CS5704 Software Engineering

Fall 2016

ApartMates

Meghendra Singh
meghs@vt.edu

Soumya Vundekode
soumyav@vt.edu

Andres Pico
andresjp@vt.edu

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

Date:	December 15, 2016
Team Number:	2
Submitted to:	Prof. Osman Balci

EXECUTIVE SUMMARY

Living in a shared apartment has a lot of benefits the biggest being low rent, but it also comes with its fair share of problems. A major problem is sharing of household responsibilities and expenses. Who will bring the groceries this week? Who will take care of the cleaning? Who will pay the cable bill this month? It becomes nearly impossible to keep track of the many tasks that keep the apartment running. It can be extremely difficult to delegate and share all the responsibilities and expenses in a shared apartment. Our cloud software application **ApartMates** will solve this problem and bring order to this chaos. All roommates can register and share tasks and expenses. Each user shall be able to:

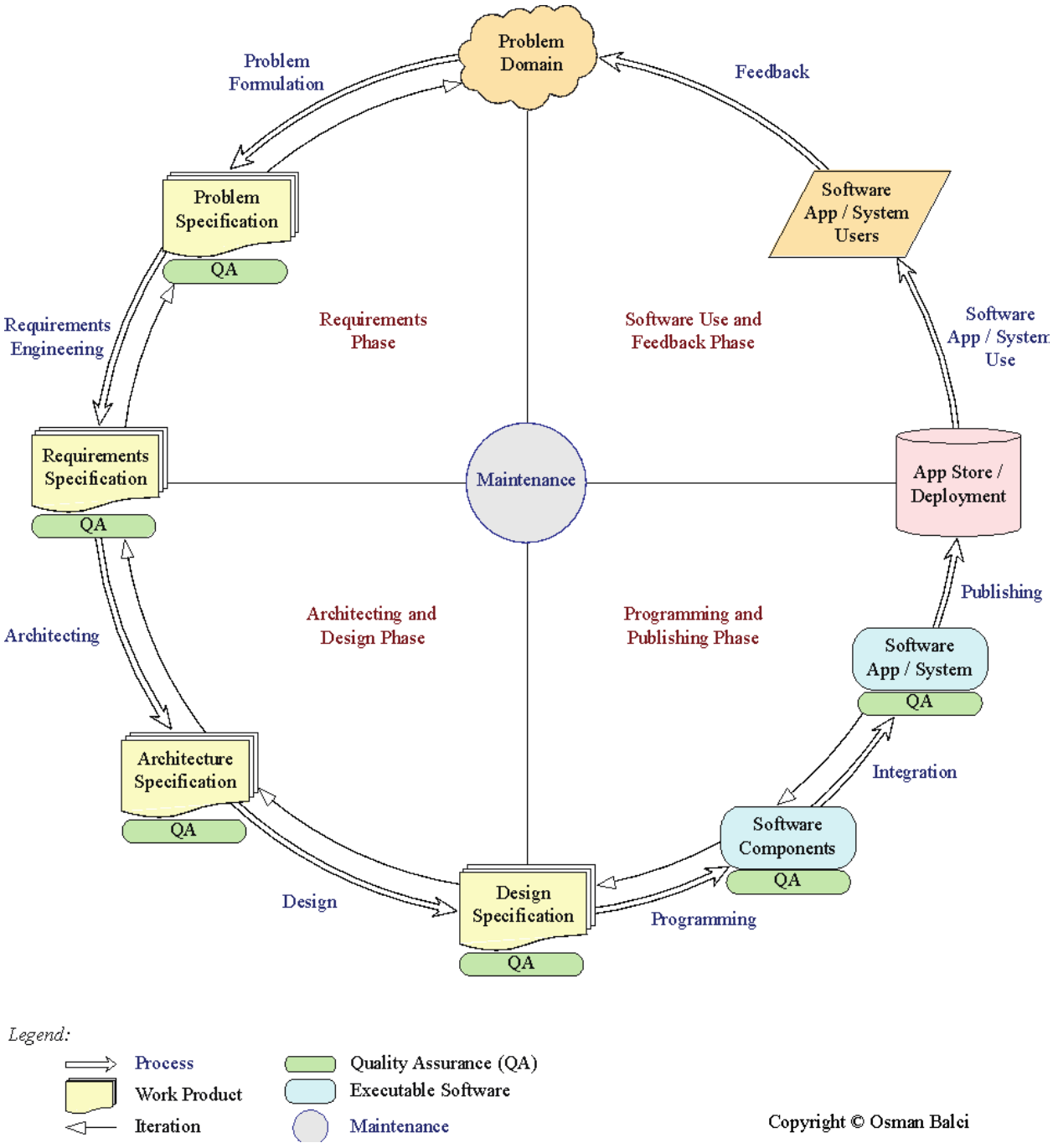
1. Create profiles using existing valid email IDs and have options for updating this profile with their preferred profile picture.
2. Create a virtual representation of their shared home (Apartment), complete with other users as their roommates (ApartMates).
3. Create task lists with deadlines and priorities shared amongst all ApartMates.
4. Earn reputation for finishing tasks before time, earn more points for finishing tasks earlier to compete with other ApartMates.
5. Keep a log of their shared expenses.
6. Maintain a record of which other ApartMates owe the user how much money.
7. Automatically generate grocery lists with approximate prices by providing their food item wish list.
8. Add existing users to their virtual Apartment.

The cloud software application shall also provide a RESTful interface for other applications to access the reputation points associated with each unique email id registered in the system. The reputation points earned by each user can also act like ‘credit ratings’ which can be used by other services to recommend the best roommates out there.

Contents

1. SOFTWARE LIFE CYCLE

A **good software engineer** develops software by following the software life cycle shown below.



A **programmer (hacker or ad-hoc developer)** develops software by looking at the problem and directly coding in an IDE. This approach is known as the **Build-and-Fix Approach**, which must never be used!

2. PROBLEM SPECIFICATION

This project will involve architecting, designing and developing a cloud software application for the purpose of allowing its users to keep track of tasks associated with their real world shared living spaces. Share these tasks with their real world roommates and keep records of the expenses they have incurred. This cloud software application will provide an interface for users to create an account, create a virtual apartment, create tasks with deadlines and priorities that represent real world tasks associated with their real world apartments. Mark these tasks as completed and delete them from the list of tasks once they are complete. Earn reputation points for finishing tasks. Generate grocery lists based on their food wish lists. A relational database management system will be used to store the data for all the users, their created apartments, tasks and expenses.

The cloud software application will be architected based on DoDAF specifications, and designed using standard UML descriptors. The developed cloud software application will allow the user to keep track of their day to day chores and expenses online.

Users earn points only for completing tasks before their deadlines so that these work as an incentive, and these reputation points are also available over a RESTful API. This would enable other services to use the reputation data in creative ways.

The cloud software application must be usable, informative, robust, responsive, user-friendly and secure. It should be designed to allow extensions like assigning tasks to individual users, summarizing expenses, email/SMS based reminders for upcoming deadlines, badges for reaching high reputation levels.

3. REQUIREMENTS SPECIFICATION

The results of our execution of the use cases-based requirements engineering process are summarized in Table 1.

Table 1. Summary of Use Cases and Functional Requirements

Use Case Number	*Use Case Name*	*Number of Functional Requirements Associated with the Use Case*
1	User Registration	5
2	View Profile	5
3	Edit Profile Information	5
4	Delete User Account	5
5	Login	9
6	Create Apartment	6
7	View Apartment	5
8	Edit Apartment	5
9	Delete Apartment	5
10	Add Tasks	5
11	View Tasks	5
12	Update Tasks	5
13	Delete Tasks	3
14	Mark Tasks	3
15	Add Expense	5
16	View Expenses	5
17	Update Expense	5
18	Delete Expense	3
19	Grocery List	2
20	Reward Points	4
21	View Roommate Profiles	3
Total:	21	98

3.1 Use Case 1: User Registration

3.1.1 Use Case "User Registration" Documentation

Use Case Documentation Template	
Use Case ID:	1
Use Case Name:	User Registration - This use case describes the process by which a Roommate can register a new account in ApartMates
Actors:	
User	
Preconditions:	
User must have a valid email account	
Flow of Events of the Primary Scenario:	
1. Use case starts when user opens ApartMates home page. 2. User selects "Register". 3. User provides the following information: a. First name b. Last name c. Email address d. Password 4. User selects "Submit" 5. User has access to new account	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
User selects 'Cancel' without submitting changes. User attempts registration with an existing email address	
Extension Points:	
None	
"Used" Use Cases:	
None	
Postconditions:	
Roommate has access to an ApartMates valid account.	

3.1.2 Functional Requirements Associated with Use Case "User Registration"

The functional requirements associated with use case "User Registration" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. User shall be able to create at most 1 account per email address
2. User shall be given the option to provide a first and last name
3. Password shall consist of minimum 6 and maximum 16 characters with at least 1 uppercase, 1 lowercase, 1 number and 1 special character.
4. User shall be able to save answer for a security question.
5. User shall be able to view the dashboard after clicking 'Submit'.

3.2 Use Case 2: View Profile

3.2.1 Use Case "View Profile" Documentation

Use Case Documentation Template	
Use Case ID:	2
Use Case Name:	View Profile - This use case describes the process by which a user can view his/her profile
Actors:	
User/Roommate	
Preconditions:	
User must have a valid ApartMates account.	
Flow of Events of the Primary Scenario:	
1. Use case starts when the user/roommate selects "My profile" 2. System displays the following information about the User: a. First name b. Last name c. Email address d. Apartment Name (If the user is a roommate, i.e., belonging to an apartment)	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
Retrieval of data is unsuccessful	
Extension Points:	
Update Profile Delete ApartMates account	
"Used" Use Cases:	
None	
Postconditions:	
ApartMates displays a screen with information about the current Roommate	

3.2.2 Functional Requirements Associated with Use Case "View Profile"

The functional requirements associated with use case "View Profile" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. User shall be able to select an option to view his/her profile
2. Profile page shall display the following information about the user: first and last name, email address, name of apartment if the user is a roommate (belongs to an apartment).
3. User shall be given the option to edit the profile.
4. User shall be able to upload his/her picture to the profile.
5. User shall be given the option to delete his/her account.

3.3 Use Case 3: Edit Profile Information

3.3.1 Use Case "Edit Profile Information" Documentation

Use Case Documentation Template	
Use Case ID:	3
Use Case Name:	Edit Profile Information - This use case describes the process by which a user can update his/her profile information
Actors:	
User	
Preconditions:	
User must have a valid ApartMates account	
Flow of Events of the Primary Scenario:	
1. Use case starts when user selects "Edit Profile" on the profile screen. 2. User changes optionally the following information about his/her account: a. User first name b. User last name c. User password 3. User selects submit and changes are stored (if any). 4. System notifies the user of changes 5. System displays the profile screen with updated information.	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
User selects cancel at any time	
Extension Points:	
None	
"Used" Use Cases:	
None	
Postconditions:	
User is notified on whether changes were successful or not.	

3.3.2 Functional Requirements Associated with Use Case "Edit Profile Information"

The functional requirements associated with use case "Edit Profile Information" are listed below. Quality of each requirement specification is evaluated with respect to the following quality

characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Users shall be able to change/update information about their account.
2. Users shall only have access to change information about their own account.
3. Users shall not leave any ‘required’ field blank.
4. Users shall have the option to change the following information about their account: first name, last name, and password
5. Password shall consist of minimum 6 and maximum 16 characters with at least 1 uppercase, 1 lowercase, 1 number and 1 special character.

3.4 Use Case 4: Delete User Account

3.4.1 Use Case "Delete User Account" Documentation

Use Case Documentation Template	
Use Case ID:	4
Use Case Name:	Delete User Account - This use case describes the process by which a user can delete his/her account information
Actors:	
Users	
Preconditions:	
User must have a valid ApartMates account	
Flow of Events of the Primary Scenario:	
1. Use case starts when user selects "Delete Account". 2. System prompts the user with action verification message. 3. User verifies deletion intent by selecting "Yes". 4. User is redirected to registration screen.	
Flow of Events of the Alternative Scenarios:	
User selects "No" when prompted about deletion intent.	
Flow of Events of the Exception Scenarios:	
Deletion of account is unsuccessful when the user is the only roommate of an apartment, in which case, the apartment needs to be deleted first before deleting user account.	
Extension Points:	
None	
"Used" Use Cases:	
None	
Postconditions:	
User is notified on whether changes were successful or not. User is redirected to registration screen.	

3.4.2 Functional Requirements Associated with Use Case "Delete User Account"

The functional requirements associated with use case "Delete User Account" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity,

understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. User shall be able to delete his/her account.
2. User shall be asked an action verification message in case the user in question changes his/her mind.
3. User shall be given the option to deny the deletion before the account is deleted.
4. User shall be redirected to registration screen.
5. User shall not be able to delete his/her account if he/she is the sole roommate of an apartment.

3.5 Use case 5: Login

3.5.1 Use Case "Login" Documentation

Use Case Documentation Template	
Use Case ID:	5
Use Case Name:	Login - This use case describes the process by which a registered user logs into the system.
Actors:	
Users	
Preconditions:	
1. Users is registered with a valid email ID. 2. Login screen is displayed.	
Flow of Events of the Primary Scenario:	
1. The use case starts when the user gets the login screen. 2. If the user selects 'Register', use use case 1: Register 3. If the user enters an email ID and password and selects 'Login': a. The system verifies the information. b. If the credentials are correct, the system displays the user's dashboard. c. If the user doesn't belong to any apartment, the dashboard provides an option to create a new apartment. d. If the user selects 'Create new apartment', use use case: Create apartment. e. If the roommate selects 'View tasks', use use case: View tasks f. If the roommate selects 'View expenses', use use case: View expenses g. If the roommate selects 'Add task', use use case: Add task h. If the roommate selects 'Add expense', use use case: Add expense 4. The use case ends.	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
1. If the email ID is not recognized, the system displays error message - "Username is not recognized. Please register." 2. Else if the password and username don't match, the system displays error message - "Username and password don't match. Please retry."	
Extension Points:	
None	
"Used" Use Cases:	
1. Register 2. Create apartment 3. View tasks 4. View expenses 5. Add task 6. Add expense	

Postconditions:
1. User is authenticated and dashboard is displayed.

3.5.2 Functional Requirements Associated with Use Case "Login"

The functional requirements associated with use case "Login" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. User shall be given the option to register or login to the system.
2. User shall enter a valid email ID and password to login to the system.
3. User shall be authenticated with the email ID and password entered.
4. Password shall consist of minimum 6 and maximum 16 characters with at least 1 uppercase, 1 lowercase, 1 number and 1 special character.
5. User shall be given the option to create an apartment, if he/she is not a roommate yet.
6. Registered user shall be given the option to add a task.
7. Registered user shall be given the option to add an expense.
8. Registered user shall be able to view all tasks.
9. Registered user shall be able to view all expenses.

3.6 Use Case 6: Create Apartment

3.6.1 Use Case "Create Apartment" Documentation

Use Case Documentation Template	
Use Case ID:	6
Use Case Name:	Create Apartment - This use case describes the process by which user creates an apartment.
Actors:	
Users	
Preconditions:	
Valid user has logged in to the system	
Flow of Events of the Primary Scenario:	
1. Use case starts when user selects "Create Apartment" 2. User enters an apartment's name and an optional apartment's address. 3. User selects submit. 4. System creates an apartment with the name given by the user and makes him/her a 'roommate' of the apartment. 5. The now 'roommate' is notified of result.	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
User selects cancel before creation of apartment is complete.	
Extension Points:	
None	
"Used" Use Cases:	
Login	
Postconditions:	
An apartment is created, and the roommate is notified	

3.6.2 Functional Requirements Associated with Use Case "Create Apartment"

The functional requirements associated with use case "Create Apartment" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity,

understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommates shall be able to start creation of apartment.
2. Roommates shall belong to 1 apartment only.
3. Roommates shall be able to cancel creation of apartment.
4. Roommates shall be able to invite other registered users to join apartment and become roommates.
5. Roommates shall be required to provide a name for the apartment.
6. Roommates shall be given the option to provide an address for the apartment.

3.7 Use Case 7: View Apartment

3.7.1 Use Case "View Apartment" Documentation

Use Case Documentation Template	
Use Case ID:	7
Use Case Name:	View Apartment - This use case describes the process by which roommates can view their apartment information.
Actors:	
Roommates	
Preconditions:	
Roommate has logged in to the system and belongs to an existing apartment.	
Flow of Events of the Primary Scenario:	
1. Use case starts when roommate selects "View Apartment". 2. System displays the following information about the apartment: a. Apartment's name b. Apartment's address (if any) c. Roommates belonging to the apartment 3. System displays the points earned by the roommate till date.	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
Retrieval of information from database is unsuccessful.	
Extension Points:	
Update Apartment	
"Used" Use Cases:	
None	
Postconditions:	
Roommate has access to information about the apartment.	

3.7.2 Functional Requirements Associated with Use Case "View Apartment"

The functional requirements associated with use case "View Apartment" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity,

understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommates shall be able to view information about their apartment.
2. Roommates shall be able to see who the roommates of the apartment are.
3. Roommates shall only be able to view only his/her own apartment details.
4. Roommates shall be given the option to edit, leave or delete the apartment.
5. Roommates shall be given the option to invite any registered user to join the apartment as roommate.

3.8 Use Case 8: Edit Apartment Information

3.8.1 Use Case "Edit Apartment Information" Documentation

Use Case Documentation Template	
Use Case ID:	8
Use Case Name:	Edit Apartment Information - This use case describes the process by which a Roommate can change the information of an apartment.
Actors:	
Roommates	
Preconditions:	
Valid roommate has logged in to the system and belongs an existing apartment.	
Flow of Events of the Primary Scenario:	
1. Use case begins when roommate selects "Edit Apartment" 2. Roommate changes optionally the following information about the apartment: a. Apartment's name b. Apartment's address (if any) 3. Roommate selects submit and changes are stored (if any). 4. System notifies roommate whether changes were successful or not 5. System displays the "View Apartment" screen.	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
Roommate selects cancel at any time before submitting changes. Changes are saved unsuccessfully.	
Extension Points:	
None	
"Used" Use Cases:	
View Apartment	
Postconditions:	
Roommate is notified, and is able to see changes.	

3.8.2 Functional Requirements Associated with Use Case "Edit Apartment Information"

The functional requirements associated with use case "Edit Apartment Information" are listed below. Quality of each requirement specification is evaluated with respect to the following

quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommates shall be able to change/update information about their apartment.
2. All roommates shall be notified of any changes made to the apartment's information.
3. Roommates shall only have access to change information about their own apartments.
4. Roommates shall not leave any blank field.
5. Roommates shall have the option to change both the name of the apartment and the address of it.

3.9 Use Case 9: Delete Apartment

3.9.1 Use Case "Delete Apartment" Documentation

Use Case Documentation Template	
Use Case ID:	9
Use Case Name:	Delete Apartment - This use case describes the process by which a Roommate can delete his/her apartment
Actors:	
Roommate	
Preconditions:	
Valid roommate has logged in to the system and belongs an existing apartment	
Flow of Events of the Primary Scenario:	
1. Use case starts when roommate selects "Delete Apartment". 2. System prompts roommate with action verification message. 3. Roommate verifies deletion intent by selecting "Yes". 4. Roommates are notified of apartment deletion.	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
Roommate selects cancel at the action verification message	
Extension Points:	
None	
"Used" Use Cases:	
View Apartment	
Postconditions:	
Roommate is notified, and is able to see changes.	

3.9.2 Functional Requirements Associated with Use Case "Delete Apartment"

The functional requirements associated with use case "Delete Apartment" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommates shall be able to view information about their apartment.
2. Roommate shall be able to delete his/her apartment.
3. Roommate shall be given a chance to confirm his/her deletion intent.
4. Roommate shall be notified of the success/failure of apartment deletion activity.
5. Roommate shall be taken back to Dashboard with ‘create apartment’ option.

3.10 Use Case 10: Add Tasks

3.10.1 Use Case "Add Tasks" Documentation

Use Case Documentation Template	
Use Case ID:	10
Use Case Name:	Add Tasks: This use case allows roommates that are associated with an apartment to create new tasks relevant to that apartment.
Actors:	
Roommates	
Preconditions:	
A valid user (i.e. a roommate) has logged into the system and is associated with the apartment for which the task is being assigned.	
Flow of Events of the Primary Scenario:	
1. The use case starts when the roommate selects Add Task. 2. The roommate enters the name for the task. 3. The roommate can set optional attributes associated with the task like: a. Task description b. Deadline c. Priority (low, medium or high) d. Reminder timestamps (Upto three reminders, before the deadline) e. Geolocation associated with the task 4. The roommate selects Submit 5. The system saves the task.	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
If the roommate selects Cancel at any time before selecting Submit.	
Extension Points:	
None as of now.	
"Used" Use Cases:	
1. Roommate information associated with the apartment, for which the task is being created.	
Postconditions:	
The task is saved in the system.	

3.10.2 Functional Requirements Associated with Use Case "Add Tasks"

The functional requirements associated with use case "Add Tasks" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommates shall be able to add 'tasks' to the system, that are associated with the user's apartment. A task shall have an alphanumeric name of arbitrary length specified by the user. A task shall have an unique ID that is generated by the system.
2. The task shall have optional attributes: Task description, Deadline, Priority, Reminders and Geolocation.
3. The deadline attribute associated with a task shall be a timestamp that is greater than the timestamp at which the task is added to the system.
4. The reminders attribute shall be a maximum of three timestamps all of which are greater than the timestamp at which the task is added to the system, and less than the deadline attribute, if a deadline attribute is specified.
5. Roommates shall be able to see the task added on the dashboard or view-tasks screen.

3.11 Use Case 11: View Tasks

3.11.1 Use Case "View Tasks" Documentation

Use Case Documentation Template	
Use Case ID:	11
Use Case Name:	View Tasks: This use case allows roommates that are associated with an apartment to view all tasks associated with their apartment.
Actors:	
Roommates	
Preconditions:	
A valid user (i.e. a roommate) has logged into the system and is associated with the apartment for which the tasks are being displayed.	
Flow of Events of the Primary Scenario:	
1. The use case starts when the roommate logs in. 2. The system displays on the dashboard - a list of all the tasks, which are associated with their apartment. 3. The roommate selects one of these tasks. 4. The system displays detailed information about the task viz. what are the attributes of the task as specified at the time of creation or update, what is the state of the task i.e. is the task marked 'complete' or 'pending'.	
Flow of Events of the Alternative Scenarios:	
There are no tasks associated with the roommates' apartment in the system. So, the use case ends.	
Flow of Events of the Exception Scenarios:	
None as of now.	
Extension Points:	
None as of now.	
"Used" Use Cases:	
None as of now.	
Postconditions:	
None as of now.	

3.11.2 Functional Requirements Associated with Use Case "View Tasks"

The functional requirements associated with use case "View Tasks" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics

and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. The user of ApartMates shall be able to view all ‘tasks’ associated with their apartment that exist in the system.
2. The system shall display all tasks that exist in the system and are associated with the viewing user’s apartment irrespective of their state i.e. ‘pending’, ‘complete’.
3. The system shall display all the details that are associated with a task in the system. These consist of: Task name, description, deadline, priority, reminders, geolocation, time required.
4. The user shall be given the option to add more or edit any existing tasks.
5. The user shall be given the option to delete any existing tasks.

3.12 Use Case 12: Update Tasks

3.12.1 Use Case “Update Tasks” Documentation

Use Case Documentation Template	
Use Case ID:	12
Use Case Name:	Update Tasks: This use case allows roommates that are associated with an apartment to update the details of the any task which is associated with their apartment.
Actors:	
Roommates	
Preconditions:	
A valid user (i.e. a roommate) has logged into the system and is associated with the apartment for which the tasks are being updated.	
Flow of Events of the Primary Scenario:	
1. The use case starts when the roommate selects a task and selects 'Edit'. 2. The roommate can update the following details associated with a task: a. Task name b. Task description c. Deadline d. Priority (low, medium or high) e. Reminder timestamps (Maximum of three reminders, before the deadline) f. Geolocation associated with the task 3. The roommate selects Submit and the system verifies that the roommate is associated with the apartment for which the task is being updated and validates the various task attributes specified. 4. The system saves the task in the state it was before the update (i.e. pending or complete) and the use case ends.	
Flow of Events of the Alternative Scenarios:	
1. If there are no tasks associated with the roommate's apartment in the system, a message is displayed specifying that "No tasks have been added yet" and the use case ends. 2. If the roommate selects Cancel at any time before selecting Submit.	
Flow of Events of the Exception Scenarios:	
3. Task name left blank. 4. Invalid timestamp selected for the deadline attribute. 5. Invalid timestamps selected for the reminders attribute. 6. Invalid roommate selected for the assignee.	
Extension Points:	
None as of now.	
"Used" Use Cases:	
Details associated with the task, which is being updated.	

Postconditions:
The task is saved in the system and marked with its earlier state (pending/complete).

3.12.2 Functional Requirements Associated with Use Case "Update Tasks"

The functional requirements associated with use case "Update Tasks" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. The user of ApartMates shall be able to update only existing 'tasks' in the system that are associated with their apartment. The task being updated shall have an alphanumeric name of arbitrary length specified by the user who is updating the task.
2. The optional deadline attribute associated with the task being updated shall be a timestamp that is greater than the timestamp at which the task is added to the system.
3. The optional reminders attribute associated with the task being updated shall be a maximum of three timestamps all of which are greater than the timestamp at which the task is added to the system, and less than the deadline attribute, if a deadline attribute is specified.
4. The roommate shall be able to mark the task as completed.
5. The roommate shall be able to see the updated task on his/her dashboard.

3.13 Use Case 13: Delete Tasks

3.13.1 Use Case "Delete Tasks" Documentation

Use Case Documentation Template	
Use Case ID:	13
Use Case Name:	Delete Tasks: This use case allows roommates that are associated with an apartment to delete any task associated with their apartment.
Actors:	
Roommates	
Preconditions:	
A valid user (i.e. a roommate) has logged into the system and is associated with the apartment for which the task is being deleted.	
Flow of Events of the Primary Scenario:	
1. The use case starts when the roommate selects a task and selects 'Delete'. 2. The system prompts the roommate asking for confirmation. 3. The user chooses delete. 4. System deletes the task. 5. Use case ends.	
Flow of Events of the Alternative Scenarios:	
Roommate selects No when prompted for confirmation, in which case, system returns to the list without deleting any task.	
Flow of Events of the Exception Scenarios:	
None	
Extension Points:	
None	
"Used" Use Cases:	
Details associated with the task, which is being deleted.	
Postconditions:	
The task is deleted from the system.	

3.13.2 Functional Requirements Associated with Use Case "Delete Tasks"

The functional requirements associated with use case "Delete Tasks" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommates shall be able to delete ‘tasks’ that correspond to their apartments.
2. Roommates shall be given a chance to confirm the deletion intent.
3. Roommates shall be able to confirm that the task has been deleted by viewing the list of existing tasks.

3.14 Use Case 14: Mark Tasks

3.14.1 Use Case "Mark Tasks" Documentation

Use Case Documentation Template	
Use Case ID:	14
Use Case Name:	Mark Tasks: This use case allows roommates that are associated with an apartment to quickly mark any task associated with their apartment as complete or pending.
Actors:	
Roommates	
Preconditions:	
A valid user (i.e. a roommate) has logged into the system and is associated with the apartment for which the task is being marked as complete or pending.	
Flow of Events of the Primary Scenario:	
1. The use case starts when the roommate views tasks on the dashboard. 2. The system displays a list of all the tasks to the roommate, which are associated with their apartment and exist in the system. 3. The roommate selects one of these tasks. 4. The system displays information about the task and asks the roommate if it wants a pending task to be marked as complete or a complete task as pending. 5. The user chooses pending or complete depending on the previous state of the task, the task state is updated and the use case ends.	
Flow of Events of the Alternative Scenarios:	
1. If there are no tasks associated with the roommates apartment in the system, a message is displayed specifying that "No tasks have been added yet" and the use case ends.	
Flow of Events of the Exception Scenarios:	
None	
Extension Points:	
None as of now.	
"Used" Use Cases:	
Details associated with the task, which is being marked as complete or pending.	
Postconditions:	
The task state is updated in the system.	

3.14.2 Functional Requirements Associated with Use Case "Mark Tasks"

The functional requirements associated with use case "Mark Tasks" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommates shall be able to mark 'tasks' that exist in the system as complete or pending.
2. The system shall only allow a pending task to be marked as complete, and a complete task to be marked as pending.
3. Roommates shall be able to see the updated information about the task being completed on the dashboard.

3.15 Use Case 15: Add Expense

3.15.1 Use Case "Add Expense" Documentation

Use Case Documentation Template	
Use Case ID:	16
Use Case Name:	Add Expense : This use case describes the process by which roommates can add an expense to their list of expenses in the system.
Actors:	
Roommates	
Preconditions:	
1. User is a roommate belonging to an apartment.	
Flow of Events of the Primary Scenario:	
1. Use case starts when the roommate clicks on 'Add expense' on the dashboard. 2. The roommate enters details of the expense like name, amount and roommates the expense is to be split between. 3. The roommate may enter non-mandatory information like date, location or brief description of expense. 4. The roommate selects 'Add'. 5. The system validates the data entered by the roommate. 6. The system adds the expense to the apartment's list of expenses. 7. The system displays the dashboard with the expense added. 8. Use case ends.	
Flow of Events of the Alternative Scenarios:	
1. If the roommate selects 'Cancel' before adding, dashboard is displayed.	
Flow of Events of the Exception Scenarios:	
1. If the data validation is unsuccessful, the system denies the add request and displays a pop-up message that the entered data doesn't meet its requirements.	
Extension Points:	
None	
"Used" Use Cases:	
None	
Postconditions:	
The expense is added to the list of the apartment's expenses.	

3.15.2 Functional Requirements Associated with Use Case "Add Expense"

The functional requirements associated with use case "Add Expense" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommate shall be able to add any expense incurred.
2. Roommate shall enter a name for the expense. The name shall consist of 8-20 characters.
3. Roommate shall enter the amount of expense incurred. The amount shall be a floating point number with maximum two decimal digits.
4. Roommate shall select the list of people involved in the expense, between whom the expense is to be shared.
5. Roommate shall be able to go back to dashboard without adding any expense.

3.16 Use Case 16: View Expenses

3.16.1 Use Case "View Expenses" Documentation

Use Case Documentation Template	
Use Case ID:	16
Use Case Name:	View Expenses : This use case describes the process by which roommates can view list of all expenses they are involved in.
Actors:	
Roommates	
Preconditions:	
Roommate is logged in and dashboard is displayed.	
Flow of Events of the Primary Scenario:	
1. Use case starts when the roommate sees the expenses list on the dashboard. 2. The system displays the list of all expenses added by the roommate. 3. The roommate selects any of the expenses and selects 'Details'. 4. The system displays information about the selected expense. 5. If the roommate selects 'Edit', use 'Update expense'. 6. The use case ends.	
Flow of Events of the Alternative Scenarios:	
1. If the roommate selects 'Back', dashboard is displayed.	
Flow of Events of the Exception Scenarios:	
None	
Extension Points:	
None	
"Used" Use Cases:	
Update expense	
Postconditions:	
User is able to view the list of all expenses and their details.	

3.16.2 Functional Requirements Associated with Use Case "View Expenses"

The functional requirements associated with use case "View Expenses" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity,

understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommate shall be able to view all the expenses of the apartment.
2. Roommate shall be able to see all the details of any selected expense.
3. Roommate shall be given the option to delete any expense.
4. Roommate shall not be able to add an expense if it is not shared with at least one other roommate.
5. Roommate shall be able to edit any expense or mark it as settled.

3.17 Use Case 17: Update Expense

3.17.1 Use Case "Update Expense" Documentation

Use Case Documentation Template	
Use Case ID:	17
Use Case Name:	Update Expense: This use case describes the process by which roommates can update an expense they are involved in.
Actors:	
Roommates	
Preconditions:	
Roommate is able to view the expense details.	
Flow of Events of the Primary Scenario:	
1. Use case starts when the roommate selects any expense on the dashboard and clicks 'Edit'. 2. Roommate edits the expense details. 3. Roommate selects 'Update'. 4. The system validates and saves the updated expense information. 5. The system displays the updated list of expenses. 7. The use case ends.	
Flow of Events of the Alternative Scenarios:	
1. If the roommate selects 'Cancel', the list of expenses is displayed.	
Flow of Events of the Exception Scenarios:	
None	
Extension Points:	
None	
"Used" Use Cases:	
Details associated with the expense, which is being deleted.	
Postconditions:	
1. Roommate is able to update the expense.	

3.17.2 Functional Requirements Associated with Use Case "Update Expense"

The functional requirements associated with use case "Update Expense" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity,

understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommate shall be able to update any existing expense he shared with other roommates.
2. Roommate shall be able to add only floating point numbers with maximum of 2 decimal points for the amount field while modifying the expense.
3. Roommate shall be able to mark any expense as settled or unsettled.
4. Roommate shall be able to return to the list of expenses without modifying by selecting cancel before submitting changes.
5. Roommate shall be able to view the updated expense in the list of expenses on dashboard.

3.18 Use Case 18: Delete Expense

3.18.1 Use Case "Delete Expense" Documentation

Use Case Documentation Template	
Use Case ID:	18
Use Case Name:	Delete Expense: This use case allows roommates that are associated with an apartment to delete any expense they added.
Actors:	
Roommates	
Preconditions:	
A valid user (i.e. a roommate) has logged into the system and is associated with the apartment for which the expense is being deleted.	
Flow of Events of the Primary Scenario:	
1. The use case starts when the roommate selects an expense and selects 'Delete'. 2. The system prompts the roommate asking for confirmation. 3. The user chooses delete. 4. System deletes the expense. 5. Use case ends.	
Flow of Events of the Alternative Scenarios:	
Roommate selects 'No' when prompted for confirmation, in which case, system returns to the list without deleting the expense.	
Flow of Events of the Exception Scenarios:	
None	
Extension Points:	
None	
"Used" Use Cases:	
Details associated with the expense, which is being deleted.	
Postconditions:	
The expense is deleted from the system.	

3.18.2 Functional Requirements Associated with Use Case "Delete Expense"

The functional requirements associated with use case "Delete Expense" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity,

understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommates shall be able to delete any expense that was added by them.
2. Roommates shall be given a chance to confirm the deletion intent.
3. Roommates shall be able to confirm that the expense has been deleted by viewing the list of existing expenses.

3.19 Use Case 19: Grocery List

3.19.1 Use Case "Grocery List" Documentation

Use Case Documentation Template	
Use Case ID:	19
Use Case Name:	Grocery List: This use case allows roommates to generate a grocery list based on the food choices they enter.
Actors:	
Roommates	
Preconditions:	
A valid user (i.e. a roommate) has logged into the system.	
Flow of Events of the Primary Scenario:	
1. The use case starts when the roommate selects Grocery List. 2. The roommate then enters one or more food choices, which will be cooked by the roommate through the week. 3. The roommate selects generate grocery list. 4. The system displays a list of grocery items that would be required to cook the food choices entered. 5. The system also displays an estimated cost for the generated grocery list and the use case ends.	
Flow of Events of the Alternative Scenarios:	
None as of now.	
Flow of Events of the Exception Scenarios:	
1. If the roommate selects Cancel at any time before selecting Generate grocery list.	
Extension Points:	
None as of now.	
"Used" Use Cases:	
None as of now.	
Postconditions:	
A grocery list is displayed to the roommate.	

3.19.2 Functional Requirements Associated with Use Case "Grocery List"

The functional requirements associated with use case "Grocery List" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. All roommates shall be able to generate a grocery list based on the food choices they have entered in the system.
2. The system shall generate an approximate cost associated with a generated grocery list.

3.20 Use Case 20: Reward Points

3.20.1 Use Case "Reward Points" Documentation

Use Case Documentation Template	
Use Case ID:	20
Use Case Name:	Reward Points - This use case describes the process by which roommates keep a number of reward points based on the tasks completed by them.
Actors:	
Roommates	
Preconditions:	
Roommate must have valid account. A roommate must have completed or missed tasks' deadlines.	
Flow of Events of the Primary Scenario:	
1. Roommate selects "View Profile" 2. Roommate selects "Reward Points" 3. System displays the number of reward points the roommate earned.	
Flow of Events of the Alternative Scenarios:	
None	
Flow of Events of the Exception Scenarios:	
Failing to successfully retrieve the amount of reward points the roommate holds.	
Extension Points:	
N/A	
"Used" Use Cases:	
N/A	
Postconditions:	
Roommates must be able to see changes in their reward points after a task has been completed in time or missed deadline.	

3.20.2 Functional Requirements Associated with Use Case "Reward Points"

The functional requirements associated with use case "Name" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommates shall receive N points for completing a task.

2. Roommates shall be penalized N points for not completing a task within the deadline.
3. Roommates shall not receive or be penalized any points if a task is deleted.
4. Roommates shall be able to retrieve the reward points of other roommates.

3.21 Use Case 21: View Roommate Profiles

3.21.1 Use Case "View Roommate Profiles" Documentation

Use Case Documentation Template	
Use Case ID:	21
Use Case Name:	View Roommate Profiles: This use case allows roommates to view the profile of any other roommate belonging to the same apartment. This would allow users to search for new roommates and also view their points, which are accumulated by doing tasks before deadlines.
Actors:	
Roommates	
Preconditions:	
A valid user (i.e. a roommate) has logged into the system	
Flow of Events of the Primary Scenario:	
1. The use case starts when the roommate logs in and dashboard is seen. 2. The system displays on the left – the list of all the roommates belonging to the apartment. 3. The system displays detailed information about each roommate like the user's full name, email and accumulated points. 4. Use case ends.	
Flow of Events of the Alternative Scenarios:	
None as of now.	
Flow of Events of the Exception Scenarios:	
None	
Extension Points:	
None as of now.	
"Used" Use Cases:	
1. None as of now.	
Postconditions:	
1. None as of now.	

3.21.2 Functional Requirements Associated with Use Case "View Roommate Profiles"

The functional requirements associated with use case "View Roommate Profiles" are listed below. Quality of each requirement specification is evaluated with respect to the following quality characteristics and found to be acceptable: Accuracy (verity, validity), Clarity (unambiguity, understandability), Completeness, Consistency, Feasibility, Modifiability, Stability, Testability, and Traceability.

1. Roommate shall be able to see the list of roommates belonging to the apartment.
2. Roommate shall be able to see their full details – name, email ID and reward points accumulated by each of them.
3. Roommates shall be able to view their profile pictures along with other details.

3.22 Non-Functional Requirements

1. The ApartMates cloud software application shall be accessed over HTTP request response based connections.
2. ApartMates shall be available at no cost for anyone to use.
3. ApartMates shall be accessed through the most popular or well known web browsers.
4. ApartMates shall be accessed through both computers and mobile devices.
5. ApartMates shall be deployed on a dedicated server with the following specifications:

    i. Dell PowerEdge T320, 64GB RAM, 2TB hard disk
    ii. Server URL: `venus.cs.vt.edu`
    iii. Located in McBryde 116 Software Engineering Lab, Virginia Tech.

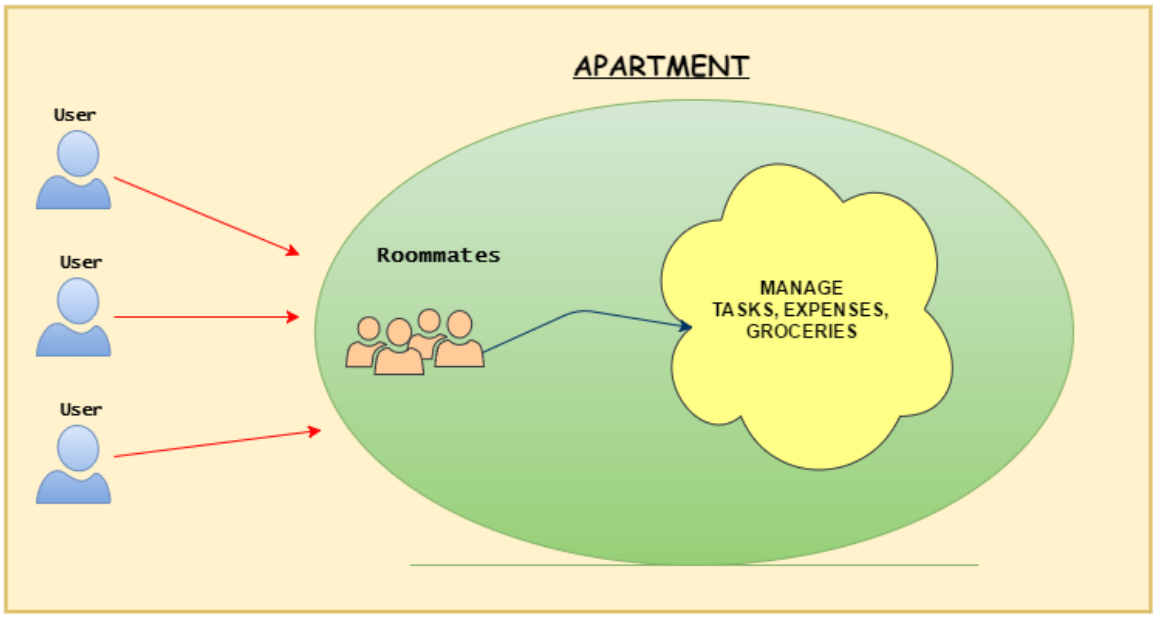
6. The ApartMates Web Service (ApartMatesWS) shall be available as a REST API.
7. ApartMates shall interoperate with other API over HTTP request response based connections.
8. ApartMates shall have high availability (available 24 hours a day, 365 days a year) except for downtime during pre-informed maintenance hours.
9. All components of ApartMates and ApartMatesWS shall be quantifiably testable (PASS/FAIL) using test cases.
10. ApartMates and ApartMatesWS source code repositories and deployed products shall be stable and maintainable.
11. ApartMates and ApartMatesWS shall be fault tolerant and be able to handle large requests using graceful degradation principle.
12. ApartMates and ApartMatesWS shall have minimal impact on the environment.
13. ApartMates shall be extendable with more features in the future.
14. ApartMatesWS shall be extendable so as to support richer features in the future.
15. ApartMates and ApartMatesWS shall be vertically scalable.
16. ApartMates and ApartMatesWS shall be deployed in a secured "authorized access only" facility so as to prevent any harm to data assets.

4. ARCHITECTURE SPECIFICATION

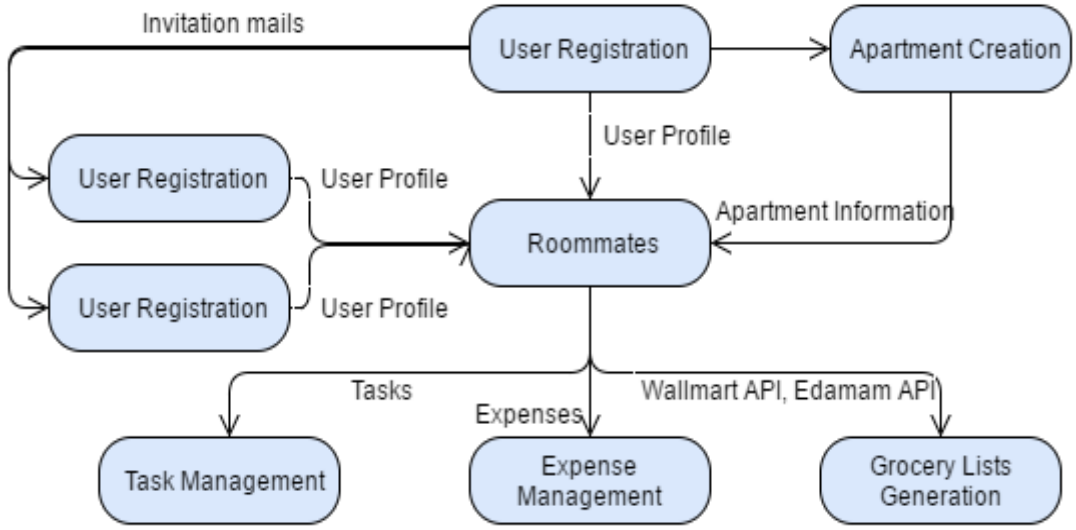
The network-centric and cloud-based architecture of our software-based solution system 'ApartMates' has been presented using the following DoDAF models.

1. **OV-1**: High-Level Operational Concept Graphic
2. **OV-2**: Operational Resource Flow Description
3. **OV-5b**: Operational Activity Model
4. **SV-1**: Systems Interface Description
5. **SV-2**: Systems Resource Flow Description
6. **SV-4**: Systems Functionality Description
7. **SvcV-1**: Services Context Description
8. **SvcV-2**: Services Resource Flow Description
9. **SvcV-4**: Services Functionality Description

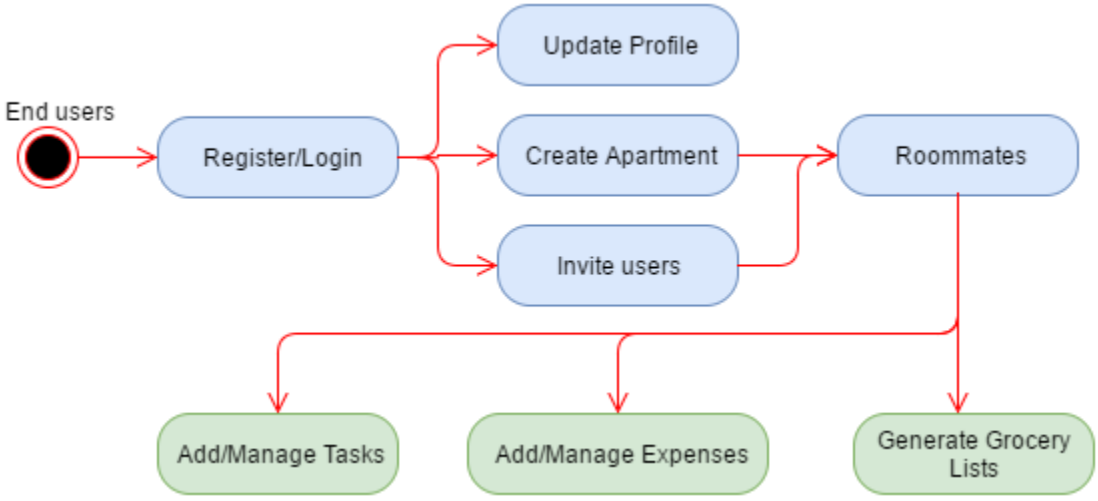
4.1 OV-1: High-Level Operational Concept Graphic



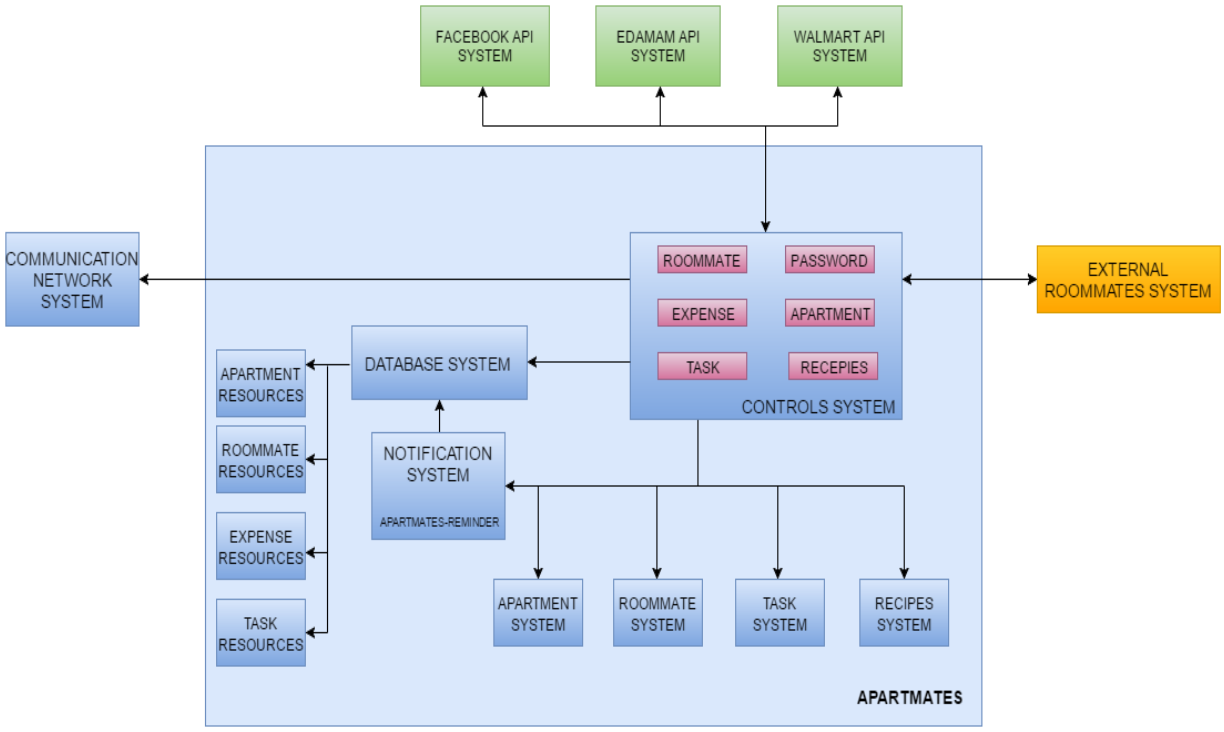
4.2 OV-2: Operational Resource Flow Description



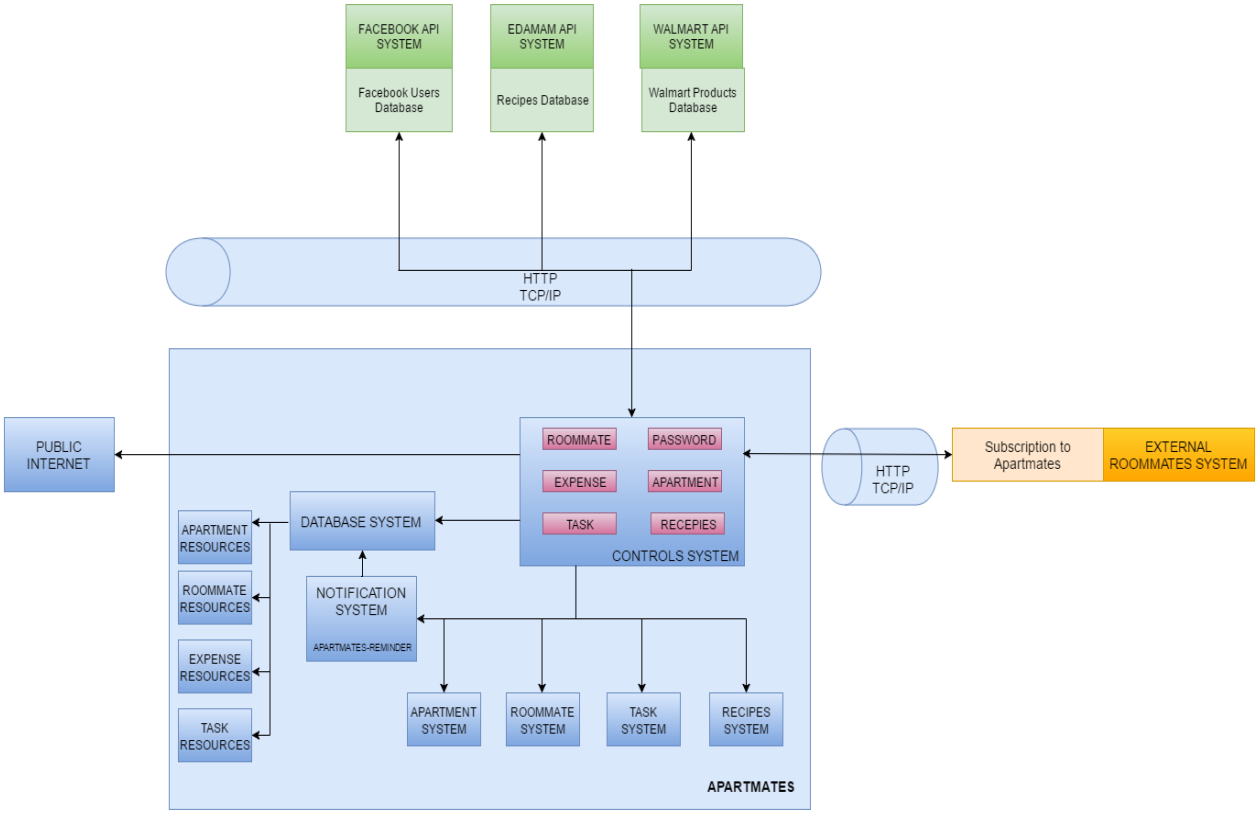
4.3 OV-5b: Operational Activity Model



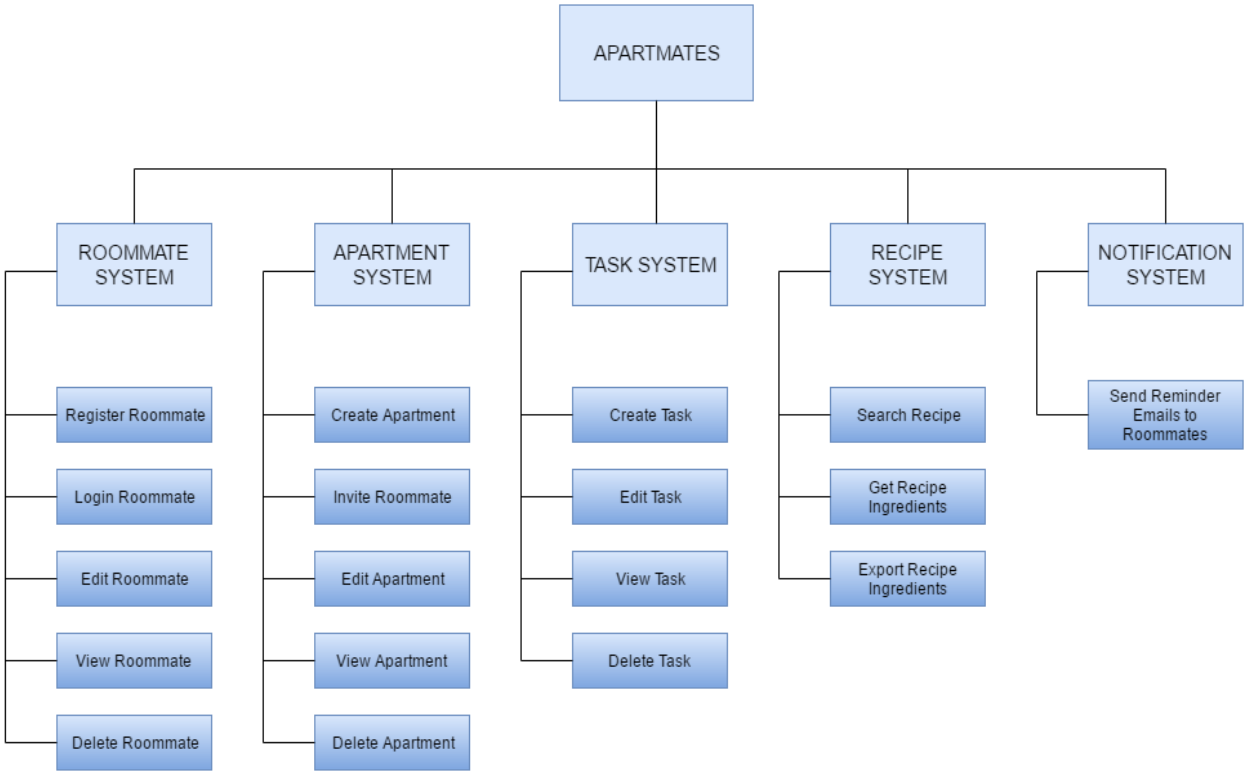
4.4 SV-1: Systems Interface Description



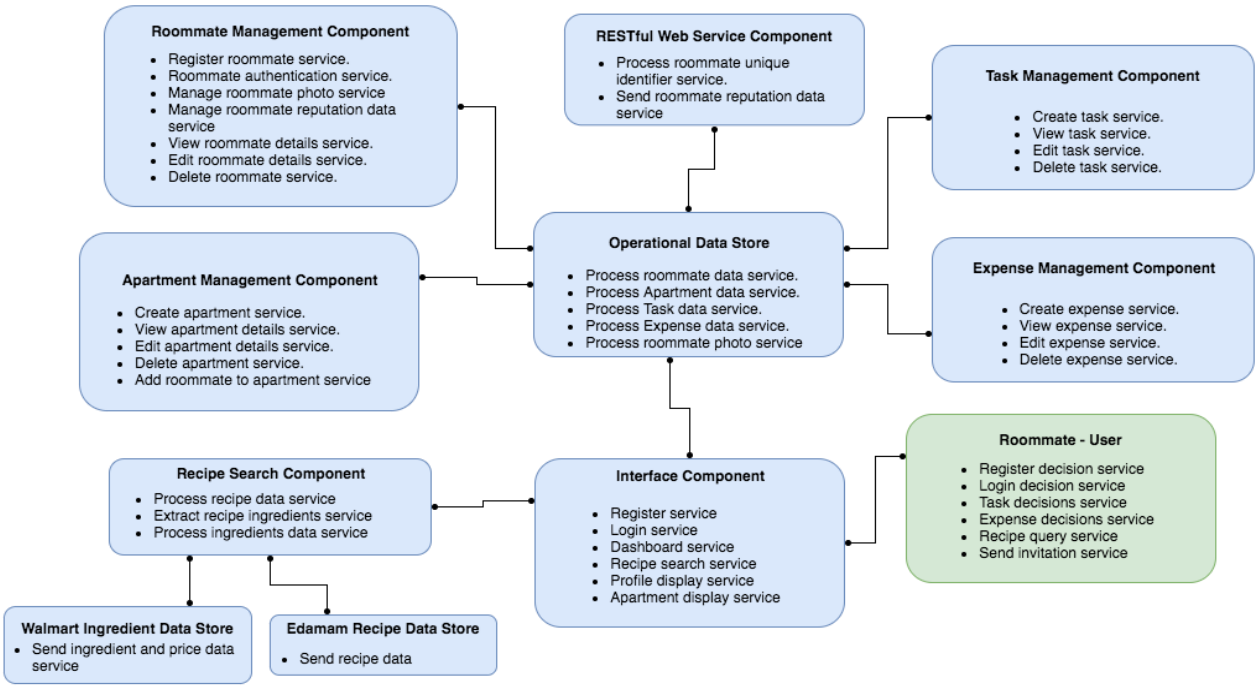
4.5 SV-2: Systems Resource Flow Description



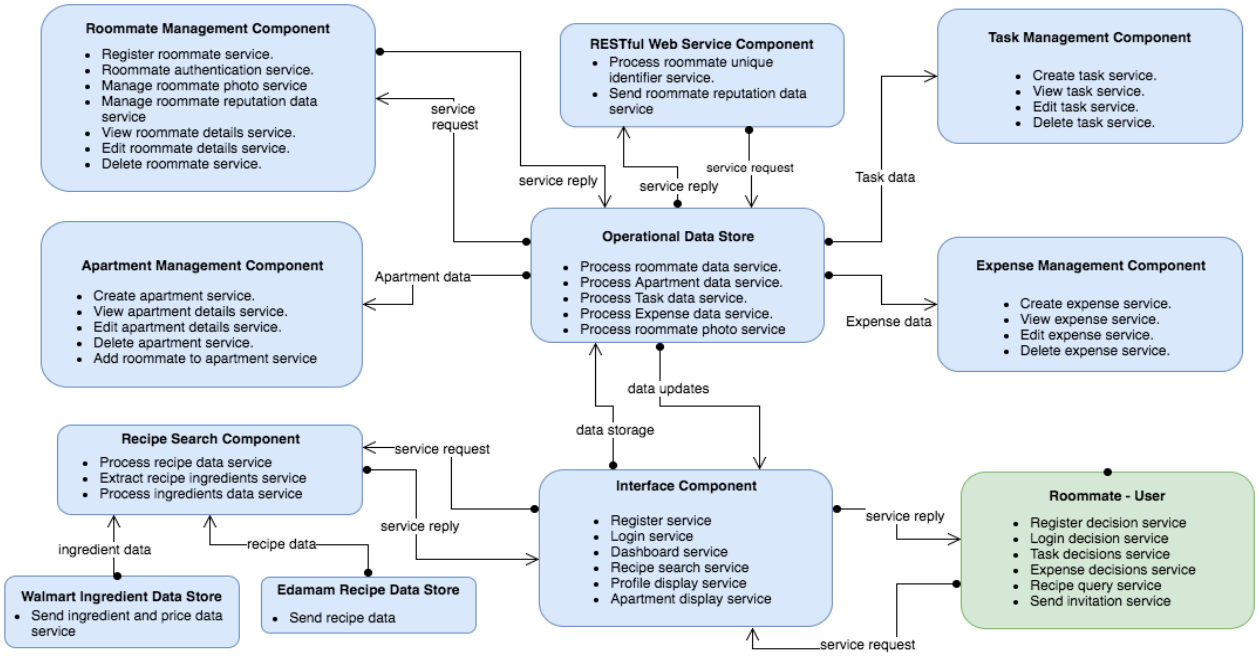
4.6 SV-4: Systems Functionality Description



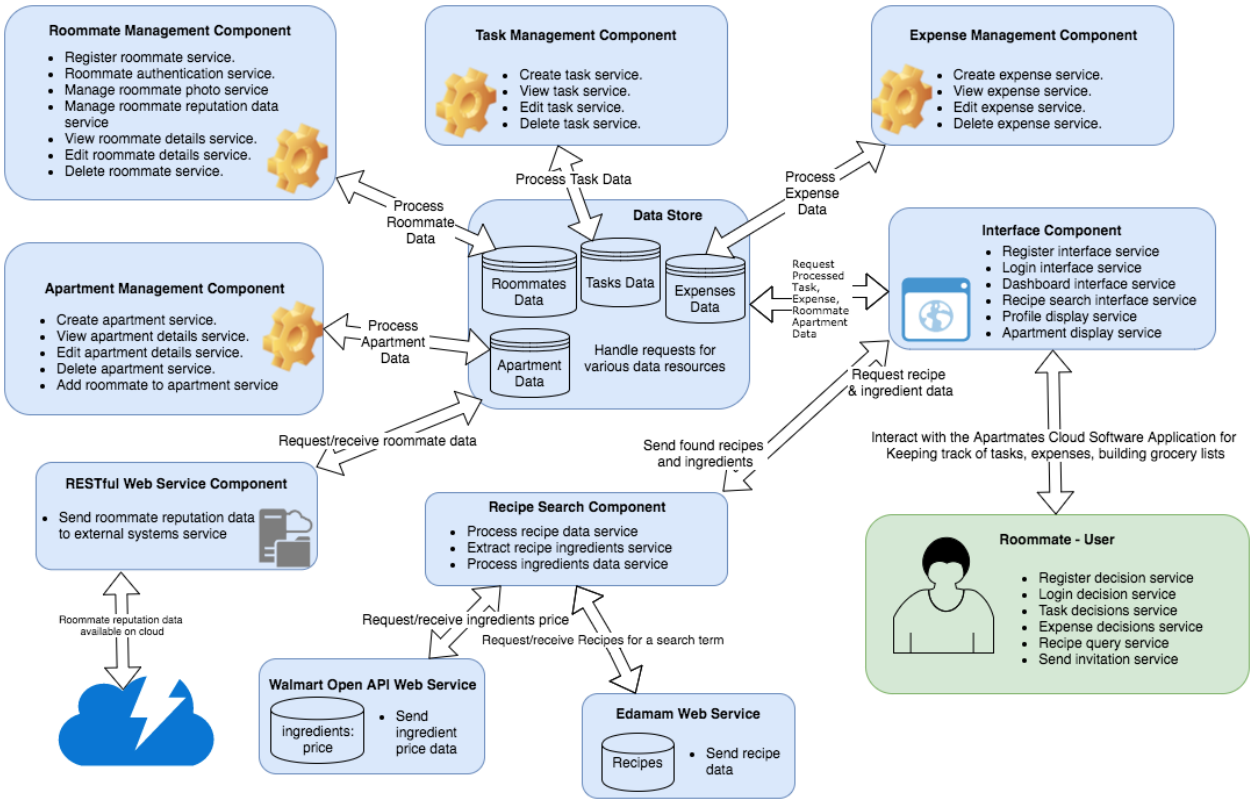
4.7 SvcV-1: Services Context Description



4.8 SvcV-2: Services Resource Flow Description



4.9 SvcV-4: Services Functionality Description

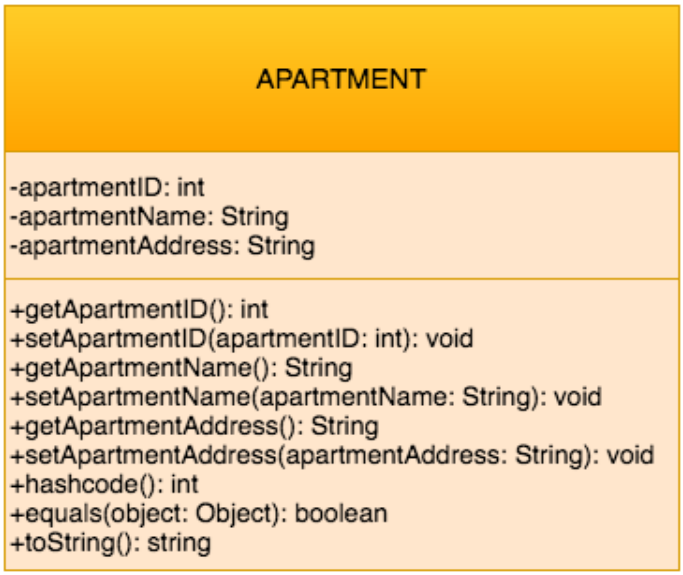


5. DESIGN SPECIFICATION

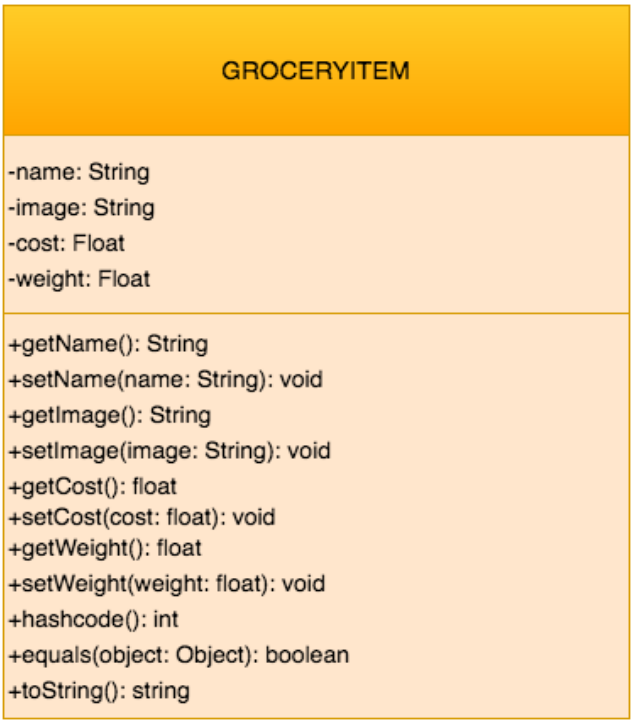
The following UML diagrams represent our object-oriented software design.

5.1 Class Diagrams

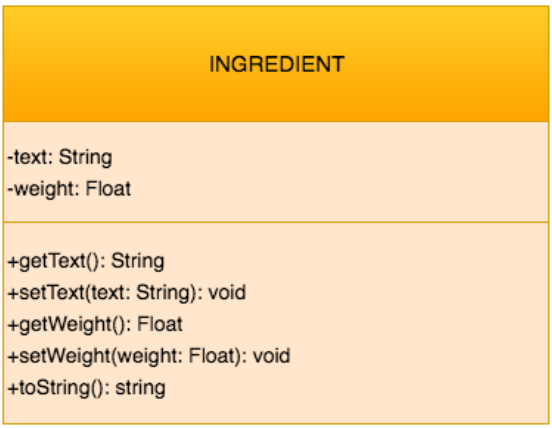
5.1.1 Apartment Class



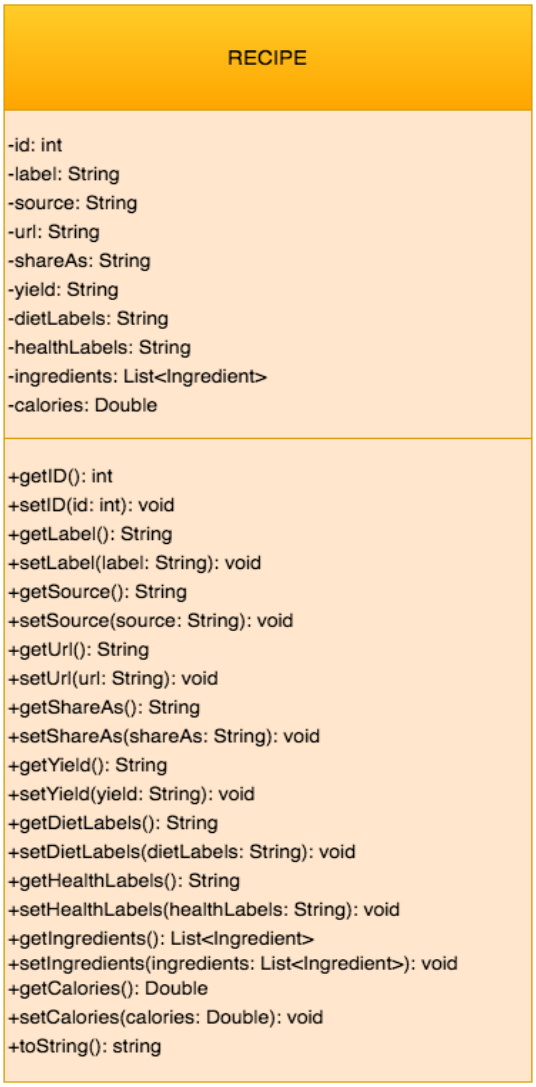
5.1.2 GroceryItem Class



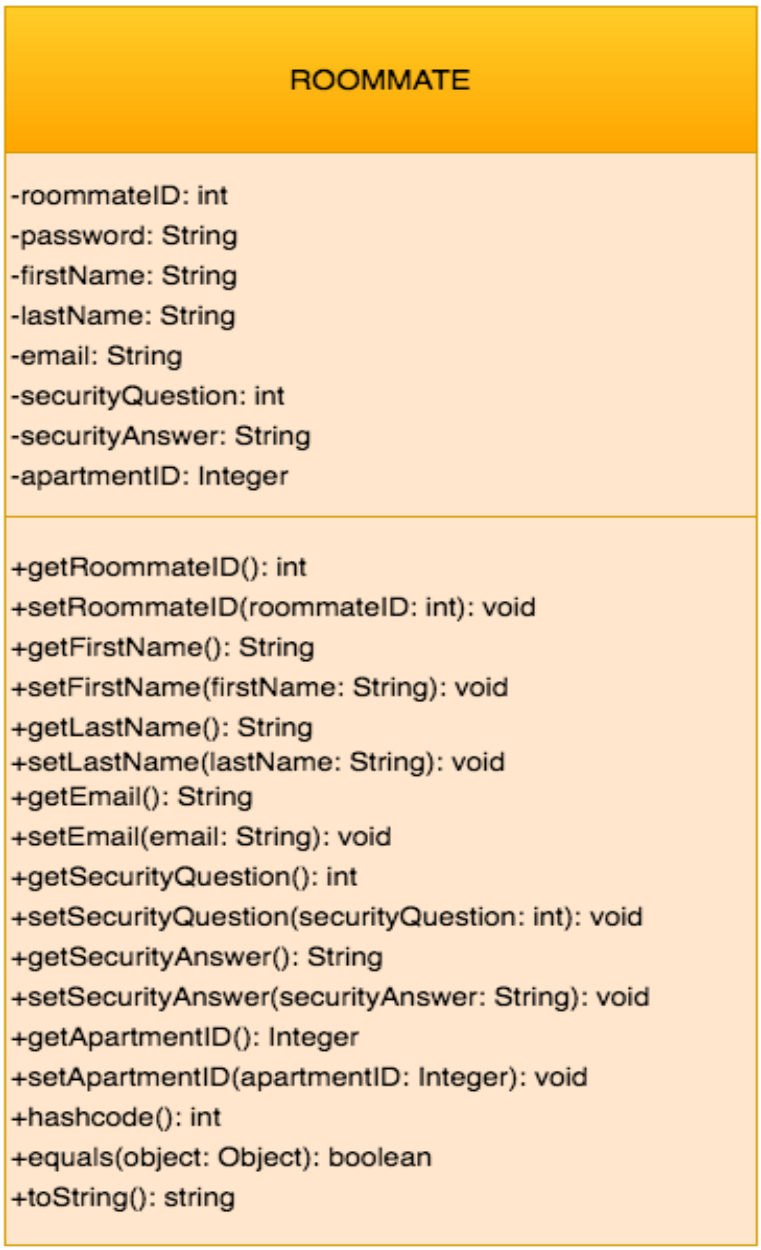
5.1.3 Ingredient Class



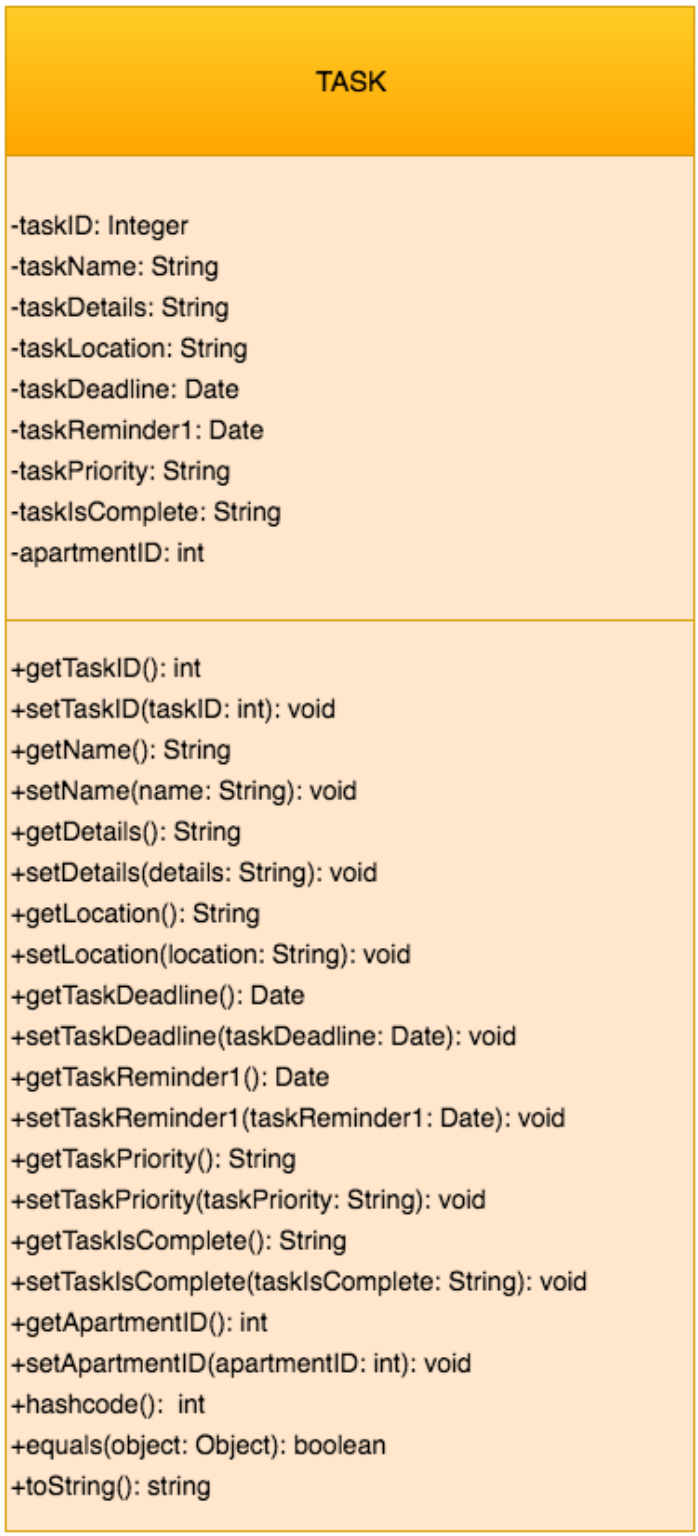
5.1.4 Recipe Class



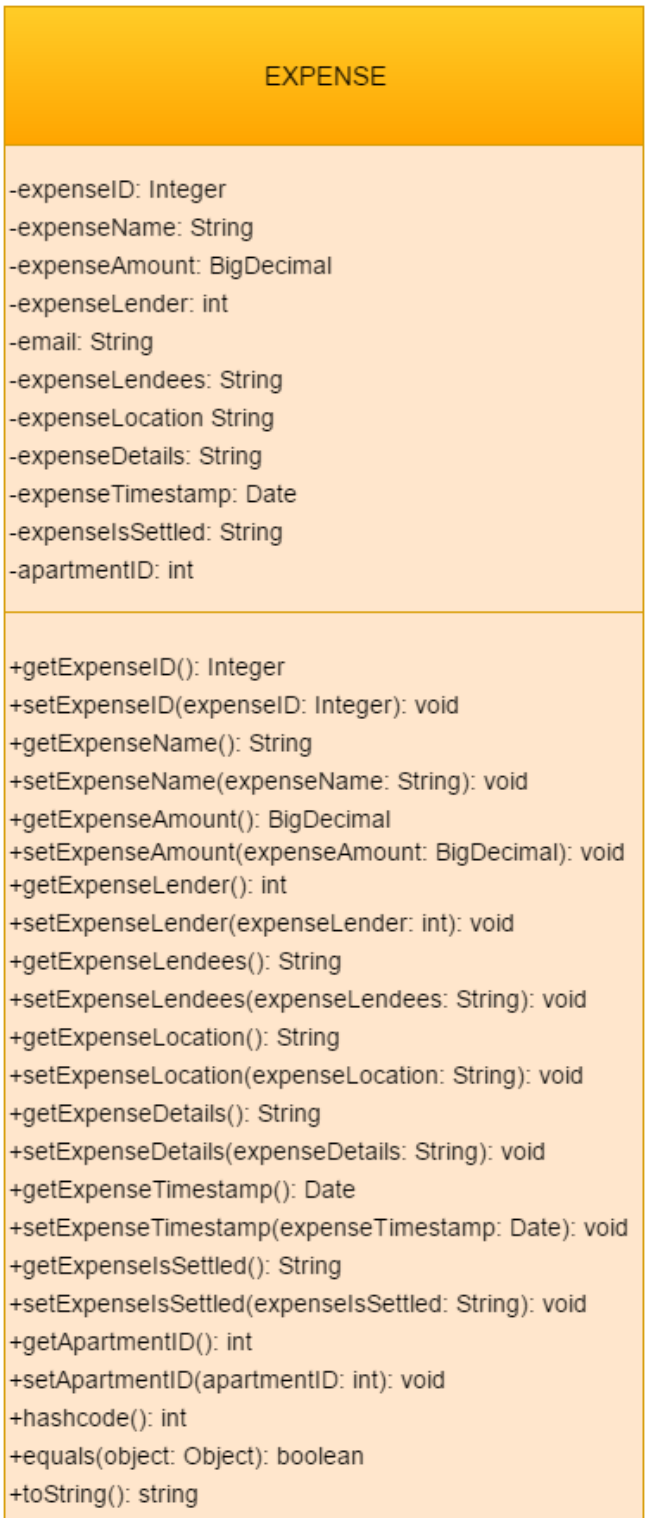
5.1.5 Roommate Class



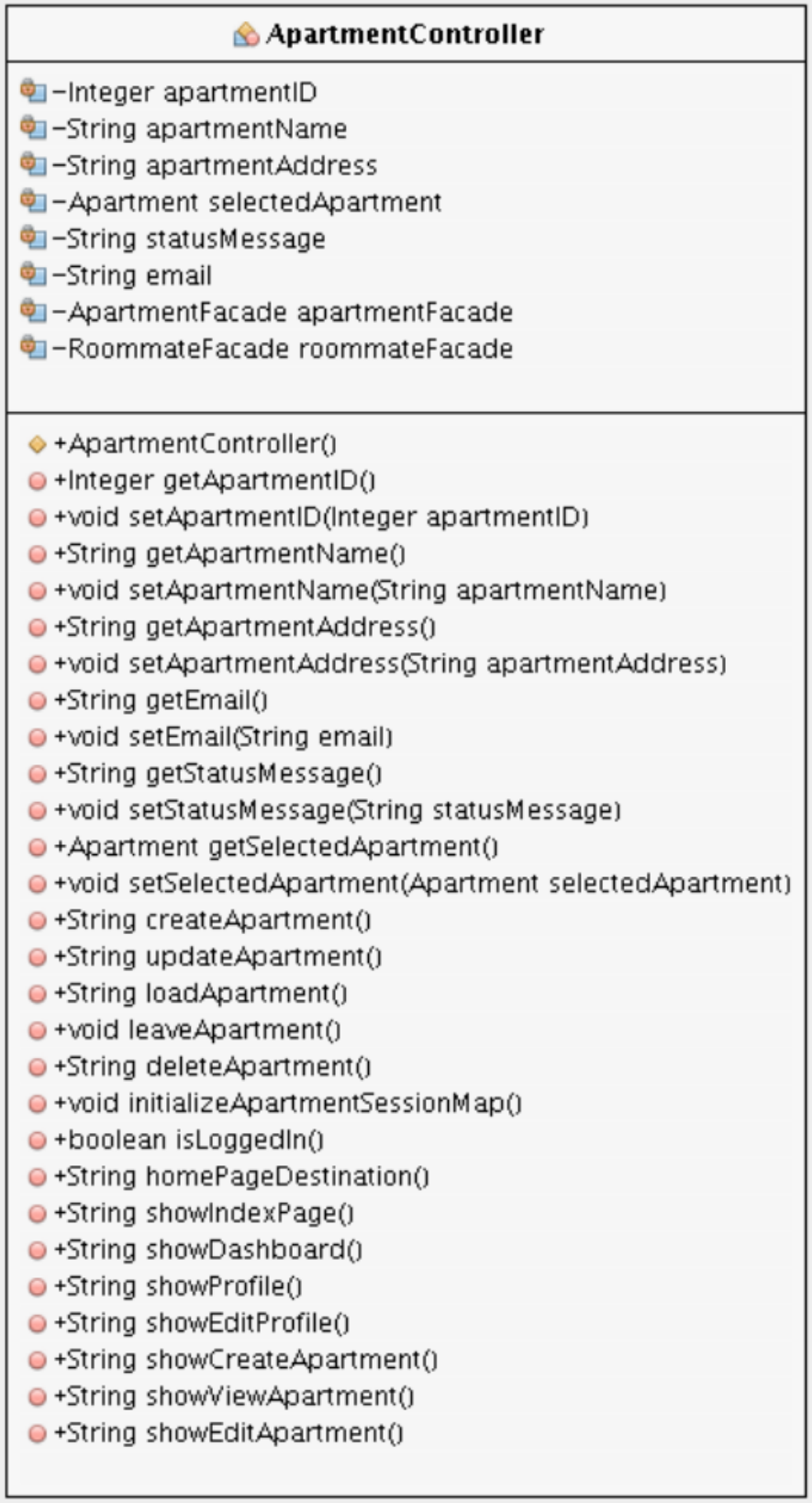
5.1.6 Task Class



5.1.7 Expense Class

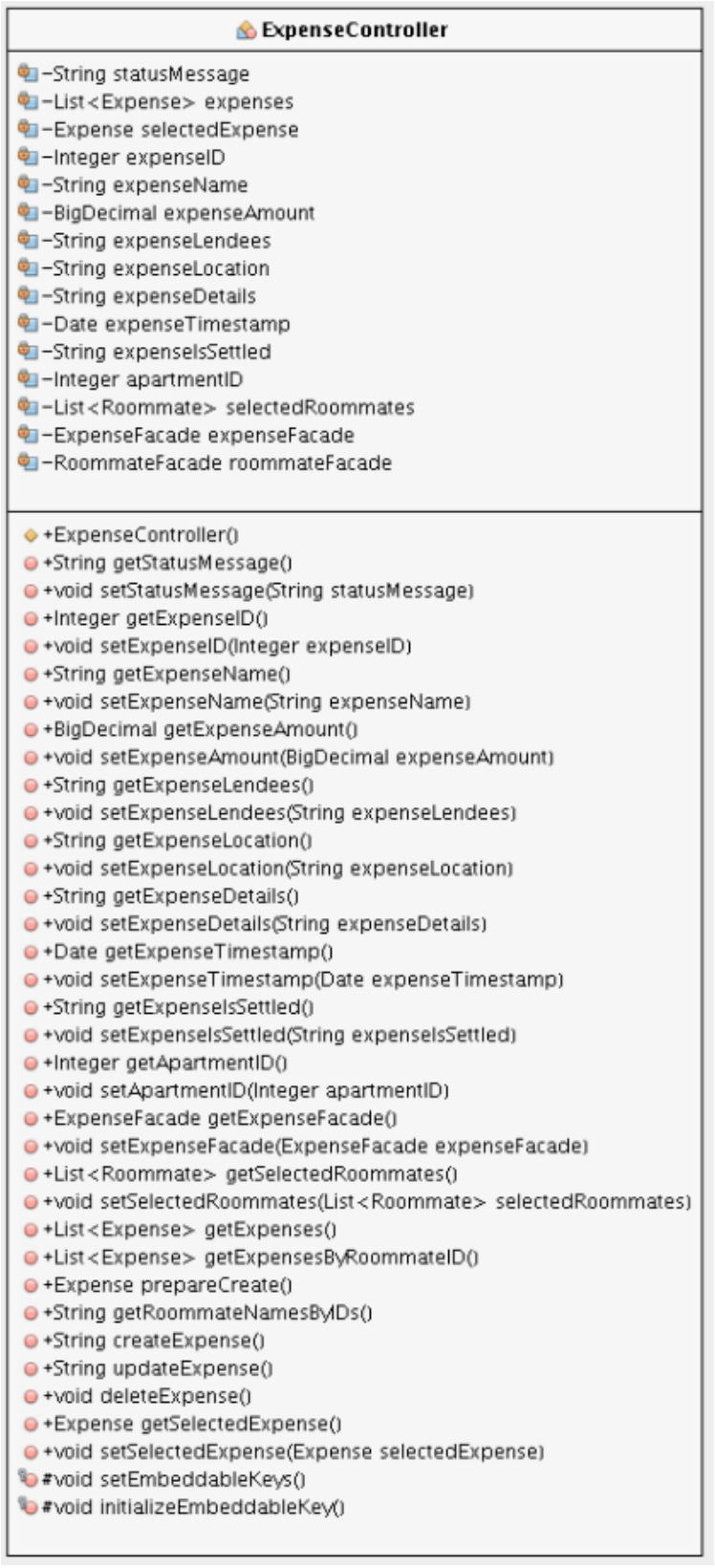


5.1.8 ApartmentController Class



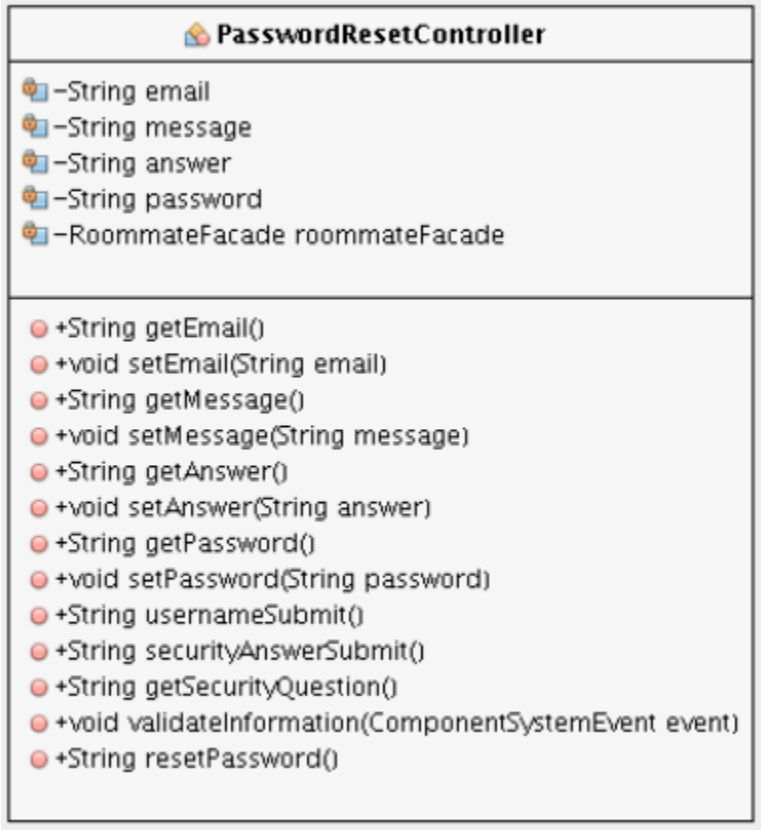
5.1.9 TaskController Class

5.1.10 ExpenseController Class

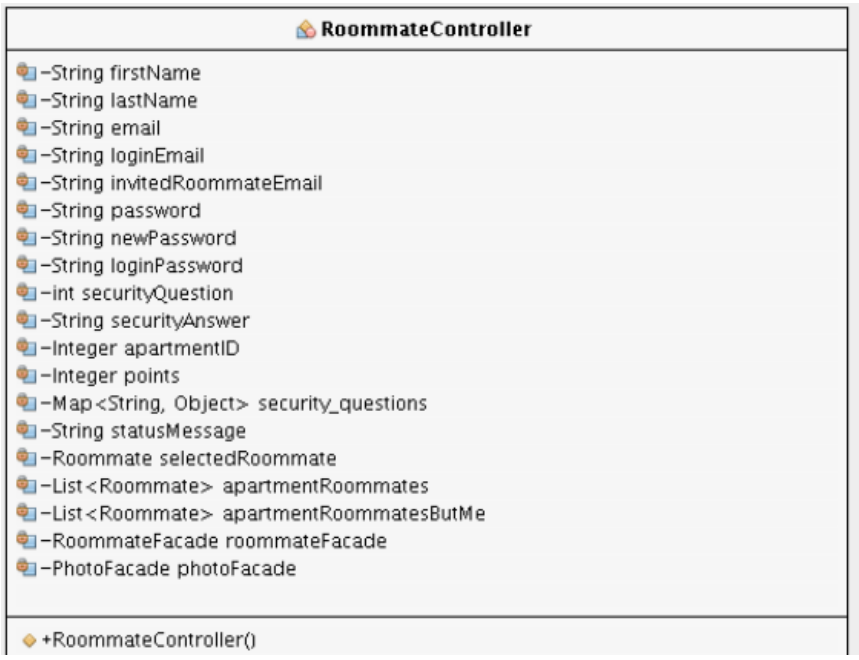


5.1.11 RecepiesController

5.1.12 *PasswordResetController*

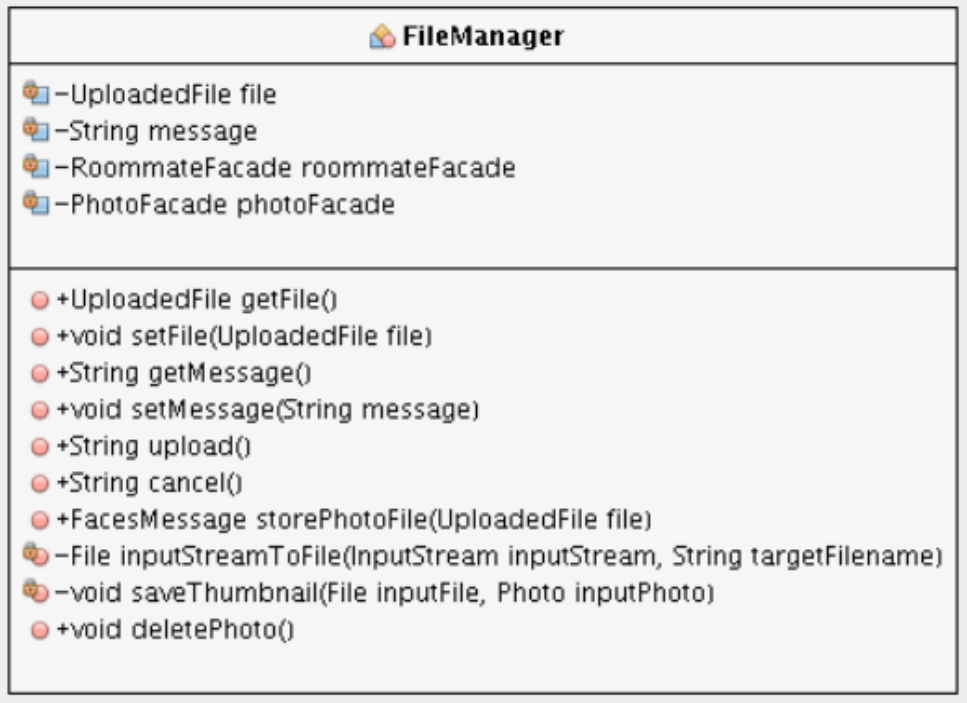


5.1.13 RoommateController



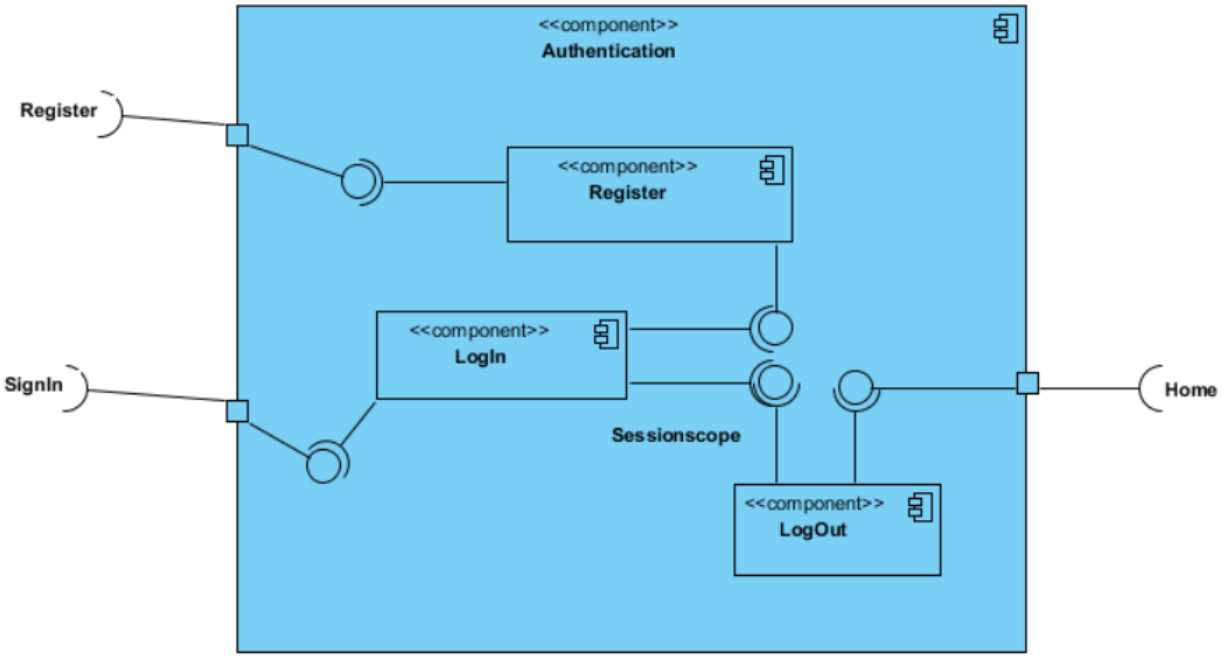

+RoommateController()
+String getFirstName()
+void setFirstName(String firstName)
+String getLastName()
+void setLastName(String lastName)
+String getEmail()
+void setEmail(String email)
+String getInvitedRoommateEmail()
+void setInvitedRoommateEmail(String invitedRoommateEmail)
+String getLoginEmail()
+void setLoginEmail(String loginEmail)
+int getSecurityQuestion()
+void setSecurityQuestion(int securityQuestion)
+String getSecurityAnswer()
+void setSecurityAnswer(String securityAnswer)
+String getPassword()
+void setPassword(String password)
+String getNewPassword()
+void setNewPassword(String newPassword)
+String getLoginPassword()
+void setLoginPassword(String loginPassword)
+Integer getApartmentID()
+void setApartmentID(Integer apartmentID)
+Integer getPoints()
+void setPoints(Integer points)
+List<Roommate> getApartmentRoommatesButMe()
+void setApartmentRoommatesButMe(List<Roommate> apartmentRoommatesButMe)
+Map<String, Object> getSecurity_questions()
+String getStatusMessage()
+void setStatusMessage(String statusMessage)
+Roommate getSelectedRoommate()
+void setSelectedRoommate(Roommate selectedRoommate)
+List<Roommate> getApartmentRoommates()
+void setApartmentRoommates(List<Roommate> apartmentRoommates)
+String photoStorageDirectoryName()
+String roommatePhoto()
+String roommatePhoto(Roommate r)
+String createRoommate()
+String loginRoommate()
+String resetPassword()
+String updateRoommate()
+String deleteProfile()
+void deletePhoto(int roommateId)
+boolean isLoggedIn()
+boolean hasAnApartment()
+void validateInformation(ComponentSystemEvent event)
+void validatePasswordChange(ComponentSystemEvent event)
+void validateRoommatePassword(ComponentSystemEvent event)
+void initializeRoommateSessionMap()
+void initializeRoommateSessionMap(Roommate roommate)
-boolean correctPasswordEntered(UIComponent components)
+String inviteRoommate()
+String logout()
+String homePageDestination()
+String showIndexPage()
+String showDashboard()
+String showProfile()
+String showEditProfile()
+String showCreateApartment()
+String showViewApartment()
+String showEditApartment()

5.1.14 FileManager

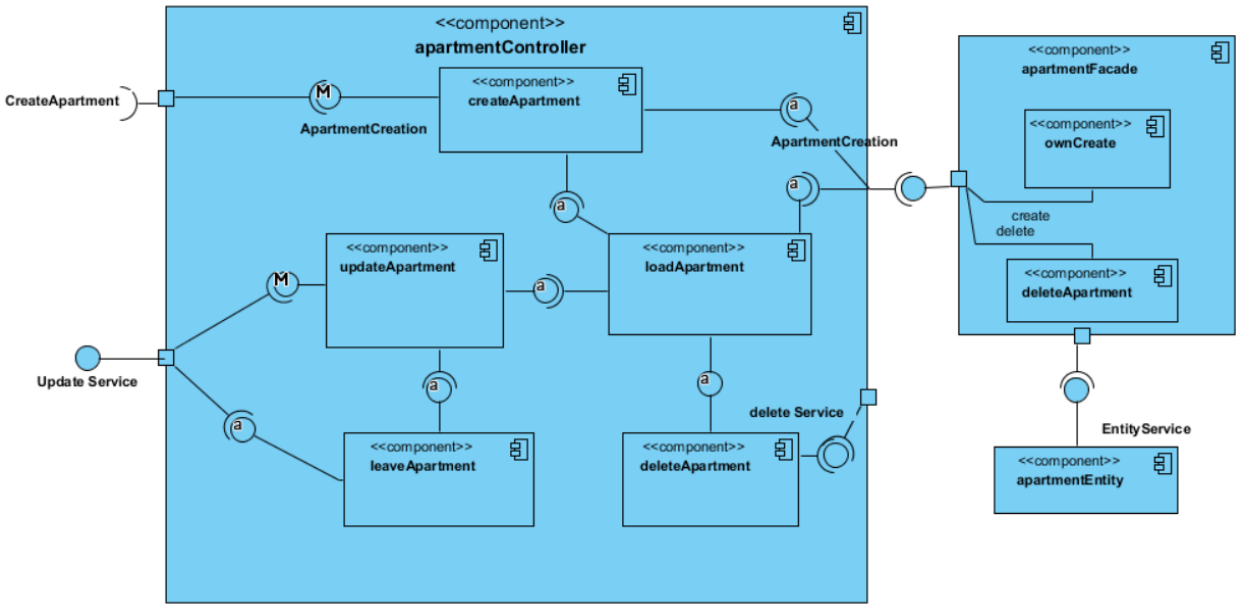


5.2 Component Diagrams

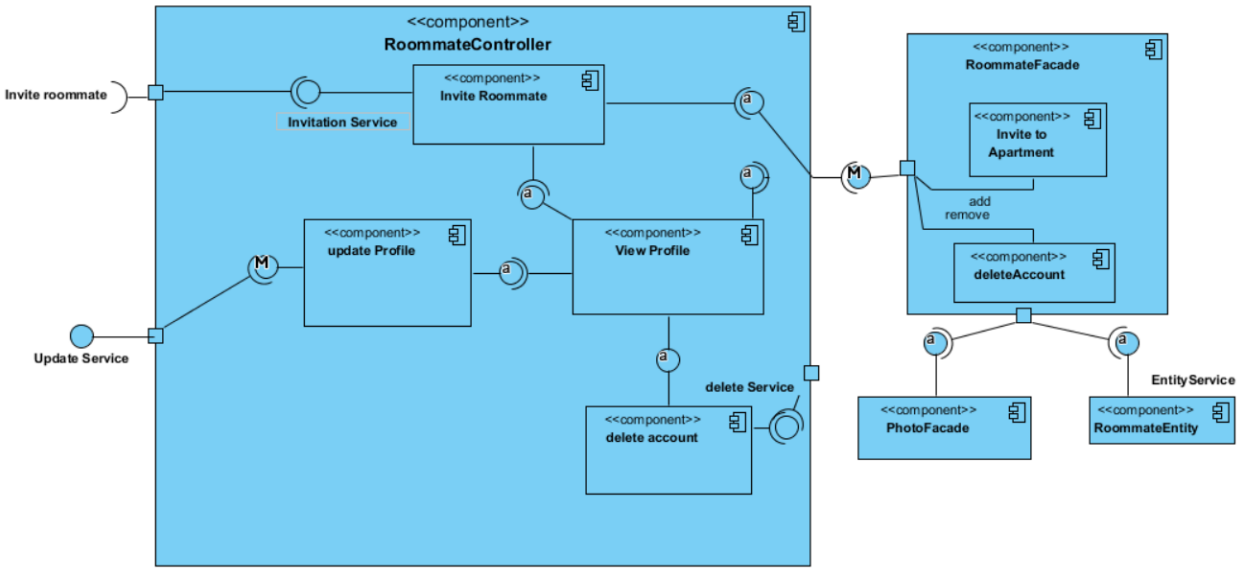
5.2.1 Authentication Component



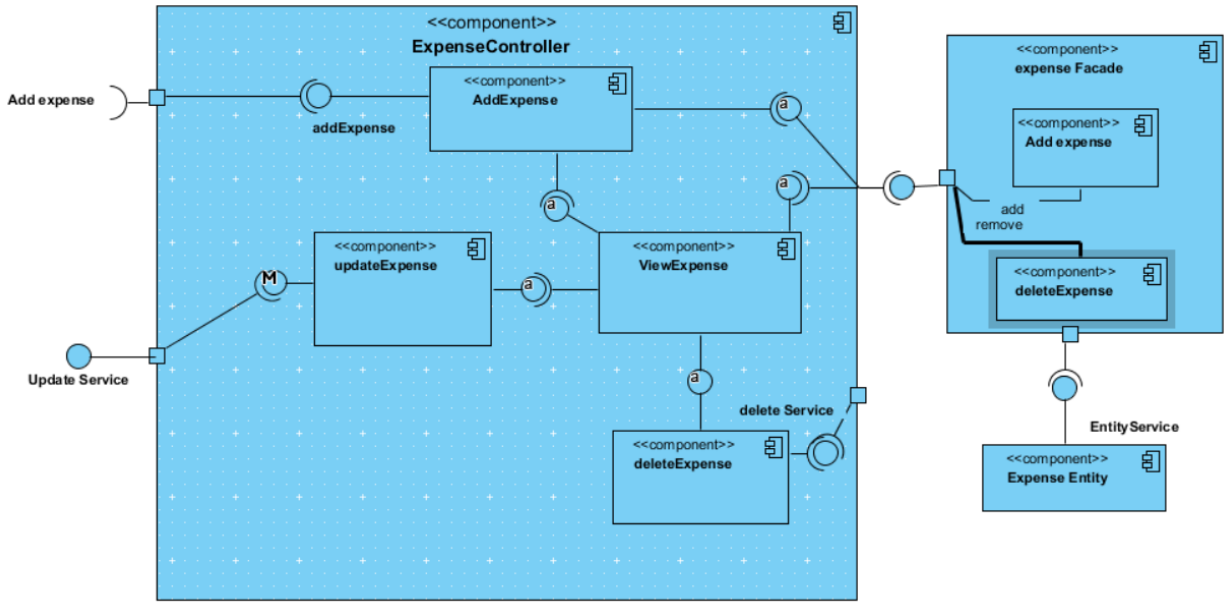
5.2.2 Apartment Component



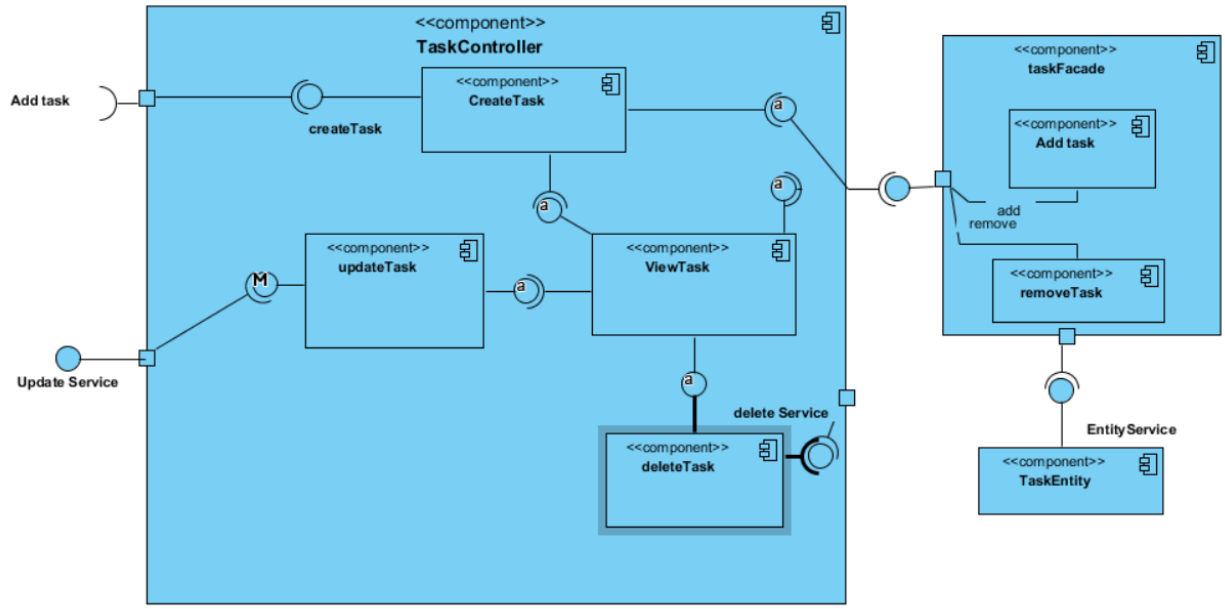
5.2.3 Roommate Component



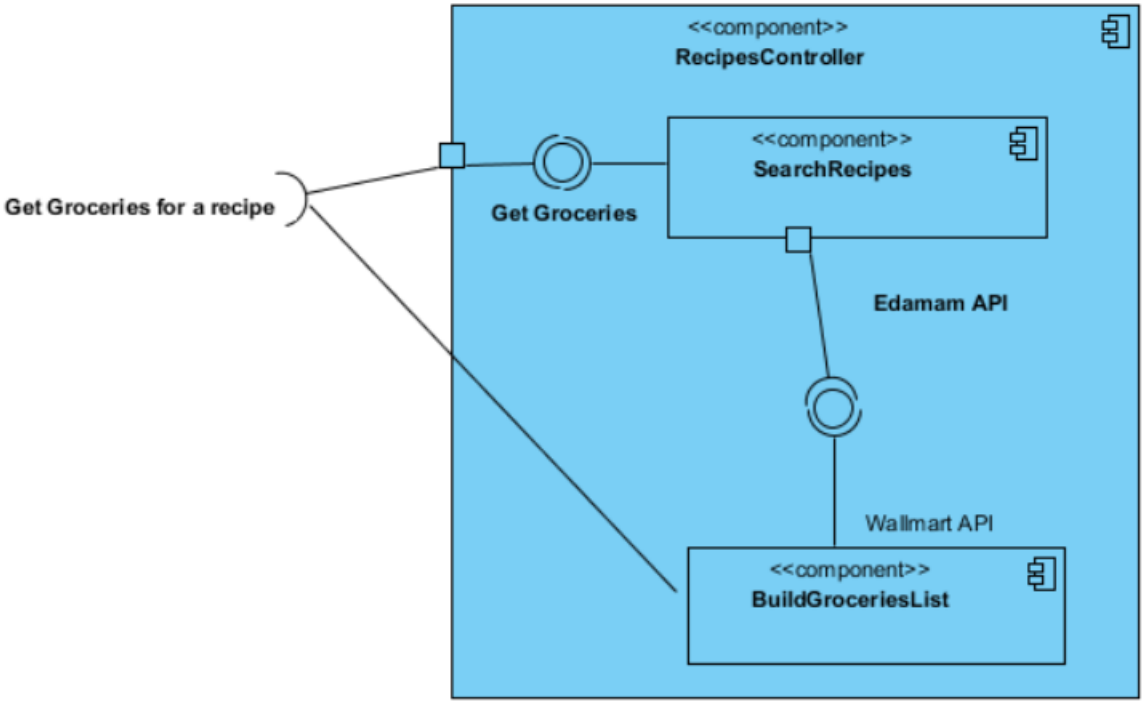
5.2.4 Expense Component



5.2.5 Task Component

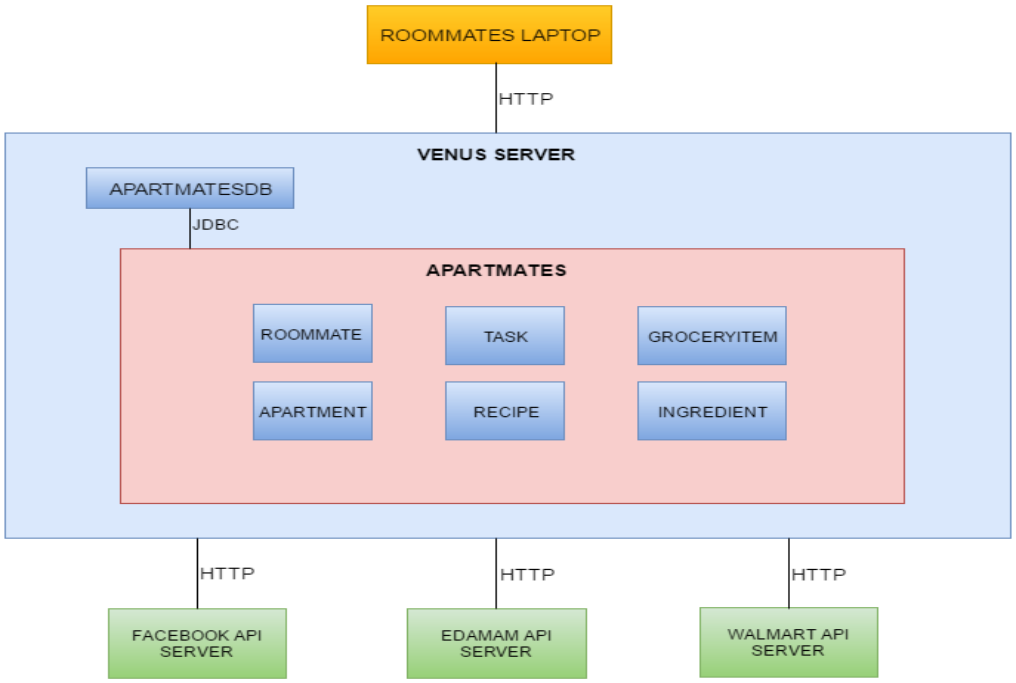


5.2.6 Groceries Component

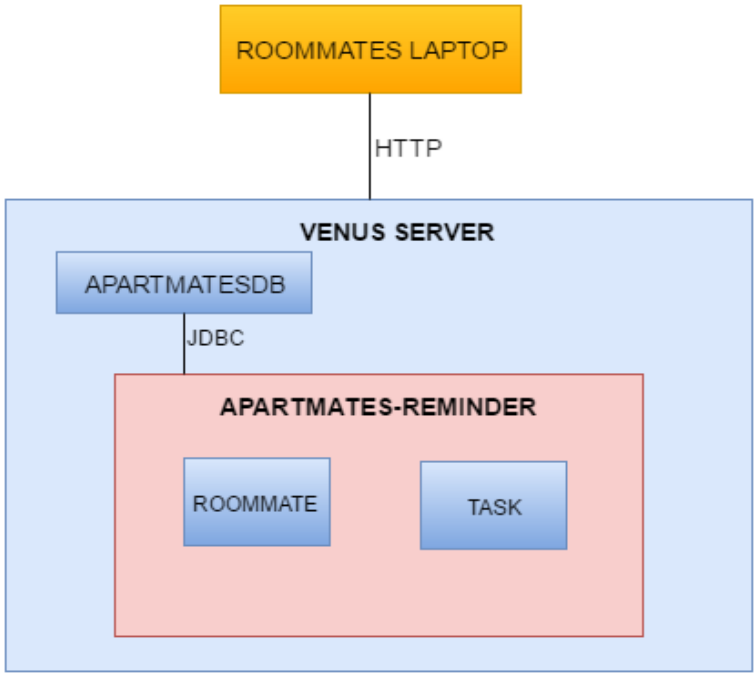


5.3 Deployment Diagrams

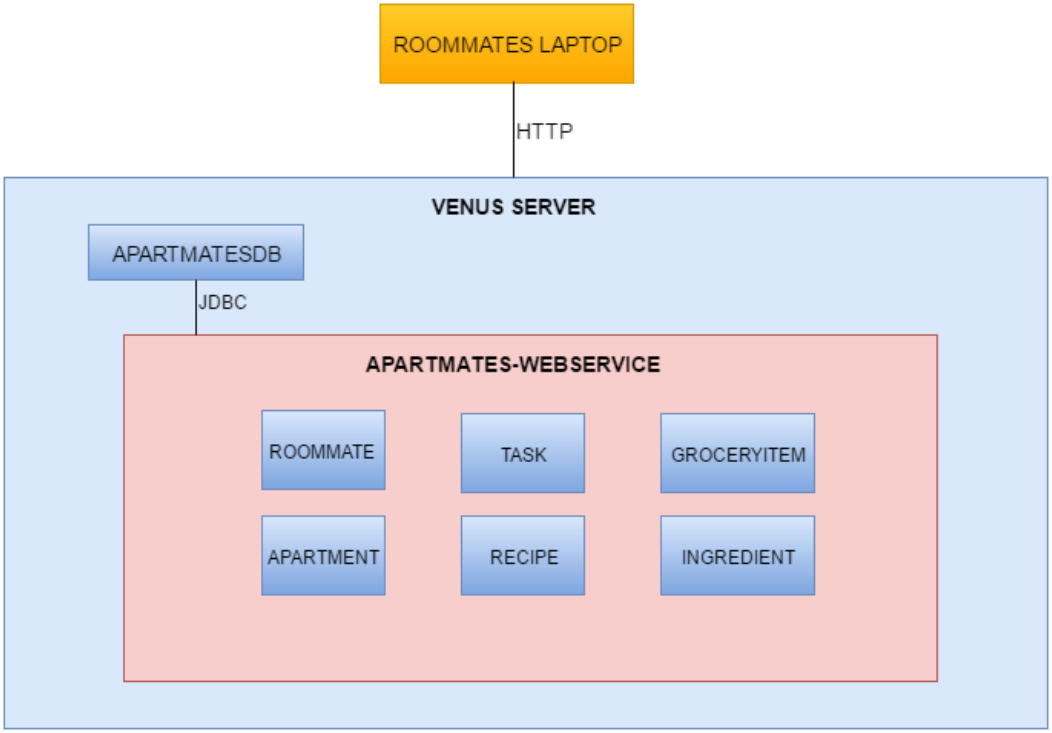
5.3.1 ApartMates Deployment Diagram



5.3.2 ApartMates-Reminder Deployment Diagram

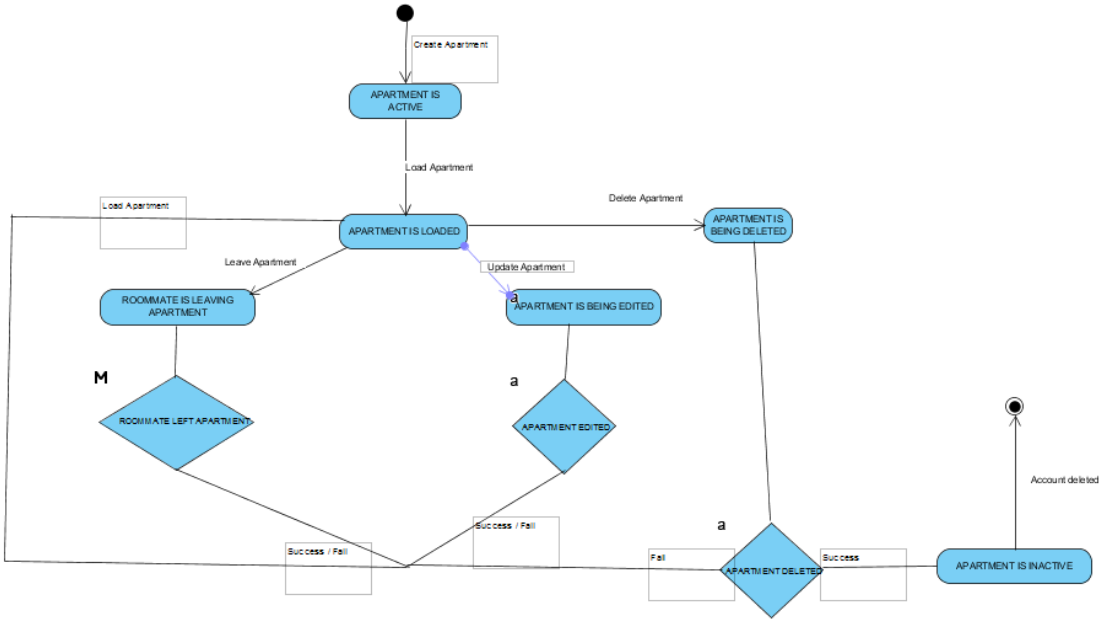


5.3.3 ApartMates-WS Deployment Diagram

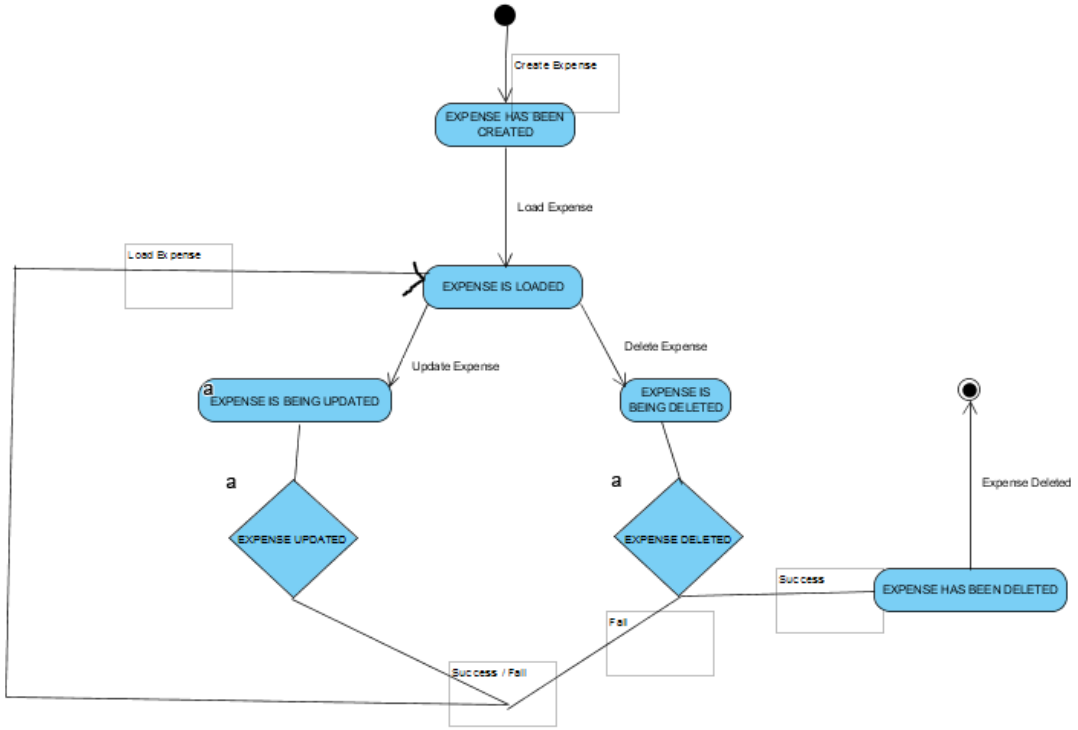


5.4 Statechart Diagrams

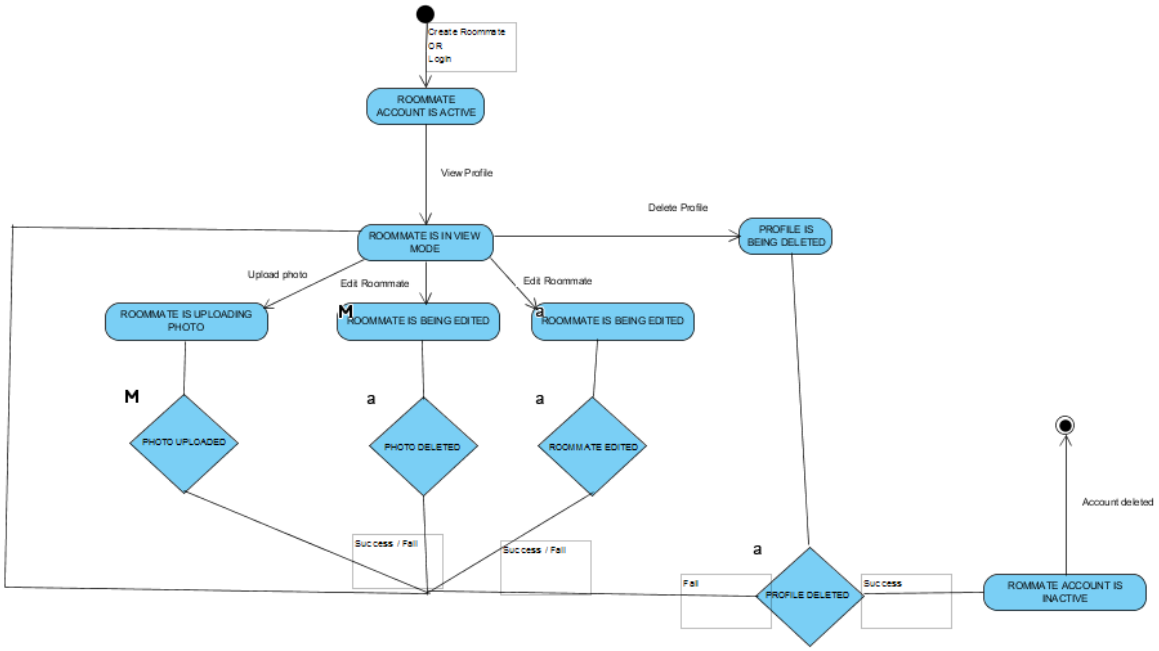
5.4.1 Apartment Statechart Diagram



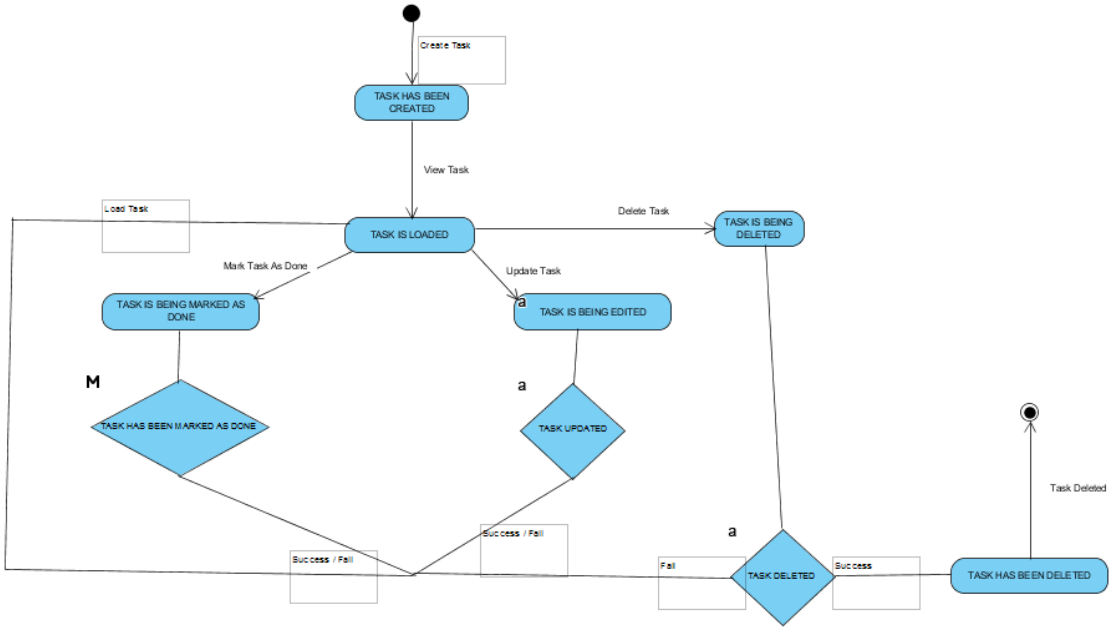
5.4.2 Expense Statechart Diagram



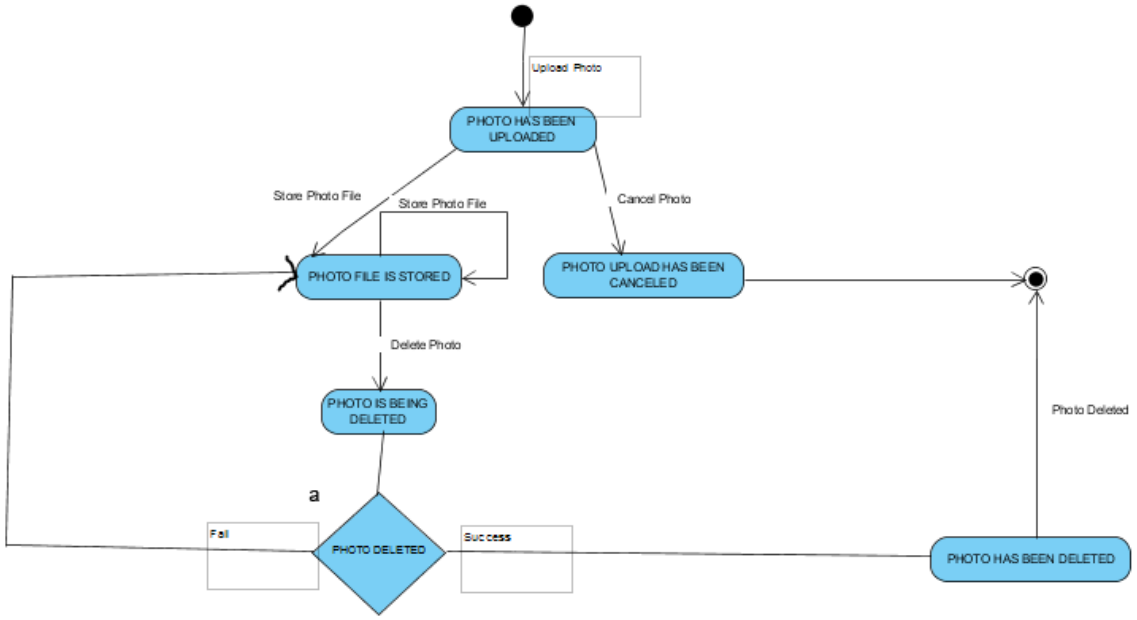
5.4.3 Roommate Statechart Diagram



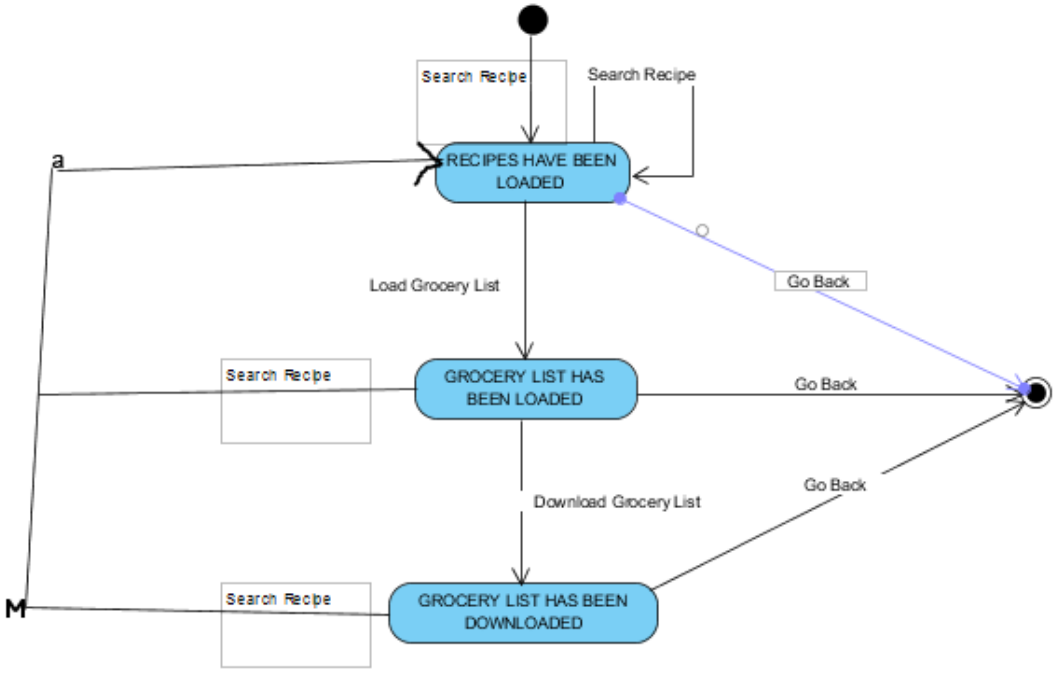
5.4.4 Task Statechart Diagram



5.4.5 Photo Statechart Diagram

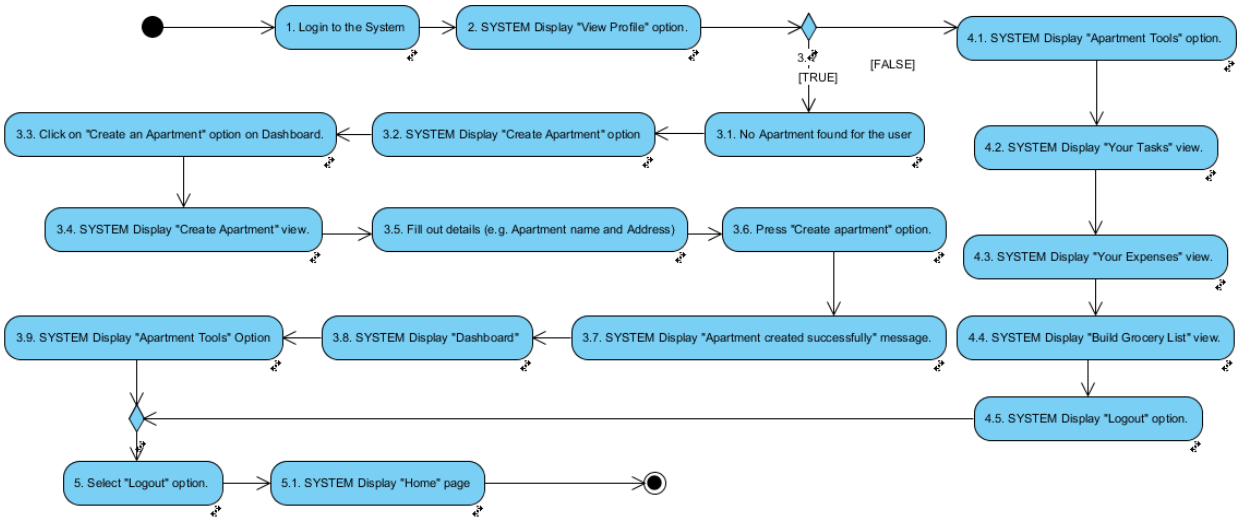


5.4.6 Recipe Statechart Diagram

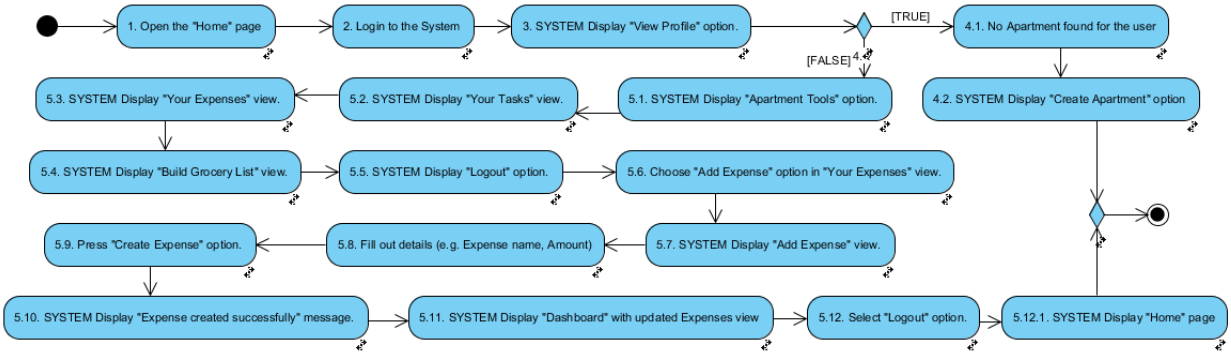


5.5 Activity Diagrams

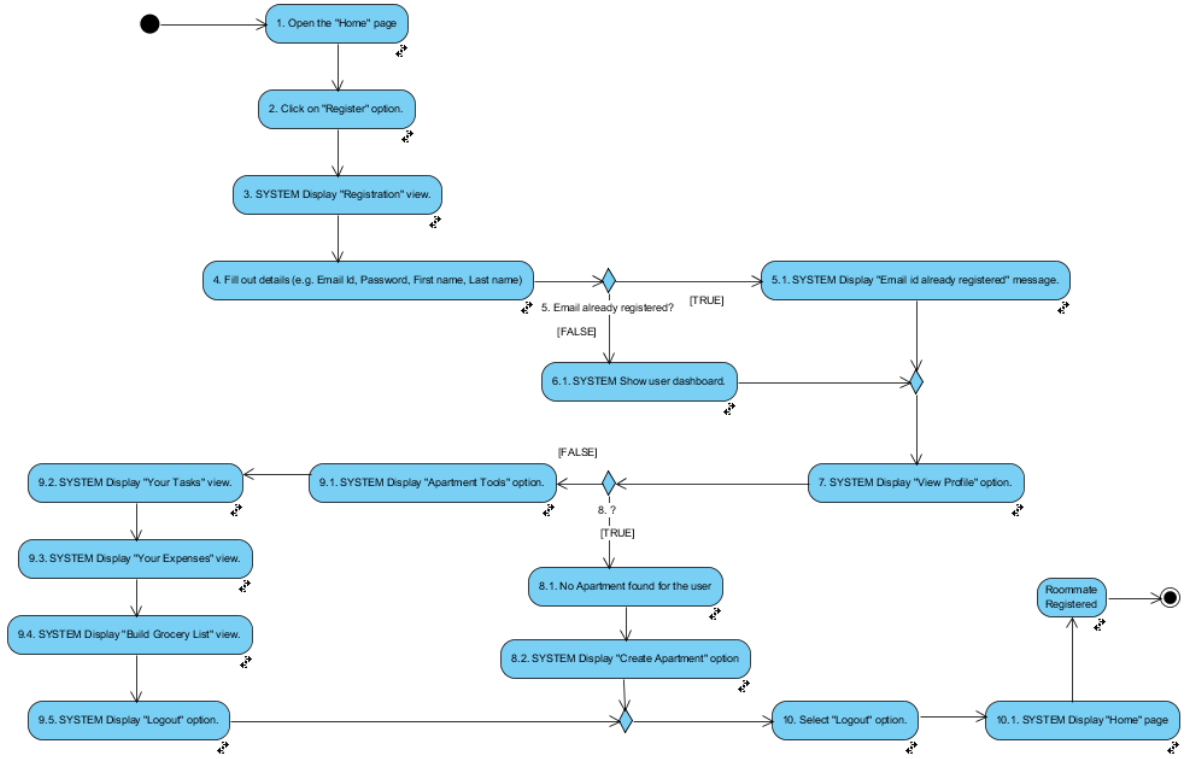
5.5.1 Create Apartment Activity Diagram



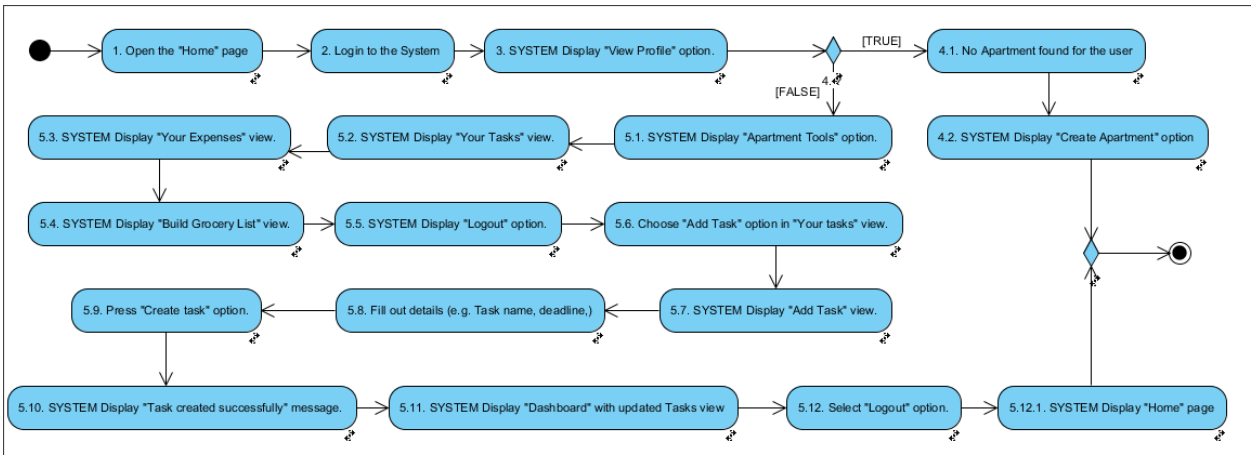
5.5.2 Create Expense Activity Diagram



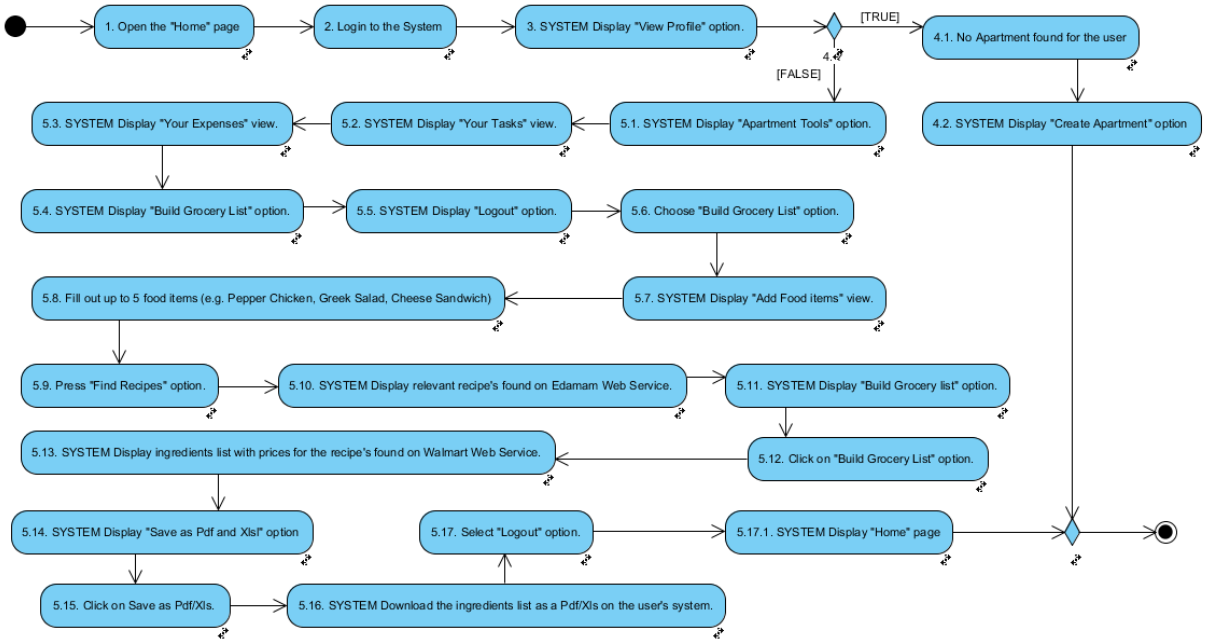
5.5.3 Create Roommate Activity Diagram



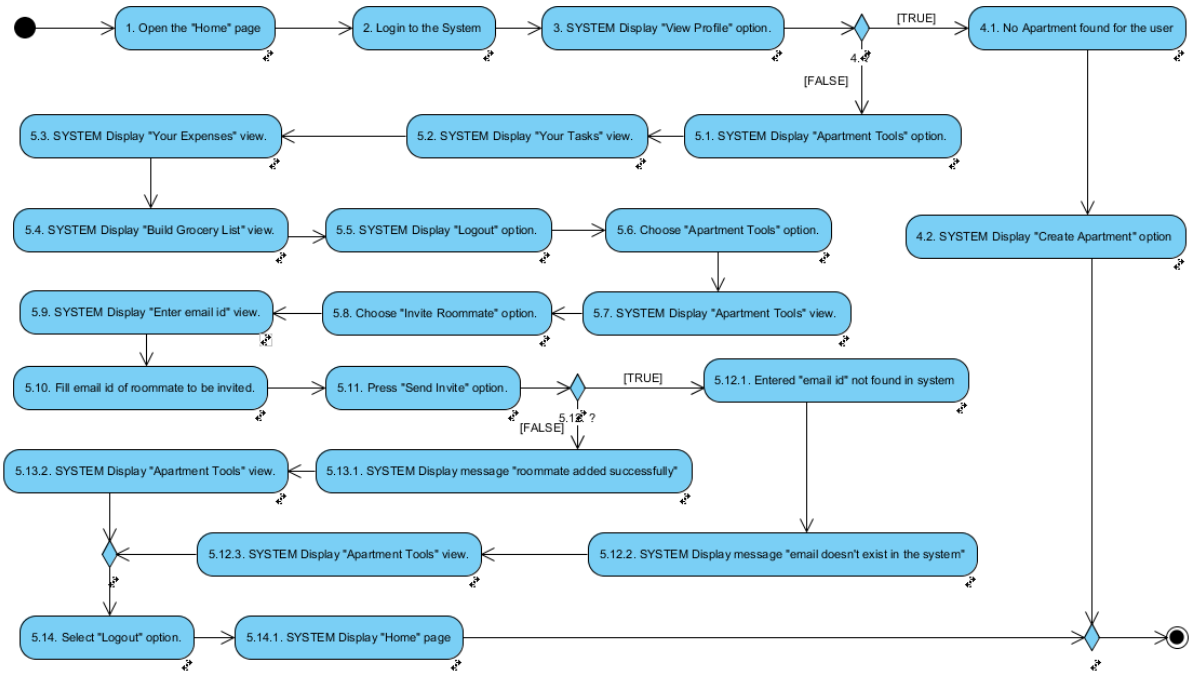
5.5.4 Create Task Activity Diagram



5.5.5 Get Grocery List Activity Diagram

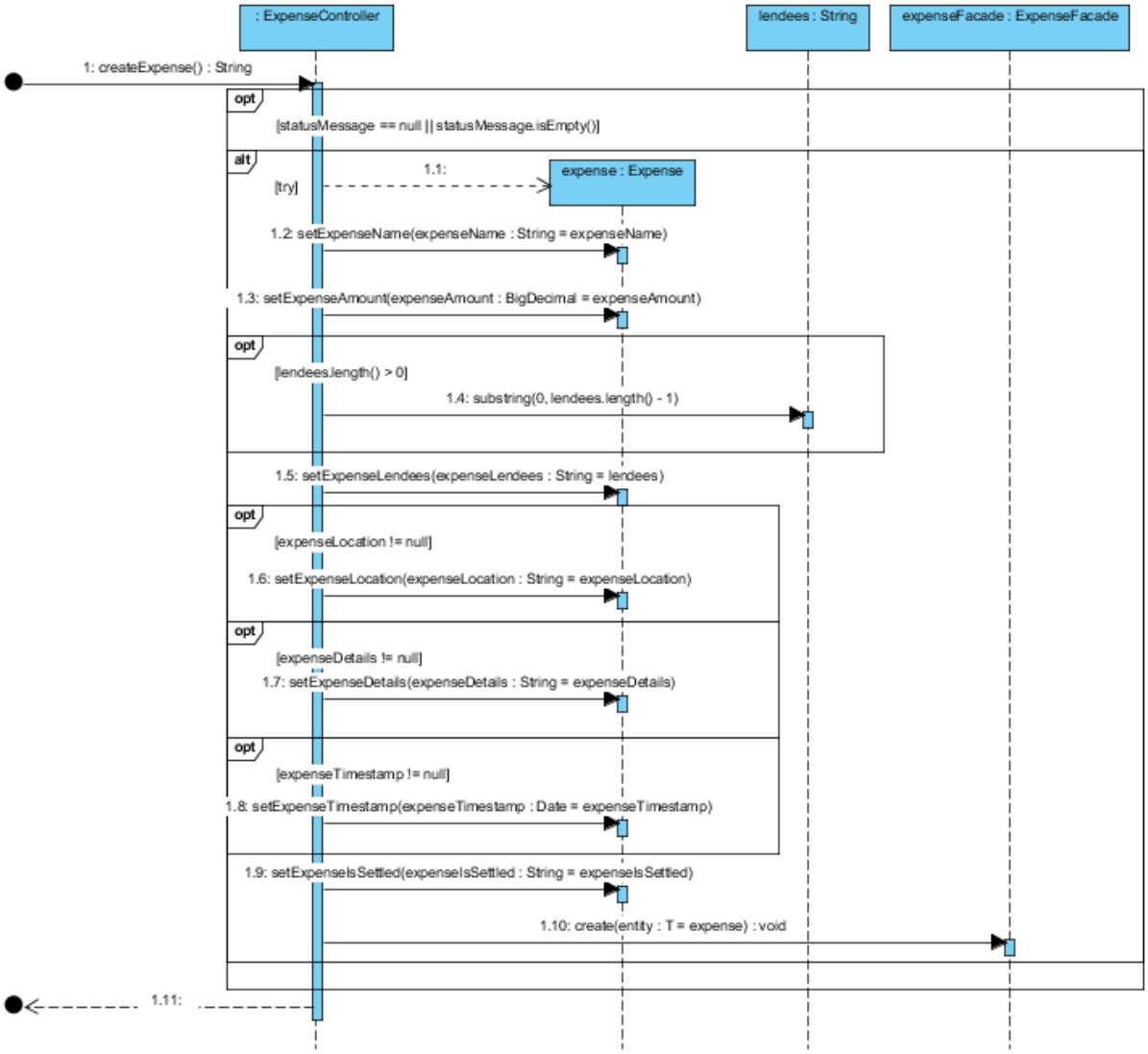


5.5.6 Invite Roommate Activity Diagram

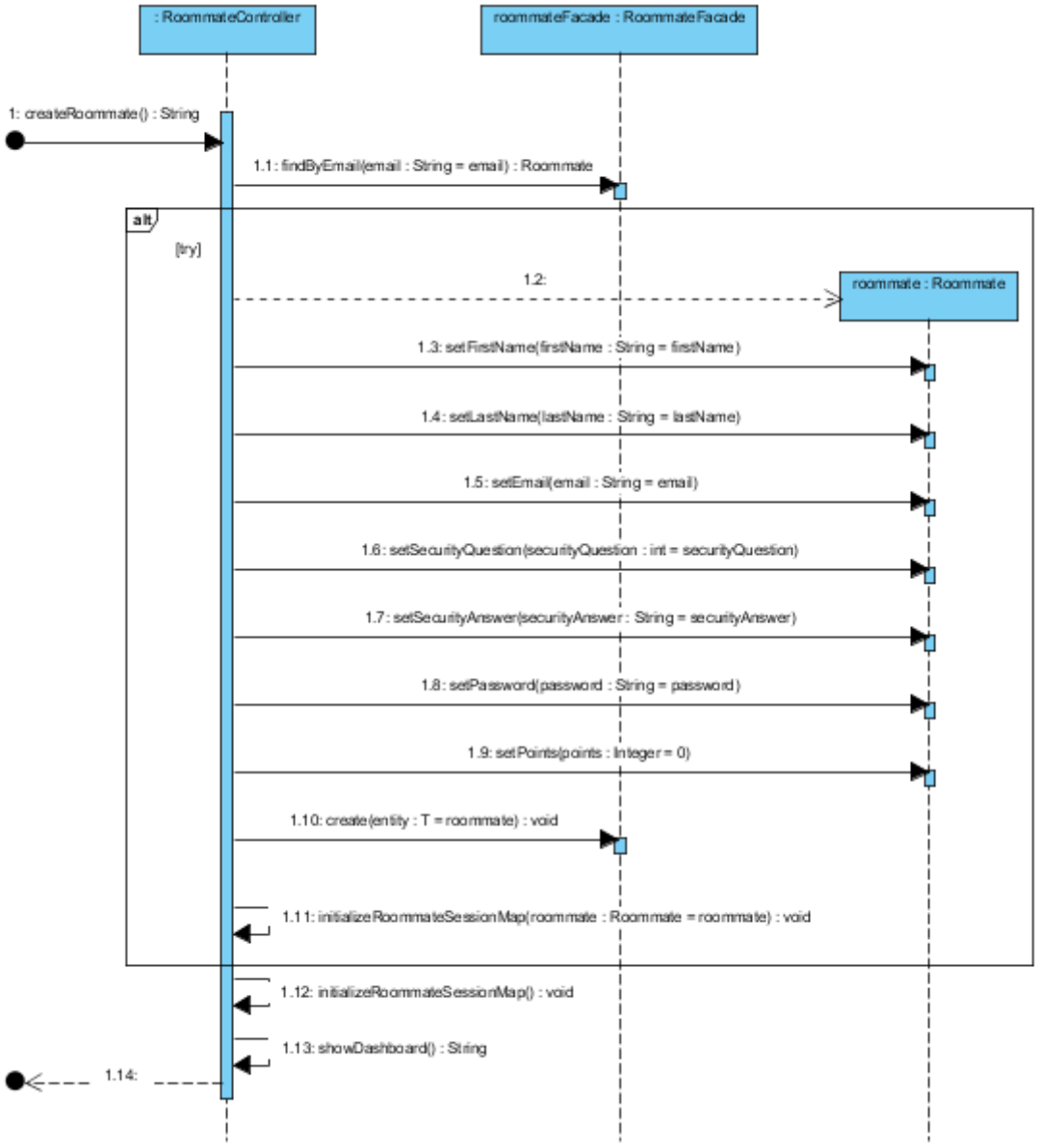


5.6 Sequence Diagrams

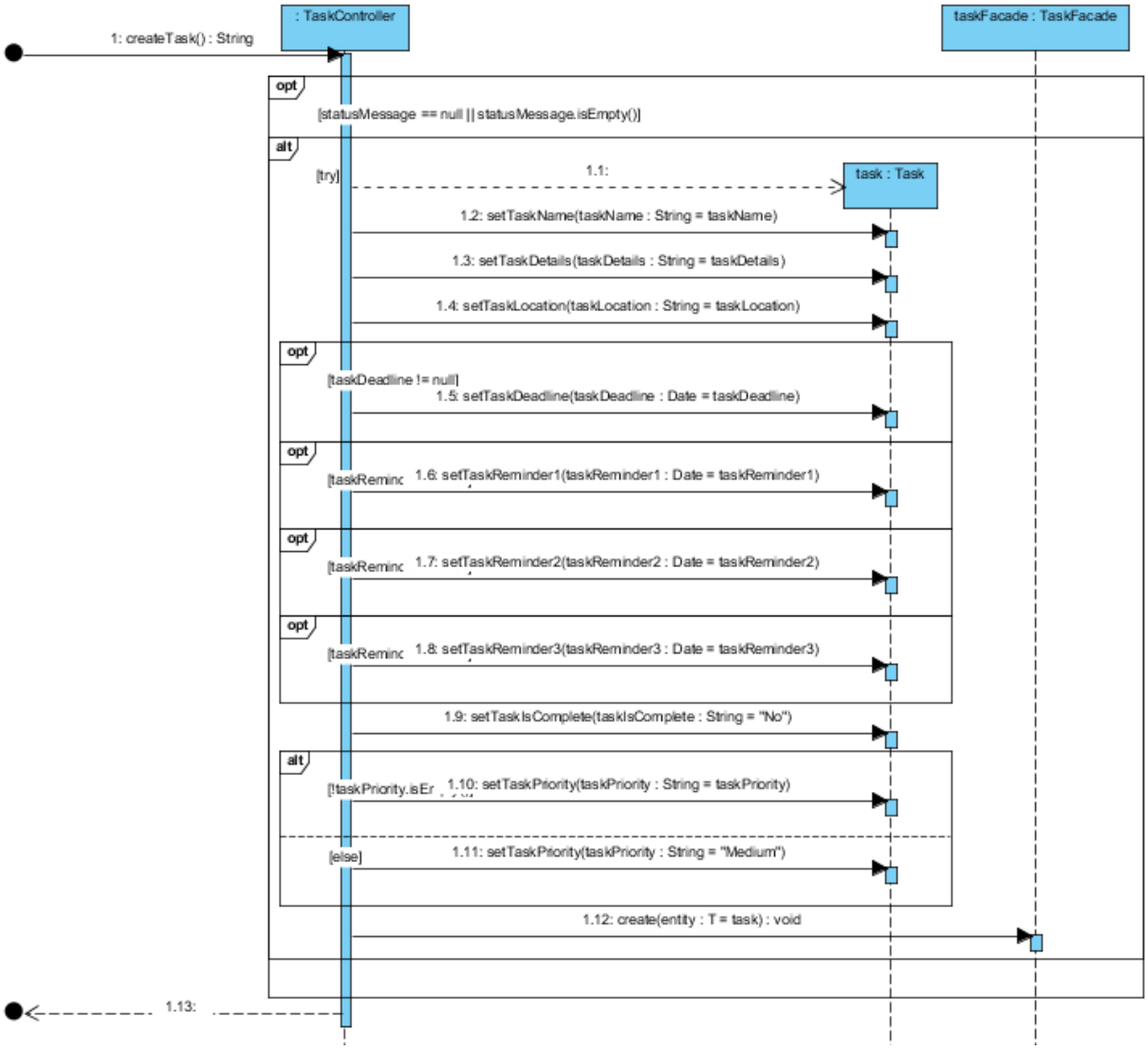
5.6.1 Create Expense Sequence Diagram



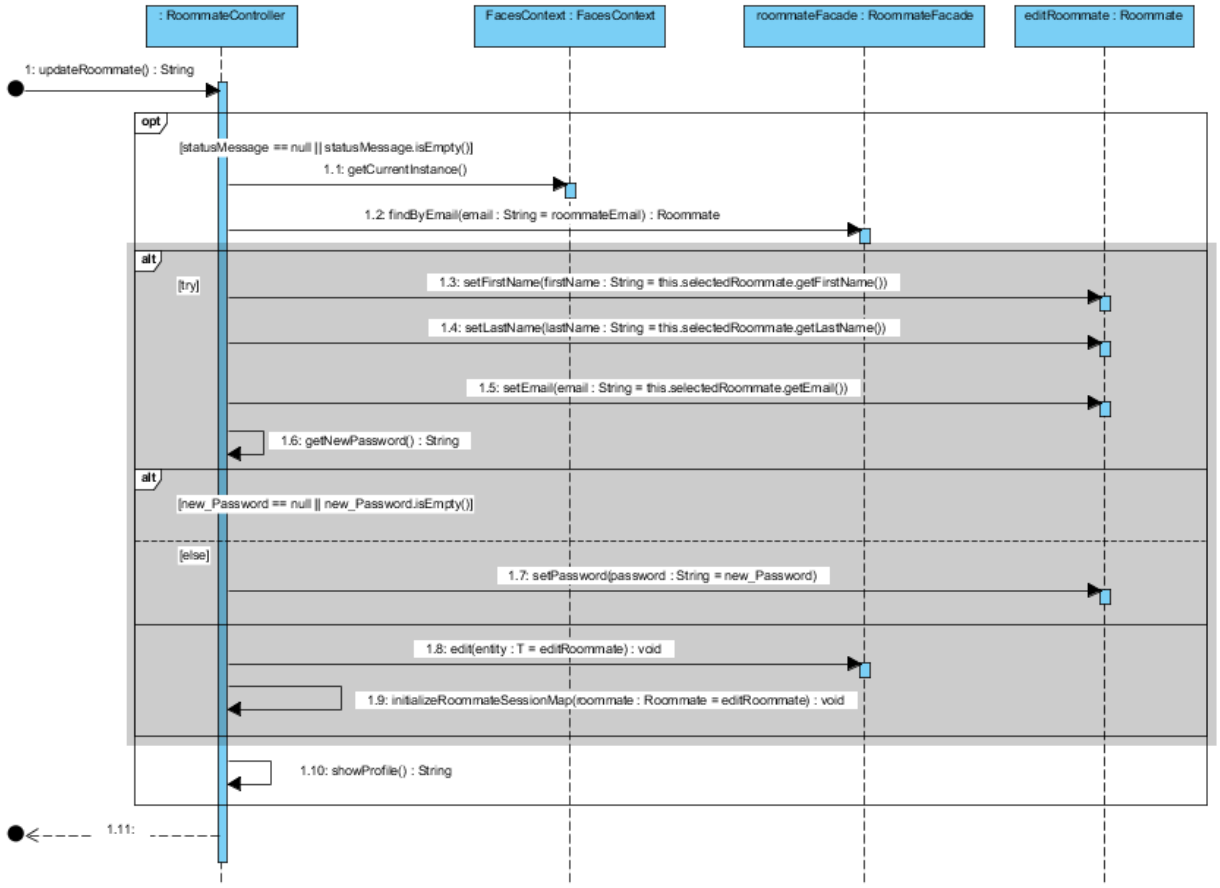
5.6.2 Create Roommate Sequence Diagram



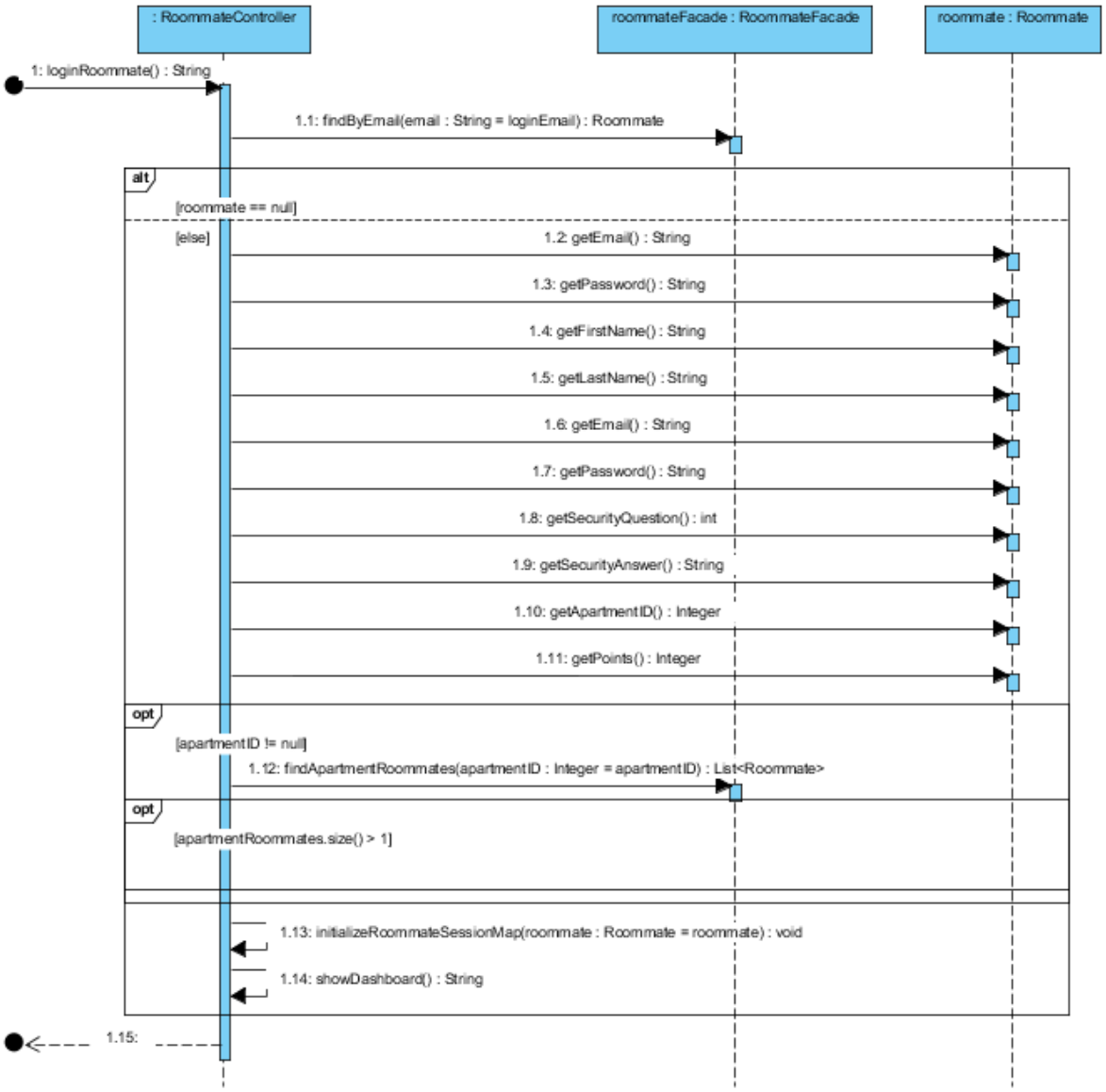
5.6.3 Create Task Sequence Diagram



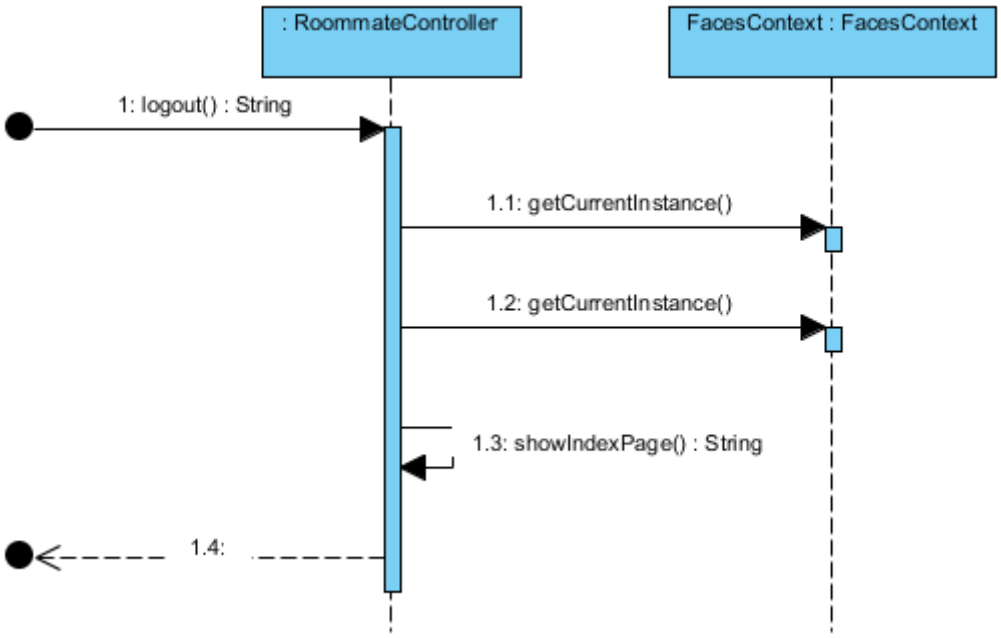
5.6.4 Update Roommate Sequence Diagram



5.6.5 Log-In Sequence Diagram

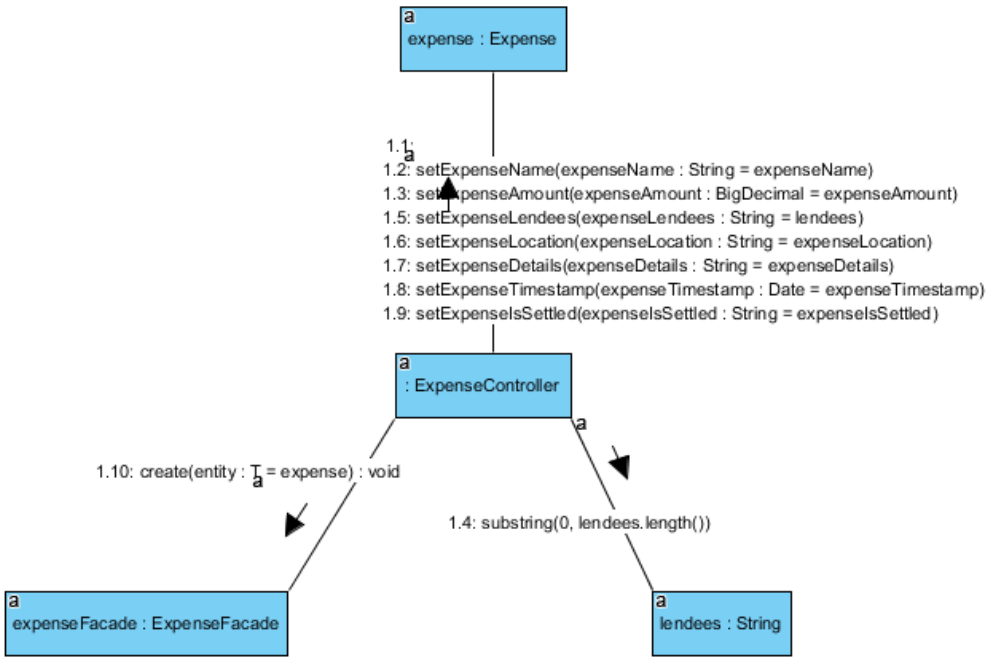


5.6.6 Logout Sequence Diagram

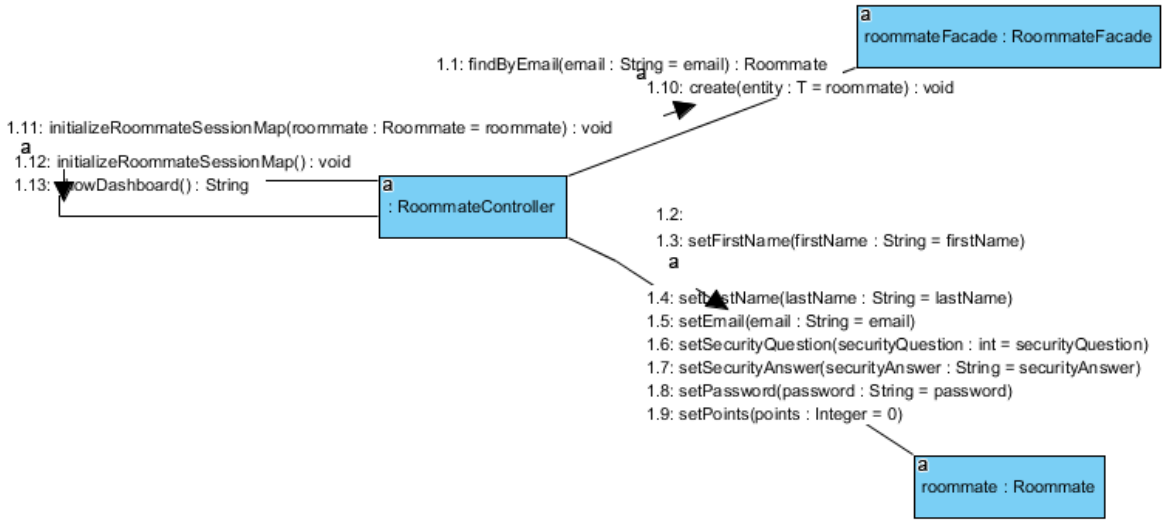


5.7 Collaboration/Communication Diagrams

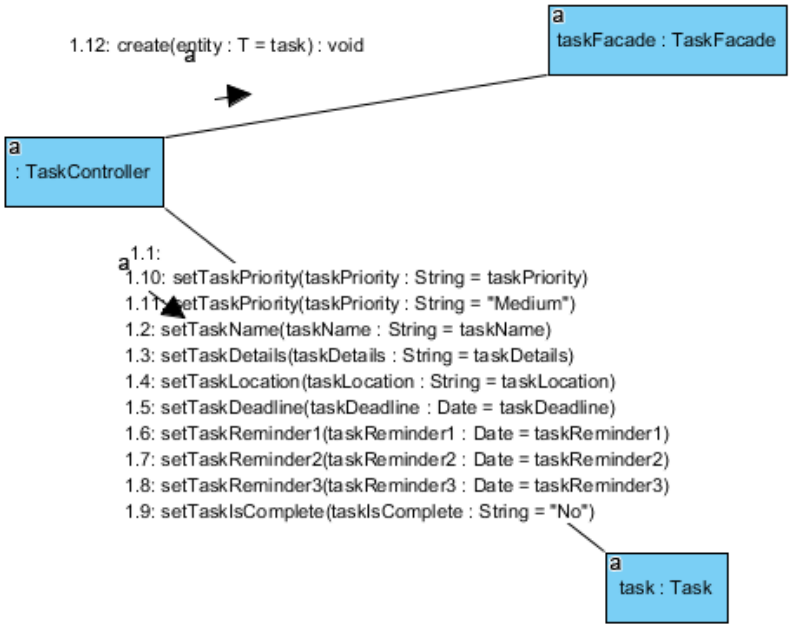
5.7.1 Create Expense Collaboration Diagram



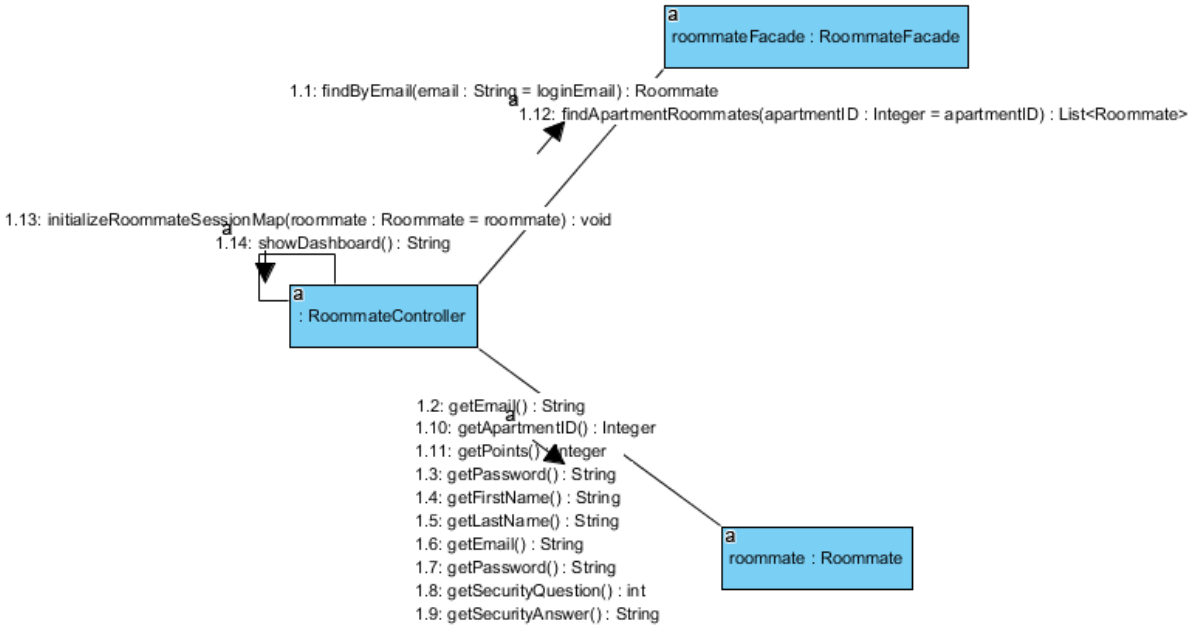
5.7.2 Create Roommate Collaboration Diagram



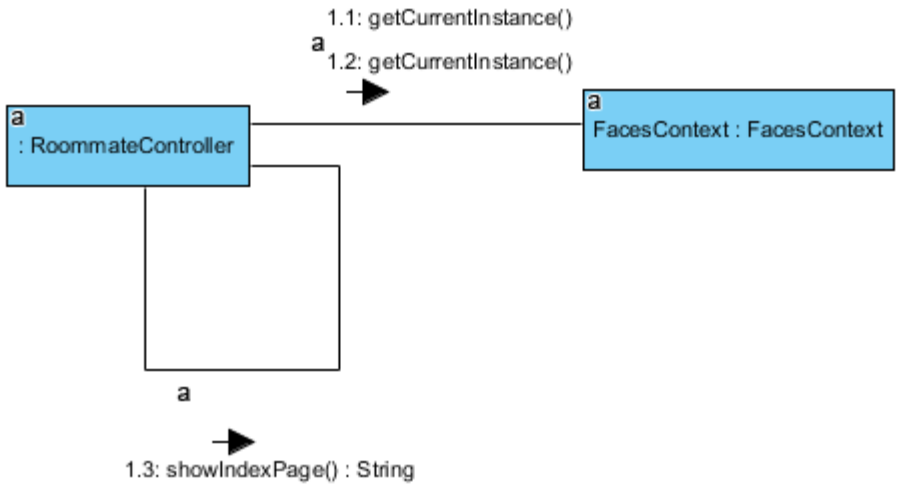
5.7.3 Create Task Collaboration Diagram



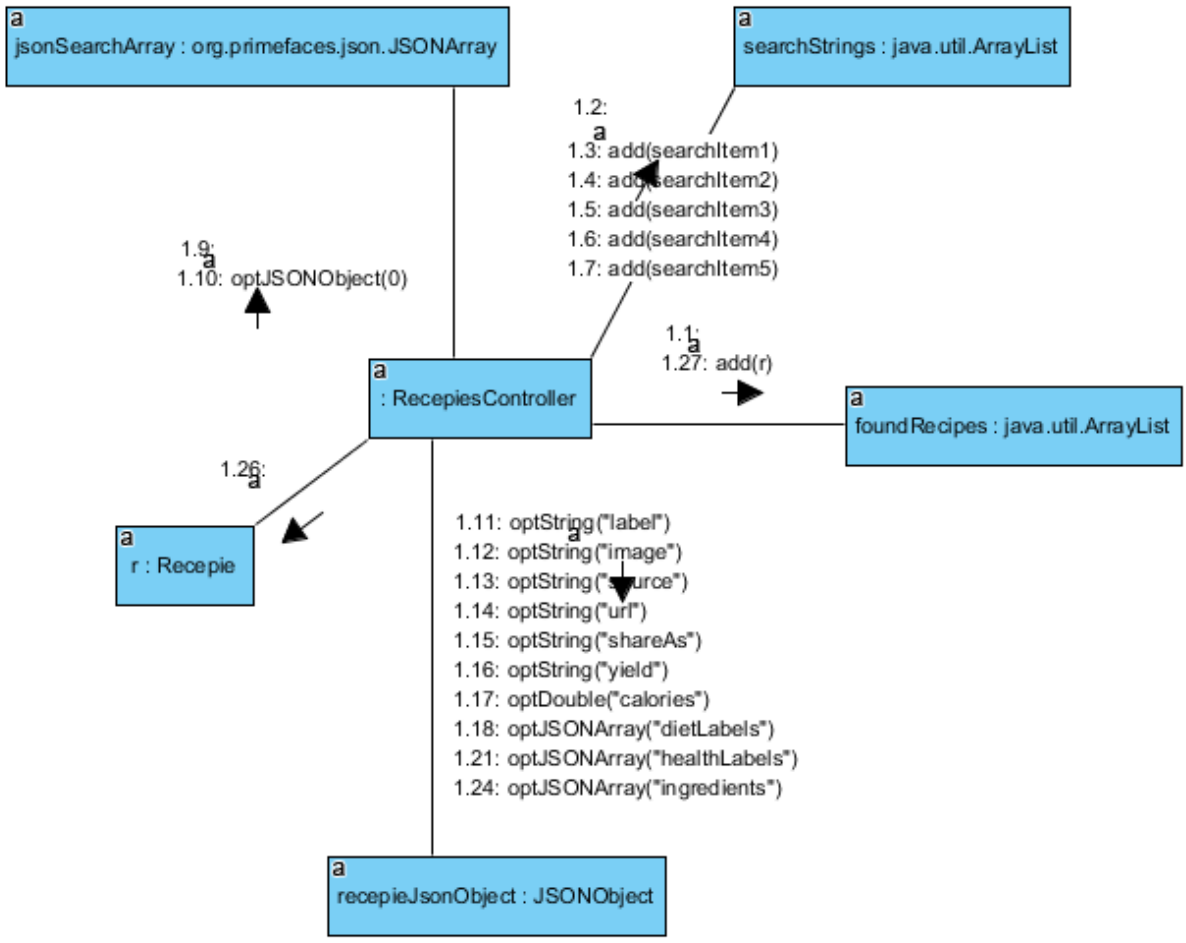
5.7.4 Login Collaboration Diagram



5.7.5 Logout Collaboration Diagram

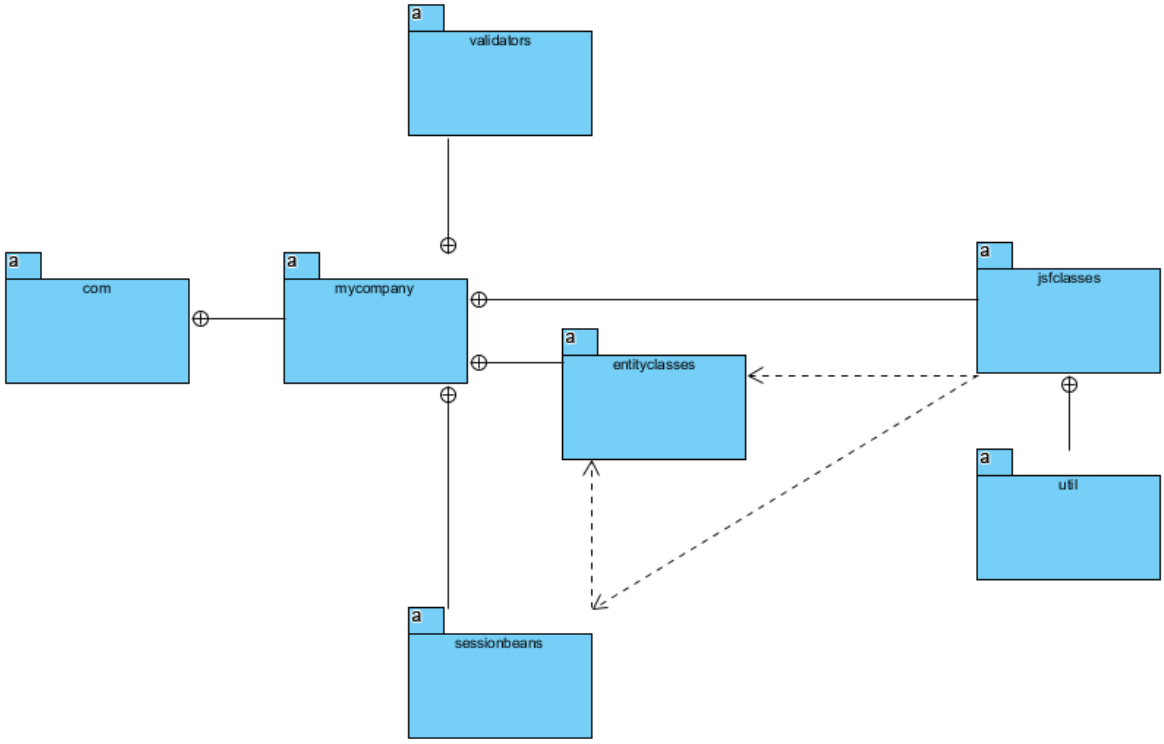


5.7.6 Recipe Search Collaboration Diagram

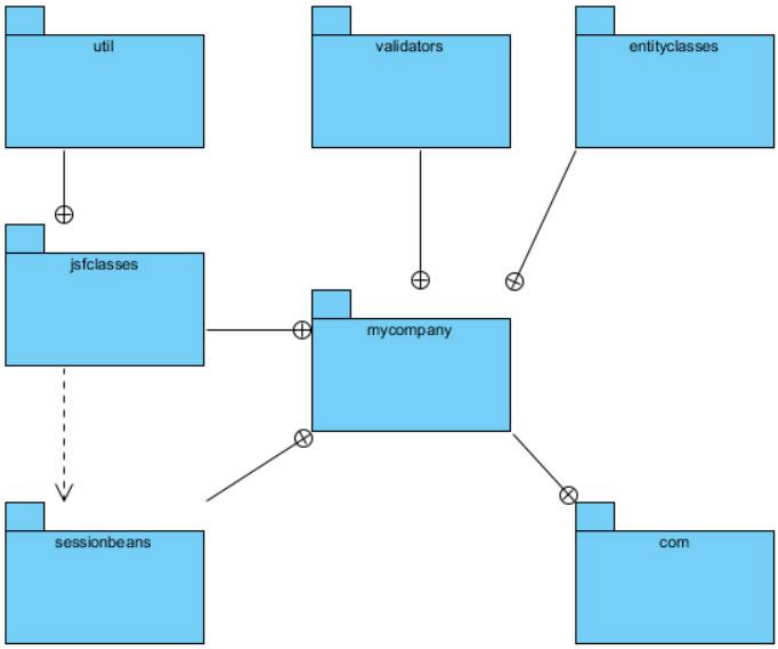


5.8 Packages

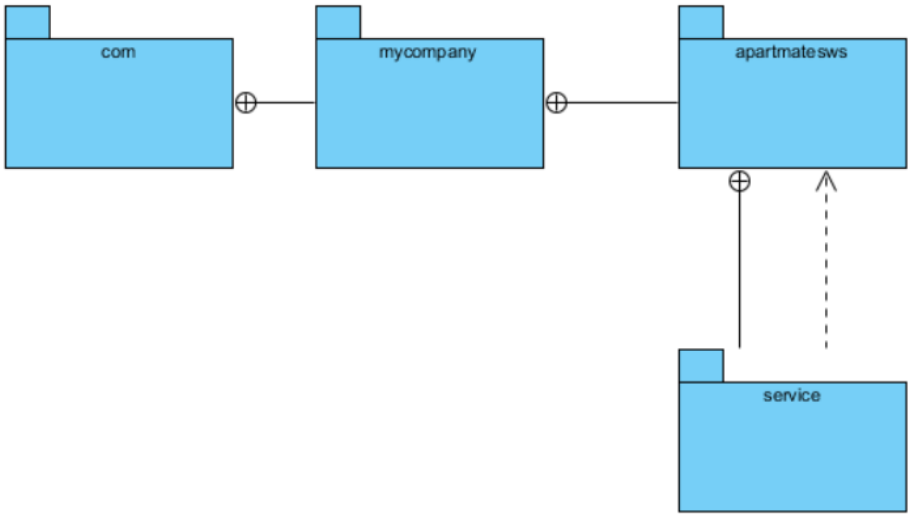
5.8.1 ApartMates Package Diagram



5.8.2 ApartMates Reminder Package Diagram



5.8.3 ApartMatesWS Package Diagram

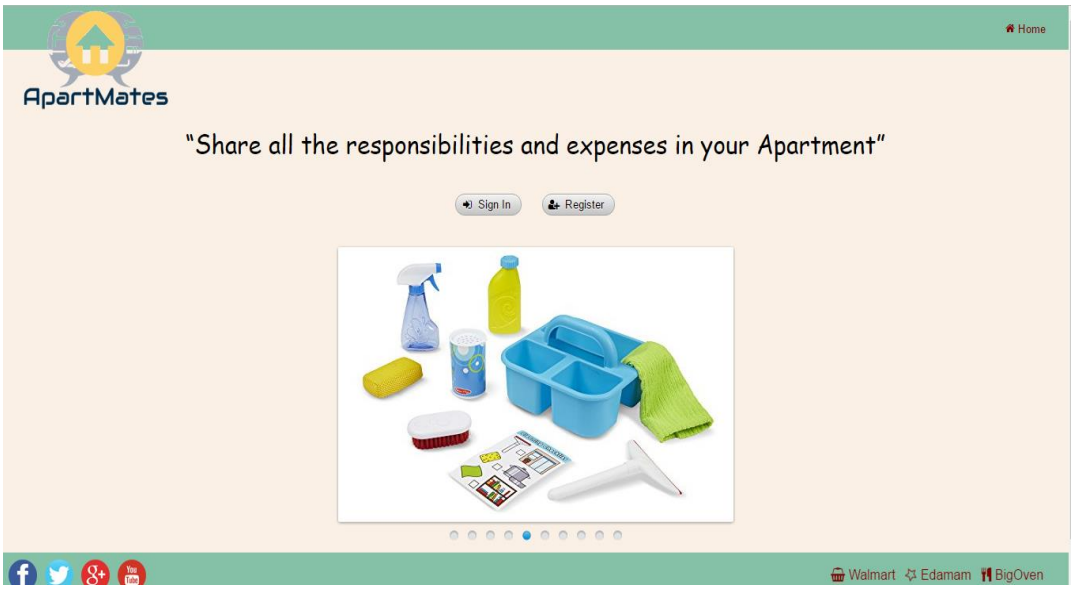


6. CLOUD SOFTWARE APPLICATION FUNCTIONALITY

Access our cloud software application "ApartMates" by navigation to the following URL: http://venus.cs.vt.edu/Apartmates/. This link will direct you to the home page of ApartMates.

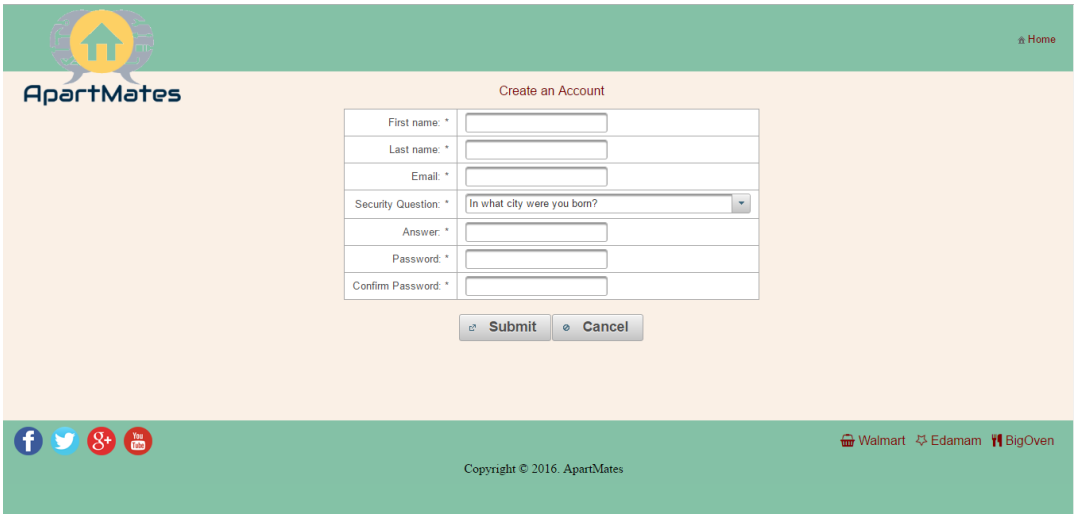
HOME PAGE:

In this page you are greeted to the home page, where you will be able to either Sign in or Register to ApartMates.



REGISTER PAGE:

In this page you are required to fill in the all of the data that appears in the screen in order to create an account.

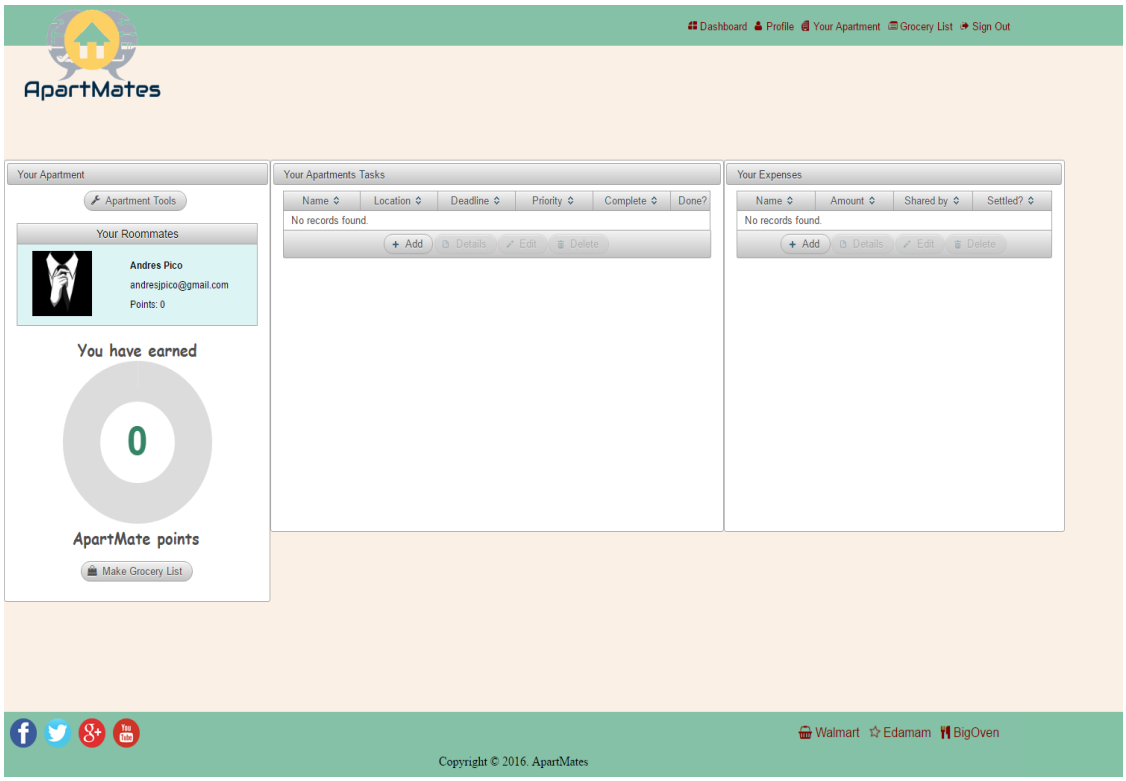


SIGN-IN PAGE:

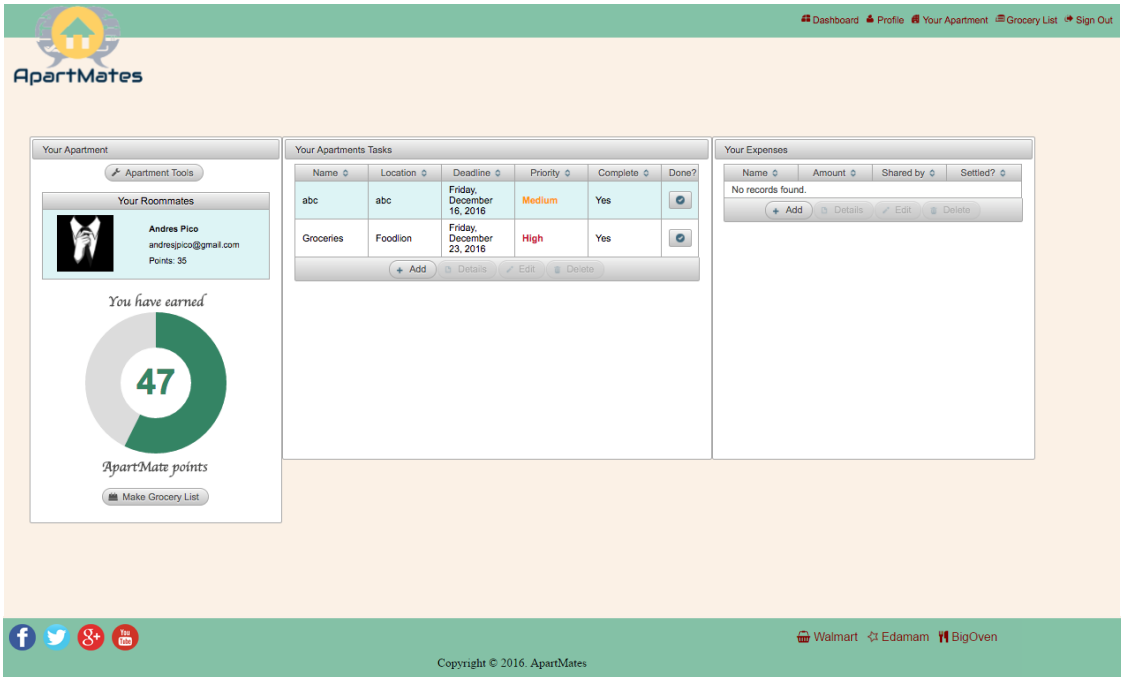
In the case that you already have an account, you will be able to log in to ApartMates by providing your email and your password.

DASHBOARD PAGE:

The dashboard is the main page for Roommates once they have logged in. In here they have access to most functions offered by ApartMates. Some of these functions include: accessing account information, accessing apartment information, creating, editing and accessing tasks, view, accessing expenses information, and accessing the grocery list page.

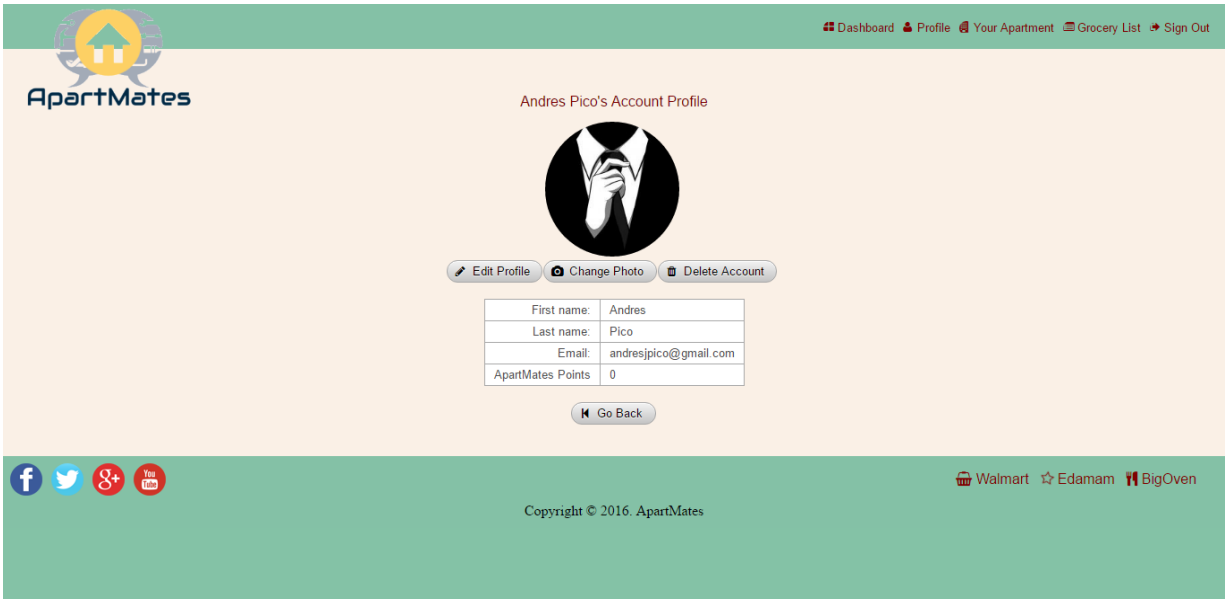


Below is the dashboard page again. But this time the picture is illustrating how the page looks like once there are new tasks available. It also shows a clear view of how the points of each user are shown on the logged-in roommate's dashboard. These reputation points are always relative to all the roommates in a household, so a roommate with large number of points shall always feel compelled to keep their points high while his or her roommates try to out beat them. The full circle represents the sum of the points of all of the roommates present in an apartment. Hence the representation also shows the amount of contribution a roommate has done in terms of tasks to an apartment.



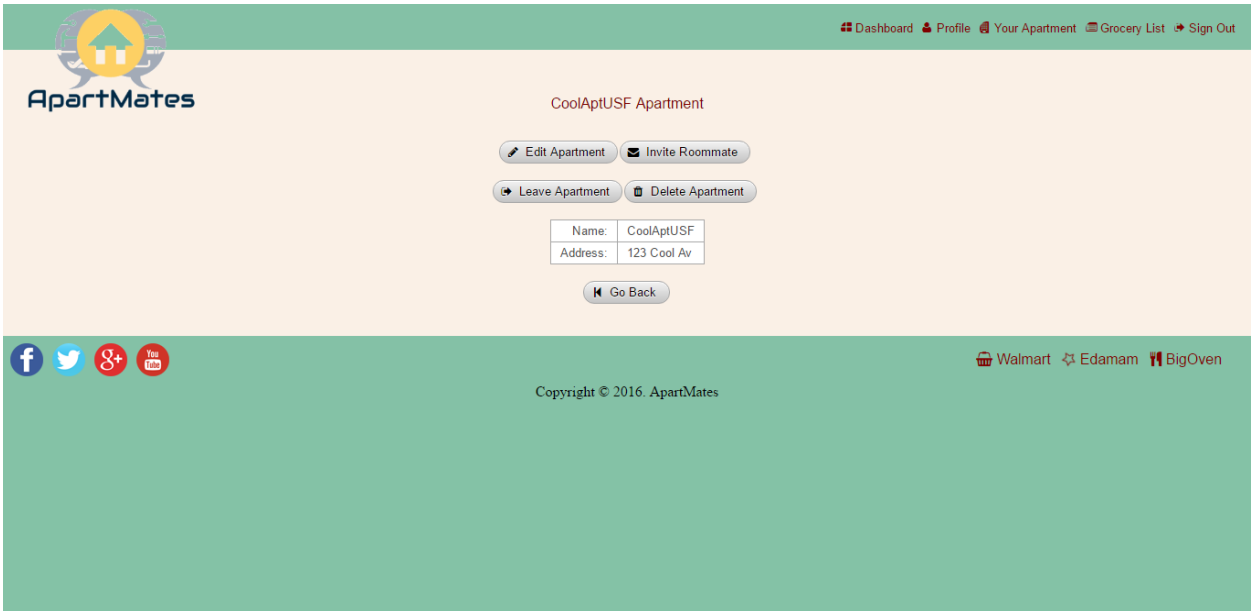
ACCOUNT PAGE:

The account page shows all the personal information about a roommate's account. Through this page, roommates will be able to upload profile pictures, edit their account information, and see the points they have gathered by completing tasks. They are also able to delete their account, but of course no one would ever want to do that.



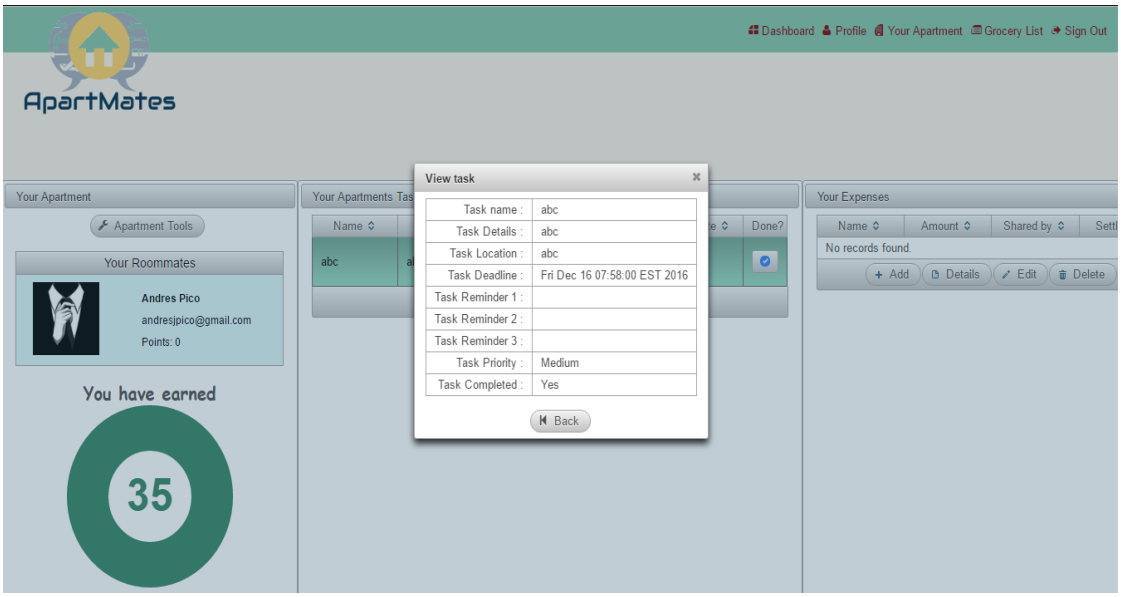
APARTMENT INFORMATION PAGE:

The apartment information page shows the most essential information about an apartment, its name and its address. In addition, this page allows roommates to invite their own roommates to be a part of their apartment. Roommates are also able to leave the apartment if they wish to do it.



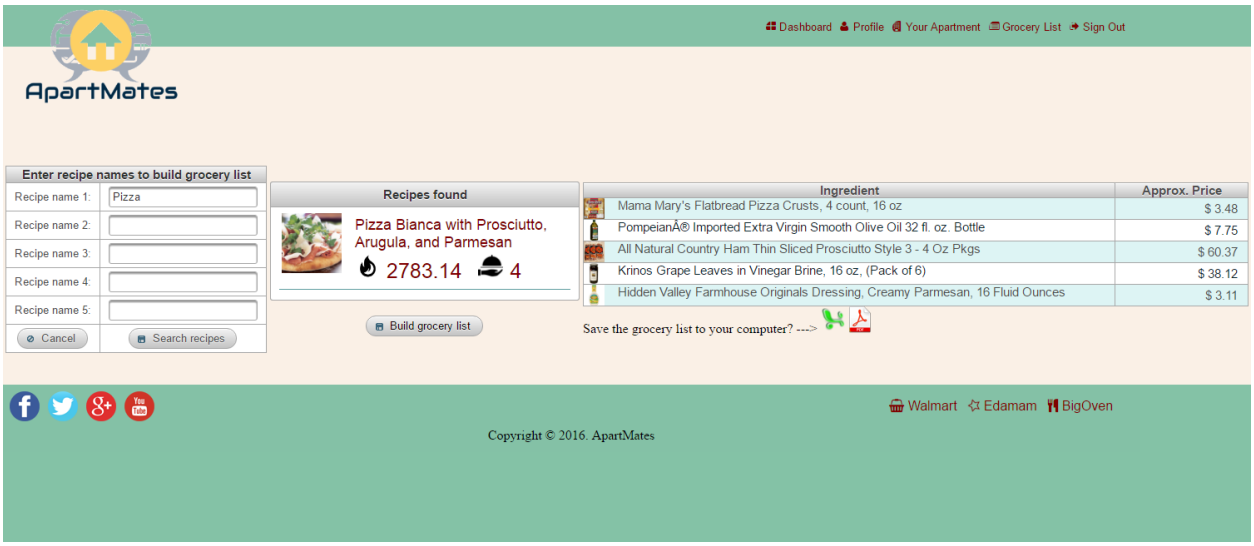
TASK INFORMATION PAGE:

As explained in the dashboard page section, roommates will have access to retrieve information about a particular task. This information will be shown to the roommate in a pop up that will appear on top of the dashboard page,



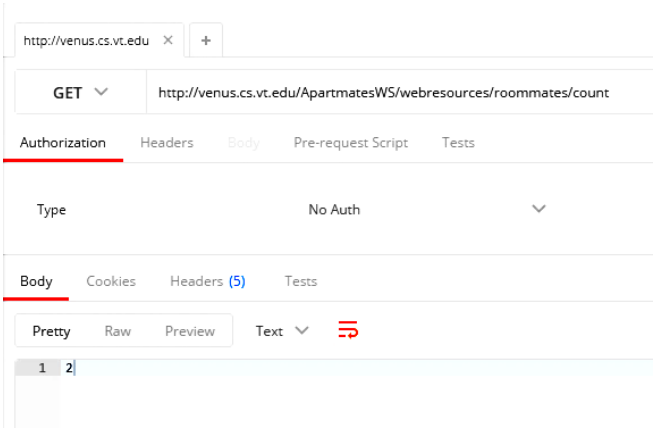
GROCERY LIST PAGE:

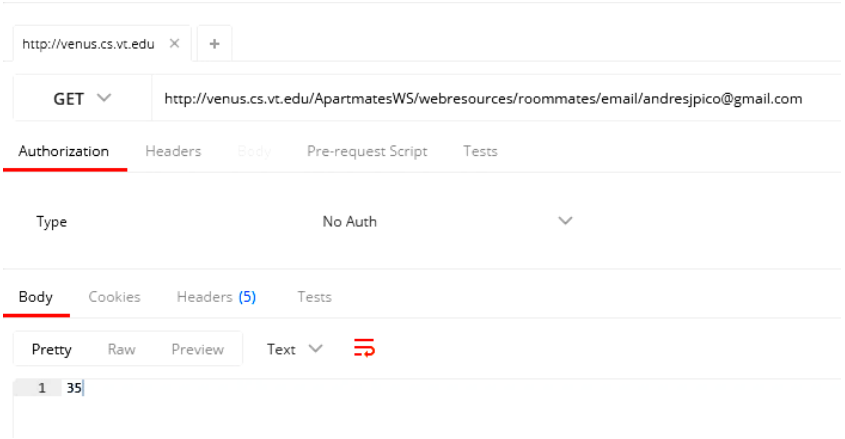
The grocery list page allows roommates to search for their preferred recipes using the Edamam API, and to generate a list of ingredients for the searched recipes along with cost estimates for each ingredient from the Walmart API. In addition, roommates can export this grocery list as a pdf and excel file in order to carry it with them to a grocery store. This is a fairly hard problem, since the ingredients available from Edamam API are English language sentences, from which individual ingredients need to be extracted with sufficient accuracy (Extracting the ingredients with 100% accuracy is an unsolved NLP research problem). We use these extracted ingredients to query Walmart Open API for getting the price of a unit of that ingredient. The extracted ingredients and their prices are shown to the user, which are available for download as pdf and excel files.



APARTMATES WEB SERVICE:

Thinking that good roommates will have the desire to show how good of a roommate they are, we built the ApartMates web service. This service will allow developers to retrieve the amount of reputatio points earned by a particular roommate given that they know the roommate's email address.





7. CONCLUDING REMARKS

ApartMates is an ideal service developed for people who share an apartment and need an organized way of distributing tasks amongst roommates. Many people experience unpleasant moments when their roommates forget to do a task, like taking out the trash in a residence where the trash truck only comes twice a week for example. For this reason and more, ApartMates allows roommates to manage tasks in a better, transparent and accountable way. Each task has a deadline, a priority, and a detail section that enables roommates with all the information needed to complete a task. In order to keep roommates motivated, a reputation points system has been implemented that will gamify the user experience and encourage friendly competition between roommates of an apartment to do more tasks and prevent deadline over dues. The points system also allows roommates to have a better understanding of who they are going to live with when looking for potential roommates. Finally, a web service is provided to allow other developers to incorporate the rewards functionalities of ApartMates into their own applications.

In conclusion, ApartMates is an application meant for people who want to carry out a more organized life when living with roommates in a shared apartment. It can also help families better organize themselves when it comes to distributing work among family members.

REFERENCES

Balci, O. (2016), "CS5704 Software Engineering Course Website," Department of Computer Science, Virginia Tech, Blacksburg, VA, http://manta.cs.vt.edu/cs5704

Oracle (2016), "NetBeans," Oracle Corporation, Redwood Shores, CA, https://netbeans.org/

Pressman, R. S. and B. R. Maxim (2015), *Software Engineering: A Practitioner's Approach*, Eight Edition, McGraw-Hill, New York, NY.