GUL-E-NARJIS

SP23-BSE-023

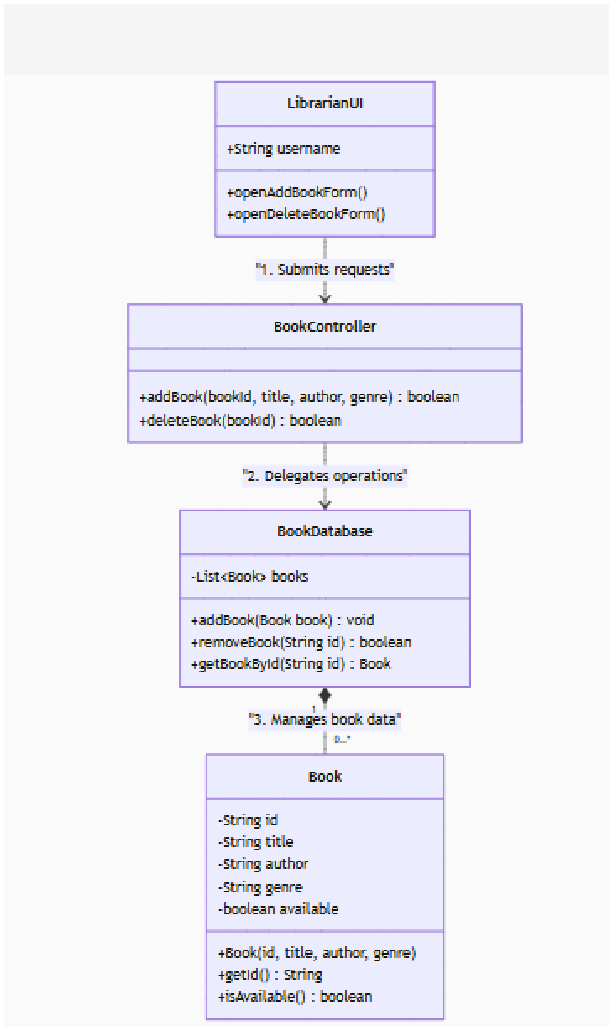
Fully dressed use case:

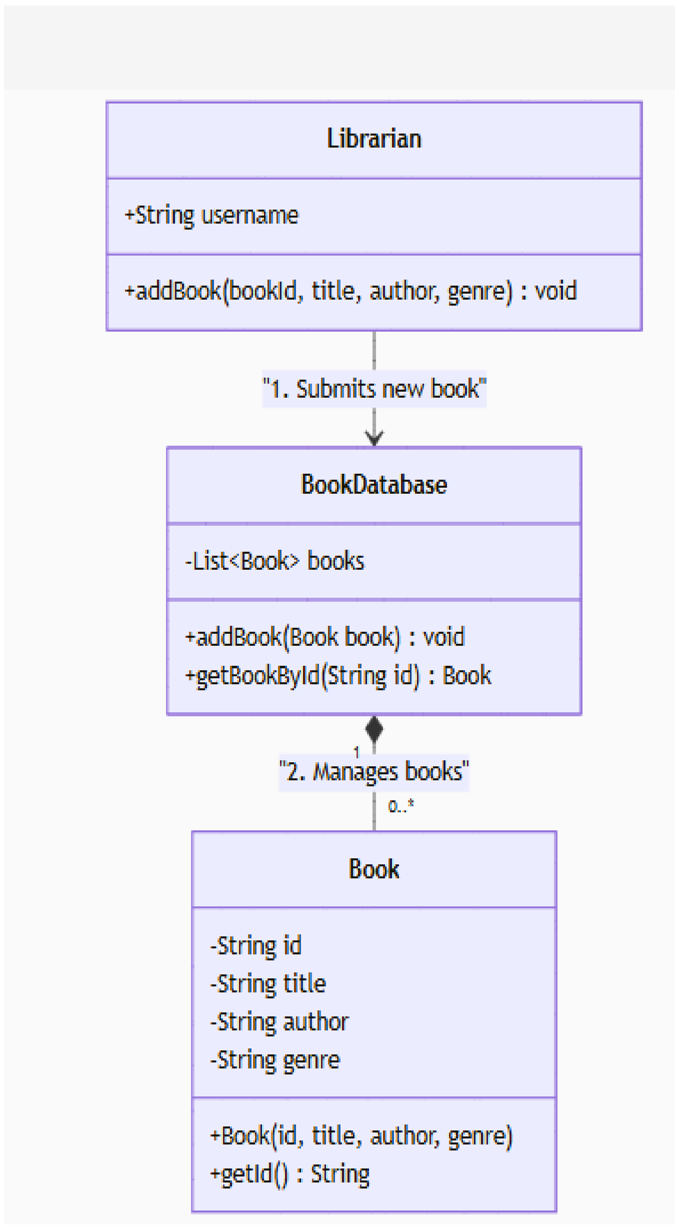
ADD BOOK:

Field	**Details**
Use Case ID	UC001
Use Case Name	Add Book
Brief Description	Allows a librarian to add a new book to the system by entering its details (ID, title, author, genre). The system validates the input and stores the book in the database.
Primary Actor	Librarian
Secondary actor	BookDatabase
Preconditions	- Librarian is logged in. - BookDatabase is accessible.
Postconditions	- New book is added to the books list in BookDatabase. - Book becomes searchable in the system.
Main Success Scenario	1. Librarian selects "Add Book" in the UI. 2. System displays a form with fields: ID, Title, Author, Genre. 3. Librarian enters details (e.g. "B004", "New Title", "New Author", "New Genre"). 4. System calls BookDatabase.addBook(new Book(...)). 5. System validates: - ID is unique (BookDatabase.getBookById(id) == null). 6. Book is added to the books list. 7. System displays: "Book added successfully!"

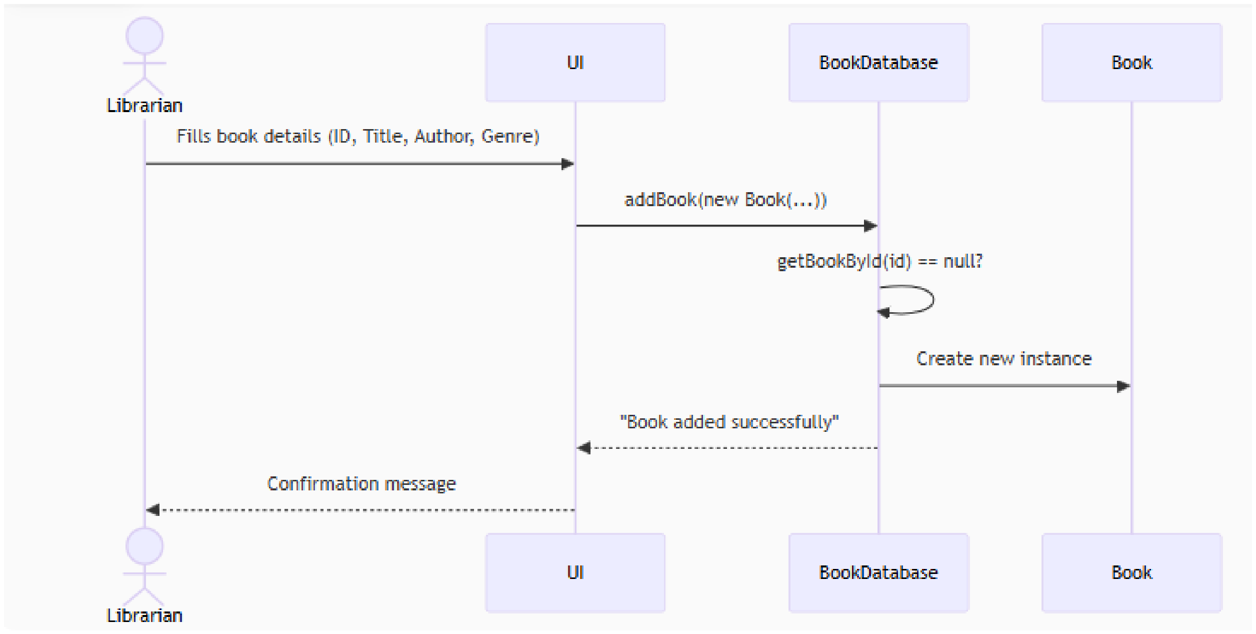
Alternative Flows	5a. Duplicate ID: System rejects with: “Error: Book ID already exists.”
Exception Flows	Database Unavailable: System shows: “Error: Could not add book. Try later.”
Business Rules	- Book ID must be unique. - All fields (title, author, genre) are mandatory.

