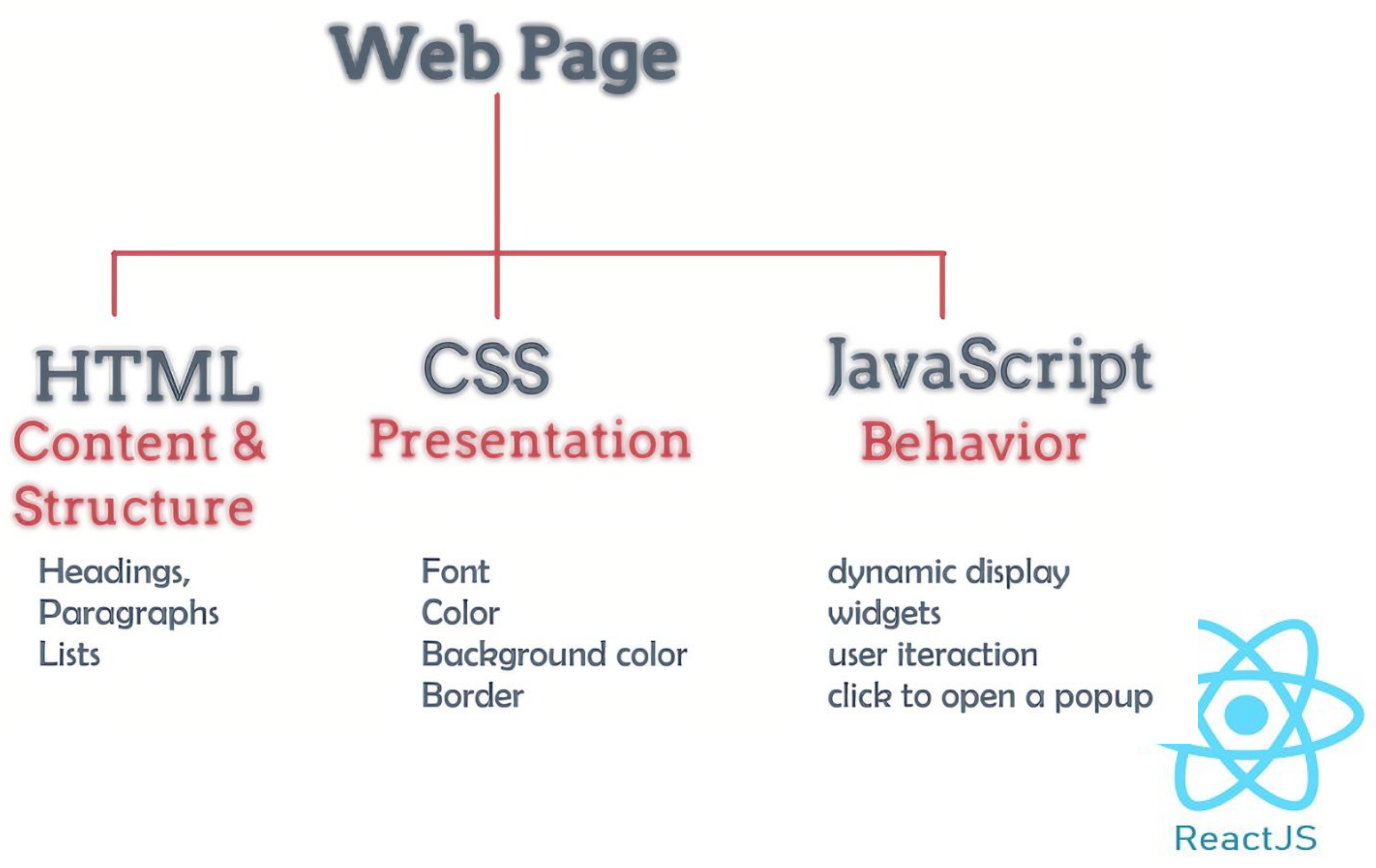
MERN: UNIT-1 React Fundamentals

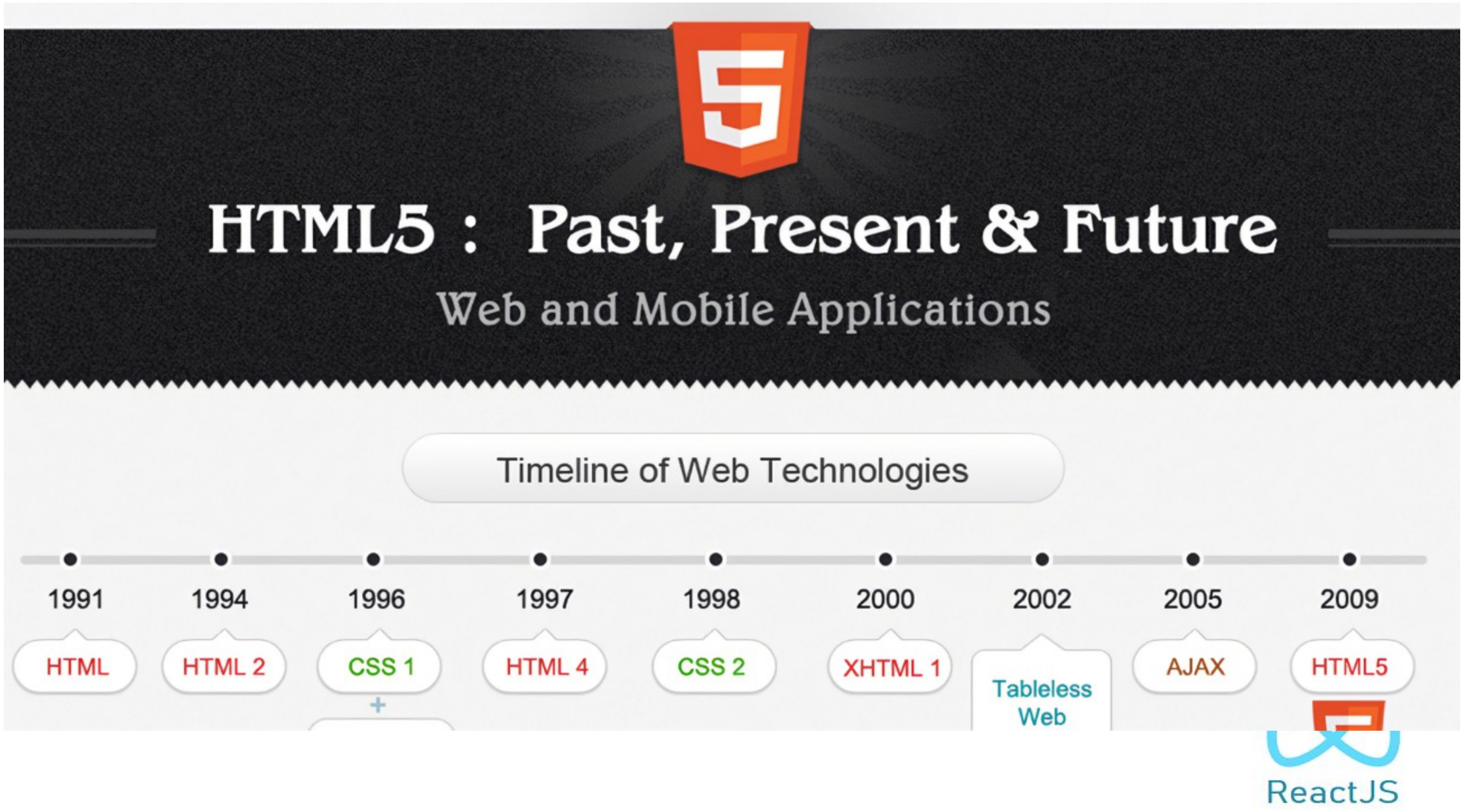
Prepared By,

M.Gouthamm, Asst.Prof, CSE, MRUH

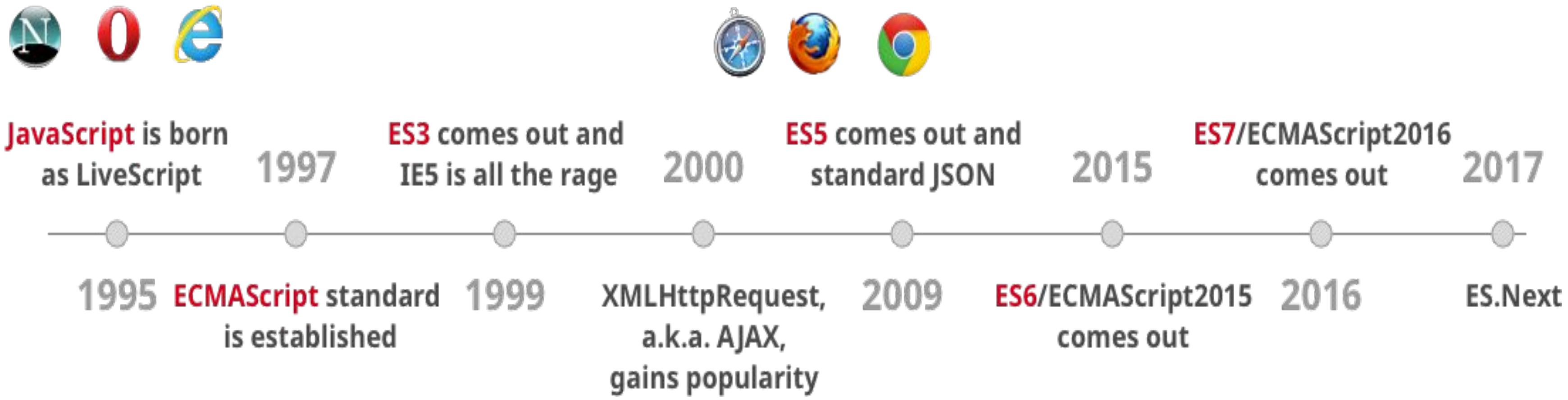
Introduction to React



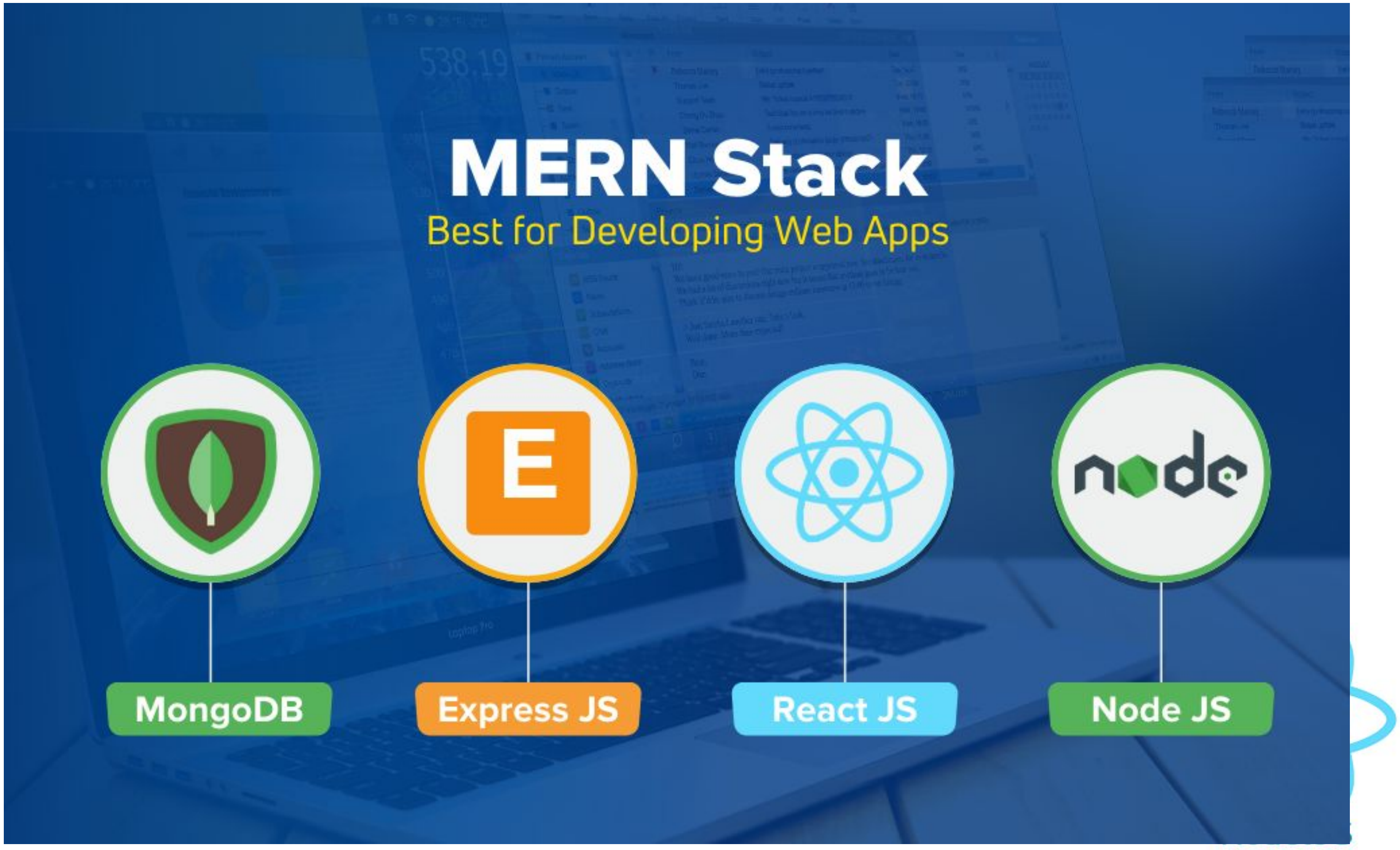
Introduction to React



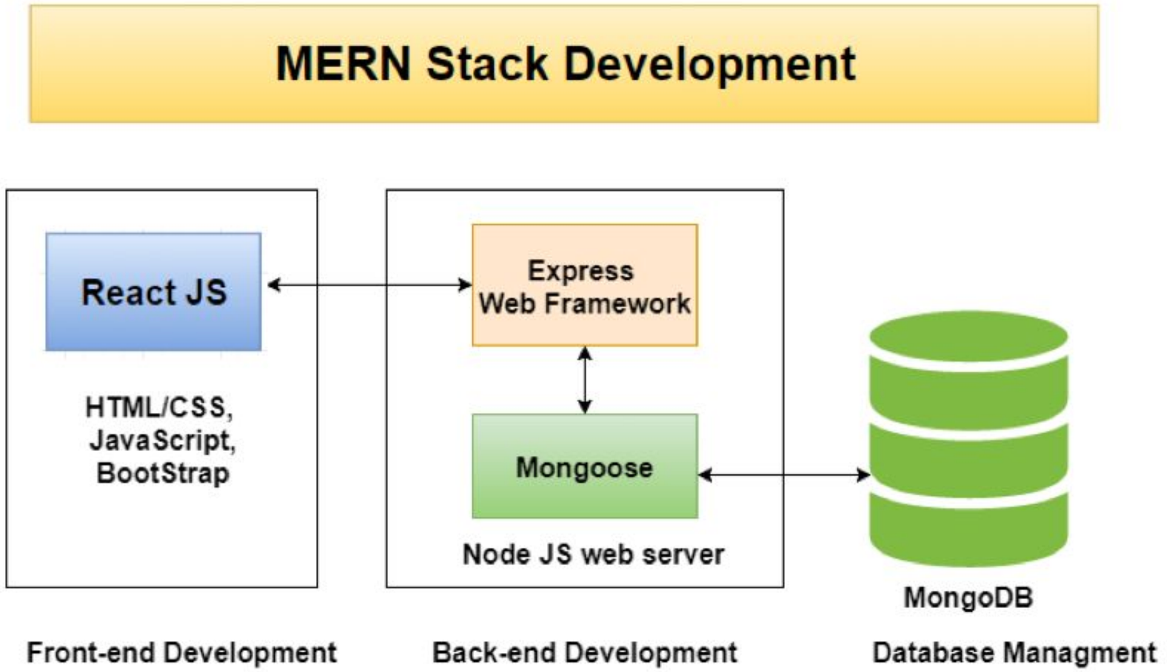
Introduction to React


JavaScript is born as LiveScript
1995
1997
ECMAScript standard is established
ES3 comes out and IE5 is all the rage
1999
2000
XMLHttpRequest, a.k.a. AJAX, gains popularity
ES5 comes out and standard JSON
2009
2015
ES6/ECMAScript2015 comes out
ES7/ECMAScript2016 comes out
2016
2017
ES.Next

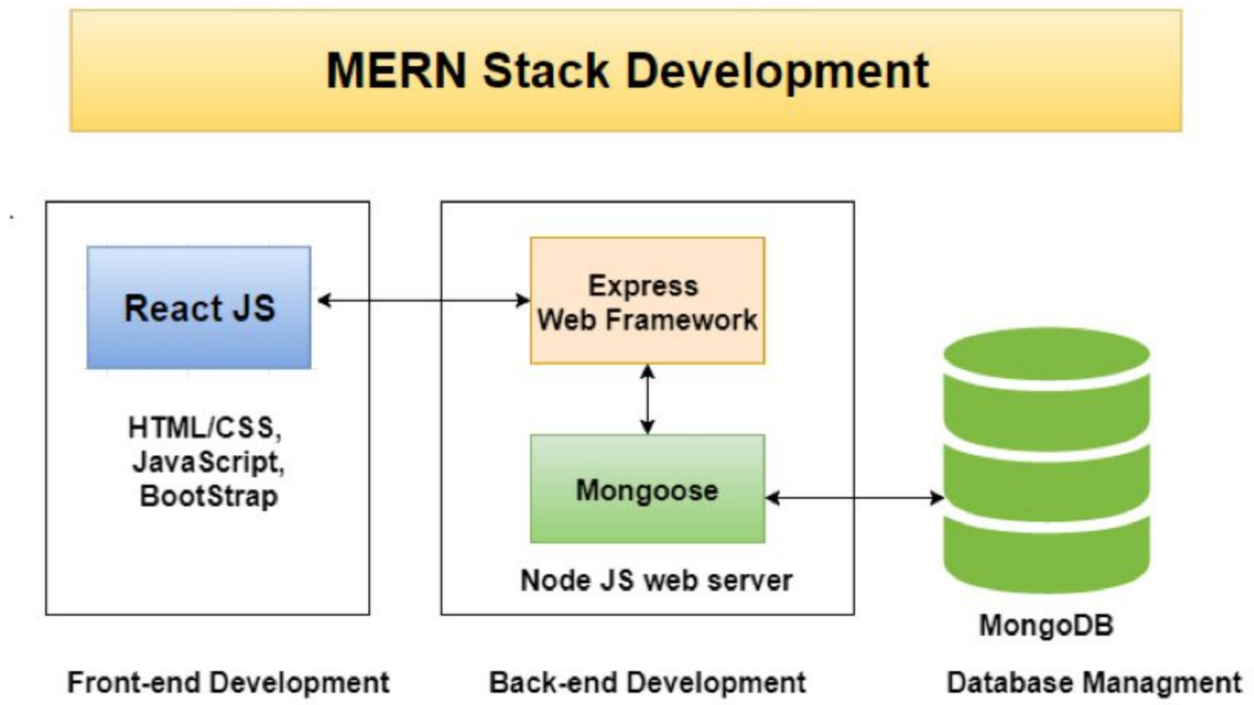
Introduction to React



Introduction to React

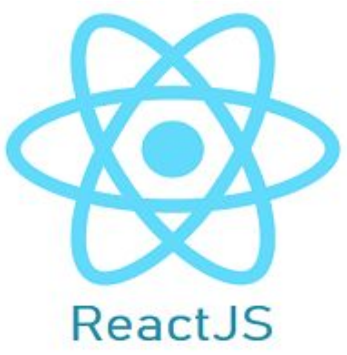


Introduction to React



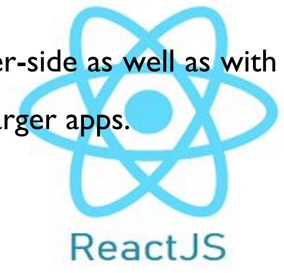
Introduction to React

- ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components.
- It is an open-source, component-based front end library which is responsible only for the view layer of the application.
- It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.



Why we use ReactJS?

