AUD
AMERICAN UNIVERSITY IN DUBAI

“Oice” Project Report

EECE457 - Mobile Applications

Spring 2014

Group Members:

Michael McCarthy	**1304023233**
Omar Hreirati	**1304023115**
Sharbel Dahlan	**1004018456**

Project Supervisor: **Dr. Hicham El Zabadani**

Due date: **April 27, 2014**

Contents

Abstract

(Sharbel)

The trend of social mobile applications is increasing due to the people's desire of having quick ways to interact. Even though text messaging apps have gained popularity due to their simplicity, applications that achieve communication in simpler ways have greater potential, such as those that are solely concentrated on voice messaging. Oice is voice messaging application that has the qualities of functionality and simplicity. Oice uses parse as a BaaS, which stores the list of users, the relations among them, and the messages that are sent between them. Due to time constraints, aside from the sophistication of implementing some of the requirements, the application was completed with some of the proposed functionality, including allowing user sign-up, storing users on the server, adding friends and creating friends relations, and recording and playing voice. The implementation of sending the voice messages to other users was not achieved due to inability to get the retrieve the voice file that was stored in a form to be uploaded on the server. More research is required to find a way to have audio files that are created on the devices to be sent to the server so that they can be transmitted between users.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

(Mike)

The project will be building a mobile application named oice, which is concentrated on voice chatting, or more specifically, a push-to-talk-like application. The main aspect of the app is its simplicity and ease of use. Oice allows you to send voice messages to your friends with just two taps easily and effectively. Your friends can listen to your messages instantly right after you finish recording them, and here is the cool part: your messages will be wiped out just after your friends finished listening to it! So no one will be able to listen to it again and it won't be stored anywhere, not even on our servers! Forget about phone calls, voicemails, VOIP apps, or even texting apps. With oice you can easily send your voice. All you have to do is tap and hold the Record button, record your message, release the button, and simply choose who you want to share it with!

1.1 Motivation (Mike)

This application allows users to connect across a medium that was previously popular (6 years ago) in the USA called Sprint / Nextel Push-to-Talk. The premise of this app is that people will want to send voice messages to their friends and family and skip the need to text or take a photo. This application will derive and mimic Snapchat in usability, accessibility and function. Users will be able to record a message, send that message to users on their friends list, and have that message be listened to only once before it is deleted; all very similar in form to Snapchat. This application is niche in that there are currently no other providers that allow for a disap- pearing voice conversation. I believe as a user that there are many situations that recording a voice message is easier that writing out a full text message. Currently, my own situation living aboard and not being able to send a quick, easy, voice message to my parents is something that bothers me. I have to log onto skype or google hangouts, press a bunch of buttons to record, and then hope that my mom logs onto the web app to see my message. I believe there is a market place for a application that deals with this dilemma and I know that I would use something like this if it was available now. Centering the application on a walkie-talkie-like premise is unique and has not been successfully implemented before. This project will change that.

1.2 Description (Omar)

The following description can appear on the app page in the play store:

Oice is more than just a Walkie-Talkie!

oice allows you to send voice messages to your friends with just two taps easily and effectively. Your friends can listen to your messages instantly right after you finish recording them, and here is the cool part: your messages will be wiped out just after your friends finished listening to it! So no one will be able to listen to it again and it won't be stored anywhere, not even on our servers!

Forget about phone calls, voicemails, VOIP apps, or even texting apps. With oice you can easily send your voice. All you have to do is tap and hold the Record button, record your message, release the button, and simply choose who you want to share it with!

So hurry up! Join your friends, start talking and let them hear you.

Features:

- Oice is currently available on Google Play Store, and it is absolutely free with no hidden payments or subscriptions.
- No need to add your friends or accept invitations, oice scans your phone book and automatically import your friends that are using oice.
- The ability to block anyone even if you don't have them on your contacts, so no spam!
- Send your message anytime you want and it will be delivered to your friends once they are available and online?.

- Messages will be automatically removed from our server and all devices once your friends finish listening to it.
- The ability to broadcast messages to multiple friends?
- Your friend will get a push notification when you send a message, and you'll be notified when your message was heard.
- An indication for messages when they are sent.
- works on all connections including Wifi, 3G, and LTE*

*Note: Oice needs an internet connection so data charges may apply, check your phone carrier for more information.

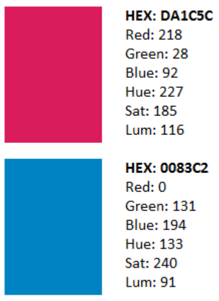
Chapter 2

Graphical User Interface Design (Sharbel)

Unlike computer programs, whose quality stems from the sophistication and functionality they provide, mobile applications need to have an inviting simple user interface, which will influence the decision of the users of whether the application is worth downloading or not. However, that does not mean the application should compromise its functionality, and this is the challenge when designing user interfaces. With this application, there was a balance sought between the functionality and the user interface. Section 2.1 describes the logo of the application and the general theme, while section 2.2 descries the graphical user interface of the application.

2.1 Logo Design and Theme

To give the application some unique character and distinguish it from other similar applications which have generaly green, blue, and orange colors dominant, it was given two main colors: The turqoise and the raspberry-red color. The HEX color codes of the raspberry-red and the turqoise are DA1C5C and 0083C2, respectively. The RGB colors of each are shown in Figure 2.1.



HEX: DA1C5C
Red: 218
Green: 28
Blue: 92
Hue: 227
Sat: 185
Lum: 116

HEX: 0083C2
Red: 0
Green: 131
Blue: 194
Hue: 133
Sat: 240
Lum: 91

FIGURE 2.1: The color codes of the oice application

The logo of the application is shown in Figure 2.3. Both the main colors of the app's theme were included in the logo. Since the name of the application means "Only Voice", the emphasis was made on the 'O' by giving it the different color than the other three characters of the name.

