Wheel share

# Introduction

The application is designed to aid the people who are in search of rides and people who want their trip to be shared. We help them save money by encouraging carpooling.

# Team members

Triet Nguyen: UI design and back-end integration

Sugeerth: Validation, Hindi language, testing.

Deneb Chen: Backend server

Duc Hoang: Quality assurance

Activity Screens & Logic Code

# Activity Main/activity\_register

This screen is where users are able to login and register for a new account. An error message will display if incorrect username or password is inputted. The register screen allows users to register for a new account. The input information will be pushed to database. We also have various fields for checking register. We ensure that password is validated with many constraints because the app involves money.

# activity\_home

This is the homepage of the application. Think of this as a Facebook application where the riders/drivers can get their notifications as and when they log in.This page stores the requests sent/received from users. A user does all interactions from this page; such as accept/reject requests, payments, and start/end trips. Requests are pulled from the data store every time a user is in the homepage.

# activity\_search

This screen allows the user to post and search for a ride. A user is allowed to enter destination, source, fare, date, number of seats, and deciding whether they are looking for a driver or passenger. A popup window will also appear if they user decides to post. Within this popup window, the user can enter a description (a personal message to the driver/rider) about the trip.

# activity\_result

This screen shows all the results from searching. Each result is displayed as an item in a List View. Users can sort the information by date, fare, and number of seats. This screen takes information from the activity\_search screen and queries the datastore and pulls the result. The result information are stored in an arraylist which gets displayed by an Adapter. Also, if no results are returned, a post field will appear similar to the one in activity\_search where users can post their search query.

# activity\_confirm

This screen displays all the search/post information. When a result is clicked on, this screen will show user information, trip information, and a description. We used this screen not only for requests but also to allow users to do payment and start/end trip in the homepage.

# AdInfo and UserInfo

These two classes store information that is pulled from the data base. We used this class for the result and home screen. We have a list of AdInfo which contains an instance of UserInfo for the poster's information. The list of AdInfo is the internal representation of our adapter. In other words, you are essentially seeing the list of AdInfo within activity\_result and activity\_home.

# XFormTask

These classes are used to handle datastore interaction. They all extends Asynctask which allows the application to run network activities in the background thread. We make use of HTTP post/get to transfer/pull data from the datastore. Once the appropriate data is found, they are sent back to the screens that requires them.

# XListAdapter

These adapter classes handles the display of the List View. A List of information such as AdInfo and UserInfo is sent to this class where they will use the appropriate data to display the list you see on activity\_result and activity\_home.

# SortByXComparator

These classes are used to sort information on the activity\_result screen. They all implement Comparator which contains the logic for sorting. Comparator is a fast and efficient way to sort Collections. In our case, we used this to sort our ArrayList.

Concurrency Issues

We did not handle the case were the same user logs in on multiple devices. The user could potentially make multiple requests to the same post which would decrement the number of seats twice (if post is driver) wasting a seat for the driver.

Future changes

**Hard Changes:**

* The implementation of GPS in the system would really benefit the riders and the drivers. Making the driver view the specific location of the rider would ease the logistics. It would see that the journey is recorded with the driver and rider so the payment is more reliable.
* Improve the existing credibility of the system by introducing the photos for each of the profile. A more descriptive profile I would suggest. Social network integration is a great way to implement this. We would also get the friend system which allows users to car pool with friends easier. Also when making a post we can automatically create an event in Facebook and make the notification.

**Easy changes:**

* The riders could send a virtual gift or a tip to the driver. This can be done by including a button which would make enable the riders send a notification to the driver by GCM (Google cloud messaging).
* We can make a better UI as android is unknown for its UI. We are currently thinking of Facebook sliding menu to make the application more presentable.

**Highly desirable:**

* The integration with Facebook is something that we think could be well appreciated with the riders and drivers. Based on the preferences of the all the drivers and riders we could make the app more fascinating.
* Making the app use the Google voice translation to input the source, destination and date could be very useful to the people who type slowly.

Design Flaws

* The Design flaw would be the credibility of the system. Any Tom, Dick and Harry could use the application and post a ride and search and join without even paying for the ride. This would affect the commuting time of the all the people involved.
* There could be concurrency issues with the payment. The payment being made with the current app is done with the escrow account, if there is a way by which we could integrate the payment module, it will be more durable. For example, if a ride has successfully been posted, then the ride will remain posted even if the system crashes.