Dartmouth Campus Guide Report

Tianlong Yun, Xiaohong Qiu, Mengjia Kong

May 29, 2014

1 Introduction

We have made a campus tour guide. Unlike the other existing tour guides, our app is community-based, just as the way wiki does. Each user is able to contribute his interested place into a shared centralized repository, which is maintained by a central server. User has access to all the current available places recommended by others and provide comments if he wants. If a place is not in the repository and other users might be interested in it, user can create a new entry for the place. The new entry includes name and a short text description. User has the option to take a couple of pictures for the place. User also can record an audio which provides a simple voice introduction about the place. Moreover, a ranking ranging from 1 to 5 can be selected for that place. Once user clicks the save button, all the information about the new place will be stored into local and server database concurrently.

Our backend server serves as a centralized agent to manipulate all the places currently available in the repository. Everyone is able to view all the places recommended by others via browser. However, only administrator has the privilege to delete a specific place. Plus we have deployed our server into Google cloud and thus got a domain name. So the user can just only download the android app and install it. Then they can play with it.

2 Architectural Design

2.1 OO design diagram

Our OO diagram of android app is in Figure 1. Our 00 diagram of server is in Figure 2.

2.2 class descriptions

2.2.1 Client

- `ManualInputPlace.java`: The class is used to create a new entry for a place. It shows a couple of buttons to perform various functions. Using these buttons, user can record an audio to provide a voice introduction, take a couple of pictures to represent the place, obtain GPS location on Google map. The taken pictures and recorded audios will be stored into web storage server. The database can just stores their URLs. User can also select a ranking level for the place. Once save button is clicked, all the information will be stored into local and server database.

- `MyPageAdapter.java`: It is subclass of PageAdapter and has two pages. One page is listview of all the places in the local database. If user clicks Sync, then the local database will be cleared first and all the places in server will be populated into the local database and automatically populate listview. The other page is a detailed presentation of a certain place. All the pictures associated with the place will show up and can be swiped horizontally. Then a ranking bar will appear below the pictures. Underneath is a Google map with GPS location indicated by a marker. Below that is voice

