Purwadhika
Digital Technology School

Fullstack Web Development

Intro to React

Job Connector Program

Outline

- Javascript DOM
- Intro to React
- Component & props
- Class based component vs functional component
- Routing

What is DOM ?

The Document Object Model (DOM) in Javascript is a way to interact with and manipulate web pages. Think of a web page as a document, like a book or a magazine. The DOM is like a tool that allows you to change, read, and update the contents of that document (web page) using Javascript.

Key points:

- **Document**: The "D" in DOM stands for "Document," which is your web page. It includes everything you see on a webpage - text, images, links, buttons, forms, and more.
- **Object**: The "O" in DOM stands for "Object." In programming, objects are like containers for data and actions. The DOM represents your web page as a structured set of objects that you can manipulate.
- **Model**: The "M" in DOM stands for "Model." It's a way to represent the structure of your web page in a logical and organized manner. This model is hierarchical, like a family tree, where elements (like paragraphs, headings, or images) are related to one another.

Src : https://en.wikipedia.org/wiki/Document_Object_Model
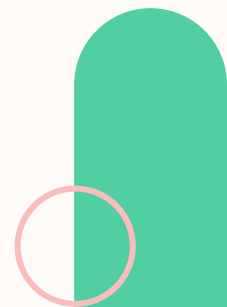
DOM Manipulation with Javascript

DOM manipulation is the process of using Javascript to interact with and change the content, structure, and style of a web page represented by the DOM.

It allows you to make dynamic and interactive web applications by altering what users see and how they interact with your web page.
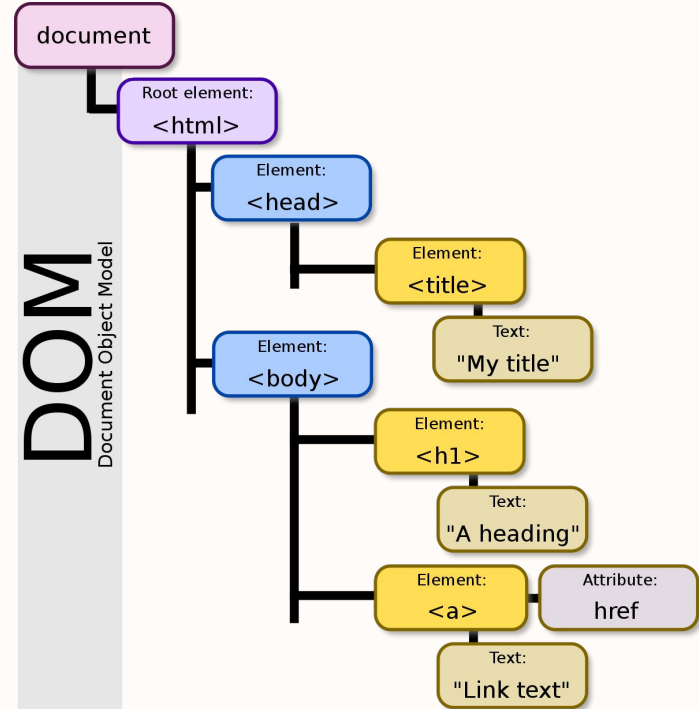
HTML + Javascript DOM

In this session you will learn about developing web application through HTML + Javascript DOM comparing to ReactJs.

