**Fall 2014**

# Senior Project Report

**Computer Science Department**

California State University, **Dominguez Hills**

# *A Recommender System for Location-Based Social Networks*

---

Prepared by

Mitch Francis

In
Partial Fulfillment of the requirements

For
Senior Design – CSC 492

Department of Computer Science
Cal State University, Dominguez Hills
**Fall 2014**

## Committee Members/Approval

| | | |
|---|---|---|
| Dr. Mohsen Beheshti | | |
| ***Faculty advisor*** | *Signature* | *Date* |
| *Committee member* | *Signature* | *Date* |
| *Committee member* | *Signature* | *Date* |
| Dr. Mohsen Beheshti | | |
| ***Department Chair*** | *Signature* | *Date* |

## Abstract

Location Based Social Networks (LBSN) enable users and organizations to selectively push information to customers meeting geospatial delivery requirements (time, proximity, etc). These services are often driven by recommender systems that suggest information to users based distance similarity and behavioral factors. Specifications for such a recommender system is provided with emphasis on improving cold start recommendations to new users. The implementation uses the Google Android platform for LSBN user interface functions, and Google Apps Engine as the backend for the recommender service.

**Keywords:** mobile, Android, recommender, algorithm, geo-social networks

## Acknowledgement

I would like to acknowledge and thank my faculty advisor, Dr. Mohsen Beheshti,

Professor and Department Chair of Computer Science for his diligence throughout the years in support of my academic career.

A personal thanks to many of my friends and peers who have supported the completion of my education:

To Matt and Tiffany Liverpool, whom have shared the wisdom and support that elder siblings often give, motivating me to see my pursuits though to completion.

To Toshimi, Linda, Eric and Huy, for being continued motivators of academic excellence.

To David, my professional mentor, for providing a supportive environment that has given me the flexibility to invest the time needed to complete this document.

## Table of Contents

## Index of Figures

# A Recommender System for Location-Based Social Networks

Mitchell Francis
*California State University Dominguez Hills*
*mfrancis5@toromail.csudh.edu*

## 1. Introduction

Mobile computing continues to evolve at an alarming rate. The cellular phone, once solely a means for telecommunication has expanded into the roles of payment management, quantified self, and social networking. Within the social networking space, there exist a sub group of interest known as location-based social networks (LBSN). Since the introduction of Dodgeball in 2000[1], a number of LBSNs have risen to commercial success through the use of Recommender systems.
Location-based recommender systems take user check-ins, geo-tagged content, and behavioral patterns, to suggest places of interest to users. For participants, this information can be of value in the discovery of new venues. For business and researchers, it represents an opportunity for targeted marketing of temporal and local events[3], or analysis of traffic patterns for emergency and disaster recovery[1]. Such systems excel when the dataset is large, but performance is limited when faced with new users or small number of check-ins. This 'cold start' represents a notable challenge with recommender systems that we will attempt to address in this work. A specification for a recommender system using the Google App Engine platform will be detailed. Additionally a Google Android Application will provide user experience capability to demonstrate our recommender system and the approach taken to improve on the the cold start problem. In improving on the cold-start problem our approach will be to allow user preference variation in place category and distance in order to determine if we can more accurately determine locations of interest to the user.

## 2. Background

Location based networks and their recommendation systems are dependent on a number of concepts in order to be effective. First and foremost, these systems will be modified by the types of content they are capable of suggesting, which for the purpose of this paper will be location, tags, and friends. A dataset of places is also required to track user check ins at various business and venues throughout an area of interest, which will be limited to the area surrounding the university. The number of methods that can be attempted for recommendation with generally fall into the content-based and collaborative filtering[1,9]. These filtering approaches have limits when it comes to handling a “cold start”[8] which should also be considered. Then, there are performance considerations in terms of the diversity of suggestions. Some understanding of historical LBSNs is also useful for designing a suitable user experience, as well as social trends in considerations for essential location-base social network functions

## 2.1 A brief History on LBSNs

Location based systems found their first commercial uses in stolen vehicle recovery systems. The FCC later enforced the requirement for mobile telecommunication companies to enable locating callers during 911 emergencies. Several early systems required user to provide their city or zip code in order for location based services to be applicable. As GPS became more pervasive in mobile phones, it became a standard mechanism for providing a users location.[11]

Location based systems are comprised of four components; a position generating device is needed to triangulate the user location. This location must then be served to a communication network, to be delivered to an application. This allows the provision of content as part of the service. With the introduction of the modern smartphone, access to location based services has grown. Unlike predecessor systems which focused on strictly reporting back the location of some person or object, LBSNs incorporate social aspects of sharing, including the geo-tagging of imagery, social events, or locations visited by the users.[1]

iBeacons are becoming a popular piece of modern location based services.[14] They represent Bluetooth Low Energy (BLE) devices with limited broadcast range for communication with smartphones or wearable devcies. iBeacon, is Apple Computer's patented technology. The service is intended to serve as a localized tracking platform with its application generally geared towards indoor use. Some applications include tracking shopping patterns and commuter traffic.

### 2.1.1 Dodgeball

Dodgeball was founded in 2000 by Denise Crowley and Alex Rainert and is considered on of the first commercially available US location-based social networks.[7] It was acquired by Google in 2005 and canceled in 2009, when Google introduced it's own serviceGoogle Latitude.
Primarily text based, Dodgeball allowed users to microblog via SMS to friends and partrons of venues in the NYC area. Users had the ability to subscribe to notifcations on the per location basis. It was primary user base were described as bar hoppers and urbanites in the New York area. [7]

### 2.1.2 Foursquare

After acquisition by Google. Dodgeball founder Dennis Crowley created a new service called Foursquare in 2008. Initially, Foursquare was strongly coupled to technology, requiring an active GPS, but after responding to feedback regarding locations with little availability to GPS, the service moved become more decoupled from the availability of positional data. While this has opened the service to phones without GPS, making it more user friendly than its competitors, it also has made cheating more pervasive on the service.[12] From a business standpoint Foursquare generates revenue from marketing partnerships.

## 2.2 Recommender Systems use in LBSNs

Recommender Systems are an integral part of location-based social network services. Focusing on place recommendation, they differ from prediction algorithms in that new venues are generated with the purpose of providing users with new experiences and establishments, rather than predicting next location to be visited by a user.[5]

Generally, recommendation methods fall into two categories. Content-based or collaborative filtering. Hybrid systems can be produced from the combination of these two filtering systems. These system generally produce recommendations in the form of location, geo-tagged content, and friend recommendation.[1] The functions of a location-based recommendation systems have improved as social data in incorporated in their models [1,8], which ultimately are an application of location prediction algorithms.

### 2.2.1 Location Recommendation

Location can be suggested based on a number of factors. At the simplest level, highly rated places can be suggested to the user, due to a overall positive experience by all members of the networked community [13]. At more complex levels, the travel distance of the use from home or hub location can be statistically evaluated from various place check-ins to determine how far a user may be willing to travel to a newly suggested venue.[4] Rhee, et al notes that such behavior has be seen in lower order animals and can be characterized by Levy Flight patterns for random foraging walks[2,4]. This work theorizes that human and lower animals followed identifiable models when foraging for food, or companionship.

### 2.2.2 Tag Recommendation

Tag recommendation is often based on evaluation of a text label associated with a given position[1]. Here, cosine similarity between tag text categories can be applied with distance algorithms to identify content that can be of interest to the user.

### 2.2.3 Friend Recommendation

Friend recommendation is based on factors such as the overall size of a users friendship map in terms of distance, similarities between place discovery patterns, and similarities between place categories[10]. Gao et al notes that friend similarity for recommendation can be quantified in terms of the following equation:

$$\mathrm{sim}(u_i, u_j) = \frac{f_i \cdot f_j}{|f_i|_2 \times |f_j|_2}$$

*Figure 1: Gao formula for user similarity*

That is two users *i* and *j* are considered similar if the cosine distance of each user's check in vectors up to the k-th element return a value close to 1.

### 2.2.4 Filtering Methods

The filtering methods used strongly determine how effective a recommender system is during cold start[8]. Content-based filtering has a distinct advantage in this regard, due to being able to pull in other data sources besides user rated check ins and those within friend networks[15]. [7] has noted that initial recommendations can be improved, by incorporating data from other social networks, or information available by way of browser cookies or mobile phone data.

Collaborative filtering on the other hand depends on the friendships and ratings of users within the social service, and represents a well established method for generating suggestions for not only places, but many other forms of content.[9] Netflix and Amazon are two well known business which apply collaborative filtering methods to recommend product and entertainment suggestions.[6]

### 2.2.5 Cold Start Problem

The cold-start problem has been discussed at length [1,8,13,15], and a number of approaches have been considered, from aggregating socio data sources external to the location-based serivces[5], to recommending the most popular venues to all users. The problem will continue to be a challenge until a better model for human behavior can be produced. From investigating of the respective approaches, a middle approach will be attempted in the implementation of our system.

### 2.2.6 Performance Measurement

Performance measurements of recommend systems are usually done by root mean square error, which is a measurement between the predicted recommendations and the actual[6]. This sort of performance metric favors a systems with larger user bases, but represent[9]

### 2.2.7 Privacy Concerns

Location-bases services and their network communities do expose a number of privacy concerns. In some cases users forgo check in behavior to avoid system reporting of places considered embarrassing, or which may cause a physical security risk.[1] The models that are capable of providing the user with qualitative information is the very mechanism the introduces fear when a level of transparency is not provided[11]. Research in the area of LBS networks have often involved the anonymizing of user data-sets[15], although often such information is not provided to the research community due to fear of data being leaked.[2]

## 3. Loqale Recommender System

For the Loqale recommender system, a hyrbid approach was taken in order to support a scalable backend, user experience that supports participation, and a cold star mechanism that may provide some improvement on performance during low participation. From the architecture in figure 1., a frontend built on the Java Google Android Platform was selectedm with communication taking place over the Internet through HTTP transmission of JSON objects.

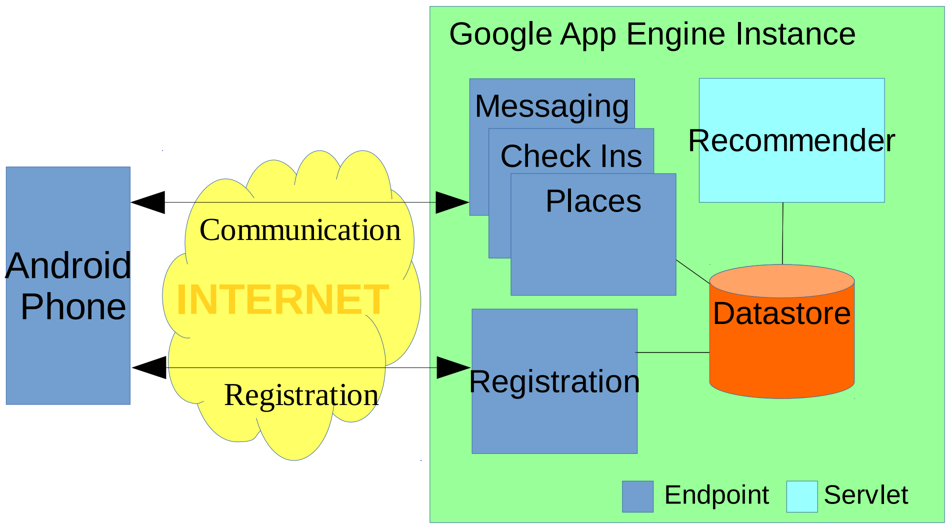


*Figure 2: Loqale Architecture*

Google App Engine provides much of the framework in the form of Cloud Endpoints, which seamlessly convert plain old java objects (POJO) into JSON. App Engine also provides support for a datastore, which we will use to house user-generated data and a fixed database of place locations.

For the frontend, the Google Android platform was selected for due to the openess of the platform and because of how pervasive android mobile devices are. The phone application will use the GPS to identify locations of interest to the users, and will integrate with client libraries created by the backend to push the data to the server.

The data set used will be relatively small due to constraints in time, and will be focused on the are around the university for which the author is in attendance of.

## 3.1 Datastore Design

Each private member of the POJO entity classes have accessors and mutators to interact with the class and to support usage by the OfyServerice class.

### 3.1.1 OfyService.java

The Objcectify Service class (Appendix 6.3.1) acts as a service for registering our entity classes for interactions with the App Engine Datastore.

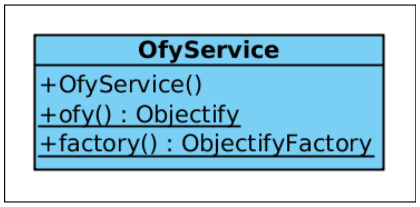


*Figure 3: Ofy Entity Service class diagram*

The *.ofy()* method returns a connector to the Objectify service allowing for the insertion access or deltion of java data objects from the data store.

The *.factory()* method interfaces with objectify factory and is used to register POJO classes for use as entity containers on the datastore

### 3.1.2 RegistrationRecord.java

The RegistrationRecord class (Appendix 6.3.2) stores the information associated with a user's android phone for the purpose of sending and receiving messages between the backend and the android GUI.

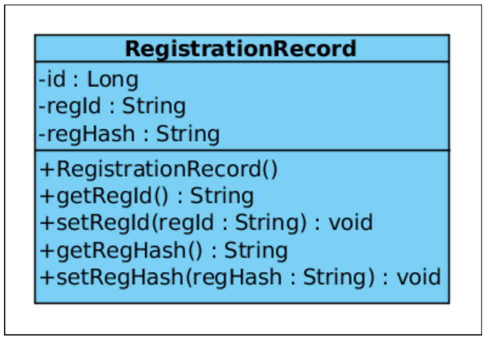


*Figure 4: RegistrationRecord entity class diagram*

The class contains a constructor and accessors/mutators for the following private members:

String reg*Id*, used for mobile device lookup of  by the recommender

*String regHash*, used for storing hash of regId

### 3.1.3 User.java

The User class (Appendix 6.3.3) stores user identification information of the recommender service.

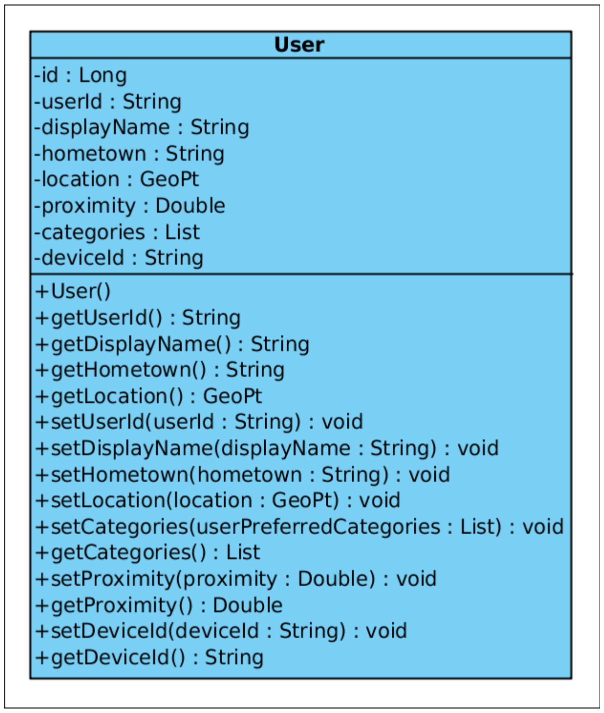

User
-id : Long
-userId : String
-displayName : String
-hometown : String
-location : GeoPt
-proximity : Double
-categories : List
-deviceId : String
+User()
+getUserId() : String
+getDisplayName() : String
+getHometown() : String
+getLocation() : GeoPt
+setUserId(userId : String) : void
+setDisplayName(displayName : String) : void
+setHometown(hometown : String) : void
+setLocation(location : GeoPt) : void
+setCategories(userPreferredCategories : List) : void
+getCategories() : List
+setProximity(proximity : Double) : void
+getProximity() : Double
+setDeviceId(deviceId : String) : void
+getDeviceId() : String

*Figure 5: User entity class diagram*

The class contains a constructor and accessors/mutators for the following private members:

String user*Id*, used for internal lookup by the recommender
*String displayName*, used for storing the preferred name
String hometown, *used to store* the hometown address
GeoPt *location*, used to store lat/long position
Double *proximity*, used to store preferred recommendation range
List<String> *categories*, used to store preferred place categories
String *deviceId*, user to store associated mobile device

### 3.1.4 Place.java

The Place class (Appendix 6.3.4) stores place information for the recommender service. Place data makes up the majority of the service data as it represents a record of city wide locations.

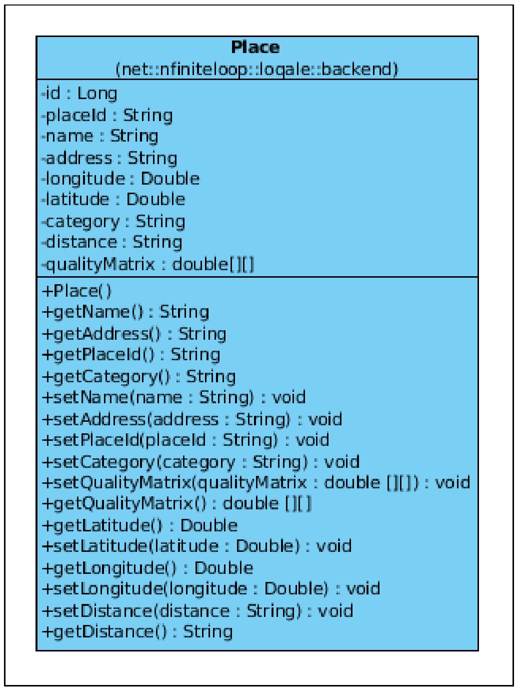

Place
(net::nfiniteloop::loqale::backend)
-id : Long
-placeId : String
-name : String
-address : String
-longitude : Double
-latitude : Double
-category : String
-distance : String
-qualityMatrix : double[][]
+Place()
+getName() : String
+getAddress() : String
+getPlaceId() : String
+getCategory() : String
+setName(name : String) : void
+setAddress(address : String) : void
+setPlaceId(placeId : String) : void
+setCategory(category : String) : void
+setQualityMatrix(qualityMatrix : double [][]) : void
+getQualityMatrix() : double [][]
+getLatitude() : Double
+setLatitude(latitude : Double) : void
+getLongitude() : Double
+setLongitude(longitude : Double) : void
+setDistance(distance : String) : void
+getDistance() : String

*Figure 6: Place entity class diagram*

The class contains a constructor and accessors/mutators for the following private members:

String p*laceId*, used for internal lookup by the recommender
*String name*, is a String field for displaying the the name
String ad*dress, to store* postal address of
Double *latitude*, used to store latitude position
Double *logitude*, used to store longitude position
String, distance, used to pass distance to client

### 3.1.5 Tag.java

The Tag class (Appendix 6.3.5) is a container for content provided by the recommender service. This content can consist of text or imagery, which is geo-tagged with a location provided by the user. The class has one constructor, eight accessors, .getName(), .getAddress

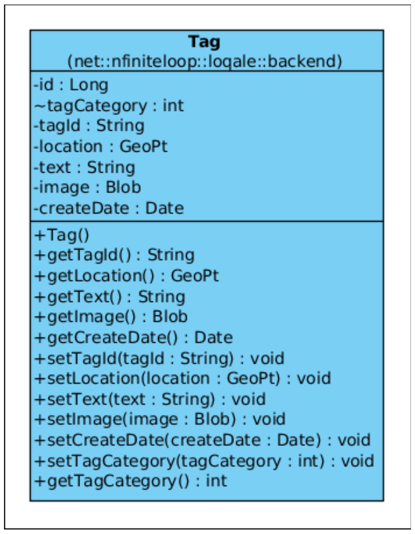

Tag
(net::nfiniteloop::loqale::backend)
-id : Long
~tagCategory : int
-tagId : String
-location : GeoPt
-text : String
-image : Blob
-createDate : Date
+Tag()
+getTagId() : String
+getLocation() : GeoPt
+getText() : String
+getImage() : Blob
+getCreateDate() : Date
+setTagId(tagId : String) : void
+setLocation(location : GeoPt) : void
+setText(text : String) : void
+setImage(image : Blob) : void
+setCreateDate(createDate : Date) : void
+setTagCategory(tagCategory : int) : void
+getTagCategory() : int

*Figure 7: Tag entity class diagram*

The class contains a constructor and accessors/mutators for the following private members:

String *tagId*, used for internal lookup by the recommender
int *tagCategory*, used
GeoPt *location*, used to mark the location of the content-based
String *text*, plain text string associated with the tag
Blob *image*, imagery associated with the tag
Date *createDate*, to track the date the tag is created, and to expire content.

