

BookWiseUD: A Scalable Library Management System

Presented by:

Wilder Steven Hernandez Manosalva

Jhon Javier Castañeda Alvarado



BookWiseUD



Addressing Library Management Challenges

Manual Systems

Inefficiencies and data inconsistencies from paper-based or fragmented digital records.

Lack of Integration

No real-time inventory, automated loans, or secure user authentication.

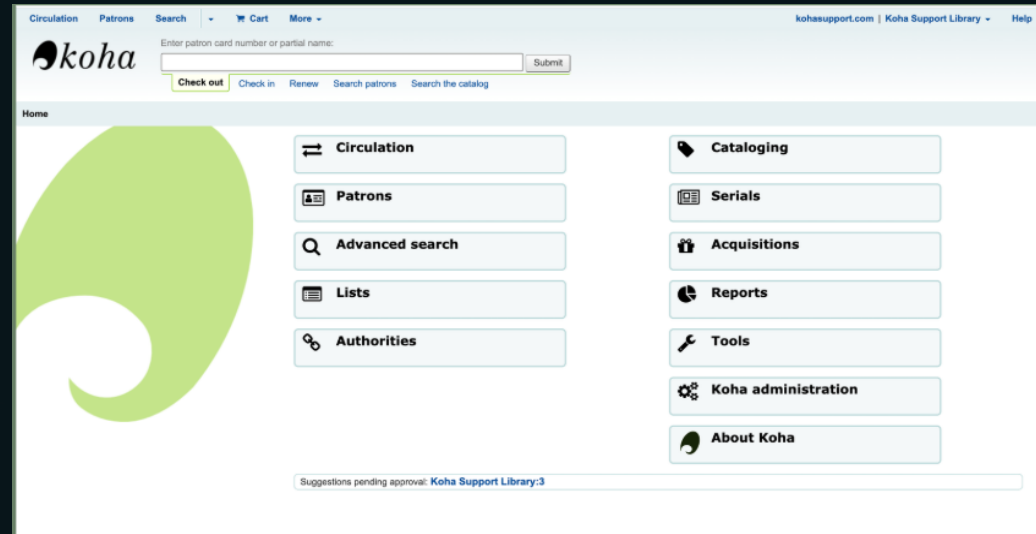
Poor User Experience

Patrons and administrators face bottlenecks and limited access to information.

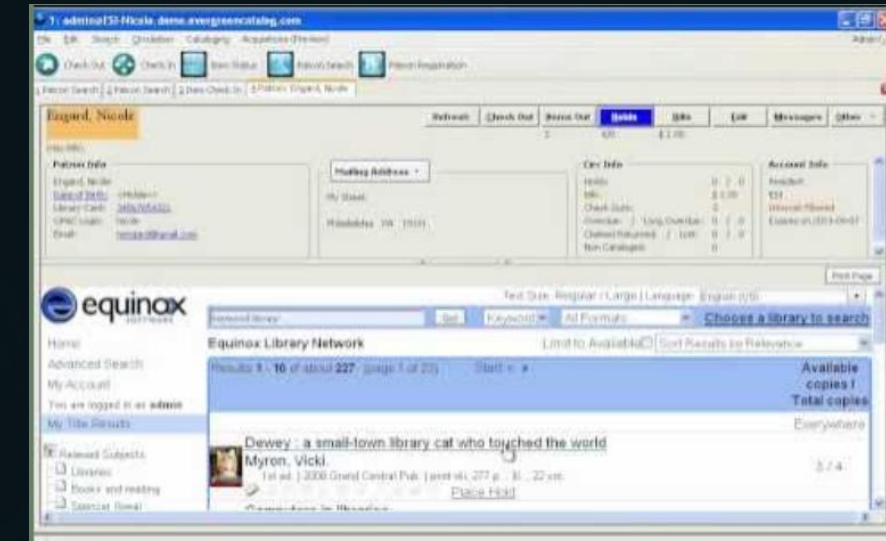


Base or references (inspiration)

