Purwadhika
Digital Technology School

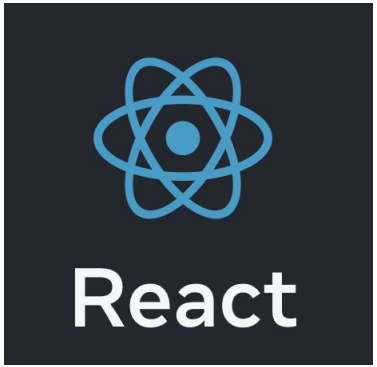
Full Stack AI Software Development

Introduction to React

Job Connector Program

What is React?

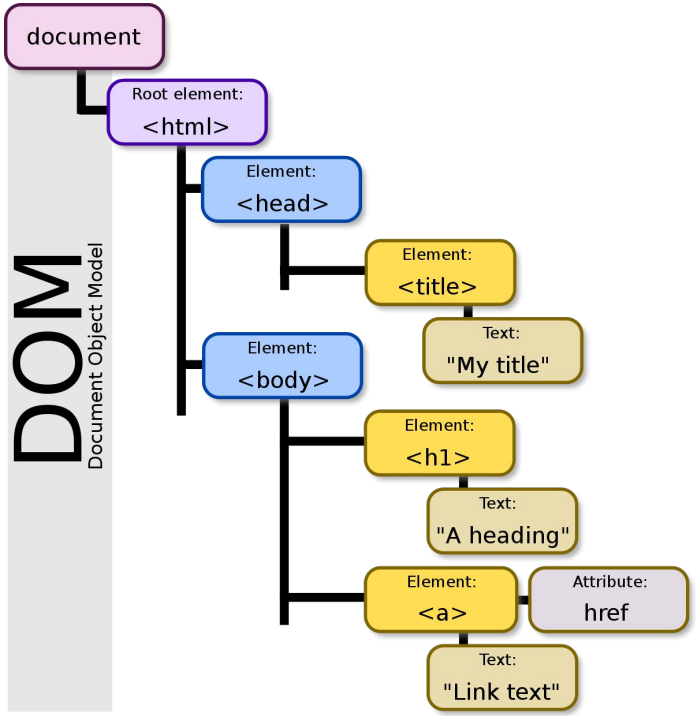
- **React** is a JavaScript library for building user interfaces, developed and maintained by Meta (formerly Facebook).
- Based on a component-based architecture, allowing developers to build encapsulated, reusable UI pieces.
- Manages application state to keep the UI automatically in sync with your data.
- Uses a **Virtual DOM** (more on this next!) to optimize rendering performance, making apps fast and responsive.



Find out more here:
https://react.dev/

What is the DOM?

- **Key Concept:** The **Document Object Model (DOM)** is a programming interface for web documents.
- **Analogy:** Think of the DOM as a tree structure where every HTML tag, piece of text, and attribute is a node or branch.
- **Function:** JavaScript uses the DOM to change the page's structure, style, and content.
- **The Problem:** Direct DOM manipulation is slow.
    - Every time you change an element, the browser may have to recalculate the layout and repaint the screen, which is computationally expensive.



DOM Manipulation Example: Text and Button Change

1. The HTML Structure (index.html)

We need two elements to target: an <h1> and a <button>.

```html
<h1 id="myHeading">Original Heading Text</h1>

<button id="myButton">Click Me to Change the Text</button>
```

2. The JavaScript (script.js)

This script runs when the button is clicked. It changes both the heading text and the button's own text.

```javascript
// 1. Select the elements we need
const headingElement = document.getElementById('myHeading');
const buttonElement = document.getElementById('myButton');

// 2. Define the function that performs the changes
function updateText() {
    // Change the text content of the heading element
    headingElement.textContent = 'The Text Has Been Updated!';

    // Change the text content of the button itself
    buttonElement.textContent = 'I Was Clicked!';

    // Optional: Add a style change too!
    headingElement.style.color = 'blue';
}

// 3. Attach the function to the button's 'click' event
buttonElement.addEventListener('click', updateText);
```

Basic Method in 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.

