LIBRAIRIE DORO

BookShop — MySQL Project

Renaud Yoganathan, Max Dambron, Souhaib Sghaier

Object-Oriented Analysis & Design + Advanced Web Programming

Our Goal

Build a simple, normalized MySQL-backed website for discovering books, browsing by genre, and posting user reviews. This version adds a more detailed database structure and an entity-relationship model to illustrate our design choices

ER Diagram

The ER diagram shows how the main entities are connected:

• Book is the central entity.

• Each Book is written by one Author, published by one Publisher, and written in one Language.

• A Book can belong to several Genres through the link table Book_Genre.

• Users can post Reviews about different books.

• Relationships are mainly one-to-many, except Book ↔ Genre, which is many-to-many.
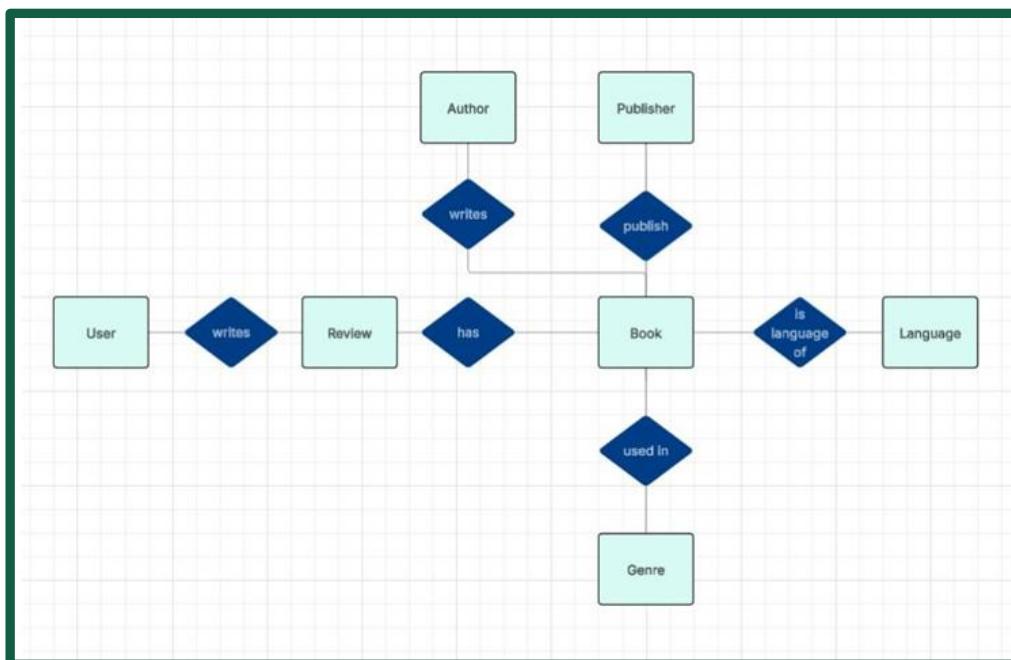
Table Structure Description

      The diagram represents the logical structure of the database used for the book application. The goal was to design a simple, normalized model where each entity has a clear and unique purpose.
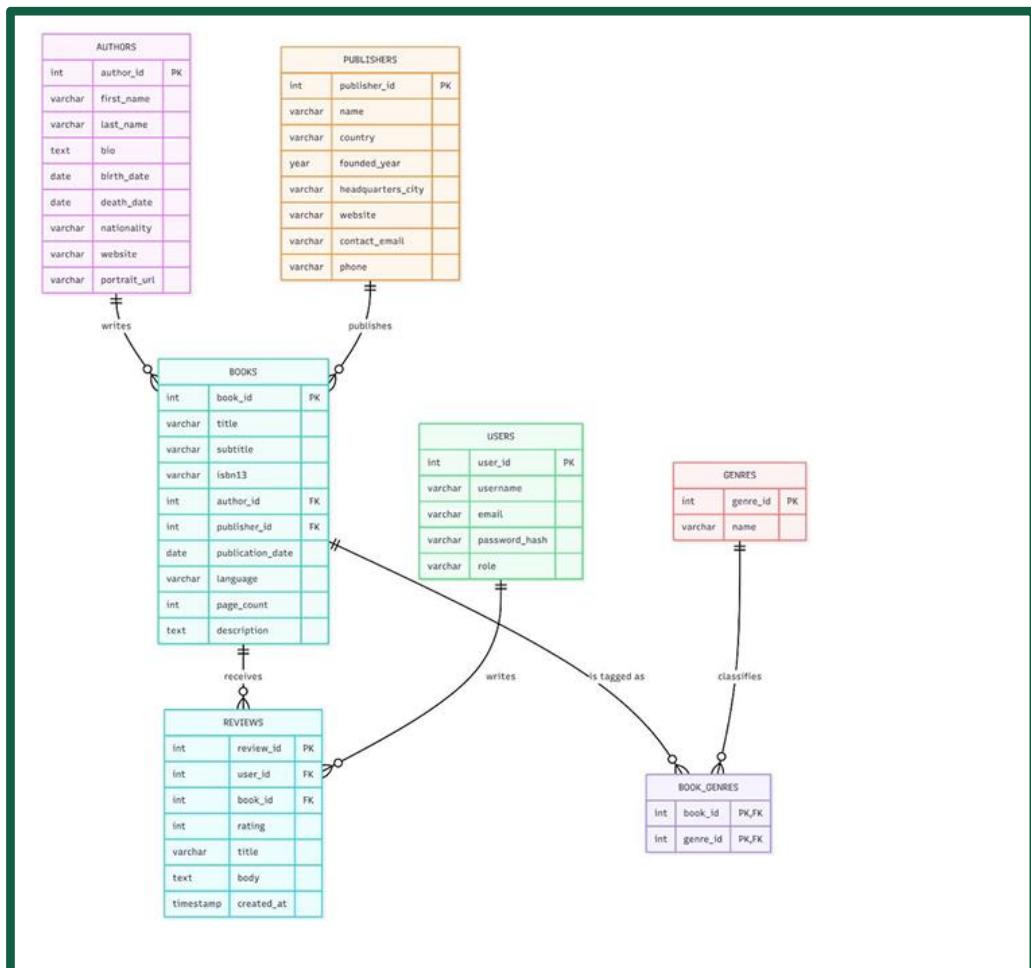
The **Books** table is the central element of the model. Each book is written by one **Author** and published by one **Publisher**, which explains the *one-to-many* relationships between these tables — one author or publisher can be linked to several books.