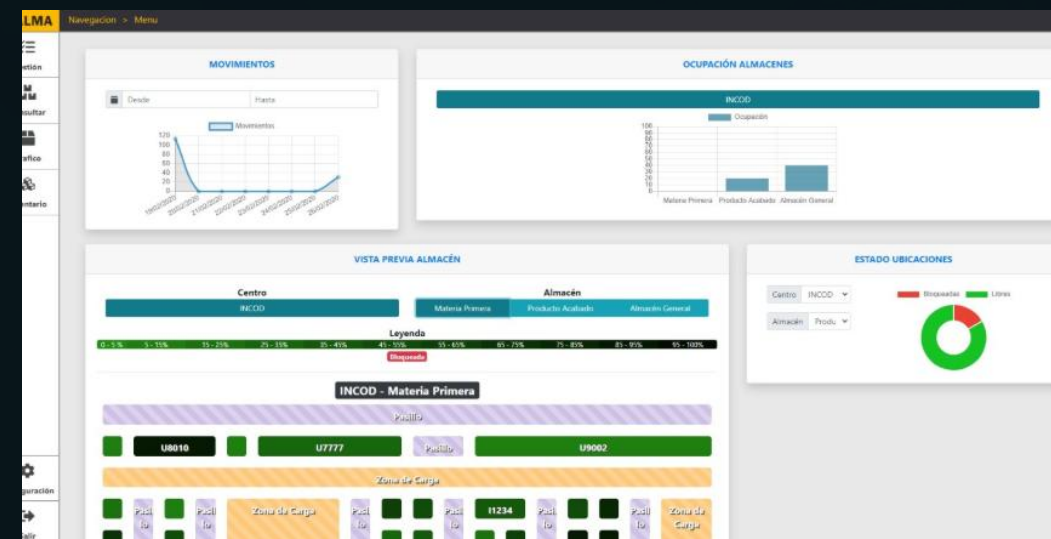
Koha Integrated Library System



Evergreen ILS



Sierra and alma software



Architectural Decisions: Dual-Service Approach

We chose a dual-service architecture over a monolith for specific advantages.

Security Isolation

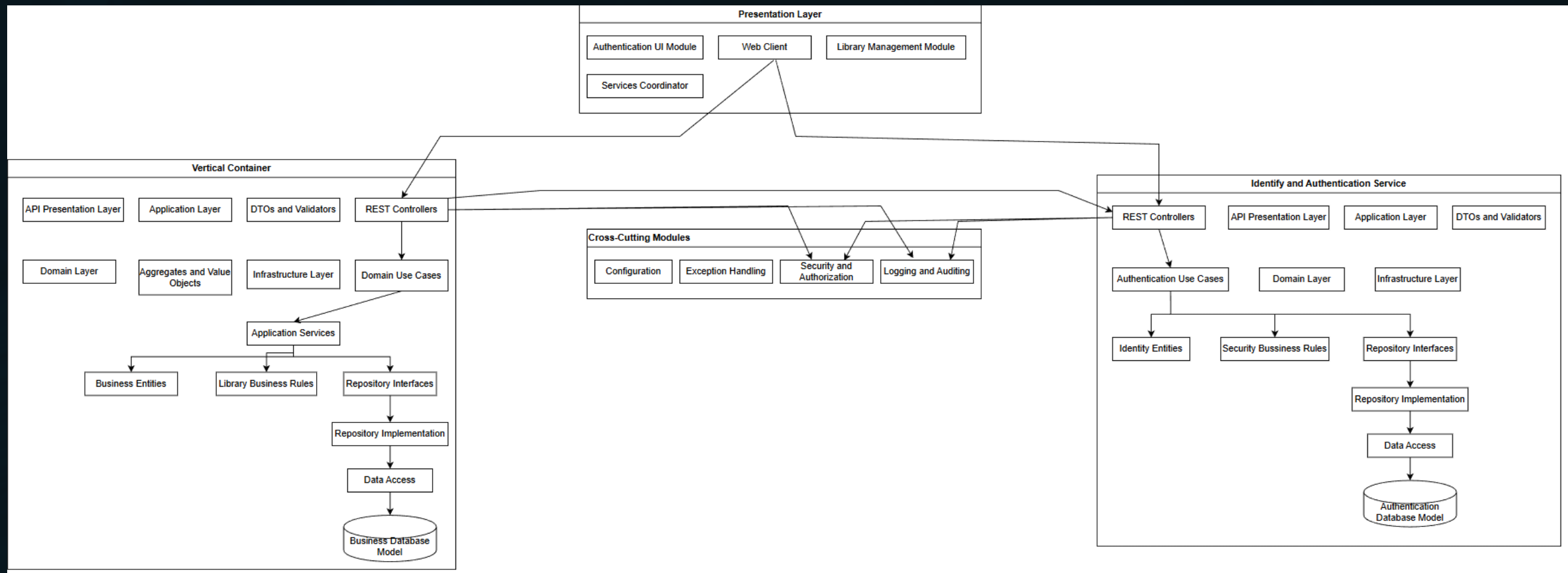
Credential storage separated from domain operations.

Technology Specialization

Spring Security for auth, FastAPI for performance.

Academic Objectives

Demonstrate distributed systems and microservices.



