Socala

Software Requirements Specifications

Benjamin Romano

CJ Guttormsson

Bryan Anderson

2016-02-14

Revision History			
Date	**Version**	**Description**	**Author**
2016-02-15	0.1	Initial Draft	
2016-03-8	1.0	Final Draft	
2016-03-25	1.1	Initial Database Models Class Diagram, Requirements fixes	Benjamin Romano
2016-03-26	1.2	Front-End Models	Benjamin Romano
2016-03-27	1.3	Android App Activity and Fragments Diagram, App Services and Contexts Diagram	Benjamin Romano
2016-03-29	1.4	Added REST API Endpoints	Benjamin Romano
2016-03-31	1.5	Added Sequence Diagrams	Benjamin Romano
2016-03-31	1.6	Requirements Doc corrections	Bryan Anderson

Table of Contents

1. Introduction

The motivation behind Socala is to simplify the process of finding common times with friends and discovering new events in your area. Socala is an Android app that seeks to be the primary way that users plan events socially. Socala's goals are to enable users to discover events in their area and to provide an easy way for groups of users to plan events at mutually agreeable times. The app will have the ability to sync with existing calendar platforms starting with Gmail. Another feature we have is the ability to add friends and view their calendars. A friend can also set which events are visible to others.

2. Glossary

Friend: Someone has given you access to their calendar.

Event Visibility: Setting for visibility of an event.

Hidden - The event is not shown to other people. To others it will appear as if the user is free during that time period.

Partial - Time blocked off by event is shown to friend, but not details.

Friends - Friends can see all the details of an event.

Location - People in the nearby area can see all details about an event.

Nearby Area: 20 mile radius around the **current** location of the user. This means that the nearby area is not necessarily static.

OAuth Token: A token provided from a service provider (in this case Google+) to allow third parties to access a user's data. With the token, we can view and change a user's calendar and events. An oauth token is received after the user accepts our application's request to get access to their data.

FAB: Floating Action Button. A button which appears near the bottom of the screen and is not docked. Usually allows adding or editing something.

3. User Requirements Definition

3.1 Functional Requirements

1. A user is able to login using Google+ and give our app permission to view calendar.
2. A user must have a Google+ account.
3. The system will only handle one of the user's Google calendars. Multiple calendars per user are not supported.
4. A user is able to find and add friends using email address. Users are able to enter an email address, and the system returns the corresponding user or a user-not-found error.
5. A user is able to find upcoming events in his or her area. The app shows the user all publicly broadcast events within a 20-mile radius of the user.
6. To use location features, the user must have GPS enabled on their phone. This is necessary to establish the user's 20-mile-radius nearby area so that upcoming local events can be suggested.
7. A user is able to create events.
8. A calendar event can be set to different visibility levels: Location, Friend, Partial, Hidden. These visibility levels are defined above.
9. The system given a gmail oauth token can scrape calendar events.
10. The system must be able to sync calendars within 10 minutes of a change.

3.2 Non-Functional Requirements

1. The system must be able to run on devices with at least Lollipop.
2. The system must use strive to drain the battery as little as possible.
3. GPS – location data will be used to recommend local events

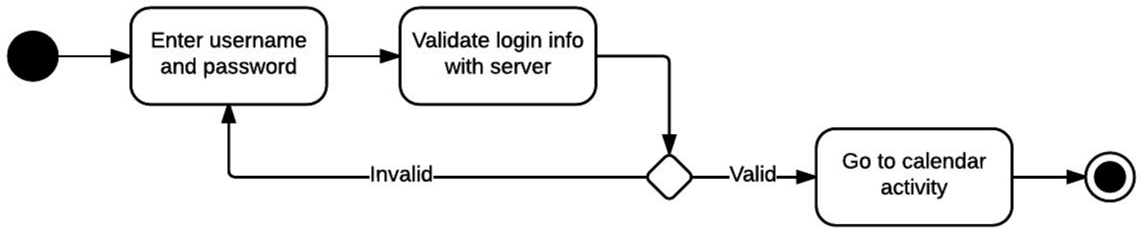
4. Data connectivity – used to sync with gmail and fetch info from other user's calendars.

4. System Architecture

The system will be composed of four activities, three fragments, two popups, a navbar, a database, a server and a service. The five activities include Login, Main, Common Times Calendar, and Event Details. The two popups are the Add Friend Popup and Calendar Options popup. The three fragments are Common Times Finder, Calendar, and Friends List. The navbar shows the different fragments and allows a user to switch between them. The Server runs a REST API which allows the android app to request resources from the database. The sync service is used to observe updates from a user's base calendar implementation (in our case Google Calendar).