Class diagram:




Librarian
+String username
+addBook(bookId, title, author, genre) : void
"1. Submits new book"
BookDatabase
-List<Book> books
+addBook(Book book) : void
+getBookById(String id) : Book
1
"2. Manages books"
0..*
Book
-String id
-String title
-String author
-String genre
+Book(id, title, author, genre)
+getId() : String

Sequence diagram:



1. Add Book System Event Design

Actor Action (Trigger)	**System Event**	**System Description / Response**
Librarian selects "Add Book"	initiateAddBook()	System loads the "Add Book" form with fields: ID, Title, Author, Genre.
Librarian fills in book details	—	Data entered manually (not a system event).
Librarian clicks "Submit"	validateBookInput(bookData)	1. Checks ID uniqueness (getBookById(id) == null) 2. Validates required fields (title, author).
(Optional) Uploads book cover	uploadBookCover(imageFile)	*[Not in your code, but added for completeness]*
System confirms addition	confirmBookAddition()	Updates UI: "Book [Title] added successfully!"

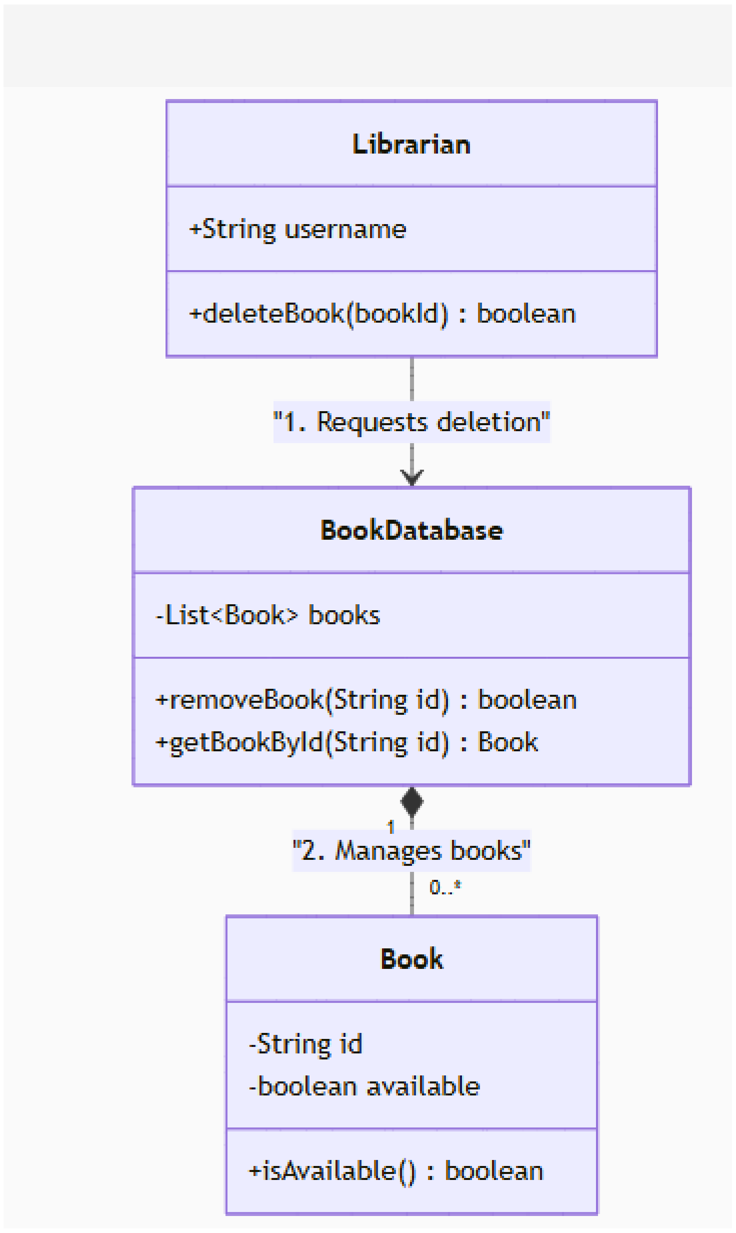
Fully dressed use case:

DELETE BOOK:

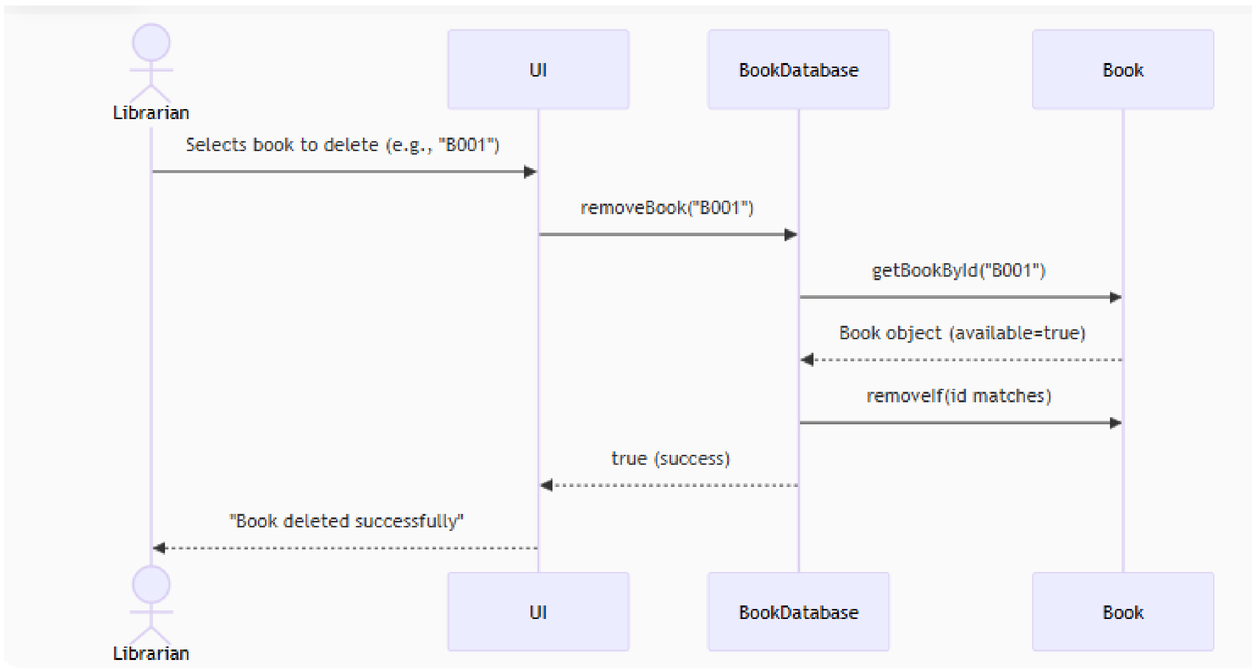
Field	**Details**
Use Case ID	UC002
Use Case Name	Delete Book
Brief Description	Allows a librarian to permanently remove a book from the system, provided it is not currently borrowed.
Primary Actor	Librarian
Secondary actor	BookDatabase
Preconditions	- Librarian is logged in. - Book exists in the books list.
Postconditions	- Book is removed from the books list. - Book is no longer visible in searches.
Main Success Scenario	1. Librarian selects "Delete Book" in the UI. 2. System displays a list of books via BookDatabase.getAllBooks(). 3. Librarian selects a book by ID (e.g. "B001"). 4. System calls BookDatabase.removeBook(id). 5. System checks: - Book exists (BookDatabase.getBookById(id) != null). - Book is not borrowed (book.isAvailable() == true). 6. Book is removed from the books list. 7. System displays: "Book deleted successfully!"
Alternative Flows	5a. Book is Borrowed: System rejects with: "Error: Book is currently borrowed and cannot be deleted."
Exception Flows	Invalid ID: System shows:

	"Error: Book not found."
Business Rules	- Only available books (isAvailable() == true) can be deleted. - Deletion is permanent (no recovery in current code).

