Thoughts

Personal sentiment tracking & analysis for Android

Table of Contents

Change History

Date	Author	Changes made
9/10/2014	Tarif Haque	Initial setup of document, Introduction to the application
9/11/2014	Tarif Haque	Functional description of system outlined, essential functional requirements specified
9/23/2014	Yudhishthir Singh	Project description, Nonfunctional requirements
9/23/2014	Vlad Caciuc	Activity diagram
9/23/2014	Tarif Haque	High level class diagram and description, key terminology
9/24/2014	Yudhishthir Singh	Use Case Diagram, Key terminology, Functional Requirements, Nonfunctional Requirements
9/24/2014	Vlad Caciuc	Description Activity diagram, Nonfunctional requirements
10/22/2014	Vlad Caciuc	Sequence Diagrams: Register new user and Add Thought
10/22/2014	Vlad Caciuc	Description Sequence Diagrams: Register New User, Login and View Feed. Edited Login Sequence Diagram.
10/22/2014	Yudhishthir Singh	Sequence Diagrams and descriptions for View Feed and Add Thought diagrams
10/29/2014	Tarif Haque	Detailed class diagrams and descriptions

Introduction

Purpose

Thoughts is a personal sentiment tracking and analysis application for Android that enables users to improve awareness of emotional state through microblogging. The

primary function of the application pertains to the use of sentiment analysis to classify user *thoughts*, which like posts in a blog, are simply text input by the user throughout time. As the application collects these *thoughts* from the user, it is able to analyze the user's thoughts for positivity/negativity and affective/emotional state. In this way, the application is able to track the user's mood and present its findings to the user.

Key Terminology

- **Sentiment analysis** is a natural language processing method used to extract subjective information from text. Using sentiment analysis, we can:
    - Assess the positivity/negativity of a text source, or the **polarity**, as a percentage.
    - Analyze and label a text source with an **affective/emotional state** (e.g. *happy*, *sad*, *angry*).

- **Text classification** refers to the task of choosing the correct **class label** for a given text input.
    - For example, a spam filtering system labels emails as "spam" or "not spam."
    - Similarly, *Thoughts* will perform text classification to determine affective/emotional state, labeling user-provided thoughts with affective states.

- **Emotion classification** is the means used to distinguish one emotion from another. The classification of emotions is contested topic in affective science and no universal theory of emotion classification has been adopted; thus, to assign affective states to *thoughts*, our proposed system must rely on trained text classifiers modeled from theories in emotion classification. Researchers' approach to emotion classification have two fundamentally different viewpoints:

    - **Discrete emotion theory** suggests that all humans have an innate set of basic emotions that are universal and cross-culturally recognizable.
        - Paul Ekman and colleagues (1972) concluded the six basic emotions are anger, disgust, fear, happiness, sadness, and surprise. Since, an additional seventh emotion, contempt, has been added to these basic emotions. Depending on the theory, the "core emotions" vary.

        - Thousands of emotion related word are simply synonyms of these core emotions (Beck 2004).

    - **Dimensional models of emotion** suggest that emotions can be characterized on a dimensional basis in groupings.

- A classifier is called **supervised** if it is built based on **training corpora** containing the correct label for each input. The *Thoughts* mobile application will interface with several supervised text classifiers, which will require prior training data (labeled *thoughts*), implemented using the Google Prediction API.

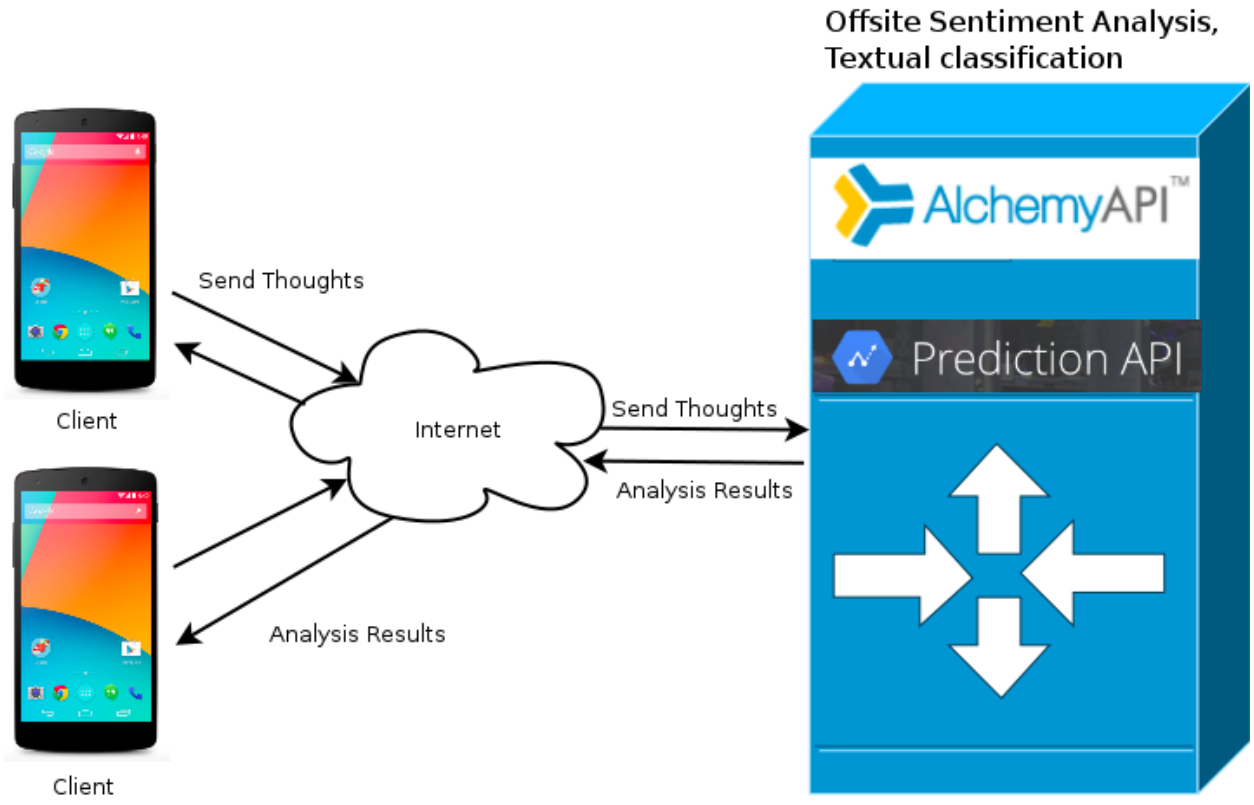
- The **Client-Server Model** is a distributed application structure that partitions tasks between providers of a service, called **Servers** and service requesters called **Clients.**