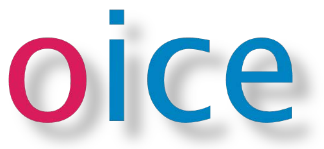


FIGURE 2.2: The main application logo

Figure **??** shows the icon of the application, which will also appear on the action bar of the application. The two overlapping chat bubbles signify a sent oice and a received oice.

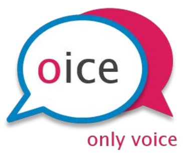


FIGURE 2.3: The icon of the application

2.2 User Interface design

In any good application, simplicity is what makes the users decide whether or not to continue using the app. With that kept in mind, the user interface of Oice was designed to be very simple - down to one main page. The reason for that is to eliminate the need to navigate through different pages to interact with other users and send or receive voice messages. Hence, the main page contains the list of contacts to which oices were sent and received, as shown in Figure 2.4.

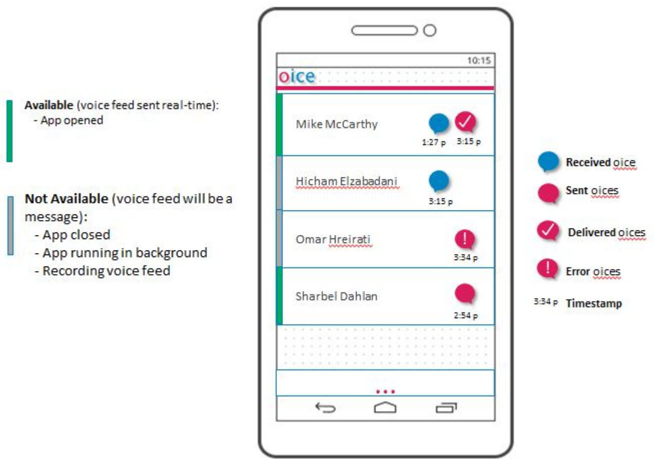


FIGURE 2.4: The GUI of the main page of the application showing the contacts list with the most recent interactions on top. The legend of the icons is shown on both sides of the screen.

In Figure 2.4 Two main icons indicate sent and recieved oices are shown. The sent icons have more indicators, suchas if a message was delivered or an error has occurred. As apparent, the complete list of the users can be expanded from the three dots on the bottom, which, if tapped, will make the list expand as shown in Figure 2.5.

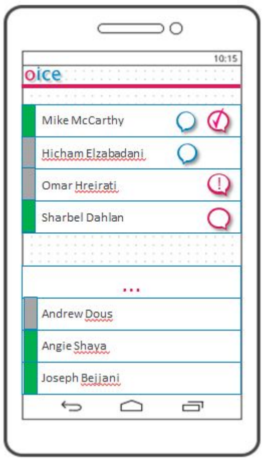


FIGURE 2.5: The rest of the contacts extended by expanding the list from the bottom of the window

Figures 2.6 and 2.7 show the GUIs related to the sign-up pages of the app.

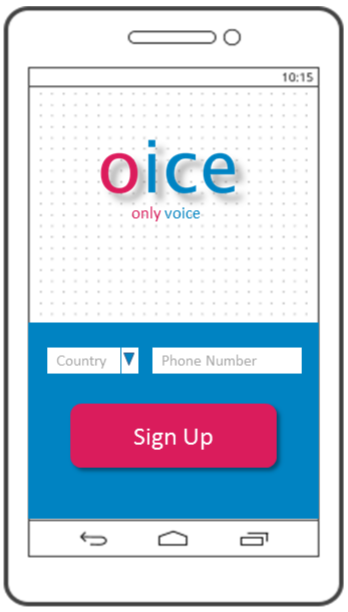


FIGURE 2.6: Sign-up page

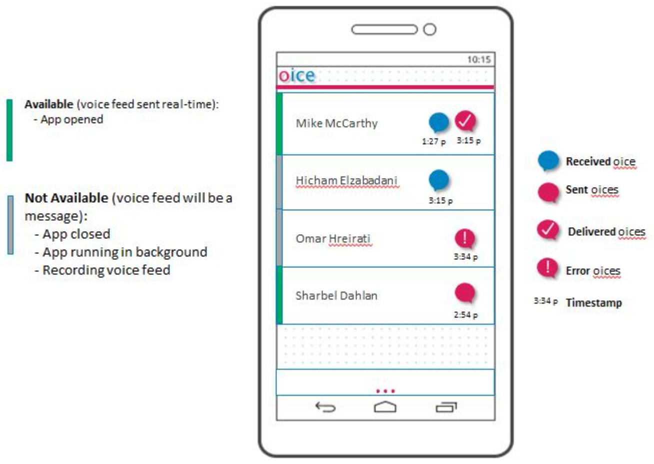


FIGURE 2.7: Sign-up page after selecting the country and number

2.3 The Actual App GUI

Figures 2.8 through 2.14 show the Final GUIs implemented. Notice that there are some differences due to changes in implementation of different parts.

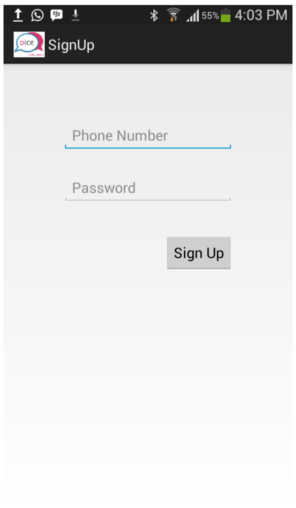


FIGURE 2.8: The sign up page that appears only for the first time the app is launched