4.1 Login Activity

The login screen is the first screen a user sees if they have not already signed into an account. On this screen a user is presented with a google+ button that takes them through the Google+ login activity. Once completed, the user is shown to the Calendar activity, which is our main activity.

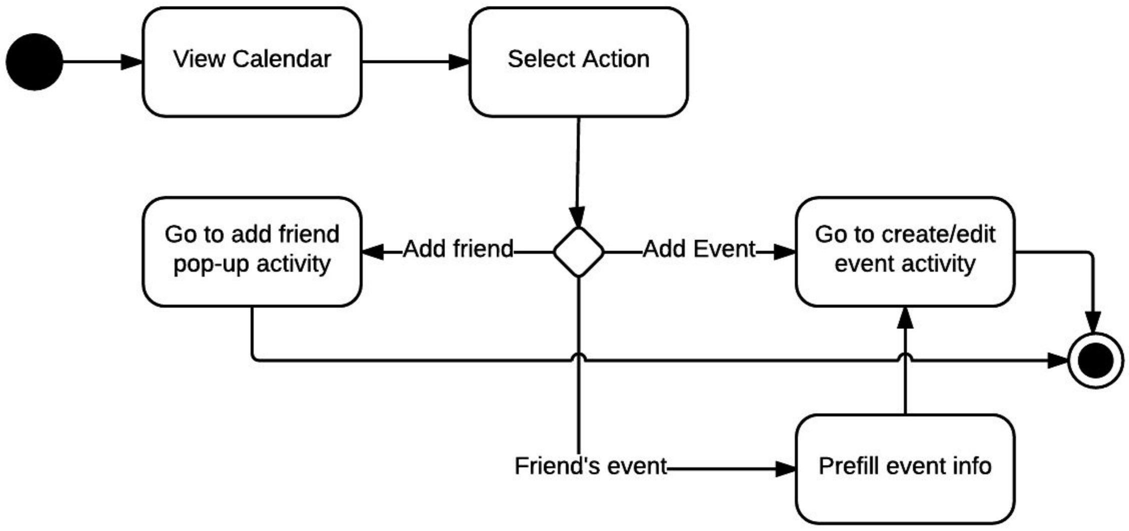


Main Activity

The main activity consists of the navbar, the android toolbar and a fragment manager. The fragment manager displays the currently selected fragment from the navbar. When the user finishes logging in they see the Main Activity with the Calendar Fragment active.

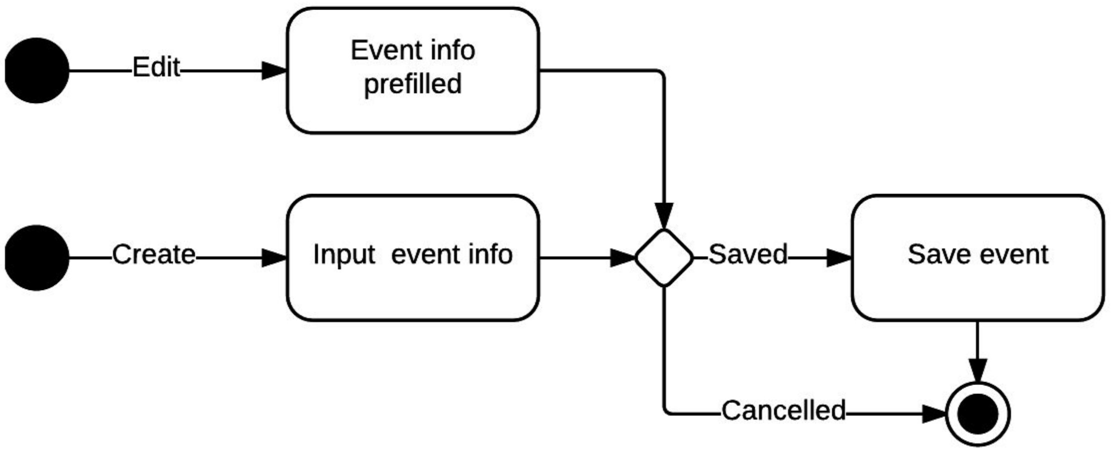
4.2 Calendar Fragment

The calendar fragment is the first fragment the user sees after logging in. In this fragment, the user is able to view their calendar and see upcoming events. On the top toolbar, there will be a gear icon which will start up the Calendar Options Popup. On this popup, a user is able to customize what is being displayed in the calendar fragment. In addition to the Calendar Options Popup, the calendar fragment will have a floating plus button in the bottom right which will allow a user to quickly open up the Create/Edit Event activity. If a user clicks on an event the user will be directed to the Create/Edit Event Activity with the event info filled in.



