EZRide

Design Document

Team 9

Qing Wei
Yinchen Yang
Spencer Smith
Guanqun Mao
Christopher Williams

Purpose

The purpose of this design document is to provide key details of EZRide. The document will include: a summary of functional requirements, general priorities, a design outline, design issues, design details and GUI Mockups.

The goal of EZRide is to allow members of groups to easily set up rides and/or meetings by providing an easy-to-use scheduler as well as a live map.

Functional Requirements Summary

- As a user, I can enter login details of several services (Google, Facebook, plain email and password, etc.) in order to login.
- As a user, I can edit my profile and settings.
- As a user, I can create/join a group.
- As a user, I can input my schedule into the app (from online services and/or manual entry).
- As a user, I can have the app analyze schedules and match me with other users in the group with similar schedules.
- As a user, I can find others using a live, interactive map.
- As a user, I can arrange a repeated event within a group.

General Priorities

Performance

In order to help with performance, a client-server architecture was chosen, as well as a thin-client. This allows the front end application to perform better by offloading much of the processing to the server. This will also save storage and battery life on the Android client.

Security

Securing user data will be a primary concern. No personal data (such as login information) will be sent in an easy to access format over the internet. Also, we will ask the user to give permission (which will be persistent to enhance usability) before retrieving data. Any secure exchanges will be made using the SHA-1 protocol.

Reliability

The client-server architecture definitely helps with reliability since it does the most to help with dividing our focus. It allows us to focus on making the client as bug free as possible, while only having to worry about the hardware on the server side, which is the only hardware we can control. The server should be able to at least be up for 99% of the time per three days.

Response Time

For all parts of the system that we can control, optimizations will attempt to be made in order to cut down on the response time. For instance, once a query for data is received by the server, it should be able to, internally, find the requested data and send it back as quickly as possible. Since we cannot control a vast majority of the connection between the client and server, these optimizations will be made in order to help mitigate the effect of any internet slowdowns the user may experience.

Battery Life

For the android client, battery life will be a main concern. We will develop the app in such a way that will avoid unnecessary processing in order to save as much battery as possible.

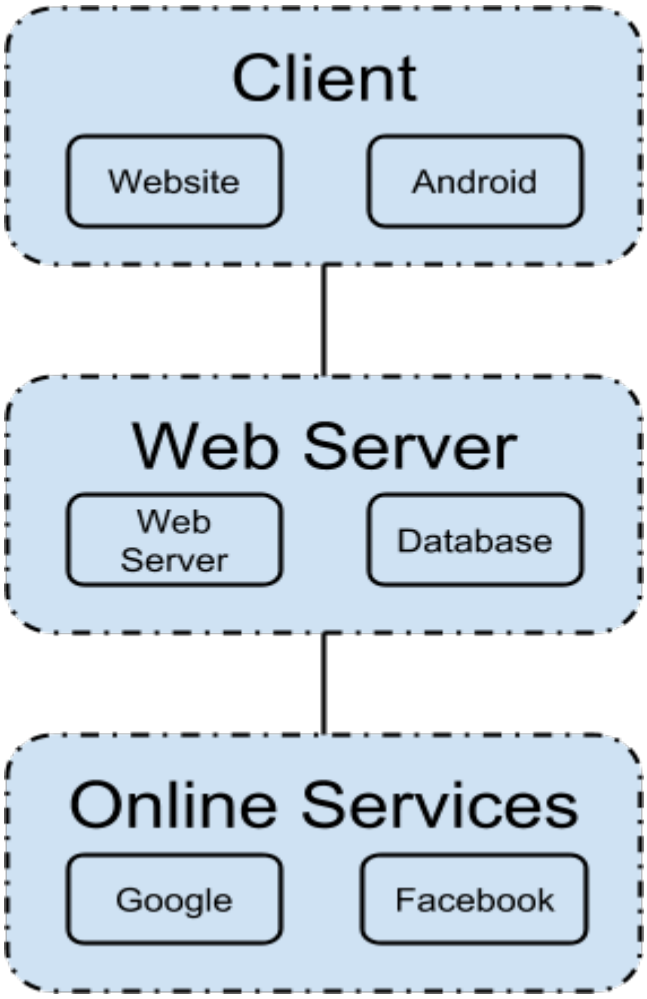
Usability

The user interface will be laid out in such a way that should be intuitive enough for a new user to figure out pretty quickly, but not be too simple so that a power user can do what they need to do. The android app will conform to the modern Google app development guidelines. This will allow easy access to several menus and functions in a good looking and functional way. The web

app will be laid out in a simple and elegant manner that should be easy to use, while still providing much utility.

Design Outline

The full system will be composed of 3 layers, 2 of which will be our responsibility to develop. The following diagram will help to illustrate these layers.



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, as well as profile details, like the user's name, picture, etc.

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/semi-permanent data may be cached in the

client (things like the user's profile data), the web server will be responsible for maintaining all of the relevant data about the users and groups and provide it to the client as requested. The database will be used to store a variety of data. The most obvious things are information about the users (an ID unique to each user, their name, profile picture, etc.) and information about the groups (an ID unique to each group, the name of the group, the users in that group, etc.). There are more specific things the database will store. For instance, if we are able to implement in-group events (events/meetings that aren't part of anyone's personal calendar, but are a part of the group's calendar), then the database will store the information about the event in the portion of the database that stores data about the groups.

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.

Design Issues

1) Software Architecture

Option 1: Client-Server
Option 2: N-Tier
Option 3: Service-Oriented

Decision: Client-Server
Reason: The N-Tier could work. But our system has two major components which are client and web server. Taken into consideration that server does most of the computations, the client will only need to present information to users and send data to the server. Client is also critical because it is the main interface for the users to communicate with server and other users.

2) Programming Language

Option 1: Java and PHP
Option 2: Objective-C and PHP
Option 3: Java and Python

Decision: Java and PHP
Reason:

1. First of all, all of us are quite familiar with Java. Android Software Developer Toolkit provided by Google in Eclipse is widely considered as the best way to implement an Android application.
2. PHP is a succinct and powerful language that is designed for web developing. It is also easy to find a free PHP server.

3) Client Design

Option 1: Thin Client
Option 2: Fat Client
Option 3: Balanced Client

Decision: Thin Client
Reason: We chose thin client because we decided to store the data on the database so that it is more effective for server to compute and send out the data to multiple users efficiently. And since our app always needs internet connection, the thin client will benefit since it only needs to send requests to exchange data and it is helpful for the server maintenance.

4) Client Application

Option 1: Android and Web application

Option 2: Android and Desktop application

Decision: Web application and Android

Reason: Considering the adaptability in mobile applications and the limited time we have for development, we agreed to build a web application rather than desktop application. Another major advantage of web application is that it's platform independent and can be accessed from anywhere. Also, it makes future support and maintenance much easier.

5) Backend Software

Option 1: Apache

Option 2: Tomcat

Option 3: ASP.NET

Decision: Apache

Reason: We chose Apache because we decided to use PHP as the server language. Apache is reliable, powerful and secure server software for our application. In addition, Tomcat and ASP.NET are for the java and C# as the backend.

6) Database System

Option 1: MySQL

Option 2: MongoDB

Option 3: Oracle

Decision: MySQL

Reason: MySQL is an Open Source and easy to use SQL database. Plus, it also works great with PHP server. It is reliable, powerful, scalable and secure. Since MySQL is a small size database, it queries the data fast and it is an ideal data manage system for our web application.

7) Version Control

Option 1: GitHub

Option 2: Bitbucket

Option 3: SVN

Option 4: Dropbox

Decision: Github

Reason: We made this decision based on the fact that all teammates are all familiar with GitHub and it's a distributed version control system. Though Bitbucket is similar to Github, we choose GitHub because we all have GitHub accounts. Centralized version control systems such as TortoiseSVN are easy to use but hard to manage a server and backups. Dropbox can synchronize files but it is not designed for controlling the codes. GitHub can help us merge codes, synchronize our codes and trace the changes in an efficient manner, therefore our decision is GitHub.

8) Signin Methods

Option 1: Google+, facebook, and EZRide login

Option 2: Only EZRide login

Decision: Google+, facebook and EZRide login

Reason: We decided that not only using our service of creating an account, but also login by using the common social app login so that it is convenient for people to sign in. In addition, we need to access people's google calendar information.

9) Data Acquisition

Option 1: API and users input their own data

Option 2: Only API

Decision: API and users input their own data

Reason: Using API(like google account) to login is only a way that simplify user's way to upload their calendars. Our application should be available to those who are not willing share their calendars from other account.

10) Users Location

Option 1: Google Maps

Option 2: Bing Maps

Decision: Google Maps

Reason: On Android phone, our first choice can only be Google Maps. Since we are using Google Map on the phone, using the same map on website can save us much time. What's more, Google Map is the most frequently used online map all over the world.

11) Cryptography

Option 1: Send encrypted data between web server and client application

Option 2: Encrypt data in server and client.

Decision: Send encrypted data between web server and client application

Reason: Data transferred between server and client contains important user information. In order to secure users' privacy, we decided to send encrypted data between server and client. Users' passwords are all encrypted before saved into database.

12) Cryptography method

Option 1: SHA-1

Option 2: RSA

Option 3: MD5

Decision: SHA-1
Reason: SHA-1 is a cryptographic hash function designed by the United States National Security Agency and published by the United States NIST as a U.S. Federal Information Processing Standard. RSA is an asymmetric encryption algorithm, so it can be decrypted (with a private key). CMU Software Engineering Institute now says that MD5 "should be considered cryptographically broken and unsuitable for further use". So from above, we made the decision to use SHA-1.

