S.No	Practicals / Topics	hrs
Lab No	**Advanced JavaScript**	4
1	**Fundamentals of JavaScript for MERN** **Topics to Cover** ● Variables and data types (let, const, var). ● this keyword and Scope ● Functions: declaration, expression, and arrow functions. ● Arrays: Methods like map(), filter(), find(). ● Objects: accessing and manipulating objects. ● ES6+ Features: Destructuring and template literals. **Project: Movie Collection Manager** **Objective**: Work with arrays and objects to manage a collection of movies dynamically. **Tasks:** ● Create an array of movie objects, where each object has fields like title, genre, rating, and release year. ● Write functions to: ● Add a new movie to the collection. ● List all movies in a specific genre. ● Find the highest-rated movie in the collection. ● Use map() to create a list of all movie titles. ● Use filter() to find movies released after a specific year. ● Log the results to the console using template literals.	
2	**Advanced JavaScript Concepts for MERN** **Topics to Cover** ● Asynchronous programming: Promises and async/await. ● Error handling (try...catch). ● JavaScript modules (export, import). ● Using JavaScript Date and Math objects. **Project: Task Reminder System** **Objective:** Build a task reminder system that schedules reminders and handles asynchronous delays. **Tasks** ● Create an array of task objects, where each task has fields like title, dueTime (in minutes from now), and priority. ● Write functions to: ○ Add a new task to the list. ○ Sort tasks by their priority.	

	○ Display tasks due within a certain timeframe (e.g., next 10 minutes). ○ Simulate sending reminders using setTimeout() based on the task's dueTime. ● Use try...catch to handle errors, such as invalid task data or missing fields. ● Export and import the task management functions using Jav aScript modules.	
	Node.js	10
3	**Introduction to Node.js and Basic Module** **Topics to Cover:** ● What is Node.js? ● Installing Node.js. ● Using the Node.js REPL. ● Core modules (fs, path, os). ● Writing and running a simple script. **Project: *System Utility Dashboard*** ● **Objective:** Create a command-line tool that provides system information. ● **Tasks:** 1. Use the os module to display the OS type, total memory, free memory, and CPU details. 2. Use the fs module to save the system information to a log file. 3. Use the path module to ensure the file is saved in a standardized format (logs/system-info.txt).	
4	**Working with Custom Modules and HTTP** **Topics to Cover:** ● Custom modules: require and module.exports. ● Using the HTTP module to create a server. ● Handling different URL paths manually. **Project: Basic Static File Server** **Objective:** Build an HTTP server to serve static files like HTML, CSS, and images. **Tasks:**	

	● Create a simple HTML page with a welcome message and a few images. ● Use the http and fs modules to serve these files when the browser requests them. ● Add logic to handle 404 errors when a file is not found.	
5	**Asynchronous Programming and File System Operations** **Topics to Cover:** ● Synchronous vs. asynchronous operations in Node.js. ● Callbacks, promises, and async/await. ● Advanced file system operations (read, write, delete files asynchronously). **Project: File Organizer Tool** **Objective:** Build a CLI tool to organize files into folders based on their type (e.g., images, documents, videos). **Tasks:** ● Accept a directory path as an input from the user. ● Use the fs module to read all files in the directory. ● Move files into folders like Images, Documents, and Others based on their extensions. ● Log the operations performed into a summary.txt file.	
6	**Building a Simple RESTful API with Core HTTP Module** **Topics to Cover:** ● What is REST? ● Creating a RESTful API with Node.js core HTTP module. ● Handling JSON data in requests and responses. **Project: *User Management System*** **Objective:** Create a RESTful API to manage user data without using Express.js. **Tasks:** ● Implement the following endpoints using the http module: ● GET /users: Return a list of all users stored in a JSON file. ● POST /users: Accept new user data in the request body and add it to the JSON file. ● DELETE /users/:id: Remove a user by their ID from the JSON file. ● Use the fs module to store and retrieve user data persistently. ● Test the API using Postman or curl.	

