Project Report

Group 3, class 2-Z

Rafael Sánchez Córdoba (315212)

Rosa Briales Marfil (315250)

María Ortiz Planchuelo (315266)

Franciszek Jan Nurkiewicz (318212)

Alexandru Dulghier (315267)

Ole Ildsgaard Hougaard

Ib Havn

VIA University College

58.209 characters

Software Technology Engineering

2nd Semester

01/06/2022

Table of content

Abstract viii
1 Introduction 1
2 Analysis 3
2.1 Requirements 3
2.2 User Stories 3
2.3 Non-Functional Requirements 5
2.4 Uses cases 6
2.4.1 Use case diagram 6
2.4.2 Use case descriptions 7
2.4.3 Activity and System Sequence diagrams 11
2.5 Domain model 13
3 Design 14
3.1 Patterns and principles 15
3.1.1 MVVM 15
3.1.2 Observer 16
3.1.3 Singleton 17
3.1.4 Adapter 18
3.1.5 Liskov Subbstitution Principle (LSP) 20
3.1.6 Dependency Inversion Principle (DIP) 21
3.1.7 Interface-Segregation Principle (ISP) 21
3.2 Class diagram 23
3.3 Design Persistence 24
3.4 Technologies 27

3.5 Graphical User Interface (GUI) design 28
4 Implementation 30
4.1 Path to add a library user 30
4.2 Patterns implementation 33
4.2.1 Singleton 33
4.2.2 Observer pattern 34
4.3 Database implementation 37
4.4 Libraries and Dependencies 38
5 Test 40
5.1 Black box: Test Cases 41
5.2 White Box: JUnit 43
5.2.1 Z-zero: 45
5.2.2 O-one 46
5.2.3 M-multiple 46
5.2.4 B-Boundaries, I-Interface definition (not relevant) 46
5.2.5 E-Exceptional behaviour (errors and exceptions), S-Simple scenarios (not relevant…… 48
6 Results and Discussion 54
7 Conclusions 55
8 Project future 56
9 Sources of information 57
10 Appendices 59

List of figures and tables

Figure 1. Use case diagram (appendix D) ... 6
Figure 2. Add librarian system sequence diagram (See appendix H) ... 11
Figure 3.Loan item system sequence diagram (See appendix H) ... 12
Figure 4. Log in activity diagram (See appendix H) ... 13
Figure 5.Domain model (appendix ??) ... 14
Figure 6. MVVM pattern class diagram (See appendix G) ... 15
Figure 7. Model class Diagram (See appendix G) ... 16
Figure 8. Property change subject pattern class diagram (See appendix G) ... 16
Figure 9. Property change listener pattern class diagram (See appendix G) ... 17
Figure 10. Singleton pattern of the DAO for Loan Books (see appendix G) ... 18
Figure 11. Interface Storage class diagram (see appendix G) ... 19
Figure 12. Adapter pattern storage class diagram (see appendix G) ... 19
Figure 13. Multimedia Item in class diagram (see appendix G) ... 20
Figure 14. Example ISP class diagram (See appendix G) ... 22
Figure 15.EER database diagram(see appendix N) ... 25
Figure 16.Relational schema database (See appendix M) ... 27
Figure 17. Add/Remove books UI (see appendix K) ... 28
Figure 18Add/Remove Librarians UI (see appendix K) ... 29
Figure 19.Lend Multimedia Item UI (see appendix K) ... 29
Figure 20. Snippet code A, addUserButtonPressed() (see appendix K) ... 30
Figure 21.Snippet code B, addLibraryUser() (see appendix K) ... 31
Figure 22. Snippet code C, addLibraryUser() (see appendix K) ... 31
Figure 23.Snippet code D addLibraryUser() (See appendix K) ... 31
Figure 24. Snipped code addLibraryUser() (See appendix K) ... 32
Figure 25.Snippet code E addLibraryUser() (See appendix K) ... 32
Figure 26.Snippet code F addLibraryUser() (See appendix K) ... 32
Figure 27.Snippet code G addLibraryUSer() (See appendix K) ... 33
Figure 28.Snippet code H prepared statements for adding a library user (See appendix K) ... 33
Figure 29.Singleton LibraryUserDaoImplementation class (see appendix K) ... 34

Figure 30.Singleton getInstance() (see appendix K) 34
Figure 31.SQL to create loan magazine table 38
Figure 32.SQL to create the magazine table 38
Figure 33.Libraries and dependencies Shared Module 39
Figure 34.Dependecies and lIbraries Server Module 39
Figure 35.Dependencies and libraries Client Module 40
Figure 36. Junit Test setUp(See appendix K)... 44
Figure 37. Junit Test case zero (See appendix K) 45
Figure 38. Test code case **z**ero (See appendix K) 45
Figure 39. Test code case **o**ne (See appendix K) 46
Figure 40.Test code case **m**ultiple (See appendix K).. 46
Figure 41.Junit Test case boundaries (See appendix K) 47
Figure 42. Junit Test case boundaries (See appendix K) 47
Figure 43. Junit Test case boundaries (See appendix K) 48
Figure 44. Test code error Title (see appendix K) 48
Figure 45. Test code error Date A (see appendix K).. 49
Figure 46. Test code Error Date B (see appendix K) 49
Figure 47. Test set error label outside the view model (see appendix K) 49
Figure 48. Test no error in the text fields (see appendix K)... 50
Figure 49.Test clear text fields after error (see appendix K).. 50
Figure 50.Test for successful add and reset of the text fields (see appendix K) 51
Figure 51. Test get_date_of the employment specifics_employment_returns_date() (See appendix K).. 51
Figure 52. Test get date of the employment specifics (See appendix K) 52
Figure 53. error_label_cannot_be_set_outside_viewmodel() (see appendix K) 52
Figure 54. Result from errorLabel setting test (see appendix K) 52
Figure 55. Result of errorLabel setting test (see appendix K).. 53
Figure 56. Observer example p1 (see appendix K)……………………………………………..35
Figure 57. Observer example p2 (see appendix K)……………………………………………..35
Figure 58. Observer example p3 (see appendix K)……………………………………………..36
Figure 59. Observer example p4 (see appendix K)……………………………………………..36

Figure 60. Observer example p5 (see appendix K)……………………………………………..36
Figure 61. Observer example p6 (see appendix K)……………………………………………..37

I hereby declare that my project group and I prepared this project report and that all sources of information have been duly acknowledged

Group member's name	Student number	Signature
Rosa Briales Marfil	315250	[illegible]
Franciszek Jan Nurkiewicz	318212	[illegible]
María Ortiz Planchuelo	315266	[illegible]
Alexandru Dulghier	315267	[illegible]
Rafael Sánchez Córdoba	315212	[illegible]

Abstract

The client of this project is a small library in Pakistan that is using a physical registry system to keep track of the transactions.

Therefore, the purpose of this project is to create a digital library system which can ease the management of resources of the library.

Due to the lack of knowledge of the Pakistani language and the complexity of including multiple types of multimedia items, the system only includes books and magazines, and it is developed in English.

The structure used in the elaboration of the system has been an iterative approach following the agile development methodology, SCRUM and Unified Process (UP).

During the iterations the analysis, design, implementation, and testing of the system are elaborated to accomplish the client wishes.

The result of the project is a multiclient-server system with a database persistence layer. Even though the system does not accomplish the purpose of the project to a complete extent, it includes all the functionality needed to fulfil all the critical priorities user stories required by the client.

1 Introduction

People have always needed to tell stories: the adventures of Greek heroes and gods, the life of great kings and queens, or just bedtime stories told to children by their parents. As more and more stories appeared, and people started writing them down, libraries were used to store this knowledge and pass it down to future generations.

Libraries have always played an essential role in society, it's concept dates back millennia. "The first systematically organized library in the ancient Middle East was established in the 7th century BCE by Assyrian ruler Ashurbanipal in Nineveh, in contemporary Iraq. It contained approximately 30,000 cuneiform tablets assembled by topic. Many of the works were archival documents and scholarly texts, but there were also works of literature, including the ancient Epic of Gilgamesh." (Vaughan, s.f.)

The goal of ancient libraries was to collect knowledge and learn from it. Developments in fields such as agriculture, architecture, medicine, art, manufacturing, war, and more were all preserved in these libraries. Access to this knowledge led to people realizing the benefits of having libraries, and they became common in cities all over the world. (Vaughan, s.f.)

As the world evolves, libraries do as well. They have come a long way from cuneiform tablets and modern libraries are a part of a complex system of education, even if interest in them has diminished in current days, they are still relevant. "In a world without libraries, it would be difficult to advance research and human knowledge or preserve the world's cumulative knowledge and heritage for future generations." (White, 2012)

"The 21 Century is called a century of peace and education explosion" (Samad, 2019). Unfortunately, not all population has access to this knowledge, specifically in Pakistan, the institutions libraries are not well managed and are not functioning properly and are failed to create a culture of reading in their society. (Samad, 2019)

The client of this project is a small library in Pakistan that is using a physical registry system to keep track of the transactions. The client is requesting a digital system to manage its current resources and reduce the librarians' workload.

The library needs a way to store and manage all the information about their multimedia items (books, magazines, films…), their library users and employees, as well as the loan period and the fines that come with the delays. It is also needed to track all the transactions.

Nowadays, libraries require some information for the library user and librarians, such as the social security number (the equivalent in Pakistan is the National Identity Card or NIC, formed of unique 13 digits which are recognized all over the country (NADRA, s.f.)), first and last name, date of birth, email and phone number. Regarding the media items, specific information such as an identification number, title, author/director, date of publishing, and genre among others.

Usually, libraries have a library manager who is in charge of hiring and firing librarians, who are responsible for creating the library user's account, adding and removing new multimedia to the library, and loaning it to the library users. Usually, their job also includes extending the reservation for a specific media item when the user requests it and managing the fines.

Moreover, some libraries give their library users the possibility of accessing the information about their borrowed items, giving a review of them, and checking which items are available in the library so they can be reserved.

The purpose of this project is to create a digital library system due to the lack of efficiency in the current physical one so it can ease the management of resources of the library.

The project involves a series of challenges, the most relevant being the library having a bad internet connection and therefore not being able to access the system. The lack of knowledge of the Pakistani language and the complexity of including multiple types of multimedia items have also been considered.

Taking this into account, the system only includes books and magazines, and it is developed in English.

The steps followed from analysis and design to implementation and testing are documented in this report and can be found in the following sections.

2 Analysis

During analysis, the focus is on understanding the problems that the creation of the library system involves, as well as the client's needs. This is done through the elaboration of requirements and diagrams, resulting in a Domain Model (Larman, 2004).

2.1 Requirements

Before enumerating the requirements, it is important to describe the primary actors that interact with the system:

Library manager: is the person who oversees the library, his responsibilities include hiring and firing librarians.

Librarians: they are the main workers of the library, their responsibilities include managing the multimedia item, meaning registering the new ones that arrive at the library in the system and deleting the ones that are no longer available. They are also in charge of registering the users who want to be members of the library and deleting the ones that no longer want to be part of it. Moreover, they are in charge of managing the loans, when a user borrows a multimedia item and when they are returned and processing the fines in case the items are not returned on time. Lastly, they have the responsibility of deleting the reviews the users leave if they are not appropriate.

Library Users: they use the library to loan multimedia items and extend said loans with the librarian's help (library users act as secondary actors in this case), but also to reserve an item or write reviews.

Following the SMART principles (YourCoach n.d.), the following user stories have been elaborated according to the client's desires.

2.2 User Stories

Critical priority:

1. As a librarian, I want to be able to add multimedia items to the system, so library users can borrow them.
2. As a library manager, I want to be able to add librarians to the system, so they can start using it once they are hired.
3. As a librarian, I want to add a library user to the system, so the librarians can lend multimedia items to them.

4. As a librarian, I want to lend a specific item of multimedia to a specific library user for 14 days, so they can borrow it.
21. As a librarian, I want to make the multimedia items available when they are returned so other library users can borrow them again.

High priority:
5. As a library manager, I want to be able to remove librarians from the system, so they cannot access it anymore if they are fired or gone.
6. As a librarian, I want to be able to remove multimedia items from the system, so multimedia items will not belong to the system anymore.
7. Both as a librarian and a library user, I want to be able to search for specific multimedia items, so I can check if the desired item is available in the library.
8. As a library user, I want to see the items I have borrowed and their information, including the remaining time, so I know when I have to return them.
9. As a librarian, I want to see the active fines of the library users when they try to borrow a multimedia item, so I can make sure that no one can borrow any items if they have a fine.
10. As a librarian, I want to be able to process a fine, so it can be removed once the library user has paid.

Low priority:
20. As a librarian, I want to be able to delete a specific library user, so they won't be able to borrow any multimedia item anymore.
12. As a library user, I want to be able to see my previous borrowed multimedia items and previous fines and reviews so I can keep track of them.
13. As a librarian, I want to be able to see library users' previously borrowed multimedia items, previous fines, and reviews so I can keep track of them.
14. As a librarian, I want to be able to extend the loan of a specific item for a specific library user up to 3 times, so they can return the item later than it was supposed to.
15. As a library user, I want to be notified of the amount of money I must pay when the time of one of my borrowings has passed, so I know I must go to the library to pay the fine.
16. As a library user, I want to be notified 24 hours before my borrowing expires so I can return the media item or request an extension.
17. As a library user, I want to be able to reserve a multimedia item for 24 hours, so I can make sure that no one else will borrow it before me.
18. As a library user, I want to be able to give a review of the multimedia items I have rented, so that other users will be able to read them.
19. As a librarian, I want to be able to delete a specific review, so I can make sure there are no inappropriate or offensive reviews.
22. As a library user, I want to be able to delete a specific review that I have previously written, so no one will see it if I made a mistake or changed my opinion.
11. As a library user, if I try to reserve a book that is not available, I want to be notified when it becomes available, so I know I can borrow it then.

2.3 Non-Functional Requirements

a. The system will be a multi client-server system.
b. The system will be implemented in English.
c. All the registered data will be permanently stored in a database unless it is specifically removed by a librarian or manager.
d. The system will include a user guide to document how to use the user interface.
e. The system will include the following information:
    - Books: isbn, title, publisher, author, edition, year of publication and genres (the mandatory information is isbn, title, publisher, year).
    - Magazines: title, publisher, volume, genre, date (day, month and year) (the mandatory information is title, publisher and date).
    - Library user: social security number (ssn), first and last name, and password (all the information is mandatory).
    - Librarian: social security number (ssn), first and last name, date of employment and password (all the information is mandatory).
    - Library manager: social security number (ssn), first and last name and password (all the information is mandatory).
f. The manager will be fixed in the system, meaning that it cannot be modified or deleted.
g. The history of loans for multimedia items will be stored in the database, including the social security number of the user who borrowed it and the dates of start and return of the loan. However, only the loans of the current library users, books and magazines are stored, so when a user, magazine or book is deleted all of their loan history is deleted as well.
h. Regarding the information to be stored, some of them will follow a specific format:
    - Isbn (numbers and "-").
    - Edition, day, month, year, volume (numbers).
    - Social security number (13 digits).
    - Date (yyyy-mm-dd).
    - Title, publisher, author, genre, first name, last name, password (no predetermined format).
i. The system will include a read me file to help the clients to set up the system.

2.4 Uses cases

2.4.1 Use case diagram

The interactions of the 3 actors with the system and the 11 use cases are shown in the following use case diagram (See figure 1). The Manager interacts with “manage librarians”; librarians interact with “manage multimedia item”, “manage library user”, “manage loan”, “search multimedia item”, “see information”, “manages fines” and “delete review”. Lastly, library users interact with “search multimedia item”, “see information”, “write review”, “delete review”, “reserve multimedia item” and “see notification”.
The use case “see notification” is extending see information”, and “reserve a multimedia item” is extending “manage loan”.

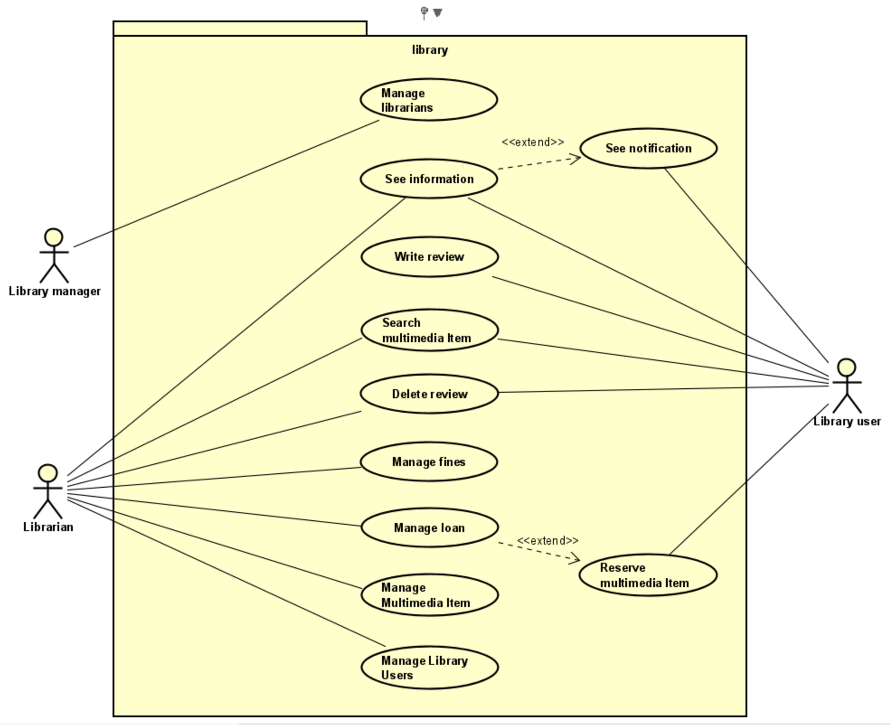


Figure 1. Use case diagram (appendix D)

2.4.2 Use case descriptions

The following use case descriptions explains a set of scenarios that leads to a common user goal. The rest of them can be found in the appendix F.

2.4.2.1 Manage multimedia item

The Use case description below (See table 1) shows the functionality of "Manage multimedia item".
This use case fulfills the user stories 1 and 6.
Below (see tables 2 and 3) are the scenarios which complement this use case: "manage books" and "manage magazines".

Use case	**Manage multimedia item**
Summary	A librarian has a multimedia item that needs to be added or removed from the system. The librarian sees a list of the multimedia items that are currently stored in the system. In case they want to add it, they will use the system to record all the information about it. The system validates and records the data. On the other hand, if they want to delete it, they will have to select the item and use the system to delete it. The multimedia item will either appear or disappear from the list.
Actor	Librarian
Precondition	The librarian is logged in the system and has chosen to manage multimedia items. In case of removing a multimedia item, said item must be stored in the system.
Postcondition	The item and all its information will be successfully added/removed from the system.
Base sequence	1. System asks to choose the type of item 2. Choose type • If chosen type is 'book' go to manage books scenario (See table 2) • If chosen type is 'magazine' go to manage magazines scenario (See table 3)
Notes	This use case fulfills the user stories 1 and 6.

Table 1. Manage Multimedia Item (See appendix F)

Scenario	**Manage books**
Base sequence	CASE ADD: 1. System shows a list of the stored books and a list of genres to select from and ask for the following information:

	Isbn, title, publisher, author, edition, year of publication and genres (the mandatory information is isbn, title, publisher, year). 2. Librarian inserts all the information. 3. About genres (OPTIONAL): CASE Add genre: 3.1 The librarian selects a genre from the list of genres and chose to add. (ES 2) 3.2 The system updates the list of added genres. CASE Remove genre: 3.1 The librarian selects a genre from the list of added genres and chooses to remove. 3.2 The system updated the list of added genres. 4. Librarian chooses to add the book. (ES 1) 5. System shows the updated list. CASE REMOVE: 1. System displays a list of the stored books. 2. OPTIONAL: 2.1. Filter to find the desired book. 2.2. System displays a list of the books that fulfill the filters. 2.3. Possibility to change the filters, go to case delete step 2.1 3. Select the book from the list and delete it. 4. The system displays the list with the remaining books.
Exception sequence	ES 1: the information is incomplete or against the system restrictions: 4a. System displays an error. Go back to main flow, case add step 1. ES 2: the genre is already on the list of genres added. 3.3a System displays an error. Go back to main flow, case add genre step 3
Notes	Regarding ES 1, the error that the system might display are the following: • Title/Publisher/Isbn cannot be empty. • Invalid date: future date (year of publication cannot be a future date). • There is already a book with that isbn in the system (isbn must be unique) • Duplicate genre (book cannot have the same genre twice). • Edition must be a natural number. Optional step 3 can be done multiple times.

Table 2. Manage Books Scenario (See appendix F)

Scenario	**Manage magazines**

Base sequence	CASE ADD: 1. System displays a list of the stored magazines and asks for the following information: Title, publisher, volume, genre, date (day, month and year) (the mandatory information is title, publisher and date) 2. Librarian inserts all the information and adds the item. (ES 1) 3. System records the information and shows the updated list. CASE REMOVE: 1. System displays a list of the stored magazines 2. OPTIONAL: 2.1. Filter to find the desired item 2.2. System displays a list of the items that fulfill the filters. 2.3. Possibility to change the filters, go to case delete step 2.1 3. Select the item from the list and delete it. 4. The system displays the list with the remaining items.
Exception sequence	ES 1: the information is incomplete or against the system restrictions: 2a. System displays an error. Go back to main flow, case add step 1.
Notes	Regarding ES 1, the errors that the system might display are the following: - Invalid date: future date (date cannot be a future date). - Title/Publisher/Day/Month/Year cannot be empty. - Invalid date (day, month and year create a date that does not exist. Ex. 2022-02-31, 2000-13-3). - Volume must be a natural number.

Table 3. Manage Magazines scenario (See appendix F)

2.4.2.2 Manage library user

The Use case description below (See table 4) shows the functionality of “Manage library user”. This use case fulfills the user stories 3 and 20.

Use case	**Manage library User**
Summary	A library user comes into the library and ask the librarian to be register in the library. In this case the librarian will add a new library user, they will record all the information about it: social security number (ssn), first and last name, and their password (see non-functional requirement e.). The system validates and records the data.

