Backend Track: Week 4 Assessment – Advanced Database Operations & Authentication

This assessment is part of a 5-week Backend learning journey. Each assessment builds progressively, guiding learners from fundamentals to advanced concepts in Node.js development.

Objective

This assessment challenges participants to deepen their understanding of database interactions by implementing more advanced queries and integrating basic user authentication into their Node.js API. Participants will learn to perform complex data manipulations, secure API endpoints, and manage user sessions. This exercise is critical for building secure, scalable, and feature-rich backend applications that handle sensitive data and multiple users.

Scenario/Problem Statement

Your project idea API now successfully persists data using a database. However, real-world applications often require more sophisticated data retrieval and management, as well as the ability to differentiate between users and protect certain data or functionalities. Imagine your project is a collaborative platform where users can create and manage their own project ideas, and only authenticated users can access or modify their ideas. Your task is to enhance your API to support advanced data queries (e.g., filtering, sorting, pagination) and implement a basic authentication mechanism to secure your API endpoints. This simulates a common and crucial aspect of modern backend development: ensuring data integrity, providing flexible data access, and safeguarding user information.

Key Concepts Covered

- **Advanced Database Queries:** Implementing filtering, sorting, and pagination for data retrieval.
- **Relational Database Concepts (if applicable):** Understanding relationships between tables (one-to-many, many-to-many) and JOIN operations.
- **Authentication:** Concepts of user authentication, including user registration, login, and session management.
- **JSON Web Tokens (JWT):** Understanding JWTs for stateless authentication.
- **Password Hashing:** Securely storing user passwords using hashing algorithms (e.g., bcrypt).
- **Middleware for Authentication:** Creating custom middleware to protect routes and verify user credentials.
- **Error Handling:** Implementing robust error handling for authentication failures and database query errors.
- **Environment Variables:** Securely managing sensitive information like database credentials and JWT secrets.

Task Instructions

Participants will enhance their existing Node.js API to include advanced database operations and a basic authentication system.

Step 1: Database Enhancement (if using SQLite/PostgreSQL)

1. **Add `users` Table:** If you are using a relational database (like SQLite or PostgreSQL), create a new `users` table to store user credentials. This table should have at least `id`, `username`, and `password` (hashed) fields. `sql CREATE TABLE users ( id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT UNIQUE NOT NULL, password TEXT NOT NULL );`
2. **Link Ideas to Users (Optional but Recommended):** Add a `userId` foreign key to your `ideas` table to link each idea to a specific user. This will allow you to

implement user-specific data access. `sql ALTER TABLE ideas ADD COLUMN userId INTEGER;`

Step 2: Implement Advanced Querying for Ideas

Enhance your `GET /api/ideas` endpoint to support query parameters for filtering, sorting, and pagination.

1. **Filtering:** Allow filtering ideas by `status` (e.g., `/api/ideas?status=Concept`).
2. **Sorting:** Allow sorting ideas by `title` or `createdAt` (if you add a timestamp) (e.g., `/api/ideas?sort=title&order=asc`).
3. **Pagination (Optional):** Implement `_limit` and `_page` query parameters for pagination (e.g., `/api/ideas?_limit=10&_page=1`).

   ```javascript // Example for SQLite (simplified) app.get("/api/ideas", (req, res) => { let query = "SELECT * FROM ideas"; const params = []; const conditions = [];

   if (req.query.status) { conditions.push("status = ?"); params.push(req.query.status); }

   if (conditions.length > 0) { query += " WHERE " + conditions.join(" AND "); }

   if (req.query.sort) { const order = req.query.order === "desc" ? "DESC" : "ASC"; query += `ORDER BY $ ${order}`; }

   // Add pagination logic here if implementing

   db.all(query, params, (err, rows) => { if (err) { res.status(400).json({"error": err.message}); return; } res.json({ "message": "success", "data": rows }); }); }); ```

## Step 3: Implement User Authentication

1. **Install Dependencies:** `bash npm install bcryptjs jsonwebtoken dotenv`
2. **Environment Variables:** Create a `.env` file in your project root to store your JWT secret: `JWT_SECRET=your_super_secret_jwt_key` Load environment variables at the top of your `server.js`: `javascript require("dotenv").config();`
3. **User Registration (POST /api/register):**
   ```

- Receive `username` and `password` from the request body.
- Hash the password using `bcryptjs`.
- Store the `username` and hashed `password` in your `users` table.
- Return a success message or the newly created user (without password).

