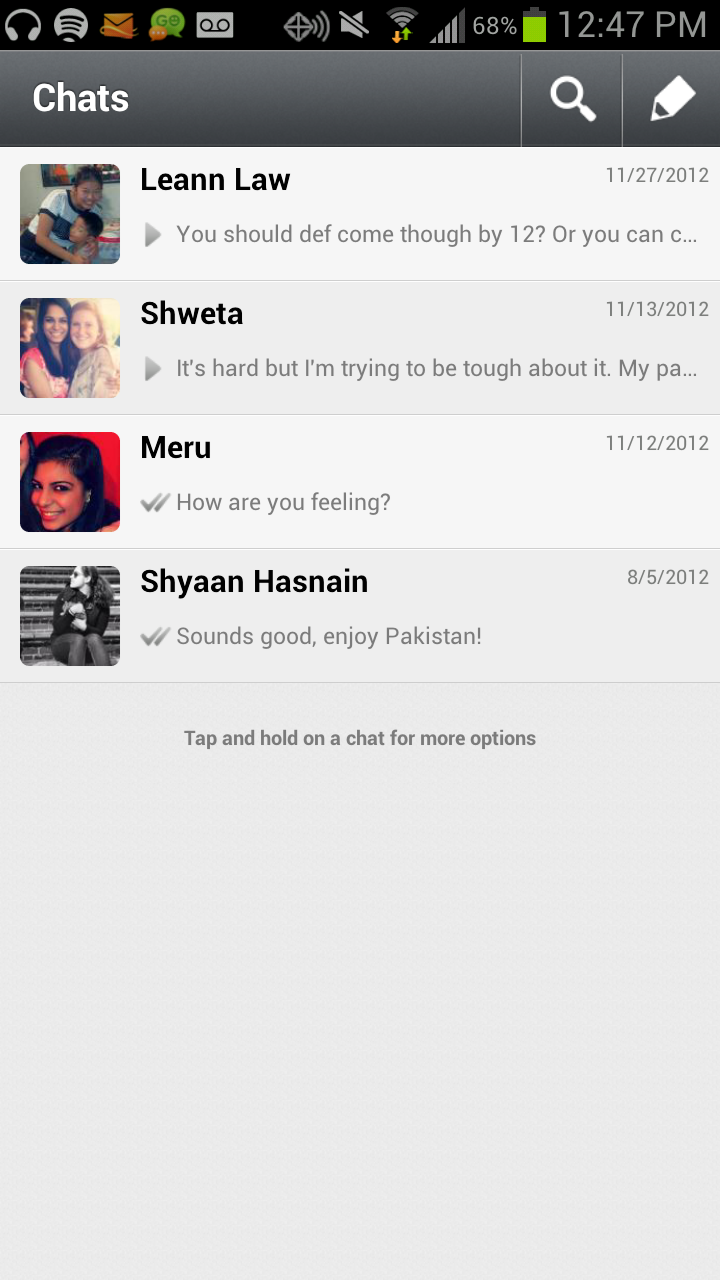
Inspiration for Project

The inspiration from my project was the texting application I currently use for my phone: Galaxy S2 running android 4.0.3 Ice Crème Sandwich. The texting application, [Go SMS Pro](https://play.google.com/store/apps/details?id=com.jb.gosms) often lagged, especially when typing or sending a message. Once my text count got into the several thousand, I would type a message but it would not display for several seconds. Furthermore, once I did submit the text, it would take several seconds for it to show up on the screen and then several seconds to return to the main contacts/thread page. This of course was frustrating, especially on a phone that had a 1.5Ghz dual-core Snapdragon with 2Gb of RAM, one of the specs for a mobile phone to data.My reason for keeping with GO SMS Pro and not switching to another texting application was its support of group chat, which other texting applications did not support or could not handle well. However, I believe the lag in the application was due to the large range of other features that came with the application such as themes, cloud backup, sms blocker, filters, fb sync, etc. See below:

Another texting application I tried out is whatsApp. This is not an sms application but a messaging application. To explain the difference, a user sends messages over wifi and the message if hosted by the whatsApp servers. The end user must also have whatsApp installed. The message does not go through the wireless network provider and is not stored in the sms dictionary hosted by the phone.



