# CS307 – Software Engineering Design Document

## HiLingual

Garrett Davidson
Noah Maxey
Nate Ohlson
Joseph Savastano
Riley Shaw
Vincent Zhang

# Table of Contents

# 1. Purpose

There are thousands of languages spoken across the world. Every day, millions of people are trying to learn to speak a new one. Whether it is for school, work, or personal interests, there are always people trying to learn. However, what most language courses fail to capture is that language is a social construct, and thus, to be taught properly, requires an appropriate social context. There have been many different attempts to make language learning more social, like Memrise and Duolingo where you can challenge your friends, or like LiveMocha where you can chat with and have your lessons graded by native speakers. However, all of them focus on the users' intrinsic motivation to simply learn the language. We plan to solve this by making language learning the natural byproduct of building relationships with other people, the same way a native language is learned.

## 1.1 Functional:

1. As a user, I would like to create an account.
2. As a user, I would like to login to an account.
3. As a user, I would like to find users who speak the language I am trying to learn.
4. As a user, I would like to choose users who speak the language I am learning.
5. As a user, I would like to search for specific users by name.
6. As a user, I would like to send messages to other users in real time.
7. As a user, I would like to correct messages sent by other users.
8. As a user, I would like to be able to translate messages that a user sent me.
9. As a user, I would like to send and receive voice messages.
10. As a user, I would like to create flashcards for vocabulary words.
11. As a user, I would like to review flashcards.
12. As a user, I would like to maintain multiple flash card sets.
13. As a user, I would like to set my personal information.
14. As a user, I would like to edit my personal information.

15. As a user, I would like to be able to report spam.
16. As a user, I would like to block specific users.
17. As a user, I would like to have a recent match list.
18. As a user, I would like to change my privacy settings.
19. As a user, I would like to change my password.
20. As a user, I would like to login through Facebook/Google.
21. As a user, I would like to be able to pay to increase my daily match limit.
22. As a user, I would like to be able to select my interests to find better matches (if time allows).
23. As a user, I would like to match with people with whom I share similar hobbies or interests (if time allows).
24. As a user, I would like to be able to pay to increase my daily message translation limit (if time allows).
25. As a user, I would like to be able to pay to increase my daily match limit (if time allows).
26. As a developer, I would like to see insights regarding how much interaction users have with each of their matches, to be able to improve the algorithm (if time allows).

## 1.2 Non-Functional:

1. We must have the app work on an iOS 9+ device.
2. App layout must support iPhone 5, 6 and 6+ screen sizes.
3. Interface should be easy to use and intuitive to navigate.
4. Messaging must be as fast and reliable as possible.
5. Personal/account information must be kept secure.
6. App-server communications must be done securely.
7. App must be resilient to network connectivity disruptions.
8. App and Server must be able to support all target languages.
9. Server must be able to authenticate users via OAuth 2.0.
10. Server must be able to integrate with various support APIs (translation, etc).

11. Server must support a large number of concurrent users.
12. Platform must be able to be run locally for testing/development.
13. Database must be fast and redundant (if time allows).
14. Platform must support automatic staging and deployment (if times allows).
15. Server must provide a maintenance/administration panel (if time allows).
16. App must have an algorithm to match users based on language abilities and shared interests (if time allows).

# 2. Design Outline

1. **Client :**
    a. The client will facilitate the interface with the user.
    b. The client will display conversations and language keyboards.
    c. The client will display other users' profiles and search options.
    d. The client will allow the user to change their profile information and chat settings.
2. **Server :**
    a. The server will provide an interface between the user and the database.
    b. The server will always be accessible from the client.
    c. All back-end calculations for the matching algorithm will occur on the server.
3. **Database :**
    a. The database will store all user information and messaging history.
    b. Typical data would include; usernames, passwords, messages, delete flags, edits, and user profiles.

## 2.1 High Level Overview of the System:

At the core of our project is an iOS application that will be the only interface for the users to the HiLingual platform. We are going to use the client-server model where the server can provide our services to many attached clients simultaneously, facilitating the

transfer of messages and storing user information, among other features. The client must always be able to see the server, as it is required for all functions on the app.

**Figure 2.2 - User Matching**

The user will request for a possible match from the server. The server will generate the list of matches through the database. The generated list will be based off of the user's information and language choices. Once completed the list will be pulled from the database and sent to the user as matches. The user can then choose a match through the list.