- The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the apps. It uses virtual DOM (JavaScript object), which improves the performance of the app.
- The JavaScript virtual DOM is faster than the regular DOM. We can use ReactJS on the client and server-side as well as with other frameworks. It uses component and data patterns that improve readability and helps to maintain larger apps.



Introduction to React

- ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library responsible only for the view layer of the application. It was created by **Jordan Walke, who was a software engineer at Facebook.**

- It was initially developed and maintained by Facebook and was later used in its products like **WhatsApp & Instagram. Facebook developed ReactJS in 2011 in its newsfeed section**, but it was released to the public in the month of **May 2013 version 0.3.0**. The Current version of React is **18.2.0 / 14 June 2022**

- Today, most of the websites are built using MVC (model view controller) architecture. In MVC architecture, **React is the 'V' which stands for view,** whereas the architecture is provided by the Redux or Flux.

ReactJS

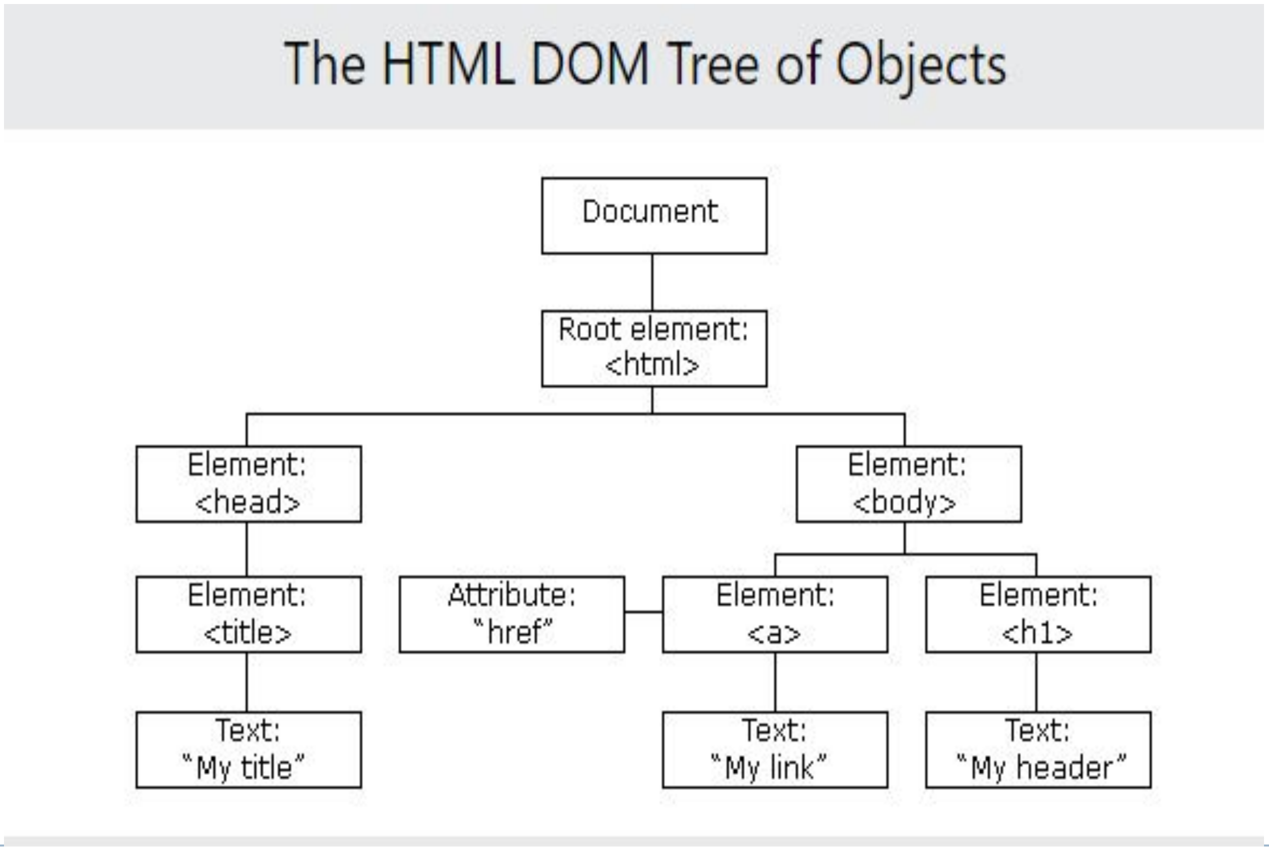
Why learn ReactJS?

