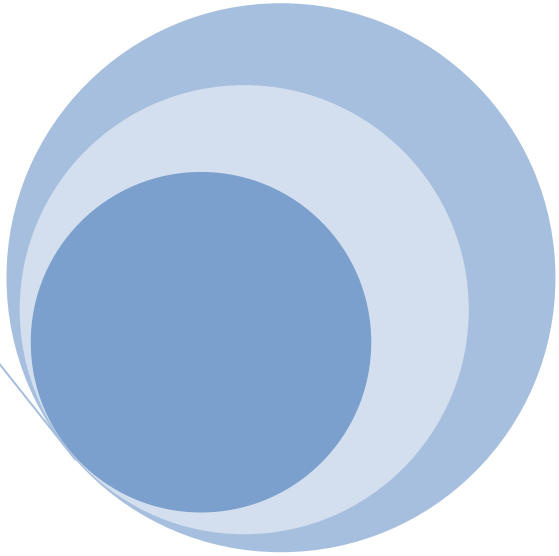

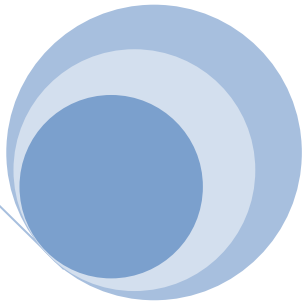

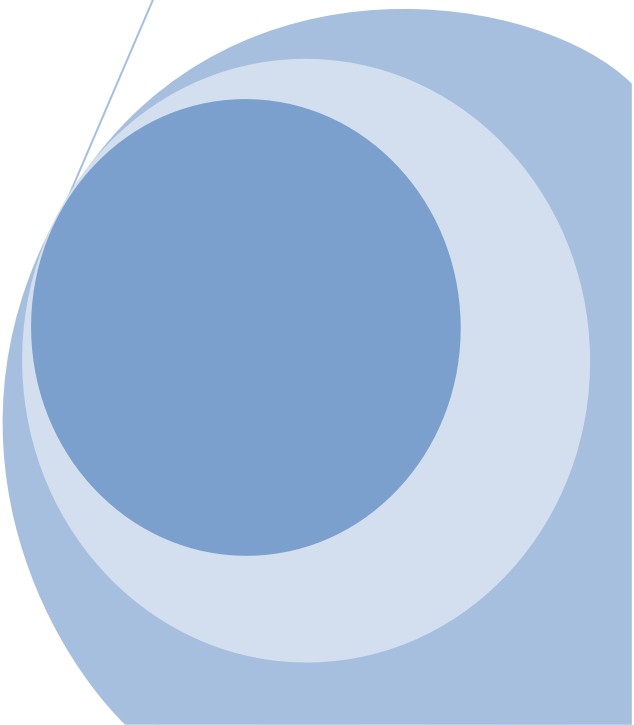

Book Management

System Requirement Specification

4/11/2012

Content

Introduction

This System Requirement Specification (SRS) document is a detail description of all requirements of Faculty Directory system. The requirements are organized in a specific way for the viewpoint of the client and development team.

System Architecture

The architecture for our book management system will have 3 layers from MVC model perspective:

1. Presentation Layer: the interface (web page) for users to access the system.
2. Logic Layer: it controls the function that our system provides to users
3. Database Layer: it is the database systems which store necessary information for our system.

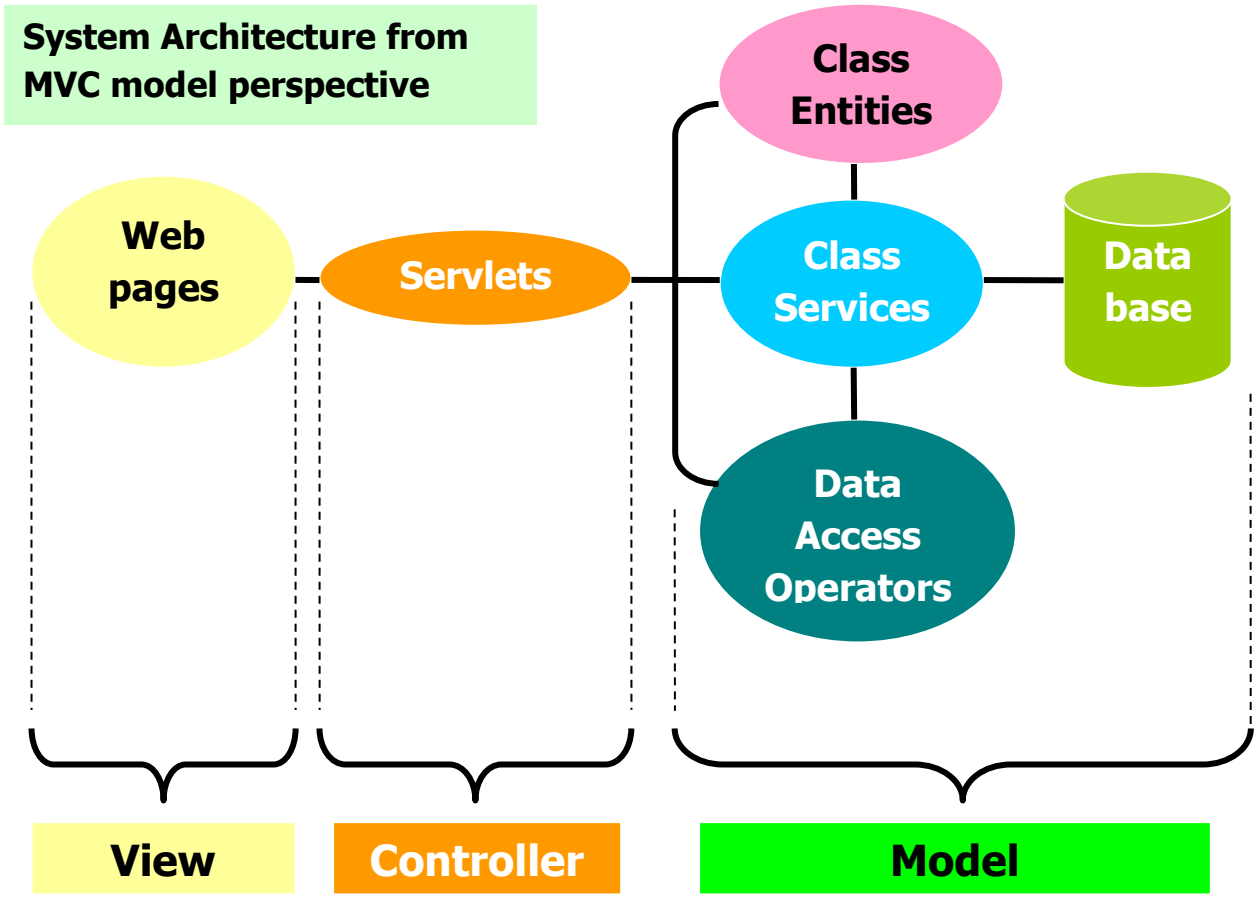

Figure 3: MVC Model

System Requirement

In this part, we will give you a brief detail of functions that we implement into our book management system in Sprint Cycle 1, which are:

1. Login
2. View library policies
3. View book list
4. View book details
5. Search for books

This session will show you normal case, what can go wrong when doing the activities, and the initial assumption may be involved when performing a function as well as some models and diagrams to explain how our system works.

Context Model

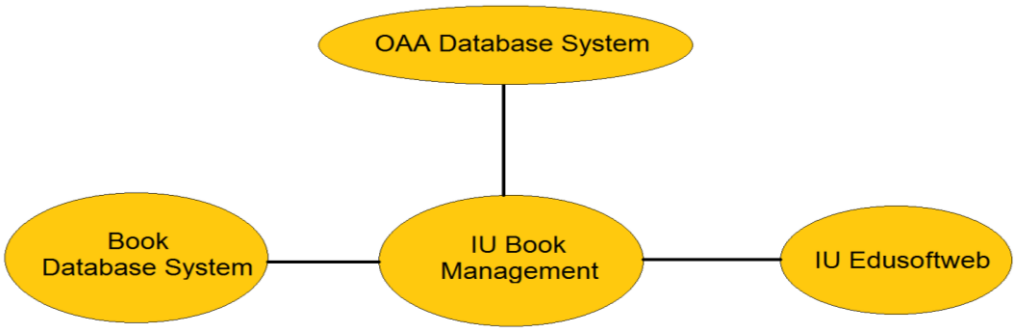

Figure 3.a: Context model

Structural Model

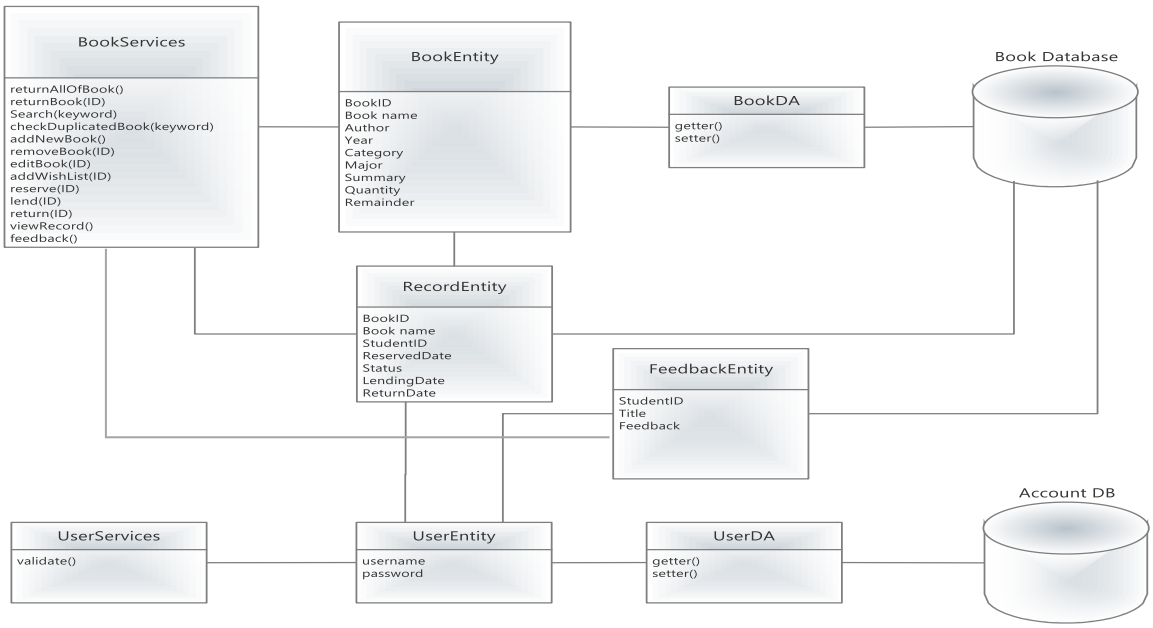

Figure 3.b: Class diagram

3.1 Login

Users can choose between 1 of 2 options:

1. Access the system as a guest.
2. Access as a librarian or student for using some oncoming addition functions which the system provide in future.

Initial conditions	User want to login the system for authority to perform his/her actions
Normal case	In the homepage, users will enter their usernames and passwords in the login box. The usernames and passwords are generated by OAA
What can go wrong?	User enters wrong username or password
Other activities	No other actions can be performed in this story
System state on completion	Data is loaded, but the system remains unchanged

3.1.1 Use Case

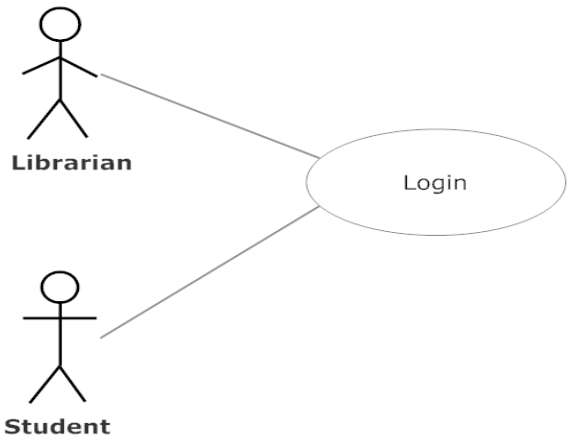

Figure 3.1.1: Use case model of **Login** function

3.1.2 Behavioral Model

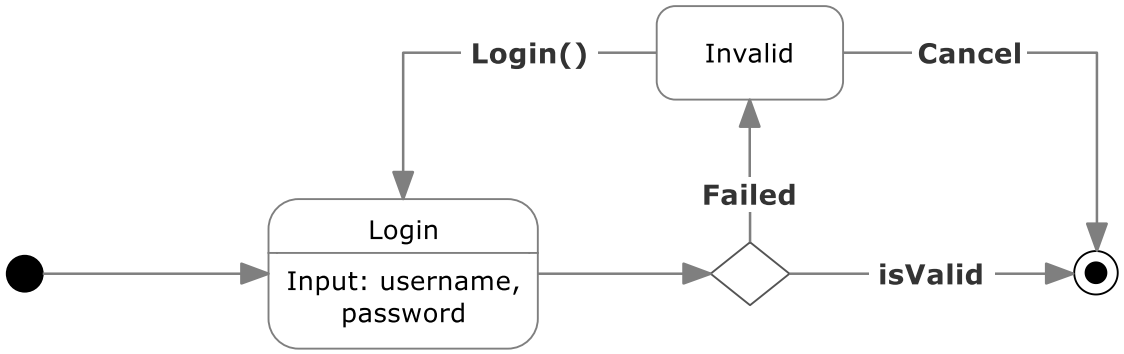

Figure 3.1.2 State diagram model of **Login** function

3.1.3 Interaction Model

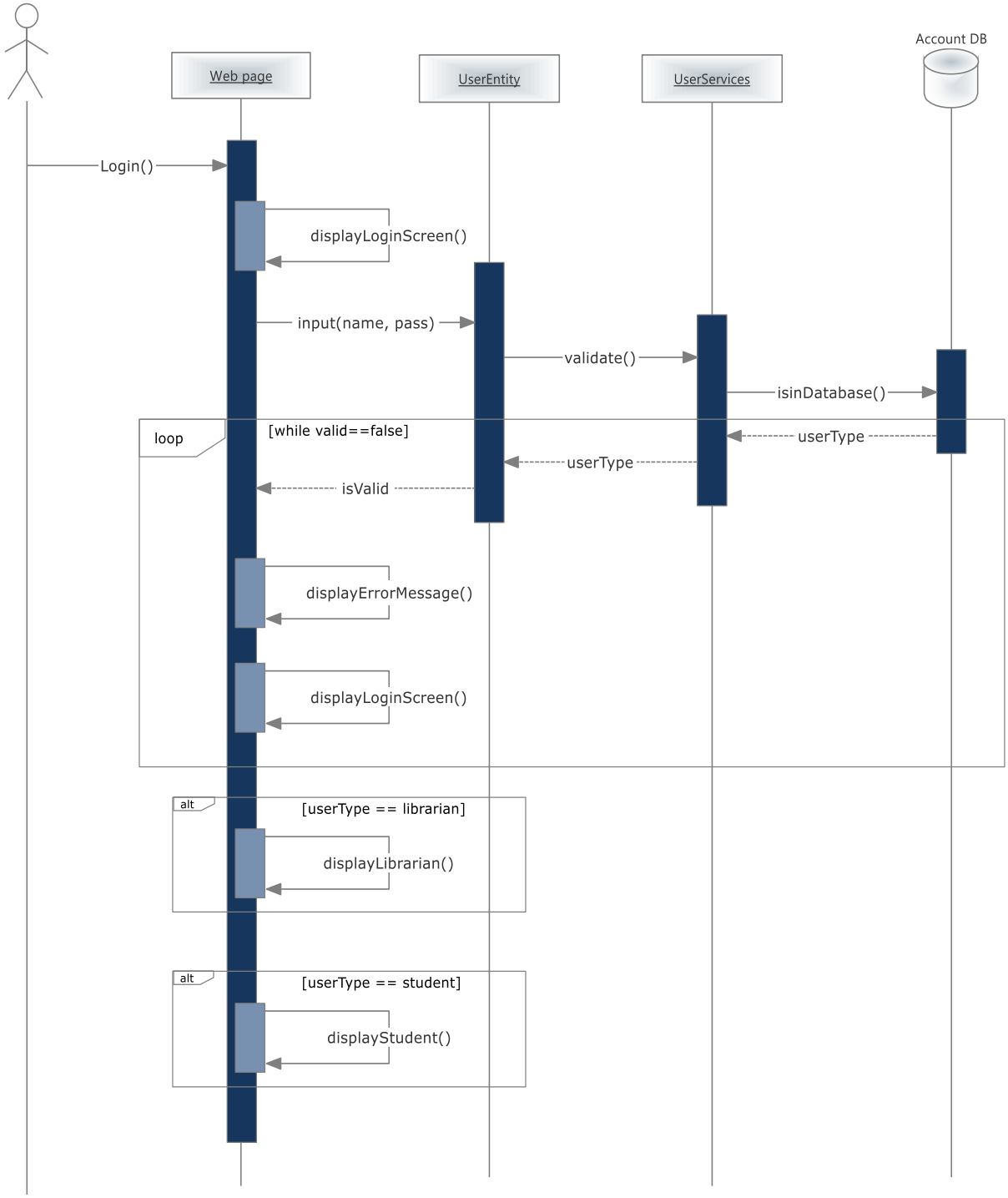

Figure 3.1.3 Sequence diagram model of **Login** function

3.2 View library policies

Initial conditions	User wants to view Library policies.
Normal case	User visits the website, login is optional, after the homepage loaded, user chooses 'Policies' to view all the rules
What can go wrong?	In this story, nothing can go wrong
Other activities	No other actions can be performed in this story
System state on completion	Data is loaded, but the system remains unchanged

3.2.1 Use Case

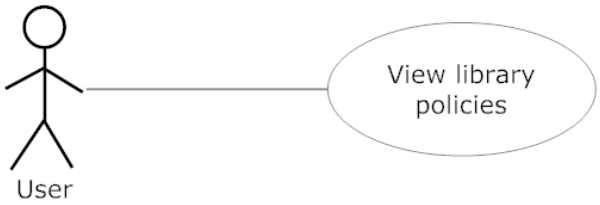

Figure 3.2.1: Use case model of **View library policies** function

3.2.2 Behavioral Model

Figure 3.2.3 Sequence diagram model of **View library policies** function

3.2.3 Interaction Model

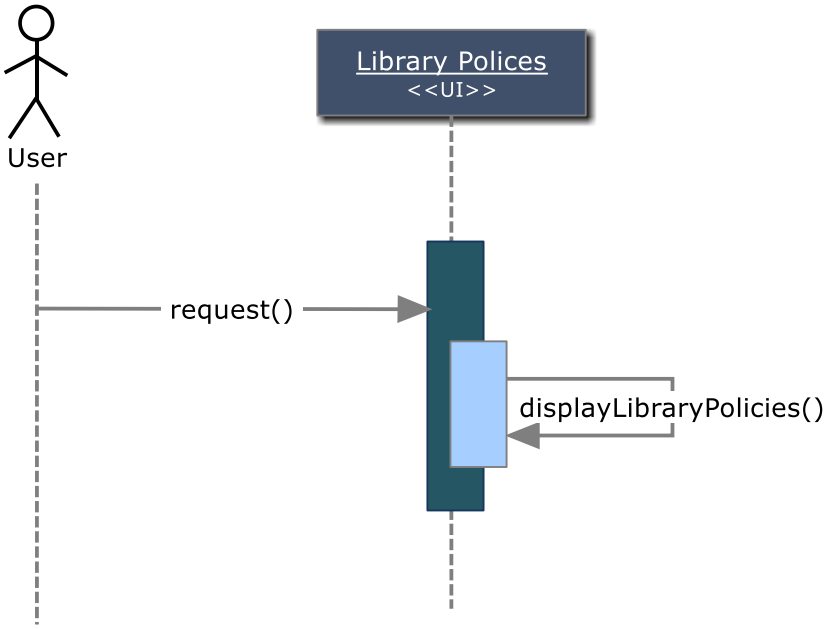

Figure 3.2.3 Sequence diagram model of **View library policies** function

3.3 View book list

Initial conditions	User wants to view the list of books that library has.
Normal case	User visits the website, login is optional, after the homepage loaded, user chooses 'View book list' to list out all the book.
What can go wrong?	In this story, nothing can go wrong
Other activities	If the list of book is currently updated by the Librarian, the new book might not be shown in the list yet until refresh.
System state on completion	Data is loaded, but the system remains unchanged

3.1 Use Case

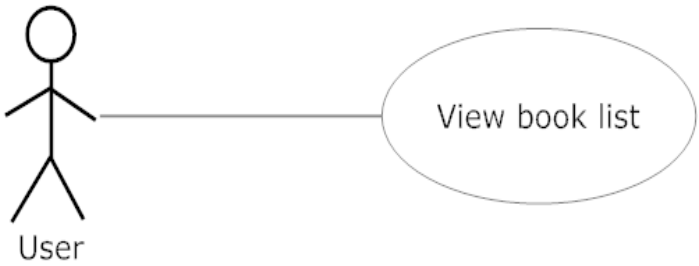

Figure 3.3.1: Use case model of **View book list** function

3.3.2 Behavioral Model

Figure 3.3.2 State diagram model of **View book list** function

3.3.3 Interaction Model

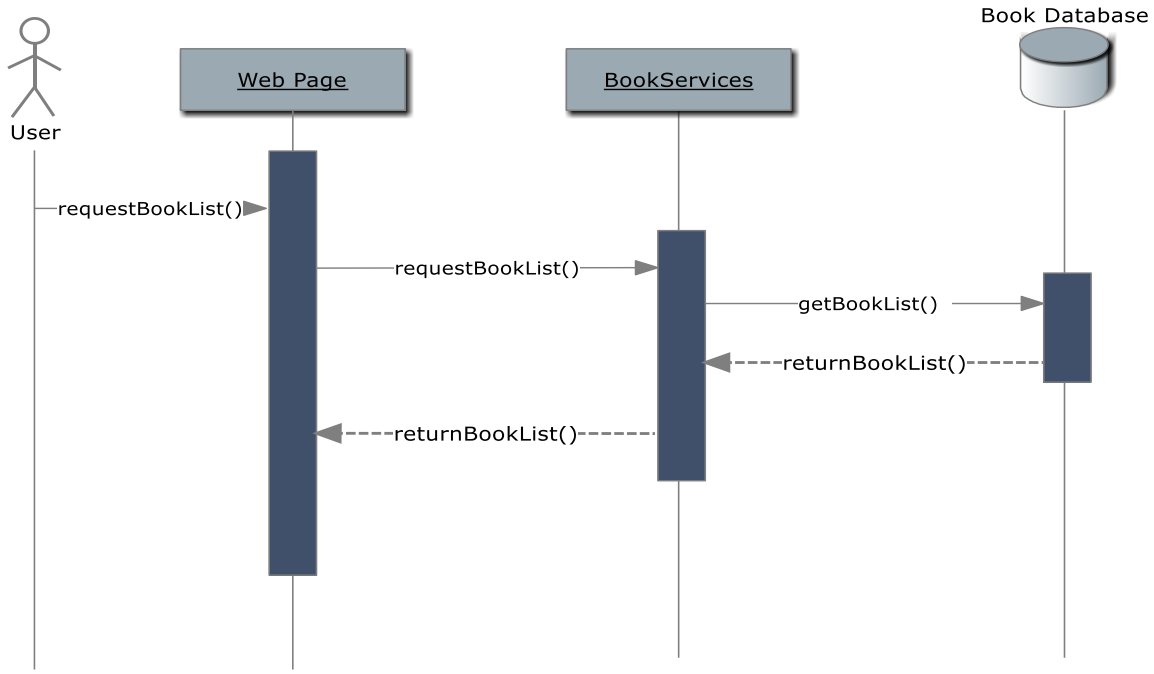

Figure 3.3.3: Sequence diagram of **View book list** function

3.4 View book details

Initial conditions	User wants to see the information about a book
Normal case	User have visited page 'Book list', then clicked on the book title. The browser will automatically connect to the details of the book.
What can go wrong?	If the requiring book is removed by the Librarian concurrently, the system won't display the book any more.
Other activities	No other action can be performed in this story
System state on completion	Data is loaded, but the system remains unchanged

3.4.1 Use Case

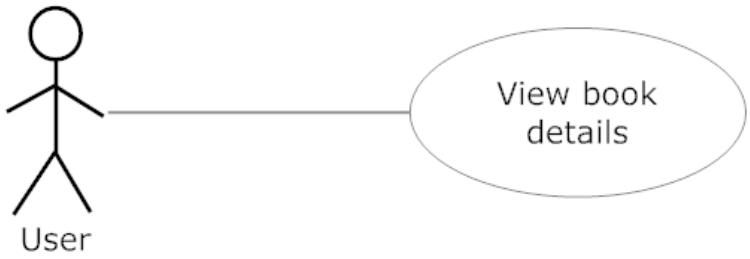

Figure 3.4.1: Use case model of **View book details** function

3.4.2 Interaction Model

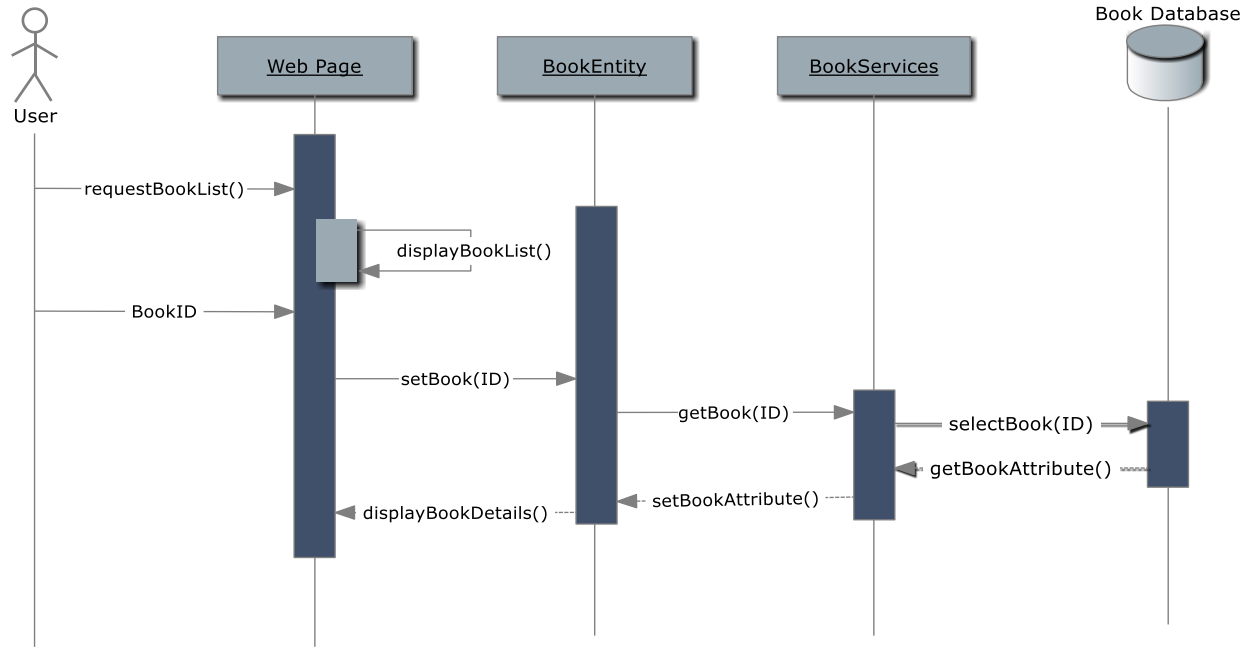

Figure 3.4.2 Sequence diagram model of **View book details** function

3.4.3 Behavioral Model

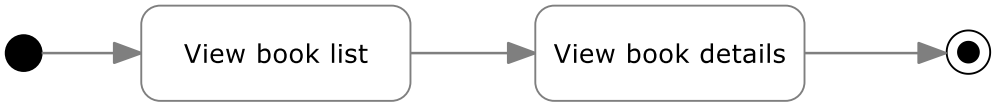

Figure 3.4.3 State diagram model of **View book details** function

3.5 Search for books

Initial conditions	User wants to search a book in the system
Normal case	User goes to 'Search' page, then types the details of the book in the search box, then clicks search. The result returns to user.
What can go wrong?	The required book may not in the database yet therefore system will return "book not found" and browse to recommend page
Other activities	No other activities is involved
System state on completion	System remains unchanged

3.5.1 Use Case

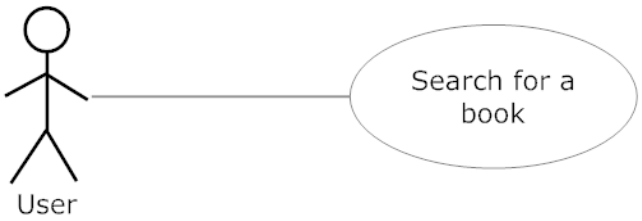

Figure 3.5.1: Use case model of **Search** function

3.5.2 Behavioral Model

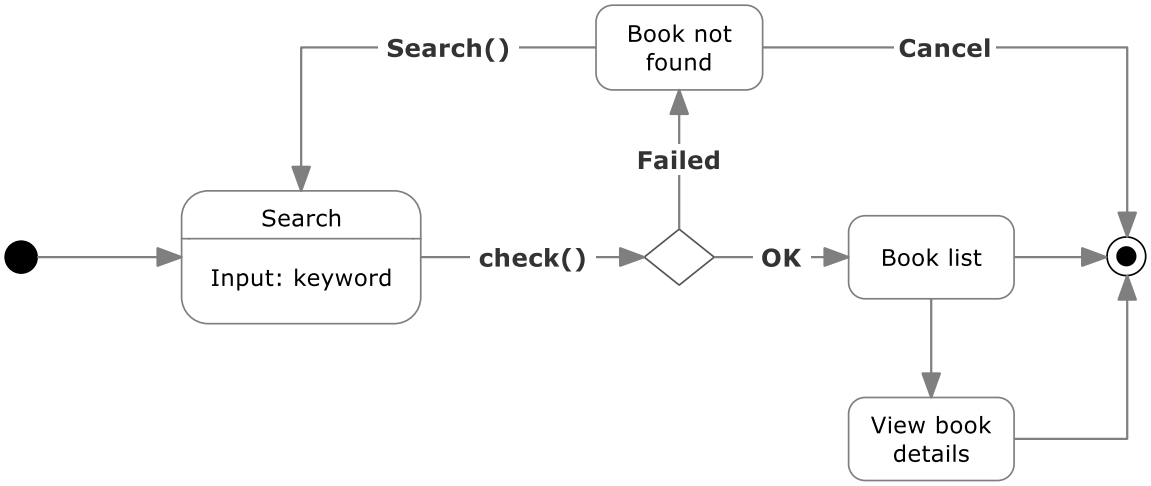

Figure 3.5.2: State diagram of **Search** function

3.5.3 Interaction Model

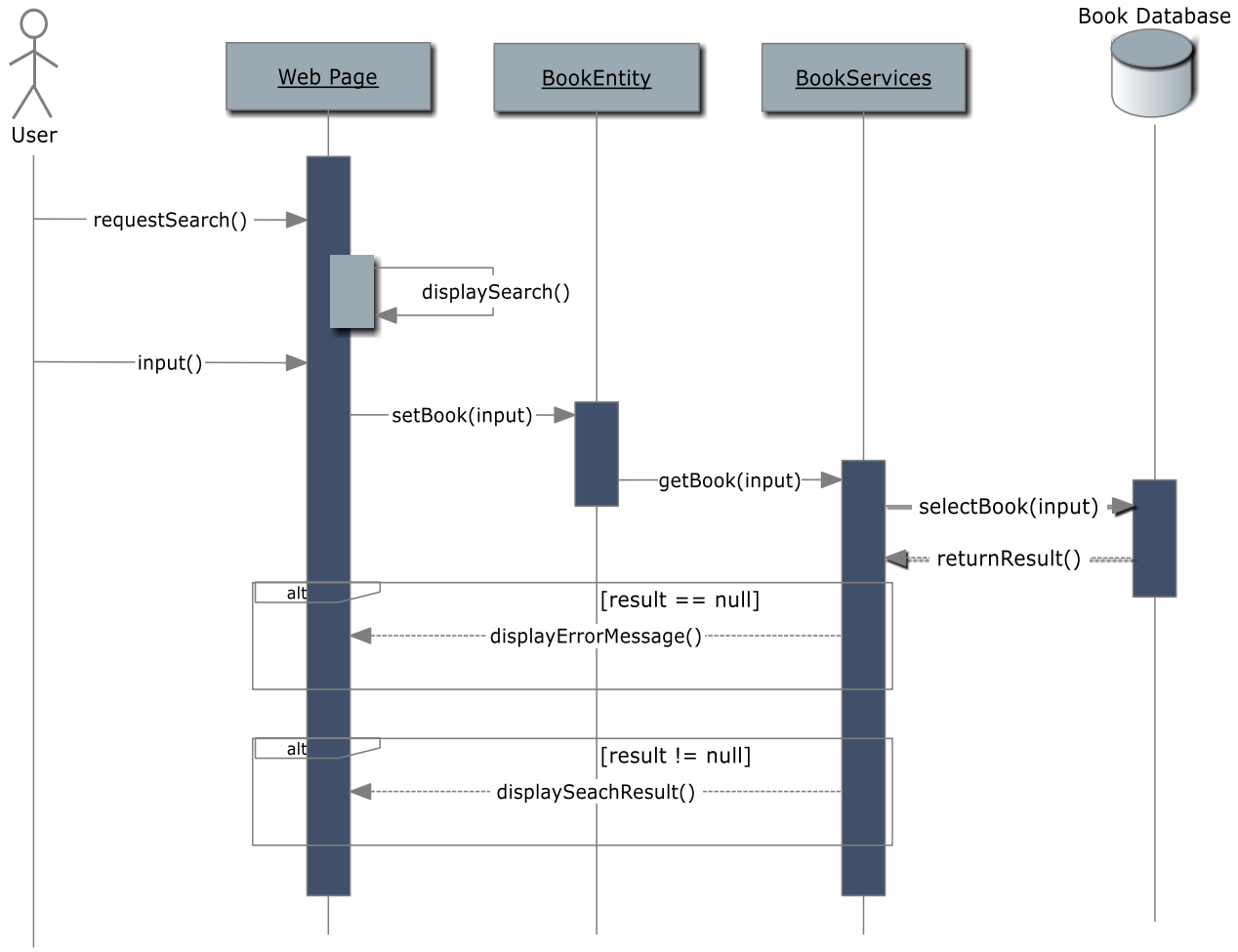

Figure 3.5.2: Sequence diagram of **Search** function

User Requirement

4.1 Functional Requirement

As we mentioned above in Part 3 - System Requirement, our book management system has 6 functions in Sprint Cycle 1.

1. **Login**: Students and librarians can use the usernames and passwords that IU generate to log in and perform other activities involved in the software.
2. **View library policies**: The web page will show all the standard rules in the library for students to follow accurately.
3. **View book list**: The web page will show the list of books that the library has.
4. **View book details**: The page will show the attributes of the book such as name, author, quantity, ranking as well as a brief summary of the book.
5. **Search a book**: This action will help users to find the book they need faster and easier.

4.2 Nonfunctional Requirement

Property	Measure
Environment	The system is developed on J2EE Framework. Programming languages are Java and HTML. Database used: My SQL Server 5.5
Speed	0.1s for response time 0.5 to 1.5 seconds for processed transactions
Reliability	The system must not return any inaccurate result except when the input information is inaccurate.
Robustness	Time to restart after failure: 1 hour. A backup system will run on the server during the time failure of the main system is being fixed.
Stability	The system must be stable when the range of users increases. Minimum number of access that the system can handle stably is 4000.
Ease of use	Library staffs can use all the functions that system provides and initialize the data after being trained in 2 hours.
Web-interface	Interesting interface instead of boring tables and buttons
Help and Support	Engineering team must reply any questions from the administrators as soon as possible.

4.3 Domain Requirement

The Library privacy will be included so that the users can follow the rules of library accurately.

(*) The primary language is English for the system is designed for International University. Besides, it will also have Vietnamese for other students studying in other universities which belong to Vietnam National University.

(*) Vietnamese version will be applied into the system after our product is completed.

Interface and Sample Code

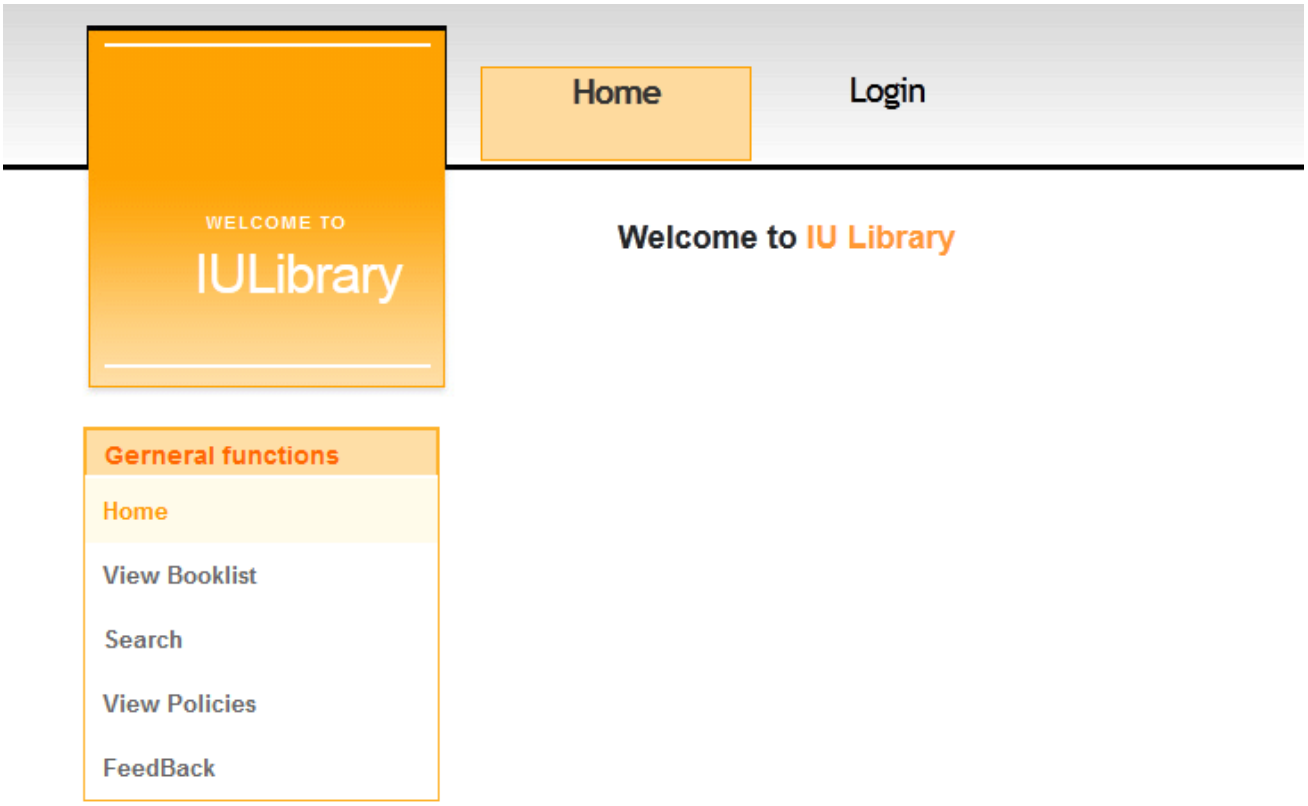

Figure 5.1: Interface of **Home page**

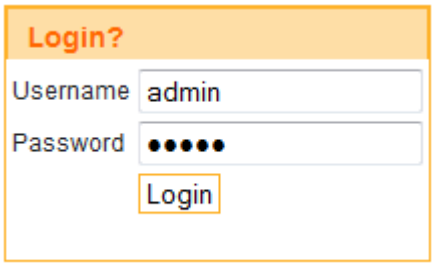

Figure 5.2: Interface of function **Login**

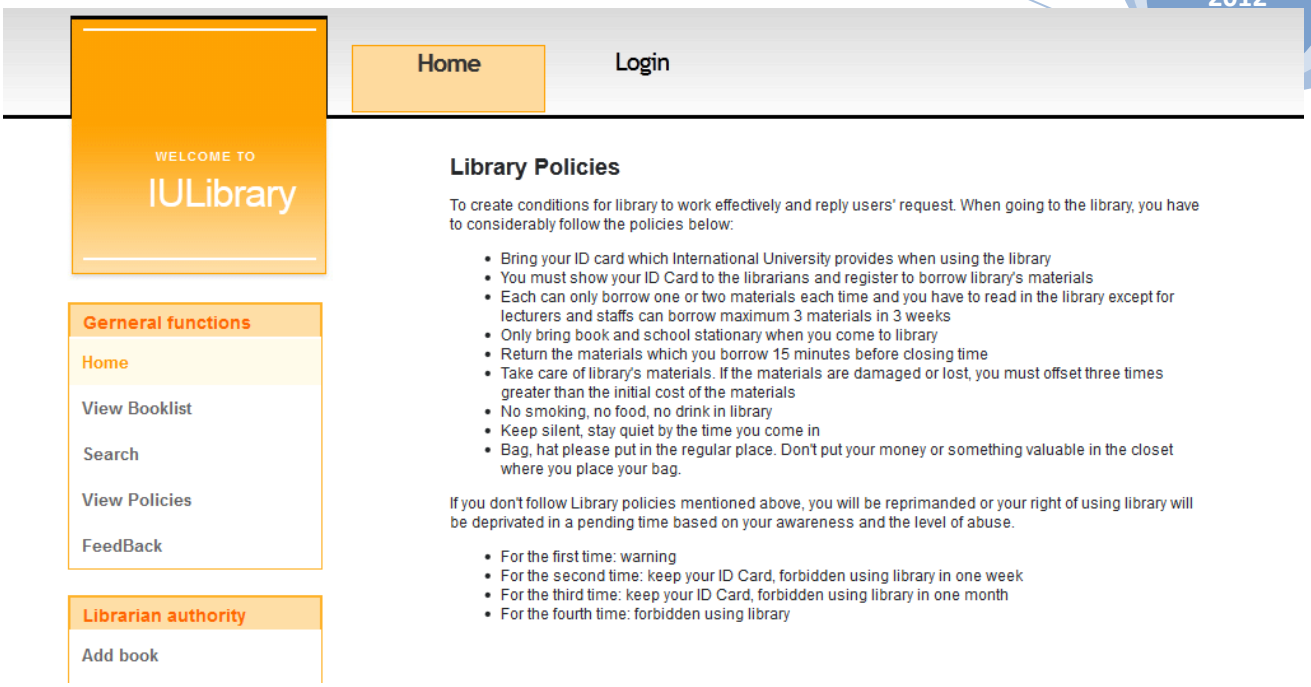

Figure 5.3: Interface of function **View library policies**

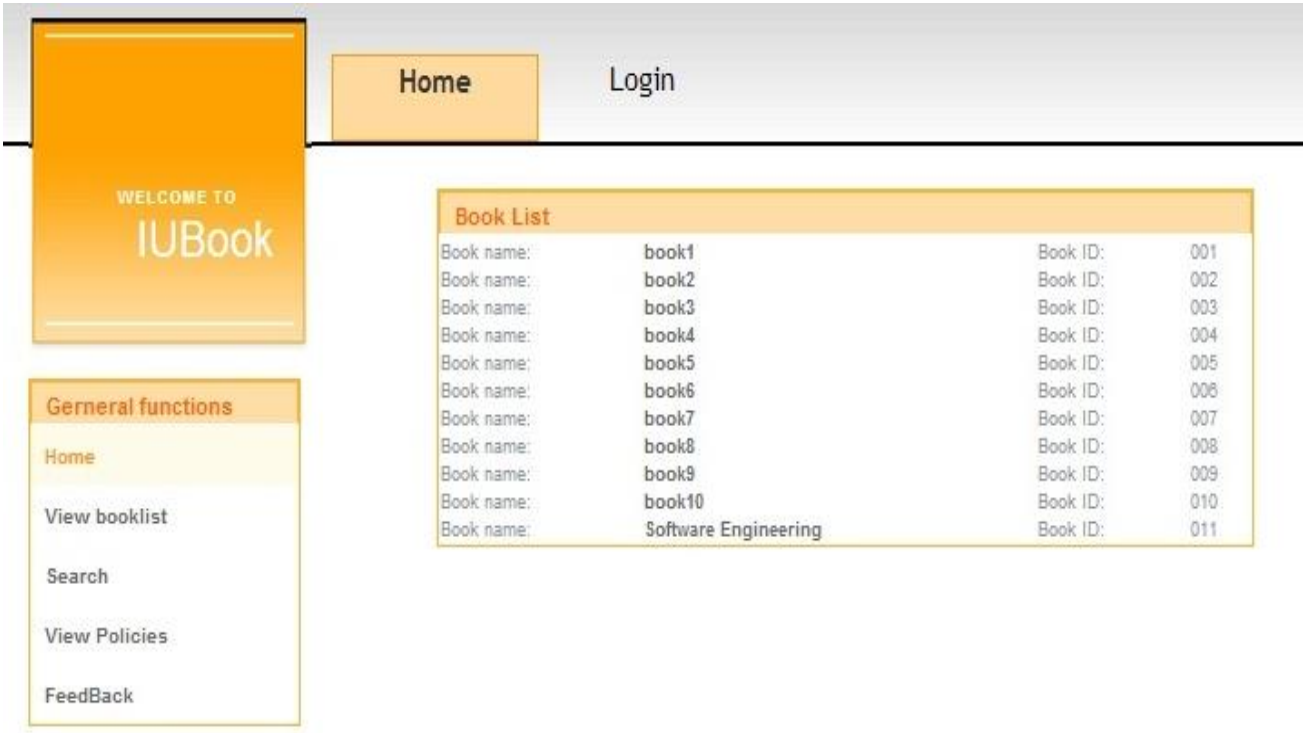

Figure 5.4: Interface of function **View book list**

Figure 5.5: Interface of function **View book details**

Figure 5.6: Interface of function **Search**

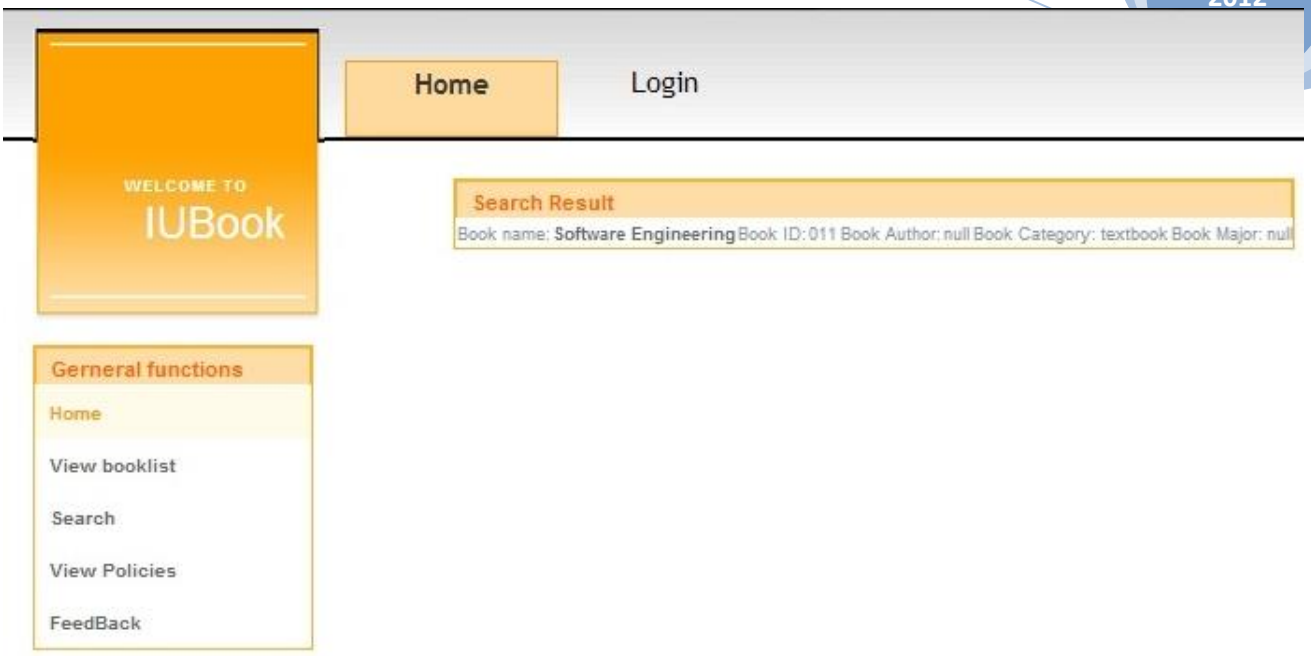

Figure 5.7: Interface of **Search Result**

```
private boolean validate(String usn,String psw){
    if(UserServices.validate(usn, psw)) return true;
    return false;
}
```

Figure 5.8: Sample code of **validate** method

```
<div id="archives" class="box_one">
        <h2>Search?</h2>
        <form action="Controller?action=search" method=POST>
            <table>
            <tr><td>Book Name<td><input type="text" name="book_name"><br>
            <tr><td>Author<td><input type="text" name="author"><br>
            <tr><td>Major<td><input type="text" name="major"><br>
            <tr><td>Catergory<td><input type="text" name="catergory"><br>
            <tr><td><td><input id=menu class=box_one type="submit" name=submit value=search>
            </table>
```

Figure 5.9: Sample code of **search** method

```
<%
String id = request.getParameter("parameter");
BookServices services = new BookServices();
Book book = services.returnBook(id);

%>
```

Figure 5.10: Sample code of **View Book list** function

```
</div>
<div id="content">
    <div id="posts">
        <div class="post">
<!--            <h1 class="title">Welcome to <span class="hgone">website</span></h2> -->
        </div>
        <div class="date">
        <%String requestAction=request.getParameter("action");
          String requestParameter=request.getParameter("parameter");
          String requestKeyword=request.getParameter("keyword");

            if(requestAction.equals("login")){%>
                <jsp:include page="login.jsp" flush="true"/><%} %>

        <%if(requestAction.equals("success")){%>
                <jsp:include page="success.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("regist")){%>
                <jsp:include page="Register.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("logfail")){%>
                <jsp:include page="logfail.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("showInfo")){%>
                <jsp:include page="showInfo.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("viewBookList")){%>
                <jsp:include page="viewBookList.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("search")){%>
                <jsp:include page="search.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("addNewBook")){%>
                <jsp:include page="addNewBook.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("homepage")){%>
                <jsp:include page="homepage.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("feedBack")){%>
                <jsp:include page="feedBack.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("viewPrivacy")){%>
                <jsp:include page="viewPrivacy.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("logout")){%>
                <jsp:include page="logout.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("searchResult")){%>
                <jsp:include page="searchResult.jsp" flush="true"/><%}%>

        <%if(requestAction.equals("viewBookDetail")){%>
                <jsp:include page="viewBookDetail.jsp" flush="true"/><%}%>
            </div>
    </div>
    <div id="navi">
        <div id="archives" class="box_one">
            <h2>Gerneral functions</h2>
            <ul>
                <li class="active"><a href="view.jsp?action=homepage">Home</a><i></i></li>