- **RESTful API -** REST or Representational State Transfer is an architectural style for distributed hypermedia (hypertext + media) systems. A RESTful web API is one that adheres to constraints specified by the REST architecture. A web application that adheres to RESTful constraints has the following features: a base URI, an Internet Media Type for the data (for example JSON), use of standard HTTP methods (GET, PUT, POST) for communication, hypertext links to reference state and hypertext links to reference related resources.

Project Description

Thoughts is built on the idea that a user may track their state of mind by providing the application text-input of what the user is thinking. Examples of *thoughts* may include mini “tweet-like” blurbs about the user’s day, blog posts, or expressions of emotions like, “I don’t feel well” or “Life is good.” *Thoughts* will be implemented on Android, the most popular mobile operating system.

A key component of the application will require sentiment analysis and classification of user thoughts, which is a computationally expensive task and requires natural language processing tools that are not feasible to implement on a mobile device locally. Thus, we opted to model the entire system using a client-server architecture. The Android application, or the client, provides the interface for user interaction: accepting thoughts, allowing the user to select thoughts for analysis, and displaying the results of those analyses. The server accepts thoughts sent for analysis and classification from mobile clients and returns the results of the analysis.



Client Side Features

Login, Registration and User Profiles

The client application allows users to register an account or login to an existing account. Every user has a profile which contains user information such as username, email, age, gender, profile picture and a set of zero or more thoughts.

Thoughts

Each *thought* is associated with a location, date and time. A thought is associated with classification results if it has been analyzed. The client allows the user to enter new thoughts, view old thoughts, select a group of thoughts for analysis and view results of analyses using a feed like view.

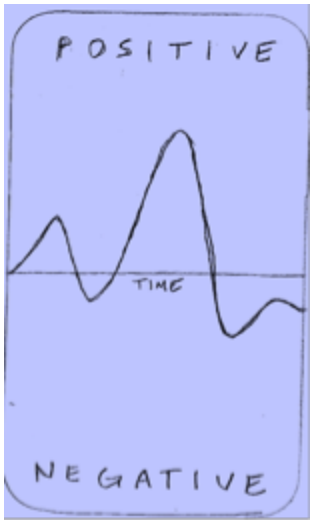
Context awareness through sensor data

We use the phones GPS to associate every thought with a location. We also associate each thought with a date and time.

Visualizations

Polarity vs Time graph

Thoughts can be classified based on polarity. A polarity classification characterises thoughts as either positive or negative on a scale from -1 to 1, with 1 being the most positive, 0 being neutral and -1 being the most negative. We use the results from this analysis to construct a polarity vs time graph. This allows the user to see changes in emotional state over time.



Discrete Emotion Graph

Discrete Emotion Theory classifies thoughts as belonging to an innate set of basic emotions. When a user selects a discrete analysis we classify each thought as belonging to one of these core emotions and display the results as a discrete emotion graph. This allows the user to easily compare the ratio of different types of thoughts.

Mood Map and Google Maps Integration

Each thought is associated with a location. We can use the results of thought classification and location to construct a map of moods. We display the classification and location on a google map.

Google+ Integration

We want to allow the user to login using their Google+ account. This will allow us to pull user information and posts from google plus.

Server Side Features

The server receives some thoughts from the client. The server then uses advanced statistical methods to analyze this data. We leverage the use of powerful sentiment analysis and classification systems developed by third parties. We will be using the *Alchemy* and *Google Prediction API* both of whom have a RESTful interface that accepts data in JSON format, perform analysis on that data and return results in JSON format. The client will send properly formatted JSON data which will be consumed by these services. The servers analyze the data and return results in JSON format which is then sent to the client for further processing. Therefore the server side of the application represents external API's and services. The design of the application will be sufficiently general such that it can be easily extended with new classification models.

Creating a custom sentiment analysis model

1. Collect & label training data
    - "sad", "Feeling kind of low...."
    - "excited", "OMG! Just had a fabulous day!"
    - "bored", "Eating eggplant. Why bother?”
2. Upload training data to cloud
3. Train the model using Google Prediction API
4. Use trained model to classify unseen thoughts!

Functional Requirements

The functional requirements map closely to the use cases of the application.

Essential

1. Create an account - The user will be able to create an account. We will ask the user to provide a username, password, age, gender and a photograph (optional). We will use this information to create a user profile. The profile will also contain all of the users thoughts.
2. Login to the application
3. Add a new thought
4. Select a set of thoughts for analysis - We allow the user to select any number of thoughts for analysis.
    a. Select a specific sentiment analysis model - Users will be able to select one of a number of different analysis models to classify their thoughts.
5. Display results of analysis - Analysis results will be visualised and displayed to the user using the phones graphics capabilities. There will be different visualizations for different analyses.

Desirable

1. After an analysis of thoughts, visualize resulting moods on a map - Leverage the Google Maps API to display a list of moods and their analyses on a map.
2. Allow user to login into application and pull profile information from Google+.

