

**React Notes**

**📌 What is React.js?**

React.js is a **JavaScript library** for building **user interfaces (UIs)**, mainly for **Single Page Applications (SPAs)**. Developed by **Facebook (Meta)**, it allows developers to create **reusable UI components** and efficiently manage application state using the **Virtual DOM**.

**Key Features:**  
✔ **Component-Based Architecture** – Reusable and modular UI.  
✔ **Virtual DOM** – Optimized rendering for better performance.  
✔ **Unidirectional Data Flow** – Ensures predictable state management.  
✔ **SEO-Friendly** – Works well with server-side rendering (SSR) using Next.js.

**📌 History of React.js**

📍 **2011** – Created by **Jordan Walke** at Facebook to improve UI performance.  
📍 **2013** – Open-sourced at **JSConf**.  
📍 **2015** – **React Native** launched for mobile development.  
📍 **2016** – **React Fiber** project started to enhance performance.  
📍 **2017** – **React 16** released with React Fiber.  
📍 **2020** – **React 17** introduced gradual upgrades and event delegation.  
📍 **2022-2023** – **React 18** added **automatic batching, concurrent rendering, and transitions** for a better user experience.

**📌 Real-World Projects Using React.js**

🚀 **Facebook & Instagram** – Core platforms built using React.  
🚀 **WhatsApp Web** – Uses React + Redux for real-time chat.  
🚀 **Netflix** – Optimized UI performance.  
🚀 **Airbnb** – Interactive booking UI with React Fiber.  
🚀 **Uber & Uber Eats** – Smooth navigation and dynamic UI.  
🚀 **Discord** – Web chat app using React.  
🚀 **Pinterest** – Handles infinite scrolling efficiently.  
🚀 **Reddit** – Modern web interface powered by React.  
🚀 **Shopify** – Uses React for admin dashboards & storefronts.

**Why Use React?**

✅ **Fast rendering** with Virtual DOM.  
✅ **Reusable components** for scalable apps.  
✅ **Strong community & ecosystem** with Next.js, Redux, and React Router.

**📌 Installation of React**

**1.installation of bun**

>npm i -g bun

>check if its install by => bun -v

**2.installation of react**

>bun create vite

>then follow the cmd for further installation

3.if there is an error of esbuild then run -> bun rebuild esbuild

**📌 Project structure**

1. **node\_modules/** - This folder contains all the dependencies (packages) installed using npm or yarn. It is automatically created when you run npm install or yarn install. You usually don’t modify anything inside this folder.
2. **public/** - This folder is for static assets like images, icons, and fonts that do not get processed by Vite. It usually contains files that should be available directly in the browser.
3. **src/** (Source Folder) - This is where your actual project code lives.
   * **app.jsx** - This is the main application component. It usually contains the core structure of your React app.
   * **main.jsx** - This is the entry point of the app. It connects React to the index.html file and renders the App component.
   * **.eslintrc.cjs** - Configuration file for ESLint, a tool that helps maintain code quality by catching errors and enforcing coding standards.
   * **.gitignore** - A file that tells Git which files or folders to ignore when committing code (e.g., node\_modules).
   * **index.html** - The main HTML file that loads your React app. It contains a <divid="root"></div> where your React components are injected.
   * **package.json** - This file contains details about your project, including dependencies, scripts, and configurations.
   * **vite.config.js** - The configuration file for Vite, a fast frontend build tool. It allows customization of settings like plugins and server behavior.

**What is JSX?**

 JSX stand for JAVASCRIPT XML . it allows you to write HTML elements in JAVASCRIPT and place them in the DOM without any **createElement()** and **appendChild()** methods

**How JSX Works**

When[React](https://www.geeksforgeeks.org/react/) processes this JSX code, it converts it into JavaScript using **Babel**. This JavaScript code then creates real HTML elements in the browser’s DOM . which is how your web page gets displayed.

**React.createElement()**

It is a core function in React that is used to create React elements without JSX. When JSX is used, it gets transpiled into React.createElement calls by Babel.

**The syntax is:**

**React.createElement(type,{props},children);   
// JSX code   
<type {props} >{children}</type**

1. type: The type of the element (e.g., 'div', 'h1', or a React component).

2. props: An object containing properties for the element.

3. children: The content inside the component, which can be text, another element, or an array of elements.

**ReactDOM :** React DOM contains the arguments that are necessary to render react elements in the browser.

**Syntax:**

**ReactDOM.render(element,containerElement);**

Parameters: ReactDOM.render() takes two arguments:

1.element: The element that needs to be rendered in the DOM.

2.containerElement: Where to render in the dom.

Example of code :-

**React components**

React components are independent, reusable building blocks in a React application that define what gets displayed on the UI.

 They accept inputs called props and return React elements describing the UI.

**There are two primary types of React components:**

**1. Functional Components**

[Functional components](https://www.geeksforgeeks.org/reactjs-functional-components/) are simpler and preferred for most use cases. They are[JavaScript functions](https://www.geeksforgeeks.org/functions-in-javascript/) that return [React](https://www.geeksforgeeks.org/react/) elements. With the introduction of [React Hooks](https://www.geeksforgeeks.org/reactjs-hooks/), functional components can also manage state and lifecycle events.

