***Book Rental System***

**1. Introduction**

The Book Rental Project is a backend application designed to manage book rentals and returns, track transactions, and calculate rent. This system handles interactions between users and books, ensuring efficient management of book inventories and user transactions.

**2. Project Overview**

* **Project Name:** Book Rental System
* **Objective:** To provide an API that manages book rentals, returns, and transactions, including tracking the status of each transaction and calculating rent.

**3. Technologies Used**

* **Programming Language:** JavaScript
* **Backend Framework:** Node.js
* **Database:** MongoDB
* **ORM/ODM:** Mongoose
* **API Documentation:** Swagger (if used)
* **Version Control:** GitLab

**4. Database Design**

The project utilizes two MongoDB databases:

1. **Books and Users Database:**
   * **Collections:**
     + books: Stores information about books, including title, author, and availability status.
     + users: Stores user information such as user ID and contact details.
2. **Transactions Database:**
   * **Collections:**
     + transactions: Records all transactions including issue and return details, calculated rent, and associated book and user IDs.

**5. API Endpoints**

**5.1. Issue Book**

* **Endpoint:** /api/transactions/issue
* **Method:** POST
* **Description:** Issues a book to a user.
* **Request Body:**

{

"bookId": "string",

"userId": "string",

"issueDate": "ISODate"

}

* **Response:**

{

"transactionId": "string",

"message": "Book issued successfully"

}

* **Flow:**
  1. Validate input data.
  2. Check if the book and user exist.
  3. Create a new transaction record.
  4. Return success message with transaction ID.

**5.2. Return Book**

* **Endpoint:** /api/transactions/return
* **Method:** POST
* **Description:** Returns a book and calculates the rent.
* **Request Body:**

{

"transactionId": "string",

"returnDate": "ISODate"

}

* **Response:**

{

"transactionId": "string",

"rent": "number",

"message": "Book returned successfully"

}

* **Flow:**
  1. Validate input data.
  2. Retrieve the transaction record.
  3. Calculate rent based on the issue and return dates.
  4. Update the transaction record.
  5. Return the calculated rent and success message.

**5.3. Query Transactions**

* **Endpoint:** /api/transactions
* **Method:** GET
* **Description:** Queries transactions by book, user, and date range.
* **Query Parameters:**
  + bookId (optional): Filter by book ID.
  + userId (optional): Filter by user ID.
  + startDate (optional): Filter by start date.
  + endDate (optional): Filter by end date.
* **Response:**

{

"transactions": [

{

"transactionId": "string",

"bookId": "string",

"userId": "string",

"issueDate": "ISODate",

"returnDate": "ISODate",

"rent": "number"

}

]

}

* **Flow:**
  + Validate query parameters.
  + Query the database based on provided filters.
  + Return the list of transactions.

**6. Data Flow**

1. **Client Request:**
   * A client sends a request to one of the API endpoints.
2. **Server Processing:**
   * The server receives the request, routes it to the appropriate controller, processes it, and performs necessary validations and calculations.
3. **Database Interaction:**
   * The server interacts with the MongoDB database to query, insert, update, or delete records as needed.
4. **Response Generation:**
   * After processing, the server generates a response and sends it back to the client.
5. **Client Receives Response:**
   * The client receives and processes the response.

**7. Implementation Details**

* **Validation:** Ensures all inputs are correctly formatted and meets criteria.
* **Error Handling:** Handles various errors such as missing data, invalid IDs, or database issues.
* **Authentication (if applicable):** Handles user authentication and authorization to secure endpoints.
* **Logging:** Tracks API usage and errors for monitoring and debugging.

**8. Challenges and Solutions**

* **Challenge:** Managing book availability and preventing multiple issues of the same book.
  + **Solution:** Implemented checks before issuing books and updated availability status upon issue and return.
* **Challenge:** Calculating accurate rent based on the issue and return dates.
  + **Solution:** Used date arithmetic to compute rent based on the rental duration.

**9. Future Improvements**

* **User Authentication:** Implement user authentication to secure transactions and ensure authorized access.
* **Performance Optimization:** Optimize database queries and indexing for better performance.
* **Enhanced Features:** Add functionality for book reservations, notifications for due dates, and reporting features.

**10. Conclusion**

The Book Rental Project successfully manages book rentals and returns, ensuring efficient transaction processing and accurate rent calculation. The implementation provides a solid foundation for further enhancements and scalability.