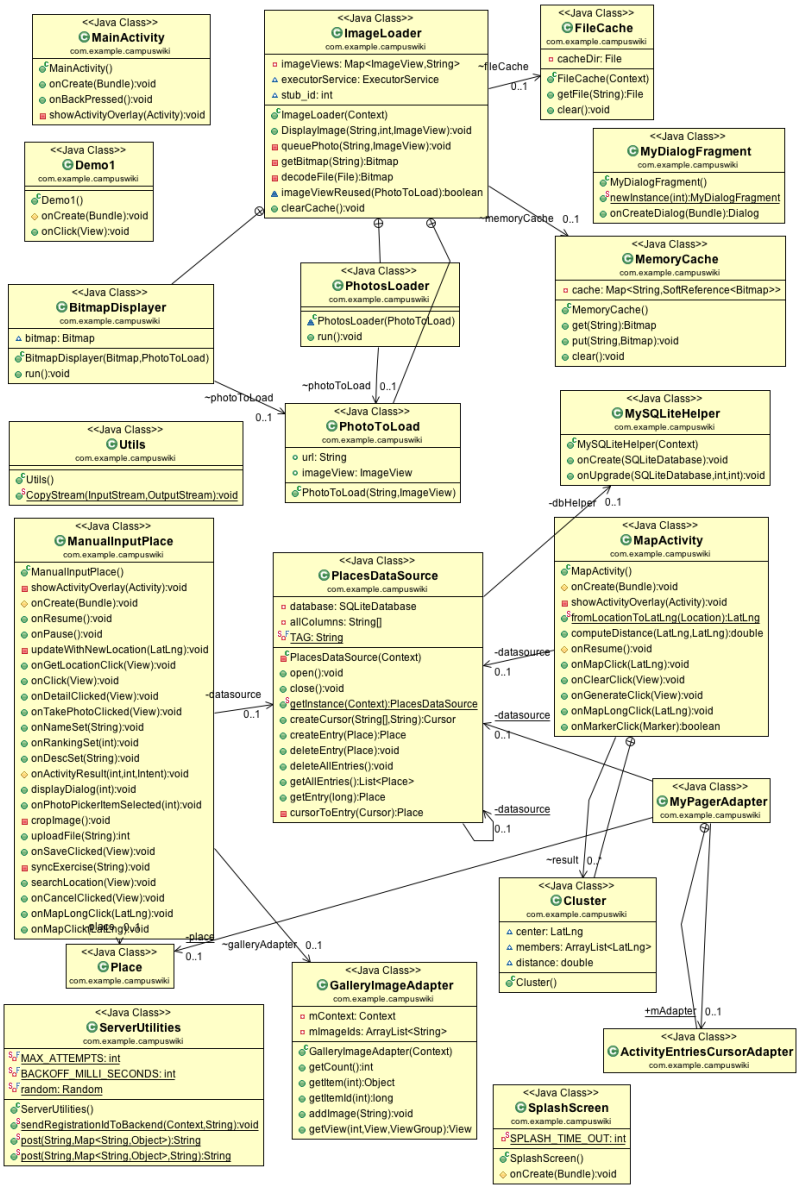

Figure 1: Dartmouth Campus Guide App Design Diagram

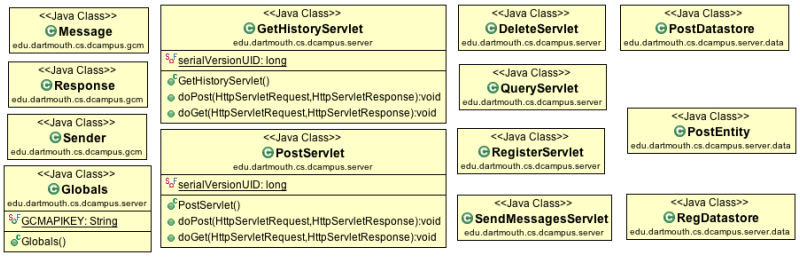

Figure 2: Dartmouth Campus Guide Server Design Diagram

introduction you can play and stop it any time. The bottom is a text description of the place followed by all comments posted by all users.

- `Place.java`: This is a bean holding all the attributes associated with a place including name, latlng, pictures, ranking, voice audio, text description, ranking etc.
- `PlaceDataStore.java`: This class provides a wrapper for SQLite related operations such as createEntry(), deleteEntry(), deleteAll(), query(), queryAll() etc.
- `ServerUtilites.java`: This class provides a wrapper for various operations on java URL connections. The most import one is post(), which is used to send requests and json-fomatted data to server.
- `MyDialogFragment.java`: This class provides centralized helper functions to handle all kinds of dialogs used in the app, such as when taking a picture, when prompting for text or when clicking recording stop.
- `Utils.java MemoryCache.java ImageLoader.java FileCache.java`: These class will download audio and pictures from server, and cache them.
- `SplashScreen.java`: This class provides a thread to handle the fade in and fade out of the splash screen.
- `PlacesDataSource.java MySQLiteHelper.java`: These two class are helper class which provide you with the routines to manipulate the SQLite datasource, including add, delete or query.
- `MapActivity.java`: This class will show the map of Dartmouth, and visual all the pre-existing locations and locations by the users. User could click on the place balloon to view the detail or they can also long click on some point to and a new entry.
- `MainAvtivity.java`: This class will initialize the page adapter.
- `Demo1.java`: The class contains routine to show the transparent activity.
- `GalleryImageAdapter.java`: This class will handle the routine for show the gallery and enlarged view of pictures

2.2.2 Server

There are two parts classes in server: Backend and Frontend. Frontend is about the webpage which will be present in a browser. Backend can also be split into two parts, webpage backend and datastore.

