DETAILED SYLLABUS:

Sr. No.	Module	Detailed Content	Hours	LO Mapping
I	HTML5	Elements, Attributes, Head, Body, Hyperlink, Formatting, Images, Tables, List, Frames, Forms, Multimedia	**02**	LO1
II	CSS3	Syntax, Inclusion, Color, Background, Fonts, Tables, lists,CSS3 selectors, Pseudo classes, Pseudo elements	**02**	LO2
III	Bootstrap	Grid system, Forms, Button, Navbar, Breadcrumb, Jumbotron	**02**	LO3
IV	JavaScript	Variables, Operators, Conditions, Loops, Functions, Events, Classes and Objects, Error handling, Validations, Arrays, String, Date	**05**	LO4
V	React	Installation and Configuration. JSX, Components, Props, State, Forms, Events, Routers, Refs, Keys.	**08**	LO5
VI	Node.js	Installation and Configuration, Callbacks, Event loops, Creating express app.	**07**	LO6

🧩 Module I – HTML5 (LO1)

Duration: 2 hours

🔹 Basics:

HTML (HyperText Markup Language) is the **structure** of a web page.
Files have `.html` extension.

Tag	Purpose
`<html>`	Root element of the page
`<head>`	Contains meta info, title, styles, links
`<body>`	Contains visible content
`<h1>` to `<h6>`	Headings

`<p>`	Paragraph
`<a href="">`	Hyperlink
`<img src="">`	Image
`<ul>`, `<ol>`, `<li>`	Lists
`<table>`, `<tr>`, `<td>`	Tables
`<form>`	For user input (contains `<input>`, `<select>`, `<textarea>`, etc.)
`<audio>`, `<video>`	Multimedia support

Example:

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
</head>
<body>
  <h1>Welcome!</h1>
  <p>This is my first webpage.</p>
  <a href="https://example.com">Visit Example</a>
</body>
</html>
```

🎨 Module II – CSS3 (LO2)

Duration: 2 hours

CSS (Cascading Style Sheets) gives **style and layout** to webpages.

- 🔹 **Syntax:**

```
selector {
  property: value;
```

```
}
```

🔹 **Ways to Add CSS:**

1. **Inline:** `<p style="color:red;">Hello</p>`
2. **Internal:** inside `<style>` tag in `<head>`
3. **External:** link CSS file using `<link rel="stylesheet" href="style.css">`

🔹 **Key Topics:**

- **Colors:** `color`, `background-color`
- **Fonts:** `font-family`, `font-size`
- **Lists/Tables:** `list-style`, `border`, `padding`
- **Selectors:** `.class`, `#id`, `element`
- **Pseudo-classes:** `:hover`, `:focus`
- **Pseudo-elements:** `::before`, `::after`

Example:

```
body {
  background-color: lightblue;
}
h1:hover {
  color: red;
}
```

🧱 Module III – Bootstrap (LO3)

Duration: 2 hours

Bootstrap is a **CSS framework** for responsive design.
Include via CDN:

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.
min.css" rel="stylesheet">
```

🔹 Main Concepts:

- **Grid System:** Uses rows and columns (12-column layout)
- **Forms:** Pre-styled form elements
- **Buttons:** `btn btn-primary`, `btn btn-success`
- **Navbar:** Responsive navigation bars
- **Breadcrumb:** Navigation path
- **Jumbotron:** Highlight large headers or info blocks

Example:

```
<div class="container">
  <div class="row">
    <div class="col-md-6 bg-light p-3">Left</div>
    <div class="col-md-6 bg-dark text-white p-3">Right</div>
  </div>
</div>
```

💻 Module IV – JavaScript (LO4)

Duration: 5 hours

JavaScript adds **interactivity** and logic to web pages.

