React.js Introduction

React.js is a **JavaScript library** used for building **user interfaces**, especially **single-page applications (SPAs)**. It allows developers to create **reusable UI components** and efficiently update the UI using a **virtual DOM**.

React makes web applications faster, more scalable, and easier to manage.

History of React.js:

Created by **Facebook**.

- **React** was created by **Jordan Walke**, a software engineer at **Facebook**.
- First used in **Facebook's News Feed**.
- Open-source and widely used in industry.
- latest version of React.js is **React 19**, released on **Dec 5, 2024**.

Why use React (Advantages)

- **Component-Based** – Reusable, modular UI components.
- **Virtual DOM** – Fast updates and rendering.
- **Declarative** – Easier to read and debug UI code.
- **Rich Ecosystem** – Large community, libraries, and tools.
- **Strong Community Support** – Active development and job demand.
- **Unidirectional Data Flow** – Predictable and manageable state.
- **JSX Syntax**– HTML + JavaScript in one place, easier development.(Allows writing HTML-like code inside JavaScript.)
- **Fast Rendering** – Uses Virtual DOM for efficient updates.
- **Reusable Components** – Promotes code reusability and maintainability.
- **Cross-Platform** – Supports web (React) and mobile (React Native).

Limitations of React

- **JSX Complexity** – Not intuitive for beginners.
- **Learning Curve** – Needs understanding of JSX, state, props, hooks, etc.
- **View Only** – Requires integration with other libraries for routing, state management.
- **Frequent Updates** – Fast changes can cause compatibility issues.
- **Boilerplate Code** – Setup can be complex without tools like Create React App or Next.js.
- **SEO Limitation** – Needs SSR (like Next.js) for better SEO.
- **Performance Overhead** – Poorly optimized components can lead to performance issues.

React vs Traditional JavaScript

Aspect	React	Traditional JavaScript
Approach	Component-based, declarative UI	Imperative DOM manipulation
DOM Updates	Uses Virtual DOM for efficient rendering	Direct DOM manipulation (slower, costly)
Code Structure	Modular, reusable components	Often monolithic, less modular
State Management	Built-in hooks (useState, useEffect)	Manual state tracking and event handling
Learning Curve	Higher, requires understanding JSX, React concepts	Lower, uses plain JS and DOM APIs
Maintainability	Easier with reusable components	Can get complex as app grows
Performance	Optimized with Virtual DOM	Can be slower with frequent DOM updates
Development Speed	Faster with React’s ecosystem and tools	Slower, requires manual handling of UI updates

Virtual DOM

Virtual DOM is a lightweight JavaScript representation of the actual DOM used by React to optimize and efficiently update the UI by minimizing direct DOM manipulations.

Virtual DOM is a lightweight in-memory copy of the real DOM used by React.

How Virtual DOM works:

- **Initial Render** – React creates a Virtual DOM from JSX.
- **State/Props Change** – A new Virtual DOM is created.
- **Diffing** – React compares the new Virtual DOM with the previous one.
- **Reconciliation** – It identifies the minimal set of changes.
- **Update Real DOM** – Only the changed parts are updated in the real DOM.

Benefits:

- Faster updates.
- Better performance.

React Reconciliation

- **React Reconciliation** is the process React uses to update the DOM efficiently when a component's state or props change.

Steps:

- **New Virtual DOM created** after state/props change.
- **Diffing Algorithm** compares new vs old Virtual DOM.
- **Minimal changes identified**.
- **Real DOM is updated** only where necessary.
- **Goal:** Fast and efficient UI updates with minimal DOM manipulation.

React Fiber

- **React Fiber** is the reimplementation of React's core algorithm for rendering and reconciliation, introduced in React 16.

  Key Features:

- **Incremental Rendering** – Splits rendering into units of work, improving responsiveness.
- **Prioritization** – Handles high-priority updates (e.g. animations) first.
- **Concurrency** – Supports async rendering for better user experience.
- **Error Handling** – Improved error boundaries and recovery.

Purpose:

To make React more efficient, flexible, and capable of handling complex UIs smoothly.

Difference Between React Fiber and React Reconciliation

Aspect	React Reconciliation	React Fiber
Definition	Process of updating the DOM efficiently	New architecture of React’s rendering engine (from v16)
Purpose	Diff old and new Virtual DOM to apply updates	Improve rendering with prioritization and interruption
Method	Old algorithm (stack-based, synchronous)	Fiber algorithm (linked list-based, asynchronous)
Performance	Less control over update timing	Allows pausing, resuming, and aborting rendering tasks
Concurrency	Not supported	Supports concurrent rendering
Error Handling	Basic	Improved with error boundaries

Components in React

- A component is a self-contained, independent, reusable piece of code that defines how a certain part of the UI should appear and behave.
- **Components in React** are the building blocks of a React application's UI.
- Types of Components in React:
- **Functional Components** – Simple JavaScript functions using hooks.
- **Class Components** – Use ES6 classes and lifecycle methods (older approach).

Features:

- Reusable and modular.
- Accept **props** and manage **state**.
- Return JSX to render UI.

Class Components

- These are ES6 classes that extend React.Component.
- They manage state and use lifecycle methods (like componentDidMount, componentDidUpdate).
- The component also requires a render() method, this method returns HTML.
- **Example:**

```
class Greeting extends React.Component {

  render() {

 return     <h1>Hello, {this.props.name}!</h1>;

        }

   }
```

Functional Components

- These are JavaScript functions that return JSX (JavaScript XML).
- With the introduction of React Hooks (like useState, useEffect), functional components can now manage state and side effects.
- They are simpler, more concise, and preferred in modern React development.
- **Example:**

```
 function Greeting(props) {

return  <h1>Hello, {props.name}!</h1>;

    }
```

Difference between Functional and Class Components

Aspect	Functional Components	Class Components
Syntax	JavaScript functions returning JSX	ES6 classes extending React.Component
State Management	Use React Hooks (useState, useEffect, etc.)	Use this.state and setState
Lifecycle Methods	Managed via Hooks (useEffect)	Built-in lifecycle methods (componentDidMount, componentDidUpdate, etc.)
this Keyword	No this context	Uses this to access props, state, and methods
Performance	Slightly faster and simpler	Slightly heavier due to class overhead
Readability	More concise and easier to read	More verbose
Hooks Support	Fully supports Hooks	Does not support Hooks directly
Introduced in	React 16.8+	Since early React versions