13) Calendar Display

Option 1: Display week/month calendar
Option 2: Display a list of availability
Decision: Display week/month calendar
Reason: Displaying week/month calendar gives users a more straightforward view of the schedule. Although displaying a list may provide more detail information, we could make the calendar clickable so users can click on each event to check the detail.

14) Offline Mode

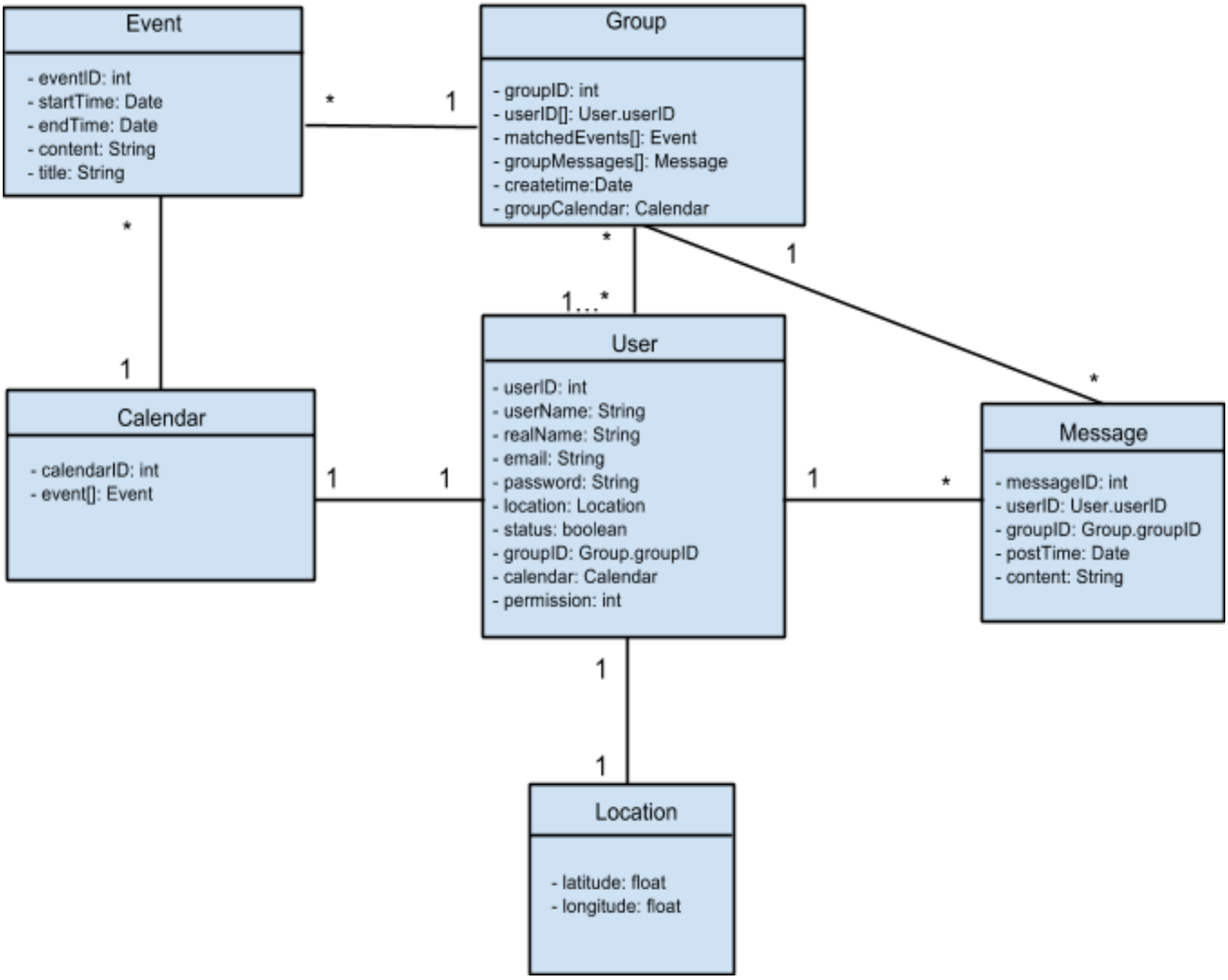
Option 1: Allow Android users to view part of the information when offline
Option 2: Users don't have access to their information when Internet is not available
Decision: Allow user to view part of the information when Internet is not available
Reason: We agreed on Option 2 because we realized that it would be beneficial for user to look up some information such as scheduled time and location for a certain event. However, the functionality will be limited to Android users since certain information could be stored in the client side, and Web users must have an internet connection.

15) Offline Mode Settings

Option 1: Offline views are always available for Android users
Option 2: Users are able to disable offline mode
Decision: Users are able to disable offline mode
Reason: Based on the previous Issue (Offline Mode), some user information is available even without the Internet connection. We realized that some users might want to keep their current location and contact information confidential. Therefore, we allow user to disable offline mode when they don't want any possible reveal of privacy.

Design Details

Class Diagram



Description of the models

User

Entity which represents the user him/herself. The model interacts with the server. The user can have the user_id for identity and profile data like userName, realName, email, and password. The location is the geographical information of the user which will be showed on the client map. The status indicates whether or not the current user needs a ride. When the client create/join the group, the serve will assign a new/existing group id to the current user. The calendar which includes the events information will be access or modify either through API or the database.

Event
The event is responsive for the events within each user's calendar. The entity will store the event ID, start time, end time, title and content of a basic event. The calendar will query the events array for more functionality.

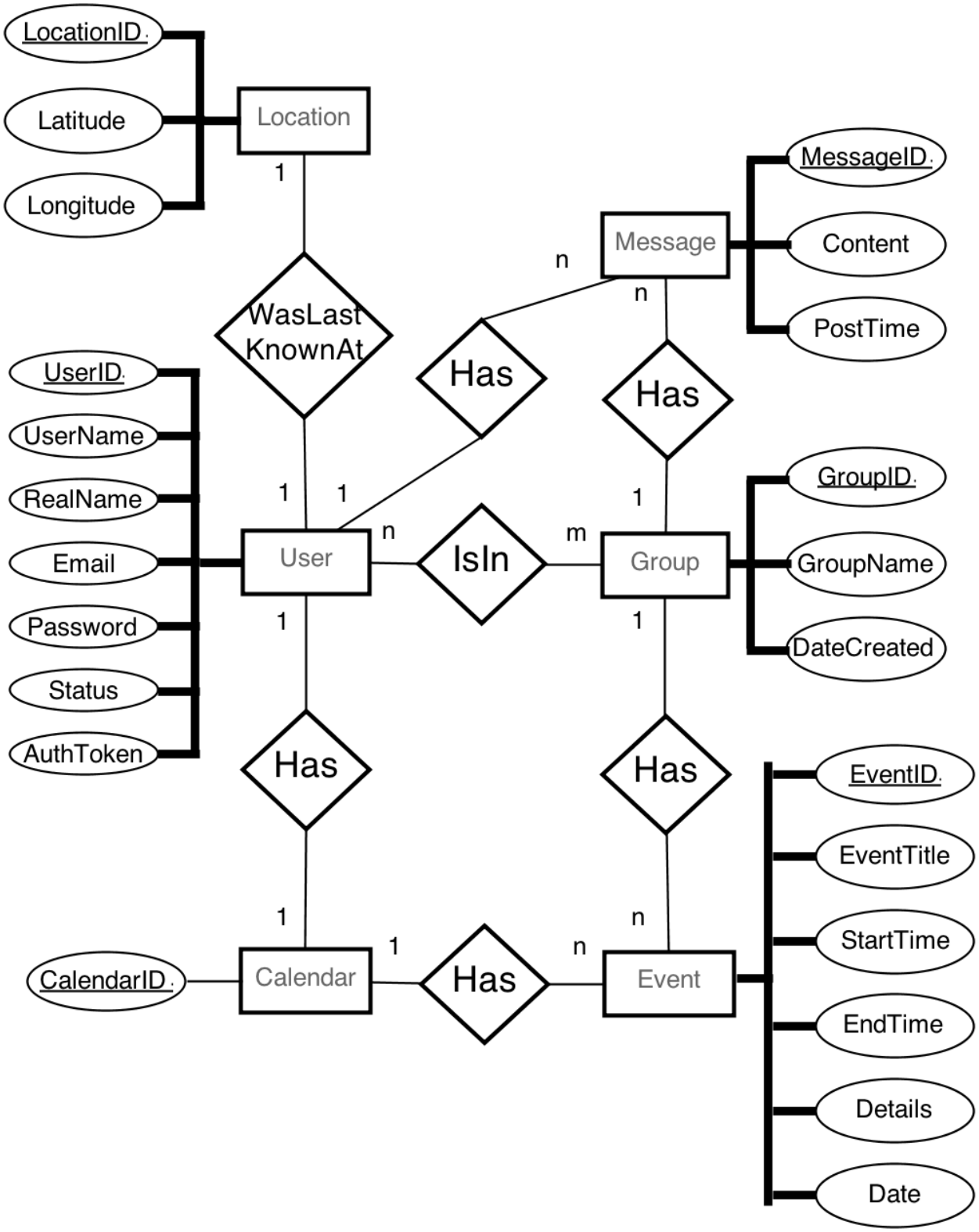
Calendar
The entity will hold the whole events list for the corresponding user and be stored on the database for further reference and computation based on the Server call.