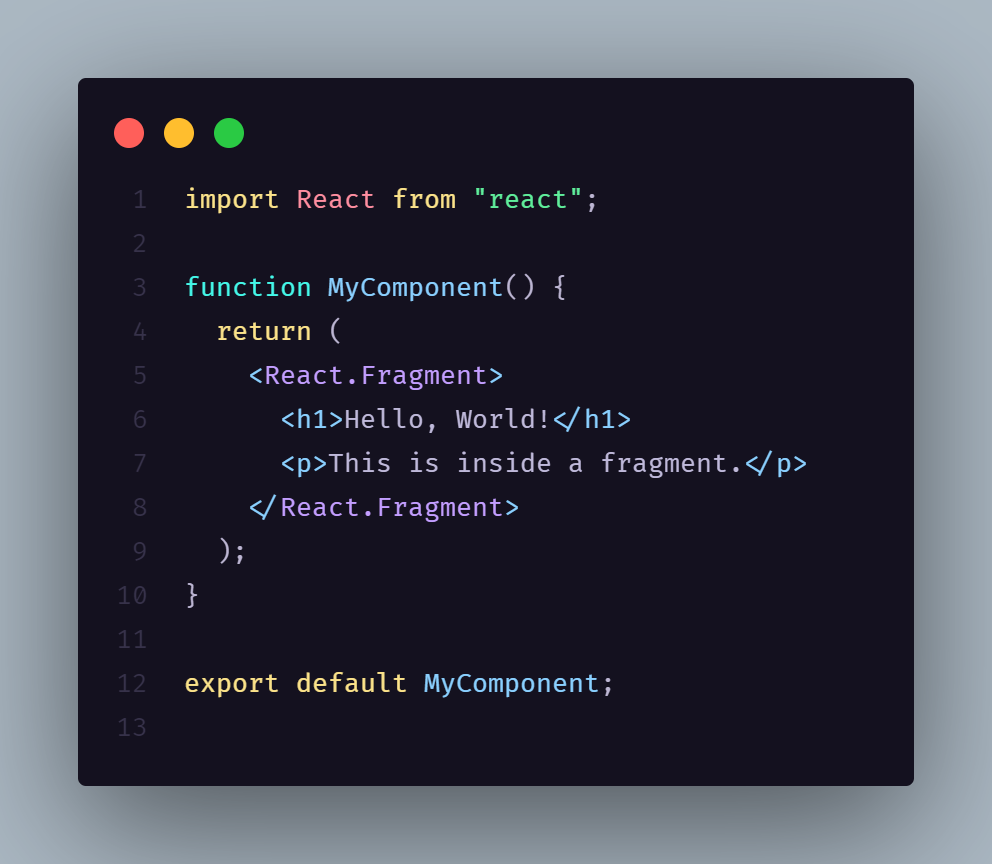
* **Stateless or Stateful:**Can manage state using React Hooks.
* **Simpler Syntax:**Ideal for small and reusable components.
* **Performance:** Generally faster since they don’t require a ‘this’ keyword.

**React fragments**

A **Fragment** in React is a special wrapper that allows grouping multiple elements **without adding extra nodes to the DOM**. This is useful when returning multiple elements inside a component without wrapping them in an unnecessary <div>.

**How to use it?**

1. **Using <React.Fragment>**

****

1. **Using shorthand <>…</> syntax {recommended}**

****

**Import and Export in React**

In React.js, **import** and **export** are used to modularize the code. They allow you to split components, functions, or variables into separate files and reuse them across your application.

**Exporting a component** 📤 → Means sharing it so other files can use it.

**Importing a component** 📥 → Means bringing it into another file so you can use it there.

**Types of Exports in React**

**In React, there are two types of exports**

1. Default Exports and Imports
2. Named Exports and Imports

**1. Default Export and Import**

A default export allows you to export a single component or variable from a file. When importing a default export, you can give it any name you choose.

**import** React **from** "react";

**const** MyComponent = () => {

**return** <h1>Hello **from** MyComponent!</h1>;

};

**export** **default** MyComponent;

**import** React **from** "react";

**import** MyComponent **from** "./components/MyComponent";

**const** App = () => {

**return** (

<div>

<MyComponent /> {*/\* Using the imported component \*/*}

</div>

);

};

**export** **default** App;

**In this code**

* **Default Export:**Exports one thing from a file (like a component).
* **Default Import:**Imports the default export from another file, naming it as needed.

**2. Named Export and Import**

Named exports allow you to export multiple components or variables from a single file. When importing a named export, you must use the exact name of the exported entity.

**export** **const** MyComponent = () => {

**return** <h1>Hello **from** MyComponent!</h1>;

};

**export** **const** AnotherComponent = () => {

**return** <h1>Hello **from** AnotherComponent!</h1>;

};

**import** { MyComponent, AnotherComponent } **from** "./components/component.js";

**const** App = () => {

**return** (

<div>

<MyComponent />

<AnotherComponent />

</div>

);

};

**export** **default** App;

This code demonstrates **Named Export and Import** in React:

* **Exporting Components:** MyComponent and AnotherComponent are exported individually using export.
* **Named Import:** They are imported using {} from components.js, meaning their names must match exactly.
* **Usage in App Component:** Both components are used inside App to render them.

1. **Combining Default and Named Exports in React**

You can export both a default component and named components from the same file. This allows flexibility in importing.

Same as but in App.jsx

**Syntax is 🡺 import {//for named component},//for default component from “./src”**

**Looping in react**

**Event Handling in React**

**Passing event handling by Props**

**Event Propagation**