In this example, the string **Original Text** changed into **Updated Text** when the Update Text button is clicked.
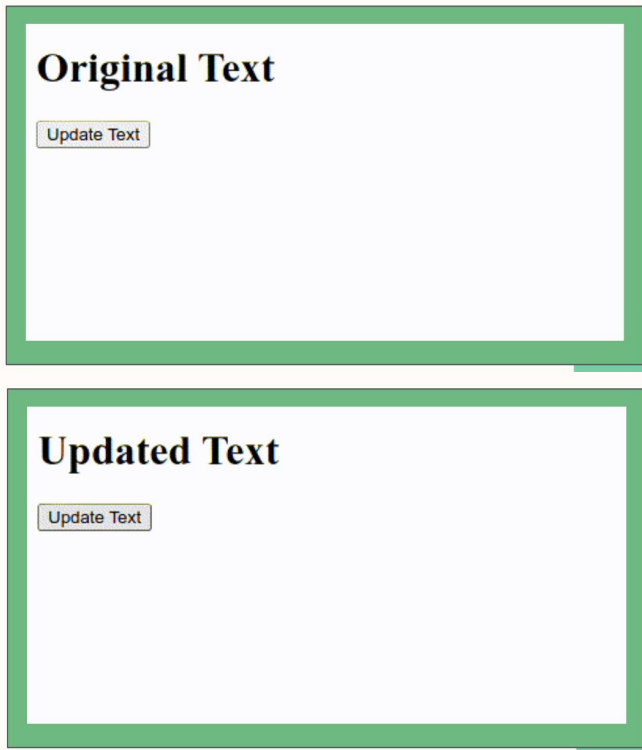
HTML + Javascript DOM

We would like to manipulate the **Original Text** in **h1** tag, change it into **Update Text.**


```html
<!DOCTYPE html>
<html>
  <head>
    <title>HTML+JavaScript DOM</title>
    <script>
      function updateText() {
        const element = document.getElementById("myElement");
        element.innerText = "Updated Text";
      }
    </script>
  </head>
  <body>
    <h1 id="myElement">Original Text</h1>
    <button onclick="updateText()">Update Text</button>
  </body>
</html>
```

index.html

HTML + Javascript DOM

Inside script tag, you could write any of javascript syntax under this html tag.

Since we would like modify **h1** html tag string value from "Original Text" into "Update Text" we need to use Javascript DOM to manipulate the html.

```html
index.html

<!DOCTYPE html>
<html>
  <head>
    <title>HTML+JavaScript DOM</title>
    <script>
      function updateText() {
        const element = document.getElementById("myElement");
        element.innerText = "Updated Text";
      }
    </script>
  </head>
  <body>
    <h1 id="myElement">Original Text</h1>
    <button onclick="updateText()">Update Text</button>
  </body>
</html>
```
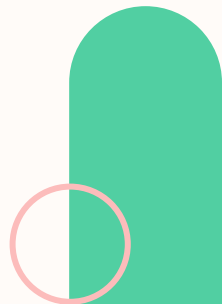
HTML + Javascript DOM

Here are some of the basic methods available in the DOM:

- getElementById(id): Retrieves an element by its unique id attribute.
- getElementsByClassName(className): Retrieves a collection of elements that have a specific class name.
- getElementsByTagName(tagName): Retrieves a collection of elements that have a specific tag name.
- querySelector(selector): Retrieves the first element that matches a specific CSS selector.
- querySelectorAll(selector): Retrieves a collection of elements that match a specific CSS selector.
- createElement(tagName): Creates a new element with the specified tag name.
- createTextNode(text): Creates a new text node with the given text content.
- appendChild(node): Appends a node as the last child of another node.

HTML + Javascript DOM

To modify the DOM (Document Object Model) in JavaScript, you can use various methods and properties provided by the DOM API. Here are some common ways to modify the DOM:

Changing Element Content:

- textContent or innerText: To change the text content of an element.
- innerHTML: To change the HTML content of an element.

```
// Example: Changing text content
const myElement1 = document.getElementById("myElement1");
myElement.textContent = "New text content";

// Example: Changing HTML content
const myElement2 = document.getElementById("myElement2");
myElement.innerHTML = "<p>This is a new paragraph.</p>";
```

HTML + Javascript DOM

Creating and Adding Elements:

- createElement: To create a new element.
- appendChild or insertBefore: To add the newly created element to the DOM.

```javascript
// Example: Creating and adding a new element
const newParagraph = document.createElement("p");
newParagraph.textContent = "This is a new paragraph.";

const parentElement = document.getElementById("parentElement");
parentElement.appendChild(newParagraph);
// OR
const siblingElement = document.getElementById("siblingElement");
parentElement.insertBefore(newParagraph, siblingElement);
```

