San Francisco State University
Department of Computer Science

CSC 895 - Development And Evaluation Of Citizen Science Application Using Android And Cloud Computing

Title: Design, Development And Evaluation Of Android Based Citizen Science Application And Cloud Computing

Author(s): **Joao Sousa**

Date: **10/23/2014**

Keywords: **Field Observations, Model View Control, Android, Google App Engine, NoSql, Dropbox, Java, XML**

Abstract:

Citizen Science projects are activities sponsored by organizations which ask society's citizens to volunteer on their project. Each Citizen Science project has its exclusive data capturing motivation and usage that can increase the projects' costs and inefficiency of the project's time frame. Some of these applications require sharing of captured images at no or low cost, sometimes not allowed by available frameworks. Developing an application for each individual project is cost and time inefficient. In order to decrease project inefficiency related to custom data capturing application, a COTS (custom off the shelf) application with data capturing mechanism that supports dynamic configuration and free data sharing like images which is configurable for each project is needed. To achieve this goal, we first evaluated of the shelf technologies that would allow development of free-to use mobile applications for citizen science that also allows free sharing of images. We then developed an Android mobile with Google App Engine framework which allows dynamic data capturing configuration and data sharing via broadcast capabilities. The solution to this problem verified that a mobile Android citizen science application can be built which is easy to configure and allows free sharing of captured images.

Keywords: Citizen science, mobile application, free sharing, Android

Table of Contents

1.0 Introduction

Citizen Science projects are activities sponsored by organizations which ask society's citizens to volunteer on their project. The Citizen Scientists (i.e. volunteers) assist with required tasks in the project and then share the results gathered with the project hosts or participants. Some of the tasks involved in such projects require data gathering, data analysis, periodic specimen monitoring and reporting. These activities involve the person signing up with the organization that is sponsoring the project, so proper communication and information sharing gets established between both parties. Lately data gathering with mobile technology developed great popularity in Citizen Science projects. With the popularity of mobile devices and apps, it has become increasingly easier for people to volunteer in Citizen Scientist projects.

Each Citizen Science project has its exclusive data capturing motivation, which increases the projects' costs and inefficiency of the project's time frame. Developing an application for each individual project is cost and time inefficient. In order to decrease project inefficiency related to custom data capturing application, a COTS (custom off the shelf) application with data capturing mechanism that supports dynamic configuration for each project is needed. To support this demand, an Android application with Google App Engine framework was used to develop a mobile application, which contains dynamic data capturing configuration and data sharing via broadcast capabilities. The goal of this project's application was to suffice the need for data capturing Citizen Science software (mobile application with backend server) with no charge for usage and is configurable dynamically to support multiple projects. The solution to this problem provided a technology that has minimal financial burden on various Citizen Science projects and minimal time constraint to get the technology set up for the project with no programming needed in order to use the mobile application for data capturing.

1.1 **Overview Of Observer Projects At SFSU**

Mobile device and Cloud technologies became great resources to support Citizen Science projects. The applications offered by these technologies make it easy for any individual to simply download a specific Citizen Science project application and then start participating in the project. The mobile phones used by these volunteers come equipped with many sensors and applications, so it is easy to capture information using phone sensors and share the data collected. With these devices users are able to capture rich data such as image, geographical location, text, and other data depending on the mobile device's equipment. Furthermore the users can share data with others at a convenient time. The need and ability to capture rich data (text and image), without errors, for Citizen Science projects makes these mobile phone applications attractive to Citizen Scientist's projects. However, there are many technologies and solutions available to build these mobile phone applications, so deciding which technology to use to build a Citizen Science project's application is a key factor to the project's success.

Choosing the right technology to build the application that best meets the project's requirements can be a daunting. Each project has its own data gathering requirements and building an app for each individual project becomes cost and time inefficient. In this project an investigation was performed on which technology best fits a general set of Citizen Science projects requirements. Then, a step further, a mobile app was built such that the mobile app is free to use. The mobile app built can be dynamically configured to accommodate data capturing requirements for each project, so one app can be used for several individual projects. This paper covers the analyses of various technologies that can support a dynamically configurable mobile application for Citizen Science projects.

In recent work done at San Francisco State University' a configurable COTS (custom off the shelf) framework that supports IOS mobile app or Android mobile devices with ability to accommodate Citizen Science projects and offer data collection configurability and data curation without programming was built and evaluated by Abishek [3]. This COTS configurable iOS mobile device application built by Abishek [3] was supported by a Mongo Database framework that provides PHP-Mongo Service APIs, a publicly accessible platform web interface and a configured Amazon Web Service (AWS) Unix server - hosted on Amazon Cloud backend. Keep in mind that various set of technologies can be used to build a COTS configurable mobile Citizen Science application.

1.2 Project Definition And Project Goals

Each Citizen Science project has its exclusive data capturing motivation that can increase the projects' costs and inefficiency of the project's time frame. Developing an application for each individual project is cost and time inefficient. In order to decrease project inefficiency related to custom data capturing application, a COTS (custom off the shelf) application with data capturing mechanism that supports dynamic configuration for each project is needed. The goal of this project's application is to analyze technologies that can be used to suffice the need for data capturing Citizen Science software (mobile application with backend server) that is free of charge for usage and configurable dynamically to support multiple projects without any software programming required. Then, once the technology is analyzed use that technology to build an application with the requirements that support dynamic data capturing configuration for Citizen Science projects, which is free to use.

1.3 **Contributions**

The contribution to this project was evaluation of the framework used for this project's application's development and then actually building the application. Several factors of the technology used to build the software for this project was analyzed. First an analysis was made to determine the amount of programming effort on backend server to get the project completed to determine if the Google App Engine requires a lot of programming to get the application up and running. Next an evaluation of the amount of programming effort on the mobile application to get project completed was conducted, so one can better estimate how much development effort the mobile application side of the framework requires. Then a study of the effort for getting the programming environment set up such as signing up with Google App Engine, configuring Eclipse with Google App Engine APIs, how helpful were the tutorials provided by Google App Engine and the effort of utilizing special authentication tokens from Google App Engine to get development environment set up.

In addition to analyzing the technology that best supports Citizen Science projects, we considered a technology that is free to use. We reviewed the costs of continuing to run the application once built based on database maintenance, database size increase, bandwidth usage, API versioning upkeep and upgrades. These various analyses were compared against other technologies in other projects to give an insight of the tools available to support COTS dynamically configurable Citizen Science projects.

After analyzing the technology and verifying which would be financially sound, the mobile app development implementation was done. The COTS mobile application requires no programming for the project's unique data capturing functionality. The main project host scientist can access a single Google App Engine project's web application interface, which provides the ability to create multiple observer projects database structure dynamically without the need to program or change the CRUD (create, read, update and delete) functionality API. A mobile Android application with a backend server for data curation with dynamic configuration capability without programming needed in order to support Citizen Science projects data collection is the other effort of this paper.

The development of the application for this project utilized modern software engineering such as User Centric Design (UCD) with focus groups to improve the software's usability. After building the core application's functionality, a focus group was performed in order to improve the user interface based on user's feedback. Once the updates were made based on previous feedbacks, the users would test the application again. Next the users provided new feedback for further usability improvement. Various rounds of User Interface testing and feedback for improvement were made.

1.4 Report Outline

The information of this report will be presented in the following order. Chapter 2 will describe the main requirements for Citizen Science projects. Chapter 3 will talk about technologies evaluated. Chapter 4 gives a design of the application built for this project, which includes the functionality, user interface, application’s architecture and design tradeoffs. Chapter 5 covers the implementation of the application including user interface (UI) and overall architecture. Chapter 6 contains the evaluation and comparison of the technologies will be conducted. Chapter 7 gives a conclusion with recommendations based on the analysis completed.

2.0 General Requirements For Citizen Science applications

For this project a User Centric Design software engineering methodology was used to achieve the final application. Section 2.1 describes typical use cases for the application and scenarios for Citizen Science projects. The requirements sections 2.2 and 2.3 mentioned below are the actual final version of the requirements, which were revised and updated several times throughout each iteration of an User Centric Design. This type of User Interface design involves the user testing the application and providing usability feedback for another round of improvements. Below are the final requirements.

2.1 Use Cases

1. A Citizen Science project host needs a free mobile application with backend server to support data collection that supports his project's data distribution requirements that broadcasts the data to all users in the project.
2. A science project host needs a free mobile app that can support multiple Citizen Science projects. Each science project has different data collection requirements.
3. A user starts the Citizen Science android mobile application while in the outdoors to collect data related to the current Citizen Science project in a region that has no cell phone or internet service available. The user creates a new species record in the app in order to capture text information about the specimen observed. An image of the specimen is taken with the phone's camera. All the text information inputted into that record is saved in the phone's application local database and the image taken is associated with the observation record created.
4. A user opens up the Citizen Science application to verify a specimen: The user notices a specimen in the laboratory that matches the description of another user's specimen,

which was broadcasted that morning without an image or image url. The user opens the record submitted by the colleague and reviews the text data provided. The user takes an image of the lab specimen and associates that image with the record.

5. A user recently updated his specimen notes on the Citizen Science mobile application and decides to broadcast the updates. All users who have the app on their Android phone receive a copy of the specimen's record along with the updates.
6. The Citizen Science project application administrator wants to verify the application's user traffic volume and data accuracy. The user then logs into the backend server at http://cloud.google.com. The user logins to cloud.google.com and selects the Citizen Science project. The usage status and traffic monitoring overview graph is displayed. The user is then able to view the database Schemas available, traffic logs and data stored in the backend server. The user selects a schema entity and views the data stored in the backend. The user then logs out.
7. The Citizen Science project administrator wants to start a new project's data collection effort. He logs into Google App Engine and creates a new project name without the need to further configure any application or pay fees.

2.2 Requirements – Functional

1. Mobile application shall allow user to capture text data
2. Mobile application shall allow user to associate image with data captured
3. Mobile application shall allow user to edit record of data captured
4. User shall only be able to read and not modify data that was captured and shared by other users
5. Administrator shall be able to add data captured

6. Administrator shall be able to edit data captured by any user
7. Administrator shall be able to delete data captured by any user
8. Mobile application shall allow user to select a project in mobile application to allow filtering of data based on project selected.
9. Mobile application shall allow user to upload a record of data collected
10. Project coordinator shall be able to view data in backend server
11. User shall be able to share their own observations

2.3 **Requirements – Non Functional**

1. User Centric Design methodology shall be used to develop user's interface.
2. Application Android services shall be free of charge for usage.
3. Application's backend server services shall be free of charge according to service provider's term and conditions.
4. Android application shall adhere to Android User Interface standards.
5. Listview image shall be at most 1,024 Kb of size, so this can easily fit within the listview display size.
6. User shall be able to dynamically add additional data capturing fields
7. Database backend shall accept a dynamic increase number of fields of data to upload
8. Application layout shall accommodate Android phones
9. Application upload shall take less than 45 seconds based on phone's signal strength
10. Application shall upload record's notes and image seamless to user's perspective
11. User shall be able to upload a record by clicking on the desired record.
12. No upload confirmation prior to uploading occurring shall take place.

13. User shall have ability to sort existing records list by latest record created or by record name.
14. Every record shall have a name and at least one notes field.
15. A record is not required to have an image associated with it. In this case a default icon shall be associated with the record
16. Application buttons shall have good size and width for ease of navigation
17. User shall be able to quit the application from main starting screen only
18. Project coordinator shall be able to curate data in backend server
19. Application shall have some type of user security or authentication mechanism

3.0 Technologies Evaluated

In comparison to this project's mobile app, other mobile apps with dynamic configuration were built. For example, Abishek [3], built an IOs mobile Citizen Science application with Mongo DB as a backend database, running on a Unix virtual server hosted on Amazon cloud. With various technologies available to develop a mobile application with dynamic configurability for Citizen Science projects, this chapter describes the analysis of technologies used to build Citizen Science projects. This chapter first compares mobile apps technology: IOs versus Android. Second this chapter analyzes the backend servers that support the mobile apps. Third this chapter covers Cloud Services and costs related to the backend servers; the main goal is to have an app that is free to use. Fourth this chapter compares Mongo DB database to GAE's, "Data Store," database API. At the end of the chapter there is a table with an overall comparison of various providers of Cloud servers that connects to mobile apps.

