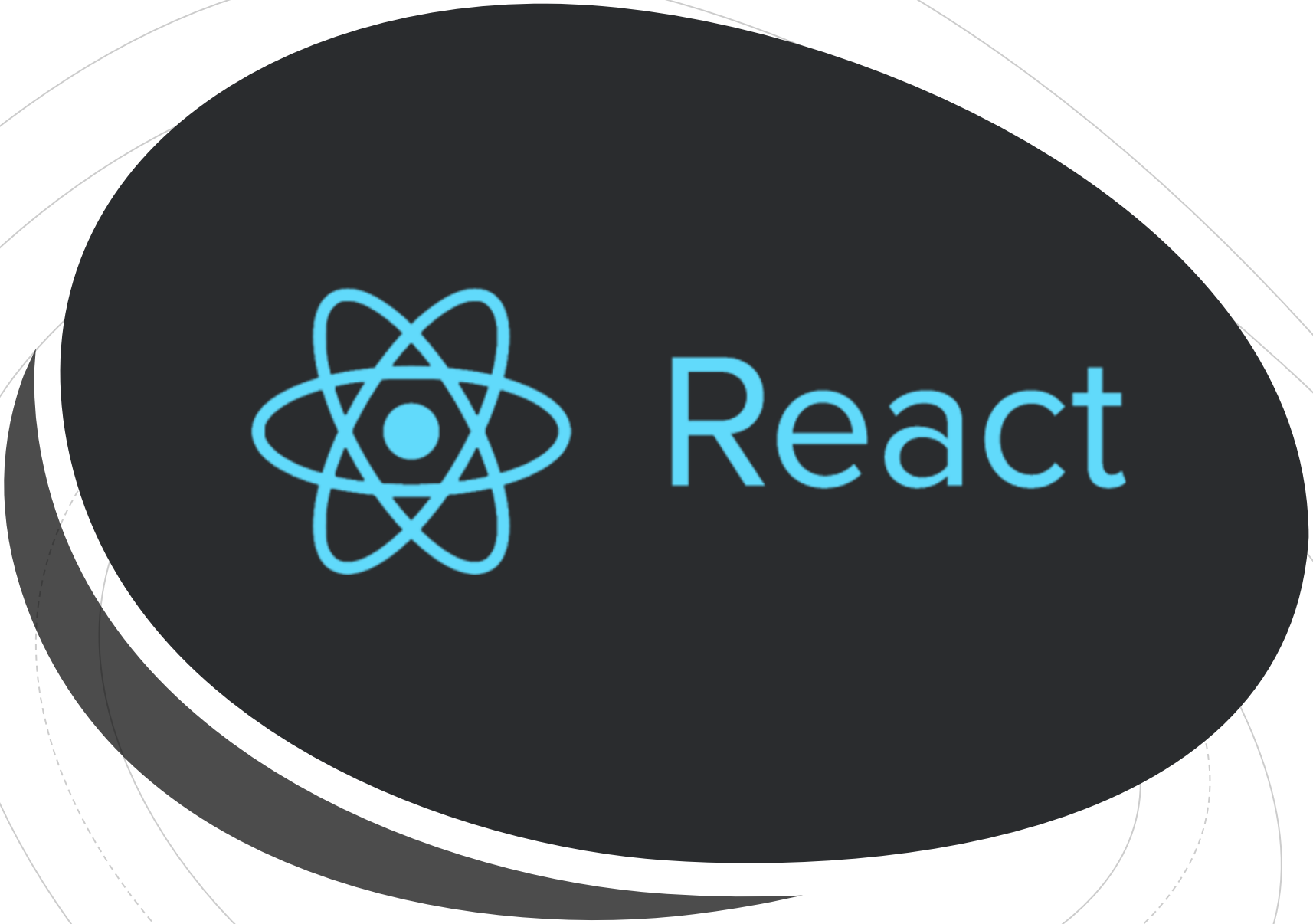
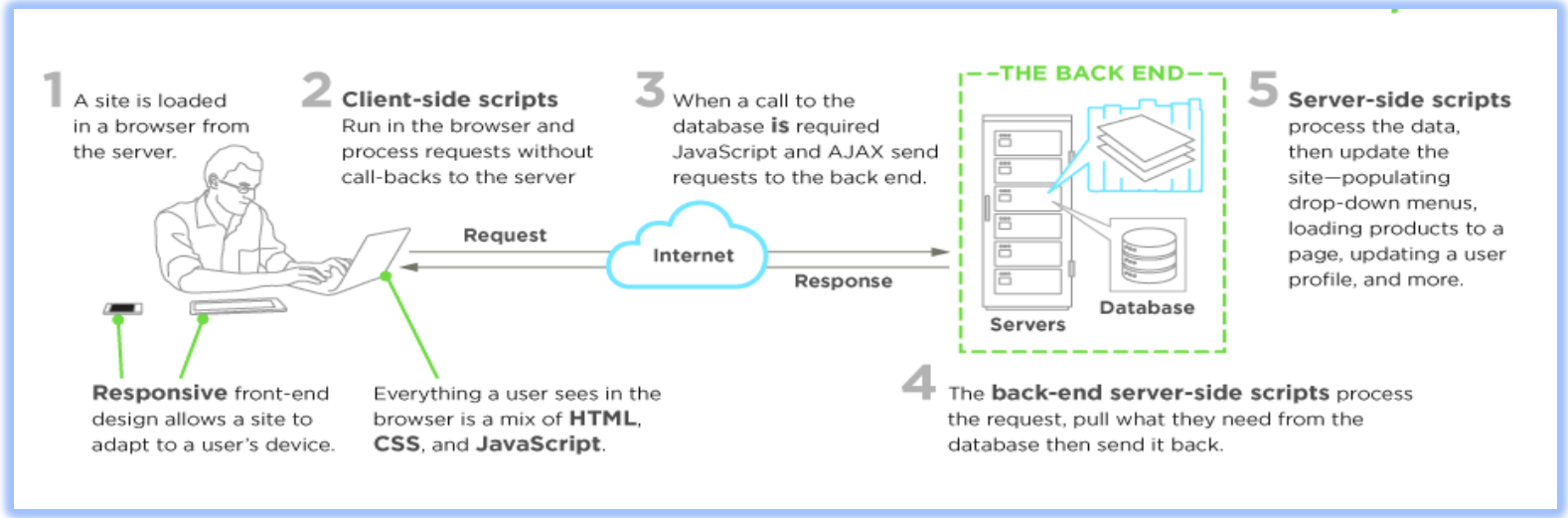


