ABOUT THE COURSE: MERN STACK WITH MONGO DB

TABLE 1	
OVERALL COURSE OBJECTIVE:	• Backend Development with Node.js and Express.js: middleware and how it's used in Express & Creating RESTful APIs using Express.js. • Database Management with MongoDB: Designing and implementing MongoDB schemas. • Full Stack Integration: Connecting the frontend and backend components of an application. • Asynchronous Programming: Understanding the basics of asynchronous programming in JavaScript. • Performance Optimization: Techniques for optimizing the performance of both frontend and backend code. • Real-world Project Experience: Working on a full-scale project to apply the learned concepts.
LEARNING OUTCOME:	• **Explore** Node.js with this comprehensive course, covering everything from setting up Express.js environments to mastering RESTful API development, asynchronous programming,error handling, debugging, and security best practices, empowering you to build robust and secure applications. • setup, schema building, CRUD operations, optimization, authentication, MERN performance, and deployment strategies for React and Node.js applications.

TABLE 2: MODULE-WISE COURSE CONTENT AND OUTCOME				
SL.NO	MODULE NAME	MODULE CONTENT	MODULE LEARNING OUTCOME	DURATION (HRS)
1	Introduction to Node.js	Intro to Backend Development What is the backend? Role and importance in web development Client-Server architecture Roles & Responsibility for Backend Developer Understanding Node.js What is Node.js? Node.js advantages and use cases Installing Node.js and npm Basic Node.js syntax and structure Node.js Fundamentals Global objects and modules The require function Creating a simple Node.js application event loop and asynchronous programming Working with	• Clients request; servers provide. • Create/ maintain server logic, databases, APIs. • Use global objects, modules. Understand async programming. • Work with different modules and handle http requests.	5

		Node.js Modules Core modules (fs, http, path, etc.) Creating custom modules Using npm packages File System and HTTP Module Reading and writing files Creating an HTTP server Handling HTTP requests and responses		
2	Fundamentals of Express.js	Introduction to Express.js What is Express.js? Settingup an Express.js application Basic routing with Express.js Middleware in Express.js Understanding middleware Built-in middleware functions Creating custom middleware Advanced Routing Route parameters Query strings Handling different HTTP methods (GET, POST, PUT, DELETE) Templating and Static Files	• Implement URL path routing.Use built-in/custom middleware. • Handle route parameters, query strings, HTTP methods. • Serve static files, render dynamic content with templating engines.	4

		Serving static files Usingtemplating engines(e.g., EJS, Pug) Rendering dynamic content		

3	Working with MongoDB	Introduction to MongoDB What is MongoDB? MongoDB vs. SQL databases Installing and setting up MongoDB Basic MongoDB Operations MongoDB data types and schema Inserting, updating, deleting, and querying documents Using MongoDB shell Mongoose ORM Introduction to Mongoose Setting up Mongoose with Node.js Defining schemas and models CRUD Operations with Mongoose Creating and reading documents Updating and deleting documents Validations and schema types	• NoSQL, document-based, flexible schema. JSON-like documents, dynamic schemas. Insert, • update, delete, query. • ODM for MongoDB in Node.js. Define schemas/models. Perform CRUD operations. Apply validations, use various schema types.	5
4	Integrating Node.js, Express,	Setting Up the Project Creatinga new	• Define routes for various functionalities.	3

	and MongoDBs	Express application Connecting to MongoDBwith Mongoose Project structure and organization Building RESTful APIs Defining API endpoints Creating CRUD routes Handling requests and responses Authentication and Security Introduction to authentication Implementing user authentication with JWT Securing routes and data	• Create, read, update, delete data. Manage within Express.js routes. Basics and JWT implementation. Secure routes, prevent unauthorized access.	
5	Advanced Concepts	Advanced Express.js Middleware for logging and debugging Error-handling middleware Optimizing performance Real-time Applications with Socket.io. Introduction to WebSockets Setting up Socket.io with Express Real-time communication between client and server.	• Logging, debugging, error handling. Optimize • application efficiency. • Understand WebSockets. Set up Socket.io with Express. Enable real-time client-server communication .	3

