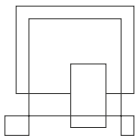


Library Client Server System

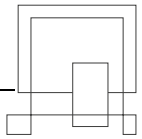
ICT Engineering

SEP2

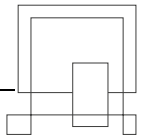
Date: 15 Dec. 2017

Supervisor:
Jens Cramer Alkjærsig
Troels Mortensen

Group:
Wangshu Xu, 260068
Shishir Sharma, 260073



Contents



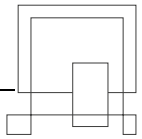
Abstract

A library always needs to deal with a large amount of the information from the reader and the book. Therefore, we started this project to develop a library operating system to provide online access to the library for users and librarians. Meanwhile, it also can manage the resource of users and librarians (included borrow information and return information).

Our system has two different operating interfaces to librarians and users. The operating system for the user is more focus on the function of borrow and return books. Another operating system is more focus on the function of the management of the user and books. For examples, it can insert and delete a book from the system.

At the beginning, we started with a scrum schedule. We distributed the work into three sprints. After that, we arranged a group meeting for each day we work. We did a small adjustment to our system by using the MVC model, it is useful to structure the implementation of the code and control function of the GUI.

We have met several challenges by connecting the GUI to the database. however, we managed to make it through.in the end, we finished this work and all the requirements have been fulfilled.



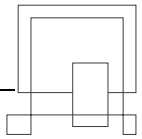
1 Introduction

Our system is developed to provide online access to the library. We have two different type of operator, the user and the administrator.

The requirement is divided into non-functional requirement and functional requirement which can be seen in page 5. After that, we did an analysis to our project. The main function for the user is to borrow a book, return a book, see book list, see customers list and the main function for the administrator is to insert a book, remove a book, insert a customer, delete a customer, see book list, see customer list. These two functions are showed to clarify the connection between classes, we made a Class diagram (3.3) and based on the class diagram we made a Sequence diagram (3.4) to indicate the interaction within these classes.

To show the visual interface, we did a GUI design (4.1) according to the Class diagram. The details and descriptions of this part can be seen in the User Guide. In the implementation of the code part, it shows some code examples with explanation of classes (Remote Library Client, Remote Library Sever, Library Database). After we did the implementation, a Junit test (6) was made to check whether we had any bug in our code.

At the last, we had Result and Discussion to specify what we had done and what we hadn’t finished and what we learned during this project.



2 Requirements

The purpose of our project is to make a Library Client Server system. We enlisted some of the requirements and then sorted them in functional and non-functional requirement.

2.1 Functional Requirements

1. **The system should be able to run in two or more computers.**
2. **All the details of books, customers and borrowed book should be stored in the database.**
3. **Insert Book to the library system as librarian which includes book id, book name, author of the book, published date and stock of books in the library.**
4. **The librarian should be able to remove book as a client in the database with the book id.**
5. **The librarian should be able to register customer in the server which includes full name, student id, cpr number and validity.**
6. **The librarian should be able to remove a customer in the database by inserting the customers cpr.**
7. **The user should be able to borrow the book as a client from the server by inserting the book id and their cpr number.**
8. **The customer should be able return the book by inserting the book id and their cpr number.**
9. The librarian should be able to see the list of books in the database which consists of book id, book name, author, published date and the number of stock of the book.
10. The librarian should be able to see the list of customers in the database which consists of full name, student id, cpr and validity.
11. The customer should be able to see list of books that are in the library.
12. The customer should be able to see if they are registered in the system or not.
13. The customer should be able to see the list of books borrowed by other customers.
14. The system should print message if any action is performed in the system.
15. The customer can take more than one book from the library.
16. The customer can’t borrow a book if they are not registered in the system.

2.2 Non-Functional Requirement

1. The system should be made using SCRUM.
2. The system should be developed in JAVA.
3. The system should be developed in the three weeks of time.
4. The system should have a simple user interface which could be used by keyboard and mouse.

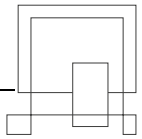
The requirement is furthermore described with some of the following diagrams,

3 Analysis

3.1 Use case

The different scenarios of our system can be explained more through the use case diagram. There are two user who can access our system, there is administrator who can insert book, delete book, add or insert user and delete the user and can see list of all the books and list of all customers. The other user are the customers who can borrow books from the library, return books and see list of books and check if they are registered in the system or not. The system has been made according to the diagram which is shown below, (The full Use case can be seen in Appendix 4)

Figure 1: Use-Case diagram.



3.2 Use Case Description

The use case description has been made to elaborate the use case diagram. It gives all the idea how each method in the system works. In the figure below, it is shown that how a user can borrow the book from the library server and how does the system work.