COMP-3123

Full Stack Development - I

What is Front-End Tool ?

- Front-end web development, also known as **client-side development** is the practice of producing **HTML, CSS and JavaScript** for a website or Web Application so that a user can see and interact with them directly.
- The challenge associated with front end development
  - *the tools and techniques used to create the front end of a website change constantly and so the developer needs to constantly be aware of how the field is developing.*
- The objective of designing a site is to ensure that when the users open up the site they see the information in a format that is **easy to read and relevant**.
- It get's complicated by the fact that users now use a **large variety of devices** with **varying screen sizes** and **resolutions** thus forcing the designer to take into consideration these aspects when designing the site.
- Developers need to ensure that the site comes up correctly in **different browsers** (cross-browser), **different operating systems** (cross-platform) and **different devices** (cross-device), which requires **careful planning** on the side of the developer.



Web Reference Link

What is React JS

https://reactjs.org/

- 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 which is faster than the regular DOM.
- ReactJS is **JavaScript library** used for building reusable UI components.
- ReactJS is one of the most **popular** JavaScript front-end libraries which has a strong foundation and a **large community**.
- According to React official documentation, following is the definition –
  - React is a library for building **composable user interfaces**.
  - It encourages to creation of **reusable UI components**, which present data that changes over time.
  - React can also render on the server using Node, and it can power native apps using **React Native**.
  - React implements **one-way reactive data flow**, which **reduces the boilerplate** and is easier to reason about than traditional data binding.

React.JS History

- Current version of React.JS is **V17.0.1** (October 2020).
- **Initial Release** to the Public (V0.3.0) was in July 2013.
- React.JS was first used in 2011 **for Facebook's Newsfeed feature** and was later used in its products like **WhatsApp & Instagram**.
- Facebook Software Engineer, **Jordan Walke**, created it.
- The **create-react-app** version 2.0 package was released in October 2018.
- Create-react-app version 2.0 supports **Babel** 7, **webpack** 4, and **Jest23**.
- 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**.

React JS Features

https://reactjs.org/

- **JSX** – JSX is JavaScript syntax extension.