	Express.js	8
7	**Introduction to Express.js** **Topics to Cover:** ● Setting up an Express project. ● Understanding the request-response cycle. ● Using middleware (express.json() and express.static()). **Project: Simple Content Delivery Server** **Objective:** Create a static file server with basic APIs for logging user visits. **Tasks:** ● Serve static HTML, CSS, and JS files from a public directory. ● Log every user visit (IP, time) to a visits.log file using middleware. ● Provide an API endpoint (GET /logs) to retrieve the log data as JSON.	
8	**Routing and Query Parameters** **Topics to Cover:** ● Defining routes for different HTTP methods (GET, POST, PUT, DELETE). ● Handling dynamic route parameters and query strings. ● Organizing routes using express.Router(). **Project: E-Commerce Product Catalog API** **Objective:** Build an API for managing an e-commerce product catalog. **Tasks:** ● GET /products: Return all products. ● GET /products/:id: Fetch a specific product by ID. ● GET /products?category=electronics: Filter products by category. ● Use route parameters and query strings effectively.	
9	**Middleware, Error Handling, Authentication, and Session Management in Express** **Topics to Cover:** ● Writing custom middleware functions. ● Implementing logging middleware for request tracking. ● Centralized error handling in Express. ● Adding authentication with JWT (JSON Web Tokens). ● Managing sessions using cookies or token-based authentication. **Project: Order Management System**	

	Objective: Create a secure API for managing orders with robust logging, error handling, authentication, and session management. **Tasks:** **1. CRUD Operations for Orders** ● Implement endpoints: ● POST /orders: Create a new order. ● GET /orders: Retrieve all orders. ● GET /orders/:id: Retrieve a specific order by ID. ● PUT /orders/:id: Update an order by ID. ● DELETE /orders/:id: Delete an order by ID. ● Use in-memory storage or a simple database (e.g., MongoDB with Mongoose) for storing orders. **2. Logging Middleware** ● Write a custom middleware function to log: ● HTTP method, URL, and timestamp of each API call. ● Save the logs to a file (server.log) using the fs module. **3. Centralized Error Handling** ● Implement a centralized error-handling middleware for: ● Invalid endpoints (404 errors). ● Missing or incorrect data in requests (400 errors). ● Server errors (500 errors). ● Use custom error classes for cleaner error handling. **4. Authentication with JWT** ● Add user authentication with login and registration endpoints: ● POST /auth/register: Register a new user. ● POST /auth/login: Authenticate a user and return a JWT token. ● Secure order endpoints so only authenticated users can access them. ● Implement middleware to validate JWT tokens in API requests. **5. Session Management** ● Use JWT tokens stored in HTTP-only cookies for session management. ● Set the cookie on login and remove it on logout: ● POST /auth/logout: Clear the authentication cookie. **6. Bonus Task: Role-Based Access Control** ● Assign roles (e.g., admin, user) during user registration. ● Restrict certain endpoints (e.g., DELETE /orders/:id) to admin users.	
10	**Building a RESTful API (Pracitce Work)** **Topics to Cover:** ● Building a complete CRUD API. ● Parsing JSON and URL-encoded data. ● Sending appropriate HTTP status codes. **Project: Personal Task Manager** **Objective:** Build a task management API for creating, updating, and managing tasks.	

	Tasks: ● CRUD operations for tasks (POST, GET, PUT, DELETE). ● Validate task input data (e.g., title and status fields) using middleware. ● Persist tasks in a tasks.json file.	
	MONGODB	6
11	**Introduction to MongoDB and Mongoose Integration** **Topics to Cover** ● What is MongoDB? NoSQL vs SQL databases ● Installing MongoDB locally and an introduction to MongoDB Atlas ● Setting up a Node.js project ● Installing and configuring Mongoose for MongoDB integration ● Understanding schemas and models in Mongoose **Project: User Profile Manager** **Objective:** Set up MongoDB and Mongoose in a Node.js application and create a basic schema to perform simple database operations. **Tasks** ● Install MongoDB locally or create a free cluster on MongoDB Atlas. ● Set up a new Node.js project and install the required dependencies (express, mongoose, dotenv). ● Create a .env file and store your MongoDB connection URI securely. ● Write a connection script in Node.js using Mongoose to connect to MongoDB. ● Define a User schema with fields like name, email, and age. ● Create a simple script to insert a user into the database and log the results in the console. ● Fetch all users from the database and display them in the console.	
12	**CRUD Operations with MongoDB** **Topics to Cover** ● Implementing Create, Read, Update, and Delete operations with Mongoose ● Introduction to RESTful API routes using Express ● Querying documents with filters and projections **Project: Task Manager API** **Objective:** Build a RESTful API with Express and Mongoose to manage	