HTML + Javascript DOM

Removing Elements:

- removeChild: To remove a child element from its parent.

```javascript
// Example: Removing an element
const elementToRemove = document.getElementById("elementToRemove");
const parentElement = elementToRemove.parentNode;
parentElement.removeChild(elementToRemove);
```

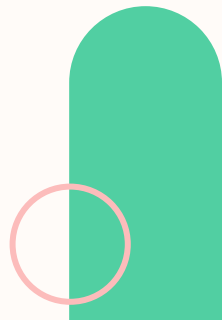# HTML + Javascript DOM

Updating Element Attributes and Styles:

- You can use the setAttribute method to set or modify attributes of an element.
- You can directly modify the style property to change the inline styles of an element.

```javascript
// Example: Updating attributes and styles
const myElement = document.getElementById("myElement");

// Update attributes
myElement.setAttribute("data-custom-attr", "value");

// Update styles
myElement.style.color = "blue";
myElement.style.fontSize = "18px";
```

HTML + Javascript DOM

Adding Event Listeners:

- Use addEventListener to attach event handlers to elements.

```javascript
// Example: Adding an event listener
const myButton = document.getElementById("myButton");
myButton.addEventListener("click", function() {
  alert("Button clicked!");
});
```
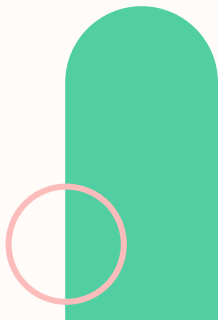
Introduction to React

Since you already know how to manipulate your HTML through DOM manipulation using javascript. Now, let me introduce you another way to manipulate your HTML through ReactJS.

What is **React**?

React is an **open-source JavaScript library** developed by Facebook. It is primarily used for building user interfaces (UI) and handling the view layer of web applications. React allows developers to create reusable UI components and efficiently manage the state of a web application, making it easier to build interactive, dynamic, and scalable front-end applications.
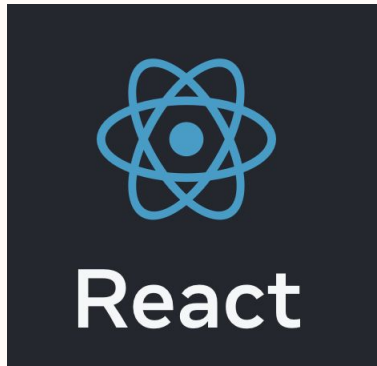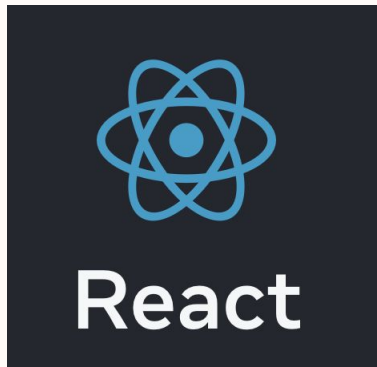
Find out more here:
https://react.dev/

Introduction to React

Why React?

React's popularity is due to its flexibility, performance, and developer-friendly approach to building modern web applications. It is widely used in web development for creating single-page applications, progressive web apps, and complex user interfaces across various industries and platforms.
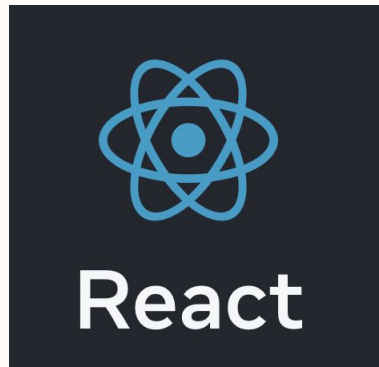

Find out more here:
https://react.dev/
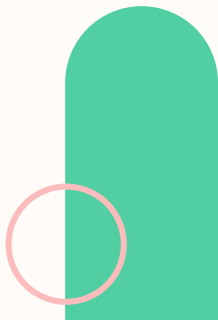
Introduction to React

