Backend Technical Assessment: Building a REST API for a Library Management System

Objective:

The goal of this test is to build a RESTful API for a library management system. The system should manage books, authors, and users. It should also handle borrowing and returning books, with additional features such as role-based access control, search, and caching.

Requirements:

1. **Environment:**

- The candidate is required to use Laravel for this test.
- The solution should use a relational database (e.g., MySQL, PostgreSQL, SQLite).

2. **Entities:**

Book:

- id: Integer, primary key
- title: String, required
- isbn: String, required, unique
- published_date: Date
- author_id: Foreign key to Author
- status: Enum [Available, Borrowed], required

Author:

- id: Integer, primary key
- name: String, required
- bio: Text, optional
- birthdate: Date, optional

User:

- id: Integer, primary key
- name: String, required
- email: String, required, unique
- password: String, required
- role: Enum [Admin, Librarian, Member], required

BorrowRecord:

- id: Integer, primary key
- user_id: Foreign key to User
- book_id: Foreign key to Book
- borrowed_at: DateTime, required
- due_at: DateTime, required
- returned_at: DateTime, optional

API Endpoints:

Books:

- GET /books: Retrieve a list of all books.
- GET /books/{id}: Retrieve details of a specific book by ID.
- POST /books: Create a new book (Admin/Librarian only).
- PUT /books/{id}: Update an existing book by ID (Admin/Librarian only).
- DELETE /books/{id}: Delete a book by ID (Admin only).
- POST /books/{id}/borrow: Borrow a book (Member only, if available).
- POST /books/{id}/return: Return a borrowed book (Member only).

Authors:

- GET /authors: Retrieve a list of all authors.
- GET /authors/{id}: Retrieve details of a specific author by ID.
- POST /authors: Create a new author (Admin/Librarian only).
- PUT /authors/{id}: Update an existing author by ID (Admin/Librarian only).
- DELETE /authors/{id}: Delete an author by ID (Admin only).

Users:

- GET /users: Retrieve a list of all users (Admin only).
- GET /users/{id}: Retrieve details of a specific user by ID (Admin only).
- POST /users: Register a new user.
- PUT /users/{id}: Update user details (Admin only or self).
- DELETE /users/{id}: Delete a user by ID (Admin only).
- POST /login: Authenticate a user and return a sanctum token.

BorrowRecords:

- GET /borrow-records: Retrieve all borrow records (Admin/Librarian only).
- GET /borrow-records/{id}: Retrieve details of a specific borrow record by ID (Admin/Librarian only).

Additional Requirements:

1. Implement **role-based access control (RBAC)**:

- Admin: Full access to all resources.
- Librarian: Can manage books and authors, view borrow records, but cannot manage users.
- Member: Can view books/authors, borrow and return books, and update their profile.

2. Implement **search functionality** for books by title, author, or ISBN.
3. Implement **pagination** for listing books, authors, and borrow records.
4. Implement **input validation** for all endpoints.
5. Implement **error handling** with appropriate status codes and error messages.
6. Include **feature tests** for as many API endpoints as possible.

Bonus:

7. Implement **rate limiting** for API requests to prevent abuse.
8. Use **Docker** to containerize the application and database for easier deployment (NOT COMPULSORY TO USE DOCKER, JUST BONUS POINTS)

Evaluation Criteria:

- **Code Quality:** Clear, maintainable, and well-documented code with adherence to best practices.
- **Architecture:** Proper design patterns, modularization, and separation of concerns.
- **Functionality:** All required features are implemented correctly, including RBAC and asynchronous tasks.
- **Performance:** Efficient data access, caching, and handling of large datasets.
- **Security:** Implementation of authentication, RBAC, and other security best practices.
- **Testing:** Comprehensive coverage of important use cases with unit and integration tests.
- **Bonus:** Extra points for implementing bonus tasks, demonstrating advanced knowledge.

Submission:

- Provide a link to a GitHub repository with the complete code and the documentation of your API.
- Include instructions in the README file on how to set up and run the application locally, as well as how to run the tests.
- Document your API using postman or swagger.
- If Docker is used, provide the necessary Dockerfile and docker-compose.yml files.

Timeline:

Kindly submit your output via email to **oluyemi@bloocodetechnology.com** within 72 hours of

receiving this email. Your interview/presentation will be scheduled after receiving and evaluating

your submission (for successful candidates only).

Additional Notes:

Feel free to use any libraries or tools you're comfortable with.

We value your creativity and problem-solving skills. Feel free to add your personal touch to the tasks/project.

All the best!

Bloocode Technology Hiring Team