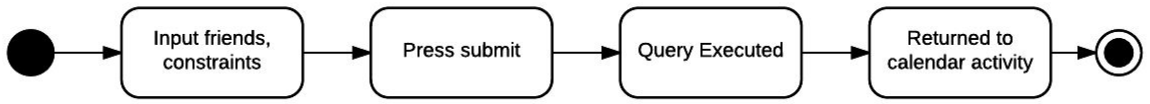
4.3 Event Details Activity

Either the Common Times Calendar Activity or the Calendar Fragment will lead to here. If the user clicked an existing event, the fields in this activity are filled with the info from the event. If the existing event is associated with the current user, the fields can be changed and the form can be saved. Otherwise, the form is read-only since the event doesn't belong to them. However, if the event is set to accept RSVPs then the user can rsvp to the event. RSVPing an event adders the user to the event as an attendee. Next if the event belongs to the current user, then the event can be edited and saved.



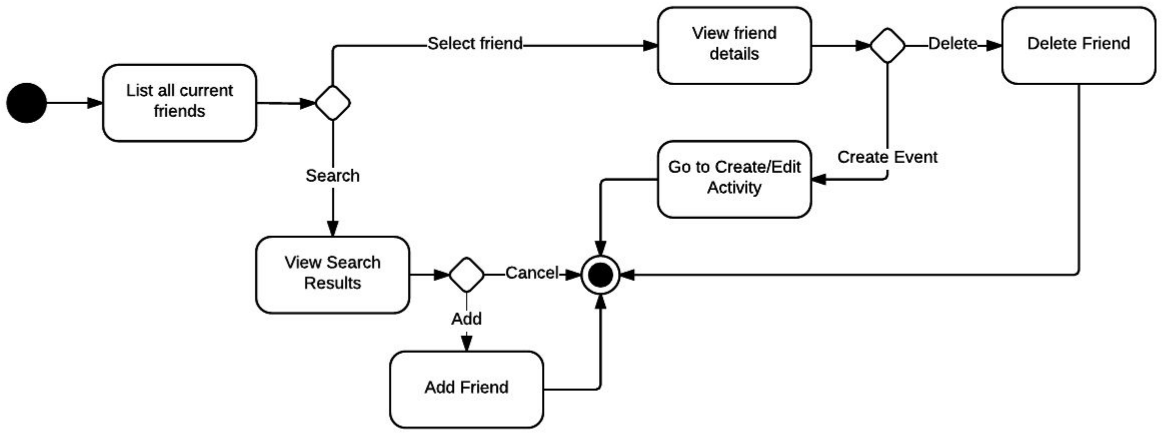
4.4 Common Time Finder Fragment

The Common Time Finder Fragment contains a form and a submit button. The form includes fields for specifying which users to add, how long the event will run and time constraints. A user can be selected from the friends list dropdown or an email can be typed manually. An error will occur if the email supplied is not a user of Calendar. Once submit is clicked, the query will be executed and the user will be returned to the calendar activity with the relevant information being presented.



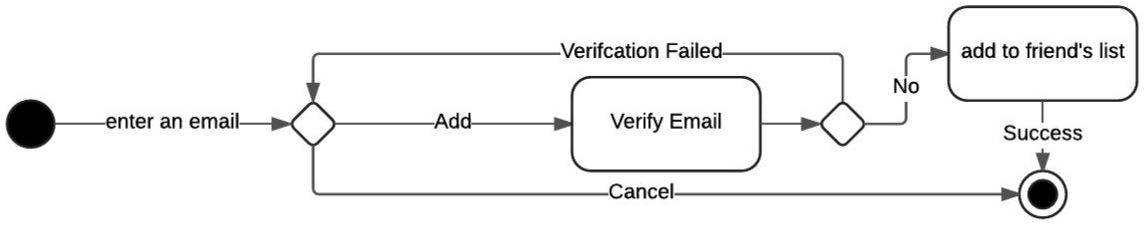
4.5 Friend List Fragment

The friend list fragment lists all friends of a user by their name (pulled from gmail info) or their email if no name is present. The user is able to delete a friend on this activity. The floating plus button on the bottom right will be displayed on this activity. The two options Add Friend and Add Event will be displayed.



4.6 Add friend popup

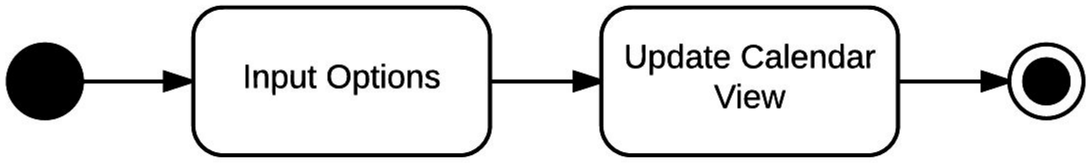
The add friend popup will display a field to enter an email address. If the email entered is associated with a user who is registered to our service, the friend will be added to a user's friend list. Otherwise an error message is displayed.



