

Library Management System

Project Documentation

<https://github.com/sirdashasan/library-management>

Prepared by: Hasan Sırdaş



May 2025

[***Click here to view the full Screenshots section. To jump to a specific screenshot, select one from the list below.***](#)

[Backend Launch – IntelliJ IDEA](#)

[Frontend Launch – VSCode](#)

[Docker Environment – Docker Desktop](#)

[Database Monitoring – pgAdmin 4](#)

[Live Application – React UI \(Frontend View\)](#)

[All Endpoints \(Postman&React UI\)](#)

[API Documentation – Swagger UI](#)

[Unit Tests – Service Layer](#)

[Integration Tests – Controller Layer](#)

[Test Coverage Report](#)

[Global Exception Handling – Error Responses](#)

[DTO Validation – Invalid Request Samples](#)

****Documentation Shortcuts****

[Technologies Used & Installation & Configuration](#)

[System Architecture](#)

[ER Diagram](#)

[Package Structure & Main Classes](#)

[Role-Based Access Control Matrix](#)

[Class Overview](#)

[Testing Overview](#)

Introduction

This project is a Library Management System developed using Java and Spring Boot. It is built using a monolithic architecture with layered design principles. The application allows librarians to manage books, users, borrow histories and overdue situations, while patrons can search, borrow and return books. Key features include role-based access control, JWT-based authentication and exception handling implemented with a global error response structure. The system also provides reactive book availability updates using Spring WebFlux and is containerized via Docker for easy deployment. Additionally, the project is documented using Swagger (OpenAPI) and includes both unit and integration tests to ensure functionality and reliability.

Technologies Used

Languages & Frameworks: Java 21, Spring Boot, Spring MVC, Spring Data JPA, Spring Security, Spring WebFlux

Security & Authentication: JWT, BCrypt, Role-Based Access Control

Database: PostgreSQL, H2 (for testing)

API Documentation: Swagger UI, OpenAPI 3

Testing: JUnit 5, Mockito, Reactor Test, Spring Security Test

Build Tool: Maven

Utilities: Lombok, dotenv-java

Deployment: Docker, Docker Compose

Frontend (Demo): React, JavaScript, Context API, Axios, Tailwind CSS

Installation

- **Clone the project repository**

```
git clone https://github.com/sirdashasan/library-management.git
```

- **Install the required dependencies and build the Project**

```
mvn clean package -DskipTests
```

- **Start the application using Docker Compose**

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

After the containers start successfully, the following resources will be available:

- **Backend API Root:** <http://localhost:8080/library/api>
- **Swagger UI:** <http://localhost:8080/library/api/swagger-ui/index.html>
- **Reactive Book Availability Stream (SSE):** <http://localhost:8080/library/api/books/availability-stream>

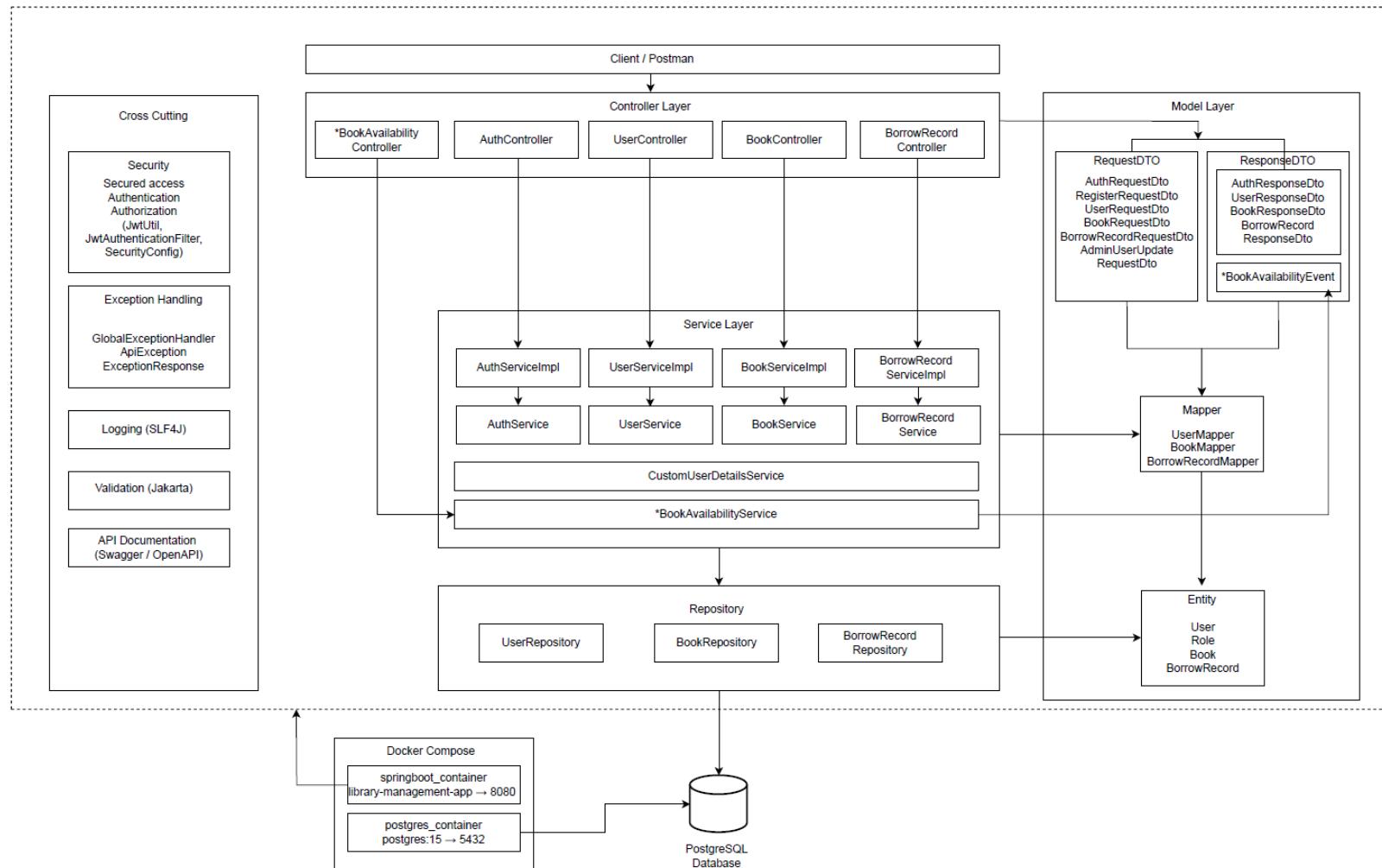
Configuration

- **application.properties:** Used for configuring the server port, context path, PostgreSQL connection, Swagger, JWT settings and other core application properties.

- **application.properties (test):** Contains in-memory H2 database settings and a separate JWT secret used for testing purposes.
- **.env file:** Stores sensitive environment variables such as DB_USERNAME, DB_PASSWORD, and JWT_SECRET. These values are not hardcoded and are injected securely via Docker Compose during runtime. **These values must be provided by the developer or deployment environment.**

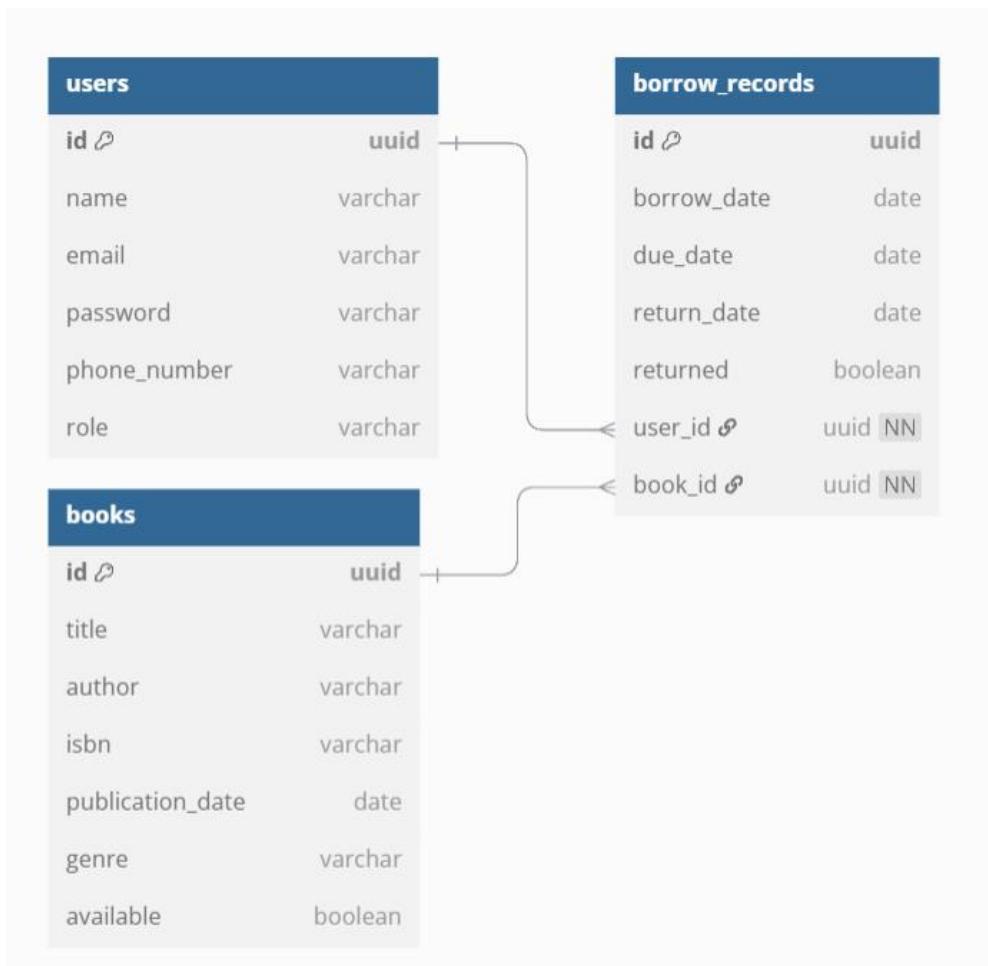
System Architecture

The application follows a layered monolithic architecture, separating concerns across controller, service, repository and model layers. The system also includes cross-cutting components such as security, validation, logging and exception handling, along with a reactive stream for book availability updates.



ER Diagram

The Entity Relationship Diagram (ERD) illustrates the core domain structure of the system, consisting of three main entities: User, Book and BorrowRecord. Each borrow record links a user and a book, forming many-to-one relationships to both User and Book entities.



Package Structure & Main Classes

The project follows a clean, layered package structure based on domain-driven principles.

com.hasan.library_management

```
├── config
│   ├── DataInitializer
│   ├── OpenApiConfig
│   └── SecurityConfig
├── controller
│   ├── AuthController
│   ├── BookAvailabilityController
│   ├── BookController
│   ├── BorrowRecordController
│   └── UserController
├── dto
│   ├── request
│   │   ├── AdminUserUpdateRequestDto
│   │   ├── AuthRequest
│   │   ├── BookRequestDto
│   │   ├── BorrowRecordRequestDto
│   │   ├── RegisterRequest
│   │   ├── UserRequestDto
│   ├── response
│   │   ├── AuthResponse
│   │   ├── BookAvailabilityEvent
│   │   ├── BookResponseDto
│   │   ├── BorrowRecordResponseDto
│   │   ├── UserResponseDto
└── entity
    └── Book
```

```
|   ├── BorrowRecord
|   ├── Role
|   ├── User
|
|   └── exceptions
|       ├── ApiException
|       ├── ErrorResponse
|       └── GlobalExceptionHandler
|
└── mapper
    ├── BookMapper
    ├── BorrowRecordMapper
    └── UserMapper
|
└── repository
    ├── BookRepository
    ├── BorrowRecordRepository
    └── UserRepository
|
└── security
    ├── JwtAuthenticationFilter
    └── JwtUtil
|
└── service
    ├── impl
    |   ├── AuthServiceImpl
    |   ├── BookServiceImpl
    |   ├── BorrowRecordServiceImpl
    |   ├── UserServiceImpl
    |
    |   ├── AuthService
    |   ├── BookAvailabilityService
    |   ├── BookService
    |   ├── BorrowRecordService
    |   ├── CustomUserDetailsService
    |   └── UserService
```

Role-Based Access Control Matrix

The table below summarizes which features are accessible by each user role in the system: Librarian and Patron. It also maps each feature to the corresponding backend API endpoint, helping to understand how role-based authorization is enforced across the application. Green check marks (✓) indicate access is granted, while red crosses (✗) indicate access is restricted.

Feature	Librarian	Patron	Endpoint	Module
View all users	✓	✗	GET /users	User
View user by ID	✓	✗	GET /users/{id}	User
View own user details	✓	✓	GET /users/me	User
Register	✓	✓	POST /auth/register	User
Login	✓	✓	POST /auth/login	User
Update any user	✓	✗	PUT /users/{id}	User
Delete a user	✓	✗	DELETE /users/{id}	User
View book list & details	✓	✓	GET /books	Book
Search books by ID	✓	✓	GET /books/{id}	Book
Search books by title (pagination)	✓	✓	GET /books/search/title	Book
Search books by author (pagination)	✓	✓	GET /books/search/author	Book
Search books by isbn (pagination)	✓	✓	GET /books/search/isbn	Book
Search books by genre (pagination)	✓	✓	GET /books/search/genre	Book
Add a book	✓	✗	POST /books	Book
Update a book	✓	✗	PUT /books/{id}	Book
Delete a book	✓	✗	DELETE /books/{id}	Book
View all borrow histories	✓	✗	GET /borrow-records	Borrow Record
View borrow record by user ID	✓	✗	GET /borrow-records/user/{userId}	Borrow Record
View own borrow records	✓	✓	GET /borrow-records/me	Borrow Record
View overdue records	✓	✗	GET /borrow-records/overdue	Borrow Record
Borrow a book	✓	✓	POST /borrow-records	Borrow Record
Return a book	✓	✓	PUT /borrow-records/return/{id}	Borrow Record
Report overdue	✓	✗	GET /borrow-records/overdue/report	Borrow Record
Book availability stream	✓	✓	GET /books/availability-stream	Book (Reactive)

Note: All endpoints listed above are relative to the base path defined in the configuration:
server.servlet.context-path=/library/api

For example, GET /users maps to GET <http://localhost:8080/library/api/users>.

Class Overview

Entity

- **User:** Represents the users table. Includes fields such as id, name, email, password, phoneNumber and role. It has a one-to-many relationship with BorrowRecord.
- **Book:** Represents the books table. Contains fields like id, title, author, isbn, publicationDate, genre and available. It has a one-to-many relationship with BorrowRecord.
- **BorrowRecord:** Represents the borrow_records table. Includes id, borrowDate, dueDate, returnDate and returned. It has many-to-one relationships with both User and Book.
- **Role (Enum):** Defines application roles as LIBRARIAN and PATRON. Used for role-based access control within the application, particularly in User entity and SecurityConfig.

DTO – Request

- **UserRequestDto:** Used for creating a new user or during user registration. Contains name, email, password, phoneNumber and role fields. Includes validation annotations for input constraints.
- **RegisterRequest:** Specifically designed for the auth/register endpoint. Similar to UserRequestDto, it includes name, email, password, phoneNumber and role, with appropriate validation applied.
- **AdminUserUpdateRequestDto:** Used by librarians (with LIBRARIAN role) to update other users information. Unlike UserRequestDto, it does not include the password field. Input validation is enforced through annotations.
- **AuthRequest:** Used for user login (auth/login endpoint). Contains only email and password. Validation is applied for both fields.
- **BookRequestDto:** Handles both creation and update operations for books. Includes fields like title, author, isbn, publicationDate and genre. Validation constraints are enforced to ensure data integrity.
- **BorrowRecordRequestDto:** Represents the request data for borrowing a book. Includes userId, bookId, borrowDate and dueDate. All fields are validated using standard validation annotations.

DTO – Response

- **UserResponseDto:** Represents the public-facing user data returned in API responses. Does not include sensitive fields such as passwords. Contains id, name, email, phoneNumber and role.
- **AuthResponse:** Returned after a successful registration or login operation. Includes a JWT token used for authenticated requests.
- **BookResponseDto:** Used in book-related API responses. Includes id, title, author, isbn, publicationDate, genre and available status.
- **BorrowRecordResponseDto:** Returned in borrowing history and related operations. Contains detailed borrowing data such as bookTitle, userName, id, userId, bookId, borrowDate, dueDate, returnDate and returned.
- **BookAvailabilityEvent:** Represents real-time book availability updates emitted by BookAvailabilityService. It is used in reactive streams (WebFlux) and excluded from Swagger documentation to avoid conflicts with MVC-based endpoints.

Mapper

- **UserMapper:** Handles the conversion between User entities and their corresponding DTOs. Includes methods like toEntity, toResponseDto and updateEntityFromAdminDto for admin-level user updates.
- **BookMapper:** Responsible for converting between Book entities and DTOs. Provides toEntity, toResponseDto and updateEntity methods to support both creation and update operations.
- **BorrowRecordMapper:** Maps between BorrowRecord entities and DTOs. Includes toEntity and toResponseDto methods to handle conversion during borrowing and return operations.

Repository

- **UserRepository:** Extends JpaRepository<User, UUID> to provide basic CRUD operations. Includes a custom method “findByEmail(String email)” used for authentication and user lookup.
- **BookRepository:** Extends JpaRepository<Book, UUID>, enabling standard CRUD functionality. Contains custom search methods for pagination and filtering:
 - findByTitleContainingIgnoreCase
 - findByAuthorContainingIgnoreCase
 - findByIsbnContainingIgnoreCase
 - findByGenreContainingIgnoreCase
 - Also includes findByIsbn to ensure uniqueness of ISBN during book creation.

- **BorrowRecordRepository:** Extends JpaRepository<BorrowRecord, UUID> and supports advanced queries related to borrowing logic. Custom methods include:

- findByUserId
- findByBookId
- findByReturnedFalse
- findByReturnedFalseAndDueDateBefore
- countByUserIdAndReturnedFalse
- existsByUserIdAndReturnedFalseAndDueDateBefore

These queries are used to handle borrowing history, overdue checks and borrowing limits.

Service

- **UserService (interface):** Defines the contract for user-related operations such as getAllUsers, getUserId, getOwnUserDetails, updateUser and deleteUser.
- **AuthService (interface):** Abstracts the authentication-related operations, including register() and login() methods.
- **BookService (interface):** Provides book-related business logic such as getAllBooks, getBookById, create, update and delete. Also includes search functionality using methods like searchByTitle, searchByAuthor, searchByIsbn and searchByGenre.
- **BorrowRecordService (interface):** Handles borrowing logic, including getAllBorrowRecords, borrowBook, returnBook, getBorrowRecordsByUserId, getOwnBorrowRecords, getOverdueRecords and generateOverdueReport.
- **BookAvailabilityService:** Enables real-time broadcasting of book availability changes (e.g., when a book is borrowed or returned). Utilizes Sinks.Many from Spring WebFlux to implement a publish–subscribe model:
 - sink.tryEmitNext(...) → emits a new BookAvailabilityEvent
 - getStream() → returns a Flux<BookAvailabilityEvent> to stream availability updates
- **CustomUserDetailsService:**
Implements Spring Security's UserDetailsService interface.
Responsible for loading user details based on the email found in the JWT token (loadUserByUsername).
Maps user roles to Spring Security authorities in the ROLE_... format, used during authentication.

Service – Impl

- **UserServiceImpl:** Implements the UserService interface and handles user-related business logic. Uses UserRepository for database access and UserMapper for converting between entities and DTOs. Throws ApiException in exceptional scenarios and uses SLF4J for logging.
- **AuthServiceImpl:**
 - **register:** Checks for existing email, saves the new user, encodes the password, and generates a JWT token.
 - **login:** Authenticates the user via AuthenticationManager and returns a JWT token upon success.

Throws ApiException on failure and logs non-sensitive information (e.g., email) using SLF4J.
- **BookServiceImpl:** Implements the BookService interface. Checks for ISBN duplication before adding a new book. Uses BookMapper.updateEntity() during book updates. All search methods return pageable results. Combines BookRepository and BookMapper in all operations. Logs operations via SLF4J and throws ApiException in case of errors.
- **BorrowRecordServiceImpl:** Implements the BorrowRecordService interface. Integrates with BookAvailabilityService to emit real-time book availability updates. Contains checkUserEligibility() method to enforce rules such as a maximum of 5 active borrowings and no overdue books. Users violating these conditions cannot borrow new books. The generateOverdueReport() method creates a detailed text report of overdue records, logs it to the console, and writes it to overdue_report.txt. Uses SLF4J for logging and throws ApiException in case of exceptions.

Controller

- **UserController:** Manages user-related endpoints with detailed Swagger/OpenAPI documentation and response codes.
 - GET /users → Returns all users
 - GET /users/{id} → Returns a specific user by ID
 - GET /users/me → Returns the authenticated user's own profile using JWT
 - PUT /users/{id} → Updates a user
 - DELETE /users/{id} → Deletes a user
- **AuthController:** Handles authentication endpoints with proper Swagger/OpenAPI annotations.
 - POST /auth/register → Registers a new user and returns a JWT token
 - POST /auth/login → Authenticates user credentials and returns a JWT token
- **BookController:** Provides endpoints for managing book resources and includes search functionalities.
 - GET /books → Retrieves all books
 - GET /books/{id} → Retrieves a book by ID
 - POST /books → Adds a new book

- PUT /books/{id} → Updates book information
 - DELETE /books/{id} → Deletes a book
 - GET /books/search/{field} → Supports search by title, author, ISBN or genre with pagination
- **BorrowRecordController:** Manages borrow/return operations with role-based access and Swagger documentation.
 - GET /borrow-records → Returns all borrow records
 - POST /borrow-records → Creates a new borrow record
 - PUT /borrow-records/return/{id} → Marks a book as returned
 - GET /borrow-records/user/{userId} → Returns borrow records of a specific user
 - GET /borrow-records/me → Returns the authenticated user's borrow history
 - GET /borrow-records/overdue → Returns a list of overdue borrow records
 - GET /borrow-records/overdue/report → Generates a text-based overdue report, logs it, and saves it as a file
- **BookAvailabilityController:** Provides a reactive streaming endpoint using Server-Sent Events (SSE).
 - GET /books/availability-stream

This endpoint streams real-time book availability changes.
Triggered inside BorrowRecordServiceImpl.borrowBook() and returnBook() after changing book status via book.setAvailable(...).

Security

- **JwtUtil:** Responsible for generating, validating and extracting information from JWT tokens. The token includes the user's email and role as claims. Key methods include:
 - generateToken(...)
 - validateToken(...)
 - getEmailFromToken(...)
 - getRoleFromToken(...)
- **JwtAuthenticationFilter:** Intercepts every HTTP request to check for a valid JWT token in the Authorization header (format: Bearer <token>). If the token is valid, the associated user details are loaded and set into the Spring Security SecurityContext. This filter runs before the UsernamePasswordAuthenticationFilter in the Spring Security filter chain.

Config

- **SecurityConfig:** Contains the full Spring Security configuration for the application. Defines public endpoints, role-based access rules, and integrates the custom JwtAuthenticationFilter into the filter chain. Includes core security settings such as CORS, CSRF, session management policy, and password encryption using BCrypt.

- **OpenApiConfig:** Configures Swagger/OpenAPI documentation for the project. Also sets up a JWT-based security scheme for authorized API testing through the Swagger UI.
- **DataInitializer:** Loads demo data when the application starts. Primarily used to speed up development and testing processes by populating the system with initial users and books.

Exceptions

- **ApiException:** Custom runtime exception that allows throwing application-specific errors with custom HTTP status codes (e.g., NOT_FOUND, BAD_REQUEST). Typically used for domain-level validations.
- **ErrorResponse:** Data transfer object (DTO) that represents structured error information including message, status code and timestamp. Used internally to build uniform error responses.
- **GlobalExceptionHandler:** Centralized exception handling class annotated with @ControllerAdvice. Handles and formats various exceptions including:
 - ApiException for custom domain errors
 - MethodArgumentNotValidException for validation failures
 - HttpMessageNotReadableException for malformed JSON input
 - BadCredentialsException for invalid login attempts
 - Generic Exception as a fallback for unexpected errors

Testing Overview

CustomUserDetailsServiceTest (Unit Test)

This test class verifies the behavior of the loadUserByUsername(...) method in the CustomUserDetailsService class.

- Uses @ExtendWith(MockitoExtension.class) to enable Mockito-based unit testing.
- The UserRepository dependency is mocked using @Mock.
- The CustomUserDetailsService is injected with its dependencies using @InjectMocks.

Test Methods:

- **loadUserByUsername_shouldReturnUserDetails_whenUserExists()**
Verifies that the correct UserDetails object is returned when a user is found by email. Also checks that the email, password and ROLE_PATRON authority are included as expected.
- **loadUserByUsername_shouldThrowException_whenUserNotFound()**
Ensures that a UsernameNotFoundException is thrown when no user is found for the given email. Verifies that the exception message matches the expected output.

BookAvailabilityServiceTest (Unit Test)

This test class verifies the behavior of the publishAvailabilityChange(...) method in the BookAvailabilityService class.

