Back-end Engineer Technical Assessment | Bosta

Library Management System

Objective: Design and implement a simple library management system to manage books and borrowers.

Functional Requirements

1- Books:

- Add a book with details like title, author, ISBN, available quantity, and shelf location.
- Update a book's details.
- Delete a book.
- List all books.
- Search for a book by title, author, or ISBN.

2- Borrowers:

- Register a borrower with details like name, email, and registered date (Keep the user details as simple as possible).
- Update borrower's details.
- Delete a borrower.
- List all borrowers.

3- Borrowing Process:

- A borrower can check out a book. The system should keep track of which books are checked out and by whom.
- A borrower can return a book.
- A borrower can check the books they currently have.
- The system should keep track of due dates for the books and list books that are overdue.

Non-functional Requirements

1. Performance: The system should be optimized for reading operations since searching and listing books/borrowers will be frequent operations.
2. Scalability: The system design should support the addition of new features in the future, like reservations or reviews.
3. Security: Ensure that user inputs are validated to prevent SQL injection or other potential security threats.

Technical Requirements

1. Programming: The task shall be implemented using NodeJs.
2. Database: Use a relational database system (e.g., PostgreSQL, MySQL).
3. API: Implement a RESTful API to support all the above operations.
4. Error Handling: The system should gracefully handle errors and provide meaningful feedback.

Bonus - Ordered descending by value (Optional)

1. The system can show analytical reports of the borrowing process in a specific period and export the borrowing process data in CSV or Xlsx sheet formats e.x.
2. Exports all overdue borrows of the last month.
3. Exports all borrowing processes of the last month.
4. Implement rate limiting for the API to prevent abuse. (Choose only two endpoints to apply the rate-limiting).
5. Dockerizing the application using docker-compose.
6. Implement basic authentication for the API.
7. Add unit tests (Adding unit tests for only one module shall be enough, choose the easiest one).

Submission

- Codebase: Share the code repository link (like GitHub).
- Database: Provide a schema diagram and any necessary setup scripts.
- Documentation: Include instructions to set up and run the application. Document the API endpoints with expected inputs/outputs.

Evaluation Criteria

- All functional requirements should be implemented.
- Code Quality: Clear naming conventions, modularity, and comments.
- Database Design: Schema design, normalization, and indexing.
- API Design: Proper use of HTTP methods, status codes, and endpoint structuring.
- Error Handling: Proper error feedback and handling of edge cases.
- (Optional) Bonus features and enhancements.