Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

Software Development Process

Assignment 03: Library Management System

Link Repository: GitLab Project

Authors of the paper:
Simone Bettelli: 859719
Andrea Messa: 856435

A.A. 2023-2024

Contents

Chapter 1

Library System

1.1 Application description

This application was designed as a prototype for an online library.
The user is allowed to register within the system in order to be able to make use of the various services dedicated to him/her. To achieve this, the application provides various functionalities such as:

- **`Register:`** allows you to register a user within the online library and assign him/her an identifier to recognise him/her within the library system.
- **`Home:`** allows the user to view the list of books he has borrowed in his name. It is also possible to display the list of books available in the library that have not yet been reserved by other users and to keep track of the history of books that have been reserved by the user over time.

Furthermore, once the user has logged into the system, on the Home page it is possible to search for books in the library catalogue. This search can be carried out in two different ways.
The first mode allows the book to be found by its `ISBN` which represents the unique code assigned to the book, the `name` of the author who wrote it and the `title` of the book itself.
The second mode allows books to be displayed via its `publisher` concatenated with a certain author of a certain `nationality`. In both cases it will then be possible to add the searched books to one's own private list with the help of a button.

1.2 Entity:

The entities that are part of our library system are as follows:

- `Book:` represents the individual book in the library. It is composed of several attributes that characterise it such as: the `ISBN` (uniquely identifies the book within the system), `title` (represents the title of the book), `author` (represents the author of the book), `genre` (indicates the genre to which the book belongs) and the `publisher` (indicates the publisher of the book).
- `Author:` represents the individual author. It is also composed of several attributes such as: `author ID` (which allows the author of the book to be uniquely identified), `name` (author's first name), `surname` (author's surname), `birthdate` (represents the author's date of birth) and `nationality` (represents the author's nationality).
- `Genre:` represents the genre of the book. It is composed of the following attributes: `genre ID` (allows us to uniquely identify the individual genre) and the `name` (indicates the name of the genre we want to associate with the book).
- `Library Member:` represents the individual library member. Each individual member, after the registration phase, will be assigned a `library member ID` in order to be identified within the library system. This is similar to the concept of a physical library card. It is also composed of several attributes such as: `member ID` (allows the member to be identified within the system), the `name` and the `surname` (these are those of the user entered during the registration phase), and `membership date` (which represents the date of account creation).
- `Ebook:` represents an electronic book. This entity inherits all attributes from Book and includes additional attributes such as `format` (i.e. the format of the Ebook) and `fileSizeMB` (i.e. the size of the Ebook expressed in MB).
- `User:` represents the user of the application. It is also composed of several attributes such as: `UserID` (which allows each individual user to be uniquely identified within the system), `name` (this attribute represents the user's first name), `surname` (represents the user's surname), `username` (this attribute represents the name that the user wants to give to his profile), `password` (this attribute represents the password that the user must enter into the system to log in), `email` (this attribute represents the user's e-mail address), `phoneNumber` (this attribute represents the user's telephone number), `city` (this attribute represents the user's home town).

