Day 7

React Js

PART 1 — React Installation & Commands

1️⃣ What You Need Before Installing React

React requires Node.js and npm (Node Package Manager).

✓ Why Node.js?

Helps run JavaScript outside the browser

Required for React development tools

Comes with npm, which installs React packages

✓ Check if Node.js is installed

Open terminal / CMD:

node -v

npm -v

If version numbers appear → Node is installed.

If not → download from the official site.

2️⃣ Install React Using Create React App (CRA)

(Best for beginners)

✓ Step 1: Create React App

npx create-react-app my-app

Explanation:

npx → executes packages

create-react-app → tool to generate a full React project

my-app → your project folder name

React automatically installs:

✓ Webpack

✓ Babel

✓ Development server

✓ React files

✓ Step 2: Move to project folder

cd my-app

✓ Step 3: Start the app

npm start

Result:

App runs on: http://localhost:3000

Auto refresh when code changes

3️⃣ Install React Using Vite

(Very fast, modern alternative)

✓ Step 1: Create Vite Project

npm create vite@latest my-vite-app

Select:

Framework → React

Variant → JavaScript

✓ Step 2: Move into folder

cd my-vite-app

✓ Step 3: Install packages

npm install

✓ Step 4: Start project

npm run dev

Runs on: http://localhost:5173

4️⃣ Important npm Commands

Command	Purpose
npm start	Run React app (CRA)
npm run dev	Run React app (Vite)

npm install package Install package

npm uninstall package Remove package

npm run build Prepare production build

5️⃣ React Project Folder Structure Explained

my-app/

|

├── node_modules/ → All installed libraries

├── public/ → index.html + images

├── src/ → React code

| ├── App.js → Main component

| ├── index.js → Entry point

| └── styles.css → CSS

|

├── package.json → Project configuration

└── README.md → Info & scripts

🌟 PART 2 — React.js Fundamentals (In-Depth + Examples)

1️⃣ React Introduction

💡 **What is React?**

React is a **JavaScript library** used to build **user interfaces**.
This means we use React to create the parts of a website a user can see and interact with.

💡 **Why React?**

- It makes building apps **faster**
- You can create **reusable components**
- It updates only the changed part of the page → **super fast**
- It's widely used in the industry

🟦 **Simple Example:**

Without React → You manually update the DOM (HTML).
With React → React handles updates for you.

2️⃣ Component Architecture

💡 What is a Component?

A **component** is like a LEGO block.
You build small pieces, then combine them to make a full application.

Examples of components:

- Button
- Navbar
- Footer
- Card
- Login form

💡 Why use Components?

- Reusable
- Easy to maintain
- Clear structure

🟦 Example: Simple Component

```
function Hello() {
  return <h1>Hello Students!</h1>;
}
```

To use it:

```
<Hello />
```

3️⃣ JSX (JavaScript XML)

💡 What is JSX?

JSX lets us write **HTML inside JavaScript**.
React uses JSX to make UI code easier and more readable.

✓ Features:

- Looks like HTML
- Can use JavaScript inside { }

🟦 **Example:**

```
const name = "Aman";


function Welcome() {
  return <h2>Welcome {name} to React!</h2>;
}
```

JSX lets you mix UI + logic easily.

🔌 Props & State

📦 PROPS (Properties)

Props = **Data passed from parent to child**
Props are **read-only** (child can't change them).

🟦 **Example:**

Parent Component:

```
<Greeting name="Riya" />
```

Child Component:

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

Output:

Hello Riya

🔄 STATE

State = **data that belongs to a component**
State can **change**, and when it changes, the UI updates automatically.

Example:

Counter, Form inputs, API data

🟦 **Example:**

```
const [count, setCount] = useState(0);
```

5️⃣ **useState & useEffect Hooks**

🟢 **useState – For Managing State**

🟦 **Example: Counter App**

```
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <>
      <h1>Count: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </>
  );
}
```

What happens?

- Initial value = 0
- Button click → state updates
- UI refreshes automatically

🔵 **useEffect – For Side Effects**

useEffect runs code at specific times, like:

- When the component loads
- When data changes

- When you fetch API data

🟦 **Example: Runs once when page loads**

```
useEffect(() => {
  console.log("Component Loaded");
}, []);
```

If the array is empty → run only once.

6️⃣ **React Router (Navigation Between Pages)**

React Router helps create **multiple pages** without refreshing the browser.
Example:
Home Page, About Page, Contact Page

💡 **Steps:**

1. Install react-router
2. Wrap app in <BrowserRouter>
3. Add routes

🟦 **Example:**

```
import { BrowserRouter, Routes, Route } from "react-router-dom";


function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </BrowserRouter>
  );
}
```

Now:

- / → Home page
- /contact → Contact page

7️⃣ API Fetching with React

We use fetch() or axios to get data from an API.

API = data provider
React = shows data on UI

🟦 Example: Fetch Users from API

```
import { useState, useEffect } from "react";


function Users() {
 const [users, setUsers] = useState([]);


 useEffect(() => {
  fetch("https://jsonplaceholder.typicode.com/users")
   .then(res => res.json())
   .then(data => setUsers(data));
 }, []);


 return (
  <>
   <h2>User List</h2>
   {users.map(user => (
    <p key={user.id}>{user.name}</p>
   ))}
  </>
 );
}
```

What happens?

1. Page loads → useEffect runs
2. API is called
3. Data is stored in state
4. UI updates and shows users