Group
A group represents a group of people who are willing to set up an appointment or share rides. Each group object contains a unique ID, namely groupID, which will be used to differentiate the groups. The variable userID is an array that contains all the members within the group. The List matched. Events contains all the time slot that match every members' schedule. In addition, plan to construct a chatting scenario in which people within the same group could negotiate on the event. Group messages stores all the messages in the chatroom, and it is an object of class Message, which will be discussed in the following context. The client will make requests to the Server that it computes the shared time block based on a query of users from the group.

Message
The message object represents the discussion within the group. The variable messageID is used to keep track of the message and groupID classifies the message to a certain group. In addition, each message records the user who posted the message, and the time it is posted. We use String to store the message content.

Location
We created a class Location to represent the coordinator system that will be used to save geographical information. It contains the latitude and longitude information of the current user.

E-R Diagram


LocationID
Latitude
Longitude
Location
1
WasLast KnownAt
1
1
UserID
UserName
RealName
Email
Password
Status
AuthToken
User
n
IsIn
m
Group
n
Has
n
Message
MessageID
Content
PostTime
Has
1
Has
1
GroupID
GroupName
DateCreated
1
Has
1
1
CalendarID
Calendar
1
Has
n
Event
n
EventID
EventTitle
StartTime
EndTime
Details
Date

User Entity
The user entity is responsible for storing all relevant data pertaining to the user. Each user entity in the database will have an unique ID, be able to store an encrypted version of the user's password, as well as whatever authentication objects need to be stored (authorization tokens for the social networking logins).

Group Entity
The group entity is responsible for storing the necessary information about each group that exists. The necessary information is not much, just an unique ID, the name of the group, and its date and time of creation.

Event Entity
The event entity is responsible for storing every single calendar event that exists within all groups. Each event has a unique ID, it's title, the start and end times, the date, and any details the creator would want to include about the event.

Calendar Entity
The responsibility of the calendar entity is very simple. All it does is keep track of every event that belongs to a specific user. Essentially, each calendar entity represents the personal calendar of the user.

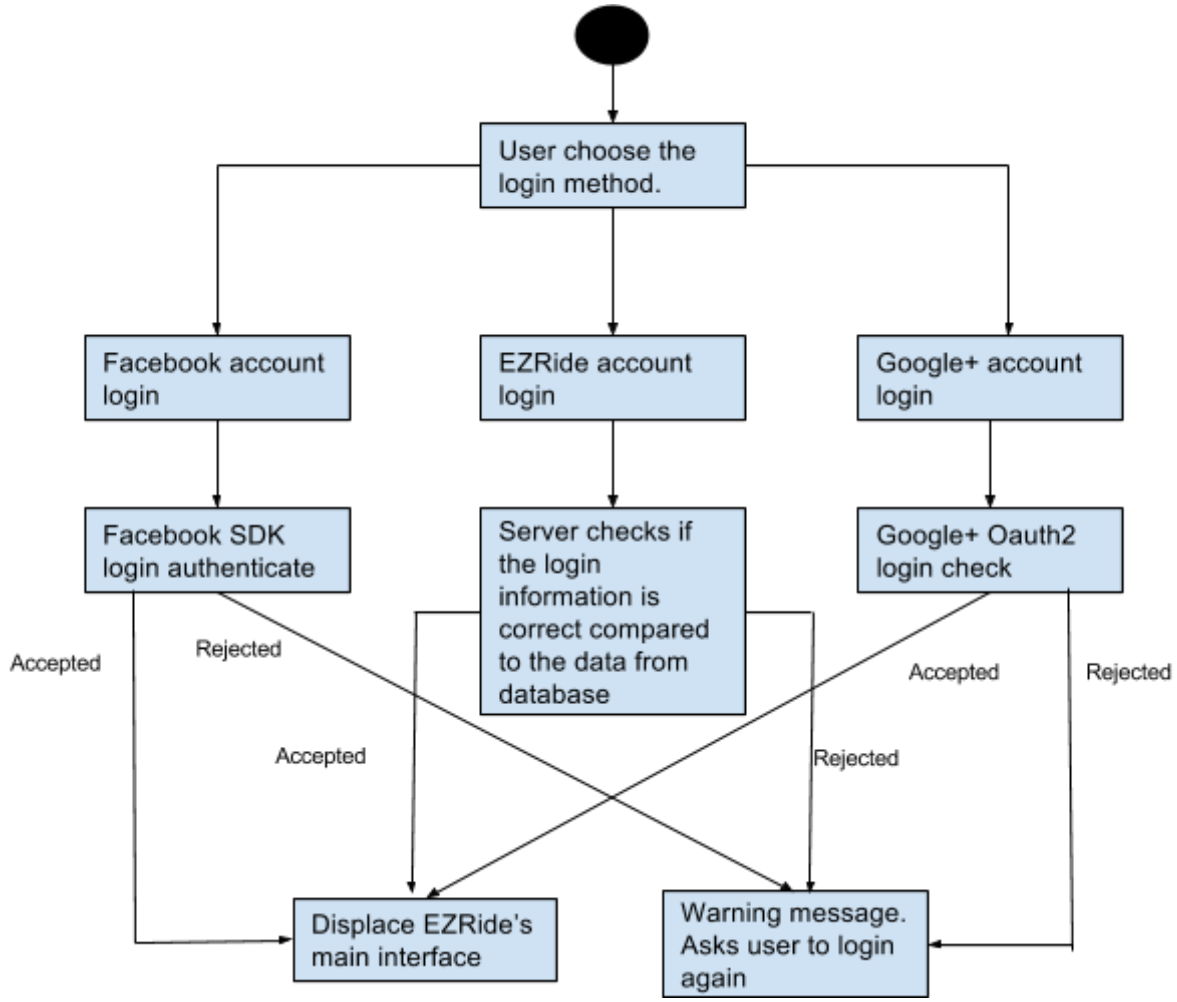
Message Entity
Each message entity is responsible for containing an individual message, as well as any data needed in order to link the messages to each respective group. A unique ID, the content of the message, and its timestamp is all that is needed.

Location Entity
The location entity is probably the most simple of all. It's only responsibility is to keep the last known location of the user in the database.

Relations
The user entity has a many-to-many relationship with the group entity. This means that many users can belong to one group, and a user can belong to many groups. Each user entity has a one-to-one relation with the calendar and location entities, meaning that each location and calendar entity can only be related to one user. Both the user and the group entities have one to many relationships with the message entity, meaning that each user and/or group can have many messages, but each message only belongs to one user and one group at a time. The calendar and group entities have one-to-many relations with the event entity, meaning that each calendar and/or group can have many events, but each event only belongs to one group and calendar at a time.