- **Components** –
  - React is all about components.
  - **Components are independent and reusable bits of code.**
  - They serve the same purpose as JavaScript functions, but work in isolation and returns HTML via a render function.
  - You need to think of everything as a component and this will help you maintain the code when working on larger scale projects.

- **Unidirectional data flow and Flux** – React implements one-way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data unidirectional.

- **License** – React is licensed under the **Facebook Inc.** Documentation is licensed under CC BY 4.0.

React Advantages

- Uses virtual DOM which is a JavaScript object. This will improve apps performance, since JavaScript **virtual DOM** is faster than the regular DOM.
- Can be used on **client and server side** as well as with other frameworks.
- **Component and data patterns** improve readability, which helps to maintain larger apps.

React Limitations

- **Covers only the view layer of the app**, hence you still need to choose other technologies to get a complete tooling set for development.
- Uses inline templating and JSX, which might **seem awkward** to some developers.

Web Reference Link

Installation and Environmental Setup

- NodeJS is the platform needed for the ReactJS development. (https://nodejs.org/en/download/)
- There are two ways to set up an environment for successful ReactJS application.
    1. Using the npm command
    2. Using the **create-react-app** command

Using the **create-react-app** command

- Install create-react-app

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

- Browse through the desktop and install the Create React App using command prompt as shown below –