- Today, many JavaScript frameworks are available in the market(like angular, node), but still, React came into the market and gained popularity amongst them. The previous frameworks follow the traditional data flow structure, which uses the DOM (Document Object Model). DOM is an object which is created by the browser each time a web page is loaded.
- It dynamically adds or removes the data at the back end and when any modifications were done, then each time a new DOM is created for the same page. This repeated creation of DOM makes unnecessary memory wastage and reduces the performance of the application.
- Therefore, a new technology ReactJS framework invented which remove this drawback. ReactJS allows you to divide your entire application into various components. ReactJS still used the same traditional data flow, but it is not directly operating on the browser's Document Object Model (DOM) immediately; instead, it operates on a virtual DOM.
- It means rather than manipulating the document in a browser after changes to our data, it resolves changes on a DOM built and run entirely in memory.

ReactJS

Why learn ReactJS?

- When a web page is loaded, the browser creates a Document Object Model of the page.
- The HTML DOM model is constructed as a tree of Objects:



React Environment Setup

- we will learn how to set up an environment for the successful development of ReactJS application.

Pre-requisite for ReactJS

1. NodeJS and NPM
2. React and ReactDOM
3. Webpack
4. Babel

Ways to install ReactJS

There are two ways to set up an environment for successful ReactJS application. They are given below.

1. **Using the npm command**
2. **Using the create-react-app command**

React create-react-app

- Starting a new React project is very complicated, with so many build tools. It uses many dependencies, configuration files, and other requirements such as Babel, Webpack, ESLint before writing a single line of React code.
- **Create React App** CLI tool removes all that complexities and makes React app simple.
- The **create-react-app** is an excellent tool for beginners, which allows you to create and run React project very quickly. It does not take any configuration manually.
- This tool is wrapping all of the required dependencies like **Webpack, Babel** for React project itself and then you need to focus on writing React code only.
- This tool sets up the development environment, provides an excellent developer experience, and optimizes the app for production.

React create-react-app

Requirements

The Create React App is maintained by **Facebook** and can works on any platform, for example, macOS, Windows, Linux, etc.

To create a React Project using **create-react-app**, you need to have installed the following things in your system.

1. **Node version**
2. **NPM version**

Run the following command to check the Node version in the command prompt.

```
1. node -v
```

Run the following command to check the NPM version in the command prompt.

```
1. npm -v
```

React create-react-app

Installation

- Here, we are going to learn how we can install React using CRA tool. For this, we need to follow the steps as given below.

Install React

- We can install React using npm package manager by using the following command. There is no need to worry about the complexity of React installation.
- The create-react-app npm package manager will manage everything, which needed for React project.

```
npm install -g create-react-app
```

Create a new React project

- Once the React installation is successful, we can create a new React project using create-react-app command. Here, I choose "reactproject" name for my project.

```
create-react-app reactproject
```

- The above command will take some time to install the React and create a new project with the name **"reactproject.“**
- Once the React project is created successfully on our system. Now, we need to start the server so that we can access the application on the browser. Type the following command in the terminal window.

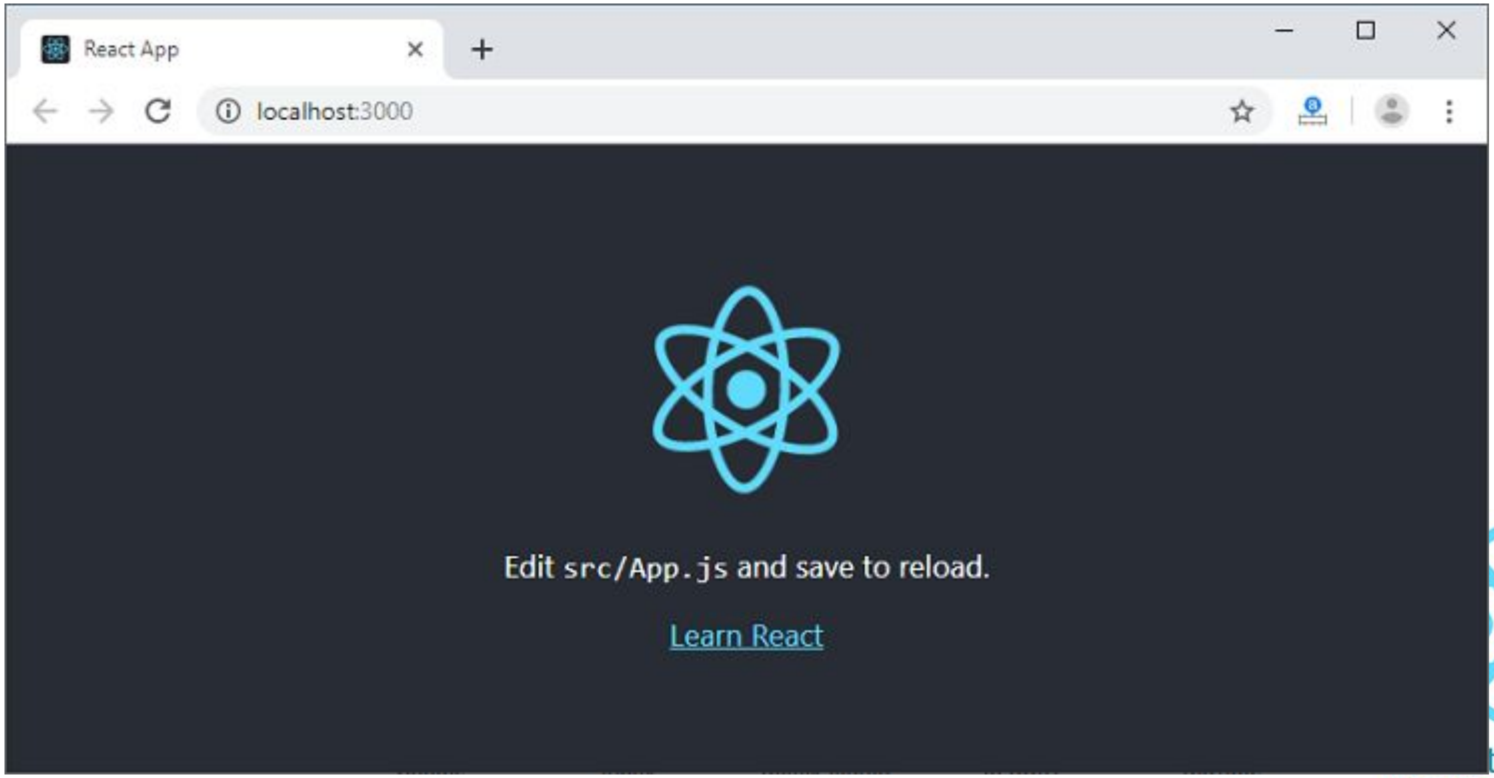
```
npm start
```

ReactJS

React create-react-app

- NPM is a package manager which starts the server and access the application at default server **http://localhost:3000.**

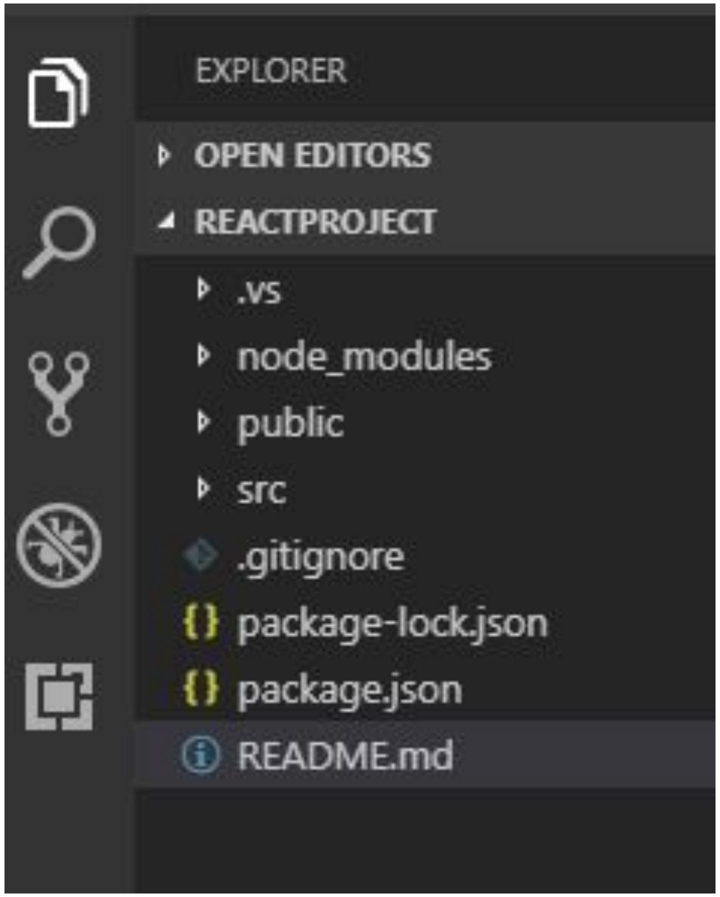
Now, we will get the following screen.



Edit src/App.js and save to reload.

React create-react-app

- Next, open the project on Code editor. Here, I am using Visual Studio Code. Our project's default structure looks like as below image.



```
EXPLORER
▸ OPEN EDITORS
◢ REACTPROJECT
  ▸ .vs
  ▸ node_modules
  ▸ public
  ▸ src
    .gitignore
  {} package-lock.json
  {} package.json
    README.md
```

React create-react-app

- In React application, there are several files and folders in the root directory. Some of them are as follows:

1. **node_modules:** It contains the React library and any other third party libraries needed.
2. **public:** It holds the public assets of the application. It contains the index.html where React will mount the application by default on the <div id="root"></div> element.
3. **src:** It contains the App.css, App.js, App.test.js, index.css, index.js, and serviceWorker.js files. Here, the App.js file always responsible for displaying the output screen in React.
4. **package-lock.json:** It is generated automatically for any operations where npm package modifies either the node_modules tree or package.json. It cannot be published. It will be ignored if it finds any other place rather than the top-level package.
5. **package.json:** It holds various metadata required for the project. It gives information to npm, which allows to identify the project as well as handle the project?s dependencies.
6. **README.md:** It provides the documentation to read about React topics.

React create-react-app

React Environment Setup

Now, open the src >> App.js file and make changes which you want to display on the screen. After making desired changes, save the file.

As soon as we save the file, Webpack recompiles the code, and the page will refresh automatically, and changes are reflected on the browser screen.

Now, we can create as many components as we want, import the newly created component inside the App.js file and that file will be included in our main index.html file after compiling by Webpack.

