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
- **Promises in JS**
    - Creation of Promise
    - Resolve
    - Reject
    - States of Promise
        - Pending
        - Rejected
        - Fulfilled
    - Accessing the data from Promise
        - Then catch
        - Async Await
        - Try catch

- **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()

        - path.resolve()
        - path.normalize()
        - path.relative()
        - path.parse()
        - path.format()
        - path.isAbsolute()
        -
    - Os
        - os.arch()
        - os.cpus()
        - os.endianness()
        - os.freemem()
        - os.homedir()
        - os.hostname()
        - os.loadavg()
        - os.networkInterfaces()
        - os.platform()
        - os.totalmem()

    - Http
        - http.createServer()
        - http.request()
        - http.get()
        - http.Server.listen()
        - http.Server.close()
        - http.Server.on()
        - http.IncomingMessage
        - http.ServerResponse
        - http.setHeader()
        - http.getHeader()

- **Asynchronous Nature of Node JS**

    - Javascript Engine
    - Execution Context
    - Callstack
    - Libuv
    - Thread Pool
    - Task Queue
    - Microtask Queue
    - Event Loop
- **Web Server**
    - Definition and Role
    - Difference Between Client and Server
    - Why Use Node.js as a Web Server?
    - Comparison with Traditional Servers
    - 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
    - HTTP Protocol Basics
    - Request Methods (GET, POST, PUT, DELETE, etc.)
    - Request and Response Structure (Headers, Body, Status Codes)
    - URL Breakdown (Protocol, Domain, Path, Query)
    - Lifecycle of an HTTP Request
    - Static vs Dynamic Content
    - What are Static Files?
    - Generating Dynamic Responses Using Node.js
    - Sending JSON Data as response
    - Sending File Content as response

    - Sending HTML File as a REsponse
    - Configuring CSS File
    - Sending Text Data as a Response
- **Routing in NodeJS**
    - Intro to Routing?
    - Definition and Importance
    - Role of Routing in Handling Different Endpoints
    - Basic Routing Using the http Module
    - Creating Routes for Different Paths
    - Handling Query Parameters and URL Parameters
- **Responses**
    - Web page as a response
    - Json as a response
    - Normal text as a response
    - Setting headers for a response

- **NPM**
    - Introduction to NPM
    - What is NPM?
    - Definition and Purpose
    - Role in Node.js Ecosystem
    - Why Use NPM?
    - Installing Node.js and NPM
        - Installation Steps (Windows, macOS, Linux)
        - Verifying Installation (node -v and npm -v)
        - Understanding NPM Versions
        - NPM CLI Versions
        - Updating NPM (npm install -g npm)

- **Packages**

    - What is an NPM Package?
    - Difference Between Local and Global Packages
- **Package Registry**
    - What is the NPM Registry?
    - Accessing the Registry via https://www.npmjs.com
- **Package Management Basics**
    - Installing, Updating, and Removing Packages
    - Difference Between Development and Production Depen
    - Working with package.json
    - What is package.json?
        - Purpose and Structure
        - Key Fields (name, version, dependencies)
        - Creating a package.json File
        - Using npm init and npm init -y
        - Manually Editing package.json
        - Understanding Dependency Versioning
        - Semantic Versioning (SemVer) Explained (^, ~, *)
        - Installing Packages
        - npm install <package> (Local Installation)
        - npm install -g <package> (Global Installation)
        - Removing Packages
        - npm uninstall <package>
        - Updating and Checking Dependencies
        - npm update
        - npm outdated
        - Lock Files (package-lock.json)
        - Alternatives to NPM
            - Yarn
            - PNPM
            - Differences Between NPM, Yarn, and PNPM
            - Creating a New Project with npm init
            - Installing and Using a Package (e.g., lodash)

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
- **Types of Web APIs**
  - REST APIs
  - SOAP APIs
  - GraphQL APIs

- **Components of a Web API**
  - Endpoints and Resources
  - HTTP Methods and Status Codes
  - Input and Output (Request Body, Query Parameters, and Responses)

- **Why Use Web APIs?**
    - Enabling Communication Between Systems
    - Supporting Multiple Platforms (Web, Mobile, IoT)

- **REST API Principles**
    - What is REST?
    - Definition (Representational State Transfer)
    - Characteristics of Restful APIs

- **REST Principles**
    - Statelessness
    - Client-Server Architecture
    - Uniform Interface
    - Resource-Based URLs

- **Best Practices for REST APIs**
    - Use Meaningful Resource Names
    - Handle Errors Gracefully
    - Version Your API
    - Secure the API (Authentication and Authorization)