The **language** field was integrated directly inside the Books table to simplify the design and avoid an unnecessary extra table.

Users are represented in the Users table and can write multiple **Reviews**; each review being related to a single book.
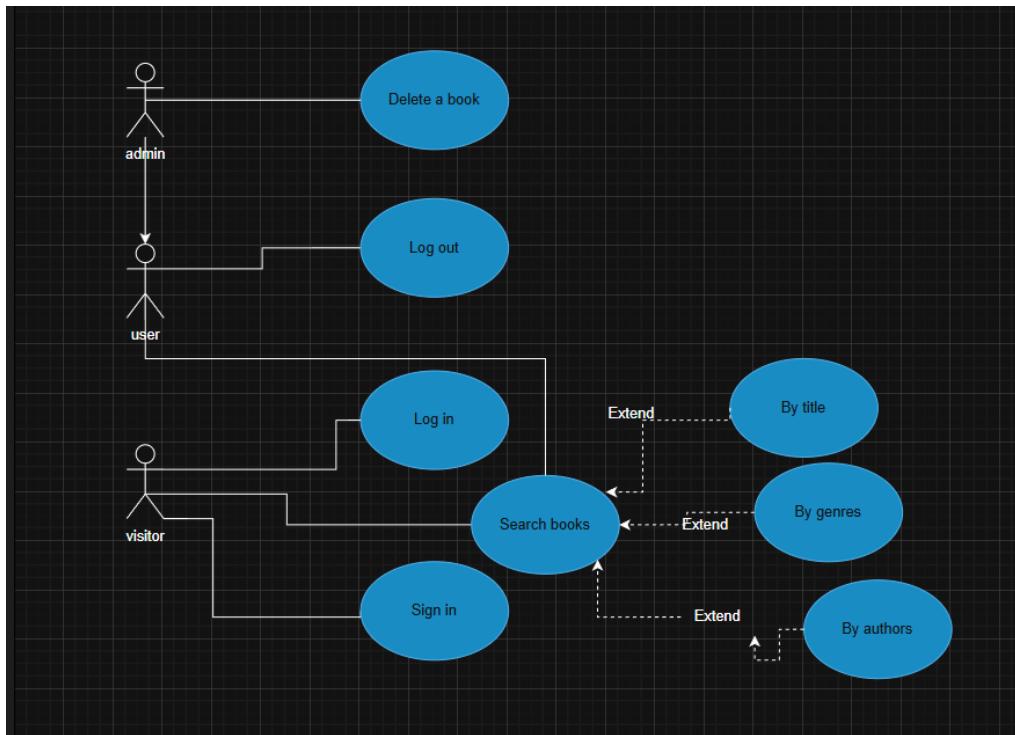
   This creates a one-to-many relation between *Users → Reviews* and *Books → Reviews*.