	On the opposite, a library user comes and asks to be no longer a user in the library. The librarian will select the library-user and use the system to delete it. The library user will either appear or disappear from the list that it is always shown.
Actor	Librarian
Precondition	The librarian is logged in to the system and has chosen to manage library users. In case of removing a library-user, it must be stored in the system.
Postcondition	The library user is added/removed from the system.
Base sequence	1. System displays a list of all the library users stored in the system CASE ADD: 2. The system asks for the following information: social security number (ssn), first and last name and their password 3. Insert library-user information and add it. (ES 1) 4. System shows the updated list. CASE REMOVE: 3. Select the librarian from the list and delete it. 4. The system displays the list with the remaining librarians.
Exception sequence	ES 1: the information is incomplete or against the system restrictions: 3a. System will display an error. Go back to main flow, case add step 2.
Notes	This use case fulfills the user stories 3,20 Regarding ES 1, the error that the system might display are the following: - First name/ Last name/ Ssn/ password cannot be empty. - The ssn must be 13 digits - There is already a library user with that ssn in the system (ssn must be unique).

Table 4. Manage library user Table (See appendix F)

2.4.3 Activity and System Sequence diagrams

To get a better understanding of the actors actions and system reaction, additional diagrams have been elaborated.

The following system sequence diagram shows the interaction between the manager and the system in the two sunny scenarios in managing librarians (add/remove). (See figure 2)

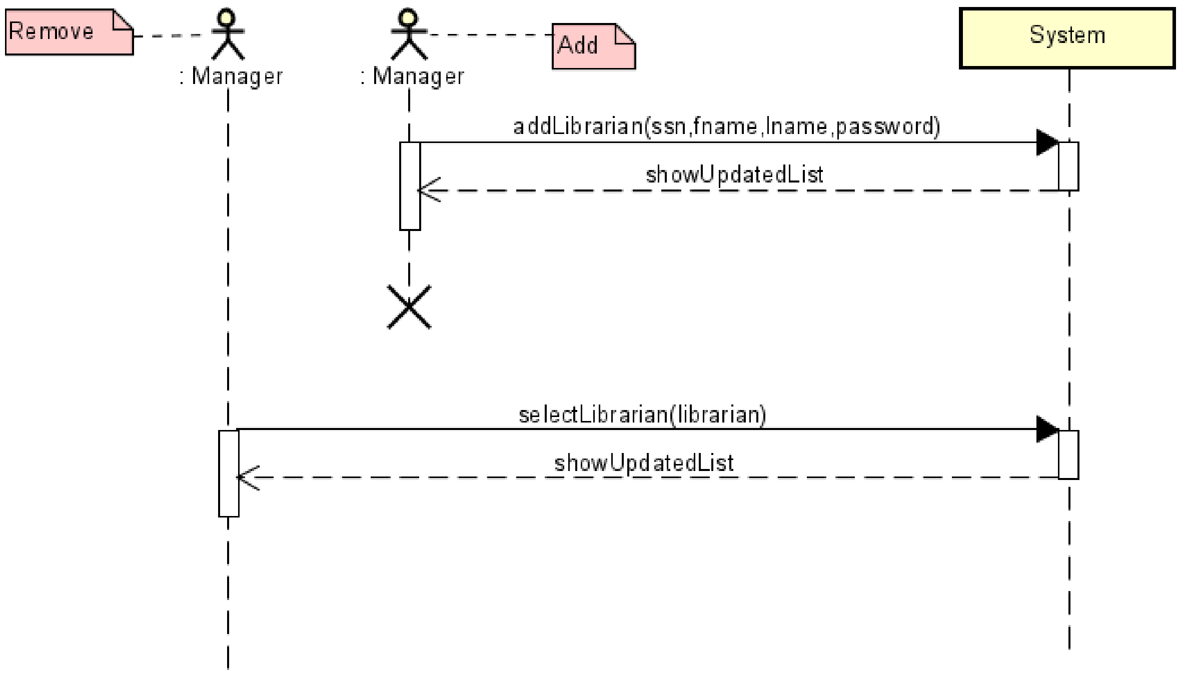

Remove
: Manager
: Manager
Add
System
addLibrarian(ssn,fname,lname,password)
showUpdatedList
selectLibrarian(librarian)
showUpdatedList

Figure 2. Add librarian system sequence diagram (See appendix H)

The following system sequence diagram shows the interaction between the librarian and the system in the sunny scenario in end loan. (See figure 3).

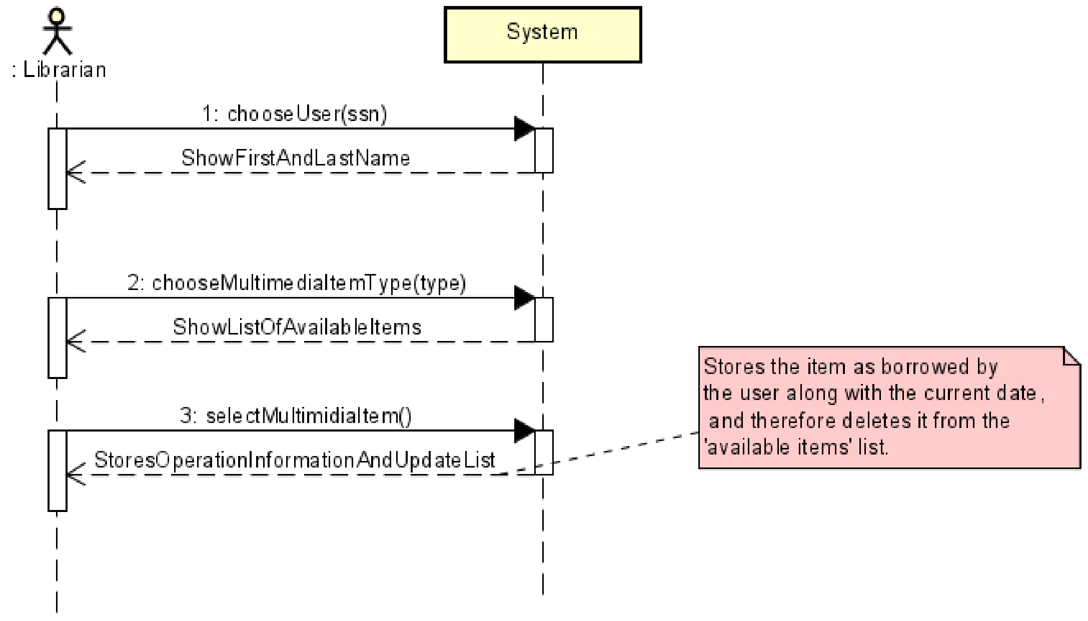

: Librarian
System
1: chooseUser(ssn)
ShowFirstAndLastName
2: chooseMultimedialtemType(type)
ShowListOfAvailableItems
3: selectMultimidialtem()
StoresOperationInformationAndUpdateList
Stores the item as borrowed by the user along with the current date, and therefore deletes it from the 'available items' list.

Figure 3.Loan item system sequence diagram (See appendix H)

The following activity diagram correspond to the scenario log in and can also be found in appendix H.

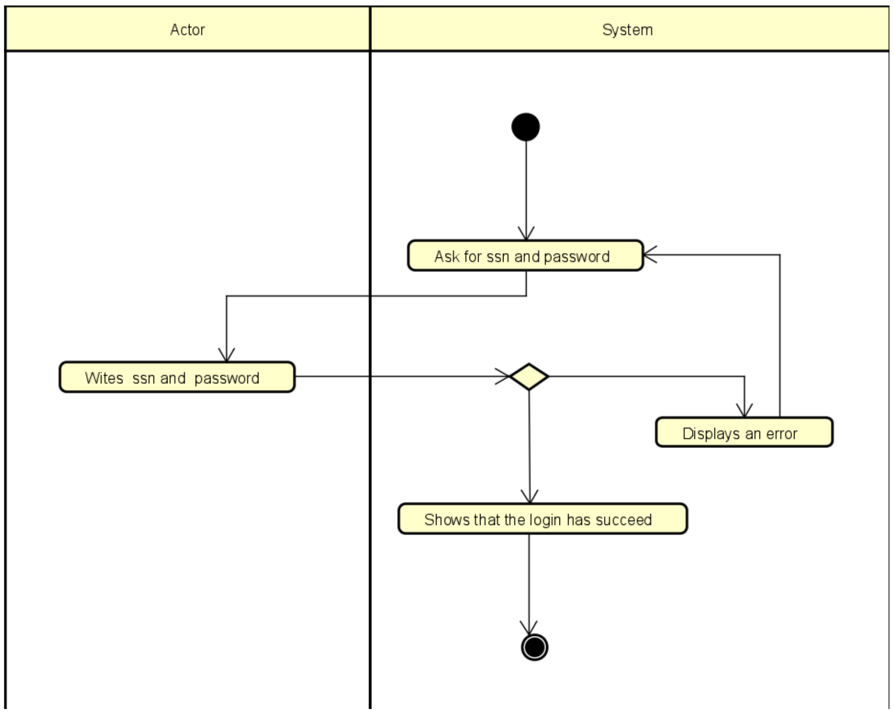


Figure 4. Log in activity diagram (See appendix H)

2.5 Domain model

The analysis concludes with the elaboration of the domain model below (see figure 5) which shows a specific overview of the problem data and its relationships.

The domain model is formed by 8 entities: library manager, librarian, fine, library user, review, multimedia item, book and magazine.

Only the entities which have been analysed thoroughly are represented which their respective attributes.

The library manager either adds or removes librarians, who can themselves add and remove library users, remove fines and add, remove or search for multimedia items. Multimedia item is either magazine or book and has between 0 to many reviews that are written by the library user and can be deleted by both librarians and library users.

Library users can loan a multimedia item or reserve them. Both loan and reserve are association classes and store the start date and, in case of the loan, the end date.

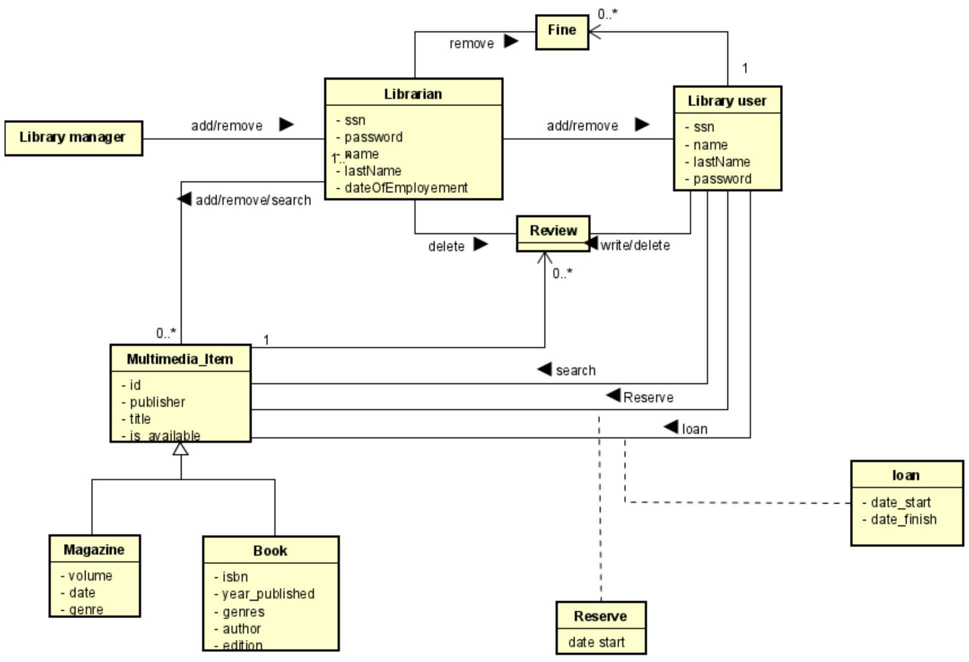


Figure 5.Domain model (appendix ??)

3 Design

This section outlines how the system is structure following a series of design patterns and principles described below.

The system follows a layered design with a clear separation between the logical layers and it is a client/server system which follows RMI.

As stated in the non-functional requirement c, the persistence layer consist of a database, which also is design through diagrams that can be found below in this section or in appendixes L,M and N.

3.1 Patterns and principles

3.1.1 MVVM

One of the architecture patterns followed is MVVM, which provides a separation of concerns. This pattern not only makes the system easier to maintain and update, but also makes unit testing much simple, being able to test the different layers independently. It can be seen clearly in the class diagram (see appendix G). The figures below show an example of this pattern. The class `AddRemoveLibrarianController` has an association to the `AddRemoveLibrarianViewModel` (see figure 6), through data binding and commands, and is the `ViewModel` the one connected to the model (see figure 7) and gets the information from it to send it to the view in a way that it can use it. This way, the controller is isolated from the model itself, but the functionality of the model can be seen in the user interface as well. (Microsoft, 2012)

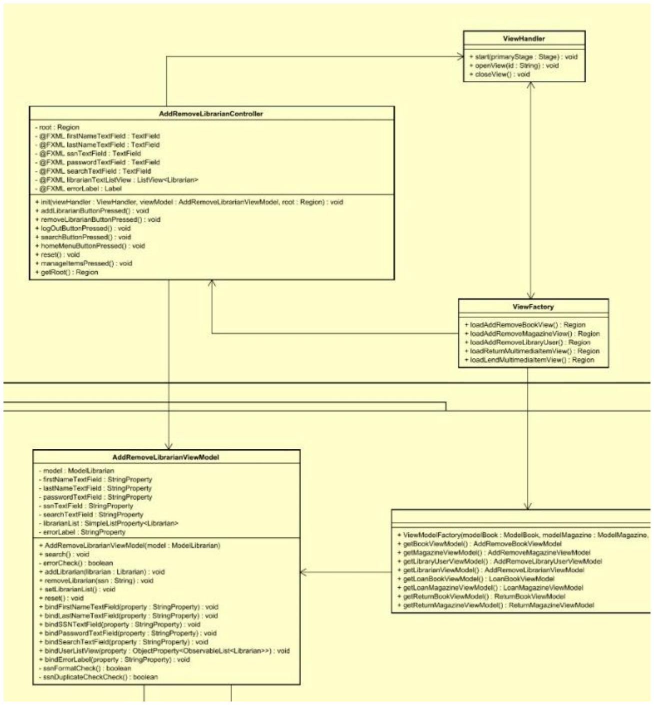

ViewHandler
+ start(primaryStage : Stage) : void
+ openView(id : String) : void
+ closeView() : void
AddRemoveLibrarianController
- root : Region
- @FXML firstNameTextField : TextField
- @FXML lastNameTextField : TextField
- @FXML ssnTextField : TextField
- @FXML passwordTextField : TextField
- @FXML searchTextField : TextField
- @FXML librarianTextListView : ListView<Librarian>
- @FXML errorLabel : Label
+ init(viewHandler : ViewHandler, viewModel : AddRemoveLibrarianViewModel, root : Region) : void
+ addLibrarianButtonPressed() : void
+ removeLibrarianButtonPressed() : void
+ logOutButtonPressed() : void
+ searchButtonPressed() : void
+ homeMenuButtonPressed() : void
+ reset() : void
+ manageItemsPressed() : void
+ getRoot() : Region
ViewFactory
+ loadAddRemoveBookView() : Region
+ loadAddRemoveMagazineView() : Region
+ loadAddRemoveLibraryUser() : Region
+ loadReturnMultimediaItemView() : Region
+ loadLendMultimediaItemView() : Region
AddRemoveLibrarianViewModel
- model : ModelLibrarian
- firstNameTextField : StringProperty
- lastNameTextField : StringProperty
- passwordTextField : StringProperty
- ssnTextField : StringProperty
- searchTextField : StringProperty
- librarianList : SimpleListProperty<Librarian>
- errorLabel : StringProperty
+ AddRemoveLibrarianViewModel(model : ModelLibrarian)
+ search() : void
- errorCheck() : boolean
+ addLibrarian(librarian : Librarian) : void
+ removeLibrarian(ssn : String) : void
+ setLibrarianList() : void
+ reset() : void
+ bindFirstNameTextField(property : StringProperty) : void
+ bindLastNameTextField(property : StringProperty) : void
+ bindSSNTextField(property : StringProperty) : void
+ bindPasswordTextField(property : StringProperty) : void
+ bindSearchTextField(property : StringProperty) : void
+ bindUserListView(property : ObjectProperty<ObservableList<Librarian>>) : void
+ bindErrorLabel(property : StringProperty) : void
- ssnFormatCheck() : boolean
- ssnDuplicateCheckCheck() : boolean
+ ViewModelFactory(modelBook : ModelBook, modelMagazine : ModelMagazine,
+ getBookViewModel() : AddRemoveBookViewModel
+ getMagazineViewModel() : AddRemoveMagazineViewModel
+ getLibraryUserViewModel() : AddRemoveLibraryUserViewModel
+ getLibrarianViewModel() : AddRemoveLibrarianViewModel
+ getLoanBookViewModel() : LoanBookViewModel
+ getLoanMagazineViewModel() : LoanMagazineViewModel
+ getReturnBookViewModel() : ReturnBookViewModel
+ getReturnMagazineViewModel() : ReturnMagazineViewModel

Figure 6. MVVM pattern class diagram (See appendix G)

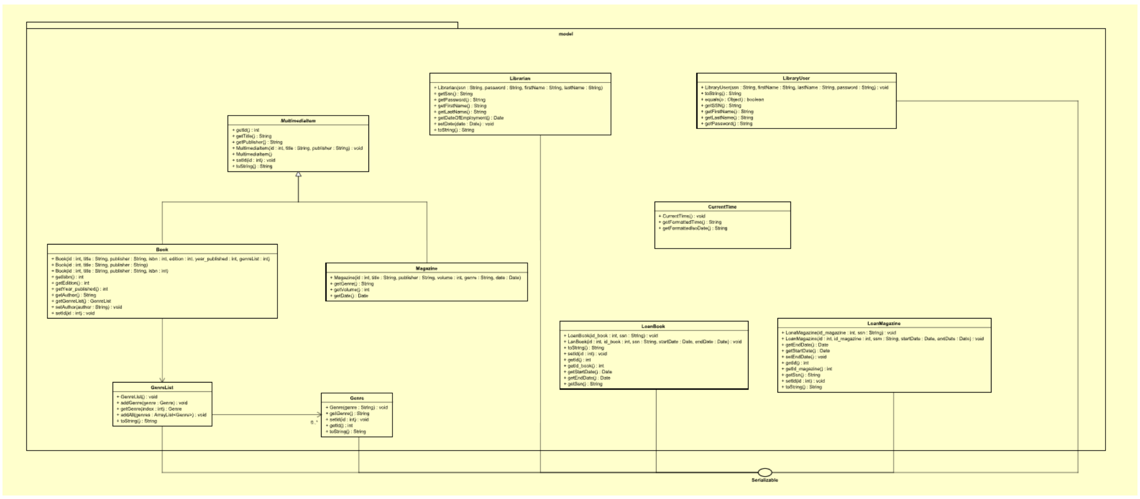


Figure 7. Model class Diagram (See appendix G)

3.1.2 Observer

As mentioned before, following the MVVM pattern the View model interacts with the model by invoking methods in the model classes. Observer pattern, as shown in the figure below (see figure 8), makes the view model and model participate in two-way data binding direction by firing `PropertyChange` events.

