Senior .NET Developer Assignment:

Starters - Warm-up Tasks

Before starting the main Library API assignment, please complete these small warm-up exercises. They are designed to test your coding fundamentals, style, and problem-solving approach, all within a simple library-related theme.

1. Check if a Book ID is a Power of Two

Implement a method that checks whether a given **Book ID** number is a power of 2.

2. Reverse a Book Title

Write a method that takes a **book title** as input and returns it reversed.
(Example: "Moby Dick" → "kciD yboM")

3. Generate Book Title Replicas

Create a method that **repeats a given book title** a specified number of times.
(Example: ("Read", 3) → "ReadReadRead")

4. List Odd-Numbered Book IDs

Implement a method that prints all odd numbers between 0 and 100, simulating the generation of **odd-numbered IDs** for certain book collections or limited editions.

Main task - Library API System

You are recommended to spend as little time as possible to solve the following problem. Ideally, it should take 5-8 hours for a person with the right competence.

Problem Statement

A public library requires a robust API-driven application to manage its books, borrowers, and lending activity. The goal is to provide librarians with reliable insights into inventory status and borrowing patterns, enabling them to optimize day-to-day operations and long-term planning. The system should expose a set of HTTP API endpoints designed to answer key business questions, such as:

Inventory Insights

What are the most borrowed books?

User Activity

Which users have borrowed the most books within a given time frame?
Estimate a user's reading pace (pages per day) based on the borrow and return duration of a book, assuming continuous reading.

Borrowing Patterns

What other books were borrowed by individuals who borrowed a specific book?

Constraints & Assumptions

The application may be prepopulated with sample data for demonstration purposes.
The API will not be publicly accessible, so security concerns are out of scope for this assignment.
User management (e.g., authentication, authorization) is not considered part of the assignment.

Expectations

- The application should be **structured cleanly** into at least two layers:
    - **API Layer**: Handles HTTP interactions.
    - **Service Layer**: Encapsulates business logic and database operations.
- Communication between layers should preferably use a **modern RPC framework** (e.g., **gRPC**).
- **Data Storage**:
    - Use **MongoDB or SQL** for data persistence.
      (You may simulate connections if needed)
- **Testing Requirements**:
    - Emphasis on **test coverage**:
        - **Unit Tests** for individual methods/classes
        - **Functional Tests** for key features
        - **Integration Tests** for database interactions
        - **System Tests** to validate complete user flows
    - Tests should be automated and easy to execute.
- **Code Quality**:
    - Clean, maintainable, and well-structured code is highly valued.¨
    - Clear separation of concerns across different layers.
    - Proper use of modern .NET patterns and best practices.

Technologies

- **Primary Language**: C#, .NET Core
- **Database**: MongoDB or SQL
- **Mandatory**: Usage of gRPC for internal service communication
- **Testing Frameworks**: NUnit, xUnit, or similar

Notes

- Feel free to reach out anytime if you have questions or need clarification.
- Submission should include a **README** file explaining how to run the project and the tests.