3.2 Mobile Apps

IOs mobile apps are very popular in today's mobile device arena. Mobile apps built with IOs technology contain specific development requirements imposed by the technology's manufacturer, which hinders the promotion of open source- free software for Citizen Science projects. The development environment constrains imposed by the IOs manufacturer requires the developer to use Objective-C in order to develop the mobile application. Furthermore publishing the application requires approval and app store fees imposed by the device's manufacturer – Apple Inc. These technological barriers have potential negative financial effects in developing a mobile application for Citizen Science projects because Objective-C is a platform specific language, and it requires a specific development platform i.e. Macintosh computer or notebook to run and test the mobile application. The IOs mobile application developed by Abishek [3], was mostly built from scratch with minimal app framework. From these points of views, one can say that the IOs mobile app imposes some technical and financial challenges to Citizen Science projects due to the technology constrains.

In contrast to the IOs mobile apps, Android mobile apps have more relaxed manufacturer regulations and thus better promotes Citizen Science projects. Android mobile apps are natively Java based applications, which is an open source development language. Furthermore, the Android app store imposes no fees during mobile app deployment.

For this project it was decided to use Google App Engine (GAE) connected Android mobile application framework. The decision was made based on the GAE's COTS ready to use development Android mobile application framework and also due to the free trial service provided by GAE backend server's APIs.

3.3. Backend Servers

Servers connected with mobile apps allow for greater functionality of services and resources. Similar to mobile apps, virtual servers are extremely popular in today's technology infrastructure market. As in Abishek's [3], a virtual Unix server was hosted on an Amazon cloud services. Using Unix server for Citizen Science projects is a great choice because there are various Unix operating systems that are free to use, which impose little to no financial burden on the project's budget. Abishek's [3] project required a virtual Unix serve to be built and configured to accommodate Mongo DB and IOs connectivity. This type of set up takes some time to build and configure. The backend server was connected to a Mongo DB technology: the operating system and database were hosted on separate servers i.e. database server and Unix server. Having the database hosted in a separate Cloud service provider helped minimize the disk space usage with Amazon Cloud server for the project built by Abishek [3]. However, the Mongo DB server itself has its limits of free space too. Furthermore, having the Unix Server call the Mongo DB uses network bandwidth, which also has a limited free usage by Amazon Cloud services. Furthermore an Amazon Cloud server instance had to be created to host the Unix virtual server. All these efforts and configurations take time and potentially bear a financial burden to the Citizen Science project.

Like Amazon Cloud, various other cloud service providers are available. Furthermore some cloud service providers offer mobile application (REST) web services. The Android mobile application connected to GAE comes with a message broadcast mechanism already built and ready to use, so little work was required to test the connectivity between the mobile app and the backend server. This broadcast mechanism served as a testing functionality to show that the mobile application was connected to the backend server and that the broadcast mechanism can send messages to all users in the project.

3.4 Cloud Service Analysis

The Cloud Service technology used by Abishek [3], Amazon Cloud, provides a 1GB virtual server with a limited 365 days of trial period. Additional restrictions apply. Overall this Cloud Service is pretty good, but it does have a fee imposed upon a set period of time. The Cloud space provided can be a Unix based or Windows based server. Abishek [3] was running an Unix server, which was customized for Citizen Science projects and connected to Mongo DB. All these configuration and connection setup takes time and adds to the overall of the Citizen Science project.

In contrast to Amazon Cloud, Google App Engine (GAE) comes with unrestricted number of days for its trial service. The free trial period provided by GAE is financially attractive for Citizen Science projects. The GAE APIs trial period has no fixed number of days imposed. In contrast the other Cloud service providers all impose a fixed number of days for a free trial of its software. With the unlimited number of days for trial of the GAE APIs and the Android COTS broadcast mechanism connected to GAE makes Android mobile app a much more attractive technology than IOs for Citizen Science projects mobile apps.

For the generous trial period and mobile app with backend server connected features, the Android GAE framework can save a lot of time and money in the project's application development phase. Furthermore the open source language with relaxed device manufacturer regulations better promotes free app development for Citizen Science projects.

Another advantage of using GAE service is that no operating system installation is required. The user needs to instantiate a project, which is also time consuming: for this project it took about three to four days to get the project fully set up and ready for development.

Good and helpful setup documentation is provided for new users to help set up the GAE connected Android framework: https://developers.google.com/eclipse/docs/endpoints-androidconnected-gae. This process can be a bit lengthy as it requires the user to generate an sh1 key from the development environment (developer's computer) in order to obtain a project's token-key from GAE. Although the GAE setup process is lengthy, for some developers GAE may be easier than setting up a Unix server and connecting it to Mongo DB and IOs. This choice could come down to the technical skills of the user. The usual drawback to using Cloud based infrastructure technology relates to cost of usage of services. Overall, Cloud based technologies are great for getting Citizen Science projects started but long term they have some costs related to maintenance and operations (M&O) of the project. Based on the analysis of the Cloud Services available, GAE seems to provide the most attractive technology to start a Citizen Science mobile app that promotes dynamic configuration for multiple projects.

An evaluation table containing pricing, effort required to get project started, APIs availability and trial period services was built to analyze best choice available for Citizen Science projects (see table 3.1 below). Note that only major Cloud Service providers with early-on mobile and backend Cloud service connected framework provider were considered for analysis listed below.

3.5 Database Technology Comparison

The backend database used in the Citizen Science project needs to accommodate dynamic fields i.e. database table's column can be added on demand. This constrain is easily accommodated by NoSQL databases.

Along with IOs mobile app, Abishek's [3], used Mongo DB technology for the backend database. Mongo DB is an open source cross platform database that promotes document based NoSQL technology. This is a free, open source technology and can easily be used as backend database for Citizen Science projects because of its compatibility with various technologies i.e. PHP and JSON and capability to dynamically accommodate table structure update that accommodates Citizen Science project requests. The concerns about this technology for Citizen Science projects is the Maintenance and Operations (M&O) once the database size begins to increase and storage space begins to incur fees. Overall Mongo DB technology is easy to use and get started, which helps speed up the development lifecycle of Citizen Science mobile applications.

For this paper's project a Google App Engine database API was used for its backend data storage. The Google App Engine (GAE) – Android connected framework provides free NoSQL database (REST) API for development and service trial through its, "Cloud Datastore," service. This database API is by default, connected to the GAE Android framework, so this also helps speed up the software deployment. The GAE database API requires no configuration to get the project started and is free with limitation of nine hours of usage per day and at most 1,000 queries per day. The service usage hours is reset at midnight every day. In contrast, Monbo DB has unlimited queries per day. Mongo DB seems like a better choice for Citizen Science projects. On the other hand, using Mongo DB requires additional configuration and set up time. Another issue related to the GAE database service is the data size.

The GAE database API, "Cloud Datastore," has limitation on the data size that can be uploaded. The database's data size limitation restricts image upload. However, GAE does have other database APIs that accommodates image upload such as, "Big Query," which is not free for usage. Due to this database limited data size insert to the database, this project opt out with additional API service to support the project's requirement to be able to share pictures. To support image upload capability, the mobile app was further customized to connect to DropBox's mobile Android API: this API is free to use with a 2

Giga Byte data usage limit. With GAE's database API and DropBox's Android API one can built a mobile app that is free to use, accommodates Citizen Science project requirements such as notes and images sharing capabilities. The data upload limitation issues is one of the drawbacks with GAE, which Abishek [3] did not have to deal with.

Table 3.1 Cloud Service Providers With Mobile App connectivity analysis of price and services

Name	IBM Blue Mix	Microsoft Azure	Amazon AWS (Amazon Web Services)	Google Cloud Platform
Provides Free Service	Temporarily	Temporarily	Yes	Yes
Ease Of Getting Started: Easy, Moderate, Extensive	Easy	Easy	Easy	Complex
Free Memory Space Provided	2 GB	0 GB	1 GB	Varies per API
Free Trial Period Credit	$0	$200	$0	$0
Trial Period Duration (days)	30	until credit runs out	365	28 days depending on /API
Provides Free APIs	Temporarily	No	Yes	Yes
Requires Credit Card To Get Started	Yes	Yes	No	No
Always On	Yes	Yes	No: 750 hours/month	NO: 500 request/user
Provides Cloud Service	Yes	Yes	Yes	Yes
Provides Virtual Machine	UNIX or Windows	UNIX or Windows	UNIX or Windows	UNIX based (free)
Pricing	$0.07 per GB-hour	$0.13 per hour	Varies: $0.013 to $0.105 per hour	free
URL	https://ace.ng.bluemix.net	https://account.windowsazure.com	https://aws.amazon.com	https://cloud.google.com

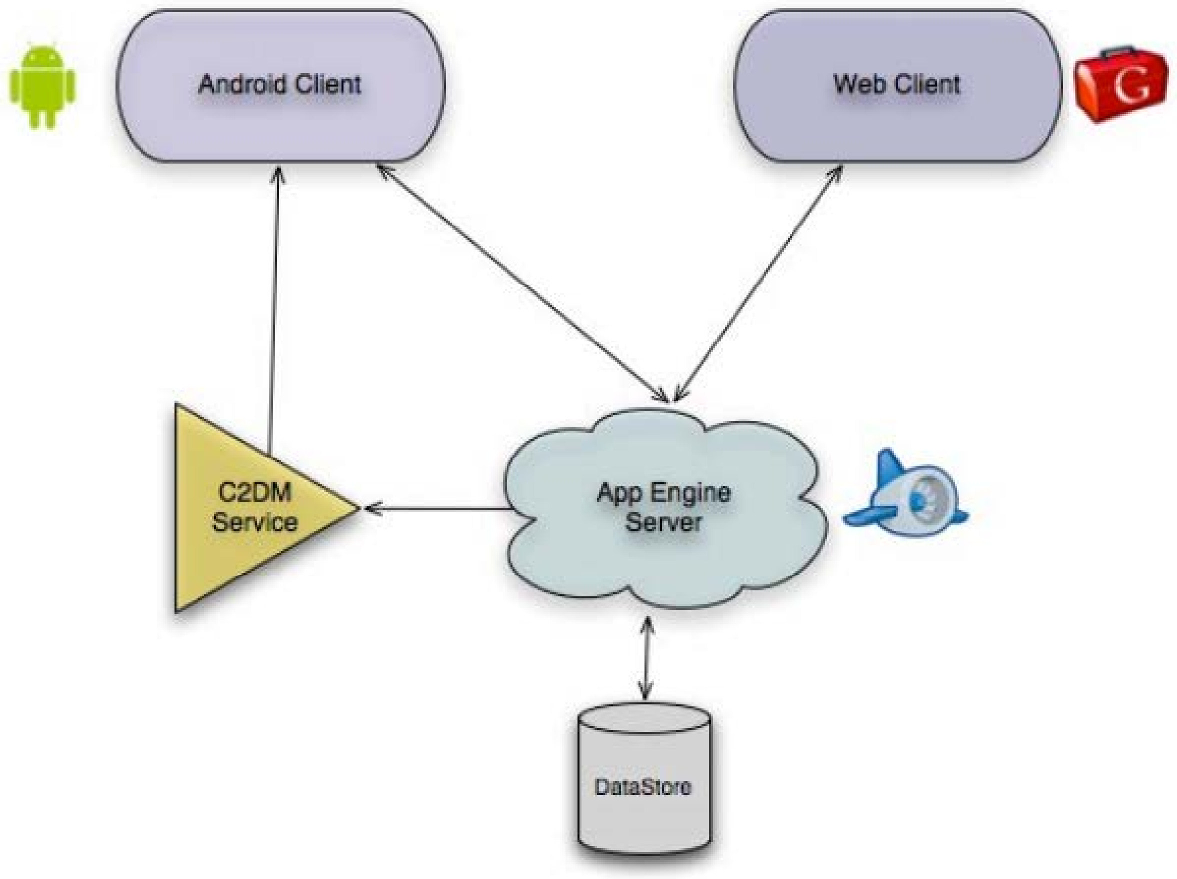
4.0 Architecture And System Design

This section of the document is organized in three sections. Section 1talks about local device's application architecture. This section goes into details of how the backend works along with it COTS functionality. Section 2 talks about the mobile application high level description of the application and its core requirements achieved. Section 3 describes the environment setup for Google App Engine (GAE) with mobile app and the tradeoffs of using this technology versus a dedicated server. Section 4 is a discussion of the GAE web and Android connected architecture. This part of the discussion talks about user interface which administrators use for APIs and data management.   Section 5 talks about the challenges of this project and how alternative implementations were used to achieve the main requirements in order to still use the GAE backend COTS functionality that promotes free usage.