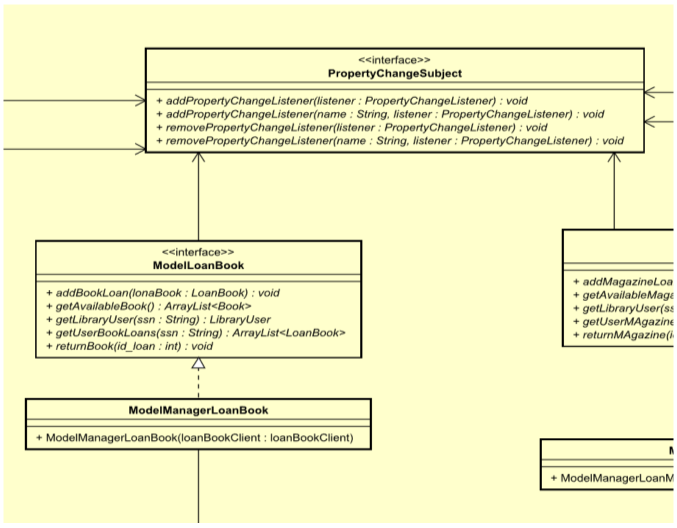

<<interface>>
PropertyChangeSubject
+ addPropertyChangeListener(listener : PropertyChangeListener) : void
+ addPropertyChangeListener(name : String, listener : PropertyChangeListener) : void
+ removePropertyChangeListener(listener : PropertyChangeListener) : void
+ removePropertyChangeListener(name : String, listener : PropertyChangeListener) : void
<<interface>>
ModelLoanBook
+ addBookLoan(lonaBook : LoanBook) : void
+ getAvailableBook() : ArrayList<Book>
+ getLibraryUser(ssn : String) : LibraryUser
+ getUserBookLoans(ssn : String) : ArrayList<LoanBook>
+ returnBook(id_loan : int) : void
ModelManagerLoanBook
+ ModelManagerLoanBook(loanBookClient : loanBookClient)
+ addMagazineLoa
+ getAvailableMaga
+ getLibraryUser(s
+ getUserMAgazine
+ returnMAgazine(i
+ ModelManagerLoanM

Figure 8. Property change subject pattern class diagram (See appendix G)

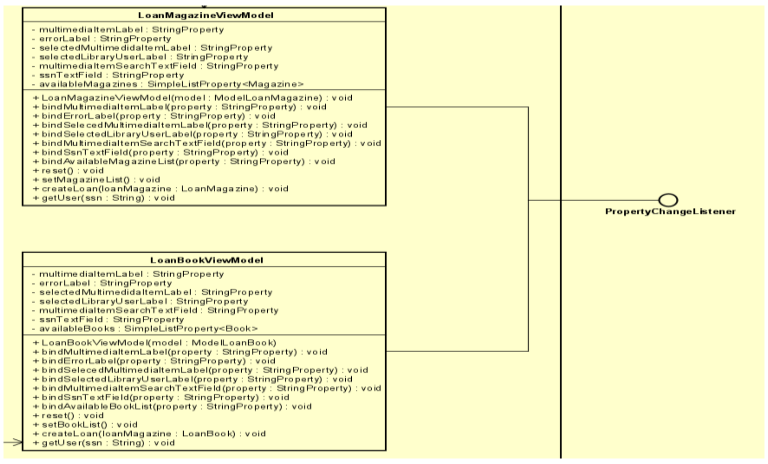

LoanMagazineViewModel
- multimediaItemLabel : StringProperty
- errorLabel : StringProperty
- selectedMultimedidaItemLabel : StringProperty
- selectedLibraryUserLabel : StringProperty
- multimediaItemSearchTextField : StringProperty
- ssnTextField : StringProperty
- availableMagazines : SimpleListProperty<Magazine>
+ LoanMagazineViewModel(model : ModelLoanMagazine) : void
+ bindMultimediaItemLabel(property : StringProperty) : void
+ bindErrorLabel(property : StringProperty) : void
+ bindSelecedMultimediaItemLabel(property : StringProperty) : void
+ bindSelectedLibraryUserLabel(property : StringProperty) : void
+ bindMultimediaItemSearchTextField(property : StringProperty) : void
+ bindSsnTextField(property : StringProperty) : void
+ bindAvailableMagazineList(property : StringProperty) : void
+ reset() : void
+ setMagazineList() : void
+ createLoan(loanMagazine : LoanMagazine) : void
+ getUser(ssn : String) : void
PropertyChangeListener
LoanBookViewModel
- multimediaItemLabel : StringProperty
- errorLabel : StringProperty
- selectedMultimedidaItemLabel : StringProperty
- selectedLibraryUserLabel : StringProperty
- multimediaItemSearchTextField : StringProperty
- ssnTextField : StringProperty
- availableBooks : SimpleListProperty<Book>
+ LoanBookViewModel(model : ModelLoanBook)
+ bindMultimediaItemLabel(property : StringProperty) : void
+ bindErrorLabel(property : StringProperty) : void
+ bindSelecedMultimediaItemLabel(property : StringProperty) : void
+ bindSelectedLibraryUserLabel(property : StringProperty) : void
+ bindMultimediaItemSearchTextField(property : StringProperty) : void
+ bindSsnTextField(property : StringProperty) : void
+ bindAvailableBookList(property : StringProperty) : void
+ reset() : void
+ setBookList() : void
+ createLoan(loanMagazine : LoanBook) : void
+ getUser(ssn : String) : void

Figure 9. Property change listener pattern class diagram (See appendix G)

3.1.3 Singleton

This creational pattern has been used to ensure that only one instance of the DAO implementations classes is created. As an example, the `LoanDAOImplementation` includes a instance variable of itself, a private constructor to make sure is not called from outside of the class and a static method `getInstance()` which is the only way of getting the object from the outside. This way, it is ensured that the different parts of the program share a single database.

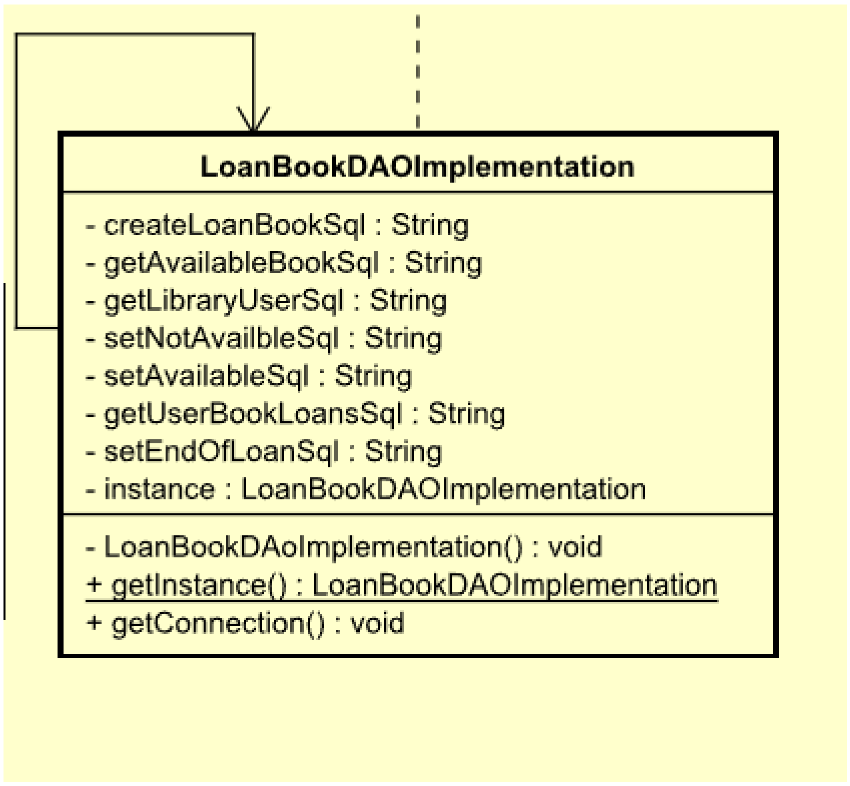
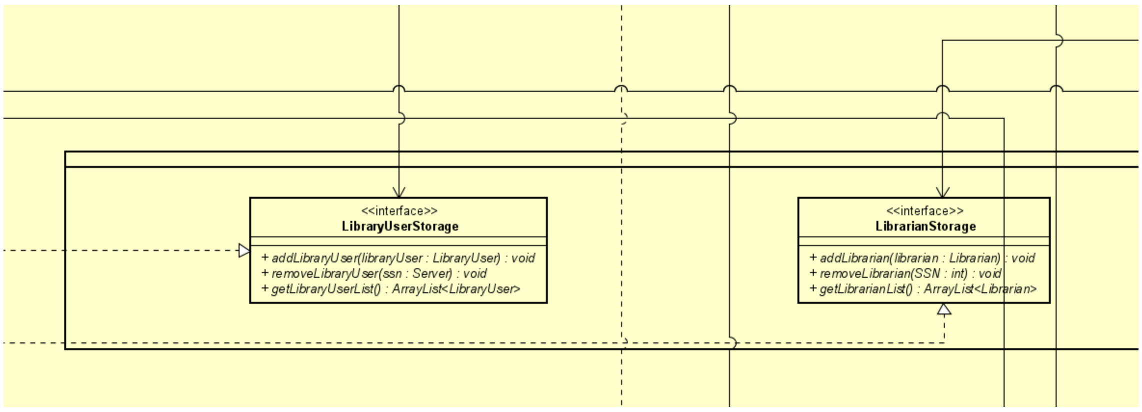

LoanBookDAOImplementation
- createLoanBookSql : String
- getAvailableBookSql : String
- getLibraryUserSql : String
- setNotAvailbleSql : String
- setAvailableSql : String
- getUserBookLoansSql : String
- setEndOfLoanSql : String
- instance : LoanBookDAOImplementation
- LoanBookDAoImplementation() : void
+ getInstance() : LoanBookDAOImplementation
+ getConnection() : void

Figure 10. Singleton pattern of the DAO for Loan Books (see appendix G)

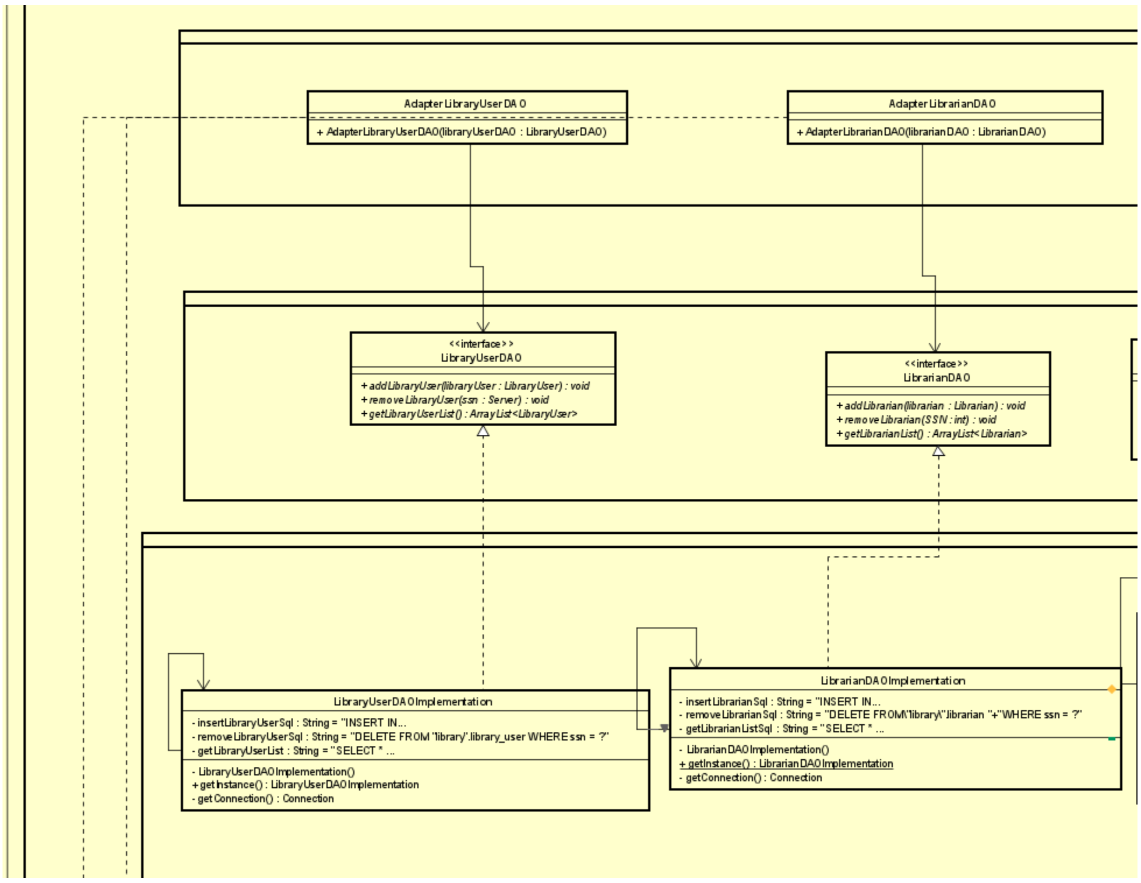
3.1.4 Adapter

A specific storage interface for each class (see figure 11) has been designed and with the use of the adapter pattern (see figure 12) it is made sure that the Single Responsibility Principle (SRP) and the Open/Close Principle (OCP) are followed. New types of adapters for different types of storage can be introduce without breaking the existing code as long as they work through the storage interface. This design allows the separation of the DAO storage from the rest of the system while testing.



<<interface>>
LibraryUserStorage
+ addLibraryUser(libraryUser : LibraryUser) : void
+ removeLibraryUser(ssn : Server) : void
+ getLibraryUserList() : ArrayList<LibraryUser>
<<interface>>
LibrarianStorage
+ addLibrarian(librarian : Librarian) : void
+ removeLibrarian(SSN : int) : void
+ getLibrarianList() : ArrayList<Librarian>

Figure 11. Interface Storage class diagram (see appendix G)



AdapterLibraryUserDAO
+ AdapterLibraryUserDAO(libraryUserDAO : LibraryUserDAO)
AdapterLibrarianDAO
+ AdapterLibrarianDAO(librarianDAO : LibrarianDAO)
<<interface>>
LibraryUserDAO
+ addLibraryUser(libraryUser : LibraryUser) : void
+ removeLibraryUser(ssn : Server) : void
+ getLibraryUserList() : ArrayList<LibraryUser>
<<interface>>
LibrarianDAO
+ addLibrarian(librarian : Librarian) : void
+ removeLibrarian(SSN : int) : void
+ getLibrarianList() : ArrayList<Librarian>
LibraryUserDAOImplementation
- insertLibraryUserSql : String = "INSERT IN...
- removeLibraryUserSql : String = "DELETE FROM 'library'.library_user WHERE ssn = ?"
- getLibraryUserList : String = "SELECT * ...
- LibraryUserDAOImplementation()
+ getInstance() : LibraryUserDAOImplementation
- getConnection() : Connection
LibrarianDAOImplementation
- insertLibrarianSql : String = "INSERT IN...
- removeLibrarianSql : String = "DELETE FROM\"library\".librarian "+"WHERE ssn = ?"
- getLibrarianListSql : String = "SELECT * ...
- LibrarianDAOImplementation()
+ getInstance() : LibrarianDAOImplementation
- getConnection() : Connection

Figure 12. Adapter pattern storage class diagram (see appendix G)

3.1.5 Liskov Subbstitution Principle (LSP)

The “L” in the SOLID principles. It states that “Subclasses should be substitutable for their base classes” (Martin, 2000). In this project this can be seen with the `MultimediaItem` abstract class, which is extended by `Book` and `Magazine` (See figure 13).

However, by including both `publisher` and `title` in the `MultimediaItem` class, this design does not follow the Open-Close Principle, given that adding a different media type (for example CDs or DVDs) would require to change it. This is due to the fact that the system is not design considering the future, and therefore adding more multimedia item types i not taken into consideration.

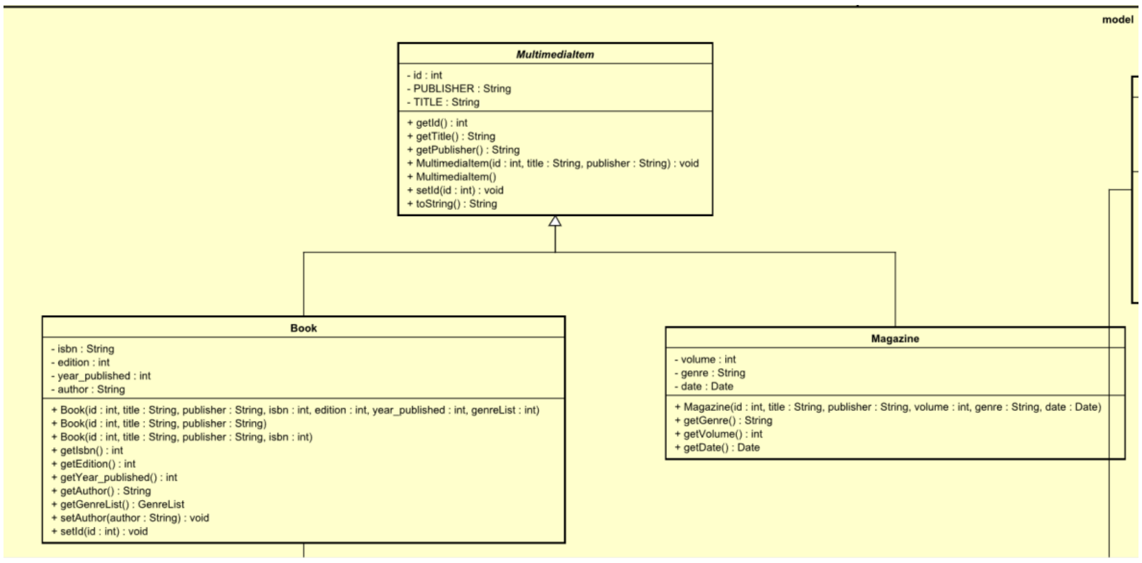

model
MultimediaItem
- id : int
- PUBLISHER : String
- TITLE : String
+ getId() : int
+ getTitle() : String
+ getPublisher() : String
+ MultimediaItem(id : int, title : String, publisher : String) : void
+ MultimediaItem()
+ setId(id : int) : void
+ toString() : String
Book
- isbn : String
- edition : int
- year_published : int
- author : String
+ Book(id : int, title : String, publisher : String, isbn : int, edition : int, year_published : int, genreList : int)
+ Book(id : int, title : String, publisher : String)
+ Book(id : int, title : String, publisher : String, isbn : int)
+ getIsbn() : int
+ getEdition() : int
+ getYear_published() : int
+ getAuthor() : String
+ getGenreList() : GenreList
+ setAuthor(author : String) : void
+ setId(id : int) : void
Magazine
- volume : int
- genre : String
- date : Date
+ Magazine(id : int, title : String, publisher : String, volume : int, genre : String, date : Date)
+ getGenre() : String
+ getVolume() : int
+ getDate() : Date

Figure 13. Multimedia Item in class diagram (see appendix G)

3.1.6 Dependency Inversion Principle (DIP)

As shown in the figure below (see figure 14) the classes in the `mediator package` (higher level) do not depend on the classes in the `client package` (lower level) but in its abstractions.

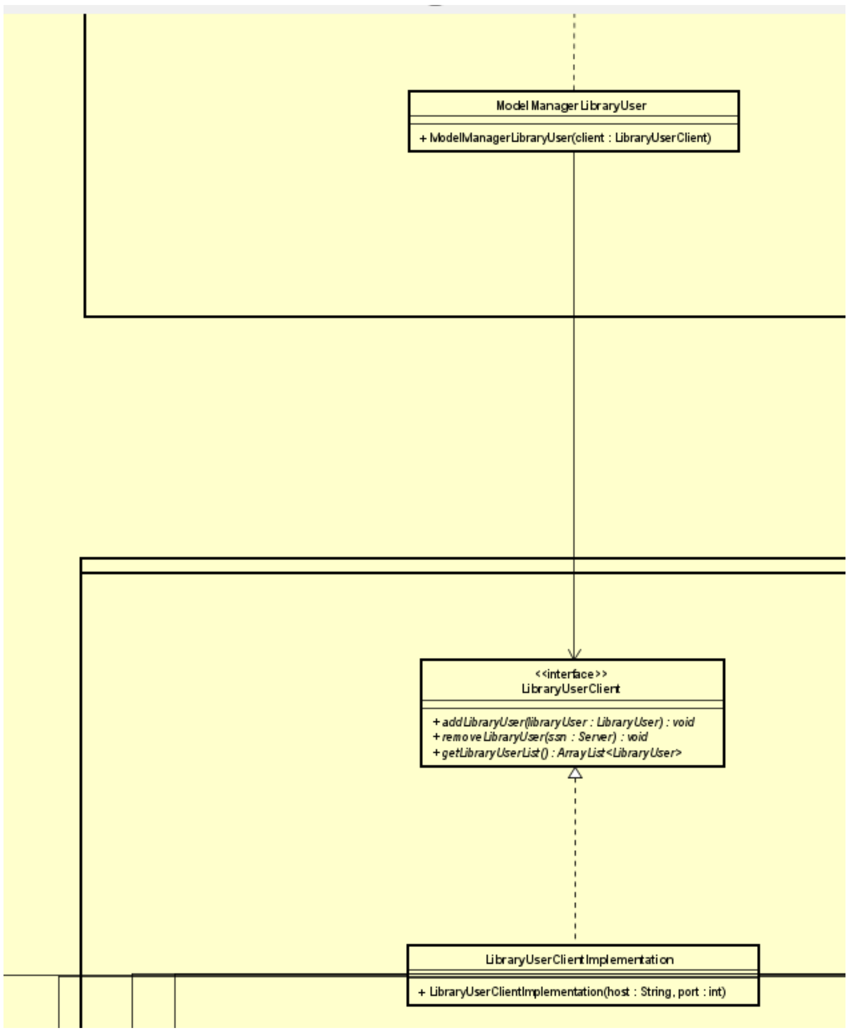

Model Manager LibraryUser
+ ModelManagerLibraryUser(client : LibraryUserClient)
<<interface>>
LibraryUserClient
+ addLibraryUser(libraryUser : LibraryUser) : void
+ removeLibraryUser(ssn : Server) : void
+ getLibraryUserList() : ArrayList<LibraryUser>
LibraryUserClientImplementation
+ LibraryUserClientImplementation(host : String, port : int)

Figure 14. Example DIP class diagram (see appendix G)

3.1.7 Interface-Segregation Principle (ISP)

The class `LibraryUserClientImplementation` just depends on `LibraryUserClient` interface and `LibrarianClientImplementation` just depends on `LibrarianClient` interface (see figure 14). Therefore, this design makes each of them depend only on things they need, following the Interface-Segregation Principle.

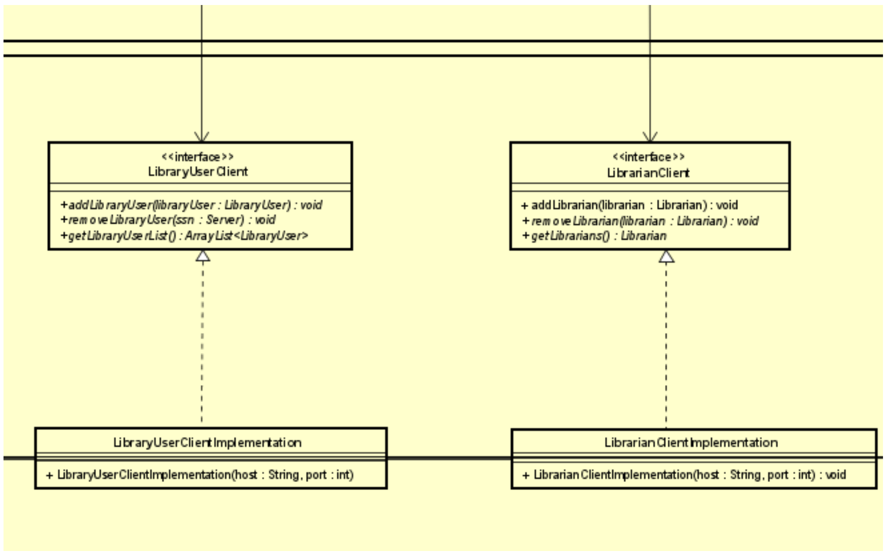

<<interface>>
LibraryUserClient
+addLibraryUser(libraryUser : LibraryUser) : void
+removeLibraryUser(ssn : Server) : void
+getLibraryUserList() : ArrayList<LibraryUser>
<<interface>>
LibrarianClient
+ addLibrarian(librarian : Librarian) : void
+ removeLibrarian(librarian : Librarian) : void
+ getLibrarians() : Librarian
LibraryUserClientImplementation
+ LibraryUserClientImplementation(host : String, port : int)
LibrarianClientImplementation
+ LibrarianClientImplementation(host : String, port : int) : void

Figure 14. Example ISP class diagram (See appendix G)

3.2 Class diagram

The previous patterns, principles and all of the analysis converge in the elaboration of a class diagram (see appendix G), which was designed based on the domain model and the Use Cases, considering all the requirements that the system must fulfill.