- Uses StepVerifier from the Reactor Test library to test the reactive stream.
- Ensures that emitted events are correctly published and can be consumed through the getStream() method.

Test Method:

- **publishAvailabilityChange_shouldEmitEvent()**
Verifies that when publishAvailabilityChange(...) is called, a corresponding BookAvailabilityEvent is emitted through the stream.
Asserts that the emitted event contains the expected bookId and available values.

UserServiceImplTest (Unit Test)

This test class verifies the behavior of user-related methods in the UserServiceImpl class.

- UserRepository and UserMapper are mocked using @Mock.
- Mocks are injected into the service class using @InjectMocks.

Test Methods:

- **getAllUsers_shouldReturnUserList()**
Ensures the correct number and order of user DTOs are returned when listing all users.

- **getUserById_shouldReturnUser_whenExists()**
Verifies that a valid user ID returns the correct user DTO.
- **getUserById_shouldThrowException_whenUserNotFound()**
Asserts that an ApiException is thrown when the user is not found for the given ID.
- **getOwnUserDetails_shouldReturnUser_whenExists()**
Verifies that a user is correctly retrieved and mapped by email.
- **getOwnUserDetails_shouldThrowException_whenUserNotFound()**
Ensures that an ApiException is thrown when no user is found for the given email.
- **updateUser_shouldUpdateUser_whenExists()**
Checks that an existing user is successfully updated and the correct DTO is returned.
- **updateUser_shouldThrowException_whenUserNotFound()**
Asserts that an ApiException is thrown when attempting to update a non-existent user.
- **deleteUser_shouldDeleteUser_whenExists()**
Ensures that a valid user is deleted successfully without throwing an exception.
- **deleteUser_shouldThrowException_whenUserNotFound()**
Verifies that an ApiException with the correct message is thrown when trying to delete a non-existent user.

BookServiceImplTest (Unit Test)

This test class verifies the correct behavior of methods in the BookServiceImpl class.

- BookRepository and BookMapper are mocked using @Mock.
- A real instance of BookServiceImpl is injected with mocked dependencies using @InjectMocks.

Test Methods:

- **getAllBooks_shouldReturnBookList()**
Ensures that all books in the database are correctly retrieved and returned as a list of DTOs.
- **getBookById_shouldReturnBook_whenExists()**
Verifies that a valid book ID returns the correct BookResponseDto.
- **getBookById_shouldThrowException_whenNotFound()**
Asserts that an ApiException is thrown when the book is not found.
- **createBook_shouldCreate_whenIsbnNotExists()**
Checks that a book is successfully created when the provided ISBN is not already in the database.
- **createBook_shouldThrowException_whenIsbnAlreadyExists()**
Ensures that an ApiException is thrown if a book with the same ISBN already exists.
- **updateBook_shouldUpdateBook_whenExists()**
Verifies that an existing book is updated successfully.

- **updateBook_shouldThrowException_whenBookNotFound()**
Asserts that an ApiException is thrown when trying to update a non-existent book.
- **deleteBook_shouldDeleteBook_whenExists()**
Ensures that a book is deleted successfully if it exists.
- **deleteBook_shouldThrowException_whenBookNotFound()**
Verifies that an ApiException is thrown when trying to delete a book that does not exist.
- **searchByTitle_shouldReturnPagedBooks()**
Ensures that the system returns paginated books filtered by title correctly.
- **searchByAuthor_shouldReturnPagedBooks()**
Verifies correct pagination and filtering when searching books by author name.
- **searchByIsbn_shouldReturnPagedBooks()**
Confirms that searching by ISBN returns the appropriate book(s).
- **searchByGenre_shouldReturnPagedBooks()**
Validates that filtering by genre works correctly and returns a paginated response.

BorrowRecordServiceImplTest (Unit Test)

This test class verifies the correctness of the borrowing and return logic in the BorrowRecordServiceImpl class.

- Dependencies such as BookRepository, UserRepository, BorrowRecordRepository, BookAvailabilityService, and BorrowRecordMapper are mocked using @Mock.
- A real instance of BorrowRecordServiceImpl is injected with these mocks using @InjectMocks.

Test Methods:

- **getAll_shouldReturnBorrowRecords()**
Ensures that all borrow records are retrieved correctly as a list.
- **borrowBook_shouldCreateBorrowRecord_whenValidRequest()**
Verifies that a borrow record is created when a valid request is made, and a BookAvailabilityEvent is emitted.
- **borrowBook_shouldThrowException_whenBookNotFound()**
Asserts that an ApiException is thrown if the requested book does not exist.
- **borrowBook_shouldThrowException_whenUserNotFound()**
Ensures an ApiException is thrown with the correct message if the user is not found.
- **borrowBook_shouldThrowException_whenBookNotAvailable()**
Verifies that an error is thrown if the book is already borrowed and unavailable.
- **borrowBook_shouldThrowException_whenUserHasAlreadyBorrowed5Books()**
Ensures that a user cannot borrow more than 5 books concurrently; an exception is thrown if the limit is exceeded.

- **borrowBook_shouldThrowException_whenUserHasOverdueBooks()**
Verifies that borrowing is blocked if the user has overdue books.
- **returnBook_shouldMarkAsReturned_whenValidRequest()**
Ensures that the borrow record is marked as returned and the book is made available again.
- **returnBook_shouldThrowException_whenRecordNotFound()**
Asserts that an ApiException is thrown if the return request references a non-existent record.
- **returnBook_shouldThrowException_whenAlreadyReturned()**
Verifies that an exception is thrown when attempting to return a book that has already been returned.
- **getBorrowRecordsByUserId_shouldReturnRecords_whenUserExists()**
Ensures that a user's borrow records are returned correctly when the user exists.
- **getBorrowRecordsByUserId_shouldThrowException_whenUserNotFound()**
Asserts that an ApiException is thrown with the correct message if the user does not exist.
- **getOwnBorrowRecords_shouldReturnRecords_whenUserExists()**
Verifies that borrow records are correctly returned based on the email extracted from the JWT token.
- **getOwnBorrowRecords_shouldThrowException_whenUserNotFound()**
Ensures that an ApiException is thrown if the user referenced in the token is not found.
- **getOverdueRecords_shouldReturnOverdueRecords()**
Ensures that overdue records are correctly filtered and returned.
- **generateOverdueReport_shouldReturnFormattedText_whenOverdueExists()**
Verifies that a properly formatted report of overdue records is generated, logged to the console, and saved as a .txt file.

AuthServiceImplTest (Unit Test)

This test class verifies the correct behavior of register and login operations in the AuthServiceImpl class under both success and failure scenarios.

- Dependencies such as UserRepository, PasswordEncoder, JwtUtil, and AuthenticationManager are mocked using @Mock.
- These mocks are injected into a real instance of AuthServiceImpl using @InjectMocks.

Test Methods:

- **register_shouldReturnToken_whenEmailIsNotRegistered()**
Verifies that when the provided email is not already registered, the user is successfully created and a JWT token is returned.
- **register_shouldThrowException_whenEmailAlreadyExists()**
Ensures that an ApiException is thrown if the email is already in use.

- **login_shouldReturnToken_whenCredentialsAreValid()**
Verifies that a valid login request results in a successful authentication and a JWT token is returned.
- **login_shouldThrowException_whenAuthenticationFails()**
Asserts that a BadCredentialsException is thrown by Spring Security when authentication fails.
- **login_shouldThrowException_whenUserNotFound()**
Ensures that an ApiException with an appropriate message is thrown if the user is not found in the database, even after successful authentication.

UserControllerTest (Integration Test)

This test class validates the integration behavior of the /users endpoints, focusing on access control, authorization based on user roles, and expected HTTP responses.

- Uses @SpringBootTest and @AutoConfigureMockMvc to load the full application context and test the API layer in a realistic environment.
- JWT tokens are generated via register/login requests and used to simulate authenticated access.
- Both LIBRARIAN and PATRON roles are tested for their access rights.

Test Methods:

- **getAll_shouldReturnOk**
Confirms that a user with a LIBRARIAN token can successfully retrieve the list of users.
- **getAll_shouldReturnForbidden_whenNoToken**
Verifies that access is denied when no token is provided.
- **getAll_shouldReturnForbidden_whenUserIsPatron**
Ensures that a PATRON role cannot access the /users list.
- **getUserById_shouldReturnUser_whenAuthorizedAsLibrarian**
Confirms that a LIBRARIAN can retrieve a specific user's information by ID.
- **getUserById_shouldReturnForbidden_whenUserIsPatron**
Asserts that a PATRON cannot access another user's data.
- **getUserById_shouldReturnNotFound_whenUserDoesNotExist**
Verifies that a 404 is returned when requesting a non-existent user ID.
- **getOwnDetails_shouldReturnOk_whenAuthorized**
Confirms that an authenticated user can retrieve their own profile using a valid token.
- **getOwnDetails_shouldReturnForbidden_whenNoToken**
Verifies that access is denied if no token is provided.
- **getOwnDetails_shouldReturnUnauthorized_whenTokenIsInvalid**
Asserts that an invalid token results in a 401 Unauthorized response.

- **updateUser_shouldUpdate_whenLibrarianAndValidData**
Confirms that a LIBRARIAN can successfully update a user's information with valid input.
- **updateUser_shouldReturnForbidden_whenUserIsPatron**
Ensures that a PATRON cannot update another user's data.
- **updateUser_shouldReturnNotFound_whenUserDoesNotExist**
Verifies that updating a non-existent user returns a 404 response.
- **updateUser_shouldReturnForbidden_whenNoToken**
Asserts that an update operation is rejected when no token is provided.
- **deleteUser_shouldDelete_whenLibrarianAndValidId**
Confirms that a LIBRARIAN can successfully delete another user.
- **deleteUser_shouldReturnForbidden_whenUserIsPatron**
Ensures that a PATRON cannot delete another user.
- **deleteUser_shouldReturnNotFound_whenUserDoesNotExist**
Verifies that deleting a non-existent user returns a 404 response.
- **deleteUser_shouldReturnForbidden_whenNoToken**
Asserts that deletion is denied when no token is provided.

BookControllerTest (Integration Test)

This test class ensures the correctness of all /books endpoints, including CRUD operations and search functionality.

It verifies JWT-based authorization, validation rules, and various error scenarios using full end-to-end tests.

- Built using @SpringBootTest and MockMvc to simulate real HTTP requests within a fully initialized Spring application context.
- Covers all basic CRUD operations and search operations by title, author, isbn, and genre.

Test Methods:

- **getAllBooks_shouldReturnOk_whenAuthorized()**
Verifies that books are successfully listed when a valid token is provided.
- **getBookById_shouldReturnOk_whenBookExistsAndAuthorized()**
Ensures that a created book can be fetched correctly by its ID.
- **getBookById_shouldReturnNotFound_whenBookDoesNotExist()**
Verifies that a 404 status is returned for a non-existent book ID.
- **createBook_shouldReturnOk_whenValidRequestAndAuthorized()**
Confirms that a book is added successfully when all required fields are provided with a valid token.
- **createBook_shouldReturnForbidden_whenNoTokenProvided()**
Ensures that a 403 Forbidden response is returned when trying to create a book without authentication.

- **createBook_shouldReturnBadRequest_whenMissingRequiredFields()**
Verifies that the API returns a 400 Bad Request when required fields are missing.
- **updateBook_shouldUpdate_whenBookExistsAndValidRequest()**
Confirms that an existing book is successfully updated with valid input.
- **updateBook_shouldReturnNotFound_whenBookDoesNotExist()**
Ensures a 404 is returned when attempting to update a non-existent book.
- **updateBook_shouldReturnBadRequest_whenValidationFails()**
Asserts that the API responds with 400 if invalid data (e.g., empty title) is submitted.
- **deleteBook_shouldDelete_whenBookExistsAndAuthorized()**
Verifies that a book can be deleted and is no longer retrievable afterward.
- **deleteBook_shouldReturnNotFound_whenBookDoesNotExist()**
Ensures that deleting a non-existent book returns a 404 response.
- **deleteBook_shouldReturnForbidden_whenNoToken()**
Confirms that deletion is denied when no JWT token is provided.
- **searchByTitle_shouldReturnResults_whenMatchingBooksExist()**
Validates that books matching the given title are returned.
- **searchByTitle_shouldReturnEmpty_whenNoMatchesFound()**
Ensures that a search with a non-matching title returns an empty result.
- **searchByAuthor_shouldReturnResults_whenMatchingBooksExist()**
Verifies that the search by author name returns matching books.
- **searchByAuthor_shouldReturnEmpty_whenNoMatchesFound()**
Confirms that no results are returned for unmatched authors.
- **searchByIsbn_shouldReturnResults_whenMatchingBookExists()**
Validates that a matching ISBN returns the correct book.
- **searchByIsbn_shouldReturnEmpty_whenNoMatchFound()**
Ensures no results are returned when the ISBN does not match any records.
- **searchByGenre_shouldReturnResults_whenMatchingGenreExists()**
Verifies that books are correctly returned when searched by genre.
- **searchByGenre_shouldReturnEmpty_whenNoMatchFound()**
Confirms that a search by non-matching genre returns an empty result.

BorrowRecordControllerTest (Integration Test)

This test class verifies the accuracy and access control of all /borrow-records endpoints, including borrowing, returning, and overdue record management.

- Built using @SpringBootTest, MockMvc, and JWT-based authentication to perform full end-to-end integration testing.

Test Methods:

- **getAllBorrowRecords_shouldReturnOk_whenAuthorized()**
Ensures that an authorized user can retrieve all borrow records successfully.
- **getAllBorrowRecords_shouldReturnForbidden_whenNoToken()**
Verifies that access is denied when no token is provided.
- **borrowBook_shouldReturnOk_whenValidRequestAndAuthorized()**
Confirms that a valid borrow request results in successful borrowing.
- **borrowBook_shouldReturnForbidden_whenNoToken()**
Asserts that borrowing is forbidden without a JWT token.
- **borrowBook_shouldReturnBadRequest_whenDueDateIsNotFuture()**
Ensures a 400 Bad Request is returned when the due date is not in the future.
- **returnBook_shouldReturnOk_whenValidIdAndAuthorized()**
Verifies that a book can be returned successfully with a valid ID and token.
- **returnBook_shouldReturnNotFound_whenRecordDoesNotExist()**
Ensures a 404 Not Found is returned when attempting to return a non-existent record.
- **returnBook_shouldReturnForbidden_whenNoTokenProvided()**
Asserts that the return operation is blocked without authentication.
- **getRecordsByUserId_shouldReturnList_whenValidUserAndAuthorized()**
Confirms that a librarian can retrieve borrow records of a specific user.
- **getRecordsByUserId_shouldReturnEmptyList_whenNoRecordsFound()**
Ensures that an empty list is returned when the user has no borrow records.
- **getRecordsByUserId_shouldReturnForbidden_whenNoToken()**
Verifies that access to another user's records is denied without a token.
- **getOwnRecords_shouldReturnList_whenUserHasBorrowedBooks()**
Confirms that an authenticated user can access their own borrow records using /me.
- **getOwnRecords_shouldReturnEmptyList_whenNoRecordsExist()**
Verifies that /me returns an empty list when the user has no borrow history.
- **getOwnRecords_shouldReturnForbidden_whenNoTokenProvided()**
Ensures that access to /me is denied without authentication.
- **getOverdueRecords_shouldReturnList_whenOverdueExists()**
Verifies that overdue records are correctly returned for books past due and not returned.
- **getOverdueRecords_shouldReturnEmptyList_whenNoOverdue()**
Confirms that an empty list is returned when no overdue records exist.
- **getOverdueRecords_shouldReturnForbidden_whenNoTokenProvided()**
Ensures that listing overdue records is forbidden without authentication.
- **generateOverdueReport_shouldReturnOk_whenAuthorized()**
Confirms that a LIBRARIAN user can successfully generate the overdue report.

- **generateOverdueReport_shouldReturnUnauthorized_whenNoToken()**
Verifies that the report generation endpoint is inaccessible without a token.
- **generateOverdueReport_shouldReturnForbidden_whenNotLibrarian()**
Ensures that a PATRON user receives a 403 Forbidden when trying to access the report generation endpoint.

AuthControllerTest (Integration Test)

This test class verifies the authentication flow of the application, specifically the behavior of the register and login endpoints.

- Built using @SpringBootTest and MockMvc to perform realistic integration testing for user authentication.

Test Methods:

- **register_shouldReturnOk_whenEmailIsNew()**
Verifies that a new user can successfully register and receives a 200 OK response.
- **register_shouldReturnBadRequest_whenEmailAlreadyExists()**
Ensures that a 400 Bad Request is returned when attempting to register with an email that is already in use.
- **login_shouldReturnOk_whenCredentialsAreCorrect()**
Confirms that a user can log in successfully with valid email and password credentials.
- **login_shouldReturnUnauthorized_whenPasswordIsWrong()**
Verifies that a 401 Unauthorized response is returned when the password is incorrect.
- **login_shouldReturnUnauthorized_whenEmailDoesNotExist()**
Ensures that logging in with a non-existent email results in a 401 Unauthorized response.

BookAvailabilityControllerTest (Integration Test)

This test class verifies the real-time event streaming functionality of the /books/availability-stream endpoint, which is powered by the BookAvailabilityService.

- It is a **reactive integration test**, implemented using WebTestClient to test the WebFlux-enabled **Server-Sent Events (SSE)** endpoint.
- The endpoint emits BookAvailabilityEvent objects when books are borrowed or returned.

Test Method:

- **streamAvailability_shouldEmitEvents()**
 - Simulates the emission of a new BookAvailabilityEvent via a separate thread.
 - Subscribes to the /books/availability-stream endpoint using WebTestClient.
 - Verifies that:
 - The stream returns 200 OK.

- The emitted event is correctly delivered to the subscriber.
- The event payload includes the expected bookId and available fields with correct values.

Screenshots - Project Startup Screenshots

Backend Launch - IntelliJ IDEA

The screenshot shows the IntelliJ IDEA interface during the startup of a Spring Boot application. The left sidebar displays the project structure under 'library-management' with 'main' selected. The 'SecurityConfig.java' file is open in the main editor, showing configuration code for security filters. The bottom terminal window shows the application logs, indicating the startup of Tomcat and the successful start of the 'LibraryManagementApplication'. The status bar at the bottom right shows the date and time as 9.05.2025, 19:51.

```
public class SecurityConfig {
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(HttpMethod.GET, ...patterns: "/borrow-records/me").hasAnyRole(...roles: "PATRON")
                .requestMatchers(HttpMethod.POST, ...patterns: "/borrow-records/**").hasAnyRole(...roles: "PATRON")
                .requestMatchers(HttpMethod.PUT, ...patterns: "/borrow-records/return/**").hasAnyRole(...roles: "LIBRARIAN")
                .requestMatchers(HttpMethod.GET, ...patterns: "/borrow-records/user/**").hasRole("LIBRARIAN")
                .requestMatchers(HttpMethod.GET, ...patterns: "/borrow-records/overdue").hasRole("LIBRARIAN")
                .requestMatchers(HttpMethod.GET, ...patterns: "/borrow-records/overdue/report").hasRole("LIBRARIAN")
                .requestMatchers(HttpMethod.GET, ...patterns: "/borrow-records").hasRole("LIBRARIAN"))
            .anyRequest().authenticated()
        )
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);
    }
    return http.build();
}
```

Terminal output:

```
ity.web.savedrequest.RequestCacheAwareFilter@24caa80, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@3420547, org.springframework.security.web.authentication.AnonymousAuthenticationFilter@642b2176, org.springframework.security.web.session.SessionManagementFilter@1488d1c, org.springframework.security.web.access.ExceptionTranslationFilter@3c3a767, org.springframework.security.web.access.intercept.AuthorizationFilter@3941bdd7]
[boot] 2025-05-09T16:48:56.042Z INFO 1 --- [library-management] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/library/api'
[boot] 2025-05-09T16:48:56.052Z INFO 1 --- [library-management] [main] c.h.l.LibraryManagementApplication : Started LibraryManagementApplication in 5.305 seconds (process running for 5.873)
[boot] 2025-05-09T16:48:56.052Z INFO 1 --- [library-management] [main] c.h.l.LibraryManagementApplication : Mock data (users, books, and overdue borrow records) loaded successfully.
```

Bottom status bar:

63:8 CRLF UTF-8 4 spaces ↻

Windows Taskbar icons: Library - ..., Dosya Ge..., rol dağılı..., docker ko..., library-m..., pgAdmin 4, returnBo..., axiosInsta..., Container..., 26°C, 9.05.2025, 19:51

Frontend Launch – VSCode

The screenshot shows a Microsoft Visual Studio Code (VSCode) interface with the following details:

- File Explorer:** Shows the project structure under the folder "LIBRARY-FRONTEND". The "src" folder contains components like BookCard.jsx, BorrowHistoryPanel.jsx, ConfirmationModal.jsx, DetailsModal.jsx, ManageBooksPanelAdmin.jsx, ManageUsersPanelAdmin.jsx, MyBooksPanel.jsx, Navbar.jsx, OverdueBooksAdmin.jsx, RecentActivityBanner.jsx, RequireAdminRoute.jsx, Sidebar.jsx, and SidebarAdmin.jsx. It also includes constants, contexts (BookFilterContext.jsx), layouts (Layout.jsx), pages (AdminPage.jsx, BookServices.jsx, Home.jsx, Login.jsx, Profile.jsx, Register.jsx), and routes (AppRoutes.jsx).
- Editor:** The "axiosInstance.jsx" file is open in the editor. The code defines an axios instance with a baseURL set to "http://localhost:8080/library/api" or "VITE_API_BASE_URL" if available. It includes an interceptor for requests that adds a "Bearer" token from localStorage to the Authorization header.
- Terminal:** The terminal shows the output of the Vite development server. It says "VITE v6.3.4 ready in 735 ms" and provides local and network URLs, along with a help command.
- Bottom Status Bar:** Shows various icons and status information, including a GitHub pull request icon, a Docker icon, a pgAdmin 4 icon, a returnBot icon, a Container icon, a weather icon (26°C), and a date/time stamp (9.05.2025).

Docker Desktop

The screenshot shows the Docker Desktop application window. The left sidebar has a 'Containers' tab selected, showing three running containers: 'library-management', 'springboot_container', and 'postgres_container'. The main area displays container usage metrics and a table of running containers. A 'Walkthroughs' section is also visible.

Containers [Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage [i](#)
0.14% / 1600% (16 CPUs available)

Container memory usage [i](#)
298.91MB / 7.26GB

Show charts

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	library-management	-	-	-	0.14%	3 minutes ago	[] [...] []
<input type="checkbox"/>	springboot_container	c146a6a296c9	library-management-app	8080:8080	0.14%	3 minutes ago	[] [...] []
<input type="checkbox"/>	postgres_container	1dd5938f0832	postgres:15	5433:5432	0%	3 minutes ago	[] [...] []

Showing 3 items

Walkthroughs [X](#)

Multi-container applications [View more in the Learning center](#)

Containerize your application [View more in the Learning center](#)

Engine running | RAM 1.40 GB CPU 0.06% Disk: 15.18 GB used (limit 1006.85 GB) | Terminal ✓ v4.40.0

Windows Taskbar icons: Dosya Ge..., rol dağılı..., docker ko..., library-m..., pgAdmin 4, returnBo..., axiosInsta..., Container..., 26°C, 19:52, 9.05.2025

Database Monitoring – pgAdmin4

pgAdmin 4

File Object Tools Help

Object Explorer Dashboard X Properties X SQL X Statistics X

General System Statistics

Database sessions

Total Active Idle

Transactions per second

Transactions Commits Rollbacks

Tuples in

Inserts Updates Deletes

Tuples out

Fetched Returned

Block I/O

Reads Hits

Database activity

Sessions Locks Prepared Transactions

Active sessions only

Search

	PID	User	Application	Client	Backend start	Transaction start	State	Wait event	Blocking PIDs
✖️	68	postgr...	PostgreSQL JDBC Dri...	172.18.0.3	2025-05-09 16:48:53 ...		idle	Client: ClientRead	
✖️	69	postgr...	PostgreSQL JDBC Dri...	172.18.0.3	2025-05-09 16:48:54 ...		idle	Client: ClientRead	
✖️	70	postgr...	PostgreSQL JDBC Dri...	172.18.0.3	2025-05-09 16:48:54 ...		idle	Client: ClientRead	
✖️	71	postgr...	PostgreSQL JDBC Dri...	172.18.0.3	2025-05-09 16:48:54 ...		idle	Client: ClientRead	
✖️	72	postgr...	PostgreSQL JDBC Dri...	172.18.0.3	2025-05-09 16:48:54 ...		idle	Client: ClientRead	

Windows Taskbar: Library - Dosya Ge... rol dağılı... docker ko... library-m... pgAdmin 4 returnBo... axiosInsta... Container... 26°C 19:53 9.05.2025

Frontend View – Live Application (React UI)

The screenshot shows a live application interface for a library system. At the top, there are two browser tabs both titled "localhost". The main window has a purple header bar with the title "LibraryApp" on the left and navigation links for "Book Services", "Login", and "Register" on the right. A message banner at the top states: "The Hunger Games was returned ✅ (19:53:41)".

