## Part 1:

Let's start with the Basic HTML Hello World page.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Namaste React</title>

</head>

<body>

<H1>Hello World</H1>

</body>

</html>

Now let's create with Simple Javascript.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Namaste React</title>

</head>

<body>

<div id="root"></div>

<script>

const heading = document.createElement("h1");

heading.innerHTML = "Hello World via Javascript!";

const root = document.getElementById('root');

root.appendChild(heading);

</script>

</body>

</html>

## Part 2:

Now to add react in the code

Search: CDN React

script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

Place these 2 lines in the code.

As of the current version these files are old and will not be updated we have to use new versions.

Now let's create a hello world program with the help of React

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Namaste React</title>

</head>

<body>

<div id="root"></div>

<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

<script>

const heading = React.createElement('h1', {}, "Hello world with React!");

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

root.render(heading);

</script>

</body>

</html>

## Part 3

Let's break above code into 2 pieces index.html and app.js

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Namaste React</title>

</head>

<body>

<div id="root"></div>

<script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

<script src="./App.js"></script>

</body>

</html>

app.js

const heading = React.createElement('h1', {}, "Hello world with React!");

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

root.render(heading);

This will work the same as the above code. Just separated the JS and HTML

Now inline

const heading = React.createElement('h1', {}, "Hello world with React!");

What is 2nd element ‘{}’ ?

It is an object that contains an attribute for the element.

app.js

const heading = React.createElement('h1', {"id":"heading"}, "Hello world with React!");

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

root.render(heading);

When we will print on the console heading

It will show

The heading is a js object

And we are passing arguments to it.

3rd argument will be named as child along with 2nd argument and that will come in the props. (properties or attributes)

What is the function of the render function?

Its job is to create elements we have passed in it and show them on screen.

Converting objects into HTML tags and putting them on DOM.

## Part 4

How do you create a Nested Structure?

Support the structure is

<div id="parent">

<div id="child">

<h1>This is h1 tag.</h1>

</div>

</div>

Now let's see how to create it with the help of React.

const parent = React.createElement(

'div',

{

id: "parent"

},

React.createElement(

'div',

{

id:'child'

},

React.createElement(

'h1',

{},

"This is h1 tag"

)

)

);

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

root.render(parent);

How to add siblings to the tags like

<div id="parent">

<div id="child">

<h1>This is h1 tag.</h1>

<h2> This is h2 tag </h2>

</div>

</div>

We have to pass an array of children

// const heading = React.createElement('h1', {"id":"heading"}, "Hello world with React!");

const parent = React.createElement(

'div',

{

id: "parent"

},

React.createElement(

'div',

{

id:'child'

},

[

React.createElement(

'h1',

{},

"This is h1 tag"

),

React.createElement(

'h2',

{},

"This is h2 tag"

)

]

)

);

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

root.render(parent);

There is some Error in the above code, when you see the console it will show

Each child in a list must have a unique key.

Now

We can see the create element is more complicated to look at here as compared to js.

Here JSX plays a role.

## Part 5:

we can use react in existing projects.

It applies to the specific module only, everything in the module will be replaced in it. Other than that module we can use anything that won't be affected by react.

Explanation by AI.

### **Part 5: Integrating React into Existing Projects**

**You can integrate React into existing projects without affecting the entire application.** This approach is useful when you want to use React in specific parts of your application while leaving the rest of the project as it is.

#### **Key Points:**

1. **Module-Specific Integration:**
   * React can be applied to a particular module or component of your application. This means only the designated parts will use React, and other parts of the application will remain unaffected.
2. **Encapsulation:**
   * React components can be encapsulated within specific modules. This ensures that React’s behavior and updates are limited to those components and do not interfere with the rest of the application.
3. **Gradual Migration:**
   * Integrating React in this way allows for a gradual migration. You can incrementally adopt React without needing to refactor the entire project at once.
4. **Non-React Parts Remain Unchanged:**
   * The sections of the project that are not using React will continue to function as before, maintaining their original behavior and technology stack.

#### **Example**

Assume you have an existing HTML project with a traditional JavaScript interface, and you want to integrate React into just one part of your application, such as a user profile section.

**HTML File (index.html):**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Existing Project</title>

</head>

<body>

<div id="profile-section"></div>

<!-- Other non-React parts of the application -->

<script src="path/to/your/traditional-js.js"></script>

<script src="path/to/react.development.js"></script>

<script src="path/to/react-dom.development.js"></script>

<script src="path/to/your-react-app.js"></script>

</body>

</html>

**React Component (your-react-app.js):**

javascript

Copy code

// Define a React component for the user profile

const UserProfile = () => {

return (

<div>

<h1>User Profile</h1>

<p>Welcome to the user profile section!</p>

</div>

);

};

// Render the React component into the specified DOM node

const root = ReactDOM.createRoot(document.getElementById('profile-section'));

root.render(<UserProfile />);

**Explanation:**

* In the above example, the React component UserProfile is rendered inside the div with the ID profile-section. This integration is limited to this specific part of the application.
* Other parts of the HTML page, such as traditional JavaScript and the rest of the application, are unaffected by the React component.

By following this approach, you can leverage React for specific functionalities or modules while preserving the existing technology stack for other parts of your project.