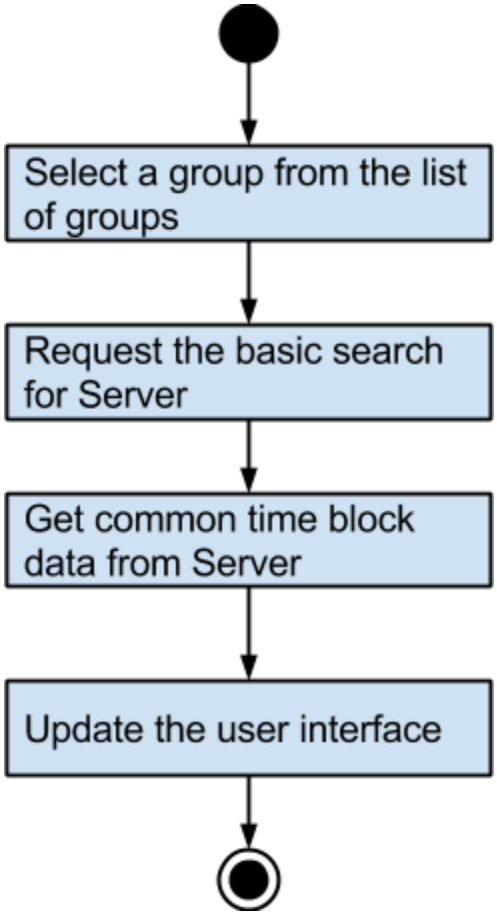
State Diagrams

User wishes to login



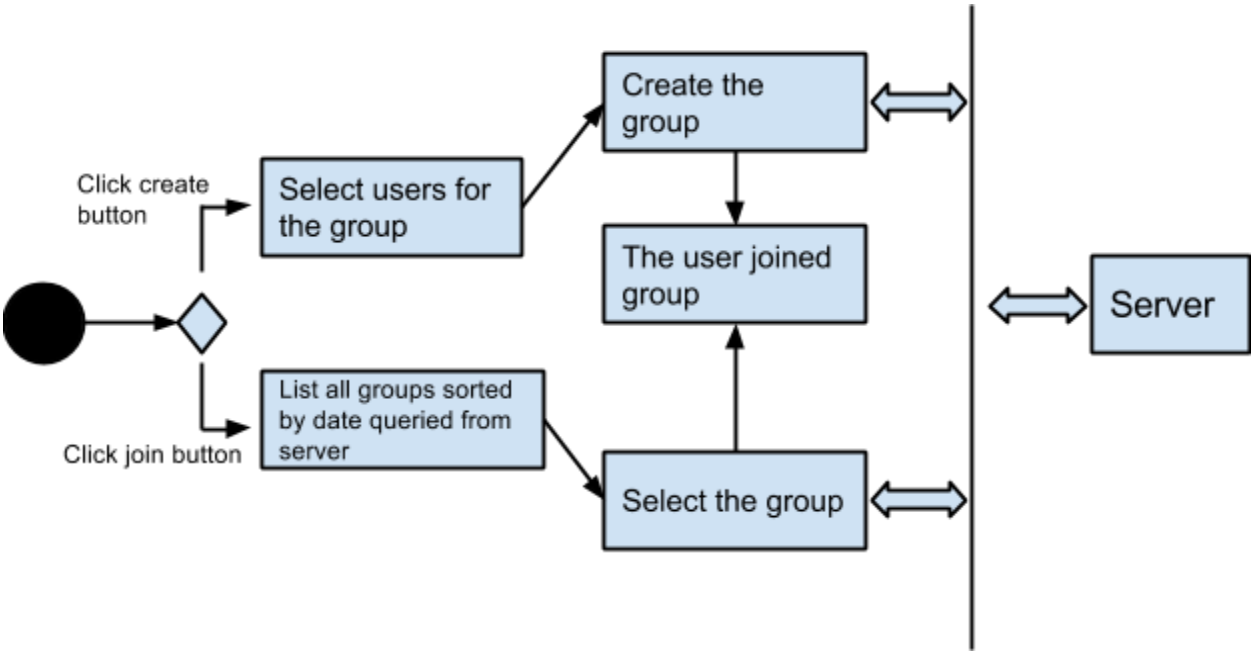
At the beginning, user will be to required to use one the way(Facebook account, Google+ account or EZRide Account) to login. For the users who used EZRide account, our server will in charge of the verification process. For the others, the login request would be sent to Facebook SDK or Google+ Oauth2. If the request is accepted, EZRide's main interface will be displayed. If the request is rejected, a warning message will be thrown out and user will be required to login again.

User wishes to receive matched calendar



This is the general function of requesting matched time block. Here, the user will first select the desired group from client and the server will receive the request from the client and the server will gather the information for computing from database. After the service done, it will send back the data to client to display.

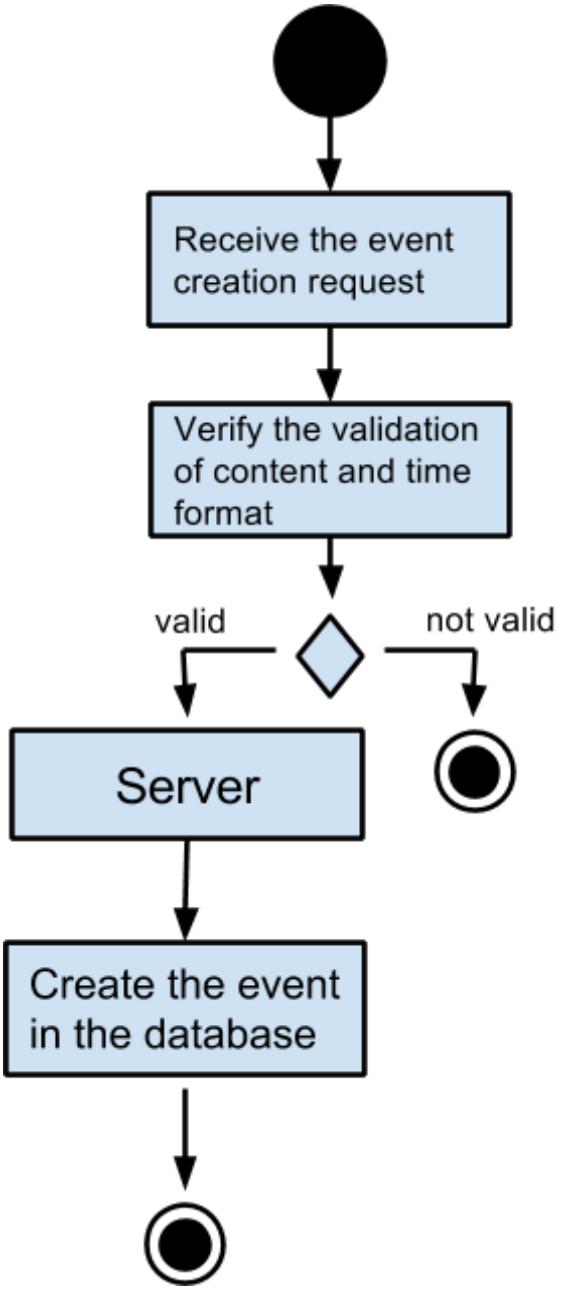
User wishes to create/join a group



If the user clicked create button, the app will require user to select users for the group. User can search for users by user ID or email address. After group members are selected, user will could confirm this creation and the data will sent to the server and save to database.
If the user clicked join button, all the current groups are displayed in list sorted by date and relevance. User can choose any group they want to join and the request will be sent to server. After the server accepted the request, user will get the a confirmation of joining the group successfully.

Activity Diagrams

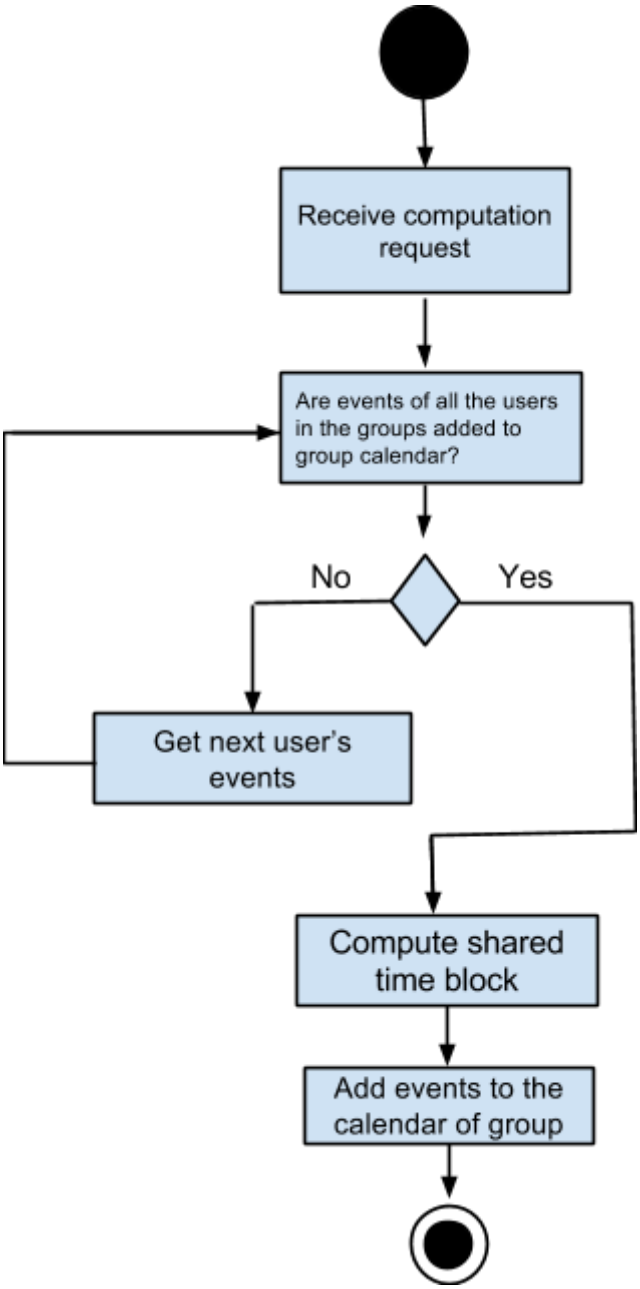
User wish to input a new event to calendar



This is the general creation event service. The frontend will check the validation of format for the event's attributes. If the validation fails, the client will send back the error message and ask user to input again and if the validation success, the client will send a creation event request to the backend. Once the server receives the request, it will query the database to create the event in

the table and match all the attributes to the corresponding entries. And once the server is done for updating, it will send back the data to GUI for display.

