Assumptions

Library-related assumptions

1. Unique identification
    - Each book has a unique ISBN
    - Multiple copies of the same ISBN are allowed; each copy is a separate Book entry

2. Borrowing constraints
    - A single copy of a book can only be borrowed by one borrower at a time
    - Once a book is returned, it becomes available for borrowing again
    - BorrowDate is automatically set when a book is borrowed
    - ReturnDate is set when a book is returned; until then, it remains null
    - The system does not track overdue or due dates - borrowing is assumed to be indefinite

3. Borrower constraints
    - Each borrower has a unique ID and email address
    - Borrowers cannot borrow more than one copy of the same book simultaneously

System/API assumptions

1. All API requests are authenticated by API user - security is out of the scope
2. Input validation is in place for required fields (e.g., ISBN, email)
3. Only one database is assumed; concurrent updates are minimal - high concurrency is out of the scope
4. The system runs on Java 17 (Spring Boot), MySQL and Docker for containerization
5. **Important:** The system assumes a uat-db and prd-db are pre-configured by the API user

Deployment/DevOps assumptions

1. Database is containerized and ephemeral for testing purposes

Limitations

1. Security/Authentication
    - Current limitation: the APIs do not implement OAuth/JWT, or other authentication/authorization mechanisms
    - Improvement: Integrate Spring Security with OAuth2/JWT to secure endpoints, ensure only authorized API users or borrowers can access or perform actions

2. Due date tracking
    - Current limitation: The system does not track borrowing due dates, overdue books, or automatic notifications

- Improvement: Add dueDate to the Borrowing entity, implement a scheduler to flag overdue borrowings, and notify borrowers via email or in-app messages

3. Concurrency/high-load handling
   - Current limitation: The service assumes low concurrency. Two users attempting to borrow the same book simultaneously could cause race conditions
   - Improvement: Use database-level locks or optimistic/pessimistic locking to prevent simultaneous borrowing of the same book

4. Limited borrower management
   - Current limitation: Borrower management is minimal (no update/delete endpoints, no borrower profiles)
   - Improvement: Add full CRUD operations for borrowers, with proper validation and history tracking

5. Deployment
   - Current limitation: The system is designed for a single containerized deployment and does not handle horizontal scaling or load balancing
   - Improvement: Use Kubernetes deployments with multiple replicas, include a service mesh or ingress, and configure health checks to allow horizontal scaling

MySQL Justification

1. Structured relational data
   - The library system has clearly defined entities: Book, Borrower, Borrowing
   - Relationships are well-suited to relational tables (e.g., Borrowing references Book and Borrower)
   - MySQL's ACID-compliant transactions ensure consistency - critical when handling borrow/return operations (atomicity, consistency, isolation, durability)

2. Referential integrity and constraints
   - MySQL supports foreign keys, unique constraints, and validation rules out-of-the-box
   - For example:
     - Borrowing.book_id must reference a valid book
     - Borrowing.return_date logic is enforced in the service but can be backed by DB constraints if needed

3. Simplicity and familiarity
   - MySQL is widely used, easy to set up in Docker containers, and integrates seamlessly with Spring Boot via JDBC/JPA
   - Ideal for a small to medium-scale project like this take-home test

API Documentation

[POST] Create Book

Endpoint: /api/books/create
Description: registers a new book in the library system

Request body:

```
{
    "isbn": "9788578439156",
    "title": "Avatar",
    "author": "James Cameron"
}
```

Field	Type	Required	Description
isbn	String	Yes	Unique book ISBN
title	String	Yes	Title of the book
author	String	Yes	Author of the book

Response body:

- 200 OK

```
{
    "author": "James Cameron",
    "id": 10,
    "isbn": "9788578439156",
    "title": "Avatar"
}
```

- 400 Bad Request: when required field is missing or invalid

```
title: Book title cannot be empty
```

- 400 Bad Request: when a book with the same ISBN already exists with conflicting title/author

```
ISBN: 9788578439156 already exists with a different name/author
```

[GET] Get Books

Endpoint: /api/books/getBooks
Description: retrieves a list of all registered books in the library system

Response:


```
[
    {
        "author": "George RR Martin",
        "id": 3,
        "isbn": "3528164190",
        "title": "Clean Codes"
    },
    {
        "author": "George Michael",
        "id": 4,
        "isbn": "9783376756818",
        "title": "Harry Potter"
    },
    {
        "author": "George Michael",
        "id": 5,
        "isbn": "9783376756818",
        "title": "Harry Potter"
    },
    {
        "author": "Abel Tesfaye",
        "id": 6,
        "isbn": "9794050977160",
        "title": "Lord of the Rings"
    },
    {
        "author": "Abel Tesfaye",
        "id": 7,
        "isbn": "9794050977160",
        "title": "Lord of the Rings"
    },
    {
        "author": "Winnie the Pooh",
        "id": 8,
        "isbn": "9793094342262",
        "title": "Avatar"
    },
```

```
    {
        "author": "Mahatma Ghandi",
        "id": 9,
        "isbn": "9785034446888",
        "title": "Ebola: The Best Thing to Ever Happen"
    },
    {
        "author": "James Cameron",
        "id": 10,
        "isbn": "9788578439156",
        "title": "Avatar"
    }
]
```


[POST] Create Borrower

Endpoint: /api/borrower/create
Description: registers a new borrower in the library system

Request body:


```
{
    "name": "Austin Teck",
    "email": "austin.teck@gmail.com"
}
```


Field	Type	Required	Description
name	String	Yes	Name of the author
email	String	Yes	Email of the author

Responses body:

- 200 OK


```
{
    "email": "austin.teck@gmail.com",
    "id": 3,
    "name": "Austin Teck"
}
```


- 400 Bad Request: when required field is missing or invalid

```
name: Borrower name cannot be empty;
```

- 400 Bad Request: when author already exists

```
Borrower email: austin.teck@gmail.com already exists
```

[POST] Borrow Book

Endpoint: /api/books/borrow
Description: borrows a book from the library system

Request body:

```
{
    "bookId": 3,
    "borrowerId": 1
}
```

Field	Type	Required	Description
bookId	String	Yes	Book ID to be borrowed
borrowerId	String	Yes	Borrower ID of borrower

Response body:

- 200 OK

```
{
    "bookId": 3,
    "borrowerId": 1,
    "title": "Clean Codes",
    "author": "George RR Martin",
    "email": "austin@gmail.com",
    "borrowDate": "2026-01-07T20:39:02.0773129",
    "returnDate": null
}
```

- 400 Bad Request: when book is already borrowed

```
Book: 3 is already borrowed
```

- 400 Bad Request: when book does not exist

```
Book: 11 not found
```

- 400 Bad Request: when borrower does not exist

```
Borrower: 10 not found
```

[POST] Return Book

Endpoint: /api/books/return
Description: returns a book to the library system

Request body:

```
{
    "bookId": 3
}
```

Field	Type	Required	Description
bookId	String	Yes	Book ID to be returned

Response body:

- 200 OK

```
{
    "bookId": 3,
    "borrowerId": 1,
    "title": "Clean Codes",
    "author": "George RR Martin",
    "email": "austin@gmail.com",
    "borrowDate": null,
    "returnDate": "2026-01-07T20:41:56.1597556"
}
```

- 400 Bad Request: when book is not borrowed

```
Book: 7 is not currently borrowed
```