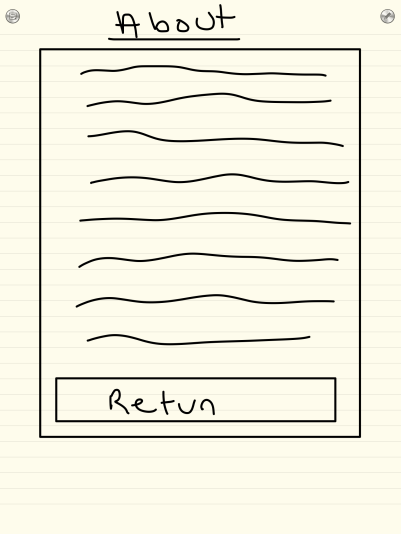
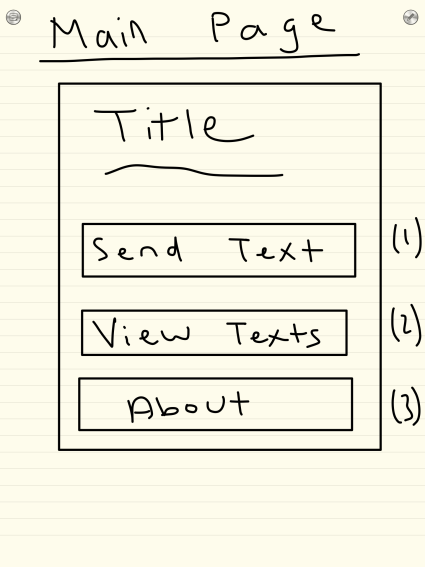
I really enjoyed the simplicity and efficiency of whats app. It had a simple, yet intuitive layout. The main page had a simple list of thread/message previews and when you clicked on a specific thread, the entire conversations came up. Easy, clean, simple. And best of all, no lag.

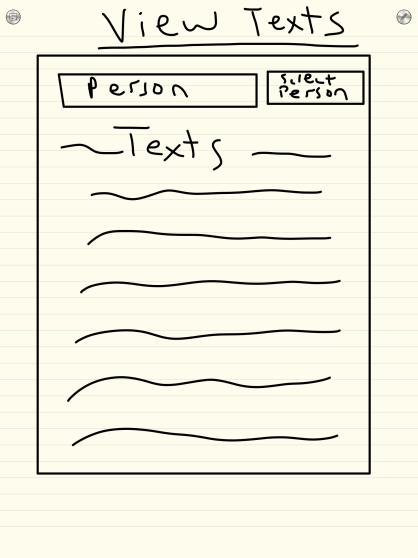
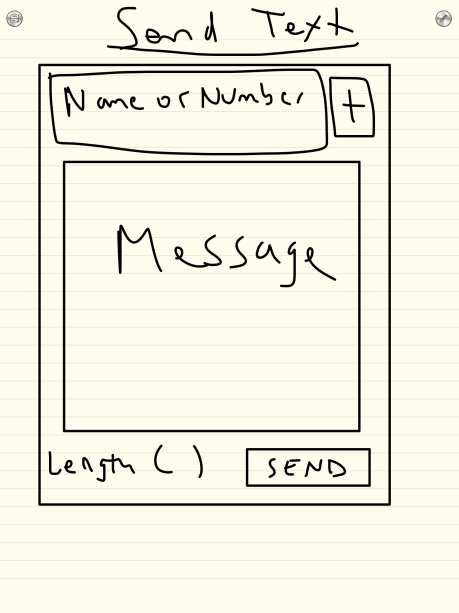


I decided to take the best of both applications and make this great application with a seamless, easy texting experience. I did understand that these programs were created by companies, startups initially, that were setup just to create this app, with professional programmers, software engineers, design artists and HC interaction majors, yet I felt I could do myself justice. Not because I was cocky, but because it seemed at the core level, these apps were simple. SMS apps are simple in general. Just send or view texts. No crazy algorithms, no layouts too crazy to code. SMS was easy. Oh how wrong I was…

Overview of User Interface

When first pondering my design, I wished it to be simple. A part of me wished to copy all the other texting applications out there and do a thread view, and clicking on a thread would bring up the conversation view, but I really wanted to create something unique. Furthermore, I wanted something simple and intuitive, something easy to explain and understand. Therefore I broke down texting into its two main components: viewing texts and sending texts. This became the inspiration for my main screen, see design layout picture below. Have a simple button, a clear message: click on this to send a text. Click on this to view a text. Click on this to get information about the app. Click here to close. Simple, clean, intuitive. I took this same approach in the sendtext layout. Choose a contact or write the contact(s) number, and send the message. One box for the number, one box for the message and a button to send the message. With the view text. A place to choose the contact to view and below, the messages to display. Everything was simple, easy!