Books can also belong to several **Genres**, and each genre can contain multiple books. This *many-to-many* relation is managed by the associative table **Book_Genres**, ensuring flexibility when classifying books.
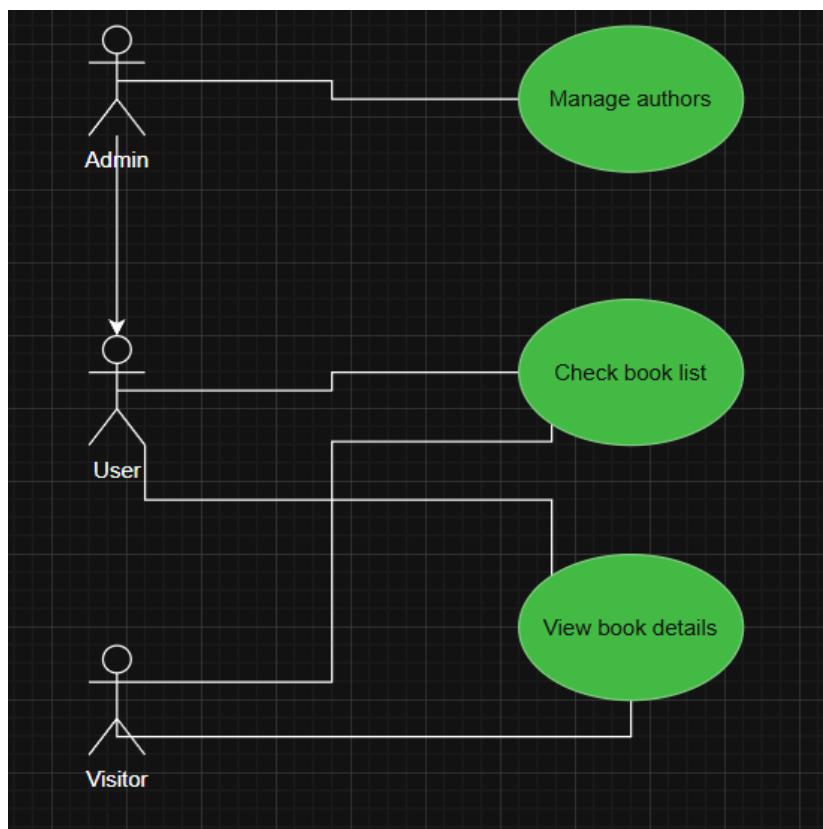
User diagram
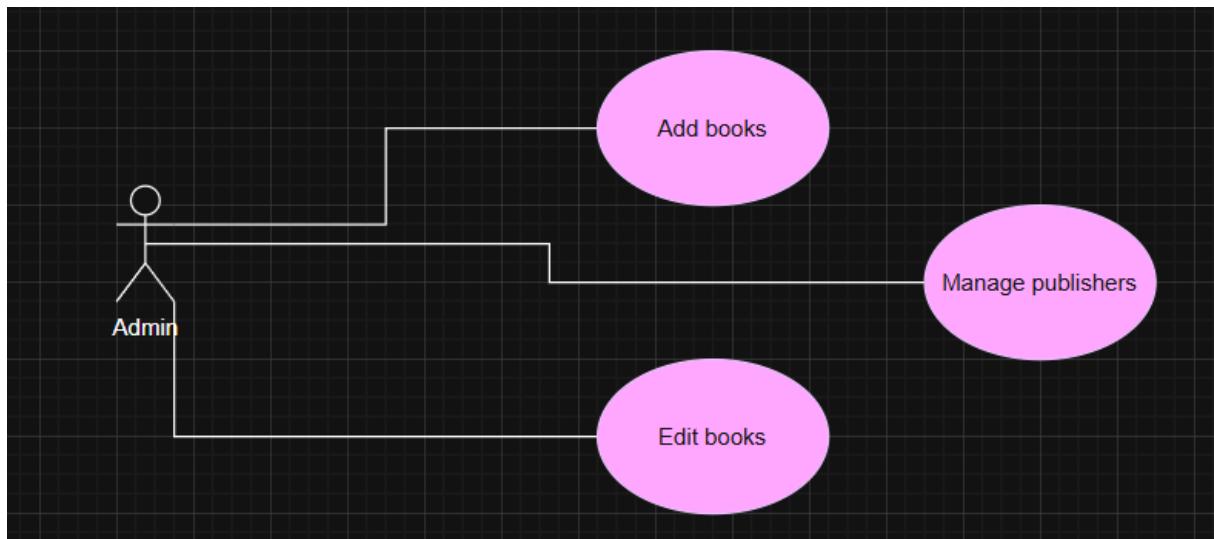
Renaud :