TABLE 3: OVERALL COURSE LEARNING OUTCOME ASSESSMENT CRITERIA AND USE-CASES

LEARNING OUTCOME	ASSESSMENT CRITERIA	USE-CASES
• Clients request; servers provide. • Create/ maintain server logic, databases, APIs. • Use global objects, modules. Understand async programming. • Work with different modules and handle http requests.	• Describe client-server architecture.Identify backend developer responsibilities.Understand basic Node.js syntax. Node.js Fundamentals Use global objects and modules.Employ require function accurately. • Explain event loop and asynchronous programming. Working with Node.js Modules.Utilize core modules (fs, http, path).Create/use custom modules. Install and use npm packages. • File System and HTTP Module Read/write files with Node.js. Configure HTTP servers and handle requests efficiently.	Use Case 1: Excel Migration Scenario: ABC Corporation, a long-standing company, has been diligently maintaining its records in a legacy CSV file for years. However, as technology evolves, they realize the need to modernize their data management system. The existing CSV file, containing outdated data, poses a challenge due to its outdated format and lack of compatibility with newer systems. Task: Reading Data from Legacy CSV File: Implement functionality to read data from the legacy CSV file, ensuring that all relevant data is captured accurately. Data Transformation:

		Develop logic to transform the data as necessary, addressing any inconsistencies or outdated information present in the legacy file. Writing Transformed Data to New CSV File: Create functionality to write the transformed data to a new CSV file, maintaining the desired format and structure. Use Case 2: File System Monitoring Scenario: In a bustling shared directory where files are frequently modified, added, or removed by multiple users, there's a pressing need for a robust monitoring solution. The objective is to develop a system capable of tracking file system events

		in real-time, logging each change for monitoring, debugging, and
		auditing purposes. As the project commences, the team focuses on designing and implementing a streamlined solution tailored to enhance operational efficiency and security in this dynamic file environment. Task: Implementing File System Monitoring Logic: Develop logic to use the chokidar package to watch for file system events such as file modifications, additions, or deletions within the specified directory. Logging File System Events: Create a mechanism to log the detected file system events to the console or a designated log

		file, providing real-time visibility into changes occurring within the directory. Handling Edge Cases: Account for potential edge cases such as handling errors, ensuring graceful shutdown, and handling large volumes of file system events efficiently.
• Implement URL path routing.Use • built-in/custo m middleware. • Handle route parameters, query strings, HTTP methods. Serve static files, render dynamic content with templating engines.	• Explain Express.js significance in the Node.js ecosystem. Describe its role in web application development. Install and • configure Express.js.Understand the project structure. Implement routing for various URL paths.Handle common HTTP methods. Use built-in and custom middleware effectively.Manage route parameters, query strings, and HTTP methods.Serve static files. Utilize templating engines for dynamic content rendering.	Use Case 1: Notes Taking App Scenario: Alice, a student, uses the Simple Note-Taking API to organize her study notes. She starts by creating a new note using a POST request with the title "Chemistry Formulas" and content containing various chemistry formulas. Next, she retrieves all her notes with a GET request to review them before an exam. After studying, Alice realizes she made a

		mistake in one formula. She updates the note using a PATCH request with the corrected content. Finally, after the exam, she deletes the note using a DELETE request to keep her notes organized and up to date.

		Task: Create a new note (POST /notes): This route creates a new note with a unique ID, using the request body's title and content fields. It then adds the new note to the notes array and responds with the newly created note and a status code 201 (Created). Get all notes (GET /notes): This route retrieves all notes stored in the notes array and responds with a JSON array of notes. Update a note by ID (PATCH /notes/:id): This route updates an existing note with the specified ID. It finds the index of the note in the notes array, updates its title and content fields with the values from the request body, and responds with the updated note. Delete a note by ID (DELETE /notes/:id): This route deletes a note with the specified ID from the notes array. It uses the filter method to create a new array without the deleted

		note and responds with a status code 204 (No Content) to indicate successful deletion. Use Case 2: CRUD Operations. Scenario: A developer is building a task management system where users can create, view, update, and delete tasks. Here's how you can map the CRUD operations to real-world actions within the task management system. Task: Create an API with POST method that enables you to add data into an array. Create an API with PUT method that enables
		you to update data in the array. Create an API with the Delete method that enables you to delete data in an array. Create an API with GET method that enables you to view all data in an array.

