CS-370

Requirement Analysis

Chris Lapp, Jacques Sarraffe, Adam Mena, Jackon Feinstein

SendR

Overview of the Problem:

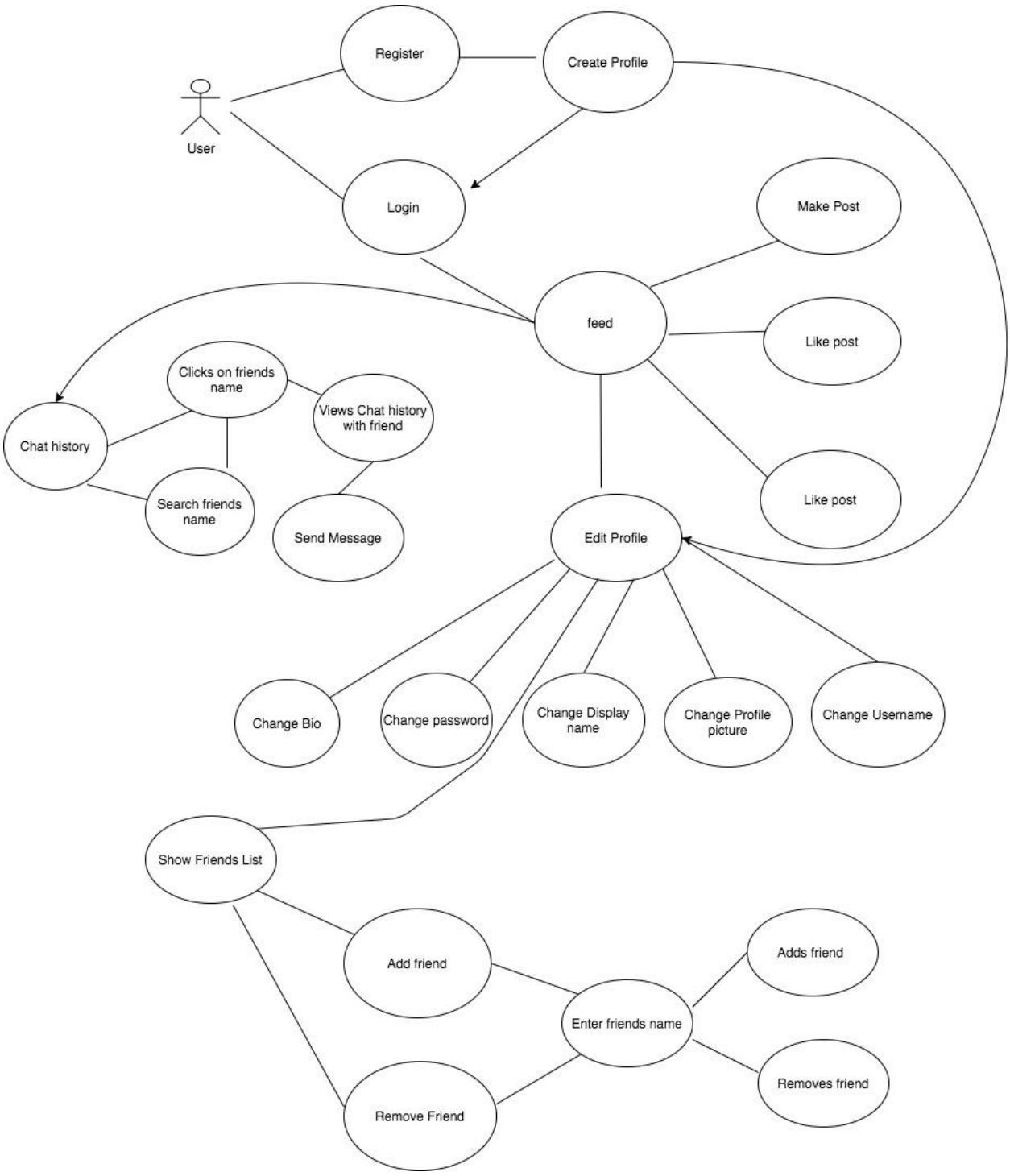
Instant messaging (IM) is a process that allows users to send and receive messages over the internet or a cellular network. Many of the current methods of IM rely on large packets of data about the user to be sent across a network. The problem with packets is that they usually require a lot of information to be delivered for even the simplest of messages. Facebook, for example, requires a large amount of data to be sent before any of the functions of the app can become usable. For example, Snapchat and Facebook both load their sponsored content before they actually load any user content.

Phone service providers offer instant messaging through text messaging, but this messaging service costs money and is not always reliable. Text messaging does have an upside though, it is completely standardized, and any customer with a cell phone capable of sending or receiving texts can send or receive texts. This is ideal but covers such a broad range of phones that it must be kept simple as to not alienate customers who choose basic phone options.

Overview of the proposed solution:

SendR is a free lightweight social networking application that lets users message their friends with the smallest amount of data possible. The app itself is not very large, so it should be able to be installed and run on any IOS or Android device. SendR lets users add their friends, message their friends, and post to a feed that can be seen and “liked” by all their friends. Users will be able to control their profile, friends list, and their feed. To pay for server time, we plan on charging users a very small amount on a subscription model.

Environment and system model



Use Cases: Register //Chris

Steps: 4

Actor actions:	System response
1. User selects register on the launch reason	2. System responds with the profile editing screen for a blank profile
3. User inputs the necessary information and presses submit	4. System saves the information and returns the user to the launch screen

Use Cases: Login //Chris

Steps: 5

Actor actions:	System response
1. User selects login from the launch screen	2. System responds with a dialog with spaces for username and password
	3. System checks username against database, if username is not found, prompts user to re-enter username
	4. System checks password against database, if password is not found, prompts user to re-enter password
	5. System moves user from launch screen to the feed page

Use Cases: Sending a message //Chris

Steps: 12

Actor Actions	**System responses**
1. User views the chat history	2. System displays users the user has messaged
3.User selects a user, or selects “search friends”	4.If a user selects search friends, system responds with a dialog

5. (OPTIONAL) User enters friend's name into search dialog	6.(OPTIONAL) system searches user's friends for username, if not found, prompts user to re-enter username. If found, prompts the user to select the user
7.(OPTIONAL) User selects friend from search dialog	8. System opens a dialog where the user can input text
9. User inputs message	
10. User selects "send message"	11. System saves and delivers message
	12. System notifies other user that they received a message

Use Cases: Replying to a message //Jacques

Steps: 7

Actor Actions	**System responses**
1. User views the chat history	2. System displays users who have messaged user by highlighting their name
3. User selects user they want to reply to	4. System opens a dialog where user can input text
4. User inputs message	
5. User selects "send message"	6. System saves and delivers message
	7. System notifies other user that they received a message

Use Cases: Posting to the feed //Jacques

Steps: 6

Actor Actions	**System responses**
1. User views the feed	
2. User selects "post to the feed"	3. System opens a dialog
3. User inputs message	
5. User selects "post to feed"	6. System saves and posts message to

	the feed

Use Cases: Commenting on a Post on the feed. //Jacques
Steps: 7

Actor Actions **System responses**

1. User views the feed	
2. User selects a post on the feed	
3. User selects “Comment”	4. System opens a dialog
5. User inputs message	
6. User selects “send message”	7. System saves and posts message as a sub post on the feed

Use Cases: Liking a post on the feed //Jackson
Steps: 4

Actor actions: System response

1. User views the feed	
2. User selects a post on the feed	
3. User selects “like”	4. System increments the like counter and bars user from liking the post

Use Case: Adding a friend //Jackson
Steps: 6

Actor actions: System response

1. User views the profile page	
2. User selects “view friends”	
3. User selects “add friend”	4. System opens a dialog
5. User inputs friend’s username	6. System searches database for user with matching username, and add’s user if the name is found. If the username is

	not found, system ask's user to re-input the username

Use Case : View Friends list //Adam
Steps:
Actor actions: System response

1. User views the profile page	
2. User selects "view friends"	3. System returns a list of the users friends

Quality requirements :

- Reliability: Our app must always be functional. If there is a problem with the backend server, we should notify users.
- Maintainability: Our app should be easy to maintain, with the code being easy to read and modify.
- Look and feel: Our app should be simple to learn and pick up. The user experience should come above all else.
- Performance: The app should run near perfectly, with no major delays in the low level tasks that the user performs.

Non-Functional Requirements

- Portability: Want the app to run on Android, and IOS, the two most popular operating systems for phones.
- We'll be using Firebase as our backend. It has automatic data synchronization, authentication services, messaging, file storage, analytics, and more.
- The frontend will be written in Javascript and for the UI React Native.