

**System Design Specification**

**for**

**Library information appication**

**Version: 2.0**

**Date: 18/06/09**

**20143692 강경석**

**20144103 조유성**

**20145026 김상범**

**20141043 양준호**

**20146110 문태현**

**20143583 김기환**

Table of Contents

Index

[[Library information appliciation]](#_Toc480998875)

[Version 2.0](#_Toc480998877)

[June 9, 2018](#_Toc480998878)

[1. overview](#_Toc480998879)

[2. System architecture](#_Toc480998881)

[2.1 major modules and their function](#_Toc480998882)

[2.2 interfaces between modules](#_Toc480998883)

[2.3 Data Description (Database)](#_Toc480998884)

[2.4 Design alternatives considered](#_Toc480998885)

[3. Process](#_Toc480998881)

[3.1 Risk Assessment](#_Toc480998882)

[3.2 Project schedule](#_Toc480998883)

[3.3 Team Structure](#_Toc480998884)

[3.4 test plan](#_Toc480998885)

[3.5 documentation plan](#_Toc480998884)

[3.6 coding style guidelines](#_Toc480998885)

1. **OVERVIEW**

Our team will provide various information about the use of library.

In user interface, this program provides the users with information about the books which the library has.  
It will provide the list of books that users have borrowed and inform users of the late fees.  
In addition to this, we will offer function that user wants. The details of the user interface function will be explained in major modules in their function(3.1).

In administrator interface, administrators can use this program to manage the list of books the library has, such as adding information from a new book or deleting information from a discarded book.  
And, this program lets administrators know the list of people who have overdue books and their late fees.

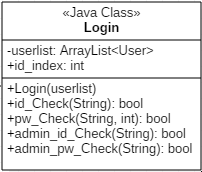
1. **System Architecture**
   1. **major modules and their function**

The modules of this library management program can be largely divided by database, administrator mode api, and user mode api.

Book information and user information are saved in the database.

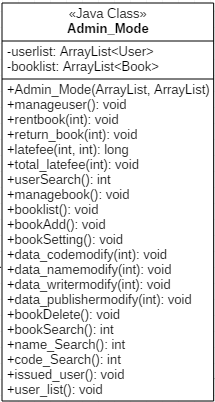
In the beginning of this program, by choosing which ID to login, it can be determined which mode will be used, the user mode or the administrator mode. The access is denied when ID and password are wrong.

**Module of Login**



The administrator can go into the administrator mode when he or she logins with the administrator ID. In the administrator mode, it is possible to access with the book database and the user database of the library. The administrator can search, add, revise, and delete the book information in the database. The administrator can also search the list of the users who have to pay late fee.

**Module of AdmistratorMode**



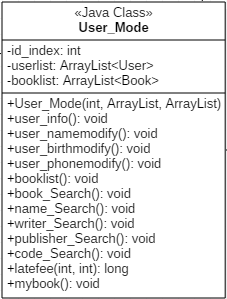
The user can go into the user mode when he or she logins with the user ID. The user mode provides general information about the library. The user can search books by the book’s name, writer, publisher, code number, which shows it can be rentable or not.

The user can know the list of books he or she loaned, and at the same time, the program provides the date of each book when it is loaned and the due date. If the user exceeds the deadline, the program shows the late fee by calculating days.

Also, in the user mode, the user can change his or her password and phone number in the user database. However, the user cannot change his or her ID, name, and birthdate.

The user can also withdraw from the library membership.

