React JS is a JavaScript Library for building front end application or user interfaces (UI).

We can build modern, fast Single Page Applications or websites with React.

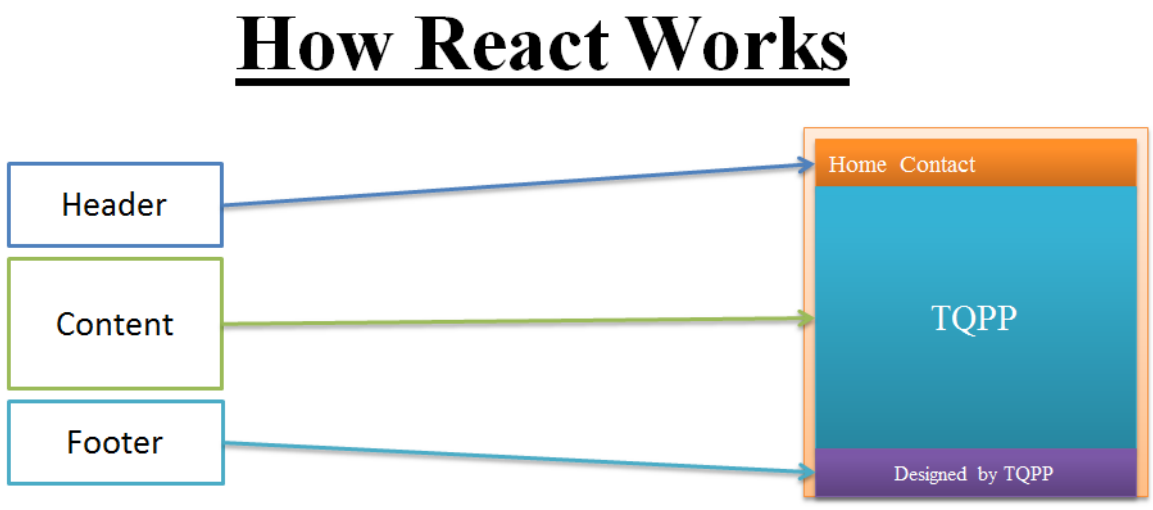
ReactJS allows us to create reusable UI components.

Component - Components are the building blocks of any React app.

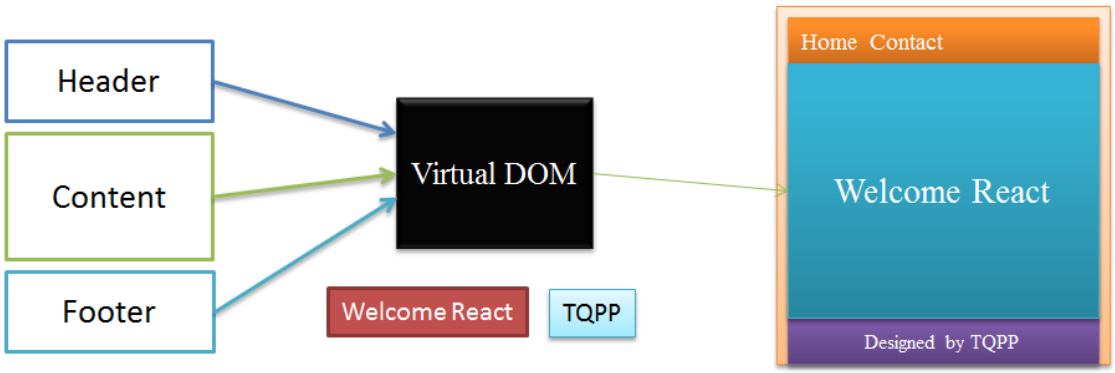
It is created by Facebook

Advantages of ReactJS

- Reusable Components
- Open Source
- Efficient and Fast
- Works in Browser
- Large Community



How React Works



Requirements

- Text Editor/Source Code Editor – Visual Studio Code, Notepad++, Atom etc
- Web Browser – Google Chrome, Firefox