- `QueryServlet.java`: This class queries all data items in datastore. There are two methods "`doPost`"and "`doGet`" in this class with same function.
- `PostServlet.java`: This class is to add a new data to datastore. There are two methods "`doPost`"and "`doGet`" in this class with same function.
- `DeleteServlet.java`: This class is to delete a data from datastore and refresh the webpage. There are two methods "`doPost`"and "`doGet`" in this class with same function.
- `GetHistoryServlet.java`: This class is to query all data items in datastore and send them to device. There are two methods "`doPost`"and "`doGet`" in this class with same function.
- `RegisterServlet.java`: This class is to register a new device to server and store the information of this device in register datastore. There are two methods "`doPost`"and "`doGet`" in this class with same function.

- `PostEntity.java`: This class is about the data structure which is stored in datastore. Only structure method in this class.
- `PostDatastore.java`: This class is about the DatastoreService which provides a datastore to store all location informations in this project. "`add`" method is to add a new data to this datastore. "`query`" is to get all data in datastore in the type of `PostEntity`. "`delete`" is to delete the specific data in the datastore.
- `RegDatastore.java`: This class is about the DatastoreService which provides a datastore to store all information of devices in this project. "`register`" is to add a new device item to this datastore. "`getDevices`" is to get all devices from datastore.
- `main.jsp`: This file is not a class but a jsp file about the webpage, which builds from html, java, css, jQuery.

3 How to run the demo

3.1 Client

1. First Start the Server in RealCloud, or just use ours: http://dartcampuswiki.appspot.com/
2. Change the IP address of the string value at App side to the IP address e.g. http://dartcampuswiki.appspot.com/
3. Run

3.2 Server

We have run the server on the real Cloud. The url is http://dartcampuswiki.appspot.com . The resources are also provided in the svn, which can be import into your eclipse and run in your own computer. The steps of running the project on the real cloud is :

1. Import the existing project in your workspace of eclipse. Maybe there are some bugs according to the environment. According to my experience, the Google App Engine SDK may be missing. Then a specific Google App Engine SDK need to be added.
2. Press run, you can see the server on http://localhost:8888 .
3. To run in the real cloud, we use Google App Engine. Create an application on Google App Engine's Application Overview and set the Application Identifier and Title. Next, deploy the project in eclipse to the app engine. Finally, you can see the server on the real cloud when you open the äpplication identifier.appspot.comïn a browser! The details is exactly like the lecture notes in cs65 course website.

4 Demo

When you start our app on a android phone, users will see splash screen in Figure 3.

When the app downloading succeeds, users will see a introduction of how to use the main view like Figure 4. Touch anywhere then the main view will show up in Figure 5. Press any item, uses can see the details of this item in Figure 6.

Press Add button, users will see a introduction of how to use the add view like Figure 7. Touch anywhere then the add view will show up in Figure 8.

In main view, press Map button, users will see a introduction of how to swipe between views like Figure 9. Touch anywhere then the map view will show up in Figure 10.

Users also can see locations in a website: http://dartcampuswiki.appspot.com/ . The view is in Figure 11.

