Software Developer - Back-End.

Maids.cc Software Developer Assessment | Back-End
Library Management System using Spring Boot

Build a Library Management System API using Spring Boot. The system should allow librarians to manage books, patrons, and borrowing records.

Requirements:

Entities:
● Create entities for:
● Book: Includes attributes like ID, title, author, publication year, ISBN, etc.
● Patron: Contains details like ID, name, contact information, etc.
● Borrowing Record: Tracks the association between books and patrons, including borrowing and return dates.

API Endpoints:
● Implement RESTful endpoints to handle the following operations:
● Book management endpoints:
● GET /api/books: Retrieve a list of all books.
● GET /api/books/{id}: Retrieve details of a specific book by ID.
● POST /api/books: Add a new book to the library.
● PUT /api/books/{id}: Update an existing book's information.
● DELETE /api/books/{id}: Remove a book from the library.
● Patron management endpoints:
● GET /api/patrons: Retrieve a list of all patrons.
● GET /api/patrons/{id}: Retrieve details of a specific patron by ID.
● POST /api/patrons: Add a new patron to the system.
● PUT /api/patrons/{id}: Update an existing patron's information.
● DELETE /api/patrons/{id}: Remove a patron from the system.
● Borrowing endpoints:
● POST /api/borrow/{bookId}/patron/{patronId}: Allow a patron to borrow a book.
● PUT /api/return/{bookId}/patron/{patronId}: Record the return of a borrowed book by a patron.

Data Storage:
● Use an appropriate database (e.g., H2, MySQL, PostgreSQL) to persist book, patron, and borrowing record details.
● Set up proper relationships between entities (e.g., one-to-many between books and borrowing records).

Validation and Error Handling:
● Implement input validation for API requests (e.g., validating required fields, data formats, etc.).
● Handle exceptions gracefully and return appropriate HTTP status codes and error messages.

Security (Optional - for extra credit):

● Implement basic authentication or JWT-based authorization to protect the API endpoints.

Aspects (Optional - for extra credit):
● Implement logging using Aspect-Oriented Programming (AOP) to log method calls, exceptions, and performance metrics of certain operations like book additions, updates, and patron transactions.

Caching (Optional - for extra credit):
● Utilize Spring's caching mechanisms to cache frequently accessed data, such as
book details or patron information, to improve system performance.

Transaction Management:
● Implement declarative transaction management using Spring's @Transactional
annotation to ensure data integrity during critical operations.

Testing:
● Write unit tests to validate the functionality of API endpoints.
● Use testing frameworks like JUnit, Mockito, or SpringBootTest for testing.

Documentation:
● Provide clear documentation on how to run the application, interact with API
endpoints, and use any authentication if implemented.

Evaluation Criteria:
● Functionality: Ensure that CRUD operations for books, patrons, and borrowing records work correctly.
● Code Quality: Evaluate the code for readability, maintainability, and adherence to best practices.
● Error Handling: Check for proper handling of edge cases and validation errors.
● Testing: Assess the coverage and effectiveness of unit tests.
● Bonus: Consider additional features, like authorization, effective usage of transactions, caching, and aspects.

Submission Requirements: Share the project via a Github/Gitlab link below. *

SUBMIT ANSWERS

RESET FORM