- Component-Based Architecture
- Declarative Programming
- Efficient Virtual DOM
- Reusability and Composability
- Rich Ecosystem
- Large and Active Community
- Backed by Facebook
- Platform Agnostic: React can be used for web development as well as mobile app development (React Native). This allows developers to reuse a significant portion of their codebase while building applications for multiple platforms.
- Open-Source and Free

Find out more here:
https://react.dev/
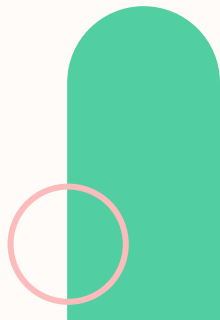
Single Page Application (SPA)

Single Page Application is a method to update the appearance without having to refresh the entire appearance of the website, so it only uses an HTML file as a media render.

This method is considered better because of its rendering speed. Because only certain elements/components are updated without having to reload the entire page by utilizing Virtual DOM. And this system is implemented in modern javascript environments, especially React JS.
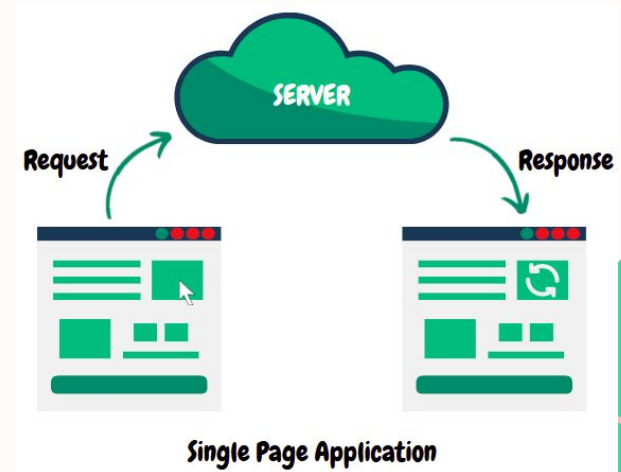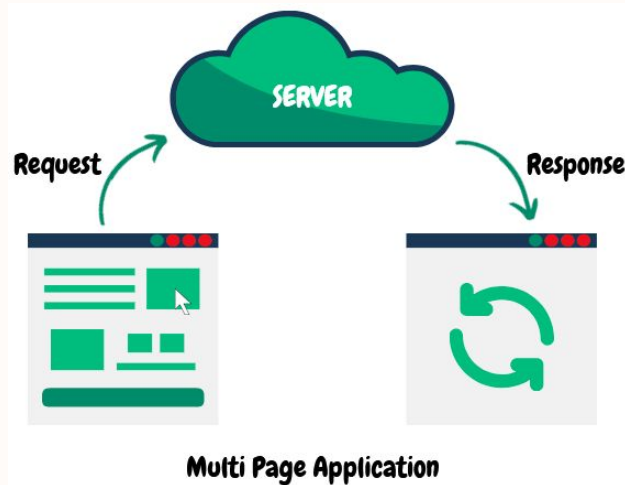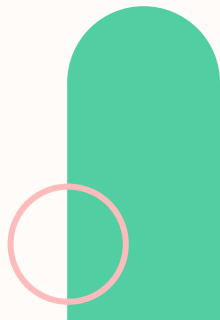
Virtual DOM

The Virtual DOM in React is a lightweight, in-memory representation of the actual DOM (Document Object Model) of a web page. It acts as an intermediary layer between your React components and the Real DOM. React uses the Virtual DOM to optimize updates and improve performance by reducing direct manipulation of the Real DOM.

The Virtual DOM allows React to efficiently track and update only the elements that have changed, minimizing the work required to keep the web page up to date. This results in faster rendering and a smoother user experience.
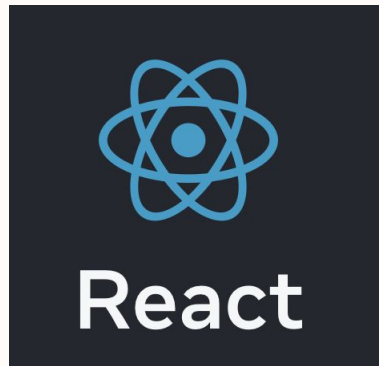
Find out more here:
https://react.dev/

Virtual DOM vs Real DOM

Virtual DOM:

- A virtual representation of the DOM.
- React updates and compares this virtual version, not the actual DOM.
- Efficiently identifies changes and updates the Real DOM.