The class diagram is formed by three main packages: `client`, `server` and `shared`.

The client package includes four subpackages: `client`, `mediator` (with the models and model managers), `view` and `viewModel`.

The `server` package includes two subpackages: `server` (with another subpackage storage and the `Communicator` class) and `persistence`, which has at the same time three subpackages on its own: `adapters`, `DAO` and `DAOImplementation`.

The `shared` package includes two subpackages: `server`, which includes all the remote interfaces shared by the `client` and `server` (implemented by the `Communicator` class in the `server` package and as instances variables in the client implementations in the `client` package) and `model`.

3.3 Design Persistence

The database design is based on an EER diagram, a relational schema, and a Global Relations Diagram.
First, conceptually modelling the information from the client results in the following EER diagram (see figure 15). One librarian adds multiple `Library_Users` and multiple `Multimedia_Items`, that must either be a `Book` or a `Magazine`.
Moreover, one `Library_User` loan many `Multimedia_Item` and a `Multimedia_Item` can be loaned by many `Library_User`. There are two attributes in this relationship, so it is being tracked the start of the loan and the end of the loan. (This design fulfils non- functional requirement d).
To conclude, every attribute that is designed to be stored in each entity is shown in figure 15, where it is important to realize that `book` has a composite attribute `genre`. Focusing attention on the attributes that unique identify each entity, it is important to mention the `ssn` as a candidate key in `Librarian` and `Library_User` and the `isbn` in `Book.` It is already being decided that the PK, the key that identify the concurrence `Multimedia_Item,` is an `id`.

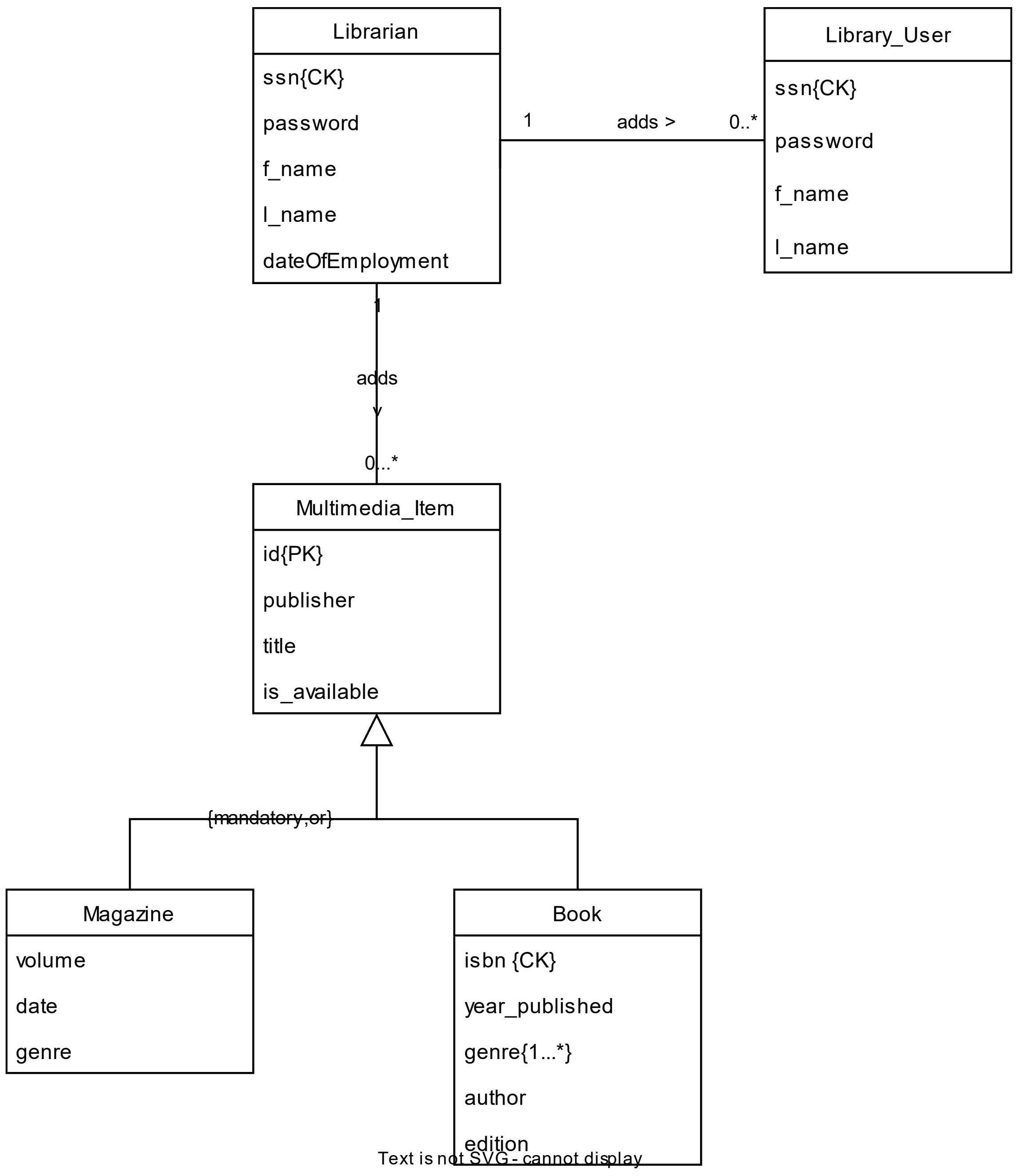


Figure 15.EER database diagram(see appendix N)

Following the mapping steps to assure that the 3rd normal form is respected (Connolly & Carolyn, 2014) a relation schema has been elaborated (see appendix M) and based on it, the global relation diagram below (see figure 16). One to many relationships have become a reference in the child relation as a foreign key: `Librarian` in `Magazine`, `Book` and `Library_User`. `Loan_Book` and `loan_Magazine` is the result of a join-relation. The attributes of the relationship , as well as the primary keys of each relation as foreign keys have been included.
Furthermore, the multivalued attribute `genres` in `Book` results in two more relations so it keeps track of each value and a reference to the entity occurrence. Typically, it is done with just an extra relation but it is designed this way so the genres are identified by an unique id.
To conclude, the attributes decided to uniquely identify each table are the `ssn` in case of `Librarian` and `Library_User`, a composite primary key for the table `Book_Genre`, and for the rest of the relations an id as surrogate key.

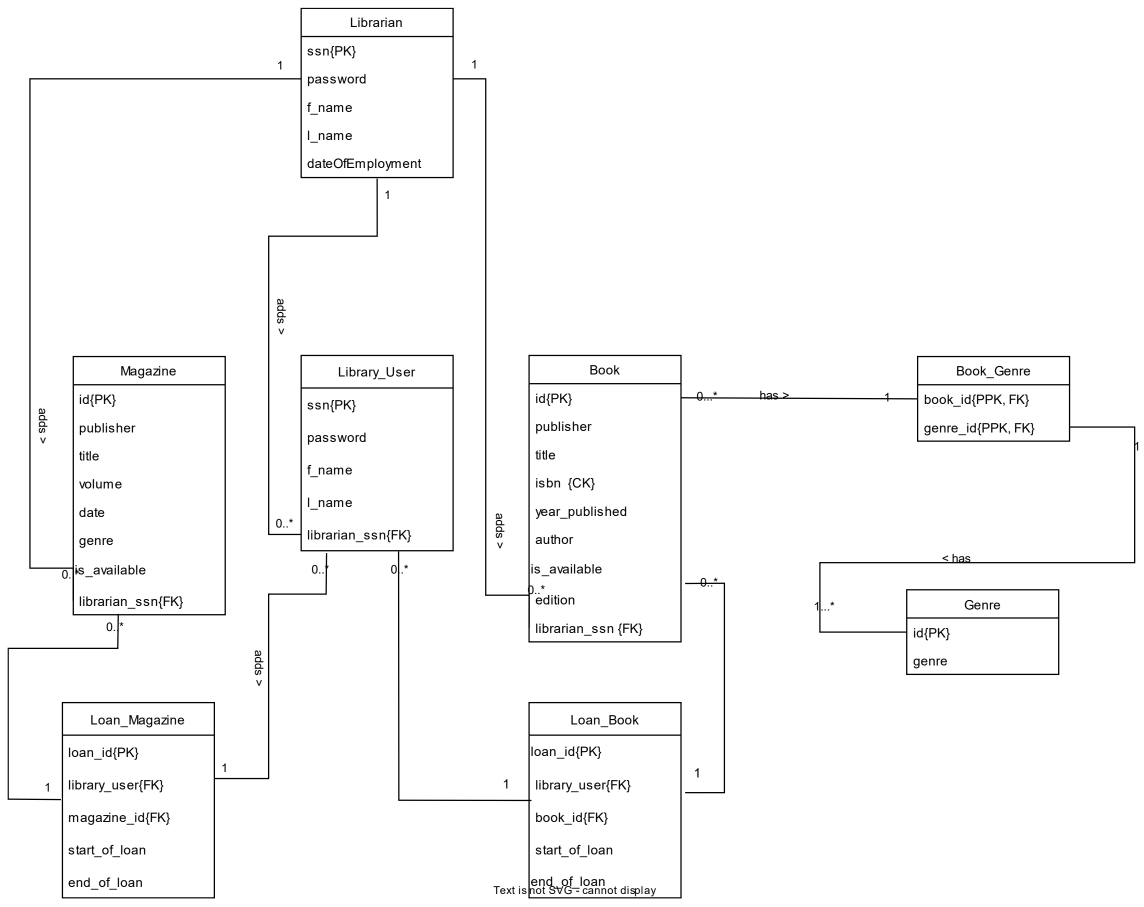


Figure 16.Relational schema database (See appendix M)

3.4 Technologies

Regarding the technologies used in this project:

Java 17 and SQL are the programming languages for the project.

The java implementation is done using IntelliJ IDEA. ‘IntelliJ IDEA is an intelligent, context-aware IDE for working with Java and other JVM languages like Kotlin, Scala, and Groovy on all sorts of applications.’ (JetBrains, 2022)

On the other hand, the SQL implementation is done using DataGrip. 'DataGrip is a database management environment for developers. It is designed to query, create, and manage databases. Supports MySQL, PostgreSQL, Microsoft SQL Server...' (JetBrains, 2022)

In terms of design, the diagrams are created using Astah (UML modeling tool), the graphical user interface (GUI) is created using Scene Builder as the main tool, and the design of the database is created using Diagrams.net, an online diagram editor.

In order to connect the system with the database, a PostgresSQL database is implemented, in this case ElephantSQL. 'ElephantSQL installs and manages PostgreSQL databases for you. ElephantSQL offers databases ranging from shared servers for smaller projects and proof of concepts, up to enterprise-grade multi-server setups.' (ElephantSQL, n.d.)

3.5 Graphical User Interface (GUI) design

Along with the system, a graphical user interface (GUI) had been designed. The graphical user interface (GUI) is designed mainly using SceneBuilder. The following figures show the design.

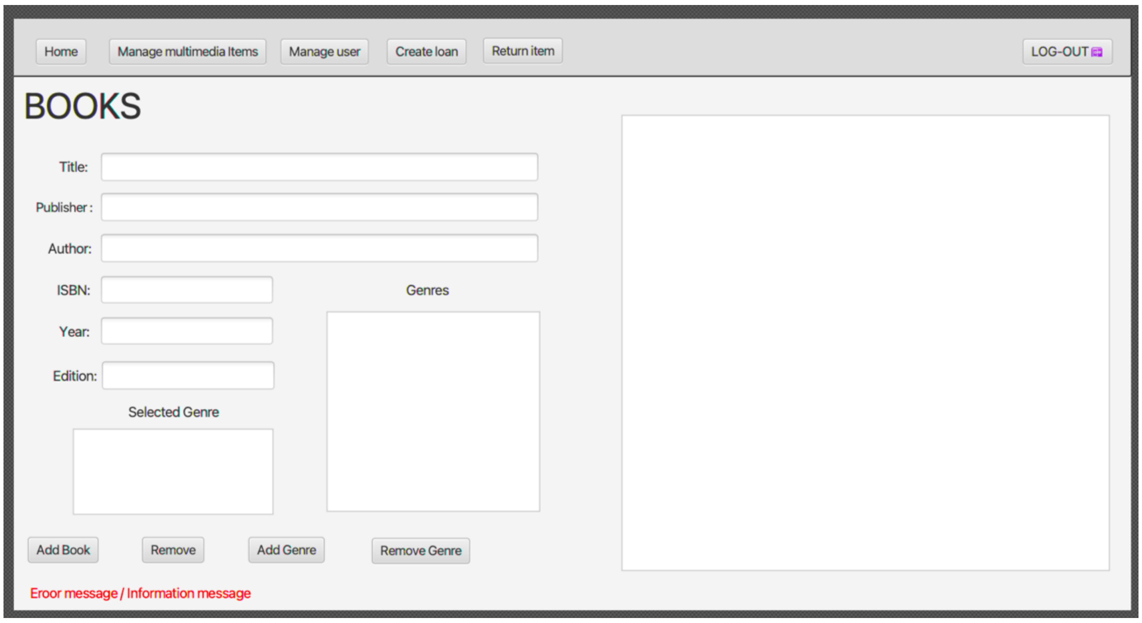


Figure 17. Add/Remove books UI (see appendix K)

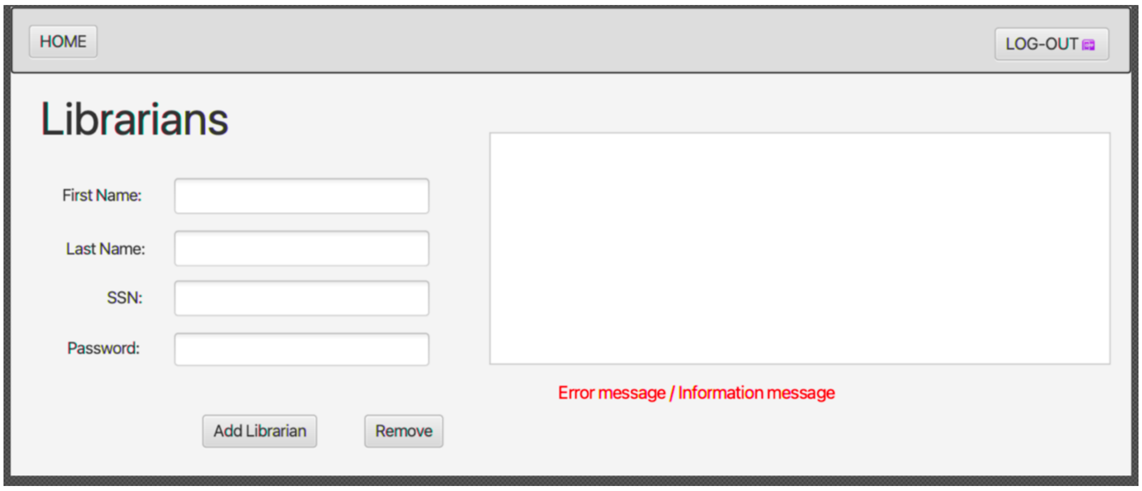


Figure 18Add/Remove Librarians UI (see appendix K)

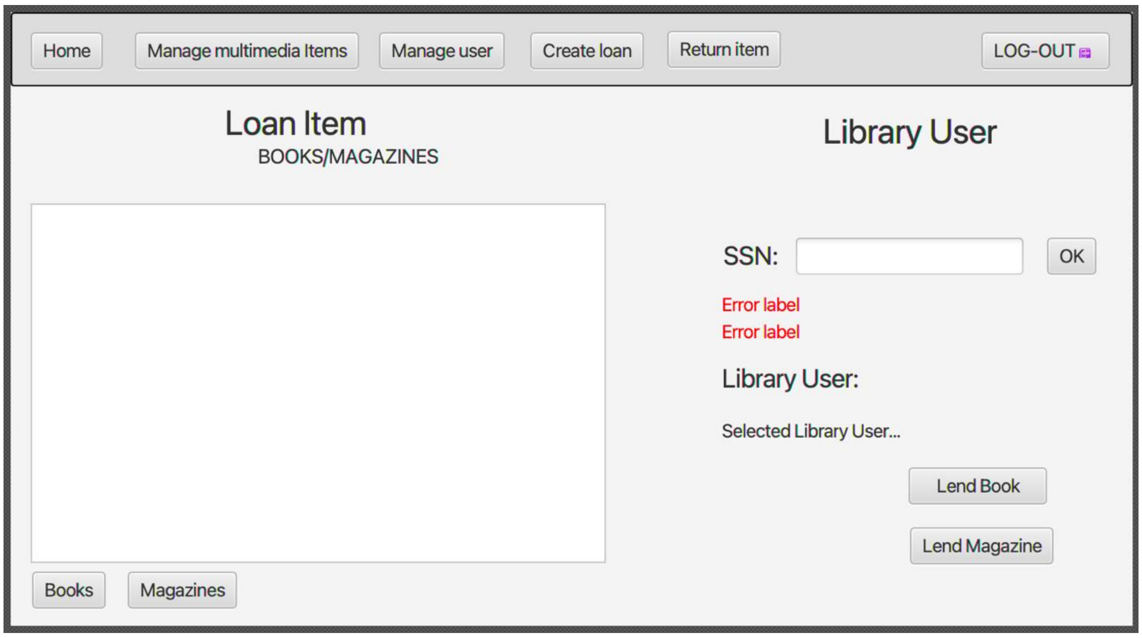


Figure 19.Lend Multimedia Item UI (see appendix K)

The design layout is kept simple and clear in all the views, it is formed mainly by labels, text fields, list views, and buttons. To be clearer and stand out for the user, the error labels are bright red. The layout is based on inserting the desire information in the text fields and using the buttons to execute a specific action.

LendMultimediaItem (see figure 19) and returnMultimediaItem contain both the same view for books and magazines. But for adding a multimedia item we have two different views with the same layout, one for books and the other one for magazines, addRemoveBook (see figure 17) and addRemoveMagazine.

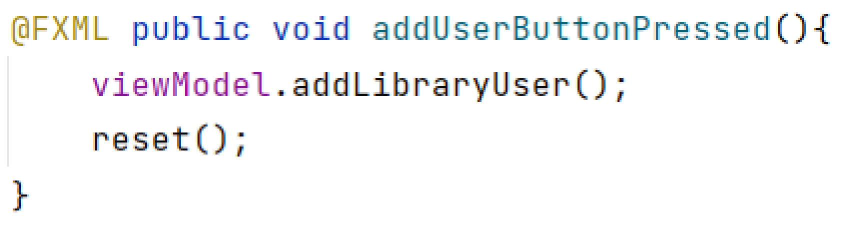
All the views have a navigation bar on the top, making it easier for the users to navigate between windows, having as the main view the home view. On the right top, there is a logout button that closes the view.

4 Implementation

4.1 Path to add a library user

Following the design, this section explains the path that the system implementation follows from the user interface to the database in order to add a library user.

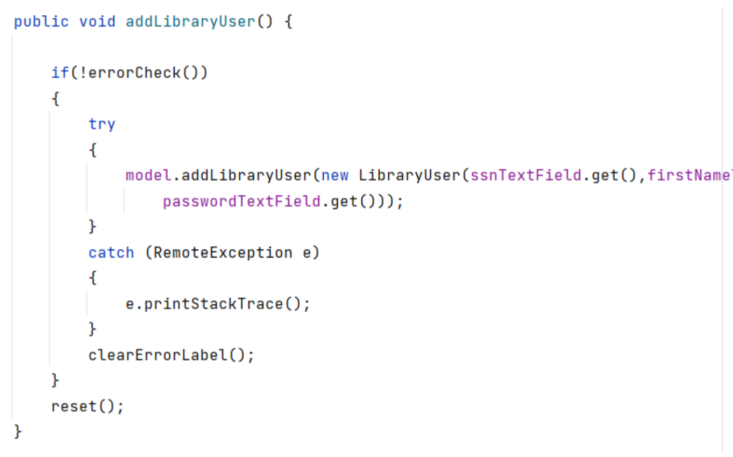
When the button is pressed in the user interface, the method `addLibraryUserButtonPressed()` (see figure 20) is called in the `AddRemoveLibraryUserViewController.`



```
@FXML public void addUserButtonPressed(){
    viewModel.addLibraryUser();
    reset();
}
```

Figure 20. Snippet code A, addUserButtonPressed() (see appendix K)

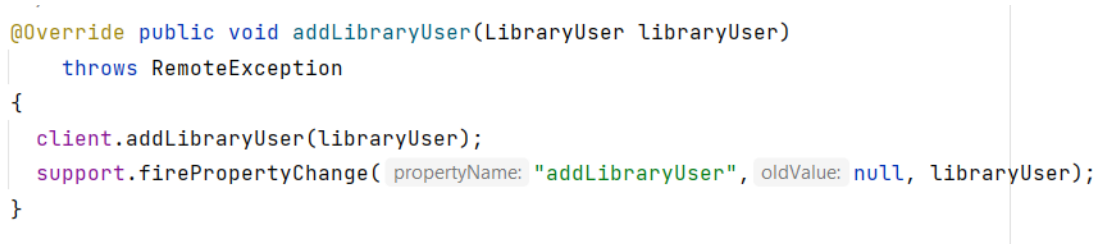
Following the MVVM pattern and as described in the design section of this report, the method delegates to the `viewModel` (see figure 21). Because all the `textFields` are binded, the `viewModel` can check if it is correct. In that case, it creates a `LibraryUser` object, sends it to the `ModelLibraryUser` interface and handles the exception that this may throw.



```java
public void addLibraryUser() {

    if(!errorCheck())
    {
        try
        {
            model.addLibraryUser(new LibraryUser(ssnTextField.get(),firstNameTextField.get(),lastNameTextField.get(),
                passwordTextField.get()));
        }
        catch (RemoteException e)
        {
            e.printStackTrace();
        }
        clearErrorLabel();
    }
    reset();
}
```

Figure 21.Snippet code B, addLibraryUser() (see appendix K)

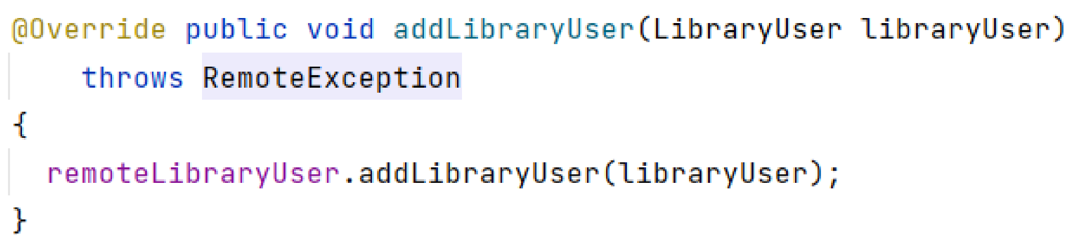
The `model`, or in this case (following the Interface-Segregation Principle), the model manager (see figure 22), receives the `LibraryUser` object and sends it to the client.



```java
@Override public void addLibraryUser(LibraryUser libraryUser)
    throws RemoteException
{
  client.addLibraryUser(libraryUser);
  support.firePropertyChange( propertyName: "addLibraryUser", oldValue: null, libraryUser);
}
```

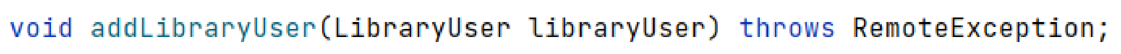
Figure 22. Snippet code C, addLibraryUser() (see appendix K)

The client itself (see figure 23) delegates to the remote interface (see figure 24) and it makes the object be in the shared package.



```java
@Override public void addLibraryUser(LibraryUser libraryUser)
    throws RemoteException
{
  remoteLibraryUser.addLibraryUser(libraryUser);
}
```

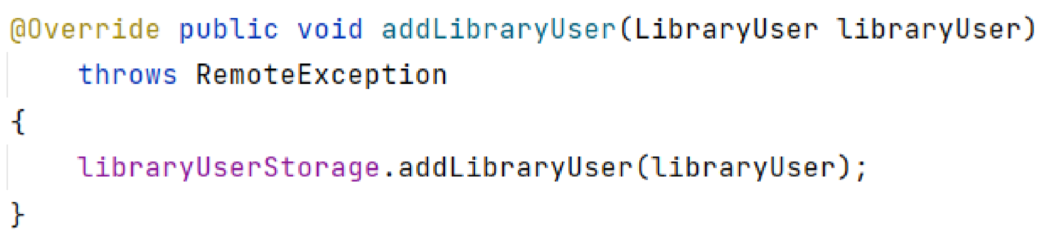
Figure 23.Snippet code D addLibraryUser() (See appendix K)



```java
void addLibraryUser(LibraryUser libraryUser) throws RemoteException;
```

Figure 24. Snipped code addLibraryUser() (See appendix K)

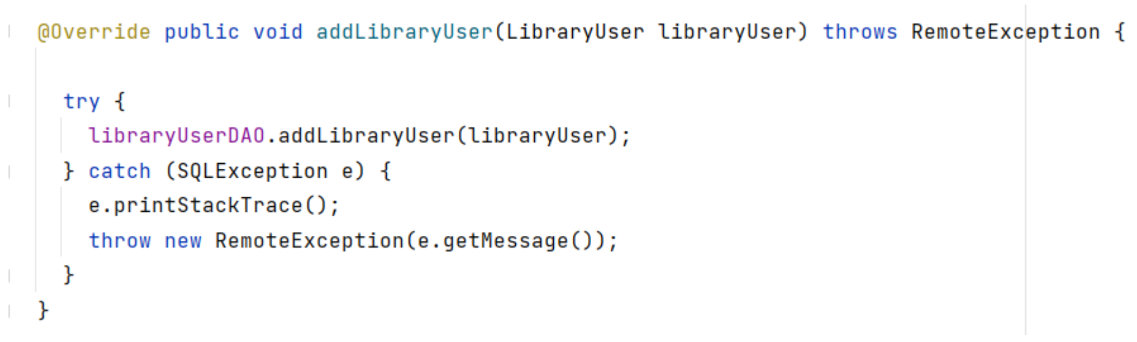
The class that is implementing the remote interface and, therefore, in charge of calling the method is the `Communicator`. The object finally goes to the `server package` and is one step closer to the database. The `Communicator,` as previous classes, only delegates the action to the storage interface.



```java
@Override public void addLibraryUser(LibraryUser libraryUser)
    throws RemoteException
{
    libraryUserStorage.addLibraryUser(libraryUser);
}
```

Figure 25.Snippet code E addLibraryUser() (See appendix K)

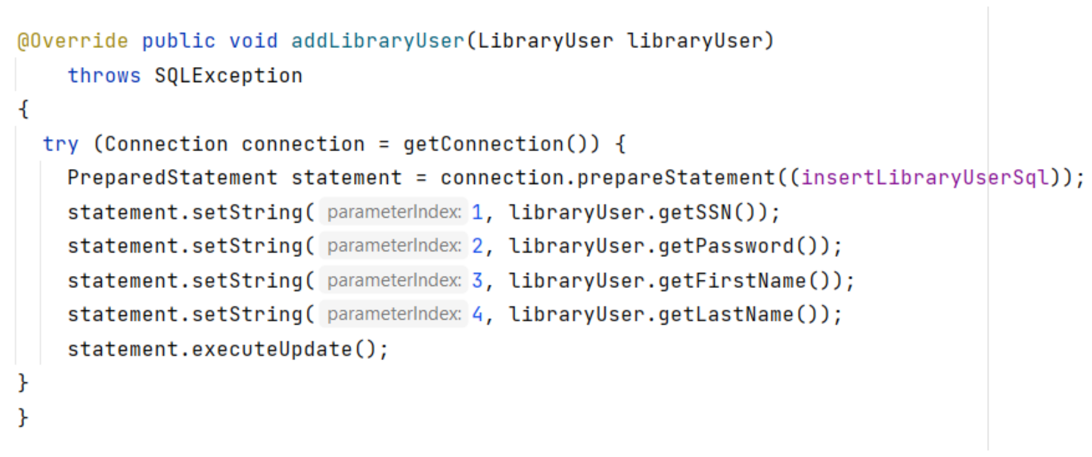
The storage interface is implemented by the `AdapterLibraryUserDAO` class (see figure 26), following the adapter pattern explained in the analysis section of this report. The object that was created in the view model is now sent to the LibraryUserDAO interface and the `LIbraryUserDAOImplementation` (see figure 27) is the responsible of adding it to the database.



```java
@Override public void addLibraryUser(LibraryUser libraryUser) throws RemoteException {

  try {
    libraryUserDAO.addLibraryUser(libraryUser);
  } catch (SQLException e) {
    e.printStackTrace();
    throw new RemoteException(e.getMessage());
  }
}
```

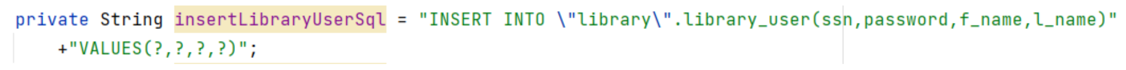
Figure 26.Snippet code F addLibraryUser() (See appendix K)

In order to add it the database, the method takes the library user as a parameter and gets its information to substitute it for the ? in the prepared statement (see figure 28). Once it has done that, it executes the update, and the user is added to the database.



```
@Override public void addLibraryUser(LibraryUser libraryUser)
    throws SQLException
{
  try (Connection connection = getConnection()) {
    PreparedStatement statement = connection.prepareStatement((insertLibraryUserSql));
    statement.setString( parameterIndex: 1, libraryUser.getSSN());
    statement.setString( parameterIndex: 2, libraryUser.getPassword());
    statement.setString( parameterIndex: 3, libraryUser.getFirstName());
    statement.setString( parameterIndex: 4, libraryUser.getLastName());
    statement.executeUpdate();
}
}
```

Figure 27.Snippet code G addLibraryUSer() (See appendix K)



```
private String insertLibraryUserSql = "INSERT INTO \"library\".library_user(ssn,password,f_name,l_name)"
    +"VALUES(?,?,?,?)";
```

Figure 28.Snippet code H prepared statements for adding a library user (See appendix K)

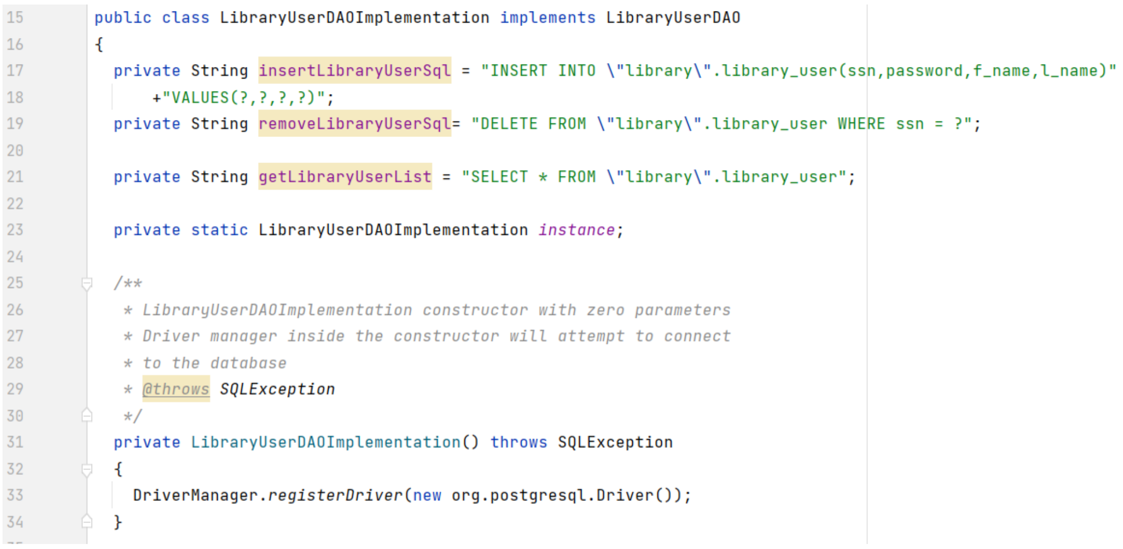
At this point, the model (see figure 22) fires the property change and the view model, which is listening to it, updates itself and shows the list, now including the new library user.

4.2 Patterns implementation

The following sections explains how some of the patterns mention in the design section of this report have been implemented, for a better understanding of them.

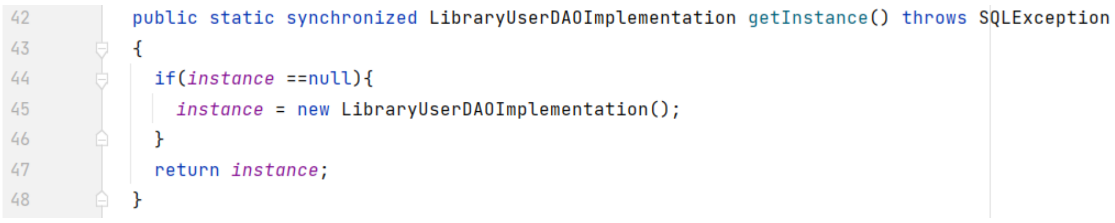
4.2.1 Singleton

According to the design, the `DAOImplementation` classes are implemented as singleton. As an example, the figure below (see figure 29) shows the static variable called `instance` (line 23) and the private constructor (line 31) from the `LIbraryUserDAOIplementation` class. This ensures that the only way to get an object of this type from outside the class is through the static method `getInstance()` (see figure 30). This method is synchronized to make sure that no more than one client (acting as threads) tries to execute the code at the same time.



```java
public class LibraryUserDAOImplementation implements LibraryUserDAO
{
  private String insertLibraryUserSql = "INSERT INTO \"library\".library_user(ssn,password,f_name,l_name)"
      +"VALUES(?,?,?,?)";
  private String removeLibraryUserSql= "DELETE FROM \"library\".library_user WHERE ssn = ?";

  private String getLibraryUserList = "SELECT * FROM \"library\".library_user";

  private static LibraryUserDAOImplementation instance;

  /**
   * LibraryUserDAOImplementation constructor with zero parameters
   * Driver manager inside the constructor will attempt to connect
   * to the database
   * @throws SQLException
   */
  private LibraryUserDAOImplementation() throws SQLException
  {
    DriverManager.registerDriver(new org.postgresql.Driver());
  }
```

Figure 29.Singleton LibraryUserDaoImplementation class (see appendix K)



```java
public static synchronized LibraryUserDAOImplementation getInstance() throws SQLException
{
  if(instance ==null){
    instance = new LibraryUserDAOImplementation();
  }
  return instance;
}
```

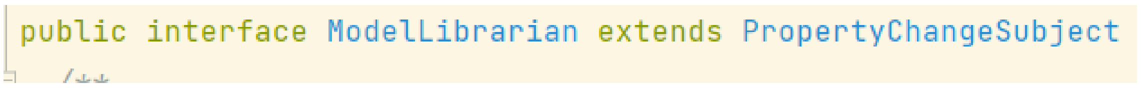
Figure 30.Singleton getInstance() (see appendix K)

4.2.2 Observer pattern

Observer pattern is implemented in order to establish communication between objects: observable and observers. Using this particular Java design pattern allows to notify observers about changing states. During the implementation process interface `PropertyChangeSubject` is created. This class contains four methods. Two `addPropertyChangeListener` methods with different parameters and two `removePropertyChangeListener` methods (with parameters: `listener` and `name,listener`). In every `ModelManager` class there is reference to `PropertyChangeSupport - support`. Whenever the state is changed `support.firePropertyChangeSupport()` is used. In addition to that, using support

allows to add and remove observers, notify them when observable state changes. Following the pattern every `ViewModel` class is equipped with `propertyChange()` method looking for `name` (type `String`) which is a message for observers.
An example for the observer pattern in the project is shown through the following figures and description:

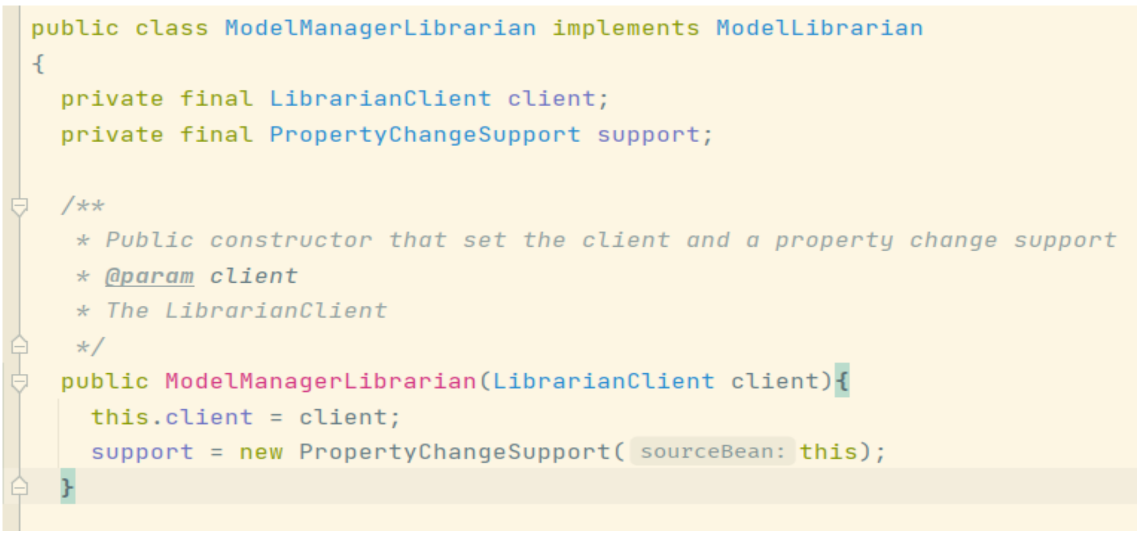
`ModelLibrarian` interface which extends the `PropertyChangeSubject` (see figure 56).



```
public interface ModelLibrarian extends PropertyChangeSubject
   /**
```

Figure 56. Observer example p1 (see appendix K)

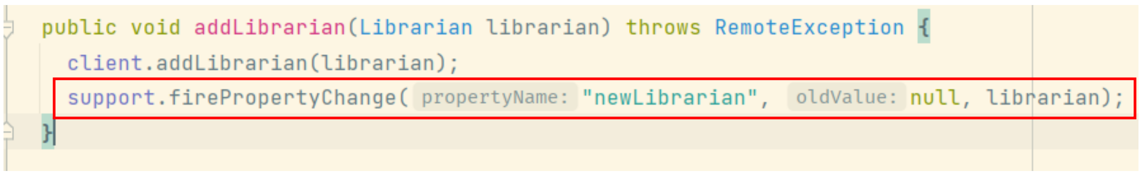
The `ModelLibrarianManager` implements `ModelLibrarian` and is added as a subject. (see figure 57).



```
public class ModelManagerLibrarian implements ModelLibrarian
{
  private final LibrarianClient client;
  private final PropertyChangeSupport support;

  /**
   * Public constructor that set the client and a property change support
   * @param client
   * The LibrarianClient
   */
  public ModelManagerLibrarian(LibrarianClient client){
    this.client = client;
    support = new PropertyChangeSupport( sourceBean: this);
  }
```

Figure 57. Observer example p2 (see appendix K)

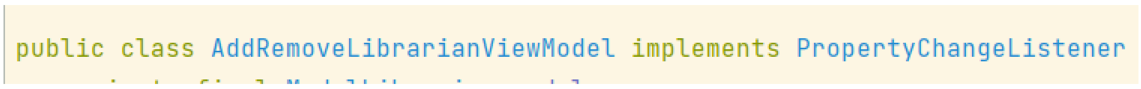
In the method of the class a property change is fired to notify the listener (see figure 58)



```java
public void addLibrarian(Librarian librarian) throws RemoteException {
    client.addLibrarian(librarian);
    support.firePropertyChange( propertyName: "newLibrarian",  oldValue: null, librarian);
}
```

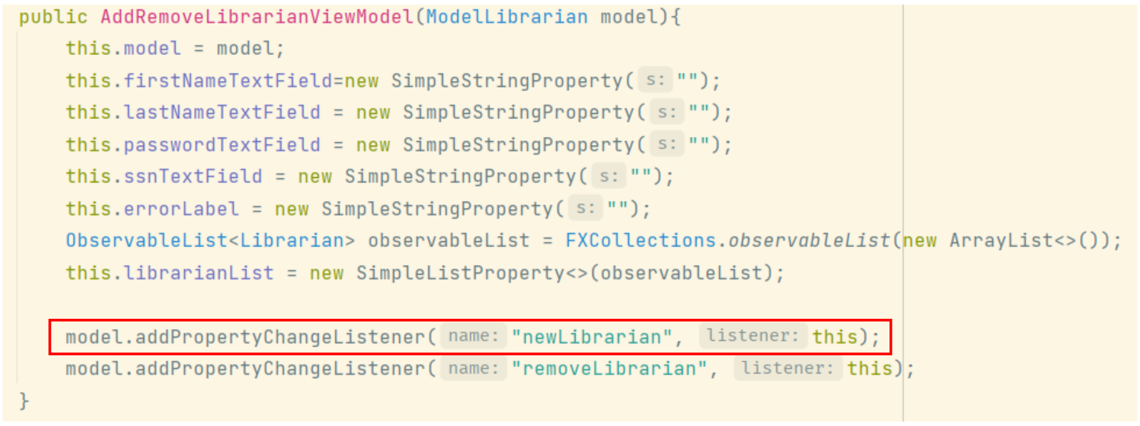
Figure 58. Observer example p3 (see appendix K)

The interested class is set as a listener (see figure 59) and in the constructor of the class listeners for the property change are added (see figure 60)



```java
public class AddRemoveLibrarianViewModel implements PropertyChangeListener
```

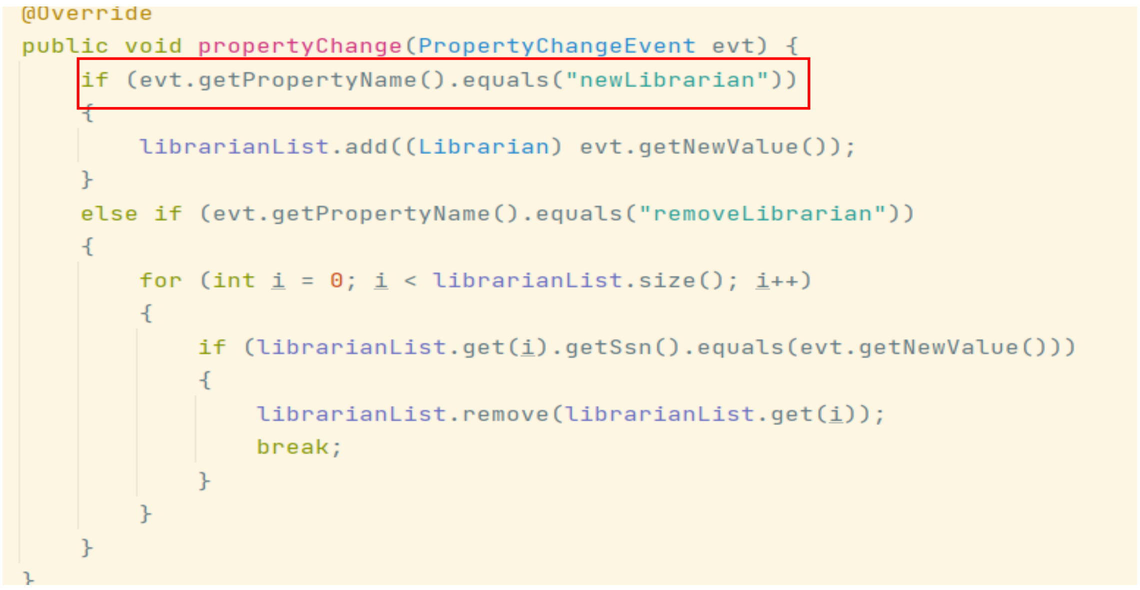
Figure 59. Observer Example p4 (see appendix K)



```java
public AddRemoveLibrarianViewModel(ModelLibrarian model){
    this.model = model;
    this.firstNameTextField=new SimpleStringProperty( s: "");
    this.lastNameTextField = new SimpleStringProperty( s: "");
    this.passwordTextField = new SimpleStringProperty( s: "");
    this.ssnTextField = new SimpleStringProperty( s: "");
    this.errorLabel = new SimpleStringProperty( s: "");
    ObservableList<Librarian> observableList = FXCollections.observableList(new ArrayList<>());
    this.librarianList = new SimpleListProperty<>(observableList);

    model.addPropertyChangeListener( name: "newLibrarian",  listener: this);
    model.addPropertyChangeListener( name: "removeLibrarian",  listener: this);
}
```

Figure 60. Observer Example p5 (see appendix K)

Finally, the behaviour of the class when a property change is detected is stated in the code of the `propertyChange` method (see figure 61).



```java
@Override
public void propertyChange(PropertyChangeEvent evt) {
    if (evt.getPropertyName().equals("newLibrarian"))
    {
        librarianList.add((Librarian) evt.getNewValue());
    }
    else if (evt.getPropertyName().equals("removeLibrarian"))
    {
        for (int i = 0; i < librarianList.size(); i++)
        {
            if (librarianList.get(i).getSsn().equals(evt.getNewValue()))
            {
                librarianList.remove(librarianList.get(i));
                break;
            }
        }
    }
}
```

Figure 61. Observer example p6 (see appendix K)

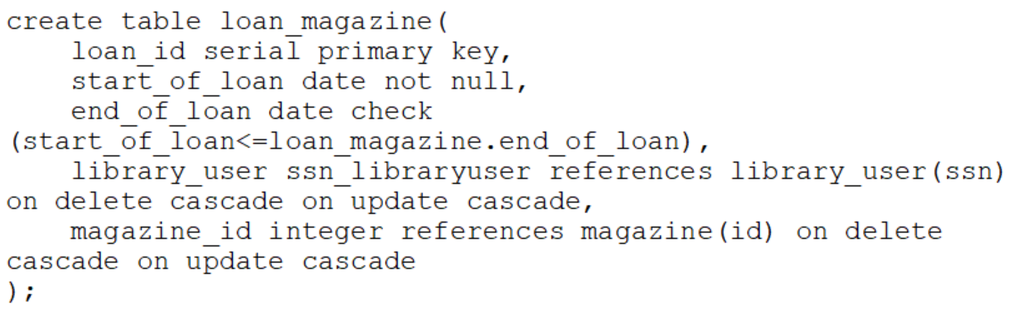
4.3 Database implementation

Database is implemented following the design and the non-functional requirements stated in analysis.

Figure 31 is an example of the sql written to create the table `loan_magazine`.

`Loan_id`, the primary key, is defined to be serial. On delete cascade is established, so, when a library user or a magazine is deleted, the information about them and their loans are removed (non-functional requirement d). Moreover, on update cascade of the foreign keys assures an update when a change occurs.

Two additional checks in `start_of_loan` and `end_of_loan` guarantee that the `start_of_loan` is not null and that a magazine is not returned (`end_of_loan`) before the `start_of_loan.`

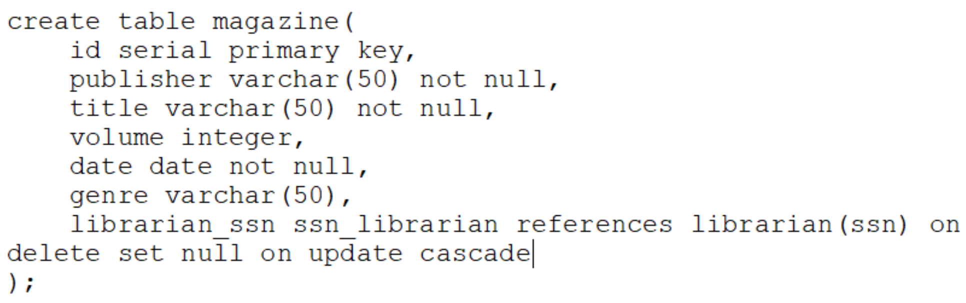


```
create table loan_magazine(
    loan_id serial primary key,
    start_of_loan date not null,
    end_of_loan date check
(start_of_loan<=loan_magazine.end_of_loan),
    library_user ssn_libraryuser references library_user(ssn)
on delete cascade on update cascade,
    magazine_id integer references magazine(id) on delete
cascade on update cascade
);
```

Figure 31.SQL to create loan magazine table

Not null constrains have been written to fulfil non-functional requirement e. Figure 32 is an example of how the mandatory information `publisher, title` and `date` contains said constrains in the table `magazine`. The foreign key that reference to the `librarian` that adds the magazine can be null and it is stablished to be set to null when the librarian no longer belongs to the system.

-

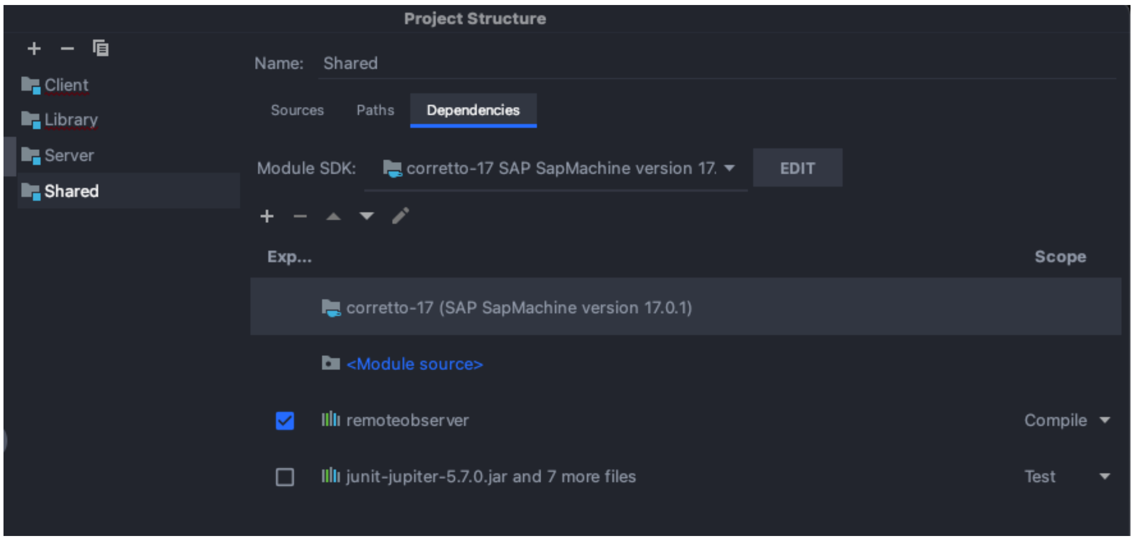


```
create table magazine(
    id serial primary key,
    publisher varchar(50) not null,
    title varchar(50) not null,
    volume integer,
    date date not null,
    genre varchar(50),
    librarian_ssn ssn_librarian references librarian(ssn) on
delete set null on update cascade
);
```

Figure 32.SQL to create the magazine table

4.4 Libraries and Dependencies

The system is implemented in three main modules: `client`, `server`, and `shared`. The `share` module contains mainly the remoteoberserver.jar library that is exported to the other modules. In order to test the code, JUnit5.7.0.jar is added to the dependencies of the `share` module.



Project Structure

Client
Library
Server
Shared

Name: Shared

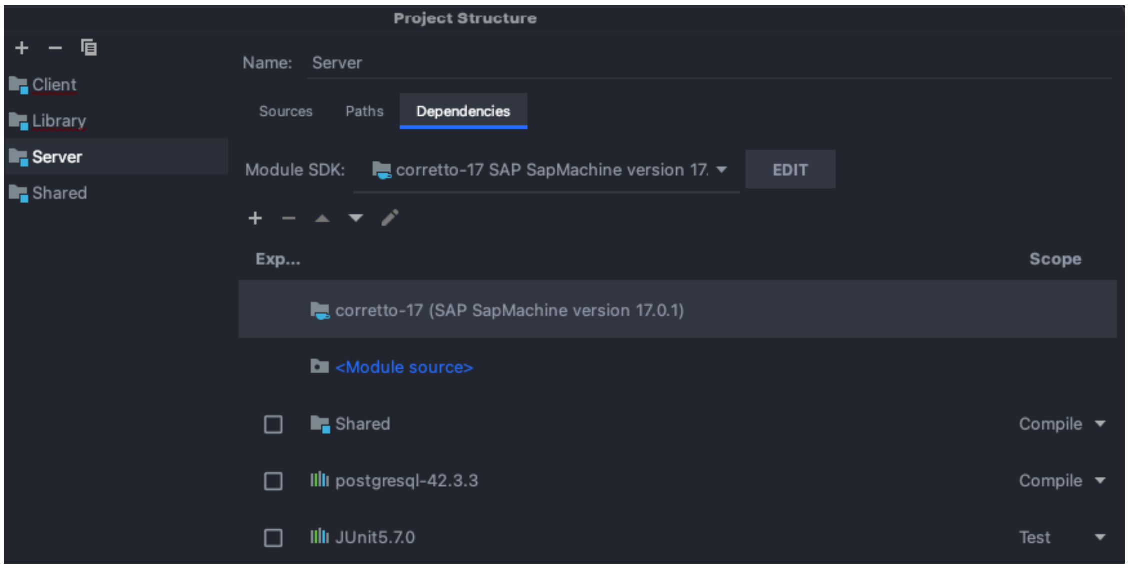
Sources | Paths | Dependencies

Module SDK: corretto-17 SAP SapMachine version 17. | EDIT

Exp...		Scope
	corretto-17 (SAP SapMachine version 17.0.1)	
	<Module source>	
✓	remoteobserver	Compile
☐	junit-jupiter-5.7.0.jar and 7 more files	Test

Figure 33.Libraries and dependencies Shared Module

The `server` module contains JUnit5.7.0.jar to test the code. The postgresql-42.3.3.jar in order to implement the database and the `shared` module is added to the dependencies so the classes in the shared module can be accessed in the server module.



Project Structure

Client
Library
Server
Shared

Name: Server

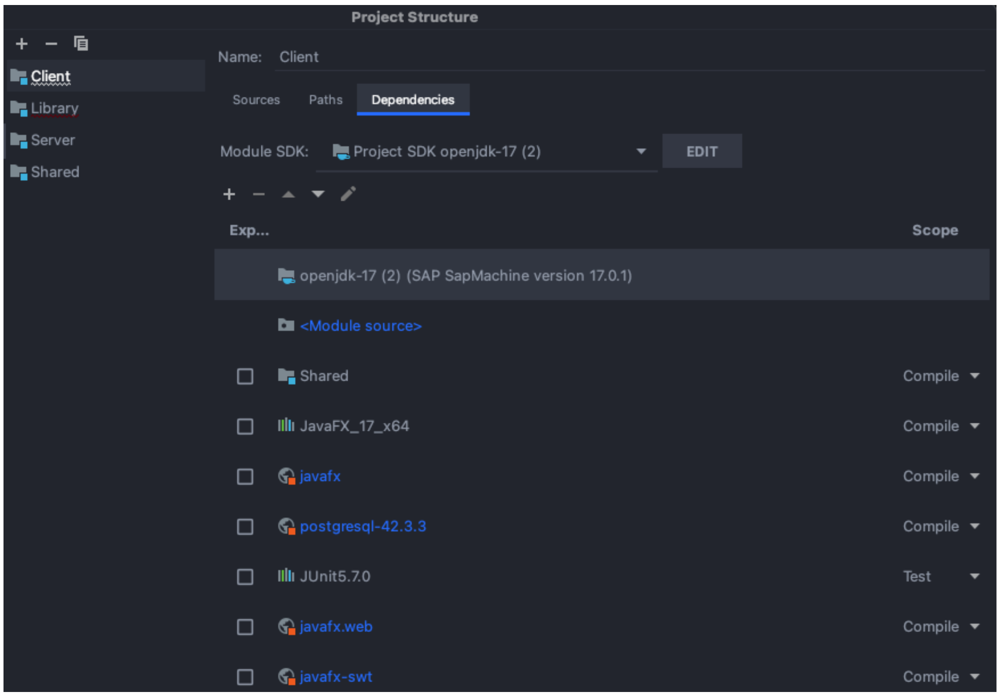
Sources | Paths | Dependencies

Module SDK: corretto-17 SAP SapMachine version 17. | EDIT

Exp...		Scope
	corretto-17 (SAP SapMachine version 17.0.1)	
	<Module source>	
☐	Shared	Compile
☐	postgresql-42.3.3	Compile
☐	JUnit5.7.0	Test

Figure 34.Dependecies and lIbraries Server Module

The `client` module contains like the other modules JUnit5.7.0 in order to test the code, the `shared` module is added to the dependencies so the classes in the shared module can be accessed in the `client` module. The GUI (graphical user interface) requires the JavaFX_17_x64.jar library in macOS and javafx-sdk-17.0.2.


Project Structure

Client
Library
Server
Shared

Name: Client

Sources | Paths | Dependencies

Module SDK: Project SDK openjdk-17 (2) | EDIT

Exp...		Scope
	openjdk-17 (2) (SAP SapMachine version 17.0.1)	
	<Module source>	
☐	Shared	Compile
☐	JavaFX_17_x64	Compile
☐	javafx	Compile
☐	postgresql-42.3.3	Compile
☐	JUnit5.7.0	Test
☐	javafx.web	Compile
☐	javafx-swt	Compile

Figure 35.Dependencies and libraries Client Module

5 Test

This section covers the results of the system testing to prove if the requirements have been fulfilled. The system has been tested using White Box testing (JUnit) and Black Box testing creating Test cases based on the Use Cases.

Test cases (see appendix E) have only been created for the use cases which have been analysed. The table below illustrates the result of said tests. It is important to highlight that “not passed” means that there is no test case for that user story because it has not been analysed or implemented and all the functionality which has been implemented is currently passing all test cases.

Use case	User Story	Passed/not passed
Manage librarian	2	Passed

	5	Passed
Reserve multimedia item	17	Not passed
	11	Not passed
See notification	15	Not passed
	16	Not passed
	11	Not passed
Write review	18	Not passed
Delete review	19	Not passed
	22	Not passed
See information	8	Not passed
	9	Not passed
	12	Not passed
	13	Not passed
Search multimedia item	7	Not passed
Manage fines	9	Not passed
	10	Not passed
Manage loans	4	Passed
	21	Passed
	14	Not passed
Manage multimedia item	1	Passed
	6	Passed (without filtering)
Manage library users	3	Passed
	20	Passed

Table 5.User stories fulfilled

5.1 Black box: Test Cases

The test cases elaborated for each use case can be found in appendix E. The system response in them all is as expected, so it is concluded that the system successfully reaction as desired based on the test cases.

Below it is shown an example of one of the test cases for the use case Add Magazine.

Normal flow:

Action no.	Action	Reaction	Result
1	Choose add magazine	System shows a list with all the magazines. System asks for the title, publisher, volume, date (day, month and year) and genre	Expected – Test passed
2	Introduce the following data: Title: Den danske spectator Publisher: Jorgen Riis Volume: 5 Day: 12 Month: 3 Year: 2005 Genre: literary review		
3	Choose to add the magazine	System adds the magazine with the following information: Title: Den danske spectator Publisher: Jorgen Riis Volume: 5 Date: 12/03/2005 Genre: literary review The magazine is shown on the list and the values of the filled fields are reset.	Expected – Test passed

Alternative flow book: Librarian inputs incorrect type of data

a. Null value

Step 1-3 in normal flow except title/ publisher/day/month/year is empty

3a	Choose to add the magazine	System displays an error indicating that it can't be empty and resets the values of the filled fields	Expected – Test passed

Go to step 2 normal flow

a. Invalid date

Step 1-3 in normal flow except the date is invalid. Ex. 31/02/2003, -4/07/2021, 05/16/2013

3a	Choose to add the magazine	System displays an error indicating that the librarian has chosen an invalid date and resets the values of the filled fields	Expected – Test passed

Go to step 2 normal flow

a. Future date

Step 1-3 in normal flow except the date is future. Ex. 21/02/2033, 04/12/2022, 05/16/2051

3a	Choose to add the magazine	System displays an error indicating that the librarian has chosen a future date and resets the values of the filled fields	Expected – Test passed

Go to step 2 normal flow

d) Volume is not an integer

Step 1-3 in normal flow except the same genre is chosen twice

3a	Choose to add the magazine	System displays an error indicating that the volume must be an integer and resets the values of the filled fields.	Expected – Test passed

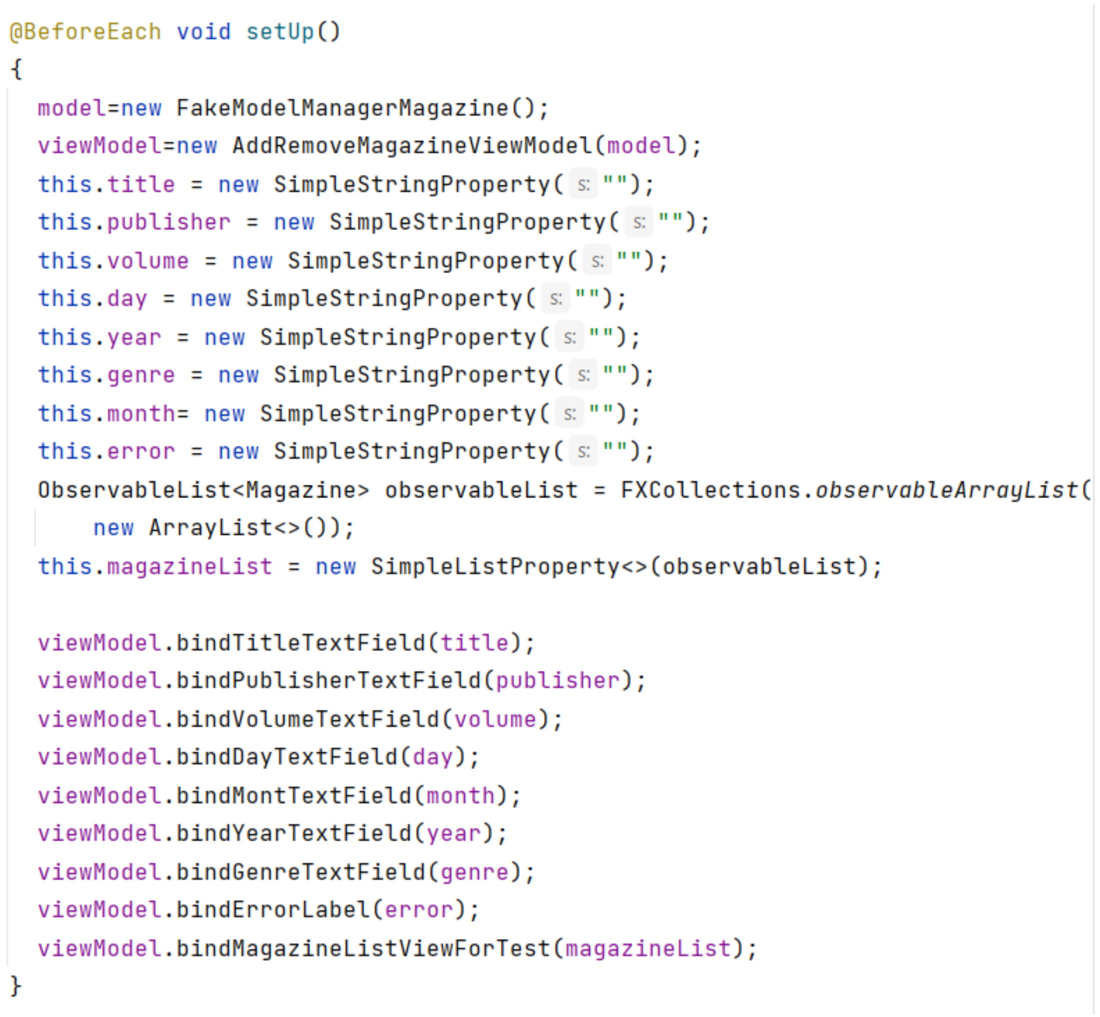
Go to step 2 normal flow

Table 6.Test case add magazines.(See Appendix E)

5.2 White Box: JUnit

Unit testing has been used to test the system implementation, with special care on following the ZOMBIES principles. Down below can be found the `AddRemoveMagazineViewModelTest`, to exemplify and illustrate said principles. The rest of the tests can be found in the test directories in the implementation (See appendix K and E).

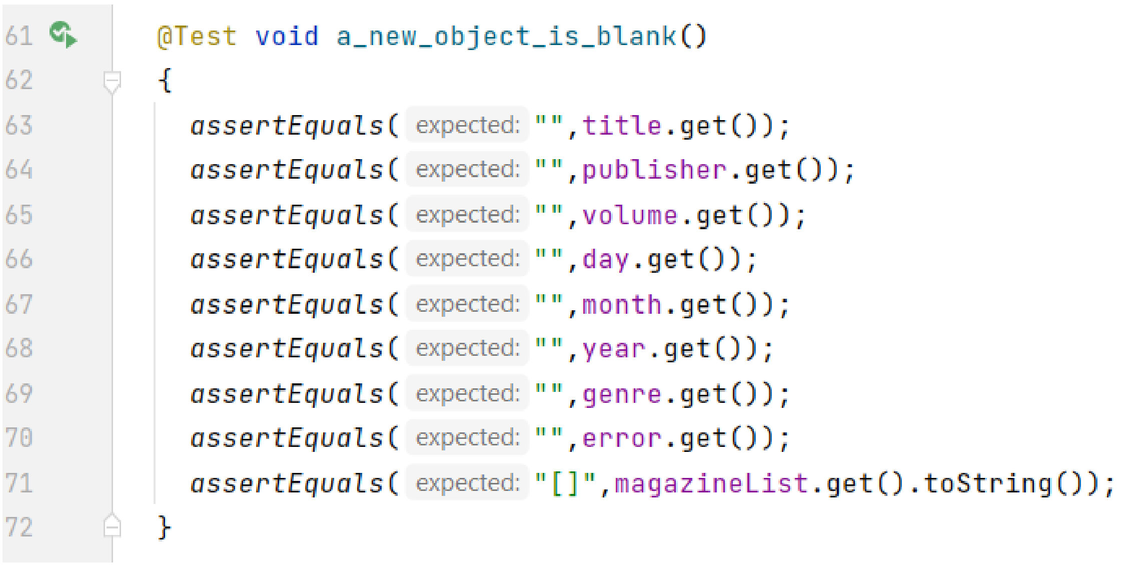
In order to test this `viewModel`, a `FakeModelManagerMagazine` is used, as well as a series of simple string properties which are binded to the `viewModel` in the setup (See figure 36).



```
@BeforeEach void setUp()
{
  model=new FakeModelManagerMagazine();
  viewModel=new AddRemoveMagazineViewModel(model);
  this.title = new SimpleStringProperty( s: "");
  this.publisher = new SimpleStringProperty( s: "");
  this.volume = new SimpleStringProperty( s: "");
  this.day = new SimpleStringProperty( s: "");
  this.year = new SimpleStringProperty( s: "");
  this.genre = new SimpleStringProperty( s: "");
  this.month= new SimpleStringProperty( s: "");
  this.error = new SimpleStringProperty( s: "");
  ObservableList<Magazine> observableList = FXCollections.observableArrayList(
      new ArrayList<>());
  this.magazineList = new SimpleListProperty<>(observableList);

  viewModel.bindTitleTextField(title);
  viewModel.bindPublisherTextField(publisher);
  viewModel.bindVolumeTextField(volume);
  viewModel.bindDayTextField(day);
  viewModel.bindMontTextField(month);
  viewModel.bindYearTextField(year);
  viewModel.bindGenreTextField(genre);
  viewModel.bindErrorLabel(error);
  viewModel.bindMagazineListViewForTest(magazineList);
}
```

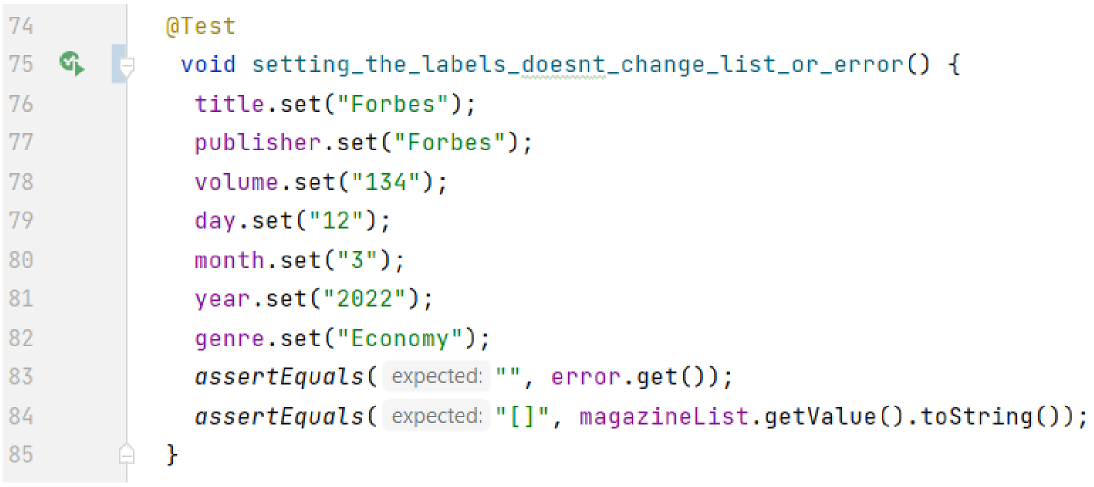
Figure 36. Junit Test setUp(See appendix K)

5.2.1 Z-zero:



```
@Test void a_new_object_is_blank()
{
   assertEquals( expected: "",title.get());
   assertEquals( expected: "",publisher.get());
   assertEquals( expected: "",volume.get());
   assertEquals( expected: "",day.get());
   assertEquals( expected: "",month.get());
   assertEquals( expected: "",year.get());
   assertEquals( expected: "",genre.get());
   assertEquals( expected: "",error.get());
   assertEquals( expected: "[]",magazineList.get().toString());
}
```

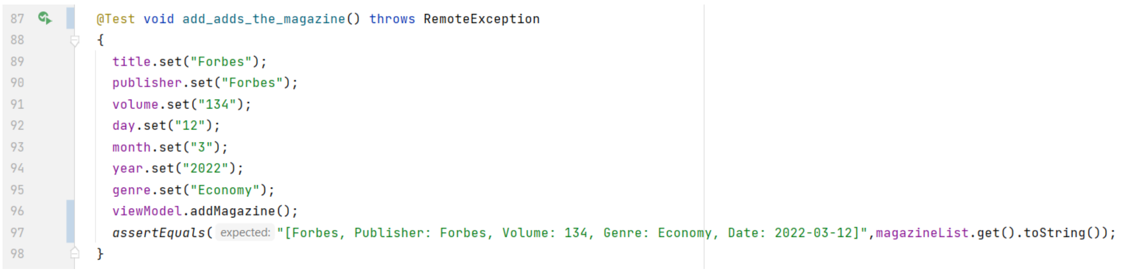
Figure 37. Junit Test case zero (See appendix K)



```
@Test
 void setting_the_labels_doesnt_change_list_or_error() {
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("3");
  year.set("2022");
  genre.set("Economy");
  assertEquals( expected: "", error.get());
  assertEquals( expected: "[]", magazineList.getValue().toString());
}
```

Figure 38. Test code case zero (See appendix K)

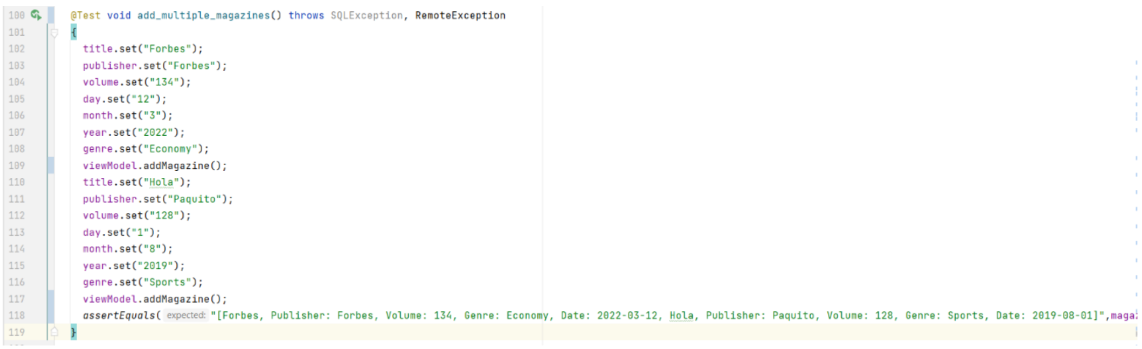
5.2.2 O-one



```
@Test void add_adds_the_magazine() throws RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("3");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "[Forbes, Publisher: Forbes, Volume: 134, Genre: Economy, Date: 2022-03-12]",magazineList.get().toString());
}
```

Figure 39. Test code case ***o****ne (See appendix K)*

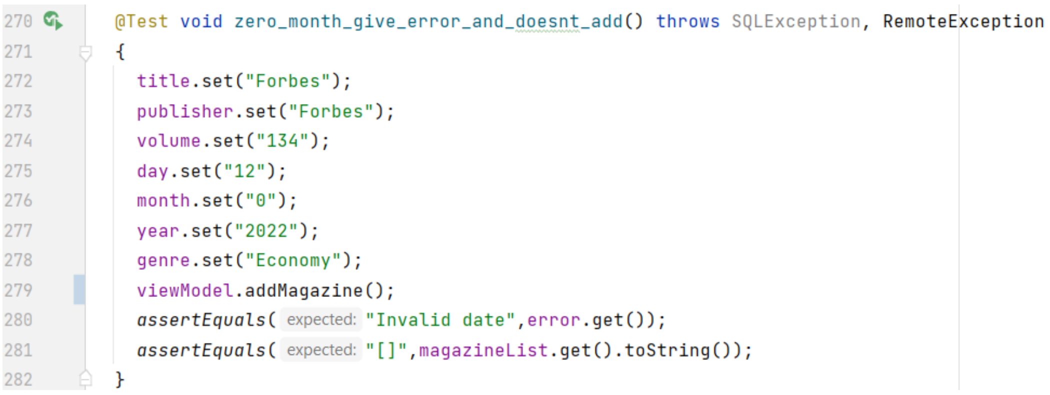
5.2.3 M-multiple



```
@Test void add_multiple_magazines() throws SQLException, RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("3");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  title.set("Hola");
  publisher.set("Paquito");
  volume.set("128");
  day.set("1");
  month.set("8");
  year.set("2019");
  genre.set("Sports");
  viewModel.addMagazine();
  assertEquals( expected: "[Forbes, Publisher: Forbes, Volume: 134, Genre: Economy, Date: 2022-03-12, Hola, Publisher: Paquito, Volume: 128, Genre: Sports, Date: 2019-08-01]",maga
}
```

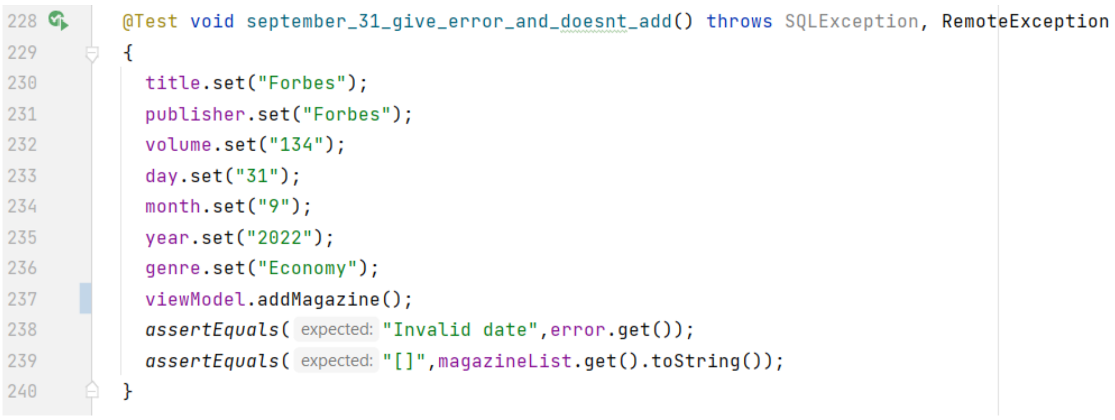
Figure 40.Test code case ***m****ultiple (See appendix K)*

5.2.4 B-Boundaries, I-Interface definition (not relevant)



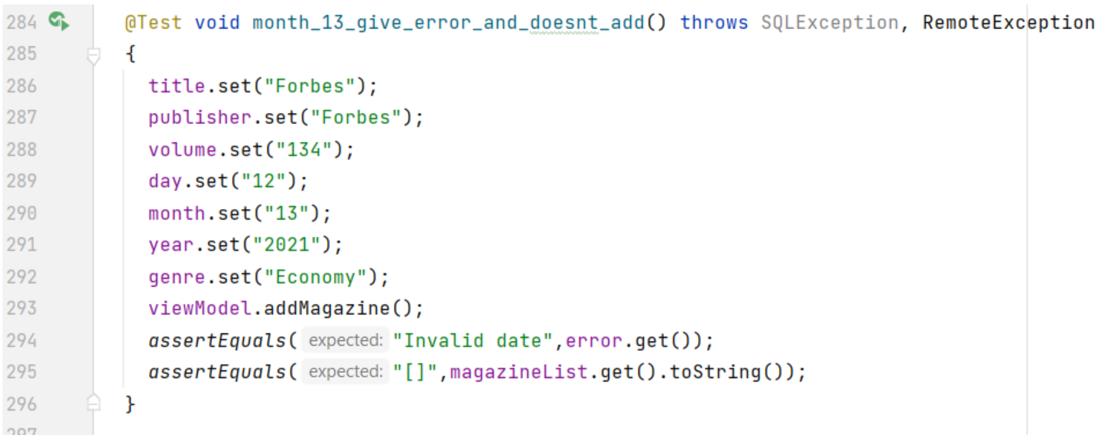
```java
@Test void zero_month_give_error_and_doesnt_add() throws SQLException, RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("0");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "Invalid date",error.get());
  assertEquals( expected: "[]",magazineList.get().toString());
}
```

Figure 41.Junit Test case boundaries (See appendix K)



```java
@Test void september_31_give_error_and_doesnt_add() throws SQLException, RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("31");
  month.set("9");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "Invalid date",error.get());
  assertEquals( expected: "[]",magazineList.get().toString());
}
```

Figure 42. Junit Test case boundaries (See appendix K)

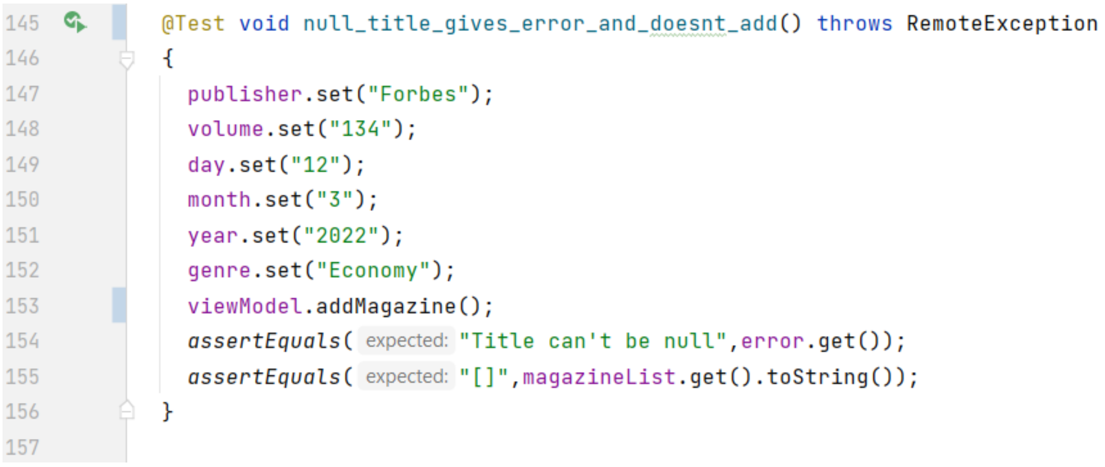


```
@Test void month_13_give_error_and_doesnt_add() throws SQLException, RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("13");
  year.set("2021");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "Invalid date",error.get());
  assertEquals( expected: "[]",magazineList.get().toString());
}
```

Figure 43. Junit Test case boundaries (See appendix K)

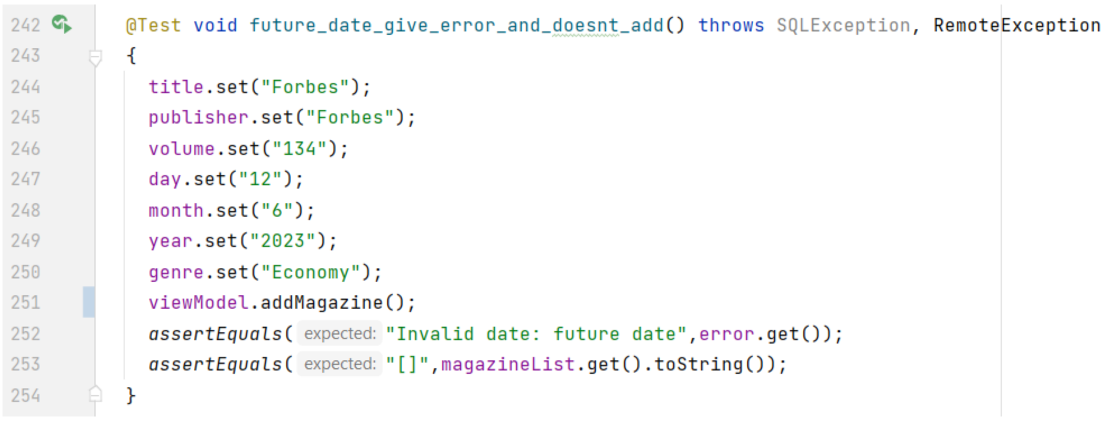
5.2.5 E-Exceptional behaviour (errors and exceptions), S-Simple scenarios (not relevant)

Only a few of the test appear in this report (see figures 44-50), to see all of them see appendix K.



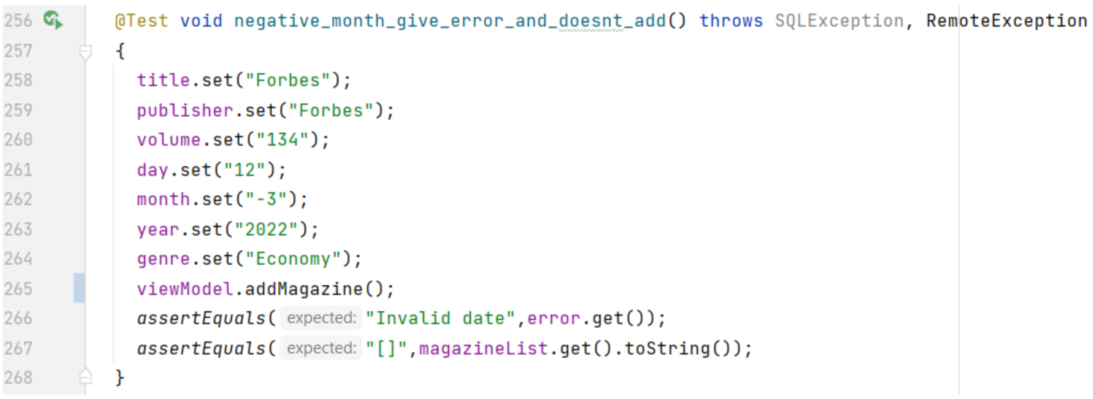
```
@Test void null_title_gives_error_and_doesnt_add() throws RemoteException
{
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("3");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "Title can't be null",error.get());
  assertEquals( expected: "[]",magazineList.get().toString());
}
```

Figure 44. Test code error Title (see appendix K)



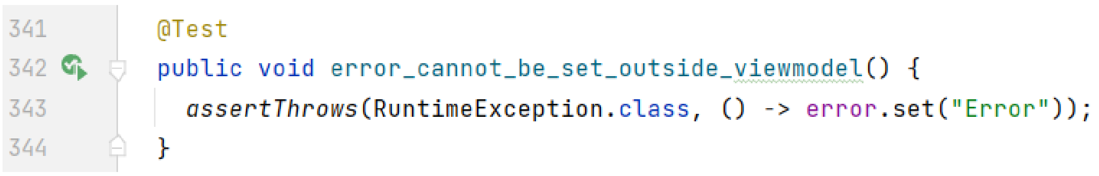
```java
@Test void future_date_give_error_and_doesnt_add() throws SQLException, RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("6");
  year.set("2023");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "Invalid date: future date",error.get());
  assertEquals( expected: "[]",magazineList.get().toString());
}
```

Figure 45. Test code error Date A (see appendix K)



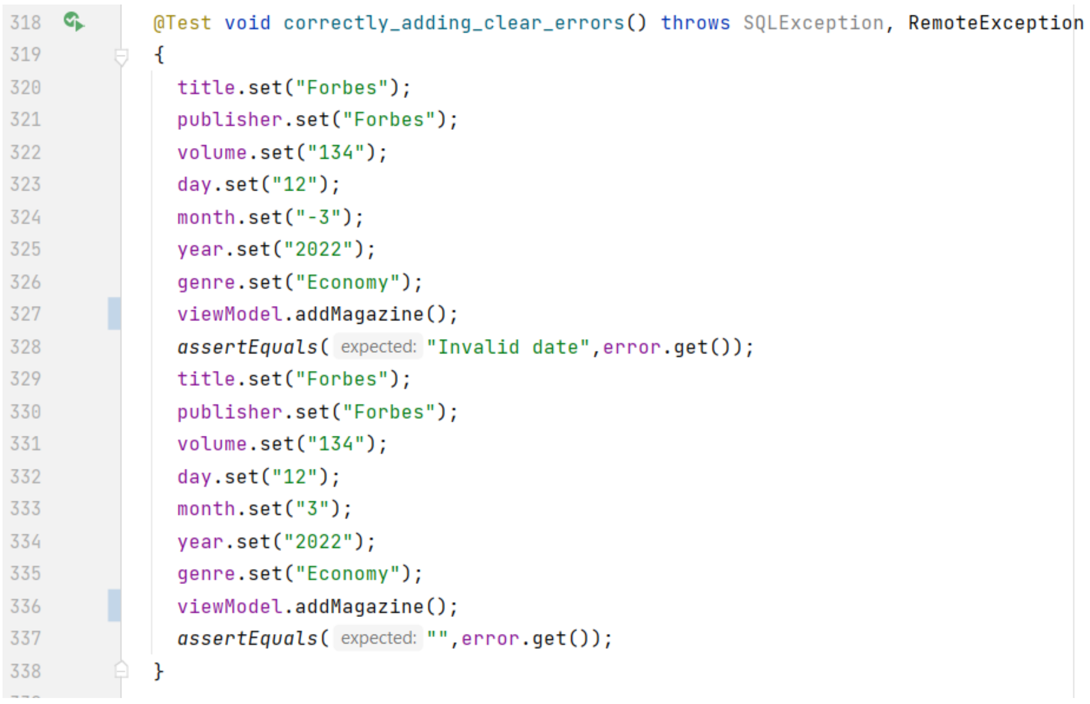
```java
@Test void negative_month_give_error_and_doesnt_add() throws SQLException, RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("-3");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "Invalid date",error.get());
  assertEquals( expected: "[]",magazineList.get().toString());
}
```