Overviewof Modules, Design and Functions

Clearly, the difficulty of this project was getting not only python to run on my phone, but to have access to both the text messages as well a GUI on my phone. Thankfully, after research, this problem was easily solved with a google project called [Scripting Layer for Android](http://code.google.com/p/android-scripting/) (sl4a). This application provides modules for different scripting languages as well as an interpreter and IDLE for editing. sl4a also provides [facades](http://www.mithril.com.au/android/doc/index.html), basically an api that allow access to a specific aspect of the android os. For example, in this project, I used the ContactsFacade to gain access to the contact name and number. I used the smsFacade to pull up text messages. To call a function in the api, call upon droid followed by the function. Note droid is defined as a dummy variable for calling a function as defined by sl4a script: **droid = android.Android().** For example **droid.phoneCall(\*)** places a call where \* is the string of the number.

One of the hardest parts of this project was finding out how sl4a handled GUIs. Here I wasted a lot of time (see time sheet) before finding an ideal one: a google project that is an extension of sl4a called [fullscreenwrapper2](http://code.google.com/p/python-for-android/wiki/fullscreenwrapper2). This ‘framework’ extends the basic [FullScreenUI](http://code.google.com/p/android-scripting/wiki/FullScreenUI) api provided by sl4a that used XML code for [Android layout](http://developer.android.com/guide/topics/ui/overview.html) (note android in general [not sl4a] also allows java to initiate layout elements but of course, this cannot be done so the GUI is limited to XML). XML is like HTML is the sense that is uses a series of nested items as well as inheritance of attributes. An explanation of an example XML file is below. Elements in the layout, whether is be an object or a type of layout, is given an id which both identifies it and in the case of fullscreenwrapper2, allows it to communicate with the python (the python code can pull data from an element of the xml as well as edit the data is some cases).

Any code written as a extension of Fullscreenwrapper2 works by defining new instances of the layout class (defined in fullscreenwrapper2). Each layout is basically a deference screen you wish to display. In my case, I would have a main screen layout instance, a send text layout instances, a view text layout instance and an about layout instance. Hariharan Srinath, the developer of fullscreenlayout2, designed each layout to require layout instance to have **\_\_init\_\_(self)** function that loads the XML but calling on a super class function, as well as an **on\_show()** and **on\_close()** function. The on\_show is needed to tell the script to make the layout accessible, as well as the place to define event handlers such as a screen, button, list press (this is where XML ids become important; event handlers wait for an element, identified by its id, to have an action imposed on it, and then calls a function). As written by Mr. Srnath on the documentation page:

“You would typically use **click\_EventHandler** &**itemclick\_EventHandlers** (for ListView) with Views. The **init** for these take the View itself & an event handler function reference to call when the event occurs as parameters. You would typically associate **key\_EventHandler** with the layout itself. The **init** for key\_EventHandler takes a key\_match\_id (defaults to "4" which is the back key), a view (defaults to None) and an event handler function reference as parameters.”

For example, **self.add\_event(key\_EventHandler("4", self,self.cancel ))** waits for the button identified by 4 (back button on andoird devices has an id of 4) to be pressed and then initiates the **self.cancel()** function which closes the current layout:

**def cancel(self,view,eventdata):**

**FullScreenWrapper2App.close\_layout()**

“The on\_close() is mainly allow you to save state before a layout dissappears if needed. You can have **pass** as the only statement.”

Once you have defined all your layouts, you need to initialize:

“Once your layout class is setup, in your main function, initialize the framework first with Android.Android(). Then show the layout using**FullScreenWrapper2App.show\_layout()** and initiate the eventloop()”

**if \_\_name\_\_ == '\_\_main\_\_':  
    droid = android.Android()  
    random.seed()  
    FullScreenWrapper2App.initialize(droid)  
    FullScreenWrapper2App.show\_layout(FIRST\_LAYOUT())  
    FullScreenWrapper2App.eventloop()**

Conclusion

To reflect on my project, it did not turn out how I had planned for several reasons. The first one was the XML layout as my only option for a GUI. This limited how the user of this application could interact with my application. For example, it would have been nice to use perhaps a picture of a cool looking text, or even just cool looking texts, instead of simple buttons to initiate actions, but then I would not be able to use event handlers and have action done when the pictures are hit. Second, the limitation of text fields meant I had to include text in the boxes initially, and could not leave it blank or with just a hint. However, these are little concerns. I would like to talk about access to the android device in general.