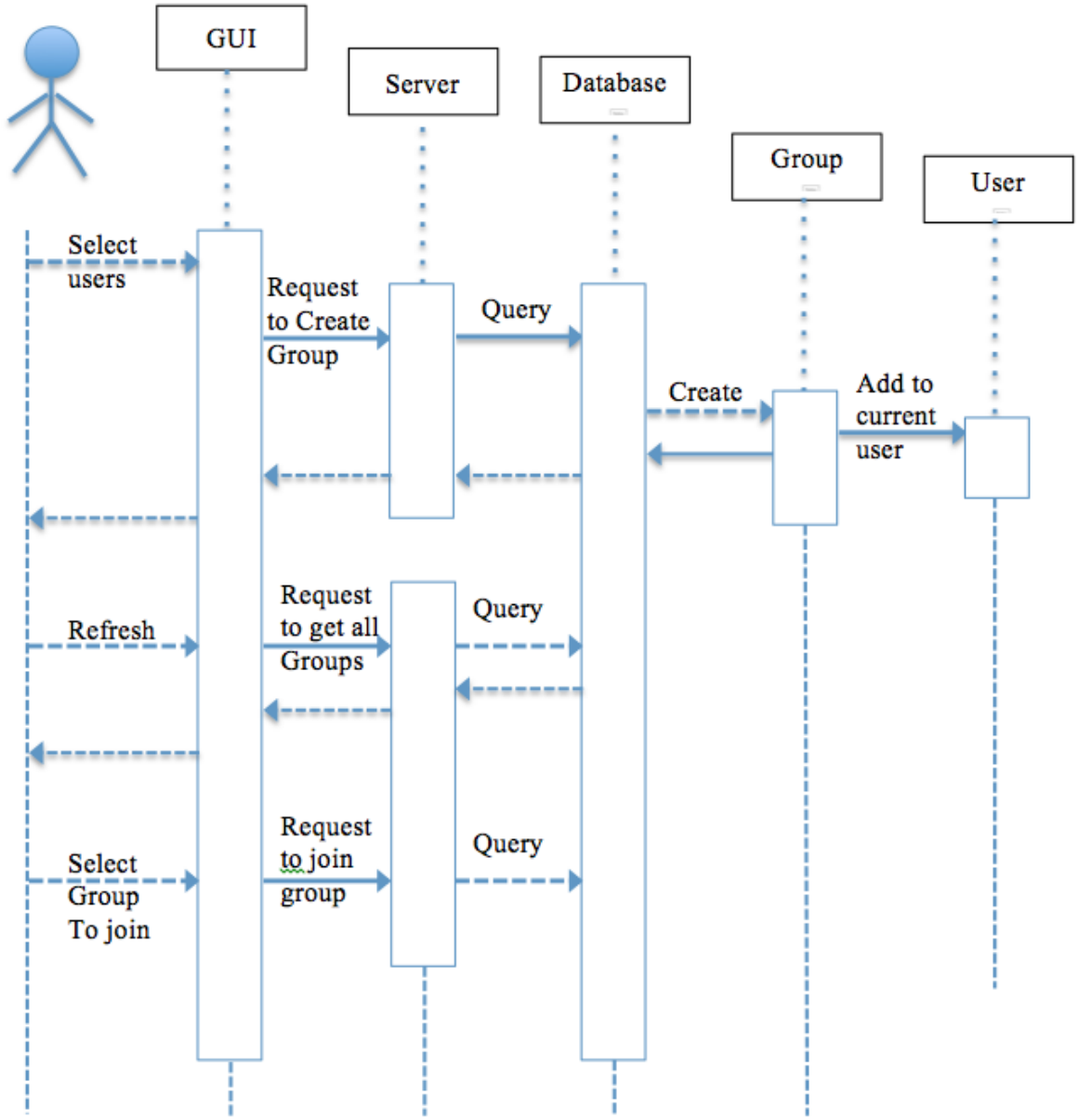
User wishes to have the matched calendar



After received computation request, server will grab the calendars of all the users in the group. After all the users' calendars are saved into group calendar, server will compute the shared time blocks and save it in every user's calendar.

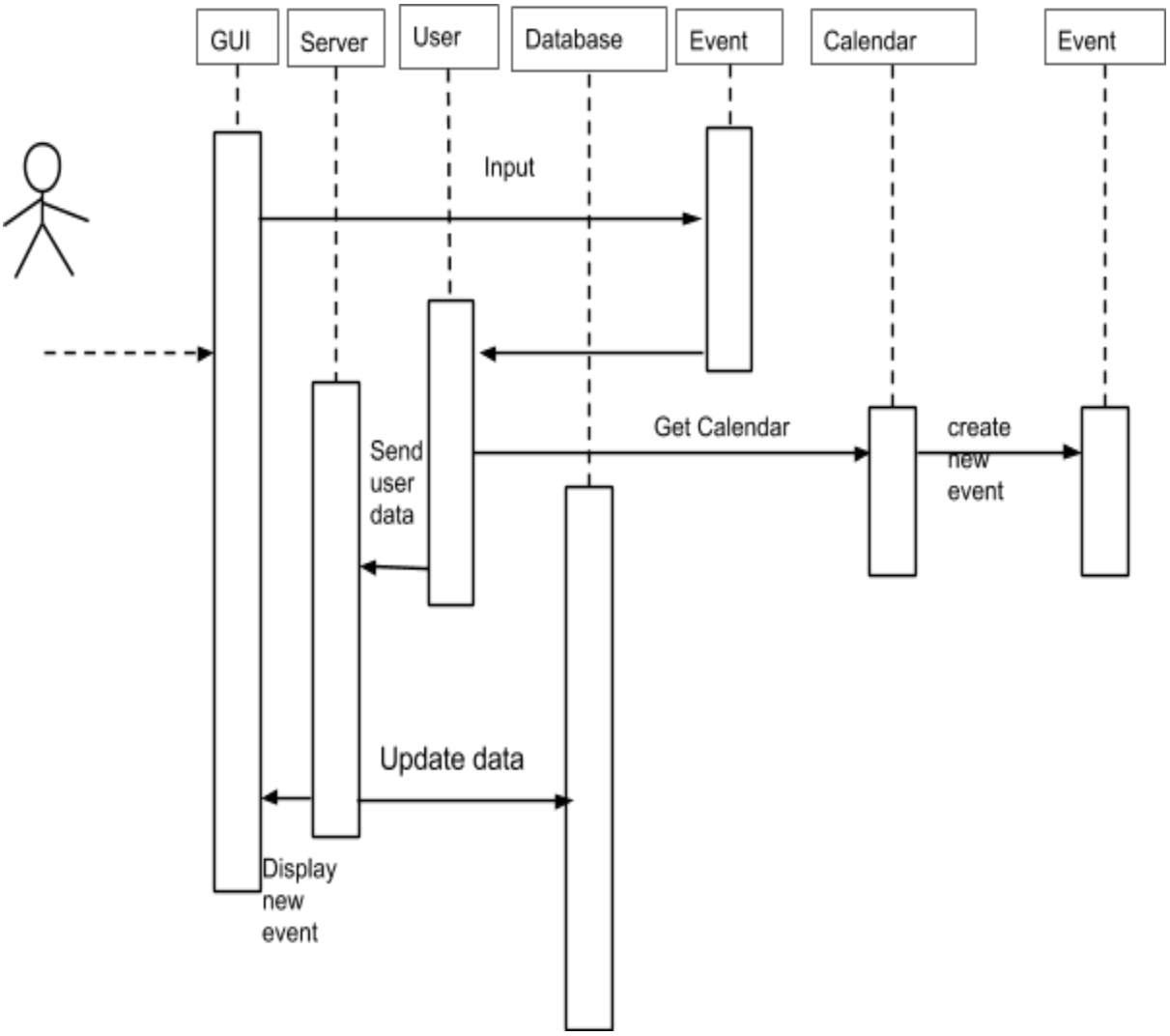
Sequence Diagrams

User query creation



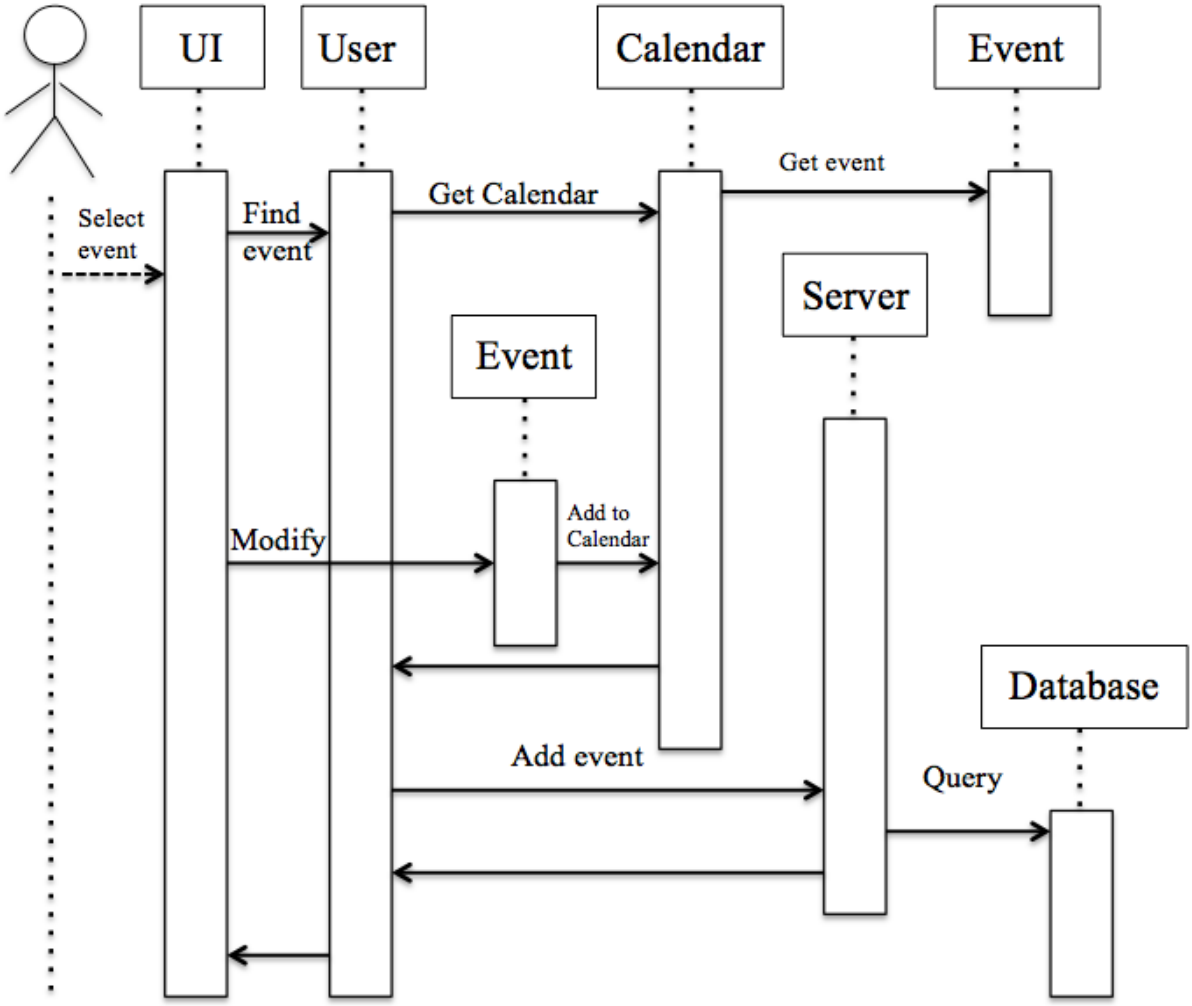
After the user presses the button of creating a group, he/she must select the people that they want to add to the group. The server sends query to the database and thus group is created in the database. After this the groupID gets updated in the user object. Similarly, users can refresh or request to join a group through interacting with the GUI. The server then sends the data to the database, and database will respond accordingly and update the corresponding information.