Figure 46. Test code Error Date B (see appendix K)



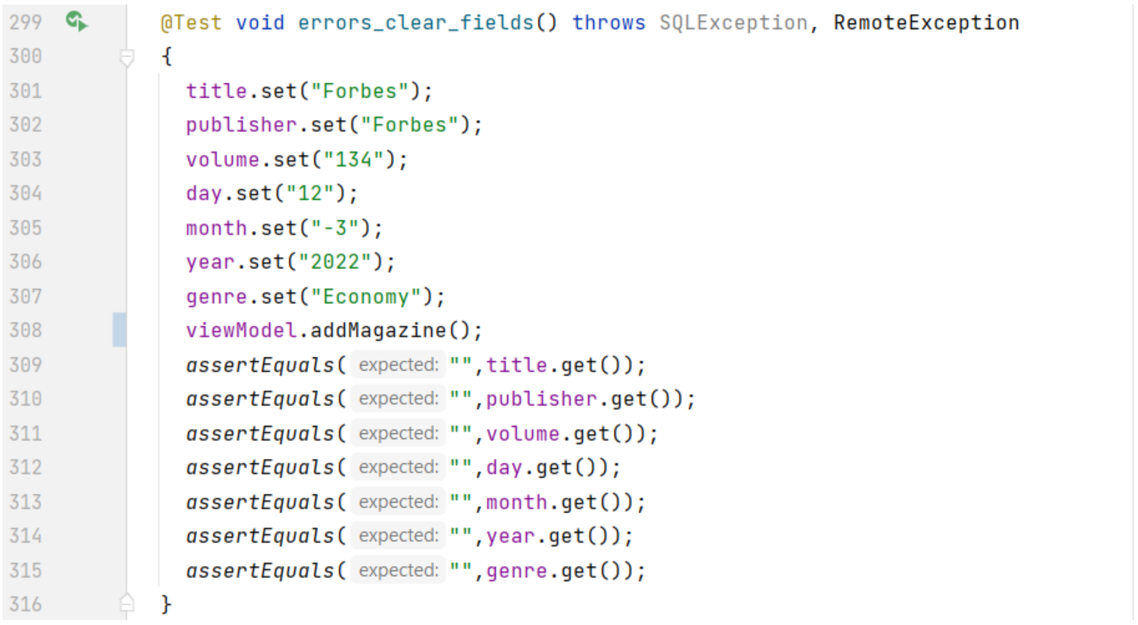
```java
@Test
public void error_cannot_be_set_outside_viewmodel() {
  assertThrows(RuntimeException.class, () -> error.set("Error"));
}
```

Figure 47. Test set error label outside the view model (see appendix K)



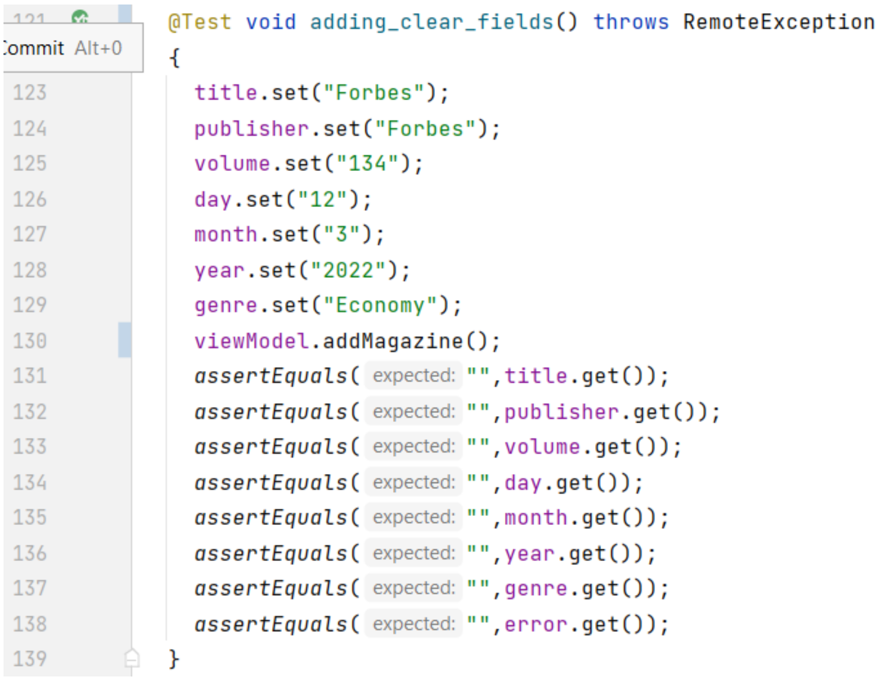
```java
@Test void correctly_adding_clear_errors() throws SQLException, RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("-3");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "Invalid date",error.get());
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("3");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "",error.get());
}
```

Figure 48. Test no error in the text fields (see appendix K)



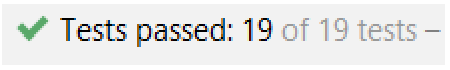
```java
@Test void errors_clear_fields() throws SQLException, RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("-3");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "",title.get());
  assertEquals( expected: "",publisher.get());
  assertEquals( expected: "",volume.get());
  assertEquals( expected: "",day.get());
  assertEquals( expected: "",month.get());
  assertEquals( expected: "",year.get());
  assertEquals( expected: "",genre.get());
}
```

Figure 49.Test clear text fields after error (see appendix K)



```
Commit  Alt+0
@Test void adding_clear_fields() throws RemoteException
{
  title.set("Forbes");
  publisher.set("Forbes");
  volume.set("134");
  day.set("12");
  month.set("3");
  year.set("2022");
  genre.set("Economy");
  viewModel.addMagazine();
  assertEquals( expected: "",title.get());
  assertEquals( expected: "",publisher.get());
  assertEquals( expected: "",volume.get());
  assertEquals( expected: "",day.get());
  assertEquals( expected: "",month.get());
  assertEquals( expected: "",year.get());
  assertEquals( expected: "",genre.get());
  assertEquals( expected: "",error.get());
}
```

Figure 50.Test for successful add and reset of the text fields (see appendix K)

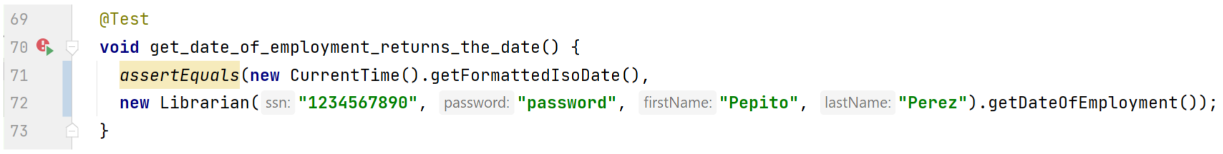


Tests passed: 19 of 19 tests –

The system was designed to be as simple as possible, both with implementation and testing in mind. However, after a mistake in the design, to remove a magazine it is needed to give the id as an argument. This id cannot be access through the `viewModel`, meaning that this functionality cannot be tested as adding.

The overall result of all JUnit tests is 118 tests passed out of 120. This chapter briefly explains them to provide information about why they fail.

The first of the mentiones failed test is `get_date_of_employment_returns_the_date()` (see figure 51)

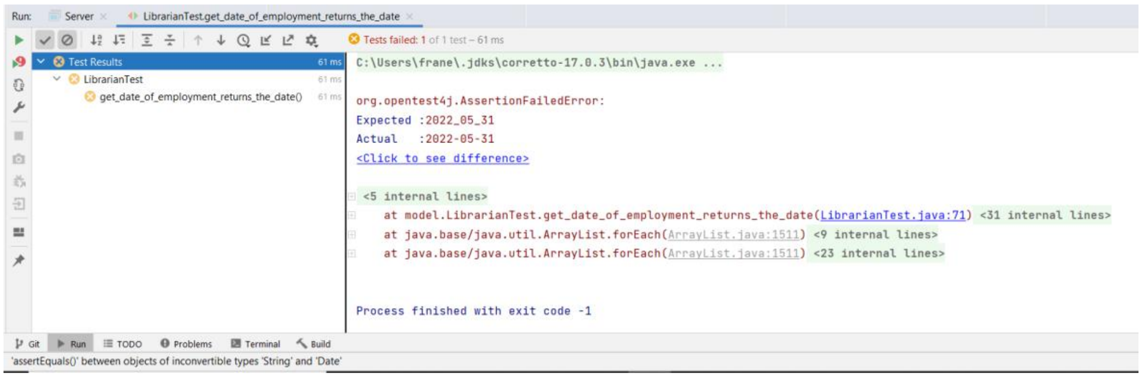


```
@Test
void get_date_of_employment_returns_the_date() {
  assertEquals(new CurrentTime().getFormattedIsoDate(),
  new Librarian( ssn: "1234567890", password: "password", firstName: "Pepito", lastName: "Perez").getDateOfEmployment());
}
```

Figure 51. Test get_date_of the employment specifics_employment_returns_date() (See appendix K)

The reason behind it is the difference between property of String and formatting of date.(See figure 22).

The expected value for the test is '2022_05_31' (day when the test was run)- it is a result of using `newCurrentTime().getFormattedIsoDate()`. Actual value is date in String format '2022-05-31'.



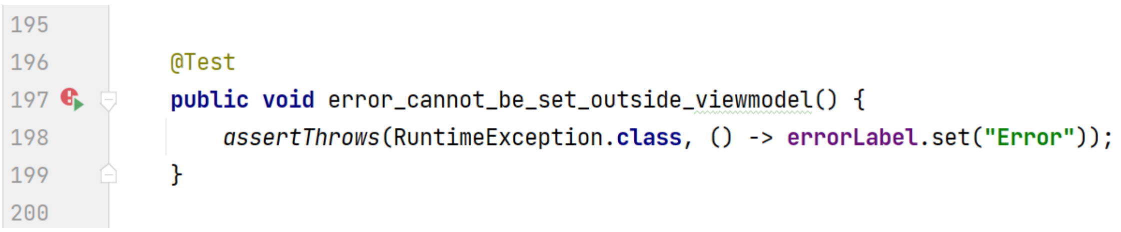
```
C:\Users\frane\.jdks\corretto-17.0.3\bin\java.exe ...

org.opentest4j.AssertionFailedError: 
Expected :2022_05_31
Actual   :2022-05-31
<Click to see difference>

 <5 internal lines>
    at model.LibrarianTest.get_date_of_employment_returns_the_date(LibrarianTest.java:71) <31 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <23 internal lines>


Process finished with exit code -1
```

Run:
Server
LibrarianTest.get_date_of_employment_returns_the_date
Tests failed: 1 of 1 test – 61 ms
Test Results 61 ms
LibrarianTest 61 ms
get_date_of_employment_returns_the_date() 61 ms
Git
Run
TODO
Problems
Terminal
Build

'assertEquals()' between objects of inconvertible types 'String' and 'Date'

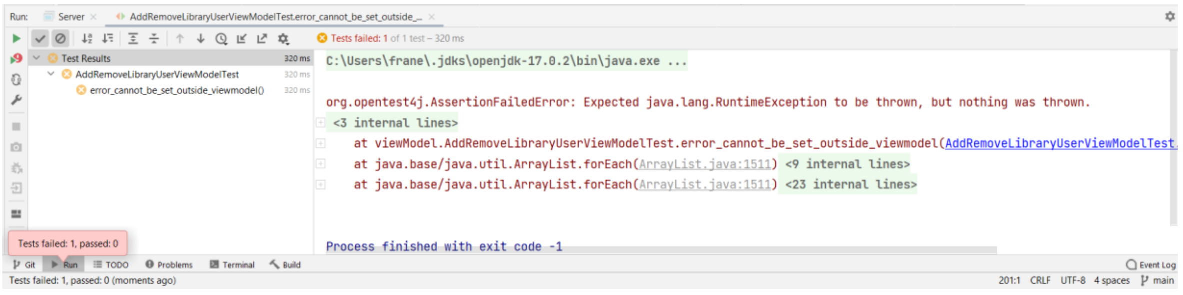
Figure 52. Test get date of the employment specifics (See appendix K)

The second test mention is `error_cannot_be_set_outside_viewmodel()` (See figure).



```java

    @Test
    public void error_cannot_be_set_outside_viewmodel() {
        assertThrows(RuntimeException.class, () -> errorLabel.set("Error"));
    }

```

Figure 53. error_label_cannot_be_set_outside_viewmodel() (see appendix K)



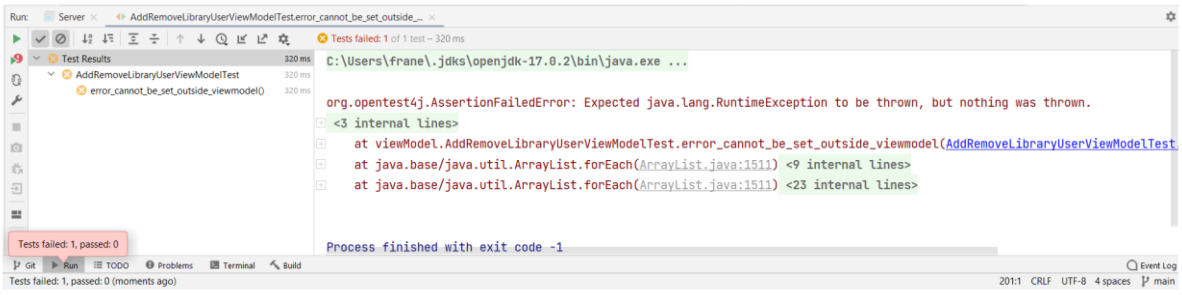
```
C:\Users\frane\.jdks\openjdk-17.0.2\bin\java.exe ...

org.opentest4j.AssertionFailedError: Expected java.lang.RuntimeException to be thrown, but nothing was thrown.
 <3 internal lines>
    at viewModel.AddRemoveLibraryUserViewModelTest.error_cannot_be_set_outside_viewmodel(AddRemoveLibraryUserViewModelTest.
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <23 internal lines>


Process finished with exit code -1
```

Run:
Server
AddRemoveLibraryUserViewModelTest.error_cannot_be_set_outside_...
Tests failed: 1 of 1 test – 320 ms
Test Results 320 ms
AddRemoveLibraryUserViewModelTest 320 ms
error_cannot_be_set_outside_viewmodel() 320 ms
Tests failed: 1, passed: 0
Git
Run
TODO
Problems
Terminal
Build
Event Log
201:1 CRLF UTF-8 4 spaces main

Tests failed: 1, passed: 0 (moments ago)

Figure 54. Result from errorLabel setting test (see appendix K)

This test check if the `errorLabel` cannot be set outside the `AddRemoveLibraryUserViewmodel`. If runtime exception is thrown `assertThrows` the test would succeed. Because no exception is thrown the test fails. It is caused by the error label being bounded bidirectional because there are some errors that are thrown from the View Controller.


Run:
Server
AddRemoveLibraryUserViewModelTest.error_cannot_be_set_outside_...
Tests failed: 1 of 1 test – 320 ms
Test Results 320 ms
AddRemoveLibraryUserViewModelTest 320 ms
error_cannot_be_set_outside_viewmodel() 320 ms
Tests failed: 1, passed: 0
Git
Run
TODO
Problems
Terminal
Build
Event Log
Tests failed: 1, passed: 0 (moments ago)
201:1 CRLF UTF-8 4 spaces main

```
C:\Users\frane\.jdks\openjdk-17.0.2\bin\java.exe ...

org.opentest4j.AssertionFailedError: Expected java.lang.RuntimeException to be thrown, but nothing was thrown.
<3 internal lines>
   at viewModel.AddRemoveLibraryUserViewModelTest.error_cannot_be_set_outside_viewmodel(AddRemoveLibraryUserViewModelTest.
   at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
   at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <23 internal lines>


Process finished with exit code -1
```

Figure 55. Result of errorLabel setting test (see appendix K)

6 Results and Discussion

This section presents the achieved results of the project.

The outcome of the project has been greatly influenced by the use of the scrum methodology and unified process. The process report explains thoroughly how they have been followed.

Most of the problems that have obstruct the project have their root in the analysis.

Regarding the user stories, some of them are not strictly following the SMART principles, for example, requirement 1 is not specific as it refers to multimedia item in general. Instead, this requirement should have been divided in two, one for book and one for magazine.

On the use cases diagrams (see appendix F), the only considered post-condition is the one associated to the main flow, when the alternative flows should also have been included.

The main flow for manage books and magazines (case remove) include the optional steps for filtering, when the user stories taken in that sprint did not include that. This mistake was discovered during the design of these use cases and, therefore, it was not included either in the design or the implementation.

Despite all the mentioned problems, the analysis for the user stories that were taken in each sprint is completed successfully, and it has been the blueprint for the further design and implementation. There are some differences between the design and analysis for example the omission of the filtering option due to the mentioned analysis problems. Despite that, the design mostly follows analysis and is as well consistent with the implementation.

The result of the system is determined by the black box testing and white box testing developed. Even though the system reacts successfully based on the test cases, it does not retrieve the information of the genres in book from the database. This means that the Junit created cannot test if the genres are added correctly and in order to test it is necessary to look into the database.

To sum up all the critical priority user stories are implemented, two high priorities, and one low priority, them all successfully passing the tests elaborated. Library users does not have functionality in the system yet. However, the library manager can add and remove librarians and librarians themselves can add and remove books, magazines, and library users. Librarians as well can manage loans for library users with the exception of extending the loans.

7 Conclusions

The purpose of the conclusion section is to compile the results from each section in the report. What is the conclusion? Did the project fulfil the requirements? Etc.

You can only comment on report contents, no new topics or content can be introduced in this section.

The purpose of the project is to create a digital library system due to the lack of efficiency in the current physical one so it can ease the management of resources of the library. The needs of the client have been represented in a list of requirements including 22 user stories (following the SMART principles) and 9 non-functional requirements that lead to 11 uses cases and the activity and system sequence diagrams that comes with them. The analysis ends with the creation of a domain model, which is the blueprint of the following design. It especially influences the design of the database, which includes a EER diagram, a relational schema, and a Global Relations Diagram.

Trying to adjust to the SOLID principles, when possible, the design of the system follows a series of patterns, including MVVM, observer, singleton and adapter among others.

The implementation strictly follows the design and the black box testing and white box testing expose that the following user stories succeed (User stories: 2,5,4,21,1,6,20, 3). The library manager can add and remove librarians. Librarians can add and remove books, magazines and library users, and manage loans for library users with the exception of extending the loans. The system does not include any functionality for library users yet.

8 Project future

Reflecting from a technical point of view, there are some functionalities that could be implemented or modified to improve the project.

Starting with the current implementation and what could change, the view controller classes should only delegate to the view model, instead of implementing the functionalities as they currently do. Furthermore, the Remote exceptions that are thrown in the controller classes should be caught in the view model.

Moreover, a Remote observer could be implemented so when there are multiple clients using the system, all their actions are automatically updated to the others and login scenario is being analysed but further design and implementation is expected to be elaborated.

Regarding the missing functionality that could be included, all of our use cases should be analyzed, designed, implemented and tested. This includes the login, reservation, filtering, reviews, notifications and fines functionalities.

As an option for scalability, the system could, in the future, include other types of multimedia items apart from books and magazines, even if this possibility has not been considered yet.

9 Sources of information

Connolly, T. & Carolyn, . B., 2014. *Database Systems. A Practical Approach to Design, Implementation,*. s.l.:Pearson.

ElephantSQL, n.d. *ElephantSQL.* [Online]
Available at: https://www.elephantsql.com

JetBrains, 2022. *IntelliJ IDEA overview.* [Online]
Available at: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

JetBrains, 2022. *Introduction DataGrip.* [Online]
Available at: https://www.jetbrains.com/help/datagrip/meet-the-product.html

Larman, C., 2004. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development,*. 3 ed. s.l.:Pearson.

Martin, R. C., 2000. *Design Principles and Design Patterns.* [Online]
Available at: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf
[Accessed 27 5 2022].

Microsoft, 2012. *The MVVM Pattern.* [Online]
Available at: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10)
[Accessed 27 5 2022].

NADRA, n.d. *NADRA.* [Online]
Available at: https://www.nadra.gov.pk/identity/identity-cnic/
[Accessed 2 3 2022].

NISO, 2010. *Scientific and Technical Reports* -, Baltimore: National Information Standards Oganization.

Samad, S., 2019. *Daily Times.* [Online]
Available at: https://dailytimes.com.pk/374359/a-country-without-libraries/
[Accessed 23 2 2022].

Vaughan, D., n.d. *Britannica.* [Online]
Available at: https://www.britannica.com/story/a-brief-history-of-libraries
[Accessed 16 2 2022].

White, B., 2012. *Wipo Magazine.* [Online]
Available at: https://www.wipo.int/wipo_magazine/en/2012/04/article_0004.html
[Accessed 16 2 2022].

10 Appendices

Appendix A: /Apendices/Project_Description(Appendix A)

Appendix B: /Apendices/Analysis/Activity_Diagram(Appendix B)

Appendix C: /Apendices/Analysis/Domain_Model(Appendix C)

Appendix D: /Apendices/Analysis/Use_Case_Diagram(Appendix D)

Appendix E: /Apendices/Analysis/Test_Cases(Appendix E)

Appendix F: /Apendices/Analysis/Use_Cases(Appendix F)

Appendix G: /Apendices/Design/Class_Diagram(Appendix G)

Appendix H: /Apendices/Design/Sequence_Diagrams(Appendix H)

Appendix I: /Apendices/Implementation_and_Testing/Junit(Appendix I)

Appendix J: /Apendices/User_Guide(Appendix J)

Appendix K: /Apendices/ Implementation_and_Testing/Library(Appendix K)

Appendix L: /Apendices/DataBase/Logical_Model(Appendix L)

Appendix M: /Apendices/ DataBase/Global_Relational_Diagram(Appendix M)

Appendix N: /Apendices/ DataBase/EER_Diagram(Appendix N)