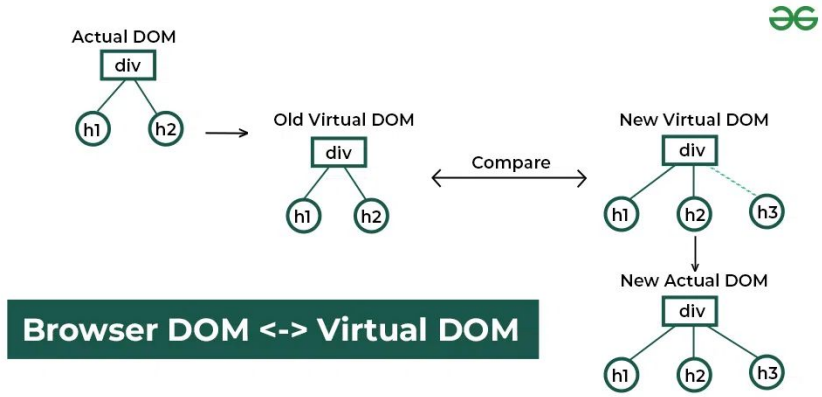
React's Solution: The Virtual DOM

1. State Changes

- React creates a Virtual DOM (VDOM), a lightweight copy of the real DOM, kept entirely in memory.
- When your application's state changes, React first updates this fast, in-memory VDOM.

2. Diffing & Reconciliation

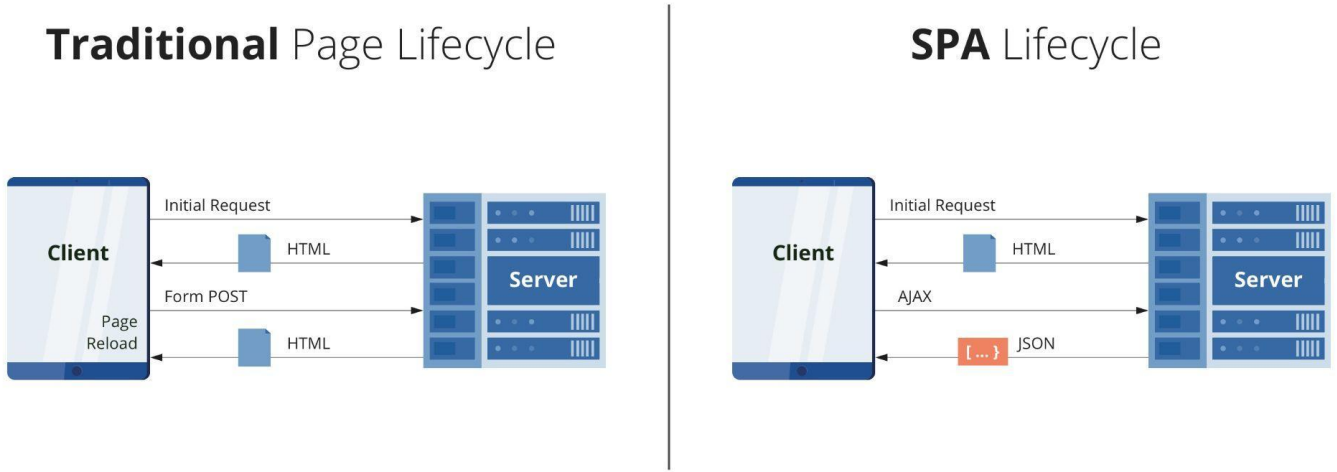
- React runs a "diffing" algorithm to compare the new VDOM with the old one, finding the minimal changes needed.
- It then "batches" these changes and applies only them to the real DOM. This process is called reconciliation.



Reference : *https://www.geeksforgeeks.org/reactjs-virtual-dom/*

Single Page Application (SPA)

Single Page Application (SPA) is a method where the UI can update without reloading the whole website. It uses one main HTML file as the render base, and only specific components are updated using the Virtual DOM. This makes rendering faster and more efficient, and this approach is used in modern JavaScript frameworks like React.