### 3.1.6 CheckIn.java

The CheckIn class (Appendix 6.3.6) container for storing user check-in data. This will be used by the recommender service to determining the users travel distance and to identify similarity amongst the user base

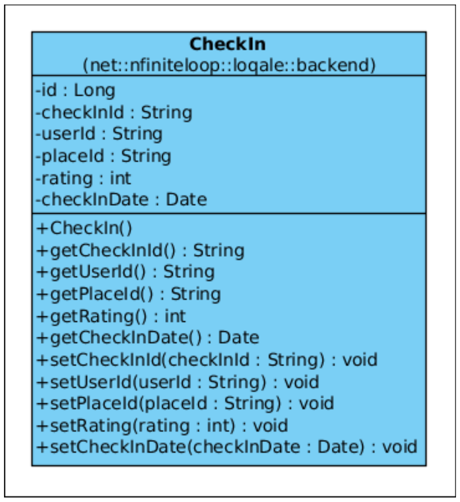


*Figure 8: CheckIn entity class diagram*

The class contains a constructor and accessors/mutators for the following private members:

String *checkInId*, used for internal lookup by th recommender
String *userId*, used for internal lookup by the recommender
String p*laceId*, used for internal lookup by the recommender
int *rating*, used to store rating of associated place
Date *checkInDate*, to store the date the user checked in

### 3.1.7 Recomendation.java

The Recommendation class (Appendix 6.3.7) is a container for storing recommendations generated by the recommender service. The Recommendation class uses the enumeration *RecommendationType* to determine the content type of the recommendation to be retrieved from the datastore. The will consist of user info, tags, or places.

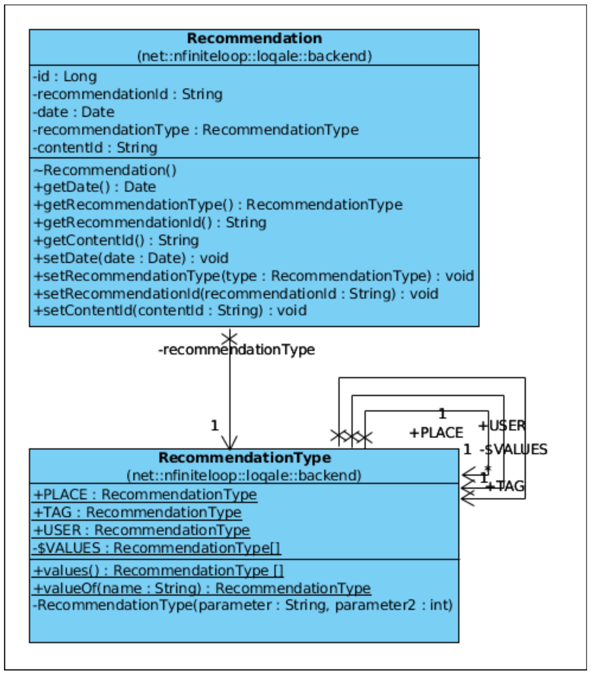

Recommendation
(net::nfiniteloop::loqale::backend)
-id : Long
-recommendationId : String
-date : Date
-recommendationType : RecommendationType
-contentId : String
~Recommendation()
+getDate() : Date
+getRecommendationType() : RecommendationType
+getRecommendationId() : String
+getContentId() : String
+setDate(date : Date) : void
+setRecommendationType(type : RecommendationType) : void
+setRecommendationId(recommendationId : String) : void
+setContentId(contentId : String) : void
-recommendationType
1
1
+PLACE
+USER
1 -$VALUES
*
1
+TAG
RecommendationType
(net::nfiniteloop::loqale::backend)
+PLACE : RecommendationType
+TAG : RecommendationType
+USER : RecommendationType
-$VALUES : RecommendationType[]
+values() : RecommendationType []
+valueOf(name : String) : RecommendationType
-RecommendationType(parameter : String, parameter2 : int)

*Figure 9: Recommendation entity class diagram*

The class contains a constructor and accessors/mutators for the following private members:

String *userId*, to associate the recommendation with a user in the datastore
RecomendationType r*ecommendationType*, to determine the content type
String content*Id*, used for internal lookup by the recommender
Date *date*, to store the date the recommendation was made

The Proile class is a composition class that is retrieve user specific information from the datastore and compiles it into a object to support the recommender class.

## 3.2 App Engine Application Design

The primary functions of the recommender system exist in the Java App Engine running on the backend of the application. Cloud Endpoints are used to support communication with mobile clients. The recommender itself, runs as a standalone servlet, and communication with the servlet occurs after each check in by the user.

### 3.2.1 Recommender.java

The Recommender class (Appendix 6.3.8) retrieves the userId of the last check in processed by the CheckInEndpoint through a HTTP GET request made by the endpoint. The details of the process will be bettered explained in the operation section of this document. The class has one constructor and one private method. *.doPost()* performs the processing of entities from the datastore. The *placeQueryLimit* member variable limits the max number of recommendations returned.

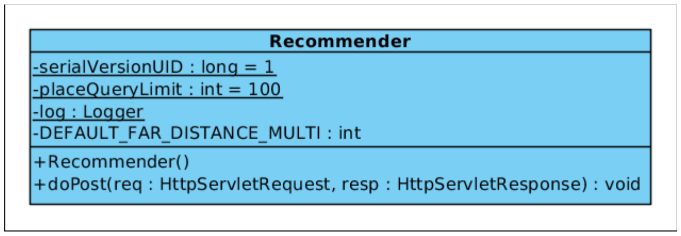

Recommender
-serialVersionUID : long = 1
-placeQueryLimit : int = 100
-log : Logger
-DEFAULT_FAR_DISTANCE_MULTI : int
+Recommender()
+doPost(req : HttpServletRequest, resp : HttpServletResponse) : void

*Figure 10: Recommender Servlet class diagram*

### 3.2.2 ContentFilter.java

The ContentFilter class (Appendix 6.3.9) processes the initial race set of recommendations generated from place check ins and will reduce them down to a list to be pushed to the user. The class has one constructor and two inhirted members from  Filter class.

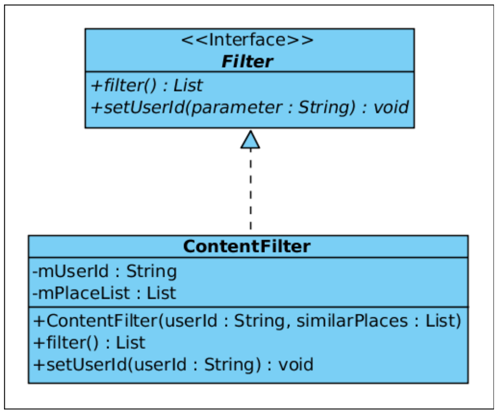

<<Interface>>
Filter
+filter() : List
+setUserId(parameter : String) : void
ContentFilter
-mUserId : String
-mPlaceList : List
+ContentFilter(userId : String, similarPlaces : List)
+filter() : List
+setUserId(userId : String) : void

*Figure 11: ContentFilter class diagram*

### 3.2.3 TagUtil.java

TagUtil class (Appendix 6.3.10) acts as a event logger for the recommender system. The class has one method. The .*recordEvent() method* logs the type event for the user and is intended to be used to share user behavior with the friends on the service.

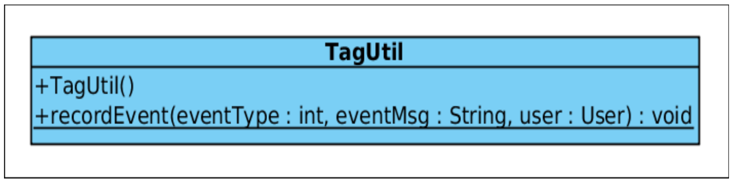

TagUtil
+TagUtil()
+recordEvent(eventType : int, eventMsg : String, user : User) : void

*Figure 12: TagUtil class diagram*

### 3.2.4 RegistrationEndpoint.java

RegistrationEnpoint class (Appendix 6.3.11) supports the storage of user device info, needed for coordinating message delivery back to the individual devices. Currently it's functionality is limited

to registration of the device and population of the user entity class. The class has one constructor, and five methods.
*.registerDevice()* adds the user's mobile device credentials to the datastore, support communication with the user.
.unregisterDevice() removes the user's mobile device credentials from the datastore.

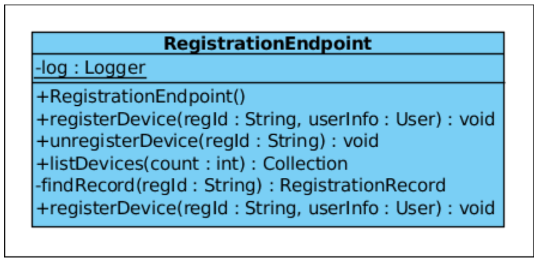

RegistrationEndpoint
-log : Logger
+RegistrationEndpoint()
+registerDevice(regId : String, userInfo : User) : void
+unregisterDevice(regId : String) : void
+listDevices(count : int) : Collection
-findRecord(regId : String) : RegistrationRecord
+registerDevice(regId : String, userInfo : User) : void

*Figure 13: RegistrationEndpoint class diagram*

### 3.2.5 PlaceEndpoint.java

The PlaceEndpoint class (Appendix 6.3.12) allows the Android frontend to request places from the datastore by a radius area of coverage. The PlaceUtil class supports processing this information in memory before returning a result list to the user. The class has one constructor and one method.
.listPlaces() method queries places from the datastore based on the distance they are from the users position, within the area specified by *radiusInMeters*.

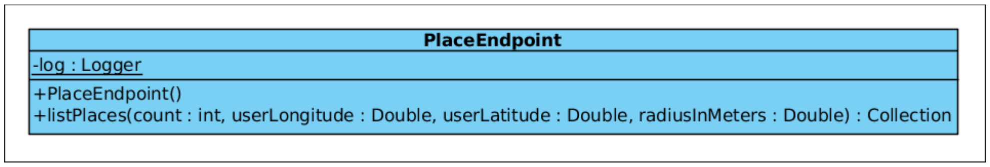

PlaceEndpoint
-log : Logger
+PlaceEndpoint()
+listPlaces(count : int, userLongitude : Double, userLatitude : Double, radiusInMeters : Double) : Collection

*Figure 14: PlaceEndpoint class diagram*

### 3.2.6 PlaceUtil.java

The PlaceUtil class (Appendix 6.3.13) is a utility class that supports the classes in the endpoint. The class has one constructor and two methods. The *.getDistanceInMeters()* applies the Haversine distance calculation to filter places down to a proximity list with. The *.getDistanceInMeters()* method is also used for distance calculations in the Recommender class and for user queries. The class also declares two member variables, radiusEarthMeters and metersToMiles for supporting distance computations.

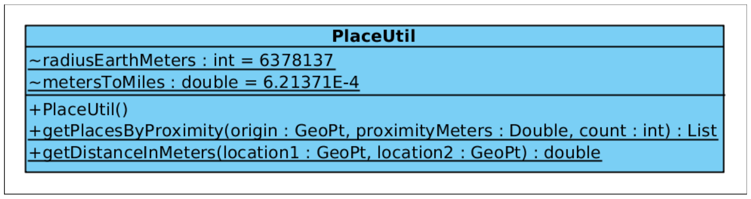

PlaceUtil
~radiusEarthMeters : int = 6378137
~metersToMiles : double = 6.21371E-4
+PlaceUtil()
+getPlacesByProximity(origin : GeoPt, proximityMeters : Double, count : int) : List
+getDistanceInMeters(location1 : GeoPt, location2 : GeoPt) : double

*Figure 15: PlaceUtil class diagram*

### 3.2.7 CheckInEndpoint.java

The CheckInEndpoint class (Appendix 6.3.14) allows the user to add check in event to the datastore. The class has one contructor and three methods. The *.checkIn()* method adds a CheckIn object to the datastore. The .*updateRecommendations() method makes a HTTP GET call to the Recommender class to generate new recommendations for*, which will make a query to the Recommender class.

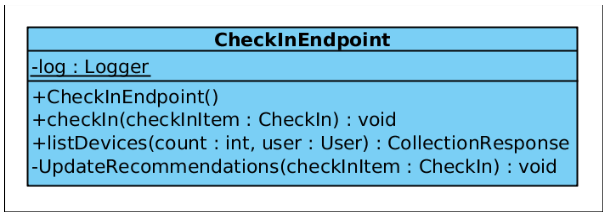

CheckInEndpoint
-log : Logger
+CheckInEndpoint()
+checkIn(checkInItem : CheckIn) : void
+listDevices(count : int, user : User) : CollectionResponse
-UpdateRecommendations(checkInItem : CheckIn) : void

*Figure 16: CheckInEndpoint class diagram*

### 3.2.8 MessagingEndpoint.java

The MessagingEndpoint class (Appendix 6.3.15) sends messages to the user from the Recommender. The class has one constructor and two methods. The .*sendMessage()* will issue messages to the devices in the service. The method .*getEventMessages()* works with the TagUtil class to provide updates to the user on the “Messages” tab of the Android application.

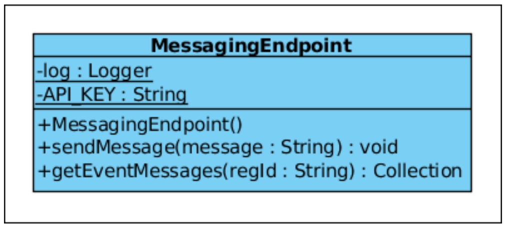
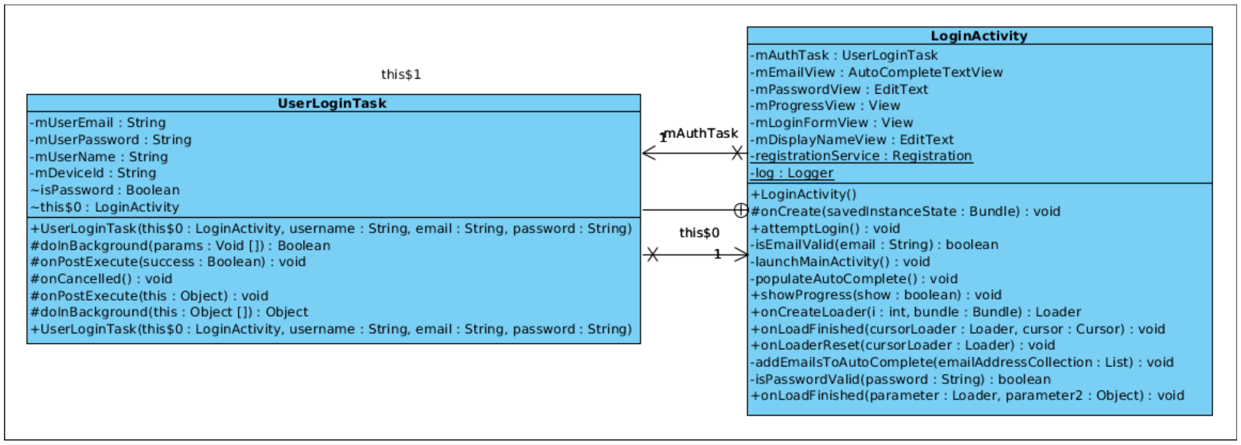

MessagingEndpoint
-log : Logger
-API_KEY : String
+MessagingEndpoint()
+sendMessage(message : String) : void
+getEventMessages(regId : String) : Collection

*Figure 17: MessagingEndpoint class diagram*

## 3.3 Android Application Design

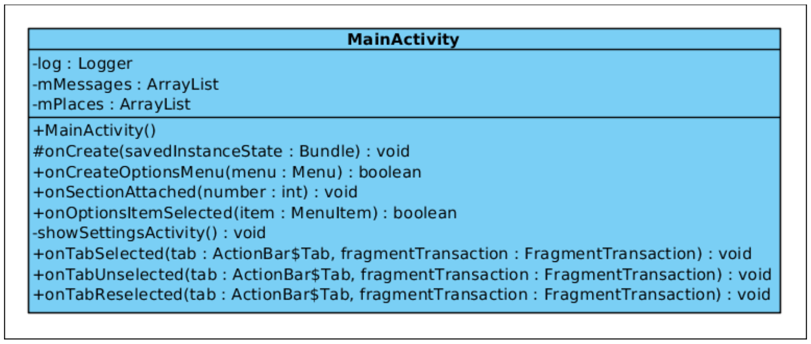
### 3.3.1 LoginActivity.java

The LoginActivity class (Appendix 6.3.16) acts as the main entry point for the android application, requiring that the user provide and email and optional password for accessing the service. The entry point for the class is the .*onCreate()* methods, with is called by the android operating system to instantiate the activity class with layout and configuration parameters specified by the source code. The method .*attemptLogin() is called when the sign in button is pressed. This* will submit the users credentials to the server by way of asynchronous call made by the UserLoginTask class .*doInBackground()* method. Upon success being returned in the .*onPostExecute()* method, the *LoginActivity.launchMainActivity()* method is called.


this$1
UserLoginTask
-mUserEmail : String
-mUserPassword : String
-mUserName : String
-mDeviceId : String
~isPassword : Boolean
~this$0 : LoginActivity
+UserLoginTask(this$0 : LoginActivity, username : String, email : String, password : String)
#doInBackground(params : Void []) : Boolean
#onPostExecute(success : Boolean) : void
#onCancelled() : void
#onPostExecute(this : Object) : void
#doInBackground(this : Object []) : Object
+UserLoginTask(this$0 : LoginActivity, username : String, email : String, password : String)
1
mAuthTask
this$0
1
LoginActivity
-mAuthTask : UserLoginTask
-mEmailView : AutoCompleteTextView
-mPasswordView : EditText
-mProgressView : View
-mLoginFormView : View
-mDisplayNameView : EditText
-registrationService : Registration
-log : Logger
+LoginActivity()
#onCreate(savedInstanceState : Bundle) : void
+attemptLogin() : void
-isEmailValid(email : String) : boolean
-launchMainActivity() : void
-populateAutoComplete() : void
+showProgress(show : boolean) : void
+onCreateLoader(i : int, bundle : Bundle) : Loader
+onLoadFinished(cursorLoader : Loader, cursor : Cursor) : void
+onLoaderReset(cursorLoader : Loader) : void
-addEmailsToAutoComplete(emailAddressCollection : List) : void
-isPasswordValid(password : String) : boolean
+onLoadFinished(parameter : Loader, parameter2 : Object) : void
*Figure 18: LoginActivity class*

### 3.3.2 MainActivity.java

The MainActivity class (Appendix 6.3.17) is a container for the MessageFragment and PlaceFragment classes. Those classes have callbacks and asynchonous functions that drive the applications functionality. The *.onCreate()* method instantiates the activities layout from XML in the layout directories. The *.onTabSelected()* method is called by default and inserts the fragments into the MainActivity class for display. The class


MainActivity
-log : Logger
-mMessages : ArrayList
-mPlaces : ArrayList
+MainActivity()
#onCreate(savedInstanceState : Bundle) : void
+onCreateOptionsMenu(menu : Menu) : boolean
+onSectionAttached(number : int) : void
+onOptionsItemSelected(item : MenuItem) : boolean
-showSettingsActivity() : void
+onTabSelected(tab : ActionBar$Tab, fragmentTransaction : FragmentTransaction) : void
+onTabUnselected(tab : ActionBar$Tab, fragmentTransaction : FragmentTransaction) : void
+onTabReselected(tab : ActionBar$Tab, fragmentTransaction : FragmentTransaction) : void
*Figure 19: MainActivity class*

### 3.3.3 MessageFragment.java

The MessageFragment class (Appendix 6.3.18) is a dynamic layout component derived from the android FragmentList class. Using the MessageAdapter class (Appendix 6.3.19) and data model class MessageItem (Appendix 6.3.20), a list of items is displayed to the user. Method MessageGetterTask.*OnCreate()* is called at instantiation of the class, and the MessageGetterTask asynchronous *.doInBackground()* method request the data to be manipulated. MessageAdapter instantiates the list from the data model object in its constructor, and the .getView() method converts the data into a displayable element.

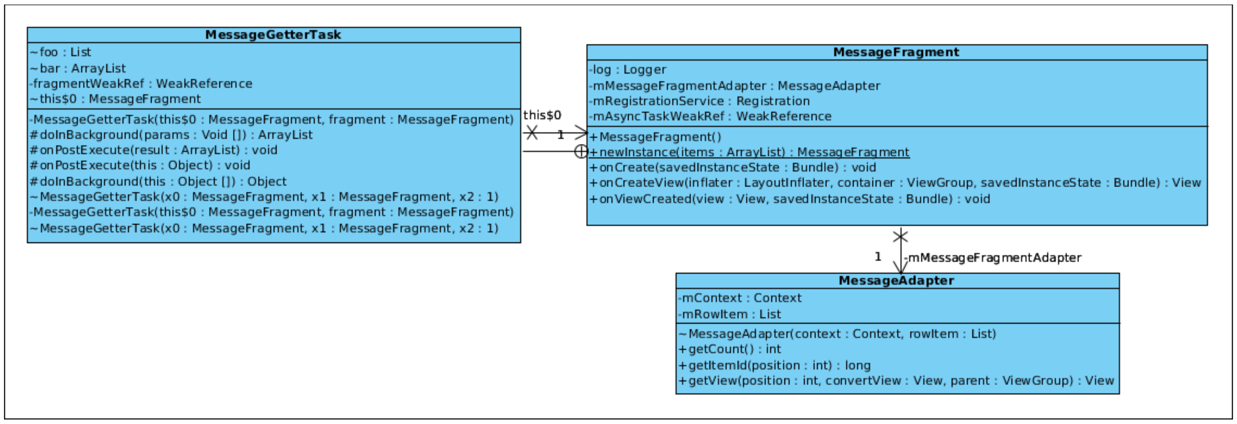

MessageGetterTask
~foo : List
~bar : ArrayList
-fragmentWeakRef : WeakReference
~this$0 : MessageFragment
-MessageGetterTask(this$0 : MessageFragment, fragment : MessageFragment)
#doInBackground(params : Void []) : ArrayList
#onPostExecute(result : ArrayList) : void
#onPostExecute(this : Object) : void
#doInBackground(this : Object []) : Object
~MessageGetterTask(x0 : MessageFragment, x1 : MessageFragment, x2 : 1)
-MessageGetterTask(this$0 : MessageFragment, fragment : MessageFragment)
~MessageGetterTask(x0 : MessageFragment, x1 : MessageFragment, x2 : 1)
this$0
1
MessageFragment
-log : Logger
-mMessageFragmentAdapter : MessageAdapter
-mRegistrationService : Registration
-mAsyncTaskWeakRef : WeakReference
+MessageFragment()
+newInstance(items : ArrayList) : MessageFragment
+onCreate(savedInstanceState : Bundle) : void
+onCreateView(inflater : LayoutInflater, container : ViewGroup, savedInstanceState : Bundle) : View
+onViewCreated(view : View, savedInstanceState : Bundle) : void
1
-mMessageFragmentAdapter
MessageAdapter
-mContext : Context
-mRowItem : List
~MessageAdapter(context : Context, rowItem : List)
+getCount() : int
+getItemId(position : int) : long
+getView(position : int, convertView : View, parent : ViewGroup) : View

*Figure 20: MessageFragment class diagram*

### 3.3.4 PlaceFragment.java

he PlaceFragment class (Appendix 6.3.19) is a dynamic layout component derived from the android FragmentList class. Using the PlaceAdapter (Appendix 6.3.22) class and data model class PlaceItem (Appendix 6.3.23), a list of items is displayed to the user. Method *.OnCreate()* is called at instantiation of the class, and the asynchronous *PlaceGetterTask.doInBackground()* method request the data to be manipulated. PlaceAdapter instantiates the list from the data model object in its constructor, and the *.getView()* method converts the data into a displayable element. The CheckInHelperTask handles request to check in to a place which is triggered when the *PlaceFragment.onListItemClick()* event is invoked.

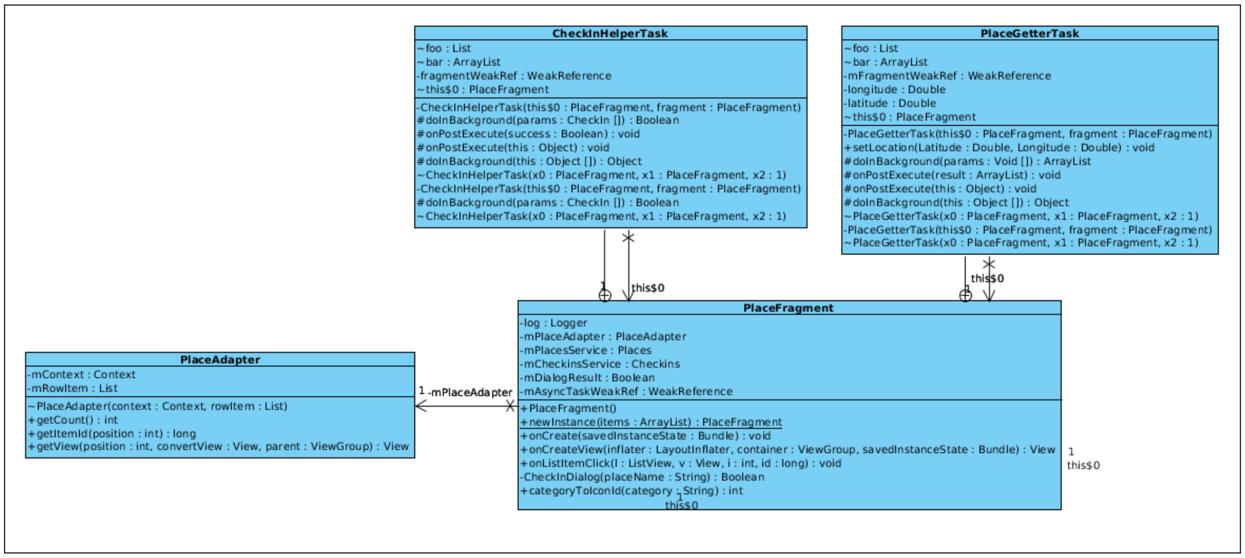

CheckInHelperTask
~foo : List
~bar : ArrayList
-fragmentWeakRef : WeakReference
~this$0 : PlaceFragment
-CheckInHelperTask(this$0 : PlaceFragment, fragment : PlaceFragment)
#doInBackground(params : CheckIn []) : Boolean
#onPostExecute(success : Boolean) : void
#onPostExecute(this : Object) : void
#doInBackground(this : Object []) : Object
~CheckInHelperTask(x0 : PlaceFragment, x1 : PlaceFragment, x2 : 1)
-CheckInHelperTask(this$0 : PlaceFragment, fragment : PlaceFragment)
#doInBackground(params : CheckIn []) : Boolean
~CheckInHelperTask(x0 : PlaceFragment, x1 : PlaceFragment, x2 : 1)
PlaceGetterTask
~foo : List
~bar : ArrayList
-mFragmentWeakRef : WeakReference
-longitude : Double
-latitude : Double
~this$0 : PlaceFragment
-PlaceGetterTask(this$0 : PlaceFragment, fragment : PlaceFragment)
+setLocation(Latitude : Double, Longitude : Double) : void
#doInBackground(params : Void []) : ArrayList
#onPostExecute(result : ArrayList) : void
#onPostExecute(this : Object) : void
#doInBackground(this : Object []) : Object
~PlaceGetterTask(x0 : PlaceFragment, x1 : PlaceFragment, x2 : 1)
-PlaceGetterTask(this$0 : PlaceFragment, fragment : PlaceFragment)
~PlaceGetterTask(x0 : PlaceFragment, x1 : PlaceFragment, x2 : 1)
1
this$0
1
this$0
PlaceFragment
-log : Logger
-mPlaceAdapter : PlaceAdapter
-mPlacesService : Places
-mCheckinsService : Checkins
-mDialogResult : Boolean
-mAsyncTaskWeakRef : WeakReference
+PlaceFragment()
+newInstance(items : ArrayList) : PlaceFragment
+onCreate(savedInstanceState : Bundle) : void
+onCreateView(inflater : LayoutInflater, container : ViewGroup, savedInstanceState : Bundle) : View
+onListItemClick(l : ListView, v : View, i : int, id : long) : void
-CheckInDialog(placeName : String) : Boolean
+categoryToIconId(category : String) : int
1
this$0
1
this$0
PlaceAdapter
-mContext : Context
-mRowItem : List
~PlaceAdapter(context : Context, rowItem : List)
+getCount() : int
+getItemId(position : int) : long
+getView(position : int, convertView : View, parent : ViewGroup) : View
1
-mPlaceAdapter

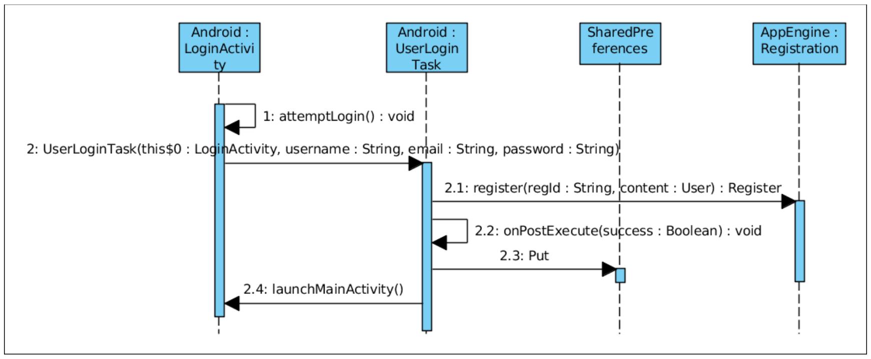
*Figure 21: PlaceFragment class diagram*

## 3.4 Loqale Operation

To interface with the recommender, new users must register their identity. Once a user account is created, the user can View the feed of other local users, check in to places, and request recommendations.

### 3.4.1 New User Registration

Upon launching the application, new users will be greeted with a login screen that will allow them to provide an email an password to create an account, or to login in using their Google+ identity to authenticate securely with the backend service.



Android : LoginActivity
Android : UserLoginTask
SharedPreferences
AppEngine : Registration
1: attemptLogin() : void
2: UserLoginTask(this$0 : LoginActivity, username : String, email : String, password : String)
2.1: register(regId : String, content : User) : Register
2.2: onPostExecute(success : Boolean) : void
2.3: Put
2.4: launchMainActivity()

*Figure 23: Login sequence diagram*

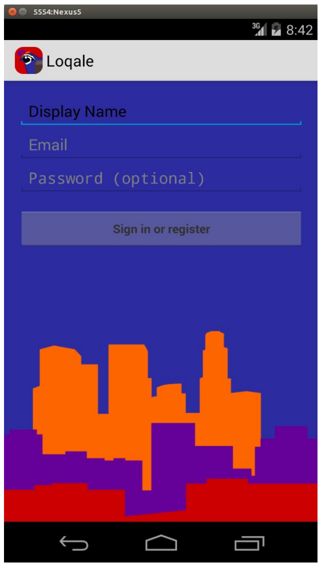
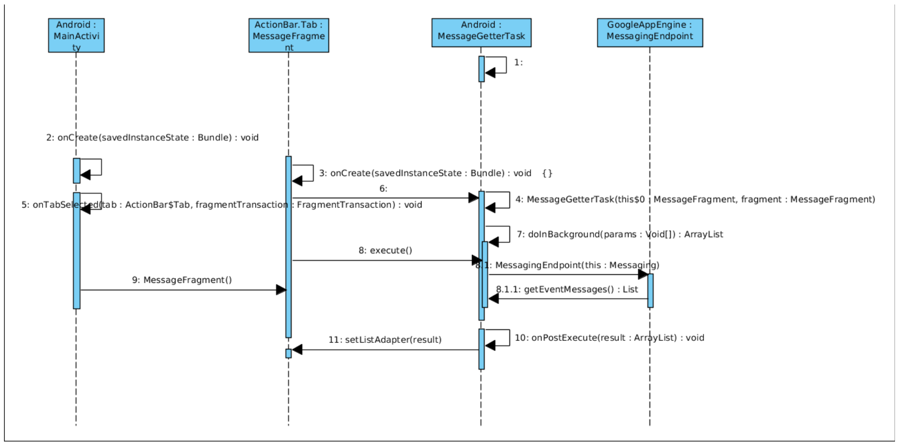


*Figure 22. Sign In Screen*

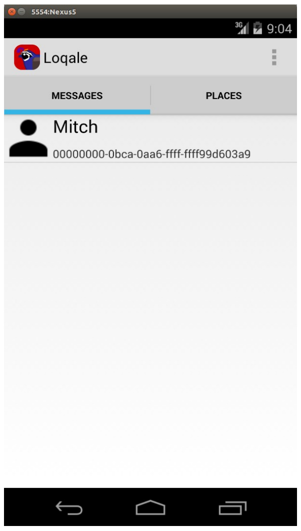
### 3.4.2 Main Screen

On the main screen, a asynchronous task that is part of the MainActivity class will retrieve feed items from the datastore within proximity to the user. These items will be a composite of users and places visited.



Android : MainActivity
ActionBar.Tab : MessageFragment
Android : MessageGetterTask
GoogleAppEngine : MessagingEndpoint
1:
2: onCreate(savedInstanceState : Bundle) : void
3: onCreate(savedInstanceState : Bundle) : void {}
4: MessageGetterTask(this$0 : MessageFragment, fragment : MessageFragment)
5: onTabSelected(tab : ActionBar$Tab, fragmentTransaction : FragmentTransaction) : void
6:
7: doInBackground(params : Void[]) : ArrayList
8: execute()
8.1: MessagingEndpoint(this : Messaging)
8.1.1: getEventMessages() : List
9: MessageFragment()
10: onPostExecute(result : ArrayList) : void
11: setListAdapter(result)

*Figure 24: Message retrieval sequence*



Mitch
00000000-0bca-0aa6-ffff-ffff99d603a9

*Figure 25. Main Screen*

### 3.4.4 Location Query

Upon switching to the Places tab, the user location will be queried from the device if available, and sent to the PlaceEndpoint *getPlacesByProximity()* function. The datastore will query the entire places list and process the locations in memory. The haversine formula for computing location distance will be applied and only the set of places in proximity are returned.

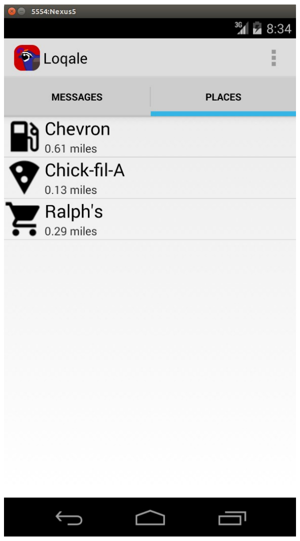
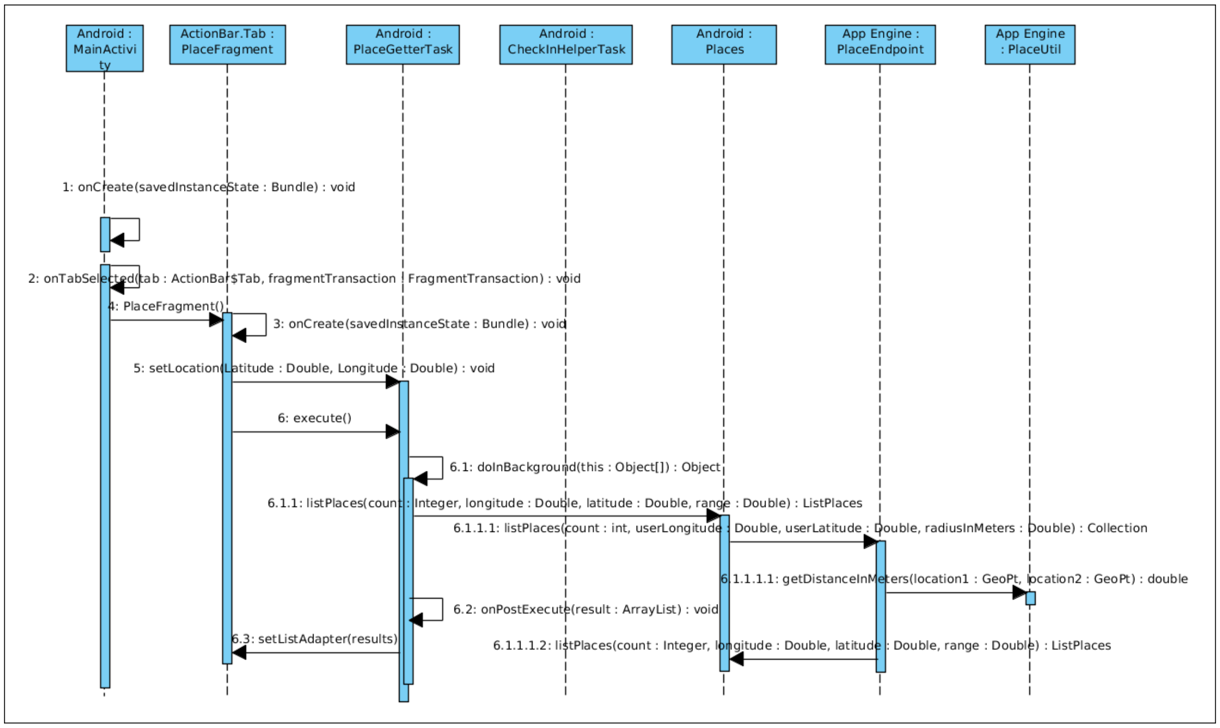


*Figure 26: Places Tab with nearby locations*

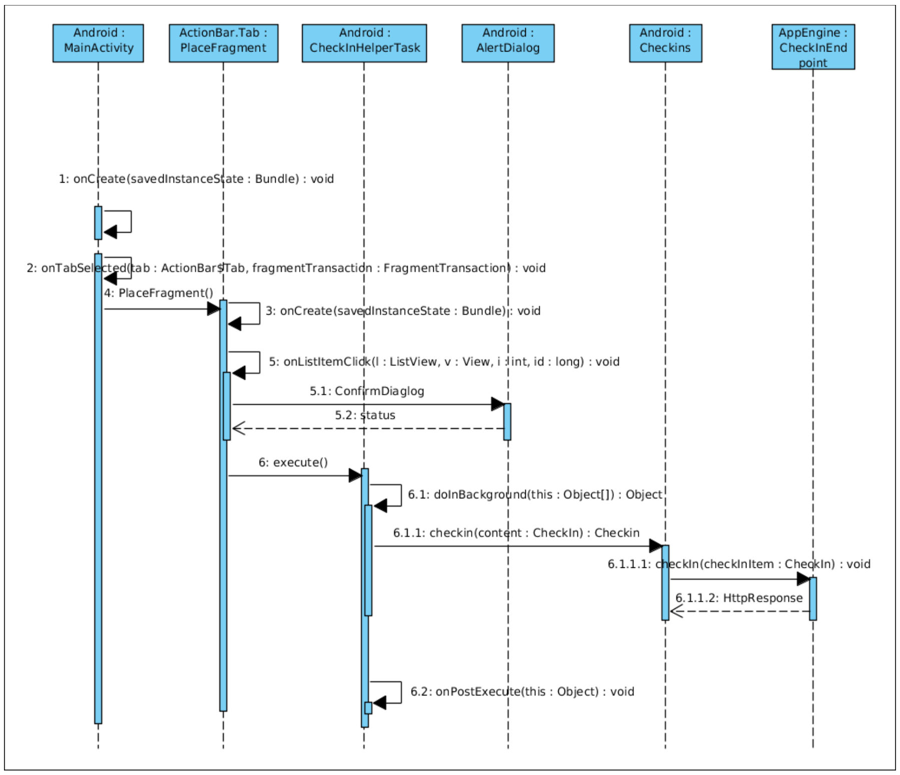


Android : MainActivity
ActionBar.Tab : PlaceFragment
Android : PlaceGetterTask
Android : CheckInHelperTask
Android : Places
App Engine : PlaceEndpoint
App Engine : PlaceUtil
1: onCreate(savedInstanceState : Bundle) : void
2: onTabSelected(tab : ActionBar$Tab, fragmentTransaction : FragmentTransaction) : void
4: PlaceFragment()
3: onCreate(savedInstanceState : Bundle) : void
5: setLocation(Latitude : Double, Longitude : Double) : void
6: execute()
6.1: doInBackground(this : Object[]) : Object
6.1.1: listPlaces(count : Integer, longitude : Double, latitude : Double, range : Double) : ListPlaces
6.1.1.1: listPlaces(count : int, userLongitude : Double, userLatitude : Double, radiusInMeters : Double) : Collection
6.1.1.1.1: getDistanceInMeters(location1 : GeoPt, location2 : GeoPt) : double
6.2: onPostExecute(result : ArrayList) : void
6.3: setListAdapter(results)
6.1.1.1.2: listPlaces(count : Integer, longitude : Double, latitude : Double, range : Double) : ListPlaces

*Figure 27: Place query sequence diagram*

### 3.4.5 Location Check In

After pressing the “Check In” button, the users location will be retrieved from their phone, using location services. An asynchronous call in the MainActivity Class will provide the location to the backend and will return places in near proximity. The user can then select to check into a place, which will be reflected in the user feed.


Android : MainActivity
ActionBar.Tab : PlaceFragment
Android : CheckInHelperTask
Android : AlertDialog
Android : Checkins
AppEngine : CheckInEndpoint
1: onCreate(savedInstanceState : Bundle) : void
2: onTabSelected(tab : ActionBar$Tab, fragmentTransaction : FragmentTransaction) : void
4: PlaceFragment()
3: onCreate(savedInstanceState : Bundle) : void
5: onListItemClick(l : ListView, v : View, i : int, id : long) : void
5.1: ConfirmDiaglog
5.2: status
6: execute()
6.1: doInBackground(this : Object[]) : Object
6.1.1: checkin(content : CheckIn) : Checkin
6.1.1.1: checkIn(checkInItem : CheckIn) : void
6.1.1.2: HttpResponse
6.2: onPostExecute(this : Object) : void

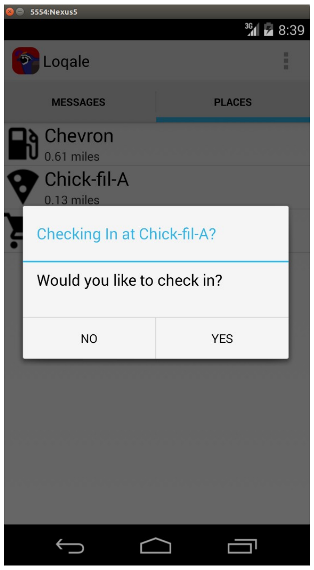


*Figure 28: Check In interaction dialog*

*Figure 29: Check In sequence diagram*

### 3.4.5 Recommendation Generation

Recommendation generation will occur with each user check in. The CheckInEndpoint class performs and HTTP GET on the Recommender servlet class, passing the user identification in the URL string of the request. For the selected user the Recommender then queries the user check in history and preferred categories, generating a list of similar places that will be filtered down to a set that will be inserted into the datastore. In this case the recommender only performs content filtering, as collaborative filtering required a user base that was not available in the time-frame this work was completed.

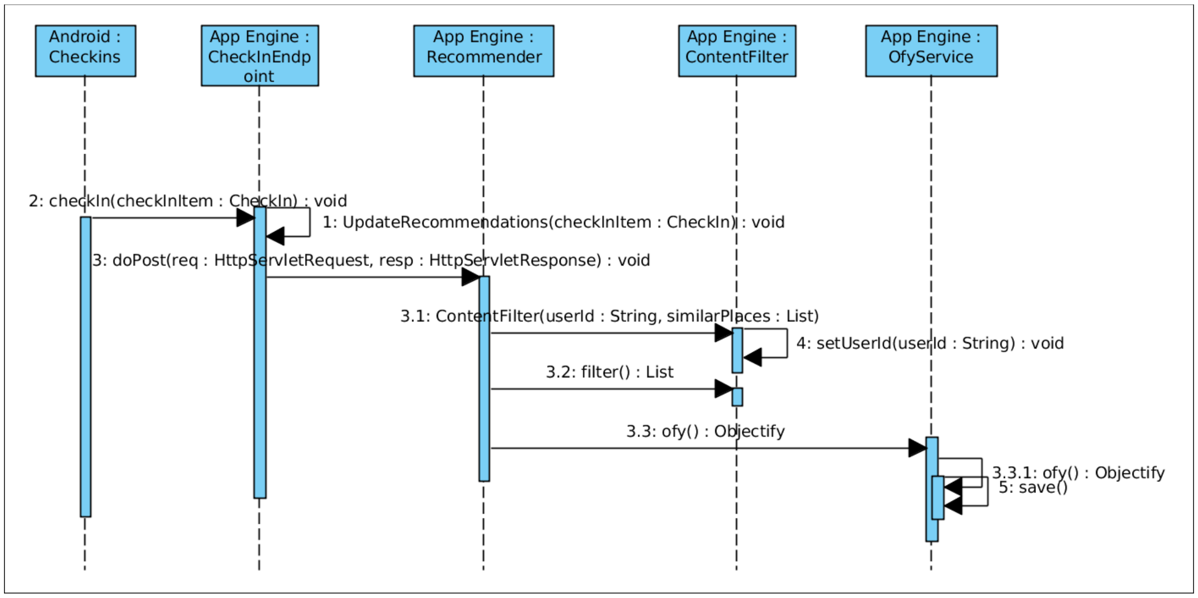
To overcome cold-start performance, the algorithm attempts to apply A/B testing of place recommendations in order to determine the users habits. This is done on the basis of some common qualities each place may have:

- cost (cheap/expensive)
- distance (close/far)
- expediency (fast/slow)
- size of business (small/corporation)

These qualities are stored in the *qualityMatrix* for each Place entity. To determine whether two places may represent some polarity to test the user with the recommeder witl compute the cross product of two locations, and if an identity matrix is generated, insert the results into the list of places to send the user.

$$\begin{bmatrix} c_i & d_i \\ e_i & s_i \end{bmatrix} \times \begin{bmatrix} c_j & d_j \\ e_j & s_j \end{bmatrix} = I_2$$

*Figure 30: Comparison of two places for polarity*

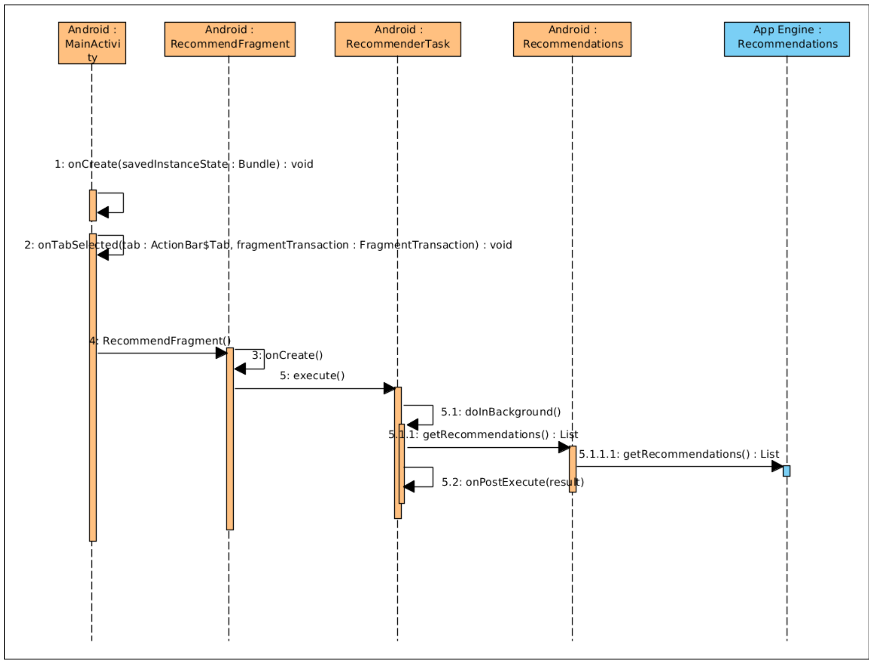


Android : Checkins
App Engine : CheckInEndpoint
App Engine : Recommender
App Engine : ContentFilter
App Engine : OfyService
2: checkIn(checkInItem : CheckIn) : void
1: UpdateRecommendations(checkInItem : CheckIn) : void
3: doPost(req : HttpServletRequest, resp : HttpServletResponse) : void
3.1: ContentFilter(userId : String, similarPlaces : List)
4: setUserId(userId : String) : void
3.2: filter() : List
3.3: ofy() : Objectify
3.3.1: ofy() : Objectify
5: save()

*Figure 31: Recommendation sequence diagram*

### 3.4.5 Request Recommendations

To request recommendations, the user navigates to the recommendation tab from the actionbar, which will execute an asynchronous Task call in the MainActivity Class to request recommendations for the user. In the event, no recommendations exist, the user will be notified. When recommendations are returned, the user has the option to favorite recommendations that they like in order to improve recommender accuracy. The classes labeled in orange reflect work that was being completed at the time of submission and will be left to future work to expand.


Android : MainActivity
Android : RecommendFragment
Android : RecommenderTask
Android : Recommendations
App Engine : Recommendations
1: onCreate(savedInstanceState : Bundle) : void
2: onTabSelected(tab : ActionBar$Tab, fragmentTransaction : FragmentTransaction) : void
4: RecommendFragment()
3: onCreate()
5: execute()
5.1: doInBackground()
5.1.1: getRecommendations() : List
5.1.1.1: getRecommendations() : List
5.2: onPostExecute(result)

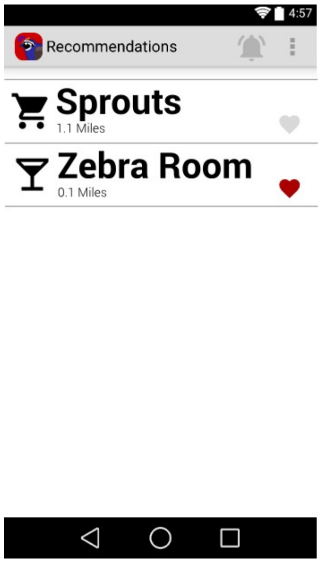


*Figure 33. Recommendation Screen*

*Figure 32: Recommendation Request sequence diagram*

## 4. Conclusions and Discussion

The performance of the recommender was not fully vetted, due to the enormity of the work and time constrains, but some analysis was considered regarding the overall design. The objectify datastore is limited to one relational operation per query, so multi-dimension bounding location queries were not possible. This required in memory manipulation of the place entities to compute distance, resulting in a O(n) accessing runtime. As future work a simplified cache of place identification keys in memory would be beneficial. The limited resources of the android front-end also reduced the number of calculations performed on the client side. The majority of calculations were done as post processing on entity queries by the backend. Entity classes were overloaded with members that could provide information in a ready-to-display manner for the fragment classes. The subjectiveness of the authors quality Matrix values should be considered, with a future method of computation based on sampling being of beneficial to future design improvements. These quality values could even see adjustment from categories selected by the user as theorized in the concluding discussion of [5] allowing dynamic variation of quality values.

## 5. References

[1] Gao, H., & Liu, H. (2014). Data Analysis on Location-Based Social Networks. In *Mobile Social Networking* (pp. 165-194). Springer New York.

[2] Zheng, Y., Zhang, L., Xie, X., & Ma, W. Y. (2009, April). Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th international conference on World wide web*(pp. 791-800). ACM.

[3] Noulas, A., Scellato, S., Mascolo, C., & Pontil, M. (2011). An Empirical Study of Geographic User Activity Patterns in Foursquare.*ICWSM*,*11*, 70-573.

[4] Rhee, I., Shin, M., Hong, S., Lee, K., Kim, S., Chong, S.: On the levy-walk nature of human mobility. IEEE/ACM Transactions on Networking (TON) 19(3), 630–643 (2011)

[5] Gao, H., Tang, J., & Liu, H. (2012, June). Exploring Social-Historical Ties on Location-Based Social Networks. In *ICWSM*.

[6] Shani, G., & Gunawardana, A. (2011). Evaluating recommendation systems. In *Recommender systems handbook* (pp. 257-297). Springer US.

[7] Humphreys, L. (2007). Mobile social networks and social practice: A case study of Dodgeball. *Journal of Computer-Mediated Communication*, *13*(1), 341-360.

[8] Gao, H., Tang, J., & Liu, H. (2014). Addressing the cold-start problem in location recommendation using geo-social correlations. *Data Mining and Knowledge Discovery*, 1-25.

[9] Breese, J. S., Heckerman, D., & Kadie, C. (1998, July). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence* (pp. 43-52). Morgan Kaufmann Publishers Inc.

[10] Zheng, Y., Zhang, L., Ma, Z., Xie, X., & Ma, W. Y. (2011). Recommending friends and locations based on individual location history. *ACM Transactions on the Web (TWEB)*, *5*(1), 5.

[11] Steiniger, S., Neun, M., & Edwardes, A. (2011). Foundations of Location Based Services Lesson 1 CartouCHe 1-Lecture Notes on LBS, V. 1.0.

[12] Glas, R. (2013). Breaking reality: Exploring pervasive cheating in Foursquare. *Transactions of the Digital Games Research Association*, *1*(1).

[13] Borne, K. (2014, February 20). Design Patterns for Recommendation Systems – Everyone Wants a Pony | MapR. Retrieved December 11, 2014, from https://www.mapr.com/blog/design-patterns-recommendation-systems-—-everyone-wants-pony#.VIljPerd_Qo

[14] Oksar, I. (2014, May). A Bluetooth signal strength based indoor localization method. In *Systems, Signals and Image Processing (IWSSIP), 2014 International Conference on* (pp. 251-254). IEEE.

[15] Quercia, D., Lathia, N., Calabrese, F., Di Lorenzo, G., & Crowcroft, J. (2010, December). Recommending social events from mobile phone location data. In*Data Mining (ICDM), 2010 IEEE 10th International Conference on* (pp. 971-976). IEEE.

## 6. Appendix
### 6.1 Proposal

TITLE: Location Based Services API for Android

Prepared by: Mitchell Francis

**Proposal:**

Location Based Services (LBS) enable users and organizations to selectively push information to customers meeting geospatial delivery requirements (time, proximity, etc). A primary use of LBS has been in the area of business marketing, but has seen usage in areas of political activism and within the military. Use cases in social networking could have value, as the social media industry is still in a growth phase. Though controversial for reasons of privacy, the concept of location based services is fascinating because I believe uses cases will become apparent with mobile computing wearables and the coming Internet of Things (IoT).

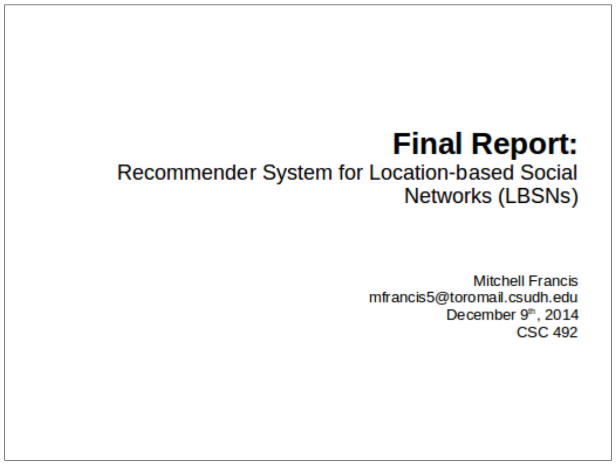
This project will explore the use of LBS as a social networking mechanism through the development of a mobile front-end application on the Google Android platform that will communicate  with a Google Apps Engine backend built in Java. The key features of this project application will be as follows. Map interface allowing the creation of pinned events. Messaging system allowing the delivery of imagery and text. Detection algorithm for proximity, time, and category on location pins. Ability to request historical pins. Privacy controls for pin and location viewing. Recommendation System for Places and People with like interest.

Mao Ye , Peifeng Yin , Wang-Chien Lee, Location recommendation for location-based social networks, Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, November 02-05, 2010, San Jose, California

Quercia, Daniele; Lathia, Neal; Calabrese, Francesco; Di Lorenzo, Giusy; Crowcroft, Jon (2010). "Recommending Social Events from Mobile Phone Location Data". 2010 IEEE International Conference on Data Mining. p. 971. doi:10.1109/ICDM.2010.152. ISBN 978-1-4244-9131-5.

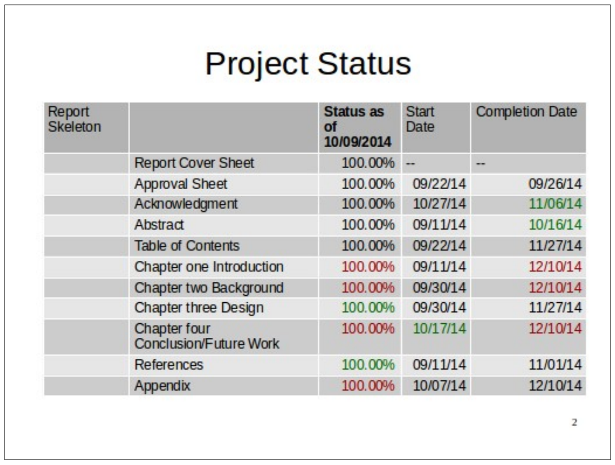
| | | |
|---|---|---|
| ***Faculty advisor*** | *Signature* | *Date* |
| *Committee member* | *Signature* | *Date* |
| *Committee member* | *Signature* | *Date* |

## 6.2 Presentation



**Final Report:**
Recommender System for Location-based Social Networks (LBSNs)

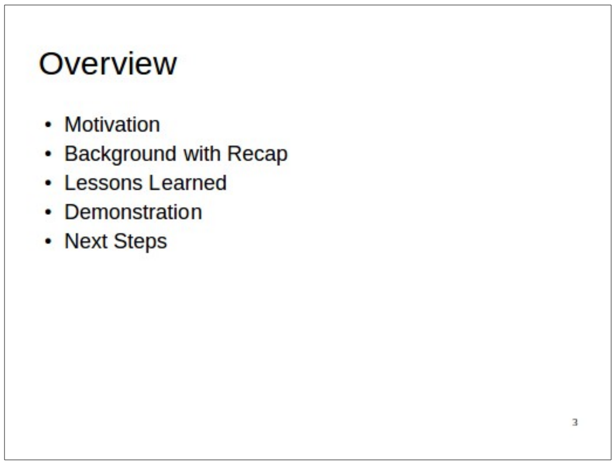
Mitchell Francis
mfrancis5@toromail.csudh.edu
December 9th, 2014
CSC 492



### Project Status

| Report Skeleton | | **Status as of 10/09/2014** | Start Date | Completion Date |
|---|---|---|---|---|
| | Report Cover Sheet | 100.00% | -- | -- |
| | Approval Sheet | 100.00% | 09/22/14 | 09/26/14 |
| | Acknowledgment | 100.00% | 10/27/14 | 11/06/14 |
| | Abstract | 100.00% | 09/11/14 | 10/16/14 |
| | Table of Contents | 100.00% | 09/22/14 | 11/27/14 |
| | Chapter one Introduction | 100.00% | 09/11/14 | 12/10/14 |
| | Chapter two Background | 100.00% | 09/30/14 | 12/10/14 |
| | Chapter three Design | 100.00% | 09/30/14 | 11/27/14 |
| | Chapter four Conclusion/Future Work | 100.00% | 10/17/14 | 12/10/14 |
| | References | 100.00% | 09/11/14 | 11/01/14 |
| | Appendix | 100.00% | 10/07/14 | 12/10/14 |

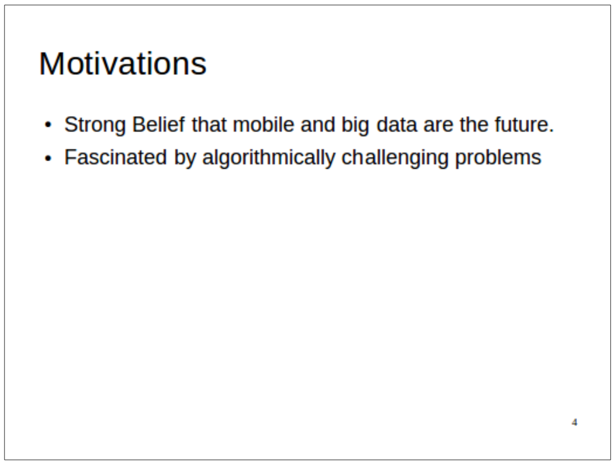
2



### Overview

- Motivation
- Background with Recap
- Lessons Learned
- Demonstration
- Next Steps

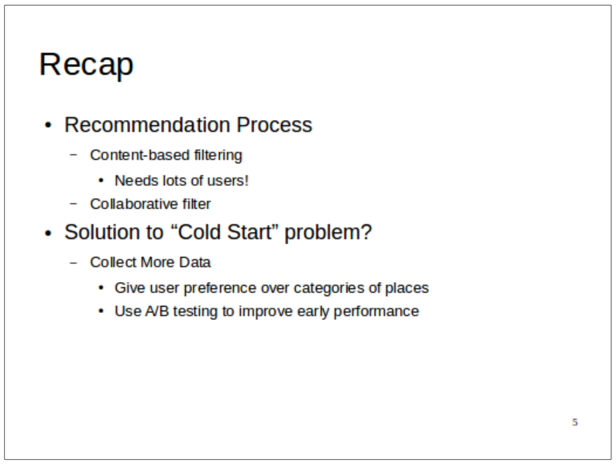
3



### Motivations

- Strong Belief that mobile and big data are the future.
- Fascinated by algorithmically challenging problems

4

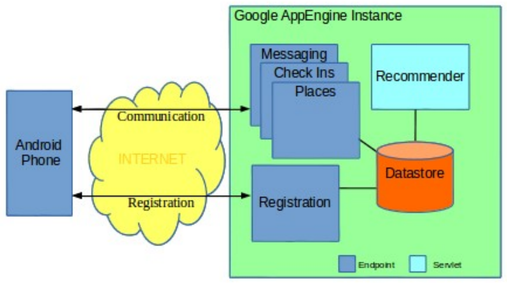


### Recap

- Recommendation Process
  - Content-based filtering
    - Needs lots of users!
  - Collaborative filter
- Solution to "Cold Start" problem?
  - Collect More Data
    - Give user preference over categories of places
    - Use A/B testing to improve early performance

5

### Recap (cont'd)*



*slide updated post review

6

## Challenges

- The Android learning curve
- Limitations of Objectify Persistence library

7

## Future Work

- Improve Android User Interface
- Encrypt communication and mobile settings.
- Refine Recommender “Quality Matrix” values
- Secure source of parseable data
- Complete testing in a production environment

8

Backup Slides

9

## Problem Statement

- Recommender systems are difficult to 'cold start' when very few user inputs are available. Can social background data be used to improve performance?
- With Android mobile users numbering in the millions, can a recommender be architected with a sufficient level of scalability?
- A suitable user experiences (UX) encourages users to participate in location tagging. What qualities will enhance this possibility?

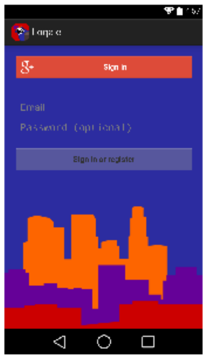
10

## Proposed Solution

- Use Google Android platform for user experience
  - Well documented API with growing development community
- Use Google App Engine for backend recommender system
  - Support for Java servlet technology, a mature web application framework
  - Supports integration with Android.
  - PaaS requires low Infrastructure maintenance. Can focus on runtime performance of application.
- Apply Gamefication to check-in process to encourage competition and participation.

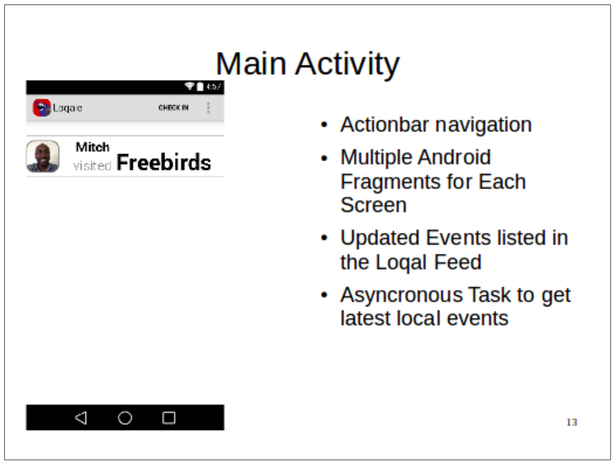
11

## Login Activity

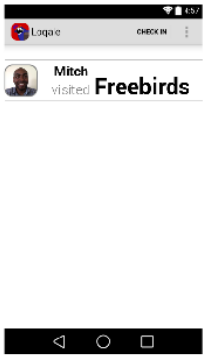


- User can login using Google+ or with email and password.

12



## Main Activity



- Actionbar navigation
- Multiple Android Fragments for Each Screen
- Updated Events listed in the Loqal Feed
- Asyncronous Task to get latest local events

13

## 6.3 Source Code

### 6.3.1 OfyService.java

```java
package net.nfiniteloop.loqale.backend;

import com.googlecode.objectify.Objectify;
import com.googlecode.objectify.ObjectifyFactory;
import com.googlecode.objectify.ObjectifyService;

/**
 * Objectify service wrapper so we can statically register our persistence classes
 * More on Objectify here : https://code.google.com/p/objectify-appengine/
 *
 */
public class OfyService {

    static {
        ObjectifyService.register(RegistrationRecord.class);

        // Recommendation classes
        ObjectifyService.register(User.class);
        ObjectifyService.register(Place.class);
        ObjectifyService.register(CheckIn.class);
        ObjectifyService.register(Tag.class);
        ObjectifyService.register(Recommendation.class);
    }

    public static Objectify ofy() {
        return ObjectifyService.ofy();
    }

    public static ObjectifyFactory factory() {
        return ObjectifyService.factory();
    }
}
```

### 6.3.2 RegistrationRecord.java

```java
package net.nfiniteloop.loqale.backend;

import com.googlecode.objectify.annotation.Entity;
import com.googlecode.objectify.annotation.Id;
import com.googlecode.objectify.annotation.Index;

import java.io.Serializable;

/** The Objectify object model for device registrations we are persisting */
@Entity
public class RegistrationRecord implements Serializable{

    @Id
    private Long id;

    @Index
    private String regId;

    private String regHash;

    public RegistrationRecord() {}

    public String getRegId() {
        return regId;
    }

    public void setRegId(String regId) {
        this.regId = regId;
    }

    public String getRegHash() {
        return regHash;
    }

    public void setRegHash(String regHash) {
        this.regHash = regHash;
    }
}
```

### 6.3.3 User.java

```
package net.nfiniteloop.loqale.backend;

import com.google.appengine.api.datastore.GeoPt;
import com.googlecode.objectify.annotation.Entity;
import com.googlecode.objectify.annotation.Id;
import com.googlecode.objectify.annotation.Index;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by vaek on 10/5/14.
 * TODO: Keys are not supported in the endpoint classes. Will resolve following the semester
 */
@Entity
public class User {
    @Id
    private Long id;
    @Index
    private String userId;
    private String displayName;
    private String hometown;
    private GeoPt location;
    private Double proximity;

    private List<String> categories = new ArrayList<String>();

    private String deviceId;

    public String getUserId() {
        return userId;
    }

    public String getDisplayName() {
        return displayName;
    }

    public String getHometown() {
        return hometown;
    }

    public GeoPt getLocation() {
        return location;
    }
```

```java
    public void setUserId(String userId) {
        this.userId = userId;
    }

    public void setDisplayName(String displayName) {
        this.displayName = displayName;
    }

    public void setHometown(String hometown) {
        this.hometown = hometown;
    }

    public void setLocation(GeoPt location) {
        this.location = location;
    }

    public void setCategories(List<String> userPreferredCategories){
        categories = userPreferredCategories;
    }

    public List<String> getCategories() {
        return categories;
    }

    public void setProximity(Double proximity) {
        this.proximity = proximity;
    }

    public Double getProximity() {
        return proximity;
    }

    public void setDeviceId(String deviceId) {
        this.deviceId = deviceId;
    }

    public String getDeviceId() {
        return deviceId;
    }
}
```

### 6.3.4 Place.java

```java
package net.nfiniteloop.loqale.backend;

import com.googlecode.objectify.annotation.Entity;
import com.googlecode.objectify.annotation.Id;
import com.googlecode.objectify.annotation.Index;
import com.googlecode.objectify.annotation.Serialize;

/**
 * Created by vaek on 10/5/14.
 */
@Entity
public class Place {
    @Id
    private Long id;
    @Index
    private String placeId;
    private String name;
    private String address;
    private Double longitude;
    private Double latitude;
    private String category;
    private String distance;

    @Serialize
    private double[][] qualityMatrix;

    public String getName(){
        return name;
    }

    public String getAddress(){
        return address;
    }

    public String getPlaceId(){
        return placeId;
    }

    public String getCategory() {
        return category;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public void setPlaceId(String placeId) {
        this.placeId = placeId;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public void setQualityMatrix(double[][] qualityMatrix) {
        this.qualityMatrix = qualityMatrix;
    }

    public double[][] getQualityMatrix() {
        return qualityMatrix;
    }

    public Double getLatitude() {
        return latitude;
    }

    public void setLatitude(Double latitude) {
        this.latitude = latitude;
    }

    public Double getLongitude() {
        return longitude;
    }

    public void setLongitude(Double longitude) {
        this.longitude = longitude;
    }

    public void setDistance(String distance) {
        this.distance = distance;
    }

    public String getDistance() {
        return distance;
    }
}
```

### 6.3.5 Tag.java

```
package net.nfiniteloop.loqale.backend;


import com.google.appengine.api.datastore.Blob;
import com.google.appengine.api.datastore.GeoPt;
import com.googlecode.objectify.annotation.Entity;
import com.googlecode.objectify.annotation.Id;
import com.googlecode.objectify.annotation.Index;

import java.util.Date;

/**
 * Created by vaek on 10/5/14.
 */
@Entity
public class Tag {
    @Id
    private Long id;
    @Index
    int tagCategory;
    private String tagId;
    private GeoPt location;
    private String text;
    private Blob image;
    private Date createDate;

    public String getTagId() {
        return tagId;
    }

    public GeoPt getLocation() {
        return location;
    }

    public String getText() {
        return text;
    }

    public Blob getImage() {
        return image;
    }

    public Date getCreateDate() {
        return createDate;
    }

    public void setTagId(String tagId) {
        this.tagId = tagId;
    }

    public void setLocation(GeoPt location) {
        this.location = location;
    }

    public void setText(String text) {
        this.text = text;
    }

    public void setImage(Blob image) {
        this.image = image;
    }

    public void setCreateDate(Date createDate) {
        this.createDate = createDate;
    }

    public void setTagCategory(int tagCategory) {
        this.tagCategory = tagCategory;
    }

    public int getTagCategory() {
        return tagCategory;
    }
}
```

### 6.3.6 CheckIn.java

```
package net.nfiniteloop.loqale.backend;

import com.googlecode.objectify.annotation.Entity;
import com.googlecode.objectify.annotation.Id;
import com.googlecode.objectify.annotation.Index;

import java.util.Date;

/**
 * Created by vaek on 10/5/14.
 */
@Entity
public class CheckIn {
    @Id
    private Long id;
    @Index
    private String checkInId;
    private String userId;
    private String placeId;
    private int rating;
    private Date checkInDate;

    public String getCheckInId() {
        return checkInId;
    }

    public String getUserId() {
        return userId;
    }

    public String getPlaceId() {
        return placeId;
    }

    public int getRating() {
        return rating;
    }

    public Date getCheckInDate() {
        return checkInDate;
    }
    public void setCheckInId(String checkInId) {
        this.checkInId = checkInId;
    }

    public void setUserId(String userId) {
        this.userId = userId;
    }

    public void setPlaceId(String placeId) {
        this.placeId = placeId;
    }

    public void setRating(int rating) {
        this.rating = rating;
    }

    public void setCheckInDate(Date checkInDate) {
        this.checkInDate = checkInDate;
    }
}
```

### 6.3.7 Recommendation.java

```
package net.nfiniteloop.loqale.backend;

import com.googlecode.objectify.annotation.Entity;
import com.googlecode.objectify.annotation.Id;
import com.googlecode.objectify.annotation.Index;

import java.util.Date;

/**
 * Created by vaek on 10/6/14.
 *
 * Recommendation Class for Recommender App
 * Generic container for Place, Tag and Friend Recommendations
 */
enum RecommendationType {PLACE, TAG, USER}

@Entity
class Recommendation {
    @Id
    private Long id;
    @Index
    private String recommendationId;
    private Date date;
    // TODO: Add private String explanation;
    private RecommendationType recommendationType;
    private String contentId;

    public Date getDate() {
        return date;
    }

    public RecommendationType getRecommendationType() {
        return recommendationType;
    }

    public String getRecommendationId() {
        return recommendationId;
    }
```

```java
    public String getContentId() {
        return contentId;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public void setRecommendationType(RecommendationType type) {
        this.recommendationType = type;
    }

    public void setRecommendationId(String recommendationId) {
        this.recommendationId = recommendationId;
    }

    public void setContentId(String contentId) {
        this.contentId = contentId;
    }
}
```

### 6.3.8 Recommender.java

```java
package net.nfiniteloop.loqale.backend;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
import java.util.Set;
import java.util.logging.Logger;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import Jama.Matrix;

import static net.nfiniteloop.loqale.backend.OfyService.ofy;

/**
 * Created by vaek on 10/5/14.
 */
public class Recommender extends HttpServlet{
    private static final long serialVersionUID = 1L;
    private static final int placeQueryLimit = 100;
    private static final Logger log = Logger.getLogger(Recommender.class.getName());
    private final int DEFAULT_FAR_DISTANCE_MULTI = 4;

    /* Working thoughts
          Think Thread Safety!!
          Think runtime!!
          Algorithm
            Build User Profile
            // get check ins
            Raw Recommendation Phase
                Is this a Cold Start?
                    Yes: Use Levy Flight Pattern to get raw recommendations
                        For each point on Path
                            Get establishment by nearest location and query by category
                            Return weighted results for category
                                (Make sure to handle collisions)

                    No: Use Profile to get raw recommendations
                        For each CheckIn
                            Get establishments category and query by category
                            Return weighted results for category
                        For each Tag (Future capability)
                            Get Label and query by label
                            return results by currency
            Filter Phase
                If Friendships
                    CF Friends to mark raw CheckIn recommendations
                    Repeat for Tags (future capability)
                If CheckIns
                    Build Levy Flight model and determine distance/frequency weights
                    Mark results with high distance/frequency weights

                If Tags (future capability)
                    Mark Results with high label frequency/or favorites
            Post Processing
                Generate Reason info for marked data
                Package marked recommendations
            Return recommendation package.
    */
    @Override
    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String userId = req.getParameter("userID");
        // Filtering module
        ContentFilter contentFilter;
        // container to store matching places
        List<Place> refinedPlaces = new LinkedList<Place>();
        List<Place> similarPlaces = new LinkedList<Place>();
        List<Recommendation> filterResults = new ArrayList<Recommendation>();
        List<User> recUser = ofy().load().type(User.class).filter("userId", userId).list();
        // this query should never be empty. Might want to check though...
        List<CheckIn> userCheckIns = ofy().load().type(CheckIn.class).filter("userId",
userId).list();
        if(userCheckIns.isEmpty()){
            // cold start logic
            // ok, so we don't know jack about this user, but we can send them a base set of
            // recommendations to jump start the engine and learn a bit about them. Of course
this
            // totally depends on them actually liking the recommendations served in order to
            // provide some dimensions to work with
            // so what are the dimensions that are easily testable?
            //    size of business (corporate chain, or small business
            //    quality/cost business (cheap/expensive)
            //     distance (close/far)
            // these can be used in combinations (far small), (far, small, cheap)
            //
            // cold start algorithm:
            //    get the categories of interest to the new user
            //    define 2 ranges to test for distance based on user settings (d, and 4d)
            //    at each range, query for places that fall into each place category ( ala A/B
testing)
            //    for range d look for places in same category,
            //      but with different qualities (exluding distance)
            //    pseudo
            //       for places in category
            //          place x cross place y = identity martrix
            //          add places to list
            //       invert matrix from  previous iteration
            //       for distance (3d < d < 4d), query locations by categories
            //       for places in category
            //          if place x cross place y = inverse identity martrix
            //             add place to list
            //    compile places and check for collisions (large business can have multiple
sites)
            //    send out cold recommendations
            Double proximityLimit = recUser.get(0).getProximity();
            List<String> preferredCategories = new ArrayList<String>();
            preferredCategories = recUser.get(0).getCategories();
            similarPlaces =
                    PlaceUtil.getPlacesByProximity(recUser.get(0).getLocation(),
                            proximityLimit,
                            placeQueryLimit);
            // TODO: Filter places by category. Or better, move category filtering into
ContentFilter
            //preferredCategories = recUser.get(0).getCategories();
            //for( String c : preferredCategories) {
            //    similarPlaces.addAll(ofy().load().type(Place.class).filter("category",
c).list());
            //}
            ListIterator<Place> iter = similarPlaces.listIterator();
            while ( iter.hasNext() ) {
                Place singlePlace = iter.next();
                similarPlaces.remove(0);
                // for each place, use the quality relationships to sam ple places for A/B test
                if (iter.hasNext()) {
                    Matrix qualityMatrix = new Matrix(singlePlace.getQualityMatrix());
                    Matrix pMatrix = new Matrix(iter.next().getQualityMatrix());
                    pMatrix = pMatrix.times(qualityMatrix);
                    if (!qualityMatrix.equals(pMatrix.inverse())) {
                        refinedPlaces.add(iter.previous());
                        refinedPlaces.add(singlePlace);
                    }
                }
                // List<Place> refinedPlaces = new LinkedList<Place>();
            }
            contentFilter =
                    new ContentFilter(recUser.get(0).getUserId(), refinedPlaces,
preferredCategories);
            filterResults = contentFilter.filter();
            ofy().save().entities(filterResults).now();
            proximityLimit *= DEFAULT_FAR_DISTANCE_MULTI;
            similarPlaces =
                    PlaceUtil.getPlacesByProximity(recUser.get(0).getLocation(),
                            proximityLimit,
                            placeQueryLimit);
            /*
                      ListIterator<Place> iter = similarPlaces.listIterator();
            while ( iter.hasNext() ) {
                Place singlePlace = iter.next();
                similarPlaces.remove(0);
                // for each place, use the quality relationships to sam ple places for A/B test
                if (iter.hasNext()) {
                    Matrix qualityMatrix = new Matrix(singlePlace.getQualityMatrix());
                    Matrix pMatrix = new Matrix(iter.next().getQualityMatrix());
                    pMatrix = pMatrix.times(qualityMatrix);
                    if (!qualityMatrix.equals(pMatrix.inverse())) {
                        refinedPlaces.add(iter.previous());
                        refinedPlaces.add(singlePlace);
                    }
                }
                // List<Place> refinedPlaces = new LinkedList<Place>();
            }
             */
        }
        else {

            Set<String> categories = new HashSet<String>();
            // get the prefereded categories from the user profile. those will have higher weight
            // for each checkin, get the category using the placeId, and look for other places in
range
            // of user home location
            // don't forget about recommendations that were liked! use that list to filter out
            // places that don't match on dimensional criteria
            for (CheckIn c : userCheckIns) {
                List<Place> checkInPlace = ofy().load().type(Place.class)
                        .filter("placeId", c.getPlaceId()).list();
                categories.add(checkInPlace.get(0).getCategory());

            }
            // first we handle the case where data exist in the db
            for (String s : categories) {
                // make sure you fix this! you could oull the whole db!
                similarPlaces.addAll(ofy().load().type(Place.class).filter("category",
s).list());
            }

            // ok, got the categories. build a list of raw places
            // challenges:
            // i need to limit the query to locations within range of the
            // friends
            // Feed the list of places to the filter
            // Filter recFilter = new RecommendationFilter( similarPlaces );
            // rec filter removes places the user has checked in to and previous recommendations
            // refinedPlaces = recFilter.getResults()

            // friendfilter iterates over friends checkins and does a placeId count.
            // looks for collisions and mark common ids
            // with higher weight
            // Filter FriendFilter = FriendFilter( refinedPlaces );
            // List recommendedPlaces = new LinkedList<Places>();
            // recommendedPlaces = refinedPlaces.getPlaces();

            // send list of recommended places to user
            ofy().save().entities(similarPlaces).now();
        }
        // send  filterResults the user

    }
}
```