Users Stories

The distribution shows high-priority core features were implemented first. Librarian stories (57% of total) emphasize administrative control, while user stories focus on self-service access.

TABLE I
USER STORIES IMPLEMENTATION SUMMARY

ID	Story Name	Sprint	Priority	Points
Librarian Stories				
US01	Receive Return	1	High	3
US02	Lend Book	1	High	5
US03	Delete User	2	High	2
US04	Update User	2	High	3
US05	Register User	2	High	3
US06	Delete Book	3	High	2
US07	Update Book	3	High	3
US08	Register Book	3	High	3
User Stories				
US09	View Available Books	1	High	3
US10	Request Book Loan	2	High	5
US11	Return Borrowed Book	3	Medium	3
US12	View Loan History	3	Medium	2
US13	Log in to System	1	High	3
US14	Recover Account/Password	1	High	3

User Story Mapping



Book Catalog Access	Book Loan Request	Book Return	Loan History	Authentication	Account Recovery
As a user, I want to view the catalog of available books so that I can know which ones I can borrow.(US88)	As a user, I want to request the loan of an available book so that I can read it for a specific period.(US10)	As a user, I want to register the return of a borrowed book so that my loan history remains updated.(US11)	As a user, I want to view my past loan history so that I can check the books I have borrowed.(US12)	As a user, I want to log in to the system using my credentials so that I can access features available for my role.(US13)	As a user, I want to recover my account when I forget my password so that I can regain access without losing my data.(US14)
Access catalog section.	Select a book from catalog.	View current borrowed books.	Access loan history section.	Access login form.	Enter registered email.
Retrieve available books from database.	Request loan through form or button.	Select a book to return.	Retrieve user loan data from database.	Enter email and password.	System sends verification link or code.
Display relevant info (title, author, category, status).	Validate availability before confirmation.	Confirm return.	Display title, loan date, and return date.	Validate credentials against database.	Open recovery form via secure link.
Implement filters by title, author, or category.	Record loan details (dates, user ID).	Update book status → Available.	Enable sorting by date or title.	Manage session token creation and expiration.	Enter new password.
Handle "no available books" message.	Display success or error message.	Display confirmation or error message.	Show message if no previous loans.	Display success or error message.	Validate and update credentials.
				Implement logout function.	Show confirmation or error message.



Book Return Management	Book Loan Management	User Management	User Registration	Book Inventory Management	Book Registration
As a librarian, I want to receive returned books from users, so that the book inventory is updated and the return process is recorded correctly.(US15)	As a librarian, I want to lend books to users, so that the loan process is tracked and books are issued correctly.(US2)	As a librarian, I want to delete a user from the system, so that inactive or incorrect accounts are removed.(US63)	As a librarian, I want to update user information, so that user records are accurate and up to date.(US84)	As a librarian, I want to register new users, so that they can borrow books and access library services.(US85)	As a librarian, I want to delete books from the system, so that obsolete or damaged books are removed from inventory.(US86)
Verify that the librarian session is active.	Verify active librarian session.	Verify active librarian session.	Verify active librarian session.	Verify active librarian session.	Verify active librarian session.
Retrieve the list of active loans from the database.	Fetch all available books.	Search user by name or ID.	Search user by name or ID.	Search book by title or ID.	Search book by ID or title.
Display the list of active loans in the frontend.	Display available books and user list.	Display user information.	Display user details in editable form.	Validate required fields (name, ID, email, etc.).	Display book information.
Allow the librarian to select a specific loan record.	Allow librarian to select book and user.	Confirm deletion request.	Allow editing of user fields (name, email, phone, etc.).	Save new user in database.	Confirm deletion.
Display loan details (user, book, loan dates).	Record new loan (loan date, return date).	Remove user record from database.	Validate input fields.	Check for duplicates before saving.	Remove book record from inventory database.
Provide a "Confirm Return" button.	Update book status → loaned.	Show confirmation or error message.	Save updated record in database.	Display confirmation or failure message.	Show confirmation or failure message.
On confirmation, update book status → Available.	Validate required information.	Validate user existence before deletion.	Show success or error message.	Validate that the book exists before deletion.	Notify success or failure.
Record return date and user ID in the database.	Notify success or failure.				
Show success or error message.					
Validate all required fields before saving.					

🏠 Para ti

🕒 Recientes

★ Marcados como fav...

🔗 Aplicaciones

📁 Planes PREMIUM

📁 Espacio + ...

Recientes

🇺🇸 project_seminar_engin...