                <li><a href="view.jsp?action=viewBookList">View booklist</a><i></i></li>
              <li><a href="view.jsp?action=search">Search</a><i></i></li>
                <li><a href="view.jsp?action=viewPrivacy">View Policies</a><i></i></li>
              <li><a href="view.jsp?action=feedBack">FeedBack</a><i></i></li>
            </ul>
        </div>
        <div id="categories" class="box_one">
            <h2>Librarian authority</h2>
            <ul>
                <li><a href="view.jsp?action=addNewBook">Add book</a></li>
                <li><a href="#">Remove book</a></li>
                <li><a href="#">Delete book</a></li>
                <li><a href="#">Lend book</a></li>
                <li><a href="#">News Room</a></li>
            </ul>
      </div>
        </div>
```

Figure 5.11: Sample code of **view** method

Risk Analysis

This section analyzes risks which our team may have during the project. In this part, we talk about the probably risk and how to handle this. The risks will have 4 kinds such as project (members, time, etc.), product (bug, error, etc.), business (competitor with the same software, technology change, etc.).

The risks level is: Low (Green), Moderate (Yellow), High (Red)

- All low level risks will be solved in one day.
- All moderate level risks with level Moderate will take time from 2 to 4 days to be fixed.
- All high level risks with level High will take from 6 to 7 days to be fixed.

Notes:

- With business risks, the project can be stop immediately.
- In the case that the risks cannot be solved in time, our company will work overtime to get the deadline.

The low level risks can be repeated all the time project running.

6.1 Project risks

Risks	Risk Description	Solution to avoid
Weak cooperation between members	Member have not work together before	Plan team with familiar members
Less team meeting	Team member's home far from each other – make the project go closely	Using Skype, Gmail to communicate, have 2 meeting/weeks at cafe shop
Team member change	Some problems with team members and need to replace by another one.	Take more time to familiar with new member. We also use pair-programming to make it as fast as possible
Member Rejection	Make the project delay or force to stop the project	New one member in our company take the empty as soon as possible. We also use pair-programming too sure about the continue
Hardware unavailability	Virus or physical affection(hardware crash, lost database).	Create backup per day to, use newest antivirus program

Table 6.1 - Project Risks

6.2 Product Risks

Risks	Risks Description	Solution to avoid
Component Compatibility	Components is not compatible to run	Using XP(Extreme Programming) to have compatible component
Tasks – Requirement changing	Customer will change the tasks	Using Scrum models – make customer change possibility
Bugs when running	Some bugs, wrong affection when using program	Tester always check to make sure the quality of product
Delivery time	Product may not finish in time	Development team work as hard as possible to finish before deadline
Error when execute	Program will not run	Tester always check to make sure the quality of product

Table 6.2 - Product Risks

6.3 Business Risks

Risks	Risks Description	Solution to avoid
Customer out of budget	Customer doesn't have ability to continue the project.	Our project use sprint cycles methods from each tasks, so customer can be pay at time they want to stop and still has the useful program with all tasks which are developed before the risk
Technology change	Customer require to run this program in different environment	Customer should pay more and decide to continue the current project or not
Program Competitor	The same program is marketed before project finish	Customer should have the plan to continued or discontinued, also the solution when project won't allow to keep going .

Table 6.3 - Business Risks