Figure 3: Splash Screen

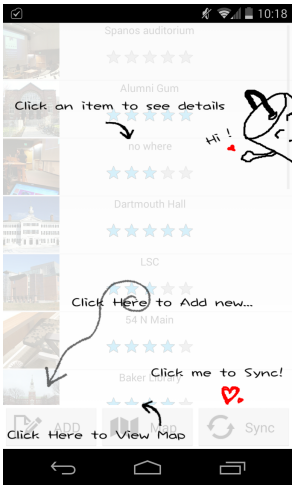

Figure 4: Introduct how to use main view

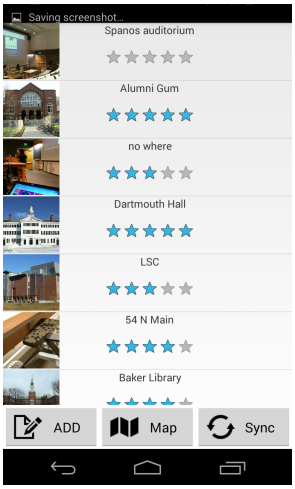

Figure 5: Main View

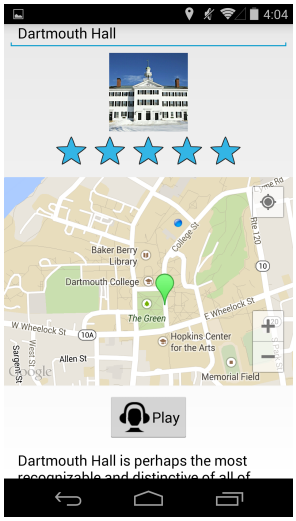

Figure 6: Details of Dartmouth Hall

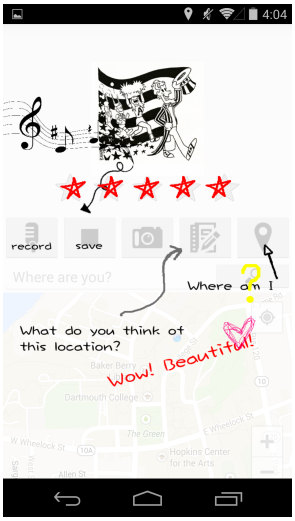

Figure 7: Introduct the add view

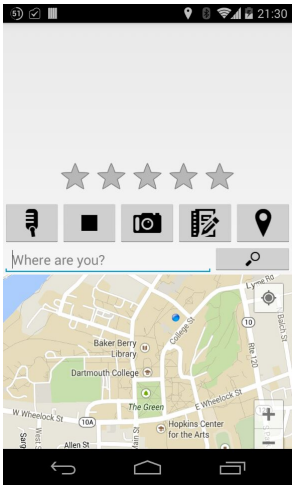

Figure 8: Add View

5 Lessons Learnt

In this project, we have used many functions which we learned from classes: Widgets of UI, Camera, Compressing images, Album, Database, Map, Google app engine, Cloud. Besides, we have some innovative ideas in this project: upload and download images and audios to server.

In the progress, we have run into many problems. Once, we want to use socket to upload files to server. But Google App Engine blocks `IOStream` to forbidden users to access this method. After trying variety of possible methods, we run website on Google App Engine but upload files to another server, from which both app and website get files.

Luckily, we went through all this problems finally. When these difficulties show up, many methods are tried to figure out them. There are some resources help us a lot: Stack Overflow and Android Developer. Furthermore, communicate more with TAs which will give you many feedbacks about the project.

Figure 9: Introduct the swipe function

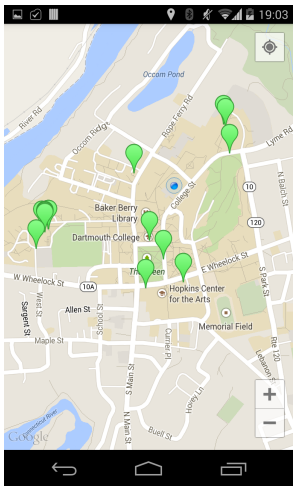

Figure 10: Map View

Figure 11: Dartmouth Campus Guide Website

6 Conclusion

One of the biggest advantages our campus tour guide over others existing ones is that it is driven and contributed by each user. Our app has deeper interaction with user and thus can better reflect and meet user needs. We use GPS and Google map as well as audio to supply a more vivid description of places. On server side, we employ two-level permission labels to manipulate the current places to avoid mal-behaviors or careless operations.

Next step is to integrate video into the app. The video should give a more vivid introduction about the place. Another improvement is if user arrives at a certain place, the corresponding audio and video should pop up once the option has been enabled. We may also consider applying our app to a broader community than campus, such as town or city or even state.