The main content area displays a grid of eight book cards, each with a purple rounded rectangular button labeled "Borrow".

Book Title	Author	ISBN	Genre	Status	Action
The Great Gatsby	F. Scott Fitzgerald	9780743273565	Classic	Available	Borrow
The Hunger Games	Suzanne Collins	9780439023528	Dystopian	Available	Borrow
The Lord of the Rings	J.R.R. Tolkien	9780544003415	Fantasy	Available	Borrow
The Road	Cormac McCarthy	9780307277671	Post-apocalyptic	Available	Borrow
To Kill a Mockingbird	Harper Lee	9780061120084	Classic	Available	Borrow
Pride and Prejudice	Jane Austen	978014139600	Romance	Available	Borrow
1984	George Orwell	9780307474278	Dystopian	Available	Borrow
Fahrenheit 451	Ray Bradbury	9781451673319	Science Fiction	Available	Borrow

At the bottom of the screen, a taskbar shows various open applications and system status indicators.

Register (Postman) – with PATRON

The screenshot shows the Postman application interface. The left sidebar has 'Project' selected, with 'Library' expanded. Under 'Library', there is a list of various API endpoints, including 'register'. The 'register' endpoint is currently selected and highlighted in grey. The main workspace shows a POST request to 'http://localhost:8080/library/api/auth/register'. The 'Body' tab is selected, showing a raw JSON payload:

```
1 {  
2   "name": "Berk",  
3   "email": "berk@gmail.com",  
4   "password": "123456",  
5   "phoneNumber": "5354214577",  
6   "role": "PATRON"  
7 }
```

Below the body, the response is shown as a 200 OK status with 163 ms duration and 599 B size. The response body is a JSON object containing a token:

```
{ } JSON ▾  
1 {  
2   "token": "eyJhbGciOiJIUzIiNiJ9.eyJzdWIiOiJiZXJiQGdtyWlsLmNvbSISInJvbGUIOiJQQVRST04iLCJpYXQiOjE3NDY4MDk20DksImV4cCI6MTc0Njg5NjA40X0.NY92Qc_Er6UI-iCQZooYs5T0GwRrqH-xpxCce1_Dvc4"  
3 }
```

At the bottom, the taskbar shows several open windows including 'Postbot', 'Runner', 'Start Proxy', 'Cookies', 'Vault', 'Trash', and others.

Register (Client View)

The screenshot shows a web browser window with two tabs open, both titled "localhost". The active tab displays a registration form for "LibraryApp". The form fields include:

- Name: Ahmet
- Email: ahmet@gmail.com
- Password: (redacted)
- Phone: 5548774459
- Role: PATRON

The "Sign Up" button is visible at the bottom of the form. A modal dialog box is overlaid on the page, containing the message "localhost:5173 web sitesinin mesajı" and "Registration successful! You can now log in." with a blue "Tamam" (OK) button.

In the top right corner of the browser, the developer tools Network tab is open, showing a single request for "register" with a status of 200 and a duration of 76 ms. The request URL is listed as "http://localhost:8080/library/api/auth/register".

At the bottom of the screen, the Windows taskbar is visible, showing various pinned icons and the current system status (19:56, 26°C, 9.05.2025).

Login (Postman) - with PATRON

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections, environments, flows, and history. The main area shows a library entry for 'POST login'.

Request Details:

- Method:** POST
- URL:** http://localhost:8080/library/api/auth/login
- Headers:** (8)
- Body:** (raw JSON)

```
1 "email": "ahmet@gmail.com",
2 "password": "123456"
```

Response Details:

- Status:** 200 OK
- Time:** 79 ms
- Size:** 600 B
- Content:** A JSON object containing a token.

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhaG1ldEBnbWFpbC5jb20iLCJyb2xlijoUEFUUK90IiwiaWF0IjoxNzQ2ODA5ODIxLCJleHAiOjE3NDY4OTYyMjF9.RiIQ91LXPN-LCDYW03jE0pBggOsElWVea8g7M_D1LhR4"
```

At the bottom, the taskbar shows various open windows and system status.

Login (Client View)

The screenshot shows a web browser window with two tabs open, both titled "localhost". The active tab displays the "LibraryApp" login page. A modal dialog box is overlaid on the page, containing the text "localhost:5173 web sitesinin mesajı" and "Login successful!". A blue "Tamam" button is at the bottom right of the modal. In the background, the main content of the app shows a banner about "The Lord of the Rings" and a "Login" button. The browser's developer tools Network tab is visible on the right, showing a single XHR request named "login" with a status of 200, initiated from "Login.jsx:19" and sent to "http://localhost:8080/library/api/auth/login". The browser's taskbar at the bottom shows various open applications like Dosya Ge..., pgAdmin 4, and Docker. The system tray indicates it's 19:57 on 09.05.2025.

localhost:5173 web sitesinin mesajı

Login successful!

Tamam

LibraryApp

The Lord of the Rings was returned

Log In

Don't have an account? [Register](#)

localhost:5173/login

localhost

localhost

Elements Console Sources Network Performance

Fetch/XHR Doc CSS JS Font Img Media Manifest Socket Wasm Other

Name Status Type Initiator Size Time

login 200 xhr Login.jsx:19 0.6 kB 70 ms

http://localhost:8080/library/api/auth/login

1 / 2 requests | 0.6 kB / 0.6 kB transferred | 0.2 kB / 0.2 kB resources

Windows Taskbar:

- Dosya Ge...
- pgAdmin 4
- login - Pr...
- axiosInsta...
- Container...
- 26°C
- 19:57
- 09.05.2025

GET /users with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various API endpoints under 'Library'. The main workspace displays a 'getAllUsers' endpoint with a 'GET' method. The 'Authorization' tab is selected, showing a dropdown set to 'Bearer Token' with a token value: dwBgQbrBmZCeKeWXPFnLzdFnstM. The 'Body' tab shows a JSON response containing two user objects:

```
[{"id": "29b42984-8326-4a09-9562-b97427b0be8c", "name": "Admin", "email": "admin@gmail.com", "phoneNumber": "5554214577", "role": "LIBRARIAN"}, {"id": "df7e6154-9f56-43c7-8be5-52998959639c", "name": "Hasan", "email": "hasan@gmail.com", "phoneNumber": "5559876543", "role": "PATRON"}]
```

The status bar at the bottom indicates the response was 200 OK with 14 ms latency and 1.05 KB size.

GET /users with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Collections' (empty), 'Environments' (empty), 'Flows' (empty), and 'History' (empty). The main area shows a 'Library' section with various API endpoints listed under 'HTTP Library / getAllUsers'. One endpoint, 'GET getAllUsers', is selected and highlighted in red. The 'Authorization' tab is active, showing 'Bearer Token' selected in the dropdown. A token value 'EZ_EDRfRiWwIMv5PWhpNRbsdSV-00' is displayed in a text input field. The 'Body' tab is active, showing the response status as '403 Forbidden'. The bottom status bar shows the system tray with icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a search bar.

Project

New Import

getAI | GET getSI | GET getSt | POST crea | PUT upda | GET getAI | GET getU | PUT upda | DEL deleti | POST crea | GET get

HTTP Library / getAllUsers

GET http://localhost:8080/library/api/users

Params Authorization Headers (7) Body Scripts Settings

Auth Type: Bearer Token

Token: EZ_EDRfRiWwIMv5PWhpNRbsdSV-00

403 Forbidden

Body Cookies Headers (13) Test Results (0/1)

Raw Preview Visualize

Postbot Runner Start Proxy Cookies Vault Trash

Online Find and replace Console

Postbot Runner Start Proxy Cookies Vault Trash

9:59 9.05.2025

GET /users – Admin Panel View

The screenshot shows a Microsoft Edge browser window with three tabs: 'Library' (active), 'localhost' (inactive), and 'Swagger UI' (inactive). The address bar shows 'localhost:5173/admin'. The main content area is titled 'Admin Panel' and displays the 'Manage Users' section. A sidebar on the left, titled 'Admin Menu', includes links for 'Manage Users' (selected), 'Manage Books', 'Borrow History', and 'Overdue Books'. The 'Manage Users' section contains a table with the following data:

ID	Name	Email	Phone	Role	Active Books	Status	Actions
29b42984-8326-4a09-9562-b97427b0be8c	Admin	admin@gmail.com	5554214577	LIBRARIAN	0	ACTIVE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
df7e6154-9f56-43c7-8be5-52998959639c	Hasan	hasan@gmail.com	5559876543	PATRON	1	OVERDUE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
152e474e-c906-48bc-a69a-bc518f932994	Berk	berk@gmail.com	5354214577	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
0b74cfab-8aac-4277-8acb-63977ef4f488	Mehmet	mehmet@gmail.com	5548794455	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
084d6d8e-02ad-431d-be31-4423320f7857	Ahmet	ahmet@gmail.com	5548774459	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>

The status column indicates two users are 'ACTIVE' (green) and one is 'OVERDUE' (red). The bottom of the screen shows the Windows taskbar with various icons and system information.

GET /users – Admin Panel View 2

The screenshot shows a browser window with three tabs: "Library" (active), "localhost" (closed), and "Swagger UI" (closed). The main content area displays the "Admin Panel" of the "LibraryApp". The panel has a purple header with the app name and a sidebar with a shield icon. A banner at the top says "The Hunger Games was returned ✅ (20:00:21)". Below it, the "Manage Users" section is shown with a table:

ID	Name	Email
29b42984-8326-4a09-9562-b97427b0be8c	Admin	admin@gm...
df7e6154-9f56-43c7-8be5-52998959639c	Hasan	hasan@gm...
152e474e-c906-48bc-a69a-bc518f932994	Berk	berk@gmai...

The developer tools' Network tab is open, showing the following requests:

Name	Status	Type	Initiator	Size	Time
me	200	xhr	Navbar.jsx:25	0.7 kB	21 ms
books	200	xhr	RecentActivityBanner,	2.5 kB	24 ms
users	200	xhr	ManageUsersPanelAd	1.2 kB	24 ms
borrow-records	http://localhost:8080/library/api/users	xhr	ManageUsersPanelAd	0.9 kB	27 ms
overdue	200	xhr	ManageUsersPanelAd	0.9 kB	29 ms
availability-stream	(pending)	events...	admin:20	0.0 kB	Pending

At the bottom of the browser window, the taskbar shows various icons and the system status bar indicates 20:00, 9.05.2025, 26°C, and battery level.

GET /users/{id} with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections, environments, flows, and history. The main workspace shows a collection named "Library" containing several API endpoints. One endpoint, "getUserById", is selected and expanded. The "Authorization" tab is active, showing a "Bearer Token" dropdown with a token value: dwBgQbrBm2ZCeKeWXPFnLzdFnstM. The "Body" tab shows a JSON response with the following data:

```
1 {  
2   "id": "084d6d8e-02ad-431d-be31-4423320f7857",  
3   "name": "Ahmet",  
4   "email": "ahmet@gmail.com",  
5   "phoneNumber": "5548774459",  
6   "role": "PATRON"  
7 }
```

The status bar at the bottom indicates the response was successful (200 OK) with a duration of 15 ms and a size of 552 B.

GET /users/{id} with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections, environments, flows, and history. The main area shows a library entry for 'getUserById'. The request details pane indicates a GET method with the URL `http://localhost:8080/library/api/users/084d6d8e-02ad-431d-be31-4423320f7857`. The 'Authorization' tab is selected, showing 'Bearer Token' as the auth type with a token value `EZ_EDRfRiWwIMv5PWhpNRbsdSV-00` highlighted. The response pane shows a 403 Forbidden status with a single character '1' in the raw response body. The bottom navigation bar includes icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a help icon.

GET /users/{id} – Admin Panel View

The screenshot shows a Microsoft Edge browser window with four tabs: 'Library' (active), 'localhost' (inactive), 'Swagger UI' (inactive), and '+'. The main content area is the 'LibraryApp' Admin Panel. The top navigation bar includes 'Book Services', 'Admin' (selected), 'Profile', and 'Logout'. A success message 'Brave New World was returned ✓ (20:00:21)' is displayed. On the left, a sidebar titled 'Admin Menu' lists 'Manage Users' (selected), 'Manage Books', 'Borrow History', and 'Overdue Books'. The central 'Admin Panel' section has a heading 'Manage Users' and a table listing five users:

ID	Name	Email	Phone	Role	Active Books	Status	Actions
29b42984-8326-4a09-9562-b97427b0be8c	Admin	admin@gmail.com	5554214577	LIBRARIAN	0	ACTIVE	<input checked="" type="checkbox"/> Delete Edit
df7e6154-9f56-43c7-8be5-52998959639c	Hasan	hasan@gmail.com	5559876543	PATRON	1	OVERDUE	<input checked="" type="checkbox"/> Delete Edit
152e474e-c906-48bc-a69a-bc518f932994	Berk	berk@gmail.com	5354214577	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> Delete Edit
0b74cfab-8aac-4277-8acb-63977ef4f488	Mehmet	mehmet@gmail.com	5548794455	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> Delete Edit
084d6d8e-02ad-431d-be31-4423320f7857	Ahmet	ahmet@gmail.com	5548774459	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> Delete Edit

The bottom taskbar shows various icons and system status: Windows logo, search, task view, Google Chrome (Library - ...), File Explorer (Dosya Ge...), Task Manager (rol dağılı...), Docker (docker ko...), pgAdmin 4, getUserB..., axiosInsta..., Container..., 26°C, 9.05.2025, 20:02.

GET /users/{id} – Admin Panel View 2

The screenshot shows a browser window with the address bar set to `localhost:5173/admin`. A modal window titled "Details" is displayed, showing user information:

Id:	084d6d8e-02ad-431d-be31-4423320f7857
Name:	Ahmet
Email:	ahmet@gmail.com
PhoneNumber:	5548774459
Role:	PATRON

In the background, the Network tab of the developer tools is open, showing a single XHR request:

Name	Status	Type	Initiator	Size	Time
084d6d8e-02ad-431d-be31-4423320f7857	200	xhr	DetailsModal.jsx:20	0.7 kB	12 ms

The initiator for the request is `DetailsModal.jsx:20`, and the URL is `http://localhost:8080/library/api/users/084d6d8e-02ad-431d-be31-4423320f7857`.

GET /users/me with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various API endpoints under 'Library'. The 'getOwnUserDetails' endpoint is selected. In the main workspace, a GET request is configured to 'http://localhost:8080/library/api/users/me'. The 'Authorization' tab is selected, showing 'Bearer Token' as the type, with a token value 'dwBgQbrBm2ZCeKeWXPFnLzdFnstM' displayed. The response section shows a 200 OK status with a response body containing user details:

```
1 {  
2   "id": "29b42984-8326-4a09-9562-b97427b0be8c",  
3   "name": "Admin",  
4   "email": "admin@gmail.com",  
5   "phoneNumber": "5554214577",  
6   "role": "LIBRARIAN"  
7 }
```

At the bottom, the taskbar shows various open windows and system status.

GET /users/me with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections, environments, flows, and history. The main workspace shows a collection named "Library" with a sub-item "getOwnUserDetails". A GET request is selected, with the URL set to `http://localhost:8080/library/api/users/me`. The "Authorization" tab is active, showing "Bearer Token" selected, and the token value is `EZ_EDRFRIWwIMv5PWhpNRbsdSV-00`. The "Body" tab shows a JSON response with the following data:

```
1 {  
2   "id": "084d6d8e-02ad-431d-be31-4423320f7857",  
3   "name": "Ahmet",  
4   "email": "ahmet@gmail.com",  
5   "phoneNumber": "5548774459",  
6   "role": "PATRON"  
7 }
```

The status bar at the bottom indicates the response was successful (200 OK) with a duration of 10 ms and a size of 552 B. The taskbar at the bottom shows several open applications, including Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a system clock showing 20:05 on 9.05.2025.

GET /users/me – Profile Page

The screenshot shows a Microsoft Edge browser window with three tabs open: "Library", "localhost", and "Swagger UI". The "localhost" tab is active and displays the "LibraryApp" profile page. The page has a purple header with the title "LibraryApp" and navigation links for "Book Services", "Profile", and "Logout". A success message "Pride and Prejudice was returned ✓ (20:05:58)" is visible. Below it, a "Profile Information" section displays the user's details: Name: Ahmet, Email: ahmet@gmail.com, and Phone: 5548774459. The taskbar at the bottom shows various pinned icons and the system clock.

Pride and Prejudice was returned ✓ (20:05:58)

Profile Information

Name: Ahmet
Email: ahmet@gmail.com
Phone: 5548774459

20:06 9.05.2025

GET /users/me – Profile Page 2

The screenshot shows a browser window with three tabs: "Library", "localhost" (active), and "Swagger UI". The address bar shows "localhost:5173/profile". The main content area displays the "LibraryApp" interface with a purple header. A message at the top says "The Great Gatsby was returned ✅ (20:06:48)". Below it, a "Profile Information" box contains the user's details: Name: Ahmet, Email: ahmet@gmail.com, and Phone: 5548774459.

The developer tools Network tab is open, showing the following requests:

Name	Status	Type	Initiator	Size	Time
me	200	xhr	Profile.jsx:11	0.7 kB	32 ms
availability-s	200	xhr	profile:20	0.0 kB	Pending
me	200	xhr	Navbar.jsx:25	0.7 kB	17 ms
books	200	xhr	RecentActivityBanner.	2.5 kB	19 ms

At the bottom of the browser window, the taskbar shows various icons and the system tray indicates a temperature of 26°C and the date/time as 9.05.2025.

PUT /users/{id} with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar includes sections for Project, Collections, Environments, Flows, History, and a search bar. The main area displays a collection named "Library" with various API endpoints listed under "updateUser". A specific PUT request is selected, targeting the URL `http://localhost:8080/library/api/users/0b74cfa8-8aac-4277-8acb-63977ef4f488`. The request body is set to raw JSON:

```
1 {
2   "name": "Mehmet Updated",
3   "email": "mehmet@gmail.com",
4   "phoneNumber": "5548794455",
5   "role": "PATRON"
6 }
```

The response section shows a green "200 OK" status with a response time of 26 ms and a size of 562 B. The response body is identical to the request body. The bottom of the screen shows the Windows taskbar with various icons and the system tray indicating the date and time.

PUT /users/{id} with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the sidebar includes sections for Project, Collections, Environments, Flows, History, and a plus sign for creating new items. The main area displays a library of API endpoints under the category 'HTTP Library / updateUser'. One endpoint, 'PUT /users/0b74cfa8-8aac-4277-8acb-63977ef4f488', is selected and shown in detail. The 'Authorization' tab is active, set to 'Bearer Token', with a token value 'EZ_EDRfRiWwIMv5PWhpNRbsdSV-00' displayed. The 'Headers' tab lists '(9)' headers. The 'Body' tab shows a raw JSON payload with a single field '1'. The response section indicates a '403 Forbidden' status with a response time of '10 ms' and a size of '389 B'. The bottom navigation bar includes links for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a search bar.

PUT /users/{id} – Admin Panel View

The screenshot shows a Microsoft Edge browser window with three tabs: 'Library' (active), 'localhost' (inactive), and 'Swagger UI' (inactive). The address bar shows 'localhost:5173/admin'. The main content area is titled 'LibraryApp' with a purple header bar containing 'Book Services', 'Admin', 'Profile', and 'Logout' links. A success message 'The Lord of the Rings was returned ✓ (20:09:46)' is displayed. On the left, a sidebar titled 'Admin Menu' includes 'Manage Users' (selected), 'Manage Books', 'Borrow History', and 'Overdue Books'. The central area is titled 'Admin Panel' and describes managing users, books, and borrowing activity. It features a 'Manage Users' section with a table showing five users:

ID	Name	Email	Phone	Role	Active Books	Status	Actions
29b42984-8326-4a09-9562-b97427b0be8c	Admin	admin@gmail.com	5554214577	LIBRARIAN	0	ACTIVE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
df7e6154-9f56-43c7-8be5-52998959639c	Hasan	hasan@gmail.com	5559876543	PATRON	1	OVERDUE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
152e474e-c906-48bc-a69a-bc518f932994	Berk	berk@gmail.com	5354214577	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
084d6d8e-02ad-431d-be31-4423320f7857	Ahmet Updated	ahmet@gmail.com	5548774459	PATRON	0	ACTIVE	<input type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
0b74cfab-8aac-4277-8acb-63977ef4f488	Mehmet Updated	mehmet@gmail.com	5548794455	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>

The taskbar at the bottom shows various open applications including Docker, pgAdmin 4, and a terminal window.

PUT /users/{id} – Admin Panel View 2

The screenshot shows a browser window with two tabs: "localhost" and "Swagger UI". The "localhost" tab displays an admin panel for managing users. On the left, there is a table with the following data:

	Role	Active Books	Status	Actions
77	LIBRARIAN	0	ACTIVE	[Edit, Delete, Info]
43	PATRON	1	OVERDUE	[Edit, Delete, Info]
77	PATRON	0	ACTIVE	[Edit, Delete, Info]
59	PATRON	0	ACTIVE	[Edit, Delete, Info]
55	PATRON	0	ACTIVE	[Edit, Delete, Info]

On the right, the browser's developer tools Network tab is open, showing a single request:

Name	Status	Type	Initiator	Size	Time
084d6d8e-02ad-431d-be31-44... http://localhost:8080/library/api/users/084d6d8e-02ad-431d-be31-4423320f7857	200	xhr	ManageUsersPanelAd	0.7 kB	14 ms

At the bottom, the taskbar shows various open applications including a library manager, file explorer, Docker, pgAdmin, and a terminal.

DELETE /users/{id} with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Collections' sidebar lists various API endpoints, including 'getOwnUserDetails', 'getAllBooks', 'getBookByld', 'searchByTitle', 'searchByAuthor', 'searchByLisbn', 'searchByGenre', 'getAllBorrowRecords', 'getBorrowRecordsByUserId', 'getOwnBorrowRecords', 'getOverdueRecords', 'getOverdueRecordsReport', 'createBook', 'borrowBook', 'register', 'login', 'updateUser', 'updateBook', 'returnBook', 'deleteUser', 'deleteBook', and several 'practiz-' prefixed endpoints. The 'deleteUser' endpoint is currently selected and highlighted.

In the main workspace, a 'DELETE' request is configured against the URL `http://localhost:8080/library/api/users/0b74cfa8-8aac-4277-8acb-63977ef4f488`. The 'Authorization' tab is selected, showing a dropdown set to 'Bearer Token' with a corresponding token value: `dwBgQbrBm2ZCeKeWXPFnLzdFnstM`.

The response section shows a green status bar indicating a **204 No Content** response with a duration of 36 ms and a size of 371 B. Below this, the 'Raw' response body is displayed as a single digit '1'.

The bottom of the screen shows the Windows taskbar with various pinned icons and the system clock indicating 20:11 on 9.05.2025.

DELETE /users/{id} with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the 'Collections' sidebar lists various API endpoints, including 'GET getOwnUserDetails', 'DELETE deleteUser', and 'PUT updateUser'. The main workspace displays a DELETE request to `http://localhost:8080/library/api/users/152e474e-c906-48bc-a69a-bc518f932994`. The 'Authorization' tab is selected, showing 'Bearer Token' as the type and a token value '74e-c906-48bc-a69a-bc518f932994' highlighted with a yellow box. The response status is '403 Forbidden' with a 5 ms duration and 389 B size. The bottom taskbar shows various icons and system status.

DELETE /users/{id} – Admin Panel View

The screenshot shows a Microsoft Edge browser window with three tabs: 'Library' (active), 'localhost' (inactive), and 'Swagger UI' (inactive). The address bar shows 'localhost:5173/admin'. The main content area is the 'Admin Panel' of 'LibraryApp', featuring a purple header with 'LibraryApp', 'Book Services', 'Admin', 'Profile', and 'Logout' links. A success message 'Fahrenheit 451 was returned ✓ (20:12:34)' is displayed. On the left, a sidebar titled 'Admin Menu' includes 'Manage Users' (selected), 'Manage Books', 'Borrow History', and 'Overdue Books'. The central 'Manage Users' section has a table with columns: ID, Name, Email, Phone, Role, Active Books, Status, and Actions. The table contains four rows of user data:

ID	Name	Email	Phone	Role	Active Books	Status	Actions
29b42984-8326-4a09-9562-b97427b0be8c	Admin	admin@gmail.com	5554214577	LIBRARIAN	0	ACTIVE	<input checked="" type="checkbox"/> trash edit
df7e6154-9f56-43c7-8be5-52998959639c	Hasan	hasan@gmail.com	5559876543	PATRON	1	OVERDUE	<input checked="" type="checkbox"/> trash edit
152e474e-c906-48bc-a69a-bc518f932994	Berk	berk@gmail.com	5354214577	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> trash edit
084d6d8e-02ad-431d-be31-4423320f7857	Ahmet Updated	ahmet@gmail.com	5548774459	PATRON	0	ACTIVE	<input checked="" type="checkbox"/> trash edit

A 'Delete' button is located at the bottom right of the table. The taskbar at the bottom shows various open applications and system icons.

DELETE /users/{id} – Admin Panel View 2

The screenshot shows a browser window with three tabs: "Library", "localhost", and "Swagger UI". The "localhost" tab displays an admin panel for managing users. The main content area shows a table of users:

Role	Books	Status	Actions
LIBRARIAN	0	ACTIVE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
PATRON	1	OVERDUE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>
PATRON	0	ACTIVE	<input checked="" type="checkbox"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>

The "Status" column uses color coding: green for ACTIVE and red for OVERDUE. The "Actions" column contains checkboxes, delete buttons, and edit buttons.

In the bottom right corner of the main content, there is a small modal or overlay with the text "User deleted successfully!".

The "Network" tab in the developer tools is open, showing an XHR request details panel. The request is for the URL `http://localhost:8080/library/api/users/084d6d8e-02ad-431d-be31-4423320f7857`. The status is 204, and the initiator is "ManageUsersPanelAd". The time taken is 15 ms.

At the bottom of the screen, the taskbar shows various icons and the system tray indicates it's 20:13 on 9.05.2025.

GET /books with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. In the center, a specific API endpoint is selected: 'HTTP Library / getAllBooks'. The 'Authorization' tab is active, showing 'Bearer Token' selected in the dropdown. A token value 'dwBgQbrBmZCeKeWXPFnLzdFnstM' is displayed in a text input field. The 'Body' tab shows a JSON response with two book entries:

```
[{"id": "e9185c46-d706-4c7a-9452-8093ce4702c1", "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "isbn": "9780743273565", "publicationDate": "1925-04-10", "genre": "Classic", "available": true}, {"id": "33206a7f-2f62-4e35-ab89-98db53e6352b", "title": "The Hunger Games", "author": "Suzanne Collins", "isbn": "9780439023528", "publicationDate": "2008-09-14"}]
```

The status bar at the bottom indicates the response was successful (200 OK) with a duration of 12 ms and a size of 2.28 KB.

GET /books with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. In the center, a specific API endpoint named 'getAllBooks' is selected under the 'Library' collection. The 'Authorization' tab is active, showing a 'Bearer Token' dropdown with the value 'EZ_EDRfRiWwIMv5PWhpNRbsdSV-00'. The 'Body' tab displays a JSON response with two book objects:

```
[{"id": "e9185c46-d706-4c7a-9452-8093ce4702c1", "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "isbn": "9780743273565", "publicationDate": "1925-04-10", "genre": "Classic", "available": true}, {"id": "33206a7f-2f62-4e35-ab89-98db53e6352b", "title": "The Hunger Games", "author": "Suzanne Collins", "isbn": "9780439023528", "publicationDate": "2008-09-14"}]
```

The status bar at the bottom shows the system tray with icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a date/time indicator of 20:14 9.05.2025.

GET /books – Home & All Books

The screenshot shows a web browser window with the URL `localhost:5173/book-services`. The page has a purple header bar with the title "LibraryApp". On the left, there's a sidebar menu with "All Books" selected. The main content area is titled "Book Services" and displays a list of books with their details and borrowing status.

Books Available:

- The Great Gatsby**
Author: F. Scott Fitzgerald
ISBN: 9780743273565
Genre: Classic
Status: Available
[Borrow](#)
- The Hunger Games**
Author: Suzanne Collins
ISBN: 9780439023528
Genre: Dystopian
Status: Available
[Borrow](#)
- The Lord of the Rings**
Author: J.R.R. Tolkien
ISBN: 9780544003415
Genre: Fantasy
Status: Available
[Borrow](#)

Books Borrowed:

- The Road**
Author: Cormac McCarthy
ISBN: 9780307277671
Genre: Post-apocalyptic
Status: Available
- To Kill a Mockingbird**
Author: Harper Lee
ISBN: 9780061120084
Genre: Classic
Status: Available
- Pride and Prejudice**
Author: Jane Austen
ISBN: 9780141439600
Genre: Romance
Status: Available

The browser taskbar at the bottom shows various open tabs and system icons.

GET /books – Home & All Books 2

The screenshot shows a browser window with three tabs: 'Library' (active), 'localhost' (background), and 'Swagger UI' (background). The main content area displays the 'LibraryApp' interface. At the top, a banner says 'To Kill a Mockingbird was returned ✅ (20:16:01)'. Below it, a sidebar has a 'Book Services' section with the text: 'View all available books, borrow new ones, and track your book history.' A 'Filter Books' search bar is present. In the main content area, a card for 'The Great Gatsby' is shown with details: Author: F. Scott Fitzgerald, ISBN: 9780743273565, Genre: Classic, and a status of 'Available'. A 'Borrow' button is at the bottom. The right side of the screen features the Chrome DevTools Network tab, which shows a timeline of requests. A table lists the requests:

Name	Status	Type	Initiator	Size	Time
availability-stream	(pending)	events...	book-services:20	0.0 kB	Pending
books	200	xhr	RecentActivityBanner,	2.5 kB	21 ms
me	200	xhr	Navbar.jsx:25	0.7 kB	19 ms
books	200	xhr	AllBooksPanel.jsx:46	2.5 kB	36 ms
me	200	xhr	AllBooksPanel.jsx:19	0.7 kB	33 ms

At the bottom of the browser window, the taskbar shows various open applications including Dosya Ge..., Spotify, Docker, pgAdmin 4, axiosInsta..., Container..., and a system tray with a 26°C temperature indicator.

GET /books/{id} with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections, environments, flows, and history. The main workspace shows a collection named "Library" containing several API endpoints. One endpoint, "getBookById", is selected and highlighted with a grey background. The request details panel shows a GET method for the URL `http://localhost:8080/library/api/books/e9185c46-d706-4c7a-9452-8093ce4702c1`. The "Authorization" tab is selected, showing "Bearer Token" as the type and a token value `dwBgQbrBm2ZCeKeWXPFnLzdFnstM` in the input field. The response panel shows a 200 OK status with a response time of 14 ms and a body size of 615 B. The response body is displayed as JSON:

```
1 {  
2   "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",  
3   "title": "The Great Gatsby",  
4   "author": "F. Scott Fitzgerald",  
5   "isbn": "9780743273565",  
6   "publicationDate": "1925-04-10",  
7   "genre": "Classic",  
8   "available": true  
9 }
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray indicating the date and time.

GET /books/{id} with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main workspace displays a collection named "Library" containing several API endpoints. One endpoint, "getBookById", is selected and highlighted in green. The request details panel shows a GET request to `http://localhost:8080/library/api/books/e9185c46-d706-4c7a-9452-8093ce4702c1`. The "Authorization" tab is selected, showing "Bearer Token" as the auth type with a token value "EZ_EDRfRiWwIMv5PWhpNRbsdSV-00". The response panel shows a 200 OK status with a response body in JSON format:

```
1 {
2   "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",
3   "title": "The Great Gatsby",
4   "author": "F. Scott Fitzgerald",
5   "isbn": "9780743273565",
6   "publicationDate": "1925-04-10",
7   "genre": "Classic",
8   "available": true
9 }
```

The bottom navigation bar includes links for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a help icon. The taskbar at the bottom shows various open applications like Dosya Ge..., pgAdmin 4, getBookB..., axiosInsta..., Container..., and a system tray with a 26°C temperature indicator.

GET /books/{id} – Admin Panel Manage Books Details

The screenshot shows a Microsoft Edge browser window with three tabs: "Library", "localhost", and "Swagger UI". The main content area displays a list of books, with one book's details modal open. The modal has a title "Details" and contains the following information:

Attribute	Value
Id	e9185c46-d706-4c7a-9452-8093ce4702c1
Title	The Great Gatsby
Author	F. Scott Fitzgerald
ISBN	9780743273565
PublicationDate	1925-04-10
Genre	Classic
Available	Yes

The taskbar at the bottom shows several pinned icons and the current time: 20:18, 9.05.2025.

GET /books/{id} – Admin Panel Manage Books Details 2

The screenshot shows a browser window with three tabs: "Library", "localhost", and "Swagger UI". The main content area displays a modal titled "Details" for a book. The modal contains the following information:

Id:	e9185c46-d706-4c7a-9452-8093ce4702c1
Title:	The Great Gatsby
Author:	F. Scott Fitzgerald
Isbn:	9780743273565
PublicationDate:	1925-04-10
Genre:	Classic
Available:	Yes

The Network tab of the developer tools shows an XHR request for the book's details. The request is labeled "e9185c46-d706-4c7a-9452-8093ce4702c1" and has a status of 200. The initiator is "DetailsModal.jsx:20" and the URL is "http://localhost:8080/library/api/books/e9185c46-d706-4c7a-9452-8093ce4702c1". The response size is 0.8 kB and the time taken is 11 ms.

At the bottom, the taskbar shows various open applications including Dosya Ge..., pgAdmin 4, getBookB..., axiosInsta..., Container..., and Docker. The system tray shows the date and time as 9.05.2025 and 20:19.

GET /books/search/title with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. In the center, a specific API endpoint named 'searchByTitle' is selected under the 'Library' collection. The 'GET' request is configured with the URL `http://localhost:8080/library/api/books/search/title?title=great&page=0&size=5`. The 'Authorization' tab is active, set to 'Bearer Token', with a token value displayed: `dwBgQbrBm2ZCeKeWXPFnLzdFnstM`. The response body is shown as JSON, containing a single book entry:

```
1 {  
2   "content": [  
3     {  
4       "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",  
5       "title": "The Great Gatsby",  
6       "author": "F. Scott Fitzgerald",  
7       "isbn": "9780743273565",  
8       "publicationDate": "1925-04-10",  
9       "genre": "Classic",  
10      "available": true  
11    },  
12    ],  
13    "pageable": {  
14      "pageNumber": 0,  
15      "pageSize": 5,  
16      "sort": {}  
17    }  
18  }  
19 }
```

The status bar at the bottom indicates the response was successful (200 OK) with a duration of 44 ms and a size of 929 B. The taskbar at the very bottom shows various open applications.

GET /books/search/title with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various API endpoints under 'Library'. The 'searchByTitle' endpoint is selected. In the main workspace, a 'GET' request is configured to 'http://localhost:8080/library/api/books/search/title?title=great&page=0&size=5'. The 'Authorization' tab is selected, showing 'Bearer Token' as the type and a token value 'EZ_EDRfRIWwlMv5PWhpNRbsdSV-00' entered in the field. The 'Headers' tab shows '(7)' entries. The 'Body' tab is set to '{} JSON'. The 'Test Results' tab shows a single result: '200 OK' with a response time of '12 ms', a size of '929 B', and a status message 'Save Response'. The response body is displayed as JSON:

```
1 {  
2   "content": [  
3     {  
4       "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",  
5       "title": "The Great Gatsby",  
6       "author": "F. Scott Fitzgerald",  
7       "isbn": "9780743273565",  
8       "publicationDate": "1925-04-10",  
9       "genze": "Classic",  
10      "available": true  
11    },  
12    ],  
13    "pageable": {  
14      "pageNumber": 0,  
15      "pageSize": 5,  
16      "sort": {}  
17    }  
18  }  
19 }
```

At the bottom, the taskbar shows various open tabs and system icons.

GET /books/search/title – All Books filter books

The screenshot shows a web browser window with three tabs: 'Library' (active), 'localhost' (inactive), and 'Swagger UI' (inactive). The address bar shows 'localhost:5173/book-services'. The main content area is titled 'Book Services' and displays a message: 'The Great Gatsby was returned ✅ (20:18:18)'. On the left, a sidebar menu has 'All Books' selected. The main content area includes a 'Filter Books' search bar with 'great' typed in and a dropdown set to 'Title'. Below this, a card for 'The Great Gatsby' by F. Scott Fitzgerald is shown, with details: ISBN: 9780743273565, Genre: Classic, and status 'Available'. A large blue 'Borrow' button is at the bottom of the card. The taskbar at the bottom shows various open applications.

Library

localhost

Swagger UI

localhost:5173/book-services

LibraryApp

Book Services Admin Profile Logout

The Great Gatsby was returned ✅ (20:18:18)

Menu

All Books

My Books

Borrow History

Book Services

View all available books, borrow new ones, and track your book history.

Filter Books

great

Title

The Great Gatsby

Author: F. Scott Fitzgerald
ISBN: 9780743273565
Genre: Classic

Available

Borrow

Dosya Ge... rol dağılı... docker ko... library-m... pgAdmin 4 searchBy... axiosInsta... Container... 26°C 9.05.2025

GET /books/search/title – All Books filter books 2

The screenshot shows a browser window with three tabs: 'Library' (active), 'localhost' (inactive), and 'Swagger UI' (inactive). The main content area displays a library application interface. On the left, there's a sidebar with the text: 'View all available books, borrow new ones, and track your book history.' Below it is a 'Filter Books' search component with a search bar containing 'great' and a dropdown menu set to 'Title'. To the right, a card for 'The Great Gatsby' by F. Scott Fitzgerald is shown, detailing its author, ISBN, genre, and status as 'Available', with a 'Borrow' button. At the bottom of the card, it says '© 2025 Library App'. On the right side of the screen, the browser's developer tools Network tab is open, showing a list of XHR requests. One request is highlighted with a red border, showing the URL: <http://localhost:8080/library/api/books/search/title?title=great&page=0&size=20>. The Network tab also includes a timeline at the top and a table below with columns: Name, Status, Type, Initiator, Size, and Time.

Name	Status	Type	Initiator	Size	Time
title?title=g&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.9 kB	11 ms
title?title=gr&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	11 ms
title?title=gre&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	14 ms
title?title=grea&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	10 ms
title?title=great&page=0&size...	200	xhr	AllBooksPanel.jsx:37	1.1 kB	12 ms

GET /books/search/author with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. In the center, a specific API endpoint named 'searchByAuthor' is selected under the 'Library' collection. The 'GET' request is configured with the URL `http://localhost:8080/library/api/books/search/author?author=scott&page=0&size=5`. The 'Authorization' tab is active, set to 'Bearer Token', with a token value `dwBgQbrBmZCeKeWXPFnLzdFnstM` displayed. The response body is shown as JSON, containing a single book entry:

```
1 {  
2   "content": [  
3     {  
4       "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",  
5       "title": "The Great Gatsby",  
6       "author": "F. Scott Fitzgerald",  
7       "isbn": "9780743273565",  
8       "publicationDate": "1925-04-10",  
9       "genre": "Classic",  
10      "available": true  
11    },  
12    {"pageable": {  
13      "pageNumber": 0,  
14      "pageSize": 5,  
15      "sort": {}  
16    }  
17  }  
18}
```

The status bar at the bottom indicates the response was successful (200 OK) with 15 ms latency and 929 B size. The taskbar at the very bottom shows various open applications.

GET /books/search/author with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. In the center, the 'Library / searchByAuthor' section is displayed. A 'GET' request is selected with the URL `http://localhost:8080/library/api/books/search/author?author=scott&page=0&size=5`. The 'Authorization' tab is active, set to 'Bearer Token', with a token value `EZ_EDRfRiWwIMv5PWhpNRbsdSV-00` shown in a yellow box. The 'Body' tab shows a JSON response with 16 lines of code. The response body is:

```
{  
    "content": [  
        {  
            "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",  
            "title": "The Great Gatsby",  
            "author": "F. Scott Fitzgerald",  
            "isbn": "9780743273565",  
            "publicationDate": "1925-04-10",  
            "genre": "Classic",  
            "available": true  
        }  
    ],  
    "pageable": {  
        "pageNumber": 0,  
        "pageSize": 5,  
        "sort": {}  
    }  
}
```

The status bar at the bottom shows the Postbot icon, a toolbar with various icons, and system information like the date and time.

GET /books/search/author – All Books filter books

localhost:5173/book-services

Book Services

All Books

My Books

Borrow History

Filter Books

scott

Author

The Great Gatsby

Author: F. Scott Fitzgerald

ISBN: 9780743273565

Genre: Classic

Available

Borrow

24°C 9.05.2025 20:23

GET /books/search/author – All Books filter books 2

The screenshot shows a browser window with three tabs: 'Library' (active), 'localhost' (inactive), and 'Swagger UI' (inactive). The main content area displays a library application interface. On the left, there's a 'Filter Books' search bar with 'scott' typed in, and a dropdown menu set to 'Author'. Below it, a book card for 'The Great Gatsby' by F. Scott Fitzgerald is shown, including its ISBN (9780743273565) and genre (Classic). A green 'Available' status and a purple 'Borrow' button are also visible. At the bottom of the card, it says '© 2025 Library App'. To the right of the application, the browser's developer tools Network tab is open, showing a list of XHR requests. One request is highlighted with a red border: 'http://localhost:8080/library/api/books/search/author?author=scott&page=0&size=20'. The Network tab includes a timeline at the top and a table below with columns: Name, Status, Type, Initiator, Size, and Time. The table lists five requests from 'AllBooksPanel.jsx:37' with various status codes (200) and sizes (e.g., 1.9 kB, 1.1 kB).

Name	Status	Type	Initiator	Size	Time
author?author=s&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.9 kB	10 ms
author?author=sc&page=0&size=...	200	xhr	AllBooksPanel.jsx:37	1.1 kB	13 ms
author?author=sco&page=0&size...	200	xhr	AllBooksPanel.jsx:37	1.1 kB	12 ms
author?author=scot&page=0&size...	200	xhr	AllBooksPanel.jsx:37	1.1 kB	10 ms
author?author=scott&page=0...	200	xhr	AllBooksPanel.jsx:37	1.1 kB	14 ms

GET /books/search/isbn with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. The main workspace displays a collection named 'Library'. A specific request titled 'GET /searchByIsbn' is selected. The 'Authorization' tab is active, showing a dropdown set to 'Bearer Token' with a token value: dwBgQbrBm2ZCeKeWXPFnLzdFnstM. The 'Body' tab shows a JSON response with one item, representing a book. The response body is as follows:

```
1 {  
2   "content": [  
3     {  
4       "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",  
5       "title": "The Great Gatsby",  
6       "author": "F. Scott Fitzgerald",  
7       "isbn": "9780743273565",  
8       "publicationDate": "1925-04-10",  
9       "genre": "Classic",  
10      "available": true  
11    },  
12    {"pageable": {  
13      "pageNumber": 0,  
14      "pageSize": 5,  
15      "sort": {}  
16    }  
17  }  
18}
```

The status bar at the bottom indicates the response was successful (200 OK) with a duration of 14 ms and a size of 929 B. The taskbar at the bottom shows various open applications including Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and several browser tabs.

GET /books/search/isbn with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. The main workspace displays a collection named 'Library / searchByIsbn'. A specific GET request is selected, which has the URL `http://localhost:8080/library/api/books/search/isbn?isbn=9780743273565&page=0&size=5`. The 'Authorization' tab is active, showing 'Bearer Token' selected. The response body is displayed as JSON, containing a single book entry:

```
1 {  
2   "content": [  
3     {  
4       "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",  
5       "title": "The Great Gatsby",  
6       "author": "F. Scott Fitzgerald",  
7       "isbn": "9780743273565",  
8       "publicationDate": "1925-04-10",  
9       "genre": "Classic",  
10      "available": true  
11    },  
12    ],  
13    "pageable": {  
14      "pageNumber": 0,  
15      "pageSize": 5,  
16      "sort": {}  
17    }  
18  }  
19 }
```

The response status is 200 OK, with a duration of 11 ms and a size of 929 B. The bottom navigation bar includes links for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a search bar.

GET /books/search/isbn – All Books filter books

The screenshot shows a web browser window with the following details:

- URL:** localhost:5173/book-services
- Page Title:** Library
- Content Area:**
 - A sidebar on the left with buttons: "All Books" (highlighted in purple), "My Books", and "Borrow History".
 - A main area titled "Filter Books" with a search input containing "9780743273565" and a dropdown menu set to "ISBN".
 - A card for "The Great Gatsby" by F. Scott Fitzgerald, showing the ISBN "9780743273565" and the genre "Classic".
 - The book status is "Available" with a blue "Borrow" button.
- Footer:** © 2025 Library App
- Taskbar:** Shows various open applications including Dosya Ge..., pgAdmin 4, searchByl..., axiosInsta..., Container..., Docker, and a weather widget showing 24°C.

GET /books/search/isbn – All Books filter books 2

The screenshot shows a browser window with two tabs: "Library" and "localhost". The "localhost" tab displays a library application interface. On the left, there's a sidebar with the text: "View all available books, borrow new ones, and track your book history." Below it is a "Filter Books" section with an input field containing "9780743273565" and a dropdown menu showing "ISBN". On the right, there's a card for the book "The Great Gatsby" by F. Scott Fitzgerald, with ISBN "9780743273565" and genre "Classic". A green "Available" status and a purple "Borrow" button are also visible. The browser's developer tools Network tab is open, showing a list of XHR requests. The table has columns: Name, Status, Type, Initiator, Size, and Time. Most requests are for "isbn?isbn=9780743273565&page=0&size=20" with various parameters. One request is highlighted with a red border and the URL "http://localhost:8080/library/api/books/search/isbn?isbn=9780743273565&page=0&size=20". The bottom of the screen shows the Windows taskbar with various icons and the system tray.

Name	Status	Type	Initiator	Size	Time
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	2.8 kB	10 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	2.8 kB	9 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	2.8 kB	12 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	2.6 kB	12 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	10 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	10 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	11 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	11 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	11 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	9 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	10 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	10 ms
isbn?isbn=9780743273565&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.1 kB	10 ms

GET /books/search/genre with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. In the center, a specific API endpoint named 'HTTP Library / searchByGenre' is selected. This endpoint is a GET request to the URL `http://localhost:8080/library/api/books/search/genre?genre=classic&page=0&size=5`. The 'Authorization' tab is active, showing 'Bearer Token' selected. A token value, `dwBgQbrBm2ZCeKeWXPFnLzdFnstM`, is displayed in a text input field. The 'Body' tab shows a JSON response with two book entries:

```
1 {  
2   "content": [  
3     {  
4       "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",  
5       "title": "The Great Gatsby",  
6       "author": "F. Scott Fitzgerald",  
7       "isbn": "9780743273565",  
8       "publicationDate": "1925-04-10",  
9       "genre": "Classic",  
10      "available": true  
11    },  
12    {  
13      "id": "38c0d359-659e-4694-bfbc-2893958b5746",  
14      "title": "To Kill a Mockingbird",  
15      "author": "Harper Lee",  
16      "isbn": "9780061120084",  
17    }  
18  ]  
19}
```

The response status is 200 OK with a duration of 12 ms and a size of 1.09 KB. The bottom navigation bar includes links for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a help icon.

GET /books/search/genre with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar includes 'Project' (selected), 'Collections' (with 'Getir Clone - Api Gateway (8080)' listed), 'Environments', 'Flows', and 'History'. The main workspace displays an 'HTTP Library / searchByGenre' collection. A specific GET request for '/books/search/genre?genre=classic&page=0&size=5' is selected. The 'Authorization' tab is active, showing 'Bearer Token' selected. A token value 'EZ_EDRfRiWwIMv5PWhpNRbsdSV-00' is displayed in a yellow-bordered box. The response section shows a 200 OK status with a response time of 11 ms and a size of 1.09 KB. The response body is a JSON array of book objects:

```
1 {  
2   "content": [  
3     {  
4       "id": "e9185c46-d706-4c7a-9452-8093ce4702c1",  
5       "title": "The Great Gatsby",  
6       "author": "F. Scott Fitzgerald",  
7       "isbn": "9780743273565",  
8       "publicationDate": "1925-04-10",  
9       "genre": "Classic",  
10      "available": true  
11    },  
12    {  
13      "id": "38c0d359-659e-4694-bfbc-2893958b5746",  
14      "title": "To Kill a Mockingbird",  
15      "author": "Harper Lee",  
16      "isbn": "9780061120084",  
17    }  
18  ]  
19}  
20}
```

At the bottom, the taskbar shows various icons including Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and system notifications. The system tray indicates it's 20:28 on 9.05.2025.

GET /books/search/genre – All Books filter books

localhost:5173/book-services

All Books

View all available books, borrow new ones, and track your book history.

Filter Books

classic

Genre

The Great Gatsby

Author: F. Scott Fitzgerald
ISBN: 9780743273565
Genre: Classic

Available

Borrow

To Kill a Mockingbird

Author: Harper Lee
ISBN: 9780061120084
Genre: Classic

Available

Borrow

2025-11-11

24°C 9.05.2025

GET /books/search/genre – All Books filter books 2

The screenshot shows a browser window with three tabs: 'Library' (active), 'localhost' (inactive), and 'Swagger UI' (inactive). The main content area displays a library interface with a search bar containing 'classic'. Below it, two book cards are shown: 'The Great Gatsby' by F. Scott Fitzgerald and 'To Kill a Mockingbird' by Harper Lee, both listed as 'Available' with a 'Borrow' button.

The right side of the screen features the Chrome DevTools Network tab. It is set to the 'Fetch/XHR' tab and shows a list of requests. The table has columns for Name, Status, Type, Initiator, Size, and Time. Seven requests are listed, all with a status of 200 and type 'xhr', initiated from 'AllBooksPanel.jsx:37'. The total size for these requests is 10.3 kB and the total time is 10 ms. A tooltip at the bottom indicates the URL: http://localhost:8080/library/api/books/search/genre?genre=classic&page=0&size=20.

