# Final Project - Cesa Backend

**Overview**
The Library Management System (LMS) is a backend API designed to provide the core functionalities for managing a library's book collection and user interactions.

**Entities**

- **User:** Represents a person who can borrow and return books.
  - **Attributes:** User ID, Name, Email, Password, Role (e.g., Student, Faculty, Staff)
- **Book:** Represents a physical or digital copy of a book.
  - **Attributes:** Book ID, Title, Author, ISBN, Availability Status
- **Loan:** Represents a transaction where a user borrows a book.
  - **Attributes:** Loan ID, User ID, Book ID, Loan Date, Due Date, Return Date (if applicable)

**Key Features**
- **User Management:** APIs for creating, updating, and deleting user accounts.
- **Book Management:** APIs for adding, removing, and updating book records, including title, author, ISBN, and availability status.
- **Borrowing and Returning:** APIs for recording book borrowing and return transactions, tracking due dates, and calculating fines.
- **Book Reservations:** APIs for placing and managing book reservations.
- **Reporting:** APIs for generating various reports, such as overdue books, popular books, and user activity.

**Technical Implementation**

- **Database:**
  - Use a relational database (e.g., PostgreSQL, MySQL) to store user, book, and loan data.
  - Utilize an Object-Relational Mapper (ORM) for simplified database interactions.
- **Caching:**
  - Implement a caching mechanism (e.g., Redis) to store frequently accessed data in memory.
  - Develop strategies for cache invalidation to ensure data consistency.
- **Error Handling and Data Validation:**
  - Employ appropriate exception handling mechanisms to catch and handle errors.
  - Validate input data to ensure it meets required format and constraints.
- **Modular Architecture:**
  - Divide the LMS into well-defined modules or components (e.g., user management, book management, reporting).
  - Design each module to be independently testable for easier development and maintenance.
- **Containerization:**

- Containerize the LMS application using Docker to provide a consistent and portable environment.
- Use Docker Compose to manage multiple containers (e.g., for the application, database, and cache) as a single unit.

---

**Important APIs**

## User Management

1. **CreateUser**
   - Request: `POST /users`
   - Body: `{"name": "John Doe", "email": "johndoe@example.com", "password": "password123", "role": "student"}`
   - Response: `{"id": 1, "name": "John Doe", "email": "johndoe@example.com", "role": "student"}`
2. **GetUserById**
   - Request: `GET /users/{userId}`
   - Response: `{"id": 1, "name": "John Doe", "email": "johndoe@example.com", "role": "student"}`
3. **UpdateUser**
   - Request: `PUT /users/{userId}`
   - Body: `{"name": "John Smith", "email": "johnsmith@example.com"}`
   - Response: `{"id": 1, "name": "John Smith", "email": "johnsmith@example.com", "role": "student"}`
4. **DeleteUser**
   - Request: `DELETE /users/{userId}`
   - Response: `{"message": "User deleted successfully"}`

## Book Management

5. **CreateBook**
   - Request: `POST /books`

    - Body: `{"title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "isbn": "9780743273565"}`
    - Response: `{"id": 1, "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "isbn": "9780743273565", "availability_status": "available"}`
6. **GetBookById**
    - Request: `GET /books/{bookId}`
    - Response: `{"id": 1, "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "isbn": "9780743273565", "availability_status": "available"}`
7. **UpdateBook**
    - Request: `PUT /books/{bookId}`
    - Body: `{"title": "The Great Gatsby (New Edition)"}`
    - Response: `{"id": 1, "title": "The Great Gatsby (New Edition)", "author": "F. Scott Fitzgerald", "isbn": "9780743273565", "availability_status": "available"}`
8. **DeleteBook**
    - Request: `DELETE /books/{bookId}`
    - Response: `{"message": "Book deleted successfully"}`

**Borrowing and Returning**

9. **BorrowBook**
    - Request: `POST /loans`
    - Body: `{"user_id": 1, "book_id": 1}`
    - Response: `{"id": 1, "user_id": 1, "book_id": 1, "loan_date": "2023-10-01", "due_date": "2023-10-15"}`
10. **ReturnBook**

- Request: `PUT /loans/{loanId}`
- Body: `{"return_date": "2023-10-10"}`
- Response: `{"id": 1, "user_id": 1, "book_id": 1, "loan_date": "2023-10-01", "due_date": "2023-10-15", "return_date": "2023-10-10"}`

**Book Reservations**

11. **ReserveBook**

- Request: `POST /reservations`
- Body: `{"user_id": 1, "book_id": 1}`
- Response: `{"id": 1, "user_id": 1, "book_id": 1}`

12. **CancelReservation**

- Request: `DELETE /reservations/{reservationId}`
- Response: `{"message": "Reservation canceled successfully"}`

**Reporting**

13. **GetOverdueLoans**

- Request: `GET /reports/overdue-loans`
- Response: `[{"id": 1, "user_id": 1, "book_id": 1, "loan_date": "2023-10-01", "due_date": "2023-10-15"}]`

14. **GetPopularBooks**

- Request: `GET /reports/popular-books`
- Response: `[{"id": 1, "title": "The Great Gatsby", "borrows": 10}]`

15. **GetUserActivity**

- Request: `GET /reports/user-activity/{userId}`
- Response: `[{"id": 1, "book_id": 1, "loan_date": "2023-10-01", "due_date": "2023-10-15"}]`