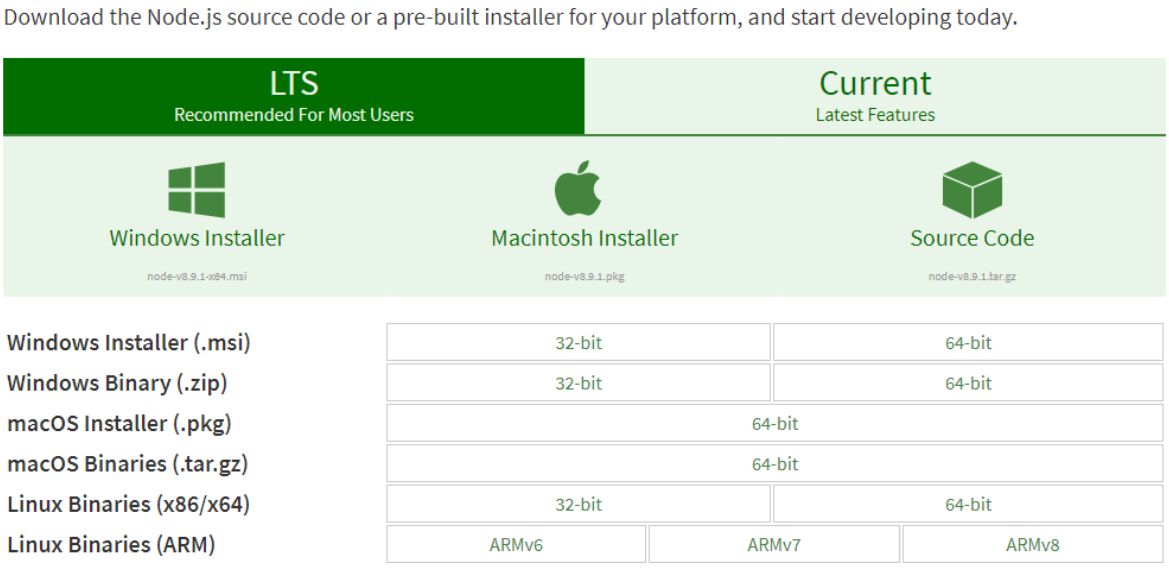
Setup:

You'll need to have Node >= 8.10 and npm >= 5.6 on your machine.

Step 1 : Install Node.js and npm (both of them are part of single download)

npm means node package manager like maven but for FE

- Node version >= 8.10
- npm >= 5.6
- To get the latest version of Node.js
  https://nodejs.org/en/download



Note : The following messages indicate you don't have node installed

'npm' is not recognized as an internal or external command

'node' is not recognized as an internal or external command

- To check the version of node installed : node –v
- To check the version of npm installed : npm –v

Install Visual Studio Code https://code.visualstudio.com/
Now our work will start:

Discuss project structure in detail , ctrl J for terminal

To create a project, run:

1. create new app in react

   npx create-react-app project name

2. set project path in terminal using cd project name then

3. run app using :- npm start

npx is a npm package runner (x probably stands for eXecute). The typical use is to download and run a package temporarily or for trials. create-react-app is an npm

package that is expected to be run only once in a project's lifecycle. Hence, it is preferred to use npx to install and run it in a single step.