4.1 Android App System And Architecture

The Google App Engine connected with Android framework is set up to promote the development of Android apps; the backend is built out and has several APIs that can be turned on. Out of the box, this framework has a simple broadcast functionality that sends message to the backend system, which is viewable through the web client. For this project we developed the Android Client to accommodate a Citizen Science project.

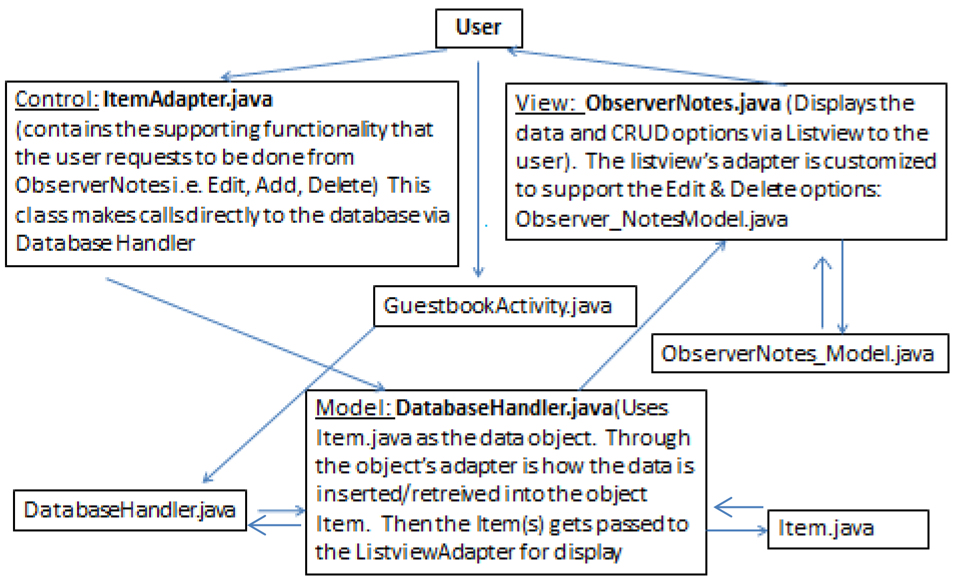
Image 4.1.1 A diagram based on the Google App Engine described in [1], which is the COTS GAE used for this project's mobile application.



The Android mobile app for this project was developed on top of the Android Client shown on the image above. In addition to the overall system architecture mentioned above, the mobile app was further architected to accommodate changes to the User Interface with minimal to no changes required for the backend or middleware. Throughout the various User Centric Design iterations various front end updates were made and the backend end did not need to get updated. Some of the middleware for the data upload had to be modified but this still did not required backend updates; the system was built with model view control architecture. The data capturing mechanism uses regular text fields to allow the user to capture field observations. The image associated with each data record is simply a text field with the image's path in the local device. Blob was considered for this application, but the backend engine APIs were not free, so we had to figure out alternatives i.e. DropBox which is discussed in details in section 4 of this chapter. The data inputted gets saved into the device's local SQLite Database. If additional text fields are needed to capture additional observations, the user may do so by using the dynamic field capturing data functionality. This functionality allocates additional data storage space in the device's local database to accommodate the additional data. The overall design of the mobile app was completed as part of this project.

4.1.1 Overall mobile device application architecture design. Note that GuestbookActivity.java was provided by GAE Android package. The remaining parts were developed as part of this project. The broadcast functionality, which is also part of the GAE Android package, is not shown on this picture.

ObserverNotes System Architecture


User
Control: ItemAdapter.java (contains the supporting functionality that the user requests to be done from ObserverNotes i.e. Edit, Add, Delete) This class makes calls directly to the database via Database Handler
View: ObserverNotes.java (Displays the data and CRUD options via Listview to the user). The listview's adapter is customized to support the Edit & Delete options: Observer_NotesModel.java
GuestbookActivity.java
ObserverNotes_Model.java
Model: DatabaseHandler.java (Uses Item.java as the data object. Through the object's adapter is how the data is inserted/retreived into the object Item. Then the Item(s) gets passed to the ListviewAdapter for display
DatabaseHandler.java
Item.java

The application was designed with the MVC architecture in mind to improve the overall presentation and keep maintainability feasible. This application has a complex Android Listview with a custom View Adapter to accommodate the Upload, Edit and Delete functionality for each data record. The various updates and modifications done to the ObseverNotes.java during focus group were easily manageable due to the MVC architecture. The Item.java and ObserverNotes_Model.java was kept the same during each update. The upload button was modified twice: it went from being a button to a row click to an icon click, which is its current functionality. The data add functionality used a different user interface layout, but still used this architecture to be able to store data locally.

The data for this package is stored in the local device's database, Android SQLite. A single table with one primary key was used to store the data. Below is a description of the local database table, "Notes:"

4.1.2 Local Android device Database description

Local Database fields:	**Data Type**	**Primary Key**
KEY_ID	int(32)	Yes
KEY_NAME	String varchar(255)	
TAXON	String varchar(255)	
COMMON	String varchar(255)	
LIFEFORM	String varchar(255)	
STATUS	String varchar(255)	
FAMILY	String varchar(255)	
BLOOM	String varchar(255)	
Data_NOTES	String varchar(255)	
Other_Notes	String varchar(255)	
IMAGE_URL	String varchar(255)	
DATE_TAKEN	String varchar(255)	
DELETED	String varchar(255)	

The table was designed and architected as part of this project. An additional table could have been created to accommodate the settings if it was not handled by the administrator. The primary key was assigned as an auto increment integer to ensure uniqueness to each record of data created by the user. The other fields were all set to string to avoid issues with data conversion/type casting when going to from the cloud. These types of issues are particularly common when dealing with date data type fields. The overall database entails of a single table in the local Android device.

4.2 Mobile Application

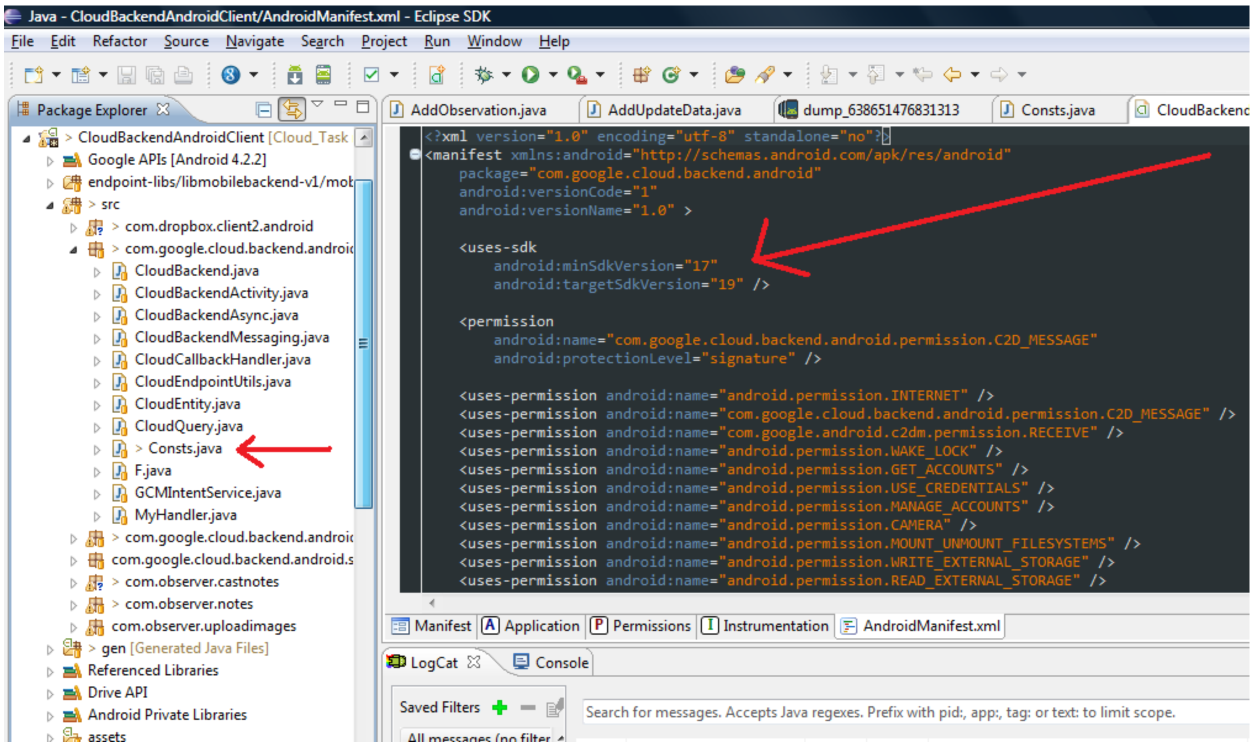
The Android mobile application built for this project accommodates local data storage along with dynamic field data capturing and capability to share the data that is stored locally during field observations. In addition the mobile application has capability to send data to a backend server API with GAE and DropBox. This mobile application is free of charge and its APIs are also free with some usage limitation; this application adheres to the Citizen Science projects requirements. Overall this application imposes no financial burden on the research project and promotes a fast software set up for the project.

4.3 Environment Setup

In order to start development for a project the proper environment and tools needs to be setup. Since this project used Cloud services provided by Google App Engine, it's worthwhile mentioning the steps taken to get the environment ready. These steps described below are the tradeoffs to traditional method of dedicated server for a project. These steps for this project replace the following: setting up a server i.e. installing an operating system, getting the backend server and database connected to the Android mobile app and testing the connectivity.

4.3.1 Google App Engine Connected Android Mobile Application Setup

Image 4.3.1 Notice the minimum Android API version 17 setting and also java file Consts.java, which is used to set up the GAE's token for the development environment.