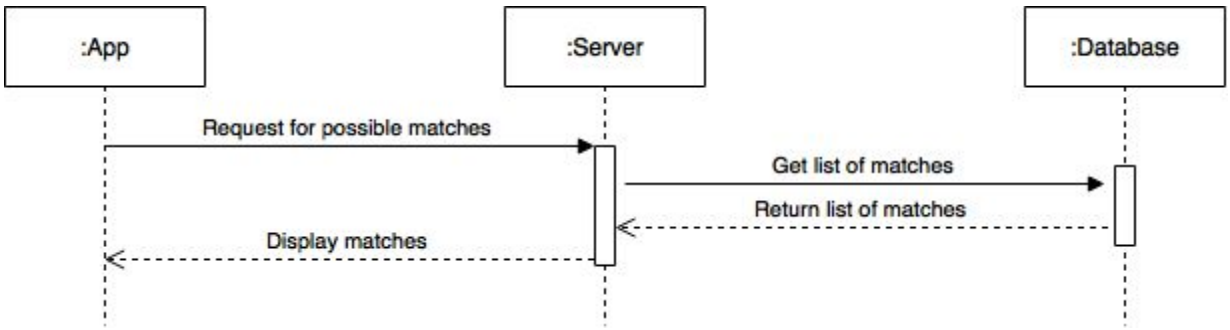


**Figure 2.3 - Sign Up**

This figure describes the process of signing up as a new user to the system. Authentication is done completely using the OAuth 2.0 standard. The server will generate a new UUID for each user and then the client will respond with a username that will correspond to the UUID.

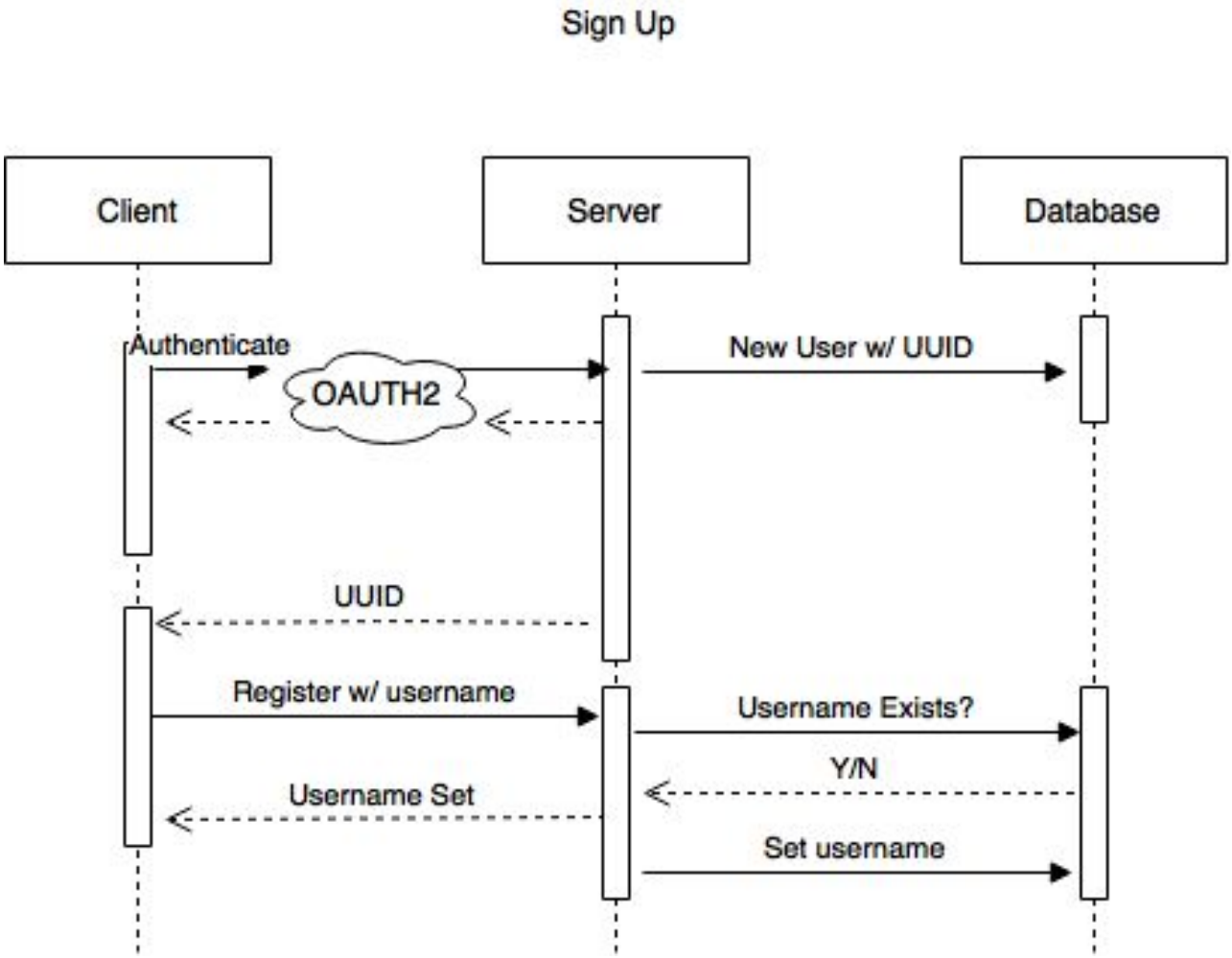
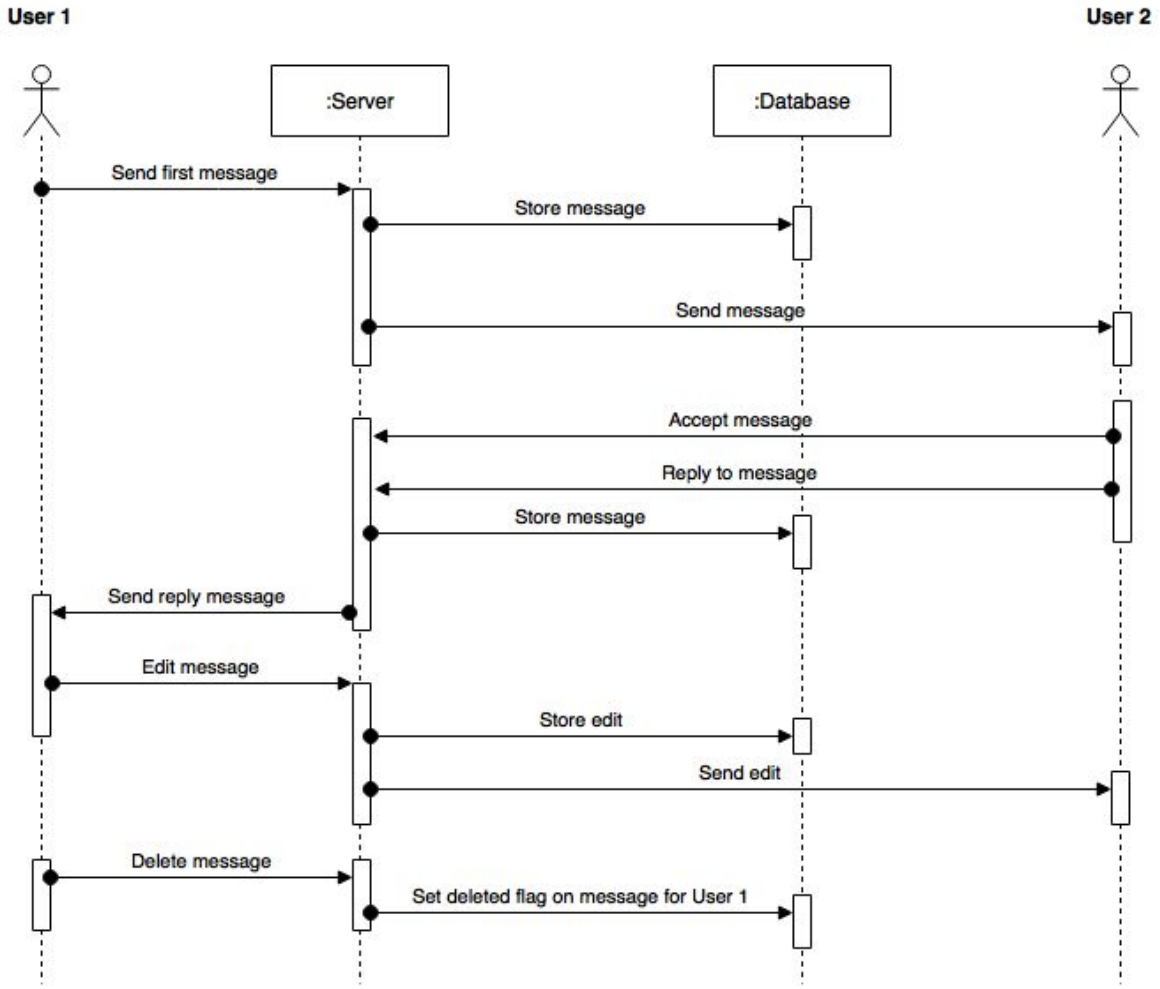