4.7 Calendar options popup

The calendar options popup will display whether or not to show upcoming events from either friends or people nearby. In the calendar options, a user can also set which calendar to display. The user could choose between showing only their own calendar or adding in other friend's calendars. Another feature of the calendar options popup is to set how to display the calendar. The user can choose between a month, week and day view.

The calendar options popup will allow users to configure the current calendar activity. The first setting is checkbox which lets a user specify whether or not to display events in the nearby area. The next is a dropdown with checkboxes at each element which lets users select which friend's calendars to show. Each element in the dropdown shows the friend's real name or email if a name is not available. Next there is a dropdown to select how to format the calendar view. The settings for this are month, day, and week.



4.8 Common Time Calendar Activity

The common time calendar is similar to the calendar fragment except the calendar options and FAB are not present. When a user clicks an empty slot. They are brought to the event details activity to create a new event. Time slots which conflict with existing events are marked off as busy so the user can not select that time frame for the event they wish to create.

4.9 Navbar

The navbar will be used to switch between the fragments. This includes Calendar Fragment, Friend List Fragment, and the Common Time Finder Fragment

4.10 Server

The server is a REST API which can be used by the mobile app to query the database.

4.11 NoSQL Database

The database will cache event information and store user info. The dataService class will interact with this database.

4.12 Sync Service

The sync service is used to keep our custom events consistent with the lower level google calendar events. This will use the Google Calendar API which allows subscribing to changes instead of having to poll every so often.

5. System Requirements Specification

Below are the models for our app. These models will be stored in a database which will be retrieved by the Server for use in the android app when requested through the REST API.

5.1 User Class

The User class contains information on an individual user, including their personal information and OAuth token. The user also has a list of friends (other instances of the User class) and a calendar (an instance of the Calendar class).

5.2 Calendar Class

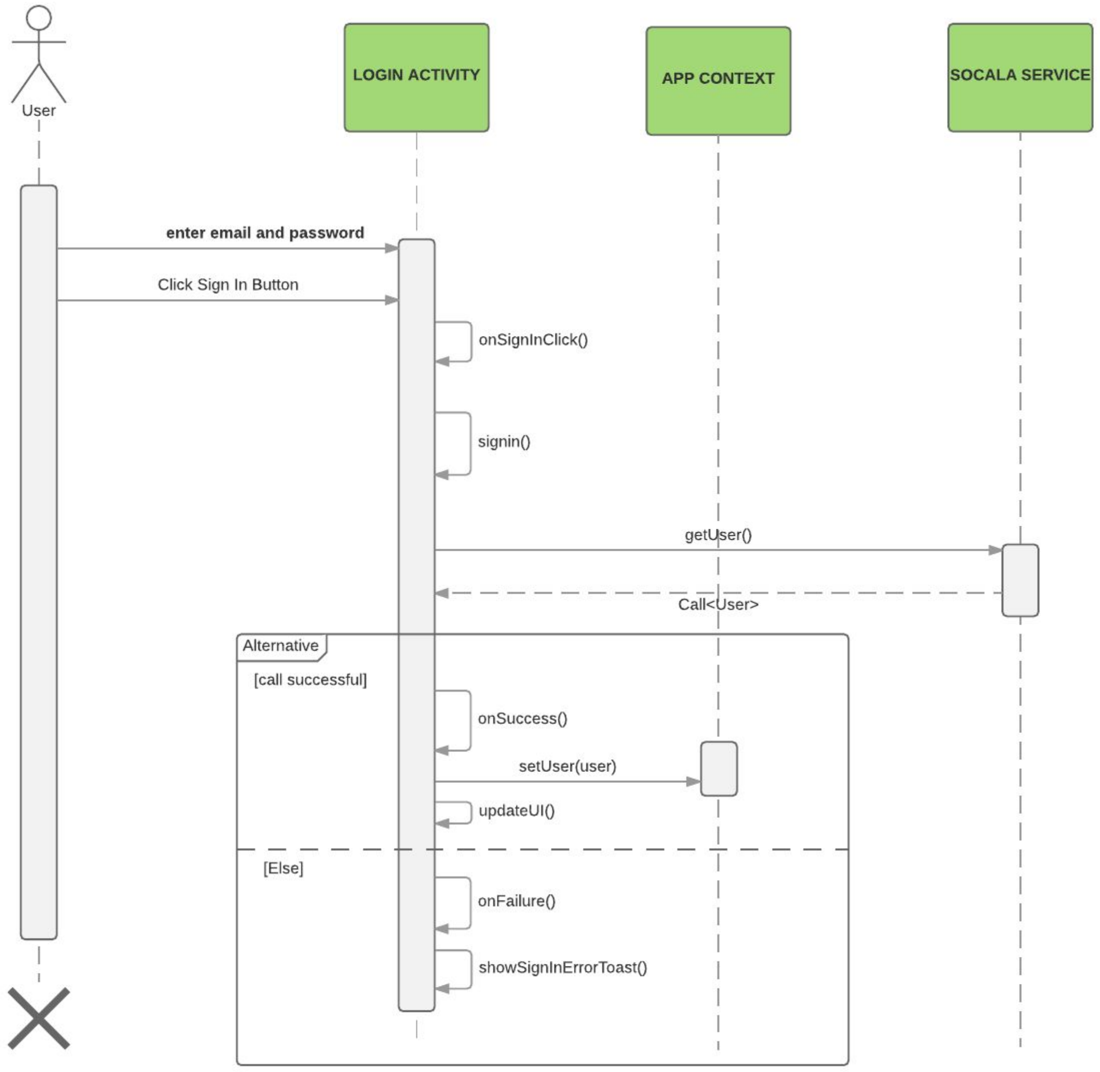
The Calendar class represents the calendar that belongs to some user (an instance of the User class). The calendar contains a list of events (instances of the Event class).