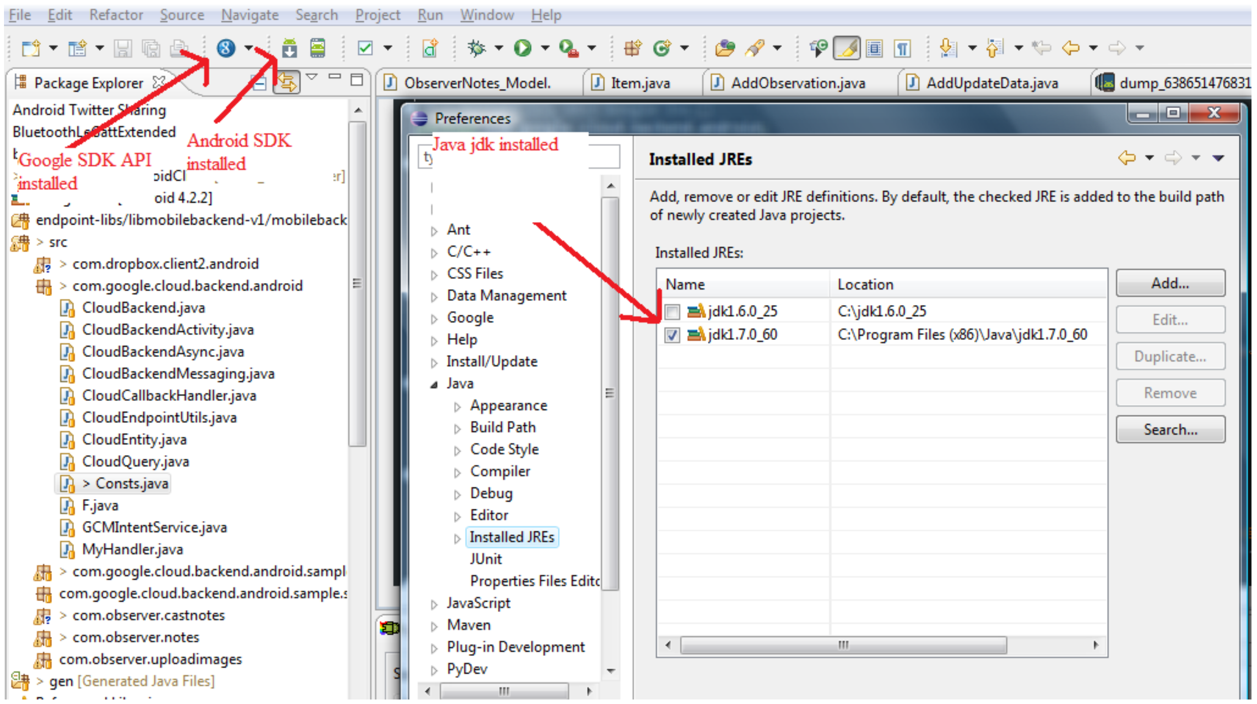

Java - CloudBackendAndroidClient/AndroidManifest.xml - Eclipse SDK
File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer
AddObservation.java
AddUpdateData.java
dump_638651476831313
Consts.java
CloudBackend
Manifest
Application
Permissions
Instrumentation
AndroidManifest.xml
LogCat
Console
Saved Filters
Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.
All messages (no filter

```
> CloudBackendAndroidClient [Cloud_Task
  Google APIs [Android 4.2.2]
  endpoint-libs/libmobilebackend-v1/mob
  > src
    > com.dropbox.client2.android
    > com.google.cloud.backend.androic
      CloudBackend.java
      CloudBackendActivity.java
      CloudBackendAsync.java
      CloudBackendMessaging.java
      CloudCallbackHandler.java
      CloudEndpointUtils.java
      CloudEntity.java
      CloudQuery.java
      > Consts.java
      F.java
      GCMIntentService.java
      MyHandler.java
    > com.google.cloud.backend.androic
    com.google.cloud.backend.android.s
    > com.observer.castnotes
    > com.observer.notes
    com.observer.uploadimages
  > gen [Generated Java Files]
  Referenced Libraries
  Drive API
  Android Private Libraries
  assets
```

```xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.cloud.backend.android"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="17"
        android:targetSdkVersion="19" />

    <permission
        android:name="com.google.cloud.backend.android.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="com.google.cloud.backend.android.permission.C2D_MESSAGE" />
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.USE_CREDENTIALS" />
    <uses-permission android:name="android.permission.MANAGE_ACCOUNTS" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

The GAE Android connected framework provides the backend and sharing capabilities with its COTS package. This Android framework requires Google API 17 or higher, Java version 1.6 installed in the development environment, the Android device needs to have Android operating system version 4.2 or higher installed on the mobile device. Otherwise the mobile application will not run. Below is an image of the COTS GAE connected Android application setup:

Image 4.3.2 Eclipse Setup for Android development with GAE



```
Android Twitter Sharing
BluetoothLeGattExtended
oidCl
oid 4.2.2]
endpoint-libs/libmobilebackend-v1/mobileback
> src
  > com.dropbox.client2.android
  > com.google.cloud.backend.android
      CloudBackend.java
      CloudBackendActivity.java
      CloudBackendAsync.java
      CloudBackendMessaging.java
      CloudCallbackHandler.java
      CloudEndpointUtils.java
      CloudEntity.java
      CloudQuery.java
      > Consts.java
      F.java
      GCMIntentService.java
      MyHandler.java
  > com.google.cloud.backend.android.sampl
  com.google.cloud.backend.android.sample.
  > com.observer.castnotes
  > com.observer.notes
  com.observer.uploadimages
> gen [Generated Java Files]
```

Installed JREs

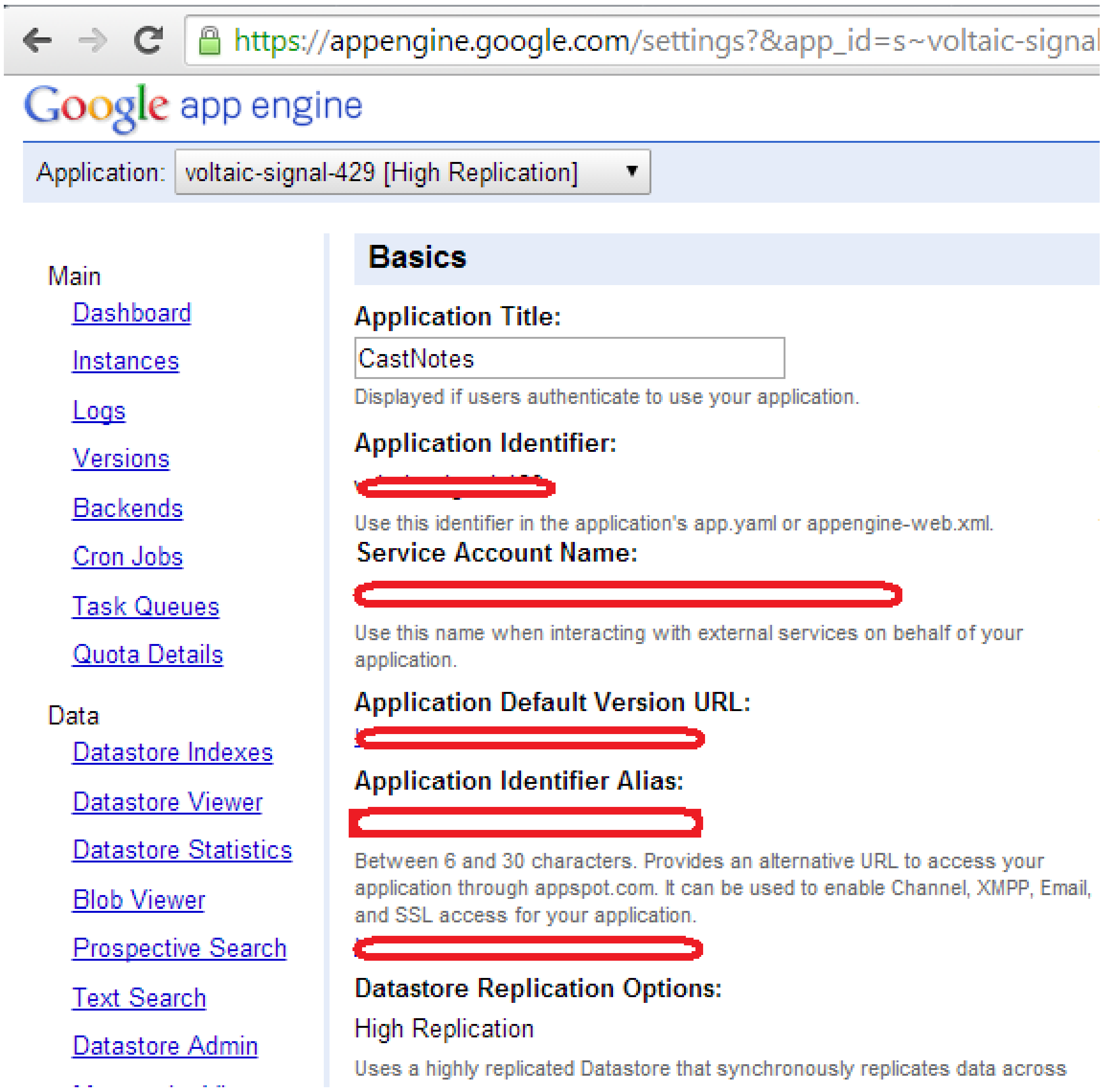
Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

Installed JREs:

	Name	Location
☐	jdk1.6.0_25	C:\jdk1.6.0_25
☑	jdk1.7.0_60	C:\Program Files (x86)\Java\jdk1.7.0_60

Above, image 4.2.2 shows a screenshot of Eclipse development environment setup for GAE with Android connected: Notice the Google API SDK icon in blue followed by the Android SDK plugins for Eclipse. The third arrow in the far right shows the Eclipse configuration using Java Development Kit (JDK) 1.7. The minimum JDK required is 1.6, but some updates to GAE's APIs require JDK 1.7.

Image 4.3.3 Google App Engine



Application: voltaic-signal-429 [High Replication]

Basics

Application Title:

CastNotes

Displayed if users authenticate to use your application.

Application Identifier:

[redacted]

Use this identifier in the application's app.yaml or appengine-web.xml.

Service Account Name:

[redacted]

Use this name when interacting with external services on behalf of your application.

Application Default Version URL:

[redacted]

Application Identifier Alias:

[redacted]

Between 6 and 30 characters. Provides an alternative URL to access your application through appspot.com. It can be used to enable Channel, XMPP, Email, and SSL access for your application.

[redacted]

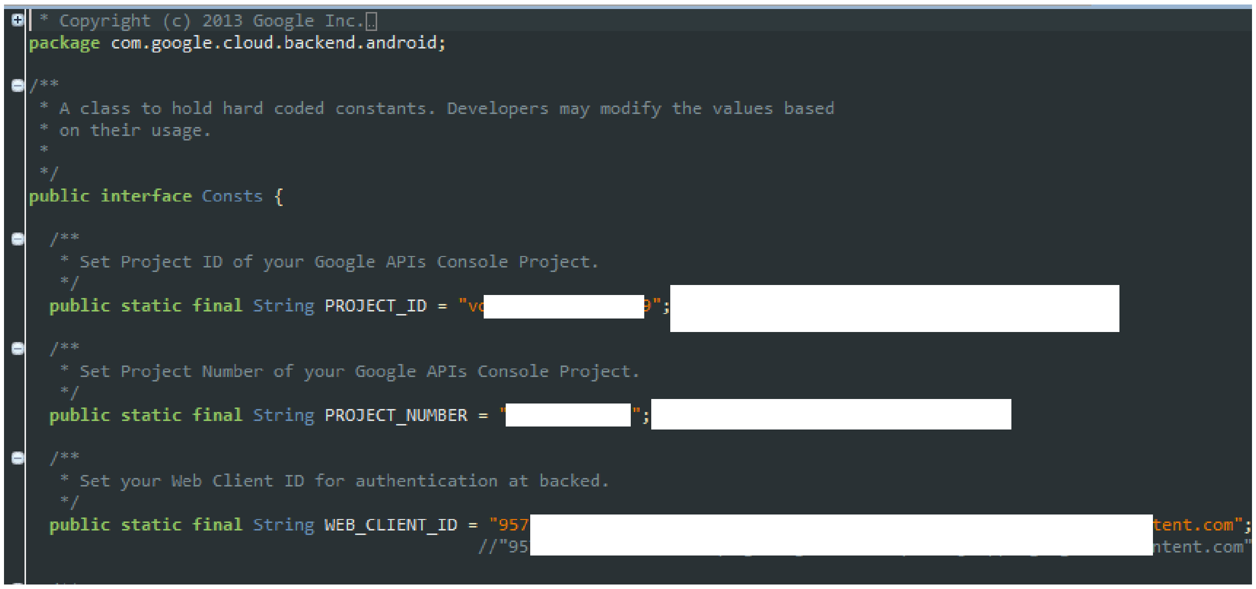
Datastore Replication Options:

High Replication

Uses a highly replicated Datastore that synchronously replicates data across

Furthermore, in order for the broadcast functionality to work, it requires proper configuration of the development environment with GAE project instantiation. During the development environment setup, the developer will obtain a token through GAE's admin web console. Below is a screenshot of the Google App Engine Console where the developer obtains the token to place into the mobile app's code:

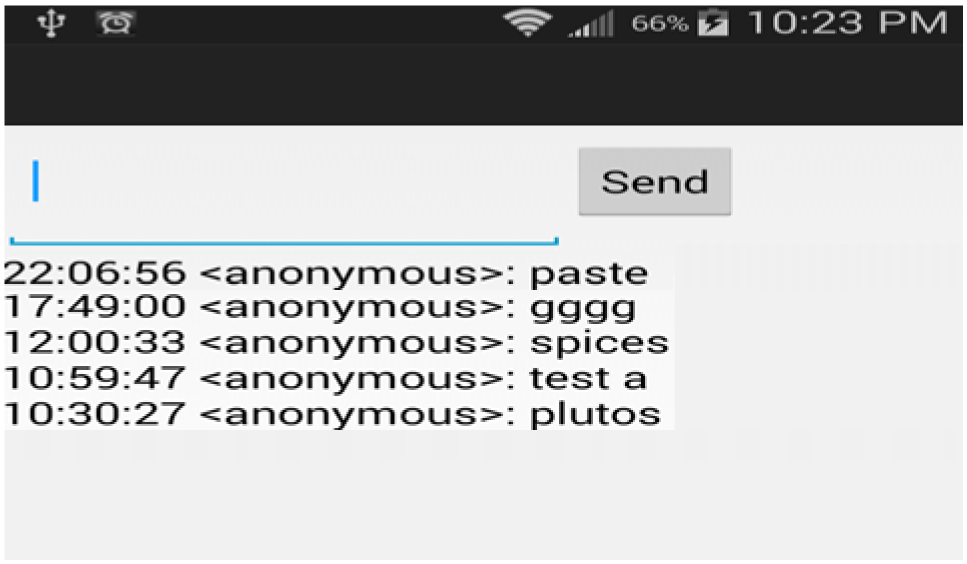
Image 4.3.4 Token placeholder for Google App Engine authentication: The keys are blanked for security purposes. The code is provided by Google as part of GAE's connected Android COTS package.