ITEM	VALUE
UseCase	Borrow Book
Summary	The customer borrows book.
Actor	User
Precondition	User wants to borrow a book from the library databse.
Postcondition	Book was borrowed. Book id anf the cpr number of the user was recorded in the dtabase.
Base Sequence	1. User enters the id of the book that they want to borrow. 2. User enters their cpr number. 3. The system checks if the book exists in the system. 4.The system checks if the user is registered in the system. 4. The system stores the information about the book borrowed and user.
Branch Sequence	
Exception Sequence	1-4. User terminates the system. 1-4. The server should be running. 2. The book and user should be registered in the database.
Sub UseCase	
Note	

Figure: Use-case Description

3.3 ER Diagram

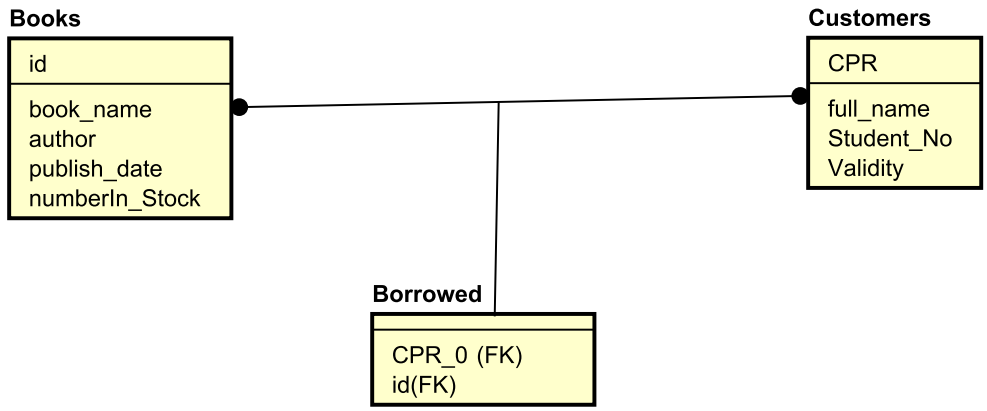


Figure 2: ER Diagram

The database in our system consists of books table and customers table. The books table and customers table are in many to many connections, so a borrowed table is created with the primary key of books and customers table. So, when customers are registered in the database they can borrow the book and when customers borrow book they are registered in the borrowed table. The figure below shows the implementation of borrowed table with the reference of books table and customers table.

```
public static void createBorrowed() throws Exception {
    try {
        Connection c = getConnection();
        PreparedStatement create = c
                .prepareStatement("CREATE TABLE IF NOT EXISTS \"library\".borrowed"
                        + "(id INTEGER REFERENCES \"library\".books "
                        + "(id), cpr INTEGER REFERENCES \"library\".customers (cpr));");
        create.executeUpdate();
    } catch (Exception e) {
        System.err.println(e.getClass().getName() + ": " + e.getMessage());
        System.exit(0);
    } finally {
        System.out.println("BORROW TABLE CREATED");
    }
```

Figure: Implementation of borrowed table

The figure below shows the implementation of books table which takes id of book, book name, author name, published date of the book and number of stock of the book.

```
public static void createBookTable() throws Exception {
    try {
        Connection c = getConnection();
        PreparedStatement create = c.prepareStatement("CREATE TABLE \"library\".books"
                + "(Id int NOT NULL, Book_Name VARCHAR(225) NOT NULL, Author VARCHAR(225) NOT NULL,
                + "publish_Date VARCHAR(20), numberin_stock int, PRIMARY KEY(Id))");
        create.executeUpdate();
    } catch (Exception e) {
        System.err.println(e.getClass().getName() + ": " + e.getMessage());
        System.exit(0);
    } finally {
        System.out.println("Books Table created");
    }
```

Figure: Implementation of books table

3.4 Class Diagram

The class diagram illustrates all the classes and the methods used in our system. To insert book in the library we have insert book method in the RemoteLibrary interface. The method insert book in RemoteLibrary is implemented by RemoteLibraryServer which adds the details of book in the database. The RemoteLibraryClient creates the instance of RemoteLibrary and creates insert book method which is called in the GUI and user insert the detail of the books.

In the figure below the classes which are coloured in yellow is the server part, the classes which are coloured in green in the client part whereas the classes coloured in blue colour are shared classes which are used by most of the classes in our system. The full class diagram can be seen in the Appendix 5.

