SOFTWARE DESIGN DOCUMENT

Version 1.1

17.01.2013

MOBCOLL PROJECT

Prepared By: ANDIOS

Murat Öksüzer

Sercan Çidem

Vedat Şahin

Fatih Osman Seçmen

Change History

*A - ADDED M - MODIFIED D – DELETED

VERSION NUMBER	DATE	NUMBER OF FIGURE, TABLE OR PARAGRAPH	A* M D	TITLE OF BRIEF DESCRIPTION
1.0	03.12.2012			Original
1.1	17.01.2013	3.2.8 3.2.9 Figure 24 Figure 25 Figure 26	A,	Synchronization Service Component, Local Database Management Component, Sequence Diagram and Class Diagram of Synchronization Service and Class Diagram of Local Database Management System are added
		3.2 Figure 29 -37	M,	Decomposition Description and User Interface Screens are modified

Preface

This document contains the system design information for MOBCOLL project. The document is prepared according to the "IEEE Standard for Information Technology – Systems Design – Software Design Descriptions – IEEE Std 1016 – 1998".

This Software Design Documentation provides a complete description of all the system design and views of MOBCOLL Project.

The first section of this document includes purpose, scope, overview, reference material and definitions and abbreviations of the project.

The second chapter of this document includes an overview of the functionality of the application. It describes the informal design generally.

The third chapter of this document will give the user a detailed description of each function of the system.

The fourth chapter of this document contains data design and data description of the project.

The fifth chapter of this document contains a general overview of what the user interface will look like.

The sixth and last chapter of this document includes of requirements matrix of the project.

Table of Contents

1. Introduction 6

1.1 Purpose 6

1.2 Scope 6

1.3 Overview 6

1.4 Reference Material 7

1.5 Definitions and Abbreviations 7

2. System Overview 7

2.1 Technologies Used 7

2.2 Application Overview 8

2.3 Design Languages 8

3. System Architecture 8

3.1 Architectural Design 8

3.2 Decomposition Description 9

3.2.1. Login and Logout Component 9

3.2.2 Setting Component 13

3.2.3 Social Component 16

3.2.4 Trade Component 18

3.2.5 Main Functionality Component 19

3.2.6 Data Service Component 23

3.2.7 Remote Server Component 24

3.2.8 Synchronization Service Component 28

3.2.9 Local Database Management Component 30

3.3 Design Rationale 31

4. Data Design 32

4.1 Data Description 32

4.1.1 Data Objects 32

4.1.2. Relationships and Complete Data Model 35

4.2 Data Dictionary 35

5. Human Interface Design 38

5.1 Overview of User Interface 38

5.2 Screen Objects and Actions 38
5.2.1 Register Screen 38
5.2.2 Login Screen 39
5.2.3 Logout Screen 39
5.2.4 User Profile Screen 39
5.2.5 My Book List Screen 40
5.2.6 Add New Book 41
5.2.7 View Profiles 41
5.2.8 History 42
5.2.9 Settings 43
5.2.10 Book Market 44
5.2.11 Search Book 45
6. Requirements Matrix 45

Table of Figures

Figure 1 - General Deployment Diagram 9
Figure 2 - Sequence Diagram of LoginHandler 10
Figure 3 - Sequence Diagram of LogoutHandler 10
Figure 4 - Sequence Diagram of RegisterHandler 10
Figure 5 - Sequence Diagram of UpdateHandler 11
Figure 6 - Class Diagram of Login-Logout Component 12
Figure 7 - Sequence Diagram of SettingsManager 13
Figure 8 – Class Diagram of Setting Component 14
Figure 9 - Sequence Diagram of socialShare 16
Figure 10 - Sequence Diagram of addFriend 17
Figure 11 – Class Diagram of Social Manager 17
Figure 12 - Sequence Diagram of TradeManager 18
Figure 13 – Class Diagram of TradeManager 18
Figure 14 - Sequence Diagram of ViewController 19
Figure 15 – Class Diagram of ViewController 20
Figure 16 - Sequence Diagram of addCoverOfBook 21
Figure 17 – Class Diagram of Create Controller 21
Figure 18 - Sequence Diagram of BarcodeScanner 22
Figure 19– Class Diagram of Barcode Scanner 23
Figure 20 – Sequence Diagram of Data Service Component 23
Figure 21 – Class Diagram of Data Service Component 24
Figure 22 – Sequence Diagram of Remote Service Component 24
Figure 23 – Class Diagram of Remote Service Component 25
Figure 24 - Class Diagram of SynchronizationService Component 29
Figure 25 - Sequence Diagram between client and SynchronizationService 30
Figure 26 – Class Diagram of Local Database Management Component 30
Figure 27 – ER Diagram 35
Figure 28 – Register Screen of Application 39
Figure 29 – Login Screen of Application 39
Figure 30 – User Profile Screen of Application 40
Figure 31 – My BookList Screen of Application 40
Figure 32 – Add New Book Screen of Application 41
Figure 33 – View Profiles Screen of Application 42
Figure 34 – History Screen of Application 43
Figure 35 – Settings Screen of Application 43
Figure 36 – Book Market Screen of Application 44
Figure 37 – Search Book Screen of Application 45

1. Introduction

1.1 Purpose

This document describes the conceptual design of the MOBCOLL Project according to the document guidelines presented in the IEEE 1016-1998 Recommended Practice for Software Design Descriptions (SDD).

The SDD shows how the software system will be structured to satisfy the requirements identified in the software requirements specification. It is a translation of requirements into a description of the software structure, software components, interfaces and data necessary for the implementation phase. In essence, the SDD becomes a detailed blueprint for the implementation activity. In a complete SDD, each requirement must be traceable to one or more design entities.

1.2 Scope

This project, MOBCOLL will be an android application of a book collection for an owner. The system will help the owner to do followings;

- Manage the information about owned books
- Track the place of books on the shelves
- Search the item with barcode or cover picture taken by the device on repository or web
- Keeping the statistical history of books borrowed or lent.
- Web shopping, gift requests and wanted item management

1.3 Overview

The purpose of this document is to help the reader visualize the solution to the project presented. This document verifies how the design meets the requirements stipulated in the SRS document through design viewpoints. The design viewpoints will cover all design elements presented before.

By using information from IEEE 1016-1998, this document will provide a direct approach to the development of this project hence reducing feature creep and pointedly determine the quality of the design.

1.4 Reference Material

IEEE, *IEEE Std 1016-1998 Recommended Practice for Software Design Descriptions*, 1998-09-23, The Institute of Electrical and Electronics Engineers, Inc., (IEEE)

IEEE, *IEEE 1016* Software Design Document (SDD) Template for CENG491

1.5 Definitions and Abbreviations

Term	Definition
Software Design Description	A document that completely describes all of the function of a proposed system and the constraints under which it must operate.
Eclipse IDE	A multi-language software development environment
Zxing	An open-source, multi-format 1D/2D barcode image processing library implemented in Java.
Jersey	Open-source JAX-RS (JSR 311) Reference Implementation for building RESTful Web services
JSON	A lightweight data-interchange format(JavaScript Object Notation)
RESTful	An approach for getting information content from a Web site by reading a designed Web page that contains an XML.
Android	A Linux based mobile phone operating system developed by Google.
Balsamiq Mockups	A graphical user interface mockup builder application.
SRS	A document that completely describes all of the functions of a proposed system and the constraints under which it must operate. For example, this document. (System Requirements Specification)

2. System Overview

2.1 Technologies Used

The system is coded with Java programming language by using Eclipse integrated development environment. We will use MySQL for database. Android SDK will be used for android application development and Jersey web service library will be used for RESTFUL web service.

2.2 Application Overview

The main goal of the project is to enable user to create a virtual library owned by him/her. The final product of MOBCOLL project will be a mobile platform for Android mobile phones or tablet systems that will enable third party mobile application developers to easily develop Android based collection applications by utilizing the common Android services.

The Mobcoll project will be an android application that is designed for users who want to keep the list of owned books. This project gives the user to an application account and by using this, user can use functionalities of the application easily. For example, Mobcoll project gives opportunities user to add new friends, add new books, recommend some of his/her books to friends, sell or buy a book and so on.

2.3 Design Languages

Unified Modeling Language (UML) 5.02 is used for graphical representations of viewpoints in MOBCOLL Project in 3.System Architecture, 4.Data Design and 5.Component Design parts. Balsamiq Mockups is used for user interface design in sixth section of this project.

3. System Architecture

3.1 Architectural Design

The Figure 1 shows the general deployment diagram of the MobColl. The MobColl

Android Application will be developed as a single Android Client (apk) and it will:

- provide the common services for default MobColl functions,

- communicate with the remote server to retrieve or update user's information.

- interact user with system.

RESTFUL Web Service will be deployed as a web application. It consist of RESTFUL web service functionality to response user request. It will act as a bridge between Android application and database. It is supposed to manage database operations like CRUD.

Database will contain necessary data for users, their book lists, etc.

The connection between these component will be carried on internet with HTTP protocol.

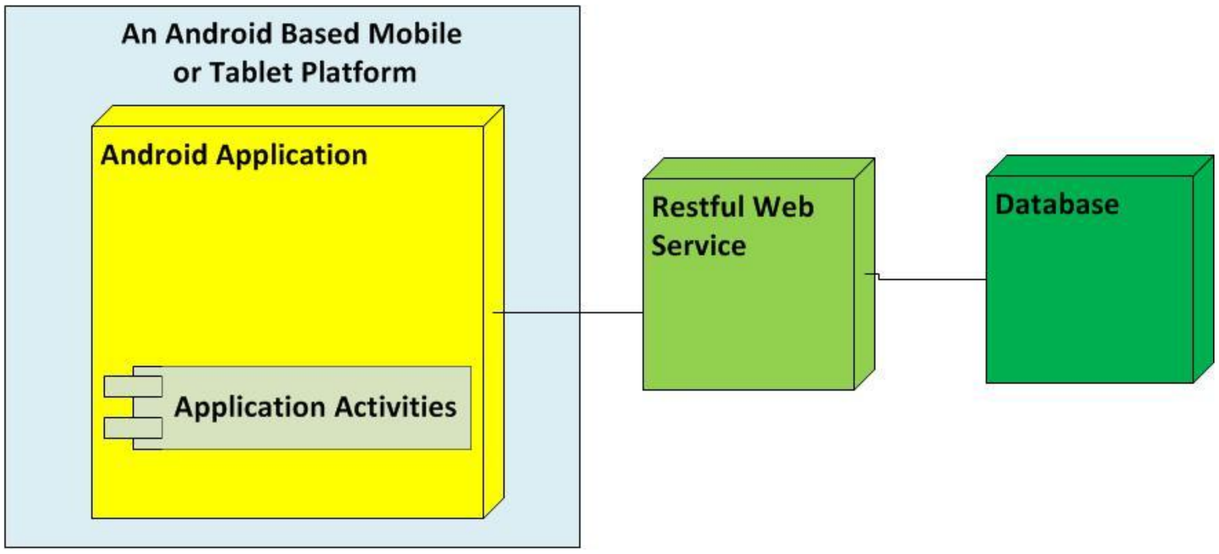


Figure 1 - General Deployment Diagram

3.2 Decomposition Description

3.2.1. Login and Logout Component

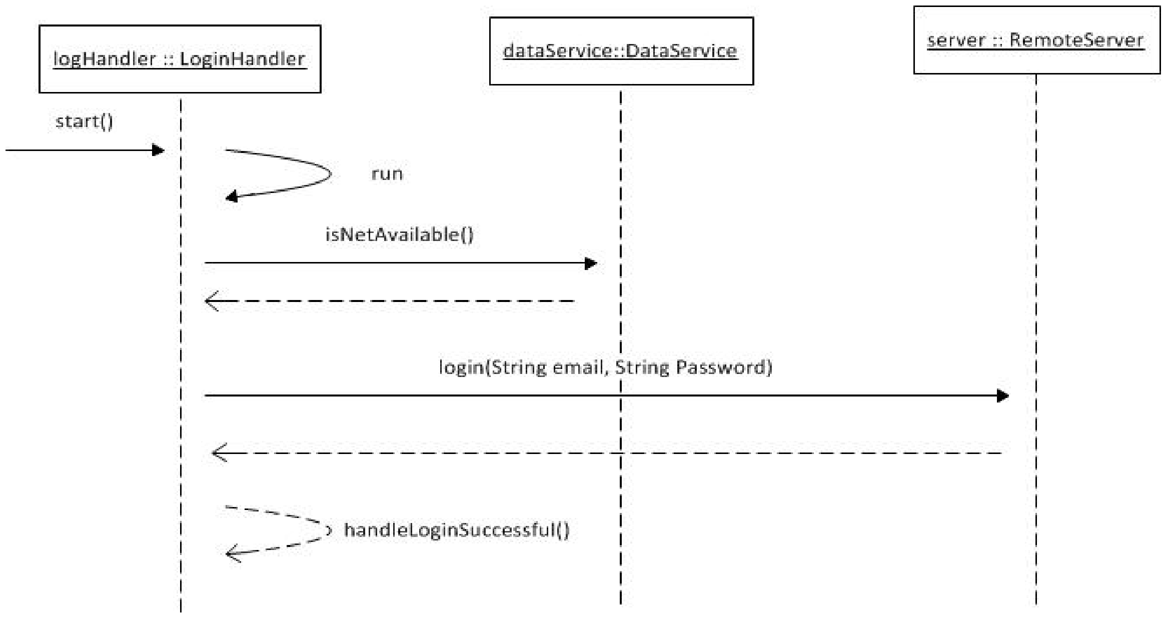


Figure 2 - Sequence Diagram of LoginHandler

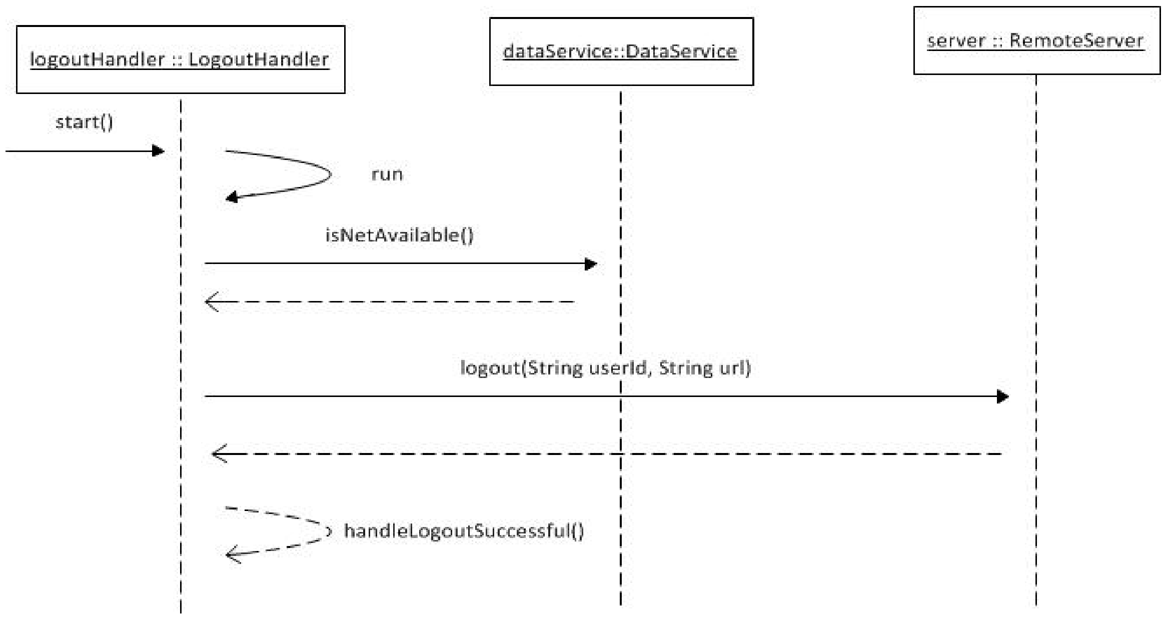


Figure 3 - Sequence Diagram of LogoutHandler

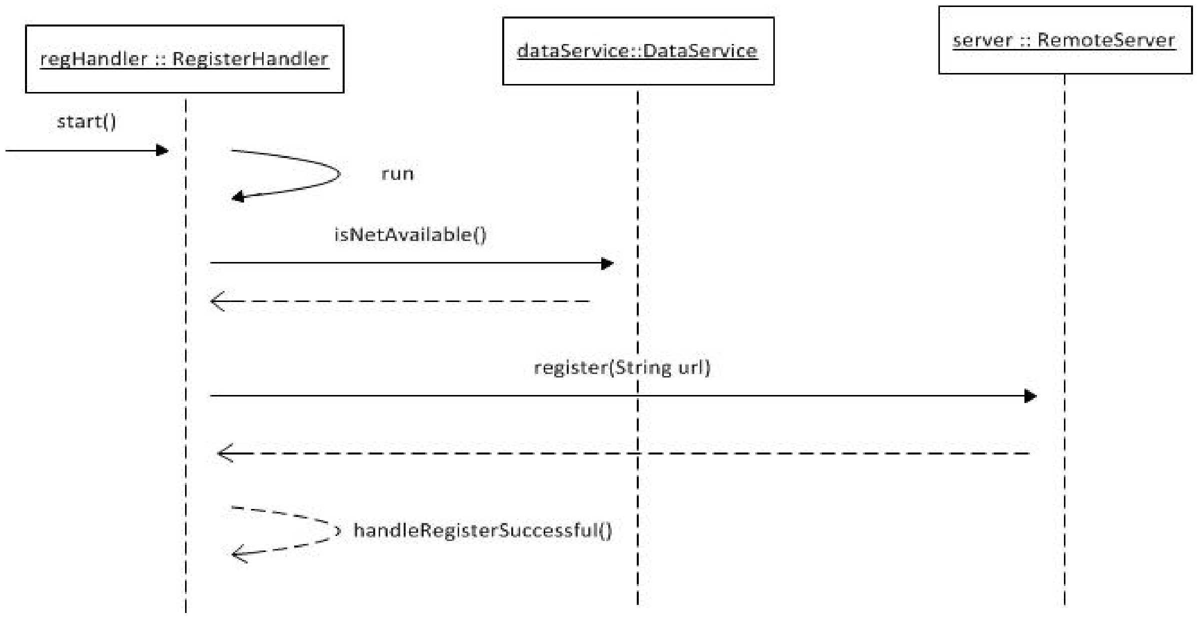


Figure 4 - Sequence Diagram of RegisterHandler

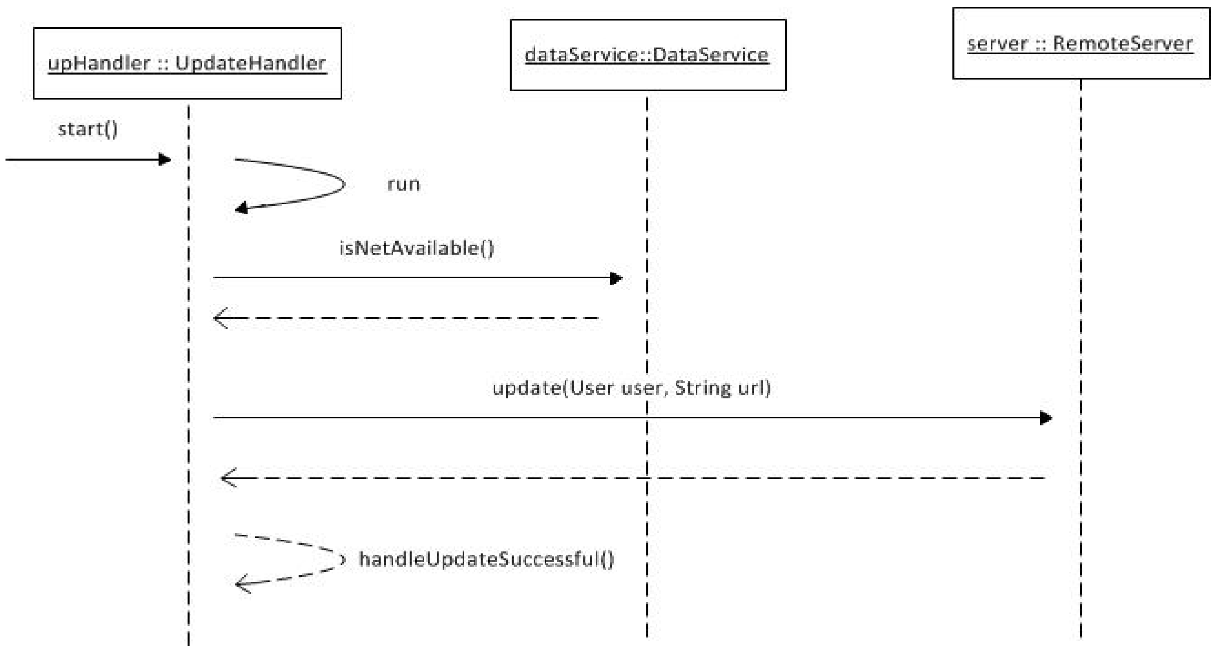
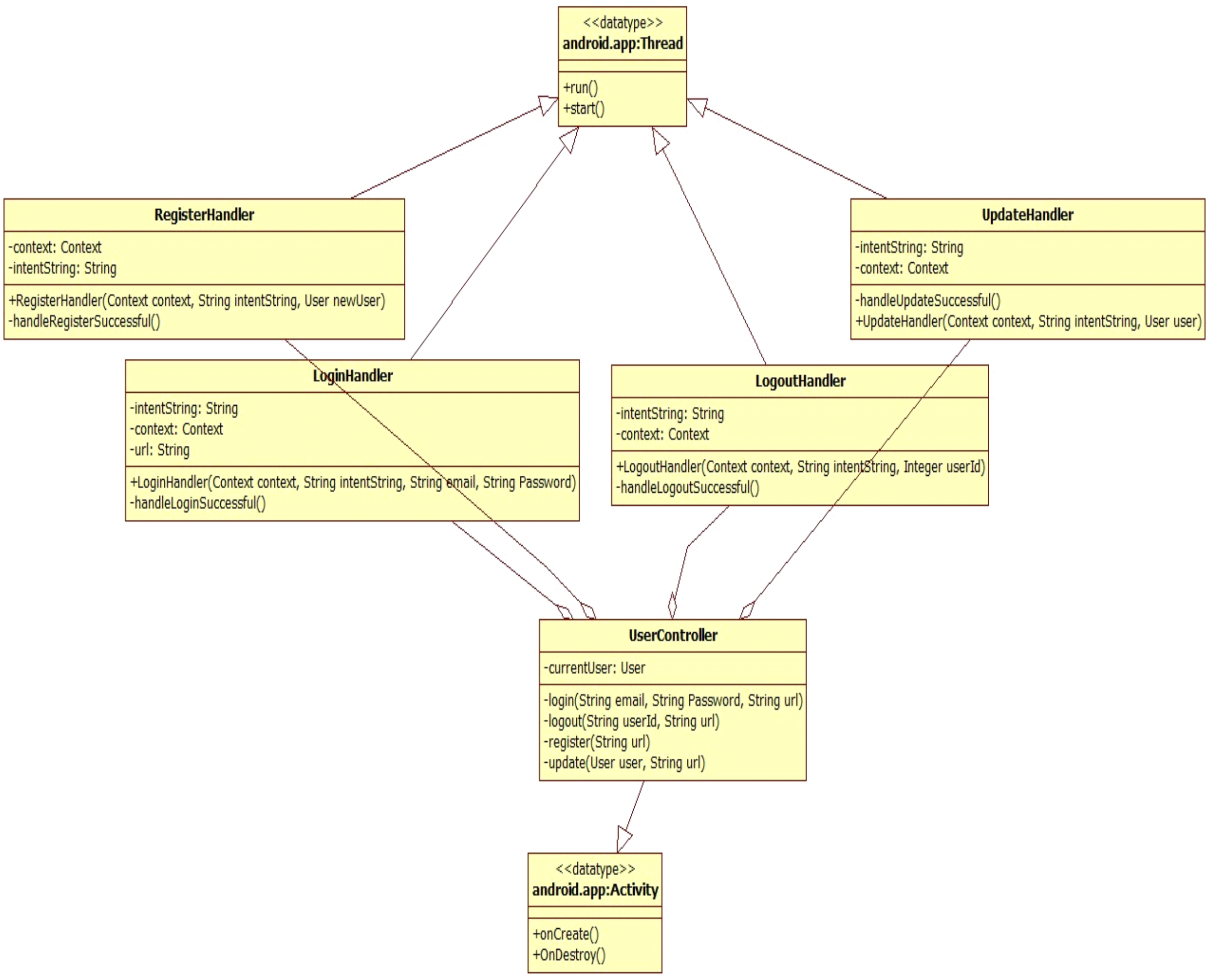


Figure 5 - Sequence Diagram of UpdateHandler

In MobColl Application, data exchange between remote server and android client for login, logout, register and update operations is achieved over JSON. For these operations, there will be four handler classes named LoginHandler, LogoutHandler, UpdateHandler and RegisterHandler. These classes are exactly the same classes shown in Figure 1. For login operation, system sends username, password and URL to the remote Server and receives a response. Which method will handle our request from Android client and give response to this request will be determined by this URL. The mechanism is just the same for logout, register and update operation. Figure 2, 3, 4 and 5 shows the sequence diagrams of these four operations.



<<datatype>>
android.app:Thread
+run()
+start()
RegisterHandler
-context: Context
-intentString: String
+RegisterHandler(Context context, String intentString, User newUser)
-handleRegisterSuccessful()
UpdateHandler
-intentString: String
-context: Context
-handleUpdateSuccessful()
+UpdateHandler(Context context, String intentString, User user)
LoginHandler
-intentString: String
-context: Context
-url: String
+LoginHandler(Context context, String intentString, String email, String Password)
-handleLoginSuccessful()
LogoutHandler
-intentString: String
-context: Context
+LogoutHandler(Context context, String intentString, Integer userId)
-handleLogoutSuccessful()
UserController
-currentUser: User
-login(String email, String Password, String url)
-logout(String userId, String url)
-register(String url)
-update(User user, String url)
<<datatype>>
android.app:Activity
+onCreate()
+OnDestroy()

Figure 6 - Class Diagram of Login-Logout Component

LoginHandler: Thread to be run when an account is being logged in. User should login automatically or manually according to the settings that user previously defined. After the successful login of the user, the normal operation sequence of the service will continue.

LogoutHandler: Thread to be run when an account is being logged out. After the successful logout of the user, this finishes the application.

RegisterHandler: Thread to be run when user creates a new account.

UpdateHandler: Thread to be run when user updates his/her account.

3.2.1.1 Login and Logout Component Design

LoginHandler:

It inherits Thread class. When we call “login” in “UserController”, we create a LoginHandler and we use “start” function in LoginHandler class. This function invokes the “run” function. In “run” function, we send a login request to remote server. “handleLoginSuccessful” function determines whether the login operation is successful or not. The remaining handler functions work similar.

UserController

This class is used for account operations in android application. When a user logs in, current user data is kept in “currentUser” variable. Other components use this variable to execute their operations. It consists of four functions listed below:

- login(String email, String password, String url) : Used for logging in to system. It uses LoginHandler.
- logout(String email, String url) : Used for logging out from system. It uses LogoutHandler.
- register(User user, String url) : Used for registering the user. It uses RegisterHandler.
- update(User user, String url) : User for updating the information of the user. It uses UpdateHandler.

3.2.2 Setting Component

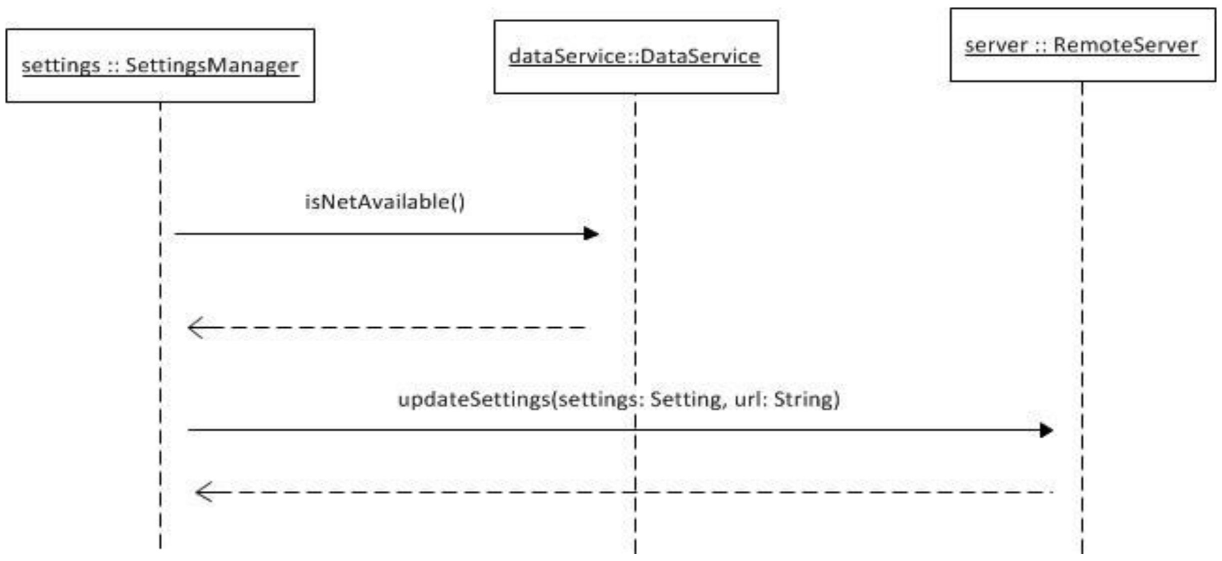

settings :: SettingsManager
dataService::DataService
server :: RemoteServer
isNetAvailable()
updateSettings(settings: Setting, url: String)

Figure 7 - Sequence Diagram of SettingsManager

In our project, between SettingsManager and RemoteServer, there will be a data flow consists of settings of the user. This sequence diagram shows how settings are saved to the server. These settings are sent to the RemoteServer if there is an available connection. Moreover, an update operation that changes settings of the user will be done via this component. The settings that will be available for change and their descriptions are listed below:

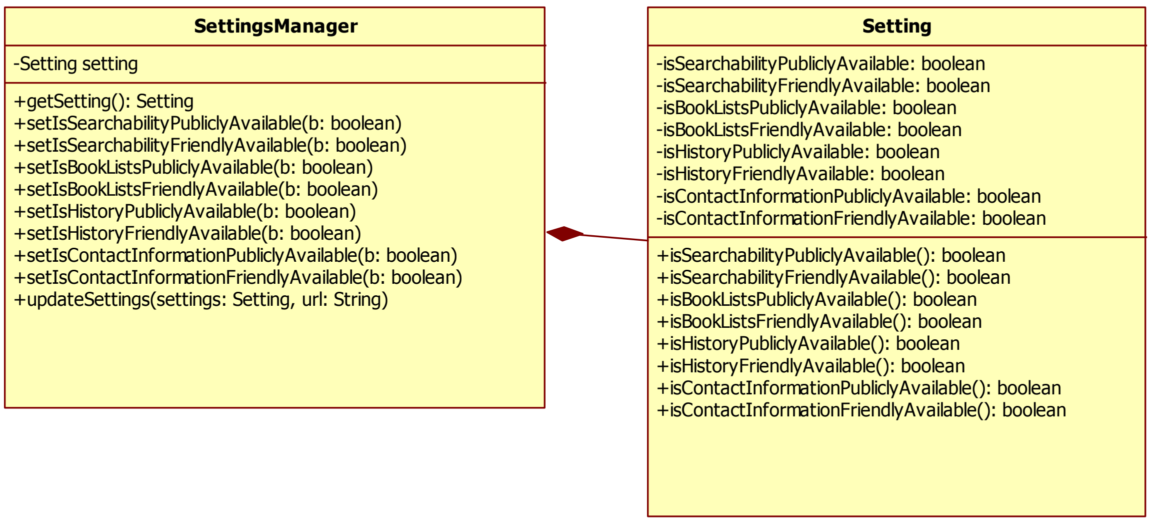

SettingsManager
-Setting setting
+getSetting(): Setting
+setIsSearchabilityPubliclyAvailable(b: boolean)
+setIsSearchabilityFriendlyAvailable(b: boolean)
+setIsBookListsPubliclyAvailable(b: boolean)
+setIsBookListsFriendlyAvailable(b: boolean)
+setIsHistoryPubliclyAvailable(b: boolean)
+setIsHistoryFriendlyAvailable(b: boolean)
+setIsContactInformationPubliclyAvailable(b: boolean)
+setIsContactInformationFriendlyAvailable(b: boolean)
+updateSettings(settings: Setting, url: String)
Setting
-isSearchabilityPubliclyAvailable: boolean
-isSearchabilityFriendlyAvailable: boolean
-isBookListsPubliclyAvailable: boolean
-isBookListsFriendlyAvailable: boolean
-isHistoryPubliclyAvailable: boolean
-isHistoryFriendlyAvailable: boolean
-isContactInformationPubliclyAvailable: boolean
-isContactInformationFriendlyAvailable: boolean
+isSearchabilityPubliclyAvailable(): boolean
+isSearchabilityFriendlyAvailable(): boolean
+isBookListsPubliclyAvailable(): boolean
+isBookListsFriendlyAvailable(): boolean
+isHistoryPubliclyAvailable(): boolean
+isHistoryFriendlyAvailable(): boolean
+isContactInformationPubliclyAvailable(): boolean
+isContactInformationFriendlyAvailable(): boolean

Figure 8 – Class Diagram of Setting Component

Setting Class:

We will use setting class in SettingManager to handle managerial incidents. This class has following attributes:

-boolean isSearchabilityPubliclyAvailable: Indicates whether the user can be found public user searches.
-boolean isSearchabilityFriendlyAvailable: Indicates whether the user can be found friends' user searches.
-boolean isBookListsPubliclyAvailable: Indicates whether the user shares own book list with public.
-boolean isBookListsFriendlyAvailable: Indicates whether the user shares own book list with

friends.
-boolean isHistoryPubliclyAvailable: Indicates whether the user shares his/her history with public.
-boolean isHistoryFriendlyAvailable: Indicates whether the user shares his/her history with friends.
-boolean isContactInformationPubliclyAvailable: Indicates whether the user shares his/her contact information with public.
-boolean isContactInformationFriendlyAvailable: Indicates whether the user shares his/her contact information with friends.

3.2.2.1 Setting Component Design

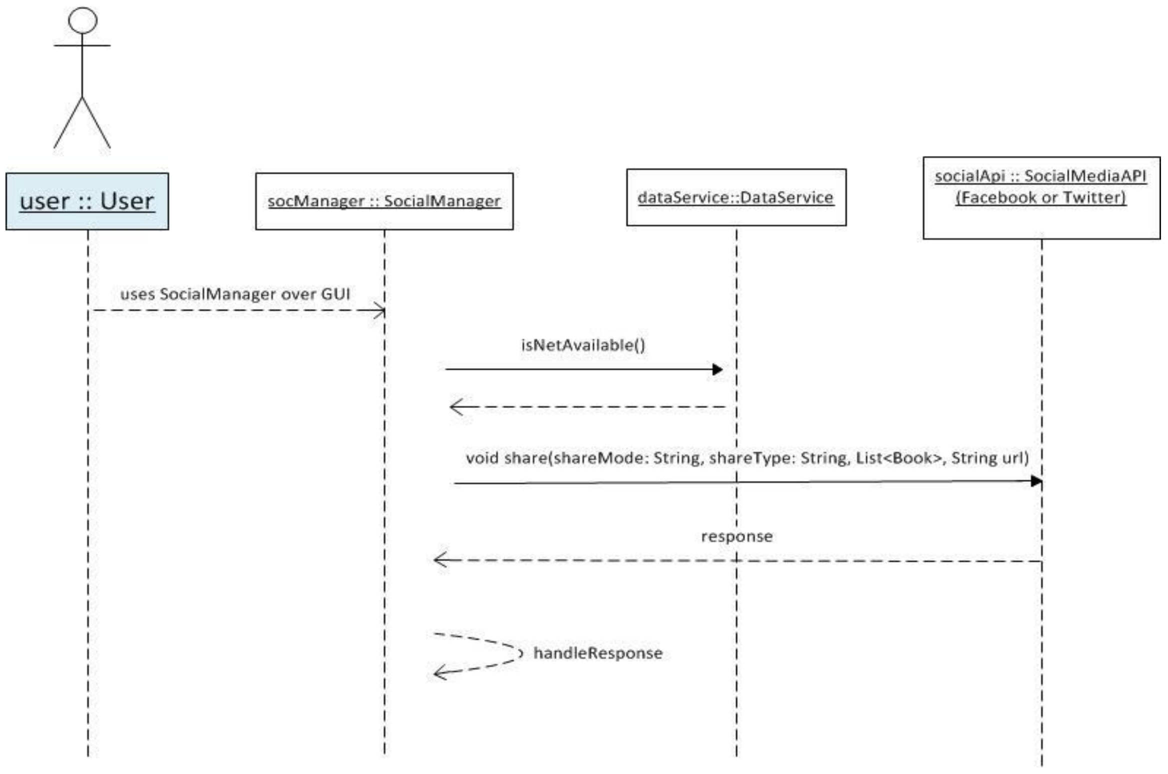
We will use SettingManager for managing settings. There is a private “setting” property in this class. This represents data model of “settings” table in ER-Diagram. There are some public functions in this class:

- getSetting() : Returns setting.
- setIsSearchabilityPubliclyAvailable(): Used for whether the user can be found public user searches.
- setIsSearchabilityFriendlyAvailable(): Used for whether the user can be found friends' user searches.
- setIsBookListsPubliclyAvailable(): Used for whether the user shares own book list with public.
- setIsBookListsFriendlyAvailable(): Used for whether the user shares own book list with his/her friends.
- setIsHistoryPubliclyAvailable(): Used for whether the user shares his/her history with public.
- setIsHistoryFriendlyAvailable(): Used for whether the user shares his/her history with his/her friends.
- setIsContactInformationPubliclyAvailable(): Used for whether the user shares his/her contact information with public.
- setIsContactInformationFriendlyAvailable(): Used for whether the user shares his/her

contact information with his/her friends.

- updateSettings(settings: Setting, url: String): Used to update user information on the server.

3.2.3 Social Component



user :: User
socManager :: SocialManager
dataService::DataService
socialApi :: SocialMediaAPI (Facebook or Twitter)
uses SocialManager over GUI
isNetAvailable()
void share(shareMode: String, shareType: String, List<Book>, String url)
response
handleResponse

Figure 9 - Sequence Diagram of socialShare

This sequence diagram shows sharing a book list on social media like Facebook or twitter. The application will use SocialMedia API for Facebook or Twitter. With this API, sharing list of a user's book is handled by sending the information of the audience, type of the list, list of books and a link. Also, adding or deleting a friend and viewing profile of friends will also be handled by this component. The sequence diagram of addFriend is shown below and deleteFriend is the same as this. After this diagram, the SocialMedia class, its methods and their definitions are listed and explained.

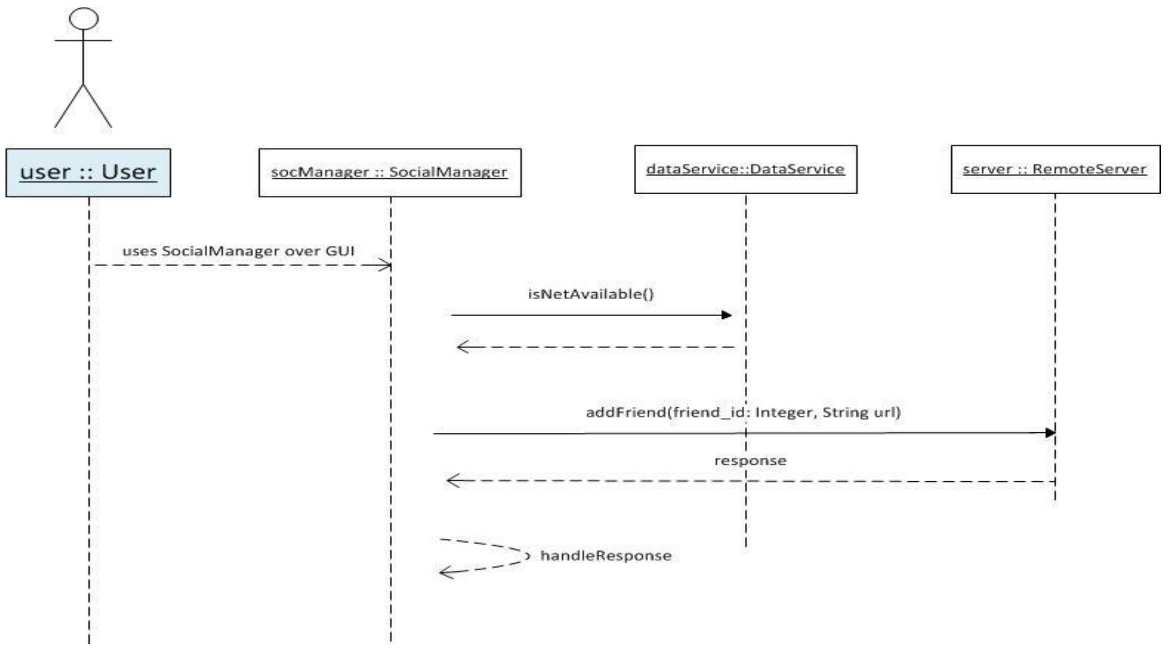


Figure 10 - Sequence Diagram of addFriend

This sequence diagram shows friend adding functionality. Remaining functions are similar.

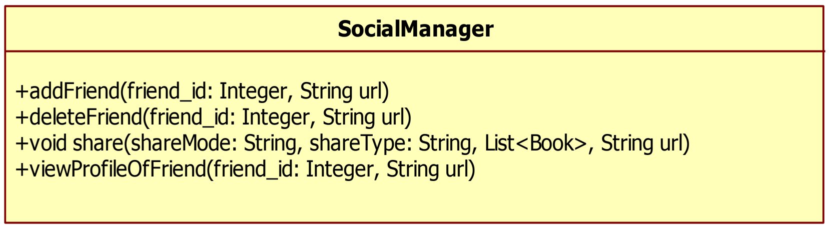

SocialManager
+addFriend(friend_id: Integer, String url)
+deleteFriend(friend_id: Integer, String url)
+void share(shareMode: String, shareType: String, List<Book>, String url)
+viewProfileOfFriend(friend_id: Integer, String url)

Figure 11 – Class Diagram of Social Manager

3.2.3.1 Social Component Design

We will use SocialManager to handle social functionalities. There are 4 public functions in this class:

- addFriend(int friend_id, string url): With this function, a user sends friend request to another user. If the requested user accepts the invitations, they become friends in database. This function has one parameter that represents requested user's id.

- deleteFriend(int friend_id, string url): With this function, a user deletes a friend. The function has one argument representing the id of the user to be deleted.

- share (shareType: String, list: List<Book>, string url) : ShareType determines which book list will be shared for example, gift request book list.

- viewProfileOfFriends(int friend_id, string url): This function shows up the profile page of a user who's id is given as a parameter to this function.

3.2.4 Trade Component

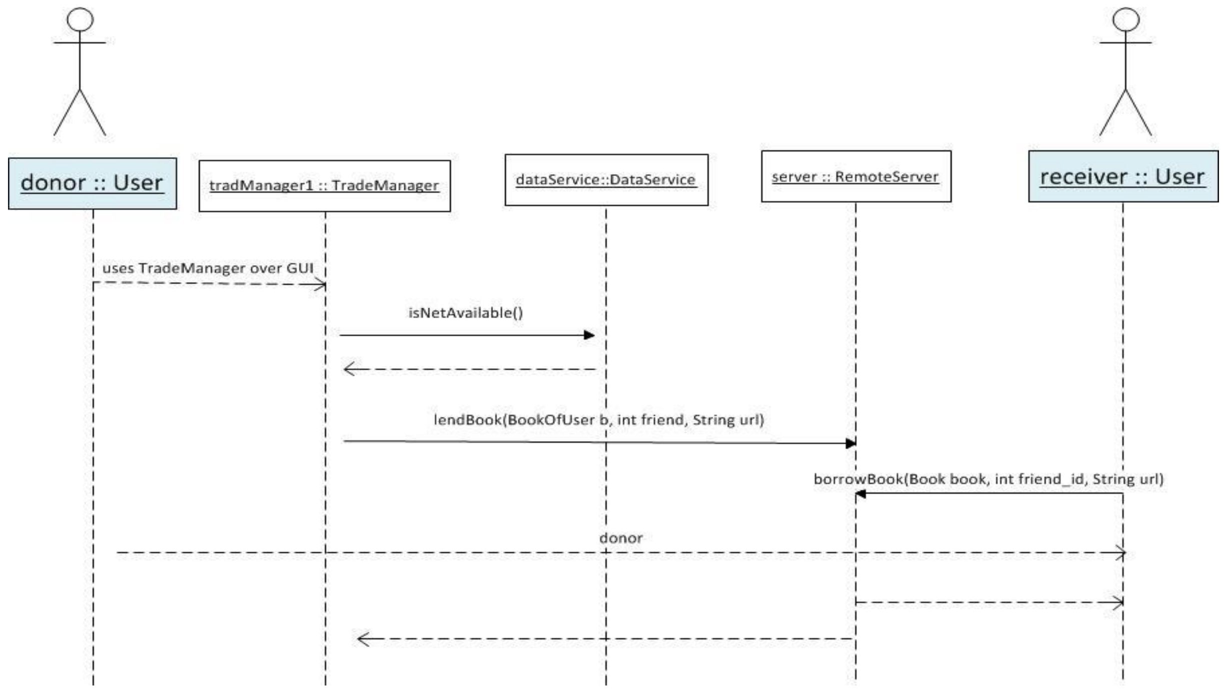

donor :: User
tradManager1 :: TradeManager
dataService::DataService
server :: RemoteServer
receiver :: User
uses TradeManager over GUI
isNetAvailable()
lendBook(BookOfUser b, int friend, String url)
borrowBook(Book book, int friend_id, String url)
donor

Figure 12 - Sequence Diagram of TradeManager

We used actors for a better understanding. In real, this event occurs when two people come together physically. We showed here lending and borrowing a book. Selling and buying transactions are the same with this representation, so we did not draw it again

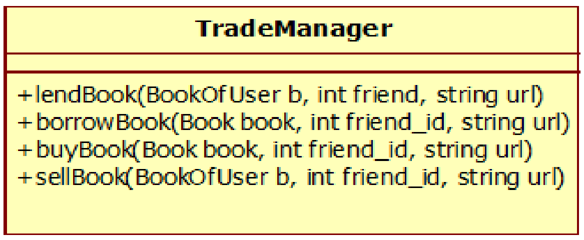

TradeManager
+lendBook(BookOfUser b, int friend, string url)
+borrowBook(Book book, int friend_id, string url)
+buyBook(Book book, int friend_id, string url)
+sellBook(BookOfUser b, int friend_id, string url)

Figure 13 – Class Diagram of TradeManager

3.2.4.1 Trade Component Design

We will user TradeManager to perform trade incidents. There are 4 public functions this class:

- lendBook (BookOfUser b, int friend, string url): This function keeps track of lending books own books to a friend.
- borrowBook(Book book, int friend_id, string url): With this function, a user can borrow a book from his/her friend founded by the friend_id.
- buyBook(Book book, int friend_id, string url): With this function, a user can buy a book from his/her friend founded by the friend_id.
- sellBook(BookOfUser b, int friend_id, string url): With this function, a user can sell a book to a user founded by the friend_id.

3.2.5 Main Functionality Component

3.2.5.1 ViewController

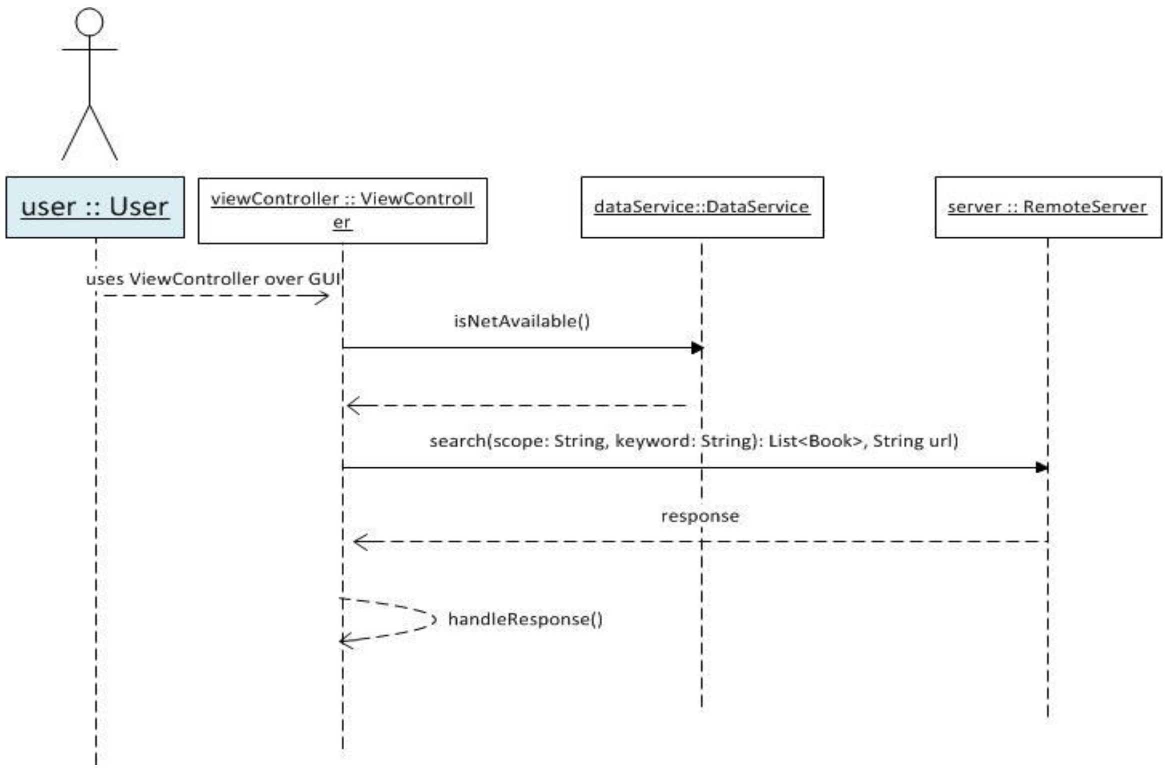

user :: User
viewController :: ViewController
dataService::DataService
server :: RemoteServer
uses ViewController over GUI
isNetAvailable()
search(scope: String, keyword: String): List<Book>, String url)
response
handleResponse()

Figure 14 - Sequence Diagram of ViewController

This sequence diagram shows how ViewController and RemoteServer are connected each other. If internet connection is available, user would control some operations like lending/borrowing book, requesting or searching a book etc. by this controller. The ViewController class has the methods to handle these operations. They are listed below and explained one by one.

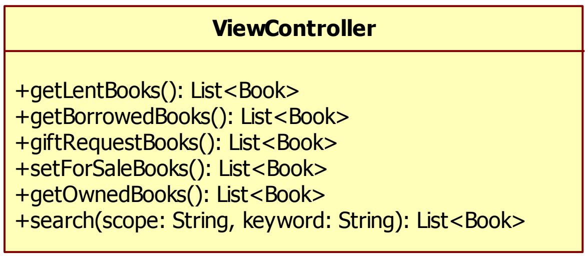

ViewController
+getLentBooks(): List<Book>
+getBorrowedBooks(): List<Book>
+giftRequestBooks(): List<Book>
+setForSaleBooks(): List<Book>
+getOwnedBooks(): List<Book>
+search(scope: String, keyword: String): List<Book>

Figure 15 – Class Diagram of ViewController

3.2.5.1.1 ViewController Component Design

This class is used to view some lists created by a user like owned books, lent books, gift request books etc. The class consists of six functions listed below:

- getLentBooks(): Returns the list of books which are lent to another user.
- getBorrowedBooks(): Returns the list of books which are borrowed from another user.
- giftRequestBooks(): Returns the list of books which are gifted request to some user.
- setForSaleBooks(): Returns the list of books which have selected as for sale before.
- getOwnedBooks(): Returns the list of books which are owned by the user, no operations have been performed on these books like lending, borrowing etc.
- search(scope: String, keyword: String):

3.2.5.2 CreateController

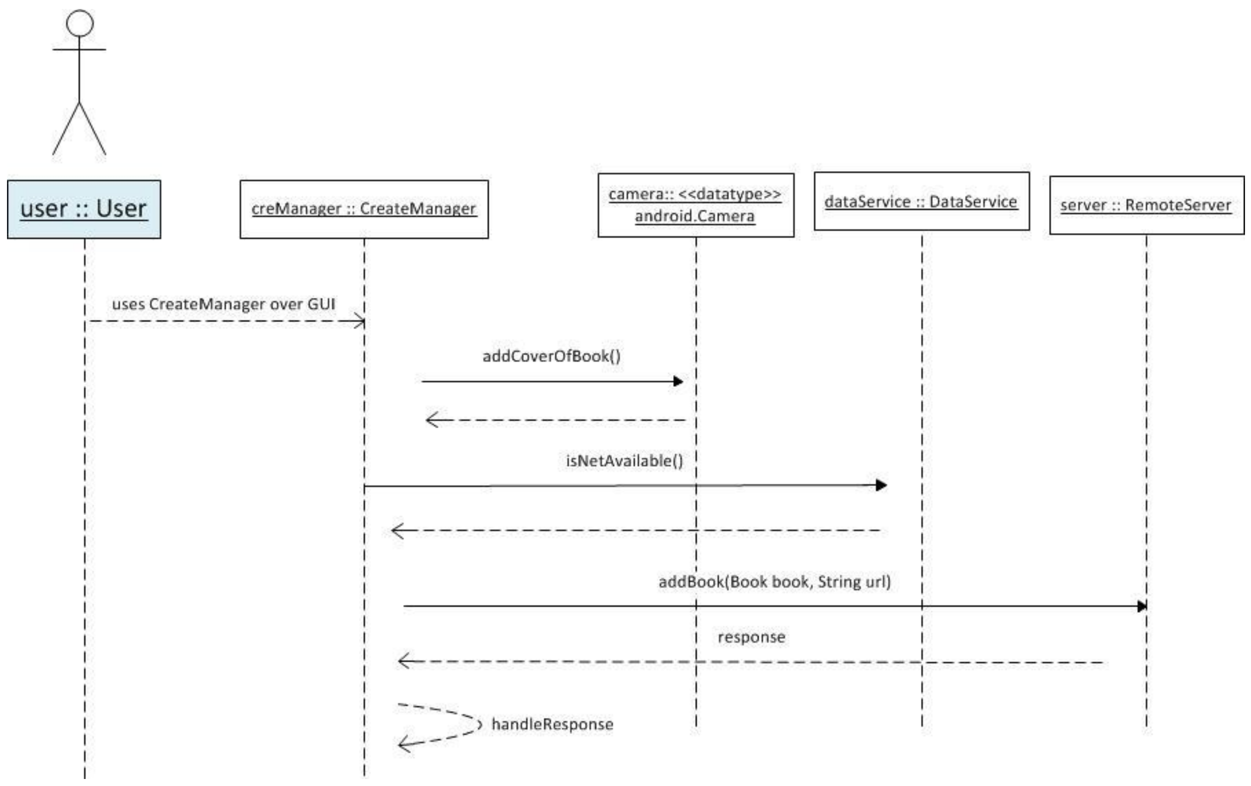

user :: User
creManager :: CreateManager
camera:: <<datatype>> android.Camera
dataService :: DataService
server :: RemoteServer
uses CreateManager over GUI
addCoverOfBook()
isNetAvailable()
addBook(Book book, String url)
response
handleResponse

Figure 16 - Sequence Diagram of addCoverOfBook

CreateManager and RemoteServer are connected via DataService again. Adding book operation is shown in this diagram. While adding new book, cover of the book can be added by taking a photo from the device's camera. Also, setting lent/borrow books or gift request/for sale is handled by this manager. These operations are listed in CreateController which is explained below.

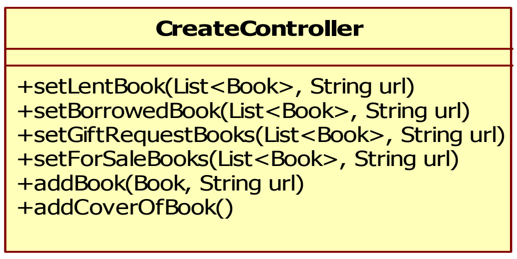

CreateController
+setLentBook(List<Book>, String url)
+setBorrowedBook(List<Book>, String url)
+setGiftRequestBooks(List<Book>, String url)
+setForSaleBooks(List<Book>, String url)
+addBook(Book, String url)
+addCoverOfBook()

Figure 17 – Class Diagram of Create Controller

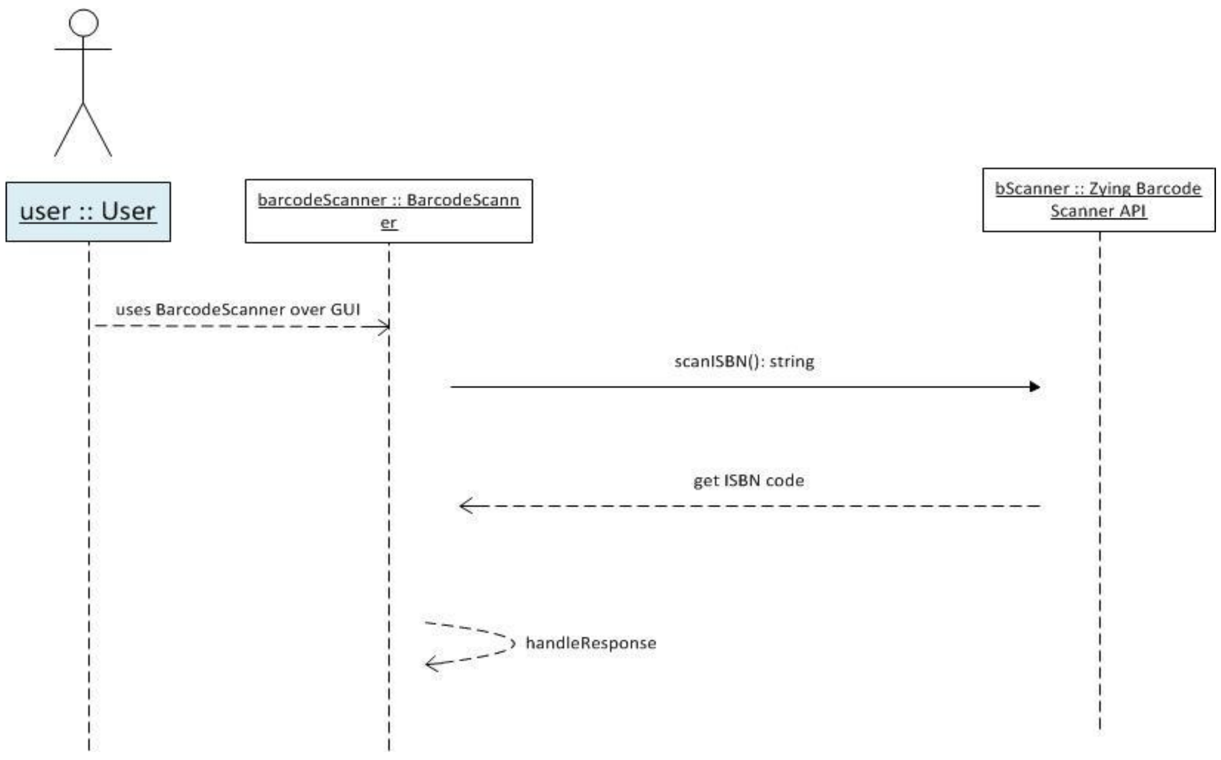
3.2.5.2.1 CreateController Component Design

This class is used to create or edit some lists to group books of a user and in this way, the user can keep track of his/her books in more efficient way. The class has six functions which are following:

- setLentBook(List<Book>, String url) : Edit operations on the list of lent books such as adding or creating.
- setBorrowedBook(List<Book>, String url) : Edit operations on the list of borrowed books.
- setGiftRequestBook(List<Book>, String url) : Edit operations on the list of gift request books.
- addBook(Book, String url) : Used to add a book to owned book list by a user.

addCoverOfBook() : Used to add cover of a book using camera of android device.

3.2.5.3 BarcodeScanner



user :: User
barcodeScanner :: BarcodeScanner
bScanner :: Zying Barcode Scanner API
uses BarcodeScanner over GUI
scanISBN(): string
get ISBN code
handleResponse

Figure 18 - Sequence Diagram of BarcodeScanner

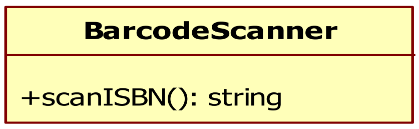


Figure 19– Class Diagram of Barcode Scanner

This class is intended to be used while adding a book. The barcode of book is scanned with the camera of android device so that the scanned ISBN will be searched. Therefore, all information about the book like title, author, category, publish date etc. will be provided. The only function is;

3.2.5.3.1 Barcode Scanner Component Design

- scanISBN(): Return the number gathered from the camera scan operation

3.2.6 Data Service Component

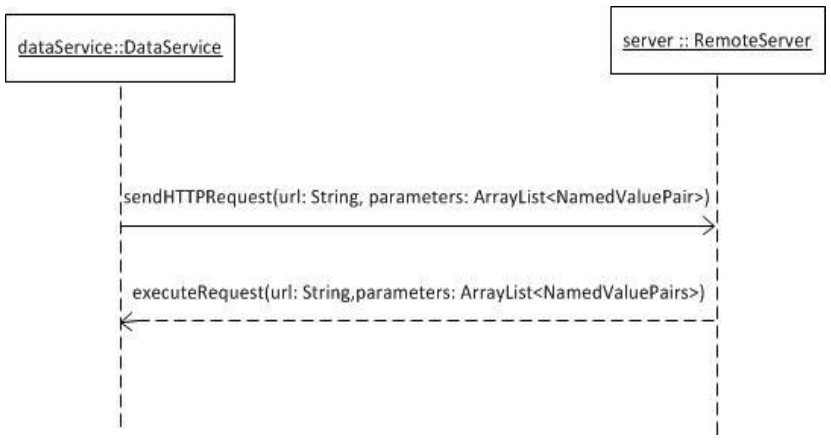

dataService::DataService
server :: RemoteServer
sendHTTPRequest(url: String, parameters: ArrayList<NamedValuePair>)
executeRequest(url: String,parameters: ArrayList<NamedValuePairs>)

Figure 20 – Sequence Diagram of Data Service Component

This diagram shows how DataService and RemoteServer are related with each other. They use HTTP protocol to handle the server connection. The related class is below and it has only two methods which will be explained next.

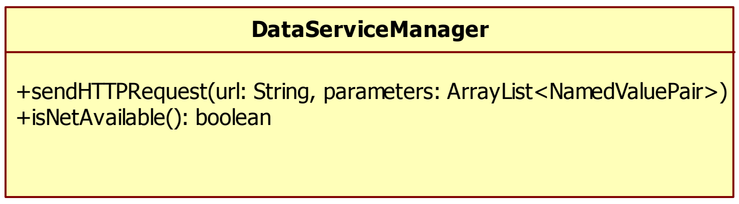

DataServiceManager
+sendHTTPRequest(url: String, parameters: ArrayList<NamedValuePair>)
+isNetAvailable(): boolean

Figure 21 – Class Diagram of Data Service Component

3.2.6.1. Data Service Component Design

sendHTTPRequest(url: String, parameters: ArrayList<NamedValuePair>): This function sends HTTP requests to remote server. These requests contain information about functions of other components.

isNetAvailable(): This function determines whether internet connection is available or not.

3.2.7 Remote Server Component

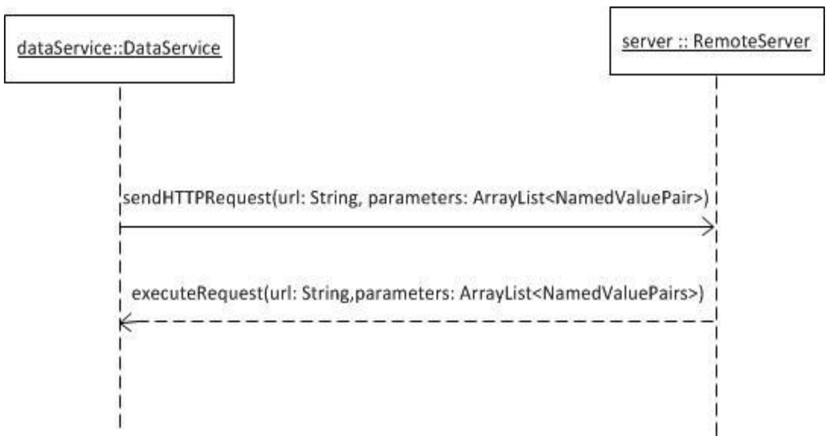

dataService::DataService
server :: RemoteServer
sendHTTPRequest(url: String, parameters: ArrayList<NamedValuePair>)
executeRequest(url: String,parameters: ArrayList<NamedValuePairs>)

Figure 22 – Sequence Diagram of Remote Service Component

This diagram shows how DataService and RemoteServer execute requests. RemoteServer sends request with parameter name and named value pairs while DataService sends HTTP request in response. The related class is RemoteServerServiceManager and its methods are listed and described below:

–

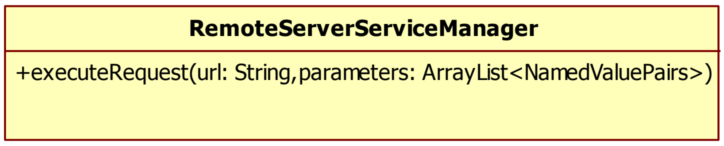

RemoteServerServiceManager
+executeRequest(url: String,parameters: ArrayList<NamedValuePairs>)

Figure 23 – Class Diagram of Remote Service Component

3.2.7.1 Remote Server Component Design

executeRequest(url: String, parameters: ArrayList<NamedValuePairs>): This function executes taken requests with their parameters and return back results to the Data Service Component.

We show "executeRequest" method as template for following methods that we use in all components.

Server side methods:

String loginUser(MultivaluedMap<String, String> params):

The "params" multivalued parameter container includes an email and a password parameter. It checks whether the given email-password pair belongs to a valid, registered user. If it is, then the method returns the id of the user in String type.

If it doesn't valid, then it returns a negative number.

public String registerUser(MultivaluedMap<String, String> params):

The "params" multivalued parameter container includes user name, lastname, email, password, and phone number. It firstly checks whether a user with given email already registered or not. If there is, it returns a negative number. Otherwise, it saves a user entry into "user" table in database and returns id of this entry.

public String addBook(MultivaluedMap<String, String> params):

The "params" multivalued parameter container includes ISBN, title, author, publisher, publishdate, language, category, type, location, and description parameters for book and user id parameter for owner of the book.

This method saves a book entry into “book” table and a book id-user id entry into “books_of_user” table in database.

public List<Book> getUserBookList(MultivaluedMap<String, String> params):

The “params” multivalued parameter container includes a user id parameter for the user whom book list is asked.

This method retrieves book entries which belongs the given user and returns it in a List data structure.

public String setApplicationSettings(MultivaluedMap<String, String> params):

The “params” multivalued parameter container includes a bunch of setting type-boolean pairs representing setting preferences. These setting types determine :

-is searchability publicly available

-is searchability friendly available

-is book list publicly available

-is book list friendly available

-is history publicly available

-is history friendly available

-is contact information publicly available

-is contact information friendly available

This method updates given parameters in the “settings” table in database.

public String addDeleteFriend(MultivaluedMap<String, String> params):

The “params” multivalued parameter container includes a friend user id and a flag parameter for determining whether adding or deleting friend.

For friend adding, this method saves a user id-friend user id pair into “friends_of_user” table in database.

For friend deleting, this method deletes the user id-friend user id pair entry in “friends_of_user” table.

public User getFriendProfileInfo (MultivaluedMap<String, String> params):

The “params” multivalued parameter container includes a friend user id whom profile information is asked. This method retrieves given user information, constructs a User object and returns it.

public String tradeOperations (MultivaluedMap<String, String> params):

The “params” multivalued parameter container includes a book id, a friend user id and a flag parameter for determining the operation. The operation can be lending, borrowing, buying, and selling.

If the operation is lending, the book entry in the “books_of_user” table is updated such that the “is_lent” boolean attribute is set to true.

If the operation is borrowing, an entry which includes book id and owner id is saved into “borrowed_books” table.

If the operation is buying, a book entry is saved into “books_of_user” table.

If the operation is selling, the book entry in the “books_of_user” table is deleted.

public List<Book> getBookLists (MultivaluedMap<String, String> params):

The “params” multivalued parameter container includes a user id parameter for whom book lists are asked, and a flag for determining which book list is asked. Lent, borrowed, requested, for sale, and owned book lists can be asked.

This method retrieves required book entries from database, adds into a List data structure and returns it.

public List<Book> searchBook (MultivaluedMap<String, String> params):

The “params” multivalued parameter container includes a user id parameter for whom book lists are searched. Moreover, it includes a keyword for searching and a scope parameter for determining in which list the search operation will be executed. . Lent, borrowed, requested, for sale, and owned book lists can be searched.

This method searches using given keyword, builds a list of book from search result and returns it.

public List<Book> setBookLists (MultivaluedMap<String, String> params):

The “params” multivalued parameter container includes a user id parameter for whom book lists are set.

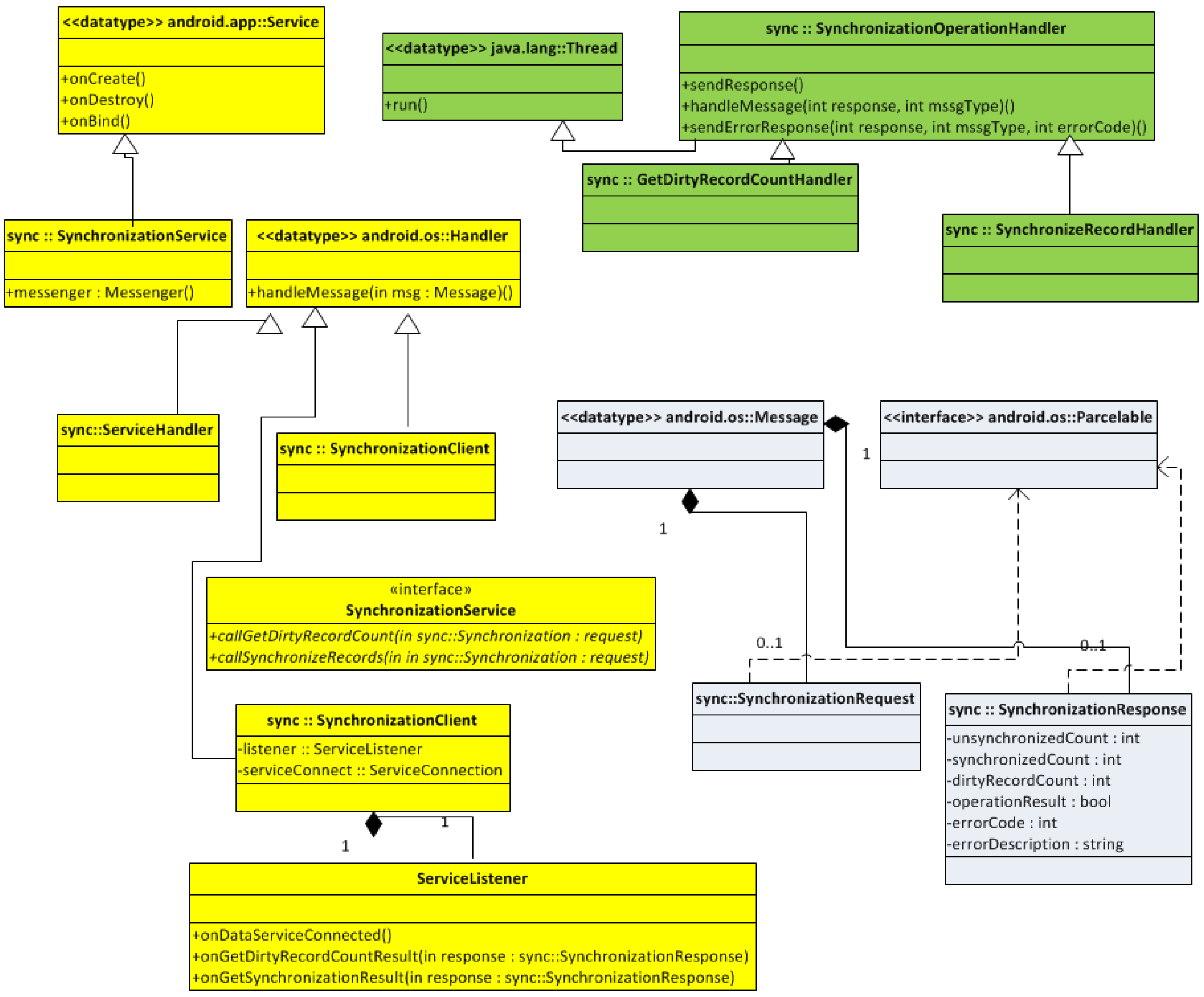
Moreover, it includes a XML string that includes input book list and a flag parameter for determining which list we should set. This list can be lent, borrowed, requested, for sale, or owned book list.

This method parses the input XML string and extracts the list of books. Then, it saves the books into database.

3.2.8 Synchronization Service Component

SynchronizationService is another communication interface as an Android Bound Service between the Mobile Book Collector Android Application and remote server which will enable the android application to check the count of unsynchronized records with remote server and synchronize them manually.

The Figure xxx illustrates the class diagram for SynchronizationService component and related classes. SynchronizationService will also be implemented as an Android Bound Service that will be accessible by Mobil Book Collector Application. Similar to DataService, the SynchronizationClient is the helper class to communicate with the service from another application. Figure 9 shows the sequence diagram to synchronize records with remote Server.



<<datatype>> android.app::Service
+onCreate()
+onDestroy()
+onBind()
<<datatype>> java.lang::Thread
+run()
sync :: SynchronizationOperationHandler
+sendResponse()
+handleMessage(int response, int mssgType)()
+sendErrorResponse(int response, int mssgType, int errorCode)()
sync :: GetDirtyRecordCountHandler
sync :: SynchronizeRecordHandler
sync :: SynchronizationService
+messenger : Messenger()
<<datatype>> android.os::Handler
+handleMessage(in msg : Message)()
sync::ServiceHandler
sync :: SynchronizationClient
<<datatype>> android.os::Message
<<interface>> android.os::Parcelable
1
1
«interface» SynchronizationService
+callGetDirtyRecordCount(in sync::Synchronization : request)
+callSynchronizeRecords(in in sync::Synchronization : request)
0..1
0..1
sync::SynchronizationRequest
sync :: SynchronizationResponse
-unsynchronizedCount : int
-synchronizedCount : int
-dirtyRecordCount : int
-operationResult : bool
-errorCode : int
-errorDescription : string
sync :: SynchronizationClient
-listener :: ServiceListener
-serviceConnect :: ServiceConnection
1
1
ServiceListener
+onDataServiceConnected()
+onGetDirtyRecordCountResult(in response : sync::SynchronizationResponse)
+onGetSynchronizationResult(in response : sync::SynchronizationResponse)

Figure 24 - Class Diagram of SynchronizationService Component

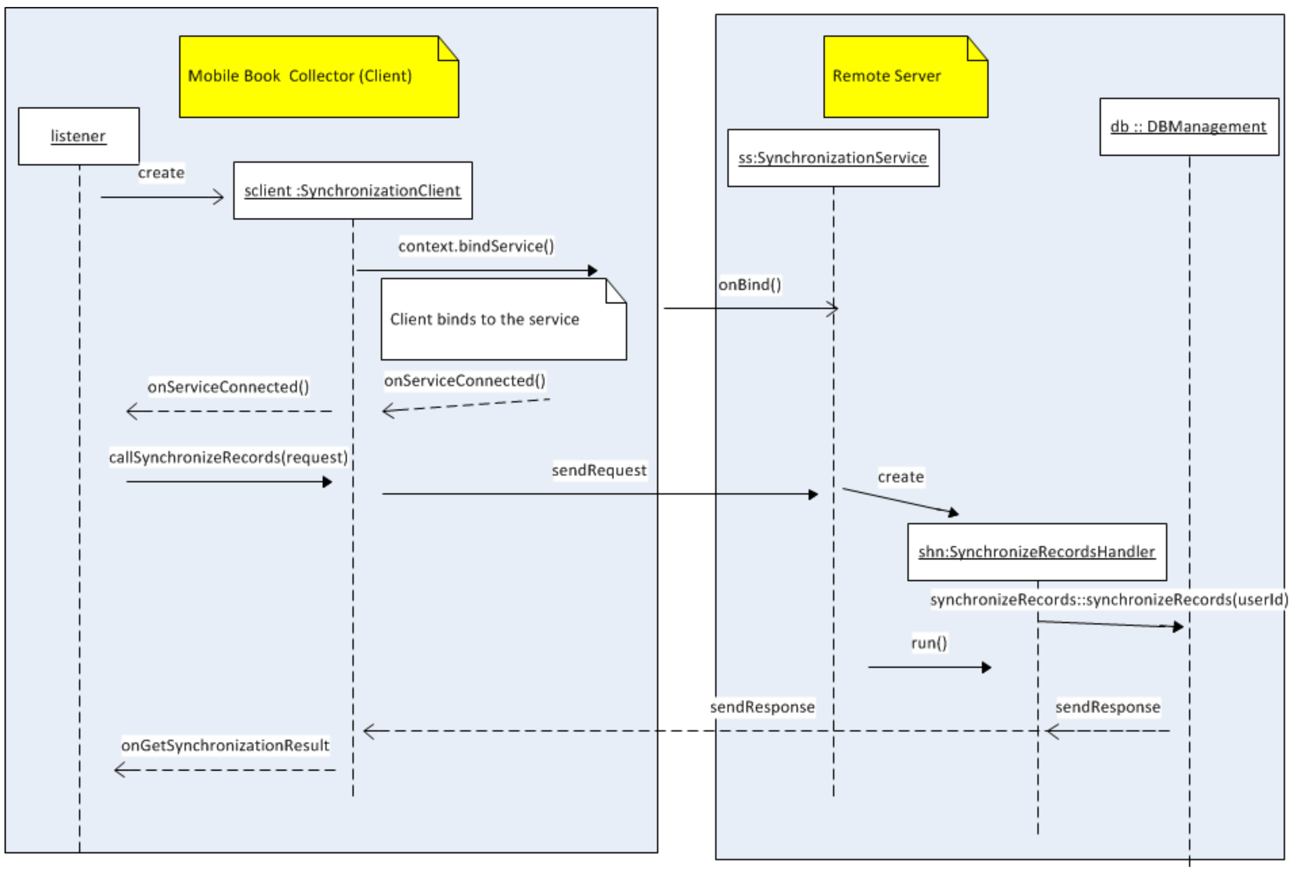

Mobile Book Collector (Client)
Remote Server
listener
create
sclient :SynchronizationClient
ss:SynchronizationService
db :: DBManagement
context.bindService()
Client binds to the service
onBind()
onServiceConnected()
onServiceConnected()
callSynchronizeRecords(request)
sendRequest
create
shn:SynchronizeRecordsHandler
synchronizeRecords::synchronizeRecords(userId)
run()
sendResponse
sendResponse
onGetSynchronizationResult

Figure 25 - Sequence Diagram between client and SynchronizationService

3.2.9 Local Database Management Component

This component manages local database operations. It is used to synchronize offline records.

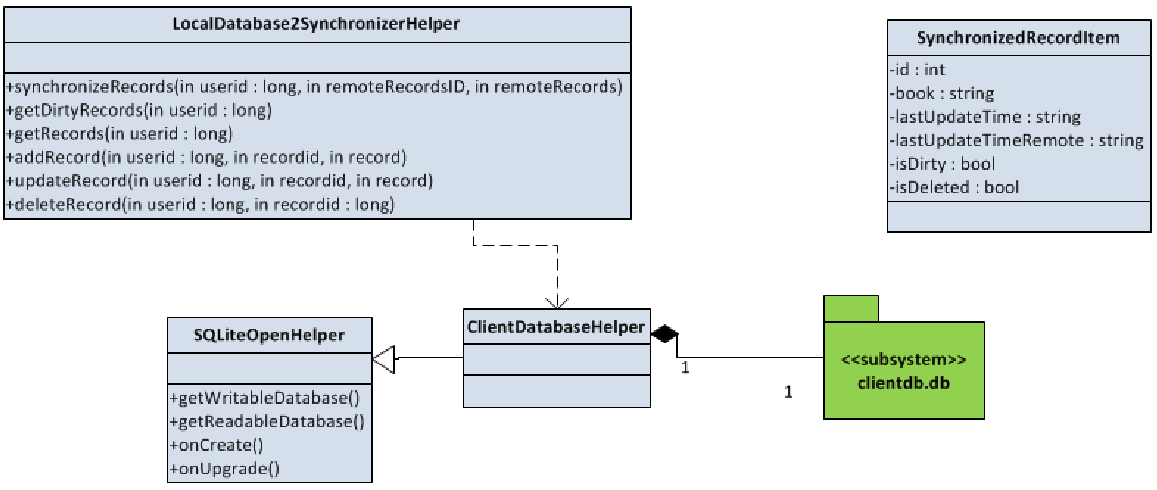

LocalDatabase2SynchronizerHelper
+synchronizeRecords(in userid : long, in remoteRecordsID, in remoteRecords)
+getDirtyRecords(in userid : long)
+getRecords(in userid : long)
+addRecord(in userid : long, in recordid, in record)
+updateRecord(in userid : long, in recordid, in record)
+deleteRecord(in userid : long, in recordid : long)
SynchronizedRecordItem
-id : int
-book : string
-lastUpdateTime : string
-lastUpdateTimeRemote : string
-isDirty : bool
-isDeleted : bool
SQLiteOpenHelper
+getWritableDatabase()
+getReadableDatabase()
+onCreate()
+onUpgrade()
ClientDatabaseHelper
1
1
<<subsystem>>
clientdb.db

Figure 26 – Class Diagram of Local Database Management Component

SynchronizeRecordItem

İd: unique record identifier of a record within the android application.

book: The record itself. The record is persisted as the JSON serialization of the object model.

lastUpdateTime: Last update time of the record within the android application.

lastUpdateTimeRemote: Last update time of the record from the remote server.

isDirty: The boolean indicator which indicates if the record persisted in the android application has changed and these changes are not synchronized with the remote server.

SQLLiteOpenHelper

It is a helper class for database open operations specific for Android SDK. It can be used to create tables also.

ClientDatabaseHelper

It is a helper class for database operations like create, delete, read and update. It uses SQLLiteOpenHelper for opening databases.

clientdb.db

It is the local database we use for offline records.

LocalDatabase2SynchronizerHelper

It is a layer between local database and android application.

3.3 Design Rationale

We wanted to build a system such that available from everywhere. Therefore, we designed a Client-Server system. Moreover, we don’t want to restrict the user to his/her device storage size. So, we designed system such that data is kept in a public database.

We chose Android for mobile platform because aproximately 50% of smartphone in market uses Android for operating system.

We wanted to use a RESTful web service because a REST service is:

- Platform-independent (you don't care if the server is Unix, the client is a Mac, or

anything else),

- Language-independent (C# can talk to Java, etc.),
- Standards-based (runs on top of HTTP), and
- Can easily be used in the presence of firewalls.

We chose Jersey library because Jersey makes it easy to build RESTful Web Services in Java.

4. Data Design

4.1 Data Description

4.1.1 Data Objects

Book:
id: Identity number given from the database for each book. This attribute is unique for any book.
title: Title of the book.
author: Author of the book.
description: A small description/summary of the book. The user will be able to edit this text.
isbn: A number consisting of 13-digit-number which is identical for the book in every country.
cover_image: Cover image of the book.
publisher: The company that has published the book.
publish_date: The date of publishing.
language: Demonstrates the language which the book is written in or translated to.
category: The general type of the book
type: The specified type of the book.
rating: Average score of the book rated by the users.
location: Indicates the place of the book in the bookshelf.

User:

id: Identity number given from the database for each user. This attribute is unique for any user.

name: The name and surname of the user.

username: User's nickname for the application.

password: The user's password.

e-mail: E-mail of the user that will be used for the communication information.

phone_number: Phone number of the user that will be used for the communication information.

description: A brief description where the user mentions about his/herself

profil_photo: A picture that may be used to face detection login.

Books-of-User :

user_id: Id number that will be taken from the user table indicating the owner of the book.

book_id: Id number taken from the book table indicating which book is owned by the user.

is_favourite: The user has selected the book as a favorite book.

status: The number demonstrating the status of the book. 0 means “owned” and 1 means “for sale”

is_lent: Is the book lent or not.

lent_date: When the lending action happens.

due_date: Due date for returning the book back.
id: Id of the user that takes the book.

Friends-of-user:
user_id, friend_id: The user with “friend_id” is a friend of the user with “user_id” .

Borrowed-book list:
book_id: Id of the book taken from “book” table.

owner_id: Id of the user that gives the book.

borrowed_date: When the borrowing action happens

due_date: Due date for returning the book back.

History:

id: Primary key.

type_id: Process type id.
date_time: When the process occurred.
description: Detailed explanation about process.
user_id: This is the user id of owner of history entry.

Process_types :
id: Primary key
type: Name of the process. It can be lending, borrowing, adding book, deleting book, selling, buying, adding friend, favoring, etc.

Message:
id: Primary key
sender_id: The user who sends the message.
receiver_id: The user who receives the message.
date: The date of the sent message
message: Information which is sent from a sender to a receiver.

Settings:
user_id: Primary key
isSearchabilityPubliclyAvailable: Whether user can be reached by public search or not.
isSearchabilityFriendlyAvailable: Whether user can be reached by a friend of that user or not.
isBookListsPubliclyAvailable: Whether user's book lists can be reached by public search or not.
isBookListsFriendlyAvailable: Whether user's book lists can be reached by a friend of that user or not**.**
isHistoryPubliclyAvailable: Whether user's history can be reached by public search or not.
isHistoryFriendlyAvailable: Whether user's history can be reached by a friend of that user or not.
isContactInformationPubliclyAvailable: Whether contact information of user can be reached by public search or not.

isContactInformationFriendlyAvailable: Whether contact information of user can be reached by a friend of that user or not.

4.1.2. Relationships and Complete Data Model

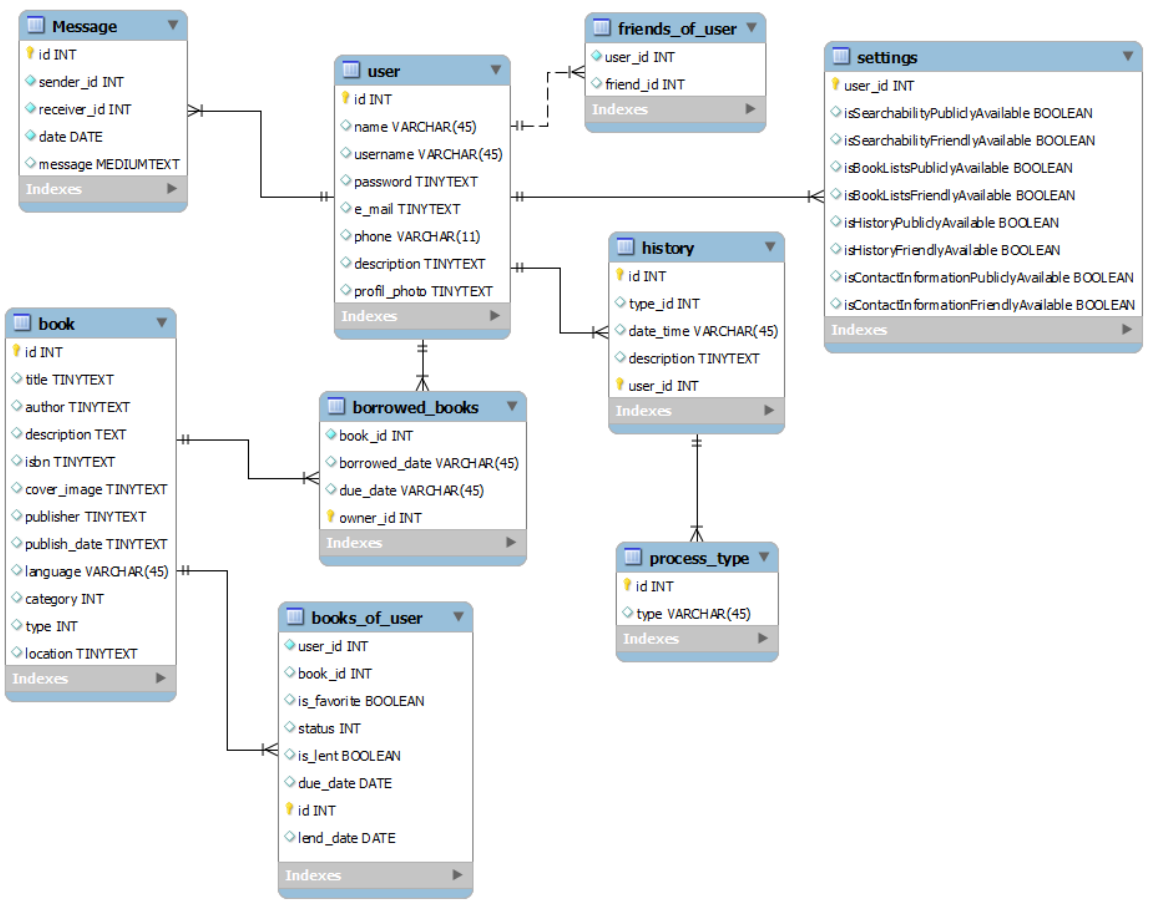


Figure 27 – ER Diagram

4.2 Data Dictionary

Alphabetically list the system entities or major data along with their types

1. Book:

Parameter:	**Parameter Type:**
author:	User
category:	String
cover_image:	String (path information)

description:	String
id:	int
isbn:	String (13-digit-number)
language:	String
location:	String
publisher:	String
publish_date:	String
rating:	Float
title:	String
type:	String

2. Books-of-User

Parameter:	**Parameter Type:**
book_id:	int
due_date:	String.
id:	int
is_favourite:	Boolean
is_lent:	Boolean
lent_date:	String
user_id:	int
status:	int (0,1 uses generally)

3. Borrowed-book list:

Parameter:	**Parameter Type:**
book_id:	int
borrowed_date:	String
due_date:	String
owner_id:	int

4. Friends-of-user:

Parameter:	**Parameter Type:**
friend_id:	int

user_id	int

5. History

Parameter:	**Parameter Type:**
date_time:	String
description:	String
id:	int
type_id:	int
user_id:	int

6. Message:

Parameter:	**Parameter Type:**
date:	String
id:	int
message:	String
receiver_id:	int
sender_id:	int

7. Process_types :

Parameter:	**Parameter Type:**
id:	int
type:	String

8. Settings:

Parameter:	**Parameter Type:**
isBookListsFriendlyAvailable:	Boolean
isBookListsPubliclyAvailable:	Boolean
isContactInformationFriendlyAvailable:	Boolean
isContactInformationPubliclyAvailable:	Boolean
isHistoryFriendlyAvailable:	Boolean
isHistoryPubliclyAvailable:	Boolean
isSearchabilityFriendlyAvailable:	Boolean
isSearchabilityPubliclyAvailable:	Boolean
user_id:	Boolean

9. User:

Parameter:	**Parameter Type:**
description:	String
e-mail:	String
id:	int
name:	String
password:	String
phone_number:	String
profil_photo:	String
username:	String

5. Human Interface Design

5.1 Overview of User Interface

The user will be prompted with the login screen when the application starts. Once the user logs into the application successfully, the profile page of the user will be prompted to him/her according to the user type. For the users who have not registered yet, there will be a button linking to the register screen. By using this application, users can keep their book lists and manage the information of their owned books.

5.2 Screen Objects and Actions

This section briefly describes the interfaces and interface components of MOBCOLL.

5.2.1 Register Screen

Register screen will be used in case of the user has not been saved to the database yet. The system will request the basic information of the user.

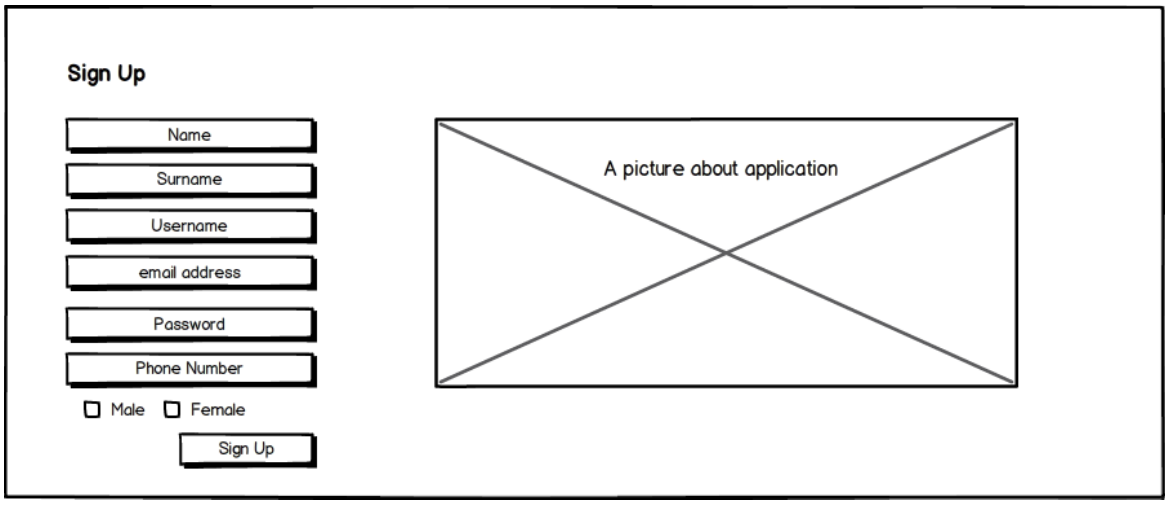


Figure 28 – Register Screen of Application

5.2.2 Login Screen

Login screen will be the first screen which the users see when MOBCOLL Application starts.

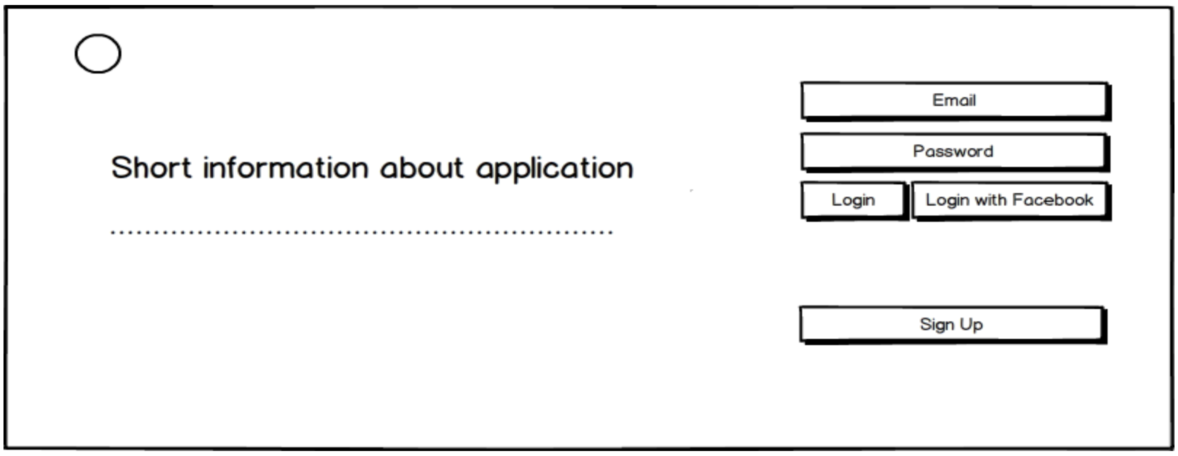


Figure 29 – Login Screen of Application

5.2.3 Logout Screen

When the logout button is pressed, the login screen will come up and the system waits for a logging in or registering.

5.2.4 User Profile Screen

When the user logs in, this screen will show up. In this screen, there will be text fields containing information about the user. If it is added, the profile picture of the user will also be in this screen. The user can also view his/her book list and friend list. These items will be placed in the screen as shown in the following figure.

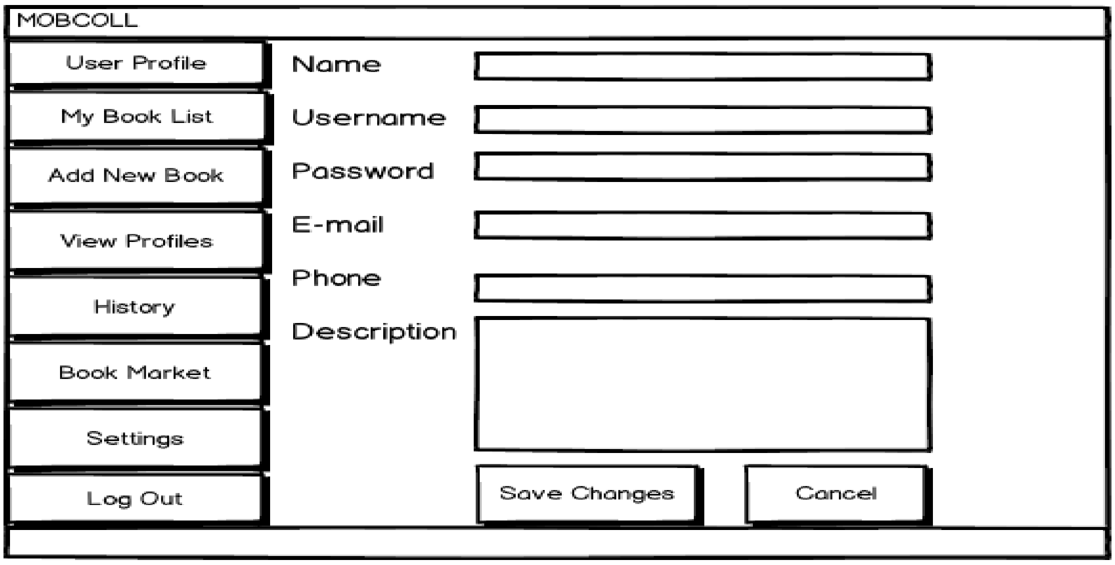


Figure 30 – User Profile Screen of Application

5.2.5 My Book List Screen

This screen consists of the list of books owned by the logged in user. A search tool to find a book in the list will be above the list. There will be buttons to recommend or selecting as for sale or gift a book after choosing from the list. Also, when a book is selected from the list, its information will be shown in the screen. These items will be placed in the screen as shown in the following figure.

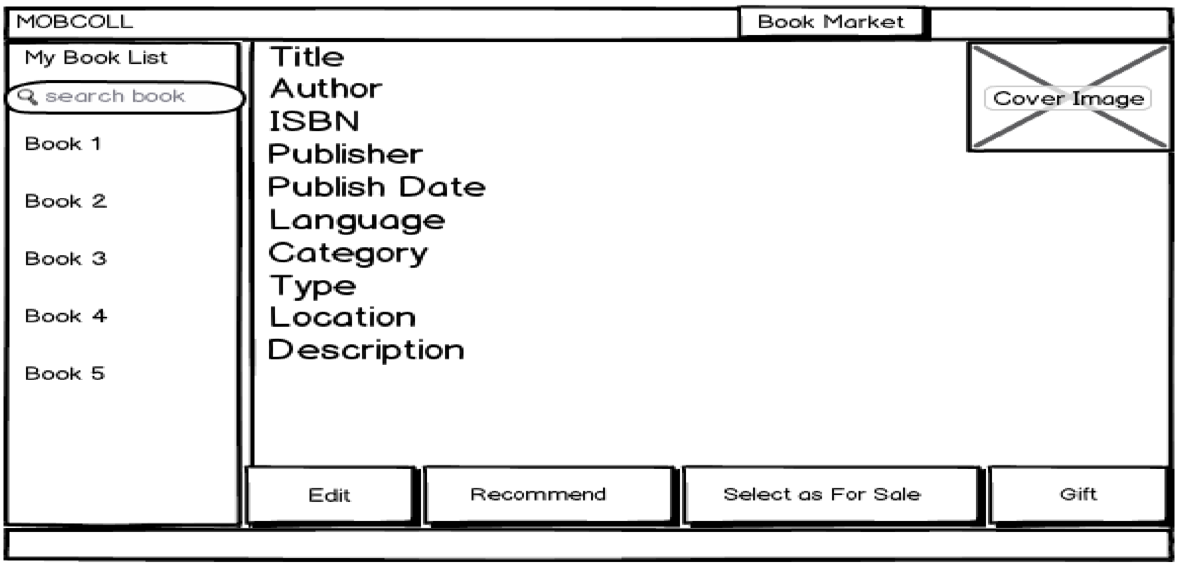


Figure 31 – My BookList Screen of Application

5.2.6 Add New Book

This screen will be shown up when the “add new book” button is pressed. The user can add a book to his/her book list by gathering information of the book using barcode reader quality of the application. While adding a new book, there will be a button to add the cover image of the book. These items will be placed in the screen as shown in the following figure.

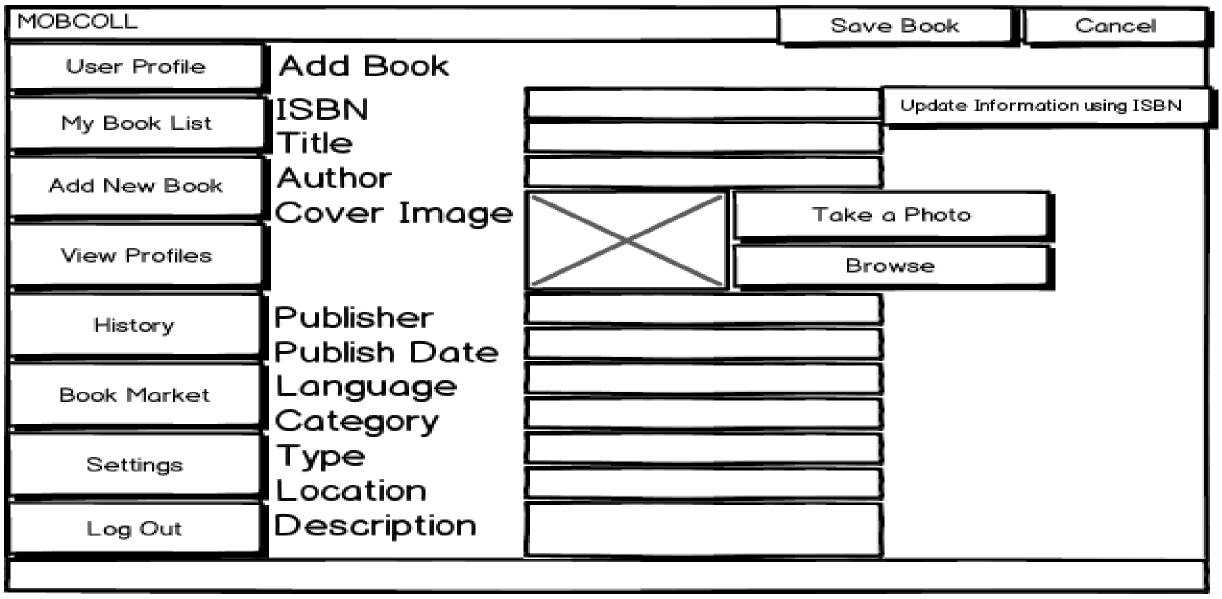


Figure 32 – Add New Book Screen of Application

5.2.7 View Profiles

A user can search reach the list of users who allow to be found in the search. The screen consists of list of users, information of the selected user and three buttons which are “add friend”, “view user history” and “send message”. “Add friend” button sends request to the selected user to be friends. “Send Message” button sends a message to the selected user. The other button will show up the transaction history of the user if he/she allows this feature. A search tool to find a user will be placed above of the user list. These items will be placed in the screen as shown in the following figure.

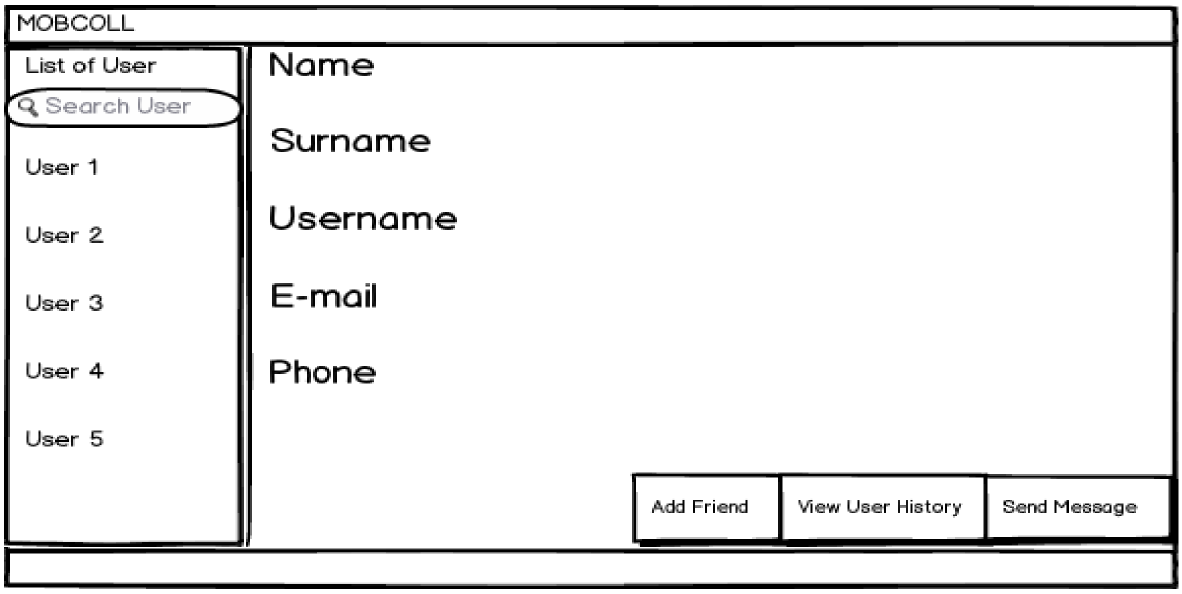


Figure 33 – View Profiles Screen of Application

5.2.8 History

All processes that a user make are saved for each user in transaction history. This list can be viewed by an audience determined by the user. The audience can be friend, everyone or no one. The history keeps track of the date, type and the users related to the process. The type of the process can be followings:

- Buying a book
- Selling a book
- Selecting a book as for sale
- Creating book list for sale
- Creating gift request
- Sharing information
- Borrowing a book
- Lending a book
- Adding a friend
- Deleting a friend
- Updating user info

These items will be placed in the screen as shown in the following figure.

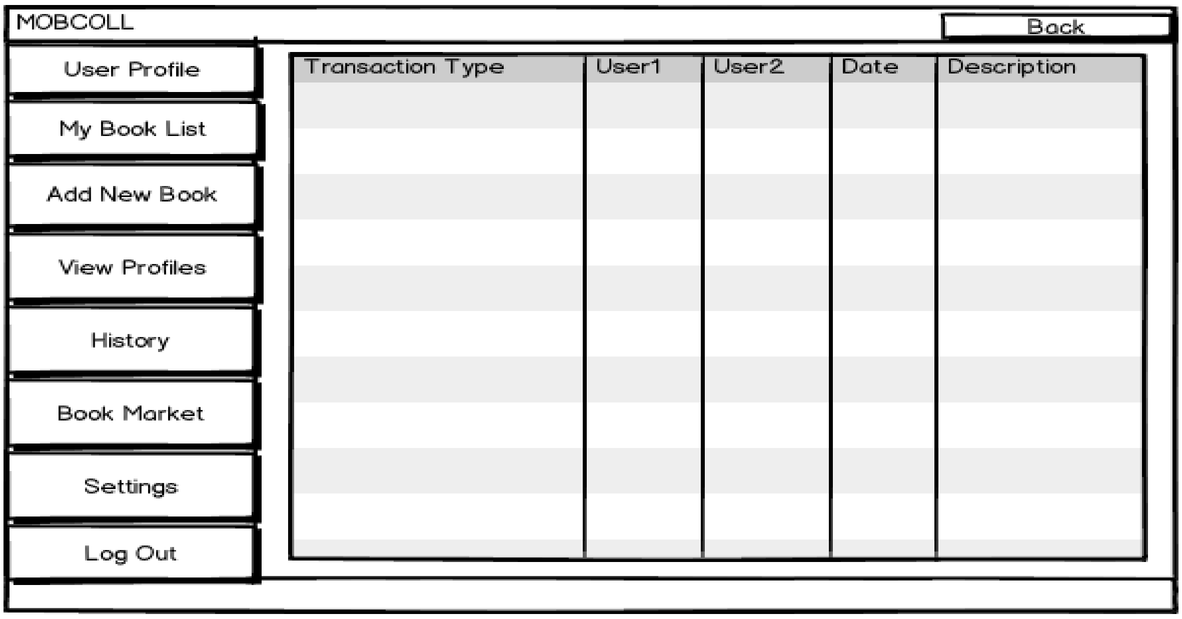


Figure 34 – History Screen of Application

5.2.9 Settings

The user will modify the privacy and account settings in this screen. In privacy settings part, user decides who will be the audience of history, contact information and search ability. The choices will be friends, everyone or no one. In account settings part, the user can update the account information like e-mail or password or username and by clicking the "save changes" button, the system saves the new settings. These items will be placed in the screen as shown in the following figure.

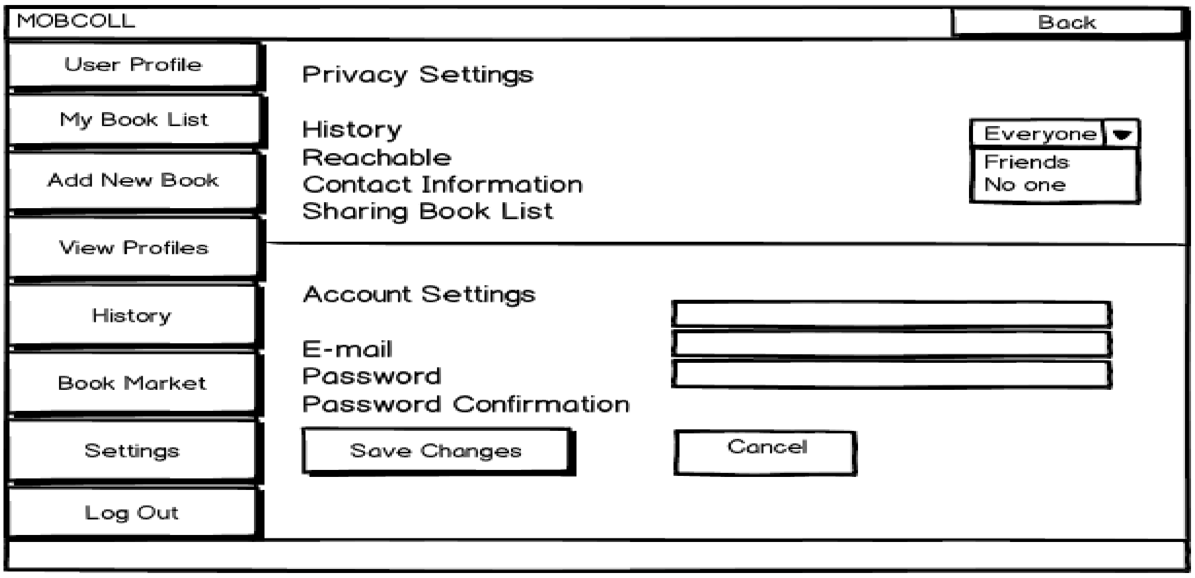


Figure 35 – Settings Screen of Application

5.2.10 Book Market

The user handles book transactions in this market tool. Using this tool, a user can:

- buy a book
- sell a book
- lend a book
- borrow a book
- create book list for sale
- view book list for sale
- view list of lent books
- view list of borrowed books
- send gift request
- recommend a book

Instead of the menu on the left, there will be market-related buttons. These items will be placed in the screen as shown in the following figure.

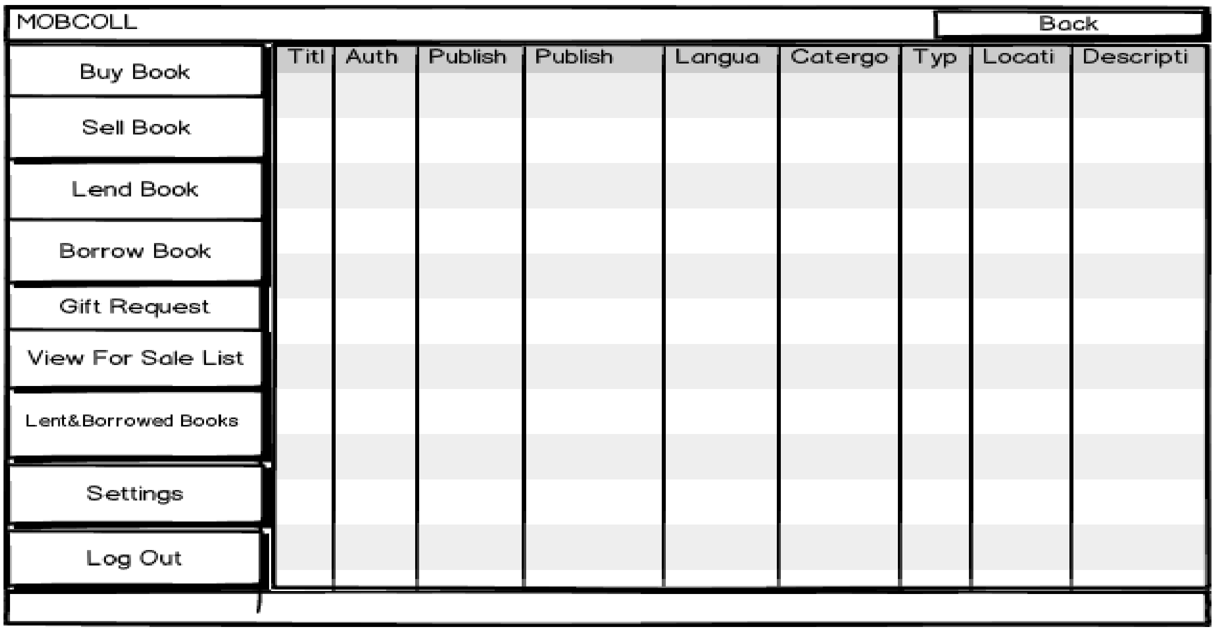


Figure 36 – Book Market Screen of Application

The user can view and create a for sale list with “view for sale list” button and view lent and borrowed books list in a joined table with “lent&borrowed books” button.

5.2.11 Search Book

User can search a book in this search tool by directly writing the title of the book to this tool. Search results will be shown in the screen. After selecting the founded books, using some buttons like “add book to my list”, basic operation about a book will be handled. These items will be placed in the screen as shown in the following figure.

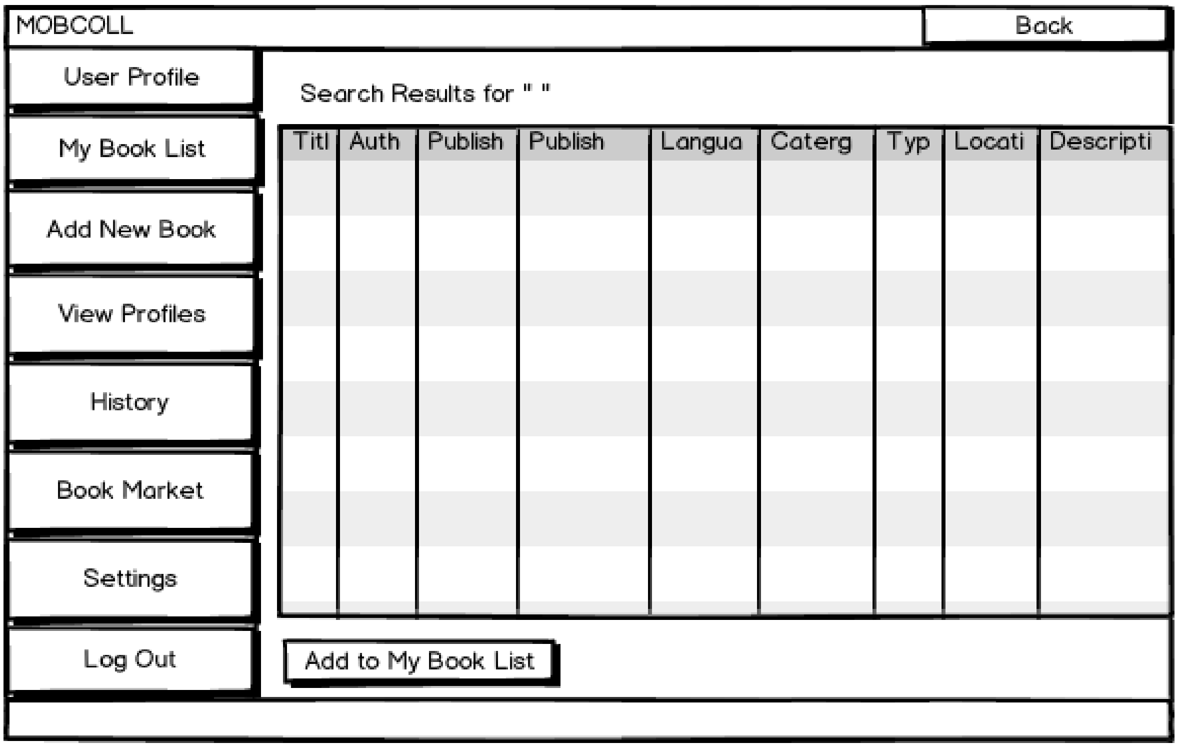


Figure 37 – Search Book Screen of Application

6. Requirements Matrix

UC stands for use case.

Components: Requirements from SRS (use cases):	Login Logout Component	Main Functionality Component	Settings Component	Social Component	Trade Component	Data Service Component	Remote Server Component
UC1	**X**					**X**	**X**
UC2	**X**					**X**	**X**

UC3	**X**					**X**	**X**
UC4	**X**					**X**	**X**
UC5		**X**				**X**	**X**
UC6		**X**				**X**	**X**
UC7		**X**				**X**	**X**
UC8		**X**				**X**	**X**
UC9		**X**				**X**	**X**
UC10		**X**				**X**	**X**
UC11				**X**		**X**	**X**
UC12				**X**		**X**	**X**
UC13			**X**			**X**	**X**
UC14				**X**		**X**	**X**
UC15				**X**		**X**	**X**
UC16					**X**	**X**	**X**
UC17					**X**	**X**	**X**
UC18				**X**		**X**	**X**
UC19		**X**				**X**	**X**
UC20				**X**		**X**	**X**
UC21		**X**				**X**	**X**
UC22					**X**	**X**	**X**
UC23					**X**	**X**	**X**
UC24		**X**				**X**	**X**