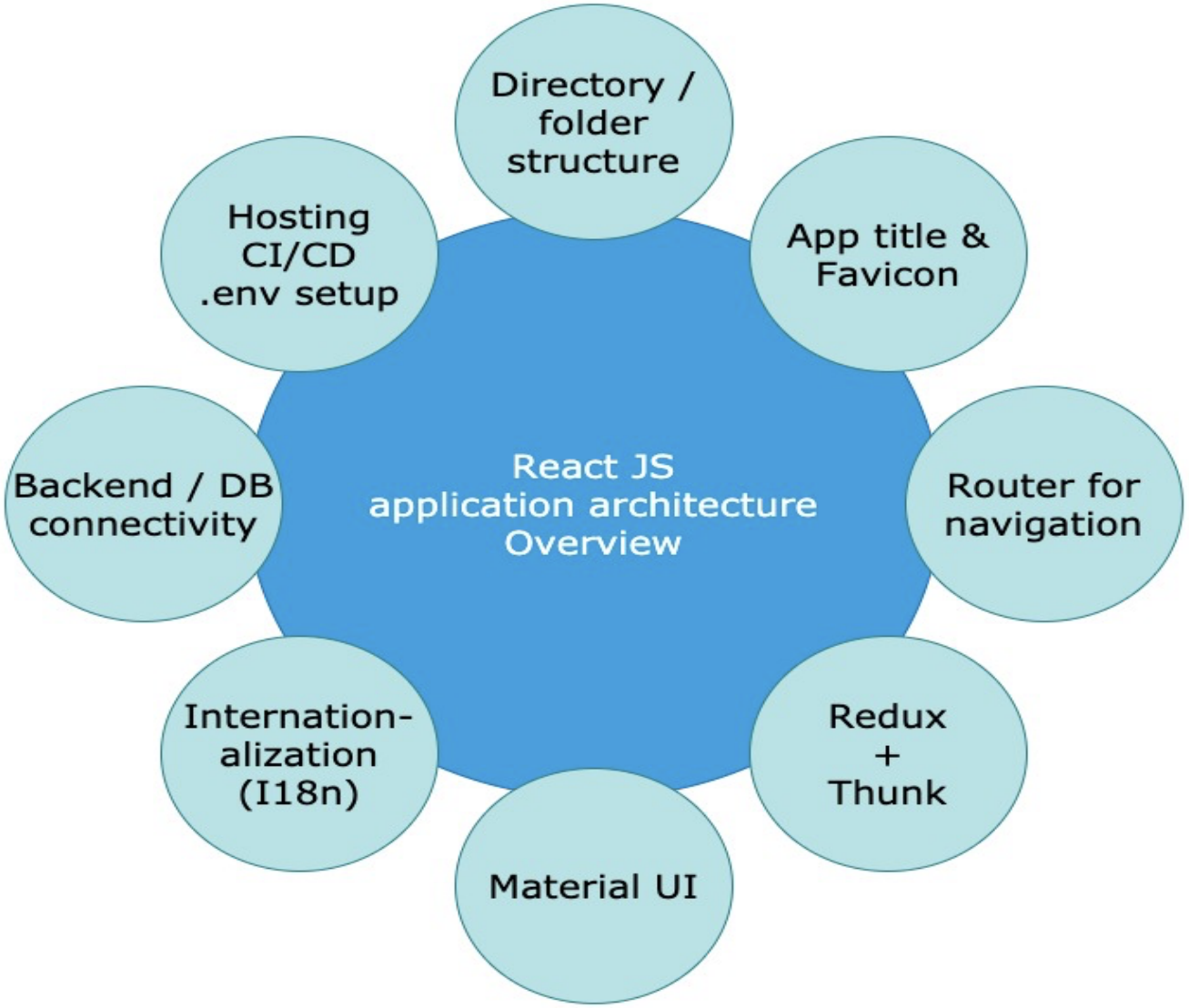
```
npx create-react-app my-app
```

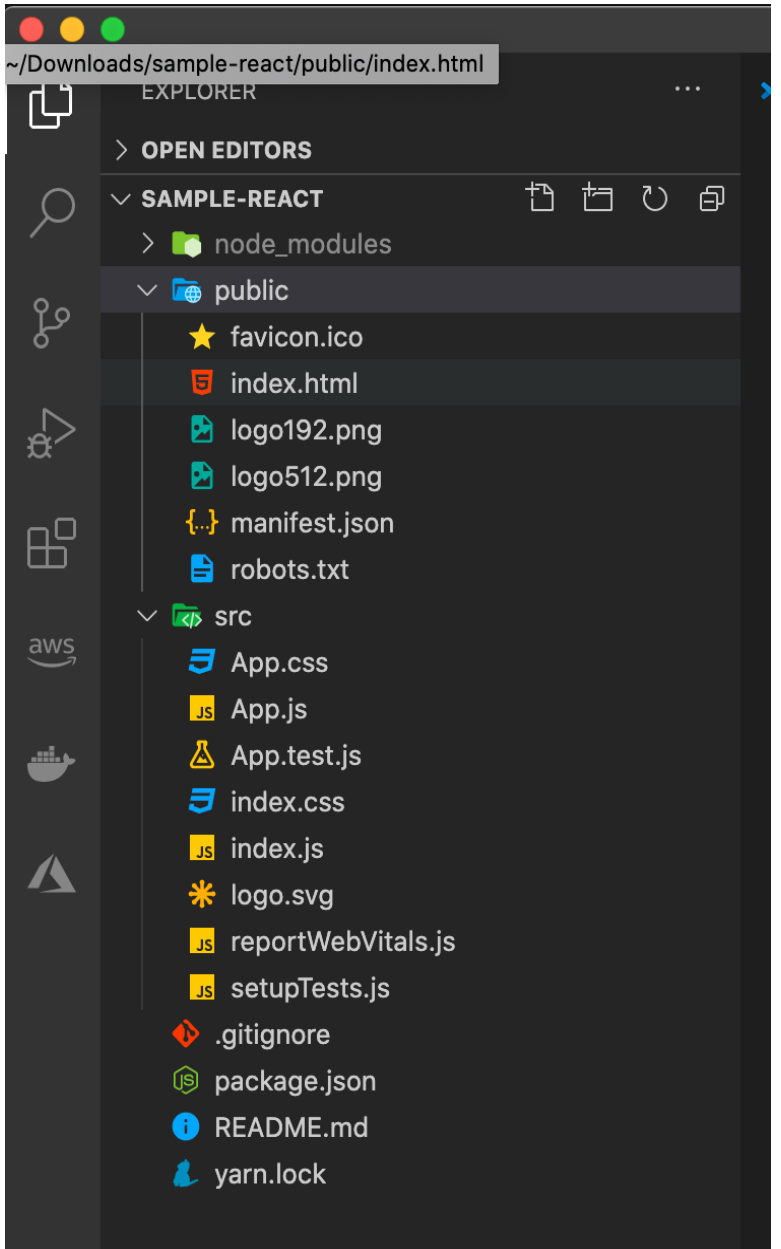
- Finally, run the project using the start command.

```
npm start
```

Web Reference Link

Architecture of React JS application





[Web Reference Link]