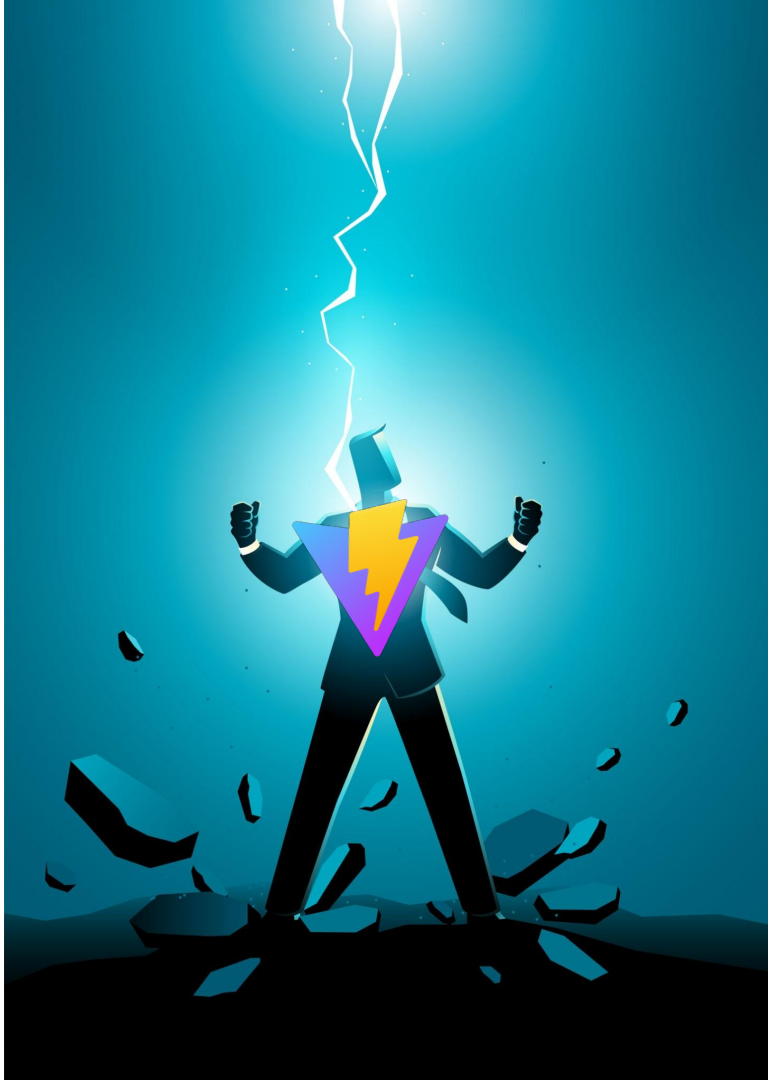


Introduction to Vite

- Vite (pronounced "veet," French for "quick") is a modern build tool and development server created by Evan You (creator of Vue.js).
- It provides an extremely fast development server and an optimized build process, replacing older, slower bundlers like Webpack (which powers Create React App).
- Key Innovation: It leverages native browser features to fundamentally change how code is served during development.

Why is Vite So Fast?

- Traditional Bundlers (e.g., Webpack): Must bundle your entire application before the dev server can start. This "cold start" gets slower as your project grows.
- Vite's Solution (Native ESM): Vite serves code over native ES Modules (ESM) in the browser. It only transforms and serves the code you actually request as you navigate.
- Instant Hot Module Replacement (HMR): Updates are handled over ESM, so only the edited module is replaced, not the whole bundle. This means changes appear near-instantly.



Vite: Production Build

- When ready for deployment, Vite switches to an optimized build process.
- Vite uses ESBuild (written in Go) to pre-bundle third-party dependencies. This is extremely fast (10-100x faster than traditional JavaScript bundlers).
- Main Bundler: Uses Rollup (a highly capable JavaScript bundler) internally for the final output.
- Output Quality: Generates highly optimized static assets with:
    - Tree-Shaking: Removal of unused code.
    - Code Splitting: Dividing the bundle into smaller, loadable chunks.

Let's Get Started

1. Create Project → **npm create vite@latest my-react-app -- --template react-ts**
2. Enter Directory → **cd my-react-app**
3. Install Deps → **npm install**
4. Run Server → **npm run dev**

Simple & Clean Project Structure

```
my-react-app/
├── node_modules/
├── public/
│   ├── vite-logo.svg
├── src/
│   ├── assets/
│   │   ├── logo.svg
│   ├── components/
│   │   ├── Header.tsx        ⟵ Using .tsx for React, or .vue for Vue
│   ├── App.tsx               ⟵ Main application component
│   └── main.ts               ⟵ Entry point (type-safe)
├── index.html
├── package.json
├── package-lock.json
├── vite.config.ts            ⟵ Configuration file is also TypeScript
└── tsconfig.json             ⟵ **Required** TypeScript configuration
```

Key Files

- **index.html**: The root HTML file (now in the project root, not in public/).
- **src/main.ts**: The entry point of your app. This is where you render your root React component.
- **src/App.tsx**: The main root component for your application.
- **src/assets/**: For static assets like images and CSS.

Code Structure

```jsx
import { useState } from "react";

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

  return (
    <div>
      <button onClick={() => setCount((count) => count + 1)}>
        count is {count}
      </button>
    </div>
  );
}

export default App;
```

About JSX

JSX stands for JavaScript XML. It is a syntax extension for JavaScript that allows you to write HTML-like structures directly within your JavaScript or TypeScript files.

Key Characteristics

- Syntactic Sugar: JSX is not standard JavaScript and is not run directly by the browser. It's simply "syntactic sugar" for calling the React.createElement() function.
- Declarative UI: It allows you to declare what the UI should look like based on your data and state, rather than manually instructing the browser how to build it (imperative approach).
- Compilation: Before the browser executes the code, a tool (like Babel or, in the case of Vite, esbuild/Rollup) converts the JSX syntax into regular JavaScript function calls.

