Backend Development with

Node JS, Express JS, and MongoDB

- **Introduction to Node JS**
  - What is NodeJS
  - Javascript Runtime
  - Open Source
  - Purpose
- **Recap of Javascript**
  - Object
  - Array
  - Callback
- **Module Concept using Common JS Module Pattern**
  - What is common js
  - How to export Functions and variables
  - How to import functions and variables
- **Callbacks**
  - Introduction to Callbacks
  - Definition of a Callback
  - Why Callbacks are Important in JavaScript
  - Passing Functions as Arguments
  - Writing a Callback Function
  - Simulating Asynchronous Behavior with Callbacks
  - Refactoring Callback Hell into Promises or async/await

- **Callback hell**
  - What is Callback hell
  - How to implement it
  - Problems with Callback hell
  - 2 Use Cases of callback hell
- **Difference between callback hell and Promises**
- **Node Module System**
  - **File**
    - Readfile
    - Writefile
    - Rename file
    - Delete file
    - Creating new file
  - **Path**
    - path.basename()
    - path.dirname()
    - path.extname()
    - path.join()
  - **Http**
    - http.createServer()
    - server.listen()
    - http.get()
    - req.on()

- **Web Server**
    - Definition and Role
    - Difference Between Client and Server
    - Why Use Node.js as a Web Server?
    - Setting Up a Basic Node.js Web Server
    - Installing Node.js
    - Writing and Running Your First Web Server with `http` Module
    - Handling Basic HTTP Requests and Responses
- **Overview on How the Web Works**
    - Client-Server Architecture
    - Role of Client
    - Role of Server
    - Request Methods (GET, POST, PUT, DELETE, etc.)
    - Request and Response Structure (Headers, Body, Status Codes)
- **Responses**
    - Web page as a response
    - Json as a response
    - Normal text as a response
    - Setting headers for a response

Introduction to Express JS

- What is Express JS?
    - Definition and Purpose
    - Why Use Express for Web Applications
    - Comparison with Vanilla Node.js (Simplifies Routing and Middleware)
- Setting Up Express
    - Installing Express (`npm install express`)
    - Creating a Basic Express Server
    - Writing and Running Your First Express App
- **Key Features of Express**

        - Lightweight and Flexible Framework
        - Middleware Support
        - Simplified Routing
        - Integration with Other Tools and Libraries
- **Understanding of Web API**
    - What is a Web API?
    - Definition and Role in Web Development
- **Components of a Web API**
    - Endpoints and Resources
    - HTTP Methods and Status Codes
    - Input and Output (Request Body, Query Parameters, and Responses)
- **REST Principles**
    - Statelessness
    - Client-Server Architecture
    - Uniform Interface
    - Resource-Based URLs

HTTP Methods in REST APIs

- Overview of HTTP Methods
- Definition and Role
- Mapping CRUD Operations to HTTP Methods

GET

- Purpose (Retrieve Data)
- Examples of GET Endpoints
- Handling Query Parameters

POST

- Purpose (Create New Resources)

- Sending Data in the Request Body
- Validating Input Data

PUT

- Purpose (Update or Replace Resources)
- Differences Between PUT and PATCH

DELETE

- Purpose (Delete Resources)
- Handling Deletion and Response Codes

- **Building REST APIs with Express**
  - Setting Up Routes
  - Defining Routes for Different HTTP Methods
  - Using Route Parameters and Query Strings
  - Working with Middleware
  - Using Built-in Middleware (express.json(), express.urlencoded())
  - Creating Custom Middleware
- **Sending Responses**
  - JSON Responses (res.json)
  - Handling Errors (res.status, next)
- **Organizing Code**
  - Separating Routes, Controllers, and Middleware
  - Using Router Instances for Modularization
- **Hands-On Exercises**
  - Setting Up a Basic Express Server
  - Creating RESTful Endpoints for a Sample Application (e.g., To-Do List, Library System)
  - Implementing CRUD Operations Using GET, POST, PUT, and DELETE
  - Sending Proper Status Codes and Responses
  - Testing API Endpoints Using Tools like Postman

- **Understanding of Middleware**
  - **What is Middleware?**
    - Definition and Role in Express
    - Middleware as a Function Intercepting Requests/Responses
  - **Middleware Execution Flow**
    - Request-Response Lifecycle in Express
    - Chaining and Execution Order

- **Routing in Express**
  - **What is Routing?**
    - Definition and Purpose
    - Routing as URL Mapping
  - **Setting Up Routes in Express**
    - `app.get`, `app.post`, `app.put`, `app.delete`
    - Route Parameters (`req.params`)
    - Query Strings (`req.query`)
  - **Dynamic Routing**
    - Capturing Parameters in Routes
  - **Router Instances**
    - Creating and Using `express.Router()`
    - Modularizing Routes into Separate Files
    - Combining Multiple Routers
- **Middleware in Routing**
  - Applying Middleware to Specific Routes
  - Grouping Middleware with Routers
- **Environment Variables**
  - **What are Environment Variables?**
    - Definition and Purpose

        - Storing Configuration Data (e.g., API Keys, Database Credentials)
    - **Using Environment Variables in Node.js**
        - Accessing Variables with process.env
        - Example: Setting Up a PORT Variable
    - **Configuring Environment Variables**
        - .env Files
        - Installing and Using dotenv Package
- **Introduction to MongoDB**
    - **What is MongoDB?**
        - Definition and Features
        - Comparison with Relational Databases
        - Use Cases for MongoDB (e.g., Big Data, IoT, Real-Time Applications)
    - **Why Choose MongoDB?**
        - Schema-less Structure
        - High Performance and Scalability
        - Flexible Data Model
- **Installation of MongoDB**
    - **Downloading MongoDB**
        - Supported Platforms (Windows, macOS, Linux)
        - Choosing the Right Version (Community vs Enterprise)
    - **Installing MongoDB**
        - Step-by-Step Installation Guide for Different Operating Systems
        - Setting Up MongoDB as a Service (Optional)
    - **Verification**
        - Running MongoDB Server (mongod)
        - Verifying Installation with Mongo Shell

