Library API - Technical Documentation

A Spring Boot RESTful API for managing library borrowers and books, with support for borrowing and returning operations. It includes input validation, business rule enforcement, concurrency control, and clear error handling.

---

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 rules: 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).
- **Concurrency Control:** JPA `@Version` field (optimistic locking).
- **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/books`

---

5. API Endpoints

Books

- **POST** `/api/books` → Add new book
- **GET** `/api/books` → List all books

Borrowers

- **POST** `/api/borrowers` → Register borrower
- **GET** `/api/borrowers` → List borrowers
- **GET** `/api/borrowers/{id}` → Get borrower details
- **POST** `/api/borrowers/{borrowerId}/borrow/{bookId}` → Borrow a book
- **POST** `/api/borrowers/{borrowerId}/return/{bookId}` → Return a book

---

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
ResourceNotFoundException	404	Borrower or book not found

---

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. Assumptions

- Book availability is tied to the unique book ID, not ISBN.
- Borrow and return endpoints are designed to be idempotent.
- Default profile uses H2; production uses MySQL via `prod` profile.