### 6.3.9 ContentFilter.java

```
package net.nfiniteloop.loqale.backend;

import java.sql.Date;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.UUID;

/**
 * Created by vaek on 11/2/14.
 */
public class ContentFilter implements Filter {
    public ContentFilter(String userId, List<Place> similarPlaces, List<String> userCategories) {
        setUserId(userId);
        mPlaceList = new ArrayList<Place>(similarPlaces);
        mCategotries.addAll(userCategories);
    }

    @Override
    public List<Recommendation> filter() {
        int count = 0;
        List<Recommendation> returnList = new ArrayList<Recommendation>();
        while( count != 10 ){
            for (Place p : mPlaceList){
                if(!mCategotries.contains(p.getCategory())){
                    continue;
                }
                Recommendation newRec = new Recommendation();
                newRec.setContentId(UUID.randomUUID().toString());
                newRec.setRecommendationType(RecommendationType.PLACE);
                newRec.setContentId(p.getPlaceId());
                newRec.setDate(new Date(System.currentTimeMillis()));
                returnList.add(newRec);
            }
        }
        // Filter will remove elements that are not distance similar to previous check-ins
        // find hubs

        // compute distance from hubs

        // for each check-in id, compute the lowest distance to a hub,  consider temporal
sequence

        // with the hub data computed process m

        return returnList;
    }

    @Override
    public void setUserId(String userId) {
        mUserId = userId;
    }

    private String mUserId;
    private List<Place> mPlaceList;
    private HashSet<String> mCategotries;
}
```

### 6.3.10 TagUtil.java

```
package net.nfiniteloop.loqale.backend;

import static net.nfiniteloop.loqale.backend.OfyService.ofy;

/**
 * Created by vaek on 12/9/14.
 */
```

```
public class TagUtil {
    public static void recordEvent(int eventType, String eventMsg, User user) {
        Tag event = new Tag();
        // 1:user_event, 2:place_event, 3_recommendation_event
        event.setTagCategory(eventType);
        event.setText(eventMsg);
        event.setTagId(user.getUserId());
        ofy().save().entity(event).now();
    }

}
```

### 6.3.11 RegistrationEndpoint.java

```
package net.nfiniteloop.loqale.backend;

import com.google.api.server.spi.config.Api;
import com.google.api.server.spi.config.ApiMethod;
import com.google.api.server.spi.config.ApiNamespace;

import java.util.ArrayList;
import java.util.Collection;
import java.util.LinkedList;
import java.util.List;
import java.util.logging.Logger;

import javax.inject.Named;

import static net.nfiniteloop.loqale.backend.OfyService.ofy;

/**
 * A registration endpoint class we are exposing for a device's GCM registration id on the
backend
 *
 * For more information, see
 * https://developers.google.com/appengine/docs/java/endpoints/
 *
 * NOTE: This endpoint does not use any form of authorization or
 * authentication! If this app is deployed, anyone can access this endpoint! If
 * you'd like to add authentication, take a look at the documentation.
 */
@Api(name = "registration", version = "v1", namespace =
    @ApiNamespace(ownerDomain = "backend.loqale.nfiniteloop.net",
        ownerName = "backend.loqale.nfiniteloop.net", packagePath=""))
public class RegistrationEndpoint {

    private static final Logger log = Logger.getLogger(RegistrationEndpoint.class.getName());

    /**
     * Register a device to the backend
     *
     * @param regId The Google Cloud Messaging registration Id to add
     */
    @ApiMethod(name = "register")
    public void registerDevice(@Named("regId") String regId, User userInfo) {
        if (findRecord(regId) != null) {
            log.info("Device " + regId + " already registered, skipping register");
            return;
        }

        RegistrationRecord record = new RegistrationRecord();
        record.setRegId(regId);
        userInfo.setUserId(regId);
        ofy().save().entity(record).now();
        ofy().save().entity(userInfo).now();

        //TagUtil.recordEvent(1, "Welcome to Loqale!", userInfo);

    }

    /**
     * Unregister a device from the backend
     *
     * @param regId The Google Cloud Messaging registration Id to remove
     */
    @ApiMethod(name = "unregister")
    public void unregisterDevice(@Named("regId") String regId) {
        RegistrationRecord record = findRecord(regId);
        if(record == null) {
            log.info("Device " + regId + " not registered, skipping unregister");
            return;
        }
        ofy().delete().entity(record).now();
    }

    /**
     * Return a collection of registered devices
     *
     * @param count The number of devices to list
     * @return a list of Google Cloud Messaging registration Ids
     */
    @ApiMethod(name = "listDevices")
    public Collection<RegistrationRecord> listDevices(@Named("count") int count) {
        List<RegistrationRecord> records =
ofy().load().type(RegistrationRecord.class).limit(count).list();
        Collection<RegistrationRecord> col = new LinkedList<RegistrationRecord>();
        col.addAll(records);
        //return CollectionResponse.<RegistrationRecord>builder().setItems(records).build();
        return col;
    }

    private RegistrationRecord findRecord(String regId) {
        return ofy().load().type(RegistrationRecord.class).filter("regId", regId).first().now();
    }

}
```

### 6.3.12 PlaceEndpoint.java

```
package net.nfiniteloop.loqale.backend;

import com.google.api.server.spi.config.Api;
import com.google.api.server.spi.config.ApiMethod;
import com.google.api.server.spi.config.ApiNamespace;
import com.google.api.server.spi.response.CollectionResponse;
import com.google.appengine.api.datastore.GeoPt;

import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.logging.Logger;

import javax.inject.Named;
```

```
/**
 * Created by vaek on 11/16/14.
 */
@Api(name = "places", version = "v1", namespace =
    @ApiNamespace(ownerDomain = "backend.loqale.nfiniteloop.net", \
            ownerName = "backend.loqale.nfiniteloop.net", packagePath=""))
public class PlaceEndpoint {
    private static final Logger log = Logger.getLogger(PlaceEndpoint.class.getName());

    /**
     * Return a collection of registered devices
     *
     * @param count The number of devices to list
     * @param userLongitude The coordinate of user device
     * @param userLatitude  The coordinate of user deivce
     * @param radiusInMeters Proximity to search
     * @return a list of Google Cloud Messaging registration Ids
     */
    @ApiMethod(name = "listPlaces")
    public Collection<Place> listPlaces(@Named("count") int count,
            @Named("longitude") Double userLongitude, @Named("latitude") Double userLatitude,
            @Named("range") Double radiusInMeters) {
        DataInjector injector = new DataInjector();
        injector.createDataSet();

        // Convert Strings into floats
        float longitude, latitude;
        longitude = userLongitude.floatValue();
        latitude = userLatitude.floatValue();
        GeoPt location = new GeoPt(latitude, longitude);

        List<Place> places = PlaceUtil.getPlacesByProximity(location,radiusInMeters,count);
        log.info(places.size() + " places returned ");

        return places;
        //return CollectionResponse.<Place>builder().setItems(places).build();
    }
}
```

### 6.3.13 PlaceUtil.java

```
package net.nfiniteloop.loqale.backend;

import com.google.appengine.api.datastore.GeoPt;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import static net.nfiniteloop.loqale.backend.OfyService.ofy;
/**
 * Created by vaek on 12/1/14.
 */


public class PlaceUtil {
    final static int radiusEarthMeters = 6378137;
    final static double metersToMiles = 0.000621371;
    public static List<Place> getPlacesByProximity(GeoPt origin, Double proximityMeters, int
count){
        List<Place> places = new ArrayList<Place>();
        double deltaLat = (float) proximityMeters.doubleValue() / radiusEarthMeters;
        double deltaLon =
                (float) proximityMeters.doubleValue() /
                        ( radiusEarthMeters * Math.cos(Math.toRadians(origin.getLongitude())) );
        float originLat = origin.getLatitude();
        float originLon = origin.getLongitude();

        //hmmm... inequality filters are limited to one operation
        // this would be expensive on a large DB...
        List<Place> results = new ArrayList<Place>();
        places.addAll(ofy().load().type(Place.class).limit(count).list());

        for(Iterator<Place> iter = places.iterator(); iter.hasNext();) {
            Place p = iter.next();
            GeoPt placeLocation =
                    new GeoPt(p.getLatitude().floatValue(), p.getLongitude().floatValue());
            double distance = PlaceUtil.getDistanceInMeters(origin,placeLocation);
            if(distance <= proximityMeters){
                // covert distance to string and pack into payload for device
                double distanceMiles = distance * metersToMiles;
                String distanceStr = String.format("%.2f", distanceMiles);
                p.setDistance(distanceStr + " miles");
                results.add(p);
            }
        }
        return results;
    }

    public static double getDistanceInMeters(GeoPt location1, GeoPt location2){

        // derived from haversine
        double radiusEarth = 6378137; // meters
        double loc1Lat = Math.toRadians(location1.getLatitude());
        double loc2Lat = Math.toRadians(location2.getLatitude());
        double latDelta = Math.toRadians(location2.getLatitude()-location1.getLatitude());
        double lonDelta = Math.toRadians(location2.getLongitude() - location1.getLongitude());

        double a = Math.sin(latDelta/2) * Math.sin(latDelta/2) +
                Math.cos(loc1Lat) * Math.cos(loc2Lat) *
                        Math.sin(lonDelta/2) * Math.sin(lonDelta/2);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

        double distance = radiusEarth * c;

        return distance;
    }


}
```

### 6.3.14 CheckInEndpoint.java

```
package net.nfiniteloop.loqale.backend;

import com.google.api.server.spi.config.Api;
import com.google.api.server.spi.config.ApiMethod;
import com.google.api.server.spi.config.ApiNamespace;
import com.google.api.server.spi.response.CollectionResponse;
import com.google.appengine.api.taskqueue.QueueFactory;
import com.google.appengine.api.taskqueue.TaskOptions;
import com.sun.javafx.tk.Toolkit;

import java.util.List;
import java.util.Queue;
import java.util.logging.Logger;
import javax.inject.Named;
import javax.jws.soap.SOAPBinding;

import static com.google.appengine.api.taskqueue.TaskOptions.Builder.withUrl;
import static net.nfiniteloop.loqale.backend.OfyService.ofy;

/**
 * Created by vaek on 11/16/14.
 */
@Api(name = "checkins", version = "v1", namespace =
    @ApiNamespace(ownerDomain = "backend.loqale.nfiniteloop.net",
            ownerName = "backend.loqale.nfiniteloop.net", packagePath=""))
public class CheckInEndpoint {
    private static final Logger log = Logger.getLogger(RegistrationEndpoint.class.getName());

    /**
     * CheckIn to a place
     *
     */
    @ApiMethod(name = "checkin")
    public void checkIn( CheckIn checkInItem) {
        ofy().save().entity(checkInItem).now();
        Place checkInPlace;
        checkInPlace =
                ofy().load().type(Place.class)
                        .filter("placeId", checkInItem.getPlaceId()).list().get(0);
        User checkedInUser =
                ofy().load().type(User.class)
                        .filter("placeId", checkInItem.getUserId()).list().get(0);

        TagUtil.recordEvent(2, "Checked In at " + checkInPlace.getName(), checkedInUser);
        UpdateRecommendations(checkInItem);
    }

    /**
     * Return a collection of registered devices
     *
     * @param count The number of devices to list
     * @return a list of Google Cloud Messaging registration Ids
     */
    @ApiMethod(name = "listcheckins")
    public CollectionResponse<CheckIn> listDevices(@Named("count") int count, User user) {
        List<CheckIn> checkIns =
                ofy().load().type(CheckIn.class).limit(count)
                        .filter("userId", user.getUserId()).list();
        return CollectionResponse.<CheckIn>builder().setItems(checkIns).build();
    }

    private void UpdateRecommendations(CheckIn checkInItem) {
        log.info("passing to recommender");
        com.google.appengine.api.taskqueue.Queue queue = QueueFactory.getDefaultQueue();

        try {
            queue.add(withUrl("/recommend").param("userId", checkInItem.getUserId()));
        }
        catch(RuntimeException e){
            log.severe("something went wrong");

        }

    }

}
```

### 6.3.15 MessagingEndpoint.java

```
package net.nfiniteloop.loqale.backend;

import com.google.android.gcm.server.Constants;
import com.google.android.gcm.server.Message;
import com.google.android.gcm.server.Result;
import com.google.android.gcm.server.Sender;
import com.google.api.server.spi.config.Api;
import com.google.api.server.spi.config.ApiMethod;
import com.google.api.server.spi.config.ApiNamespace;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.logging.Logger;
import javax.inject.Named;

import static net.nfiniteloop.loqale.backend.OfyService.ofy;
// This is code I extended from an android studio template. I added getEventMessages
// to support the Main feed Android application
/**
 * An endpoint to send messages to devices registered with the backend
 *
 * For more information, see
 * https://developers.google.com/appengine/docs/java/endpoints/
 *
 * NOTE: This endpoint does not use any form of authorization or
 * authentication! If this app is deployed, anyone can access this endpoint! If
 * you'd like to add authentication, take a look at the documentation.
 */
@Api(name = "messaging", version = "v1", namespace =
    @ApiNamespace(ownerDomain = "backend.loqale.nfiniteloop.net",
            ownerName = "backend.loqale.nfiniteloop.net", packagePath=""))
public class MessagingEndpoint {
    private static final Logger log = Logger.getLogger(MessagingEndpoint.class.getName());

    /** Api Keys can be obtained from the google cloud console */
    private static final String API_KEY = System.getProperty("gcm.api.key");

    /**
     * Send to the first 10 devices
     *
     * @param message The message to send
     */
    public void sendMessage(@Named("message") String message) throws IOException {
        if(message == null || message.trim().length() == 0) {
            log.warning("Not sending message because it is empty");
            return;
        }
        // crop longer messages
        if (message.length() > 1000) {
            message = message.substring(0, 1000) + "[...]";
        }
        Sender sender = new Sender(API_KEY);
        Message msg = new Message.Builder().addData("message", message).build();
        List<RegistrationRecord> records =
                ofy().load().type(RegistrationRecord.class).limit(10).list();
        for(RegistrationRecord record : records) {
            Result result = sender.send(msg, record.getRegId(), 5);
            if (result.getMessageId() != null) {
                log.info("Message sent to " + record.getRegId());
                String canonicalRegId = result.getCanonicalRegistrationId();
                if (canonicalRegId != null) {
                    // if the regId changed, we have to update the datastore
                    log.info("Registration Id changed for " + record.getRegId()
                            + " updating to " + canonicalRegId);
                    record.setRegId(canonicalRegId);
                    ofy().save().entity(record).now();
                }
            } else {
                String error = result.getErrorCodeName();
                if (error.equals(Constants.ERROR_NOT_REGISTERED)) {
                    log.warning("Registration Id " + record.getRegId()
                            + " no longer registered with GCM, removing from datastore");
                    // if the device is no longer registered with Gcm, remove it from the
datastore
                    ofy().delete().entity(record).now();
                }
                else {
                    log.warning("Error when sending message : " + error);
                }
            }
        }
    }

    @ApiMethod(name = "getMessages")
    public Collection<Tag> getEventMessages(@Named("regId") String regId) {
        List<Tag> messages = new ArrayList<Tag>();

        messages = ofy().load().type(Tag.class).filter("regId ==", regId).list();
        return messages;
    }
}
```

### 6.3.16 LoginActivity.java

```
package nfiniteloop.net.loqale;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.annotation.TargetApi;
import android.app.Activity;
import android.app.LoaderManager;
import android.content.Context;
import android.content.CursorLoader;
import android.content.Intent;
import android.content.Loader;
import android.content.SharedPreferences;
import android.database.Cursor;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.provider.ContactsContract;
```

```java
import android.telephony.TelephonyManager;
import android.text.TextUtils;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.extensions.android.json.AndroidJsonFactory;
import com.google.api.client.googleapis.services.AbstractGoogleClientRequest;
import com.google.api.client.googleapis.services.GoogleClientRequestInitializer;

import net.nfiniteloop.loqale.backend.registration.Registration;
import net.nfiniteloop.loqale.backend.registration.model.User;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import java.util.logging.Logger;

// This is a skeleton class that I extended to support authenticating with backend
// service. The template included a form for email and password. I extended
// it with asynchronous call to backend, validation of stored creditentials and theming

/**
 * A login screen that offers login via email/password and via Google+ sign in.
 * <p/>
 * ************ IMPORTANT SETUP NOTES: ************
 * In order for Google+ sign in to work with your app, you must first go to:
 * https://developers.google.com/+/mobile/android/getting-started#step_1_enable_the_google_api
 * and follow the steps in "Step 1" to create an OAuth 2.0 client for your package.
 * // TODO: I was not able to extend this class to support oauth in a timely fashion.

 */
public class LoginActivity extends Activity implements LoaderManager.LoaderCallbacks<Cursor> {

    /**
     * Keep track of the login task to ensure we can cancel it if requested.
     */
    private UserLoginTask mAuthTask = null;


    // UI references.
    private AutoCompleteTextView mEmailView;
    private EditText mPasswordView;
    private View mProgressView;
    private View mLoginFormView;
    private EditText mDisplayNameView;

    private static Registration mRegistrationService = null;
    private static final Logger log = Logger.getLogger(LoginActivity.class.getName());

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        SharedPreferences prefs =
                getSharedPreferences(LoqaleConstants.PREFS_NAME, Context.MODE_PRIVATE);
        String deviceId = prefs.getString("deviceId", "");

        // Set up the login form
        mDisplayNameView = (EditText) findViewById(R.id.user_disp_name);


        mEmailView = (AutoCompleteTextView) findViewById(R.id.email);
        populateAutoComplete();

        mPasswordView = (EditText) findViewById(R.id.password);
        mPasswordView.setOnEditorActionListener(new TextView.OnEditorActionListener() {
            @Override
            public boolean onEditorAction(TextView textView, int id, KeyEvent keyEvent) {
                if (id == R.id.login || id == EditorInfo.IME_NULL) {
                    attemptLogin();
                    return true;
                }
                return false;
            }
        });

        Button mEmailSignInButton = (Button) findViewById(R.id.email_sign_in_button);
        mEmailSignInButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                attemptLogin();
            }
        });

        mLoginFormView = findViewById(R.id.login_form);
        mProgressView = findViewById(R.id.login_progress);

    }

    /**
     * Attempts to sign in or register the account specified by the login form.
     * If there are form errors (invalid email, missing fields, etc.), the
     * errors are presented and no actual login attempt is made.
     */
    public void attemptLogin() {
        if (mAuthTask != null) {
            return;
        }

        // Reset errors.
        mEmailView.setError(null);
        mPasswordView.setError(null);

        // Store values at the time of the login attempt.
        String email = mEmailView.getText().toString();
        String password = mPasswordView.getText().toString();
        String username = mDisplayNameView.getText().toString();
        log.info("username[" + username +"," +password + " " + email +"] " + password.length());
        boolean cancel = false;
        View focusView = null;


        // Check for a valid password, if the user entered one.
        if (!TextUtils.isEmpty(password) && !isPasswordValid(password)) {
            mPasswordView.setError(getString(R.string.error_invalid_password));
            focusView = mPasswordView;
            cancel = true;
        }

        if (TextUtils.isEmpty(username)){
            mDisplayNameView.setError(getString(R.string.error_field_required));
            focusView = mDisplayNameView;
            cancel = true;
        }

        // Check for a valid email address.
        if (TextUtils.isEmpty(email)) {
            mEmailView.setError(getString(R.string.error_field_required));
            focusView = mEmailView;
            cancel = true;
        } else if (!isEmailValid(email)) {
            mEmailView.setError(getString(R.string.error_invalid_email));
            focusView = mEmailView;
            cancel = true;
        }

        if (cancel) {
            // There was an error; don't attempt login and focus the first
            // form field with an error.
            focusView.requestFocus();
        } else {
            // Show a progress spinner, and kick off a background task to
            // perform the user login attempt.
            showProgress(true);
            mAuthTask = new UserLoginTask(username, email, password);
            mAuthTask.execute((Void) null);
        }
    }

    private boolean isEmailValid(String email) {
        //TODO: Replace this with your own logic
        return email.contains("@");
    }

    private void launchMainActivity(){
        Intent intent;
        intent = new Intent(this, MainActivity.class);
        startActivity(intent);
    }

    private void populateAutoComplete() {
        getLoaderManager().initLoader(0, null, this);
    }

    /**
     * Shows the progress UI and hides the login form.
     */
    @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)
    public void showProgress(final boolean show) {
        // On Honeycomb MR2 we have the ViewPropertyAnimator APIs, which allow
        // for very easy animations. If available, use these APIs to fade-in
        // the progress spinner.
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB_MR2) {
            int shortAnimTime =
getResources().getInteger(android.R.integer.config_shortAnimTime);

            mLoginFormView.setVisibility(show ? View.GONE : View.VISIBLE);
            mLoginFormView.animate().setDuration(shortAnimTime).alpha(
                    show ? 0 : 1).setListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    mLoginFormView.setVisibility(show ? View.GONE : View.VISIBLE);
                }
            });

            mProgressView.setVisibility(show ? View.VISIBLE : View.GONE);
            mProgressView.animate().setDuration(shortAnimTime).alpha(
                    show ? 1 : 0).setListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    mProgressView.setVisibility(show ? View.VISIBLE : View.GONE);
                }
            });
        } else {
            // The ViewPropertyAnimator APIs are not available, so simply show
            // and hide the relevant UI components.
            mProgressView.setVisibility(show ? View.VISIBLE : View.GONE);
            mLoginFormView.setVisibility(show ? View.GONE : View.VISIBLE);
        }
    }

    @Override
    public Loader<Cursor> onCreateLoader(int i, Bundle bundle) {
        return new CursorLoader(this,
                // Retrieve data rows for the device user's 'profile' contact.
                Uri.withAppendedPath(ContactsContract.Profile.CONTENT_URI,
                        ContactsContract.Contacts.Data.CONTENT_DIRECTORY),
ProfileQuery.PROJECTION,

                // Select only email addresses.
                ContactsContract.Contacts.Data.MIMETYPE +
                        " = ?", new String[]{ContactsContract.CommonDataKinds.Email
                .CONTENT_ITEM_TYPE},

                // Show primary email addresses first. Note that there won't be
                // a primary email address if the user hasn't specified one.
                ContactsContract.Contacts.Data.IS_PRIMARY + " DESC");
    }

    @Override
    public void onLoadFinished(Loader<Cursor> cursorLoader, Cursor cursor) {
        List<String> emails = new ArrayList<String>();
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            emails.add(cursor.getString(ProfileQuery.ADDRESS));
            cursor.moveToNext();
        }

        addEmailsToAutoComplete(emails);
    }

    @Override
    public void onLoaderReset(Loader<Cursor> cursorLoader) {
    }

    private interface ProfileQuery {
        String[] PROJECTION = {
                ContactsContract.CommonDataKinds.Email.ADDRESS,
                ContactsContract.CommonDataKinds.Email.IS_PRIMARY,
        };

        int ADDRESS = 0;
        int IS_PRIMARY = 1;
    }


    /**
     * Check if the device supports Google Play Services.  It's best
     * practice to check first rather than handling this as an error case.
     *
     * @return whether the device supports Google Play Services
     */

    private void addEmailsToAutoComplete(List<String> emailAddressCollection) {
        //Create adapter to tell the AutoCompleteTextView what to show in its dropdown list.
        ArrayAdapter<String> adapter =
                new ArrayAdapter<String>(LoginActivity.this,
                        android.R.layout.simple_dropdown_item_1line, emailAddressCollection);

        mEmailView.setAdapter(adapter);
    }

    private boolean isPasswordValid(String password) {
        //TODO: Replace this with your own logic
        return password.length() > 4;
    }


    /**
     * Represents an asynchronous login/registration task used to authenticate
     * the user.
     */
    public class UserLoginTask extends AsyncTask<Void, Void, Boolean> {

        private final String mUserEmail;
        private final String mUserPassword;
        private final String mUserName;
        private String mDeviceId;
        Boolean isPassword = false;

        public UserLoginTask(String username, String email, String password) {
            mUserEmail = email;
            mUserPassword = password;
            mUserName = username;
            if(!mUserPassword.isEmpty()) {
                isPassword = true;
            }

        }

        @Override
        protected Boolean doInBackground(Void... params) {
            if (mRegistrationService == null) { // Only do this once
                // TODO: Clean up begin
                Registration.Builder builder = new
Registration.Builder(AndroidHttp.newCompatibleTransport(),
                        new AndroidJsonFactory(), null)
                        // options for running against local devappserver
                        // - 10.0.2.2 is localhost's IP address in Android emulator
                        // - turn off compression when running against local devappserver
                        .setRootUrl("http://10.0.2.2:8080/_ah/api/")
                        .setGoogleClientRequestInitializer(new GoogleClientRequestInitializer() {
                            @Override
                            public void initialize(AbstractGoogleClientRequest<?>
abstractGoogleClientRequest) throws IOException {
                                abstractGoogleClientRequest.setDisableGZipContent(true);
                            }
                        });
                // TODO: clean up end
                mRegistrationService = builder.build();
            }
            SharedPreferences prefs = getSharedPreferences(LoqaleConstants.PREFS_NAME,
Context.MODE_PRIVATE);
            String storedPass = "";
            String storedEmail = prefs.getString("email", "");
            Boolean isPass = prefs.getBoolean("is_pass",false);
            log.info(storedEmail + " " + storedPass);
            if (isPass) {
                storedPass = prefs.getString("password", "");
            }
            User userInfo = new User();

            if (storedEmail.isEmpty()) {
                try {
                    final TelephonyManager tm = (TelephonyManager)
getBaseContext().getSystemService(Context.TELEPHONY_SERVICE);

                    final String tmDevice, tmSerial, androidId;
                    tmDevice = "" + tm.getDeviceId();
                    tmSerial = "" + tm.getSimSerialNumber();
                    androidId = "" +
android.provider.Settings.Secure.getString(getContentResolver(),
android.provider.Settings.Secure.ANDROID_ID);

                    UUID deviceUuid = new UUID(androidId.hashCode(), ((long) tmDevice.hashCode()
<< 32) | tmSerial.hashCode());
                    mDeviceId = deviceUuid.toString();

                    Log.v(null, "sending user["+ mUserEmail +"] with mDeviceId[" + mDeviceId
+"]");
                    userInfo.setDeviceId(mDeviceId);
                    userInfo.setUserId(mUserEmail);
                    userInfo.setDisplayName(mUserName);
                    userInfo.setProximity(2000.0);

                    mRegistrationService.register(mDeviceId, userInfo).execute();
                } catch (IOException e) {
                    e.printStackTrace();
                }


                return true;
            }
            else{
                // existing user cases
                // no password
                if( !isPass && (storedEmail.equalsIgnoreCase(mUserEmail)))
                {
                    return true;
                }
                // user with password
                if ( (storedEmail == mUserEmail) && (storedPass == mUserPassword) ){
                    // nothing to do since the user has already authenticated
                    return true;
                }
                return false;
            }
        }

        @Override
        protected void onPostExecute(final Boolean success) {
            mAuthTask = null;
            showProgress(false);

            if (success) {
                // TODO: Encrypt the credentials
                SharedPreferences prefs = getSharedPreferences(LoqaleConstants.PREFS_NAME,
Context.MODE_PRIVATE);
                SharedPreferences.Editor editor = prefs.edit();
                editor.putString("username", mUserName);
                editor.putString("email", mUserEmail);
                editor.putBoolean("is_pass", isPassword);
                if( isPassword) {
                    editor.putString("password", mUserPassword);
                }
                editor.commit();
                launchMainActivity();
                finish();
            } else {
                mPasswordView.setError(getString(R.string.error_incorrect_password));
                mPasswordView.requestFocus();
            }
        }

        @Override
        protected void onCancelled() {
            mAuthTask = null;
            showProgress(false);
        }
    }
}
```

### 6.3.17 MainActivity.java

```
package nfiniteloop.net.loqale;

import android.app.ActionBar;
import android.app.FragmentTransaction;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
import android.support.v4.view.ViewPager;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;

/**
 * Created by vaek on 11/5/14.
 */
public class MainActivity extends FragmentActivity implements ActionBar.TabListener {
    // Logging for debugging
    private Logger log = Logger.getLogger(MainActivity.class.getName());

    //container class for places fragment
    private ArrayList<MessageItem> mMessages;
    private ArrayList<PlaceItem> mPlaces;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);
        mMessages = new ArrayList<MessageItem>();
        mPlaces = new ArrayList<PlaceItem>();

        final ActionBar actionbar = getActionBar();
        actionbar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);


actionbar.addTab(actionbar.newTab().setText(R.string.title_messages).setTabListener(this));
        actionbar.addTab(actionbar.newTab().setText(R.string.title_places).setTabListener(this));
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main, menu);

        return super.onCreateOptionsMenu(menu);
    }

    public void onSectionAttached(int number) {

    }


    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            showSettingsActivity();
        }
        return super.onOptionsItemSelected(item);
    }

    private void showSettingsActivity() {
        Intent intent;
        intent = new Intent(this, SettingsActivity.class);
        startActivity(intent);
    }

    @Override
    public void onTabSelected(ActionBar.Tab tab, FragmentTransaction fragmentTransaction) {
        if (tab.getPosition() == 0) {
            MessageFragment messageFragment = MessageFragment.newInstance(mMessages);
            getSupportFragmentManager().beginTransaction().replace(R.id.drawer_layout,
messageFragment).commit();
        } else if (tab.getPosition() == 1) {
            PlaceFragment placeFragment = PlaceFragment.newInstance(mPlaces);
            getSupportFragmentManager().beginTransaction().replace(R.id.drawer_layout,
placeFragment).commit();
        }

    }

    @Override
    public void onTabUnselected(ActionBar.Tab tab, FragmentTransaction fragmentTransaction) {

    }

    @Override
    public void onTabReselected(ActionBar.Tab tab, FragmentTransaction fragmentTransaction) {

    }
}
```

### 6.3.18 MessageFragment.java

```
package nfiniteloop.net.loqale;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.extensions.android.json.AndroidJsonFactory;
import com.google.api.client.googleapis.services.AbstractGoogleClientRequest;
import com.google.api.client.googleapis.services.GoogleClientRequestInitializer;

import net.nfiniteloop.loqale.backend.registration.Registration;
import net.nfiniteloop.loqale.backend.registration.model.RegistrationRecord;
import net.nfiniteloop.loqale.backend.registration.model.RegistrationRecordCollection;

import java.io.IOException;
import java.lang.ref.WeakReference;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;

/**
 * Created by vaek on 11/27/14.
 */
```

```
public class MessageFragment extends ListFragment {
    private Logger log = Logger.getLogger(MessageFragment.class.getName());
    private MessageAdapter mMessageFragmentAdapter;
    private Registration mRegistrationService;
    private WeakReference<MessageGetterTask> mAsyncTaskWeakRef;


    public static MessageFragment newInstance(ArrayList<MessageItem> items) {
        MessageFragment mf = new MessageFragment();
        Bundle bundle = new Bundle();
        bundle.putParcelableArrayList("items", items);
        mf.setArguments(bundle);

        return mf;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //items.add((MessageItem) savedInstanceState.getParcelableArrayList("messages").get(0));
        setRetainInstance(true);
        MessageGetterTask mg = new MessageGetterTask(this);
        this.mAsyncTaskWeakRef = new WeakReference<MessageGetterTask>(mg);
        mg.execute();


    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.loqale_messages,container,false);

        return view;
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        MessageGetterTask mg = new MessageGetterTask(this);
        this.mAsyncTaskWeakRef = new WeakReference<MessageGetterTask>(mg);
        mg.execute();

    }

    public class MessageGetterTask extends AsyncTask<Void, Void, ArrayList<MessageItem> > {

        List<RegistrationRecord> foo = new ArrayList<RegistrationRecord>();
        ArrayList<MessageItem> bar = new ArrayList<MessageItem>();
        private WeakReference<MessageFragment> fragmentWeakRef;

        private MessageGetterTask (MessageFragment fragment) {
            this.fragmentWeakRef = new WeakReference<MessageFragment>(fragment);
        }

        @Override
        protected ArrayList<MessageItem> doInBackground(Void... params) {
            if (mRegistrationService == null) { // Only do this once
                Registration.Builder builder =
                        new Registration.Builder(AndroidHttp.newCompatibleTransport(),
                        new AndroidJsonFactory(), null)
                        // options for running against local devappserver
                        // - 10.0.2.2 is localhost's IP address in Android emulator
                        // - turn off compression when running against local devappserver
                        .setRootUrl("http://10.0.2.2:8080/_ah/api/")
                        .setGoogleClientRequestInitializer(new GoogleClientRequestInitializer() {
                            @Override
                            public void initialize(AbstractGoogleClientRequest<?>
abstractGoogleClientRequest) throws IOException {
                                abstractGoogleClientRequest.setDisableGZipContent(true);
                            }
                        });
                // end options for devappserver
                mRegistrationService = builder.build();
            }
            try {
                RegistrationRecordCollection ugh = mRegistrationService.listDevices(1).execute();
                foo.addAll(ugh.getItems());

                if(!foo.isEmpty()) {
                    MessageItem mi = new MessageItem();

                    mi.setMessage(foo.get(0).getRegId());
                    SharedPreferences prefs = getActivity()
                            .getSharedPreferences(LoqaleConstants.PREFS_NAME,
Context.MODE_PRIVATE);
                    String loqaleUser = prefs.getString("username", "Loqale User");
                    mi.setUsername(loqaleUser);
                    int drawableId = R.drawable.ic_person_black_36dp;
                    mi.setPicture(getResources().getDrawable(drawableId));
                    bar.add(mi);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }

            return bar;
        }

        @Override
        protected void onPostExecute(final ArrayList<MessageItem> result) {
            if (this.fragmentWeakRef.get() != null) {
                mMessageFragmentAdapter = new MessageAdapter(getActivity(), result);
                setListAdapter(mMessageFragmentAdapter);
            }

        }
    }

}
```

### 6.3.19 MessageAdapter.java

```
package nfiniteloop.net.loqale;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;
```

```java
import java.util.ArrayList;
import java.util.List;

/**
 * Created by vaek on 11/30/14.
 */
public class MessageAdapter extends ArrayAdapter<MessageItem> {

    private Context mContext;
    private List<MessageItem> mRowItem;

    MessageAdapter(Context context, List<MessageItem> rowItem) {
        super(context,R.layout.loqale_messages, rowItem);
        this.mRowItem = new ArrayList<MessageItem>();
        this.mRowItem.addAll(rowItem);
        this.mContext = context;
    }

    @Override
    public int getCount() {

        return mRowItem.size();
    }

    @Override
    public long getItemId(int position) {

        return mRowItem.indexOf(getItem(position));
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            LayoutInflater inflater = LayoutInflater.from(getContext());
            convertView = inflater.inflate(R.layout.loqale_messages, null);
        }

        MessageItem item = getItem(position);
        ((ImageView)
convertView.findViewById(R.id.list_pic)).setImageDrawable(item.getPicture());
        ((TextView) convertView.findViewById(R.id.list_username)).setText(item.getUsername());
        ((TextView) convertView.findViewById(R.id.list_msg)).setText(item.getMessage());
        return convertView;

    }

}
```

### 6.3.20 MessageItem.java

```java
package nfiniteloop.net.loqale;

import android.graphics.drawable.Drawable;
import android.os.Parcel;
import android.os.Parcelable;

/**
 * Created by vaek on 11/28/14.
 */
public class MessageItem implements Parcelable {
    private Drawable picture;
    private String username;
    private String message;

    public MessageItem() {}

    public MessageItem(String username, String message, Drawable picture) {
        this.picture = picture;
        this.username = username;
        this.message = message;
    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel parcel, int i) {
        parcel.writeString(username);
        parcel.writeString(message);

    }

    public void setUsername(String username){
        this.username = username;
    }

    public String getUsername() {
        return username;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public Drawable getPicture() {
        return picture;
    }

    public void setPicture(Drawable picture) {
        this.picture = picture;
    }
}
```