Real DOM:

- The actual web page structure.
- Directly updating it is less efficient and can be slow.
- React minimizes direct manipulation to enhance performance.

Virtual DOM   Real DOM
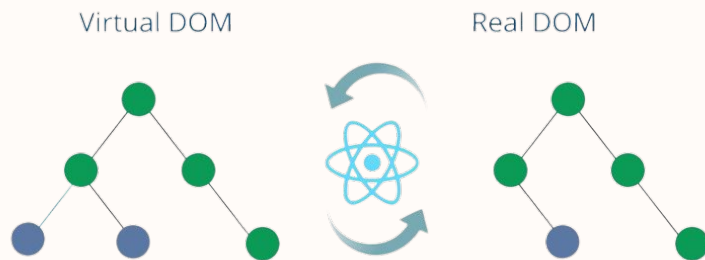
Create a New React Project

React is considered a library, not a framework, as it primarily focuses on building user interfaces and leaves other architectural decisions, such as state management and routing, to be addressed by developers using additional libraries or solutions.

To create a project using React, you can use frameworks like **NextJS**, **Gatsby**, **Vite** etc. In this session, we will user **Vite** to create our first project.

References :

- https://vitejs.dev/guide/

Create a new Vite project :

- ***npm create vite@latest***

```
PS I:\PURWADHIKA> npm create vite@latest
Need to install the following packages:
create-vite@5.1.0
Ok to proceed? (y) y
√ Project name: ... vite-project
√ Select a framework: » React
√ Select a variant: » TypeScript

Scaffolding project in I:\PURWADHIKA\vite-project...

Done. Now run:

  cd vite-project
  npm install
  npm run dev
```
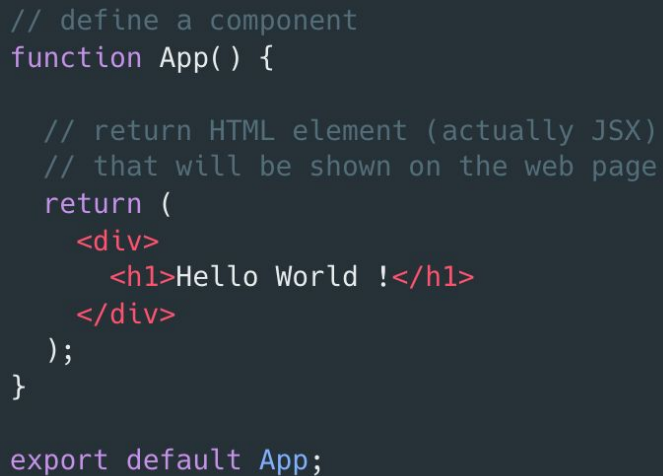
- ***cd vite-project***
- ***npm install***
- ***npm run dev***

Code Structure

```
// define a component
function App() {

  // return HTML element (actually JSX)
  // that will be shown on the web page
  return (
    <div>
      <h1>Hello World !</h1>
    </div>
  );
}


export default App;
```

In general, this is the code structure of the React JS project.

About JSX (or TSX)

JSX is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file. Although there are other ways to write components, most React developers prefer the conciseness of JSX, and most codebases use it

When you user typescript, you will use TSX instead.

Src : https://react.dev/learn/writing-markup-with-JSX

Characteristics

- JSX is not mandatory to use there are other ways to achieve the same thing but using JSX makes it easier to develop react application.
- JSX allows writing expression in { }. The expression can be any JS expression or React variable.
- To insert a large block of HTML we have to write it in a parenthesis i.e, ().
- JSX produces react elements.
- JSX follows XML rule.
- After compilation, JSX expressions become regular JavaScript function calls.
- JSX uses camelcase notation for naming HTML attributes. For example, tabindex in HTML is used as tabIndex in JSX.
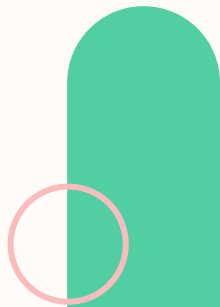
Components

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.