Event creation



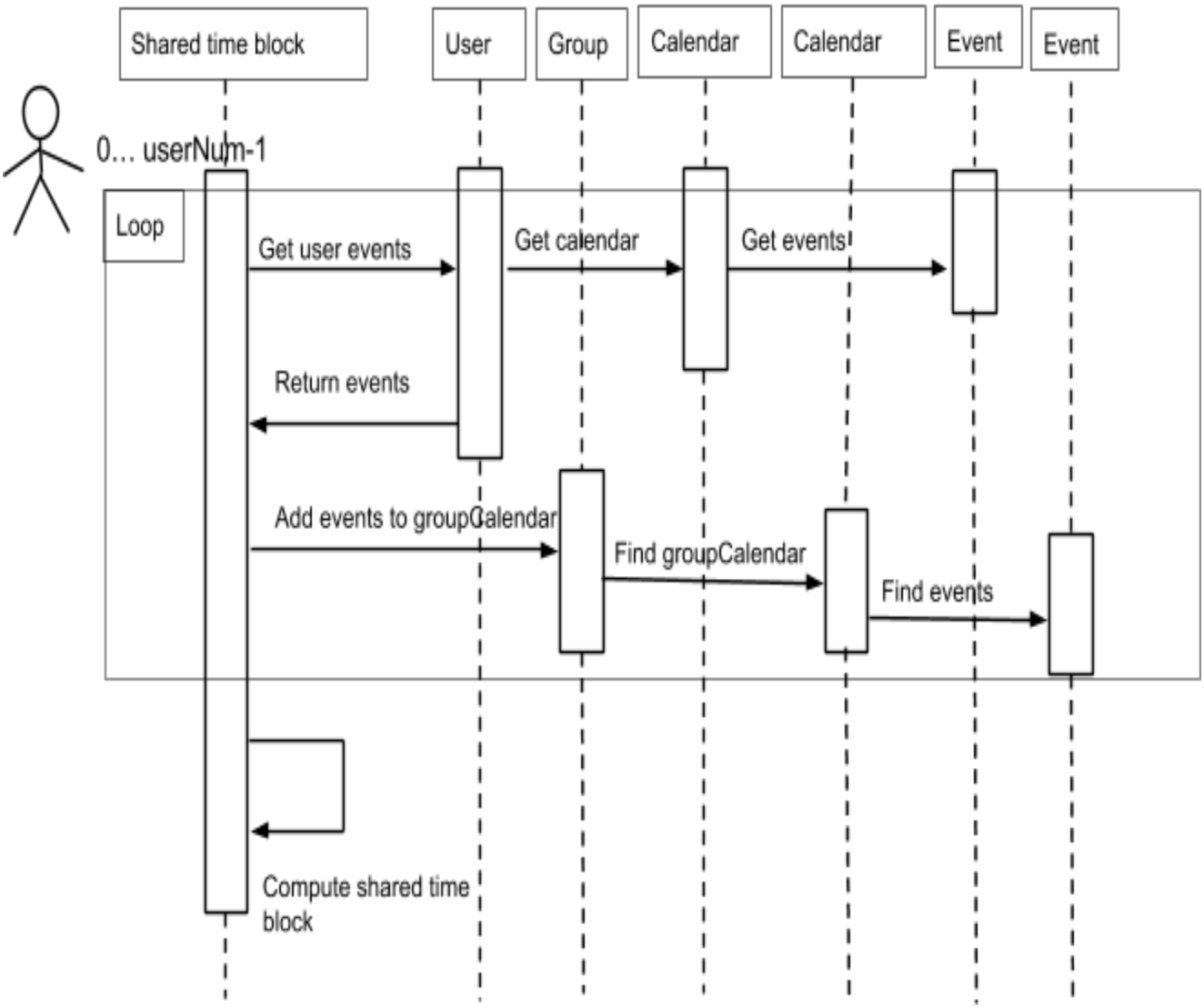
User presses the button on the GUI to create an event, and he/she must input necessary information such as event title, event content, start time and end time. The system gets calendar from the corresponding user object, and create a new event on the calendar. After this, user sends data relevant date to the server. Server updates the database and displays new event on the GUI.

Event modification



To modify an event, user first interacts with the UI. The system gets the calendar and then validate an event using eventID. User can edit the event by manipulating the text filed in the UI, the modified event is then added to the calendar. Subsequently, the event is passed to the server and server queries the database. Finally information will be passed to the UI to be displayed as multiple scheduled events.