5.3 Event Class

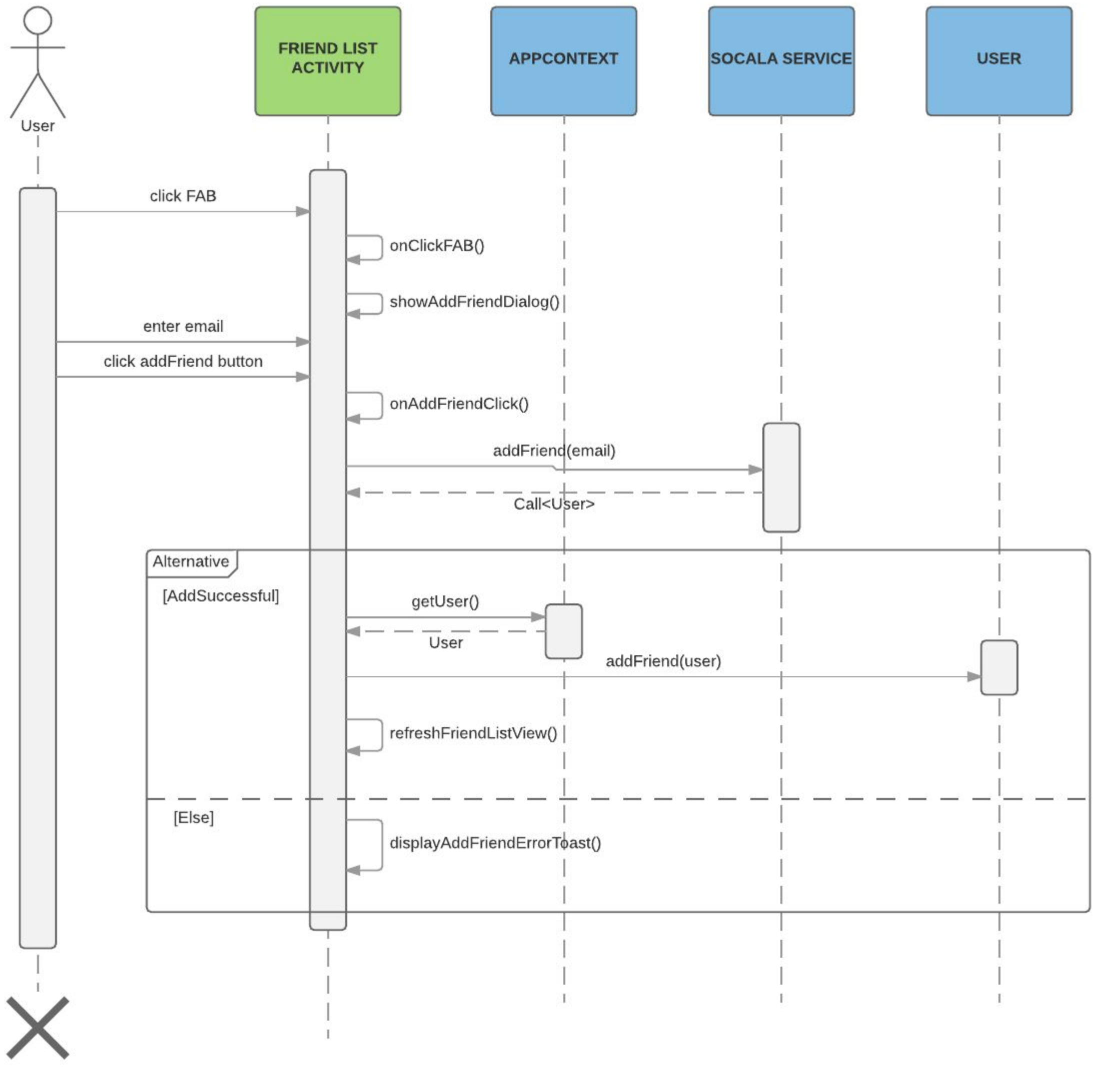
The Event class represents an event that users may place on their calendars. It contains a beginning and end time as well as identifying information (a name, a location, an image, and a description). It also contains a list of calendars (instances of the Calendar class) that have chosen to list the event, along with that calendar’s chosen privacy level (whether its participation is visible to all, visible only to friends, visible only as an unmarked block of busy time, or visible to no one).