```
 * Copyright (c) 2013 Google Inc.
package com.google.cloud.backend.android;

/**
 * A class to hold hard coded constants. Developers may modify the values based
 * on their usage.
 *
 */
public interface Consts {

  /**
   * Set Project ID of your Google APIs Console Project.
   */
  public static final String PROJECT_ID = "v[redacted]9"; [redacted]

  /**
   * Set Project Number of your Google APIs Console Project.
   */
  public static final String PROJECT_NUMBER = "[redacted]"; [redacted]

  /**
   * Set your Web Client ID for authentication at backed.
   */
  public static final String WEB_CLIENT_ID = "957[redacted]tent.com";
                                          //"95[redacted]ntent.com"
```

Once the tokens obtained from GAE console is placed inside the, "Consts.java," (show in Image 4.3.2 above) file that comes with GAE's COTS Android app, the device is able to identify itself with GAE backend server through a C2DM service (see image 4.3.6 for a diagram). Above is an image of the token setup in the, "Consts.java," file.

Image 4.3.5 A trial broadcast of random messages once the GAE connected Android mobile app was setup



```
22:06:56 <anonymous>: paste
17:49:00 <anonymous>: gggg
12:00:33 <anonymous>: spices
10:59:47 <anonymous>: test a
10:30:27 <anonymous>: plutos
```

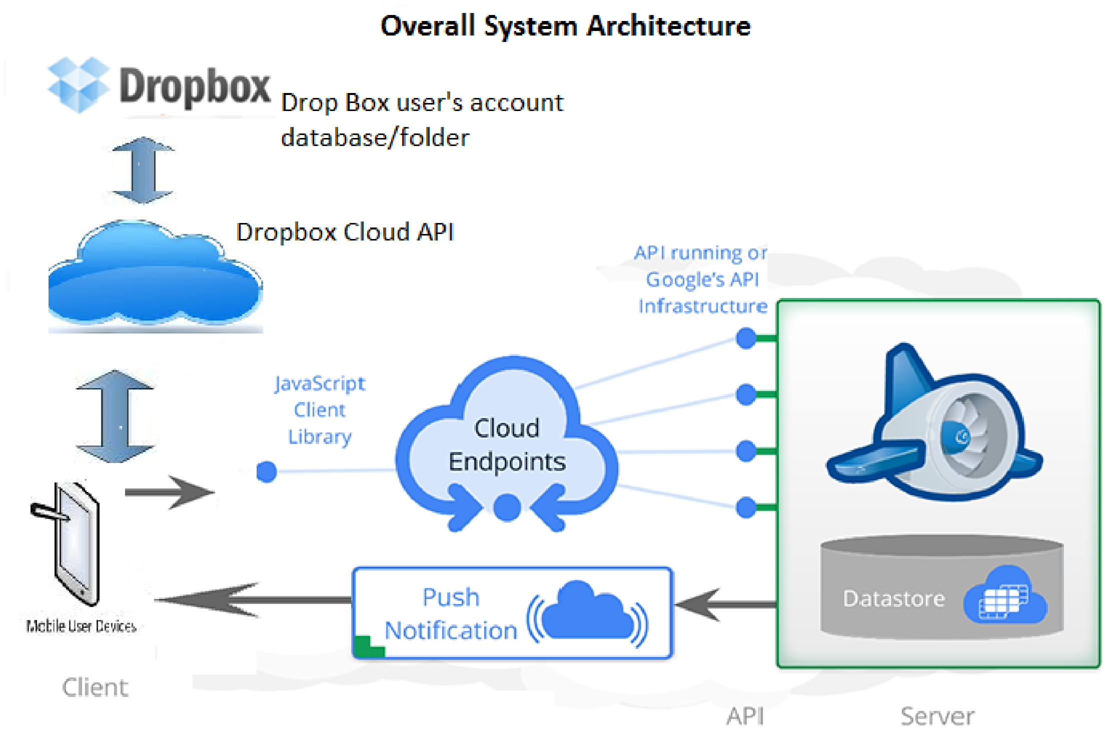
Upon setup of the development environment and proper configuration of the application, the first trial run already allows the tester to broadcast a message. The image above is a screenshot of the initial trial broadcast.

This ability to quickly start development with a mobile app that is halfway built is very attractive, but there are some drawbacks. For example the free APIs have limited data size, so images cannot be uploaded and shared. This is one of the tradeoffs to Mongo DB or other technology used to build Citizen Science mobile app because other technology may not have imposed usage limitations.

4.4 Google App Engine And APIs

To share the data, the application's device shall have connectivity to the internet (web connection) or cellular phone service connection. The data sharing mechanism allows the user to select a field observation record from list of data saved on the local device. Upon selecting a field the user is prompted to authenticate into DropBox for the image sharing mechanism if the image has not been shared yet. Otherwise the record is broadcasted immediately through GAE's API mechanism. Once the user is authenticated with DropBox, the image gets uploaded and the image's url is obtained from the DropBox API. Then the image's url is updated into the data record in the device's local database. Then, the record gets broadcasted through GAE's text upload mechanism.

Image 4.3.1 A diagram of the modified GAE connected Android application set up to upload images to DropBox


Overall System Architecture
Dropbox
Drop Box user's account database/folder
Dropbox Cloud API
API running on Google's API Infrastructure
JavaScript Client Library
Cloud Endpoints
Datastore
Push Notification
Mobile User Devices
Client
API
Server

Above is a diagram of the overall mobile application for this project.

The, "Client," mobile user device in image 4.3.1 has a highly modified version of the COTS Android application. The mobile application contains the backend server connectivity with capability to send and receive messages to and from the backend server. In addition, this connectivity was slightly modified at the Android side of the connection to save other user's broadcasted data to the device's local database. A broadcast is simply an inquiry to get the data in the backend database, which takes place every time the application is started.

Below is a snipped of the code used to ensure existing records versus data received.

```
// convert posts into string and update UI
  private void updateGuestbookUI() {
    final StringBuilder sb = new StringBuilder();

    String getBroadcast = "";

    if(BROADCAST_SET == 0){
            //loop to display every message received from the cloud
          for (CloudEntity post : posts) {
             sb.append(sdf.format(post.getCreatedAt()) + getCreatorName(post) +
"Broadcasted:  " + post.get("name")
                + "\n");

             //break down the messages received into tokens and
             //parse them to make it insertable into the database
             if(getBroadcast == ""){
               //this is the first item
               getBroadcast += post.get("url").toString()+"@#"+
                                   post.get("name").toString()+"@#"+
                                   post.get("notes").toString();
             }
             else{
               //these are the remaining items
                   getBroadcast += "|"+  post.get("url").toString()+"@#"+
                                         post.get("name").toString()+"@#"+
                                         post.get("notes").toString();
              }
          }


          //set the text view section to display broadcast
          tvPosts.setText(sb.toString());



          //start intent to insert broadcasted messages into database
          Intent intent=new Intent(getBaseContext(),AddUpdateData.class);
             intent.putExtra("UPDATE", "broadcasted");
             //pass data to be stored in DB
             intent.putExtra("Guestbook", getBroadcast);

             intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
             startActivityForResult(intent, BROADCAST_INSERT);
    }
  }//end receiving all posts
```

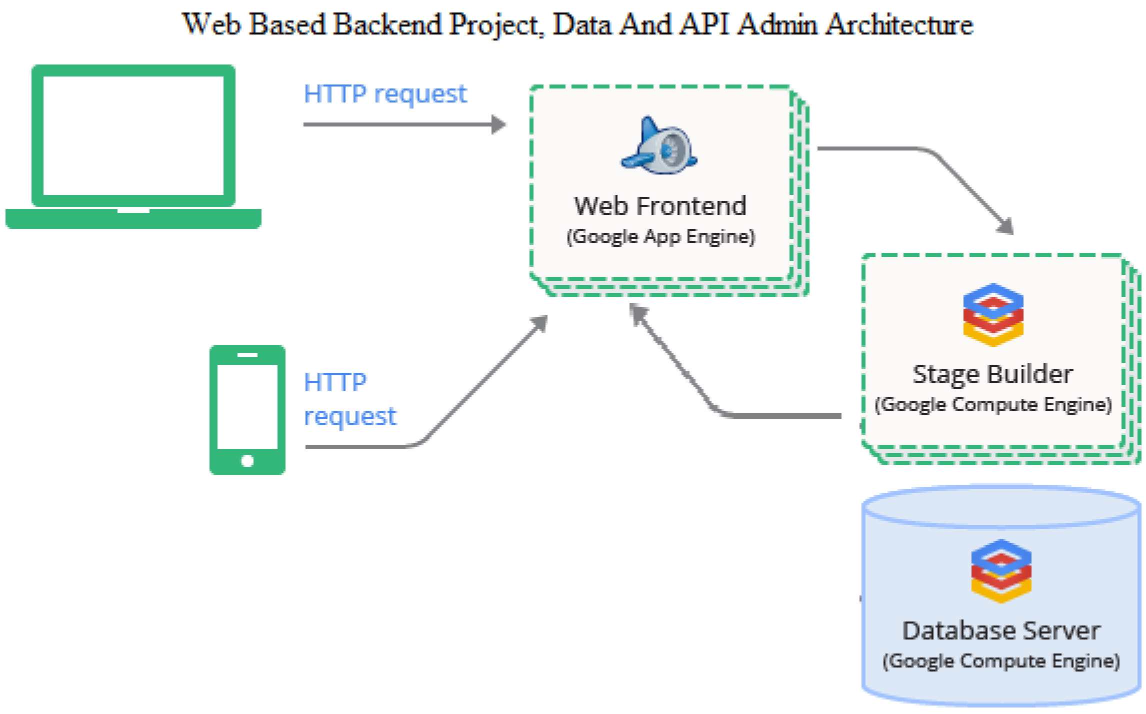
Duplicate records get filtered during a broadcast. The broadcast of records received is compared to the local device's database records to ensure that the existing record does not already exist, so duplicate records are not inserted. If the record recently received does not exist in the local list of records, then an instance of the data's model is created with the new data. Then the data model object is passed onto a database instance which inserts the record into the database.

The Google App Engine (GAE) backend server allows user to store data and broadcast it to other recipients through its APIs. The sharing of the data occurs only when an Android Observer app calls the GAE API to store data locally. Then, the GAE backend framework broadcasts the data to other users that have the app on the Android device. Therefore the Android application can only share data once it's connected to the web or cell phone connectivity service. As data gets uploaded, it's broadcasted to other users whom receive broadcasts upon start of the application. For this project, a slight code modification was done, so newly received information via broadcast gets inserted to the device local storage. This allows for every user to have the same copy of data set locally. Furthermore, under settings, the user can select a project to filter data. Through project selection the user only sees the data for the desired Citizen Scientist project. Overall the GAE backend server has ability to store and share data and the local mobile app has been modified to store the new data received.

Through [1] we can verify that the database on GAE, a NoSQL Google App Engine database via Data Store API can be used to support an Android Observer Application.

The Data Store API provides database transactions, which the Android Device will be able to consume to support various project(s) with dynamic data fields. Through the Data Store API, the user can upload a variety of data fields' data i.e additionally fields added through the configuration. The NoSQL database promotes unstructured data read and or write, so various application configurations can write i.e. upload data to the GAE database without the need of a new table being created. The GAE database APIs promotes application metadata sharing, which is ideal for Citizen Science project that require dynamic data field functionality.

Image 4.4.3 Diagram of how project administrator monitors the backend server APIs and data: A high level diagram of the Admin Console

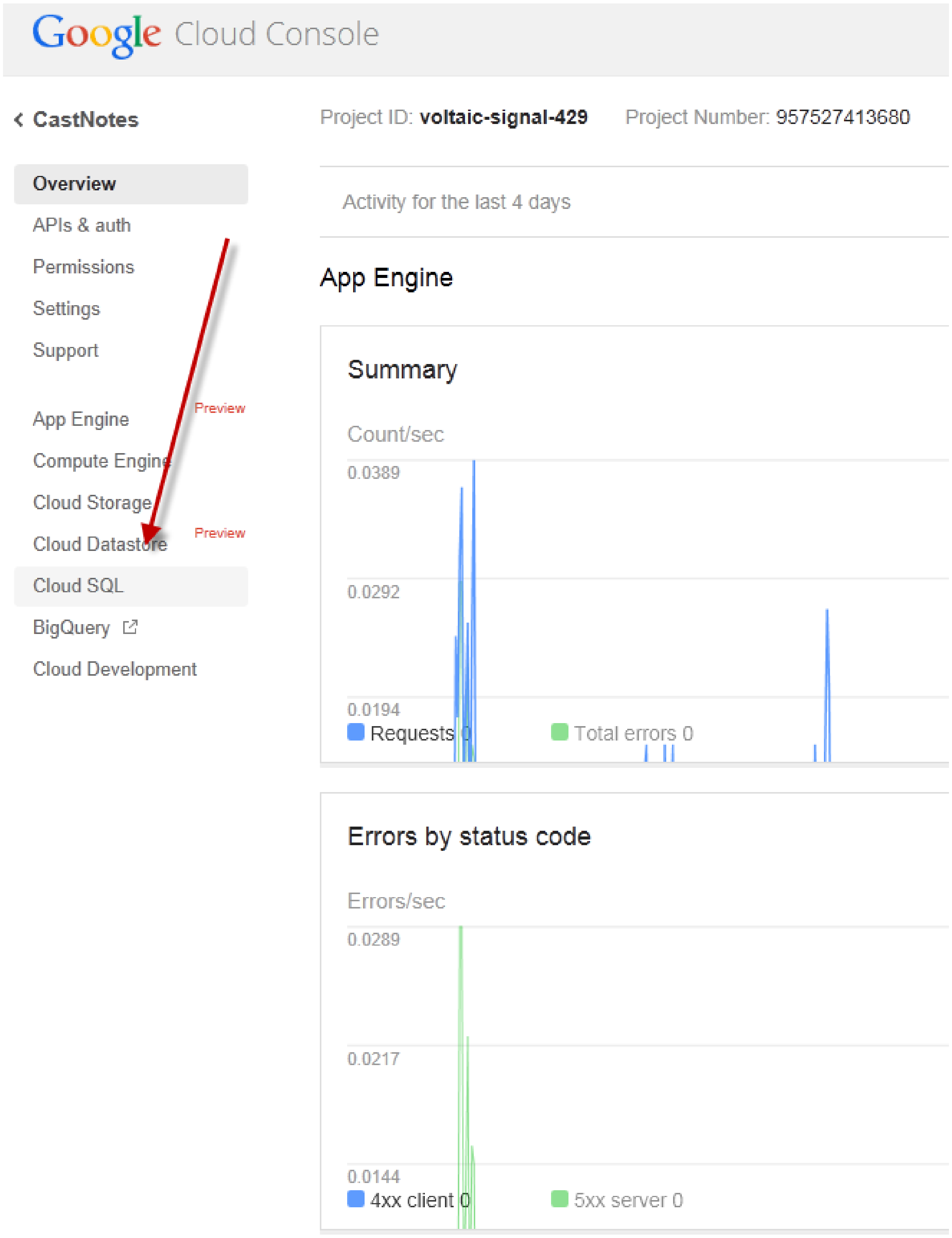


Note that Google App engine provides some free trial features and APIs with specific quotas. Therefore the backend server's APIs can be kept in, "Off," mode until data sharing is executed. If the free daily quota limit is reached, the user will not be able to upload or download data until the next day when the free quota is reset.

Below is an image of the GAE Admin Console.

In addition to the APIs provided by GAE, through the, “Web Client,” administrators can monitor and clean the data uploaded, “Android Client.” Admin users can view the database’s data via web browser. Through GAE’s documentation [2] indicates that Google App engine provides Admin SDK API for free. This API allows users to update, review or delete undesired data that has been uploaded into Google App engine cloud database. The Administration Console feature provided by Google App engine has three levels of access a user can obtain for this functionality: Viewer, Developer, and Owner [1]. Through the Administration Console the user can maintain the database, application’s backend instance usage, monitor usage traffic and other features provided through the Administration Console. The GAE Administration console provides ability to maintain the backend sever and its stored data.

Image 4.4.4 This image has an arrow indicating where to click to view the backend database



Project ID: **voltaic-signal-429** Project Number: 957527413680

Activity for the last 4 days

App Engine

Summary

Errors by status code

4.5 Project Challenges

As part of the Citizen Science project requirements, images captured during field observations need to be shared along with field observation notes. There were some major challenges to overcome while building the mobile application: First, sharing the images was one of the challenges in this project due to the image file's size constraints imposed by Google App Engine's free API service. To share the image becomes more challenging as the image needs to be uploaded before sharing it. The second challenge was the ability to accommodate dynamic field configuration for any project. The third challenge is the ability to capture broadcasted data locally without creating duplicate records. Another difficult challenge for this project is to build a mobile application that is free to use and adheres to the technical requirements. Hence the image file sharing challenges.

The GAE connected with Android COTS framework restricts large data sharing capability with financial burdens. As a workaround to financial restrictions, additional API was used in this project: DropBox API for Android was used to comply with image sharing for Citizen Science projects. As an alternative solution to mitigate large data sharing (text and image sharing), this project uses DropBox Android mobile API to upload images and then share the image's url. This tradeoff actually works better for every user using the mobile app because the DropBox API minimizes the network usage for each user by only sharing the image's url. Furthermore without needing to upload or download the image with the notes at once allows for faster data sharing capabilities i.e. already uploaded images do not need to be re-uploaded. Lastly, if a user does not wish to download the image and simply view it online, it can easily be done with the image’s link.

5.0 User Interface Design

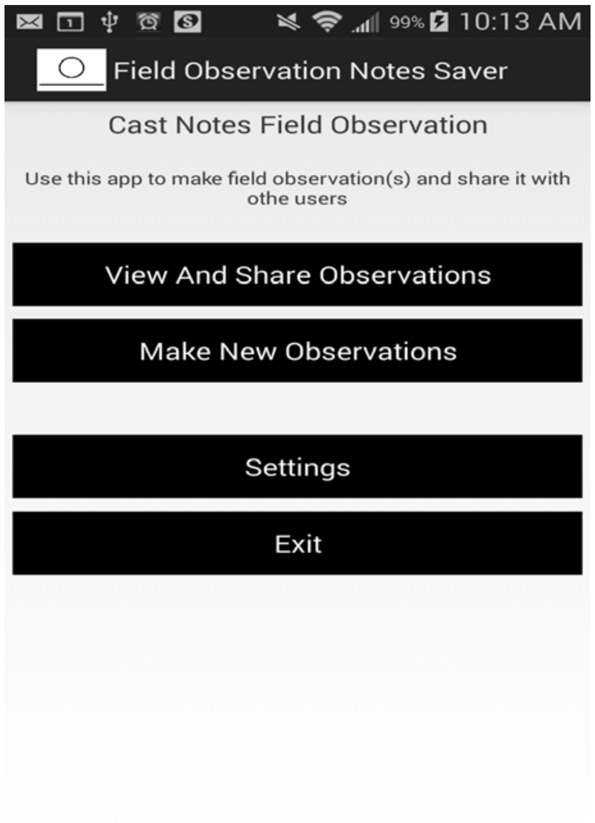
This chapter covers the User Interface design methods followed by functionality built. Various Focus Groups were used for User Interface design refinement. User Centric Design engineering design methodology was used during the Focus Groups in order to achieve the current user interface. The last section of this chapter covers the focus group outcomes, which were done to achieve the final mobile app version.

5.1 User Interface Design Methods

The User Interface of this application was first built with the GAE's COTS framework. Once all the backend functionality was built then, various rounds of focus group were performed to reach the current version of the application. See the screenshots shown in section 4.2. User Centric Design (UCD) was used for the user interface development of this project during the focus groups. Through the revisions of this document, the reviewer also tested the application. Then the tester provided feedback about the usability of the application and what changes should be made. Pictures of the focus groups drawings are included later in this chapter.

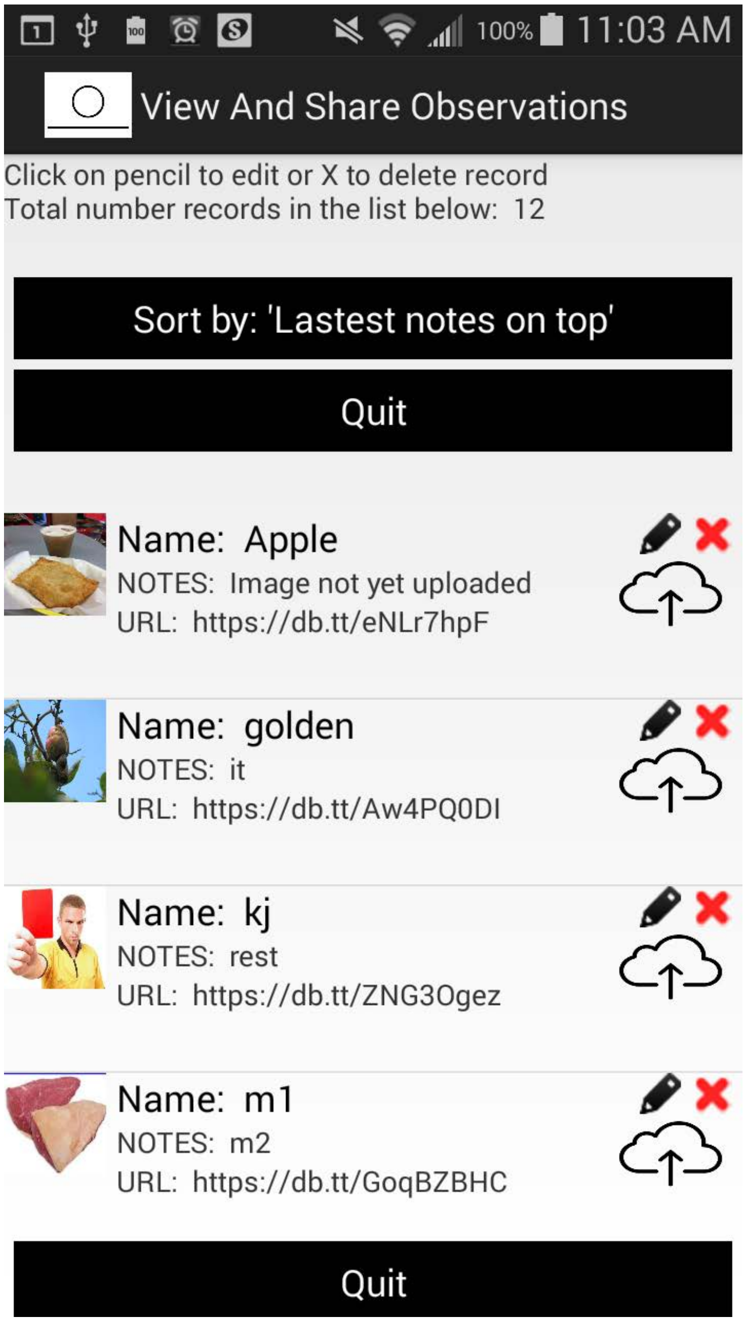
5.2 Application User Interface Functionality

Below is a screenshot of the application's main screen.


99% 10:13 AM
Field Observation Notes Saver
Cast Notes Field Observation
Use this app to make field observation(s) and share it with othe users
View And Share Observations
Make New Observations
Settings
Exit

The user interface for the mobile application contains interactive buttons with labels to allow the user to easily navigate throughout the application and perform desired tasks such as take notes, upload a record, associate image with notes, edit notes, select project to filter data by and quit the application. The, “Exit,” button closes the application. The, “Settings,” button allows user to pick which project to set this application for. By default the settings is set to, “All,” for all projects. When the setting is set to, “All,” notes taken are set to be shared for all projects.

Below is the screenshot of the “View And Share Observation,” activity of the application.



Click on pencil to edit or X to delete record
Total number records in the list below: 12

Name: Apple
NOTES: Image not yet uploaded
URL: https://db.tt/eNLr7hpF

Name: golden
NOTES: it
URL: https://db.tt/Aw4PQ0DI

Name: kj
NOTES: rest
URL: https://db.tt/ZNG3Ogez

Name: m1
NOTES: m2
URL: https://db.tt/GoqBZBHC

The, “View And Share Observation,” button allows user to view and share existing observations made. Other users’ data broadcasted can be viewed but not shared or modified. Furthermore, the list of existing observation contains an image, sort button that allows sort by latest notes, notes' name and notes broadcasted. The Quit button simply returns the use to the previous activity screen.

The user is able to edit existing record by clicking on the pencil icon of that record for further editing. If the record does not contain a pencil it means the record was received from other user of the application and is not editable by this user. Note that the project administrator can edit, create or delete data from the backend server.

The user is able to delete a record from the local device by clicking on the red x icon of the record that the user desired to delete. However, the record is not deleted from the backend database server. Only the administrator has permission to delete a record from backend database server.

Each record is clickable and takes the user to a more detailed view of the record, which displays the texts related to that record and its image.

The upload button uploads the record information to the cloud. If the record is not uploaded yet, it'll first upload its image to the cloud (DropBox). Upon successful upload of image the system gets the image's url and updates the local record. Then the record gets sent to GAE for database insertion and broadcast.

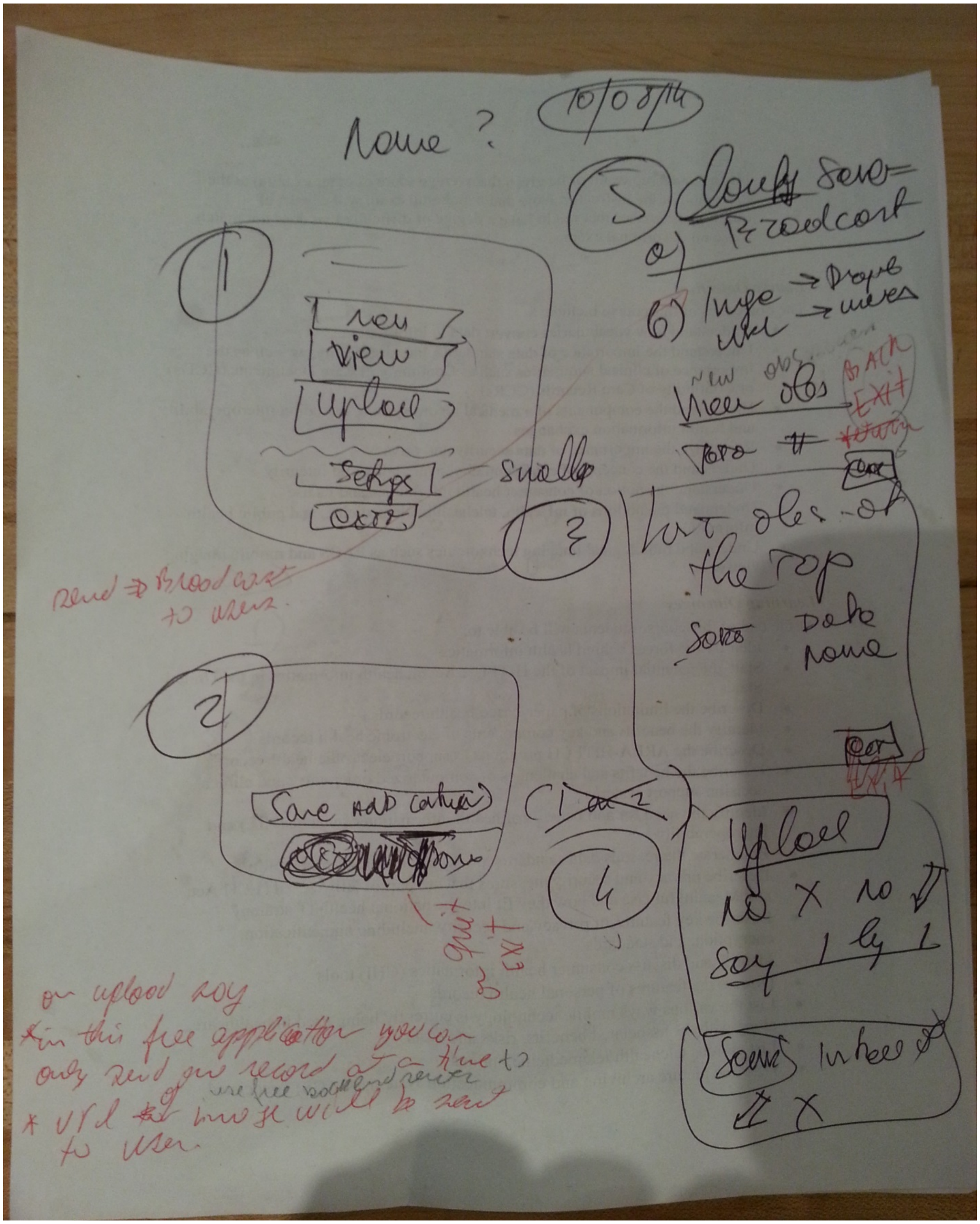
Note that records that have not been uploaded yet have a white background. The records that have been uploaded have a grey background. The user is also able to sort the list by broadcasted records.

The quit button will take the user back to the application's main screen.