Name	Status	Type	Initiator	Size	Time
genre?genre=genre=c&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	2.0 kB	10 ms
genre?genre=genre=cl&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.3 kB	10 ms
genre?genre=cla&page=0&size=20	200	xhr	AllBooksPanel.jsx:37	1.3 kB	9 ms
genre?genre=clas&page=0&size=...	200	xhr	AllBooksPanel.jsx:37	1.3 kB	10 ms
genre?genre=class&page=0&size=...	200	xhr	AllBooksPanel.jsx:37	1.3 kB	8 ms
genre?genre=classi&page=0&size=...	200	xhr	AllBooksPanel.jsx:37	1.3 kB	9 ms
genre?genre=classic&page=0...	200	xhr	AllBooksPanel.jsx:37	1.3 kB	10 ms

7 / 14 requests | 9.7 kB / 9.7 kB transferred | 5.6 kB / 5.6 kB resources

Windows Taskbar icons include: Search, Library, Dosya Ge..., Spotify, Docker, Docker Compose, pgAdmin 4, searchBy..., axiosInsta..., Container..., 24°C, 9.05.2025, 20:29.

POST /books with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections, environments, flows, and history. The main workspace shows a collection named "Gear Clone - Product Service". Within this collection, a specific POST request titled "createBook" is selected. The request URL is `http://localhost:8080/library/api/books`. The "Body" tab is active, showing a JSON payload:

```
1 {  
2   "title": "The Feeling Good Book2",  
3   "author": "Andrew Sharmaz",  
4   "isbn": "9780451524936",  
5   "publicationDate": "1980-07-01",  
6   "genre": "Personal Development"  
7 }
```

The response section shows a green "200 OK" status with a response time of 23 ms and a response size of 630 B. The response body is identical to the request body, indicating a successful creation of the book.

POST /books with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Collections' (empty), 'Environments' (empty), 'Flows' (empty), and 'History' (empty). The main area shows a 'Library' section with various API endpoints listed under 'createBook'. A specific POST request to 'http://localhost:8080/library/api/books' is selected. The 'Authorization' tab is active, showing 'Bearer Token' selected in the dropdown. A token value 'EZ_EDRfRiWwIMv5PWhpNRbsdSV-00' is displayed in a text input field. The 'Body' tab is selected, showing a single character '1' in the raw data. The status bar at the bottom indicates a 403 Forbidden response with 9 ms duration and 389 B size.

Project

New Import

POST crea ● GET getAl ● GET getBr ● PUT upda ● DEL deleti ● GET searc ● GET searc ● GET searc ● POST borr ● GE

HTTP Library / createBook

POST http://localhost:8080/library/api/books

Params Authorization Headers (9) Body Scripts Settings

Auth Type: Bearer Token

Token: EZ_EDRfRiWwIMv5PWhpNRbsdSV-00

Body Cookies Headers (13) Test Results (0/1)

Raw Preview Visualize

403 Forbidden 9 ms 389 B Save Response

Online Find and replace Console

Postbot Runner Start Proxy Cookies Vault Trash

24°C 9.05.2025

POST /books – Admin panel manage books

The screenshot shows the LibraryApp Admin Panel interface. On the left, there's a sidebar with the title "Admin Menu" and several options: "Manage Users", "Manage Books" (which is highlighted in purple), "Borrow History", and "Overdue Books". The main content area has a header "Admin Panel" with the sub-header "Manage users, books, and borrowing activity in the system." Below this is a section titled "Manage Books" with a table listing various books. The table columns are: Title, Author, ISBN, Date, Genre, Status, and Actions. The "Actions" column contains icons for edit, delete, and refresh. A message at the top of the main content area says "The Feeling Good Book was borrowed (20:31:51)". The browser address bar shows "localhost:5173/admin". The taskbar at the bottom includes icons for various applications like Dosya Ge..., Docker, and pgAdmin 4.

Title	Author	ISBN	Date	Genre	Status	Actions
The Great Gatsby 2	F. Scott Fitzgerald	9780743273566	10.04.1930	Classic	Available	
The Great Gatsby	F. Scott Fitzgerald	9780743273565	1925-04-10	Classic	Available	
The Hunger Games	Suzanne Collins	9780439023528	2008-09-14	Dystopian	Available	
The Lord of the Rings	J.R.R. Tolkien	9780544003415	1954-07-29	Fantasy	Available	
The Road	Cormac McCarthy	9780307277671	2006-09-26	Post-apocalyptic	Available	
To Kill a Mockingbird	Harper Lee	9780061120084	1960-07-11	Classic	Available	
Pride and Prejudice	Jane Austen	9780141439600	1813-01-28	Romance	Available	
1984	George Orwell	9780307474278	1949-06-08	Dystopian	Available	
Fahrenheit 451	Ray Bradbury	9781451673319	1953-10-19	Science Fiction	Available	
Brave New World	Aldous Huxley	9780060850524	1932-09-01	Science Fiction	Available	
The Feeling Good Book	Andrew Sharman	9780451524932	1980-07-01	Personal Development	Not Available	
The Feeling Good Book2	Andrew Sharman2	9780451524936	1980-07-01	Personal Development	Available	

POST /books – Admin panel manage books 2

The screenshot shows a browser window with three tabs: "Library", "localhost", and "Swagger UI". The "localhost" tab displays the Admin Panel interface for managing books. The main content area shows a success message: "1984 was returned ✓ (20:31:51)". Below this, the "Admin Panel" header is visible, followed by the "Manage Books" section. This section contains a table with the following data:

Title	Author	ISBN	Date
The Great Gatsby 2	F. Scott Fitzgerald	9780743273566	1930-04-10
The Great Gatsby	F. Scott Fitzgerald	9780743273565	1925-04-10
The Hunger Games	Suzanne Collins	9780439023528	2008-09-14
The Lord of the Rings	J.R.R. Tolkien	9780544003415	1954-07-29
The Road	Cormac McCarthy	9780307277671	2006-09-26
To Kill a Mockingbird	Harper Lee	9780061120084	1960-07-11

The right side of the screen shows the Network tab of the developer tools. It lists a single request: "books" (Status: 200, Type: xhr, Initiator: "ManageBooksPanelAction", Size: 0.8 kB, Time: 12 ms). The request URL is "http://localhost:8080/library/api/books". The developer tools also show a timeline with several short green bars representing network activity.

PUT /books/{id} with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various API endpoints. In the center, a specific PUT request to 'http://localhost:8080/library/api/books/91c0e1ce-9207-4bc4-89c4-9e6cc077ac8e' is selected. The 'Body' tab is active, displaying a JSON payload:

```
1 {  
2   "title": "The Feeling Good Book 2 Updated",  
3   "author": "Andrew Sharman",  
4   "isbn": "9780451524936",  
5   "publicationDate": "1980-07-01",  
6   "genre": "Personal Development"  
7 }
```

The response details show a status of 200 OK, 25 ms duration, and 638 B size. Below the body, the response JSON is displayed:

```
1 {  
2   "id": "91c0e1ce-9207-4bc4-89c4-9e6cc077ac8e",  
3   "title": "The Feeling Good Book 2 Updated",  
4   "author": "Andrew Sharman",  
5   "isbn": "9780451524936",  
6   "publicationDate": "1980-07-01",  
7   "genre": "Personal Development",  
8   "available": true  
9 }
```

The bottom of the screen shows the Windows taskbar with various icons and the system tray.

PUT /books/{id} with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' section. The 'PUT updateBook' endpoint is selected and highlighted in grey. The main workspace displays the configuration for this PUT request:

- Method:** PUT
- URL:** <http://localhost:8080/library/api/books/91c0e1ce-9207-4bc4-89c4-9e6cc077ac8e>
- Authorization:** Bearer Token (selected from dropdown)
- Token:** EZ_EDRfRiWwIMv5PWhpNRbsdSV-00 (copy/paste icon)
- Body:** Raw (checkbox checked)
- Response Status:** 403 Forbidden (highlighted in red)
- Response Headers:** 7 ms, 389 B
- Buttons:** Save Response, Copy, Share, Send

The status bar at the bottom shows the system tray and taskbar icons.

PUT /books/{id} – Admin panel manage books

The screenshot shows a web browser window with the URL `localhost:5173/admin`. The page title is "Admin Panel" with a shield icon. The main content area is titled "Manage Books" and displays a table of book records. The table has columns: Title, Author, ISBN, Date, Genre, Status, and Actions. The "Actions" column contains icons for edit, delete, and refresh. A purple button labeled "+ Add Book" is located at the top right of the table. On the left side, there is a sidebar with the "Admin Menu" header and several options: "Manage Users", "Manage Books" (which is highlighted in purple), "Borrow History", and "Overdue Books". The status bar at the bottom shows various system icons and the date/time `9.05.2025 20:36`.

Title	Author	ISBN	Date	Genre	Status	Actions
The Great Gatsby	F. Scott Fitzgerald	9780743273565	1925-04-10	Classic	Available	
The Hunger Games	Suzanne Collins	9780439023528	2008-09-14	Dystopian	Available	
The Lord of the Rings	J.R.R. Tolkien	9780544003415	1954-07-29	Fantasy	Available	
The Road	Cormac McCarthy	9780307277671	2006-09-26	Post-apocalyptic	Available	
To Kill a Mockingbird	Harper Lee	9780061120084	1960-07-11	Classic	Available	
Pride and Prejudice	Jane Austen	9780141439600	1813-01-28	Romance	Available	
1984	George Orwell	9780307474278	1949-06-08	Dystopian	Available	
Fahrenheit 451	Ray Bradbury	9781451673319	1953-10-19	Science Fiction	Available	
Brave New World	Aldous Huxley	9780060850524	1932-09-01	Science Fiction	Available	
The Feeling Good Book	Andrew Sharman	9780451524932	1980-07-01	Personal Development	Not Available	
The Great Gatsby 2 Updated	F. Scott Fitzgerald	9780743273566	10.04.1930	Classic	Available	
The Feeling Good Book 2 Updated	Andrew Sharman	9780451524936	1980-07-01	Personal Development	Available	

PUT /books/{id} – Admin panel manage books 2

The screenshot shows a browser window with three tabs: "Library", "localhost", and "Swagger UI". The "localhost" tab displays an admin panel for managing books. On the left, there's a sidebar with a search bar and some icons. The main content area shows a table of books:

Title	Author	ISBN
The Lord of the Rings	J.R.R. Tolkien	9780544003
The Road	Cormac McCarthy	9780307277
To Kill a Mockingbird	Harper Lee	9780061120
Pride and Prejudice	Jane Austen	9780141439
1984	George Orwell	9780307474
Fahrenheit 451	Ray Bradbury	9781451673
Brave New World	Aldous Huxley	9780060850
The Feeling Good Book	Andrew Sharman	9780451524
The Great Gatsby 2 Updated	F. Scott Fitzgerald	9780743273
The Feeling Good Book 2 Updated	Andrew Sharman	9780451524

On the right, the browser's developer tools Network tab is open, showing a single request to "http://localhost:8080/library/api/books/25447f36-6259-469d-95ab-cb1edd5c22d7" with a status of 200, type xhr, initiator "ManageBooksPanelA", size 0.8 kB, and time 11 ms.

DELETE /books/{id} with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' collection. In the center, a specific DELETE request to '/books/{id}' is selected. The request URL is set to `http://localhost:8080/library/api/books/91c0e1ce-9207-4bc4-89c4-9e6cc077ac8e`. The 'Authorization' tab is active, showing 'Bearer Token' selected, with the token value `dwBgQbrBm2ZCeKeWXPFnLzdFnstM` displayed. The response status is 204 No Content, with a response time of 13 ms and a size of 371 B. The bottom navigation bar shows several open tabs and system icons.

DELETE /books/{id} with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various API endpoints. In the center, a specific DELETE request to 'http://localhost:8080/library/api/books/25447f36-6259-469d-95ab-cb1edd5c22d7' is selected. The 'Authorization' tab is active, showing 'Bearer Token' selected. A token value 'EZ_EDRfRiWwIMv5PWhpNRbsdSV-00' is displayed in a highlighted box. The response status is '403 Forbidden'. At the bottom, the taskbar shows several open applications including Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a system clock indicating 20:38 on 9.05.2025.

Project

New Import

HTTP Library / deleteBook

DELETE http://localhost:8080/library/api/books/25447f36-6259-469d-95ab-cb1edd5c22d7

Params Authorization Headers (7) Body Scripts Settings

Auth Type: Bearer Token

Token: EZ_EDRfRiWwIMv5PWhpNRbsdSV-00

403 Forbidden

Body Cookies Headers (13) Test Results (0/1)

Raw Preview Visualize

Postbot Runner Start Proxy Cookies Vault Trash

20:38 9.05.2025

DELETE /books/{id} – Admin panel manage books

localhost:5173 web sitesinin mesajı

Are you sure you want to delete this book?

Tamam iptal

Title	Author	ISBN	Date	Genre	Status	Actions
The Great Gatsby	F. Scott Fitzgerald	9780743273565	1925-04-10	Classic	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
The Hunger Games	Suzanne Collins	9780439023528	2008-09-14	Dystopian	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
The Lord of the Rings	J.R.R. Tolkien	9780544003415	1954-07-29	Fantasy	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
The Road	Cormac McCarthy	9780307277671	2006-09-26	Post-apocalyptic	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
To Kill a Mockingbird	Harper Lee	9780061120084	1960-07-11	Classic	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Pride and Prejudice	Jane Austen	9780141439600	1813-01-28	Romance	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
1984	George Orwell	9780307474278	1949-06-08	Dystopian	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Fahrenheit 451	Ray Bradbury	9781451673319	1953-10-19	Science Fiction	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
Brave New World	Aldous Huxley	9780060850524	1932-09-01	Science Fiction	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
The Feeling Good Book	Andrew Sharman	9780451524932	1980-07-01	Personal Development	Not Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
The Great Gatsby 2 Updated	F. Scott Fitzgerald	9780743273566	1930-04-10	Classic	Available	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

Windows Taskbar icons: Library, Dosya Ge..., rol dağılı..., docker ko..., library-m..., pgAdmin 4, deleteBo..., axiosInsta..., Container..., 23°C, 9.05.2025, 20:39

DELETE /books/{id} – Admin panel manage books 2

The screenshot shows a web browser window with three tabs: "Library", "localhost", and "Swagger UI". The "localhost" tab displays an admin panel for managing books. On the left, there is a sidebar with icons for file operations like copy, move, delete, and refresh. The main content area shows a table of books:

	Date	Genre	Status	Action Buttons
0	09-14	Dystopian	Available	[Edit] [Delete] [Details]
5	1954-07-29	Fantasy	Available	[Edit] [Delete] [Details]
1	2006-09-26	Post-apocalyptic	Available	[Edit] [Delete] [Details]
4	1960-07-11	Classic	Available	[Edit] [Delete] [Details]
0	1813-01-28	Romance	Available	[Edit] [Delete] [Details]
8	1949-06-08	Dystopian	Available	[Edit] [Delete] [Details]
9	1953-10-19	Science Fiction	Available	[Edit] [Delete] [Details]
4	1932-09-01	Science Fiction	Available	[Edit] [Delete] [Details]
2	1980-07-01	Personal Development	Not Available	[Edit] [Delete] [Details]

The "Network" tab in the developer tools shows an XHR request for a book deletion:

Name	Status	Type	Initiator	Size	Time
25447f36-6259-469d-95ab-cb1... http://localhost:8080/library/api/books/25447f36-6259-469d-95ab-cb1edd5c22d7	204	xhr	ManageBooksPanelA...	0.5 kB	10 ms

At the bottom, the taskbar shows various open applications including Dosya Ge..., pgAdmin 4, and docker ko... The system tray indicates a temperature of 23°C and a date of 9.05.2025.

GET /borrow-records with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' collection. The main area displays a successful GET request to `/borrow-records`. The 'Authorization' tab is selected, showing a dropdown set to 'Bearer Token' with a token value: `dwBgQbrBm2ZCeKeWXPFnLzdFnstM`. The response status is `200 OK` with a response time of 12 ms and a size of 1.23 KB. The response body is a JSON array containing two borrow records:

```
[{"id": "c48cd897-3cccd-4ea6-bf43-2f376f16338f", "bookTitle": "The Great Gatsby", "userName": "Berik", "userId": "df7e6154-9f56-43c7-8be5-52998959639c", "bookId": "3be45ae8-60e9-410d-8f82-8ba20e5c10ff", "borrowDate": "2025-04-19", "dueDate": "2025-04-29", "returnDate": null, "returned": false}, {"id": "09324569-0093-45c9-b7d9-1dbb2ed578be", "bookTitle": "The Feeling Good Book", "userName": "Hasan", "userId": "09324569-0093-45c9-b7d9-1dbb2ed578be", "bookId": "3be45ae8-60e9-410d-8f82-8ba20e5c10ff", "borrowDate": "2025-04-19", "dueDate": "2025-04-29", "returnDate": null, "returned": false}]
```

At the bottom, the taskbar shows several open tabs and system icons.

GET /borrow-records with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections and environments. The main workspace displays a collection named "HTTP Library / getAllBorrowRecords". A specific GET request for "/borrow-records" is selected, with the URL set to "http://localhost:8080/library/api/borrow-records". The "Authorization" tab is active, showing "Bearer Token" selected. A token value "EZ_EDRfRiWwIMv5PWhpNRbsdSV-00" is displayed in a highlighted box. The response section shows a red "403 Forbidden" status with a response size of 389 B. The bottom taskbar includes icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a search bar.

GET /borrow-records – Admin panel borrow history

The screenshot shows a Microsoft Edge browser window with the following details:

- Address Bar:** localhost:5173/admin
- Tab Bar:** Library, localhost:8080/library/api/book, Swagger UI
- Header:** LibraryApp, Book Services, Admin, Profile, Logout
- Message Bar:** The Great Gatsby was borrowed (20:46:05)
- Left Sidebar (Admin Menu):**
 - Manage Users
 - Manage Books
 - Borrow History** (selected)
 - Overdue Books
- Center Content (Admin Panel):**

Admin Panel

Manage users, books, and borrowing activity in the system.

Borrow History

User	Book	Borrowed	Due	Returned	Return Date	User Info
Hasan	The Feeling Good Book	2025-04-19	2025-04-29	No	-	Hasan i
Berk	The Great Gatsby	2025-05-09	2025-05-16	No	-	Berk i
Berk	The Hunger Games	2025-05-09	2025-05-16	Yes	2025-05-09	Berk i
- Taskbar:** Shows various open applications including Docker, pgAdmin 4, and Container... along with the date and time (20:46, 9.05.2025).

GET /borrow-records – Admin panel borrow history 2

The screenshot shows a browser window with three tabs: "Library" (active), "localhost:8080/library/api/book", and "Swagger UI". The main content area displays the "LibraryApp" Admin Panel. The title bar says "Admin Panel" with a shield icon. Below it, a message says "Pride and Prejudice was returned ✓ (20:46:05)". The main section is titled "Borrow History" with a clock icon. It contains a table with four columns: User, Book, Borrowed, and Due. The table has three rows:

User	Book	Borrowed	Due
Hasan	The Feeling Good Book	2025-04-19	2025-04-29
Berk	The Great Gatsby	2025-05-09	2025-05-16
Berk	The Hunger Games	2025-05-09	2025-05-16

The browser's developer tools Network tab is open, showing a single request to "http://localhost:8080/library/api/borrow-records" with a status of 200 ms. The initiator is "BorrowHistoryPanelArea". The bottom of the screen shows the Windows taskbar with various icons.

GET /borrow-records/user/{userId} with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. The left sidebar displays a project structure with various collections, environments, flows, and history. The main workspace is titled "HTTP Library / getBorrowRecordsByUserId". A GET request is selected, with the URL set to `http://localhost:8080/library/api/borrow-records/user/152e474e-c906-48bc-a69a-bc518f932994`. The "Authorization" tab is active, showing "Bearer Token" selected and a token value "dwBgQbrBm2ZCeKeWXPFnLzdFnstM" entered. The "Body" tab shows a JSON response with two items, each representing a borrow record. The first item is for "The Great Gatsby" by "Berk". The second item is for "The Hunger Games" by "Berk". Both records have a due date of "2025-05-16" and a return date of "null". The status bar at the bottom indicates the response was successful (200 OK) with a 19 ms response time and 981 B size.

```
[{"id": "c48cd897-3cccd-4ea6-bf43-2f376f16338f", "bookId": "e9185c46-d706-4c7a-9452-8093ce4702c1", "borrowDate": "2025-05-09", "dueDate": "2025-05-16", "returnDate": null, "returned": false}, {"id": "e47a7da6-00ff-4cd8-b8d5-992fb2ca1bdc", "bookId": "e9185c46-d706-4c7a-9452-8093ce4702c1", "borrowDate": "2025-05-09", "dueDate": "2025-05-16", "returnDate": null, "returned": false}]
```

GET /borrow-records/user/{userId} with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections and environments. The main area displays a collection named "HTTP Library / getBorrowRecordsByUserId". A specific GET request is selected, which has failed with a 403 Forbidden status. The request URL is `http://localhost:8080/library/api/borrow-records/user/152e474e-c906-48bc-a69a-bc518f932994`. The "Authorization" tab is active, showing "Bearer Token" selected, and the token value is `EZ_EDRfRiWwIMv5PWhpNRbsdSV-00`. The response body is empty, indicating the failure.

GET /borrow-records/user/{userId} – Admin panel borrow history details

The screenshot shows a web browser window with three tabs: "Library", "localhost:8080/library/api/book", and "Swagger UI". The address bar indicates the URL is "localhost:5173/admin". A modal dialog box titled "Details" is displayed, containing two separate sections of borrow record information.

Record 1:

BookTitle:	The Great Gatsby
UserName:	Berk
Id:	c48cd897-3ccd-4ea6-bf43-2f3 76f16338f
UserId:	152e474e-c906-48bc-a69a-bc 518f932994
BookId:	e9185c46-d706-4c7a-9452-80 93ce4702c1
BorrowDate:	2025-05-09
DueDate:	2025-05-16
ReturnDate:	null
Returned:	No

Record 2:

BookTitle:	The Hunger Games
UserName:	Berk
Id:	e47a7da6-00ff-4cd8-b8d5-992 fb2ca1bdc
UserId:	152e474e-c906-48bc-a69a-bc 518f932994
BookId:	33206a7f-2f62-4e35-ab89-98d b53e6352b
BorrowDate:	2025-05-09
DueDate:	2025-05-16
ReturnDate:	2025-05-09
Returned:	Yes

GET /borrow-records/user/{userId} – Admin panel borrow history details 2

The screenshot shows a browser window with three tabs: 'Library', 'localhost:8080/library/api/book', and 'Swagger UI'. The main content area displays a modal titled 'Details' containing two sets of borrowing information. The first set is for 'The Great Gatsby' borrowed by 'Berk' (Id: c48cd897-3ccd-4ea6-bf43-2f376f16338f, UserId: 152e474e-c906-48bc-a69a-bc518f932994, BookId: e9185c46-d706-4c7a-9452-8093ce4702c1, BorrowDate: 2025-05-09, DueDate: 2025-05-16, ReturnDate: null, Returned: No). The second set is for 'The Hunger Games' borrowed by 'Berk' (Id: e47a7da6-00ff-4cd8-b8d5-992fb2ca1bdc, UserId: 152e474e-c906-48bc-a69a-bc518f932994, BookId: 33206a7f-2f62-4e35-ab89-98db53e6352b, BorrowDate: 2025-05-09, DueDate: 2025-05-16, ReturnDate: 2025-05-09, Returned: Yes). The Network tab in the developer tools shows a single XHR request to 'http://localhost:8080/library/api/borrow-records/user/152e474e-c906-48bc-a69a-bc518f932994' with a status of 200 and a response size of 1.1 kB. The browser taskbar at the bottom includes icons for File, Search, Task View, Google Chrome, Dosya Ge..., Spotify, Docker, library-m..., pgAdmin 4, getBorro..., axiosInsta..., Container..., Weather (23°C), Date (9.05.2025), and Time (20:50).

Details

BookTitle:	The Great Gatsby
UserName:	Berk
Id:	c48cd897-3ccd-4ea6-bf43-2f376f16338f
UserId:	152e474e-c906-48bc-a69a-bc518f932994
BookId:	e9185c46-d706-4c7a-9452-8093ce4702c1
BorrowDate:	2025-05-09
DueDate:	2025-05-16
ReturnDate:	null
Returned:	No

BookTitle:	The Hunger Games
UserName:	Berk
Id:	e47a7da6-00ff-4cd8-b8d5-992fb2ca1bdc
UserId:	152e474e-c906-48bc-a69a-bc518f932994
BookId:	33206a7f-2f62-4e35-ab89-98db53e6352b
BorrowDate:	2025-05-09
DueDate:	2025-05-16
ReturnDate:	2025-05-09
Returned:	Yes

1 / 2 requests | 1.1 kB / 1.1 kB transferred | 0.6 kB / 0.6 kB resources

GET /borrow-records/me with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. The left sidebar displays a collection named "Gear Clone - Product Service" containing various API endpoints. The main workspace shows a specific endpoint: `HTTP Library / getOwnBorrowRecords`. This endpoint is a `GET` request to `http://localhost:8080/library/api/borrow-records/me`. The `Authorization` tab is selected, showing a dropdown set to `Bearer Token` and a text input field containing the token `dwBgQbrBm2ZCeKeWXPFnLzdFnstM`. The response section shows a `200 OK` status with a response body of `[]`.

