    </script>

</body>

</html>

* One last thing we can say that why we are doing that much we can directly write the div tag with classname=”app-content” in html and we can get the desired result in this case. So why we are not doing that?
* Answer to this question is React allow us to use dynamic content while HTML don’t allow us to do so and static code is of no use in long run. That’s why we are doing this.
* For example: - Here we are printing random number on web page. Which we can’t do with HTML.
* <script type="text/babel">
* class App extends React.Component {
* render() {
* return (
* <div className="app-content">
* <h1>Hey, ninjas</h1>
* <h1> { Math.random() \* 10 } </h1>
* </div>
* )
* }
* }
* ReactDOM.render(<App />, document.getElementById('app'));
* </script>

**Video 5. From 00:00**