Globally Installation

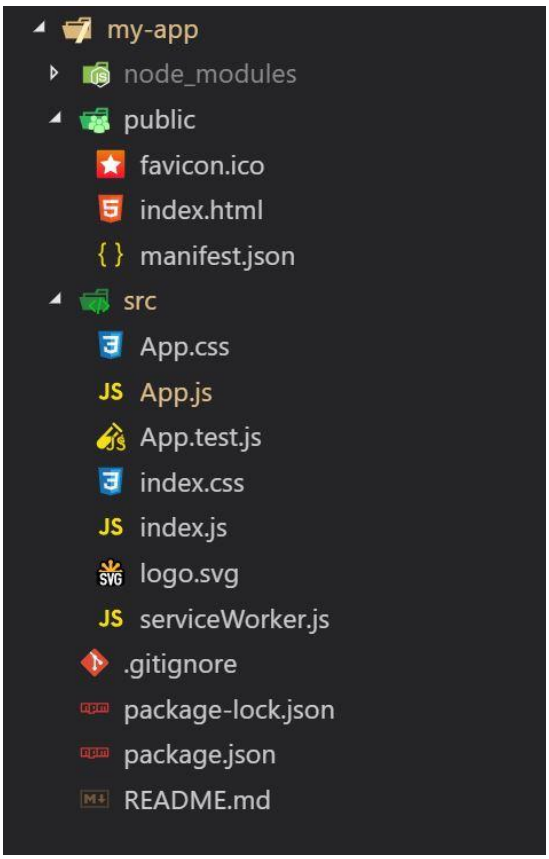
npm install create-react-app -g

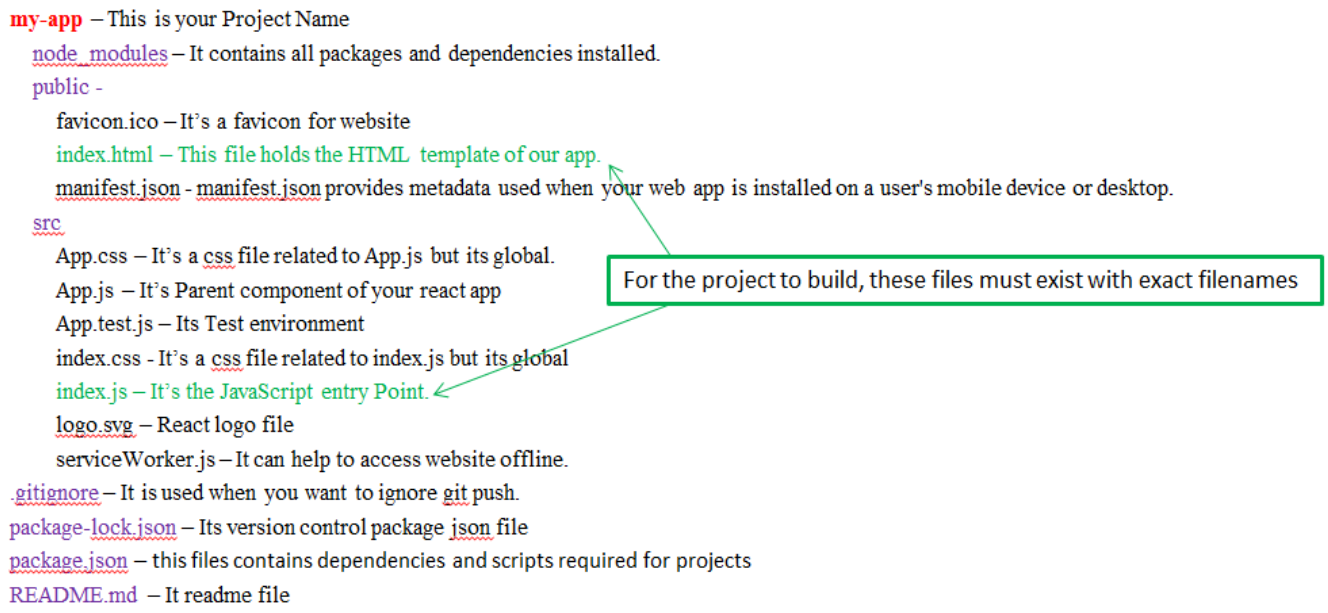
create-react-app app-name

npm start

File structure

Root level- 3 folder and 4 files





How Does Virtual DOM Work?

Like the actual DOM, the Virtual DOM is a node tree that lists elements and their attributes and content as objects and properties. React's render() method creates a node tree from React components and updates this tree in response to mutations in the data model, caused by actions.

Each time the underlying data changes in a React app, a new Virtual DOM representation of the user interface is created

This is where things get interesting. Updating the browser's DOM is a three-step process in React.

1. Whenever anything may have changed, the entire UI will be re-rendered in a Virtual DOM representation.
2. The difference between the previous Virtual DOM representation and the new one will be calculated.
3. The real DOM will be updated with what has actually changed. This is very much like applying a patch.

What is the benefit of Virtual DOM?

Each time you make a change in the code, DOM will be completely updated and rewritten. This is an expensive operation and consumes lots of time. In this point, **React** provides a solution: **The Virtual DOM.**

So when something changes:

- React first creates an exact copy of the DOM
- Then React figures out which part is new and only updates that specific part in the **Virtual DOM**
- Finally, React copies only the new parts of the **Virtual DOM** to the **actual DOM**, rather than completely rewriting it.

render() Method

render()- return null or some html code

The render() method is the only required method in a class component. It examines this.props and this.state .

It returns one of the following types:

React elements – These are created via JSX(Not required).

For example, and <App /> are React elements that instruct React to render a DOM node, or another user-defined component, respectively.

Arrays and fragments - It is used to return multiple elements from render.

Portals – It is used to render children into a different DOM subtree.

String and numbers - These are rendered as text nodes in the DOM.

Booleans or null - It renders nothing. (Mostly exists to support return test && <Child /> pattern, where test is boolean.)

Rendering is the process of transforming your react components into DOM (Document Object Model) nodes that your browser can understand and display on the screen.

