
□ UNIT 3 – React.js and Node.js

---

1. Introduction to React.js

📌 What is React.js?

React.js is an **open-source JavaScript library** developed by **Facebook (Meta)**.
It is used for **building interactive user interfaces (UIs)** and **single-page applications (SPAs)** where the page does not reload after every change.

React focuses only on the **view layer (V)** of the **MVC architecture**.
It uses **components** to build reusable and maintainable UI parts.

□ Why React.js?

Before React, updating data dynamically in web apps was slow because it involved changing the **real DOM** frequently.
React introduced a **Virtual DOM**, which updates efficiently and improves performance.

React allows developers to:

- Build dynamic web apps faster.
- Reuse code with components.
- Write cleaner and maintainable UI logic.

□ Main Features of React.js

1. **Component-Based Architecture**
   - Applications are divided into small, independent pieces called **components**.
   - Each component has its own logic, data (state), and UI structure.
   - Components can be reused throughout the application.
2. **Virtual DOM**
   - A lightweight copy of the real DOM.
   - When data changes, React updates the virtual DOM first, compares it with the previous version (Diffing), and updates only the changed parts in the real DOM (Reconciliation).
   - This improves app speed and efficiency.
3. **JSX (JavaScript XML)**
   - JSX allows writing HTML-like syntax inside JavaScript code.
   - It makes code easy to understand and debug.

    - Example:
    - `const element = <h1>Hello, React!</h1>;`
4. **Unidirectional Data Flow**
    - Data flows only in one direction — from **parent to child** through **props**.
    - This ensures predictable data handling and easy debugging.
5. **Declarative UI**
    - Developers describe what the UI should look like, and React handles the rendering automatically.
6. **Performance and Reusability**
    - React's Virtual DOM and component-based approach make web apps fast and scalable.

🧠 Core Concepts of React

Concept	Description
Component	Reusable block of UI that represents part of the screen.
Props (Properties)	Data passed from parent to child component. Immutable.
State	Local data managed by a component, can change over time.
Hooks	Special functions like `useState()` and `useEffect()` to use state and lifecycle features in functional components.
Lifecycle Methods	Functions that execute at specific points of a component's existence (e.g., mounting, updating, unmounting).

💻 Example: Basic React Component

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

export default Welcome;
```

To render this in your app:

```
<Welcome name="Student" />
```

🏗 React Architecture

```
Components (UI Elements)
      ↓
State / Props
      ↓
Virtual DOM
      ↓
Reconciliation
      ↓
Real DOM (Browser)
```

🌟 Advantages of React.js

- High performance due to Virtual DOM.
- Easy to learn and integrate.
- Component reusability saves development time.
- Strong community support and open-source libraries.
- Can also build mobile apps using **React Native**.

⚠️ Limitations

- React is a library, not a full framework — needs external tools (like Redux or Router).
- JSX has a learning curve.
- Frequent updates may break older code.

🌍 Popular Applications Built with React.js

- Facebook
- Instagram
- Netflix (frontend)
- Airbnb
- WhatsApp Web

---

2. Introduction to Node.js

📌 What is Node.js?

Node.js is an **open-source, cross-platform, server-side JavaScript runtime environment** built on **Google Chrome's V8 JavaScript engine**.
It allows JavaScript to run **outside the browser**, enabling developers to build the backend (server-side) using JavaScript.

⚙️ Key Features of Node.js

1. **Asynchronous & Event-Driven**
   - Node.js executes multiple operations simultaneously using callbacks or promises.
   - This makes it suitable for real-time applications.
2. **Non-blocking I/O model**

   - Node.js performs I/O operations (like reading files, database calls) asynchronously, allowing it to handle thousands of requests efficiently.
3. **Single-Threaded Architecture**
   - Uses a single thread with an event loop for handling concurrent clients.
   - Reduces overhead of thread management.
4. **Fast Performance**
   - Built on V8 engine, it compiles JavaScript directly to machine code.
5. **NPM (Node Package Manager)**
   - Comes bundled with Node.js.
   - Allows developers to install and use third-party packages easily.
6. **Cross-Platform**
   - Runs on Windows, macOS, and Linux.

□ Node.js Architecture

```
Client Requests
     ↓
Event Queue
     ↓
Event Loop (Single Thread)
     ↓
Worker Pool (Handles Async Tasks)
     ↓
Response to Client
```

- The **Event Loop** constantly checks for pending requests.
- Heavy operations like file access are handled by the **Worker Pool** asynchronously.
- This makes Node.js fast and non-blocking.

💡 Example: Simple HTTP Server

```
const http = require('http');

http.createServer((req, res) => {
  res.end('Hello from Node.js Server!');
}).listen(3000, () => console.log('Server running on port 3000'));
```

Output:
When you open `http://localhost:3000`, it displays:
☞ “Hello from Node.js Server!”

🔧 Modules in Node.js

Modules are reusable JavaScript files that can export functions or variables.

1. Core Modules

Provided by Node.js itself.
Examples: `http`, `fs`, `url`, `path`, `os`.

2. Local (Custom) Modules

Modules created by developers.

Example:

```
// math.js
exports.add = (a, b) => a + b;

// app.js
const math = require('./math');
console.log(math.add(5, 10));
```

3. Third-party Modules

Installed using **NPM**.

Example:

```
npm install express
```

□ Common Node.js Commands

Command	Description
`node filename.js`	Runs a Node program
`npm init`	Creates a package.json file
`npm install <pkg>`	Installs a package
`npm uninstall <pkg>`	Removes a package
`npm start`	Runs start script

□ Advantages of Node.js

- Uses JavaScript for both client and server.
- Scalable and high-performance.
- Excellent for real-time web apps.
- Huge NPM ecosystem.

□ Limitations

- Not suitable for CPU-intensive applications.

- Callback nesting (callback hell) can make code complex.
- Requires good understanding of asynchronous programming.

🌍 Popular Apps Built Using Node.js

- Netflix
- LinkedIn
- PayPal
- Uber
- eBay

---

3. Difference Between React.js and Node.js

Feature	React.js	Node.js
Type	Front-end Library	Back-end Runtime Environment
Developed By	Facebook (Meta)	Ryan Dahl
Usage	Building UI	Building server-side apps
Language	JavaScript + JSX	JavaScript
Platform	Browser	Server
DOM	Uses Virtual DOM	Doesn’t use DOM
Example	Facebook UI	Netflix Backend

---

4. Summary

- **React.js** is used for **front-end (UI)** development.
- **Node.js** is used for **back-end (server)** development.
- Together, they allow **Full Stack JavaScript** development.
- Both are efficient, fast, and highly scalable for modern web applications.