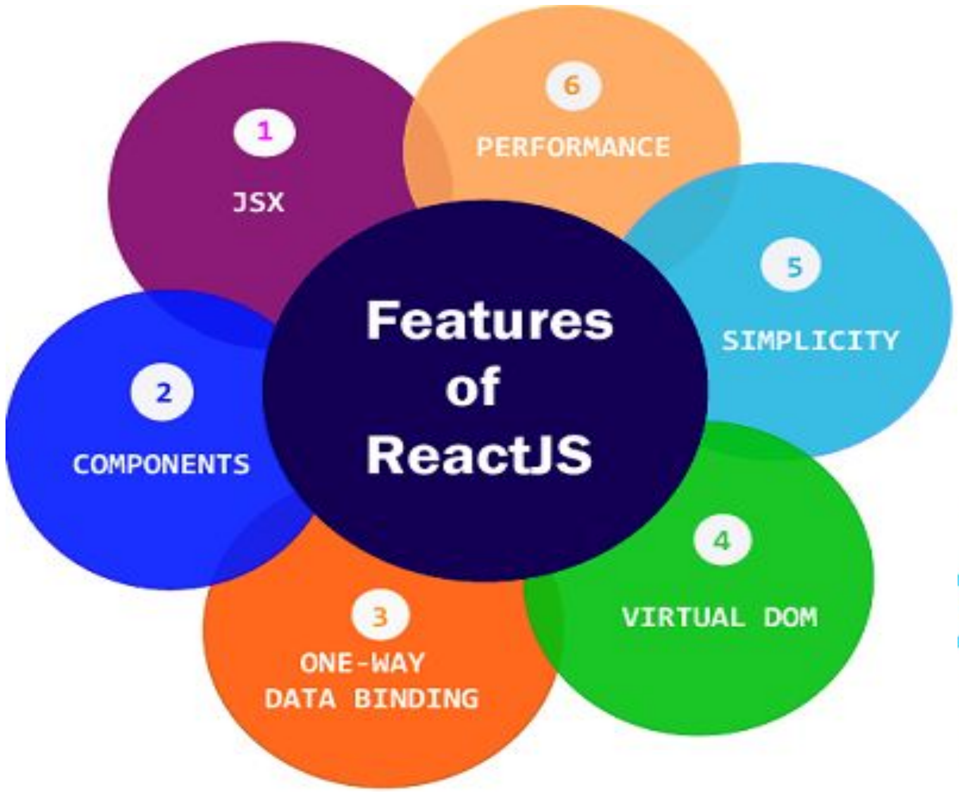
React Features

React Features

Currently, ReactJS gaining quick popularity as the best JavaScript framework among web developers. It is playing an essential role in the front-end ecosystem.

The important features of ReactJS are as following.

1. **JSX**
2. **Components**
3. **One-way Data Binding**
4. **Virtual DOM**
5. **Simplicity**
6. **Performance**



React Features

1. JSX

- JSX stands for JavaScript XML. It is a JavaScript syntax extension. Its an XML or HTML like syntax used by ReactJS. This syntax is processed into JavaScript calls of React Framework.
- It extends the ES6 so that HTML like text can co-exist with JavaScript react code. It is not necessary to use JSX, but it is recommended to use in ReactJS.

2. Components

- ReactJS is all about components. ReactJS application is made up of multiple components, and each component has its own logic and controls.
- These components can be reusable which help you to maintain the code when working on larger scale projects.

One-way Data Binding

- ReactJS is designed in such a manner that follows unidirectional data flow or one-way data binding. The benefits of one-way data binding give you better control throughout the application.
- If the data flow is in another direction, then it requires additional features. It is because components are supposed to be immutable and the data within them cannot be changed.
- Flux is a pattern that helps to keep your data unidirectional. This makes the application more flexible that leads to increase efficiency.

ReactJS

React Features

4. Virtual DOM

- A virtual DOM object is a representation of the original DOM object. It works like a one-way data binding. Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation.
- Then it checks the difference between the previous DOM representation and new DOM. Once it has done, the real DOM will update only the things that have actually changed. This makes the application faster, and there is no wastage of memory.

5. Simplicity

- ReactJS uses JSX file which makes the application simple and to code as well as understand.
- We know that ReactJS is a component-based approach which makes the code reusable as your need. This makes it simple to use and learn.

6. Performance

- ReactJS is known to be a great performer. This feature makes it much better than other frameworks out there today. The reason behind this is that it manages a virtual DOM.
- The DOM is a cross-platform and programming API which deals with HTML, XML or XHTML. The DOM exists entirely in memory. Due to this, when we create a component, we did not write directly to the DOM.
- Instead, we are writing virtual components that will turn into the DOM leading to smoother and faster performance.

ReactJS

Pros and Cons of ReactJS

Advantage of ReactJS

1. Easy to Learn and Use
2. Creating Dynamic Web Applications Becomes Easier
3. Reusable Components
4. Performance Enhancement
5. The Support of Handy Tools
6. Known to be SEO Friendly
7. The Benefit of Having JavaScript Library

Disadvantage of ReactJS

1. The high pace of development
2. Poor Documentation
3. View Part
4. JSX as a barrier

Pros and Cons of ReactJS

Advantage of ReactJS

1. **Easy to Learn and Use:** ReactJS is much easier to learn and use. Any developer who comes from a JavaScript background can easily understand and start creating web apps using React.
2. **Creating Dynamic Web Applications Becomes Easier:** To create a dynamic web application specifically with HTML was tricky, which requires complex coding, but React JS solved that issue and makes it easier. It provides less coding and gives more functionality.
3. **Reusable Components:** A ReactJS web application is made up of multiple components, and each component has its logic and controls. These components can be reused wherever you need them. The reusable code helps to make your apps easier to develop and maintain.
4. **Performance Enhancement:** ReactJS improves performance due to virtual DOM. The React Virtual DOM exists entirely in memory and is a representation of the web browser's DOM. Due to this, when we write a React component, we did not write directly to the DOM. Instead, we are writing virtual components that react will turn into the DOM, leading to smoother and faster performance.
5. **The Support of Handy Tools:** ReactJS support a handy set of tools which make the task of the developers understandable and easier. It also allows you to select particular components and examine and edit their current Props and State.

Pros and Cons of ReactJS

Advantage of ReactJS

6. Known to be SEO Friendly

- Traditional JavaScript frameworks have an issue in dealing with SEO. The search engines generally having trouble in reading JavaScript-heavy applications.
- Many web developers have often complained about this problem. ReactJS overcomes this problem that helps developers to be easily navigated on various search engines.
- It is because React.js applications can run on the server, and the virtual DOM will be rendering and returning to the browser as a regular web page.

7. The Benefit of Having JavaScript Library

- Today, ReactJS is choosing by most of the web developers. It is because it is offering a very rich JavaScript library.
- The JavaScript library provides more flexibility to the web developers to choose the way they want.

8. Scope for Testing the Codes

- ReactJS applications are extremely easy to test. It offers a scope where the developer can test and debug their codes with the help of native tools.

Pros and Cons of ReactJS

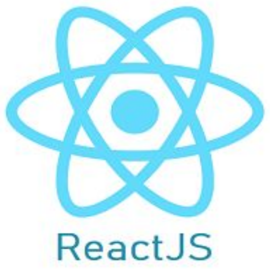
Disadvantage of ReactJS

1. The high pace of development

- The high pace of development has an advantage and disadvantage both. In case of disadvantage, since the environment continually changes so fast, some of the developers not feeling comfortable to relearn the new ways of doing things regularly.
- It may be hard for them to adopt all these changes with all the continuous updates. They need to be always updated with their skills and learn new ways of doing things.

2. Poor Documentation

- It is another cons which are common for constantly updating technologies. React technologies updating and accelerating so fast that there is no time to make proper documentation.
- To overcome this, developers write instructions on their own with the evolving of new releases and tools in their current projects.



Pros and Cons of ReactJS

Disadvantage of ReactJS

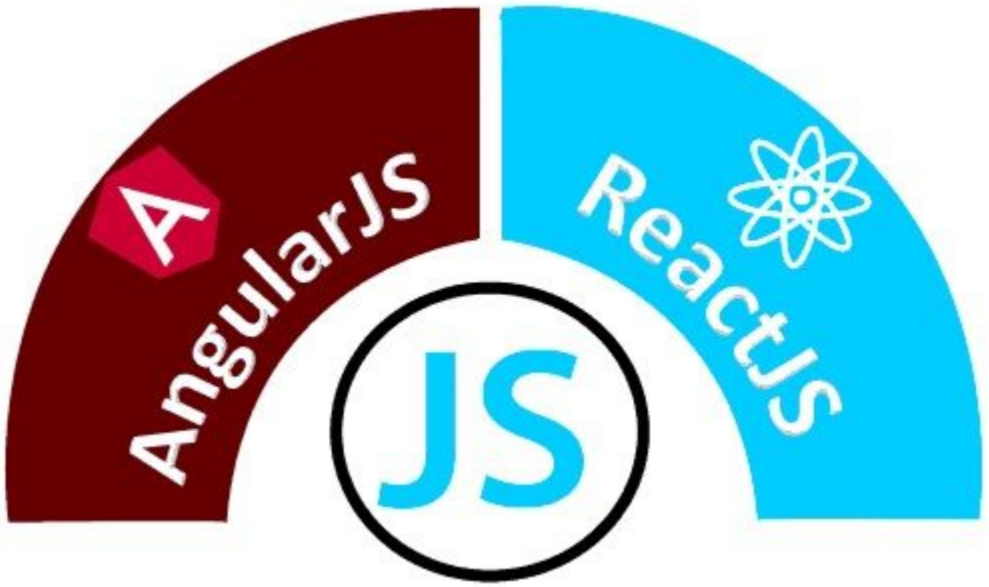
3. View Part

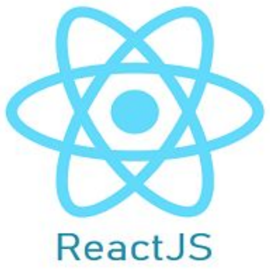
- ReactJS Covers only the UI Layers of the app and nothing else. So you still need to choose some other technologies to get a complete tooling set for development in the project.

4. JSX as a barrier

- ReactJS uses JSX. It's a syntax extension that allows HTML with JavaScript mixed together.
- This approach has its own benefits, but some members of the development community consider JSX as a barrier, especially for new developers. Developers complain about its complexity in the learning curve.

AngularJS Vs. ReactJS





AngularJS Vs. ReactJS

	AngularJS	ReactJS
Author	Google	Facebook Community
Developer	Misko Hevery	Jordan Walke
Initial Release	October 2010	March 2013
Latest Version	Angular 1.7.8 on 11 March 2019.	React 16.8.6 on 27 March 2019
Language	JavaScript, HTML	JSX
Type	Open Source MVC Framework	Open Source JS Framework
Rendering	Client-Side	Server-Side
Packaging	Weak	Strong
Data-Binding	Bi-directional	Uni-directional
DOM	Regular DOM	Virtual DOM
Testing	Unit and Integration Testing	Unit Testing
App Architecture	MVC	Flux
Dependencies	It manages dependencies automatically.	It requires additional tools to manage dependencies.
Routing	It requires a template or controller to its router configuration, which has to be managed manually.	It doesn't handle routing but has a lot of modules for routing, eg., react-router.
Performance	Slow	Fast, due to virtual DOM.
Best For	It is best for single page applications that update a single view at a time.	It is best for single page applications that update multiple views at a time.

ReactJS - Architecture

- React is a library to create user interface in a web application. React's primary purpose is to enable the **developer to create user interface using pure JavaScript**. Normally, every user interface library introduces a new template language (which we need to learn) to design the user interface and provides an option to write logic, either inside the template or separately.
- Instead of introducing new template language, React introduces three simple concepts as given below −

React elements

- JavaScript representation of HTML DOM. React provides an API, **React.createElement** to create React Element.