6. Sequence Diagrams

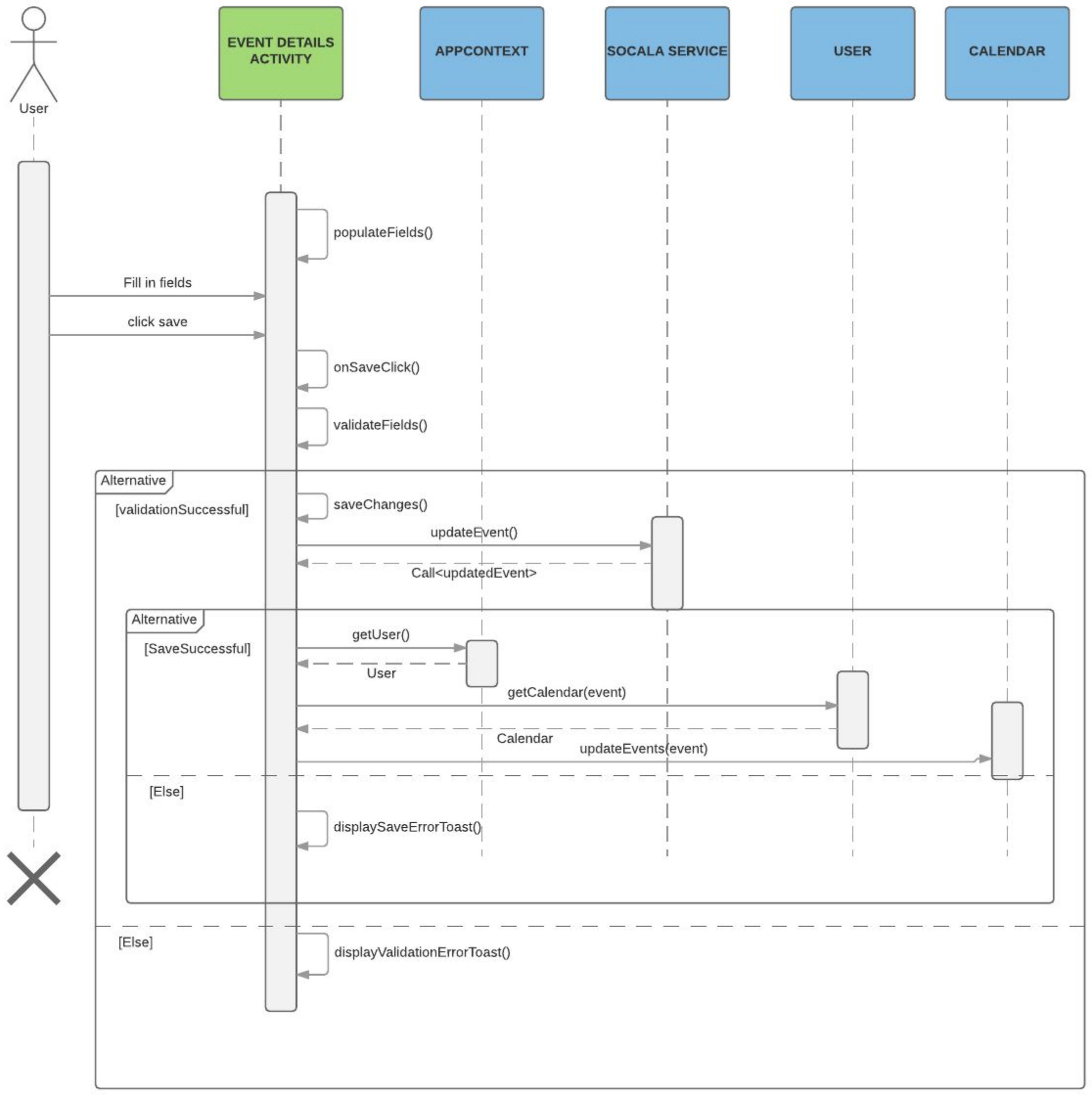
6.1 SignIn



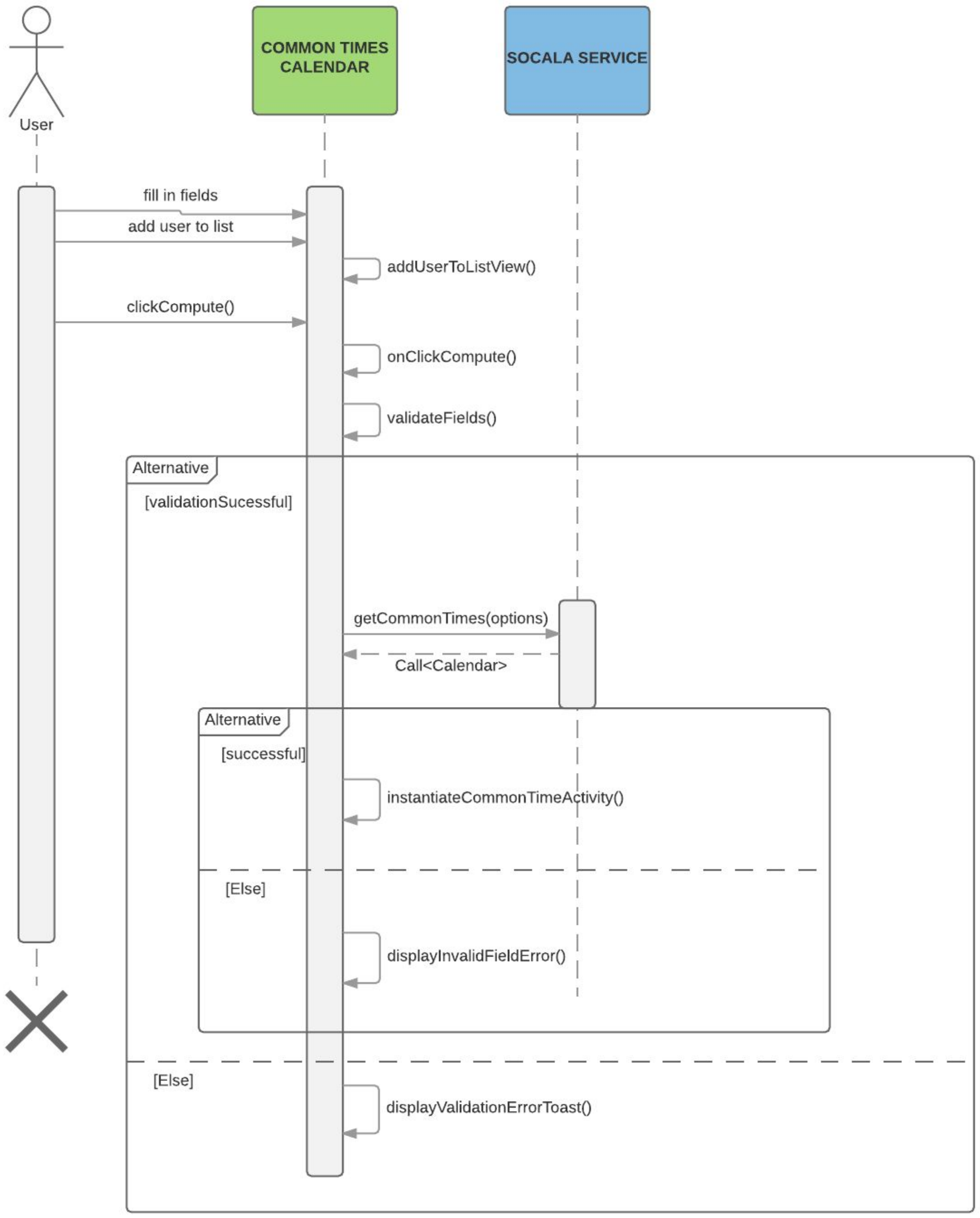