☰ Más espacios

🔗 Confluence

👥 Equipos

... Más

🔍 Buscar

+ Crear

💎 Versión de prueba de Premium

🔔 ? ⚙️ SH

Espacio

🇺🇸 project_seminar_engineering_software

⚙️ ...

🌐 Resumen

📅 Backlog

📊 Tablero

📄 Código

📅 Cronograma

📄 Páginas

📄 Formularios

+

🔍 Buscar en el ba...

👤 SH JC

☰ Filter

Espacio

🇺🇸 project_seminar_engineering_software

⚙️ ...

🌐 Resumen

📅 Backlog

📊 Tablero

📄 Código

📅 Cronograma

📄 Páginas

📄 Formularios

+

🔍 Buscar tablero

👤 SH JC

☰ Filtro

+ Crear

📌 TO DO

+ Crear

📌 IN PROGRESS

📌 IN REVIEW

📌 TESTING

📌 DONE 103

📌 Validate that the book exists before deletion.

📌 ☒ SERUM-71

📌 US01 Receive Return

📌 BOOK RETURN MANAGEMENT

📌 ☒ SERUM-3

📌 Display book information.

📌 ☒ SERUM-67

📌 Access catalog section.

📌 ☒ SERUM-86

📌 Display user details in editable form.

📌 ☒ SERUM-52

📌 Notify success or failure.

📌 ☒ SERUM-38

Completar sprint

BookWiseUD: Our Solution

A modular web-based system for Universidad Distrital Francisco José de Caldas.

React Frontend
Intuitive user interface for
interaction.



Java Spring Boot

Authentication service with
MySQL.

Python FastAPI

Operational service with
PostgreSQL.

Key Functional Requirements

- 1 User Authentication
JWT token issuance and role-based access control.
- 2 Book & User Management
CRUD operations for librarians, user registration.
- 3 Loan Lifecycle
Creation, return, and history tracking.
- 4 Catalog & Search
Browsing and real-time availability for all authenticated users.



Validation & Testing Strategy

A multi-layered approach ensures reliability and performance.

Unit Testing

96% coverage for Python backend, 97% for Java auth service.

Acceptance Testing

100% pass rate for critical user journeys using Cucumber/Behave.

Stress Testing

4.84ms average response time under simulated load (JMeter).

CI/CD Automation

GitHub Actions pipeline for continuous integration and deployment.

```
(venv) PS C:\Users\jjavi\OneDrive\Documentos\Programacion\SpringBoot\Authentication\BookWiseUD\Bc
----- coverage: platform win32, python 3.12.0-final-0 -----
Name                               Stmts   Miss Branch BrPart  Cover   Missing
-----
app\__init__.py                     0       0      0      0    100%
app\auth_utils.py                   58       3     18      1     95%    70, 99-100
app\crud\__init__.py                 0       0      0      0    100%
app\crud\books.py                   35       0      6      0    100%
app\crud\category.py                11       5      0      0     55%    11-18
app\crud\loans.py                   37       0      8      2     96%    44->47, 54->57
app\crud\users.py                   44       0     16      0    100%
app\database.py                     15       4      0      0     73%    17-21
app\main.py                         19       2      4      0     91%    34, 38
app\models\__init__.py               6       0      0      0    100%
app\models\book.py                  14       0      0      0    100%
app\models\category.py               9       0      0      0    100%
app\models\loan.py                  13       0      0      0    100%
app\models\user.py                  12       0      0      0    100%
app\routers\__init__.py              0       0      0      0    100%
app\routers\books.py                30       0     16      0    100%
app\routers\category.py             12       1      4      0     94%    16
app\routers\loans.py                70       1     36      4     95%    106->109, 128, 142->147, 144->147
app\routers\stats.py                16       0      2      0    100%
app\utils.py                        44       2     12      3     91%    33, 46->48, 53
-----
TOTAL                               530     19    138     11     96%
Coverage HTML written to dir htmlcov

===== 128 passed in 2.27s =====
```

28 | private ObjectMapper objectMapper;

PROBLEMS 24 | OUTPUT | DEBUG CONSOLE | TERMINAL | TEST RESULTS | PORTS

ControllerTest)

%TESTE 5,testLogin_WithEmptyUsername(com.example.Authentication.Auth.AuthControllerTest)

%TESTS 6,testLogin_WithInvalidJson(com.example.Authentication.Auth.AuthControllerTest)

%TESTE 6,testLogin_WithInvalidJson(com.example.Authentication.Auth.AuthControllerTest)

%TESTS 7,testRegister_Success(com.example.Authentication.Auth.AuthControllerTest)

%TESTE 7,testRegister_Success(com.example.Authentication.Auth.AuthControllerTest)

%RUNTIME7479

Test Runner for Java

✓ AuthControllerTest \$(symbol-namespace) com.example.Authenticati...

