**What is ReactJS?**

- Open source library for building UI
- JS library not a framework
- Focus on UI

**Why?**

- It's a project created & maintained by Facebook
- More than 100k starts on github
- Huge community i.e. you are able to find 1000's of article when you are face obstacle while starting a react.

**Features**

1. Component based architecture

Eg. header, footer sidenav & main content

2. Reusable code

Eg. article component in react can be reuse in article data on angular & vuejs also, for enterprise projects reusable code is plus point.

3. React it declarative

Eg. Tell react what you want & react will build the actual UI, like go to artist & ask them to draw landscape you don't tell them how to draw it, it's up to you that is declarative.

4. It will handle efficiently updating & rendering of the components

5. DOM updates are handles gracefully in react.

**Prerequisites**

- html,css and js fundamentals
- ES6
- js - this keyword, filter, map & reduce
- ES6 - let & const, arrow function, template literals, default parameter, object literals, rest & spread operator and destructuring assignment.

**How to Create React App**

1. **npx:** npx is npm pakage runner which allow us to create app without having the configuration of react.
   Syntax: npx create-react-app project_name

2. **npm:**

   Syntax: npm install create-react-app –g and create-react-app project_name

**Folder Structure**

- package.json: This file contains the dependencies & script require for the project.
- node_modules folder: In which all the dependencies are installed & generated when you run create-react-app
- public folder: containing the index.html
- src folder: you will be working with the most during development.

**Components:**

- It describe a part of ui
- They are re-usable and can be nested inside other components
- Two types: functional & class component
- They are the building blocks of any react app

**Types of Component:**

1. **Functional Component (Stateless/Dumb/Presentational):**
    - It looks like a simple function that will return a simple or regular html.
    - It's a JS component that accepts an input of properties as a props & return html that describe the UI.

2. **Class Component (Stateful/ Smart/ Container):**

- It's a regular class of es6 that extends react component class from the react library.
- It must contains render() that will in term return simple html.
- It is basically es6 class similar to functional component can optionally accepts a props & return html, apart from that its can also maintain a private internal state, it can maintain some info which is private to that component & use that info to describe the UI.

| **Functional Component** | **Class Component** |
|---|---|
| **Simple function:** recieving props & return declaration. Try to use functional component as much as possible.<br><br>Absence of 'this' keyword which is encountered in class based component. this keyword is quite tricky for beginner & functional component will help you in that aspect.<br><br>**solution without using state:** force to think a solution without having a use of state.<br><br>mainly responsible for the UI<br><br>Stateless/Dumb/Presentational | More feature rich.<br><br>Maintain their own private data known as state.<br><br>Maintain the complex UI logic.<br><br>Provide lifecycle hooks<br><br>Stateful/ Smart/ Container |

**JSX**

- JavaScript XML(JSX) - Extension to the JS language syntax.
- With a react library it's an extension to write XML like code for elements & components
- JSX tags have tag name, attributes, and children.

**Why JSX**

- JSX is not necessity to write React applications.
- JSX makes your react code simpler and elegant.
- JSX ultimately transpiles to pure js which is understood by the browser.

**JSX Syntax Changes:**

Class => className

For => htmlFor

camelCase property naming convention

1. onclick => onClick

2. tabindex => tabIndex

**Props**:

Its short form of property is an optional input that your component can accept also allow component to be dynamic, props are immutable mean readonly.

**State:**

It is nothing but a object privately maintained inside a component, it can be changed within the component.

**Props vs State**

| **Props** | **State** |
|---|---|
| 1. props get passed to the component<br>2. function parameter<br>3. props are immutable<br>4. props - functional component<br>5. this.props - class component | 1. state is managed within the component<br>2. variables declared in the function body<br>3. state can be changed<br>4. useState Hook - functional component<br>5. this.state - class component |

**setState:**

- Always make use of setState and never modify the state directly.
- Code has to be executed after the state has been updated? Place that code in the call back function which is second argument to the setState method.

- When you have to update state based on the previous state value, pass a function as an args instead of the regular object.

**Destructuring:**

Extract the properties from an object that is known as destructuring.