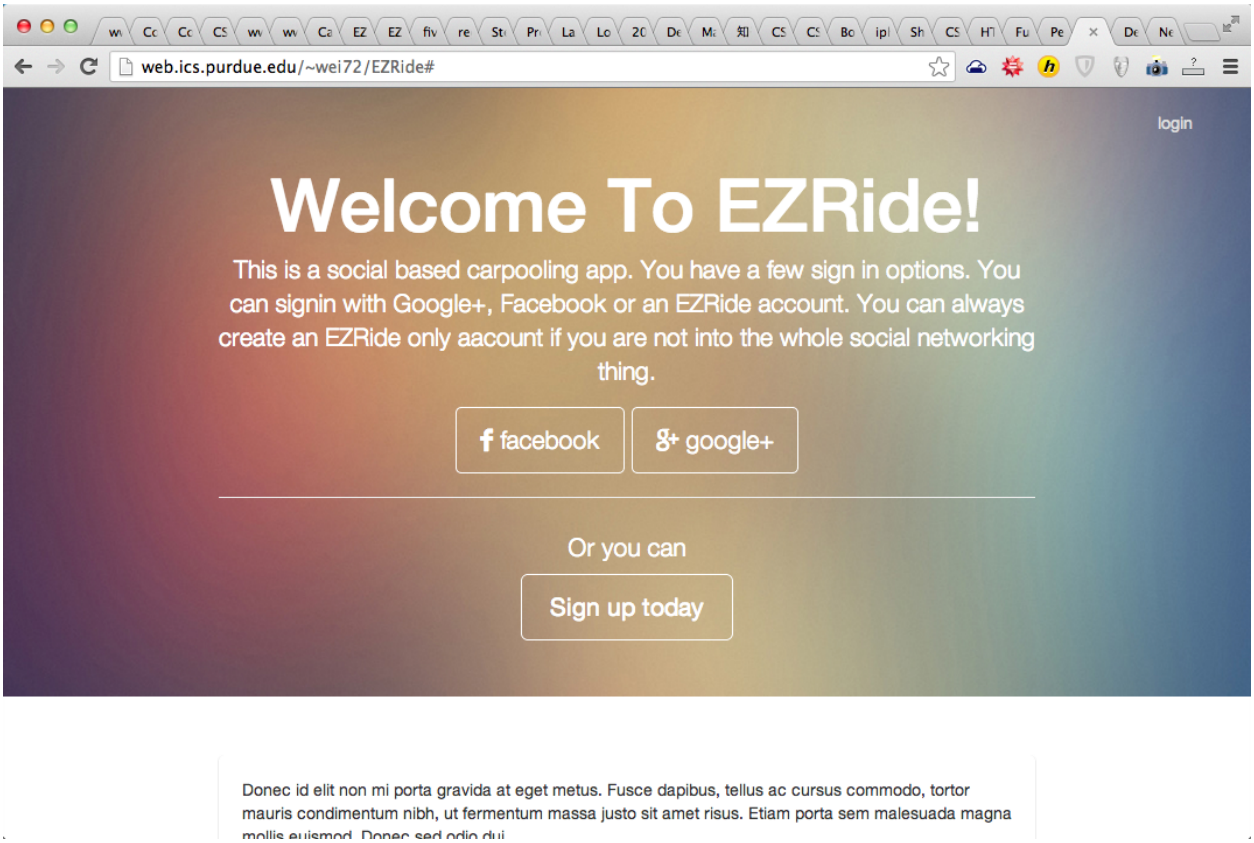
Event Match Calculation



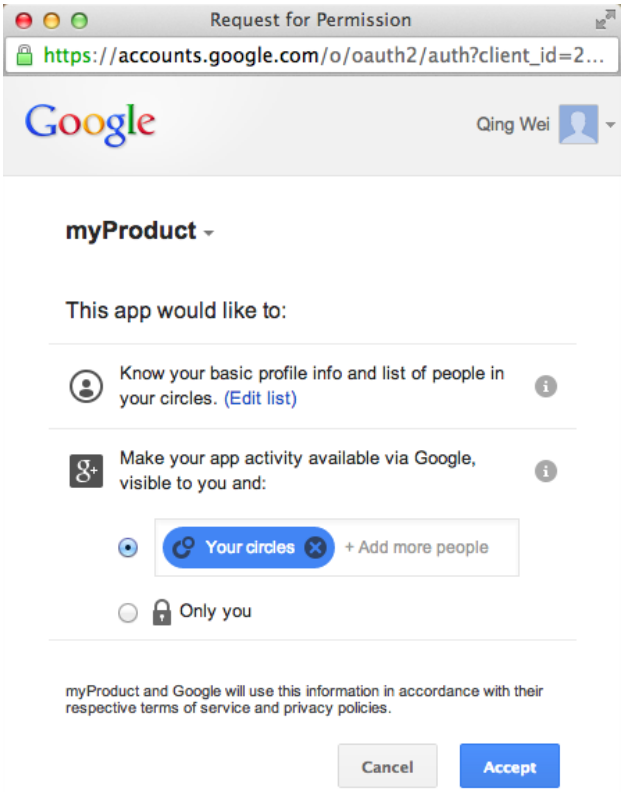
The algorithm to calculate the matched time is relatively straightforward. In each iteration of the user array, it first gets the user's events from the calendar. Each group object has a groupCalendar. The system finds the groupCalendar and marks off the time slots that are unavailable for each user.

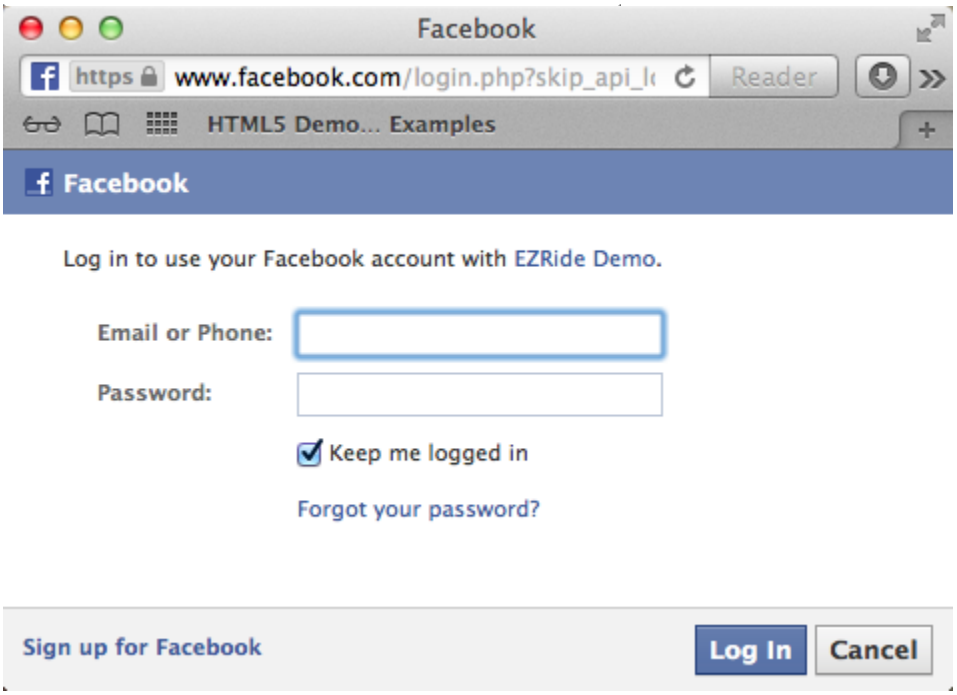
GUI Mock Ups

Login page of the web application

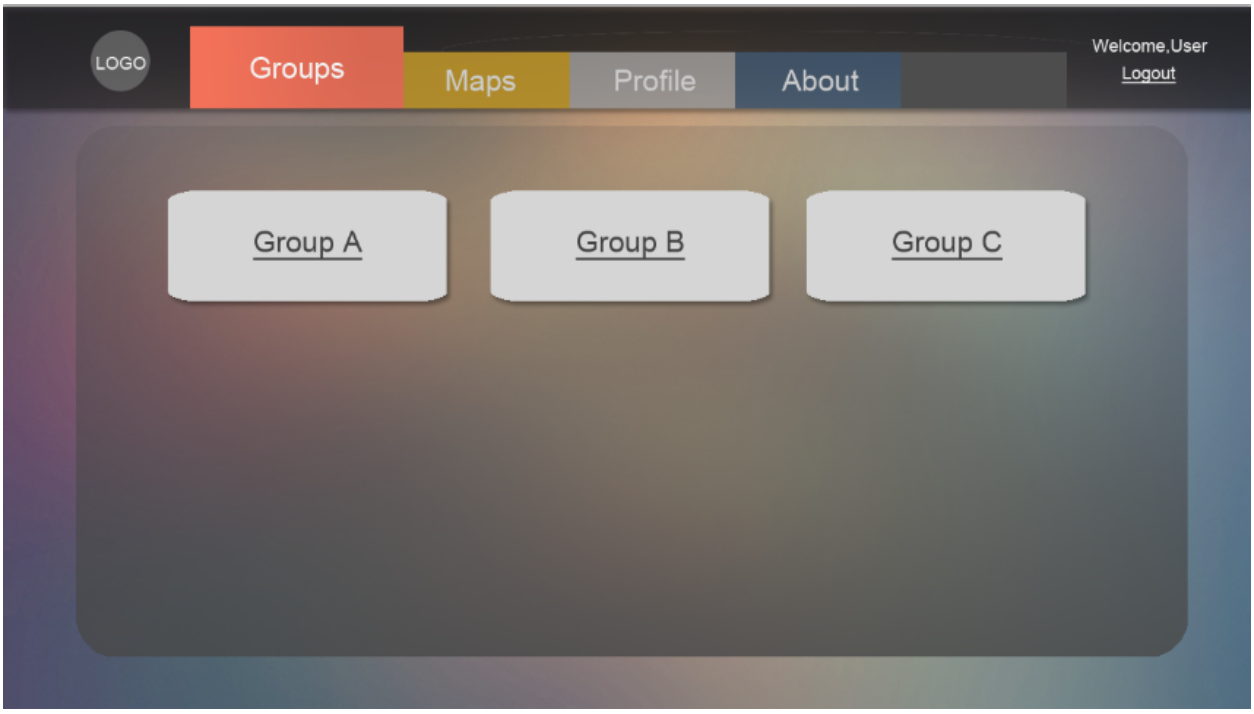


Authorization: user wants to login from google+ or facebook.