• NoSQL, document-based, flexible schema. JSON-like documents, dynamic • schemas. Insert, update, delete, query. • ODM for MongoDB in Node.js. Define schemas/models. Perform CRUD operations. Apply validations, use various schema types.	• Describe MongoDB's features such as flexible schema, scalability, and document-oriented storage. Explain key differences between • MongoDB and SQL databases, such as data model, query language, and scalability. Demonstrate understanding of MongoDB data types and schema. Perform basic operations like • inserting, updating, deleting, and querying documents using the MongoDB shell. Explain the purpose of Mongoose as an Object Data Modeling (ODM) library for MongoDB in Node.js applications. Describe Mongoose's role in simplifying interactions with MongoDB. Successfully set up Mongoose within a Node.js application. Define schemas and models effectively, ensuring proper structure and validation rules. Execute CRUD operations using Mongoose methods for creating, reading, updating, and deleting documents. Implement validations to ensure data integrity. Utilize different schema types as needed for	Use Case 1: Event Management System Scenario: Let's say you're planning a conference next month and need to organize it efficiently. You start by using the command line interface to create a new event. The system prompts you to enter details like the event name, date, location, and organizer's information. Once you provide these details, the system saves them to the database. Later, you want to review all upcoming events to ensure everything is on track. You use the system to retrieve a list of events, complete with organizer details. Seeing that everything looks good, you proceed with updating some event details. Maybe the event location changed to a hybrid setup, so you update that information with a simple command. Finally, after the successful completion of your

	various data structures.	conference, you use the system to delete the event from the database, keeping your records clean and organized. Task: To create events by giving the inputs asked in the terminal. Find the events which are created successfully. To retrieve all the events in the database. To update the event location to Hybrid by giving its _id. To delete any event by giving

		its _id. Use Case 2: Schemas and Models. Scenario: Imagine we are building a simple blogging platform where users can create posts. Each post will have a title, content, author name, and date of creation. We will use Mongoose to define a schema for the posts collection and create a model to interact with the database.

		Task: Creates a new document using the ProductSchema and saves it to the database. Example: Inserts a product with name "redmi", price 5000, color "black", and an additional field "range" with value "sss". Updates a document in the database that has the name "lava", setting its price to 10000. Deletes a document from the database that has the name "mi". Reads all documents from the "products" collection and logs the number of documents found.

• Define routes for various functionalities. Create, read, update, delete • data. Manage within Express.js routes. Basics and JWT implementation. Secure routes, prevent unauthorized access.	• Clearly define and document API endpoints, specifying their purpose and functionality. Develop CRUD routes that effectively handle data operations (Create, Read, Update, Delete). • Demonstrate proficiency in handling HTTP requests and responses within Express.js routes. Demonstrate understanding of authentication basics. Implement user • authentication using JWT and ensure secure routes and data protection. Utilize middleware for logging, debugging, and error handling effectively. Implement error-handling middleware to gracefully manage errors and exceptions. Optimize application performance through efficient coding practices and resource utilization. Explain WebSockets' purpose and identify suitable use cases. Set up Socket.io with Express to enable real-time communication between client and server, ensuring seamless data exchange.	Use Case 1: Task Management API. Scenario: Imagine you are building a task management application where users can sign up, log in, and manage their tasks. Upon signing up, users can create tasks, update their status (incomplete, in-progress, completed), delete tasks, and view tasks based on their status. The API handles user authentication securely, ensuring that only authenticated users can access task-related endpoints. Task: User Authentication: Create routes for user sign-up, sign-in, and sign-out. Implement logic to hash passwords securely using bcrypt and generate JWT tokens for authentication. Task Management: Define schemas and models for tasks using Mongoose. Create routes for creating, reading, updating, and deleting tasks. Implement

		functionality to retrieve tasks based on their status. Security and Validation: Implement middleware for verifying JWT tokens to protect authenticated routes. Validate user input and ensure data integrity and security throughout the application. Testing and Deployment: Test the API endpoints using tools like Postman or automated testing frameworks. Deploy the application to a production environment, ensuring scalability and performance optimizations. Use Case 2: User

