Software Design Specification

for

College Living

Version 2.0

Prepared by Andrew Blake Dunn,
Derek Overlock, Lumin Shi

The University of Alabama

March 16, 2013

Revision History

Name	Date	Reason For Changes	Version
Derek Overlock, Andrew Blake Dunn, Lumin Shi	2/2/2013	Initial draft	0.1
Derek Overlock	2/11/2013	Functional requirements	0.2
Andrew Blake Dunn	2/11/2013	Use Case Diagrams, Activity Diagrams	0.3
Lumin Shi	2/11/2013	Nonfunctional Requirements	0.4
Derek Overlock, Lumin Shi	2/13/2013	Class Diagrams	0.5
Andrew Blake Dunn	2/13/2013	Activity Diagrams	0.6
Derek Overlock, Lumin Shi, Andrew Blake Dun	2/13/2013	Finished initial draft	1.0
Derek Overlock	2/17/2013	Updated overall description, functional requirements and non-funcitonal requirements	1.1
Derek Overlock, Andrew Blake Dunn, Lumin Shi	3/6/2013	Extended SRS for Design Document	1.2
Derek Overlock, Andrew Blake Dunn, Lumin Shi	3/16/2013	Completed SDS Document	2.0

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to outline the requirement specifications for the College Living application. This document will specify the product's scope, feature requirements (functional and non-functional), user interfaces, and UML documentation of certain use cases, activity diagrams, and class diagrams.

1.2 Product Goals

College Living is a social-networking tool that assists college students with finding available apartments and compatible roommates. The application is also meant to help students find other students with common interests, providing a safe environment for them to interact. Apartment complexes will register available units with a web service. Students will also be able to view these listings.

1.3 Product Scope

College Living utilizes the device's global positioning system (GPS) to localize this application. Other users that are close to the user will be listed on the **Roomies** screen. Local apartment units will be shown to the user on the **Pads**. A database will store user information (login, profile, location, etc.), user messages, and apartment listings. Users will initially answer a set of questions. These answers will be used to rate their compatibility with other users on the **Roomies** screen.

1.4 Definitions

Android Operating System
: An open-source operating system used for mobile phones

Compatibility Score
: A rating given to each user to demonstrate the level of compatibility between the rated user and the student

Global Positioning System (GPS)
: A global positioning system is a system that receives signals from satellites for navigation and tracking purposes

JavaScript Object Notation (JSON)
: Lightweight, human-readable, text-based standard for sending data between systems.

MySQL
: An open-source database management system

Pads
: Activity screen displaying potential apartment units for the user

PHP: Hypertext Processor (PHP)

An open-source server-side programming language

Roomies

Activity screen displaying potential roommates for the user

2 Overall Description

2.1 Product Perspective

On-campus student living has become difficult to sustain for both students and university administrators. The cost of on-campus student living is increasing every year as demand increase. As an example to demonstrate this problem, in 2011 72% percent of undergraduate students at The University of Alabama lived off-campus. College Living will help this increasing problem by providing viable living options for college students.

2.2 Product Features

This product consists of two domains: a web-service for the apartment complexes, and a mobile application for college students. The web-service will allow the apartment complexes to upload information about their available units, including monthly rent, pictures and amenities. The mobile application will be an interface for college students to view these apartment listings. Furthermore, the mobile application will also be an interface for college students to interact with other local students that share common interests.

2.3 Operating Environment

There are two distinct domains for this product: a web-service and a mobile application. The web-service will provide an interface for apartment complexes to advertise their available units to college students. The mobile application will be used by college students to search for available apartments along with potential roommates.

2.4 Geo-spatial Searching

This feature will require the use of GPS to locate the user. Based off of the user's location, both other users and available apartments in close proximity will be listed in their respective screen.

2.5 Apartment Web Service

This feature will allow apartment management to publish information about their available property. Information includes: monthly rent, amenities, lease length, and photos of the unit.

2.6 User Listing

Using geo-spatial searching, the user list will provide a grid view of User Tiles in proximity to the user. The tile will display the user's primary picture, display name, and compatibility score. By clicking on the User Tile, a user can look at that particular user's information. Information includes: compatibility score, age, basic description, etc.

2.7 Apartment Listing

Using geo-spatial searching, the user list will provide a grid view of Apartment Tiles in proximity to the user. By clicking on the Apartment Tile, a user can look at that particular apartment unit's information. Information includes: monthly rent, amenities, lease length, and photos of the unit.

2.8 Constraints

The network bandwidth experienced by the user's device will determine the loading time of certain areas of this software, particularly the user listing and apartment listing screens.

2.9 Dependencies

The mobile application will require a data connection (cellular or Wi-Fi), along with location services. The web-service will require a standards-compliant browser that supports file uploading, form posting and JavaScript.

3 Functional Requirements

3.1 Geo-spatial searching

Name	Description
Location Services	The system will utilize the device's GPS to obtain the location of the user.
Location Save	The system will push the user's location to the database.
Location Update	The system will periodically request the user's location from the device's GPS and update this information in the database.
Data pull from database	The system will query the database to find other user's within the user's specified radius.

3.2 Apartment Web Service

Name	Description
Data pull from database	The system will query the database to find and display the registered apartment units.

3.3 User Listing

Name	Description
Data pull from database	Using the user's location, the system will pull local student information from the database.
Grid View	The system will show the user a grid of tiles that represent local students. The tile will display the users primary picture, along with their display name and compatibility score.
Timed Refresh	The grid view will periodically refresh to update the grid of users in proximity.

3.4 User Interaction

Name	Description
Display User Profile	When the user selects a student tile, the system will display the user profile of that selected user.
User Messaging	When the user sends a message to another user, the system will save the message to the database.
Message pull from database	If a user is online and has unread messages saved in database, the system will pull these messages from the database and clear them out of the database.

3.5 Apartment Listing

Name	Description
Data pull from database	Using the user's location, the system will pull local apartment unit information from the database.
Grid View	The system will show the user a grid of tiles that represent local students.
Timed Refresh	The grid view will periodically refresh to update the grid of users in proximity.

4 Nonfunctional Requirements

4.1 Geo-spatial searching

Name	Description
GPS Services	System uses GPS services, and it has to be enabled for location matching.

4.2 Apartment Web Service

Name	Description
Modern Browser	Web service requires Javascript enabled.
Server Implementation	MySQL and PHP is required to build the web service.
Internet Access	Web service requires basic access to the Internet.

4.3 User listing

Name	Description
GPS Services	GPS services is required to locate user's current location.
Internet Access	Cellular network or WI-FI is required to get data from the server, and they can also be used to approximate user's location.
JSON API	JSON API is required to get/update information based on user's request.
Compatibility Score	During sign up, users are required to answer a set of questions so they can be scored for compatibility with other users.

4.4 Apartment listing

Name	Description
GPS Services	GPS services is required to locate user's current location.

Internet Access	Cellular network or WI-FI is required to get data from the server, and they can also be used to approximate user's location.
JSON API	JSON API is required to get/update information based on user's request.

5 UML Diagrams

5.1 Use Cases

5.1.1 Student Use Case

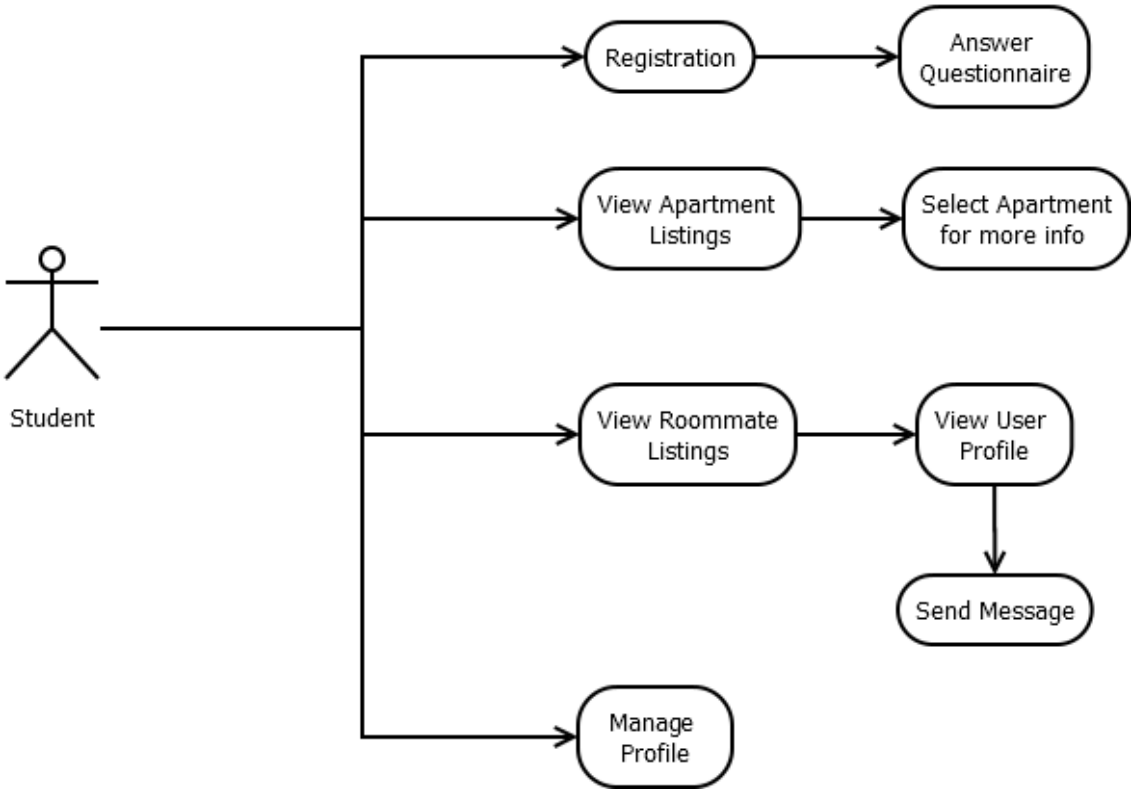


Fig. 5.1 Use Case diagram for student users

This diagram shows the actions that the students can perform. The student will first have to register through the registration page. Once registered, he/she will answer a set of questions about his/her lifestyle. The student will also be able view apartment listings, view roommate listings, or manage his/her own profile. If the student views the grid of apartment listings, the user can select an apartment tile to get more information. If the student views the roommate listings, the user can select a user tile and interact with the user by sending him/her a message. Finally, the user can manage his/her own profile information by selecting 'Manage Profile.'

5.1.2 Apartment Contact Use Case

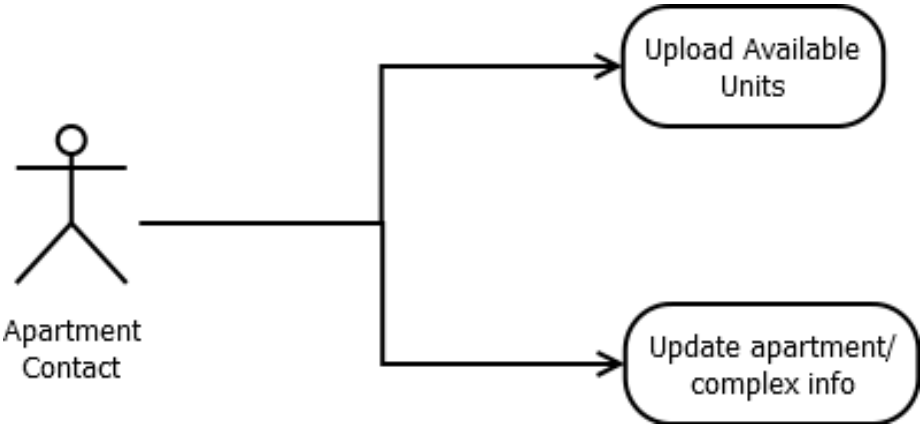


Fig 5.2 Use case for apartment contact

This diagram shows the actions the apartment contacts can use via the web service. An apartment contact will be able to upload information pertaining to available units. They will also be able to delete and update this information.

5.2 Activity Diagram

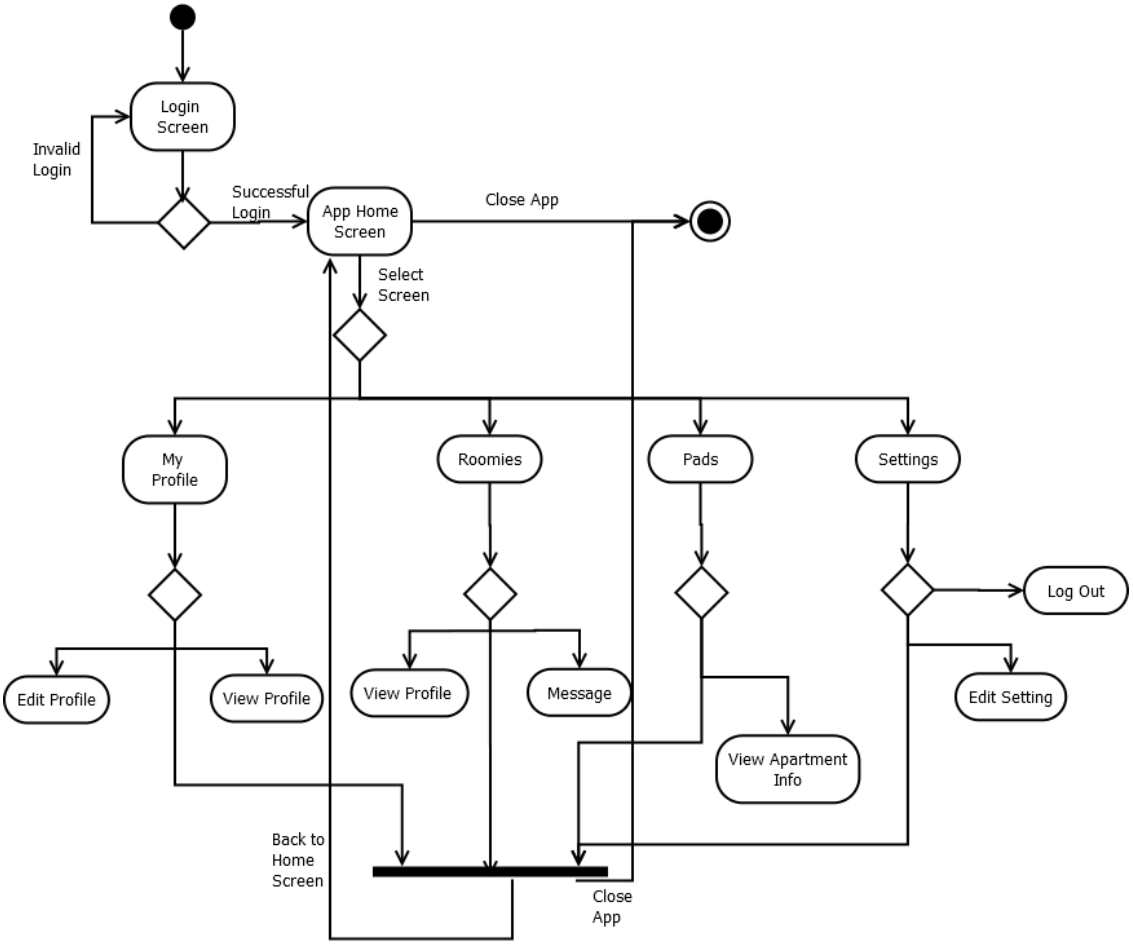


Fig. 5.2.1 High-level activity diagram visualizing the domain of this system

This diagram is a high-level depiction of the domain of this system. The student will login and choose which activity he/she would like to execute: *My Profile, Roomies, Pads, Settings,* or *Log Out*. This diagram also depicts the high-level actions inside each acitivity that a user can execute.

5.2.1 Viewing Available Apartments

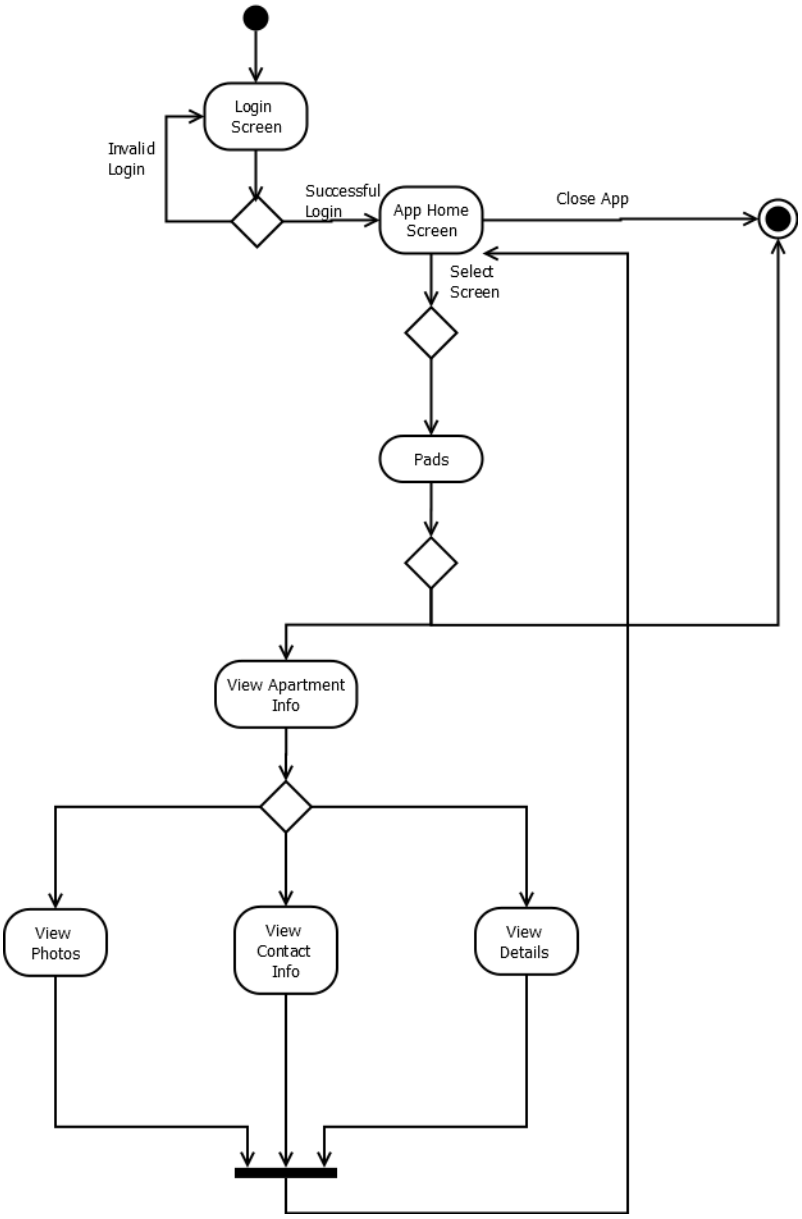


Fig. 5.2.2 Activity diagram of viewing available apartments

This activity diagram shows the process for a student to view available apartments. The user will log in and select the 'Pads' button. In this screen, a grid of tiles will show, each tile representing an available apartment unit. A user can select an individual tile to view more information about that unit. From there, the user can view photos and details about the unit, and also view the apartment's contact information.

5.2.2 Viewing/Interacting Potential Roommates

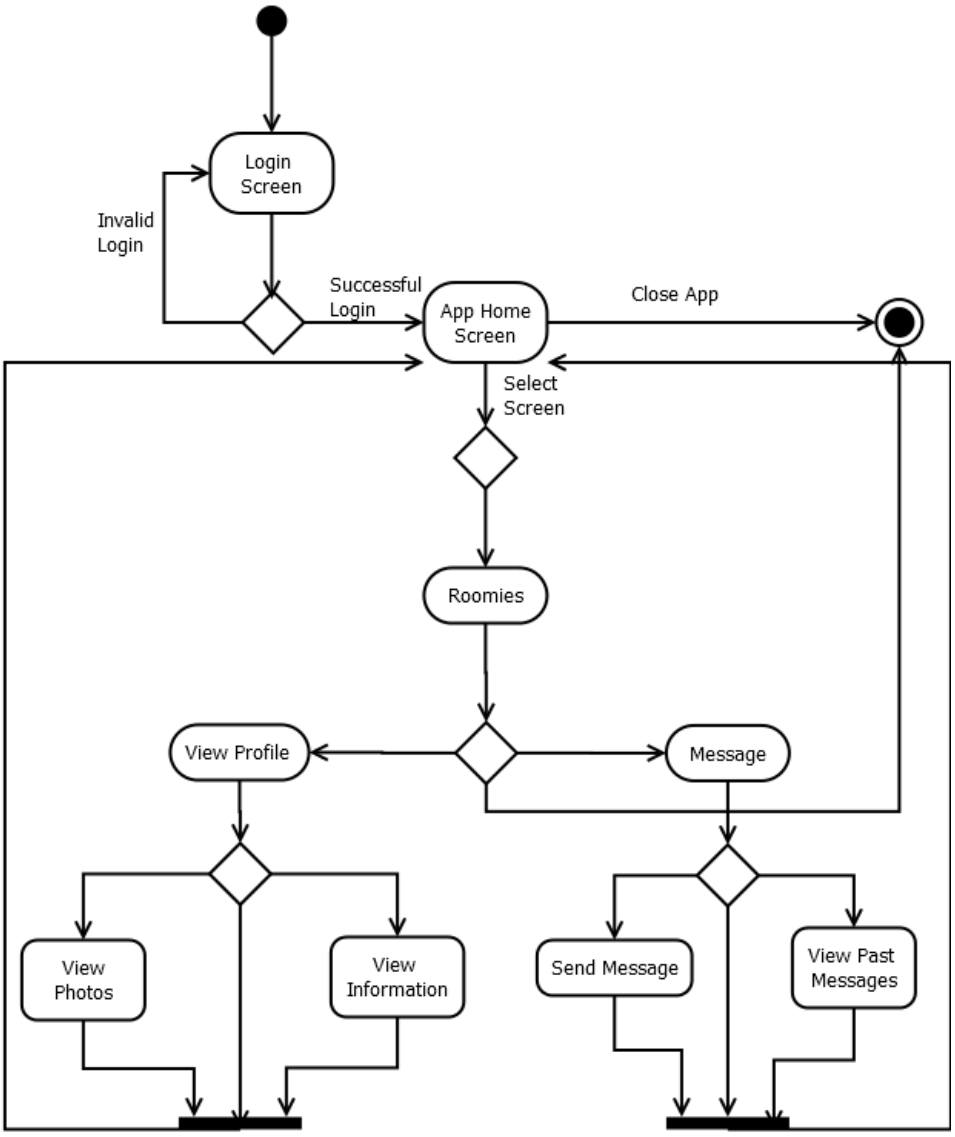


Fig. 5.2.3 Activity diagram for viewing (and interacting with) potential roommates

This diagram shows the process for a student to view and communicate with potential roommates. The student will log in and select the 'Roomies' button. In this screen, a grid of tiles representing local potential roommates will show. The student can select a tile to view the user's profile and to message the user.

5.2.3 Viewing/Modifying User Profile

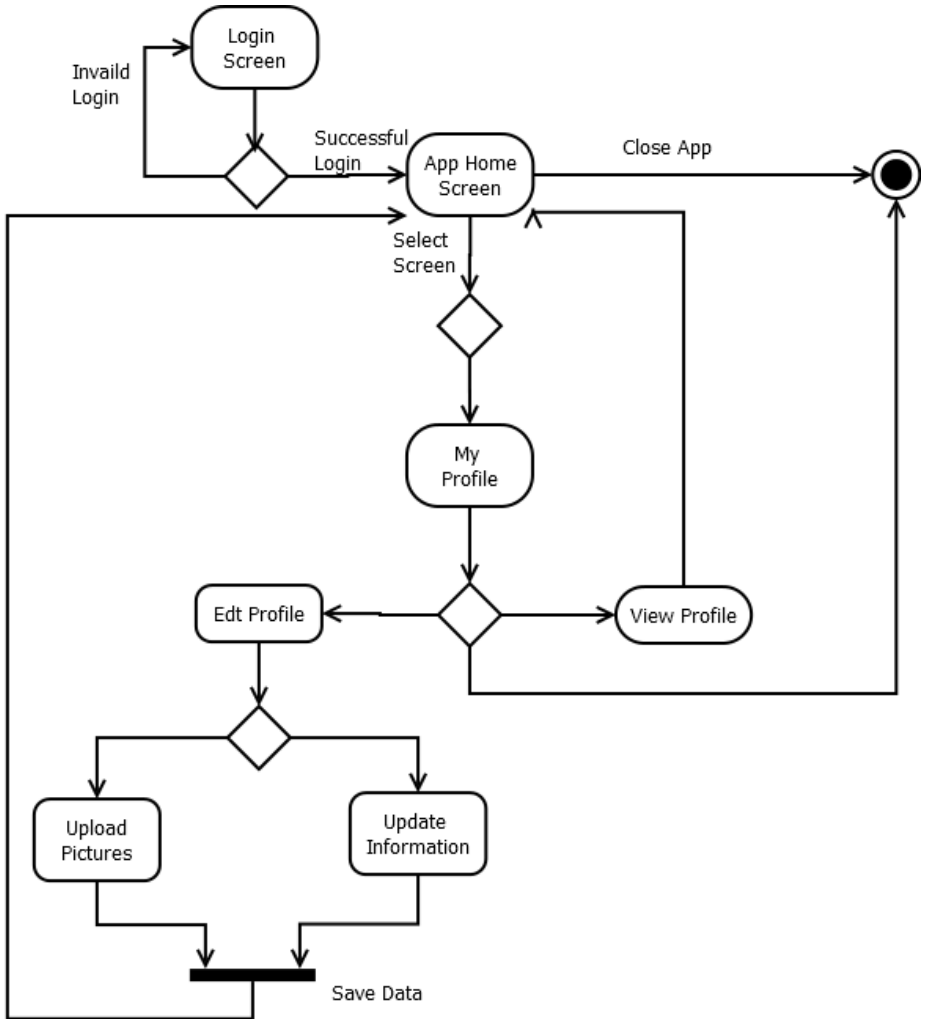


Fig. 5.2.4 Activity diagram for viewing and modifying user profile

This diagram shows the process for a student to view and edit his/her own profile. The student will log in and select the 'My Profile' button on the home screen. At this screen, the student can either view their profile or edit their profile. If they choose to edit their profile, they may upload pictures and/or update information. Finally, the user will then save the changes.

5.2.4 Settings Management

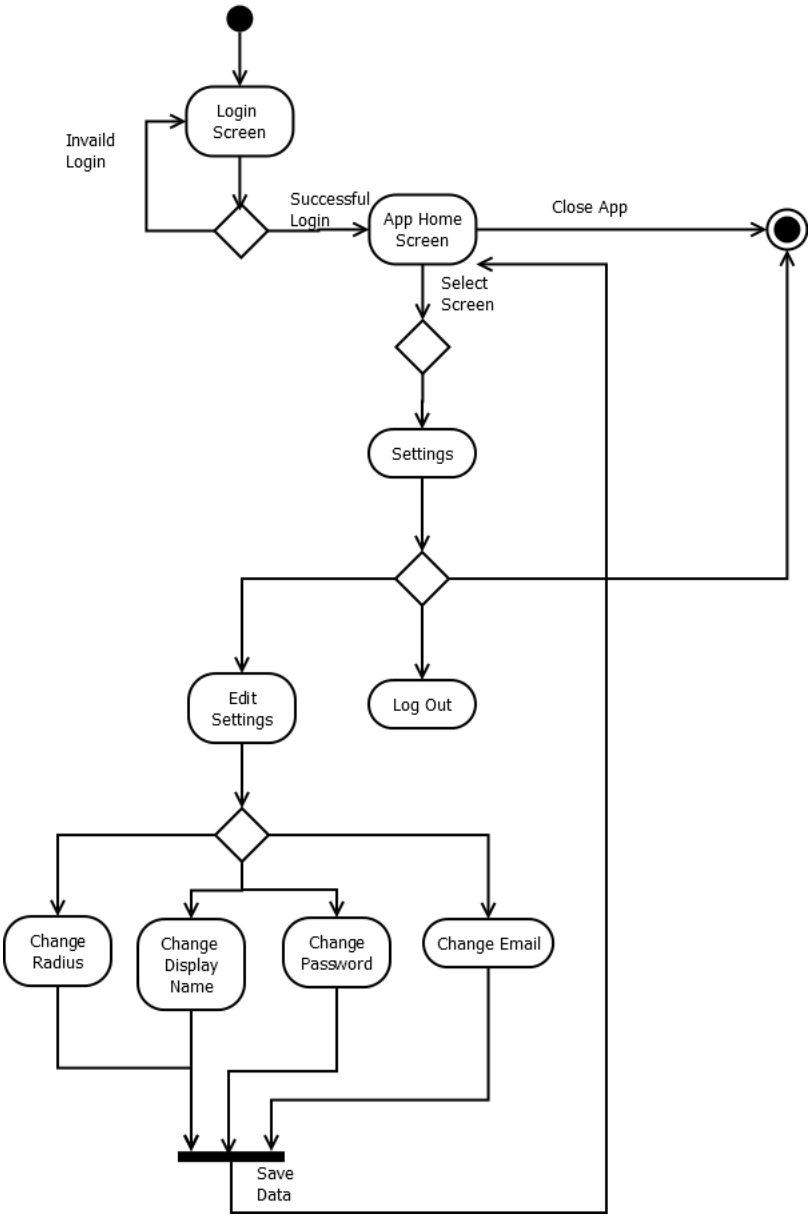


Fig. 5.2.5 Activity diagram of settings management

This diagram shows the process for a student to manage the application's settings. The student will log in and select 'Settings' on the home screen. At this screen, the student can edit the search radius, along with his/her display name, password and e-mail address.

5.3 Class Diagram

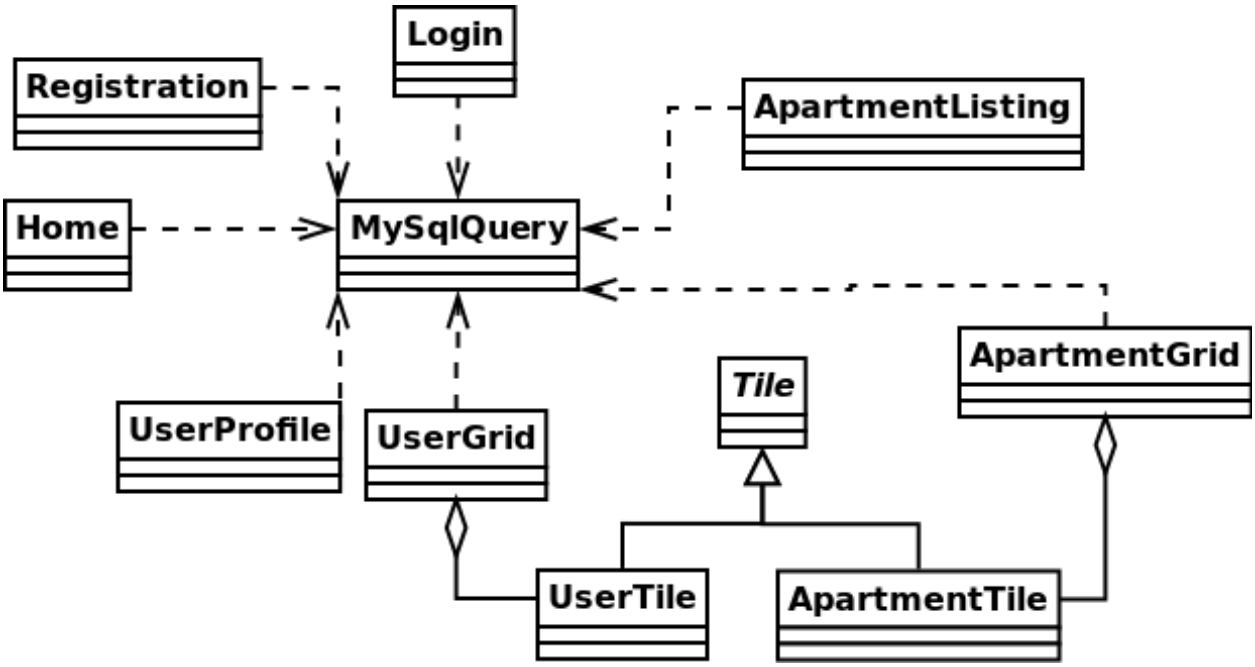
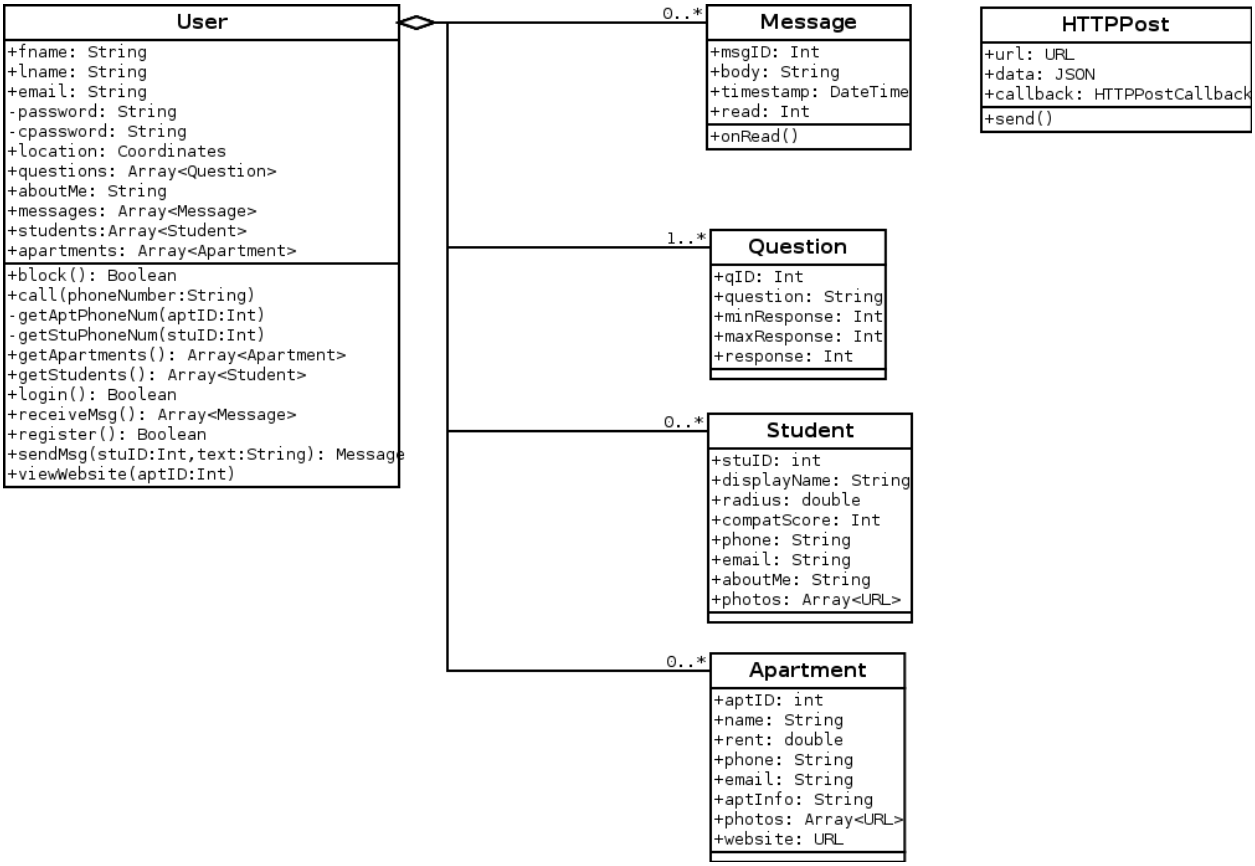


Fig. 5.3 High-level class diagram of system

This diagram shows the relationships between the classes in the system. There will be ten different logic classes for this android application: *ApartmentGrid, ApartmentListing, Home, Login, MySqlQuery, Tile (UserTile and ApartmentTile), UserGrid,* and *UserProfile*. These logic classes will use the *MySqlQuery* class to query the MySQL database for information.

6 Design Specification

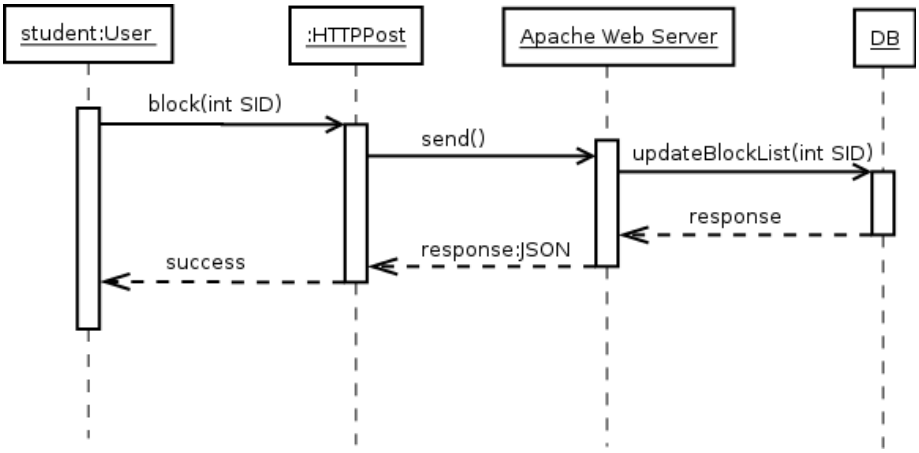
6.1 Detailed Class Diagram



This detailed class diagram shows the relationship between objects in the system. The *User* class is a class that represents the logged in user. This object will communicate with other local users through the *sendMsg()* and *receiveMsg()* methods. Other users are represented by the *Student* class, and are held in the *User*'s *students* property. Local apartments are represented by the *Apartment* class, and are held in the *User*'s *apartments* property. The *Student* and *Apartment* class simply hold information for each student and apartment object. The *Question* class represents each question used for the questionnaire during registration and is held by the *question* property in *User.* All communication between the *User* and server will be done through *HTTPPost*. The *Apartment*, *Message*, *Question* and *Student* classes will all contain *get* and *set* methods.