DOM manipulation is extremely slow. In contrast, manipulating React elements is much, much faster. React makes the most of this by creating a virtual representation of what the DOM should look like called the Virtual DOM

React Element

React.createElement()- required min 3 parameters

1. String define html tag to be render ex. div tag
2. Pass any optional properties or not pass any value to send null
3. Children for the html ex i.e. for div tag

Using createElement() Method

React.createElement("h1", null, "Hello TQPP");

Using JSX

<h1>Hello TQPP</h1>

JavaScript XML (JSX)

JSX stands for JavaScript XML. It is a syntax extension to JavaScript.

JSX is a preprocessor step that adds XML syntax to JavaScript.

JSX produces React "elements". It is possible to create element without JSX but JSX makes React a lot more elegant.

It is recommended to use JSX with React to describe what the UI should look like.

JSX is easier to read and write. **Babel** transform these expressions into a actual JavaScript Code.

It also allows React to show more useful error and warning messages.

Way of Creatring Components

- Components are the building blocks of any React app.
- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.
- Always start component names with a capital letter.
- React treats components starting with lowercase letters as DOM tags. For example, represents an HTML div tag, but <App /> represents a component requires App to be in scope.

Function Components

It is a JavaScript function which accepts a single “props” object argument with data and returns a React Element.

Syntax:-

```
function  func_name ( ) {
  return  React Element;
}
```

Ex:-

```
Function  Student( ){
  return  <h1>Hello Jay</h1>
}
```

```
const  Student = ( ) =>{
  return <h1>Hello Jay</h1>
}
```

Syntax:-

```
Function  func_name (props) {
  return  React Element;
}
```

Ex:-

```
function  Student(props){
  return <h1>Hello Jay</h1>
}
function Student(props){
  return <h1>Hello {props.name}</h1>
}
const Student = (props) =>{
  return <h1>Hello {props.name}</h1>
}
```

Class Component

A class component requires you to extend from React.Component. The class must implement a render() member function which returns a React component to be rendered, similar to a return value of a functional component. In a class-based component, props are accessible via this.props.

Syntax:-

```
class class_name extends Component {
  render( ){
      return React Element
   }
}
```

Ex:-

```
class Student extends Component {
  render( ){
      return <h1>Hello Jay</h1>
   }
}
class Student extends Component {
  render( ){
       return <h1>Hello {this.props.name}</h1>
   }
}
```

Composing Components

Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail.

Ex:-

```
function Student(){
  return <h1>Hello Jay</h1>
}
function App( ){
  return (
   <div>
          <Student / >
          <Student / >
         <Student / >
   </div>
);   }
ReactDOM.render(<App />, document.getElementById(“root”));
```

Functional vs Class Component

Use functional components if you are writing a presentational component which doesn't have its own state or needs to access a lifecycle hook. You cannot use setState() in your component because Functional Components are plain JavaScript functions,

Simple functions
Use Functional components as much as possible
Absence of this' keyword
Solution without using state
Mainly responsible for the UI
Stateless/ Dumb/ Presentational

Use class Components if you need state or need to access lifecycle hook because all lifecycle hooks are coming from the React.Component which you extend from in class components.

More feature rich
Maintain their own private data - state
Complex UI logic
Provide lifecycle hooks
Stateful/ Smart/ Container.

Properties (props)

Props: - Reusing same component in different component then props (properties) come in picture

Props is optional input that the your component can accept it also allow the component to be dynamic

Pass your business logic from one component to another

If you have to don't know what is going to pass as props or if you have to pass in dynamic html content passes In between component tag and in the component definition simply render the content using **props.children .** If it all specify it is render the browser and if nothing is pass props. Children simply render nothing

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object "props".

Props with function component:

Greet.js

JS App.js | JS Hello.js | JS Greet.js | JS Welcome.js

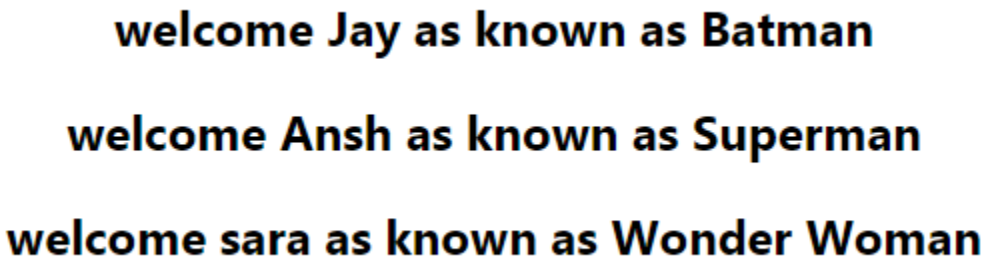
```
1   import React from 'react'
2
3   const Greet=(props )=>{
4       return(
5          <div>
6       <h1> Hello {props.name} as known as {props.heroName}</h1>
7
8       </div>
9       )
10  }
11   export default Greet;
```

