Week 12: Express Middleware, Error Handling, and Database Integration

Day 1: Introduction to Express Middleware

Theoretical Overview

Middleware functions are the backbone of the Express framework. A middleware function is a function that has access to the Request object (req), the Response object (res), and the next middleware function in the application's request-response cycle. These functions can execute any code, make changes to the request and response objects, end the request-response cycle, or call the next middleware function in the stack.

The Role of the next() Function

The next() function is a callback that passes control to the next middleware function. If the current middleware function does not end the request-response cycle (e.g., by sending a response), it must call next() to pass control to the next function. Otherwise, the request will be left hanging, and the client will eventually time out.

Day 2: Types and Binding of Middleware

Application and Router Level Middleware

Middleware can be bound to different levels of the application:

- **Application-level Middleware**: Bound to an instance of the app object using app.use() or app.METHOD(), where METHOD is the HTTP method of the request.
- **Router-level Middleware**: Functions exactly like application-level middleware, except it is bound to an instance of express.Router(). This is essential for the to separate different routes levels.

Built-in and Third-party Middleware

Express comes with several built-in middleware functions:

- **express.static**: Used for serving static files such as images, CSS, and JavaScript files for the **GadgetShop** frontend.
- **express.json()**: A built-in middleware that parses incoming requests with JSON payloads.
- **Response Compression**: Using third-party middleware like compression to reduce the size of the response body, improving performance for high-traffic apps.

Day 3: Execution Order and Custom Middleware

Writing Custom Middleware

Custom middleware allows us to inject specific logic into our **Gadget API**. For example, a middleware function can log the URL and method of every incoming request to the console or check if a user is logged in before allowing access to the **Phone Book** data.

The Importance of Order

Middleware execution is determined by the order in which functions are loaded. Middleware loaded first is executed first. This is why logging and security middleware are typically placed at the top of the file, while error-handling middleware is placed at the bottom.

Day 4: Raising and Handling Errors

Error Handling Middleware

Express comes with a default error handler, but custom error-handling middleware is necessary for professional applications. Error-handling middleware is defined with four arguments instead of three: (err, req, res, next).

- **Raising Errors**: When an error occurs (e.g., a database connection failure), the developer calls next(err) to skip all remaining non-error middleware and jump straight to the error handler.
- **Handling Errors**: The handler can log the error details and send a clean, user-friendly JSON message and a 500 status code back to the frontend.

Day 5: Accessing MongoDB from Node.js

Programmatic Connection

To make our **Gadget API** truly dynamic, we must connect it to MongoDB. This involves:

1. **Installing the Driver**: Using npm install mongodb to add the official Node.js driver.
2. **Connection URL**: Defining the connection string (usually mongodb://localhost:27017).
3. **MongoClient**: Initializing the client and establishing a connection to the server.

Performing Database Operations

Once connected, we can perform operations within our Express routes:

- **Retrieving Documents**: Using find() or findOne() to fetch product data for the **GadgetShop**.
- **Inserting Documents**: Using insertOne() to save a new contact in the **Phone Book** assignment.