Max :
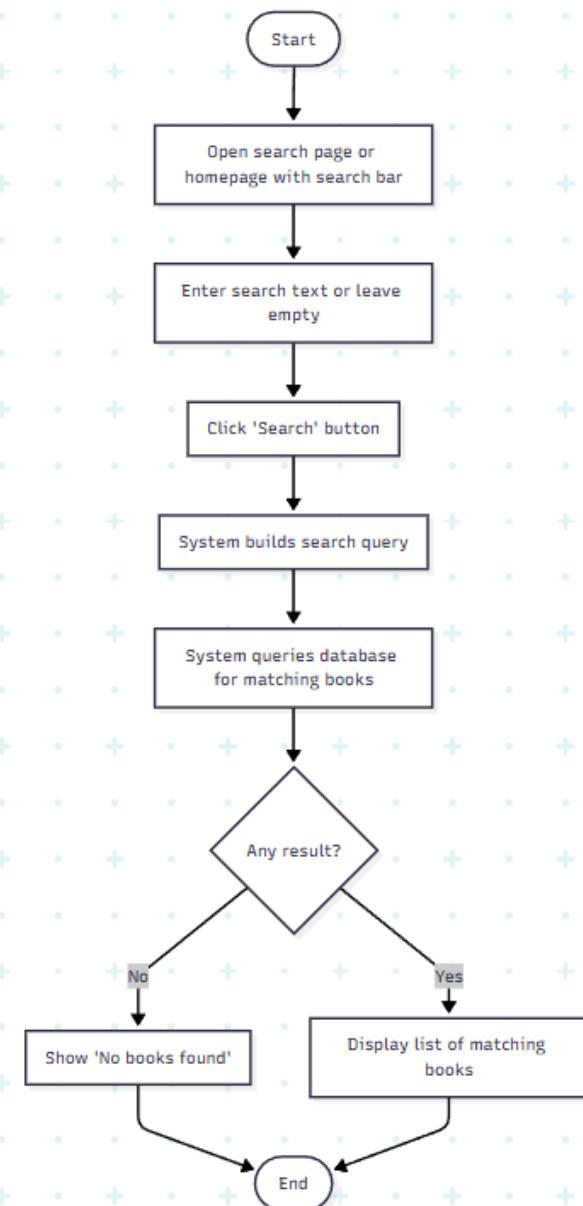
Souhaib :
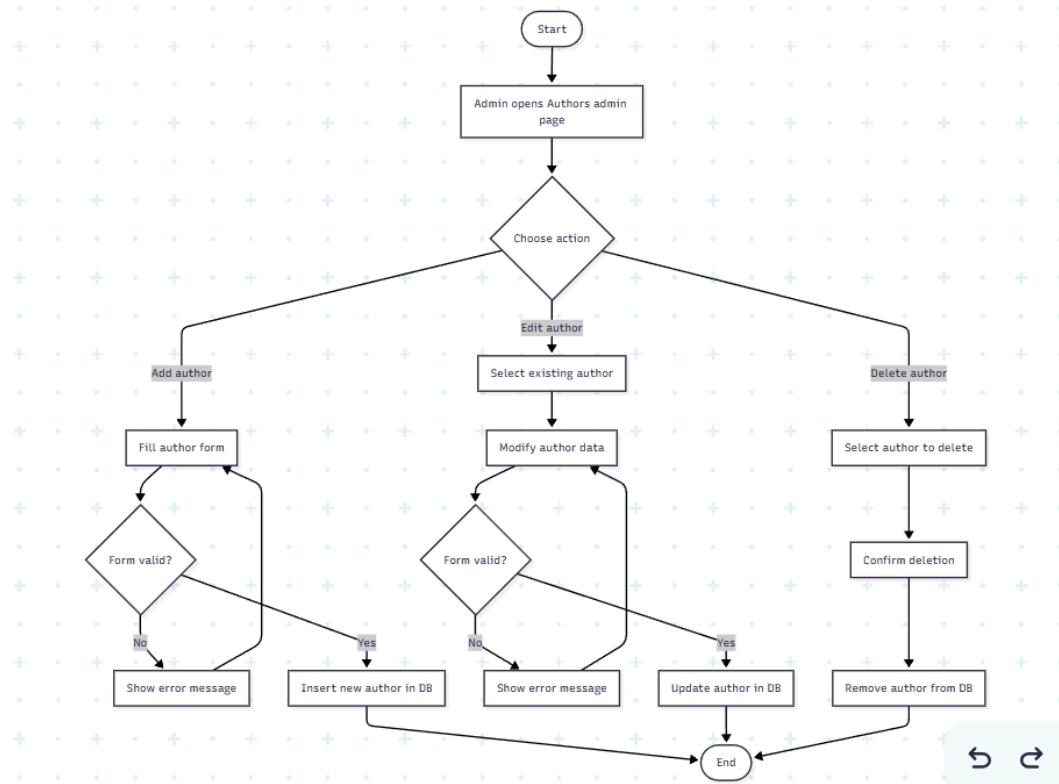
Activity diagram
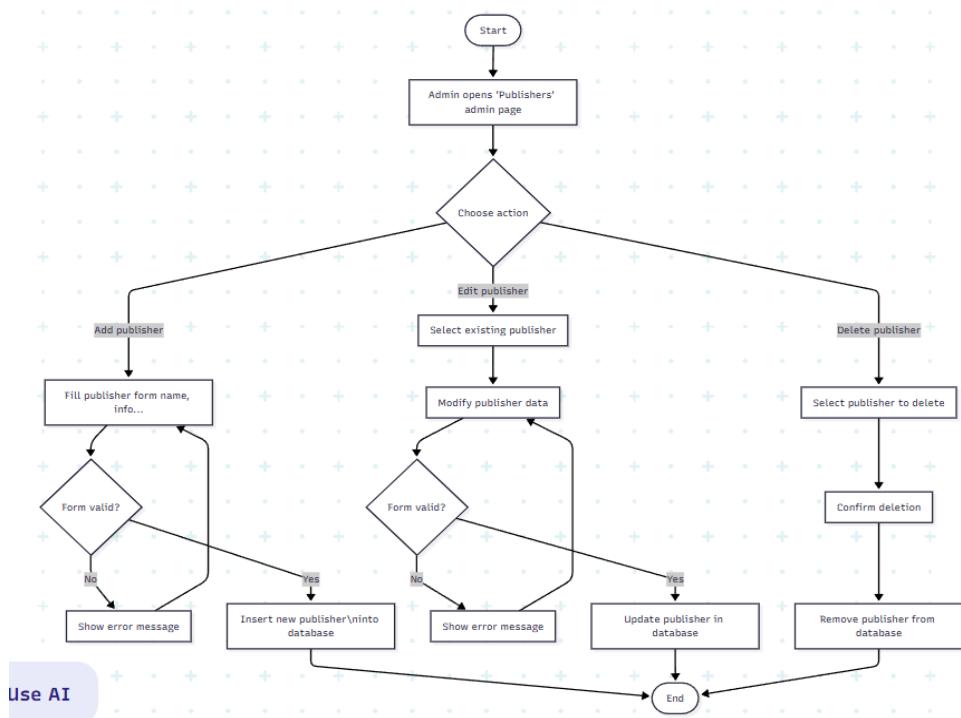
Renaud : sign in, log in, search book global

Max : manage authors

Start
↓
Admin opens Authors admin page
↓
Choose action
- Add author → Fill author form → Form valid? → No → Show error message / Yes → Insert new author in DB
- Edit author → Select existing author → Modify author data → Form valid? → No → Show error message / Yes → Update author in DB
- Delete author → Select author to delete → Confirm deletion → Remove author from DB
↓
End

Souhaib : add books, edit books, manage publishers

Start
↓
Admin opens 'Publishers' admin page
↓
Choose action
- Add publisher → Fill publisher form name, info... → Form valid? → No → Show error message / Yes → Insert new publisher\ninto database
- Edit publisher → Select existing publisher → Modify publisher data → Form valid? → No → Show error message / Yes → Update publisher in database
- Delete publisher → Select publisher to delete → Confirm deletion → Remove publisher from database
↓
End

Use AI

Wireframes

There is one wireframe per person.