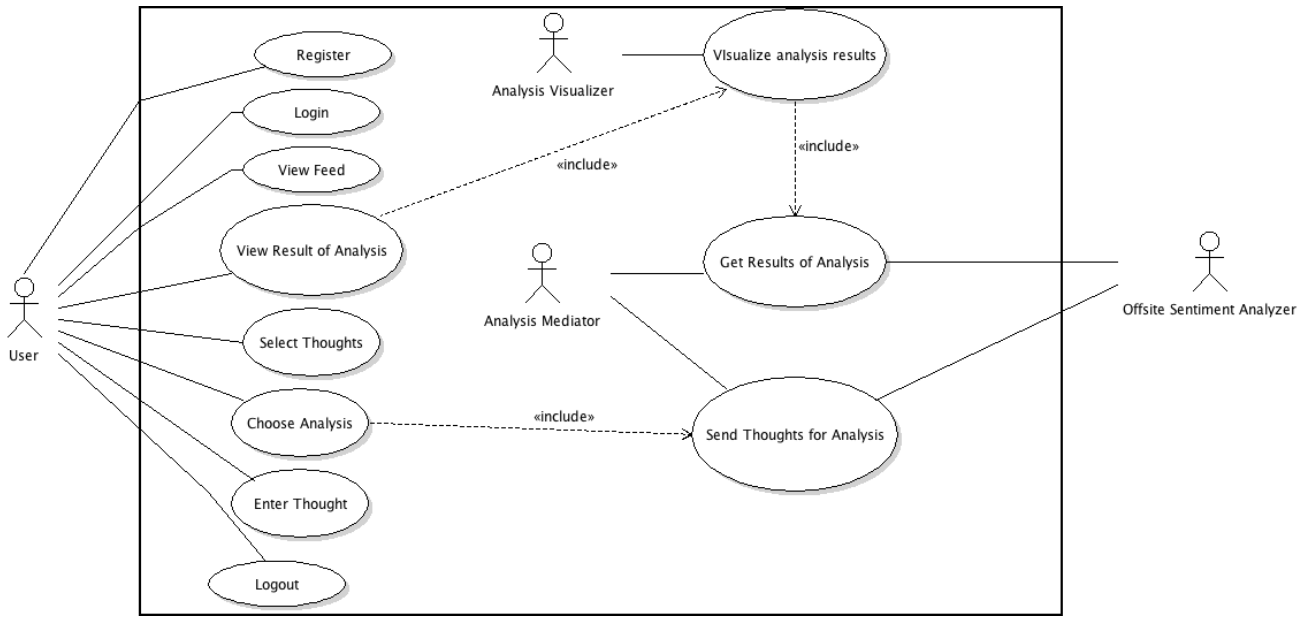
Nonfunctional Requirements

1. Login information must be stored securely.
2. The application needs internet access to send thoughts to the sentiment analyzer.
3. Communication between the client and server needs to be fast so that the application feels responsive and the user is not kept waiting.
4. The Google Maps API is required to implement the mood map feature
5. The Google Prediction API is required to perform emotional classification

6. The Alchemy API is required to perform polarity classifications
7. Access to a graphics framework is required to perform visualizations.

UML Diagrams

Use Case Diagram

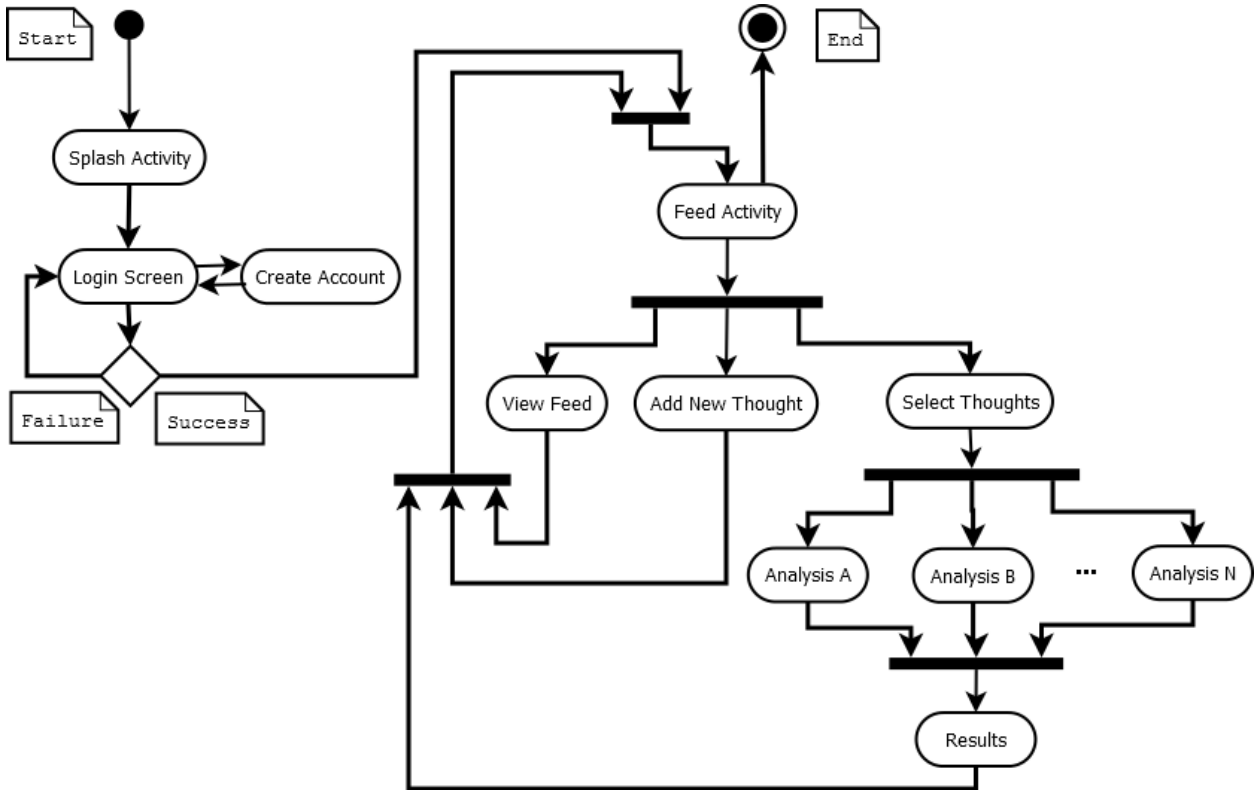


The use case diagram describes the different actors and the actions these actors can take. The User is the primary actor. The user can register new accounts, login. Once logged in a user can view a feed of his thoughts, select thoughts out of that feed and choose an analysis to be performed on selected thoughts. The user can add new thoughts which are added to the feed. Finally a user can view results of analysis.