**Module of UserMode**



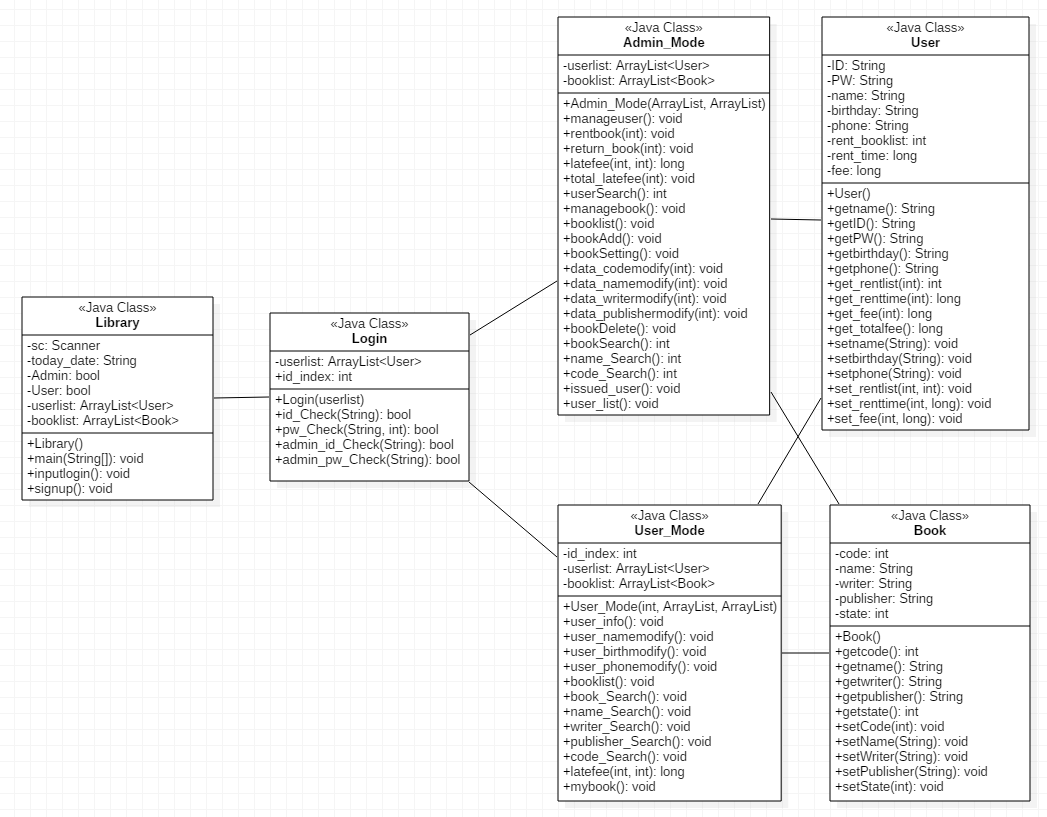
These functions of the API are the main modules of our program.

If we have remaining time, it is possible to add more functions in addition. For example, if the user apply for new books, the administrator can know the list of the books that are requested. If the requested books come into the library, the administrator can send messages to the user who requested the books. Secondly, the user may extend the due date of the loan date. Thirdly, if the user wants to loan the book that is already loaned, the user may reserve the book to loan.

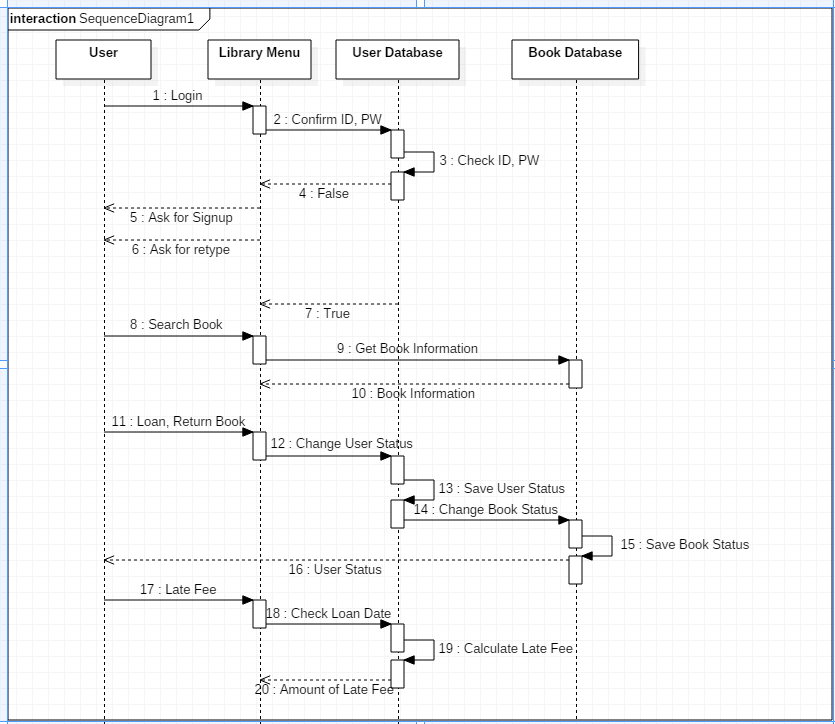
The user mode module and the administrator mode module exchange information and interact with each other through the database. This interaction will be described thoroughly in 3.2