🔹 Basics:

- **Variables:** `var`, `let`, `const`
- **Operators:** `+`, `-`, `*`, `/`, `==`, `===`
- **Conditions:** `if`, `else if`, `switch`
- **Loops:** `for`, `while`, `do-while`
- **Functions:** `function add(a,b){return a+b;}`
- **Events:** `onclick`, `onchange`
- **Objects & Classes:** Encapsulate data and behavior
- **Error Handling:** `try...catch`
- **Validation:** Check form inputs
- **Arrays, Strings, Dates:** Common data structures

Example:

```
<button onclick="showMessage()">Click Me</button>
<script>
  function showMessage(){
    alert("Hello JavaScript!");
  }
</script>
```

⚛️ Module V – React (LO5)

Duration: 8 hours

React is a **JavaScript library for building UI components**.

🔹 Setup:

- Install Node.js first

Create app:

```
npx create-react-app myapp
cd myapp
npm start
```

-

🔹 Core Concepts:

- **JSX:** HTML-like syntax in JavaScript
- **Components:** Building blocks of React
- **Props:** Data passed between components
- **State:** Stores component data
- **Forms & Events:** Handle user input
- **Router:** Navigate between pages
- **Refs:** Access DOM elements directly
- **Keys:** Unique IDs in lists

Example Component:

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

🟢 Module VI – Node.js (LO6)

Duration: 3 hours

Node.js allows **JavaScript to run on the server**.

🔹 Installation & Configuration:

- Install from nodejs.org

Check version:

```
node -v
npm -v
```

-

🔹 Core Concepts:

- **Callbacks:** Functions called after task completion
- **Event Loop:** Handles asynchronous operations
- **Express.js:** Framework to build web servers easily

Example Express App:

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello from Node.js!');
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

🧠🔥 Important JavaScript Topics (Module IV)

1️⃣ Basics of JavaScript

- **What it is:** A scripting language used to make web pages dynamic.
- **Runs in:** Browser (client-side) or on server (Node.js).
- **Adding JS:**
  - Inline → `<button onclick="msg()">Click</button>`
  - Internal → `<script> ... </script>`
  - External → `<script src="script.js"></script>`

2️⃣ Variables

- Used to store data values.
- **Keywords:**
  - `var` → function-scoped, old method
  - `let` → block-scoped (preferred)
  - `const` → constant value

Example:

```
let name = "Vaishnavi";
const PI = 3.14;
```

-

3️⃣ Data Types

- **Primitive:** String, Number, Boolean, Undefined, Null, Symbol, BigInt
- **Non-Primitive:** Object, Array, Function

Example:

```
let str = "Hello";
let num = 25;
let isOn = true;
```

-

4️⃣ Operators

- **Arithmetic:** `+ - * / % ++ --`
- **Assignment:** `= += -= *=`
- **Comparison:** `== != === > < >= <=`
- **Logical:** `&& || !`
- **Ternary:** `condition ? value1 : value2`

Example:

```
let result = (marks >= 40) ? "Pass" : "Fail";
```

-

5️⃣ Conditional Statements

Used for decision making.

```
if(condition) {
  // code
} else if(condition2) {
```

```
  // code
} else {
  // code
}
```

Also: `switch` statement for multiple conditions.

6️⃣ Loops

Used for repeating tasks.

- `for`, `while`, `do...while`

Example:

```
for(let i=1; i<=5; i++){
  console.log(i);
}
```

-

7️⃣ Functions

Reusable block of code.

```
function add(a, b) {
  return a + b;
}
let result = add(2, 3);
```

Arrow function:

```
const square = n => n * n;
```

-

8️⃣ Events

Used to perform actions based on user interaction.