4. **User Login (POST /api/login):**
   - Receive `username` and `password` from the request body.
   - Retrieve the user from the database by `username`.
   - Compare the provided password with the stored hashed password using `bcryptjs.compare()`.
   - If credentials are valid, generate a JWT using `jsonwebtoken` and send it back to the client.
5. **Authentication Middleware:** Create a middleware function (e.g., `authMiddleware.js`) to verify JWTs. This middleware should:
   - Extract the token from the `Authorization` header.
   - Verify the token using `jsonwebtoken.verify()`.
   - If valid, attach the user information (e.g., `userId`) to the `req` object and call `next()`.
   - If invalid or missing, send a 401 Unauthorized response.

   ```javascript // authMiddleware.js const jwt = require("jsonwebtoken");

   const authMiddleware = (req, res, next) => { const token = req.header("Authorization"); if (!token) { return res.status(401).json({ message: "No token, authorization denied" }); }

   try { const decoded = jwt.verify(token.split(" ")[1], process.env.JWT_SECRET); req.user = decoded.user; // Assuming your JWT payload has a 'user' object next(); } catch (e) { res.status(401).json({ message: "Token is not valid" }); } };

   module.exports = authMiddleware; ```
   ```

Step 4: Secure API Endpoints

1. **Protect Idea Endpoints:** Apply the `authMiddleware` to your CRUD endpoints for ideas. For example, only authenticated users should be able to create, update, or delete ideas. ```javascript const auth = require("./authMiddleware");

   // Protect these routes app.post("/api/ideas", auth, (req, res) => { /* ... */ }); app.put("/api/ideas/:id", auth, (req, res) => { /* ... */ }); app.delete("/api/ideas/:id", auth, (req, res) => { /* ... */ }); `` 2. **User-Specific Data (Optional but Recommended):** If you added `userId` to your `ideas` table, modify your CRUD operations to ensure users can only access/modify their own ideas.

Step 5: Test Authentication and Protected Routes

1. **Register a User:** Make a `POST` request to `/api/register`.
2. **Login User:** Make a `POST` request to `/api/login` to get a JWT.
3. **Access Protected Route:** Use the obtained JWT in the `Authorization` header (e.g., `Bearer YOUR_JWT_TOKEN`) to access a protected idea endpoint (e.g., `POST /api/ideas`). Verify that requests without a valid token are rejected.

Step 6: Documentation and Submission

1. **Update `README.md`:** Update your `README.md` file to include:
   - Details on advanced querying capabilities (filtering, sorting, pagination).
   - Comprehensive documentation for authentication endpoints (`/api/register`, `/api/login`).
   - Explanation of how JWTs are used for securing routes.
   - Instructions on how to set up environment variables.
   - Technologies used (Node.js, Express.js, Database, bcryptjs, jsonwebtoken, dotenv).
   - Challenges faced and solutions.
2. **Code Comments:** Ensure all new authentication logic, middleware, and advanced database queries are well-commented.

Deliverables

Participants are required to submit a single ZIP file containing the entire Node.js API project. The submission should include:

- The complete API project folder.
- A comprehensive `README.md` file.
- All source code files (including authentication middleware).
- The database file (if applicable).
- A `.env` file (with placeholder secret, or instructions to create one).

Evaluation Criteria

Assessments will be evaluated based on the following criteria, with a total of 100 points:

- **Advanced Querying (30 points):**
  - Successful implementation of filtering and sorting for ideas.
  - (Optional) Successful implementation of pagination.
- **User Registration (20 points):**
  - Functional `POST /api/register` endpoint.
  - Correct password hashing using `bcryptjs`.
- **User Login & JWT Generation (25 points):**
  - Functional `POST /api/login` endpoint.
  - Correct password comparison.
  - Successful generation and return of a valid JWT.
- **Authentication Middleware & Route Protection (15 points):**
  - Correct implementation of JWT verification middleware.
  - Successful protection of at least one API route.
- **Code Quality & Security (10 points):**
  - Clean, readable, and well-organized code.

    - Proper use of environment variables for secrets.
    - Robust error handling for authentication failures.

This assessment elevates the backend application to a production-ready level by incorporating advanced data management and crucial security features. Mastering authentication and advanced database interactions is essential for building robust, secure, and scalable web services that can handle real-world user demands.

Looking Ahead

In Week 5, you will focus on preparing your backend application for production deployment, including topics like containerization (Docker), continuous integration/continuous deployment (CI/CD), and strategies for monitoring and scaling your API. This will ensure your application is not only functional and secure but also ready for real-world usage and growth.

Recommended Resources

- **bcrypt.js GitHub:** https://github.com/dcodeIO/bcrypt.js
- **jsonwebtoken GitHub:** https://github.com/auth0/node-jsonwebtoken
- **dotenv GitHub:** https://github.com/motdotla/dotenv
- **OWASP Top 10:** https://owasp.org/www-project-top-ten/
- **SQL Joins Explained:** https://www.w3schools.com/sql/sql_join.asp

---