* 1. **interfaces between modules**

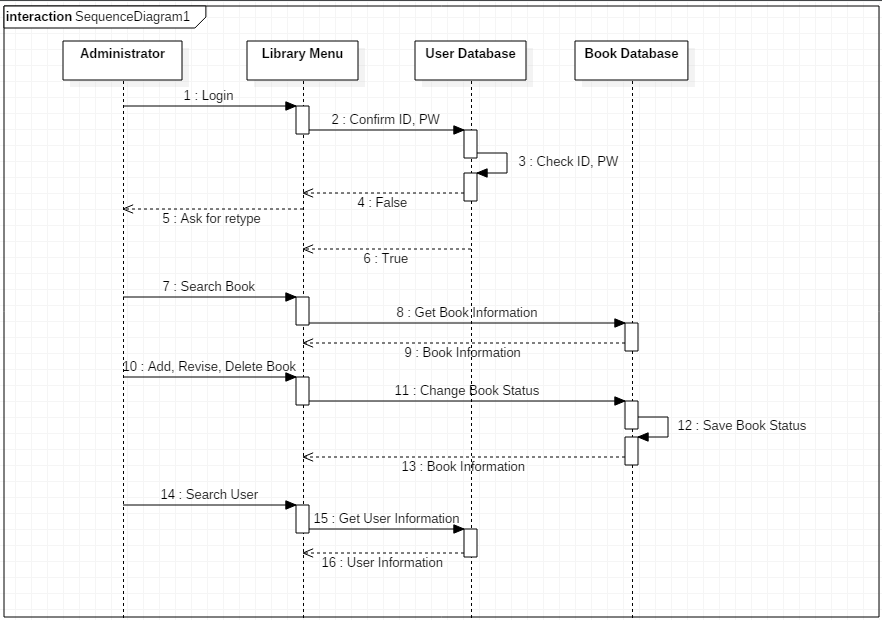
**UML Class Diagram**

****

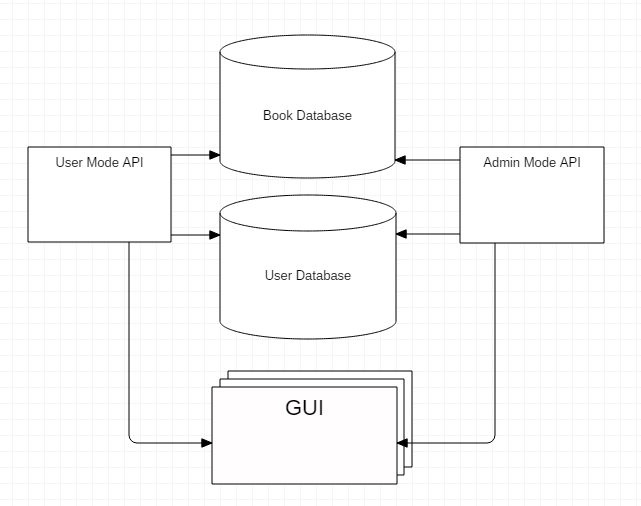
**UML Sequence Diagram about UserMode**



**UML Sequence Diagram about UserMode**

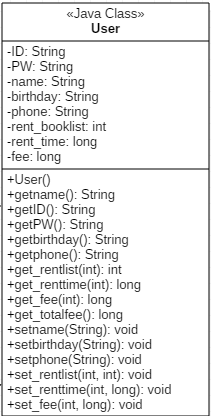
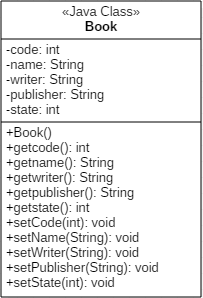