6.2 Add Friend



6.3 Add/Edit event

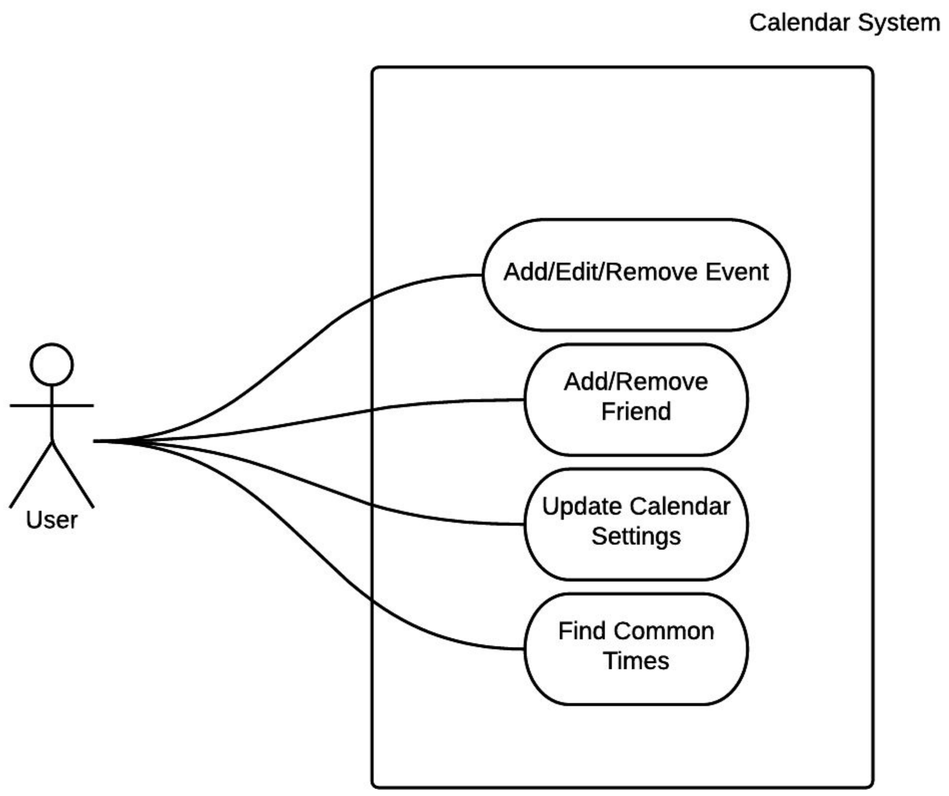


