Week 14: Assignment Work - Gadget API (Backend)

Day 1: RESTful API Design and Project Setup

Theoretical Overview

The **Gadget API** serves as the data layer for the MERN stack application. We designed this backend using the REST (Representational State Transfer) architectural style. This ensures that the frontend React application can communicate with the MongoDB database through a series of standardized HTTP requests.

Architecture and Dependencies

- **Node.js Runtime**: Leveraged the V8 engine for high-performance server-side JavaScript execution.
- **Express Framework**: Used to handle complex routing and middleware logic efficiently.
- **MongoDB Node Driver**: Installed the official driver (npm install mongodb) to enable programmatic access to the database.
- **NPM Management**: Utilized package.json to manage dependencies and versioning for the backend project.

Day 2: MongoDB Schema Design and Connection Logic

Data Modeling for Gadgets

Following the MongoDB data model, we defined a "Gadgets" collection. Unlike relational databases, we utilized a flexible JSON-like document structure to accommodate varying product specifications.

- **Fields**: Every document includes _id (auto-generated), name, price, description, category, and imageURL.
- **Connection Setup**: We implemented a connection URL mongodb://localhost:27017 to establish a link between the Express server and the Mongo server.
- **Database Operations**: Developed a dedicated module to retrieve the "Gadget" collection before the server starts listening for requests.

Day 3: Implementing CRUD Controller Logic

The Controller Pattern

To maintain "Separation of Concerns," logic was moved from route definitions into dedicated controller functions.

- **Read (GET)**: Implemented the find() method to retrieve the full catalog and the findOne() method to fetch details for a single gadget by its unique ID.
- **Create (POST)**: Developed endpoints to insert new documents into the collection, handling the request body through middleware.
- **Update and Delete**: Created logic for updateOne() and deleteOne() to manage inventory and product removal.

Day 4: Express Middleware and Request Parsing

Enhancing the Request-Response Cycle

Middleware was integrated into the **Gadget API** to preprocess incoming data and secure endpoints.

- **JSON Parsing**: Utilized express.json() middleware to handle POST request bodies containing product data.

- **Static Assets**: Configured express.static to serve product images stored on the server to the React frontend.
- **Error Handling**: Wrote custom error-handling middleware to catch database connection failures and send appropriate status codes.
- **CORS Support**: Implemented Cross-Origin Resource Sharing to allow the frontend (running on a different port) to access the API.

Day 5: Integration with React and Testing

Full-Stack Connectivity

The final day was dedicated to ensuring the **GadgetShop** (Frontend) and **Gadget API** (Backend) communicated seamlessly.

- **API Calls**: Integrated the useEffect hook in React to load the data from the Express server upon component mounting.
- **Asynchronous Handling**: Utilized Promises and async/await syntax to manage the non-blocking I/O nature of the data fetch.
- **Reflection**: Documented the successful programmatic access to MongoDB from Node, as practiced in the syllabus lab exercises.