Required Modules in React JS application

```
{
 "name": "reactApp",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "scripts": {
  "start": "webpack-dev-server --mode development --open --hot",
  "build": "webpack --mode production"
 },
 "keywords": [],
 "author": "",
 "license": "ISC",
 "dependencies": {
  "react": "^16.8.6",
  "react-dom": "^16.8.6",
  "webpack-cli": "^3.3.1",
  "webpack-dev-server": "^3.3.1"
 },
 "devDependencies": {
  "@babel/core": "^7.4.3",
  "@babel/preset-env": "^7.4.3",
  "@babel/preset-react": "^7.0.0",
  "babel-core": "^6.26.3",
  "babel-loader": "^8.0.5",
  "babel-preset-env": "^1.7.0",
  "babel-preset-react": "^6.24.1",
  "html-webpack-plugin": "^3.2.0",
  "webpack": "^4.30.0"
 }
}
```

Webpack is used for module packaging, development, and production pipeline automation. We will use **webpack-dev-server** during development, **webpack** to create production builds, and **webpack CLI** provides a set of commands. Webpack compiles these into a single file(bundle).

Babel is a JavaScript compiler and transpiler used to convert one source code to others. It compiles React JSX and ES6 to ES5 JavaScript which can be run on all browsers. We need **babel-loader** for JSX file types, **babel-preset-react** makes your browser update automatically when any changes occur to your code without losing the current state of the app. ES6 support requires **babel-preset-env** Babel preset.

JSX & HTML View

- React uses JSX for **templating** instead of regular JavaScript.
- Following are some **pros** that come with it.
  - It is faster because it performs optimization while compiling code to JavaScript.
  - It is also type-safe and most of the errors can be caught during compilation.
  - It makes it easier and faster to write templates, if you are familiar with HTML.

Using JSX

- JSX looks like a regular HTML in most cases. Look at the code from **App.jsx** where we are returning **div**.

If we want to return more elements, we need to wrap it with one container element.

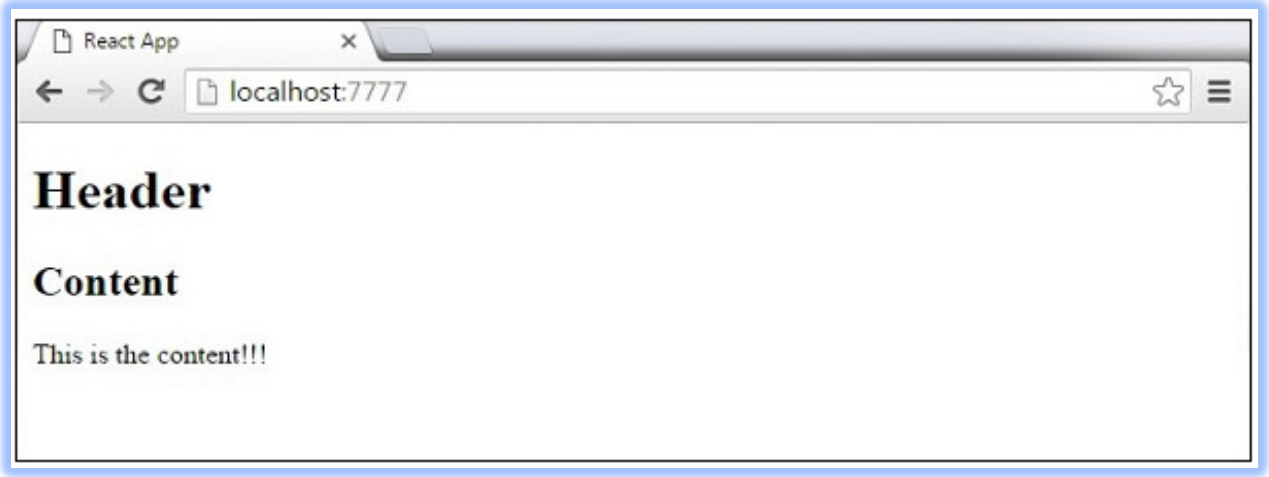
Example JSX

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            Hello World!!!
         </div>
      );
   }
}
export default App;
```