Renaud

```
| BOOKS APP (logo)      | Dashboard | Logout   |

|    ADMIN - MANAGE AUTHORS               |

| [Add New Author] (button)               |

| Existing Authors                        |
|  _____    |
| |  John Smith  |  Edit  |  Delete    |  |
| |_____|    |
|  _____    |
| |  Jane Doe   |  Edit  |  Delete     |  |
| |_____|    |

| Add / Edit Author (modal or section)    |
| Name: [_____]              |
| Bio:  [_____]              |
| (Save)   (Cancel)                       |

| Footer                                  |
```

Max

```
| BOOKS APP (logo)       | Home | Login    |

| Search: [_____] (Search)            |

| Book List                          |
|  _____        |
| | [Cover]  Book Title  ★★★★☆  |      |
| | Author: John Smith        |       |
| | (View details)            |       |
| |_____|         |
|  _____        |
| | [Cover]  Another Title ★★★☆☆ |    |
| | Author: Jane Doe          |       |
| | (View details)            |       |
| |_____|         |

| Footer (optional)                  |
```

Souhaib
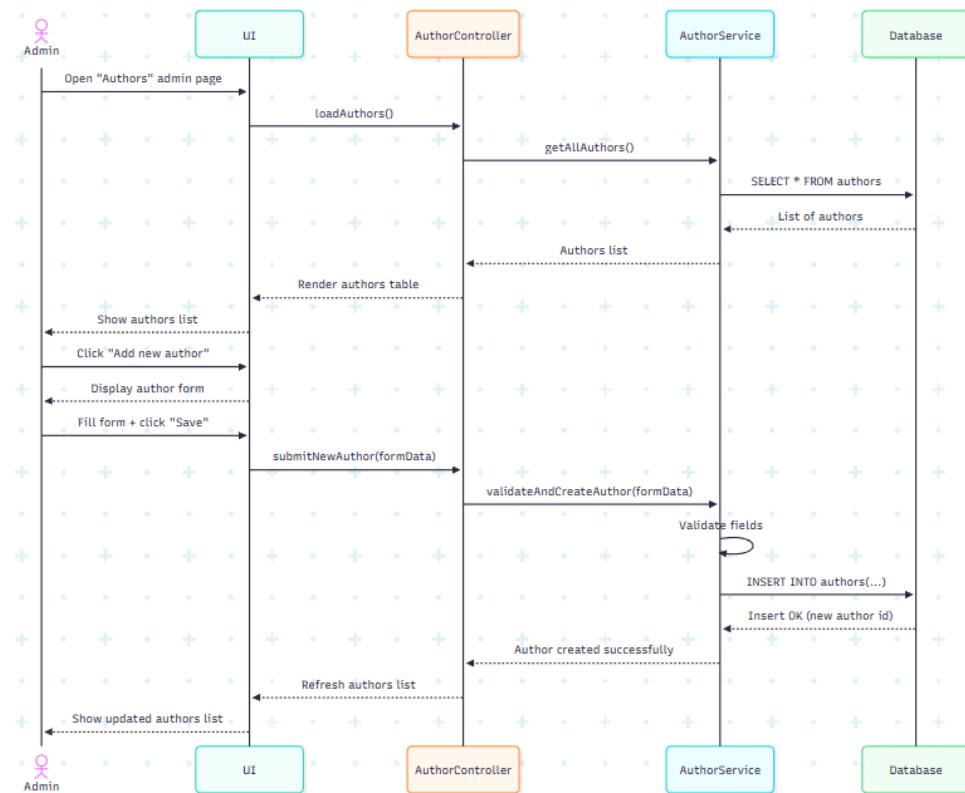
```
 _____
|                                        |
| BOOKS APP (logo)    | Admin Panel | Logout |
|_____|


|          ADD A NEW BOOK              |
|_____|


| Title:          [_____]    |
| Author:         [Dropdown ▼]          |
| Publisher:      [Dropdown ▼]          |
| Language:       [Dropdown ▼]          |
| Genres:         [ ] Fantasy  [ ] Drama |
|                 [ ] Action   [ ] Sci-Fi |
|                                        |
| Description:    [_____]    |
|                 [_____]    |
|_____|


| ( Save )            ( Cancel )        |
|_____|


| Footer                                 |
|_____|
```
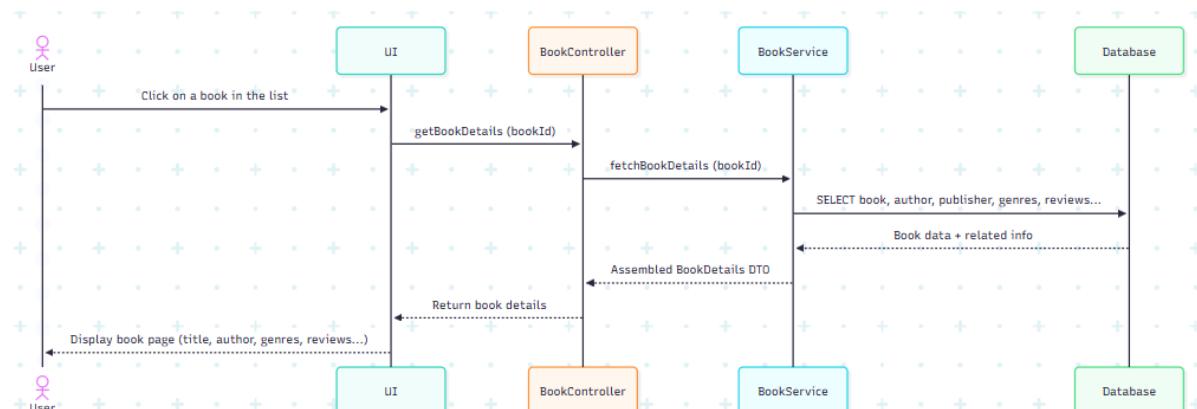
Sequence diagram