Event	Trigger
`onclick`	Mouse click
`onchange`	Form field change
`onmouseover`	Mouse hover
`onkeyup`	Key press release

Example:

```
<button onclick="greet()">Click me</button>
<script>
  function greet(){ alert("Hello!"); }
</script>
```

9️⃣ Arrays

Used to store multiple values in one variable.

```
let fruits = ["Apple", "Mango", "Banana"];
console.log(fruits[1]); // Mango
```

Common Methods:
`push()`, `pop()`, `shift()`, `unshift()`, `length`, `sort()`, `forEach()`

🔟 Strings

```
let msg = "Hello World";
console.log(msg.length);
console.log(msg.toUpperCase());
console.log(msg.substring(0,5));
```

1️⃣1️⃣ Date Object

```
let today = new Date();
console.log(today.getFullYear());
console.log(today.toDateString());
```

1️⃣2️⃣ Objects

Used to store key-value pairs.

```
let student = { name: "Amit", age: 20 };
console.log(student.name);
```

1️⃣3️⃣ Classes and Objects (OOP in JS)

```
class Car {
  constructor(name, year){
    this.name = name;
    this.year = year;
  }
  start(){
    return this.name + " started";
  }
}
let myCar = new Car("BMW", 2022);
console.log(myCar.start());
```

1️⃣4️⃣ Error Handling

Use `try...catch` to handle runtime errors.

```
try {
  let x = y + 10;
} catch(err) {
  console.log("Error: " + err.message);
```

```
}
```

15️⃣ Form Validation

Used to check if user inputs are correct.

```
function validate() {
  let name = document.getElementById("uname").value;
  if(name == "") {
    alert("Name is required!");
    return false;
  }
}
```

16️⃣ DOM Manipulation (Extra but Important)

JS can access and change HTML elements.

```
document.getElementById("demo").innerHTML = "Updated!";
```

Important javascript topics

⚡1️⃣Arrow Functions

🔹 What it is:

A **shorter way to write functions** introduced in **ES6 (modern JavaScript)**.
They make code cleaner and automatically handle the `this` keyword differently (important in React).

🔹 Syntax:

```
// Normal function
function add(a, b) {
  return a + b;
}

// Arrow function
const add = (a, b) => a + b;
```

🔹 Key Points:

- No need for the `function` keyword.
- If there's only **one line**, `return` is **implicit**.
- If there's only **one parameter**, parentheses are optional.

🔹 Examples:

```
// Multiple parameters
const multiply = (x, y) => x * y;

// Single parameter
const square = n => n * n;

// Multiple statements need {}
const greet = name => {
```

```
  console.log("Hello " + name);
  console.log("Welcome!");
}
```

🔹 Why it's useful:

- Shorter syntax
- Common in React components and callbacks
- Better for inline functions like event handlers

🌐2️⃣ DOM (Document Object Model)

🔹 What it is:

DOM is the **structure of a webpage** represented as a **tree of objects**.
It allows JavaScript to **access and change HTML and CSS dynamically**.

Think of the DOM like a **bridge between HTML and JavaScript**.

Example HTML:

```
<h1 id="title">Hello World</h1>
<button onclick="changeText()">Click me</button>
```

🔹 Accessing Elements:

```
// Select elements
document.getElementById("title");
document.getElementsByClassName("classname");
document.getElementsByTagName("p");
document.querySelector("#title");
document.querySelectorAll("p");
```

🔹 Changing Content and Styles:

```
// Change text
```

```
document.getElementById("title").innerHTML = "Welcome to JavaScript!";

// Change style
document.getElementById("title").style.color = "blue";
```

🔹 Creating and Adding Elements:

```
let newPara = document.createElement("p");
newPara.innerText = "This is a new paragraph.";
document.body.appendChild(newPara);
```

🔹 Event Handling with DOM:

```
<button id="btn">Click</button>
<p id="msg"></p>

<script>
document.getElementById("btn").addEventListener("click", function() {
  document.getElementById("msg").innerText = "Button was clicked!";
});
</script>
```

⚙️3️⃣ Callback Functions

A **function passed as an argument** to another function.
Used in asynchronous operations like event handling, API calls, etc.

```
function greet(name, callback) {
  console.log("Hi " + name);
  callback();
}

function done() {
  console.log("Greeting complete!");
}
```

```
greet("Vaishnavi", done);
```

🔄4️⃣ Promises

Used to handle **asynchronous operations** (like data fetching) in a clean way.

```
let promise = new Promise((resolve, reject) => {
  let success = true;
  if(success) resolve("Data loaded");
  else reject("Error loading data");
});

promise.then(result => console.log(result))
       .catch(error => console.log(error));
```

⏳5️⃣ Async / Await

A simpler way to use Promises.

```
async function fetchData() {
  try {
    let response = await fetch("https://api.example.com/data");
    let data = await response.json();
    console.log(data);
  } catch(error) {
    console.log("Error:", error);
  }
}
```

🧩 1. React Installation & Setup

You can create a React project using:

```
npx create-react-app myapp
cd myapp
npm start
```

✅ This sets up React, React-DOM, and Webpack automatically.
The app runs at http://localhost:3000

⚛️ 2. JSX (JavaScript XML)

JSX lets you write HTML inside JavaScript.

```
function App() {
  return <h2>Hello, React JSX!</h2>;
}
export default App;
```

✅ JSX is compiled into JavaScript using Babel.
✅ Must have **one root element** (e.g., a `<div>` or `<>` fragment).

🧱 3. Components

Components are reusable building blocks in React.

Functional Component:

```
function Welcome() {
  return <h3>Welcome Component</h3>;
}
export default Welcome;
```

✅ Components return UI and can be reused multiple times.

📦 4. Props (Properties)

Props are used to **pass data** from parent to child components.

```
function Greeting(props) {
  return <h4>Hello, {props.name}</h4>;
}

function App() {
  return <Greeting name="Vaishnavi" />;
}
```

✅ Props are **read-only** values.
✅ Used like function parameters.

🔄 5. State

State stores **dynamic data** inside a component.

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

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

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

✅ `useState()` returns a variable and a function to update it.
✅ When state changes, UI automatically re-renders.

⚙️ 6. Event Handling

React uses camelCase event names.

```
function ButtonClick() {
  const handleClick = () => alert("Button Clicked!");
  return <button onClick={handleClick}>Click Me</button>;
}
```

✅ Use functions instead of inline JavaScript.
✅ Don't use `onclick` (use `onClick`).

🌐 7. Conditional Rendering

Render elements based on conditions.

```
function Login({ isLoggedIn }) {
  return (
    <div>
      {isLoggedIn ? <p>Welcome back!</p> : <p>Please log in.</p>}
    </div>
  );
}
```

✅ Works like `if-else` in JSX using the ternary operator.

🔁 8. Lists and Keys

Render multiple items using `.map()`.

```
function ListExample() {
  const items = ["Apple", "Banana", "Cherry"];
  return (
```

```
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
}
```

✅ Each element in a list needs a unique **key** prop.

🪝 9. useEffect Hook

Used to perform **side effects** (like fetching data or timers).

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

function DemoEffect() {
  useEffect(() => {
    console.log("Component Mounted");
  }, []);
  return <p>Check the console log!</p>;
}
```

✅ The empty `[]` dependency runs it only once (on mount).

🎯 10. useRef Hook

Used to access **DOM elements** or store mutable values.

```
import { useRef } from "react";

function FocusInput() {
  const inputRef = useRef(null);
  return (
    <div>
```

```
      <input ref={inputRef} type="text" />
      <button onClick={() => inputRef.current.focus()}>Focus</button>
    </div>
  );
}
```

✅ `useRef` doesn't trigger re-renders like state.

🧭 11. React Router (Single Page App)

Used for navigation without reloading the page.

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

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> | <Link to="/about">About</Link>
      </nav>

      <Routes>
        <Route path="/" element={<h3>Home Page</h3>} />
        <Route path="/about" element={<h3>About Page</h3>} />
      </Routes>
    </BrowserRouter>
  );
}
```

✅ Enables multiple pages in a **Single Page Application (SPA)**.

🧠 12. Forms in React

React uses **controlled components** for forms.

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

function FormExample() {
  const [name, setName] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Hello, ${name}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={name} onChange={(e) =>
setName(e.target.value)} />
      <button type="submit">Submit</button>
    </form>
  );
}
```

✅ React handles form input values using **state**.

🧩 1. Node.js Environment Setup

✅ **Install Node.js**
Download from https://nodejs.org
After installation, check:

```
node -v
npm -v
```

✅ **Initialize a project**

```
mkdir node-demo
cd node-demo
npm init -y
```

This creates a `package.json` file for dependencies.

⚙️ 2. Creating a Simple Node.js Server

```
// index.js
const http = require("http");

const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello from Node.js Server!");
});

server.listen(3000, () => console.log("Server running on port 3000"));
```

✅ Run: `node index.js`
✅ Visit: http://localhost:3000

🚀 3. Express.js Introduction

Express simplifies handling routes and middleware.

Install Express:

```
npm install express
```

Example:

```
// app.js
const express = require("express");
const app = express();

app.get("/", (req, res) => {
  res.send("Hello from Express!");
});

app.listen(3000, () => console.log("Server running on port 3000"));
```

✅ Visit http://localhost:3000

🧭 4. Routing in Express

Different routes for different pages:

```
app.get("/", (req, res) => res.send("Home Page"));
app.get("/about", (req, res) => res.send("About Page"));
app.get("/contact", (req, res) => res.send("Contact Page"));
```

✅ Each route responds to a specific URL path.

💾 5. Handling GET and POST Requests

```
app.use(express.urlencoded({ extended: true })); // Parse form data

app.get("/form", (req, res) => {
  res.send(`<form method="POST" action="/submit">
              <input name="name" />
```

```
            <button>Submit</button>
          </form>`);
});

app.post("/submit", (req, res) => {
  res.send(`Hello ${req.body.name}`);
});
```

✅ Demonstrates form handling with GET/POST methods.

🧱 6. Middleware in Express

Middleware runs **before route handling**.

```
app.use((req, res, next) => {
  console.log(`Request URL: ${req.url}`);
  next(); // Move to next middleware or route
});
```

✅ Commonly used for logging, authentication, or validation.

📂 7. Serving Static Files

Used to serve images, CSS, JS files.

```
app.use(express.static("public"));
```

✅ Place your files (e.g., `index.html`, `style.css`) inside `/public` folder.
✅ Access via: `http://localhost:3000/index.html`

🧩 8. Template Engine (EJS)

Install:

```
npm install ejs
```

Example:

```
app.set("view engine", "ejs");

app.get("/", (req, res) => {
  res.render("index", { title: "EJS Example" });
});
```

views/index.ejs

```
<h2><%= title %></h2>
<p>This is rendered using EJS Template.</p>
```

✅ EJS allows embedding JS variables in HTML.

🌐 9. Using Fetch / API Requests

```
app.get("/data", (req, res) => {
  res.json({ name: "Vaishnavi", course: "Web Dev" });
});
```

✅ Returns JSON data — useful for connecting backend & frontend.

🧠 10. Asynchronous Programming (Promises / Async–Await)

```
function getData() {
  return new Promise(resolve => {
    setTimeout(() => resolve("Data Loaded"), 1000);
  });
}
```

```
async function showData() {
  const result = await getData();
  console.log(result);
}

showData();
```

✅ Demonstrates handling async code cleanly with `async/await`.

🧰 11. Express Router (Modular Routing)

```
// routes/user.js
const express = require("express");
const router = express.Router();

router.get("/", (req, res) => res.send("User Home"));
router.get("/profile", (req, res) => res.send("User Profile"));

module.exports = router;

// app.js
const userRoutes = require("./routes/user");
app.use("/user", userRoutes);
```

✅ Helps organize routes in separate files.

🔒 12. Basic Error Handling

```
app.use((req, res) => {
  res.status(404).send("404 - Page Not Found");
});
```

✅ Catches any route not defined in your app.

👩‍💻 13. Connecting Node + Database (Concept)

For MongoDB:

```
npm install mongoose
```

Example (concept only):

```
const mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/testdb");
```

✅ Allows CRUD operations from Node to database.