UNIVERSITY OF PISA

Large Scale and Multi-Structured Databases

Year 2019/20

BibliOS

A simple software system for managing a library

GERARDO ALVARO

MARCO BONGIOVANNI

RICCARDO POLINI

GIULIO SILVESTRI

INDEX

1. INTRODUCTION

BibliOS is a stand-alone software application designed as a support for public and/or private libraries. With its built-in functions for registering users, borrowing and/or returning books, it aims to be an easy-to-use tool for libraries in order to improve and speed-up their management.

The idea is that a library will have a room with a certain number of computers called *totems*. Totems can be either traditional PCs or touch screen displays. A Customer entering the library will use the totem to check the availability of a desired book and request to borrow it.

A numbered ticket will be printed and the customer may move to the librarian's desk. The librarian will see the entry for a certain book request and, if it matches the number on the customer's receipt, he/she will confirm the request and provide the physical copy of the book.

The same process will be replicated in case of a book return. This time, the loan will be closed only once the librarian has confirmed that the book was physically returned.

This solution is similar to what has been implemented in many *McDonald's* restaurants.

2. REQUIREMENTS ANALYSIS

2.1. Application Actors

The actors of the application are the *Librarian* and the C*ustomer*.

The first one maintains the Catalogue of available books in the library with their details, approves loan and return requests for specific books and records new customers.

The second one can browse the catalogue in the library to check a book's availability and request its loan.

2.2. Functional and Non-Functional Requirements

The *functional* requirements of this application, divided with respect to the two actors, are as follows:

- The customers can browse the catalogue of books in the library.
- The customers can verify the availability of a certain book.
- The customers can request to borrow a certain book.
- The customers can request to return a previously borrowed book.
- The customer can cancel an undesired loan request.

- The librarian can add new customers to the application.
- The librarian can add/remove new books or copies to the catalogue.
- The librarian can confirm borrow/return requests.
- The librarian can shut down the application.

The *non-functional* requirements of the application are:

- The application's interface must be user-friendly.
- The application must have a low response time.
- The application will store information in a relational Database (SQL).
- The application must guarantee data consistency.
- The application must be reliable: no system crashes, exceptions are handled etc.

3. UML DIAGRAMS

3.1. Use-Case Diagram

See the linked PDF file for an high-resolution version of the diagram.

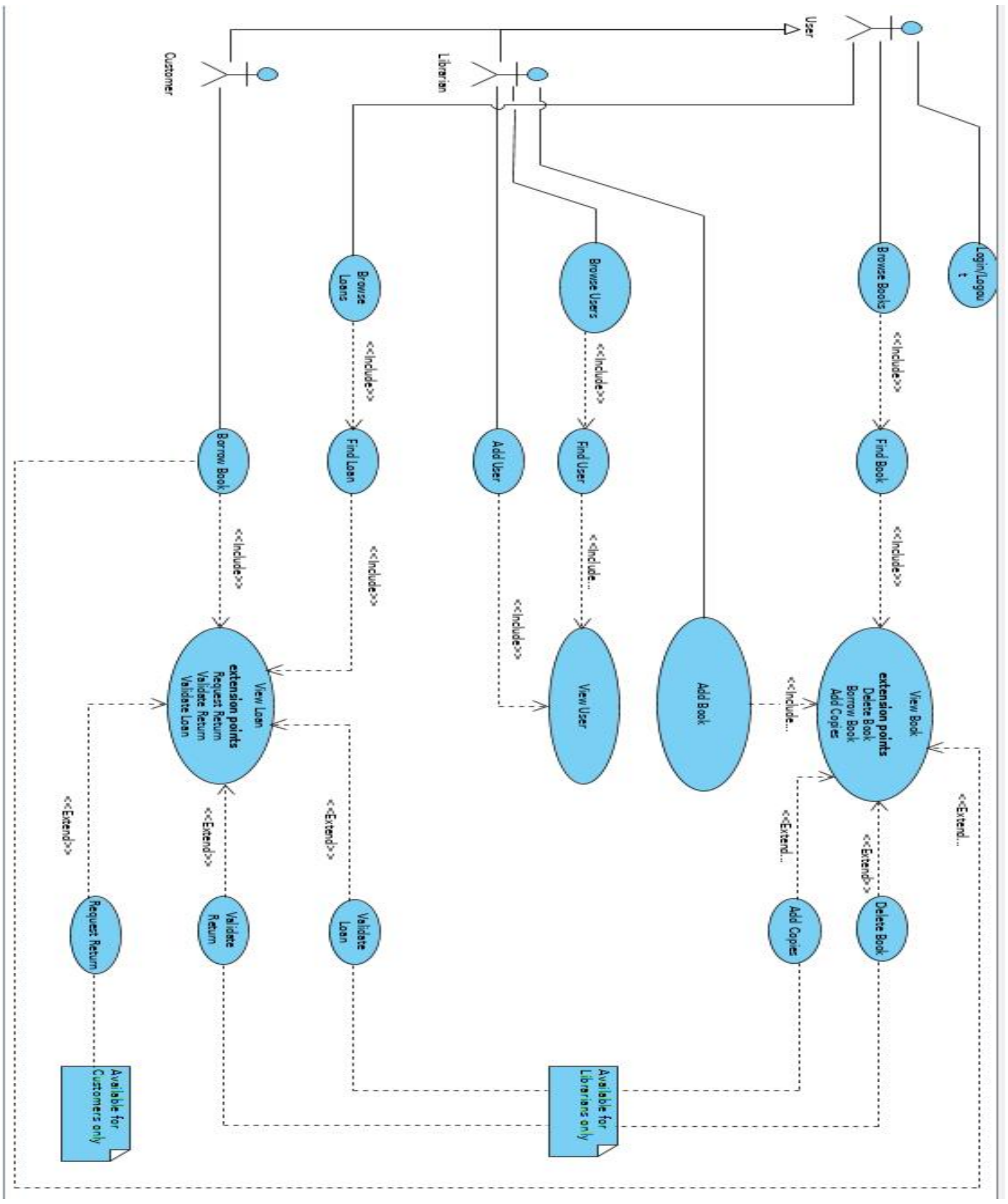


Figure 1: Use-Case Diagram

In the above figure is reported the Use-Case diagram in which we can see: the actors of the application, librarian and customer generalized into user; the use cases and some notes to specify when a use case is available only to the librarian or only to the customer (if it is not already obvious from the diagram).

3.2. Class Diagram

In *Figure 2* are reported the main entities of the application and the relationships among them. Librarian and Customer are generalized into User; a customer can request one or several book loans (or none at all), obviously each loan is made by a single customer; a book can be lent (as long as there are available copies) or not, and obviously a single loan is of just one book.

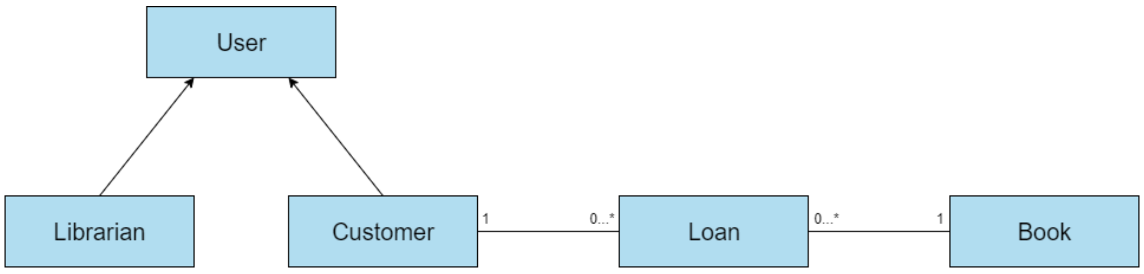


Figure 2: Class Diagram

4. ENTITIES AND DATABASE ORGANIZATION

4.1. Main Entities

The main entities involved in the application are:

- user (idUser, name, surname, privilege)
- book (ISBN, title, author, numCopies, category)
- loan (ISBN, idUser, status)

4.2. E-R Diagram of the DB

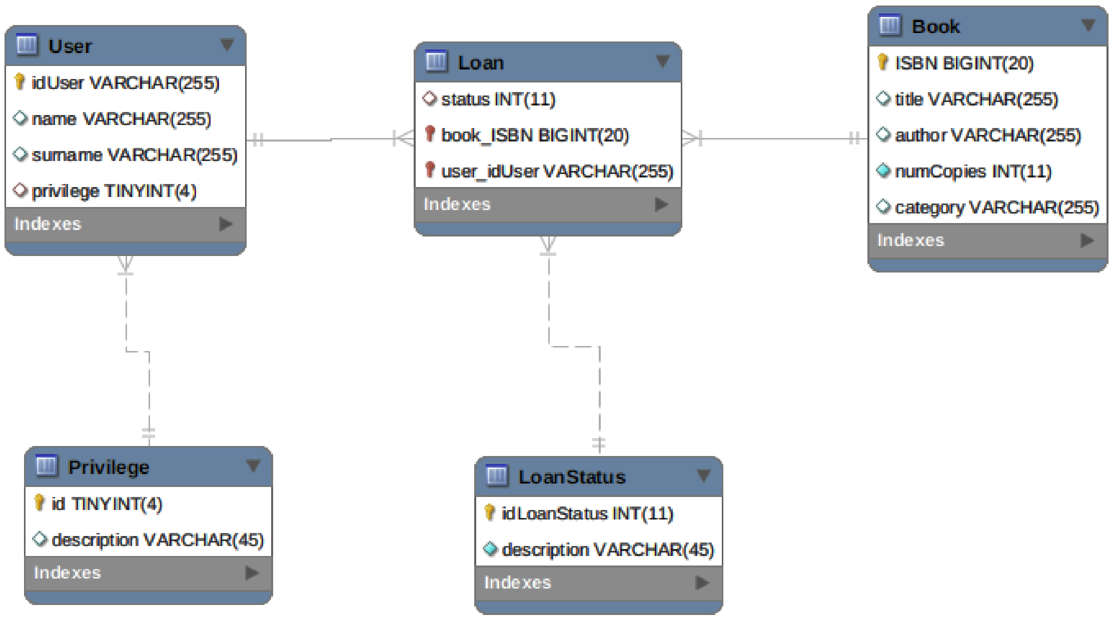


Figure 3: E-R Diagram

The tables *privilege* and *loanstatus* have been inserted to describe enum attributes of their parent entities. A similar approach should have been followed if we wanted to implement book category filter.

The former clarifies whether a user is a customer (id=0) or a librarian (id=1), the latter specifies a meaning for the three possible values of the attribute *status* in the *loan* table:

- **0 (requested)**: a customer requested a book and is waiting to receive it.
- **1 (granted)**: the librarian confirmed the loan request and delivered the book to the customer.
- **2 (returning)**: a customer returned a book and is waiting for librarian approval.

Please note that after the librarian approval for the book return, the loan ends and the corresponding tuple in the *loan* table is deleted.

5. SOFTWARE ARCHITECTURE

BibliOS has a Client-side and a Server-side.

The Client-side is composed of the GUI implemented in *javafx* and *FXML* which interacts with the user (Customer or Librarian), and by the various Interface Controllers.

The Server-side is composed by the Hibernate Entity Manager and, of course, by the Database.

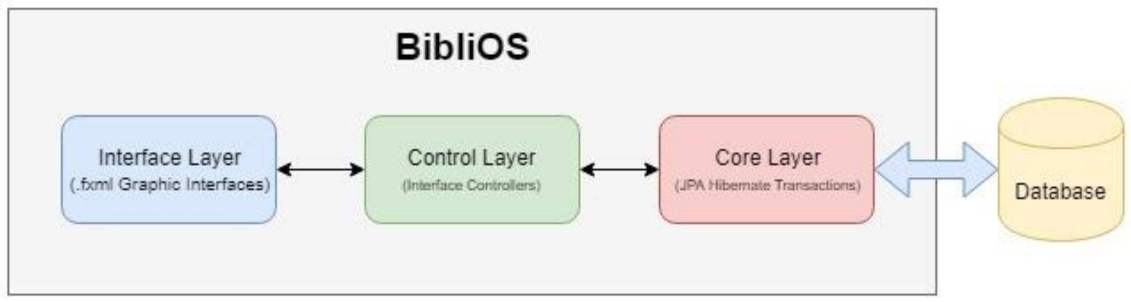


Figure 4: Software architecture of BibliOS

5.1. Repository

BibliOS is a *Maven* project. The project repository is organized as follows:

- **`./`** : contains the *makefile* and the *POM* file used to generate the maven dependencies and build the project.
- **`/src`**: contains the main source files of the application, more specifically:
    - **`/src/main/java`**: contains the *.java* source codes.
    - **`/src/main/resources`**: contains the *.fxml* files used for the GUI and the application's image logo.
- **`/target`**: generated after the first compile process. Will contain the *.class* files, the libraries, drivers and the .jar executable.

6. INSTRUCTION MANUAL

Installation

After downloading *BibliOS* on a UNIX-like system, import the Database schema **`bibliosDB.sql`** in a MySQL Server application. After this operation, open the terminal inside the project folder.

If you do not already have *maven* installed, make sure you have a working internet connection and run the following command:

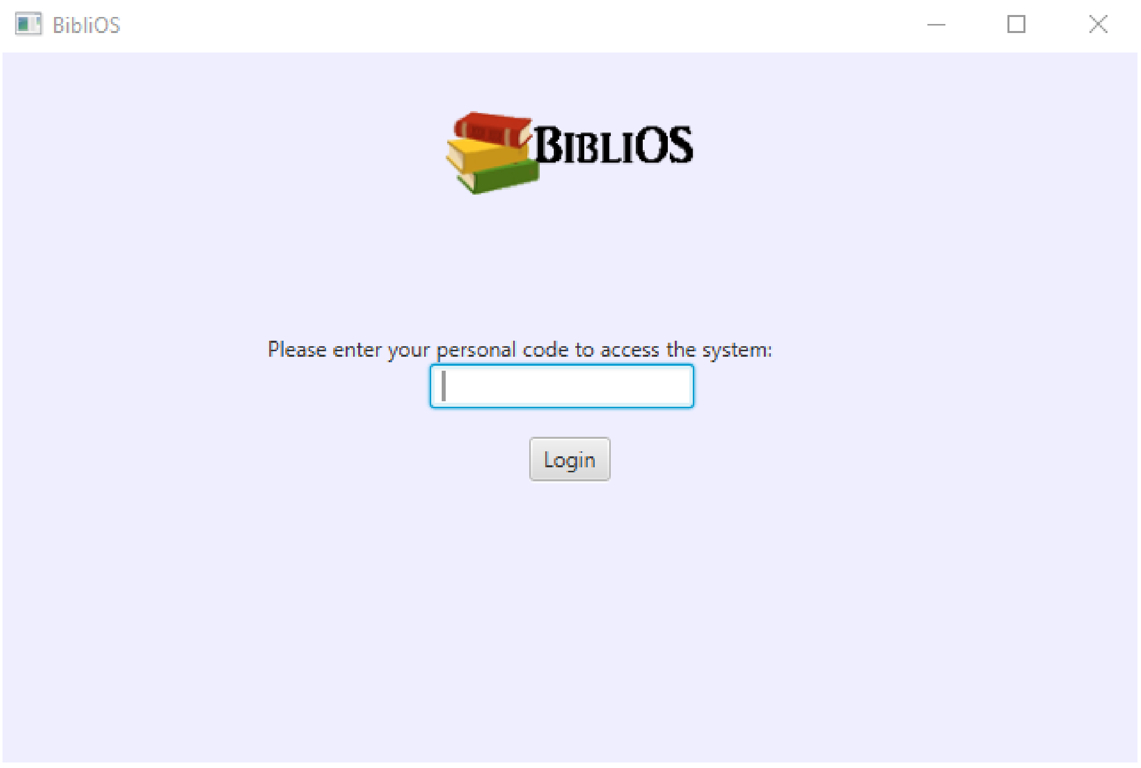
- **`sudo apt install maven`**

BibliOS requires **`java`** (version 11 or higher) and **`javafx`** (version 11 or higher) to run, install these if you do not already have them.

After installing all the necessary packages, you can run *BibliOS* by executing the following command:

- **`make`**

Login Interface



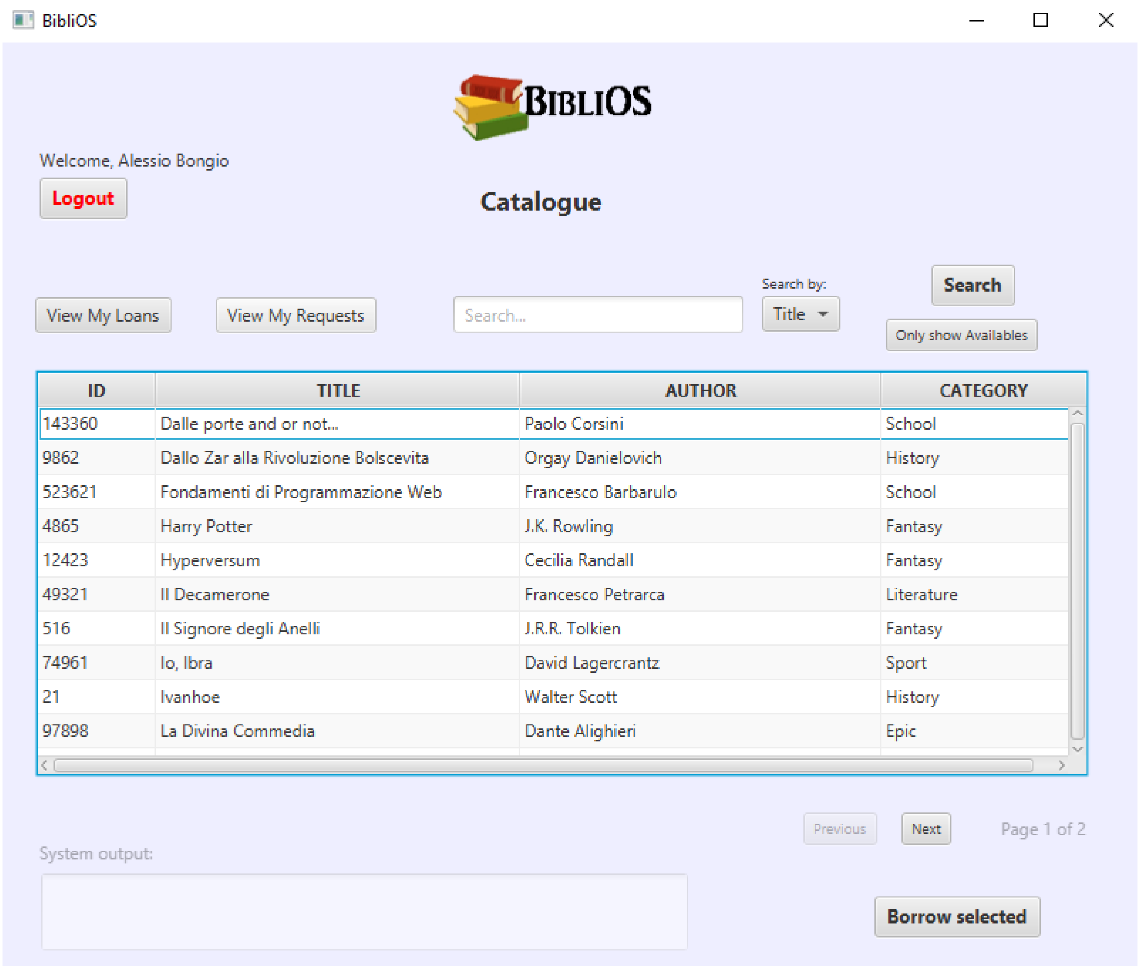
Please enter your personal code to access the system:

Figure 5: Login interface in BibliOS

The login screen of *BibliOS* contains:

- **Login field** where the user (customer or librarian) can insert his personal code to acces the main functionalities of *BibliOS*.
- **Login button** which redirects the user to the appropriate interface, checking if he is a customer or a librarian.

Customer Interface



ID	TITLE	AUTHOR	CATEGORY
143360	Dalle porte and or not...	Paolo Corsini	School
9862	Dallo Zar alla Rivoluzione Bolscevita	Orgay Danielovich	History
523621	Fondamenti di Programmazione Web	Francesco Barbarulo	School
4865	Harry Potter	J.K. Rowling	Fantasy
12423	Hyperversum	Cecilia Randall	Fantasy
49321	Il Decamerone	Francesco Petrarca	Literature
516	Il Signore degli Anelli	J.R.R. Tolkien	Fantasy
74961	Io, Ibra	David Lagercrantz	Sport
21	Ivanhoe	Walter Scott	History
97898	La Divina Commedia	Dante Alighieri	Epic

Figure 6: Customer interface in BibliOS

The customers of *BibliOS* will have available only a single interface in which there are:

- **Logout** button used to return in the login screen.
- **Search field** in which the customer can search for a certain book.
- **Searched By**: the book search will be exploit basing on the attribute selected in this menu (Title or Author).
- **Search** button starts the book search.
- **Books table** which contains all the books in the catalogue (ID, Title, Author, Category).

- **View My Loans** which shows a table which contains the books the customer borrowed recently.
- **View My Request** which shows a table which contains the books the customer requested to borrow.
- **Only Show Availables** which filter the books to show only the available ones.
- **Next/Previous Page**: buttons which scroll the table pages containing the books information.
- **System Output** Area used to show system messages to the customer.
- **Borrow Selected**: the customer can use this button to request the loan of the selected book.

Librarian Menu Interface

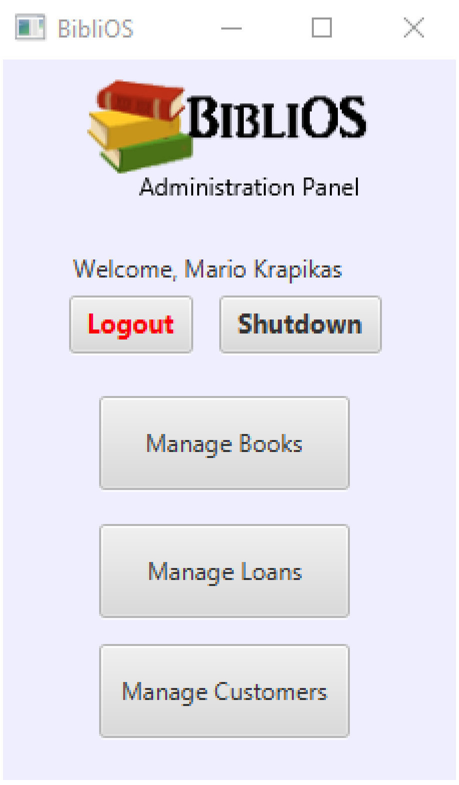
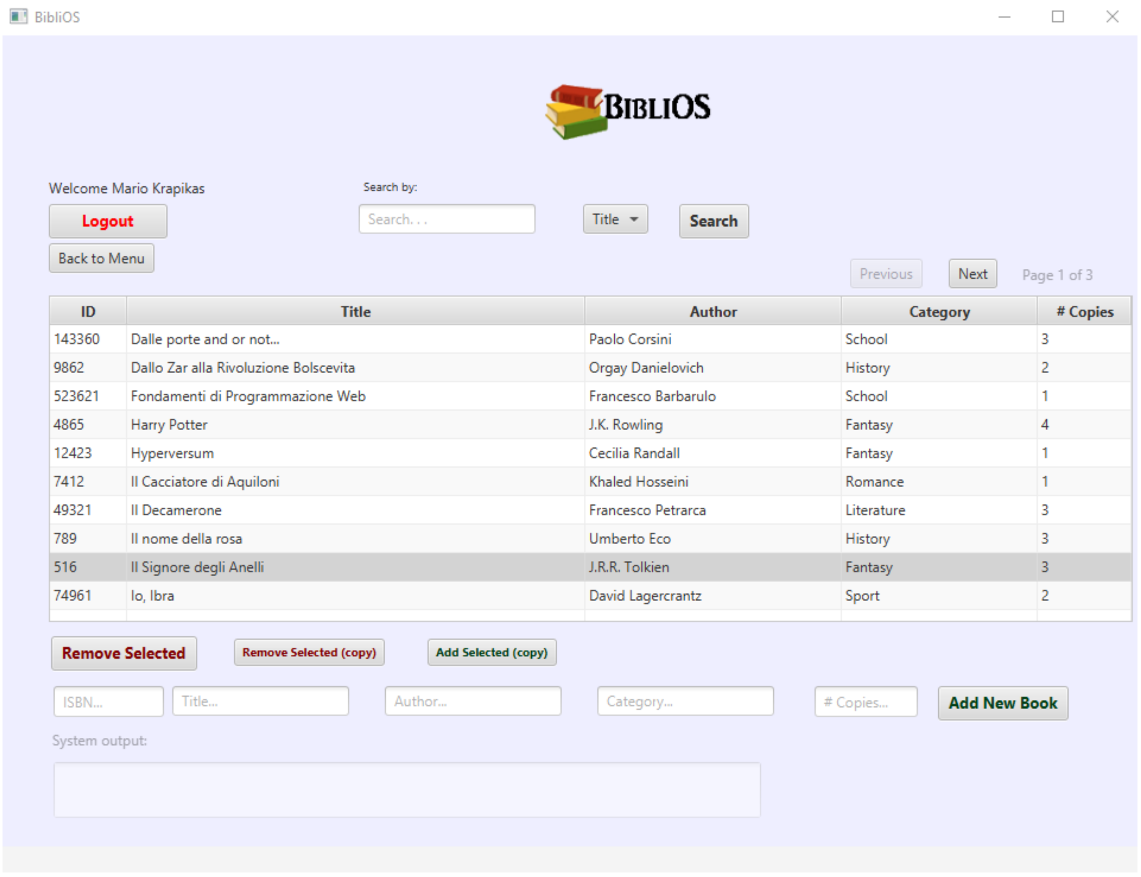


Figure 7: Librarian administration panel in BibliOS

The librarians using *BibliOS*, after the login, will find an administration panel containing:

- **Shutdown**: button used to shutdown the entire application.
- **Manage Books**: it redirects the librarian to the books interface.
- **Manage Loans**: it redirects the librarian to the loans interface.
- **Manage Customers**: it redirects the librarian to the customers interface.

Librarian Books Interface



ID	Title	Author	Category	# Copies
143360	Dalle porte and or not...	Paolo Corsini	School	3
9862	Dallo Zar alla Rivoluzione Bolscevita	Orgay Danielovich	History	2
523621	Fondamenti di Programmazione Web	Francesco Barbarulo	School	1
4865	Harry Potter	J.K. Rowling	Fantasy	4
12423	Hyperversum	Cecilia Randall	Fantasy	1
7412	Il Cacciatore di Aquiloni	Khaled Hosseini	Romance	1
49321	Il Decamerone	Francesco Petrarca	Literature	3
789	Il nome della rosa	Umberto Eco	History	3
516	Il Signore degli Anelli	J.R.R. Tolkien	Fantasy	3
74961	Io, Ibra	David Lagercrantz	Sport	2

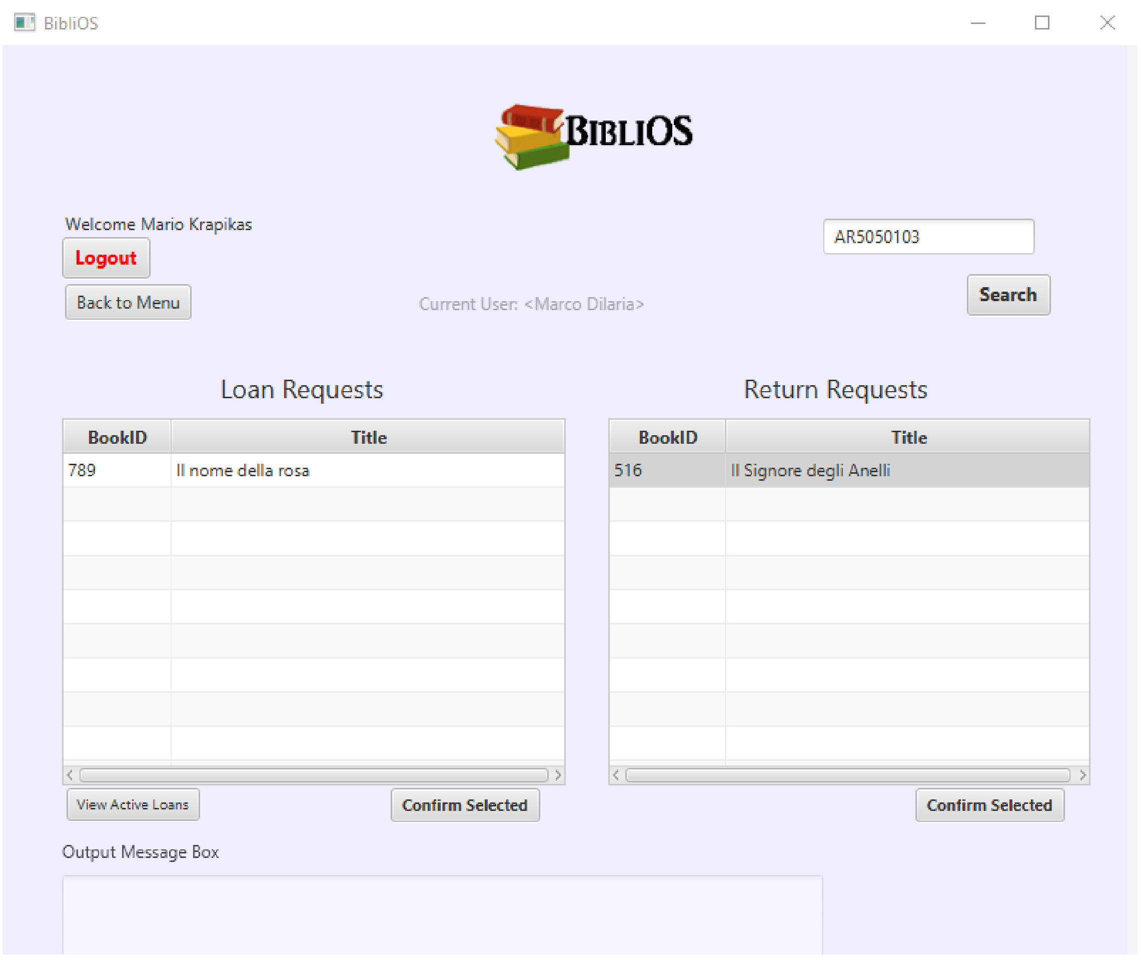
Figure 8: Librarian interface to manage books in BibliOS

The books interface for librarians in *BibliOS* contains:

- **Logout** button used to return in the login screen.
- **Back to Menu** button used to return to the main librarian screen.
- **Search field** in which the librarian can search for a certain book.
- **Searched By**: the book search will be exploit basing on the attribute selected in this menu.
- **Search** button starts the book search.
- **Books table** which contains all the books in the catalogue (ID, Title, Author, Category, Number of Copies).
- **Next/Previous Page**: buttons which scrolls the table pages containing the books information.
- **Remove Selected** removes all the copies of the selected book from the catalogue.

- **Remove Selected(Copy)** removes one of the copies of the selected book from the catalogue.
- **Add Selected(Copy)** adds one of the copies of the selected book from the catalogue.
- **Five text fields** (ISBN, Title, Author, Category, # Copies) where librarian can insert the informations about a new book he wants to add to the catalogue.
- **Add New Book** adds a new book basing on the informations inserted in the previous fields.
- **System Output** Area used to show system messages to the librarian.

Librarian Loans Interface



AR5050103

Loan Requests

BookID	Title
789	Il nome della rosa

Return Requests

BookID	Title
516	Il Signore degli Anelli

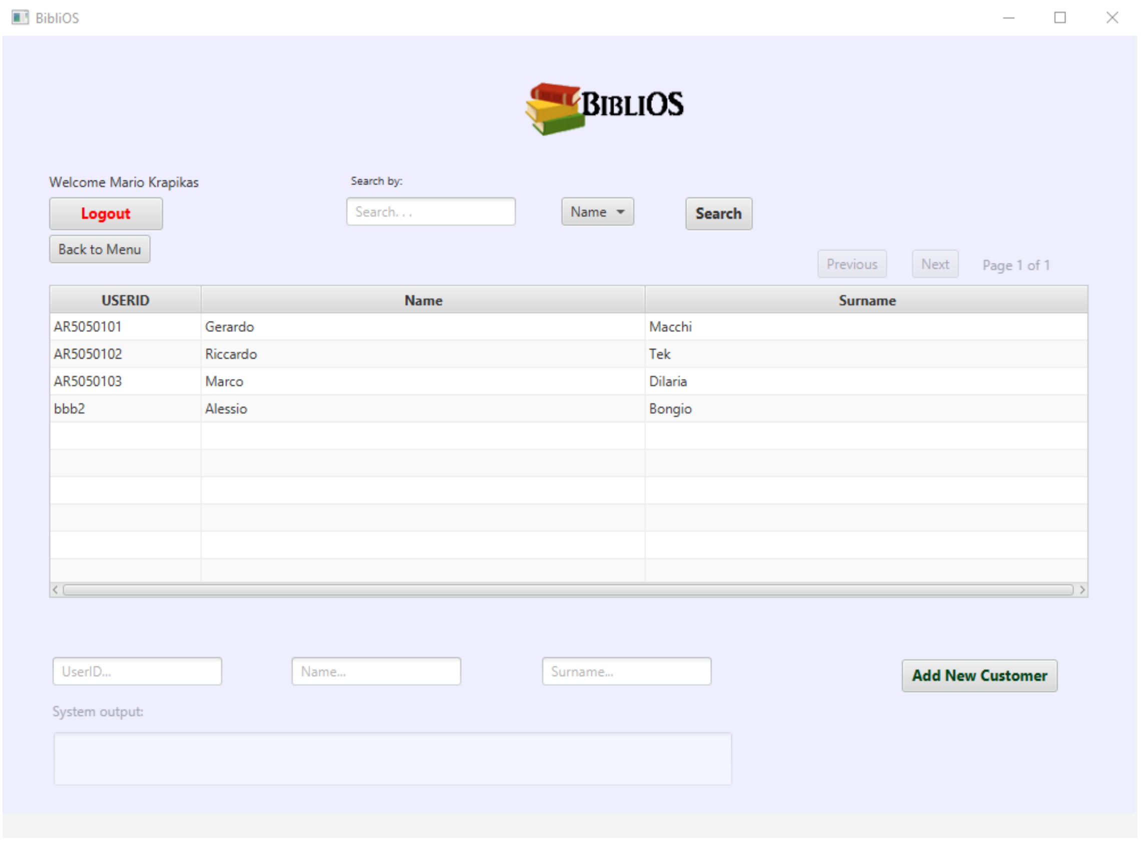
Figure 9: Librarian interface to manage loans in BibliOS

The loans interface for librarians in BibliOS contains:

- **Logout** button used to return in the login screen.
- **Back to Menu** button used to return to the main librarian screen.
- **Search field** in which the librarian can use to search the loan/return requests for a certain customer.
- **Search** button starts the requests search.

- **Loan Requests** table which contains all the books requested by the searched customer (BookID, Title).
- **View Active Loans** shows the table which contains the ongoing loans regarding the searched customer (BookID, Title).
- **Return Requests table** which contains all the books the searched customer wants to return to the library (BookID, Title).
- **The first Confirm Selected** confirms the selected loan request regarding the searched customer.
- **The second Confirm Selected** confirms the selected return request regarding the searched customer.
- **System Output** Area used to show system messages to the librarian.

Librarian Customers Interface



USERID	Name	Surname
AR5050101	Gerardo	Macchi
AR5050102	Riccardo	Tek
AR5050103	Marco	Dilaria
bbb2	Alessio	Bongio

Figure 10: Librarian interface to manage customers in BibliOS

The customers interface for librarians in BibliOS contains:

- **Logout** button used to return in the login screen.
- **Back to Menu** button used to return to the main librarian screen.
- **Search field** in which the librarian can use to search for a certain customer.
- **Searched By**: the customer search will be exploit basing on the attribute selected in this menu (Name or Surname).

- **Search** button starts the customer search.
- **User table** which contains all the customers using the application (UserID, Name, Surname).
- **Next/Previous Page**: buttons which scroll the table pages containing the customers information.
- **Threes text fields** (UserID, Name, Surname) where librarian can insert the informations about a new customer he wants to add to the application.
- **Add New Customer** adds a new customer basing on the informations inserted in the previous fields.
- **System Output Area** used to show system messages to the librarian.