1.3 UML Diagram Complete

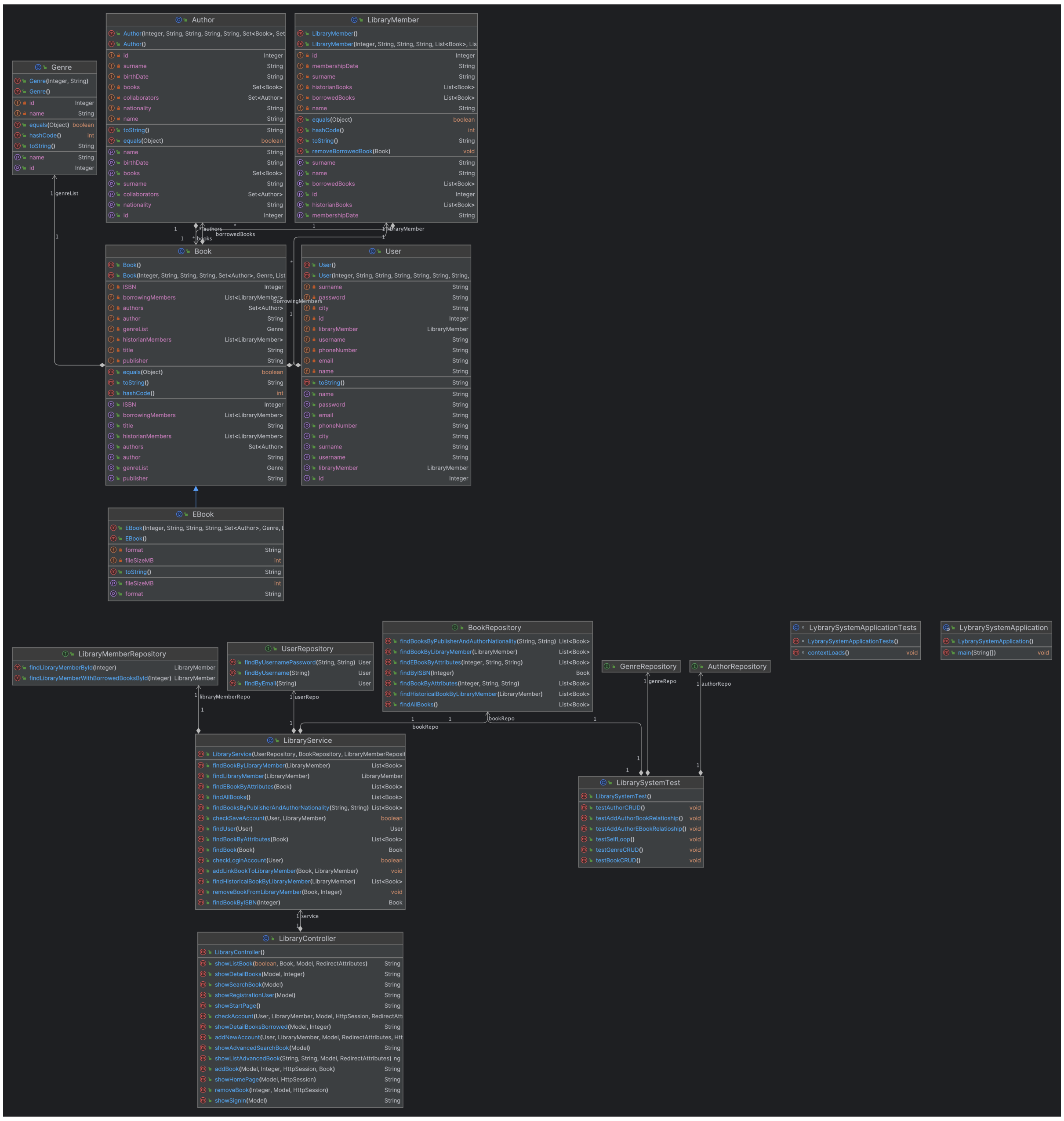
Genre
Genre(Integer, String)
Genre()
id Integer
name String
equals(Object) boolean
hashCode() int
toString() String
name String
id Integer
Author
Author(Integer, String, String, String, String, Set<Book>, Set
Author()
id Integer
surname String
birthDate String
books Set<Book>
collaborators Set<Author>
nationality String
name String
toString() String
equals(Object) boolean
name String
birthDate String
books Set<Book>
surname String
collaborators Set<Author>
nationality String
id Integer
LibraryMember
LibraryMember()
LibraryMember(Integer, String, String, String, List<Book>, Lis
id Integer
membershipDate String
surname String
historianBooks List<Book>
borrowedBooks List<Book>
name String
equals(Object) boolean
hashCode() int
toString() String
removeBorrowedBook(Book) void
surname String
name String
borrowedBooks List<Book>
id Integer
historianBooks List<Book>
membershipDate String
1 genreList
1
1
authors
*
borrowedBooks
books
libraryMember
borrowingMembers
Book
Book()
Book(Integer, String, String, String, Set<Author>, Genre, List
ISBN Integer
borrowingMembers List<LibraryMember>
authors Set<Author>
author String
genreList Genre
historianMembers List<LibraryMember>
title String
publisher String
equals(Object) boolean
toString() String
hashCode() int
ISBN Integer
borrowingMembers List<LibraryMember>
title String
historianMembers List<LibraryMember>
authors Set<Author>
author String
genreList Genre
publisher String
User
User()
User(Integer, String, String, String, String, String, String,
surname String
password String
city String
id Integer
libraryMember LibraryMember
username String
phoneNumber String
email String
name String
toString() String
name String
password String
email String
phoneNumber String
city String
surname String
username String
libraryMember LibraryMember
id Integer
EBook
EBook(Integer, String, String, String, Set<Author>, Genre, L
EBook()
format String
fileSizeMB int
toString() String
fileSizeMB int
format String
LibraryMemberRepository
findLibraryMemberById(Integer) LibraryMember
findLibraryMemberWithBorrowedBooksById(Integer) LibraryMember
UserRepository
findByUsernamePassword(String, String) User
findByUsername(String) User
findByEmail(String) User
BookRepository
findBooksByPublisherAndAuthorNationality(String, String) List<Book>
findBookByLibraryMember(LibraryMember) List<Book>
findEBookByAttributes(Integer, String, String) List<Book>
findByISBN(Integer) Book
findBookByAttributes(Integer, String, String) List<Book>
findHistoricalBookByLibraryMember(LibraryMember) List<Book>
findAllBooks() List<Book>
GenreRepository
AuthorRepository
LybrarySystemApplicationTests
LybrarySystemApplicationTests()
contextLoads() void
LybrarySystemApplication
LybrarySystemApplication()
main(String[]) void
libraryMemberRepo
userRepo
bookRepo
genreRepo
authorRepo
LibraryService
LibraryService(UserRepository, BookRepository, LibraryMemberReposit
findBookByLibraryMember(LibraryMember) List<Book>
findLibraryMember(LibraryMember) LibraryMember
findEBookByAttributes(Book) List<Book>
findAllBooks() List<Book>
findBooksByPublisherAndAuthorNationality(String, String) List<Book>
checkSaveAccount(User, LibraryMember) boolean
findUser(User) User
findBookByAttributes(Book) List<Book>
findBook(Book) Book
checkLoginAccount(User) boolean
addLinkBookToLibraryMember(Book, LibraryMember) void
findHistoricalBookByLibraryMember(LibraryMember) List<Book>
removeBookFromLibraryMember(Book, Integer) void
findBookByISBN(Integer) Book
LibrarySystemTest
LibrarySystemTest()
testAuthorCRUD() void
testAddAuthorBookRelatioship() void
testAddAuthorEBookRelatioship() void
testSelfLoop() void
testGenreCRUD() void
testBookCRUD() void
service
LibraryController
LibraryController()
showListBook(boolean, Book, Model, RedirectAttributes) String
showDetailBooks(Model, Integer) String
showSearchBook(Model) String
showRegistrationUser(Model) String
showStartPage() String
checkAccount(User, LibraryMember, Model, HttpSession, RedirectAtt
showDetailBooksBorrowed(Model, Integer) String
addNewAccount(User, LibraryMember, Model, RedirectAttributes, Htt
showAdvancedSearchBook(Model) String
showListAdvancedBook(String, String, Model, RedirectAttributes) ng
addBook(Model, Integer, HttpSession, Book) String
showHomePage(Model, HttpSession) String
removeBook(Integer, Model, HttpSession) String
showSignIn(Model) String