**Figure 2.4 - Chat Flow**

A user initiates a new conversation with another user. The first message is sent to the server, which stores it in the database and sends the message to the new user. The new user can choose whether or not to accept the contact request. Once accepted, he/she can reply to the message or edit the message. If the user chooses to edit the message, the edit is sent to the server which stores it in the database. If the user

deletes a message it deletes the message only for that user and sets a flag on the message object, so the database knows the message is deleted for that user.



# 3. Design Issues

- **Issue 1**: How should users be matched
    - Option A. Each user will decide if they would like to be matched.
    - Option B. Users can simply send messages to other users regardless of being matched
    - Option C. Have a profile option to allow messages from all users or only in accepted conversations.

  *Decision*: Option C – Users can decide whether or not to accept a conversation from another person, but users can send a conversation request to any other user.
- **Issue 2**:  Should users have a friend's list
    - Option A. Users have a distinct friend's list
    - Option B. There is no distinction between users who have been messaged and friends

  *Decision*: Option B – A friends list would not make sense in the context of this app because there is no distinction between friends and people you have conversed with.
- **Issue 3**: Should users have a limit to the amount of people they can view in the find tab
    - Option A. Yes, users should have a daily limit and they cannot receive more
    - Option B. Yes, users should have a daily limit and can pay for more
    - Option C. No, users should have an unlimited amount

  *Decision*: Option B – This prevents abuse and also allows for us to monetize the app.
- **Issue 4**: How messages are stored and deleted
    - Option A. Store messages just on device for each user.
    - Option B. Stored messages on device and server are synced for each user.

*Decision*: Option B – Allows the user to be able to access history across devices

- **Issue 5**: How should we moderate user content
    - Option A. Unmoderated
    - Option B. Specific users are considered moderators and allowed to police user content
    - Option C. Users are able to report inappropriate content and block other users

  *Decision*: Option C – For simplicity we decided to allow users to block any user they don't wish to be in contact with. Also, if it feels like a user account is spamming you can report them to the administrator.
- **Issue 6**: Display Name
    - Option A. Require every users' first and last name to be displayed.
    - Option B. Only show users' first name and have the user decide if and when to display their last name.
    - Option C. Should a username/displayname only be used.

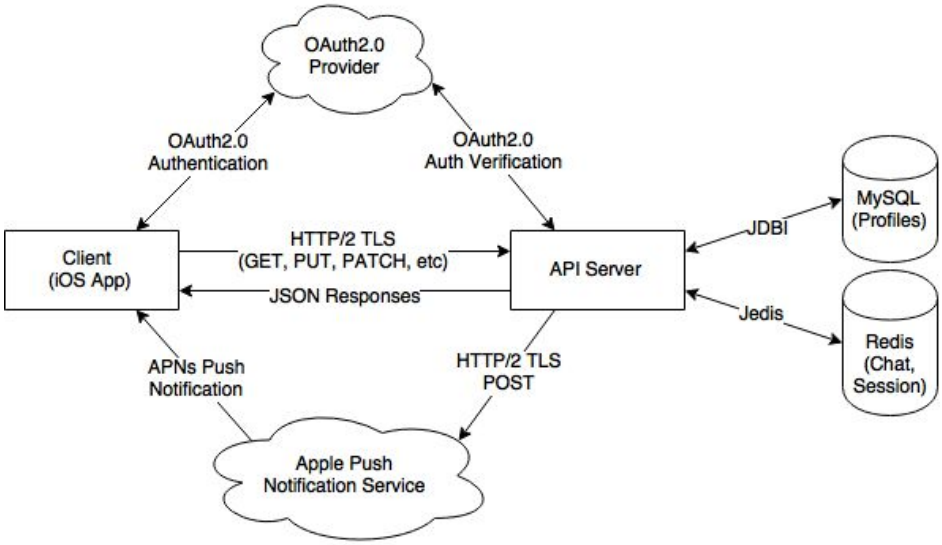
  *Decision*: Option C – We decided to show only a username or display because disclosing a user's full name could be considered a privacy concern and the paradigm for first names changes across cultures
- **Issue 7**: Should gender be a field used for matching users.
    - Option A. No, gender should not have an influence on the matching algorithm
    - Option B. Yes, the user should be able to choose what gender they would like to be matched with.

  *Decision*. Option A – No. Gender should not have influence in the matching algorithm, but users should have the choice to display gender in case they want to search for a specific gender out of our unbiased search results.

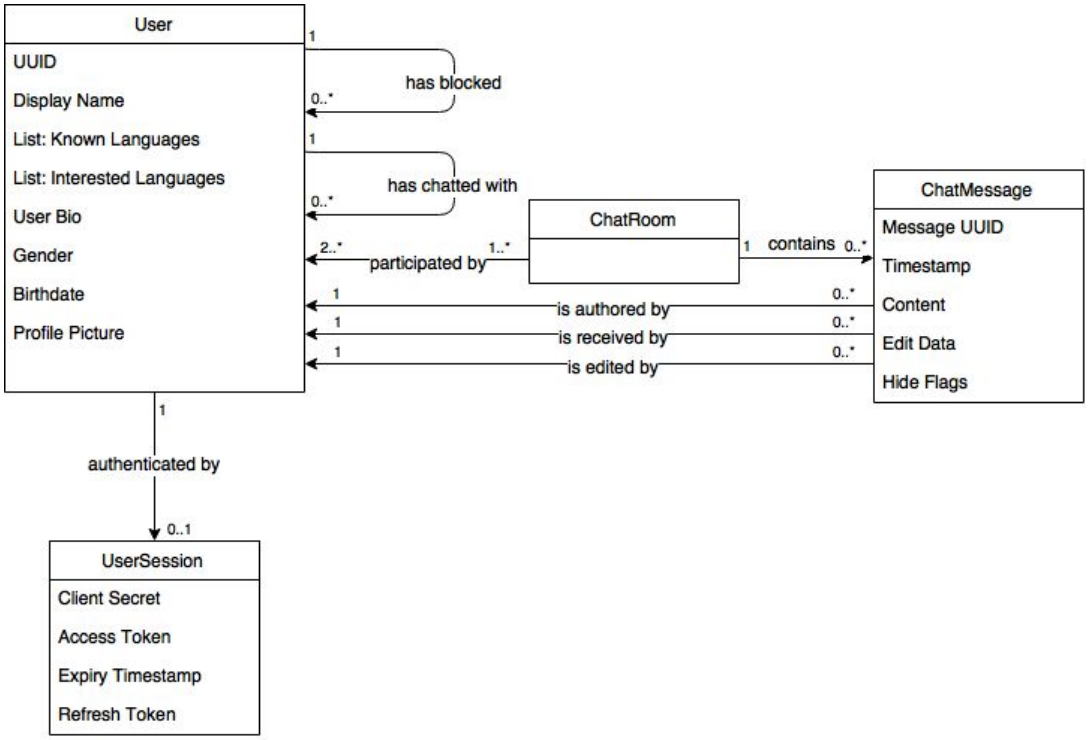
# 4. Design Details

## 4.1 - Class Design

### Figure 4.1.1 - System Overview




### Figure 4.1.2 - Data Model and Relations



## 4.2 - Description of Classes and Models

User:

- Class that contains all information regarding the user
- Authenticated by the User Session class to grant access to communication with the server
- User has the ability to add other Users to a blocked list to hide all communications with them

User Session:

- Class holds the information required for authenticating users
- User Session is created by a User object when an attempt to be authenticated is made and is kept for the duration of the session

Chat Room:

- The Chat Room object is dedicated to holding a list of Chat Messages and a list of participating Users

Chat Message:

- Class that contains the information needed when a User object has sent a message to a Chat Room object
- A Chat Message object can be edited, received and authored by a User object

## 4.3 Mock Diagrams for UI

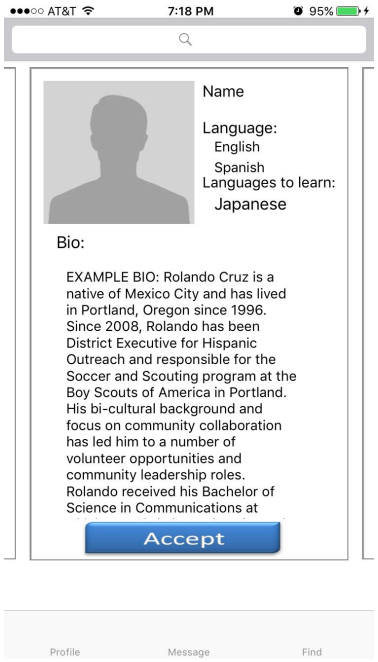



Figure 4.31 - User Profile Page

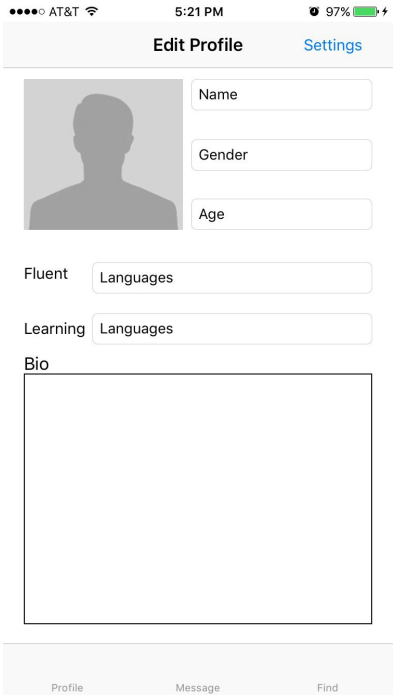



Figure 4.3.2 - User Profile Edit Page

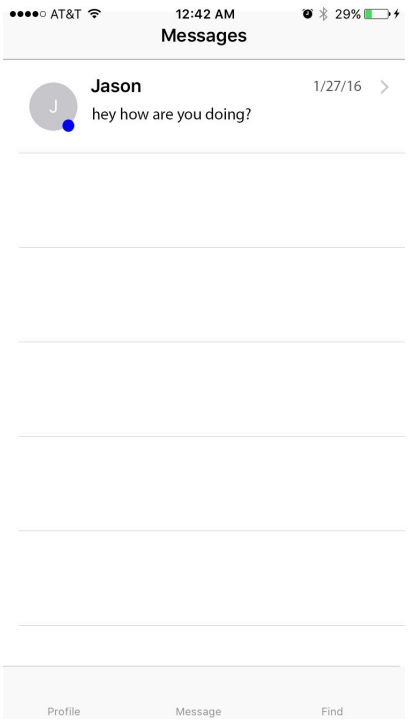



Figure 4.3.3 - User Message Table

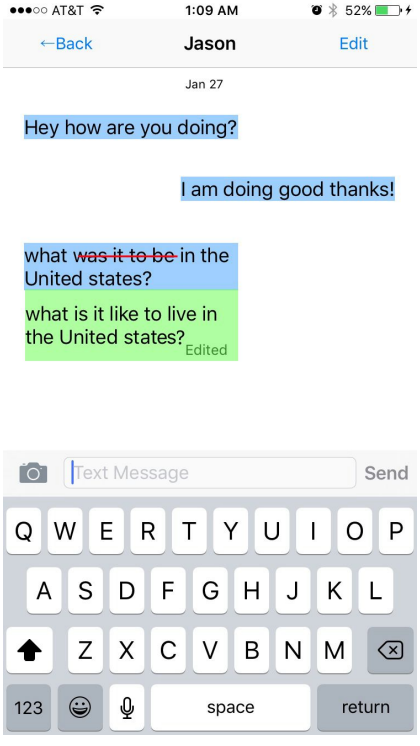



Figure 4.3.4 - User Messaging Page