React.js Notes

# What is React

* JavaScript library to build Dynamic and interactive user interfaces.
* Developed at Facebook in 2011.
* Currently the most widely used JavaScript library for front-end development.
* Used to create single-page applications.

# Working of DOM

* Browser takes HTML documents and creates DOM
* JS helps us modify DOM based on user actions or events.
* In big applications, working with DOM becomes complicated.

# Problems with JavaScript

* React has a simpler mental model
* JS is cumbersome
* JS is Error-prone
* JS is Hard to maintain

# Working of React

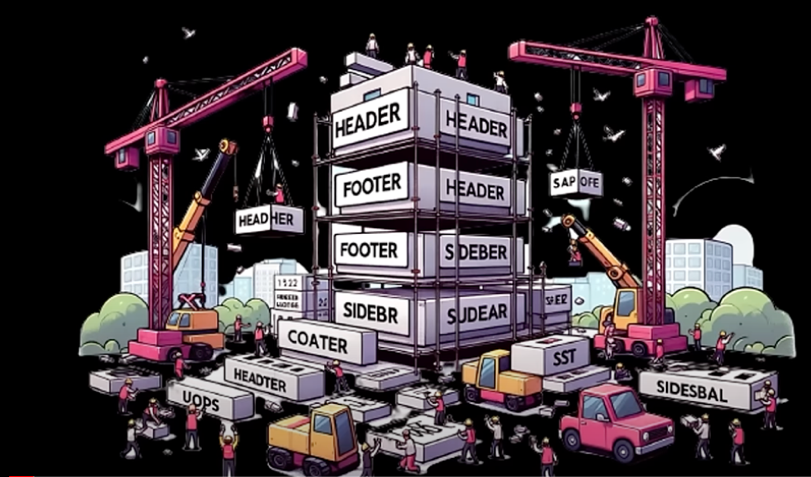
* No need to worry about querying and updating the DOM.
* React creates a web page with small and reusable components.
* React will take care of creating and updating DOM elements.
* It saves a lot of time, cheezein assn hai, pahele se likhe hua hai.

# JS Vs React

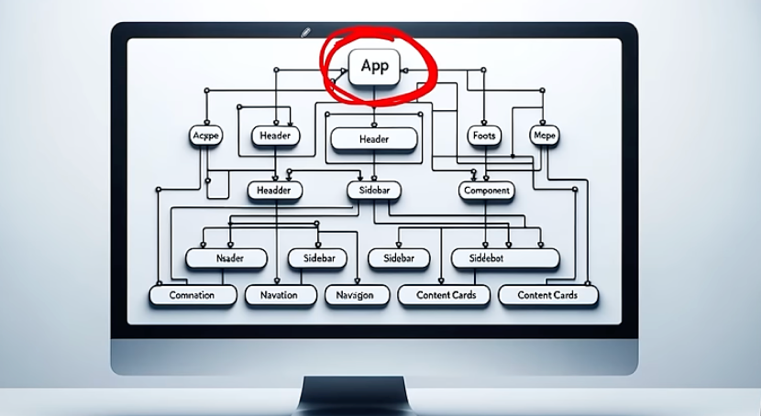
* JS is imperative: You define steps to reach your desired state.
* React is Declarative: You define the target UI state and then React figures out how to reach that state.

# Introduction to Components

* Components help us write reusable, modular and better-organized code.



* React application is a tree component with App Components as the root bringing everything together.



# Create a React App

* The official tool is CRA (Create React APP)
* Vite is a modern tool for creating React Project
* Vite produces Quick and Small bundle sizes.
* Vite: Use npm run dev to launch the dev server.
* User npm start for CRA.
* Npm is a node package manager
* In terminal use: npm create vite@latest

We are telling the node that in your package manager create a vite with the version name. You can specify the latest version of Vite.

* Summary: Steps to create a react project with vite

Step 1: npm create vite@latest

Step 2: name the project name, and project package, select framework, select the variant

Step 3: npm install

Step 4: npm run dev

Step 5: copy the url eg <http://localhost:5173/> and paste it in the browser.

# Project Structure

* **node\_modules**/ has all the installed node packages
* **public/** Directory: Contains static files that don’t change.
* **Src/** Directory: Main folder for the React code.

1. **components/:** Reusable parts of the UI, like buttons or headers
2. **assets/:** Images, fonts, and other static files.
3. **styles/:** CSS or stylesheets.

* **package.json** contains information about the project like name, version, and dependencies on other react packages.
* **Vite.config.js** contains vite config.

# File Extensions

* .js

Stands for JavaScript, contains regular JavaScript code. Used for general logic and components

* .jsx

Stands for JavaScript XML. Combines JavaScript with HTML-like tags

Make it easier to design UI components.

# Class vs Function Components

* Class Components

-Stateful: can manage state (like methods and property)

-Lifecycle: Access to lifecycle methods

-Verbose: More boilerplate code.

-Note Preferred anymore.

* Functional Components

-Initially stateless (by default you cannot give value or no property)

-Ca use Hooks for state and effects.