There is one sequence diagram per person.
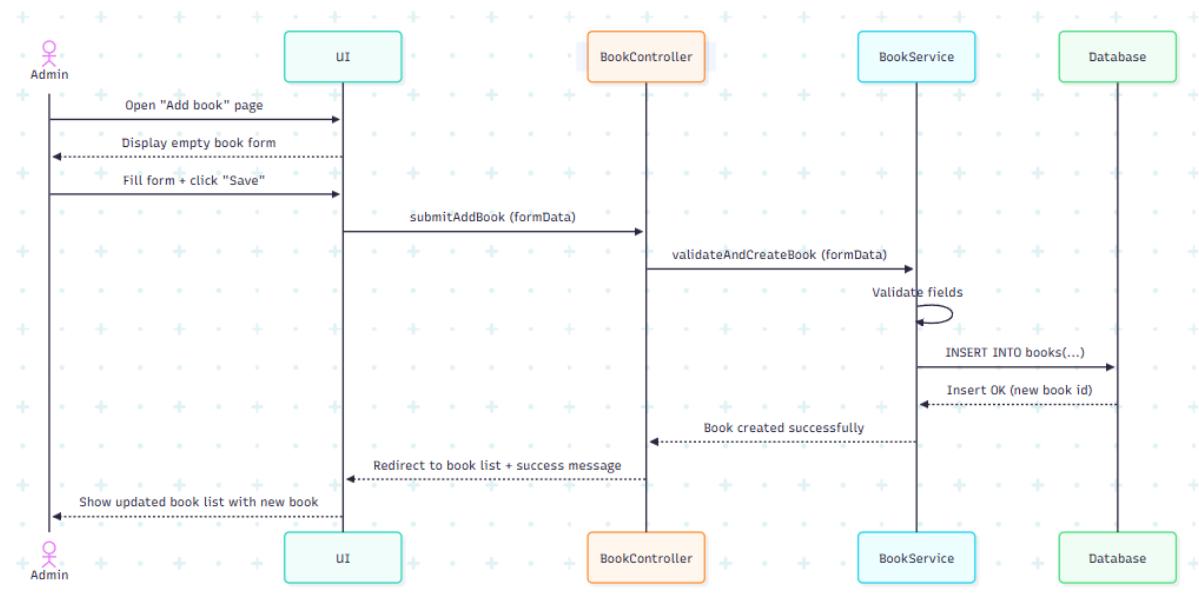
Renaud : manage authors

Max : view books details

Souhaib : add book

Sequence diagram with participants: Admin, UI, BookController, BookService, Database

- Admin → UI: Open "Add book" page
- UI → Admin: Display empty book form
- Admin → UI: Fill form + click "Save"
- UI → BookController: submitAddBook (formData)
- BookController → BookService: validateAndCreateBook (formData)
- BookService → BookService: Validate fields
- BookService → Database: INSERT INTO books(...)
- Database → BookService: Insert OK (new book id)
- BookService → BookController: Book created successfully
- BookController → UI: Redirect to book list + success message
- UI → Admin: Show updated book list with new book

Class diagram

The class diagram shows the internal structure of the **Books App** using object-oriented design.

It separates the system into three layers: the domain layer (Book, Author, Publisher, User, etc.), the application layer (controllers and services), and the Persistence layer (repository interfaces).

The relationships between classes describe how books, authors, publishers, and reviews interact, while inheritance between User and Admin illustrates role specialization.

The dependencies between controllers, services, and repositories highlight the system's data and logic flow.

Overall, the diagram provides a clear and structured view of how the application is organized.

Component diagram
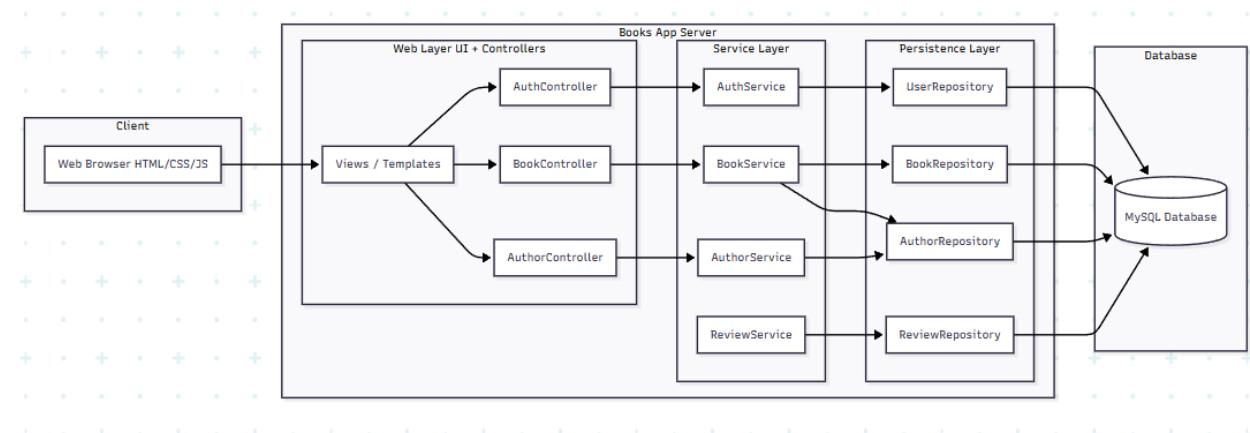
The component diagram illustrates the high-level architecture of the Books App.

