Library API - Technical Documentation

A Spring Boot RESTful API for managing library borrowers and books, supporting borrowing and returning operations. It includes input validation, business rule enforcement, clear error handling, and testing instructions.

---

1. Task Summary

This API supports:

- Registering a new borrower with a unique email.
- Registering new books with ISBN validation rules.
- Listing all books in the system.
- Borrowing and returning books for a specific borrower.

Rules enforced:

- ISBN uniqueness: same ISBN → same title & author; multiple copies allowed.
- Only one borrower per book copy at a time.
- Borrow/return operations must follow ownership rules.

---

2. Architecture Overview

- **Controller Layer:** Receives HTTP requests, validates input.
- **Service Layer:** Implements business rules.
- **Repository Layer:** Database access via Spring Data JPA.
- **Database:** H2 (dev/testing), MySQL (production).
- **Validation:** Jakarta Bean Validation & custom exceptions.

---

3. Technologies

- Java 17+
- Spring Boot, Spring Data JPA (Hibernate)
- Lombok, Jakarta Validation API
- Databases: H2 / MySQL
- JUnit 5, Mockito, JaCoCo (test coverage)
- Docker, Kubernetes

---

4. API Base Path

Base path: `/api` \ Example: `http://localhost:8080/api`

---

5. API Endpoints

Books

Register a Book

- **POST** `/api/books`
- **Request:**

```
{
  "title": "Chair",
  "author": "Da 'neil oman",
  "isbn": "96225"
}
```

- **Response:**

```
{
  "id": 1,
  "isbn": "96225",
  "title": "Chair",
  "author": "Da 'neil oman"
}
```

Get All Books

- **GET** `/api/books`
- **Response:**

```
[
    {
        "id": 1,
        "isbn": "96225",
        "title": "Chair",
        "author": "Da 'neil oman"
    }
]
```

Borrowers

Register a Borrower

- **POST** `/api/borrowers`
- **Request:**

```
{
  "name": "test",
  "email": "test@example.com"
}
```

- **Response:**

```
{
  "id": 1,
  "name": "test",
  "email": "test@example.com"
}
```

Get All Borrowers

- **GET** `/api/borrowers`
- **Response:**

```
[
    {
        "id": 1,
        "name": "test",
        "email": "test@example.com",
        "books": []
    }
]
```

Get Borrower by ID

- **GET** `/api/borrowers/{id}`
- **Response:**

```
{
    "id": 1,
    "name": "test",
    "email": "test@example.com",
    "books": []
}
```

Borrow a Book

- **POST** `/api/borrowers/{borrowerId}/borrow/{bookId}`
- **Response:**

```
{
    "id": 1,
    "name": "test",
    "email": "test@example.com",
    "message": "Book Borrowed Successfully",
    "bookResponse": {
        "id": 1,
        "isbn": "96225",
        "title": "Chair",
        "author": "Da 'neil oman"
    }
}
```

Return a Book

- **POST** `/api/borrowers/{borrowerId}/return/{bookId}`
- **Response:**

```
{
    "id": 1,
    "name": "test",
    "email": "test@example.com",
    "message": "Book Returned Successfully",
    "bookResponse": {
        "id": 1,
        "isbn": "96225",
        "title": "Chair",
        "author": "Da 'neil oman"
    }
}
```

---

6. Exception Handling

Exception	HTTP Status	Description
DuplicateIsbnException	422	ISBN exists with different title/author
BookAlreadyBorrowedException	422	Book already borrowed
BookAlreadyReturnedException	422	Book already returned
BookNotBorrowedException	422	Borrower tries to return a book not borrowed
DuplicateBorrowerException	422	Borrower email already exists
ResourceNotFoundException	404	Borrower or book not found
ConflictException	409	Generic conflict

---

7. Running Locally

```
git clone <repository-url>
cd library-api
mvn clean install
mvn spring-boot:run
```

Access: `http://localhost:8080/api`

---

8. Running with Docker

```
docker compose up --build
```

---

9. Running with Kubernetes

```
docker build -t library-api:latest .
docker tag library-api:latest <docker-repo>/library-api:latest
docker push <docker-repo>/library-api:latest

kubectl apply -f k8s/mysql-deployment.yaml
kubectl apply -f k8s/mysql-service.yaml
kubectl wait --for=condition=ready pod -l app=mysql --timeout=120s

kubectl apply -f k8s/deployment.yaml
kubectl apply -f k8s/service.yaml
kubectl rollout status deployment/library-api
```

---

10. Unit Tests & Coverage

```
mvn test
mvn jacoco:report
```

Open coverage report: `target/site/jacoco/index.html`

---

11. Using Postman

1. Open Postman.
2. Import `Library API.postman_collection.json`.
3. Set `baseUrl` (e.g., `http://localhost:8080/api`).

4. Send individual requests or run the collection.

---

12. Testing the API

- Use Postman to test each endpoint.
- Validate request and response bodies as per examples above.
- Ensure proper HTTP status codes are returned for exceptions.
- Test borrow and return operations, including edge cases.

---

13. Assumptions

- Book availability is tied to the unique book ID, not ISBN.
- Borrow and return endpoints are idempotent.
- Default profile uses H2; production uses MySQL via `prod` profile.
- Multiple books with same ISBN are considered separate copies.
- Only one borrower per book ID at a time.