I found THE BIGGEST challenge of this entire project was pulling the texts in a quick fashion and displaying them in a appealing manner. (As a side note: For pulling the texts, the smsFacade returned only texts I have received, not sent. I have tried doing research on this but I am still unaware why sl4a sms api only calls received messages. Perhaps android only stores received messages and its up to to the SMS application to create a database of sent texts itself. Therefore, the viewtexts layout on my application only shows sent messages). When you call the texts, they are returned to you by id (ids are assigned based on when the text was received – so they are ordered by time). It includes the date sent (in terms of seconds), message, if opened/unopened, the number that sent the message, as well as many other attributes. This was fine. But to display the messages how I wished, I had to match the number, to the name in my contact list as well as they type of phone that sent the message (cell,work, home based on how it was defined in the contacts list). And I had to do this efficiently. Therefore, when the application first loaded I created a dictionary of all this, and had the user wait. But therefore, if you receive a text while in the application, it will not display until the application is reopened and the dictionaries recreated. I am really curious to see how current applications deal with this issue and do it efficiently as clearly, my dictionaries take several seconds to load and this is not true with GO SMS Pro. Perhaps this is benefit of using java as opposed to python on android. Furthermore, how messages, when sent or received, are automatically shown on the screen. I assume this is an issue with the sl4a list view, another limitation of the platform.

All in all, although I cannot say I plan to use my messaging application in comparison to GO SMS Pro (I’ll just have to delete my texts often so it does not lag), I learned a lot in this project. From HTML, Javascript, CSS,XML for android layout, to learning how to do A LOT of research before attempting a project (as to not waste time figuring out the best UI), I would say this project is a success!

Explanation of sample XML (XML in Blue, Explanation in Red)

Define layout name. Insert XML declaration which specific what version of XML encoding used. (Note: utf-8 means we are encoding in Unicode 8 bit)

sendTextLayout="""<?xml version="1.0" encoding="utf-8">

Defines namespace by calling page that declares all views and attributes allowed by android XML layout.

Here we are using a specific layout, [linear layout](http://developer.android.com/reference/android/widget/LinearLayout.html)

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

Give the layout and id and define if its vertical or horizontal. In this case vertical, so each element of this layout is places below each other.

android:id="@+id/background"

android:orientation="vertical"

As this is the top layout, we want it to fit the screen both in height and width

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

Define background color as black!

android:background="#ff000000">

Define first element, a place to insert test (edit text element). Give it an id. As we are in linear view, we wish it to fit the width of the screen. The height here is 0dp (density independent pixel). This is used generally with layout weight. Layout weight tells android how much screen space an element of a layout should take up. For example, in vertical layout, is an element has a weight of 2, and another has a weight of 1, the first element will take up 2/3 of the screen height, and 1/3 for the other. The layout\_height is written as 0dp since it is calculated by its weight. Note: this is the only element in this vertical layout that has a weight so after given the other elements the height they need, fill the rest of the height with this element. The text tells what text to display in the textbox and the hint is like text, but is like faded so its like embedded.

<EditText android:id="@+id/edit\_SMS"

android:layout\_weight="1"

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:text="Enter\_SMS\_Here"

android:hint="Enter SMS Here" />

Define the usual: id, width, height, text. Margins just define margins around the object.

Note: sp = scale independent pixels.

<Button

android:id="@+id/sendSMS"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="Send Text"

android:layout\_marginBottom="5sp"

android:layout\_marginLeft="2sp"

android:layout\_marginRight="5sp"

android:layout\_marginTop="5sp"/>

Embed a horizontal layout in a vertical layout

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

Activity the tools UI editor uses to render layout

xmlns:tools="http://schemas.android.com/tools"

Wish width to expand to whole width(match parents) and height to take up just enough space (wrap content)

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="horizontal">

Define id, margins, text. Wish for item to take up just enough space both in height and width

<TextView

android:id="@+id/currentContact"

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content"

android:layout\_marginLeft="5sp"

android:layout\_marginRight="10sp"

android:text="Current Contact" />

Note the use of weights but this time the width is 0dp.

<EditText android:id="@+id/contact"

android:layout\_weight="1"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:text= "Type\_Contact\_Number\_Here"

android:hint="Type Contact Number Here" />

Close the layouts and you are done!

</LinearLayout>

</LinearLayout>

"""