The Analysis Mediator is an actor responsible for communicating with the offsite sentiment analyzer. It sends thoughts to the analyzer along with a request for a certain analysis, the offsite mediator responds to this request by performing analysis on this data and sending results back to the mediator. The analysis mediator acts when a user decides to analyze a set of thoughts.

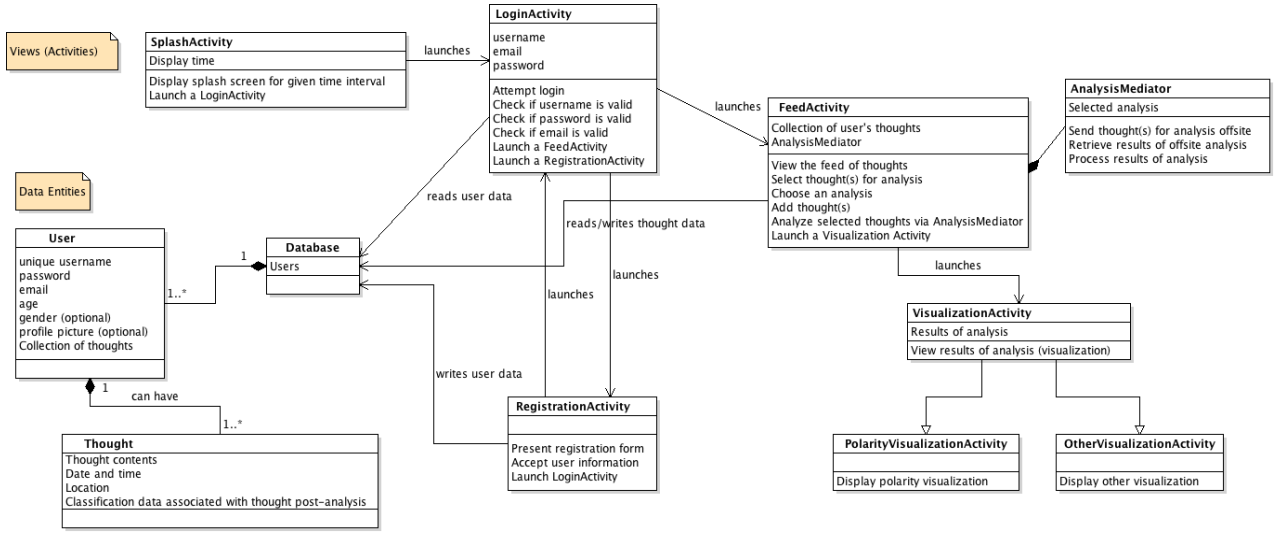
The Analysis Visualizer gets results from the mediator and creates a visualization based on the results and type of analysis. These visualizations are displayed to the user when they choose to view the results of an analysis.

Main Activity Diagram



The main activity diagram illustrates the high-level flow of the application. When the application begins, a splash screen appears for several seconds. Next, a login menu is presented, in which the user registers (creates a new profile) or signs in. After logging in, the user thoughts are displayed in a feed, in which the last thought entered is displayed first. From the feed, there will be several options the user can choose from: adding new thoughts or selecting thoughts for analysis, or exiting the application. After the user has selected thoughts for analysis, he or she chooses the type of analysis to be performed. After the results of the analysis are displayed, the user is forwarded to the "feed activity", where he or she can start the entire process again, or leave the application.

High Level Class Diagram



An easy way to think of the modern “mobile app” is visualizing the views (UI screens) that facilitate the sequence of activities the user will perform. Thus, the initial high-level class diagram for our system primarily addresses user interaction and data concerns. We conceived a set of views (Activities in Android) which provide an interface for the user to do something, and defined the data entities needed to populate these views.

Activities

In Android, an Activity is defined as a single, focused thing that the user can do. In addition, an Activity takes care of creating a window for you in which you can place your UI. Modulating our system into classes that represent these Activities provides a good starting point for design.

- SplashActivity - displays a splash screen before the login screen appears
- LoginActivity - provides the user a means to login to the application
- RegistrationActivity - provides the user a means to register/create a new profile
- FeedActivity - the central activity of the application, where the user is able to view his or her feed of thoughts, select thoughts for analysis, and add new thoughts
    - A FeedActivity has an AnalysisMediator that sends thoughts for analysis offsite and retrieves the results of offsite analysis, processing the results

in a suitable format that will ultimately be passed to the appropriate VisualizationActivity.

- VisualizationActivity (and extensions) - after a set of thoughts have been analyzed, extensions of this class present the analysis visually; each extension corresponds to a specific analysis