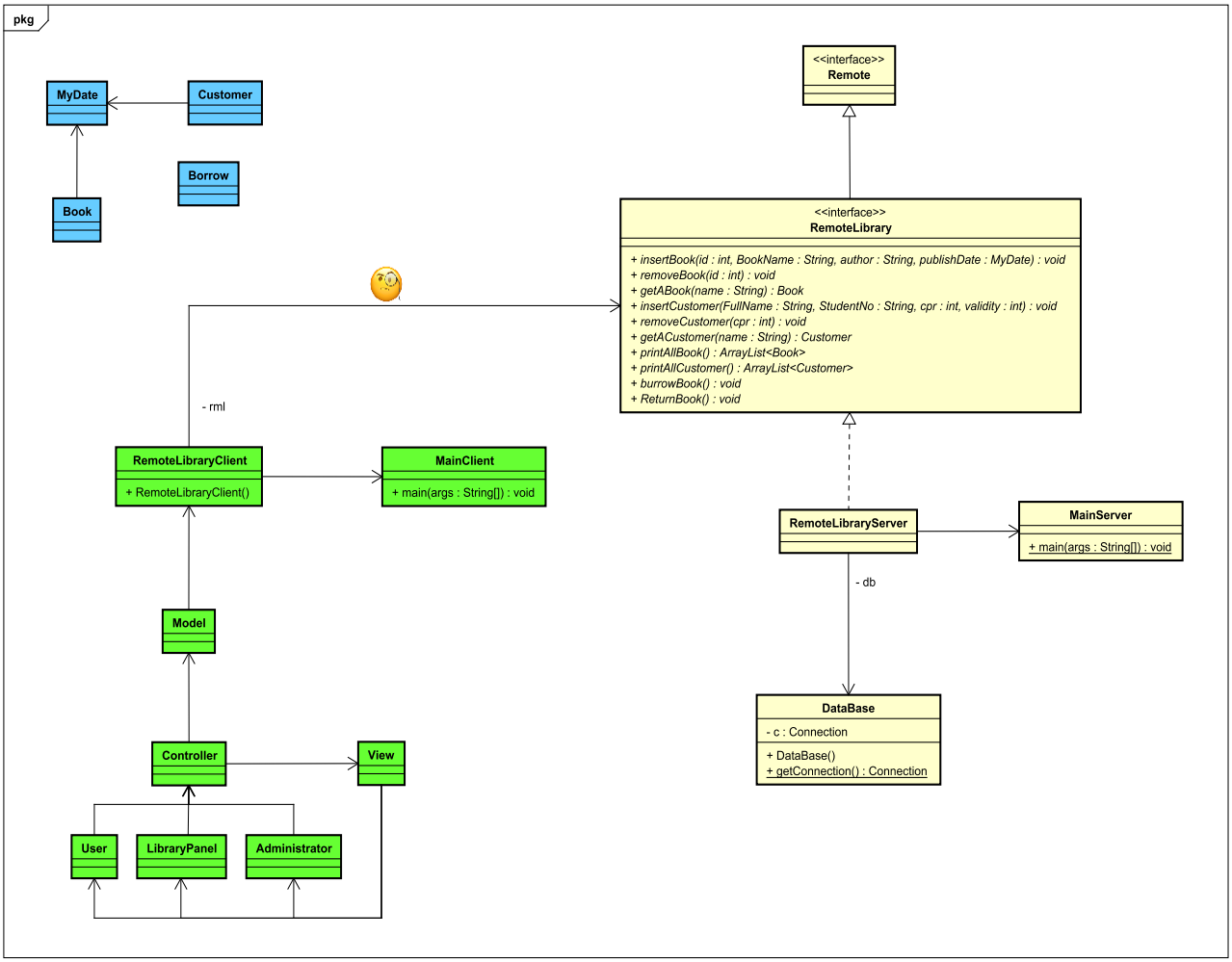
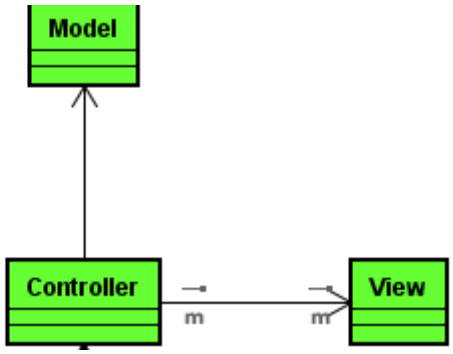


Figure: Class Diagram

4 Design

4.1 Model View Controller(MVC)

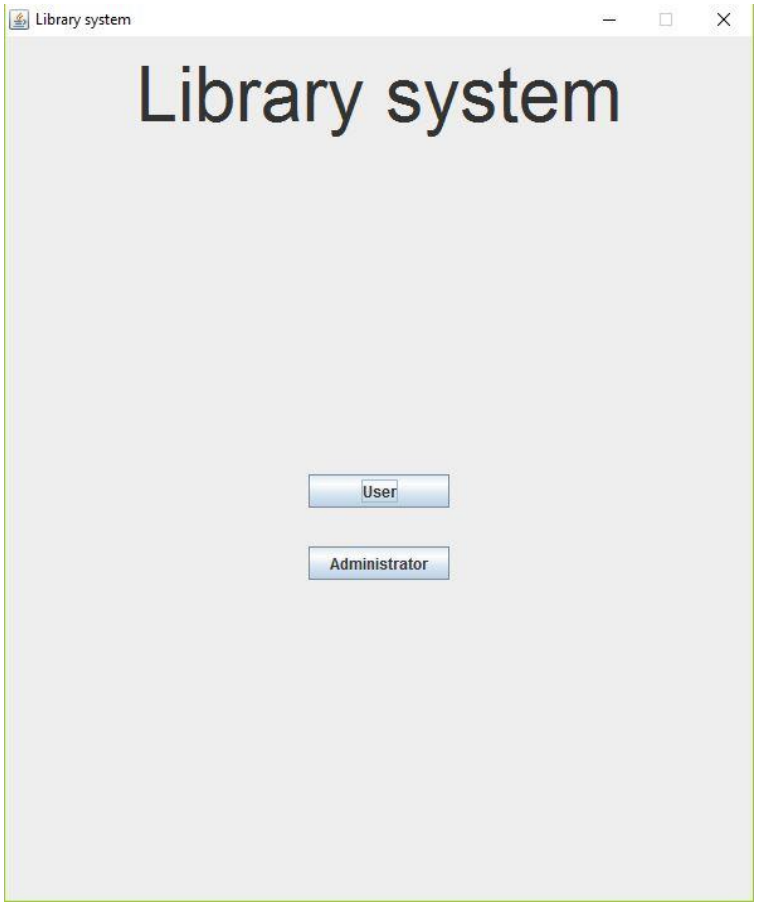
(Model-View-Controller) is a software architecture model in software engineering. The software system is divided into three basic parts: Model, View and Controller.
The purpose of MVC pattern is to realize a kind of dynamic program design, make subsequent modification and expansion of the program simplified, and make it possible to recycle some part of the program. In addition, this model simplifies the complexity and makes the program structure more intuitive. Software system through its own basic part of the separation also gives the basic functions of each part at the same time. The full MVC model can be seen in the class diagram Appendix 5.