### 6.3.21 PlaceFragment.java

```
package nfiniteloop.net.loqale;

import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.SharedPreferences;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;

import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.extensions.android.json.AndroidJsonFactory;
import com.google.api.client.googleapis.services.AbstractGoogleClientRequest;
import com.google.api.client.googleapis.services.GoogleClientRequestInitializer;
import com.google.api.client.util.DateTime;

import net.nfiniteloop.loqale.backend.checkins.Checkins;
import net.nfiniteloop.loqale.backend.checkins.model.CheckIn;
import net.nfiniteloop.loqale.backend.places.Places;
import net.nfiniteloop.loqale.backend.places.model.Place;
import net.nfiniteloop.loqale.backend.places.model.PlaceCollection;
import net.nfiniteloop.loqale.backend.registration.model.RegistrationRecord;

import java.io.IOException;
import java.lang.ref.WeakReference;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.logging.Logger;

/**
 * Created by vaek on 11/27/14.
 */
public class PlaceFragment extends ListFragment {
    private Logger log = Logger.getLogger(PlaceFragment.class.getName());
    private PlaceAdapter mPlaceAdapter;
    private Places mPlacesService;
    private Checkins mCheckinsService;
    private Boolean mDialogResult;
    private WeakReference<PlaceGetterTask> mAsyncTaskWeakRef;

    //test harness containers
    // TODO: Implement real location gatherer.

    public static PlaceFragment newInstance(ArrayList<PlaceItem> items) {
        PlaceFragment mf = new PlaceFragment();
        Bundle bundle = new Bundle();
        bundle.putParcelableArrayList("items", items);
        bundle.putString("longitude", "-118.3503218");
        bundle.putString("latitude", "33.8640103");
        mf.setArguments(bundle);

        return mf;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //items.add((PlaceItem) savedInstanceState.getParcelableArrayList("messages").get(0));
        setRetainInstance(true);
        PlaceGetterTask mg = new PlaceGetterTask(this);
        this.mAsyncTaskWeakRef = new WeakReference<PlaceGetterTask>(mg);
        mg.execute();


    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        // select an layout to inflate
        View view = inflater.inflate(R.layout.loqale_messages,container,false);

        return view;
    }

    @Override
    public void onListItemClick(ListView l, View v, int i, long id) {
        // stackOverflow
        // https://stackoverflow.com/questions/2478517/how-to-display-a-yes-no-dialog-box-in-
android
        mDialogResult = false;
        CheckInHelperTask ck = new CheckInHelperTask(this);
        super.onListItemClick(l, v, i, id);

        PlaceItem selectedItem;
        selectedItem = (PlaceItem) l.getItemAtPosition(i);

        Boolean result = CheckInDialog(selectedItem.getPlaceName());
        CheckIn placeCheckIn = new CheckIn();
        SharedPreferences prefs = getActivity()
                .getSharedPreferences(LoqaleConstants.PREFS_NAME, Context.MODE_PRIVATE);
        String userDevice = prefs.getString("deviceId", "");
        log.info("we get here");
        if(!userDevice.isEmpty()) {
            log.info("but how about here?");
            placeCheckIn.setUserId(userDevice);
            placeCheckIn.setCheckInDate(new DateTime(System.currentTimeMillis()));
            placeCheckIn.setPlaceId(selectedItem.getPlaceId());

            ck.execute(placeCheckIn);
        }
    }

    private Boolean CheckInDialog(String placeName) {
        // stackOverflow
        // https://stackoverflow.com/questions/2478517/how-to-display-a-yes-no-dialog-box-in-
android
        final AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle("Checking In at " + placeName + "?" );
        builder.setMessage("Would you like to check in?");

        builder.setPositiveButton("YES", new DialogInterface.OnClickListener() {

            public void onClick(DialogInterface dialog, int which) {
                mDialogResult = true;
                dialog.dismiss();
            }

        });

        builder.setNegativeButton("NO", new DialogInterface.OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.dismiss();
            }
        });
        AlertDialog alert = builder.create();
        alert.show();

        return mDialogResult;

    }

    public int categoryToIconId(String category) {
        Map<String, Integer> categoryPicKeys = new HashMap<String, Integer>();
        categoryPicKeys.put("GAS", R.drawable.ic_local_gas_station_black_36dp );
        categoryPicKeys.put("BAR", R.drawable.ic_local_bar_black_36dp);
        categoryPicKeys.put("CFE", R.drawable.ic_local_cafe_black_36dp );
        categoryPicKeys.put("GRC", R.drawable.ic_local_grocery_store_black_36dp );
        categoryPicKeys.put("FOD", R.drawable.ic_local_pizza_black_36dp );
        categoryPicKeys.put("SCH", R.drawable.ic_school_black_36dp );
        categoryPicKeys.put("RST", R.drawable.ic_restaurant_menu_black_36dp );

        return categoryPicKeys.get(category);
    }

    public class PlaceGetterTask extends AsyncTask<Void, Void, ArrayList<PlaceItem> > {

        List<Place> foo = new ArrayList<Place>();
        ArrayList<PlaceItem> bar = new ArrayList<PlaceItem>();
        private WeakReference<PlaceFragment> mFragmentWeakRef;
        private Double longitude = new Double(-118.3503218);
        private Double latitude = new Double(33.8640103);

        private PlaceGetterTask(PlaceFragment fragment) {
            this.mFragmentWeakRef = new WeakReference<PlaceFragment>(fragment);
        }
        // must call this first!!
        public void setLocation(Double Latitude, Double Longitude) {
            latitude = Latitude;
            longitude = Longitude;
        }
        @Override
        protected ArrayList<PlaceItem> doInBackground(Void... params) {
            if (mPlacesService == null) { // Only do this once
                Places.Builder builder = new Places.Builder(AndroidHttp.newCompatibleTransport(),
                        new AndroidJsonFactory(), null)
                        // options for running against local devappserver
                        // - 10.0.2.2 is localhost's IP address in Android emulator
                        // - turn off compression when running against local devappserver
                        .setRootUrl("http://10.0.2.2:8080/_ah/api/")
                        .setGoogleClientRequestInitializer(new GoogleClientRequestInitializer() {
                            @Override
                            public void initialize(AbstractGoogleClientRequest<?>
abstractGoogleClientRequest) throws IOException {
                                abstractGoogleClientRequest.setDisableGZipContent(true);
                            }
                        });
                // end options for devappserver
                mPlacesService = builder.build();
            }
            try {
                int count=10;
                Double range = new Double(1000.0);
                PlaceCollection ugh = mPlacesService.
                        listPlaces(count, longitude, latitude, range).execute();
                if(!ugh.isEmpty()) {
                    log.info(ugh.size() +" places returned");
                    foo.addAll(ugh.getItems());
                    for (Place p : foo) {
                        PlaceItem pi = new PlaceItem();
                        pi.setPlaceName(p.getName());
                        pi.setLongitude(p.getLongitude());
                        pi.setLatitude(p.getLatitude());
                        pi.setDistance(p.getDistance());
                        pi.setPlaceId(p.getPlaceId());
                        // we need a map here!
                        int picId = this.mFragmentWeakRef.get()
                                .categoryToIconId(p.getCategory());
                        pi.setPicCategory(getResources().getDrawable(picId));
                        bar.add(pi);
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }

            return bar;
        }

        @Override
        protected void onPostExecute(final ArrayList<PlaceItem> result) {
            if (this.mFragmentWeakRef.get() != null) {
                mPlaceAdapter = new PlaceAdapter(getActivity(), result);
                setListAdapter(mPlaceAdapter);
            }

        }
    }

    public class CheckInHelperTask extends AsyncTask<CheckIn, Void, Boolean> {

        List<RegistrationRecord> foo = new ArrayList<RegistrationRecord>();
        ArrayList<MessageItem> bar = new ArrayList<MessageItem>();
        private WeakReference<PlaceFragment> fragmentWeakRef;

        private CheckInHelperTask (PlaceFragment fragment) {
            this.fragmentWeakRef = new WeakReference<PlaceFragment>(fragment);
        }

        @Override
        protected Boolean doInBackground(CheckIn... params) {
            if (mCheckinsService == null) { // Only do this once
                Checkins.Builder builder = new
Checkins.Builder(AndroidHttp.newCompatibleTransport(),
                        new AndroidJsonFactory(), null)
                        // options for running against local devappserver
                        // - 10.0.2.2 is localhost's IP address in Android emulator
                        // - turn off compression when running against local devappserver
                        .setRootUrl("http://10.0.2.2:8080/_ah/api/")
                        .setGoogleClientRequestInitializer(new GoogleClientRequestInitializer() {
                            @Override
                            public void initialize(AbstractGoogleClientRequest<?>
abstractGoogleClientRequest) throws IOException {
                                abstractGoogleClientRequest.setDisableGZipContent(true);
                            }
                        });
                // end options for devappserver
                mCheckinsService = builder.build();
            }
            try {
                CheckIn checkInItem = params[0];
                log.info("Testing...");
                mCheckinsService.checkin(checkInItem).execute();

            } catch (IOException e1) {
                e1.printStackTrace();
            }
            return true;
        }

        @Override
        protected void onPostExecute(Boolean success) {
            if (success) {
                // TODO: Pop a dialog
            }

        }
    }

}
```

### 6.3.22 PlaceAdapter.java

```
package nfiniteloop.net.loqale;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by vaek on 11/27/14.
 */
public class PlaceAdapter extends ArrayAdapter<PlaceItem> {

    private Context mContext;
    private List<PlaceItem> mRowItem;

    PlaceAdapter(Context context, List<PlaceItem> rowItem) {
        super(context,R.layout.loqale_places, rowItem);
        this.mRowItem = new ArrayList<PlaceItem>();
        this.mRowItem.addAll(rowItem);
        this.mContext = context;
    }

    @Override
    public int getCount() {

        return mRowItem.size();
    }

    @Override
    public long getItemId(int position) {
        return mRowItem.indexOf(getItem(position));
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        if (convertView == null) {
            LayoutInflater inflater = LayoutInflater.from(getContext());
            convertView = inflater.inflate(R.layout.loqale_places, null);
        }


        PlaceItem item = getItem(position);
        ((ImageView) convertView.findViewById(R.id.pic_place_category))
                .setImageDrawable(item.getPicCategory());
        ((TextView) convertView.findViewById(R.id.place_name)).setText(item.getPlaceName());
        ((TextView)
convertView.findViewById(R.id.place_distance)).setText(item.getDistance().toString());

        return convertView;

    }

}
```

### 6.3.23 PlaceItem.java

```
package nfiniteloop.net.loqale;

import android.graphics.drawable.Drawable;
import android.os.Parcel;
import android.os.Parcelable;

/**
 * Created by vaek on 11/28/14.
 */
public class PlaceItem implements Parcelable {
    private Drawable picCategory;
    private String placeName;
    private Double longitude;
    private Double latitude;
    private String distance;
    private String placeId;

    public PlaceItem() {}

      public  PlaceItem(Drawable  pic,  String  name,  Double  latitude,  Double  longitude,  String
distance, String placeId) {
        this.picCategory = pic;
        this.placeName = name;
        this.longitude = longitude;
        this.latitude = latitude;
        this.distance = distance;
        this.placeId = placeId;
    }

    @Override
    public int describeContents() {
        return 0;
    }
    @Override
    public void writeToParcel(Parcel parcel, int i) {
        parcel.writeString(placeName);
        parcel.writeDouble(latitude);
        parcel.writeDouble(longitude);
        parcel.writeString(distance);
        parcel.writeString(placeId);
    }

    public void setPicCategory(Drawable picCategory) {
        this.picCategory = picCategory;
    }

    public void setLongitude(Double longitude) {
        this.longitude = longitude;
    }

    public void setLatitude(Double latitude) {
        this.latitude = latitude;
    }

    public void setPlaceName(String placeName) {
        this.placeName = placeName;
    }

    public Drawable getPicCategory() {
        return picCategory;
    }

    public Double getLongitude() {
        return longitude;
    }

    public Double getLatitude() {
        return latitude;
    }

    public String getPlaceName() {
        return placeName;
    }

    public String getDistance() {
        return distance;
    }

    public void setDistance(String distance) {
        this.distance = distance;
    }

    public void setPlaceId(String placeId) {
        this.placeId = placeId;
    }

    public String getPlaceId() {
        return placeId;
    }
}
```

### 6.3.24 RecommenderAdpater.java

```
package nfiniteloop.net.loqale;
```

```
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.List;

/**
 * Created by vaek on 12/6/14.
 */
public class RecommendAdapter extends ArrayAdapter<RecommendItem> {

    Context context;
    List<RecommendItem> rowItem;

    RecommendAdapter(Context context, List<RecommendItem> rowItem) {
        super(context,R.layout.loqale_recommendations, rowItem);
        this.rowItem = new ArrayList<RecommendItem>();
        this.rowItem.addAll(rowItem);
    }

    @Override
    public int getCount() {

        return rowItem.size();
    }

    @Override
    public long getItemId(int position) {

        return rowItem.indexOf(getItem(position));
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        if (convertView == null) {
            LayoutInflater inflater = LayoutInflater.from(getContext());
            convertView = inflater.inflate(R.layout.loqale_places, null);
        }

                                             ImageView     picCategoryIcon     =     (ImageView)
convertView.findViewById(R.id.pic_place_category);
        TextView txtPlaceName = (TextView) convertView.findViewById(R.id.place_name);
        TextView txtPlaceDistance = (TextView) convertView.findViewById(R.id.place_distance);
        ImageView picLike = (ImageView) convertView.findViewById(R.id.pic_like);

        RecommendItem item = getItem(position);
        picCategoryIcon.setImageDrawable(item.picCategory);
        txtPlaceName.setText(item.placeName);
        txtPlaceDistance.setText(item.placeDistance);

        return convertView;

    }

}
```

### 6.3.25 RecommenderItem.java

```
package nfiniteloop.net.loqale;

import android.graphics.drawable.Drawable;

/**
 * Created by vaek on 12/6/14.
 */
public class RecommendItem {
    Drawable picCategory;
    String placeName;
    String placeDistance;
    Drawable picLike;

    public RecommendItem(Drawable pic, String name, String distance, Drawable likepic) {
        this.picCategory = pic;
        this.placeName = name;
        this.placeDistance = distance;
        this.picLike = likepic;

    }
}
```

### 6.3.26 LoqaleConstants.java

```
package nfiniteloop.net.loqale;

/**
 * Created by vaek on 12/4/14.
 */
public class LoqaleConstants {
    public static final String PREFS_NAME = "LoqaleData";
}
```

### 6.3.27 activity_login.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:background="@drawable/login_background"
    tools:context="nfiniteloop.net.loqale.LoginActivity">

    <!-- Login progress -->
    <ProgressBar
        android:id="@+id/login_progress"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:visibility="gone"/>

    <ScrollView
        android:id="@+id/login_form"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        >
        <LinearLayout
            android:id="@+id/email_login_form"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <EditText
                android:id="@+id/user_disp_name"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="textPersonName"
                android:text="Display Name"
                android:maxLength="16"
                android:maxLines="1"
                android:ems="10" />

            <AutoCompleteTextView
                android:id="@+id/email"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="@string/prompt_email"
                android:inputType="textEmailAddress"
                android:maxLines="1"
                android:singleLine="true"/>

            <EditText
                android:id="@+id/password"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="@string/prompt_password"
                android:imeActionId="@+id/login"
                android:imeActionLabel="@string/action_sign_in_short"
                android:imeOptions="actionUnspecified"
                android:inputType="textPassword"
                android:maxLines="1"
                android:singleLine="true"/>

            <Button
                android:id="@+id/email_sign_in_button"
                style="?android:textAppearanceSmall"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="16dp"
                android:text="@string/action_sign_in"
                android:textStyle="bold"/>

        </LinearLayout>
    </ScrollView>

</LinearLayout>
```

### 6.3.28 loqale_messages.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <ImageView
        android:id="@+id/list_pic"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentTop="true"/>
    <TextView
        android:id="@+id/list_username"
        android:layout_width="wrap_content"
        android:layout_height="30dp"
        android:layout_alignParentBottom="false"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/list_pic"
        style="@android:style/TextAppearance.Holo.Large"/>
    <TextView
        android:id="@+id/list_msg"
        android:layout_width="wrap_content"
        android:layout_height="20dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/list_pic"/>
    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </ListView>
</RelativeLayout>
```

### 6.3.29 loqale_places.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <ImageView
        android:id="@+id/pic_place_category"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentTop="true"/>
    <TextView
        android:id="@+id/place_name"
        android:layout_width="wrap_content"
        android:layout_height="30dp"
        android:layout_alignParentBottom="false"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/pic_place_category"
        style="@android:style/TextAppearance.Holo.Large"/>
    <TextView
        android:id="@+id/place_distance"
        android:layout_width="wrap_content"
        android:layout_height="20dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/pic_place_category"/>
<ListView
    android:id="@android:id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    </ListView>
</RelativeLayout>
```

### 6.3.30 loqale_recommendations.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <TableRow>
        <ImageView
            android:id="@+id/pic_place_category"
            android:layout_width="50dp"
            android:layout_height="50dp"/>
        <TextView
            android:id="@+id/place_name"
            android:layout_width="wrap_content"
            android:layout_height="30dp"/>
        <TextView
            android:id="@+id/place_distance"
            android:layout_width="wrap_content"
            android:layout_height="20dp"/>
        <ImageView
            android:id="@+id/pic_like"
            android:layout_width="50dp"
            android:layout_height="50dp"/>
    </TableRow>
</TableLayout>
```

### 6.3.31 DataInjector.java

```
package net.nfiniteloop.loqale.backend;

import static net.nfiniteloop.loqale.backend.OfyService.ofy;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

/**
 * Created by vaek on 12/7/14.
 */
// TODO: Implement a script to automate injection from known data source
public class DataInjector {
    // inject self id
    public DataInjector () {

    }
    double[][] cheapClose = new double[][]{
        {1.0, 0.0},
        {0.0, 0.0}
    };

    double[][] expensiveFar = new double[][]{
        {1.0, 0.0},
        {0.0, 1.0}
    };

    double[][] expensiveClose = new double[][]{
            {1.0, 0.0},
            {0.0, 1.0}
    };

    double[][] cheapFar = new double[][]{
            {0.0, 0.0},
            {1.0, 0.0}
    };

    double[][] off = new double[][]{
            {0.0, 0.0},
            {0.0, 0.0}
```

```
};

public Boolean createDataSet() {
    User userMitch = new User();
    userMitch.setDisplayName("Mitch");
    userMitch.setHometown("Torrance");
    userMitch.setProximity(2000.0);
    userMitch.setUserId("vaektor.phresh@gmail.com");
    // preference categories
    List<String> categories = new ArrayList<String>();
    categories.add("GAS");
    categories.add("BAR");
    categories.add("CFE");
    categories.add("FOD");
    userMitch.setCategories(categories);
    if (findUserRecord(userMitch) == null) {
        ofy().save().entity(userMitch).now();
    }
    // inject places

    Place place = new Place();
    place.setAddress("3960 Artesia Blvd, Torrance, CA 90504");
    place.setLatitude(33.8714422);
    place.setLongitude(-118.3445953);
    place.setCategory("GAS");
    place.setPlaceId("1");
    place.setQualityMatrix(cheapClose);
    place.setName("Chevron");
    if (findPlaceRecord(place) == null) {
        ofy().save().entity(place).now();
    }
    Place place2 = new Place();
    place2.setAddress("1400 W 190th St #F, Torrance, CA 90501");
    place2.setLatitude(33.8583);
    place2.setLongitude(-118.3021);
    place2.setCategory("CFE");
    place2.setPlaceId("2");
    place2.setQualityMatrix(cheapClose);
    place2.setName("Starbucks");
    if (findPlaceRecord(place2) == null) {
        ofy().save().entity(place2).now();
    }
    Place place3 = new Place();
    place3.setAddress("1413 Hawthorne Blvd, Redondo Beach CA, 90278");
    place3.setLatitude(33.867005);
    place3.setLongitude(-118.353856);
    place3.setCategory("GRC");
    place3.setPlaceId("3");
    place3.setQualityMatrix(cheapClose);
    place3.setName("Ralph's");
    if (findPlaceRecord(place3) == null) {
        ofy().save().entity(place3).now();
    }


    Place place4 = new Place();
    place4.setAddress("1000 East Victoria St, Carson, CA 90504");
    place4.setLatitude(33.8671);
    place4.setLongitude(-118.2586);
    place4.setCategory("EDU");
    place4.setPlaceId("4");
```

```
place4.setQualityMatrix(off);
place4.setName("Cal State Dominguez Hills");
if (findPlaceRecord(place4) == null) {
    ofy().save().entity(place4).now();
}

Place place5 = new Place();
place5.setAddress("20240 Avalon Blvd, Carson, CA 90746");
place5.setLatitude(33.83952);
place5.setLongitude(-118.2471891);
place5.setCategory("CFE");
place5.setPlaceId("5");
place5.setQualityMatrix(cheapFar);
place5.setName("Starbucks");
if (findPlaceRecord(place5) == null) {
    ofy().save().entity(place5).now();
}

Place place6 = new Place();
place6.setAddress("17453 S Central Ave,Carson, CA 90746");
place6.setLatitude(33.83952);
place6.setLongitude(-118.2471891);
place6.setCategory("GAS");
place6.setPlaceId("6");
place6.setQualityMatrix(cheapClose);
place6.setName("Chevron");
if (findPlaceRecord(place6) == null) {
    ofy().save().entity(place6).now();
}

Place place7 = new Place();
place7.setAddress("1515 Hawthorne Blvd, Redondo Beach, CA 90278");
place7.setLatitude(33.854634);
place7.setLongitude(-118.3771421);
place7.setCategory("GRC");
place7.setPlaceId("7");
place7.setQualityMatrix(cheapClose);
place7.setName("Sprouts Farmers Market");
if (findPlaceRecord(place7) == null) {
    ofy().save().entity(place7).now();
}

Place place8 = new Place();
place8.setAddress("20930 Hawthorne Blvd, Torrance, CA 90503");
place8.setLatitude(33.839543);
place8.setLongitude(-118.353008);
place8.setCategory("BAR");
place8.setPlaceId("8");
place8.setQualityMatrix(expensiveClose);
place8.setName("Zebra Room");
if (findPlaceRecord(place8) == null) {
    ofy().save().entity(place8).now();
}

Place place9 = new Place();
place9.setAddress("18200 Hawthorne Blvd, Torrance, CA 9050");
place9.setLatitude(33.865056);
place9.setLongitude(-118.3522);
place9.setCategory("FOD");
place9.setPlaceId("9");
place9.setQualityMatrix(cheapClose);
```

```
        place9.setName("Chick-fil-A");
        if (findPlaceRecord(place9) == null) {
            ofy().save().entity(place9).now();
        }

        Place place10 = new Place();
        place10.setAddress("17916 Hawthorne Blvd, Torrance, CA 90504");
        place10.setLatitude(33.8336389);
        place10.setLongitude(-118.3511089);
        place10.setCategory("FOD");
        place10.setPlaceId("10");
        place10.setQualityMatrix(cheapClose);
        place10.setName("Jack In the Box");
        if (findPlaceRecord(place10) == null) {
            ofy().save().entity(place10).now();
        }

        Place place11 = new Place();
        place11.setAddress("17916 Hawthorne Blvd, Torrance, CA 90504");
        place11.setLatitude(33.873067);
        place11.setLongitude(-118.37438);
        place11.setCategory("FOD");
        place11.setPlaceId("11");
        place11.setQualityMatrix(cheapClose);
        place11.setName("Papa John's Pizza");
        if (findPlaceRecord(place11) == null) {
            ofy().save().entity(place11).now();
        }
        return true;
    }
    private User findUserRecord(User record ) {
                              return    ofy().load().type(record.getClass()).filter("userId",
record.getUserId()).first().now();
    }
    private Place findPlaceRecord(Place record ) {
                             return    ofy().load().type(record.getClass()).filter("placeId",
record.getPlaceId()).first().now();
    }
}
```