App.js

JS App.js JS Hello.js JS Greet.js JS Welcome.js

```
1   import React from 'react';
2   import './App.css';
3   import Greet from './components/Greet';
4   import StateExample from './components/StateExample';
5   import Welcome from './components/Welcome';
6
7   function App() {
8     return (
9       <div className="App">
10
11          <Greet name="Jay" heroName="Batman" />
12          <Greet name="ansh" heroName="Superman" />
13          <Greet name="sara" heroName="Wonder Woman" />
```

Output:



Add children in component using: - **{props.children}**

Greet.js

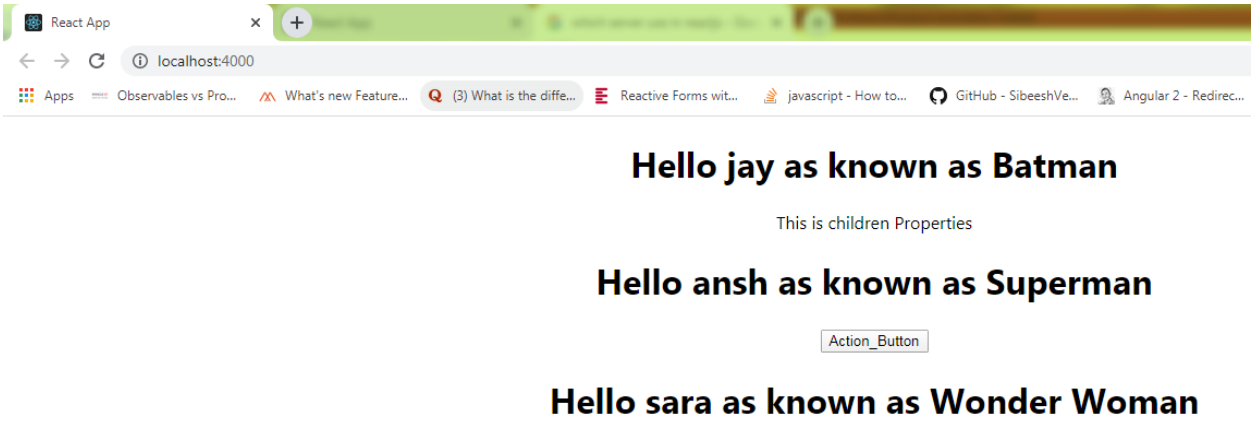
```
import React from 'react'

const Greet=(props )=>{
    return(
       <div>
    <h1> Hello {props.name} as known as {props.heroName}</h1>
    { props.children}
    </div>
    )
}
 export default Greet;
```

App.js

JS App.js ✖ | JS Hello.js | JS Greet.js | JS Welcome.js

```
1   import React from 'react';
2   import './App.css';
3   import Greet from './components/Greet';
4   import StateExample from './components/StateExample';
5   import Welcome from './components/Welcome';
6
7   function App() {
8     return (
9       <div className="App">
10          <Greet name="jay" heroName="Batman">
11            <p>This is children Properties</p>
12          </Greet>
13          <Greet name="ansh" heroName="Superman">
14            <button>Action_Button</button>
15          </Greet>
16          <Greet name="sara" heroName="Wonder Woman" />
```

Output:



Props with class component:

Unlike functional component were we specify the props parameter in class component the properties are available throw this.props which reserved in class components

Props are immutable i.e value can't be changed

Welcome.js

App.js | Hello.js | Greet.js | Welcome.js

```
1  import React,{Component} from 'react'
2  class Welcome extends Component
3  {
4      render()
5      {
6
7          return(
8     <h2> welcome {this.props.name} as known as {this.props.heroName}</h2>
9          );
10     }
11 }
12 export default Welcome
```

App.js

JS App.js ✖ JS Hello.js JS Greet.js JS Welcome.js

```
1   import React from 'react';
2   import './App.css';
3   import StateExample from './components/StateExample';
4   import Welcome from './components/Welcome';
5
6   function App() {
7     return (
8       <div className="App">
9         <Welcome name="Jay" heroName="Batman" />
10        <Welcome name="Ansh" heroName="Superman" />
11        <Welcome name="sara" heroName="Wonder Woman" />
12
13      </div>
14    );
15  }
```

Output: