

☑ Also known as React.js/ React.JS

Free & Open-source JavaScript library for building user interfaces (UIs).

Created by Meta (Facebook) & is now maintained by Meta & community of developers.

HTML/CSS/JS code refreshes the whole page unexpectedly upon interaction (like clicking a button, submitting a form, etc.)

- React is used to build *SPAs (Single Page Applications)*
- Creates *reusable UI components*
- Develops dynamic, fast & scalable front end applications.

🔗 **ReactJS uses a Virtual DOM mechanism to efficiently update the HTML DOM.**

Instead of reloading the entire DOM, the virtual DOM updates only the specific elements that have changed, resulting in faster performance.

 React follows a Component-Driven Architecture

The user interface is built by composing independent, reusable pieces called components. Each component manages its own logic and rendering, making development more modular, scalable, and easier to maintain.

A Component is a reusable UI element.

React Element : an object that describes the properties of an actual DOM node which will be created by React.

Simple Structure of a React Elements:

```
{
  type: "button"
  props: {
    classname: "btn"
  }
}
```

✳️ **SPAs(Single Page Application)**

A web application that loads a single HTML page and dynamically updates content as users interact with it is known as a Single Page Application (SPA).

In an SPA, the entire page doesn't reload with every interaction; only the necessary parts of the page are updated, providing a smoother and faster user experience.

- It uses client side rendering (CSR) [Javascript Updates the DOM]
- Communicates to server via APIs (e.g. REST, GraphQL etc.)

Example: Facebook, Gmail, Twitter

☑ Pros

- Faster Navigation (No full page reloads)
- Smoother UX
- Easier to build complex, interactive UIs
- Good for real time updates

✗ Cons

- Slower initial loads (Heavy JS Bundles)
- Search engines historically struggled with Javascript heavy SPAs. But solutions like Next.JS do help.
- Browser history management needs extra efforts (e.g React Router)

✱ MPAs(Multi Page Application)

A traditional web app where each page is a separate HTML document loaded from the server.

- It uses Server side rendering (SSR) [Server generates HTML for each request]

Example: Amazon, Wordpress, Old school sites

☑ Pros

- Better SEO out of the box as each page is a separate HTML page.
- faster initial loads (Less Javascript code, server handles rendering)
- Easier to scale as pages can be cached independently.

✗ Cons

- Slower navigation (full page reloads upon each interaction)
- Less interactivity (more clunky UX)
- Harder to maintain state (e.g Shopping Cart across pages)

📌 Where to use SPA & MPA**

- If you need a dynamic, app alike experience you need to use SPA (e.g SaaS Tool, Social Network Site)
- If you prioritize SEO, Simplicity, Server-Driven content you need to use MPA (e.g Blogs, News etc.)

☞ Hybrid Approach

Modern frameworks like **Next.JS (React)**, **Nuxt.JS (Vue)** allows **SSR + SPA** hybrid models for a better SEO & Optimization.

This approach is best for both cases.



React application is made up of Multiple Components

- Each of them are **responsible for outputting a small, reusable piece of HTML**.
- **Components can be nested within other components to allow complex applications** to be built out of simple building blocks.



Key Features of React

1. **Component Based Architecture** : React apps are built using reusable components, making code easier to manage & scale
2. **Virtual DOM** : React uses a Virtual DOM to efficiently update only the parts of the page that change, improving performance.
3. **Declare Syntax** : Instead of manually updating the DOM like Vanilla JS, you describe how the UI should look & React handle updates automatically.
4. **Strong Ecosystem** : React has a massive community, tons of libraries (like Redux, Next.JS & Extensive documentation)
5. **Works with other frameworks** : React can be integrated with backend/mobile apps. (Node.JS, Django) [React Native]



Library

A collection of pre-written functions/modules that you can call as needed. It only provides the boilerplate for the project.

Key Idea : You can control the application's flow & structure. The library will provide you utility.

Analogy : Like a toolbox, you'll pick which tools (functions) to use & when to use.

Example: React (UI Library), Lodash (Utility Function), JQuery (DOM Manipulation)

Characteristics :

1. Flexibility : Uses only what I need
2. Less Opinionated : No Strict rules on application structure
3. More Responsibility : You decide how to integrate everything together.



Framework

A fully featured structure that dictates how to build an application (includes library, tools, rules)

Key Idea : It controls the application's flow. The library will provide you utility.

Analogy : Like a blueprint, you build within it's rules.

Example: Angular, Django, Ruby on Rails

Characteristics :

1. Built in tools like tools for routing, state management etc. is already included.
2. Scalability : Enforces best practices for large teams.
3. Less flexibility : Must follow the framework's convention.

🔗 React A Framework or Library

React is a library (for UI components), but it's an ecosystem (React Router, Redux, Next.JS) can feel like a framework.

Next.JS is a framework built on React as it adds routing, SSR and other opinionated features.

➡ Why does Library/Framework matter?

Libraries give you freedom but requires more decisions. While Frameworks speed up the development but limit flexibility.

- We use library if we need lightweight control (e.g React for dynamic UI)
- We use a framework if we want to Structure/Reliability (e.g Angular for Enterprise Applications)

Features	Library (e.g React)	Framework (e.g Angular)
Control Flow	You call the library	Framework calls your code
Flexibility	High (Pick & Choose)	Low (Follow Conventions)
Use Case	Add functionality to an app	Build fullstack apps
Learning Curve	Easier to Start	Steeper
Use Case	Lightweight	Often heavier

☑ Virtual DOM

The Virtual DOM (VDOM) is a lightweight, in-memory representation of the real Document Object Model (DOM) used by the libraries like React to optimize UI updates.

It's a core reason why react is so fast.

Working Procedure

1. Initial Render : React creates a Virtual DOM tree (basically a JavaScript Object) that mirrors the real DOM.

HTML

```
<div>
  <h1> Hello React </h1>
</div>
```

React DOM

```
{type: 'div',
  props: {
    children: {
      type: 'h1',
      props: {
        children: "Hello React!"}}}}}
```

Every HTML element becomes an object with:

- type: The tag name as a string
- props: An object containing all properties (including children)

The children prop is special:

- Can be a string (for text nodes)
- Can be another object (for nested elements)
- Can be an array (for multiple children)

2. State/Props Change : When data changes (e.g `setState`), React creates a new Virtual DOM tree.

3. Diffing (Reconciliation) : React compares the new VDOM with the previous DOM (Diffing Algorithm) to find out minimal changes that are needed.

4. Batch Update to Real DOM : Only the changed parts are updated in real DOM (avoiding full renders, which is time & memory consuming)

Why do we use Virtual DOM?

- It minimizes direct DOM manipulations (slowest parts of the web application)
- Batches multiple updates into a single render cycle.
- Cross Platform : React Native can also use similar concepts for making mobile UIs

Features	Virtual DOM	Real DOM
Speed	Fast (JS Objects)	Slow (Browser regenerates code everytime)
Updates	Batched & optimized	Immediate & memory costly
Memory	Lightweight (Less memory)	Heavy (Browser Rendered)

Reconciliation

React's algorithm for efficiently upgrading the DOM when state/props change.

- It's the diffing process that compares the new Virtual DOM with the previous one to determine the minimal changes needed for the real DOM.

➡ How does it work?

- Trigger : A component's state/props change
- New Virtual DOM tree : React creates a new virtual DOM tree.
- Diffing : React compares the new tree with the old one. (reconciliation)
- Update : Only the changed parts are applied to the real DOM.

📄 Reconciliation v/s Virtual DOM?

Virtual DOM is a lightweight copy of real DOM while Reconciliation is an algorithm that compares Virtual DOM snapshot with the old DOM to update the real DOM efficiently.

- Reconciliation minimizes costly DOM operations.
- Gives smooth UI & updates feel instantaneous.
- Complex diffing can slow down a lot of large scale applications.

👉 JSX

JavaScript XML/ JSX is a syntax extension for JavaScript, primarily used with React to describe UI components.

Till now we've separately been using HTML, CSS, JS

Through JSX we can write HTML like code in JavaScript, which later on will be transformed into JavaScript code by React library.

- JSX & React are different. We can write React without JSX.

React.createElement => Object => HTMLElement(render)

```
const Heading = React.createElement ("h1", {id: "heading"}, "This is React")
```

JSX - HTML like/ XML like syntax

```
const jsxHeading = <h1 id="heading"> This is React </h1>;
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(heading);
```

As a better example:

```
<!DOCTYPE html>
<html>
  <body>
    <div id="root"></div>

    <script src="https://unpkg.com/react@18.3.1/umd/react.development.js">
```

```

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

  <script>
    function App() {
      return React.createElement(
        "div",
        {},
        React.createElement("h1", {}, "Hello World")
      );
    }

    const rootContainer = document.getElementById("root");
    const root = ReactDOM.createRoot(rootContainer);
    root.render(App());
  </script>
</body>
</html>

```

- A basic HTML element with `id="root"` — this is the mount point for the React app.
- `App` is a React component. It returns a React element (`<div><h1>Hello World</h1></div>`) using `React.createElement`.
- No JSX is used here — it's all vanilla JS.
- `rootContainer` gets the `div#root`.
- `ReactDOM.createRoot()` creates a root for the React app (React 18 style).
- `root.render(App());` renders the result of calling the `App` function.

JSX is not pure JS as JS engine doesn't understand JSX. JSX code gets transpiled (converted into the code browser can understand) prior going into the JSX engine.

Component Structure + JSX Structure:

```

import React from 'react';

function MyComponent() {
  return (
    <div>
      <h1>Hello World!</h1>
      <p>This is Pratik Speaking</p>
    </div>
  );
}

export default MyComponent;

```

In XML we can create our custom tags which we call user defined tags

JSX => React.createElement => ReactElement- JS Object => HTML Element (at the time of Render)

We need to use cammel case while declaring attributes in JSX.

🌟 Key Features of JSX

- Embedded Expressions : You can embed JS expressions inside JSX using `{}`

```
const name = 'Alice'  
const greeting = <h1> Hello, {name}!</h1>
```

- Self closing tags like ``
- JSX use `className` instead of `class`
- Inline CSS styles are written in Javascript objects:

```
<div style = {{color: 'red', fontSize: '20px'}}>  
  Styled  
</div>
```

- We can use React components like HTML tags:

```
function Welcome (props){  
  return <h1> Hello, {props.name} </h1>  
}
```

How JSX Works:

- Welcome is a functional component that accepts props as its parameter
- It returns JSX that renders an `<h1>` heading
- The component uses the name props inside curly braces `{}` to display dynamic content.

Using the Component:

```
const element = <Welcome name="John Doe" />;
```

What Happens When This Renders:

- React creates an instance of the Welcome component
- It passes the props object `{ name: 'John Doe' }` to the component
- The component returns: `<h1>Hello, John Doe</h1>`
- This JSX gets converted to actual DOM elements

React converts this JSX to something like:


```
{type: Welcome,
  props: {
    name: 'John Doe'}}}
```

Why Use JSX

- It's easier to visualize UI structure.
- It's optimized to React.

Babel compiles JSX into `React.createElement()` calls:

JSX:

```
<div id='app'>
  Hello World!
</div>
```

JavaScript:

```
React.createElement("div", {id : "app"}, "Hello World")
```

Babel is a JavaScript transpiler. Transpiles modern JavaScript (ES6+) into backward-compatible versions (ES5)

Converts newer JS syntax into older syntax that can run in older browsers. (Different developers writting different versions of JavaScript, henceforth Babel is used)

Example: Arrow functions `() => {}` → `function() {}`

☒ Installing React Boilerplate through `npm`

What's NPM?

