EZRide

Product Backlog

Team 9

Qing Wei
Yinchen Yang
Spencer Smith
Guanqun Mao
Christopher Williams

1. Problem Statement

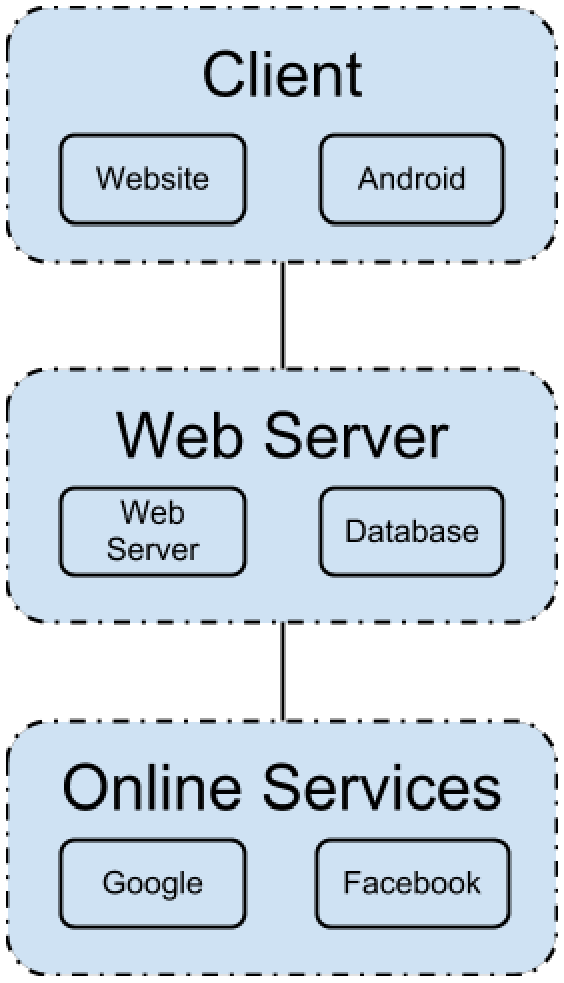
Today, organizing rides for a large group of friends can be tedious and frustrating. We would like to make the process much easier by making all relevant data available to each member of the group in a simple and easy to use set of applications.

2. Background Information

Carpooling can be a good way to save money, and lessen your impact on the environment, but it can also be extremely difficult to organize rides within a larger group. There exists an opportunity to develop a quick and easy method to find others interested in or already traveling to your target location. Scheduling apps such as Noodle Poll can quickly give an easy time schedule polling. New scheduling, mapping and location based technologies allow for a multitude of data to be gathered in order to help this process along.

3. Environment and System Models

The full system will be composed of 3 layers, 2 of which will be our responsibility to develop. The first layer, which is the layer that we will not be developing, is the third party layer, which consists of services such as Google+ or Facebook. This will be used to provide user data, such as the user's calendar. The second layer is the web server and the database. The web server itself should be constructed as a MVC framework that includes model, controller, and view. Models grab the data from the database or process the data. Controllers connect the models and views, and view here is simply our clients' interface. This will allow us to offload as much processing as possible from the client and will act as a central data hub. While temporary data may be cached in the client, the web server will be responsible for maintaining all of the relevant data about the members of a group and provide it to the client as requested. The final layer is the client application, which, when necessary, will query the web server for data in response to the user's input. As for possible future additions, when the client is running on the background, the client would send html requests to the web server to decide if a push notification is needed and ajax requests of the client in the first layer of system models.



4. Functional Requirements

4.1 Interaction between client and web server

- Client can log into online services via the web server.
- Client can send a query to the server and receive a response and display the relevant data.

4.2 Interaction between user and client

- User can enter login details of several services (Google, Facebook, plain email and password, etc.) in order to login.
- User can edit profile and settings.
- User can create/join a group.
- User can tell client where to find schedule (from online services and/or manual entry).

- User can have client analyze schedules and match user with other users in the group with similar schedules.
- User can find others using interactive map.

4.3 Interaction between web server and online services

- Web server will query needed data from the online services when necessary.

4.4 Interaction between web server and database

- Web server will utilize the database to store info about users.
    - Database will store things such as login keys, schedule details, account settings, profile settings, etc.
- Web server will utilize the database to store info about groups.
    - Database will store things such as group members, group settings, etc.
- Web server will be able to query the database whenever necessary.

5. Non-Functional Requirements

5.1 Performance

The system as a whole should be able to perform well and attempt to mask slowdowns. For instance, if say the internet speeds between the server and the client become slow, the client should not become unresponsive. It should still be interactive while data is synced asynchronously. As much of the processing load will be given to the web server as possible, since we do not know the exact hardware of the client. This will enable us to help keep performance as consistent as possible across clients.

5.2 Security

Securing user data will be a primary concern. No personal data (such as login information) will be sent as plaintext over the internet. Also, we will ask the user to give permission (which will be persistent to enhance usability) before retrieving data.

5.3 Reliability

Reliability will be a main concern, mainly because if any portion of the system fails, the system as a whole becomes next to useless. While performance is a main concern as well, if the system is glitchy and unreliable, performance does not matter. We will attempt to balance these two requirements in a way that will maintain a reliable system while achieving adequate performance.

5.4 Usability

We would like to provide a web app and an Android app. Both of them are designed to be user-friendly. Both will maintain the basic standards of their respective platform. For instance, the Android app will conform to the modern Google app development guidelines. We will conduct hallway testing and ask people without any programming experience to use our application, and evaluate their feedback in order to improve usability.

6. Use Cases

Case: User wants to create or join a group

1. The user moves to the create/join page.
2. In order to create a group the user decides to name the group and create privacy settings such as password, and public/private.
3. The user next sends out invitations based off Google Contacts, text, or e-mail.
4. In order to join groups, the user may search for groups in the database, or enter the information from their invitation.
5. Then, the user must enter security information such as a password needed to join.

Case: User wants to input his/her schedule into the group

1. The user enters his/her Google Calendar information to their profile or manually enters their schedule into the app.
2. Then in the settings for each group the user is involved in, the user may choose to allow members of the group to view his or her schedule.

Case: User wishes to retrieve the other members who have similar schedules

1. In the group page, the user selects the “Matching” button.
2. The application then displays other users in the group who have identical leaving or arriving times to the user, or the next closest user by time.
3. From this display the user may choose to call, text, or email the users that have been matched to his or her schedule.

Case: User wants to see the current location and status of other members

1. In the group page, the user selects the “Map” button.

2. A google maps generated map is displayed with the location of the other active members of the group.

Case: Users want to set their status

1. In the group page, the user selects the “Settings” button.
2. Here the user may select to become active or inactive, in which active members location is displayed on the map, and inactives are not.

Case: User can chat with other people in the group

1. In the group page, the user selects the “Members” button.
2. This displays a list of the total members of the group.
3. The user may choose to call, text, email, or use the in app chat with the users.

Case: User can post a bulletin message to the group.

1. In the group page, the user selects the “Board” button.
2. Here the user can post a message that the entire group can view on the bulletin board page.