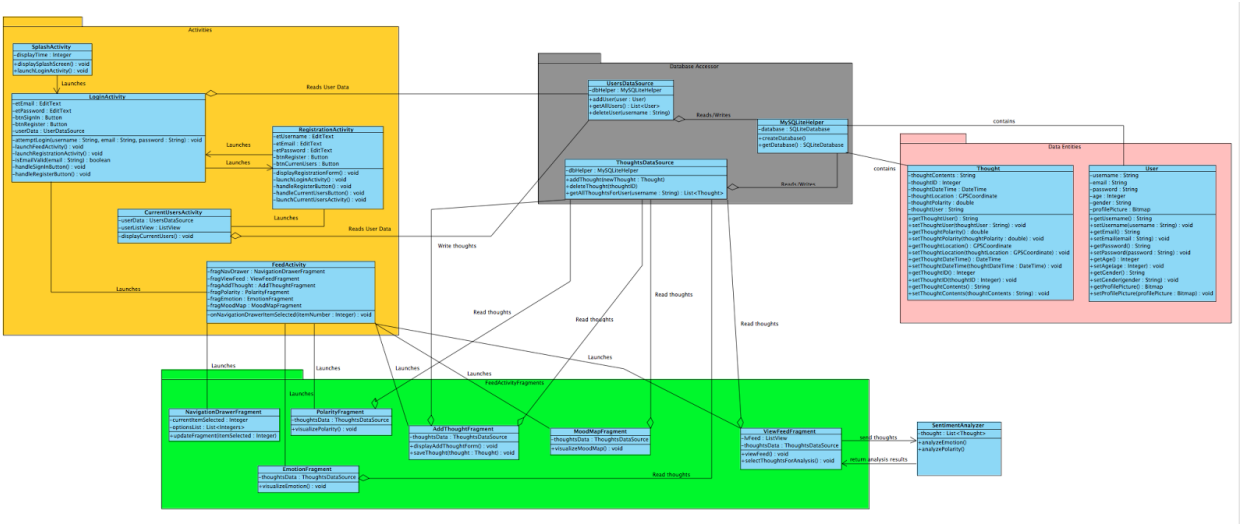
Data Model

Our data needs are straightforward: the application must store, somewhere, user personal data and thoughts associated with a given user. In our high level class diagram, we modeled our database as a globally accessible class that can only be instantiated once; in the design phase, this would presumably be achieved through the singleton design pattern. Activities interact with the single instance of the Database as needed. Ultimately, because the application must persist user and thought data across use sessions, an SQLite Database may be necessary; these technical design decisions will be finalized later in the project.

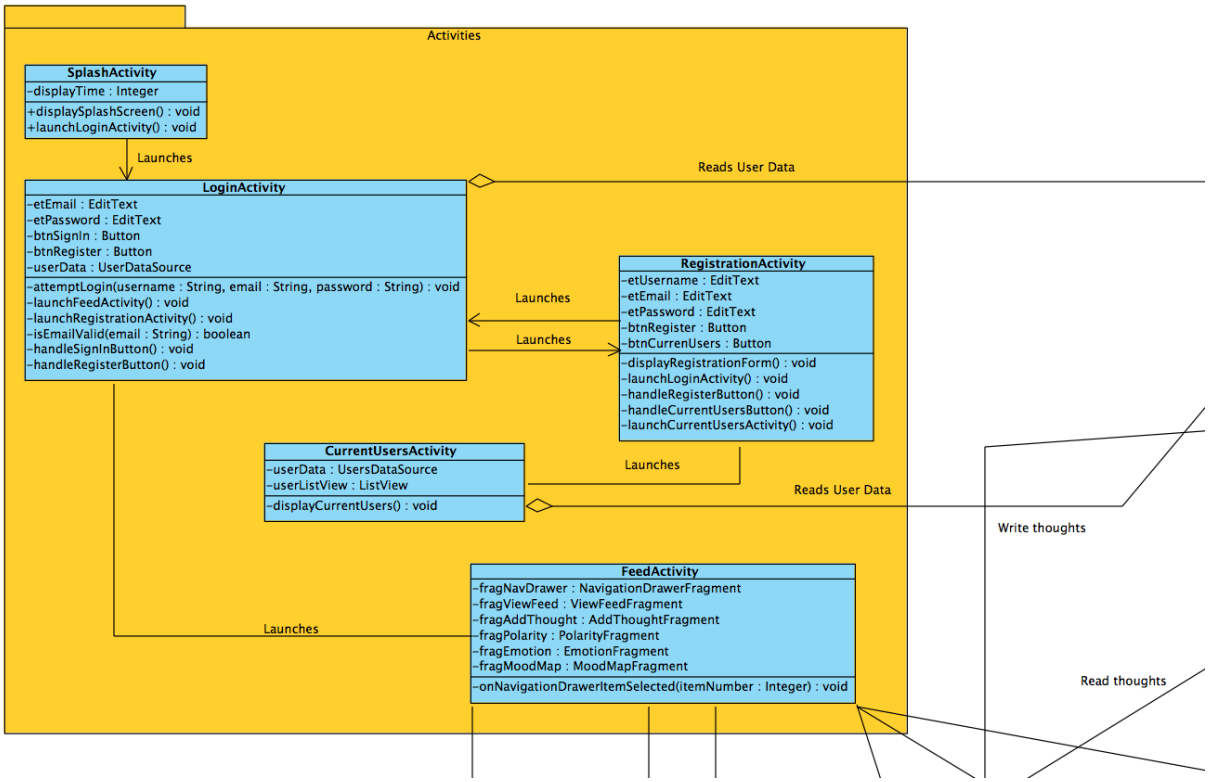
Design Documents

Detailed Class Diagram

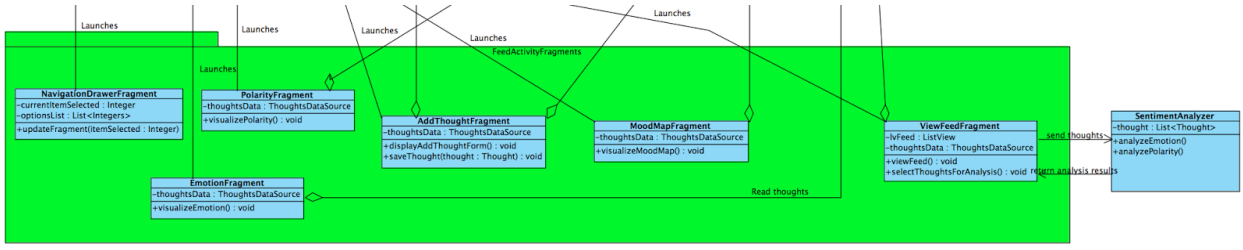
Our overall design underwent some changes more specific to Android during this iteration. We also made the decision to keep all user and thought data in a local SQLDatabase in our first prototype of the application. We split our design into four packages: Activities, Fragments associated with the Thought Feed (the central Activity of the application), Database Accessor Classes, and Data Entities. The SentimentAnalyzer class represents a “black box” that will perform sentiment analysis. Currently, this sentiment analysis will be performed using AlchemyAPI and Google Prediction API, as discussed previously. The overall diagram is shown below.



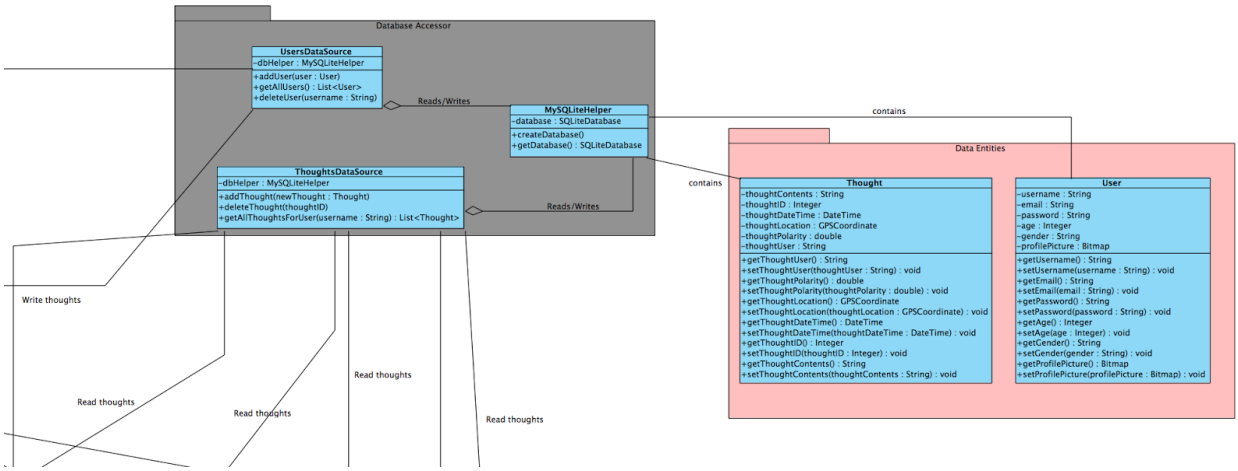
We approached the front-end of the application by detailing the classes that specify the individual “screens” of the application (Activities). The SplashActivity presents a splash screen for a specified interval and launches the LoginActivity, which allows the user to login or register through the RegistrationActivity. From the RegistrationActivity, the user may launch the CurrentUsersActivity, which displays a list of current users. The central Activity of the application is the FeedActivity, which will contain a navigation drawer allowing the user to spawn different Fragments that handle different use cases (for example, adding a thought). Fragments are dynamic UI components in Android that “fill in” an Activity. Upon selecting an item in the navigation drawer, the FeedActivity exchanges its main view with the selected Fragment.



The Fragments launched by the FeedActivity are detailed below. The default fragment will the ViewFeedFragment, which interfaces with the offisite SentimentAnalyzer. We chose this design so we may easily add fragments for new analyses on thoughts. Currently, two fragments represent the UI portion of specific analyses on thoughts (EmotionFragment and PolarityFragment). The MoodMapFragment will present a UI that maps thoughts to a Google Map that visualizes the location of thoughts.

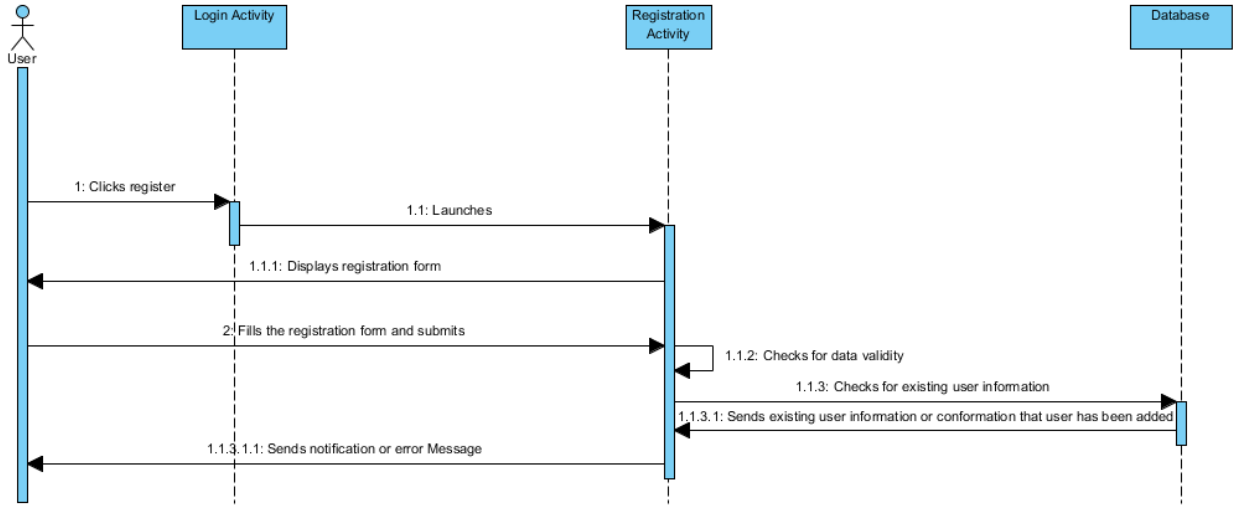


These packages model our Data Entities (Thoughts and Users) and Database Accessor classes. A MySQLHelper classes helps the application initialize the database and stores database properties, like table and column names. We use Data Access Object (DAO) Classes (UsersDataSource and ThoughtsDataSource) to manage data from the SQLiteDatabase, as described in this tutorial.


Database Accessor
UsersDataSource
-dbHelper : MySQLiteHelper
+addUser(user : User)
+getAllUsers() : List<User>
+deleteUser(username : String)
Reads/Writes
MySQLiteHelper
-database : SQLiteDatabase
+createDatabase()
+getDatabase() : SQLiteDatabase
contains
Data Entities
ThoughtsDataSource
-dbHelper : MySQLiteHelper
+addThought(newThought : Thought)
+deleteThought(thoughtID)
+getAllThoughtsForUser(username : String) : List<Thought>
Reads/Writes
contains
Thought
-thoughtContents : String
-thoughtID : Integer
-thoughtDateTime : DateTime
-thoughtLocation : GPSCoordinate
-thoughtPolarity : double
-thoughtUser : String
+getThoughtUser() : String
+setThoughtUser(thoughtUser : String) : void
+getThoughtPolarity() : double
+setThoughtPolarity(thoughtPolarity : double) : void
+getThoughtLocation() : GPSCoordinate
+setThoughtLocation(thoughtLocation : GPSCoordinate) : void
+getThoughtDateTime() : DateTime
+setThoughtDateTime(thoughtDateTime : DateTime) : void
+getThoughtID() : Integer
+setThoughtID(thoughtID : Integer) : void
+getThoughtContents() : String
+setThoughtContents(thoughtContents : String) : void
User
-username : String
-email : String
-password : String
-age : Integer
-gender : String
-profilePicture : Bitmap
+getUsername() : String
+setUsername(username : String) : void
+getEmail() : String
+setEmail(email : String) : void
+getPassword() : String
+setPassword(password : String) : void
+getAge() : Integer
+setAge(age : Integer) : void
+getGender() : String
+setGender(gender : String) : void
+getProfilePicture() : Bitmap
+setProfilePicture(profilePicture : Bitmap) : void
Write thoughts
Read thoughts
Read thoughts
Read thoughts
Read thoughts

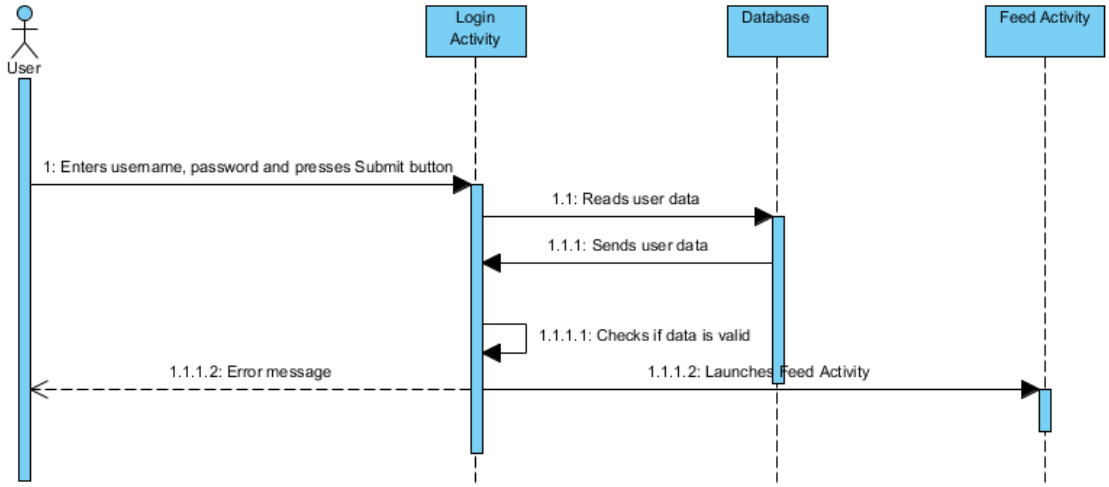
Sequence Diagrams

1. Register New User



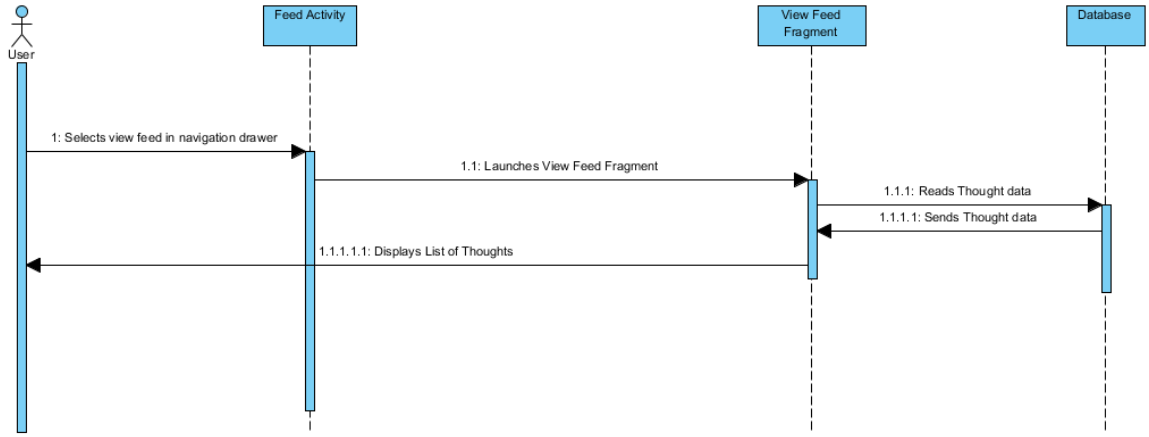
When the application is opened a splash activity is displayed for a short amount of time and then the login activity is launched. The login activity allows users to register a new account. The user clicks the register button which launches the registration activity. The registration activity displays a form to the user. The user inputs their username, email, gender and age then submits the form. The registration activity checks to see if the entered data is well formed and then checks with the database for existing users with the same information. If no such user is found the registration activity inserts the new users information in the database and sends the user a notification of success or failure depending on the response from the database.

