Assignment: Building a Backend Service for a URL Shortening Service
Objective: In this assignment, you will develop a backend service for a URL shortening service using Node.js, with your choice of MySQL or MongoDB as the database. You will also deploy the service to a cloud hosting platform such as Heroku, Render, Vercel, Netlify, or GitHub Pages. Additionally, you will create guidelines for testing the API, including what to test on the deployed API.
Requirements:
Backend Development:

1. Set up a Node.js project and an Express.js server.
2. Implement user registration and authentication using JWT (optional, for user-specific short URLs).
3. Create a database schema, either in MongoDB or MySQL. The schema should include tables/collections for `urls`.
4. Develop API endpoints for the following functionalities:
    - URL shortening (generate short URLs)
    - URL redirection (short URL to the original URL)
    - URL management (view, update, delete short URLs)
5. Implement proper error handling and validation for user inputs.
6. Use environment variables to store sensitive data like database credentials and JWT secret (if using authentication).
7. Implement data validation and sanitization for user inputs.
8. Ensure security measures to prevent misuse and unauthorized access.

Database Choice:

- Choose either MongoDB or MySQL for data storage. Include appropriate connection code, database setup, and schema creation scripts in your project.

Deployment:

- Deploy the backend service to one of the following cloud hosting platforms:
    - Heroku
    - Render
    - Vercel
    - Netlify (for serverless functions)
    - GitHub Pages (if using serverless functions)

Documentation:

1. Write clear documentation on how to set up and run the project locally.
2. Include a README file explaining the project's structure, dependencies, and how to run the server.
3. Document the API endpoints, their usage, and expected responses.

Unit Testing:

1. Use a testing framework like Mocha, Jest, or Jasmine to write unit tests.
2. Cover critical functions and endpoints with tests.
3. Perform both positive and negative tests to ensure code reliability.
4. Automate the testing process as part of your deployment pipeline.

API Testing Guidelines: To test the deployed API, you should follow these guidelines:

1. API Endpoint Testing:

- Test each API endpoint individually to ensure they perform the intended actions.

2. Positive Testing:

- Test API endpoints with valid inputs to verify that they return the expected responses.
- For example, you can test URL shortening with a valid URL.

3. Negative Testing:

- Test API endpoints with invalid inputs to ensure that they handle errors appropriately.
- For example, test URL shortening with invalid or missing data.

4. Authentication Testing (if applicable):

- If user-specific short URLs are implemented, test authentication endpoints (e.g., user login) by providing valid and invalid tokens.
- Ensure that unauthorized users cannot access protected endpoints.

5. Endpoint Validation Testing:

- For endpoints that require specific data formats (e.g., URLs), test them with both valid and invalid formats.

6. Endpoint Security Testing:

- Test security measures such as rate limiting and authentication (if applicable) to ensure they are effective.

7. API Response Testing:

- Validate that the API returns appropriate HTTP status codes (e.g., 200 for success, 401 for unauthorized) and error messages.

Submission Guidelines:

- Submit the codebase on a version control platform like GitHub.
- Ensure that the project is well-structured and follows best practices for code quality, including comments and a clean codebase.

Assessment: The assignment will be assessed based on the following criteria:

- Functionality: Does the backend service meet the requirements and work as expected?
- Code Quality: Is the code well-structured, clean, and maintainable?
- Documentation: Is the documentation clear and comprehensive?

- Deployment: Has the service been successfully deployed to the chosen cloud platform?
- Testing: Are there sufficient unit tests to ensure code reliability?

[BONUS] Swagger Link: Implement Swagger for your API endpoints following this guide: How to Document an Express API with Swagger UI and JSDoc.
Note:

- You may use any libraries or packages that you find suitable for the project.