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

- **HTTP Status Codes**
    - **Overview**
        - Categories of Status Codes (1xx, 2xx, 3xx, 4xx, 5xx)
    - **Commonly Used Status Codes**
        - 200 (OK)
        - 201 (Created)
        - 400 (Bad Request)
        - 404 (Not Found)
        - 500 (Internal Server Error)
    - **How to Send Status Codes in Express**
        - res.status().send()

- **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 or cURL

- **Understanding of Middleware**
  - **What is Middleware?**
    - Definition and Role in Express
    - Middleware as a Function Intercepting Requests/Responses
  - **Types of Middleware**
    - Built-in Middleware (e.g., express.json, express.urlencoded)
    - Third-party Middleware (e.g., morgan, cors)
    - Custom Middleware
  - **Middleware Execution Flow**
    - Request-Response Lifecycle in Express
    - Chaining and Execution Order
  - **Common Use Cases**
    - Logging
    - Authentication and Authorization
    - Data Validation
    - Error Handling

- **2. Custom Middleware**
  - **What is Custom Middleware?**
    - User-defined Functions for Specific Tasks
  - **How to Create Custom Middleware**
    - Syntax and Structure of Middleware (req, res, next)
  - **Middleware Parameters**
    - req (Request Object)
    - res (Response Object)
    - next (Function to Pass Control to the Next Middleware)

- **Examples of Custom Middleware**
    - Logging Middleware
    - Authentication Middleware
- **Placing and Using Middleware**
    - app.use for Global Middleware
    - Route-Specific Middleware
    - Middleware Priority 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

- **Error Handling**
  - **What is Error Handling?**
    - Capturing and Managing Errors in Express
    - Importance of Consistent Error Responses
- **Custom Error-Handling Middleware**
  - Structure (err, req, res, next)
  - Creating a Centralized Error Handler
- **Handling 404 Errors**
  - Setting Up a Default Route for Unmatched Paths
- **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

- **Data Types in MongoDB**
  - **Overview of Supported Data Types**
    - Common Types: String, Number, Boolean, Array, Object
    - Special Types: ObjectId, Date, Binary, Null
  - **Examples of Storing Data Using Various Types**

- **Embedded Documents**
  - **What are Embedded Documents?**
    - Storing Related Data Within a Single Document
  - **Use Cases for Embedded Documents**
    - Benefits (Performance and Simplicity)
  - **Examples**
    - Nested Objects and Arrays

- **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
  - **Aggregation Operators**
    - $sum, $avg, $group, $match
  - **Logical Operators**
    - $and,
    - $or,
    - $not,

        - $nor

    - **Element Operator**
        - $exists — Matches documents where the field exists or does not exist.
        - $type — Matches documents where the field is of a specified BSON data type.
    - Evaluation Operators
        - $text — Performs text search on indexed fields.
        - $where — Matches documents that satisfy a JavaScript expression.
    - **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.
    - **Projection Operators (Optional Query Enhancement)**
        - $slice — Limits the number of elements returned from an array.
        - $meta — Includes metadata in query results (e.g., text search scores).
        - $elemMatch — Projects matching array 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

- **Indexes in MongoDB**
  - **Introduction to Indexes**
    - What are Indexes?
    - Purpose of Indexes (Improved Query Performance)
    - Impact of Indexes on Read and Write Operations
  - **Single Field Index**
    - Definition and Use Case
    - Creating a Single Field Index
    - Querying with Single Field Indexes
    - Examples and Hands-On Practice
  - **Compound Field Index**
    - Definition and When to Use
    - Creating Compound Indexes
    - Syntax: db.collection.createIndex({ field1: 1, field2: -1 })
    - Importance of Index Order in Compound Indexes
    - Use Cases: Sorting and Filtering
    - Examples and Practice
  - **Multikey Index**
    - What is a Multikey Index?
    - Indexing Array Fields in MongoDB
    - Limitations of Multikey Indexes
    - Not Allowed on Fields with Both Arrays and Other Indexed Types
    - Examples of Querying with Multikey Indexes
  - **Text Index**
    - Overview of Text Indexes
    - Full-Text Search Capabilities in MongoDB
    - Creating Text Indexes
    - Using db.collection.createIndex({ field: "text" })
    - Querying with Text Indexes

- **Introduction to Aggregation in MongoDB**
  - **What is Aggregation?**
    - Definition and Purpose
    - Difference Between Aggregation Framework and Query Language
    - Common Use Cases (Data Transformation, Grouping, Analysis)
  - **Aggregation Pipeline**
    - Definition and Components
    - Pipeline Stages Overview
      - $lookup
      - $match (Filter Data)
      - $group (Group Documents)
      - $project (Transform Output Fields)
      - $sort (Order Documents)
      - $limit
- **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 or cURL 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 Features