	tasks in a MongoDB collection. **Tasks** ● Define a Task schema with fields like title, description, status (e.g., "Pending", "Completed"), and dueDate. ● Set up the following API endpoints: ● POST /tasks: Add a new task to the database. ● GET /tasks: Retrieve all tasks. ● GET /tasks/:id: Retrieve a specific task by ID. ● PUT /tasks/:id: Update task details by ID. ● DELETE /tasks/:id: Delete a task by ID. ● Test the API endpoints using Postman or Thunder Client. ● Use filters to query tasks based on their status or dueDate. ● Handle basic errors, such as invalid task IDs or missing fields.	
13	**Relationships and Advanced Features** **Topics to Cover** ● Creating relationships in MongoDB using Mongoose (ref and populate) ● Validating data with Mongoose validation rules ● Error handling in Mongoose and Express ● Modularizing routes and controllers **Project: Blog Application API** **Objective** Implement a relational data model to create and manage blog posts with associated authors using Mongoose and Express. **Tasks** ● Define schemas for Author and BlogPost: ● Author: name, email. ● BlogPost: title, content, createdAt, author (reference to Author). ● Set up the following API endpoints: ● POST /authors: Add a new author. ● POST /blogposts: Add a new blog post and associate it with an author. ● GET /blogposts: Retrieve all blog posts, including author details (use populate). ● GET /blogposts/:id: Retrieve a specific blog post by ID with author details. ● DELETE /blogposts/:id: Delete a blog post by ID. ● Add Mongoose validation to ensure all required fields are provided and correctly formatted (e.g., email validation). ● Handle errors gracefully, such as missing or invalid IDs and validation failures.	

	REACT.JS	12
14	**Introduction to React** **Topics to Cover** ● Introduction to React and its advantages ● Setting up a React environment using Vite ● Understanding JSX syntax ● Functional components and props **Project: Personal Profile Card** **Objective:** Learn the basics of React by building reusable components and passing data using props. **Tasks:** ● Set up a React project using Vite. ● Create a functional component for a profile card. ● Use props to display a user's name, photo, and a short bio. ● Style the card using basic CSS.	
15	**State Management with useState** **Topics to Cover** ● React state management with the useState hook ● Event handling basics in React **Project:** Simple Counter App **Objective:** Understand how to manage and update state dynamically in a React application. **Tasks** ● Create a functional component with a state variable for the counter. ● Add buttons to increment, decrement, and reset the counter. ● Display the counter value dynamically on the screen.	
16	**Lists and Conditional Rendering** **Topics to Cover** ● Rendering lists with .map() ● Keys in React lists ● Conditional rendering techniques **Project: To-Do List App** **Objective:** Learn how to dynamically render and manage a list of items with conditional rendering. **Tasks:**	

	● Create a state variable to store a list of tasks. ● Render the task list dynamically using .map(). ● Add functionality to add and remove tasks. ● Display a message when the task list is empty.	
17	**Forms and Controlled Components** **Topics to Cover** ● Handling user input in forms ● Controlled components ● Basic form validation **Project: Feedback Form** **Objective:** Understand how to handle and validate user input in a React form. **Tasks:** ● Create a form with input fields for name, email, and feedback message. ● Use state variables to control the form inputs. ● Validate the inputs (e.g., ensure fields are not empty). ● Display submitted feedback data below the form.	
18	**Context API for State Management** **Topics to Cover** ● Context API for state management ● Creating and using Context Providers and Consumers **Project:** Theme Switcher App **Objective:** Learn how to use the Context API to manage global state across components. **Tasks** ● Set up a Context Provider to manage theme state. ● Implement a toggle button to switch between light and dark themes. ● Use the theme context to dynamically update the app's background and text colors.	
19	**React Effects with useEffect** **Topics to Cover**	