The system is divided into three main layers: the Web Layer, which contains the UI and controllers; the Service Layer, which implements the business logic; and the Persistence Layer, which handles data access through repository components.

The client interacts with the application through the browser, which communicates with controllers.

Controllers delegate operations to services, and services retrieve or store data through repositories connected to the database.

This diagram provides a clear overview of how the major components of the application interact and how responsibilities are distributed across layers.
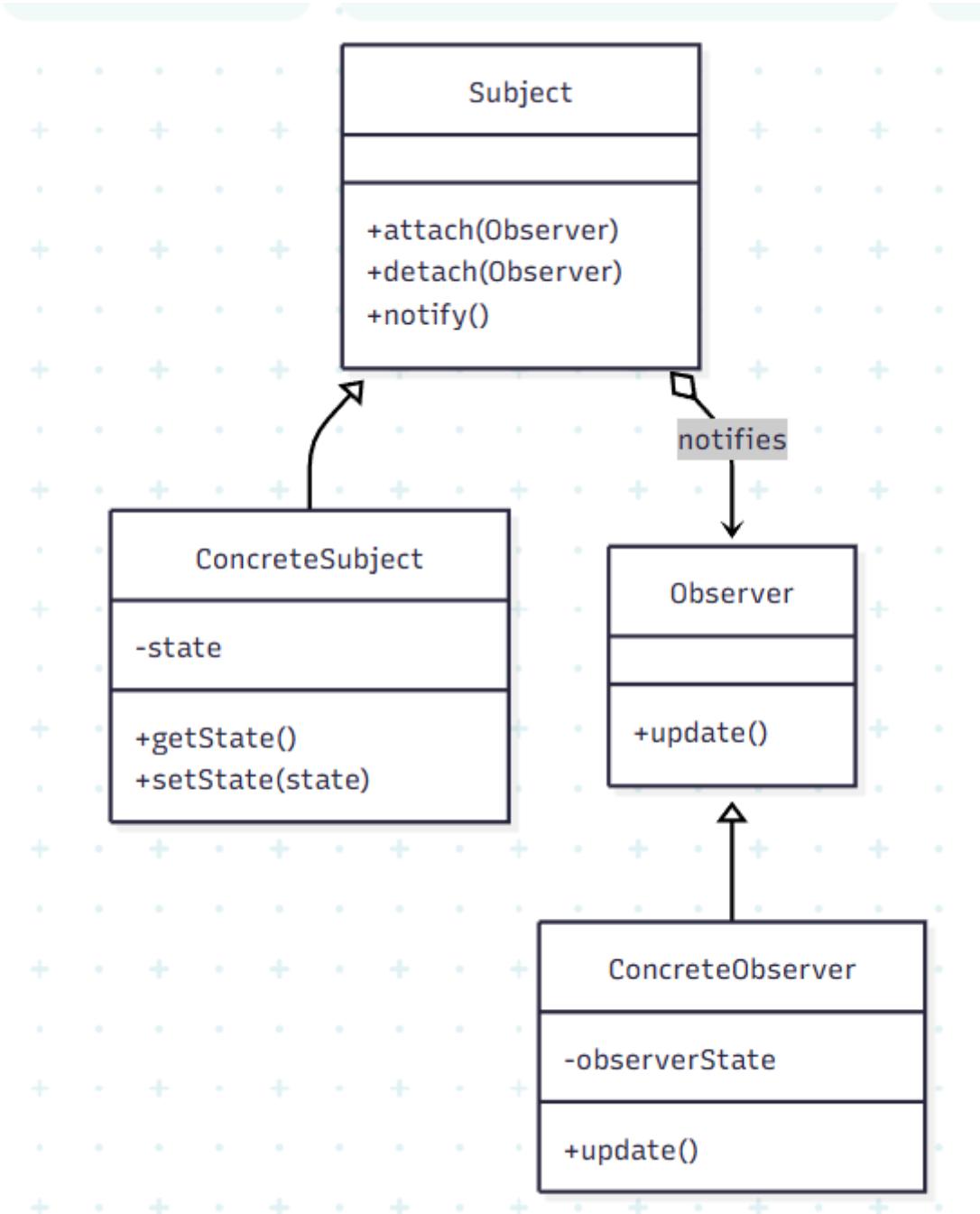
Behavioral diagram