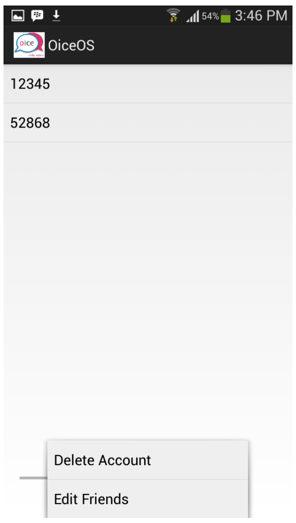
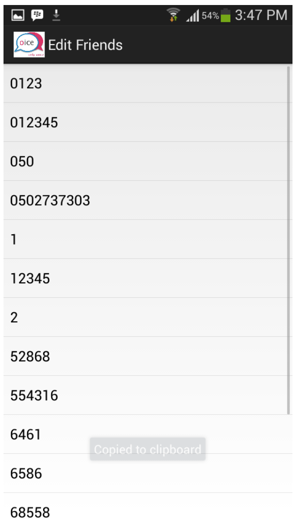


FIGURE 2.9: The menu offering a delete account and edit friends to add friends


3:47 PM

Edit Friends

0123

012345

050

0502737303

1

12345

2

52868

554316

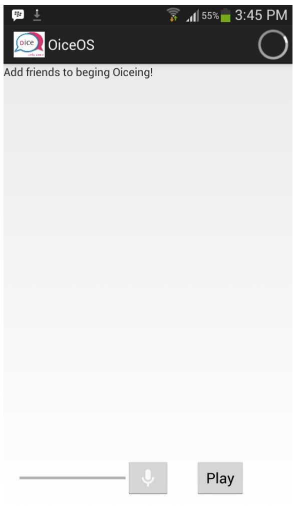
6461

6586

68558

Copied to clipboard

FIGURE 2.10: The edit friends activity which allows to add users that are on the parse server.


3:45 PM

OiceOS

Add friends to beging Oiceing!

Play

FIGURE 2.11: Loading the contacts at the main page

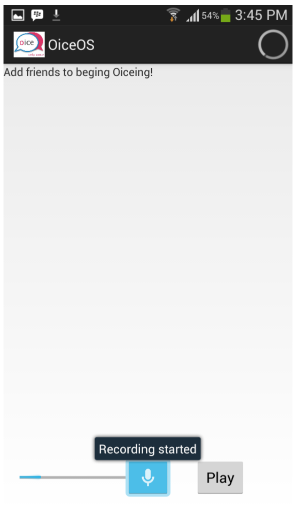


FIGURE 2.12: The recording button is tapped down. Notice the toast and the progress bar loading

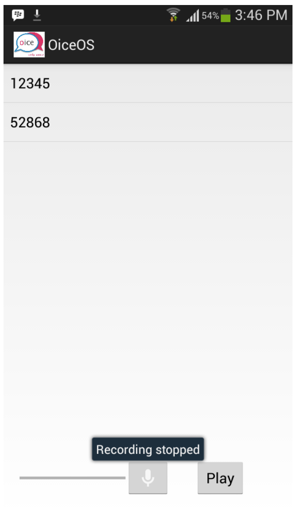


FIGURE 2.13: The recording stopped after releasing the record button. Notice the toast and the progress bar that stopped loading

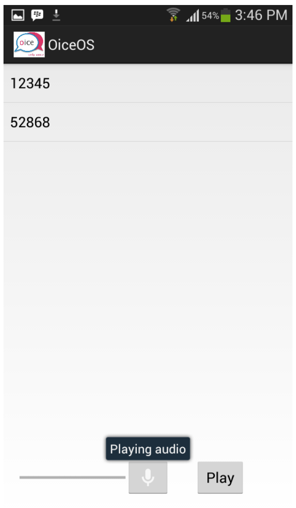


FIGURE 2.14: The play button is tapped to listen to the recording

Chapter 3

Backend Design (Sharbel)

In order to make the application to function, there needs to be a server to carry all the back-end work. For this project, the Backend-as-a-service (BaaS) tool that is used is provided by `Parse.com`. Each app created on Parse has its own application ID and client key that are applied to the SDK install [1]. A guide on several useful parts of the project, including objects, queries, files, and users, are found on Parse.

A well designed system is programmed such that there are clearly defined interfaces between components that let each component look and act like a black box for the other parts of the system. Figure 3.1 shows an object from parse called ParseUser, which is acting as this black box that communicates with the parse backend. The user information is passed as an input, the ParseUser Class communicates with the back end, and the output is a new ParseObject that has new user information.

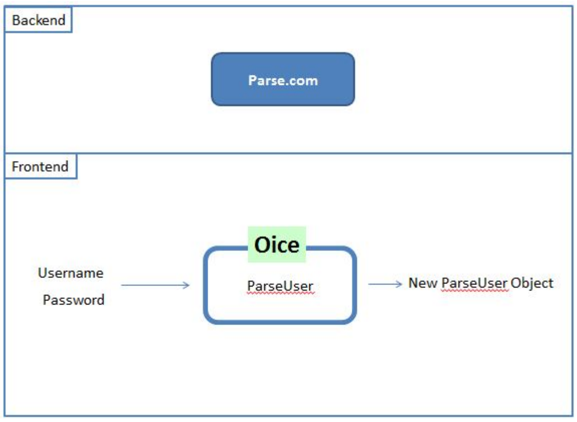


FIGURE 3.1: The interaction between parse and the user class.

There are actually two interfaces that work here. The first is the interface of the ParseUser object which includes the public methods and properties needed. The second interface is between the ParseUser class and the parse.com backend, which is known as the web interface since it is used over the web.

An example of an interaction with the parse object is calling the sign up method. Figure 3.2 shows what happens when the sign up method is called from parse.