- **Installation of Mongo Shell**
    - **What is Mongo Shell?**
        - Definition and Purpose
        - Interaction with MongoDB Server
    - **Installing Mongo Shell**
        - Standalone Installation (if required)
        - Using the Shell with MongoDB Tools
- **Connecting Mongo Shell with MongoDB Server**
    - **Starting the MongoDB Server**
        - Running the mongod Command
    - **Connecting to the Server via Mongo Shell**
        - Starting Mongo Shell (mongo)
        - Default Connection to localhost and Port 27017
- **Creating the Database**
    - **Overview of MongoDB Databases**
        - How Databases are Created Dynamically
    - **Creating a Database**
        - Using use <database-name>
        - Verifying Created Databases with show dbs
- **Collections**
    - **What is a Collection?**
        - Collections vs Tables in Relational Databases
    - **Creating Collections**
        - Dynamic Creation on Data Insertion
        - Using db.createCollection()
    - **Listing and Dropping Collections**
        - Commands (show collections, db.collection.drop())

- **BSON Format**

    - **What is BSON?**
        - Definition and How it Differs from JSON
        - Binary-Encoded JSON for Efficient Storage
    - **Key Features of BSON**
        - Support for Data Types Like Date, Binary, ObjectId
- **CRUD Operations**
    - **Create**
        - Inserting Documents (db.collection.insertOne, db.collection.insertMany)
    - **Read**
        - Retrieving Data with find() and Query Filters
    - **Update**
        - Modifying Documents with updateOne, updateMany, and $set
    - **Delete**
        - Removing Documents with deleteOne and deleteMany
- **Relations in MongoDB**
    - **Types of Relations**
        - One-to-One,
        - One-to-Many,
        - Many-to-Many
    - **Modeling Relationships**
        - Embedded vs Referenced Approach
    - **Examples of Each Relation Type**
- **Operators in MongoDB**
    - **Query Operators**
        - $eq — Matches values equal to a specified value.
        - $ne — Matches values not equal to a specified value.
        - $gt — Matches values greater than a specified value.
        - $gte — Matches values greater than or equal to a specified value.
        - $lt — Matches values less than a specified value.

        - $lte — Matches values less than or equal to a specified value.
        - $in — Matches values in an array of specified values.
        - $nin — Matches values not in an array of specified values.
    - **Update Operators**
        - $set, $unset, $inc, $push
    - **Logical Operators**
        - $and,
        - $or,
    - **Array Operator**
        - $all — Matches arrays containing all specified elements.
        - $elemMatch — Matches documents where at least one array element satisfies specified conditions.
        - $size — Matches arrays with a specified number of elements.
    - **Hands-On Exercises**
        - Installing and Setting Up MongoDB
        - Creating a Database and Adding Collections
        - Performing CRUD Operations on Sample Data
        - Modeling Embedded Documents and Relationships
        - Writing Queries with Operators
- **Mongoose**
    - Introduction to Mongoose
        - What is Mongoose?
        - Benefits of Using Mongoose with MongoDB
        - Schema vs. Collection vs. Document
    - Defining Schemas
        - Creating a Schema
        - Adding Field Types and Validation
        - Using Schema Methods and Statics

    - Working with Models
        - Creating a Model from a Schema
        - CRUD Operations with Models
        - create(),
        - find(),
        - findById(),
        - updateOne(),
        - deleteOne(),
- **Authentication and Authorization using JWT**
    - **What is JWT (JSON Web Token)?**
        - Overview and Structure of JWT (Header, Payload, Signature)
        - Benefits of Using JWT for Authentication
    - **Implementing Authentication with JWT**
        - Setting Up Registration and Login Endpoints
        - Generating JWT Tokens
        - Storing Tokens on Client (Cookies or Local Storage)
    - **Authorization with JWT**
        - Protecting Routes Using Middleware
        - Verifying Tokens on Protected Routes
    - **Refreshing Tokens**
        - Why Token Expiry is Important
        - Implementing Refresh Tokens
- **Integration of Node.js, Express, and MongoDB**
    - Setting Up the Environment
        - Installing Dependencies (express, mongoose, dotenv)
        - Configuring MongoDB Connection with mongoose.connect()
    - **Building an Express Server**
        - Creating Routes for CRUD Operations
        - Middleware for Parsing JSON and Handling Errors
        - Connecting with MongoDB

        - Defining and Using Mongoose Models in Routes
        - Handling Query Results (e.g., find, save, update)
        - Using Try-Catch for Route Handlers
        - Postman for API Testing

- **Integration with React**
    - **Overview of MERN Stack**
        - Why Use React with Node.js, Express, and MongoDB?
        - Architecture of a Full-Stack MERN Application
    - **Connecting Frontend and Backend**
        - Setting Up Proxy in React for API Requests
        - Using axios or fetch for HTTP Requests
    - **Managing State in React**
        - Storing Fetched Data in State
        - Using Context API or Redux for Global State Management
        - Authentication with React and JWT
        - Storing JWT in Cookies or Local Storage
        - Using JWT for Protected Routes in React
        - Implementing Login and Logout Feature