Class diagram:



Sequence diagram:



2. Delete Book System Event Design

Actor Action (Trigger)	System Event	System Description / Response
Librarian selects "Delete Book"	initiateDeleteBook()	System displays list of books via getAllBooks().
Librarian selects a book (by ID)	—	Manual selection (not a system event).
Librarian confirms deletion	validateDeleteRequest(id)	1. Checks book exists (getBookById(id) != null) 2. Verifies isAvailable() == true.
System processes deletion	removeBookFromDB(id)	Removes book from books list and updates UI

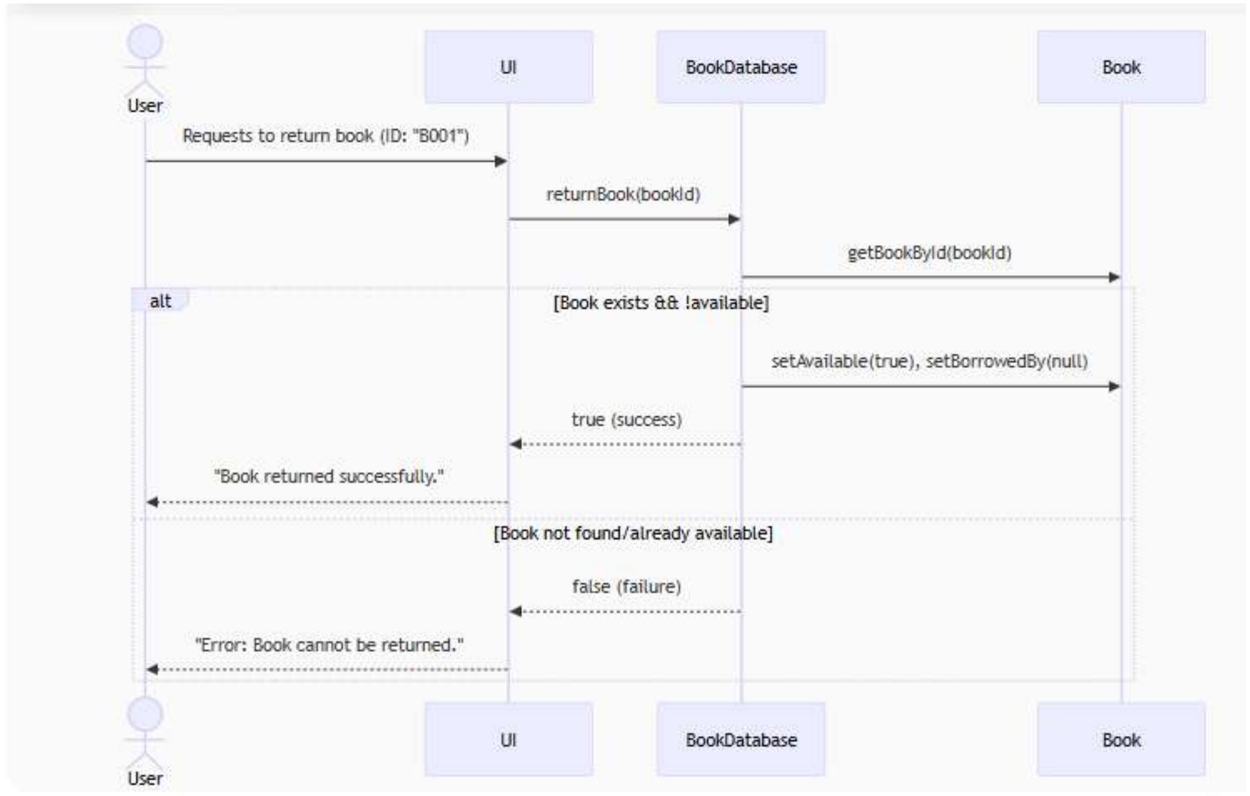
Fully dressed use case:

RETURN BOOK:

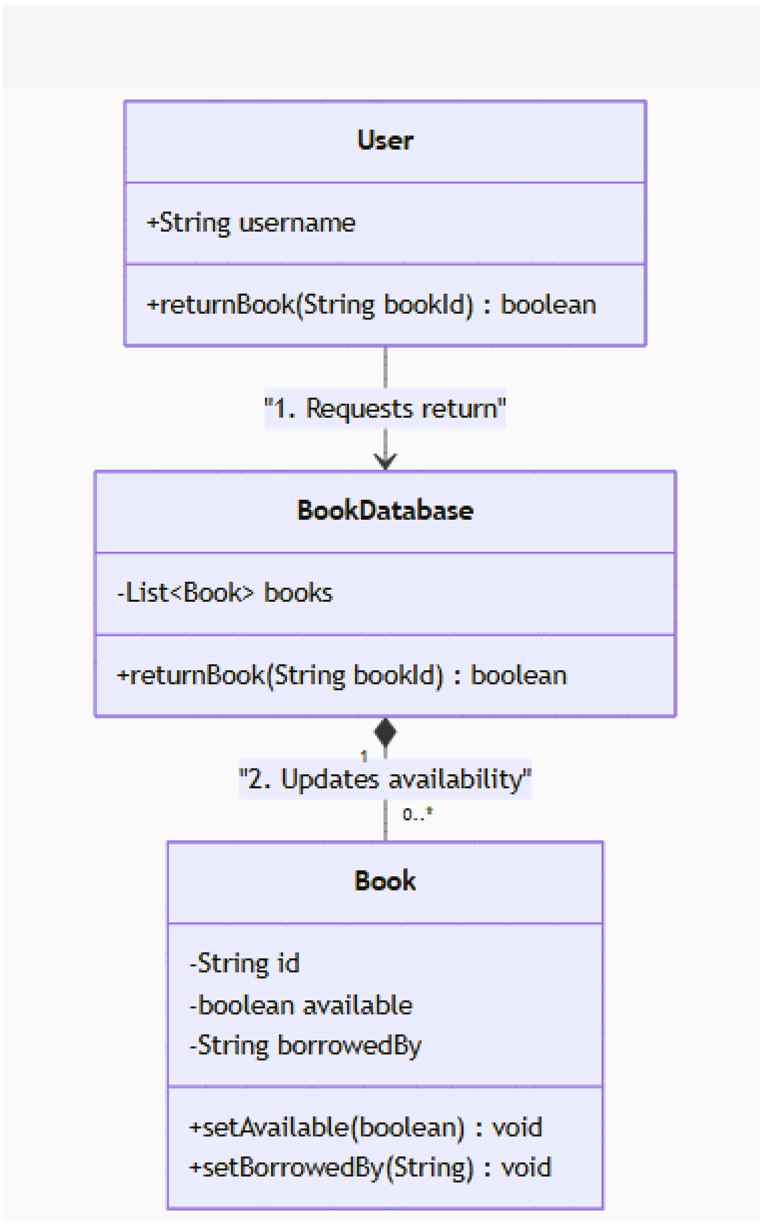
Field	**Description**
Use Case ID	UC004
Use Case Name	Return Book
Primary Actor	Member / Librarian
Secondary Actor	BookDatabase System
Brief Description	Allows users to return a borrowed book, updating its availability status.
Preconditions	- User is logged in. - Book exists and is currently borrowed (book.isAvailable() == false).
Postconditions	- Book status is set to available. - borrowedBy field is reset to null.
Basic Flow	1. User selects "Return Book" and provides the book ID. 2. System calls BookDatabase.returnBook(bookId). 3. System validates: - Book exists (getBookById(bookId) != null) - Book is not already available (!book.isAvailable()) 4. System updates book: - available = true - borrowedBy = null 5. System confirms: "Book [Title] returned successfully."

Alternative Flows	3a. Book Already Available:System shows error: "This book is not currently borrowed."
Exception Flows	- Invalid Book ID: System shows error: "Book not found." - Database Error: System shows error: "Return failed. Please try later."
Business Rules	- Only the borrower or librarian can return a book. - Audit logs may track return actions.