Components come in two types, **Class components** and **Functional components**.
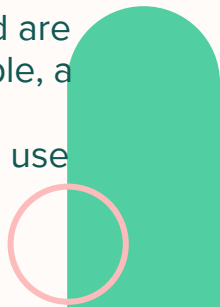
Class Based Component VS Function Component

Class Component

- Class components make use of ES6 class and extend the Component class in React.
- Sometimes called "smart" or "stateful" components as they tend to implement logic and state.
- React lifecycle methods can be used inside class components (for example, componentDidMount).
- You pass props down to class components and access them with this.props

Functional Component

- Sometimes referred to as "dumb" or "stateless" components as they simply accept data and display them in some form; that is they are mainly responsible for rendering UI.
- React lifecycle methods (for example, componentDidMount) cannot be used in functional components.
- There is no render method used in functional components.
- These are mainly responsible for UI and are typically presentational only (For example, a Button component).
- Functional components can accept and use props.

Class Based Component VS Function Component

```
class App extends React.Component {
  render() {
    return <h1>Hello world !</h1>;
  }
}
```

```
function App() {
  return <h1>Hello world!</h1>;
}
```

Props

React allows us to pass information to a Component using something called props (stands for properties).

Props are objects which can be used inside a component.

Src : https://react.dev/learn/passing-props-to-a-component

Props

if you want passing props to another component , add some interface to children component. Its to make sure that the only props item that can be passing to children component, has been written it in interface

For example, only props name and email that can be passing to Profile component. If somebody add props to Profile, it triggered warning that props is not written in interface Iprofile

```
function App() {
  return (
    <div>
      <h1>Hello world!</h1>
      <Profile name="David" email="david@mail.com" />
    </div>
  );
}
```

```
interface IProfile {
  name: string;
  email: string;
}

export default function Profile(props: IProfile) {
  return (
    <div>
      <h2>{props.name}</h2>
      <div>email : {props.email}</div>
    </div>
  );
}
```

Add Styling

There is several ways to styling component in React TS:

1. CSS Classes (Using CSS stylesheet outside component)
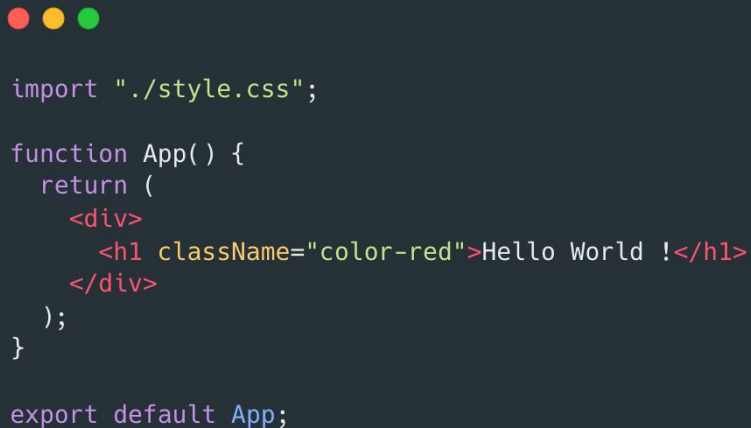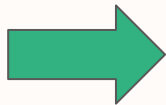2. Inline Styles (Direct styling in an element)
3. Styled components

CSS Classes

Create a css stylesheet as usual, import that CSS file to the component we will use, and then give the class name to the element to be styled into the "className" props. Take a look at the following example.

```css
.color-red {
  color: red;
}
```

```jsx
import "./style.css";

function App() {
  return (
    <div>
      <h1 className="color-red">Hello World !</h1>
    </div>
  );
}

export default App;
```
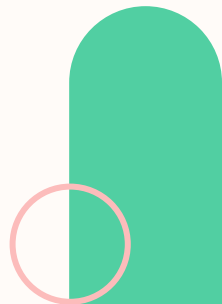
Inline Styles