```
import React from 'react';

class App extends React.Component {
   render() {
      return (
         <div>
            <h1>Header</h1>
            <h2>Content</h2>
            <p>This is the content!!!</p>
         </div>
      );
   }
}
export default App;
```



Single-Page Application using React?

What is a single page application?

- A single page application (SPA) is essentially a webpage that interacts with the web browser **dynamically** by **rewriting the current web page** with the data obtained from the webserver.
- In a single page application, the **webpage does not reload the page** during its runtime and instead works within a browser.

Single-page Application

- **Reloading**: Single-page applications work inside a browser and do not require page reloading during webpage executing.
- **UI/UX**: Offers outstanding user experience by -
  - Imitating a natural environment with the browser by eliminating wait time and page reloads.
  - It consists of a single web page that loads all content using JavaScript.
  - It requests the markup and data independently and renders pages straight to the browser.
- **Examples**: Gmail, Google Maps, Facebook, GitHub.

Web Reference Link

Why choose a single-page application?

The benefits of choosing single-page applications (SPA) are:

- SPA is **quicker** since all the webpage resources are loaded only once throughout the application, and data is the only resource that is transmitted.
- SPA **caches local storage** effectively as it sends one request, stores all the data, and uses it even when offline.
- SPA **simplifies and streamlines development** activities as it eliminates the need to write code to render pages on the server.
- SPA **can be debugged with ease with Chrome** as it is possible to investigate page elements and monitor network operations.

When not to use single-page applications?

While SPA does have its advantages, there are certain cases when it is not suitable to use it:

- **SEO**: It is **difficult** and tricky to optimize SPA for SEO since its content is loaded by **AJAX (Asynchronous JavaScript and XML).** *SPAs are not suitable for cases where SEO is critical for business success.*
- **Javascript**: It requires users to **enable Javascript** for proper application and action loading. So it is not suitable for instances where JavaScript might be disabled on the user side.
- **Security**: SPA is also less secure in comparison to MPA, making it unsuitable for highly sensitive applications. SPA has a **cross-site scripting (XSS)** and allows attackers to inject client-side scripts into the web application.
- **Slow**: While the user experience of SPAs on runtime is fast, it is **slower to download** and can also be slowed down if there are memory leaks in JavaScript.

Using State

- The state object is where you **store property values** that belongs to the component.
- When the **state object changes**, the component **re-renders**.
- The state object is initialized in the **constructor** of the component.
- It can be set by using the **setState()** method and calling setState() method triggers UI updates.

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  changeColor = () => {
    this.setState({color: "blue"});
  }
  render() {
    return (
      <div>
        <h1>My {this.state.brand}</h1>
        <p>
          It is a {this.state.color}
          {this.state.model}
          from {this.state.year}.
        </p>
        <button
          type="button"
          onClick={this.changeColor}
        >Change color</button>
      </div>
    );
  }
}
```

Using State

- We should always try to make our state as **simple as possible** and minimize the number of stateful components.
- **For example**, if we have ten components that need data from the state, we should create one container component that will keep the state for all of them.

```
import React from 'react';

class App extends React.Component {
    constructor(props) {
        super(props);

        this.state = {
            header: "Header from state...",
            content: "Content from state..."
        }
    }
    render() {
        return (
            <div>
                <h1>{this.state.header}</h1>
                <h2>{this.state.content}</h2>
            </div>
        );
    }
}
export default App;
```

State Example

```
import React from 'react';
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      header: "Header from state...",
      content: "Content from state..."
    }
  }
  render() {
    return (
      <div>
        <h1>{this.state.header}</h1>
        <h2>{this.state.content}</h2>
      </div>
    );
  }
}
export default App;
```

Using Props

- Props are **arguments** passed into React components.
- Props are passed to components **via HTML attributes**.
- When we need immutable data in our component, we can just add props to **render()** function and use it inside our component.
- You can also set default property values directly on the component **constructor** instead of adding it to the render() element.

Props Example

```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand}!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    return (
      <div>
      <h1>Who lives in my garage?</h1>
      <Car brand="Ford" />
      </div>
    );
  }
}

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

Default Props Example

```
import React from 'react';
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.headerProp}</h1>
        <h2>{this.props.contentProp}</h2>
      </div>
    );
  }
}
App.defaultProps = {
  headerProp: "Header from props...",
  contentProp:"Content from props..."
}
export default App;
```

State v/s Props

The main difference between state and props are

- **Props**
  - **props** are immutable
  - props use to pass data in the child component
  - props change a value outside a component(child component)
  - ***props*** *are used by a component* ***to get data from external environment*** *i.e. another* ***component*** *(pure, functional or class) or a* ***general class*** *or* ***JavaScript*** *code.*
- **State**
  - **state** use inside a class component
  - state change a value inside a component
  - If you render the page, you call **setState()** to update DOM (update page value)
  - ***states*** *are used* ***to manage the internal environment of a component*** *means the* ***data changes*** *inside the* ***component***

Thank You