Figure 1.1: UML Diagram

1.3.1 Relationships between entities:

The system includes the following relationships between entities:

- `Many-to-Many:` A book may have multiple authors and an author may write multiple books.
- `Self-Loop:` An author may collaborate with other authors, represented as a self-loop relationship.
- `Many-to-One`: A book has only one genre, but one genre can be associated with many books.
- `Many-to-Many:` A library member may borrow several books and a book may be borrowed by several library members.
- `One-to-One:` A user is associated with a unique library member ID and vice versa.

1.4 CRUD Operations

CRUD stands for `Create`, `Read`, `Update`, `Delete` i.e., the basic operations used to manage persistent storage, typically in databases. These operations allow basic interactions with the data:

- `Create:` Adding new data or records to the database.
- `Read:` Retrieving or querying existing data from the database.
- `Update:` Modify or update existing data in the database.
- `Delete:` Remove or delete existing data from the database.

In our case, the system supports the following operations:

- `Create:`. Add new books, authors, genres, library members, users.
- `Read:` View information about books, authors, genres, library members, and users. Display books borrowed by a library member, historical books, and all books in the library.
- `Update:` Edit information on existing books, authors, genres, and library members.
- `Delete:` Remove books, authors, genres, and library members from the system.

1.5 Application structure

The architecture of our project follows the standard `MVC` (Model-View-Controller) which is a design pattern widely used in the software development world to organize code in a modular way and facilitate maintenance.
Our application follows the following structure:

- **`Model:`** represents the application data and business logic. The model classes contain the objects that represent the domain entities. This package contains all the classes listed above.
- **`View:`** responsible for displaying the data to the user and managing the interaction with it. In the specific context, the view often corresponds to the front-end part of the application, which has been implemented with HTML, CSS and JavaScript.
- **`Controller:`** handles user requests, coordinates the necessary actions, and returns the appropriate result. In our application, the controller is annotated with @Controller. Methods within a controller are annotated with @RequestMapping or other similar annotations to map HTTP requests to specific methods.
- **`Repository:`** responsible for communication with the database. In the specific context, the CrudRepository interface of Spring Data JPA is used to simplify CRUD operations on the database.
- **`Service:`** is involved in handling transactions between the controller and repositories.