		Authorization Scenario: A developer wants to implement user authentication and authorization in their web application using JWT. They choose Node.js with Express.js for the backend and MongoDB as the database. This allows users to securely sign up, sign in, and update their profile information. Task: Creating routes for user authentication (/user/signup, /user/signin, /user/update/:userMail). Implementing controller functions for handling user signup, signin, and update operations. Implementing signup (/user/signup) route to create a new user in the database after hashing the password. Implementing signin (/user/signin) route to authenticate users by comparing hashed passwords and issuing JWT tokens. Implementing update (/user/update/:userMail) route to allow users to update their email address after verifying JWT token. Generating JWT tokens using jsonwebtoken

		package after successful user authentication (signup and signin). Implementing a middleware (verifyUser) to verify JWT tokens and protect routes that require authentication (/user/update/:userMail).

• Logging, debugging, error handling. • Optimize application • efficiency. Understand WebSockets. Set up Socket.io with Express. Enable real-time client-server communication.	• Utilize middleware effectively to log relevant information and debug issues within Express.js applications. Implement error-handling middleware to • gracefully handle errors and exceptions, providing informative responses to users. • Demonstrate techniques to optimize application performance, such as efficient code design, minimizing database queries, and utilizing caching mechanisms. Understand the purpose and benefits of WebSockets in real-time applications.Successfully set up Socket.io with Express to enable bidirectional, real-time communication between clients and the server. Implement features leveraging real-time capabilities effectively.	Use Case 1: Backend Testing Scenario: Picture yourself creating a web application that demands user authentication. Users must be able to sign up with a valid email and password, sign in using their credentials, and access protected routes. Your application relies on MongoDB as the database, Express.js for the backend server, and JWT for handling authentication tokens securely. Your goal is to ensure accurate validation of user inputs, secure password hashing, and a smooth, dependable authentication process.

		Task: Implement User Routes: Create Express.js routes for user signup and signin functionalities. Include validation checks using express-validator for email format and password length during signup. Implement logic to hash passwords using bcrypt before storing them in the database during signup. User Authentication with JWT: Generate JWT tokens upon successful user sign in. Use JWT to authenticate and authorize user access to protected routes. Set up middleware (e.g., verifyUser) to validate JWT tokens and grant access to authenticated users. Handle User Requests: Implement route handlers for user signup and signin requests. Handle errors and return appropriate responses for invalid inputs, existing users, and

		authentication failures. Testing with Jest and Supertest: Set up Jest and Supertest for testing your API endpoints. Write unit and integration tests

		to ensure proper functioning of user signup, signin, and authentication flows. Test scenarios like valid inputs, invalid inputs, existing users, successful authentication, and failed authentication. Environment Variables and Security: Use dotenv to manage environment variables like database URI, JWT secret, etc. Ensure sensitive information (e.g., database credentials, JWT secret) is kept secure and not exposed in your codebase. Use Case 2: Chat App Scenario: A Company is facing communication challenges due to the

		inefficiency of email communication, leading to delays in project coordination and collaboration. Abc decides to implement a chat app that allows employees to communicate in real-time, and collaborate on projects more effectively. The app will be accessible desktop ensuring that employees can stay connected from anywhere. To achieve this, you decide to use Express.js to create a server that serves static files and handles WebSocket connections using Socket.IO. You create a basic HTML/CSS/JavaScript frontend where users can input their username, send messages, and see the chat history. Overall, this application provides a seamless and interactive chat experience for users, making it easy for them to communicate in real-time. Task: User Register: Allow users to set name.

		Real-time Messaging: Enable users to send and receive
		messages instantly. User Presence: Display the online/offline status of users and show when they are typing. Message History: Store and display chat history, allowing users to scroll through past messages.

TABLE 4: LIST OF FINAL PROJECTS (10 PROJECTS THAT COMPREHENSIVELY COVER ALL THE LEARNING OUTCOME)

SL.NO	FINAL PROJECT
1	Excel Migration
2	File System Monitoring
3	Notes Taking App
4	CRUD Operations for task management
5	Event Management System
6	Schemas and Models
7	Task Management API
8	User Authorization
9	Backend Testing
10	Chat App