✓ testLogin_Success() \$(symbol-class) AuthControllerTest < \$(symbol-n...

✓ testLogin_WithEmptyUsername() \$(symbol-class) AuthControllerTes...

✓ testLogin_WithInvalidJson() \$(symbol-class) AuthControllerTest < \$(s...

✓ testRegister_ServiceThrowsException() \$(symbol-class) AuthContro...

✓ testRegister_Success() \$(symbol-class) AuthControllerTest < \$(symbol...

> 8 older results

Performance Insights: Stress Testing Results

TEST COVERAGE ANALYSIS BY CRITICAL MODULE

Module	Cov.	Risk	Interpretation
auth_utils.py	95%	Low	Token validation tested
crud/books.py	100%	Low	All CRUD paths covered
crud/loans.py	96%	Low	Loan lifecycle validated
routers/users.py	96%	Low	RBAC well tested
utils.py	91%	Low-Med	Transforms mostly covered
crud/category.py	55%	Med	Basic CRUD only
routers/category.py	94%	Low	Endpoints well tested
routers/loans.py	95%	Low	Endpoints validated

STRESS TESTING RESULTS WITH CONTEXTUAL ANALYSIS

Metric	Value	Interpretation
Total Requests	90	Small concurrent load
Success Rate	57/60 (95%)	Excellent reliability
Failed Auth	3/60 (5%)	Normal variability
Avg Response	4.84ms	Below 50ms target
95th Percentile	8.00ms	Consistent performance
Max Response	52ms	Acceptable worst case
Throughput	22.66 req/s	Supports 100 users

APDEX (Application Performance Index)

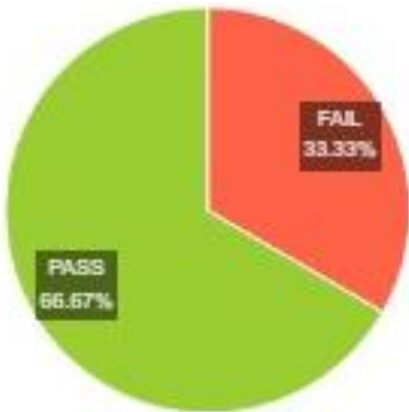
Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.667	500 ms	1 sec 500 ms	Total
0.000	500 ms	1 sec 500 ms	GET /users-1
0.000	500 ms	1 sec 500 ms	POST Java /auth/login
0.000	500 ms	1 sec 500 ms	GET /loans
0.000	500 ms	1 sec 500 ms	POST Java /auth/register
0.000	500 ms	1 sec 500 ms	GET /users
0.000	500 ms	1 sec 500 ms	GET /loans-1
1.000	500 ms	1 sec 500 ms	POST /categories
1.000	500 ms	1 sec 500 ms	GET /books
1.000	500 ms	1 sec 500 ms	GET /categories-1
1.000	500 ms	1 sec 500 ms	GET /users-0
1.000	500 ms	1 sec 500 ms	GET /categories-0
1.000	500 ms	1 sec 500 ms	GET /books-0
1.000	500 ms	1 sec 500 ms	GET /books-1
1.000	500 ms	1 sec 500 ms	POST /books
1.000	500 ms	1 sec 500 ms	GET /categories
1.000	500 ms	1 sec 500 ms	GET /books/available
1.000	500 ms	1 sec 500 ms	GET Python /health
1.000	500 ms	1 sec 500 ms	GET /loans-0

All checks have passed

2 successful checks

✓ CI - E2E (Compose + Behave + JMeter) / e2e (push) Successful in 3m [Details](#)

✓ CI / test-and-build (push) Successful in 1m [Details](#)



Architectural Trade-offs: Retrospective Assessment

Was the dual-service architecture justified for this project?

Realized Advantages

- Security isolation achieved.
- Academic learning objectives met.
- Focused testing simplified.

Realized Challenges

- High operational complexity.
- Data consistency issues.
- Significant development overhead.
- Network latency at scale.

While academically valuable, a monolithic architecture would have been more practical for the project's scale (100 concurrent users).



Conclusion & Future Recommendations

BookWiseUD is functional and robust, but architectural choices highlight key lessons.

1

API Gateway

Implement for simplified frontend and centralized JWT validation.

2

Event-Driven Sync

Use message brokers for user data consistency.

3

Centralized Logging

Essential for production debugging across services.

For similar projects at this scale, a well-designed monolithic backend with DDD alignment is recommended over service separation.



THANK YOU VERY MUCH



BookWiseUD