**System Architecture Diagram**



* 1. **data description(data base)**

**Conceptual Schema**

****

**External schema**

**(1) User**

1) User’s view

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Password | Name | Birthday | Phonenum | Rent\_booklist | Rent\_time | Rent\_fee |

2) Administer’s view

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Password | Name | Birthday | Phonenum | Rent\_booklist | Rent\_time | Rent\_fee |

* User can change all of elements of user\_info except ID, Rent\_booklist, Rent\_time, Rent\_fee
* Administer can search all elements user\_info except Password
* Administer can change Rent\_booklist, Rent\_time and Rent\_fee of user\_info.

**(2)Book**

1)User’s view

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Code | Name | Writer | Publisher | State |

2)Administer’s view

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Code | Name | Writer | Publisher | State |

* Adminster can add, change, delete all elements of book\_info
* User only can search all elements of book\_info.
  1. **design alternatives considered**

Our design pattern is Repository architecture.

★Another design pattern

1) Layered Architecture Pattern

*Pros) Lower layers are used by other higher layers. Layer normalization is easy and it is easy to define layer levels. Changing a layer does not affect other layers.*

* In our program, the layers between the database, user mode api, and administrator mode api are clearly divided.

*Cons) Wide application is difficult. Under certain circumstances, a particular layer may be unnecessary.*

* In our program, the administrator and the user cannot both access the database at the same time.

2) Model-View-Controller Pattern

*Pros) By separating the business logic from the user interface, you can create an application that can easily fix the visual elements of the application or the business logic running behind it without affecting each other.*

* The management of GUI, and other APIs of our program can be separated.

*Cons) It increases complexity, and can cause unnecessary updates to user behaviour.*

* It can increase the complexity of our program.

3) Broker Pattern

*Pros) Dynamic changes, additions, deletions, and reassignments of objects are possible, making deployment transparent to developers.*

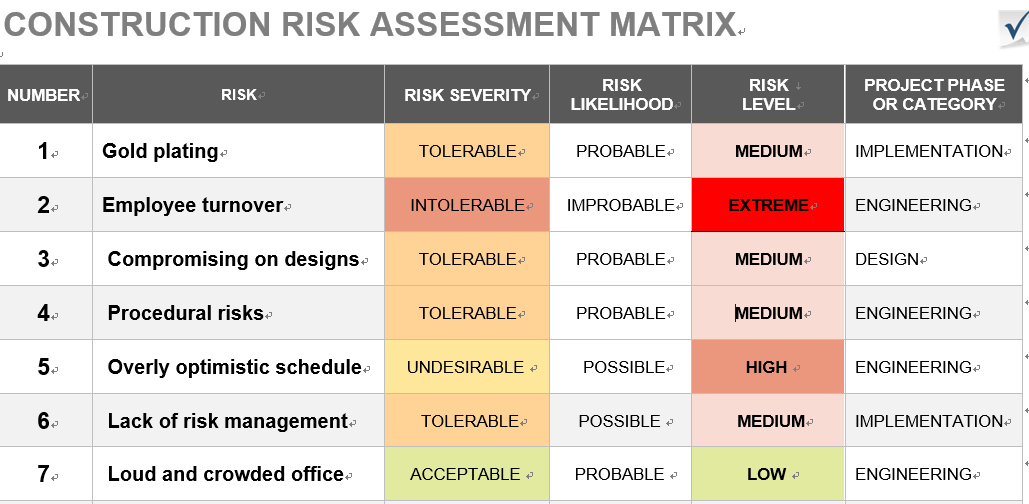
* The making process of this program is transparent and clear.

*Cons) It’s hard to fix errors.*

* When the database, the user mode api, and the administrator api have
* errors, it’s hard to fix them.

**3. Process**

**3.1 Risk assessment**

****

1. Gold plating

Developers sometimes like to show off their skills by adding unnecessary features. For instance, a developer might add Flash to a basic login module to make it look ‘stylish’. Again, this is a waste of programming hours.

2. Employee turnover

Every project has a number of developers working on it. When a developers leaves, he or she may take critical information with him/her. This can delay, and sometimes derail an entire project.