2. Login



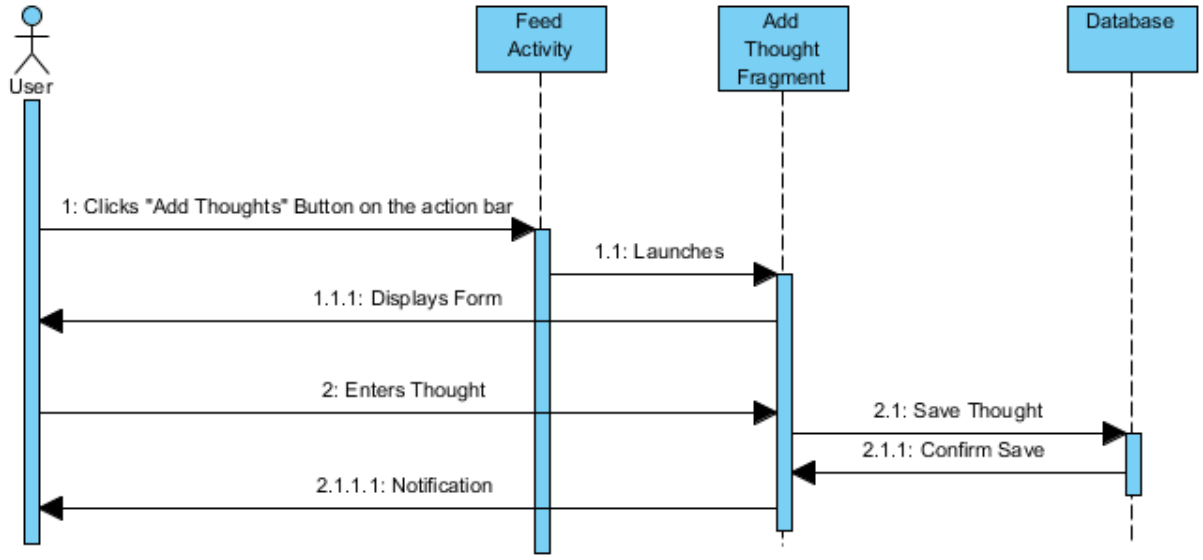
When the application is first opened a splash activity is displayed for a brief moment and then the login activity is launched. The login activity allows users to enter their username and password. Once this information submitted the login activity reads the user data from the database and checks if the credentials are valid. If everything is ok the user is logged in and the feed activity is launched. If the login fails, an error message is sent to the User.

3. View Feed



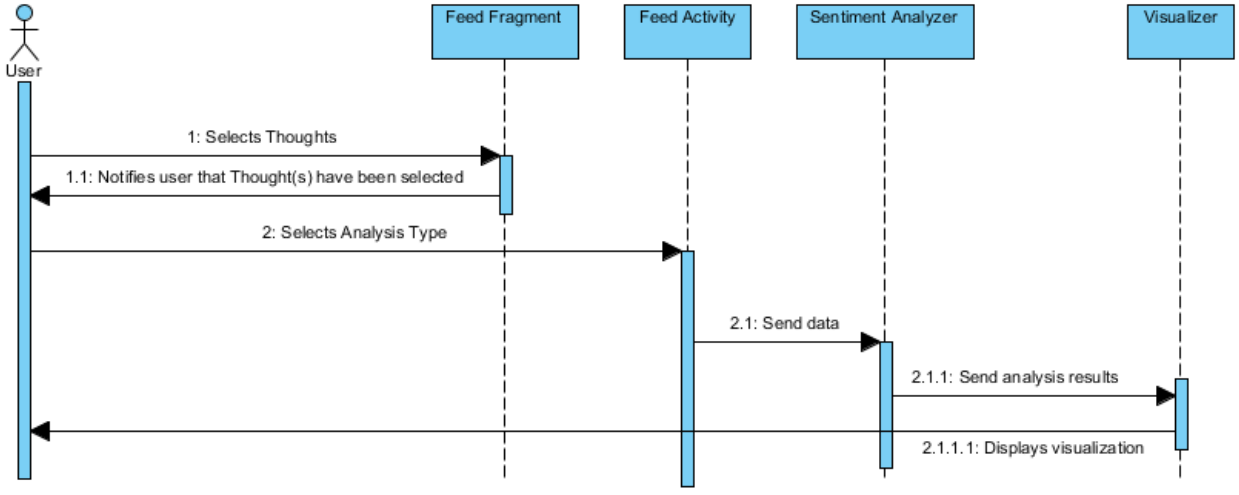
If a user wants to view a list of previously entered thoughts they simply select the view feed option in the navigation drawer of the feed activity. The feed activity launches the view feed fragment which reads a list of previously entered thoughts and populates the main view of the feed activity with this list.

4. Add Thought



To add a new thought the user selects the add thought button on the action bar of the feed activity which launches the add thought fragment. The add thought fragment is simply a form that accepts open ended input. Once the thought is entered the add thought fragment saves the thought in the database, confirms the query was successful and notifies the user.

5. Select Thoughts and Analysis Type



Sending thoughts for analysis is a two step process. First the user selects a group of one or more thoughts to analyze. Selecting thoughts for analysis is as simple as tapping the thought on the feed fragment. Once a thought is selected it's background changes so the user can see all the selected thoughts at a glance. Next the user selects an analysis type and the feed activity

sends the thoughts to the off site sentiment analyzer. The sentiment analyzer is a black box that performs analysis on the thought and sends those results to the visualizer. The visualizer displays a graph of the analysis to the user.

References

Activities in Android
http://developer.android.com/reference/android/app/Activity.html

Creating a sentiment analysis model - Google Prediction API
https://cloud.google.com/prediction/docs/sentiment_analysis

Learning to classify text
http://www.nltk.org/book/ch06.html

Document classification / text categorization
http://en.wikipedia.org/wiki/Document_classification

Excellent introduction to sentiment analysis
http://www.lct-master.org/files/MullenSentimentCourseSlides.pdf