**Binding event handler:**

1. onClick={this.changeHandler.bind(this)}

2. onClick={( ) => this.changeHandler( )}

3. this.changeHandler = this.changeHandler.bind(this) inside constructor

4. changeHandler = ( ) => { this.setState({message:'good bye'}) } as a property of class

**Methods as props**

1. onClick={props.changeHandler} // default function

2. onClick={( ) => props.changeHandler('child')} // parameterized function

**Conditional rendering**

1. if/else

2. element cariables

3. ternary conditional operator

4. short circuit operator

Note:

1. JSX is not worked in if/else
2. Recommanded: short circuit or ternary

Eg 1. if(this.state.isLogin){

return Hello Admin

}else{

return Hello Guest

}

Eg 2. let message

if(this.state.isLogin){

message = Hello Admin

```
} else{
message = <div>Hello Guest</div>
}
return (<div>{meassage}</div>)
```

Eg 3. return this.state.isLogin ? ( Hello Admin ) : ( Hello Guest )

Eg 4. return this.state.isLogin && Hello Admin

**Key props**

Key help react to identify which item in the list have changed or added or removed to handle UI efficiently.

1. Key is a special string attribute you need to include when creating lists of elements.

2. Keys give the elements a stable identity.

3. Key help react identify which item have changed, are added, or removed.

4. Help in efficient update of the user interface.

**Index as key**

Reference: https://codepen.io/gopinav/pen/gQpepq

**When to use index as key?**

1. The items in your list do not have a unique id.

2. The list is a static list and will not change.

3. The list will never be reordered or filtered.

**Styling React Components**

1. css stylesheets: conditionally(ternary) add className

2. inline styling: style={obj} // obj = {fontSize:'72px' }

3. css modules: appStyle.modules.css // import styles from "appStyle.modules.css" only apply child not subchild

4. css in js libraries

**Lifecycle Hooks**

**1. Mounting:** When an instance of a component is being created and inserted into the DOM.

**methods:** constructor, static getDerivedStateFromProps, render and componentDidMount

**2. Updating**: When component is being re-rendered as a result of changes to either its props or state

**methods**: static getDerivedStateFromProps, render, getSnapshotBeforeUpdate,shouldComponentUpdate, componentDidUpdate

**3. Unmounting**: When a component is being removed from the DOM.

**methods**: componentWillUnmount

**4. Error Handling**: When there is an error during rendering, in a lifecycle method,

or in the constructor of any child component.

**methods**: static getDerivedStateFromError and componentDidCatch

**constructor:** a special function that will get called whenever a new component is created.

**pure component:** it is use to prevent unnecessary rendered component & give you a performance boost.

1. create component by extending the PureComponent class

2. it implements shouldComponentUpdate lifecycle method by preforming a shallow comparison on the props & state of the component.

3. if there is no diff, the component is not re-rendered -- performance boost

4. it is a good idea to ensure that all the children components are also pure to avoid unexpected behaviour.

5. Never mutate the state. Always return a new Object that reflects the new state.

**API Integration using Axios**

1. npm install axios

2. import axios from 'axios' inside component.

3. set state posts:[]

4. componentDidMount(){

```
        axios.get('path')
.then(res => {
        console.log(red);
        this.setState({  posts:res.data})
}).catch(error => {
        console.log(error);
})}
```

**Higher Order Component:**

1. To share common functionality between components.
2. A pattern where a function takes a component as an args and returns a new component.

   const NewComponent = higherOrderComponent(Original)

**Router in React:**

Command: install react-router-dom –save

**<Router>**: The router that keeps the UI in sync with the URL.

**<Link>**: Renders a navigation link

**<Route>**: Renders a UI component depending on the URL.

**<Switch>**:  will make the path matching exclusive rather than inclusive (as if you were using <Route> components).

For example, even if you duplicate the route for the Messages component:

When visiting the /messages path, the Messages component will be rendered only once.

In a web application, you have two options:

1. **<BrowserRouter>,** which uses the HTML5 History API.

Eg: http://localhost:3000/route/subroute

2. **<HashRouter>**, which uses the hash portion of the URL. (window.location.hash)

Eg: http://localhost:3000/#/route/subroute

**Example of Router:**

```
import { BrowserRouter as Router, Link, Route, Switch, Redirect } from 'react-router-dom';

<Router>
        <Link to='/home'>Home</Link>
        <Link to='/about'>About</Link>
        <Link to='/form'>Form</Link>
        <Switch>
          <Route path='/home' component={Home} />
                <Route path='/about' component={About} />
                <Route path='/about' component={About} />
                <Route path='/form' component={Form} />
                <Redirect to='/home' />
        </Switch>
</Router>
```

**Reference:**

https://blog.logrocket.com/react-router-dom-set-up-essential-components-parameterized-routes-505dc93642f1/

**Redux:**

**Boilerplate**

**Reference:** https://github.com/Angelfire/React-Redux-Thunk-Boilerplate