NPM is a Node Package Manager. It's the default package manager for the Node.js runtime environment. `npm` dosen't have a full form. Here's a quick overview:

- ◊ NPM is a tool to manage JavaScript packages/libraries.
 - Used to install, share, and version-control packages.
 - Comes pre-installed with Node.js.
 - It's a package manager but it's abbreviation is not node package manager.

It comes bundled with Node.js and is the default tool for managing dependencies in JavaScript/Node.js projects.

- World's largest software registry and package manager for JavaScript. A standard repository for all the packages.

`npm` Commands Cheat Sheet

Essential npm commands for JavaScript/React development.

Package Installation

Command	Description
<code>npm install</code>	Install all dependencies from <code>package.json</code>
<code>npm install <package></code>	Install a specific package (e.g., <code>npm install react</code>)
<code>npm install <package> --save-dev</code>	Install as dev dependency
<code>npm install -g <package></code>	Install globally
<code>npm install <package>@<version></code>	Install specific version

Development & Scripts

Command	Description
<code>npm start</code>	Start development server
<code>npm run build</code>	Create production build
<code>npm test</code>	Run tests
<code>npm run <script></code>	Run custom script
<code>npm run dev</code>	Start dev mode (Vite/Next.js)

Project Setup

Command	Description
<code>npm init</code>	Create new <code>package.json</code>
<code>npm init -y</code>	Quick init with defaults
<code>npm init <initializer></code>	Scaffold project (e.g., <code>npm create vite@latest</code>)

- npm uses `package.json` & `node_modules`
- All the configuration will be inside `package.json` file for the project which is a configuration for npm.
- Our projects are dependent on a lot of packages. These packages are called dependencies to the project. npm manages these packages.

Bundler

A bundler helps you to clean your code prior sending to production. It bundles/packs your app so it can be sent to production.

Example : `webpack`, `vite`, `parcel` etc.

-> `create-React-app` uses babel & webpack.

◊ Why Do We Need Bundlers?

- Dependency Management: It combines imported modules into a single file.
- Resolves `import/require` statements.
- Code Optimization: Minifies, compresses, and tree-shakes (removes unused code).
- Improves load time.
- Supports TypeScript, SCSS, and modern JS (ES6+).

◊ Bundler vs. Compiler vs. Transpiler

Tool	Purpose
Bundler	Combines files, manages dependencies
Compiler	Converts code to machine code (e.g., TypeScript → JS)
Transpiler	Converts modern JS to older syntax (e.g., Babel)

◊ When to Use a Bundler?

- SPAs (React, Vue, Angular)
- Libraries (Publishing to npm)
- Legacy Browser Support (via polyfills)

◊ Popular JavaScript Bundlers

Bundler	Key Features	Used With
Webpack	Highly configurable, plugin system	React, Vue, Angular
Vite	Ultra-fast (ESM + Rollup), HMR	Next.js, Svelte
Parcel	Zero-config, fast builds	Small to medium apps

Dependencies

Dependencies are external packages/libraries that your project relies on to function properly.

Types of Dependencies

1. Regular Dependencies (dependencies)

Packages required for your application to run in production

- Installed with `npm install <package>`
- Listed in `package.json` under `dependencies`

Example: json

```
"dependencies": {
  "react": "^18.2.0",
  "express": "^4.18.2"}
```

2. **Development Dependencies (devDependencies)**

Packages only needed during development (testing, building, etc.)

- Installed with `npm install <package> --save-dev`
- Not included in production builds

Example: json

```
"devDependencies": {  
  "jest": "^29.5.0",  
  "webpack": "^5.76.0"}
```

npm uses Semantic Versioning (SemVer):

- `^1.2.3`: [Caret] Allow patch and minor updates (1.x.x) [Safe]
- `~1.2.3`: [Tilde] Allow only patch updates (1.2.x)
- `1.2.3`: Exact version
- `*`: Latest version (not recommended)

Example: `npm install -D parcel` [Dev dependency installed of parcel bundler]