Styling directly inside the element with props "style={}" and then put object in which there is css command. Take a look at the following example.

```
function App() {
  return (
    <div>
      <h1 style={{ color: "green" }}>Hello World !</h1>
    </div>
  );
}

export default App;
```

Styled Components

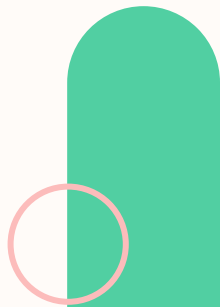```
import styled from "styled-components";

// Styled component named StyledButton
const StyledButton = styled.button`
  background-color: black;
  font-size: 32px;
  color: white;
`;

function Component() {
  // Use it like any other component.
  return <StyledButton> Login </StyledButton>;
}
```

Install:

***npm i styled-components***

Src : https://styled-components.com/docs

# Routing

Routing is a process in which a user is directed to different pages based on their action or request. **React Router** is used to define multiple routes in the application.

First install React Router dependencies on your React JS project: ***npm install react-router-dom@6***

Src : https://reactrouter.com/en/main/start/tutorial

# React Router

Once your project is set up and React Router is installed as a dependency, open the src/index.ts in your text editor. Import BrowserRouter from react-router-dom near the top of your file and wrap your app in a <BrowserRouter>:

```ts
import * as React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import "./index.css";
import App from "./App";

const root = ReactDOM.createRoot(
  document.getElementById("root")
);

root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

React Router

Now you can use React Router anywhere in your app! For a simple example, open src/App.ts and replace the default markup with some routes:

```
import * as React from "react";
import { Routes, Route } from "react-router-dom";
import "./App.css";

function App() {
  return (
    <div className="App">
      <h1>Welcome to React Router 🕴️</h1>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="about" element={<About />} />
      </Routes>
    </div>
  );
}
```

React Router

Now, still in src/App.js, create your route components:

```
function Home() {
  return (
    <>
      <main>
        <h2>Welcome to the homepage 🧍 </h2>
        <p>You can do this, I believe in you 🔥 </p>
      </main>
      <nav>
        <Link to="/about" >About</Link>
      </nav>
    </>
  );
}
```
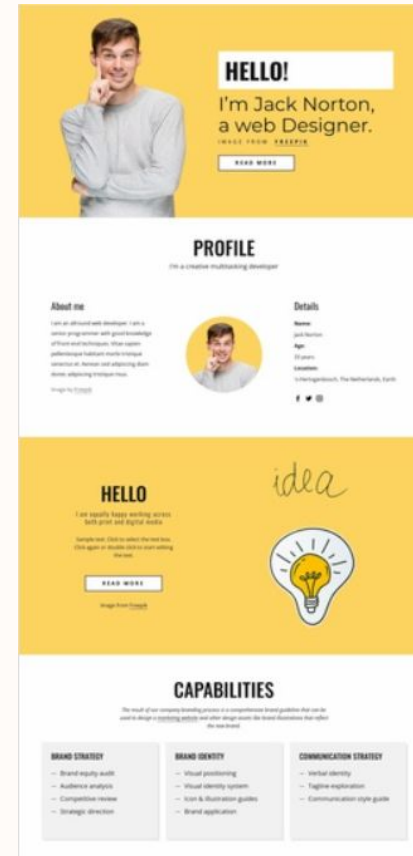
```
function About() {
  return (
    <>
      <main>
        <h2>Who are we ? 🤨</h2>
        <p>
          That feels like an existential question, don't you think ? 🤔
        </p>
      </main>
      <nav>
        <Link to="/">Home</Link>
      </nav>
    </>
  );
}
```

Exercise

Create your Portfolio Website using ReactJS. You can find out layout
reference from:

- https://nicepage.com
- https://www.behance.net
- https://www.figma.com

Thank You!