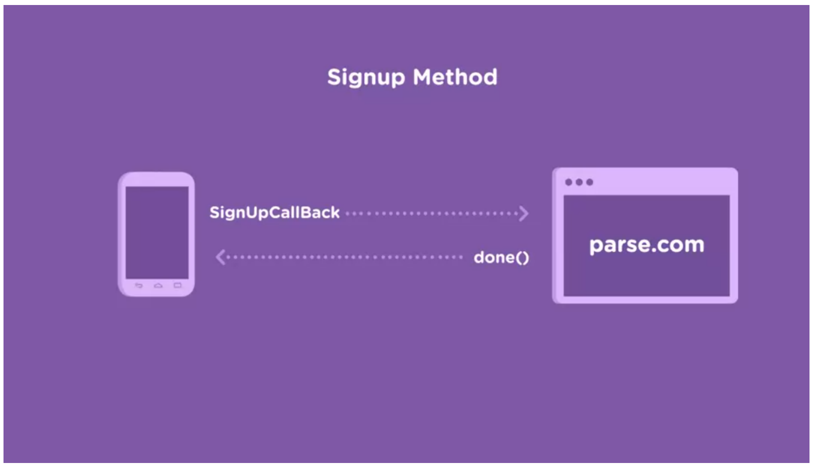


FIGURE 3.2: The sign up method on parse

The SignUpCallBack() method is an object defined in the parse SDK. As long as the correct SignUpCallBack object is given to parse, it will be able to use it. So, when the sign up operation finishes in the background, the done() method is called. The code written in the done method will be executed.

Another significant portion in which the parse backend is used is setting up a query for users. When a user is selected from the EditFriends activity (explained in Chapter 5) The activity is displayed.

The method that gets called every time an activity is displayed is the onResume() method, which is part of the activity's life cycle, shown in Figure 3.3.

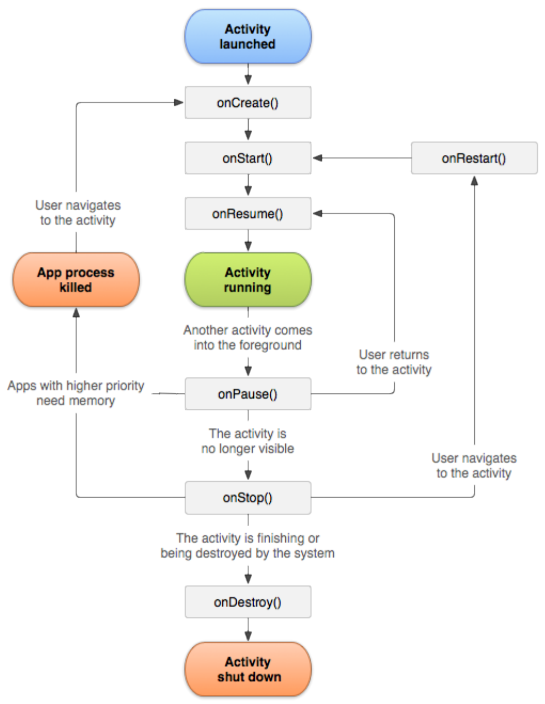

Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

FIGURE 3.3: The activity lifecycle

Mike

Starting the App

The main component of the app is the User. We want to create a page for the user to log in. We will use a username, password and email address. To help get an idea for the layout of our screen I used a website called mobilepatterns.com. Our users should only have to sign up once so I want that to be a less obvious second choice. Most of the time our users will have the app running in the background, so I want the main screen to be the inbox screen. This will be the main activity eventually. Before that we need the log in and the sign up. We'll call this the LoginActivity. We set this as portrait only. Here it displays the username and password with a button below it to sign up – which hardly will ever be used. Next we create a new activity called SignUpActivity whose parent is LoginActivity. This is pretty much the same page as the LoginActivity except it takes an email address as well.

After these were set up I worked on the back button and navigation tabs. I went to the manifest and selected signupactivity and set the loginactivity as its parent class. I also needed to reorder the stack of the pages and make sure that when I hit the back button on the loginactivity page I wasn't directed to the inbox screen. To do this I added intent flags in the MainActivity.

Adding Users / Not Going Back to Log In Screen

First I created a parse.com account to handle my backend. Then I downloaded the sdk and dragged the jar file into my lib folder which has support files. Back in Eclipse I created a new java class called Oice/Ribbit application, which extends the Application. I then imported the initialize method, which gives the application id and client id for our backend. Then I added the internet permissions from the parse website into the manifest xml file. I also copy and pasted the analytics (which we wont use) into our MainActivity file.

The next step was creating a new user in parse. I did this in the sign up activity by creating a new user and setting the username, password and email. Below this I created flags to make sure that a duplicate email wasn't already used and to make sure the user fills out all the fields required to signup.

Next I read through parse documentation about login in in background. When ever you use a login session the user is cached on disk. So we can use the implementation of that in the main activity by calling the main user. If it is null we should call the login screen. Else we should go to the inbox since they have already signed in.

What if the user wants to log out. Then we call the log out class method from current user, which clears the cache. I added this option under the main.xml in menu as a tab option. I then updated the main activity to log out the user and push them back to the login in screen – erasing the cache of the user on the disk.

I also added a window feature progress bar to the login activity to give a spinning effect that runs in parallel as the user logs in and get verified on the back end by parse. I did this in the loginActivity class.

Tabs for Activity

Back when I created this I had tabs set up for the MainActivity to allow for swipe and inbox and friends. In mainactivity wecan see fragments. Next create a new class called sectionspageadapter. Then I cut and pasted in the part from main that handled with sectionpager adapter. Then changed the getCount method to 2 to display two tabs. Under getPageTitle I changed the name for the title to be “Inbox” and “Friends”. Then I created a new class called inboxfragment. Extended List fragment here since it’s a list. After that I went to main activity and copy and pasted the onCreateView activity to the inbox fragment. Next I added a list view in the xml layout and modified the code to reflect where an inboxed message would appear with name along side it and an icon. In main activity I also implemented the action bar and set item of the viewpager to the tab that was selected.

Friends

Back end we use a parse relation to relate friends. First need to go to optionitemselected to add a new friend. Then we need to add a new blank activity called EditFriendsActivity. Use a list view in this and change the screen orientation in the manifest to portrait. EditFriends class we need to add the onResume method. Here we use the parse query to find our list of parse users. I read the sdk for help on this one. I next created a new class called ParseConstants. Changed it to final and added a class name and field name. Then I used editfriends class to query for users in background once again using the parse documentation as a guide.

Now in editfriends we set the list view to allow multiple users to be selected for adding friends. Use the getListView method for this. Now we need to update the backend. In parse I looked at the parserelation class under objects for the documentation and followed this. I did this in the onlistitemclicked method in the EditFriends file. Next I declare a parseRelation by using parseuser. And then add the friendsRelation in the EditFriends class. We should now be able to see the friend relationships on the parse backend. I also added checkmarks once the list is loaded. In editfriends I added the listadapter method and addFriendCheckmarks which pulls the friend relation from the backend using the parse relation query. Here we can look through the database to find the user.

Displaying the friend list comes next. We do this in the FriendsFragment. We can steal a bunch of code from the editFriendsActivity like getting the user and creating a relationship. Then we should add a statement to display the message to the user if they are not in the system. We should also update the query to sort by username in alphabetical order. We use fragments so that both pieces can load in parallel while the app is open. I also added a delete method that can be found in the editFriendsActivity on listitemclicked. Here I call the remove method to remove the relationship. This is then implemented with parse on the backend through the remove method.

Sending Messages

Create a new activity called RecipientsActivity which is nearly the same as the friends-fragment. This will load all the users friends and let the user select the one/ones to message. We use a list view for the xml file to display the friends. We can copy a lot of the code from the friendsFragment. We query for the friend to bring up the list. Now we let the user check their friends using list item checked. No in mainactivity we add the intent recipient class. Start the activity and pass in the URI file. Next we add the send button. We put it in RecipientsActivity. Now we create an onlistitemclick for the button. Once again we need to define a new parse object for it to store the file. We call it a parseobject and create a new object in the recipientsactivity class. We should also update the parseconstants class. We next create the parseobject which is protected and add the message to the class. We have to convert the file to a byte array. For this we use a ParseFile object to store the message in the recipientactivity class. I also had to update the main activity to add the recipients. One of the things that I also did was check to make sure there is storage space available for the app to store the message on. There is a line that checks for this by looking for storage on the mobile phone. This all takes place in the recipientsactivity. In filehelper class we convert the file to a compressed version that helps with sending. With the recipientsactivity we add the message which is now compressed to the saveinbackground method.

Receiving Message

Retrieving the message is very similar to sending the message. OnResume is added in the main activity. We need to query the messages that have the same id as ourselves – the ones that are meant for us and not the entire data base of objects. We need to use the where equal to method in parse to find the id in the system that matches our user. We also need to update the some code in the constants class. We can copy a bunch of code from our friendsfragment and copy it into the main activity. Now we update the list layout with icons. We also create a new class called messageadapter that adapts our data from parse to use in the layout. After this I created a new activity called viewimageactivity. From our inbox fragment we go to the viewimage activity. In the inbox fragment we have our message that we create. This links back to the send feature and allows the image to upload to the back end.

Chapter 4

Workflow (Use-case Scenarios)

An important part of software development is organizing the workflow through use-case scenarios. Before jumping into the development of code, the use-cases were designed to ensure that the app contains a complete functionality according to the specified scope. Not only does it explain each function of the app, but also it makes the flow of the application clear to the developers. Initially, some workflow was chosen for the app, which will be explained below. Figure 4.1 shows the use-case diagram that was initially set for the oice app.

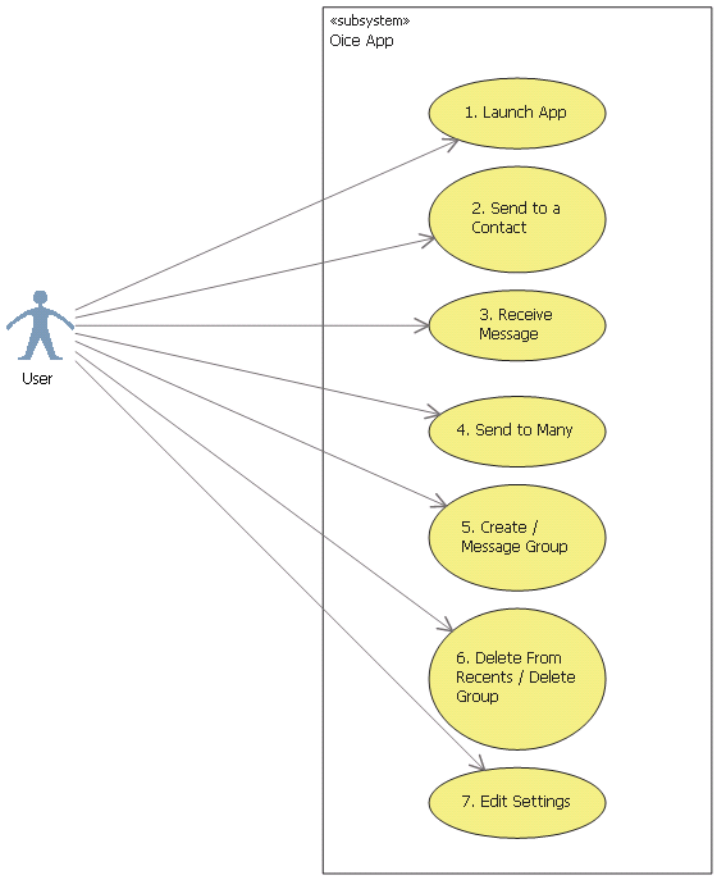


FIGURE 4.1: The Use-case diagram of the Oice app

4.1 Launch App (Mike)

Priority	High
Complexity	Medium
Time	1 Day
Actors	User
GUIs	GUI 2.4

Description

The user opens the application and will be directed to the main page. The application will access the phonebook and update the contact list in the application.

Pre-condition

The user does not have the app open. The app is either closed or running in the background.