GET /borrow-records/me with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. The left sidebar displays a project structure with various API endpoints listed under 'Library'. The main workspace shows a specific GET request for '/borrow-records/me' with a PATRON token. The response body is displayed as JSON, showing two borrow records.

Request URL: http://localhost:8080/library/api/borrow-records/me

Authorization: Bearer Token (Token value:)x6ZmvJku_Fpv6m92jcwQArnnnjqyA0

Response Body (JSON):

```
[{"id": "c48cd897-3cccd-4ea6-bf43-2f376f16338f", "bookTitle": "The Great Gatsby", "userName": "Berk", "userId": "152e474e-c906-48bc-a69a-bc518f932994", "bookId": "e9185c46-d706-4c7a-9452-8093ce4702c1", "borrowDate": "2025-05-09", "dueDate": "2025-05-16", "returnDate": null, "returned": false}, {"id": "e47a7da6-00ff-4cd8-b8d5-992fb2ca1bdc", "bookTitle": "The Hunger Games", "userName": "Berk", "userId": "152e474e-c906-48bc-a69a-bc518f932994", "bookId": "e9185c46-d706-4c7a-9452-8093ce4702c1", "borrowDate": "2025-05-09", "dueDate": "2025-05-16", "returnDate": null, "returned": false}]
```

GET /borrow-records/me – My Books

The screenshot shows a web browser window with three tabs at the top: 'Library' (active), 'localhost:8080/library/api/book' (inactive), and 'Swagger UI' (inactive). The address bar shows 'localhost:5173/book-services'. The main content area has a purple header bar with the text 'LibraryApp' on the left and 'Book Services', 'Profile', and 'Logout' on the right.

The main content area features a sidebar on the left with a 'Menu' icon and three options: 'All Books' (light blue background), 'My Books' (purple background, currently selected), and 'Borrow History' (light blue background).

The central area has a title 'Book Services' with a blue book icon. Below it, a message says 'View all available books, borrow new ones, and track your book history.'

A sub-section titled 'My Books' with a green book icon displays two entries:

- The Great Gatsby** - Borrowed (Due: 2025-05-16) Return
- The Hunger Games** - Returned (Returned: 2025-05-09)

The taskbar at the bottom shows various open applications and system icons, including a search bar, file explorer, Spotify, Docker, pgAdmin, and a terminal window showing '23°C 9.05.2025'.

GET /borrow-records/me – My Books 2

The screenshot shows a browser window with three tabs: "Library" (active), "localhost:8080/library/api/book" (inactive), and "Swagger UI" (inactive). The main content area displays the "LibraryApp" interface, specifically the "Book Services" section. A message at the top states "The Hunger Games was returned ✅ (20:52:49)". Below this, there's a "My Books" section with two entries:

- The Great Gatsby** - Borrowed [Return] (Due: 2025-05-16)
- The Hunger Games** - Returned (Returned: 2025-05-09)

To the right of the application interface, the browser's developer tools are open, specifically the Network tab. It shows a single request named "me" with the following details:

Name	Status	Type	Initiator	Size	Time
me	200	xhr	MyBooksPanel.jsx:18	1.1 kB	10 ms

Below the table, the URL "http://localhost:8080/library/api/borrow-records/me" is highlighted. At the bottom of the browser window, the taskbar shows various open applications, and the system tray indicates the date and time as "9.05.2025".

GET /borrow-records/overdue with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' collection. The main area displays a specific request for 'getOverdueRecords'. The 'Authorization' tab is selected, showing 'Bearer Token' as the type and a token value: `!IWxPZzEkO-x1zOinR9A1S99puzA4gE`. The 'Body' tab shows a JSON response with one item:

```
1 [  
2 {  
3     "bookTitle": "The Feeling Good Book",  
4     "userName": "Hasan",  
5     "id": "09324569-0093-45c9-b7d9-1dbb2ed578be",  
6     "userId": "df7e6154-9f56-43c7-8be5-52998959639c",  
7     "bookId": "3be45ae0-60e9-410d-8f82-8ba20e5c10ff",  
8     "borrowDate": "2025-04-19",  
9     "dueDate": "2025-04-29",  
10    "returnDate": null,  
11    "returned": false  
12 }]  
13 ]
```

The status bar at the bottom indicates a 200 OK response with 8 ms latency and 705 B size.

GET /borrow-records/overdue with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the sidebar includes 'Project' (selected), 'Collections' (with 'Library'), 'Environments', 'Flows', and 'History'. The main area displays a 'Library / getOverdueRecords' collection. A specific GET request for '/borrow-records/overdue' is selected, with the URL set to 'http://localhost:8080/library/api/borrow-records/overdue'. The 'Authorization' tab is active, showing 'Bearer Token' selected in the dropdown. A token value '(x6ZmvJku_Fpv6m92jcwQArnnnjqyA0)' is displayed in a text input field. The 'Body' tab is selected, showing a raw JSON response with a single digit '1'. The status bar at the bottom indicates a '403 Forbidden' error with a 7 ms response time and 389 B size.

GET /borrow-records/overdue – Admin panel overdue books

The screenshot shows a Microsoft Edge browser window with three tabs: "Library" (active), "localhost:8080/library/api/book" (inactive), and "Swagger UI" (inactive). The address bar shows "localhost:5173/admin". The main content area is titled "Admin Panel" and displays the "Overdue Books" section. A message at the top says "To Kill a Mockingbird was returned ✅ (20:56:31)". The "Overdue Books" table has the following data:

User ID	User	Book	Borrowed	Due	Returned	Return Date
df7e6154-9f56-43c7-8be5-52998959639c	Hasan	The Feeling Good Book	2025-04-19	2025-04-29	No	-

The left sidebar, titled "Admin Menu", includes links for "Manage Users", "Manage Books" (which is currently selected), "Borrow History", and "Overdue Books". The bottom taskbar shows various open applications and system status.

GET /borrow-records/overdue – Admin panel overdue books 2

The screenshot shows a browser window with three tabs: "Library" (active), "localhost:8080/library/api/book", and "Swagger UI". The main content area displays the "LibraryApp Admin Panel" with a purple header and a sidebar icon. Below the header, a message says "The Great Gatsby was borrowed (20:56:31)". The main section is titled "Overdue Books" and contains a table with one row:

User ID	User	Book	B
df7e6154-9f56-43c7-8be5-52998959639c	Hasan	The Feeling Good Book	21!

The developer tools Network tab shows a request for "overdue" at "http://localhost:8080/library/api/borrow-records/overdue" with a status of 200 OK. The initiator is "OverdueBooksAdmin". The request took 7 ms.

At the bottom, the taskbar shows various open applications including Dosya Ge..., pgAdmin 4, getOverdue..., axiosInsta..., Container..., and Docker icons.

GET /borrow-records/overdue/report with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. In the center, a specific API endpoint is selected: `HTTP Library / getOverdueRecordsReport`. The request method is `GET`, and the URL is `http://localhost:8080/library/api/borrow-records/overdue/report`. The 'Authorization' tab is active, showing 'Bearer Token' selected. A token value is displayed in a yellow-bordered input field: `!IWxPZzEkO-x1zOinR9A1S99puzA4gE`. Below the authorization section, a note states: "The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization." The 'Body' tab is selected, showing the raw response content:

```
1 | Overdue Book Report
2 =====
3 User: Hasan (ID: df7e6154-9f56-43c7-8be5-52998959639c)
4 Book: The Feeling Good Book (ID: 3be45ae8-60e9-410d-8f82-8ba20e5c10ff)
5 Borrowed: 2025-04-19
6 Due Date: 2025-04-29
7 Returned: X No
8 -----
9
```

The response status is `200 OK` with a response time of `22 ms` and a size of `699 B`. The bottom navigation bar includes links for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a search bar.

GET /borrow-records/overdue/report with PATRON Token – (403 Forbidden)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various collections, environments, flows, and history. The main workspace displays an 'HTTP Library / getOverdueRecordsReport' collection. A specific GET request for '/library/api/borrow-records/overdue/report' is selected. The 'Authorization' tab is active, showing 'Bearer Token' as the auth type with a token value: `)x6ZmvJku_Fpv6m92jcwQArnnnjqyA0`. The response status is 403 Forbidden, with a 5 ms duration and 389 B size. The bottom taskbar includes icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and system notifications.

Project

HTTP Library / getOverdueRecordsReport

GET http://localhost:8080/library/api/borrow-records/overdue/report

Params Authorization Headers (7) Body Scripts Settings

Auth Type: Bearer Token

Token: `)x6ZmvJku_Fpv6m92jcwQArnnnjqyA0`

403 Forbidden 5 ms 389 B Save Response

Body Cookies Headers (13) Test Results (0/1)

Raw Preview Visualize

1

Online Find and replace Console

Postbot Runner Start Proxy Cookies Vault Trash

27°C 9.05.2025

GET /borrow-records/overdue/report – Admin panel overdue books

Brave New World was returned ✓ (20:56:31)

Admin Panel

Manage users, books, and borrowing activity in the system.

Overdue Books

User ID	User	Book	Borrowed	Due	Returned	Return Date
df7e6154-9f56-43c7-8be5-52998959639c	Hasan	The Feeling Good Book	2025-04-19	2025-04-29	No	-

Generate Overdue Report

Overdue Books

Brave New World was returned ✓ (20:56:31)

20:59
9.05.2025

GET /borrow-records/overdue/report – Admin panel overdue books 2

The screenshot shows a browser window with three tabs: "Library", "localhost:8080/library/api/book", and "Swagger UI". The main content area displays an admin panel for managing borrowing activity. A modal dialog is open, showing a table with the following data:

ok	Borrowed	Due	Returned	Return Date
e eling	2025-04-19	2025-04-29	✗ No	-
ok				

Below the table is a "Generate Overdue Report" button. The browser's developer tools Network tab shows two files being downloaded:

- overdue_report (6).txt (275 B)
- overdue_report (5).txt (275 B)

The status bar at the bottom of the screen shows the following information:

1 / 2 requests | 0.8 kB / 0.8 kB transferred | 0.3 kB / 0.3 kB resources

Windows taskbar icons include: Library, Dosya Ge..., rol dağılı..., docker ko..., library-m..., pgAdmin 4, getOverd..., axiosInsta..., Container..., 27°C, 9.05.2025, 20:59.

POST /borrow-records with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various API endpoints under 'Library'. In the center, a 'POST' request is selected for the endpoint `/borrowBook`. The 'Body' tab is active, showing a JSON payload:

```
1 {
2   "userId": "152e474e-c906-48bc-a69a-bc518f932994",
3   "bookId": "f4a71b5b-27bf-4398-802c-6e93b3b745c1",
4   "borrowDate": "2025-05-09",
5   "dueDate": "2025-05-16"
6 }
```

The response details show a status of **200 OK**, a duration of **24 ms**, and a size of **695 B**. The response body is also displayed in JSON format:

```
1 {
2   "bookTitle": "Fahrenheit 451",
3   "userName": "Berk",
4   "id": "331e0f1a-817b-45d9-8c5d-9b7740417146",
5   "userId": "152e474e-c906-48bc-a69a-bc518f932994",
6   "bookId": "f4a71b5b-27bf-4398-802c-6e93b3b745c1",
7   "borrowDate": "2025-05-09",
8   "dueDate": "2025-05-16",
9   "returnDate": null,
10  "returned": false
11 }
```

At the bottom, the taskbar shows several open windows including Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a system tray with icons for Postman, Docker, pgAdmin, borrowBook, and Container.

POST /borrow-records with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections and environments. The main workspace displays a POST request for the endpoint `/borrow-records`. The 'Authorization' tab is selected, showing a dropdown set to 'Bearer Token' with a token value: `)x6ZmvJku_Fpv6m92jcwQArnnnjqyA0`. The 'Body' tab shows a JSON payload:

```
1 {  
2   "bookTitle": "1984",  
3   "userName": "Berk",  
4   "id": "835169bd-adf6-4b44-aed6-4694a1fc4437a",  
5   "userId": "152e474e-c906-48bc-a69a-bc518f932994",  
6   "bookId": "3232f429-bce4-4ade-bdd3-b3576a397070",  
7   "borrowDate": "2025-05-09",  
8   "dueDate": "2025-05-16",  
9   "returnDate": null,  
10  "returned": false  
11 }
```

The response status is 200 OK, with a response time of 23 ms and a size of 685 B. The bottom navigation bar includes links for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a search bar.

POST /borrow-records – All Books & Home

localhost:8080/library/api/book x | Swagger UI x | +

localhost:5173/book-services

Menu

All Books

My Books

Borrow History

Book Services

View all available books, borrow new ones, and track your book history.

Filter Books

Search by title... Title

The Lord of the Rings
Author: J.R.R. Tolkien
ISBN: 9780544003415
Genre: Fantasy
Available

The Road
Author: Cormac McCarthy
ISBN: 9780307277671
Genre: Post-apocalyptic
Available

To Kill a Mockingbird
Author: Harper Lee
ISBN: 9780061120084
Genre: Classic
Available

Pride and Prejudice
Author: Jane Austen
ISBN: 9780141439600
Genre: Romance
Available

Brave New World
Author: Aldous Huxley
ISBN: 9780060850524
Genre: Science Fiction
Available

The Feeling Good Book
Author: Andrew Sharman
ISBN: 9780451524932
Genre: Personal Development
Not Available

The Great Gatsby

The Hunger Games

Fahrenheit 451

Windows Taskbar: Library, Dosya Ge..., Spotify, docker ko..., library-m..., pgAdmin 4, borrowB..., axiosInsta..., Container..., 27°C, 9.05.2025, 21:02

POST /borrow-records – All Books & Home 2

The screenshot shows a browser window with three tabs: "Library", "localhost:8080/library/api/book", and "Swagger UI". The main content area displays a modal dialog with a checkmark icon and the text "Borrowed!". Below it, a message states: "You have successfully borrowed 'The Lord of the Rings'." A "Close" button is at the bottom of the modal. The "localhost:8080/library/api/book" tab is active. In the top right corner of the browser, there is a developer tools Network tab open, showing a single request named "borrow-records" with a status of 200, type "xhr", initiator "BookCard.jsx:32", size 0.9 kB, and time 18 ms. The URL for this request is http://localhost:8080/library/api/borrow-records. The status bar at the bottom of the screen shows various icons and the date/time 9.05.2025.

localhost:5173/book-services

Dimensions: iPhone SE ▾ 375 x 667 97% ▾ No throttling ▾

Network

Name Status Type Initiator Size Time

borrow-records 200 xhr BookCard.jsx:32 0.9 kB 18 ms
http://localhost:8080/library/api/borrow-records

1 / 2 requests | 0.9 kB / 0.9 kB transferred | 0.3 kB / 0.3 kB resources

21:03 9.05.2025

PUT /borrow-records/return/{id} with LIBRARIAN Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar has 'Project' selected, displaying a list of API endpoints. The main area shows a 'PUT' request to 'http://localhost:8080/library/api/borrow-records/return/331e0f1a-817b-45d9-8c5d-9b7740417146'. The 'Authorization' tab is selected, showing 'Bearer Token' and a token value. The response body is displayed as JSON, indicating a successful 200 OK status.

PUT http://localhost:8080/library/api/borrow-records/return/331e0f1a-817b-45d9-8c5d-9b7740417146

Authorization: Bearer Token
Token: !IWxPZzEkO-x1zOinR9A1S99puzA4gE

Body:

```
[{"bookTitle": "Fahrenheit 451", "userName": "Berk", "id": "331e0f1a-817b-45d9-8c5d-9b7740417146", "userId": "152e474e-c906-48bc-a69a-bc518f932994", "bookId": "f4a71b5b-27bf-4398-802c-6e93b3b745c1", "borrowDate": "2025-05-09", "dueDate": "2025-05-16", "returnDate": "2025-05-09", "returned": true}]
```

PUT /borrow-records/return/{id} with PATRON Token – (200 OK)

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections and environments. The main area displays a collection named "HTTP Library / returnBook". A specific PUT request is selected, with the URL set to `http://localhost:8080/library/api/borrow-records/return/835169bd-adf6-4b44-aed6-4694afc4437a`. The "Authorization" tab is active, showing "Bearer Token" selected, with a token value displayed in a yellow-bordered field: `)x6ZmvJku_Fpv6m92jcwQArnnnjqyA0`. The "Body" tab shows a JSON response with the following content:

```
1 {  
2   "bookTitle": "1984",  
3   "userName": "Berk",  
4   "id": "835169bd-adf6-4b44-aed6-4694afc4437a",  
5   "userId": "152e474e-c906-48bc-a69a-bc518f932994",  
6   "bookId": "3232f429-bce4-4ade-bdd3-b3576a397070",  
7   "borrowDate": "2025-05-09",  
8   "dueDate": "2025-05-16",  
9   "returnDate": "2025-05-09",  
10  "returned": true  
11 }
```

The status bar at the bottom indicates a 200 OK response with 14 ms latency and 692 B size.

PUT /borrow-records/return/{id} – My Books

localhost:8080/library/api/book × | localhost:5173/book-services | Swagger UI

localhost:5173/book-services

LibraryApp

Book Services | Profile | Logout

1984 was borrowed (21:02:36)

Menu

All Books

My Books

Borrow History

Book Services

View all available books, borrow new ones, and track your book history.

My Books

Book Title	Status	Due Date	Action
The Great Gatsby	Borrowed	2025-05-16	Return
The Hunger Games	Returned	2025-05-09	
The Lord of the Rings	Borrowed	2025-05-16	Return
Fahrenheit 451	Returned	2025-05-09	
1984	Returned	2025-05-09	

21:05
9.05.2025

PUT /borrow-records/return/{id} – My Books 2

The screenshot shows a browser window with three tabs: "Library", "localhost:8080/library/api/book", and "Swagger UI". The main content area displays a modal dialog with a purple circular icon, the text "Returned!", and the message "You have successfully returned \"The Great Gatsby\"." A "Close" button is at the bottom of the modal. The "Network" tab in the developer tools is selected, showing a single XHR request to "http://localhost:8080/library/api/borrow-records/return/c48cd897-3ccd-4ea6-bf43-2f376f16338f" with a status of 200 ms. The browser's taskbar at the bottom shows various open applications.

Dimensions: iPhone SE ▾ 375 x 667 97% ▾ No throttling ▾

localhost:5173/book-services

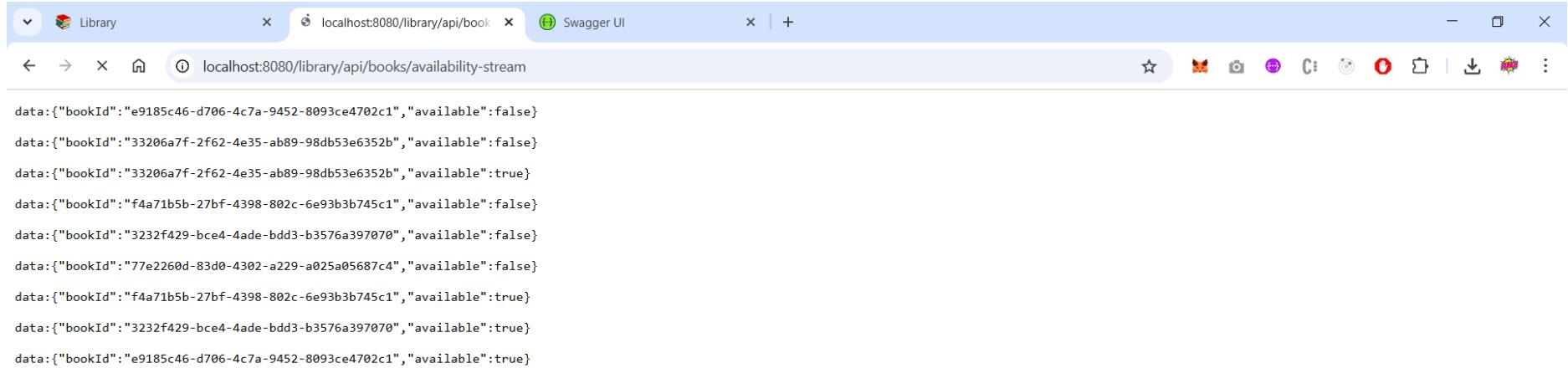
Network

Name	Status	Type	Initiator	Size	Time
c48cd897-3ccd-4ea6-bf43-2f376f16338f	200	xhr	MyBooksPanel.jsx:39	0.9 kB	12 ms

1 / 2 requests | 0.9 kB / 0.9 kB transferred | 0.3 kB / 0.3 kB resources

21:06 9.05.2025

GET /books/availability-stream (reactive)



The screenshot shows a browser window with three tabs: "Library", "localhost:8080/library/api/book", and "Swagger UI". The active tab displays the URL "localhost:8080/library/api/books/availability-stream". The page content is a JSON stream of book availability data, with each line representing a new event:

```
data:{"bookId":"e9185c46-d706-4c7a-9452-8093ce4702c1","available":false}
data:{"bookId":"33206a7f-2f62-4e35-ab89-98db53e6352b","available":false}
data:{"bookId":"33206a7f-2f62-4e35-ab89-98db53e6352b","available":true}
data:{"bookId":"f4a71b5b-27bf-4398-802c-6e93b3b745c1","available":false}
data:{"bookId":"3232f429-bce4-4ade-bdd3-b3576a397070","available":false}
data:{"bookId":"77e2260d-83d0-4302-a229-a025a05687c4","available":false}
data:{"bookId":"f4a71b5b-27bf-4398-802c-6e93b3b745c1","available":true}
data:{"bookId":"3232f429-bce4-4ade-bdd3-b3576a397070","available":true}
data:{"bookId":"e9185c46-d706-4c7a-9452-8093ce4702c1","available":true}
```



GET /books/availability-stream (reactive) 2

A real-time banner below the navbar displays book borrow/return events using Server-Sent Events (SSE).

The screenshot shows a web application titled "LibraryApp" running on localhost:5173. At the top, there are two browser tabs both labeled "localhost". The main content area has a purple header bar with the title "LibraryApp" on the left and links for "Book Services", "Login", and "Register" on the right. A real-time banner at the top of the main content area displays the message "The Hunger Games was returned ✅ (19:53:41)". Below the banner, the page features a "Welcome" section with a book icon and the text "Explore the books in our library." The main content is organized into a grid of eight book cards, each with a purple "Borrow" button.

Book Title	Author	ISBN	Genre	Status	Action
The Great Gatsby	F. Scott Fitzgerald	9780743273565	Classic	Available	Borrow
The Hunger Games	Suzanne Collins	9780439023528	Dystopian	Available	Borrow
The Lord of the Rings	J.R.R. Tolkien	9780544003415	Fantasy	Available	Borrow
The Road	Cormac McCarthy	9780307277671	Post-apocalyptic	Available	Borrow
To Kill a Mockingbird	Harper Lee	9780061120084	Classic	Available	Borrow
Pride and Prejudice	Jane Austen	9780141439600	Romance	Available	Borrow
1984	George Orwell	9780307474278	Dystopian	Available	Borrow
Fahrenheit 451	Ray Bradbury	9781451673319	Science Fiction	Available	Borrow

Swagger UI – Interactive API Documentation (<http://localhost:8080/library/api/swagger-ui/index.html>)

The screenshot shows the Swagger UI interface for the Library Management API. The top navigation bar includes tabs for 'Library' and 'localhost:8080/library/api/book'. The active tab is 'Swagger UI' with the URL 'localhost:8080/library/api/swagger-ui/index.html'. The main header features the 'Swagger' logo and the URL '/library/api/v3/api-docs'. A green 'Explore' button is on the right.

Library Management API 1.0 OAS 3.0

/library/api/v3/api-docs
OpenAPI documentation for the Spring Boot-based Library Management System
Contact Hasan

Servers: http://localhost:8080/library/api - Generated server url Authorize

user-controller

GET /users/{id} Get user by ID

PUT /users/{id} Update user information

DELETE /users/{id} Delete a user by ID

GET /users Get all users

GET /users/me Get details of the authenticated user

Taskbar icons include: Dosya Ge..., rol dağılı..., docker ko..., library-m..., pgAdmin 4, returnBo..., axiosInsta..., Container..., 24°C, 9.05.2025, 21:07.

Swagger UI 2

The screenshot shows the Swagger UI 2 interface running in a browser window. The URL in the address bar is `localhost:8080/library/api/swagger-ui/index.html`. The page displays two main sections: **borrow-record-controller** and **book-controller**.

borrow-record-controller

- PUT** `/borrow-records/return/{id}` Return a borrowed book
- GET** `/borrow-records` Get all borrow records
- POST** `/borrow-records` Borrow a book
- GET** `/borrow-records/user/{userId}` Get borrow records by user ID
- GET** `/borrow-records/overdue` Get overdue borrow records
- GET** `/borrow-records/overdue/report` Generate overdue report
- GET** `/borrow-records/me` Get borrow records of the authenticated user

book-controller

- GET** `/books/{id}` Get book by ID
- PUT** `/books/{id}` Update book information
- DELETE** `/books/{id}` Delete a book by ID
- GET** `/books` Get all books

The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray.