When an admin adds a new book, the **BookService** can act as a Subject.
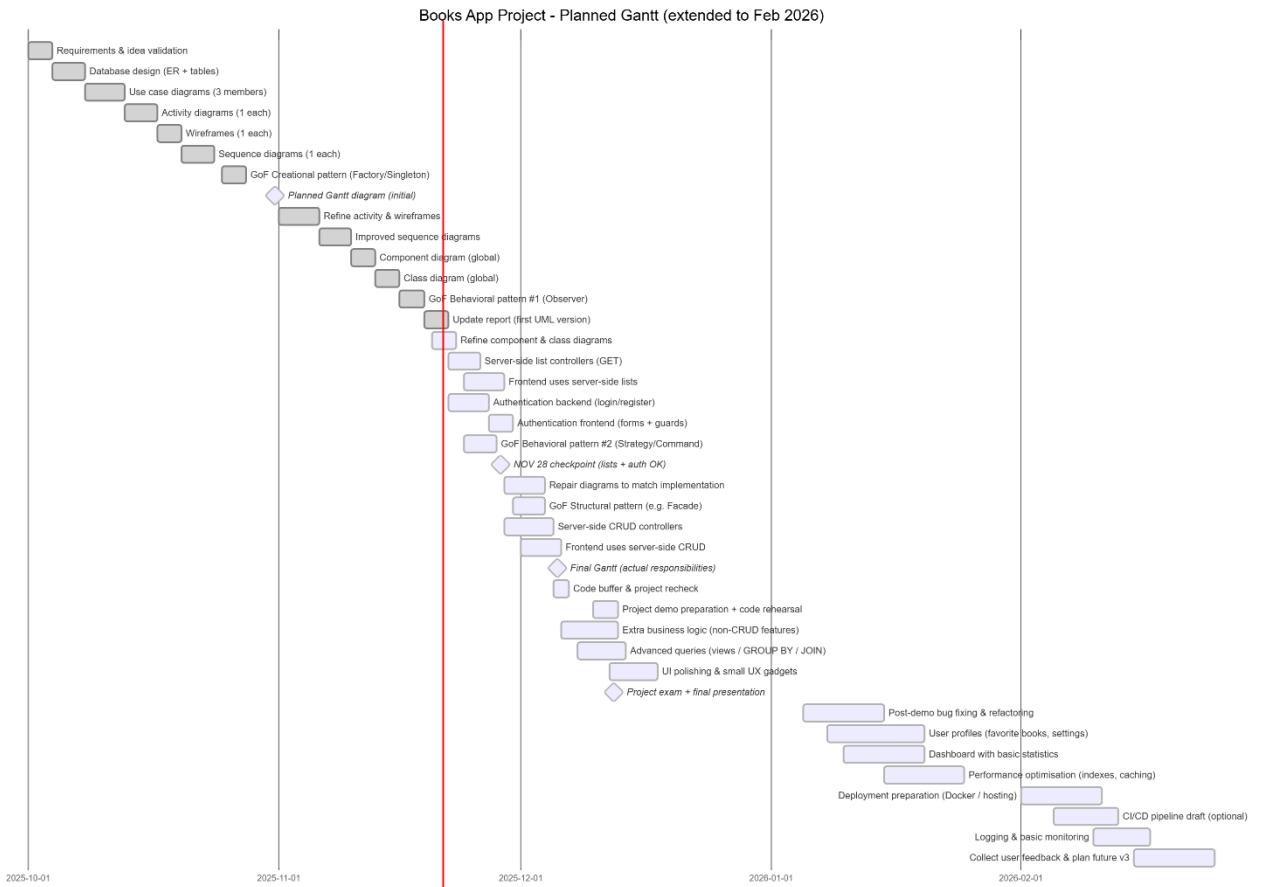
Any UI component displaying the list of books becomes an Observer.

When the service inserts the new book, it calls notify(), and all observers automatically refresh their displayed data.

This makes the system easier to maintain and prevents inconsistencies in the interface.

Gantt diagram

Books App Project - Planned Gantt (extended to Feb 2026)

Task	
Requirements & idea validation	
Database design (ER + tables)	
Use case diagrams (3 members)	
Activity diagrams (1 each)	
Wireframes (1 each)	
Sequence diagrams (1 each)	
GoF Creational pattern (Factory/Singleton)	
Planned Gantt diagram (initial)	
Refine activity & wireframes	
Improved sequence diagrams	
Component diagram (global)	
Class diagram (global)	
GoF Behavioral pattern #1 (Observer)	
Update report (first UML version)	
Refine component & class diagrams	
Server-side list controllers (GET)	
Frontend uses server-side lists	
Authentication backend (login/register)	
Authentication frontend (forms + guards)	
GoF Behavioral pattern #2 (Strategy/Command)	
NOV 28 checkpoint (lists + auth OK)	
Repair diagrams to match implementation	
GoF Structural pattern (e.g. Facade)	
Server-side CRUD controllers	
Frontend uses server-side CRUD	
Final Gantt (actual responsibilities)	
Code buffer & project recheck	
Project demo preparation + code rehearsal	
Extra business logic (non-CRUD features)	
Advanced queries (views / GROUP BY / JOIN)	
UI polishing & small UX gadgets	
Project exam + final presentation	
Post-demo bug fixing & refactoring	
User profiles (favorite books, settings)	
Dashboard with basic statistics	
Performance optimisation (indexes, caching)	
Deployment preparation (Docker / hosting)	
CI/CD pipeline draft (optional)	
Logging & basic monitoring	
Collect user feedback & plan future v3	

2025-10-01   2025-11-01   2025-12-01   2026-01-01   2026-02-01

Gant diagram based on User case diagram

Books App - Use Case Implementation Gantt (Extended far beyond deadlines)

Visitor Use Cases
- Browse book list
- Search books basic
- Filter by title author genre
- View book details
- Sign up create account
- Add dark mode toggle
- Add scroll to top button
- Improve book list card layout
- Add quick search suggestions

User Use Cases
- Login
- Logout
- Search books advanced filters
- View extended details with reviews
- Save favorite books
- Reading history page
- User profile page
- Star based rating UI

Admin Use Cases
- Add books
- Edit books
- Delete books
- Manage authors
- Manage publishers
- Admin dashboard basic stats
- Bulk upload CSV import
- Auto resize book covers
- Preview changes before saving

System Shared Features
- Improve search speed index db
- Pagination for big lists
- Simple caching layer
- Redesign error pages 404 500
- Simple book recommendations algo
- Weekly newsletter feature
- Achievement badges reading levels

Future Crazy Ideas
- AI based book recommendations
- Voice search find books
- Image recognition book cover
- Social feed what friends read
- Gamification xp levels leaderboards
- Mobile companion app basic features

2025-11-01   2025-12-01   2026-01-01   2026-02-01   2026-03-01   2026-04-01   2026-05-01   2026-06-01