TSX: TypeScript + JSX

Concept	Description	Example Syntax
TSX	The file extension for **T**ype**S**cript files containing J**X**.	`MyComponent.tsx`
Why Use It?	**Static Type Checking.** Catches errors (like misspellings or wrong prop types) *before* you run the code, significantly increasing stability in large apps.	`interface UserProps { name: string; age: number; }`
Type Definition	You define the structure and types for your component's `props`, ensuring strong contracts between parent and child components.	`const User: React.FC<UserProps> = ({ name, age }) => {...}`

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**.

Component Communication: Props (Properties)

What are Props?

Props are a mechanism for passing data from a parent component to a child component. They are read-only and immutable within the child component.

- **Direction:** Unidirectional Data Flow (Parent ➔ Child).
- **Immutability:** A child component must never modify its props. They are the component's external configuration.
- **Common Use:** Passing data, styles, or even function references (callbacks).

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

Props Example

Passing Data (Parent **App.tsx**):

```tsx
import { WelcomeMessage } from "./WelcomeMessage";

function App() {
  return (
    <div>
      <h1>Hello World!</h1>
      <WelcomeMessage name="Jhon Doe" />
    </div>
  );
}

export default App;
```

Receiving Data (Child **WelcomeMessage.tsx**):

```tsx
type Message = {
  name: string;
  message: string;
};

// Child Component
export function WelcomeMessage(props: Message) {
  // Access via props.name
  return <h2>Hello, {props.name}!</h2>;
}
```

Add Styling

There is several ways to styling component in React:

- Global CSS
- CSS Modules
- Inline Styles

Styling in React: Global CSS

Standard stylesheets imported once. Styles cascade and apply to the entire application.

Your CSS file (src/index.css):

```css
/* This style applies to the whole app */
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
}

/* This is a global class */
.button-global {
  padding: 10px 15px;
  background-color: dodgerblue;
  color: white;
  border: none;
  border-radius: 4px;
}
```

Import in your entry point (src/main.jsx):

```jsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css' // <-- IMPORTED HERE

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

Usage in any component (src/App.jsx):

```jsx
function App() {
  // This class is not imported, it's just known globally
  return <button className="button-global">Click Me!</button>;
}
```

Styling in React: CSS Modules

CSS file is named ***.module.css**. It's imported as a JavaScript object, and class names are automatically hashed to be unique.

Your CSS Module file (src/components/Button.module.css):

```css
/* This class is local to the component that imports it */
.button {
  padding: 10px 15px;
  background-color: darkgreen;
  color: white;
  border: none;
  border-radius: 4px;
}

.button:hover {
  background-color: limegreen;
}
```

Your Component (src/components/Button.jsx):

```jsx
import React from 'react';
// Import the styles as a 'styles' object
import styles from './Button.module.css';

function Button() {
  // Use the hashed class name from the 'styles' object
  return (
    <button className={styles.button}>
      This Button is Scoped
    </button>
  );
}
// Rendered class will be unique, like "Button_button__a1b2c"
```

Styling in React: Inline Styles

Styles are defined as a JavaScript object and passed to the style attribute. Property names must be **camelCase**.

Static Inline Style

```jsx
// Styles are defined as a JS object
const headerStyle = {
  color: '#61DAFB',
  fontSize: '32px',
  borderBottom: '2px solid #61DAFB' // camelCase
};

function Header() {
  return <h1 style={headerStyle}>My React App</h1>;
}
```

Dynamic Inline Style

```jsx
function WelcomeMessage({ isError }) {
  // Style changes based on the 'isError' prop
  const messageStyle = {
    color: isError ? 'red' : 'green',
    padding: '15px',
    border: isError ? '1px solid red' : '1px solid green',
  };

  return (
    <div style={messageStyle}>
      {isError ? 'Error!' : 'Success!'}
    </div>
  );
}
```

React Router DOM

Enables client-side routing in a Single-Page Application (SPA) by mapping URL paths to components without a full page reload.

- Installation: **npm install react-router-dom**
- Routes Definition: Use the **<Routes>** and **<Route>** components to define path-to-element mappings:

```
<Routes>
  <Route path="/" element={<HomePage />} />
  <Route path="/users/:id" element={<UserProfile />} />
</Routes>
```

- Navigation: Use the **<Link to="/path">** component instead of the standard <a> tag for smooth client-side transitions.

Exercise

Create your Profile Website using React . You can find out layout reference from :

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

This exercise is introduction for code challenge project when started in Day 04.

Thank you