Swagger UI 3

The screenshot shows the Swagger UI 3 interface running in a browser window. The title bar indicates the URL is `localhost:8080/library/api/swagger-ui/index.html`. The main content area is divided into two sections: **book-controller** and **auth-controller**.

book-controller contains the following endpoints:

- GET /books/{id}** Get book by ID
- PUT /books/{id}** Update book information
- DELETE /books/{id}** Delete a book by ID
- GET /books** Get all books
- POST /books** Add a new book
- GET /books/search/title** Search books by title
- GET /books/search/isbn** Search books by ISBN
- GET /books/search/genre** Search books by genre
- GET /books/search/author** Search books by author

auth-controller contains the following endpoints:

- POST /auth/register** Register a new user
- POST /auth/login** Login a user

The browser's taskbar at the bottom shows various open tabs and system icons.

Swagger UI 4

The screenshot shows the Swagger UI 4 interface running in a web browser. The title bar indicates the window is titled "Swagger UI". The address bar shows the URL "localhost:8080/library/api/swagger-ui/index.html". The main content area is titled "Schemas" and lists several API schema definitions:

- AdminUserUpdateRequestDto >
- UserResponseDto >
- BorrowRecordResponseDto >
- BookRequestDto >
- BookResponseDto >
- BorrowRecordRequestDto >
- RegisterRequest >
- AuthResponse >
- AuthRequest >

The browser's taskbar at the bottom shows various open tabs and pinned icons, including "Dosya Ge...", "rol dağılı...", "docker ko...", "library-m...", "pgAdmin 4", "returnBo...", "axiosInsta...", "Container...", "24°C", and the date "9.05.2025".

Swagger UI 5

The screenshot shows the Swagger UI 5 interface running in a web browser. The title bar indicates the window is titled "Swagger UI". The address bar shows the URL "localhost:8080/library/api/swagger-ui/index.html". The main content area displays a vertical list of API models, each with a link to expand:

- BookRequestDto >
- BookResponseDto >
- BorrowRecordRequestDto >
- RegisterRequest >
- AuthResponse >
- AuthRequest >
- PageBookResponseDto >
- PageableObject >
- SortObject >

The browser's taskbar at the bottom shows various open tabs and pinned icons, including "Swagger UI", "Dosya Ge...", "rol dağılı...", "library-m...", "pgAdmin 4", "returnBo...", "axiosInsta...", "Container...", and "24°C". The system tray shows the date and time as "9.05.2025" and "21:09".

Unit Test – CustomUserDetailsServiceTest

The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "library-management". The "test/java" directory contains several test classes under the package "com.hasan.library_management". The class "CustomUserDetailsServiceTest" is currently selected and highlighted in the code editor.
- Code Editor:** The content of "CustomUserDetailsServiceTest.java" is displayed. It contains a single test method: "loadUserByUsername_shouldReturnUserDetails_whenUserExists()". The code uses Mockito annotations (@Mock, @InjectMocks) and the Spring Test framework (@Test). The test arranges a user repository to return a user with a specific email, acts by calling the service's loadUserByUsername method, and asserts that the returned UserDetails object matches the expected user details.
- Run Tab:** The "Run" tab is active, showing the test class "CustomUserDetailsServiceTest" and its methods: "loadUserByUsername_shouldThrowExcept" (1sec 186 ms), "loadUserByUsername_shouldReturnUserDetails" (96 ms), and "loadUserByUsername_shouldReturnUserDetails_whenUserExists" (1sec 282 ms). All tests have passed.
- Output Tab:** The output window displays the test results and some Java HotSpot VM warnings about dynamic agent loading and sharing support for boot loader classes.
- Bottom Bar:** The bottom bar shows the file path "library-management > src > test > java > com > hasan > library_management > service > CustomUserDetailsServiceTest", and various system status icons.

Unit Test – BookAvailabilityServiceTest

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The left sidebar shows the project structure under "library-management". It includes a "templates" folder, an "application.properties" file, and a "test" directory containing "java" and "resources" sub-directories. Inside "java", there are packages for "com.hasan.library_management.controller" and "com.hasan.library_management.service". The "BookAvailabilityServiceTest" class is selected in the "service" package.
- Code Editor:** The main editor window displays the code for `BookAvailabilityServiceTest.java`. The current method being tested is `publishAvailabilityChange_should EmitEvent()`. The code uses StepVerifier to verify the service's behavior when publishing availability changes.
- Run Tab:** The bottom-left tab bar shows the "Run" tab is active, with the configuration "BookAvailabilityServiceTest" selected.
- Toolbars and Status Bar:** Various toolbars are visible along the top and right edges. The status bar at the bottom provides build information: "library-management > src > test > java > com > hasan > library_management > service > BookAvailabilityServiceTest > publishAvailabilityChange_should EmitEvent". It also shows system details like "27:6 CRLF UTF-8 4 spaces", a timestamp "21:12 9.05.2025", and system icons for Docker, pgAdmin, and Container.

Unit Test – AuthServiceImplTest

The screenshot shows a Java project structure in the left sidebar under the 'Project' view. The current file is `AuthServiceImplTest.java`, which contains unit tests for the `AuthService`. The code uses Mockito annotations like `@ExtendWith(MockitoExtension.class)` and `@Mock` to mock dependencies such as `UserRepository`, `PasswordEncoder`, `JwtUtil`, and `AuthenticationManager`. It also includes an `@InjectMocks` annotation and a `AuthService` field.

In the bottom right corner of the editor, there are two yellow warning icons. The bottom status bar shows the path `library-management > src > test > java > com > hasan > library_management > service > impl > AuthServiceImplTest` and the terminal output:

```
21:12:38.150 [main] INFO com.hasan.library_management.service.impl.AuthServiceImpl -- Registering user with email: test@example.com
21:12:38.150 [main] WARN com.hasan.library_management.service.impl.AuthServiceImpl -- Registration failed. Email already registered
21:12:38.155 [main] INFO com.hasan.library_management.service.impl.AuthServiceImpl -- Registering user with email: test@example.com
21:12:38.155 [main] INFO com.hasan.library_management.service.impl.AuthServiceImpl -- User registered successfully: test@example.com
```

The bottom navigation bar includes icons for Swagger, Dosya Ge..., Docker, pgAdmin 4, returnBo..., axiosInsta..., Container..., and a weather widget showing 24°C at 9.05.2025.

```
29
30     @ExtendWith(MockitoExtension.class)  sirdashasan
31     class AuthServiceImplTest {
32
33         @Mock 5 usages
34         private UserRepository userRepository;
35
36         @Mock 1 usage
37         private PasswordEncoder passwordEncoder;
38
39         @Mock 2 usages
40         private JwtUtil jwtUtil;
41
42         @Mock 2 usages
43         private AuthenticationManager authenticationManager;
44
45         @InjectMocks 5 usages
46         private AuthServiceImpl authService;
```

Run AuthServiceImplTest

AuthServiceImplTest (com.hasan.library_management) 1 sec 399 ms

- Tests passed: 5 of 5 tests – 1 sec 399 ms
- login_shouldThrowException_whenAuther 1 sec 329 ms
- login_shouldReturnToken_whenCredentialsAreValid 22 ms
- login_shouldThrowException_whenUserNotFound 41 ms
- register_shouldThrowException_whenEmailAlreadyExists 3 ms
- register_shouldReturnToken_whenEmailIsNotRegistered 4 ms

Process finished with exit code 0

45:17 CRLF UTF-8 4 spaces

21:12:38.150 [main] INFO com.hasan.library_management.service.impl.AuthServiceImpl -- Registering user with email: test@example.com
21:12:38.150 [main] WARN com.hasan.library_management.service.impl.AuthServiceImpl -- Registration failed. Email already registered
21:12:38.155 [main] INFO com.hasan.library_management.service.impl.AuthServiceImpl -- Registering user with email: test@example.com
21:12:38.155 [main] INFO com.hasan.library_management.service.impl.AuthServiceImpl -- User registered successfully: test@example.com

24°C 9.05.2025

Unit Test – BookServiceImplTest

The screenshot shows a Java development environment with the following details:

- Project Structure:** The project is named "library-management" and contains a "main" module. The "test" module is expanded, showing "java" and "com.hasan.library_management" packages. Inside "com.hasan.library_management", there are "controller", "service", and "impl" packages. The "impl" package contains several test classes: AuthServiceImplTest, BookServiceImplTest (which is currently selected), BorrowRecordServiceImplTest, UserServiceImplTest, and BookAvailabilityServiceTest.
- Code Editor:** The code editor displays the "BookServiceImplTest.java" file. The code imports various Mockito annotations and defines a class named "BookServiceImplTest". It uses @Mock annotations to mock "BookRepository" and "BookMapper". It also uses @InjectMocks to inject "BookService".

```
package com.hasan.library_management.service.impl;

import ...;

import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.test.context.junit.jupiter.extension.ExtendWith;

@ExtendWith(MockitoExtension.class)
public class BookServiceImplTest {

    @Mock
    private BookRepository bookRepository;

    @Mock
    private BookMapper bookMapper;

    @InjectMocks
    private BookService bookService;

    private Book book; 62 usages
    private UUID bookId; 8 usages
}
```
- Run Tab:** The "Run" tab shows the test class "BookServiceImplTest" has been run. The output indicates 13 tests passed in 1 second and 417 milliseconds. The log shows several INFO messages from the "BookServiceImpl" class, such as deleting books with specific IDs and a WARN message about a book not found for deletion.
- Status Bar:** The status bar at the bottom shows the current time as 21:13, date as 09.05.2025, and system information like CRLF, UTF-8, 4 spaces, and a battery icon.

Unit Test – BorrowRecordServiceImplTest

The screenshot shows the VS Code interface with the following details:

- Project Explorer:** Shows the project structure under "library-management". The "test/java/com.hasan.library_management/service/impl" folder contains several test classes, with `BorrowRecordServiceImplTest.java` currently selected.
- Code Editor:** Displays the code for `BorrowRecordServiceImplTest.java`. The class is annotated with `@ExtendWith(MockitoExtension.class)`. It includes private fields for `BorrowRecordRepository`, `BookRepository`, `UserRepository`, and `BookAvailabilityService`, all of which are mocked.
- Run View:** Shows the test results for `BorrowRecordServiceImplTest`. It indicates 16 tests passed in 1 second and 499 milliseconds. The log output shows various INFO and WARN messages from the application code.
- Bottom Status Bar:** Provides information about the current file (`BorrowRecordServiceImplTest`), encoding (`CRLF`), line count (`31:1`), and date/time (`9.05.2025 21:14`).

Unit Test – UserServiceImplTest

The screenshot shows a Java development environment with the following details:

- Project Structure:** The project is named "library-management" and contains a "main" module. The "test" module is expanded, showing "java" and "service" packages. Under "service", there is an "impl" package which contains several test classes: AuthServiceImplTest, BookServiceImplTest, BorrowRecordServiceImplTest, and UserServiceImplTest. The "UserServiceImplTest.java" file is currently selected.
- UserServiceImplTest.java Content:**

```
26  class UserServiceImplTest {  
27      @Mock 10 usages  
28          private UserRepository userRepository;  
29  
30      @Mock 5 usages  
31          private UserMapper userMapper;  
32  
33      @InjectMocks 10 usages  
34          private UserServiceImpl userService;  
35  
36          private User user; 32 usages  
37          private UUID userId; 9 usages  
38  
39      @BeforeEach 1 sirdashasan  
40          void setUp() {  
41              userId = UUID.randomUUID();  
42              user = new User();  
43              user.setId(userId);  
44              user.setName("Hasan");  
45              user.setEmail("hasan@gmail.com");  
46      }  
47  }
```
- Run Tab:** The "Run" tab is active, showing the test configuration for "UserServiceImplTest".
- Output Tab:** The output shows the test results:
 - 9 tests passed: 9 of 9 tests – 1 sec 355 ms
 - Logs from the application:

```
21:14:43.679 [main] INFO com.hasan.library_management.service.impl.UserServiceImpl -- Updating user with ID: 4ee92647-ba14-  
21:14:43.680 [main] WARN com.hasan.library_management.service.impl.UserServiceImpl -- User not found with ID: 4ee92647-ba14  
21:14:43.686 [main] INFO com.hasan.library_management.service.impl.UserServiceImpl -- Updating user with ID: e70dfaee-fb38-  
21:14:43.686 [main] INFO com.hasan.library_management.service.impl.UserServiceImpl -- User updated successfully with ID: e7
```
 - Process finished with exit code 0
- Bottom Status Bar:** Shows the current time (21:14), date (9.05.2025), and system status (CRLF, UTF-8, 4 spaces).

Integration Test – AuthControllerTest

The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "library-management". It contains a main application class ("main") and a test directory. Inside the test directory, there are several Java test classes under the package "com.hasan.library_management.controller":
 - AuthControllerTest** (selected in the tree)
 - BookAvailabilityControllerTest
 - BookControllerTest
 - BorrowRecordControllerTest
 - UserControllerTest
- AuthControllerTest.java Content:**

```
23  @TestInstance(TestInstance.Lifecycle.PER_CLASS)
24  @TestPropertySource(properties = {
25      "server.servlet.context-path="
26  })
27  class AuthControllerTest {
28
29      @Autowired 6 usages
30      private MockMvc mockMvc;
31
32      @Autowired 6 usages
33      private ObjectMapper objectMapper;
34
35      @Autowired 1 usage
36      private UserRepository userRepository;
37
38      private final String email = "auth_test@example.com"; 5 usages
39      private final String password = "123456"; 3 usages
40
41      @BeforeEach
42      void setup() {
```
- Run Tab:** Shows a single test run: "AuthControllerTest" with a duration of 994 ms. All 5 tests passed.
- Output Tab:** Displays the log output for the test run, showing various Spring and Hibernate logs.
- System Tray:** Shows standard Windows system tray icons like taskbar, search, file explorer, and network.

Integration Test – BookAvailabilityControllerTest

The screenshot shows the VS Code interface with the following details:

- Project Explorer:** Shows the project structure under "library-management". The "test/java/com.hasan.library_management/controller" folder contains several test classes: AuthControllerTest, BookAvailabilityControllerTest (selected), BookControllerTest, BorrowRecordControllerTest, and UserControllerTest.
- Code Editor:** The active file is `BookAvailabilityControllerTest.java`. The code imports various Spring framework annotations and classes, including `@SpringBootTest`, `LocalServerPort`, `WebTestClient`, and `AssertionsForInterfaceTypes.assertThat`. It defines a test class `BookAvailabilityControllerTest` with a private attribute `BookAvailabilityService availabilityService`.
- Run View:** The "Run" tab is selected, showing the result of running the `BookAvailabilityControllerTest`. The test `streamAvailability_shouldEmitEvents()` passed in 1 sec 173 ms. The output window shows log entries from the application, including shutdown hooks and HikariDataSource metrics.
- Status Bar:** The status bar at the bottom shows the file path `library-management > src > test > java > com > hasan > library_management > controller > BookAvailabilityControllerTest`, the time `19:39`, and other system information like temperature and date.

Integration Test – BookControllerTest

The screenshot shows the VS Code interface with the following details:

- Project Explorer:** Shows the project structure under "library-management". The "test/java/com.hasan.library_management/controller" folder contains several test classes: AuthControllerTest, BookAvailabilityControllerTest, BookControllerTest (which is selected), BorrowRecordControllerTest, and UserControllerTest.
- Code Editor:** The active file is `BookControllerTest.java`. The code defines a test class `BookControllerTest` with various test methods for the BookController.
- Run View:** The "Run" tab is open, showing the result of running the `BookControllerTest`. It indicates 20 tests passed in 881ms. The log output shows standard Spring Boot startup logs and a message: "Mock data (users, books, and overdue borrow records) loaded successfully."
- Bottom Status Bar:** Displays the file path "library-management > src > test > java > com > hasan > library_management > controller > BookControllerTest", the status bar with "34:27 CRLF UTF-8 4 spaces", and the system tray with icons for Swagger, Dosya Ge..., Docker, pgAdmin 4, returnBo..., axiosInsta..., Container..., and a weather widget showing 24°C at 9.05.2025.

Integration Test – BorrowRecordControllerTest

The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "library-management" and contains a "main" module. The "test" module is expanded, showing sub-directories "java" and "service", and files including "AuthControllerTest", "BookAvailabilityControllerTest", "BookControllerTest", "BorrowRecordControllerTest" (which is selected), and "UserControllerTest".
- BorrowRecordControllerTest.java Content:** The code is a JUnit test for the BorrowRecordController. It uses Mockito to perform a POST request to "/auth/register" with JSON input. It then logs in using an AuthRequest with email "patron@example.com" and password "123456". Finally, it performs a POST request to "/auth/login" with the same JSON input and expects a successful response.
- Run Tab:** The "Run" tab is active, showing the test class "BorrowRecordControllerTest" and its methods: "getOwnRecords_shouldReturnList_whenUser", "returnBook_shouldReturnForbidden_whenNoTc", "getOwnRecords_shouldReturnForbidden_when", "generateOverdueReport_shouldReturnOk_wher", "borrowBook_shouldReturnOk_whenValidRequ", "getRecordsByUserId_shouldReturnList_when", "returnBook_shouldReturnOk_whenValidIdAnd", and "borrowBook_shouldReturnRarRequest whenD".
- Test Results:** The test results show 20 tests passed in 1 second and 650 milliseconds. The log output shows various Spring and Hibernate initialization messages.
- System Tray:** The taskbar at the bottom includes icons for Swagger, Dosya Ge..., rol dağılı..., docker ko..., library-m..., pgAdmin 4, returnBo..., axiosInsta..., Container..., and a weather widget showing 24°C and the date 9.05.2025.

Integration Test – UserControllerTest

The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "library-management" and contains a "main" module. The "test" module is expanded, showing sub-directories "java" and "service". Inside "java", there are several test classes: AuthControllerTest, BookAvailabilityControllerTest, BookControllerTest, BorrowRecordControllerTest, UserControllerTest (which is currently selected), and BookAvailabilityServiceTest.
- UserControllerTest.java Content:** The code defines a test class for the UserController. It imports various Spring testing annotations and classes, including TestPropertySource, MockMvc, and MvcResult. The class is annotated with @SpringBootTest, @AutoConfigureMockMvc, and @TestInstance(PER_CLASS). It also uses @TestPropertySource to set the context path. The class contains a private MockMvc field and an @Autowired annotation for it.
- Run Tab:** The "Run" tab is active, showing the test configuration for "UserControllerTest".
- Output Window:** The output window displays the test results:
 - A green checkmark indicates "Tests passed: 17 of 17 tests – 1 sec 222 ms".
 - Log entries from the application's logger show INFO messages related to user deletion and JPA entity closing.
 - The message "Process finished with exit code 0" is displayed at the bottom.
- Bottom Status Bar:** The status bar shows the current time (33:15), encoding (CRLF), character set (UTF-8), and code style (4 spaces). It also includes icons for Docker, pgAdmin 4, Swagger, and other development tools.

Service Layer Test Coverage

Element ^	Class, %	Method, %	Line, %	Branch, %
com.hasan.library_management	100% (11/11)	100% (64/64)	98% (239/24...)	94% (17/18)
controller	100% (5/5)	100% (24/24)	100% (35/35)	100% (0/0)
service	100% (6/6)	100% (40/40)	98% (204/20...)	94% (17/18)
impl	100% (4/4)	100% (38/38)	98% (197/201)	94% (17/18)
AuthService	100% (1/1)	100% (4/4)	100% (26/26)	100% (2/2)
BookServiceImpl	100% (1/1)	100% (13/13)	100% (50/50)	100% (2/2)
BorrowRecordServiceImpl	100% (1/1)	100% (12/12)	95% (91/95)	92% (13/14)
UserService	100% (1/1)	100% (9/9)	100% (30/30)	100% (0/0)
AuthService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
BookAvailabilityService	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
BookService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
BorrowRecordService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
CustomUserDetailsService	100% (1/1)	100% (1/1)	100% (6/6)	100% (0/0)
UserService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)

Controller Layer Test Coverage

Element ^	Class, %	Method, %	Line, %	Branch, %
com.hasan.library_management	100% (11/11)	100% (64/64)	98% (239/24...)	94% (17/18)
controller	100% (5/5)	100% (24/24)	100% (35/35)	100% (0/0)
AuthController	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
BookAvailabilityController	100% (1/1)	100% (1/1)	100% (3/3)	100% (0/0)
BookController	100% (1/1)	100% (9/9)	100% (14/14)	100% (0/0)
BorrowRecordController	100% (1/1)	100% (7/7)	100% (9/9)	100% (0/0)
UserController	100% (1/1)	100% (5/5)	100% (7/7)	100% (0/0)
service	100% (6/6)	100% (40/40)	98% (204/20...)	94% (17/18)

Service & Controller Layers Test Coverage

Coverage	CustomUserDetailsServiceTest		:	-	
Element ^		Class, %	Method, %	Line, %	Branch, %
com.hasan.library_management	100% (11/11)	100% (64/64)	98% (239/24...)	94% (17/18)	
controller	100% (5/5)	100% (24/24)	100% (35/35)	100% (0/0)	
service	100% (6/6)	100% (40/40)	98% (204/20...)	94% (17/18)	

Exceptions

GET /users/{id} – Query user with invalid ID

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main workspace displays a collection named "Library / getUserById". A specific GET request is selected, which has failed with a 404 Not Found error. The request URL is `http://localhost:8080/library/api/users/084d6d8e-02ad-431d-be31-4423320f7857`. The "Authorization" tab is active, showing "Bearer Token" selected. The response body is displayed as JSON:

```
1 {  
2   "errors": [  
3     {  
4       "field": "internal",  
5       "message": "User not found with id: 084d6d8e-02ad-431d-be31-4423320f7857"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:10:11.325564444",  
9   "status": 404  
10 }
```

The status bar at the bottom shows the following icons and information: Online, Find and replace, Console, Postbot, Runner, Start Proxy, Cookies, Vault, Trash, Windows taskbar icons (File Explorer, Task View, Start button, Search, Edge browser), a system tray icon (Getir Java S...), a progress bar (Getir Bitirm...), a file icon (exceptions), a message icon (library-man...), a container icon (Containers ...), a Docker icon (docker kom...), a library manager icon (Library Man...), a red exclamation mark icon (getUserById...), a weather icon (23°C), a battery icon, a signal strength icon, a date and time (12.05.2025), and a clock icon.

PUT /admin/users/{id} – Update non-existent user

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library / updateUser' collection. The 'updateUser' endpoint is selected and highlighted in grey. In the main workspace, a 'PUT' request is configured with the URL `http://localhost:8080/library/api/users/0b74cfa8-8aac-4277-8acb-63977ef4f489`. The 'Authorization' tab is selected, showing 'Bearer Token' as the type and a redacted token value. The 'Headers' tab shows '(9)' entries. The 'Body' tab is selected, showing an empty JSON object: `{ } JSON`. The response panel shows a 404 Not Found error with the message: `404 Not Found 94 ms 593 B`. The response body is a JSON object with an 'errors' array containing one error: `"field": "internal", "message": "User not found with id: 0b74cfa8-8aac-4277-8acb-63977ef4f489"`. The status code is listed as 404.

```
1 {  
2   "errors": [  
3     {  
4       "field": "internal",  
5       "message": "User not found with id: 0b74cfa8-8aac-4277-8acb-63977ef4f489"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:21:34.129138575",  
9   "status": 404  
10 }
```

DELETE /admin/users/{id} – Delete a user that does not exist

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' section. In the center, a specific DELETE request is selected:

- Method:** DELETE
- URL:** http://localhost:8080/library/api/users/152e474e-c906-48bc-a69a-bc518f932992
- Authorization:** Bearer Token (with token value UxUOWF31Pmr8N3MkVfdS1lmVRqTg)
- Headers:** (7 items listed)
- Body:** (Empty JSON object: {})

The response details show a 404 Not Found status with the following JSON body:

```
1 {  
2   "errors": [  
3     {  
4       "field": "internal",  
5       "message": "User not found with id: 152e474e-c906-48bc-a69a-bc518f932992"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:22:49.190327264",  
9   "status": 404  
10 }
```

At the bottom, the taskbar shows several open tabs and system icons.

GET /books/{id} – Book not found

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections, environments, flows, and history. The main workspace shows a collection named "Library / getBookById". A specific GET request for "getBookById" is selected. The request URL is set to `http://localhost:8080/library/api/books/e9185c46-d706-4c7a-9452-8093ce4702c1`. The "Authorization" tab is active, showing "Bearer Token" selected in the dropdown. The "Body" tab shows a JSON response with an error message:

```
1 {  
2   "errors": [  
3     {  
4       "field": "internal",  
5       "message": "Book not found with id: e9185c46-d706-4c7a-9452-8093ce4702c1"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:26:19.572429125",  
9   "status": 404  
10 }
```

The status bar at the bottom indicates the response was a 404 Not Found with a timestamp of 2025-05-12T15:26:19.572429125 and a size of 593 B.