	● Lifecycle methods in functional components ● Using the useEffect hook for side effects ● Fetching data from APIs **Project: Weather App** **Objective**: Understand how to use the useEffect hook to fetch and display data from an external API. **Tasks** ● Fetch weather data for a user-input city using an API (e.g., OpenWeatherMap). ● Display the city name, temperature, and weather conditions. ● Update the displayed data when the user searches for a new city.	
20	**Component Styling (Advance learner)** **Topics to Cover** ● Styling React components ● CSS Modules and inline styles ● Responsive design principles **Project: Responsive Product Card** **Objective**: Learn how to style React components using modern CSS techniques. **Tasks** ● Create a product card component with name, price, and description. ● Use CSS Modules to style the card. ● Add responsive styling to adjust layout for different screen sizes.	
21	**React Router** **Topics to Cover** ● Routing with react-router-dom v7 ● Creating multiple pages in a React app ● Navigation with Link and useNavigate **Project: Multi-Page Blog** **Objective**: Learn to build multi-page applications using React Router. **Tasks** ● Set up routes for a homepage, blog list, and individual blog post pages.	

	● Use React Router to navigate between pages. ● Display blog post details dynamically based on route parameters.	
22	**Advanced State Management with useReducer (advance learner)** **Topics to Cover** ● Managing complex state with the useReducer hook ● Understanding reducer functions and actions **Project**: **Expense Tracker App** **Objective**: Learn to manage complex state logic with the useReducer hook. **Tasks** ● Set up a reducer function to manage a list of expenses. ● Add functionality to add and delete expense items. ● Display a summary of total expenses dynamically.	
	TESTING	4
23	**Backend Testing** **Topics to Cover in Testing** **1. Introduction to Testing:** ● Importance of testing in software development. ● Types of testing: Unit testing, integration testing, end-to-end (E2E) testing. ● Overview of popular testing libraries: ○ Backend: Jest, Supertest (for APIs). ○ Frontend: React Testing Library. **2. Writing Unit Tests for Backend:** ● Setting up Jest for a Node.js project. ● Writing tests for utility functions. ● Mocking dependencies and data. **3. Testing APIs:** ● Using Supertest to test Express APIs. ● Writing tests for CRUD endpoints. ● Checking status codes and response payloads. **Project: Testing a Task Manager App**	

	Objective: Test the key features of a "Task Manager" app for managing tasks. **Backend Testing:** ● **Setup:** Use Jest and Supertest to test the Node.js backend API. ● **Endpoints to Test:** 1. `POST /tasks`: Test if a task is created with valid data. 2. `GET /tasks`: Test if all tasks are returned. 3. `PUT /tasks/:id`: Test if a task is updated correctly. 4. `DELETE /tasks/:id`: Test if a task is deleted by ID.	
24	**Frontend Testing** **Topics to Cover** **1. Testing React Components:** ● Setting up React Testing Library. ● Writing tests for functional components. ● Simulating user interactions (e.g., button clicks, form submissions). **2. Basic E2E Testing (Optional if time permits):** ● Introduction to Cypress for E2E testing. ● Writing a simple test to simulate user workflows. **Project: Testing a Task Manager App** **Objective**: Test the key features of a "Task Manager" app for managing tasks. **Frontend Testing:** ● **Setup:** Use React Testing Library for testing the React frontend. ● **Components to Test:** 1. **TaskList Component:** ■ Renders a list of tasks. ■ Displays "No tasks available" if the list is empty. 2. **AddTask Form:** ■ Verifies the form is submitted when valid data is entered. ■ Checks if the form clears after submission.	
	Project - Full Stack Application	14