Assignment for the role of Backend Developer

Objective: The goal of this assignment is to evaluate your knowledge and practical skills in building RESTful APIs using Node.js, Express, SQL, and MongoDB. You are expected to design a system that interacts with both a relational database (SQL) and a NoSQL database (MongoDB), and demonstrate best practices for building scalable and maintainable backend services. Use SQL in case you are not familiar with MongoDB.

Task Overview:

You are required to implement a simple **Task Management Application** with the following functionalities:

- Users can register, log in, and manage their tasks.
- Tasks can be associated with specific users.
- Tasks should support basic CRUD operations (Create, Read, Update, Delete).
- A task should have a title, description, due date, status (e.g., Pending, In Progress, Completed), and priority (Low, Medium, High).
- User credentials should be stored securely and should be hashed.
- Implement both SQL and MongoDB to store different types of data:
    - Use **SQL** for user data (users, login credentials).
    - Use **MongoDB** for task data (task descriptions, statuses, etc.).

Requirements:

1. **Technologies to use**:
    - **Node.js** (Backend framework)
    - **Express.js** (Routing and handling API endpoints)
    - **MongoDB** (NoSQL database for tasks)
    - **MySQL/PostgreSQL** (Relational database for users)
    - **JWT (JSON Web Token)** for user authentication
2. **Functionality**:
    - **User Registration & Authentication**:
        - Implement endpoints to register a user (POST /register).
        - Implement login functionality (POST /login) that returns a JWT upon successful authentication.
        - User data (username, email, password) should be stored in the SQL database.
        - Passwords should be hashed using bcrypt or a similar hashing library.
    - **Task Management**:
        - Create a task (POST /tasks): Accepts the task title, description, due date, priority, and status.

        - Retrieve a list of tasks (GET /tasks): Should support filtering by status, priority, or due date.
        - Update a task (PUT /tasks/{taskId}): Allows for updating the status and priority of a task.
        - Delete a task (DELETE /tasks/{taskId}): Removes the task from the system.
        - Task data should be stored in MongoDB.
3. **Authentication and Authorization**:
    - Implement JWT-based authentication for all endpoints.
    - Ensure that users can only access or modify their own tasks.
    - Protect the task-related routes with middleware that verifies the JWT.
4. **Validation**:
    - Validate input for all endpoints. For example:
        - Ensure the user provides valid email formats.
        - Ensure required fields for tasks (title, description) are provided.
        - Ensure that due dates are in a valid format (e.g., ISO 8601).
5. **Database Design**:
    - **SQL (Relational)**: Users table should have fields like:
        - id (Primary Key, auto-increment)
        - email (Unique)
        - password (Hashed)
        - createdAt (Timestamp)
    - **MongoDB (NoSQL)**: Tasks collection should have fields like:
        - userId (Reference to the user who created the task)
        - title
        - description
        - dueDate
        - status
        - priority
        - createdAt (Timestamp)
6. **Documentation**:
    - Provide a README file explaining:
        - How to set up the project and run it locally (including required environment variables).
        - The structure of your API endpoints.

- Any assumptions or decisions you made during development.

Bonus Features (Optional):

- Filter based report on the status of task.
- Implement **task deadlines notifications** (send an email or log a notification when a task is nearing its due date).
- Implement **pagination** for retrieving the list of tasks (optional but encouraged).

Evaluation Criteria:

- **Code Quality**: Clear, readable, and well-documented code.
- **Design**: Well-structured and maintainable code with proper separation of concerns (e.g., models, routes, controllers).
- **Correctness**: The functionality should work as described in the requirements.
- **Security**: Secure handling of passwords, JWTs, and sensitive data.
- **Performance**: Efficient queries to both the SQL and MongoDB databases.
- **Testing**: Proper validation and handling of edge cases.

Submission Instructions:

- Create a GitHub repository and push your code there.
- Share the repository link along with any instructions needed to run the application.

Deadline: 1 week from the assignment start date.