5.2 Focus Group

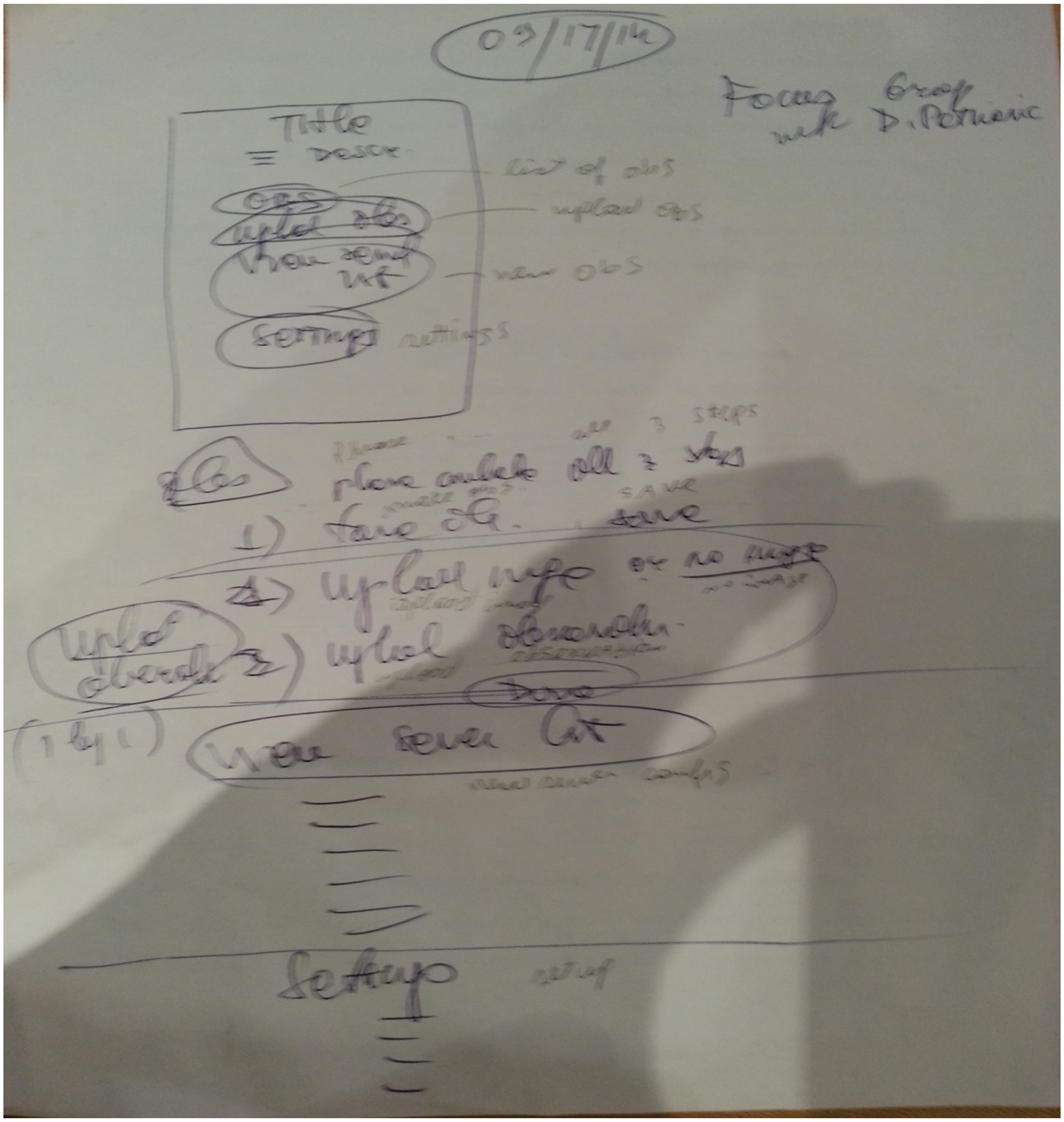
A User Centric Design approach was used to develop this application. An initial version was built with the COTS mobile app provided with GAE’s framework. All the core functionalities (upload image, share data, dynamic data capturing) were built before the first Focus Group. Then various rounds of updates were made.

Image 5.2.1 Shows second Focus Group iteration


10/08/14
Name ?
5
Daily save = Broadcast
a)
b) Image URL → Dropb → users
New obs
View obs
BACK
EXIT
Save #
OK
1
new
view
Upload
Setups
EXIT
scroll
3
List obs of the top
Sort Date name
OK
EXIT
send ⇒ Broadcast to users
2
Save and continue
quit
EXIT
(1 or 2)
4
Upload
no X no
say 1 by 1
Send
In here
on upload say
* in this free application you can only send one record at a time to use free backend server
* URL of image will be sent to user.

Throughout the review process of this project various iterations of user testing and functionality review were conducted to improve the application's usability. Below are screen shots of the drawings and revision of the User Interfaces.

Image 5.2.3 First Focus Group iteration



At the first Focus Group the broadcast functionality was in the main screen of the application and much updates were done since this first iteration. This is just to show how much change took place throughout the focus groups.

6.0 Evaluation

This chapter covers the Android mobile app and backend server test results. This chapter first talks about the tests objectives which is to ensure data is captured and shared as expected. Next part of the chapter covers the test plan, which covers the functional and nonfunctional requirements. Lastly the test results are presented: executed and its expected results versus actual results.

6.1 Quality Assurance (QA)

6.1.1 Test Objective

The application testing covered the core functionalities required by Citizen Science projects. The main functionalities entail ability to capture data, associate rich information data such as images with text, share the data through a backend server and have ability to use these functionalities with various Citizen Science projects. All these core functionalities shall be free to use in the Android mobile app and with minor restrictions in the backend database.

6.1.2 Test Plan

Based on the functional and nonfunctional requirements the test cases were derived to ensure quality assurance was met. The requirements for this project cover a variety of functionality from backend, mobile application, front end web and mobile application and pricing (free of use). The test cases cover these requirements from different levels of testing and user’s point of view.

Test Cases Table

Test Case	Expected Result	Test Result	Notes
Mobile application shall allow user to capture text data	User can input text and save data		
Each record of data has a unique identifier in database	Each record is saved locally in the device and has a primary key		
Each record of data is saved locally in device	Each record can be viewed in the application's front end and database backend		
Each record of data has a title	During new data being added, "Name:," is a required text field and user shall input text in this field before begin able to save a record		
Mobile application shall allow user to capture dynamic data	During new data being added user can click "Add Additional Text Fields," button to capture additional data		
Mobile application shall allow user to associate image with data captured	During new data being added, once the user clicks "Save And Continue," user is prompted to select an image from device's gallery to pick an image. Otherwise a default Android Icon is associated with the record		
Mobile application shall allow user to edit record of data captured	During, "View And Share Observations," user can click on pencil icon to edit record and image of record		
User shall only be able to read and not modify data that was captured and shared by other users	During, "View And Share Observations," no pencil icon to edit record is available		
Administrator shall be able to add data captured	User logins to Google App Engine interface and adds a record to the database		

Administrator shall be able to edit data captured by any user	User logins to Google App Engine interface and updates a record to the database		
Administrator shall be able to delete data captured by any user	User logins to Google App Engine interface and deletes a record to the database		
Mobile application shall allow user to select a project in mobile application to allow filtering of data based on project selected.	User clicks on, "Settings," button from main activity and is able to select a project		
Mobile application shall allow user to upload a record of data collected	During, "View And Share Observations," user can click on pencil icon to Cloud Icon record and image of record		
Project coordinator shall be able to view data in backend server	User logins to Google App Engine interface and views records in the database		
Application Android services shall be free of charge for usage.	Project administrator can add a new project in the backend database by adding a new database and inserting a record with project name.		
Application's backend server services shall be free of charge according to service provider's term and conditions.	Project administrator can monitor usage traffic and turn off APIs based on usage to stay within free bandwidth		

Listview image shall be at most 1,024 Kb of size, so this can easily fit within the listview display size.	User notices that a large image associated with record is copied and a smaller image is created to be associate with the record		
Database backend shall accept a dynamic increase number of fields of data to upload	User uploads a record that contains additional dynamically crate and project administrator can verify that additional data was added to the backend database.		
Application layout shall accommodate Android phones	User with Android phone containing Jelly Bean 4.2 and higher can install and run the application		
Application upload shall take less than 45 seconds based on phone's signal strength	User with internet and phone service provider signals shares a record without timeout or delays in less than 45 seconds.		
Application shall upload record's notes and image seamless to user's perspective	User updates a record's information, which already has image Url associated with it. The record is uploaded seamlessly without any further user interaction		
No upload confirmation prior to uploading occurring shall take place.	User clicks on upload button and record gets uploaded		
User shall have ability to sort existing records list by latest record created or by record name.	Under "View And Share," activity user can click on sort button to select a sort criteria. User selects to sort by, "Latest Created," and list is sorted by the latest record created.		

A record is not required to have an image associated with it. In this case a default icon shall be associated with the record	User adds a new record and does not select an image for the data. At, "View And Share," activity the record has an Android Icon associated with the data		
Application buttons shall have good size and width for ease of navigation	Users can easily read and click the buttons in main view and all other activities including Upload, Delete and Edit		
User shall be able to quit the application from main starting screen only	User clicks on "Exit," button and application closes		
Project coordinator shall be able to curate data in backend server	Project coordinator logins to backend system: Google App Engine web interface and reviews data, updates and deletes some records without errors.		
Application shall have some type of user security or authentication mechanism	Project coordinator logins to Google App Engine web interface. Furthermore, the coordinator views data broadcasted has user's phone information related to each record for security purposes.		

6.1.3 Test Results

Based on the Test Plan test cases, the test results matched expectations.

Test Cases results table

Test Case	Expected Result	Test Result	Notes
Mobile application shall allow user to capture text data	User can input text and save data	Passed	
Each record of data has a unique identifier in database	Each record is saved locally in the device and has a primary key	Passed	
Each record of data is saved locally in device	Each record can be viewed in the application's front end and database backend	Passed	
Each record of data has a title	During new data being added, "Name:," is a required text field and user shall input text in this field before being able to save a record	Passed	
Mobile application shall allow user to capture dynamic data	During new data being added user can click "Add Additional Text Fields," button to capture additional data	Passed	
Mobile application shall allow user to associate image with data captured	During new data being added, once the user clicks "Save And Continue," user is prompted to select an image from device's gallery to pick an image. Otherwise a default Android Icon is associated with the record	Passed	
Mobile application shall allow user to edit record of data captured	During, "View And Share Observations," user can click on pencil icon to edit record and image of record	Passed	

User shall only be able to read and not modify data that was captured and shared by other users	During, "View And Share Observations," no pencil icon to edit record is available	Not tested	
Administrator shall be able to add data captured	User logins to Google App Engine interface and adds a record to the database	Passed	
Administrator shall be able to edit data captured by any user	User logins to Google App Engine interface and updates a record to the database	Passed	
Administrator shall be able to delete data captured by any user	User logins to Google App Engine interface and deletes a record to the database	Passed	
Mobile application shall allow user to select a project in mobile application to allow filtering of data based on project selected.	User clicks on, "Settings," button from main activity and is able to select a project	Passed	
Mobile application shall allow user to upload a record of data collected	During, "View And Share Observations," user can click on pencil icon to Cloud Icon record and image of record	Passed	