Post-condition

The user is in the main page of the application GUI 2.4.

Steps

1. User taps on the app icon from the applications menu.
2. GUI 2.4 appears.

Exceptions

If in step 1 the user is launching the app for the first time:

1. The user will be directed to the sign-up page GUI 2.6. Then, the user chooses the country from the drop-down menu, enters the phone number in the adjacent text-box, and taps on the *Sign-up* button.

2. The user will be directed to the following page GUI 2.7 and will receive an SMS with the verification code which he/she should enter to the provided *verification code* text-box . Then the user taps on *Start Oice!* button and will be directed to the main page of the application GUI 2.4.

Error Conditions

If, in step 2 of the *Exceptions*, the user does not receive and SMS in 1 minute, a *Resend Verification Code* link will appear.

4.2 Send to a Contact (Sharbel)

Priority	High
Complexity	Hard
Time	3 Day
Actors	User
GUIs	GUI 2.4

Description

The user sends a voice feed to a contact from the list.

Pre-condition

The user is on the GUI 2.4 before sending the voice feed.

Post-condition

The user is on the GUI 2.4 after sending the voice feed.

Steps

1. On GUI 2.4 the user taps and holds on a contact. The contact's name gradually becomes highlighted, and the user starts recording the voice feed to be sent to the contact.
2. When the user releases the contact name, the voice feed or message is sent to the contact.
3. An icon will appear next to the contact's name signifying that the message has been sent. The number of sent messages will be indicated on the icon.

Exceptions

Error Conditions

4.3 Receive Message (Omar)

Priority	High
Complexity	Hard
Time	3 Day
Actors	User
GUIs	GUI 2.4

Description

A user receives a voice message by another user and can listen to it by a tap on the sender.

Pre-condition

User is on GUI 2.4 without message received.

Post-condition

User is on GUI 2.4 with message received.

Steps

1. User receives message and the blue bubble icon appears on the contact from which the message was sent.
2. The user taps on that contact.
3. The voice message is played.
4. The voice message disappears after it is played (the bule bubble icon will disappear).

Exceptions

The user clicks on any contact without a sent oice, and nothing appears.

Error Conditions

None

4.4 Send to Many (Omar)

Description

In this use case, the user will be able to send one voice feed to multiple users.

Priority	Low
Complexity	Easy
Time	1 Day
Actors	User
GUIs	-

Pre-condition

The main page is shown; displaying the contact list with no contact selected.

Post-condition

The main page is shown; displaying the contact list with no contact selected. (This page is shown after the multicast has been sent).

Steps

1. The user selects names to send the voice feed to by swiping the name right
2. The name that has been swiped right is now highlighted
3. Once the user has finished selecting names, he can now press and hold on any of the highlighted names to record the voice feed
4. The names will not be highlighted anymore; returning to the Post-Condition state

Exceptions

- If the user wants to unselect someone from the names, they must swipe left to un-highlight that name
- If the user wants to cancel his multicast before it is sent, they must hit the back button on the user interface of the phone

Error Conditions

- If there is a connection error it will display a message asking to be resent

4.5 Create / Message Group (Sharbel)

Priority	Low
Complexity	Hard
Time	3 Days
Actors	User
GUIs	-

Description

The user slides right on multiple contacts consecutively to select them, then tap on Create Group button. The application creates a group entry on the devices of all the group members, then the server will send a notification to the members to inform them that they were added to an Oice Group., the group will function as any other contact in the list.

Pre-condition

The user is in the main page of Oice, viewing the contacts list.

Post-condition

A group that includes multiple members is created, and will act as a normal contact.

Steps

1. The user slides on a contact entry to select it.
2. The selected entry will be highlighted to indicate that it is now selected.
3. The user repeats step 1 to select all the desired members.
4. When more than 1 contacts are selected a Create Group button will appear in the lower right corner of the screen.
5. When the user click on the button, Oice will create a group entry on all the members' devices and will notify all them that they have been added to an Oice Group.

Exceptions

After creating a group, it will act as a normal entry in the contacts list except when the user receive a new group message, the name of the sender of each message will be displayed for each message separately until it's heard.

Error Conditions

None

4.6 Delete From Recents / Delete Group (Mike)

Priority	Low
Complexity	Easy
Time	3 Day
Actors	User
GUIs	-

Description

In this use-case, the user will be able to delete a name/group from the position in their list on the home page, effectively sending them to the bottom, and refreshing the page to most recent.

Pre-condition

Home page displays names/groups in most recent order.

Post-condition

Home page displays names/groups in most recent order – with the previous selected entry appearing at the bottom.

Steps

1. The user swipes left to remove a name/group from the list.
2. The list is now refreshed to display a new home page, with the removed name at the bottom of the list. [1]

Exceptions

None

Error Conditions

None

[1] Users will not be able to undo their delete.

4.7 Change Settings (Omar)

Priority	Medium
Complexity	High
Time	4 Days
Actors	User
GUIs	-

Description

The user access Oice settings by clicking the Options button in his Android device, then tapping Settings from the available options. The available options in the settings page are the following:

- *About*: Includes some basic information about Oice including the logo, version, Help, FAQ, and a contact us link.
- *Invite friends*: The user will have the choice to send an invitation to a friend via SMS, or post an invitation to the user's Facebook or Twitter account.
- *Delete account*: This will enable the user to unlink his Oice account with his phone number to be able to add a new number instead.
- *Contacts Settings*: This will allow the user to edit their contacts list (delete, add, or rearrange), and will also include a block list which enables the user to block other users in Oice in order to stop receiving messages from them and prevent them from adding the user to a group.
- *Notifications*: this will include the basic notifications settings like choosing a sound for new message and new group message notifications. . . etc.
- *Network Status*: This will show the current network state of Oice which helps users to determine if they are connected to Oice's server or having connection issues. The status field will have 3 states: Connecting, Connected, and No Connection.

Pre-condition

The user is in the main page of Oice, viewing the contacts list.

Post-condition

The user is in the settings page where he/she can change Oice's settings.

Steps

1. The user clicks on the options button on their Android device.
2. The user taps on Settings.
3. The user is taken to the settings page.

Exceptions

None

Error Conditions

Pressing Android's settings button while recording or listening to a message might cause an issue, lead to loss of a received message, or interruption of service. The issue is solved by blocking Android's settings button while recording or listening to a message.

4.8 Refined Use-case (Sharbel)

Since this is a design process, there must be some refining that goes on after more implementation details are apparent. For this project, some of the features were scaled down hoping to make the project target achievable. There was no harm in doing that since this is part of the software design process, in which versions of the software accumulate simple features, rather than having all the features at once in one version. The latter option can reflect negatively on the application since it can be full of bugs, and there will not be as much room for feedback from the users of the application. Figure 4.2 contains a newer version of the use-case diagram, with most of the use-cases similar to those described in the previous sections.

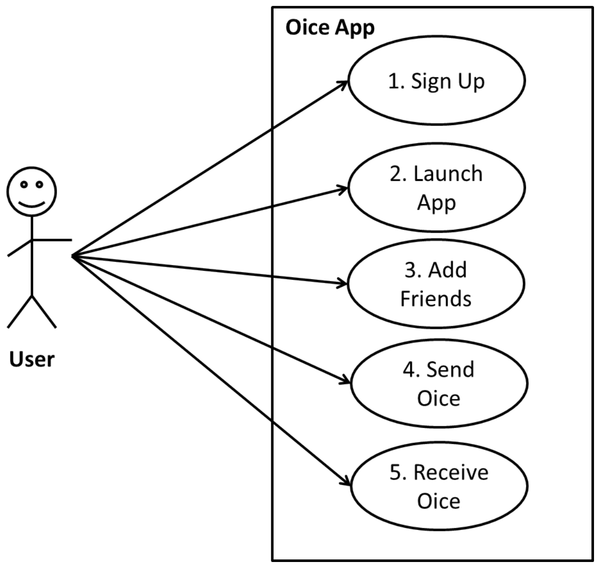


FIGURE 4.2: The new use-case diagram after parts of the scope were refined.

Chapter 5

Interaction of Classes (Omar)

In this project, five classes were used. Each class is responsible of handling many functions in the app as well as interacting with other classes to make it more integrated and useful. Although the app has only one page displaying at any given time, there are many interactions happening in the background without being noticed by the user. The reason why the app uses only the main page for most of the functionalities is that simple and clean interfaces usually attracts more users because they find it easy to use the app without any difficulty and confusion, so simplicity is an important factor in app designing. That's why the app was designed with the least classes and activities possible. The classes that were used in the application are:

In this project, five classes were used:

- Signup
- Main
- EditFrineds
- OiceOSApplication
- ParseConstants

5.1 Signup

The Signup class is responsible about the users' registration in the app, for both new user and the existing ones (if they delete their current account). When users install and run the application for the first time, they will be taken to the sign up page where they have to enter their account information (phone number and password), then it will communicate with Parse to push the provided information and try to match them with the database to see if the information already exists. It will display an error message for the user if he entered a phone number that is already in the database. If it is new information, the user will be added to the database with the information he provided. The Signup class will then interact with the Main class to log the user in, then users don't have to sign in later if they kept the app in the background or even completely kill it. The Signup class only interacts with the Main class to send the user to the main page after they sign in, and to handle the user when he is sent back from Main if they delete their account. It also interacts with Parse servers to send the users' information to the backend in order to store them in the database.

5.2 Main

The Main class is the most used class in the application, it communicates with all the other classes in the application, and it handles many functions for the users. When a registered user start the app, he will be taken to the main page where he will be able to see a list of his friends. From the main window the user can record audio, display a progress bar showing his current progress through the maximum amount time limit of the recorded file. The main page also includes an option menu which will display two options; the first one is to delete account in case the user got a new phone or if he wants to change the number he registered with, this option will sign him out and send him back to the Signup screen. The other option is the Edit Friends menu where users can manage their friends lists (more about this later). The activity will also display a loading spinner when the friends list is being fetched from the database. The Main class interacts with the Signup by receiving the user that has just finished registration and was sent by the Signup activity, the second interaction with Signup activity is by sending the user back to the Signup page when they delete their account. It also interacts with EditFriends by sending the user to the edit friends page and refresh the friends list after the user gets back from the edit friends page. In addition, Main interacts with OiceOSApplication class by obtaining the Parse unique application key so that it can communicate with parse. Finally, Main interact with ParseConstants class to obtain parse key strings that will be useful to organize the strings that uses parse (more on this later).

5.3 EditFriends

The EditFriends class is mainly responsible for managing the friends list in the Main activity, and modifying the relationships between users in Parse database which will be reflected to the app. When the user enters the EditFriends page, he will be able to see all the friends that he already has highlighted in the list of users. The user can add friends by simple taping on their phone number in the list to highlight it, or they can remove friends also by typing on the number to remove the highlight. All changes will be made to the users relation Coolum in parse database and will be reflected in the app when the user gets back to the main page. The edit friends page also displays a loading spinner to indicate to the user that it is currently trying to connect to parse database to obtain the list of users available. EditFriends activity interacts with one of the classes only, which is the Main class to receive users that requested the edit friends page in the main page, and send them back again when they are done modifying. All the other interactions are done with parse to obtain list of users and list of friends that the user already have, then modifying users relationships to add or remove friends and then save the changes to the database.

5.4 OiceOSApplication

This class is only used to initialize the connection with Parse and to send Parse unique application key to be recognized by the server to know which application the user is working on.

5.5 ParseConstants

This class is used only twice during the implementation of the app. Its purpose was just to store the string constants that are going to be used in the tables of Parse database. Because using a hard coded strings to identify a specific row or Coolum in the database tables caused many issues. And it helps organizing the values of the strings that are used to communicate with the backend in case there was a need to change the name in the future, or in case of the need to change the backend altogether. It will be easier to update those constants directly instead of going all over the code searching for them.

Chapter 6

Contribution (Sharbel)

Initially, each team member was assigned to a different task so that the work is done in parallel. At the beginning few weeks, the project was concentrated on the design of the interface and the scope of the project. The initial task division, which is shown in Table 6.1, was done with little knowledge of the implementation details of the project. Given the lack of experience of the members in developing applications, as well as getting exposed to developing an app which is based on back-end equally as it is based on front-end, the tasks division was discovered to be not sufficient for the project to be completed. Hence, each team member had to do more work than expected.

Table 6.1: Initial task division in the report

Member	**Contribution**
Mike	Users (sign-up, Comparison with Phone Contacts using Contact List Providers)
Omar	Files (uploading audio files on the server)
Sharbel	Queries (retrieving the files from the server)

Later on, the activities that each member of the team did have changed, with each member requiring to do more research on more topics to make the development of the application possible. The member contribution in both the project and the report are listed in Tables 6.2 and 6.3, respectively.

,

Table 6.2: Member contribution in the project

Member	**Contribution**
Mike	Learning to use Parse (BaaS), Creating Splash (XML and Java), Creating user Login,
Omar	Learning to use Parse (BaaS), Learning to transmit basic file types on Parse, Implementing Audio Capturing from device, Implementing OnLongClickListener on the Queries, Implementing OnTouchListener on a ListView, Recording audio with the onTouchListener on queried items of a ListView, activity for adding friends, logging out of the users
Sharbel	Learning to use Parse (BaaS), experimenting with objects on Parse, implementing Queries from parse, creating ListView Activities to show query results from parse, creating user class on Parse creating user relations on parse, implementing user sign-up and user classes, Implementing Audio Capturing from device, Adding activity for adding friends available on parse server.

Table 6.3: Member contribution in the report

Member	**Contribution (Section)**
Mike	**Introduction**: Motivation (1.1), **Workflow**: Launch App (4.1), 3 Delete From Recents/ Delete Group (4.6)
Omar	Description (1.2), **Workflow**: Send to many (4.4), Change Settings(4.7), Receive Message(4.3), **Summary, Problems, & Future** (Chapter 7), **Interaction of Classes** (Chapter 5)
Sharbel	**Abstract**, **GUI Design**: logo & theme, UI design, (Chapter 2), **Backend Design** (Chapter 3), **Workflow**: Create/Message Group(4.5), Send to a contact (4.2), Refined Use-cases (4.8) General report editing and layout

Chapter 7

Summary, Problems Faced, and Future Work (Omar)

In this project, an Android application was created from scratch using Java, Parse.com was used as the backend for the app, saving the database of the users and the files being transferred across the application. Parse also stored the relationships between users. The following functionalities were implemented:

- Signup for new users and adding them to the application's database in the back-end using Parse.
- Users don't need to login as the app will keep them logged in even when the app is working in the background or when it is resumed.
- Protecting the app from the problem of crashing or getting to the wrong activity when the user press the back button.
- The ability to delete an account (and sign out), deleting the account will lead to the signup page to register for a new account.
- When the information used to sign up matches an existing account information in Parse databases, an error will be displayed informing the user that this account already exists.
- Loading the friends list from Parse using the relationships between users in Parse, and display them as a listview in the main page of the app.
- The ability to add friends, and remove friends. The EditFriends activity highlights the items that represents users that are already friends with the user.

- A spinner that appears when the list of friends is loading, and it disappear when the loading is done.
- An ImageButton that can record voice on the memory card when the user taps and holds on it, and the recording will stop when they remove their hand.
- A progress bar that activates when the user taps on the record button, it keeps track of the progress of the recording (up to 15 seconds) and it will stop and reset when the user stops recording.
- A play button that can play the most recent recording by the user.

Issues that were faced:

- Using onTouchListener and onClickListener for the same listview caused issues.
- Taping any element with an OnTouchListener method implemented will cause an app crash.
- The intent for the voice file, and how it can be sent to the backend to be delivered to the recipient.

Future Implementations and Updates:

- Make the app able to send the audio files to a specific user in the database after storing it in the memory card.
- Choosing a friend after recording to be sent to him.
- Replacing the phone number of the users in the friends list and EditFriends activity with the actual names of the users by matching the numbers with the address book of the phone.

References

[1] "Parse documentation." https://www.parse.com/docs/android_guide. Accessed: 12/2/2014.

[2] "Audio capture." http://developer.android.com/guide/topics/media/audio-capture.html. Accessed: 21/2/2014.

[3] "Managing audio playback." http://developer.android.com/training/managing-audio/index.html. Accessed: 23/2/2014.

[4] "Managing the activity lifecycle." http://developer.android.com/guide/components/activities.html#Lifecycle. Accessed: 27/2/2014.

[5] "Onclicklistener." http://developer.android.com/reference/android/view/View.OnClickListener.html. Accessed: 28/2/2014.

[6] "Onclicklistener identiy a button." http://stackoverflow.com/questions/3320115/android-onclicklistener-identify-a-button. Accessed: 28/2/2014.

[7] "Onlongclicklistener." http://developer.android.com/reference/android/view/View.OnLongClickListener.html. Accessed: 28/2/2014.

[8] "Onlongclicklistener." http://stackoverflow.com/questions/4621439/use-both-onclicklistener-and-onlongclicklistener-in-listview-android-
Accessed: 28/2/2014.

[9] "Onitemclicklistener." http://developer.android.com/reference/android/widget/AdapterView.OnItemClickListener.html. Accessed: 28/2/2014.

[10] "arrayadapter listview." http://stackoverflow.com/questions/18405299/onitemclicklistener-using-arrayadapter-for-listview. Accessed: 28/2/2014.