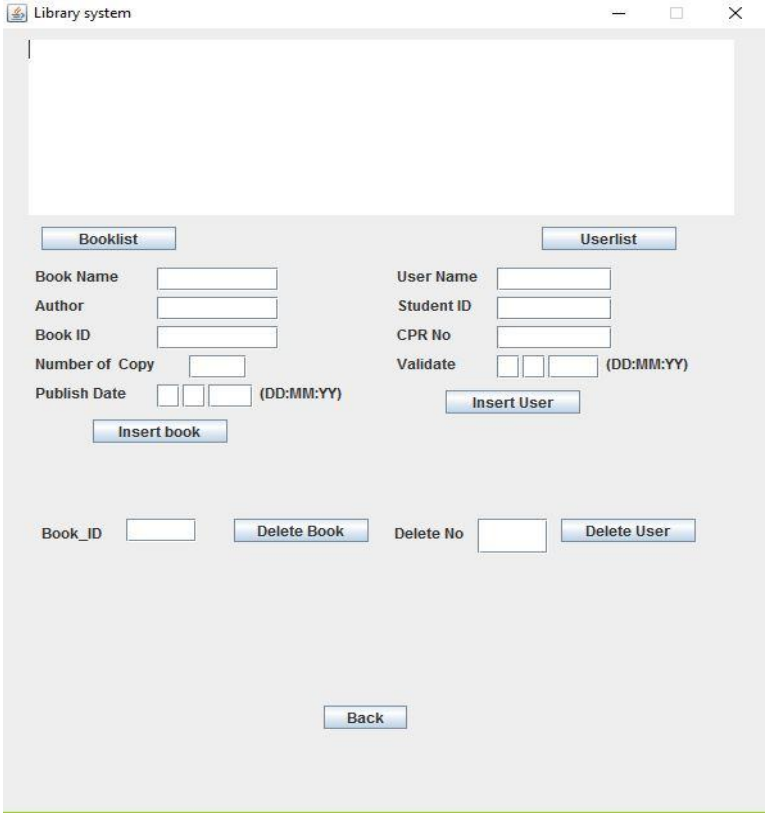
(Controller) - responsible for forwarding the request and the request processing. We used this Controller class to send the request to model class.it contains the main method of the user and administrator.
(View) - Interface designer for graphical interface designed. View class of our code can modify all the function of panels in our GUI design.
(Model) - we use model class to store the data onto the database. We use this MVC model to structure the code, try to split the JDBC code and GUI design code. The controller class is to ensure that the model class will update when the view part changed.

4.2 GUI Design

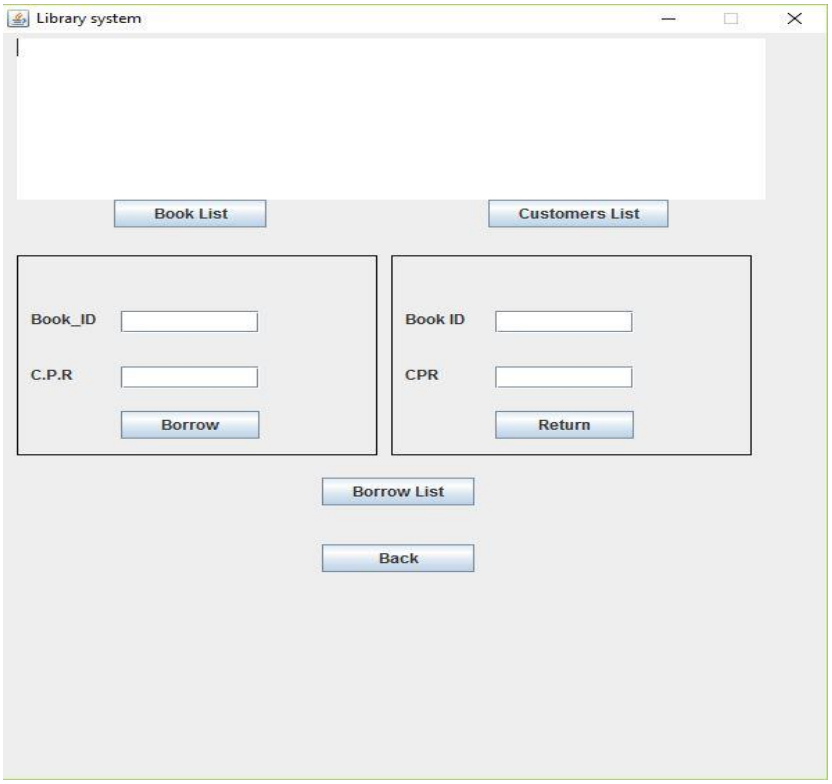
This is the main page of our GUI design, you can choose the different types of user in here. Administrator is for librarian.



The figure below shows the administrator panel, we can insert/delete a book or a customer in this panel.



The figure shows the user panel, we can borrow/return a book in here. We can also see the book list, borrow list and customer list in here.



The full user guide with the installation of the program can be seen in the appendix: 2

4.3 Class Diagram Design

The class diagram design gives us a brief idea about the design and how the system interacts in our system. The figure below shows the connection between client and server and connection of server and the database,

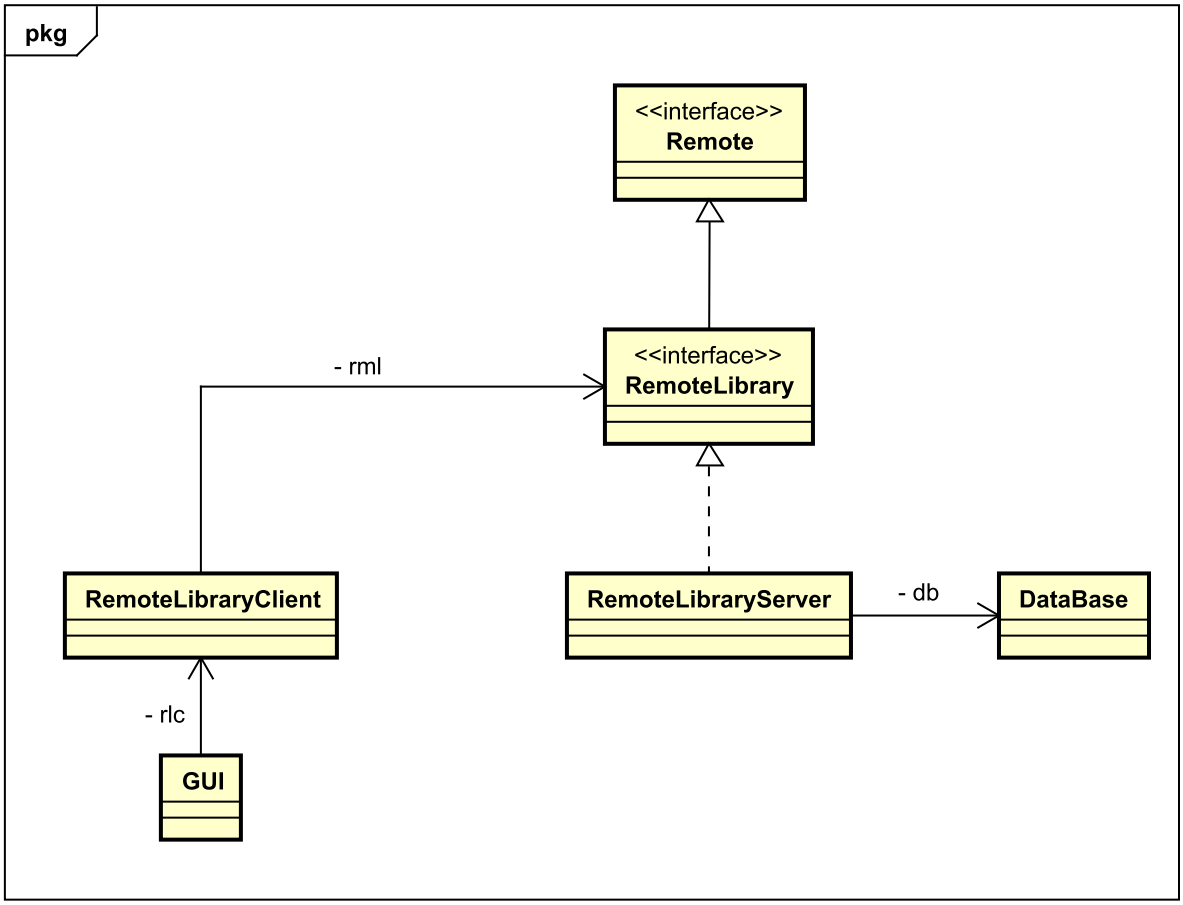


Figure: Design of class diagram

From the figure above, we can see that RemoteLibrary extends Remote interface, the RemoteLibrary interface consists of all the method i.e. insert book, delete book, see book, insert customer, delete customer etc. And these methods are implemented by Remote Library Server which stores all the data modified by the user in the database class. The client instantiates the RemoteLibrary class and calls all the required method from it. To make it easier for the user to interact with the system we have created GUI where the user inserts the required data and thus the information gets stored in the database through client and server.

The class diagram can be seen with all the methods in the appendix(Appendix 5) along with the detail view of GUI.

4.4 Sequence Diagram

The figure below shows the sequence diagram of insert book (The full Sequence diagram can be seen in Appendix 6.)

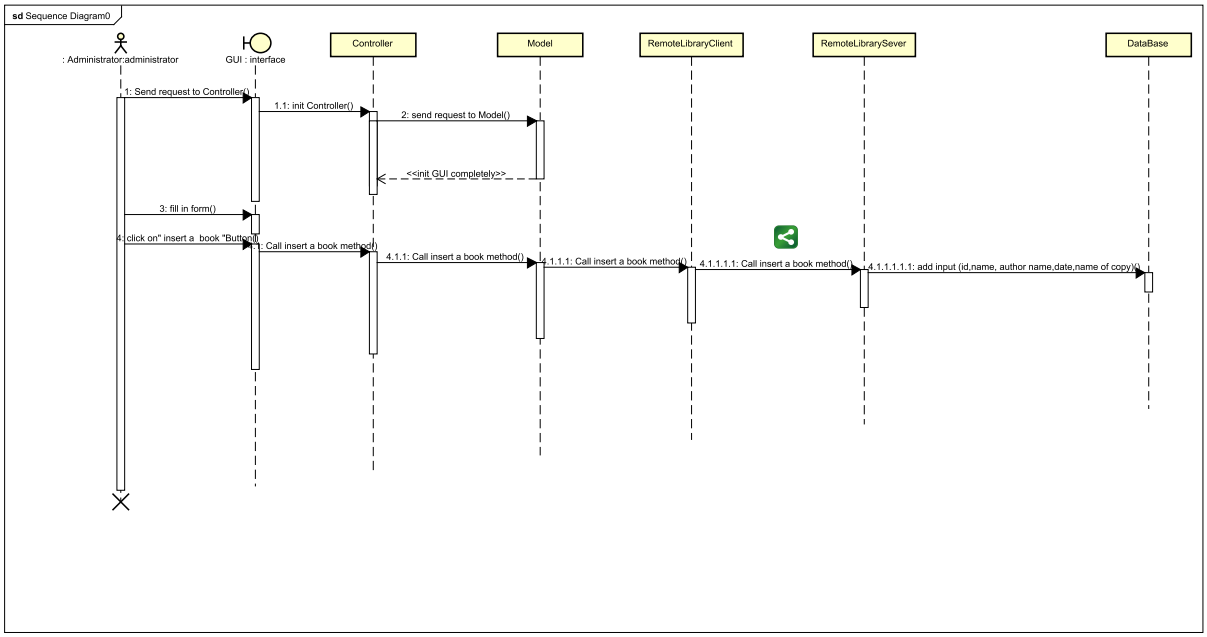
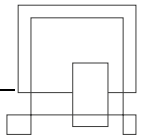


Figure: Sequence Diagram

The sequence diagram above shows how books are inserted in our system. To insert we should coordinate the system through administrator. The administrator runs client and gets a GUI panel. Then the administrator fills the form for insert book and when the administrator clicks on “Insert Book” button the GUI class calls insert book method from controller. The controller class calls insert book method from model, model class calls the Remote Library Client class insert book method, which calls method in the server, Remote Library Server class which then stores the data entered by administrator in the database.



5 Implementation

To join the client and server we have used Remote Method Invocation(RMI). The client runs the system and the insert data, these data are stored to the database through server.

5.1 Remote Library Client

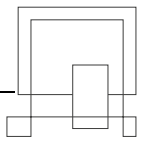
The remote library client instantiates (Remote Library) the server part and calls methods from the server, and from the main method we run the graphical user interface. In the figure below, we can see the implementation of RemoteLibraryClient, in the constructor we set path of the client to the server, then we have void method to insert book.

```
public RemoteLibraryClient() throws MalformedURLException, RemoteException,
        NotBoundException {
    rml = (RemoteLibrary) Naming.lookup("rmi://localhost:1099/hTl");
}

public void insert(int id, String BookName, String author,
        MyDate publishDate, int numbers) throws MalformedURLException,
        RemoteException, NotBoundException {
    rml.insertBook(id, BookName, author, publishDate, numbers);
}

public static void main(String[] args) throws MalformedURLException,
        RemoteException, NotBoundException {
    System.out.println("Client Connected");

    RemoteLibraryClient rlc = new RemoteLibraryClient();
    Model mdl = new Model(rlc);
    Controller ct = new Controller(mdl);
    View vi = new View(ct);
    vi.start(ct);
}
```

5.2Remote Library Server

The Remote Library Server implements Remote Library and inserts data in the database. In main method remote library server creates a registry with the port number 1099, and instantiate the server, as the type of interface. We put the server instance in the registry and assign it a name.

```
public class RemoteLibraryServer implements RemoteLibrary{

public RemoteLibraryServer() throws RemoteException {
    UnicastRemoteObject.exportObject(this, 0);
}

public void insertBook(int id, String BookName, String author,
        MyDate date, int numbers) throws RemoteException {
 db.insertbook(id, BookName, author, date, numbers);
```



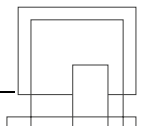
```
public static void main(String[] args) {
    try {
        LocateRegistry.createRegistry(1099);
        RemoteLibrary server = new RemoteLibraryServer();
        Naming.rebind("hTl", server);
        System.out.println("Server Started!");
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

5.3 Library Database

The Database class creates a connection between java and PostgreSQL database. The openConn() method helps to open the connection to the database and s.close() method is used to close the connection of the database.

```
private Connection c;

public DataBase() {
    c = getConnection();
}

public Connection getConnection() {
    Connection c = null;
    try {
        Class.forName("org.postgresql.Driver");
    } catch (Exception e) {
        System.err.println(e.getClass().getName() + ": " + e.getMessage());
        System.exit(0);
    }
    return c;
}

private void openConn() {
    try {
        c = DriverManager.getConnection(
                "jdbc:postgresql://localhost:5432/postgres", "USERNAME",
                "PASSWORD");
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

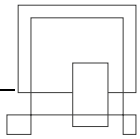
The figure shows a void method used to insert books to the table in database through the client. The insert book takes id, book name, author name, published date and number of stock of that book.



```
public void insertbook(int id, String BookName, String author,
        MyDate publishDate, int numbers) {
    openConn();
    Statement stmt = null;
    try {
        stmt = c.createStatement();
        String sql = "";

        sql = "INSERT INTO\"library\".books(id, Book_Name, Author, publish_Date, numberin_stock) VALUES ('"
                + id
                + "', '"
                + BookName
                + "','"
                + author
                + "','"
                + publishDate.toString() + "','" + numbers + "' );";
        stmt.executeUpdate(sql);
        stmt.close();
        c.close();
    } catch (Exception e) {
        e.printStackTrace();
        System.err.println(e.getClass().getName() + ": " + e.getMessage());
        System.exit(0);
    }
    System.out.println(BookName + " is now registered in the system");
```

The figure below shows return method of all the customers details. We created Customer class containing name, student number, cpr and validity of the membership in library. We created the array list of customers and returned all value received from database to the client.

```
public ArrayList<Customer> getAllCustomers() {
    openConn();
    try {
        PreparedStatement stmt = c
                .prepareStatement("SELECT * FROM \"library\".customers");
        ResultSet rs = stmt.executeQuery();
        ArrayList<Customer> customers = new ArrayList<>();

        while (rs.next()) {
            int cpr = Integer.parseInt(rs.getString("cpr"));
            String date = rs.getString("validity");
            String[] dates = date.split("/");
            int day = Integer.parseInt(dates[0]);
            int month = Integer.parseInt(dates[1]);
            int year = Integer.parseInt(dates[2]);
            MyDate valid = new MyDate(day, month, year);

            Customer customer = new Customer(rs.getString("full_name"),
                    rs.getString("student_no"), cpr, valid);

            customers.add(customer);
        }
        rs.close();
        stmt.close();
        c.close();
        return customers;
```



6 Test

The main purpose of test is to check if we fully fulfill the requirement that we come up with. For testing different methods in our system, we have used Junit test. Junit is a simple framework of writing reusable test sets, a subset of XUnit. There are three reasons that we use Junit test for our implementation of code.

1.The test framework can help us to purposely test the written program.it can also help us to minimize the bug in the code to ensure the correctness and stability of the system.
2.Many people just simply test the main method and then “sysout” output consoled observation when they are writing their own code. This is very boring and non-standard. The Disadvantages of this test method are the test method can’t be running together, the test results from the program own observations can determine the program logic.
3.Thanks to Junit's assertion mechanism, we can directly compare the results of our expectations and the results of the program to ensure that the predictability of the results
By using this Junit test, we have done a Junit test of our input method. We have inserted specific test method into different method of these classes to ensure that our input method can get the data that the user filled in.

6.1 Test Specifications

We take insert a book and insert a customer as our examples to show how we conduct Junit test in our code part. The full test code can be seen in Appendix 3.in order to test the method” insert a book”. Firstly, we create a testbook constructor and add it in @Before.it means the system will run this method before it run each sub method.
After that, we give the value to the testbook, (11,” new book”, “author”, date ,4) according to its type. Then, we use the assert Equals method to compare it with the expect data which will acquired by the original method. If the value we gave is not equal to the data we expect, it means our system is not work as our expectation. Therefore, we need to correct some part within our code.

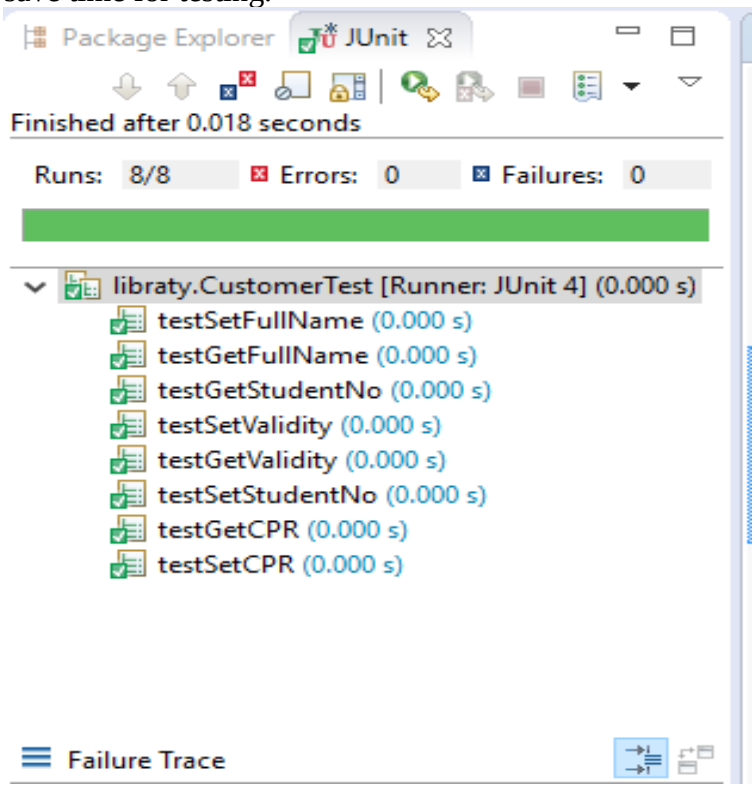
```
“
package library;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class BookTest
{  MyDate date ;
   private Book testbook;

   @Before
   public void setUp() throws Exception
   { date= new MyDate(11,12,2017);
   testbook = new Book(11,"newbook", "author", date, 4);
   }



   @Test
   public void testGetNumOfCopies()
   {
      assertEquals(4, testbook.getNumOfCopies());
      System.out.println("testGetNumOfCopies()");
```