7. MOVING TO A KEY-VALUE MODEL

7.1. Conversion Study

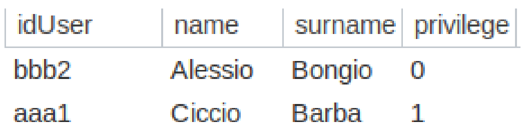
A key-value store is a database that uses an array of keys where each key is associated with only one value in a collection. It can be useful to move from a relational model to a key-value one if the application is *performance-driven* and data organization and management is less important.

The first thing that needs to be assessed is to find out how to represent tables in a key-value model.

In general, any Relational Database Model (RDBM) table can be represented in a key-value schema as follows:

```
$table_name:$primary_key_value:$attribute_name = $value
```

Let's consider, for example, just few tuples of table *user* of BibliOS' database.



idUser	name	surname	privilege
bbb2	Alessio	Bongio	0
aaa1	Ciccio	Barba	1

Figure 4: 'user' table example

For this table, the key-value schema can be defined as:

```
user:bbb2:name = "Alessio"

user:bbb2:surname = "Bongio"

user:bbb2:privilege = "0"

user:aaa1:name = "Ciccio"

user:aaa1:surname = "Barba"

user:aaa1:privilege = "1"
```

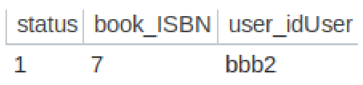
At this point, all the reported data from the table *user* is stored in a key-value form. Obviously, in our scenario, the same reasoning can be replicated for the table *book*.

Now let's move on to see how relationships are represented in a key-value schema.

In this model, foreign keys can be used as an identifier of the key to represent relationships. In order to define the key configuration, we have to include one more additional identifiers for each foreign key, so it can be represented as follows:

```
$table_name:$primary_key_value:$foreign_key_value:$attribute_name = $value
```

Taking into account a tuple from the table *loan* where the attributes *book_ISBN* and *user_idUser* are the foreign keys:



status	book_ISBN	user_idUser
1	7	bbb2

Figure 5: 'loan' table example

In a key-value model, this corresponds to:

```
loan:7:bbb2:status = "1"
```

Please note that in table *loan* a tuple is uniquely identified by the pair of foreign keys (book_ISBN, user_idUser).

We can store all these in the database using *put* operations and considering that *levelDB* stores keys in a lexicographical order.

The last thing to analyse if we want to move to a key-value schema is how can we perform query operations, considering that the only provided operations are:

- To retrieve a value by key.
- To set a value by key.
- To delete values by key.

So, for example, if we want to *select* users with a certain privilege, it's not possible to execute a simple query like:

```
SELECT idUser FROM user WHERE privilege = "?";
```

to retrieve required data, but we have to implement a separate Java method to achieve this functionality.

```java
public List<Integer> getidUserList(String attribute, String value) {
        List<Integer> userIDs = new ArrayList();

        DBIterator keyIterator = levelDBStore.iterator();
        keyIterator.seek(bytes("user")); // moves the iterator to the keys starting with "user"

        try {
            while (keyIterator.hasNext()) {

                String key = asString(keyIterator.peekNext().getKey()); // key arrangement : user:$idUser:$attribute_name = $value
                String[] keySplit = key.split(":"); // split the key

                int userID = Integer.parseInt(keySplit[1]);

                if (keySplit[keySplit.length - 1].equals(attribute)) { // check the attribute
                    String storedValue = asString(levelDBStore.get(bytes(key)));

                    if(storedValue.equals(value)){ // check the value
                        userIDs.add(userID); // if both checks are valid, user id is added
                    }
                }

                if (!keySplit[0].equals("user")) { // breaking condition : prefix is not "user"
                    break;
                }

                keyIterator.next();
            }
        } finally {
            keyIterator.close();
        }
        return userIDs; // return resulted user ids
    }
```

Code 1: retrieve data method

In the above method we use a *LevelDB* iterator to go through the keys with prefix "user" and select their *id* if the condition is matched.

In a similar way we can implement any method to retrieve data according to the requirement. Even though this operation seems much more complicated than executing a SQL query, in a well-defined key-value schema the performance is guaranteed to be higher due to fast reads/writes operations in key-value stores.

7.2. Feasibility of a Key-Value Model for BibliOS

Obviously moving to a key-value database will have some strengths and weaknesses that can be summarized as follows:

PROS:

- Increases application performances.
- Highly optimized solution for scalability.

CONS:

- More complex data modelling.
- Data management is more expensive.

Considering the previously discussed factors, we can conclude that in the case of this application, moving to a Key-Value model is not convenient.

Our application is a business-model therefore we need well organized data. Moreover, we are not interested in scalability since a Library will continue to operate in the same manner (more or less) in the foreseeable future.

We are not interested in having maximum performance since BibliOS will have, at most, tens of users operating simultaneously, not hundreds or thousands.

In our application, we often have to perform queries such as: "retrieve the loans of a specific user with a certain status". These kind of operations are very costly in Key-Value Databases since they require indexing.

In the end, BibliOS is not a performance-driven application, but rather one for which data organization and management is very important. For this reason, and for the previously mentioned, we think that it's not worth to convert our Relational Model to a Key-Value Model.

8. CONCLUSIONS

The proposed application is only provided with the main features requested for this task however, it would be possible to implement other functionalities that a realistic context would otherwise require such as: extending a loan, knowing on which shelf a certain book is placed, notifications for expired loans etc.