PUT /books/{id} – Book not found during update

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints. In the center, a specific request is being viewed: a PUT operation named "updateBook". The URL is set to `http://localhost:8080/library/api/books/91c0e1ce-9207-4bc4-89c4-9e6cc077ac8e`. The "Authorization" tab is selected, showing "Bearer Token" as the type and a redacted token value. The "Body" tab is selected, showing an empty JSON object. The response pane at the bottom displays a 404 Not Found error with the following JSON payload:

```
1 {  
2   "errors": [  
3     {  
4       "field": "internal",  
5       "message": "Book not found with id: 91c0e1ce-9207-4bc4-89c4-9e6cc077ac8e"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:27:39.163512266",  
9   "status": 404  
10 }
```

The status bar at the bottom shows the Windows taskbar with various icons and the system tray indicating the date and time.

DELETE /books/{id} – Book not found during delete

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' collection. The 'deleteBook' endpoint is selected and highlighted with a grey background.

The main workspace displays a DELETE request to `http://localhost:8080/library/api/books/25447f36-6259-469d-95ab-cb1edd5c2d72`. The 'Authorization' tab is selected, showing 'Bearer Token' as the type and a redacted token value in the 'Token' field.

The response body shows a JSON object with an error message:

```
1 {  
2   "errors": [  
3     {  
4       "field": "internal",  
5       "message": "Book not found with id: 25447f36-6259-469d-95ab-cb1edd5c2d72"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:28:47.667615192",  
9   "status": 404  
10 }
```

The status bar at the bottom indicates the response was a 404 Not Found error with a timestamp of 2025-05-12T15:28:47.667615192 and a size of 593 B.

POST /books – Duplicate ISBN conflict

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various API endpoints under 'Library'. The 'createBook' endpoint is selected. In the main workspace, a POST request is configured to 'http://localhost:8080/library/api/books'. The 'Authorization' tab is selected, showing 'Bearer Token' as the type with a placeholder 'Token'. The 'Body' tab shows a JSON response with an error message:

```
1 {  
2   "errors": [  
3     {  
4       "field": "internal",  
5       "message": "A book with this ISBN already exists: 9780743273565"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:31:20.802637228",  
9   "status": 409  
10 }
```

The status bar at the bottom indicates the response was a 409 Conflict with a timestamp of 2025-05-12T15:31:20.802637228 and a size of 583 B.

POST /borrow-records – Book not found

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various API endpoints. In the center, a 'POST' request is being made to `http://localhost:8080/library/api/borrow-records`. The request body is a JSON object:

```
1 {
2   "userId": "152e474e-c906-48bc-a69a-bc518f932994",
3   "bookId": "3232f429-bce4-4ade-bdd3-b3576a397070",
4   "borrowDate": "2025-05-09",
5   "dueDate": "2025-05-16"
6 }
```

The response status is 404 Not Found, with a timestamp of `2025-05-12T15:33:56.409742431` and a status code of 404. The response body contains an error message:

```
1 {
2   "errors": [
3     {
4       "field": "internal",
5       "message": "Book not found with id: 3232f429-bce4-4ade-bdd3-b3576a397070"
6     }
7   ],
8   "timestamp": "2025-05-12T15:33:56.409742431",
9   "status": 404
10 }
```

At the bottom, the taskbar shows several open windows and system icons.

POST /borrow-records – Book not available

The screenshot shows the Postman application interface. On the left, the 'Library' collection is selected, displaying various API endpoints. The 'borrowBook' endpoint is highlighted. In the main workspace, a POST request is configured to 'http://localhost:8080/library/api/borrow-records'. The 'Body' tab is active, showing the following JSON payload:

```
1 {
2   "userId": "152e474e-c906-48bc-a69a-bc518f932994",
3   "bookId": "c85e41bf-a2a6-41e6-a07b-d855fb737d12",
4   "borrowDate": "2025-05-12",
5   "dueDate": "2025-05-16"
6 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T15:35:18.938Z and a status of 400. The response body is:

```
1 {
2   "errors": [
3     {
4       "field": "internal",
5       "message": "Book is currently not available for borrowing"
6     }
7   ],
8   "timestamp": "2025-05-12T15:35:18.938Z",
9   "status": 400
10 }
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system clock indicating 18:35 on 12.05.2025.

POST /borrow-records – User not found

The screenshot shows the Postman application interface. On the left, the 'Project' sidebar lists various API endpoints under 'Library'. In the center, a 'POST' request is selected for the endpoint `/borrow-records`. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   "userId": "25e35e62-e3b8-4d06-a445-5822fe828ca1",  
3   "bookId": "7de6996c-59b4-4b5d-b18b-b7e6ee3410ad",  
4   "borrowDate": "2025-05-12",  
5   "dueDate": "2025-05-16"  
6 }
```

The response status is 404 Not Found, with a timestamp of 2025-05-12T15:37:24.894695696 and a status code of 404. The response body is a JSON object with an 'errors' field containing one error message: "User not found with id: 25e35e62-e3b8-4d06-a445-5822fe828ca1".

POST /borrow-records – Borrow limit exceeded

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections, environments, flows, and history. The main workspace shows a POST request to `http://localhost:8080/library/api/borrow-records`. The request body is set to raw JSON:

```
1 {
2   "userId": "25e55e62-e3b8-4d06-a445-5822fe828ca1",
3   "bookId": "1b67f350-a45f-4316-995c-4afc69a8c138",
4   "borrowDate": "2025-05-12",
5   "dueDate": "2025-05-16"
6 }
```

The response status is 400 Bad Request, with a timestamp of `2025-05-12T15:39:55.095Z` and status 400. The response body is:

```
1 {
2   "errors": [
3     {
4       "field": "internal",
5       "message": "You have reached the maximum limit of 5 borrowed books."
6     }
7   ],
8   "timestamp": "2025-05-12T15:39:55.095Z",
9   "status": 400
10 }
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system clock indicating 18:39 on 12.05.2025.

POST /borrow-records – User has overdue books

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main area displays a POST request to `http://localhost:8080/library/api/borrow-records`. The request body is set to raw JSON:

```
1 {
2   "userId": "9add54e9-272d-4d9f-8e75-e8e41f455adf",
3   "bookId": "1b67f350-a45f-4316-995c-4afc69a8c138",
4   "borrowDate": "2025-05-12",
5   "dueDate": "2025-05-16"
6 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T15:40:57.742491413 and a status of 400. The response body contains an error message:

```
1 {
2   "errors": [
3     {
4       "field": "internal",
5       "message": "You have overdue books. Please return them before borrowing more."
6     }
7   ],
8   "timestamp": "2025-05-12T15:40:57.742491413",
9   "status": 400
10 }
```

At the bottom, the taskbar shows various open windows and system notifications.

PUT /borrow-records/return/{id} – Borrow record not found

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' collection. The 'returnBook' endpoint is selected and highlighted with a grey background. In the main workspace, a 'PUT' request is configured with the URL `http://localhost:8080/library/api/borrow-records/return/835169bd-adf6-4b44-aed6-4694afc4437a`. The 'Authorization' tab is active, set to 'Bearer Token', and a token value is displayed in a text input field. The 'Headers' tab shows 9 items. The 'Body' tab is selected, showing a JSON response with an error message. The response body is:

```
{ } JSON ▾
```

```
1 {  
2   "errors": [  
3     {  
4       "field": "internal",  
5       "message": "Borrow record not found with id: 835169bd-adf6-4b44-aed6-4694afc4437a"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:42:12.453977548",  
9   "status": 404  
10 }
```

The status bar at the bottom indicates the response was a 404 Not Found error with a timestamp of 2025-05-12T15:42:12.453977548 and a size of 602 B.

PUT /borrow-records/return/{id} – Book already returned

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' collection. The main area displays a 'PUT' request for the endpoint `/library/api/borrow-records/return/a43c14fa-9f2d-4ea9-97f8-e98deee7c3f3`. The 'Authorization' tab is selected, showing 'Bearer Token' as the type and a redacted token value. The 'Body' tab shows a JSON payload with an error message. The response section indicates a **400 Bad Request** status with a timestamp of `2025-05-12T15:43:53.630038599Z`.

PUT /library/api/borrow-records/return/a43c14fa-9f2d-4ea9-97f8-e98deee7c3f3

Authorization: Bearer Token

```
{ "errors": [ { "field": "internal", "message": "This book has already been returned" } ], "timestamp": "2025-05-12T15:43:53.630038599Z", "status": 400 }
```

400 Bad Request

GET /borrow-records/user/{userId} – User not found

The screenshot shows the Postman application interface. On the left, a sidebar lists various API endpoints under a 'Project' collection. The main area displays a 'getBorrowRecordsByUserId' endpoint. The 'Authorization' tab is selected, showing 'Bearer Token' as the type and a redacted token value. The 'Body' tab shows a JSON response with an error message. The status bar at the bottom indicates the request took 291 ms and returned 593 B of data.

HTTP Library / getBorrowRecordsByUserId

GET http://localhost:8080/library/api/borrow-records/user/152e474e-c906-48bc-a69a-bc518f932994

Params Authorization Headers (7) Body Scripts Settings

Auth Type: Bearer Token

Token: (Redacted)

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Body Cookies Headers (14) Test Results (0/1)

404 Not Found 291 ms 593 B Save Response

{ } JSON ▾

```
1 {  
2   "errors": [  
3     {  
4       "field": "internal",  
5       "message": "User not found with id: 152e474e-c906-48bc-a69a-bc518f932994"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:44:45.950198045",  
9   "status": 404  
10 }
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

Getir Java S... Getir Bitirm... exceptions library-man... Containers -... docker kom... Library Man... getBorrowR... 22°C 18:44 12.05.2025

POST /auth/register – Email already registered

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library / register' collection. The 'register' endpoint is selected. In the main workspace, a POST request is configured to 'http://localhost:8080/library/api/auth/register'. The 'Body' tab contains a JSON payload:

```
1 {  
2   "name": "admin",  
3   "email": "admin@gmail.com",  
4   "password": "123456",  
5   "phoneNumber": "5354214577",  
6   "role": "PATRON"  
7 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T15:46:03.91976084 and a status of 400. The response body is a JSON object with an 'errors' field containing one error message: "Email is already registered".

At the bottom, the taskbar shows several open windows, including 'register - Pr...', and the system tray indicates the date and time as 12.05.2025 18:46.

POST /auth/login – User exists but wrong password

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints. In the center, a specific POST request to `http://localhost:8080/library/api/auth/login` is selected. The request body is set to raw JSON:

```
1 {
2   "email": "admin@gmail.com",
3   "password": "12345j6"
4 }
```

The response status is **401 Unauthorized**, with a timestamp of `2025-05-12T15:46:53.944443203` and a status code of `401`. The response body is a JSON object containing an error message:

```
1 {
2   "errors": [
3     {
4       "field": "credentials",
5       "message": "Invalid email or password"
6     }
7   ],
8   "timestamp": "2025-05-12T15:46:53.944443203",
9   "status": 401
10 }
```

At the bottom, the taskbar shows several open tabs and system notifications.

POST /auth/login –

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Project' tab. In the center, a specific POST request to '/auth/login' is selected. The request details show the URL as `http://localhost:8080/library/api/auth/login`. The 'Body' tab is selected, showing a raw JSON payload:

```
1 {
2   "email": "admin2@gmail.com",
3   "password": "123456"
4 }
```

The response section shows a status of **401 Unauthorized**, a timestamp of `2025-05-12T15:51:01.6679529`, and a status code of `401`. The response body is a JSON object with an 'errors' field containing a single error message: `"field": "credentials", "message": "Invalid email or password"`.

At the bottom, the taskbar shows several open tabs and system notifications.

Validations

POST /auth/login – Missing email

The screenshot shows the Postman interface with a failing API call to `http://localhost:8080/library/api/auth/register`.

Request Details:

- Method: POST
- URL: `http://localhost:8080/library/api/auth/register`
- Headers: (8)
- Body (raw JSON):

```
1 {
2     "name": "admin",
3     "email": "",
4     "password": "123456",
5     "phoneNumber": "5354214577",
6     "role": "PATRON"
7 }
```

Response Status: 400 Bad Request

Response Body (JSON):

```
1 {
2     "errors": [
3         {
4             "field": "email",
5             "message": "Email is required"
6         }
7     ],
8     "timestamp": "2025-05-12T15:52:48.788174654",
9     "status": 400
10 }
```

Postman Footer:

- Postbot
- Runner
- Start Proxy
- Cookies
- Vault
- Trash

18:53 22°C 12.05.2025

POST /auth/login – Invalid email format

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' section, including 'register'. The main workspace displays a POST request to 'http://localhost:8080/library/api/auth/register'. The request body is set to raw JSON:

```
1 {  
2   "name": "admin",  
3   "email": "admin-gmail.com",  
4   "password": "123456",  
5   "phoneNumber": "5354214577",  
6   "role": "PATRON"  
7 }
```

The response status is 400 Bad Request, with a timestamp of "2025-05-12T15:54:19.649840182" and a status of 400. The response body is:

```
1 {  
2   "errors": [  
3     {  
4       "field": "email",  
5       "message": "Invalid email format"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:54:19.649840182",  
9   "status": 400  
10 }
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray.

POST /auth/login – Missing password

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Project' tab. In the center, a specific POST request to '/register' is being tested. The request URL is `http://localhost:8080/library/api/auth/register`. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "name": "admin",
3   "email": "admin@gmail.com",
4   "password": "",
5   "phoneNumber": "5354214577",
6   "role": "PATRON"
7 }
```

The response status is 400 Bad Request. The response body is:

```
1 {
2   "errors": [
3     {
4       "field": "password",
5       "message": "Password is required"
6     },
7     {
8       "field": "password",
9       "message": "Password must be at least 6 characters"
10    }
11  ],
12  "timestamp": "2025-05-12T15:55:13.758533051",
13  "status": 400
14 }
```

At the bottom, the taskbar shows several open windows and system status.

POST /auth/login – Password too short

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Project' tab, including 'GET getOwnUserDetails', 'GET getAllBooks', and 'POST register'. The main workspace displays a POST request to 'http://localhost:8080/library/api/auth/register'. The request body is set to raw JSON:

```
1 {  
2   "name": "admin",  
3   "email": "admin@gmail.com",  
4   "password": "123",  
5   "phoneNumber": "5354214577",  
6   "role": "PATRON"  
7 }
```

The response status is '400 Bad Request' with a timestamp of '2025-05-12T15:56:20.852736697' and a status code of '400'. The response body is:

```
1 {  
2   "errors": [  
3     {  
4       "field": "password",  
5       "message": "Password must be at least 6 characters"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T15:56:20.852736697",  
9   "status": 400  
10 }
```

The bottom navigation bar includes icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a help icon. The system tray shows the date and time as '12.05.2025 18:56'.

POST /users – Missing name

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Project' tab. In the center, a specific POST request to 'register' is selected. The request URL is `http://localhost:8080/library/api/auth/register`. The 'Body' tab is active, showing a JSON payload:

```
1 {
2   "name": "",
3   "email": "admin@gmail.com",
4   "password": "123",
5   "phoneNumber": "5354214577",
6   "role": "PATRON"
7 }
```

The response status is 400 Bad Request. The response body is:

```
1 {
2   "errors": [
3     {
4       "field": "name",
5       "message": "Name is required"
6     },
7     {
8       "field": "password",
9       "message": "Password must be at least 6 characters"
10    }
11  ],
12  "timestamp": "2025-05-12T15:58:30.290424144",
13  "status": 400
14 }
```

At the bottom, the taskbar shows several open tabs and system icons.

POST /users – Invalid phone number format

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main area displays a POST request to the '/register' endpoint of a library API. The request body is set to raw JSON:

```
1 {
2   "name": "admin",
3   "email": "admin@gmail.com",
4   "password": "123456",
5   "phoneNumber": "",
6   "role": "PATRON"
7 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T15:59:42.432467582 and a status of 400. The response body contains error details:

```
1 {
2   "errors": [
3     {
4       "field": "phoneNumber",
5       "message": "Invalid phone number. Please enter a 10-digit number e.g. 5541234567"
6     },
7     {
8       "field": "phoneNumber",
9       "message": "Phone number is required"
10    }
11  ],
12  "timestamp": "2025-05-12T15:59:42.432467582",
13  "status": 400
14 }
```

At the bottom, the taskbar shows various open windows and system status.

POST /users – Missing role

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. In the center, a specific collection named "HTTP Library / register" is selected. A POST request is being made to the URL `http://localhost:8080/library/api/auth/register`. The request body is set to raw JSON:

```
1 {  
2   "name": "admin",  
3   "email": "admin@gmail.com",  
4   "password": "123456",  
5   "phoneNumber": "5354214577",  
6   "role": ""  
7 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T16:00:36.740785255 and a status of 400. The response body is:

```
1 {  
2   "errors": [  
3     {  
4       "field": "role",  
5       "message": "Invalid role value. Must be 'LIBRARIAN' or 'PATRON'."  
6     }  
7   ],  
8   "timestamp": "2025-05-12T16:00:36.740785255",  
9   "status": 400  
10 }
```

At the bottom, the Postman footer includes links for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a help icon. The system tray shows icons for Getir Java S., Getir Bitirm..., exceptions, library-man..., Containers ..., docker kom..., Library Man..., and register - Pr... along with the date 12.05.2025 and time 19:00.

POST /books – Missing title

The screenshot shows the Postman application interface. On the left, the sidebar displays various collections, environments, flows, and history. The main workspace shows a collection named "HTTP Library / createBook". A specific POST request is selected, which is intended to create a book entry. The request URL is `http://localhost:8080/library/api/books`. The request body is set to raw JSON and contains the following data:

```
1 {
2   "title": "",
3   "author": "Andrew Sharman2",
4   "isbn": "9780743273565",
5   "publicationDate": "1980-07-01",
6   "genre": "Personal Development"
7 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T16:02:19.750079997 and a status code of 400. The response body is a JSON object containing an error message:

```
1 {
2   "errors": [
3     {
4       "field": "title",
5       "message": "Title is required"
6     }
7   ],
8   "timestamp": "2025-05-12T16:02:19.750079997",
9   "status": 400
10 }
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system clock indicating 19:02 on 12.05.2025.

POST /books – Missing author

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main workspace displays a collection named "HTTP Library / createBook". A specific POST request is selected, targeting the URL `http://localhost:8080/library/api/books`. The request body is set to raw JSON, containing the following data:

```
1 {  
2   "title": "The Feeling Good Book2",  
3   "author": "",  
4   "isbn": "9780743273565",  
5   "publicationDate": "1980-07-01",  
6   "genre": "Personal Development"  
7 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T16:02:45.621342006 and a status code of 400. The error message in the response body is:

```
1 {  
2   "errors": [  
3     {  
4       "field": "author",  
5       "message": "Author is required"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T16:02:45.621342006",  
9   "status": 400  
10 }
```

At the bottom, the task bar includes icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a search bar. The system tray shows the date (12.05.2025), time (19:02), battery level (22°C), and network status.

POST /books – Invalid ISBN format

The screenshot shows the Postman application interface. The left sidebar contains a 'Project' section with various API endpoints listed under 'Collections'. One endpoint, 'POST createBook', is selected and highlighted in grey. The main workspace displays a POST request to 'http://localhost:8080/library/api/books'. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   "title": "The Feeling Good Book2",  
3   "author": "Andzrew Shazman2",  
4   "isbn": "12345",  
5   "publicationDate": "1980-07-01",  
6   "genre": "Personal Development"  
7 }
```

The response status is '400 Bad Request' with a timestamp of '2025-05-12T16:03:07.681903641' and a status code of '400'. The response body is a JSON object with an 'errors' field containing one error message: 'ISBN must be exactly 13 digits'.

POST /books – Missing publication date

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main area displays a collection named "HTTP Library / createBook". A specific POST request to `http://localhost:8080/library/api/books` is selected. The request body is set to raw JSON:

```
1 {  
2   "title": "The Feeling Good Book2",  
3   "author": "Andrew Sharmaz",  
4   "isbn": "9780743273565",  
5   "publicationDate": "",  
6   "genre": "Personal Development"  
7 }
```

The response status is 400 Bad Request, with the following JSON error message:

```
1 {  
2   "errors": [  
3     {  
4       "field": "publicationDate",  
5       "message": "Publication date is required"  
6     }  
7   ],  
8   "timestamp": "2025-05-12T16:03:34.873724351",  
9   "status": 400  
10 }
```

At the bottom, the taskbar shows various icons and the system clock.

POST /books – Missing genre

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main area displays a collection named "HTTP Library / createBook". A specific POST request is selected, which is intended to create a book entry. The request URL is `http://localhost:8080/library/api/books`. The request body is set to raw JSON and contains the following data:

```
1 {
2   "title": "The Feeling Good Book2",
3   "author": "Andrew Sharman2",
4   "isbn": "9780743273565",
5   "publicationDate": "1980-07-01",
6   "genre": ""
7 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T16:03:50.049580724 and a status code of 400. The response body shows an error message indicating that the "genre" field is required:

```
1 {
2   "errors": [
3     {
4       "field": "genre",
5       "message": "Genre is required"
6     }
7   ],
8   "timestamp": "2025-05-12T16:03:50.049580724",
9   "status": 400
10 }
```

At the bottom, the taskbar shows various open windows and the current date and time (12.05.2025, 19:03).

POST /borrow-records – Missing userId

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main area displays a POST request to `http://localhost:8080/library/api/borrow-records`. The request body is set to raw JSON:

```
1 {
2   "userId": "",
3   "bookId": "1b67f350-a45f-4316-995c-4afc69a8c138",
4   "borrowDate": "2025-05-12",
5   "dueDate": "2025-05-16"
6 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T16:21:39.163397383 and a status code of 400. The error message is: "User ID is required".

At the bottom, the taskbar shows several open tabs and system icons.

POST /borrow-records – Missing bookId

The screenshot shows the Postman application interface. On the left, the sidebar lists various API endpoints under the 'Library' collection. The 'borrowBook' endpoint is selected. In the main workspace, a POST request is configured to 'http://localhost:8080/library/api/borrow-records'. The 'Body' tab is active, showing a JSON payload:

```
1 {
2   "userId": "9add54e9-272d-4d9f-8e75-e8e41f455adf",
3   "bookId": "",
4   "borrowDate": "2025-05-12",
5   "dueDate": "2025-05-16"
6 }
```

The response status is '400 Bad Request' with a timestamp of '2025-05-12T16:22:14.603134502' and a status code of '400'. The response body contains an error message:

```
1 {
2   "errors": [
3     {
4       "field": "bookId",
5       "message": "Book ID is required"
6     }
7   ],
8   "timestamp": "2025-05-12T16:22:14.603134502",
9   "status": 400
10 }
```

The bottom of the screen shows the Windows taskbar with various icons and system information.

POST /borrow-records – Missing borrowDate

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main area displays a POST request to `http://localhost:8080/library/api/borrow-records`. The request body is set to raw JSON:

```
1 {
2   "userId": "9add54e9-272d-4d9f-8e75-e8e41f455adf",
3   "bookId": "1b67f350-a45f-4316-995c-4afc69a8c138",
4   "borrowDate": "",
5   "dueDate": "2025-05-16"
6 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T16:22:28.753758576 and status 400. The error message in the response body is:

```
1 {
2   "errors": [
3     {
4       "field": "borrowDate",
5       "message": "Borrow date is required"
6     }
7   ],
8   "timestamp": "2025-05-12T16:22:28.753758576",
9   "status": 400
10 }
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system clock indicating 19:22 on 12.05.2025.

POST /borrow-records – Missing dueDate

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main area displays a POST request to `http://localhost:8080/library/api/borrow-records`. The request body is set to raw JSON:

```
1 {
2   "userId": "9add54e9-272d-4d9f-8e75-e8e41f455adf",
3   "bookId": "1b67f350-a45f-4316-995c-4afc69a8c138",
4   "borrowDate": "2025-05-12",
5   "dueDate": ""
6 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T16:22:45.610473034 and status 400. The error message in the response body is:

```
1 {
2   "errors": [
3     {
4       "field": "dueDate",
5       "message": "Due date is required"
6     }
7   ],
8   "timestamp": "2025-05-12T16:22:45.610473034",
9   "status": 400
10 }
```

At the bottom, the taskbar shows various open tabs and system icons.

POST /borrow-records – Due date not in future

The screenshot shows the Postman application interface. On the left, the sidebar lists various collections, environments, flows, and history. The main workspace displays a POST request to `http://localhost:8080/library/api/borrow-records`. The request body is set to raw JSON:

```
1 {
2   "userId": "9add54e9-272d-4d9f-8e75-e8e41f455adf",
3   "bookId": "1b67f350-a45f-4316-995c-4afc69a8c138",
4   "borrowDate": "2025-05-12",
5   "dueDate": "2025-05-01"
6 }
```

The response status is 400 Bad Request, with a timestamp of 2025-05-12T16:23:05.075985856 and a status code of 400. The response body contains an error message:

```
1 {
2   "errors": [
3     {
4       "field": "dueDate",
5       "message": "Due date must be in the future"
6     }
7   ],
8   "timestamp": "2025-05-12T16:23:05.075985856",
9   "status": 400
10 }
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray.