

CS30700

Design Document

**Group 22**

- Torin Hylan

- Max Finder

- Jack Foster

- Tom Buckley

- Kayleigh Francis

- John Werner

**Table of Contents**

1 Purpose……………………………………………………………………. 3

1.1 Functional Requirements……………………………………….. 3-5

1.2 Non-Functional Requirements……………………………………. 6

2 Design Outline…………………………………………………………….. 7

2.1 Interactions Between Individual System Components…………. 7-8

2.2 High Level Structural Overview………………………………... 8-9

3 Design Issues…………………………………………………………….. 10

3.1 Functional Issues………………………………………………… 10

3.2 Non-Functional Issues…………………………………………… 13

4 Design Issues…………………………………………………………….. 13

**1 Purpose**

Every year, thousands of students at Purdue are matched up in dorm housing. However, the process for students being matched up in the dorms is simply a single questionnaire and gives no flexibility for students to make decisions or discuss with their roommate(s). Besides dorms, even more students live in off-campus housing and are looking for roommates, but do not have a convenient means of search. There are ways to search for roommates online, but other users won’t always be students at Purdue.

Due to these drawbacks, we believe a better solution is a necessity. This is where our application, UniRoom, comes into play. UniRoom will have a sleek, tinder-esque UI, which is familiar to many college students. UniRoom will allow users to browse a data driven machine learning generated queue, which will provide them with a list of roommates tailored just for them. The purpose of UniRoom is to innovate while simplifying the old style of finding roommates in a new, modern, and more relevant manner for all Purdue students.

With a more structured way of presenting personal information to other users, there needs to be a variety of options in showing other users who you are and what living with you will look like. This is a new challenge when compared to Facebook, where students will post a paragraph about anything they choose for other students to see. To achieve the ability for users to share just as much about themselves, if not more, UniRoom will allow users to share a bio, interests, area of study, sleeping habits, cleanliness, relationship status, among many other factors. Not only will that information be visible to other users (with permission of the user), but the feed that is shown will be based on those same factors to ensure the best chance for the user to find their best fit.

**1.1 Functional Requirements**

1. User Profile

*As a user,*

1. I would like to be able to create an account.
2. I would like to be able to sign in to my account with a username and password.
3. I would like to be able to reset my password through email.
4. I would like to have security questions.
5. I would like to be able to sign out of my account.
6. I would like to upload pictures of myself to my account.
7. I would like to delete my account.
8. I would like to edit my account.
9. I would like to view my questionnaire responses on my profile.
10. I would like to edit my questionnaire responses on my profile.
11. I would like to be able to add various social media usernames / links to my profile.
12. I would like to control the range of other viewable accounts.
13. Questionnaire

*As a user,*

1. I would like to be able to fill out a questionnaire.
2. I would like to enter my major.
3. I would like to enter my hometown.
4. I would like to enter my pet peeves.
5. I would like to say whether I am introverted / extroverted.
6. I would like to say whether I am very studious or not very studious.
7. I would like to fill out a section with things I plan on bringing with me to school.
8. I would like to say whether or not I plan on living in the dorms.
9. I would like to say how many roommates I plan on having.
10. I would like to have a section where I can fill out additional information.
11. I would like to enter my age.
12. I would like to specify my gender.
13. I would like to note if I have been vaccinated against Covid-19.
14. I would like to specify where I would like to live.
15. Account Viewing

*As a user,*

1. I would like to disable my account from the queue.
2. I would like to view another's user profile.
3. I would like to close out of viewing another’s profile and be returned back to the queue.
4. I would like to view another user's profile pictures.
5. I would like to view another user’s profile information from their questionnaire answers.
6. I would like certain questions on a user’s profile to be highlighted noting that we both answered them the same.
7. I would like certain questions on a user’s profile to be highlighted noting that we both answered them the opposite.
8. I would like to view another user’s social media via links.
9. I would like to see whether another user has been vaccinated for Covid-19.
10. Security

*As a user,*

1. I would like to choose what parts of my profile other users can see
2. I would like to block other users from messaging me and viewing my account.
3. I would like to view the application's code of conduct.
4. I would like to report an account that violates the application’s code of conduct.
5. I would like to view the application’s privacy policy.
6. I would like to request that my profile and personal information be completely removed from the application, it’s database, and the match queue.

*As an admin,*

1. I would like to ensure accounts have read the code of conduct.
2. I would like to verify each account and it’s entered information.
3. I would like to ban certain accounts via their email address.
4. I would like to verify users have consented to the applications privacy policy.
5. Matched Profiles

*As a user,*

1. I would like to be able to click on a button that redirects me to the matched profiles section.
2. I would like to view a list of all matched profiles.
3. I would like to click on a matched profile to begin instant messaging with that user.
4. I would like to remove other users from my list of matched profiles.
5. I would like to view read receipts on messages sent by other users.
6. I would like to view timestamps on my messages.
7. I would like to have a checklist of things me and my match plan on bringing with us.

**1.2 Non-Functional Requirements**

1. Performance

*As a user,*

1. I would like to see another profile in under 1 second.
2. I would like for the application to load in under 5 seconds.
3. (after clicking login) I would like to login in under 30 seconds.
4. I would like for my messages to be sent in under 10 seconds.
5. I would like to be able to submit a crash report to the site's admins.

*As an admin,*

1. I would like to be able to host multiple users on the site at once.
2. I would like the application to support 5,000 users.
3. I would like for the application to never crash.
4. I would like one email update per day with a summary of my users and their data (if time allows).
5. I would like to receive detailed crash logs from users when they occur.
6. I would like to scale the application to function for other college campuses (if time allows).
7. Appearance

*As a user,*

1. I would like to be able to choose between a light or dark mode.
2. I would like for the UI to feel clean and organized, as if everything feels like it is where it should be.
3. Security

*As a user,*

1. I want my private data to be secure and **encrypted** (if time allows) on the site.

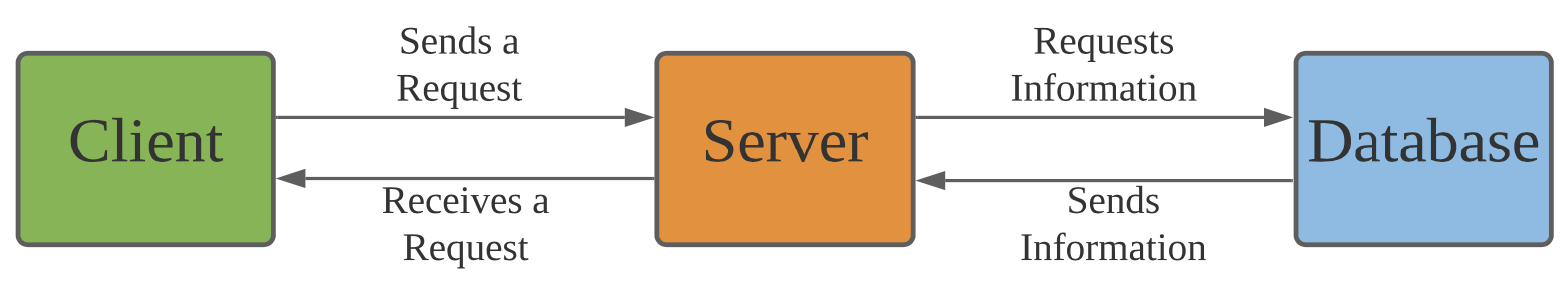
*As an admin,*

1. I would like to have a password strength checker upon password creation.
2. I would like to require some form of two factor authentication when a user creates an account.

**2 Design Outline**

Our project will be a mobile application which allows users to browse, match, and chat with other individuals on the application with the focus being finding roommates for the upcoming school year. This application will utilize a client-server model with our one server hosting a multitude of clients simultaneously. The server will also use a database where it will pull in user profiles and data.

1. Client
2. When a user logs into their account, they will be logging onto a client with account data pulled in from their user profile within our database.
3. The client will be the basis for all of the applications UI.
4. The client will send requests to the server such as: give me the next account in the match queue.
5. Server
6. All client requests will be handled via the server.
7. The server will send queries to the database to modify, delete, or return data.
8. The server will also handle responses from the database accordingly.
9. Database
10. The database will receive and respond to server queries and return the necessary data back to the server.
11. The database will store all user and site data used throughout the application.



**2.1 Interactions Between Individual System Components**

Upon starting the application, the client may request user data from the server, such as the query of potential matches, or for logging in to their account. This will then prompt the server to send a query to the database. The database will then retrieve that data and send it back to the server, where the data will be interpreted before sending the client the requested data or any error that occurred in the process. Consequently, the user interface would reflect the success of the request by providing the user with the requested data or an error message in the event of an error.

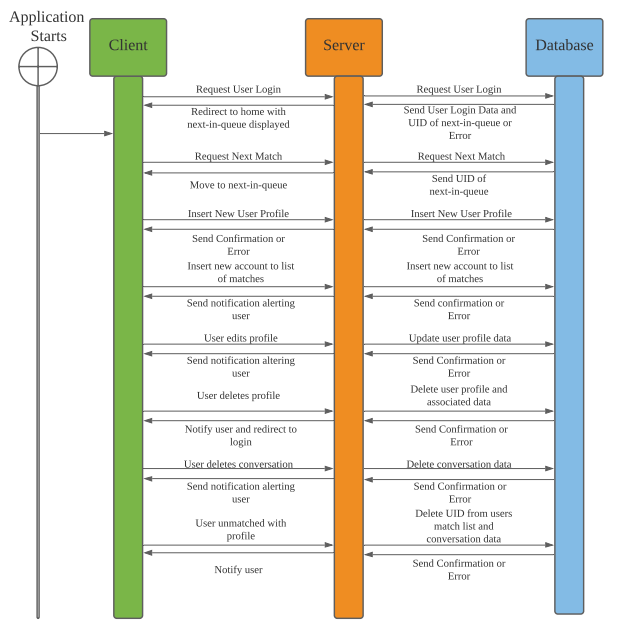
Another quite common interaction would be a client requesting to insert user data to the database. We would see this type of interaction typically when creating a profile for the first time, in addition to matching with another user. The interaction between the various components would be that first, the client would send a request to the server to add data to the database. The server would then validate the data, and then send a query to the database to add the data as long as there were no errors in validation. Next, the database will add the data in the proper location assuming no errors. If there was an error at any point in the process, the server would send the client an error message to display in the user interface. In the event of no errors, the user would receive a prompt from the server such as “Profile created successfully”.

Similar to adding user data to the database, clients can also edit or remove user data. This behavior would occur when a profile is updated or deleted, or a user unmatches with a match. The interaction between the various components would be similar to inserting data. First, the client would send a request to the server to modify or remove data. Then, the server would process that request and send a query to the database as long as the request is valid. The database would receive the query and go on to modify or remove the existing data, and upon successful completion, the database would send any necessary data back to the server. The server would acknowledge the successful request and display a prompt to the user such as “Profile information edited/removed successfully”.

**BE MORE SPECIFIC FOR WHAT TYPE OF DATA**

**2.2 High-level structural overview**

On the next page is a visual representation and breakdown of a comprehensive overview of our high level system components. These components will interact in the manner described above in 2.1, but to reiterate, these interactions are the various calls made between our client, server, and database. These calls are generally add, insert, edit, and remove. However, as mentioned these are the general forms of these calls. The more specific calls are seen below at each right facing arrow and the answers are shown via the left facing arrows.



**3 Design Issues**

The following subsections 3.1 and 3.2 outline both functional and non-functional issues we need to account for when designing our system and its components.

**3.1 Functional Issues**

***Issue 1: What do users need to create an account?***

**Options:**

1. Name, Email, Username, and Password
2. Name, Email, Username, Password, and Birthday, Year in School
3. Name, Email, Username, Password, Phone Number, and Birthday, Year in School
4. Name, Email, Username, Password, Phone Number, and Birthday, Year in School, Gender

**Choice:**

Option d.

**Justification:**

We will by default need users’ birthdays to ensure that the age limit of our app is met and to display with the users’ profile, but the methodology of logging in can be done in multiple ways. The first option will be the most accessible, where someone could use our app even if they do not have phone service via a username and password that we can encrypt, store, and reference from our database. The second option will be that a user can sign in via their phone number, as we can use the twilio API (or others) to send a verification text to the user, ensuring security.

***Issue 2: What do we ask the users to survey basic roommate compatibility?***

**Options:**

* 1. Use Y/N questions
     1. Do you smoke?
     2. Do you like to have people over?
     3. Do you play videogames?
     4. Do you go to bed early?
     5. Do you plan on partying?
     6. Do you have a car?
     7. Are you neat?
     8. Do you want pets?
     9. Are you introverted?
     10. Do we need to check before having someone over?
     11. Do you want to do joint grocery shopping?
     12. Do you have a significant other?
  2. Use multiple choice, and Y/N questions

1. How frequently do you have people over?
   1. Never
   2. Once a week
   3. 2-3 times a week
   4. 4-5 times a week
   5. 6-7 times a week
2. Do you smoke, if so how often?
   1. Never
   2. Once a week
   3. 2-3 times a week
   4. 4-5 times a week
   5. 6-7 times a week
3. How frequently do you play videogames?
   1. Never
   2. 1-4 hours a week
   3. 5-8 hours a week
   4. 9-12 hours a week
   5. 13 or more hours a week
4. What time do you go to bed during the week?
   1. 6-8pm
   2. 8-10pm
   3. 10-12pm
   4. 12-2am
   5. 2am or later
5. What time do you go to bed on the weekend?
   1. 8-10pm
   2. 10-12pm
   3. 12-2am
   4. 2-4am
   5. 4am or later
6. How clean are you?
   1. Very messy
   2. Somewhat messy
   3. Not necessarily messy or clean
   4. Somewhat clean
   5. Very clean
7. How would you like to handle chores?
   1. Assign tasks to each person
   2. Do them whenever you notice they need to be done
   3. You don’t want to do any chores
8. Do you have a car?
   1. Yes
   2. No
9. Do you want pets in the room?
   1. No
   2. Yes, dog(s)
   3. Yes, cat(s)
   4. Yes, (other)
10. Are you introverted or extroverted?
    1. Definitely introverted
    2. Somewhat introverted
    3. Neither introverted or extraverted
    4. Somewhat extraverted
    5. Definitely extraverted
11. Do we need to check before having someone over?
    1. Never, no
    2. Sometimes, yes
    3. Most of the time, yes
    4. Always, yes
12. Do you want to do joint grocery shopping?
    1. Yes
    2. No
13. Do you have a significant other?
    1. Yes, on campus
    2. Yes, not on campus
    3. No

**Choice:**

Option b.

**Justification:**

By asking multiple choice questions in addition to Y/N questions we will be able to capture a more specific view of each user, and better feed our machine learning algorithm to effectively show new user profiles. Only Y/N questions would also limit the number of questions we could ask. While we don’t want an obnoxiously long survey, having many options is nice so we can decide what we ask users and not have to worry about running out of questions. Giving a spectrum of options also helps the user by allowing them to choose a more accurate option instead of making them feel like they just have to choose the better of two options.

***Issue 3: What information will users be able to share in the feed?***

**Options:**

1. Basic information and bio
2. Basic information, bio, and interests
3. Basic information, bio, interests, and area of study
4. Basic information, bio, interests, area of study, and social media

**Choice:**

Option d.

**Justification:**

Choosing roommates is a contracted commitment, so we want to make sure that users are able to make the most educated decisions possible about other profiles that they are viewing. Seeing additional information like interests, area of study, and social media accounts allows users to make better informed decisions about committing to their next roommate. Each option to share information will be completely up to the user, so the capability will be there whether the user decides to share each piece of information or not.

**3.2 Non-Functional Issues**

***Issue 3: What languages will the iOS app be developed in?***

**Options:**

1. Swift
2. JavaScript, React.js, React Native
3. Flutter

**Choice:**

Option b.

**Justification:**

Since our group is a mix of Mac users and Windows users, we decided that Swift wouldn’t be a convenient option. Secondly, the members of our group had more familiarity with JavaScript and React Native than Flutter. Without having to start from ground zero, we will be able to jump the learning curve with JavaScript and React Native much faster, allowing our group to dive into more complex features of UniRoom earlier in the development process.

***Issue 4: Where will user data be stored?***

**Options:**

1. Firebase
2. Google Cloud Platform
3. AWS

**Choice:**

Option a.

**Justification:**

We selected Firebase as our data storage platform as it is an easily accessible cloud based storage solution. Firebase was also selected since our group is not very familiar with databases, and Firebase is an easy to learn platform that has a lot of documentation and resources online. With the help of outside resources to aid us through the process, our group felt that Firebase would be the simplest way to meet our database needs.

**4 Design Details**

(a) Include class level design of the system (i.e. class diagrams) and be as detailed as you can.

* **User**

| * user\_id: long integer * profile : *Profile* Object * email : string * password : string * security\_questions: Hashmap of question strings and answer strings * feed: *Feed* Object * messages : *Messages* Object * gave\_left\_swipe\_list : list of long integers (*user\_id*) * gave\_right\_swipe : list of long integers (*user\_id*) * match\_list : list of long integers (*user\_id*) * unmatched\_list : list of long integers (*user\_id*) * visible : boolean * activity: integer | * create\_account() * prompt\_login() * verify\_login() * reset\_password() * prompt\_security\_questions() * verify\_security\_questions() * delete\_account() * open\_feed() * open\_messages() * unmatch\_user() * report\_user() |
| --- | --- |

* **Profile**

| * first\_name : string * last\_name : string * birthday : react-datetime * age : integer * gender : list of strings * year\_in\_school : integer * survey : HashMap * my\_items : list of strings | * conduct\_survey() * edit\_profile() * change\_picture() |
| --- | --- |

* **Feed**

| * user\_queue : list of long integers (*user\_ids*) * current\_user: long integer (*user\_id)* | * swipe\_right() * swipe\_left() * report\_user() * new\_match() * dequeue\_profile() * enqueue\_profile() ~ machine learning |
| --- | --- |

* **Messages**

| * message\_list : list of *Conversation* objects | * start\_conversation() * delete\_conversation() * unmatch\_user() * report\_user() |
| --- | --- |

* **Conversation**

| * match : long integer (*user\_id)* * text\_list : list of *Text* objects * checklist : *Checklist* object | * write\_text() * send\_text() * delete\_text() * unmatch\_user() * report\_user() |
| --- | --- |

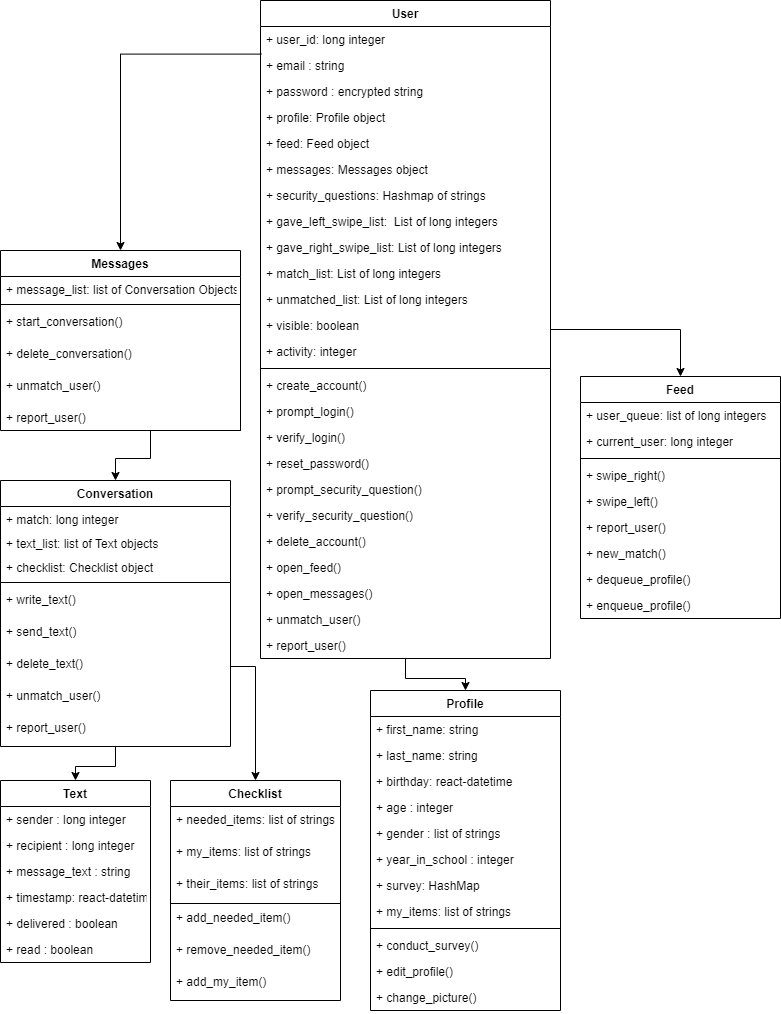
* **Text**

| * sender : long integer (*user\_id)* * recipient : long integer (*user\_id)* * message\_text : string * timestamp: react-datetime * delivered: boolean * read: boolean |  |
| --- | --- |

* **Checklist**

| * needed\_items : list of strings * my\_items : list of strings * their\_items : list of strings | * add\_needed\_item() * remove\_needed\_item() * add\_my\_item() |
| --- | --- |

**Class Diagram**



(b) Describe the classes and interactions between the classes.

**User class**

* The user class holds all the crucial data for owning an account, being the user\_id, email, password, and security questions.
* The user class will interact with its profile object on a regular basis to gather important information about the user’s profile. The information in the profile object is crucial to determining which users pop up in the feed.
* The user class holds a feed object, which will include all the potential matches that the user will see. The feed object will utilize all the other user’s profiles to determine if the user is likely to match with another as roommates before adding them to the user queue the user will view in the feed.
* The user class holds the messages object which will house all the conversations with matches. There will be conversations that have started as well as a list of matches that users can start a conversation with.
* The user class also holds a list of user\_ids that were swiped right on, swiped left on, and matched with. There will also be a list of user\_ids that were unmatched with or reported.
* The user class holds a boolean called visible that determines if the user wants to be seen by other users and have access to swiping in their feed.
* The user class holds a value called activity to show the number of minutes that the user was active on the app in the last 7 days. This is to prevent users from being seen in other feeds when they are not active themselves.
* The user class directly interacts with the profile, feed, and messages objects, which will be the main tabs in the mobile application.

**Profile class**

* The profile class holds the basic information about the user such as: first and last name, age and date of birth, gender, and year in school. Some of this data will be displayed in the feed for other users and some is used in the background to better determine potential matches.
* The profile class contains the survey data that is crucial to the use of the machine learning algorithm. The survey data will be used to originally determine potential matches until there is enough swiping data to utilize machine learning. The feed object will utilize swiping habits and the survey data to determine who the user is most likely to match with based on the survey information of the other users.
* The profile class will allow users to modify their profile by changing data that was previously stored in the database.

**Feed class**

* The feed class will contain a user\_queue that will consist of between 10-20 user\_ids at a time. These users will be enqueued based on a machine learning algorithm that will consider previous swipes and survey data from the user and other users on the application.
* The feed class will temporarily store the user\_id of the currently displayed user so that the user can more easily interact with the features of the feed and the other user’s profile.
* The feed class will respond to left swipes by adding the current user\_id to the left swipe list and move on to the next user.
* The feed class will respond to right swipes by first adding the current user\_id to the right swipe list. Then it will check if the other user has also swiped right, and if so, add the current user\_id to the match\_list as well. Both lists are stored in the User object.
* The feed class allows users to report the user that they are viewing if there is a violation of any policy or there is inappropriate content in the profile.

**Messages class**

* The messages class will utilize the match\_list that is stored in the User object to list out all the matches that have been made. Some of these matches may also correspond with a conversation object, and some may not. Matches that do not correspond with a conversation object can have a conversation started with them.
* The messages class also allows users to report or unmatch other users. If a match is being inappropriate in the conversation, then they can be reported, which leads to further investigation. If a match is simply not going to work out as a roommate, then they can be unmatched, which has no consequences with the application policies.
* The messages class allows conversations to be deleted if need be.

**Conversation class**

* The conversation class will show a one to one conversation with a matched user, called match. This allows matches to further discuss with each other before committing to being roommates.
* The conversation class will most importantly hold a list of Text objects, called text\_list, which can be sent by either user and received by the other. Text objects can also be deleted from the text\_list if necessary.
* The conversation class, like the Messages class, allows users to report or unmatch other users. If a match is being inappropriate in the conversation, then they can be reported, which leads to further investigation. If a match is simply not going to work out as a roommate, then they can be unmatched, which has no consequences with the application policies.

**Text class**

* The text class will hold the basic data that makes up a single text that is sent, including the sender, recipient, the message\_text itself, and other information like, delivered and read receipts, as well as a timestamp.
* The text class will update the status of the delivered and read receipts as messages are delivered and as the matched user opens the conversation, respectively.

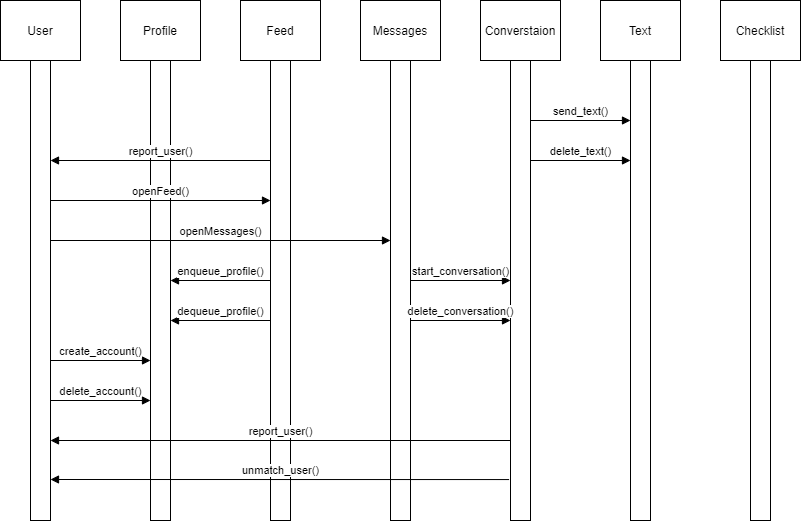
**Checklist class**

* The checklist class starts with a list of needed\_items for the room in which the matches would be sharing if they decided to be roommates.
* The checklist class will check off the needed items based on the list of items that each user already has, named my\_items and their\_items.
* The checklist class allows users to decide if they no longer need an item at all or if they decide there is an additional item that they would need for the room.

(c) Add sequence diagrams for different activities in the system, which will be helpful at the later stages of your project.