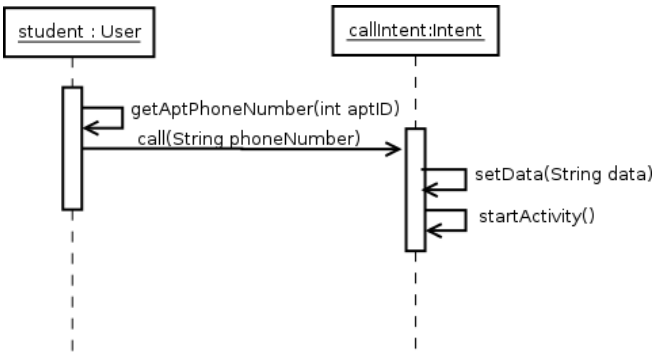
6.2 Sequence Diagrams

6.2.1 Blocking a User



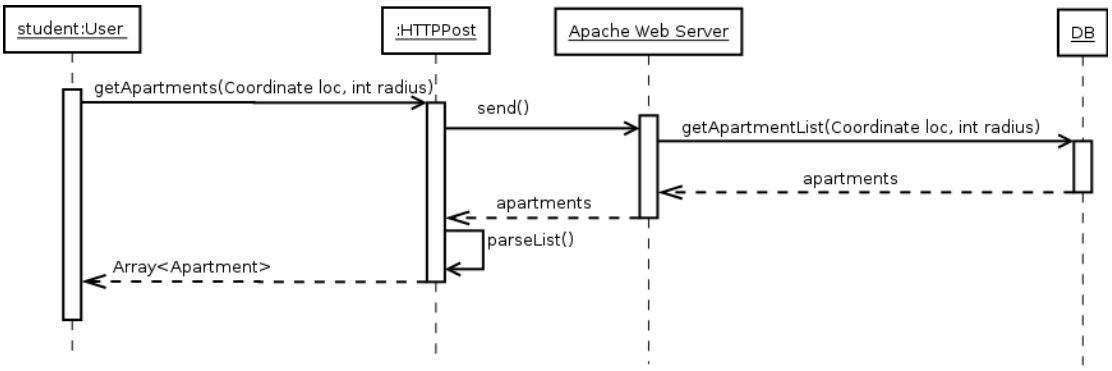
A user can block other students from communicating with them. Once the User class has been initiated with the students credentials and logged in, the *block()* method can be called with the other student's ID number. This method will send a request to the *HTTPPost* class, which will forward the request to the Apache web server. The server will call a method to update the blocked user list in the database. The database will return a Boolean to the server, indicating whether this operation was successful. The server will encode this Boolean as a JSON object and return it to the application.

6.2.2 Calling an Apartment



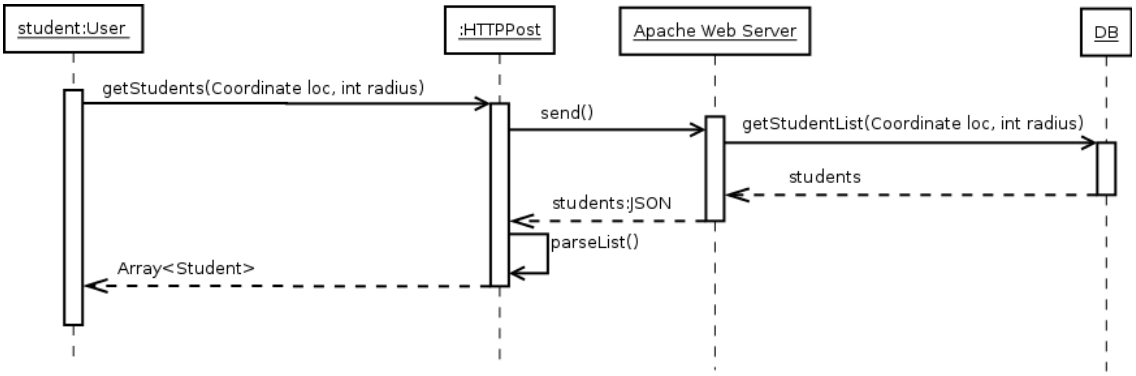
A user can call an apartment complex to get more information about their units. When the user wants to call an apartment, the *User* class will use the *getAptPhoneNumber()* method with the apartment's ID number. This method will return a phone number for the apartment. This phone number will get passed into the *call()* method. This method will set up a new *Intent*, with the phone number passed into the *setData()* method. Once the activity is started, the application will call the supplied phone number.

6.2.3 Getting a List of Local Apartments



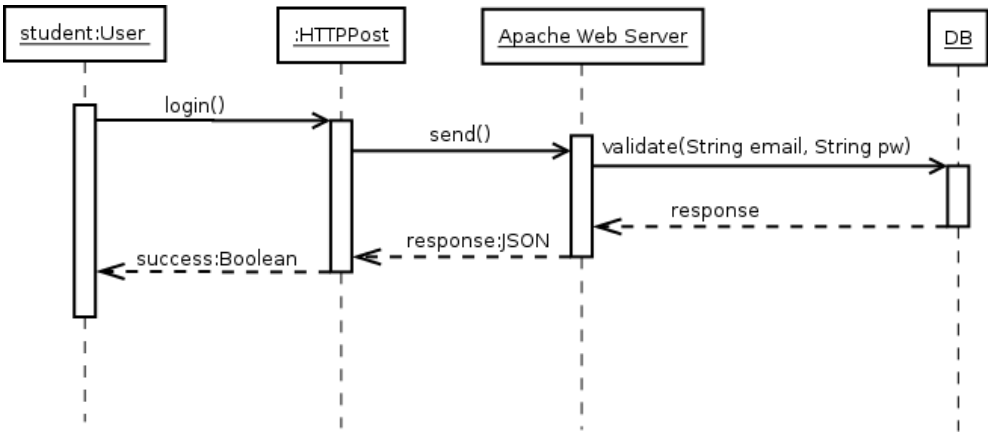
The application needs a list of local apartments for the user to interact with. The *User* class will call the *getApartments()* method with the user's current location and a search radius. This method will send a request to *HTTPPost*, which will forward the request to the Apache server. The server will query the database for a list of apartments that are close to the user. This list will get returned to the application, first as a JSON response. It will then be parsed into an array of *Apartment* instances.

6.2.4 Getting a List of Local Students



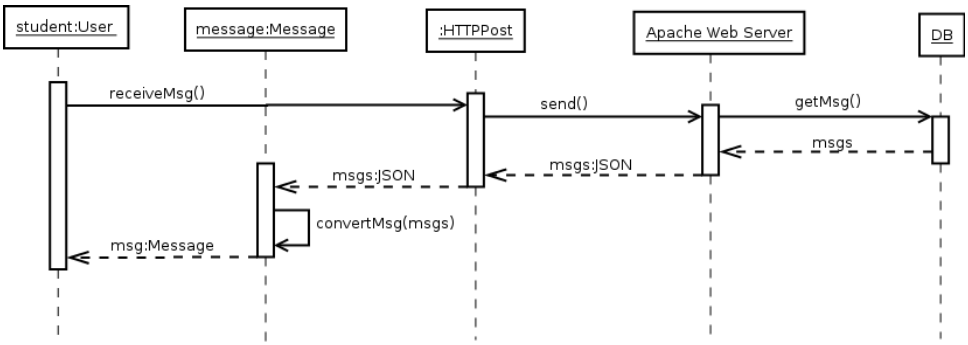
In order to get a list of the possible roommates for the user, user will call the *getStudents()* with the user's location and search radius. The *HTTPPost* will then send the request to the Apache server. The server will pull the data from the database. Then the data will be translated by the server into a JSON object. *HTTPPost* will call *parseList()* to convert the JSON object to an array of student information for the *User* class to store.

6.2.5 User Logging In



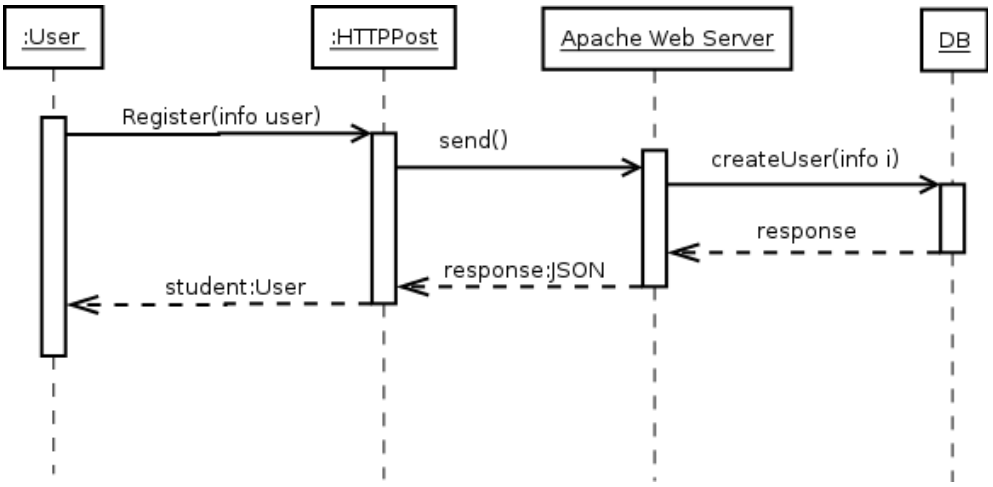
A student will have to login first in order the College Living. A student will fill in his/her login information and call the *login()* method. The *HTTPPost* send the request to the Apache server. The web server will call *validate()* to verify the user's request with the database. The server will send the response back to *HTTPPost*, and *HTTPPost* will send the success boolean back to user.

6.2.6 Receiving Messages



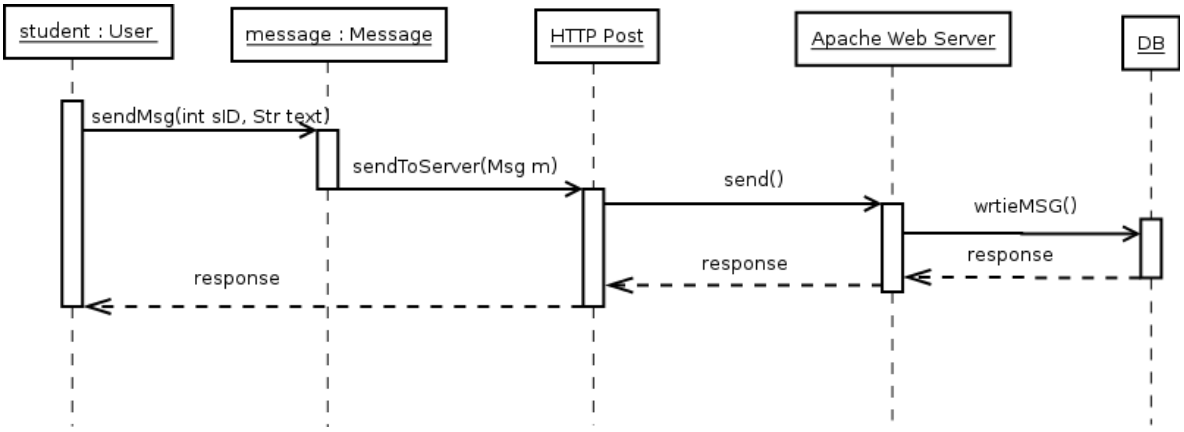
Student will need to receive messages from the server. *User* will have to call *receiveMsg()*, which will use *HTTPPost* to send a request to the Apache server. The server will retrieve the messages in the database for the current user with the corresponding student ID. Then the web server will convert the data to a JSON object and send back to *HTTPPost*. *HTTPPost* will call the *convertMsg()* method in the *Message* class to translate the JSON object to readable text and send back to the student.

6.2.7 User Registration



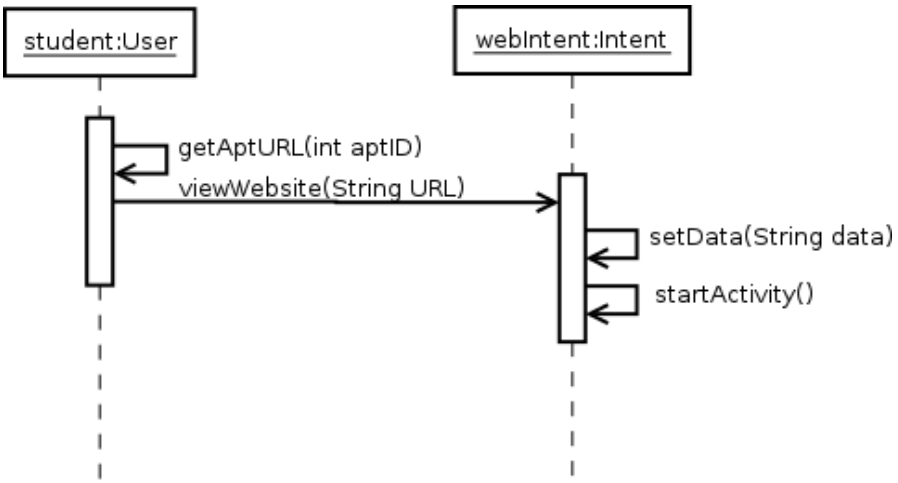
Users will have to register with College Living the first time they use the application. After opening the app the static method Register() will be called. This will bring the user to a registration form, after the user finishes filling the form out the information will be sent via *HTTPPost* to the Apache server. The web server will then add the new user to the database. Finally, a response will be sent back to the user showing either a successful or unsuccessful register.

6.2.8 Sending a Message



To send a message the user will call the *sendMsg*() method with the student ID they wish to send the message to and the text of the message. Then the Message class will create the message and send it to the server through *HTTPPost*. Next the server will send the message to the database where it is stored until the receiver gets the message from the database. Finally, the user will receive a response of whether the sending of the message was successful or not.

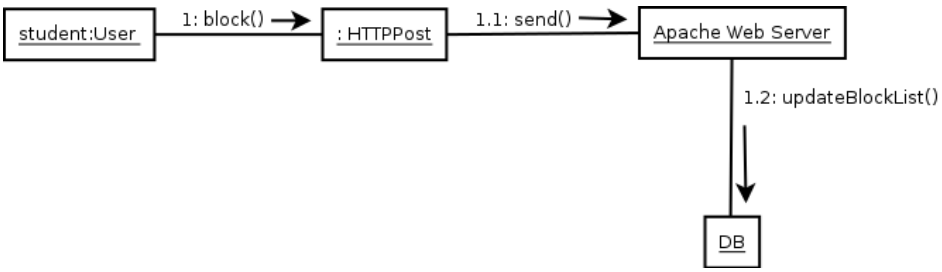
6.2.9 Viewing an Apartment's Website



The user will use the *getAptURL()* method with the apartment's ID to get an apartment's URL address. Then the *viewWebsite()* method is called with the URL address. An Intent is created with apartment's URL passed in, then an activity is started that takes the user to the chosen apartment's website.

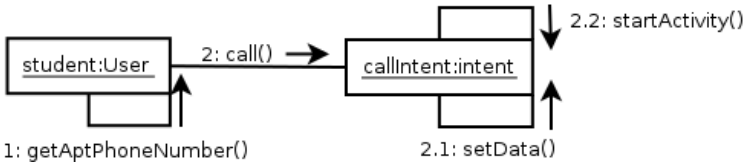
6.3 Communication Diagrams

6.3.1 Blocking a User



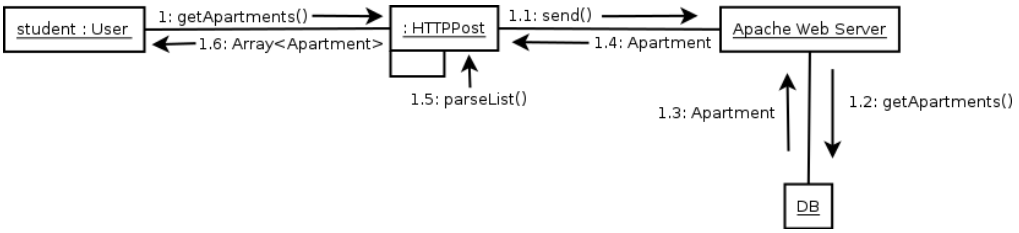
A user can block other students from communicating with them. Once the User class has been initiated with the students credentials and logged in, the *block()* method can be called with the other student's ID number. This method will send a request to the *HTTPPost* class, which will forward the request to the Apache web server. The server will call a method to update the blocked user list in the database. The database will return a Boolean to the server, indicating whether this operation was successful. The server will encode this Boolean as a JSON object and return it to the application.

6.3.2 Calling an Apartment



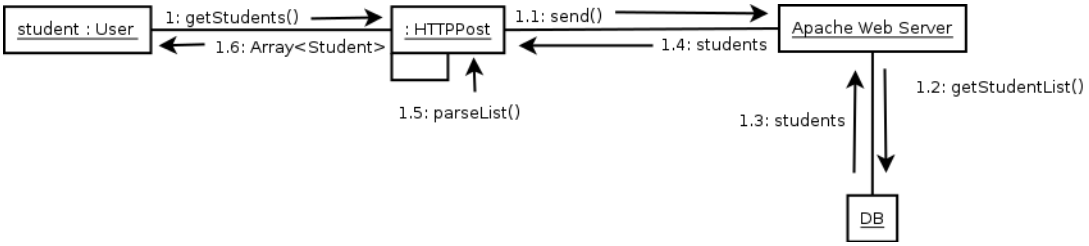
A user can call an apartment complex to get more information about their units. When the user wants to call an apartment, the *User* class will use the *getAptPhoneNumber()* method with the apartment's ID number. This method will return a phone number for the apartment. This phone number will get passed into the *call()* method. This method will set up a new *Intent*, with the phone number passed into the *setData()* method. Once the activity is started, the application will call the supplied phone number.

6.3.3 Getting a List of Local Apartments



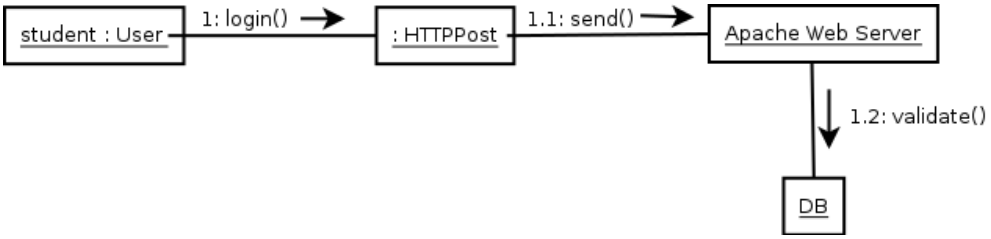
The application needs a list of local apartments for the user to interact with. The *User* class will call the *getApartments()* method with the user's current location and a search radius. This method will send a request to *HTTPPost*, which will forward the request to the Apache server. The server will query the database for a list of apartments that are close to the user. This list will get returned to the application, first as a JSON response. It will then be parsed into an array of *Apartment* instances.

6.3.4 Getting a List of Local Students



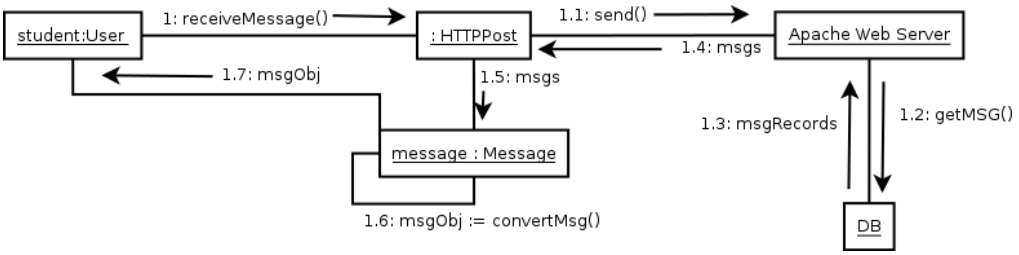
In order to get a list of the possible roommates for the user, user will call the *getStudents()* with the user's location and search radius. The *HTTPPost* will then send the request to the Apache server. The server will pull the data from the database. Then the data will be translated by the server into a JSON object. *HTTPPost* will call *parseList()* to convert the JSON object to an array of student information for the *User* class to store.

6.3.5 User Logging In



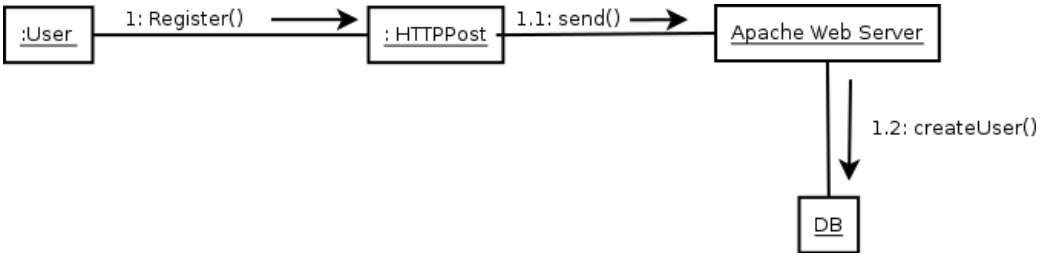
A student will have to login first in order the College Living. A student will fill in his/her login information and call the *login()* method. The *HTTPPost* send the request to the Apache server. The web server will call *validate()* to verify the user's request with the database. The server will send the response back to *HTTPPost*, and *HTTPPost* will send the success boolean back to user.

6.3.6 Receiving Messages



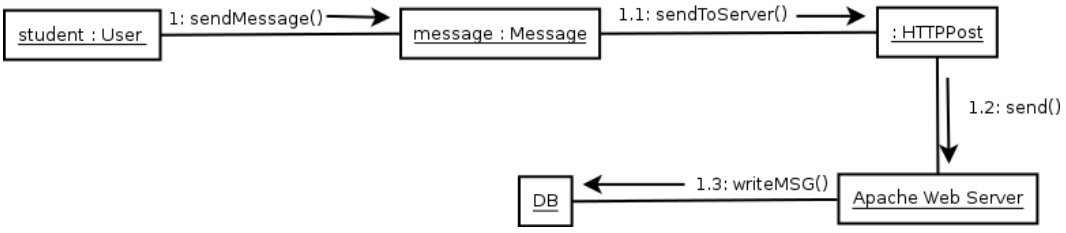
Student will need to receive messages from the server. *User* will have to call *receiveMsg()*, which will use *HTTPPost* to send a request to the Apache server. The server will retrieve the messages in the database for the current user with the corresponding student ID. Then the web server will convert the data to a JSON object and send back to *HTTPPost*. *HTTPPost* will call the *convertMsg()* method in the *Message* class to translate the JSON object to readable text and send back to the student.

6.3.7 User Registration



Users will have to register with College Living the first time they use the application. After opening the app the static method Register() will be called. This will bring the user to a registration form, after the user finishes filling the form out the information will be sent via *HTTPPost* to the Apache server. The web server will then add the new user to the database. Finally, a response will be sent back to the user showing either a successful or unsuccessful register.

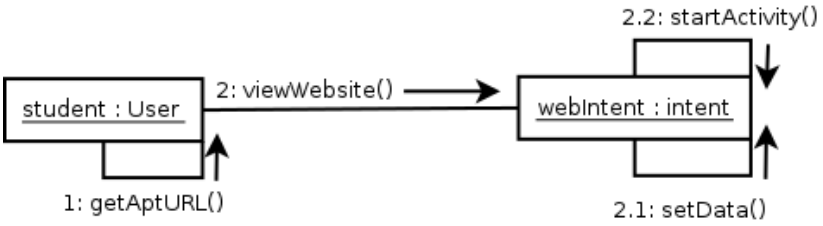
6.3.8 Sending Messages



To send a message the user will call the *sendMsg*() method with the student ID they wish to send the message to and the text of the message. Then the Message class will create the message and send it to the server through *HTTPPost*. Next the server will send the message to

the database where it is stored until the receiver gets the message from the database. Finally, the user will receive a response of whether the sending of the message was successful or not.

6.3.9 Viewing Apartment Website



The user will use the *getAptURL()* method with the apartment's ID to get an apartment's URL address. Then the *viewWebsite()* method is called with the URL address. An Intent is created with apartment's URL passed in, then an activity is started that takes the user to the chosen apartment's website.