JSX

- A JavaScript extension to design user interface. JSX is an XML based, extensible language supporting HTML syntax with little modification. JSX can be compiled to React Elements and used to create user interface.

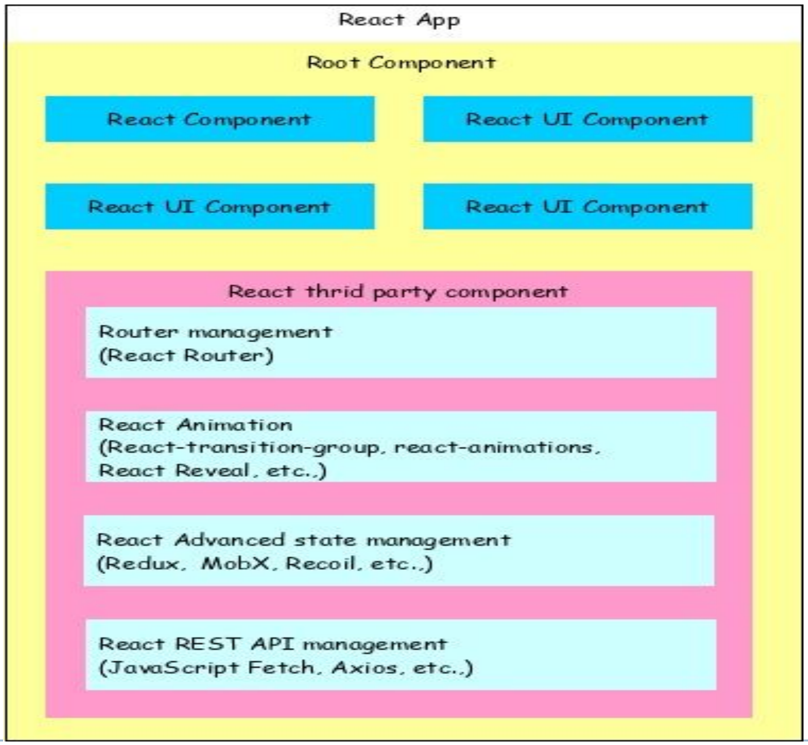
React component

- React component is the primary building block of the React application. It uses React elements and JSX to design its user interface. **React component is basically a JavaScript class** (extends the React. component class) or **pure JavaScript function.** React component has properties, state management, life cycle and event handler. React component can be able to do simple as well as advanced logic.

ReactJS - Architecture

Architecture of the React Application

- React library is just UI library and it does not enforce any particular pattern to write a complex application. Developers are free to choose the design pattern of their choice.
- React community advocates certain design pattern. One of the patterns is Flux pattern. React library also provides lot of concepts like Higher Order component, Context, Render props, Refs etc., to write better code.
- React Hooks is evolving concept to do state management in big projects.


React App
Root Component
React Component
React UI Component
React UI Component
React UI Component
React thrid party component
Router management (React Router)
React Animation (React-transition-group, react-animations, React Reveal, etc.)
React Advanced state management (Redux, MobX, Recoil, etc.)
React REST API management (JavaScript Fetch, Axios, etc.)

ReactJS - Architecture

1. React app starts with a single root component.
2. Root component is build using one or more component.
3. Each component can be nested with other component to any level.
4. Composition is one of the core concepts of React library. So, each component is build by composing smaller components instead of inheriting one component from another component.
5. Most of the components are user interface components.
6. React app can include third party component for specific purpose such as **routing, animation, state management, etc.**

ReactJS - Architecture

Example

Next, create a file, hello.html and write a simple React application.

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8" />
      <title>React Application</title>
   </head>
   <body>
      <div id="react-app"></div>
      <script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin></script>
      <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js" crossorigin></script>
      <script language="JavaScript">
         element = React.createElement('h1', {}, 'Hello React!')
         ReactDOM.render(element, document.getElementById('react-app'));
      </script>
   </body>
</html>
```

ReactJS - Architecture

Output

Next, open your favorite browser. Enter http://localhost:3000 in the address bar and then press enter.

Here, we are using two API provided by the React library.

React.createElement

Used to create React elements. It expects three parameters –

1. Element tag
2. Element attributes as object
3. Element content - It can contain nested React element as well

ReactDOM.render

Used to render the element into the container. It expects two parameters –

1. React Element OR JSX
2. Root element of the webpage

ReactJS - Architecture

Nested React element

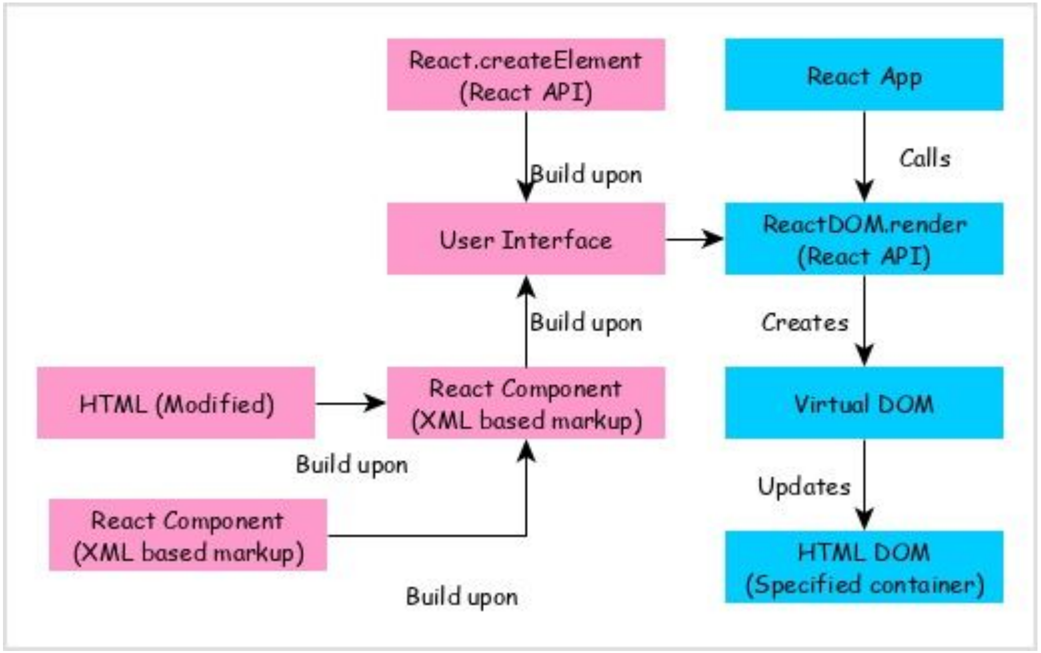
As React.createElement allows nested React element, let us add nested element as shown below −

Example

```
<script language="JavaScript">
   element = React.createElement('div', {}, React.createElement('h1', {}, 'Hello React!'));
   ReactDOM.render(element, document.getElementById('react-app'));
</script>
```

ReactJS - Architecture

By analyzing the application, we can visualize the workflow of the React application as shown in the below diagram.


React.createElement (React API)
Build upon
User Interface
Build upon
HTML (Modified)
React Component (XML based markup)
Build upon
React Component (XML based markup)
Build upon
React App
Calls
ReactDOM.render (React API)
Creates
Virtual DOM
Updates
HTML DOM (Specified container)

React app calls ReactDOM.render method by passing the user interface created using React component (coded in either JSX or React element format) and the container to render the user interface.

ReactDOM.render processes the JSX or React element and emits Virtual DOM.

Virtual DOM will be merged and rendered into the container.

React JSX (Javascript XML)

- All of the React components have a render function. The render function specifies the HTML output of a React component. **JSX(JavaScript XML)**, is a React extension which allows writing JavaScript code that looks like HTML.
- In other words, JSX is an HTML-like syntax used by React that extends ECMAScript so that HTML-like syntax can co-exist with JavaScript/React code.
- The syntax is used by preprocessors (i.e., transpilers like babel) to transform HTML-like syntax into standard JavaScript objects that a JavaScript engine will parse.
- JSX provides you to write HTML/XML-like structures (e.g., DOM-like tree structures) in the same file where you write JavaScript code, then preprocessor will transform these expressions into actual JavaScript code.
- Just like XML/HTML, JSX tags have a tag name, attributes, and children.
- JSX makes React code more readable and expressive by combining UI markup and JavaScript logic seamlessly. (File Extension is .jsx instead of .js)

Example

- Here, we will write JSX syntax in JSX file and see the corresponding JavaScript code which transforms by preprocessor(babel).

JSX File: <div>Hello World!!!</div>

Corresponding Output

```
React.createElement("div", null, "Hello World!!!")
```

React JSX (Javascript XML)

Why JSX?

- Using JSX with React is not mandatory, as there are various options to achieve the same thing as JSX; but it is helpful as a visual aid while working with UI inside a JavaScript code.

1. JSX performs optimization while translating the code to JavaScript, making it faster than regular JavaScript.

2. React uses components that contain both markup and logic in a single file, instead of separate files.

3. Most of the errors can be found at compilation time, as the data flow is unidirectional.

4. Creating templates becomes easier with JSX.

5. We can use JSX inside of conditional statements (if–else) and loop statements (for loops), can assign it to variables, accept it as arguments, or return it from functions.

6. Using JSX can prevent Cross Site Scripting attacks, or injection attacks.

React JSX (Javascript XML)

Nested Component without JSX

Example:

MyComponentWithoutJSX.js

```
import React from 'react';

// Component without JSX
const MyComponentWithoutJSX = () => {
  return React.createElement('div', null, [
    React.createElement('h1', null, 'Hello, World!'),
    React.createElement('p', null, 'This is a React component without JSX.'),
  ]);
}
export default MyComponentWithoutJSX;
```

App.js

```
import MyComponentWithoutJSX from './Components/MyComponentWithoutJSX';
<MyComponentWithoutJSX/>
```

React JSX (Javascript XML)

1. HTML-Like Syntax:

```
const element = <h1>Hello, JSX!</h1>;
```

2. Nested Elements in JSX

To use more than one element, you need to wrap it with one container element. Here, we use div as a container element which has three nested elements inside it.

App.JSX

```
<div>
    <h1>Malla Reddy University</h1>
    <h2>Department of CSE</h2>
    <p>Welcome to MERN Stack Classes!!!!!</p>
  </div>
```

React JSX (Javascript XML)

3. Attributes in JSX

- JSX supports HTML like attributes. All HTML tags and its attributes are supported. Attributes has to be specified using camelCase convention (and it follows JavaScript DOM API) instead of normal HTML attribute name.
- For example, class attribute in HTML has to be defined as className.

The following are few other examples −

- htmlFor instead of for
- tabIndex instead of tabindex
- onClick instead of onclick

Here are some key points about JSX attributes:

1. Basic Attribute Syntax:

- JSX attributes are written using the same syntax as HTML attributes, within the opening tag of an element.
- JSX attributes are written using camelCase. For example, class in HTML becomes className in JSX.

App.JSX

```
<div>
    <h1 className=“my-class”>Malla Reddy University</h1>
   <h2 style={{color:’red’}}>Department of CSE</h2>
    <p>Welcome to MERN Stack Classes!!!!!</p>
  </div>
```

React JSX (Javascript XML)

4. Expressions in JSX

JSX supports expression in pure JavaScript syntax. Expression has to be enclosed inside the curly braces, { }.

Expression can contain all variables available in the context, where the JSX is defined. Let us create simple JSX with expression.

Example:

```
const name = 'John';
const age = 30;
<h1 style={{color:'red', fontWeight:'120px'}}>Hello, {name}!</h1>
<p className="p">You are {age} years old.</p>
```

5. Functions in JSX

JSX supports user defined JavaScript function. Function usage is similar to expression. Let us create a simple function and use it inside JSX.

Example:

```
var cTime = new Date().toTimeString();
<div><p>The current time is {cTime}</p></div>,
```

React JSX (Javascript XML)

6. Conditional Rendering:

You can use JavaScript's conditional operators like if and ternary expressions to conditionally render JSX elements.

Example:

```
const isLoggedIn = true;
<h2>{isLoggedIn ? 'Welcome back!!!!' : 'Please log in...'}</h2>
```

7. Fragments:

JSX allows you to return multiple elements with wrapping them in a single parent element using React Fragments (<>...</> or <React.Fragment>...</React.Fragment>).

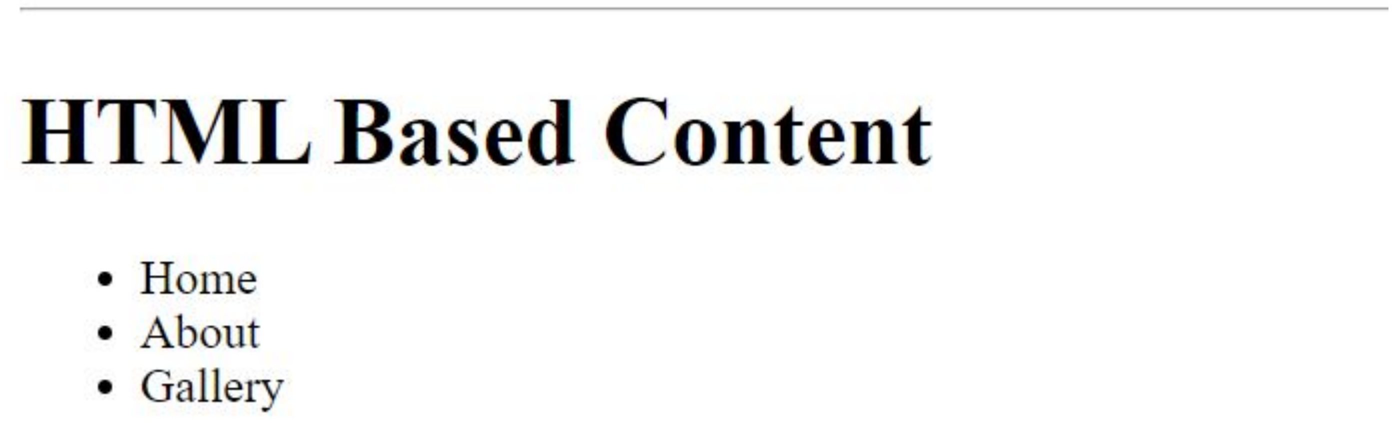
Example:

```
const element = (
  <>
    <h1>Heading 1</h1>
    <p>Paragraph 1</p>
  </>
);
```

React JSX (Javascript XML)

Example:

Create a web page with unordered list page names Home , About, Gallery



HTML Based Content

- Home
- About
- Gallery

React JSX (Javascript XML)

HTML Code:

```
<div class="container">
      <h1>HTML Based Content</h1>
      <ul>
         <li>Home</li>
         <li>About</li>
         <li>Gallery</li>
      </ul>
   </div>
```

React JSX (Javascript XML)

Javascript Code:

```
<h1>JavaScript Based HTML Content</h1>
   <script>
      let ul=document.createElement('ul');
      let li1=document.createElement('li');
      li1.innerText="Home";
      let li2=document.createElement('li');
      li2.innerText="About";
      let li3=document.createElement('li');
      li3.innerText="Gallery";
      ul.appendChild(li1);
      ul.appendChild(li2);
      ul.appendChild(li3);
      document.body.append(ul);
   </script>
```

ReactJS

React JSX (Javascript XML)

Without JSX Code in React:

Modify index.js

```
Using CreateElement

let h1=React.createElement('h1',{},"React Based HTML Content without JSX Code using createElement");
let li1=React.createElement('li',{},"Home");
let li2=React.createElement('li',{},"About");
let li3=React.createElement('li',{},"Gallery");
let ul=React.createElement('ul',{},li1,li2,li3);
let div=React.createElement('div',{},h1,ul);
```

React JSX (Javascript XML)

With JSX Code in React:

Modify index.js

```
Using Nested Element
let element=(
  <div>
    <h1>React Based HTML Content with JSX Code using Nested Elements....</h1>
    <ul>
      <li>Home</li>
      <li>About</li>
      <li>Gallery</li>
    </ul>
  </div>
);
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
 //div
element);
```

React JSX

With JSX Code in React:

Modify App.js

```
function App() {
return(
      <div>
          <h1>React Based HTML Content with JSX Code using Nested Elements....</h1>
          <ul>
            <li>Home</li>
            <li>About</li>
            <li>Gallery</li>
          </ul>
        </div>
      );
} export default App;
```

index.js

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<React.StrictMode>
           <App />    </React.StrictMode>
);
```

Naming Conventions

Naming conventions vary across programming languages and communities, but here are some common naming conventions similar to CamelCase:

1. CamelCase:

CamelCase is a convention where compound words or phrases are written without spaces, and each word or abbreviation within the phrase begins with a capital letter.
The first letter may or may not be capitalized.

Example:

```
myVariableName, calculateTotalPrice, getUserInfo
```

2. PascalCase:

Similar to CamelCase, but the first letter of the compound word is also capitalized.

Example:

```
MyVariableName, CalculateTotalPrice, GetUserInfo
```

Naming Conventions

3. Snake_case (or snake_case):

Words are written in lowercase letters, separated by underscores.

Example:

```
my_variable_name, calculate_total_price, get_user_info
```

4. UPPER_SNAKE_CASE (or SCREAMING_SNAKE_CASE):

Words are written in uppercase letters, separated by underscores.

Example:

```
MY_VARIABLE_NAME, CALCULATE_TOTAL_PRICE, GET_USER_INFO
```

5. kebab-case (or dash-case or hyphen-case):

Words are written in lowercase letters, separated by hyphens.

Example:

```
my-variable-name, calculate-total-price, get-user-info.
```

ReactJS

React Components

- Earlier, the developers write more than thousands of lines of code for developing a single page application. These applications follow the **traditional DOM structure**, and making changes in them was a very challenging task.

- If any mistake found, it **manually searches the entire application** and update accordingly. The **component-based approach** was introduced to overcome an issue.

- In this approach, the entire application is divided into a **small logical group of code**, which is known as components.

- A Component is considered as the core **building blocks** of a React application. It makes the task of building UIs much easier.

- Each component exists in the **same space, but they work independently** from one another and merge all in a parent component, which will be the final UI of your application.

- A React component represents a **small chunk of user interface** in a webpage. The primary job of a React component is to **render its user interface and update** it whenever its internal state is changed.

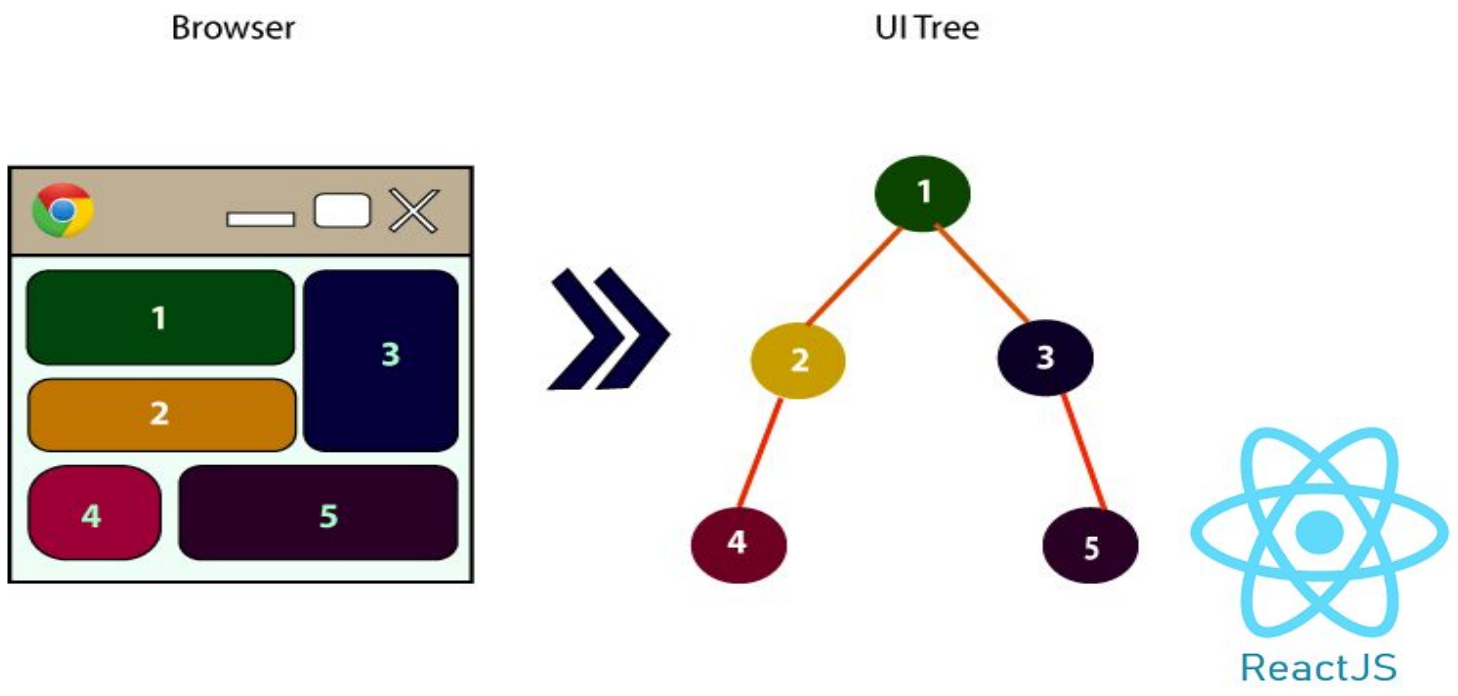
- In addition to rendering the UI, it **manages the events** belongs to its user interface.

ReactJS

React Components

- Every React component have their **own structure, methods** as well as APIs. They can be reusable as per your need.

- For better understanding, consider the entire UI as a tree. Here, the **root is the starting component**, and each of the **other pieces becomes branches**, which are further divided into sub-branches.



React Components

React component provides below functionalities.

- **Initial rendering of the user interface.**
- **Management and handling of events.**
- **Updating the user interface whenever the internal state is changed.**

React component accomplish these feature using three concepts −

1. **Properties** − Enables the component to receive input.

2. **Events** − Enable the component to manage DOM events and end-user interaction.

3. **State** − Enable the component to stay stateful. Stateful component updates its UI with respect to its state.

There are two types of components in React. They are −

1. **Function Components**

2. **Class Components**

React Components

Function Components

- A function component is literally defined as JavaScript functions. In React, function components are a way to write components that only contain a render method and don't have their own state.
- They are simply JavaScript functions that may or may not receive data as parameters. We can create a function that takes props(properties) as input and returns what should be rendered.
- This React component accepts a single object argument and returns a React element. Note that an element in React is not a component, but a component is comprised of multiple elements.

Following is the syntax for the function component in React:

Welcome.js

```
const Functioncomponent=() => {
    const greet = "Hi, This is Function Based Component!!!";
    const greet1 = "Hello "
    return (
        <div className="container">
            <h3 style={{color:'red'}}>Hello, This is First Component!!!!</h3>
            <h4 style={{color:'blue', fontSize:‘25px'}}> {greet}</h4>
        </div>
    );
}export default Functioncomponent;
```

React Components

Student-info.js

```
import React from "react";

const Studentinfo=()=>{
return(
    <h3>Create a table and display Student Details </h3>
      // Add Code Here

);
}
export default Studentinfo;
```

React Components

Class Components

Class components are more complex than functional components. It requires you to extend from React.

Component and create a render function which returns a React element.