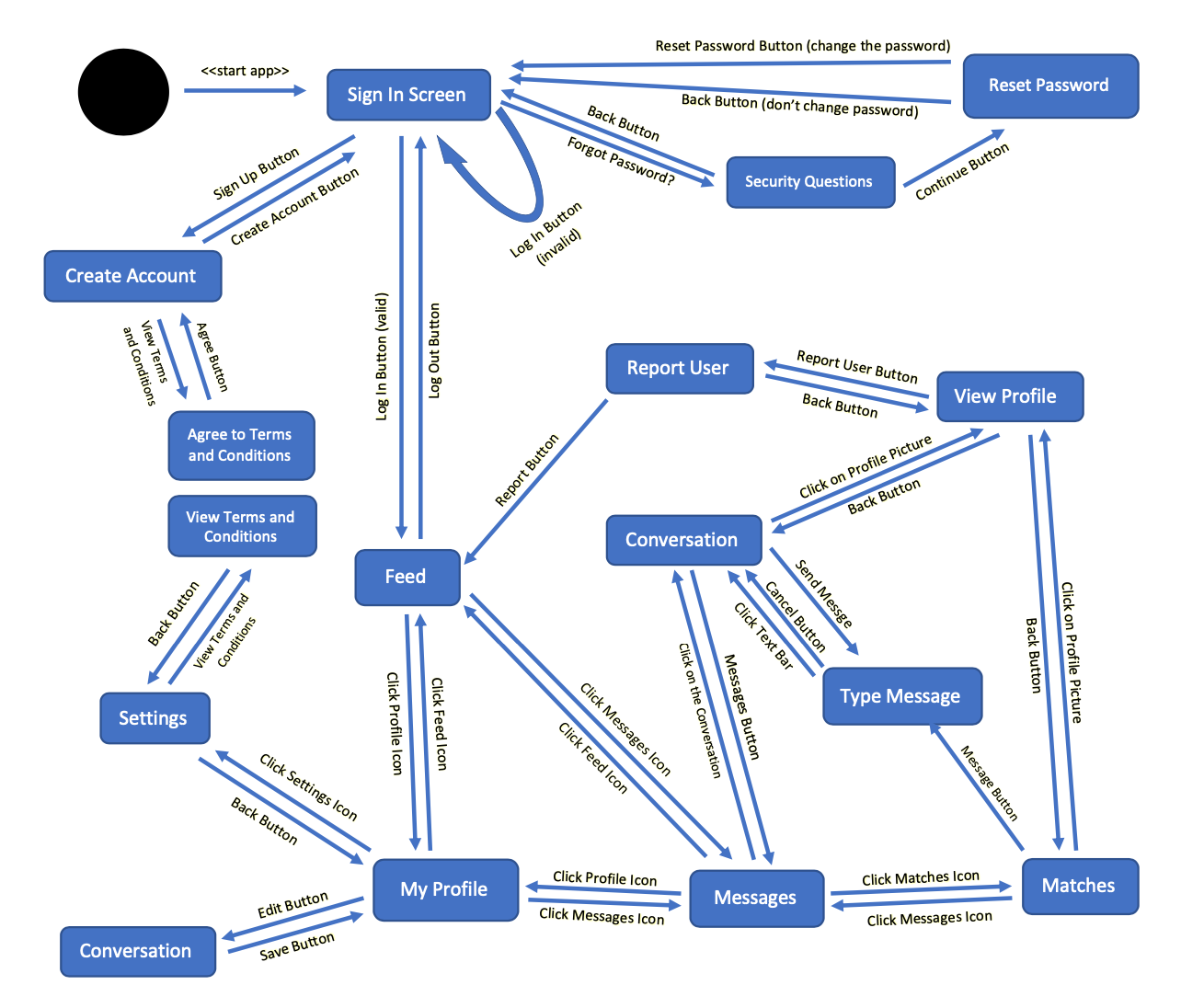
1. Login
   1. Prompt email and password
   2. User presses submit
   3. Check email and password sign in with the database
   4. Show success or failure
2. Swipe through feed
   1. Machine learning prepares a queue in the background prior to opening feed
   2. User opens their feed with a full queue of 10 users
   3. User swipes left or right
   4. The user that was swiped on is placed in a list
   5. The next users data is retrieved from the database
   6. When the queue has size less than 10 users, the machine learning algorithm will enqueue users until the user queue becomes full.
3. Start or Open a Conversation
   1. User opens the conversation tab
   2. A list of open conversations appears, as well as a list of people you can start conversations with
   3. User selects conversation to open/start
4. Send a message
   1. In conversation user can view previous messages and timestamps
   2. User types message and taps send
   3. Message is sent in real time to other user
   4. The sending user will see when the message can be seen by the recipient
   5. The sending user will see when the recipient opens the conversation and reads the message

**Sequence Diagram**



(d) If necessary, try to also include activity diagrams (or state diagrams) and UI mockups.

**State Diagram**

****

**Total: 14 points (14% of your project grade)**