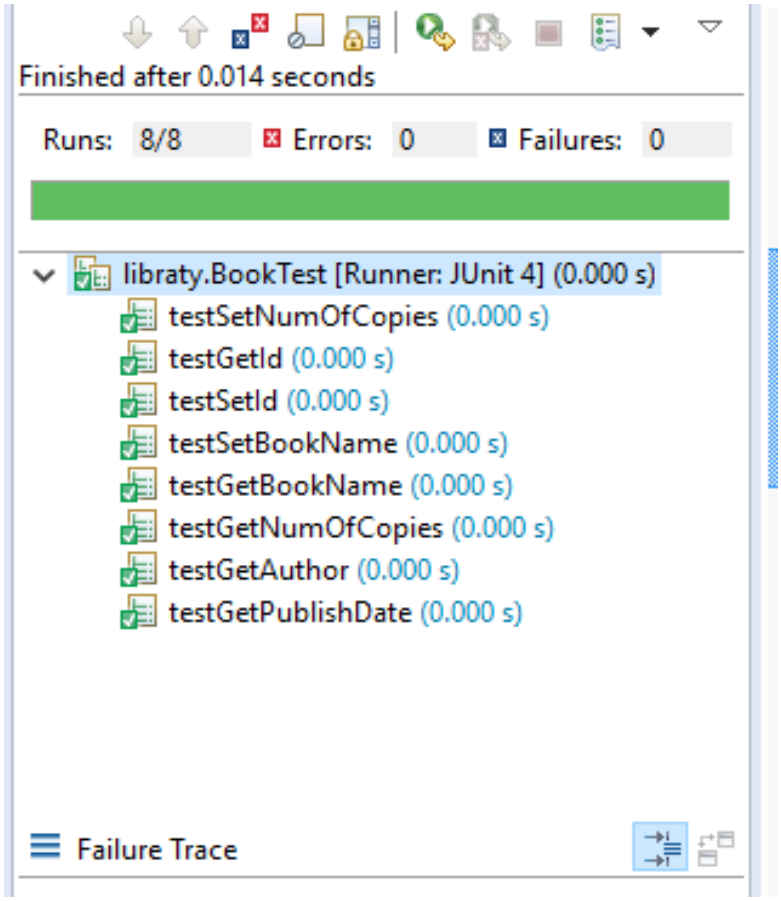
The test of function “insert a customer” is also done like that way, we first give the value to a dummy constructor “test customer” (“steven”,11,11223, date) according to its type. After that we use the assert Equals method to compare it with the expect data which will acquired by the method “insert a customer” by the customer class. If the value we gave is equal to the data we

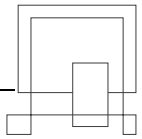
expect.it is obviously that the system is work under our expectation and these part of the code is fine.

```
public void setUp() throws Exception
{ date= new MyDate(12,1,2017);
   testcustomer = new Customer("steven", "11", 11123, date);
}
@Test(timeout = 1000)
public void testGetFullName()
{
   assertEquals("steven", testcustomer.getFullName());
   System.out.println("testGetFullName()");
}

@Test(timeout = 1000)
public void testSetFullName()
{
   String FullName ="steven";

   testcustomer.setFullName("steven");

   assertEquals(testcustomer.getFullName(), FullName);
}
@Test(timeout = 1000)
public void testGetStudentNo()
```

To prevent the situation “Dead loop” we have set a fixed running times at the beginning of each tested method. We run each of the methods three times to make sure that the result more fair and credible. The final test results show that our program works well without bugs. After this test, we found the Junit test help us a lot on testing the code and can obviously save time for testing.



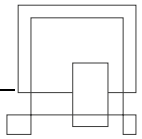

Finished after 0.014 seconds
Runs: 8/8
Errors: 0
Failures: 0
libraty.BookTest [Runner: JUnit 4] (0.000 s)
testSetNumOfCopies (0.000 s)
testGetId (0.000 s)
testSetId (0.000 s)
testSetBookName (0.000 s)
testGetBookName (0.000 s)
testGetNumOfCopies (0.000 s)
testGetAuthor (0.000 s)
testGetPublishDate (0.000 s)
Failure Trace



7 Results and Discussion

Our system has achieved all the requirements according to the list of requirements. During the design we can basically follow the normative methods and steps. Firstly, determine the requirement and the main purpose. Then, analyze and decide the design. Finally, we implement and tested the code. We all believe that our system can work properly on eclipse.

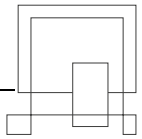
The interface is clearly and easy to operate. However, our original basic design is the user and the administrator can log in and log out with their own user names and passwords. However taking the realistic situation into consideration, we removed this function from our list of requirement.



8 Conclusion

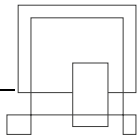
The purpose of our system is to provide online access to the library system. The requirements based on this purpose are fulfilled in this project. The design, implementation and the test part were successfully as well.

This system can create, manage and modify the data by several types of user (administrator and user).



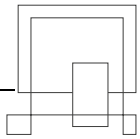
9 Project future

If we had a chance to change some more things in our system, we would set a log-in part for the administrator and the user using their details which would be stored in the database. The system that we have created right now changes the data every time directly in the database but if we got more time we would create a local storage such as hashmap in the model class. So, that when ever the user changes anything in our system it stays in the local memory unless the user saves the data and then it is stored in the database.



10 References

1. VIA College, 2017.IT-SDJ2X-A17 Session Material. [online] Available at: https://studienet.via.dk/Class/IT-SDJ2X-A17/Session%20Material/Forms/Default.aspx [Accessed 27 September 2017].
2. Ken Schwaber and Jeff Sutherland, October 2011, The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game, Scrum.org



11 List of Appendixes

1. **Project Description**
2. **User Guide**
3. **Source code**
4. **Use Case Diagrams**
5. **Class Diagrams**
6. **Sequence Diagram**