You can pass data from one class to other class components. You can create a class by defining a class that extends Component and has a render function.

Example:

Classcomponent.js

```
import React from "react";
class MyClassComponent extends React.Component {
    render() {
      return (
        <div>
            <h2>This is an example of  Class component.</h2>
     <h3>Create a table and display Employee Details </h3>
          //Add Code Here
            <First/>
            <Second/>
        </div
      );
  }
} export default MyClassComponent;
```

React Components

```
class First extends React.Component {
    render() {
       return (
          <div>
             <h3>First Class Component....</h3>
          </div>
       );
    }
 }

 class Second extends React.Component {
    render() {
       return (
          <div>
             <h3>Second Class Component....</h3>
          </div>
       );
    }
 }
```

React Components

App.js

```
import Firstcomponent from './Components/Welcome';
import Studentinfo from './Components/student-info';

        <h3>Components</h3>
          <div class="container-fluid">
            <ol>
              <li >Function Components</li>
              <ol>
                <li><Functioncomponent/></li>
                <li><Studentinfo/></li>
              </ol>
                <li>Class Based Components</li>
          <ul>
                  <li><MyClassComponent/></li>
                <ul>
            </ol>
          </div>
```

React Components

Render component react

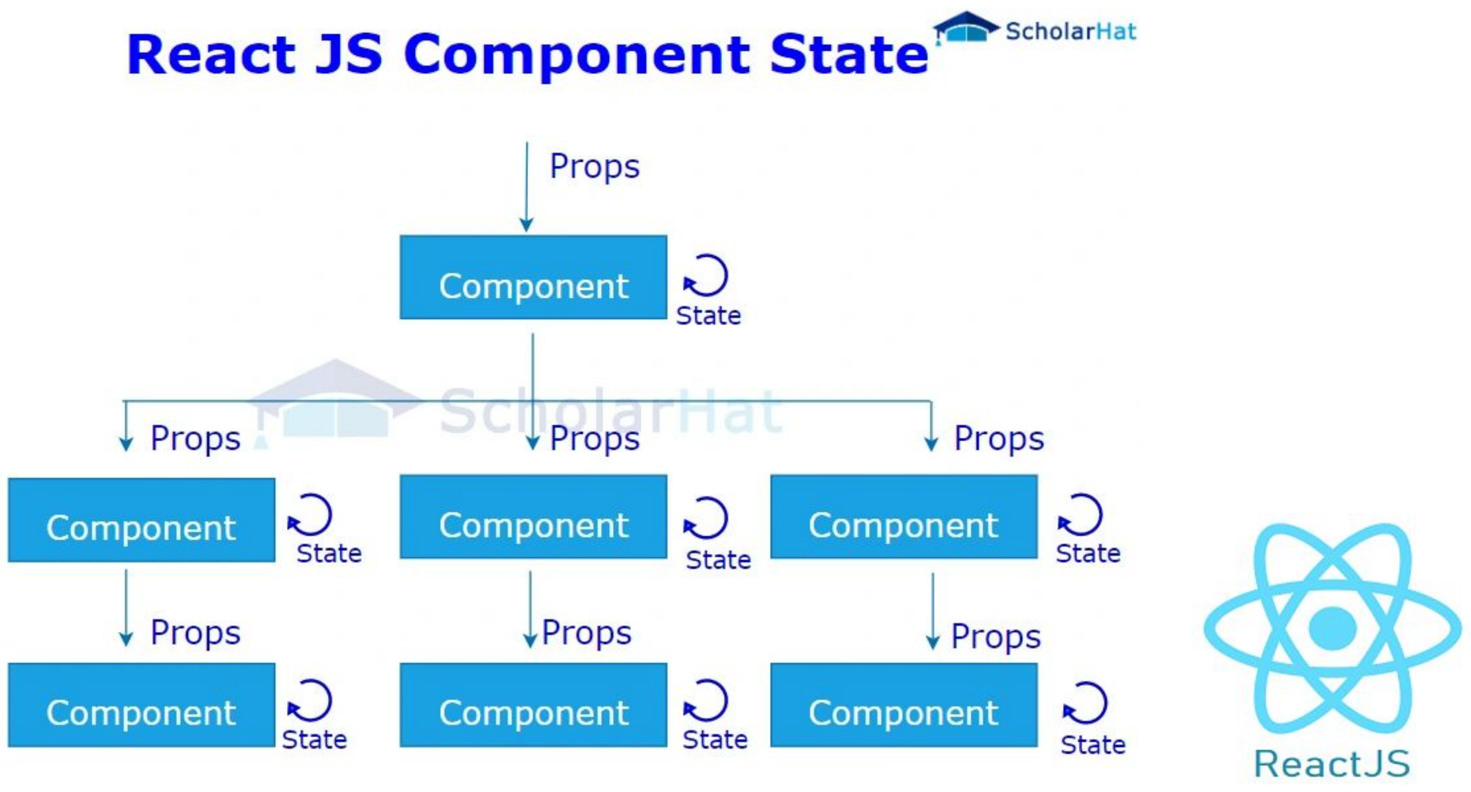
- Render component react entails putting them on the user interface. The core building blocks of React apps are components, which contain reusable and independent UI elements.

- A component is instantiated and displayed on the screen when it is rendered. The Virtual DOM, used by React, is a digital replica of the actual Document Object Model (DOM).

- When the state or props of the application change, React can effectively update and re-render only the relevant components.

React Components

A React component is in charge of maintaining its own state and events while producing a portion of the user interface. It can be viewed as a standalone component that receives data, processes it, and then outputs a React element that specifies what should be shown on the screen.



React Props

- Props stand for "Properties." They are read-only components. It is an **object which stores the value of attributes of a tag** and work similar to the HTML attributes. It gives a way to **pass data from one component to other components**. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.
- Props are immutable so we cannot modify the props from inside the component. Using the **"this.props"**, we can make the props available inside the components. Then, the dynamic data can be rendered through the render method. We need to add the props to the **reactDOM.render() in the Main.js file** of our ReactJS project of ReactJS if we need immutable data in the component. Then we can use it in the component where we want to use those dynamic data.

Below is the syntax:

// Passing Props

```
<ComponentName property1="value" property2="value" property3="value" />
```

Syntax:

// Accessing props

```
props.propName;
```

- The '**this.props**' is a kind of global object which stores all of a component's props. we can also pass props **to function-based components and class-based components.**
- To access a prop from a function we do not need to use the 'this' keyword anymore. Functional components accept props as parameters and can be accessed directly.

React Props

Example-1:

Propcomponent.js

```
import React from "react";
const Propscomponent=(props)=>{
    return(
        <div>
            <h2 className="text-danger">Hello,
           <span style={{color:'blue'}}>{props.name}</span>
      </h2>
           <h3>Welcome to MERN Stack Development Class</h3>
        </div>
    );
}
export default Propscomponent;
```

React Props

Example-2:

Propcomponent.js

```
import React from "react";
const Propscomponent=(props)=>{
    return(
        <div>
     <div className="container" >
            <h4>Example-2:</h4>
            <h2 className="text-danger">Hello,
          <span style={{color:'blue'}}>{props.name}</span>
     </h2>
            <h2 className="text-danger">Your Hall-ticket No. is:
          <span style={{color:'blue'}}>{props.hallticket}</span>
     </h2>
        </div>
     </div>
    );
}export default Propscomponent;
```

React Props

Example- 1 & 2:

App.js

```
function App(){
Return(
      <div className=“App">
      <div class="container">
              <h2>Props in React</h2>
              <Propscomponent name="Students"/>
              <Propscomponent name="Student" hallticket="2211CS010001"/>
           </div>
      </div>

);
}
```

React Props

Example-3:

Propcomponent.js

// Props through Functions

```
import React from "react";
const Propscomponent=(props)=>{
 console.log(props);
    return(
        <div>
      <div className="container" >
           <h1>Fucntion Based Props....</h1>
                 <h3>User Name:{props.uname} </h3>
                 <h3>Age:{props.age} </h3>
      </div>
     </div>
    );
}export default Propscomponent;
```

React Props

Example-3:

Propcomponent.js

```
// Props through classes
import React from "react“;
class PropsClass extends React.Component{
render(){
    return(
        <div>
                <h1>Class Based Props....</h1>
                <h3>User Name:{this.props.uname} </h3>
                <h3>Age:{this.props.age} </h3>
            </div>
    );
}}export default PropsClass;
```

App.js

```
<div>
     <h2>Props in React......</h2>
          <PropsClass uname="Jack" age="45"/>
</div>
```

React Props

Example -4:

Assignment

- Create a Student component and able to display Student details which vary in htno, name, mailid, and mobileno to users.

Hint: using Object destructing render the student data.

React Props

Example -4:

Solution: Studemtinfo.js

```
const StudentComponent=(props)=>{
    console.log(props);
    const {student}=props;
    return(
     <div>
               <h1>Object Destructing through props...</h1>
               <table className=" table table-striped">
                   <tbody>
                       <tr> <td>RollNo.:</td> <td>{student.RollNo}</td> </tr>
                       <tr> <td>Name:</td>    <td>{student.Name}</td>    </tr>
                       <tr> <td>Mobile:</td>  <td>{student.Mobile}</td>  </tr>
                       <tr> <td>Email:</td>   <td>{student.Email}</td>  </tr>
                   </tbody>
               </table>
           </div>
);}export default StudentComponent;
```

React Props

Example -4:

Solution: App.js

```
function App() {

const student = {
    RollNo: "2211CS010001",
    Name: "ABC",
    Mobile: "12345678",
    Email:"abc@gmail.com"
  }
return (
    <div className="App">
      <h2>Props in React......</h2>
          <div>   <StudentComponent student = {student}/>         </div>
    </div>
  );
}
```

State in React

- State in React concept is essential for enabling components to handle and store data. A component's state can vary over time and is represented by it, making it dynamic and interactive. Stateful components, which govern the application's behavior and rendering, are at the core of React.

- Any type of data, including characters, numbers, booleans, arrays, and even objects, can be stored in the state. The **setState() method** can be used to update it after initialization, which sets default values. React forces a re-rendering of the component when the state changes, altering the UI to reflect the new state.

State in React Components

- A default set of values must be declared in the component's constructor to define the state in a React component. The **this.state** object contains the state's data.

In React, there are two ways to define state:

1. **Class components**
2. **Hooks**

State in React

Here's a basic overview of how states work in React:

- **Initialization:** You typically initialize state in the constructor of a class component or by using the useState hook in a functional component.

```
// Class component
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            count: 0
        };
    }
}
// Functional component with useState hook
import React, { useState } from 'react';

function MyComponent() {
    const [count, setCount] = useState(0);
}
```

State in React

2. **Updating State:** You should never directly modify the state object. Instead, React provides methods like setState for class components and the updater function returned by useState for functional components to update the state.

```
// Class component
this.setState({ count: this.state.count + 1 });

// Functional component with useState hook
setCount(count + 1);
```

3. **Rendering:** Whenever state changes, React re-renders the component, ensuring that the UI reflects the updated state.

```
// Render method in class component
render() {
    return <div>{this.state.count}</div>;
}
// Functional component
return <div>{count}</div>;
```

State in React

4. Stateful vs Stateless Components: Components can either have state (stateful) or not have state (stateless). Stateful components manage their **own state**, while stateless components **receive data via props** and don't manage any internal state.

```
// Stateful component
class Counter extends React.Component {
    constructor(props) {
        super(props);
        this.state = { count: 0 };
    }

    render() {
        return <div>{this.state.count}</div>;
    }
}

// Stateless component
function Greeting(props) {
    return <h1>Hello, {props.name}!</h1>;
}
```

ReactJS

State in React

Example - 1: Using States & Setstate method changing message content.

