# NodeJS Training - Day 5

By : LAKSHMIKANT DESHPANDE (M.Tech)

# Database Connection with Express Framework

- Day 4 Recap
- Middleware
- Deployment in Render
- Swagger
- Unit testing with Jest, Chai, Mocha etc etc…
- Database Connection
- Database CRUD Operations - Database queries and schemas
- Authentication and Authorization

# Session 1

- Day 4 Recap
  - Web Server
  - What is Express JS
  - HTTP Basics, Verbs, Status Codes
  - Handling HTTP Routes
  - Navigation Route options
  - Middleware

# Session 2

- Deployment in Render
- Postgres/MS SQL Database Connection
- Swagger

# Session 3

- Unit testing with Jest, Chai, Mocha etc etc…
- CRUD Operations with Database
- Database Queries and Schemas

# Session 4

- Live Assignment

# Middleware

**What is Middleware?**

Middleware functions are executed sequentially in the order they are added to the middleware stack. Each middleware function can perform operations such as:

- **Processing**: Read or modify req (request) and res (response) objects.
- **Ending**: Send a response to the client and end the request-response cycle.
- **Passing Control**: Call next() to pass control to the next middleware function.

# Types of Middleware

- **Application-Level Middleware**: Applied globally to the entire app.
- **Router-Level Middleware**: Applied to specific routes.
- **Built-In Middleware**: Provided by Express for common tasks.
- **Third-Party Middleware**: External libraries for additional features.
- **Error-Handling Middleware**: Catches and handles errors.

# Building Database Connection

Building a database connection in Node.js typically involves using a database driver or an Object-Relational Mapping (ORM) library.

- **MongoDB**: Use mongoose for easy interaction and connection.
- **MySQL**: Use mysql2 for connection to MySQL databases.
- **PostgreSQL**: Use pg for PostgreSQL databases.
- **SQLite**: Use sqlite3 for file-based SQLite databases.
- **ORM (Sequelize)**: Provides a unified way to interact with SQL databases, supporting multiple types like MySQL, PostgreSQL, and SQLite.

Each database and library has its own methods and options for managing connections, so always refer to the official documentation for detailed instructions and advanced configurations.

# CRUD - Database queries and schemas

- **Database Queries for CRUD** i,e Create, Read, Update, Delete Records
- **Schema Definition:**
  - Structure of a database defined by tables, columns, and relationships.
  - Determines how data is stored, organized, and manipulated.
- **Schema Design Best Practices**
  - **Normalization:** Organize tables to reduce redundancy and dependency.
  - **Relationships:** Clearly define relationships (one-to-one, one-to-many, many-to-many).
  - **Indexes:** Use indexes to improve query performance.
  - **Constraints:** Apply constraints to enforce data integrity (e.g., NOT NULL, UNIQUE).

# Error Handling

- **Use try-catch Blocks:** Wrap your asynchronous code with try-catch to catch errors and pass them to the next middleware.
- **Custom Error Classes:** Create custom error classes to handle different types of errors.
- **Centralized Error Handling:** Use a centralized error handling middleware to manage all errors in one place.
- **Environment-Specific Errors:** Show detailed error messages in development and generic messages in production.
- **Validation Errors:** Validate incoming data and handle validation errors appropriately.

# Authentication & Authorization

**Authentication** and **Authorization** are two critical aspects of securing applications.

- **Authentication:** Verifies user identity and issues tokens.
- **Authorization:** Manages access to resources based on roles or permissions.
- **Middleware:** Essential for adding authentication and authorization logic to routes.

**Best Practices**

- **Hash Passwords:** Always hash passwords before storing them in the database.
- **Use HTTPS:** Protect tokens and sensitive data during transmission.
- **Validate Input:** Sanitize and validate user input to prevent security vulnerabilities.
- **Handle Token Expiry:** Implement token expiration and refresh tokens if needed.