3. Compromising on designs

In order to get stuck into the next ‘real’ tasks, developers tend to rush the design-process. This is a waste of programming hours, as designing is the most critical part of software development.

4. Procedural risks

Day-to-day operational activities might hamper due to improper process implementation, conflicting priorities, or a lack of clarity in responsibilities.

5. Overly optimistic schedule

Time goes unexpectedly to fast.

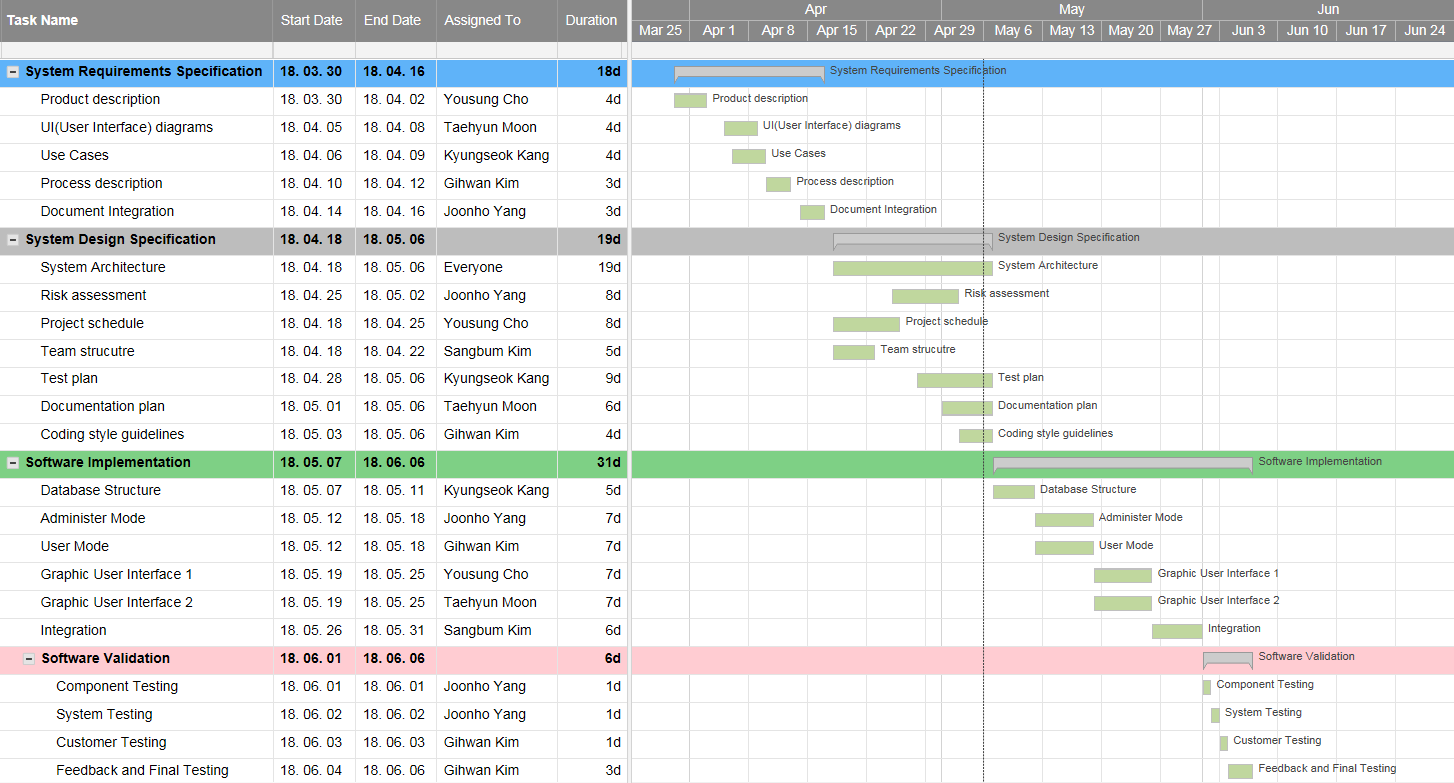
6. Lack of risk management

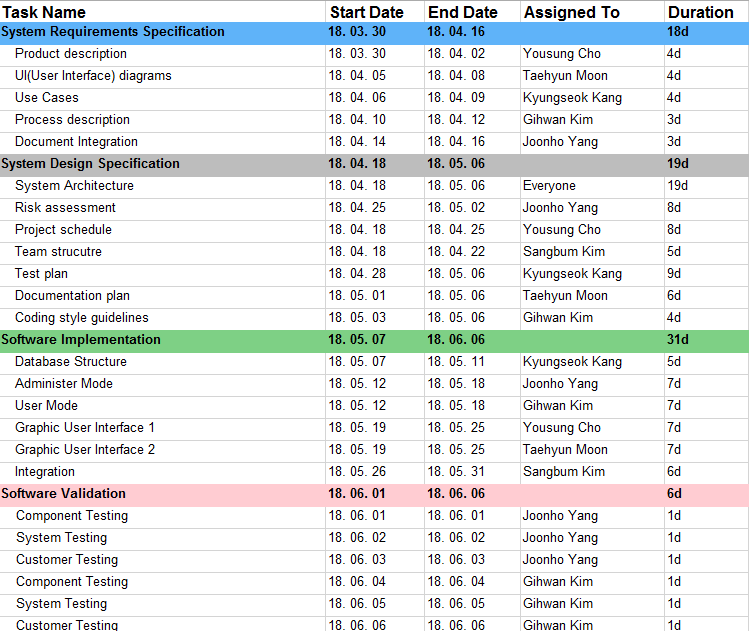
Projects are always potentially error-prone.

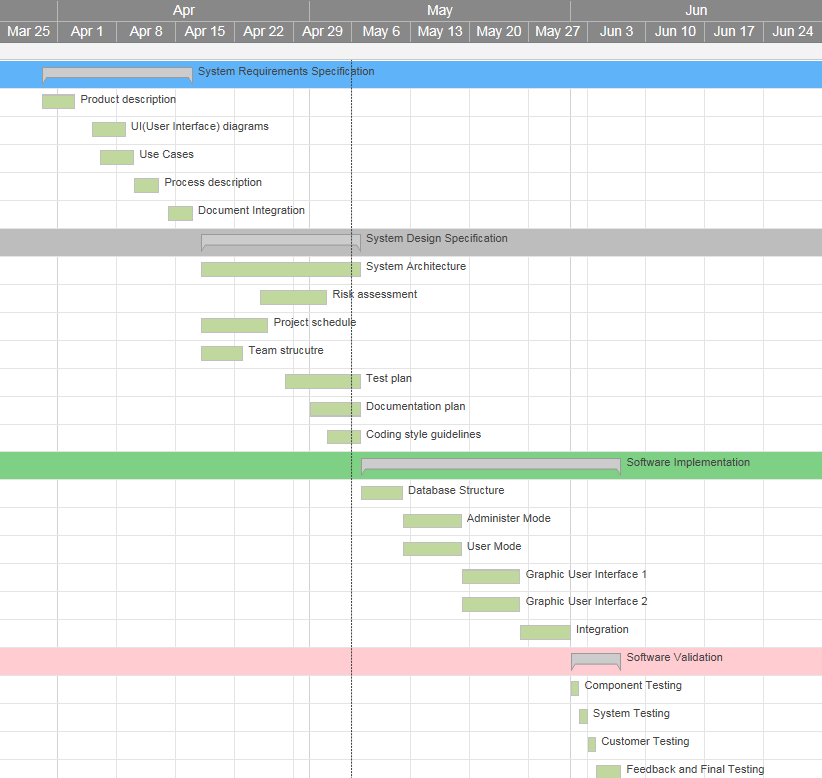
7. Loud and crowded office

Loud and crowded office can decrease efficiency of work.