Sequence diagram:


User
UI
BookDatabase
Book
Requests to return book (ID: "B001")
returnBook(bookId)
getBookById(bookId)
alt
[Book exists && !available]
setAvailable(true), setBorrowedBy(null)
true (success)
"Book returned successfully."
[Book not found/already available]
false (failure)
"Error: Book cannot be returned."
User
UI
BookDatabase
Book

Class diagram:


User
+String username
+returnBook(String bookId) : boolean
"1. Requests return"
BookDatabase
-List<Book> books
+returnBook(String bookId) : boolean
1
"2. Updates availability"
0..*
Book
-String id
-boolean available
-String borrowedBy
+setAvailable(boolean) : void
+setBorrowedBy(String) : void

4. Return Book System Event Design

Actor Action (Trigger)	System Event	System Description / Response
User selects "Return Book"	initiateReturn()	System prompts for bookID.
User enters book ID	—	Manual input (not a system event).
User submits return request	validateReturnRequest(id)	1. Checks book exists (getBookById(id) != null) 2. Verifies isAvailable() == false.
System processes return	updateBookStatus(id)	Sets available=true, borrowedBy=null, and confi

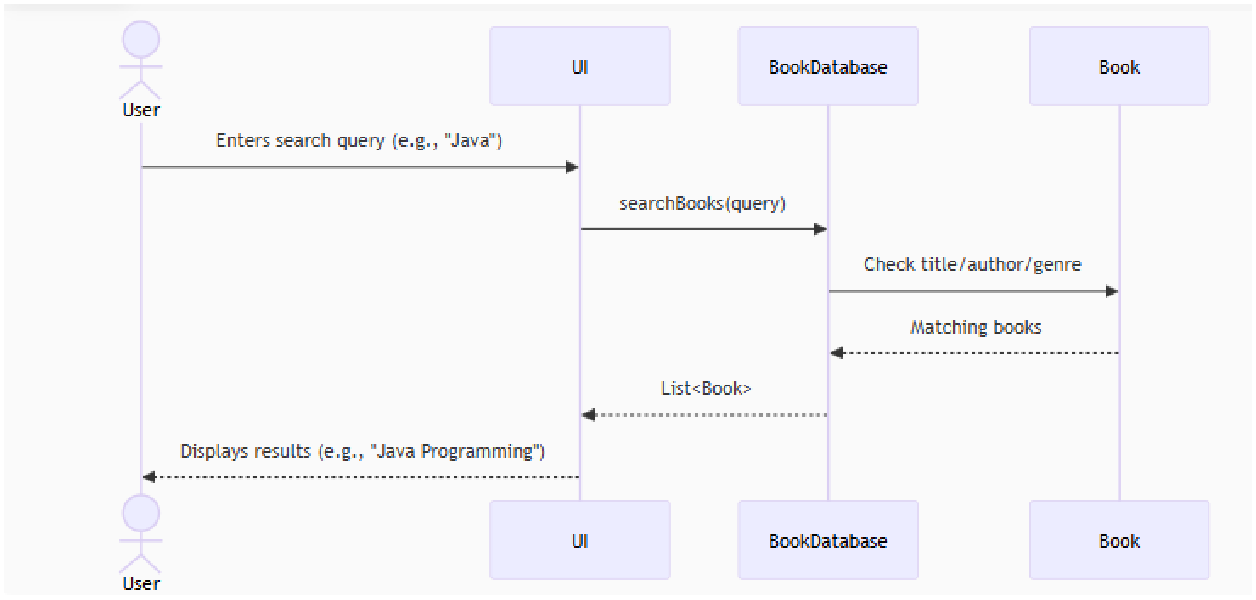
Fully dressed use case:

SEARCH BOOK:

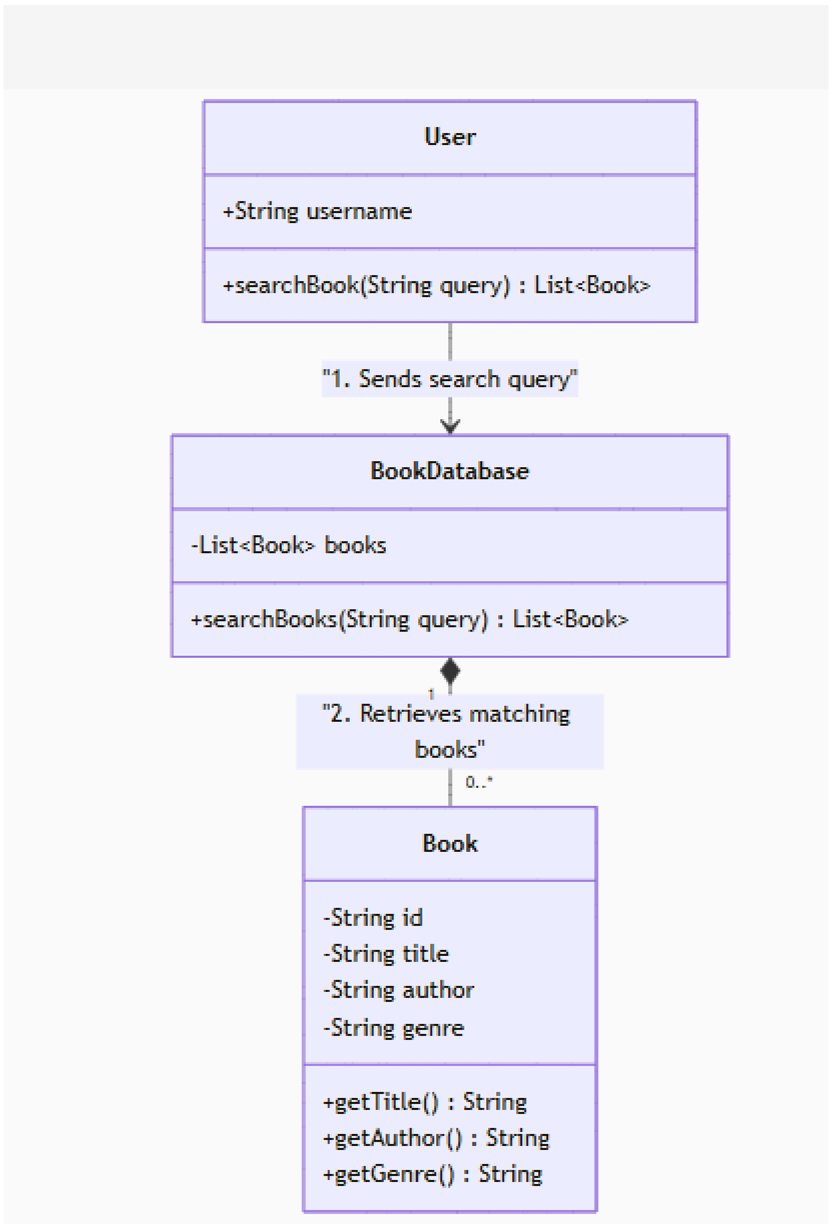
Use Case Element	Description
Use Case ID	UC004
Use Case Name	Search Book
Primary Actor	User (Member/Librarian)
Secondary Actor	Book System
Brief Description	Allows users to search for books by title, author, or genre.
Preconditions	- User is logged in (optional, depending on system requirements).- BookDatabase is accessible.
Postconditions	- Search results are displayed.
Basic Flow	1. User enters a search query (title/author/genre). 2. System calls BookDatabase.searchBooks(query). 3. System retrieves matching books.

	4. System displays the list of matching books (title, author, genre, availability).
Alternative Flows	2a. No Results Found: - System displays "No books found matching your query."
Exception Flows	Database Unavailable: - System shows error: "Search unavailable. Please try later."
Business Rules	- Search is case-insensitive. - Partial matches are allowed (e.g., "Java" matches "Java Programming").

Sequence diagram:



Class diagram:



3. Search Book System Event Design

Actor Action (Trigger)	**System Event**	**System Description / Response**
User enters search query	initiateSearch(query)	System prepares to search by title, author, or genre.
User clicks "Search"	executeSearch(query)	1. Calls searchBooks(query) 2. Returns List<Book> of matches (case-insensitive).
System displays results	displaySearchResults()	Shows books or "No results found".