-Simpler and more concise

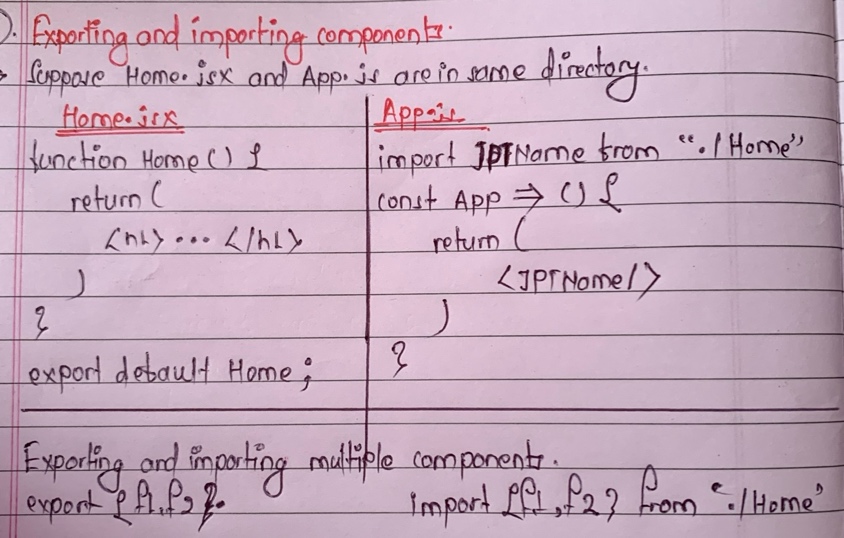
- More Popular.

# Introduction to JSX file

* **Definition:** JSX determines how the UI will look whenever the component is used.
* **Not HTML:** Though it resembles HTML, you’re writing JSX, which stands for JavaScript XML.
* **Conversation:** JSX gets converted to regular JavaScript code.
* **Babeljs.io/repl**  is a tool that allows you to see how JSX is transformed into JavaScript.

# Exporting and Importing components

* **Enables** the use of components in other parts.
* **Default Export:** Allows exporting a single component as the default from a module.
* **Named Export:** Allows exporting multiple items from a module.
* **Importing:** To use an exported component, you need to import it in the destination file using import syntax.

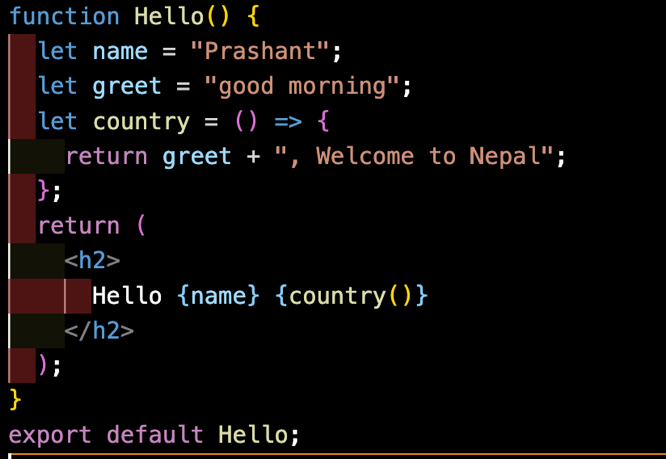
****

# Other Important Points

* **Naming:** Must be capitalized; lowercase for default HTML.
* **HTML:** Unlike vanilla JS where you can’t directly write HTML, in React, you can embed HTML-like syntax using JSX.
* **CSS:** In React, CSS can be directly imported into component files, allowing for modular and component-specific styling.

# Dynamic Components

* **Dynamic Content:** JSX allows the creation of dynamic and interactive UI components.
* **JavaScript Expressions:** Using {}, we can embed any JS expression directly within JSX. This includes variables, function call, and more.



# Reusable components

* Modularity: Components are modular, allowing for easy reuse across different parts of an application.
* Consistency: Reusing components ensures UI consistency and reduces the chance of discrepancies.
* Efficiency: Reduces development time and effort by avoiding duplication of code.
* Maintainability: Changes made to a reused component reflect everywhere it’s used, simplifying updates and bug fixes.

# Including Bootstrap

* **Responsive:** Mobile-first design for all device sizes.
* **Components:** Pre-styled elements like buttons and navbars
* **Customizable:** Modify default styles as needed.
* **Cross-Browser:** Consistent look across browsers.
* **Open-Source:** Free with community support.ls

**i. Install:** npm I [bootstrap@5.3.2](mailto:bootstrap@5.3.2)

**ii. import:** import “bootstrap/dist/css/bootstrap.min.css”

🡪 it is basically written in main.JSX file because this is CSS part.

# Fragments

* Render method will only render a single root node inside it at a time.
* In React components, the render method is used to display content on the screen, but when rendering multiple elements, developers often use a ‘div’ tag as a wrapper. React 16.2 introduced Fragments as a cleaner alternative, allowing multiple elements to be rendered without the need for an additional ‘div’ wrapper.
* Return multiple elements without a wrapping parent.
* Syntax:

1. <React.Fragment>……</React.Fragment>
2. Short: <>…</>

# Fragments

# Fragments

# Fragments

# Fragments

# Fragments

# Fragments