**3.2 Project schedule**







Mar 30 ~ April 16

- System Requirements Specification

April 18 ~ May 6

- System Design Specification

May 7 ~ May 31

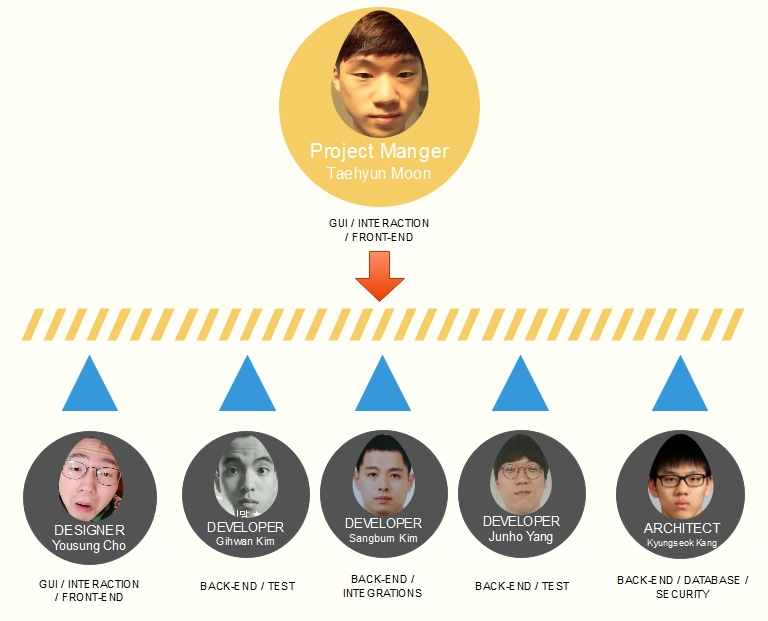
- System Implementation

June 1 ~ June 10 (change)

- System Validation

Our group has been meeting twice a week, from 13:00pm to 16:00pm on Thursday and Saturday.  
We will also schedule meetings in addition to those times when necessary.

**3.3 Team structure**



### **Project Team Assignment Roster**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Group** | **Role** | **Responsibility** |
| Taehyun Moon | Front-end Team | Core Team leader (Project manager) | •Leading & managing Project Team. •Schedule authority for project. •Splits the tasks among the members of our group •Also support Front-end Part. |
| Yousung Cho | Front-end Team | Designer | •Developing the GUI. •Translate a project engineer’s ideas into the appropriate software. •Design decisions will be made through group discussion and Designer makes a final review. |
| Kyungseok Kang | Back-end Team | Architect | •Creating H/W & S/W architecture. •Make sure the software and system architectures are in synchronization •Manage risk identification and risk mitigation strategies associated with the architecture. |
| Sangbum Kim | Back-end Team | Developer | •Developing the solutions. •Integration with other modules. •Review whether team have coded the code guidelines •Risk management that can occur when integrating |
| Junho Yang | Back-end / Test team | Developer | •Developing the solutions •Defining the appropriate tests required and any associated Test Data •Gathering and managing the Test Data and evaluating the outcome of each test cycle. |
| Gihwan Kim |

**3.4 Test Plan**

Test Plan Unit testing will be performed on each major class to verify conformity to specification. We will use a combination of black and white box testing.

Usability tests will be performed as the application becomes functional from the perspective of a user. Initially the testers will be only our group members, but as the app becomes more complete we will try to include other users as well.

At first these tests can be done on just our own infrastructure, but at some point we will need to rental books or return books at library to test some features.

This will potentially be easy because we can add some book information and actually test it by borrowing it and returning it.   
These functions are executed the same way each time, so the reliability of the usability test can be guaranteed.

For all of the basic functionality of our application we expect these tests to be adequate.  
However it will be almost impossible to test the entire application comprehensively in a real world scenario without a larger user base and well-timed failures of the library system.

As bugs appear they will be added to mobile talk by the team members who discover them.  
When they are discovered how to reproduce the bug and how its behaviour differs from expectations should be logged.

**3.5 Documentation plan**

In order to help users figure out how to use our library info application, we will provide various documents.

**3.6 Coding style guidelines**

java: <https://google.github.io/styleguide/javaguide.html>

There are a few options available to ensure that we follow these style guides.  
To enforce coding style for Java, we can use the XML code style configuration file provided by Google, in Android Studio.  
It is likely that these tools do not completely enforce every rule in the above style guides.  
So additionally, we can review each other’s code to make sure it conforms to the relevant coding style guidelines.