//Class Component which changes the State of variable.

```
class WelcomeStateComponent extends React.Component{
    constructor(){
        super()
        this.state={    message:'Hi, Welcome Visitor....'   }}
    changemessage(){
        this.setState({ message:'Thanks for Subscribing my channel....'    })
    }
render(){
    return(
        <div>
              {/* <h3>Welcome Visitor!!!!</h3>   */}
              <h3>{this.state.message}</h3>
              <button onClick={()=> this.changemessage()} >Subscribe</button>
        </div>
    );
}
}
export default WelcomeStateComponent;
```

State in React

Example - 2: Using States & Setstate method incrementing a count value.

// class component which increments count variable using states.

```
class CountComponent extends React.Component{
    constructor(){
        super()
        this.state ={     count:0        }}
    countincrement(){
        this.setState({  count:this.state.count+1        })
        console.log(this.state.count);
    }
    render(){
        return(
            <div>
                    <h3>Initial Counter Value is: <b className='text-danger'> {this.state.count}
</b> </h3>
                    <button onClick={()=>this.countincrement()}>Increment</button>
            </div>
        );    }
}export default CountComponent;
```

State in React

Example - 3: In this example, we are going to add a button to the render() method. Clicking on this button triggers the toggleDisplay() method which displays the desired output.

// Display content based on Toggle button through states

```
class ToggleButtonComponent extends React.Component{
    constructor(){
        super()
        this.state = { display: false }
        console.log('Component this', this);
        this.toggleDisplay = this.toggleDisplay.bind(this);
      }

      toggleDisplay(){
        this.setState({ toggleDisplay:!this.state.toggleDisplay   });
      }
```

State in React

Example - 3: In this example, we are going to add a button to the render() method. Clicking on this button triggers the toggleDisplay() method which displays the desired output.

```
render(){
        return(
            <div>
                  <h1 className='text-primary'>Example to Display Toggle Button content using
states....</h1>
                        {
                               this.state.toggleDisplay?( <div>
                                        <h3>The content is dispalyed based on toggle button...</h3>
                                        <h3 className='text-danger'>Hello World!!!!</h3>
                                        <button onClick={()=>this.toggleDisplay()}>Hide</button>
                                   </div>
                               ):(  <div>
                                        <button onClick={()=>{this.toggleDisplay()}}>Show
more</button>
                                    </div>
 )       }   </div>
        )    } } export default ToggleButtonComponent;
```

State in React

Difference between State and Props

SN	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.
5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
6.	Stateless component can have Props.	Stateless components cannot have State.
7.	Props make components reusable.	State cannot make components reusable.
8.	Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

Hooks in React

- Until React 16.8, function components are simply stateless component. To add state to a component, we need to convert the function component into class based component.

- Also, function component does not have option to manipulate lifecycle events of a component. To enable state and lifecycle events in the function component, React introduced a new concept called Hooks.

- Hooks are plain JavaScript functions having **access to state and lifecycle events of the component** in which it is used / applied. In general, hooks starts with **use keyword**. React comes with a few built-in hooks and allows the creation of custom hooks as well.

Hooks in React

Benefits of using React Hooks

Benefits of using React Hooks in your development process include:

- **Reusability and modularity:** By enabling you to develop personalized Hooks that contain functionality that is readily shared across numerous components, Hooks enhance reusability. This aids in efficiently organizing and maintaining your codebase.

- **Elimination of class components:** When dealing with complicated state management, hooks take the place of class components, which can be verbose and challenging to reason about. Your code will be simpler to read and comprehend since hooks offer a cleaner, more logical way to handle state and side effects.

- **Performance improvement:** Hooks are designed for re-renders, allowing for selective updates to only the necessary components. This results in notable performance gains, especially for more complex applications. Updates are quicker and more effective thanks to hooks' improved rendering control.

Hooks in React

Understanding the different types of hooks in react

React Hooks come in various flavors, each of which has a distinct function. The most common types of hooks in React are listed below:

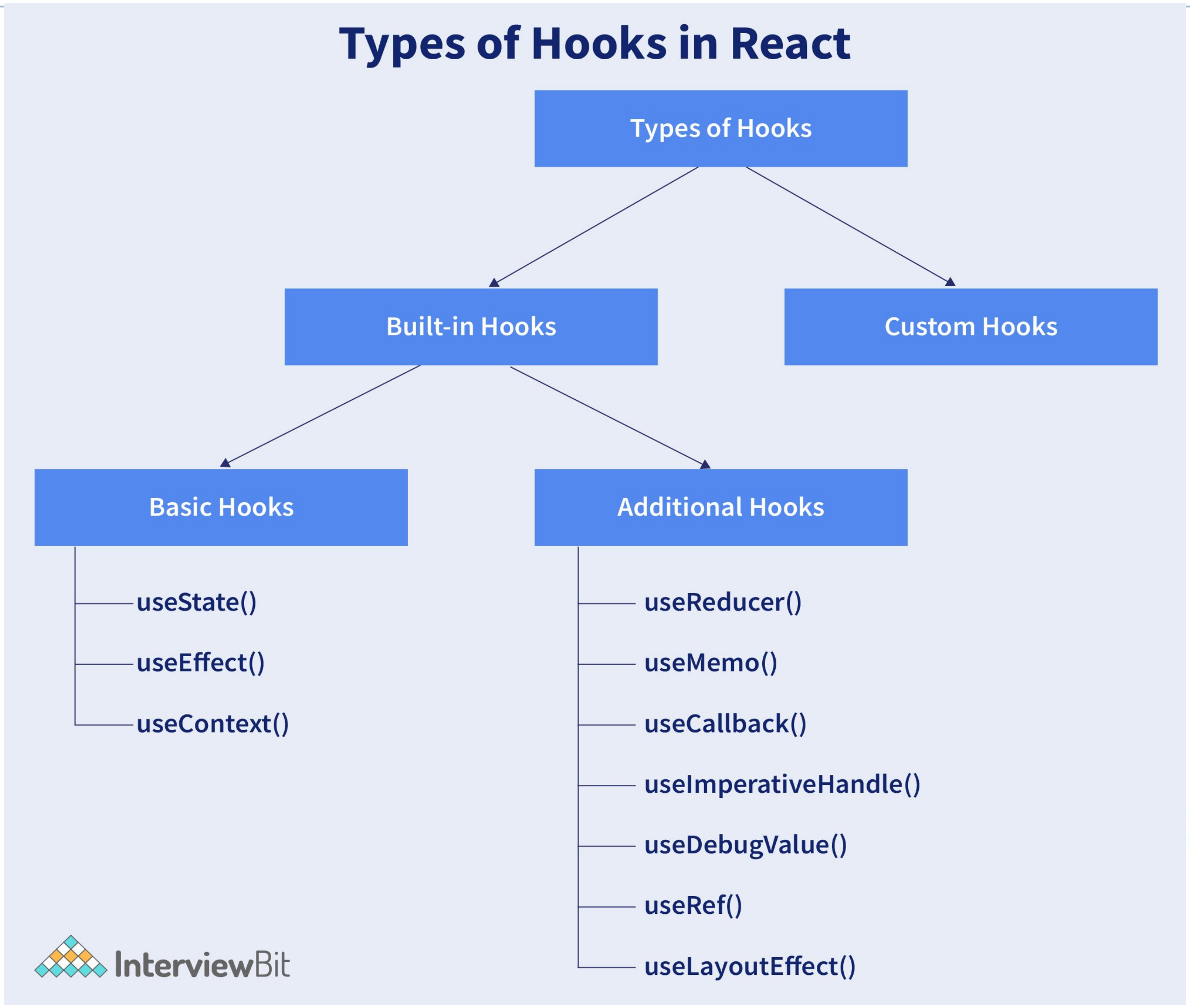
- **State Hooks:** State can be added to your functional components using state hooks like **useState.** You can **declare a state variable and change its value inside** of a functional component using the **useState Hook**. State management is made simpler and class components are no longer required.

- **Effect Hooks:** You can execute side effects in your functional components using effect hooks like **useEffect.** The DOM can be **altered, data can be fetched, and events may be subscribed to**. The useEffect Hook makes sure that your side effects are executed without sacrificing performance after each render.

- **Context Hooks:** You can access the React context within your functional components by using context hooks like **useContext.** Without having to **explicitly pass props at each level, context allows components to communicate data between them.** You may quickly consume context values within your components with the useContext Hook.

- **Custom Hooks:** User-defined Hooks that contain reusable logic are known as custom hooks. By merging already-existing Hooks or by adding unique functionality, bespoke Hooks can be made. As a result, your code is more modular & maintainable since you can abstract away complicated functionality into a reusable Hook.

Hooks in React



Hooks in React

Advantages of Hooks

Function component have many advantages over class based component when used along with Hooks. They are as follows −

1. Hooks are easy to understand and quick to start coding.

2. The complexity of the application can be kept minimum in a large application. In class based component, the complexity (state management and handling lifecycle events) grows as the project grows.

3. this in class (component) is hard to understand for a beginner in JavaScript programming. Since Function component and hooks don't depends on this, developers can quickly start coding in React without steep learning curve.

4. Stateful logic can be reused easily between components.

5. Function component can be used along with class based component, which makes it easily adoptable in the existing project of any size.

6. Function component can be written is few lines of code compared to class based components.

Hooks in React

Disadvantages of Hooks

Hooks are alternative methodology to create components and it has its own disadvantages as well. They are as follows −

1. Hooks should be called only at top level and should be avoided inside a condition, loop or nested function.

2. Hooks are specialized features that they may not be best fit for certain situation and we may have to restore to class based component.

3. React handles the internals of hooks without exposing the core for optimization, which makes it less flexible and not suitable for certain scenarios.

Reactjs – Using useState

- useState is a basic React hook, which allows a function component to maintain its own state and re-render itself based on the state changes.

The signature of the useState is as follows −

```
const [ <state>, <setState> ] = useState( <initialValue> )
```

where,

- **initialValue −** Initial value of the state. state can be specified in any type (number, string, array and object).
- **state −** Variable to represent the value of the state.
- **setState −** Function variable to represent the function to update the state returned by the useState.

The signature of the setState function is as follows −

```
setState( <valueToBeUpdated> )
```

Reactjs – Using useState

Examaple-1: Rendering the count value in function component without useState.

UseStateExample.js

```
const UseStateExample=()=>{
    let count=0;
    function updatecount(){
        ++count;
        console.log(count);
    }
    return(
        <div>
            <h3>Rendering Count value on updation </h3>
            <h2>Counter Value:<b className="text-danger">{count}</b></h2>
            <button onClick={updatecount}>increment</button>
        </div>
    );
}
export default UseStateExample;
```

Reactjs – Using useState

Examaple-1: Rendering the count value in function component with useState.

UseStateExample.js

```
const UseStateExample=()=>{

let [count, setcount] = useState(0);
    const updatecount=()=>{
        setcount(++count);
        console.log(count);
    }
return(
        <div>
            <h3>Rendering Count value on updation </h3>
            <h2>Counter Value:<b className="text-danger">{count}</b></h2>
            <button onClick={updatecount}>increment</button>
        </div>
    );
}
export default UseStateExample;
```

Reactjs – Using useState

Example-2:

Here's an example of how to use setState() to change the state:

```
import React, { useState } from 'react';

function UseStateExample() {

  const [count, setCount] = useState(0);

  const increment = () => {
   setCount(count + 1);
  };

  const decrement = () => {
   setCount(count - 1);
  };

  const reset = () => {
   setCount(0);
  };
```

Reactjs – Using useState

Here's an example of how to use setState() to change the state:

```
return (
   <div>
    <h2>Counter</h2>
    <p>Count: {count}</p>
    <button onClick={increment}>Increment</button>
    <button onClick={decrement}>Decrement</button>
    <button onClick={reset}>Reset</button>
   </div>
  );
 }
 export default UseStateExample;
```

App.js

```
<UseStateExample/>
```

End Of UNIT-1