Project coordinator shall be able to view data in backend server	User logins to Google App Engine interface and views records in the database	Passed	
Application Android services shall be free of charge for usage.	Project administrator can add a new project in the backend database by adding a new database and inserting a record with project name.	Passed	

Application's backend server services shall be free of charge according to service provider's term and conditions.	Project administrator can monitor usage traffic and turn off APIs based on usage to stay within free bandwidth	Passed	
Listview image shall be at most 1,024 Kb of size, so this can easily fit within the listview display size.	User notices that a large image associated with record is copied and a smaller image is created to be associate with the record	Passed	
Database backend shall accept a dynamic increase number of fields of data to upload	User uploads a record that contains additional dynamically crate and project administrator can verify that additional data was added to the backend database.	Passed	
Application layout shall accommodate Android phones	User with Android phone containing Jelly Bean 4.2 and higher can install and run the application	Passed	This application is only for Android SDK level 17 or above
Application upload shall take less than 45 seconds based on phone's signal strength	User with internet and phone service provider signals shares a record without timeout or delays in less than 45 seconds.	Passed	This is relative to the device’s signal’s strength
Application shall upload record's notes and image seamless to user's perspective	User updates a record's information, which already has image Url associated with it. The record is uploaded seamlessly without any further user interaction	Passed	
No upload confirmation prior to uploading occurring shall take place.	User clicks on upload button and record gets uploaded	Passed	

User shall have ability to sort existing records list by latest record created or by record name.	Under "View And Share," activity user can click on sort button to select a sort criteria. User selects to sort by, "Latest Created," and list is sorted by the latest record created.	Passed	
A record is not required to have an image associated with it. In this case a default icon shall be associated with the record	User adds a new record and does not select an image for the data. At, "View And Share," activity the record has an Android Icon associated with the data	Passed	
Application buttons shall have good size and width for ease of navigation	Users can easily read and click the buttons in main view and all other activities including Upload, Delete and Edit	Passed	
User shall be able to quit the application from main starting screen only	User clicks on "Exit," button and application closes	Passes	
Project coordinator shall be able to curate data in backend server	Project coordinator logins to backend system: Google App Engine web interface and reviews data, updates and deletes some records without errors.	Passed	
Application shall have some type of user security or authentication mechanism	Project coordinator logins to Google App Engine web interface. Furthermore, the coordinator views data broadcasted has user's phone information related to each record for security purposes.	Passed	Only assigned user has coordinator privileges

Overall the test results passed the test cases as expected.

6.2 User Interface Evaluation

6.2.1 Focus Group

The User Interface went through various rounds of modification. An analysis of the User Interface was performed throughout the testing of the Android mobile application functionality. The review of the User Interface was done through a Focus Group and various recommendations for better usability were done. Once the updates were completed another round of testing with Focus Group was performed. For this project we performed three formal Focus Groups to further improve the application's usability. A Focus Group was performed instead of the Usability Testing for various reasons. First, the Focus Group was used early on the development of the application. Furthermore, the Focus Group made more sense because the application's audience (Citizen Scientists) can have different types of background in terms of software usability. This approach to use Focus Group allowed for numerous opinions to be gathered in a short period of time. Overall the Focus Group gave a good amount of ideas to develop the Android application for better usability. In addition we were able to determine the features of highest values and improve its usability within three rounds of Focus Group.

7.0 Conclusion And Recommendations

Throughout this project an Android mobile app was built to support Citizen Science projects. An analysis of various technologies was done to determine which one best fits Citizen Science projects' requirements and promotes free usage service. The mobile app for this project required the ability to dynamically capture observations based on project's requirements, so little to no programming is required to get various projects started with just one application. The application has means to share text and images with support of Google App Engine and DropBox APIs. The overall User Interface for the mobile application was accomplished through Focus Group iterations.

7.1 Accomplishments

After a thorough analysis of Cloud Services that provide mobile application connected as COTS framework, it was determine that Google App Engine (GAE) provided the best option to build a Citizen Science project that has free service usage. The Android mobile app provided with GAE was further architected and implemented to support the dynamic data capturing. With the dynamic data capturing configuration this mobile application can support various Citizen Science projects without the need to program. Android mobile application with Cloud Services as backend support is the best option to support a dynamically configurable Citizen Science application because of the low cost of usage of the technology. Cloud Services need to be free or have a free usage service in order for this approach to work.

Some of the challenges encountered while building this mobile app was the ability to maintain the usage of the Cloud Services free of charge. This entailed having to use an additional to GAE, alternate, Cloud Service provider: DropBox. DropBox was used to upload images, and then share the image's url. GAE has APIs to upload images, but it's not free. DropBox' APIs is free and it allows images to be uploaded but does not support a NoSQL API. GAE has the NoSQL APIs with limited bandwidth (only text allowed). Another challenge during this project was the dynamic field data capturing mechanism. When additional fields are added, the database needs to support these additional fields. Once this mechanism was put in place, receiving broadcasted messages inherited the dynamic field mechanism to ensure proper data is displayed to user during, "View And Share," activity of the mobile application.

The architecture of the mobile application was built with a Model View Control approach, which allowed for easy change of functionality of the User Interface. The Focus Groups provided lots of good ideas and instant feedbacks, which required changes to the User Interface and sometimes to the middleware. These changes were easily implemented without any modification required to the backend (database handler and ListView row adapter) of the mobile application. The architecture design of the mobile application was successful as the software maintenance was manageable.

7.2 Future Work

Further work may be done to implement further data gathering with other mechanism available on the mobile device i.e. gyroscope, or gps coordinates. Also, it is needed to find a better set of APIs from various Cloud Service providers that support other functionalities such as Blob. Lastly, incorporating other mobile devices into the GAE backend server can be helpful. It was observed in this paper that IOs has costs imposed by the manufacturer, but experiment it with GAE is an option worth investigating. Windows phone is also another option worthwhile investigating as a method to support Citizen Science projects.

Bibliography:

[1] Google App Engine. (2013, November 19). "*Admin Console*."
[Online]. https://developers.google.com/appengine/docs/adminconsole/ [03/04/2014]

[2] Google App Engine. (2014, March 3). "Overview of App Engine Features."
[Online]. https://developers.google.com/appengine/ [03/04/2014]

[3] Savant, Abhishek Amol. (2013, December 21). "*Configurable Restful Mongo DB Framework for Mobile Applications Implementation Plan*," Dep. Of Computer Science, San Francisco State University, San Francisco, CA, March, 12, 2014.

References:

[4] Google, "*Assembling Your Mobile Solution*," https://cloud.google.com/developers/articles/mobile-application-solutions

[5] Google, 12/17/2013, "*Google App Engine*," https://developers.google.com/appengine/docs/java/endpoints/

[6] Google, "*Google Cloud Platform*," https://cloud.google.com/resources/articles/mobile-backend-starter-api-reference

[7] Chitan Khetiya, 06/01/2013, "*Android Local SQLite Database Example*," http://chintankhetiya.wordpress.com/2013/06/01/sqlite-database-example/

[8] Google, "*Google Cloud Platform*," https://cloud.google.com/

[9] Google, 11/12/2013, "*Google Drive SDK*," https://developers.google.com/drive/quickstart-android

[10] Google, 12/17/2013, "*Google App Engine*," https://developers.google.com/appengine/docs/quotas

[11] Google, 12/10/2013, "*Writing files to the Blobstore using the Files API (Deprecated)*," https://developers.google.com/appengine/docs/python/blobstore/#Python_Writing_files_to_the_Blobstore

[12] Google, "*Cloud Backend API Overview*," https://cloud.google.com/developers/articles/mobile-backend-starter-api-reference

[13] Google, "*Google Cloud Platform*," https://cloud.google.com/products/bigquery/

[14] Google, 10/02/2013, "*Google Cloud Platform*," https://developers.google.com/cloud/samples/mbs/

[15] DropBox,08/12/2014, ”*Datastore API*”, https://www.dropbox.com/developers/datastore

[16] DropBox,07/24/2014, ”*Core API*”, https://www.dropbox.com/developers/core/docs