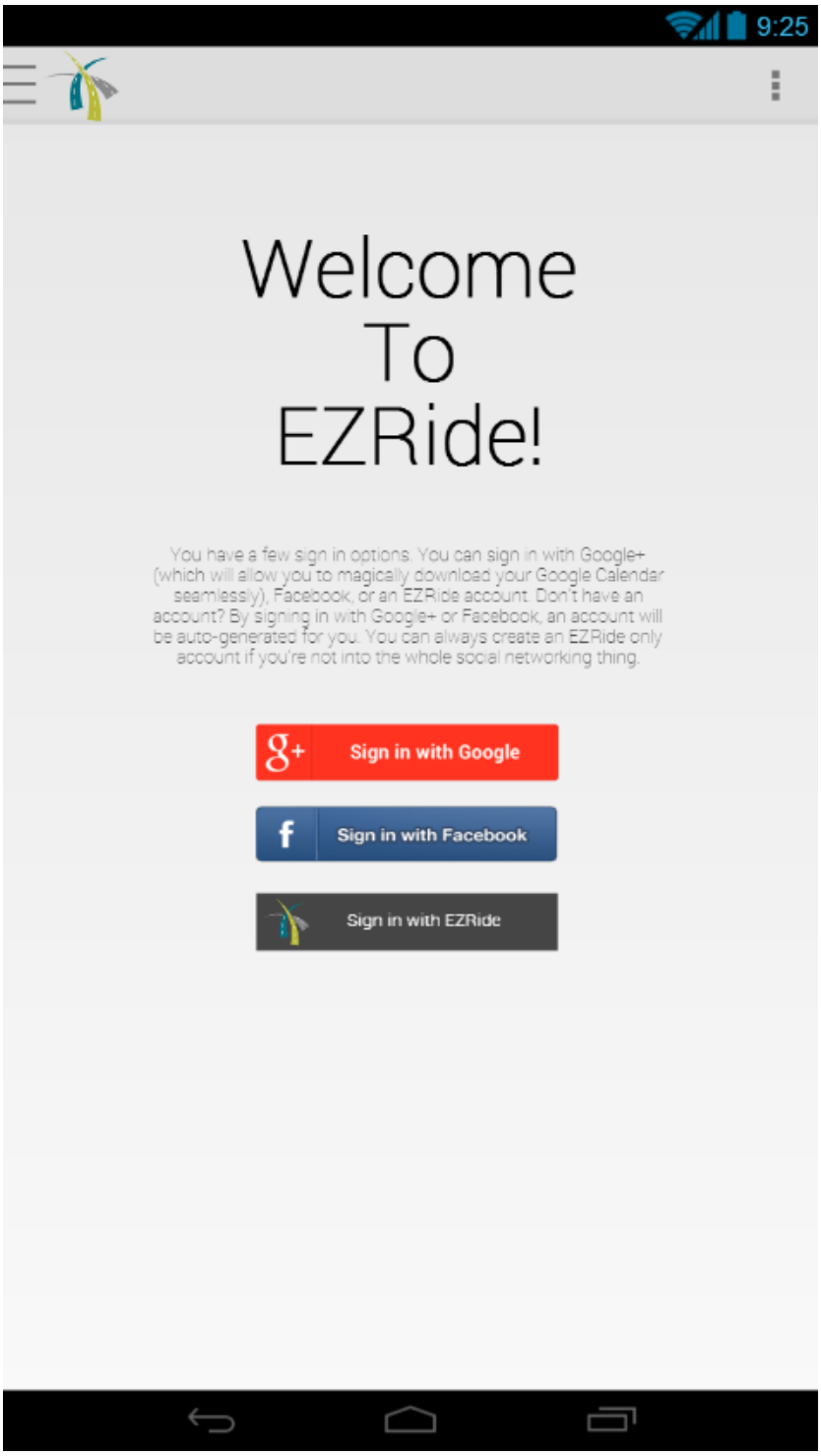




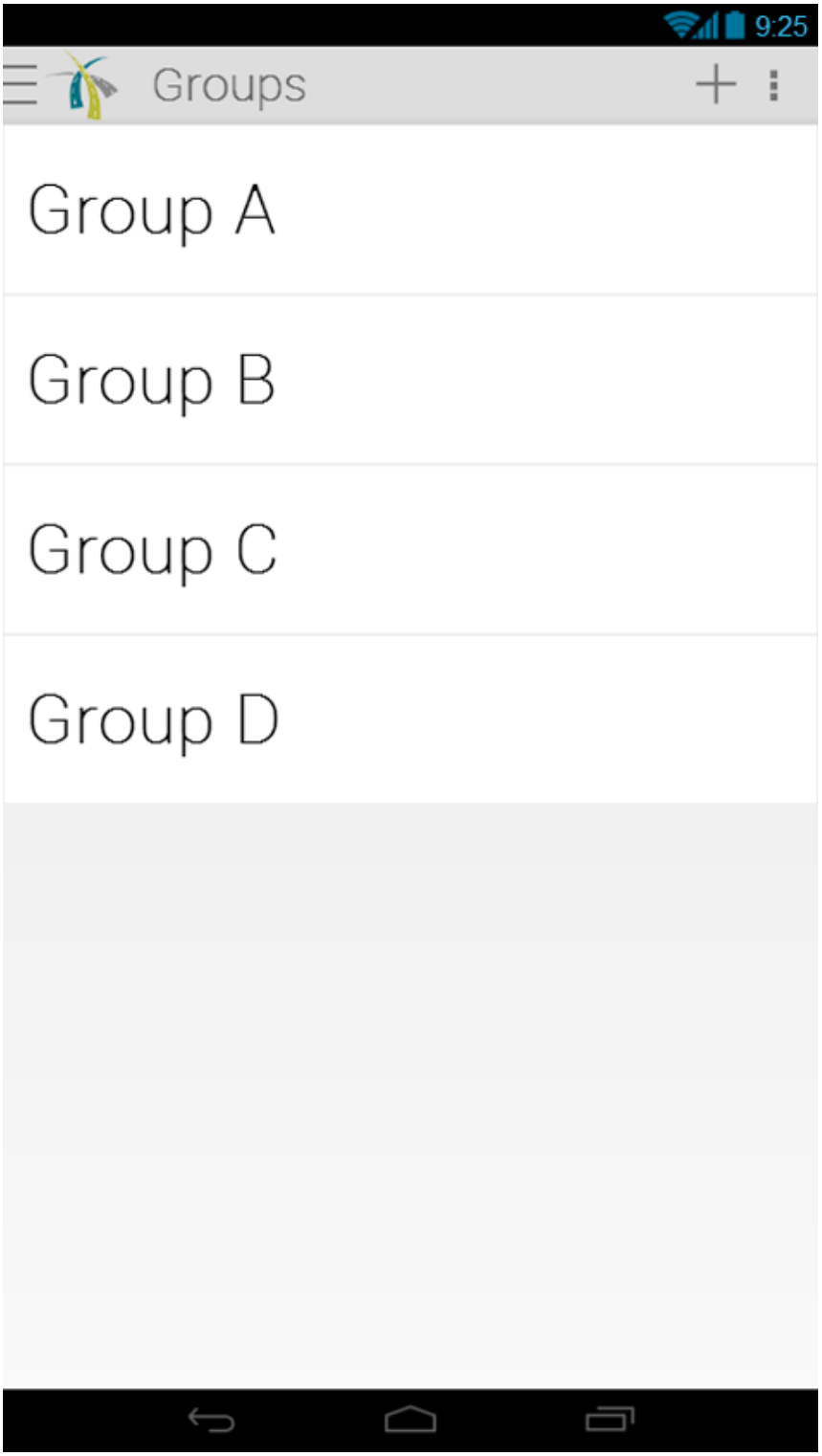
Main panel of web application



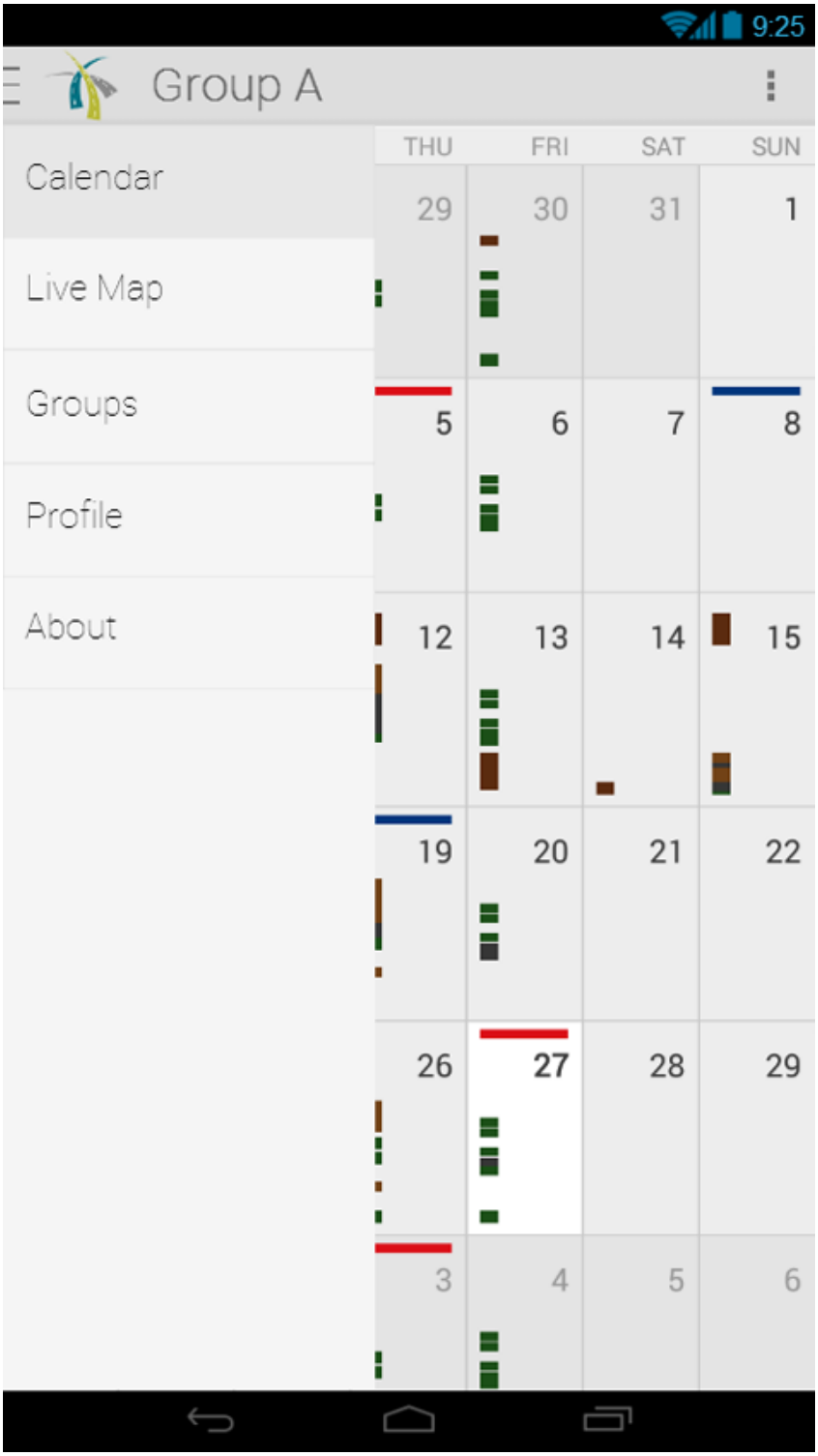
User wants to login from the Android phone



User wants to choose a group to join from a list of groups



User can view their own calendar and events on the phone



User can see the real-time location information on the Android map.