6.4 Find Common Time

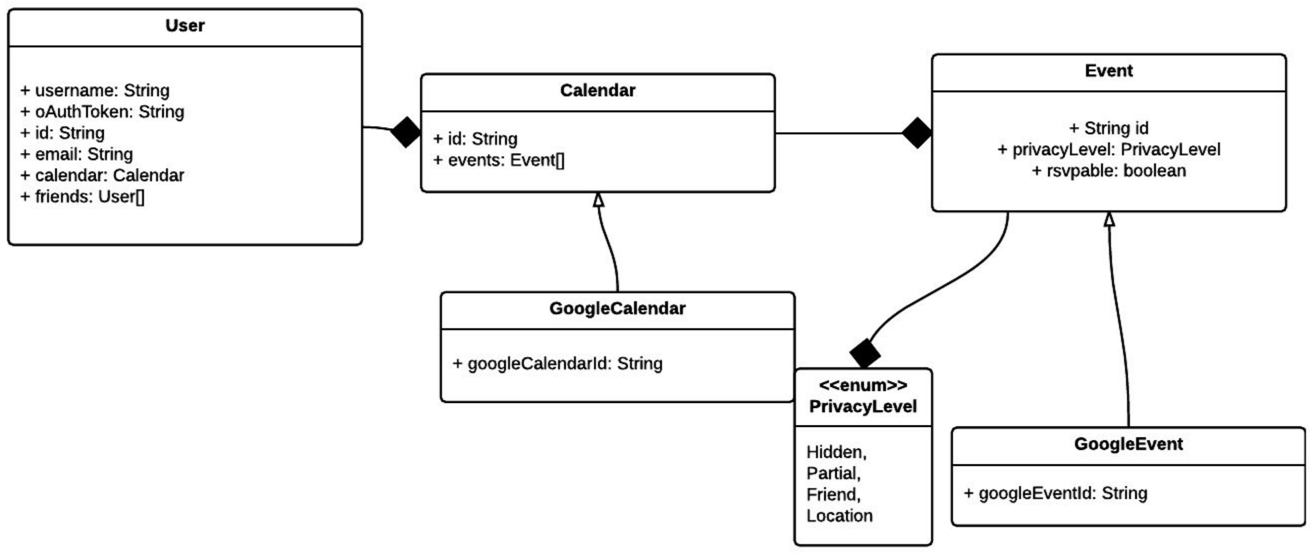


7. System Architecture

7.1 Use Case Diagram

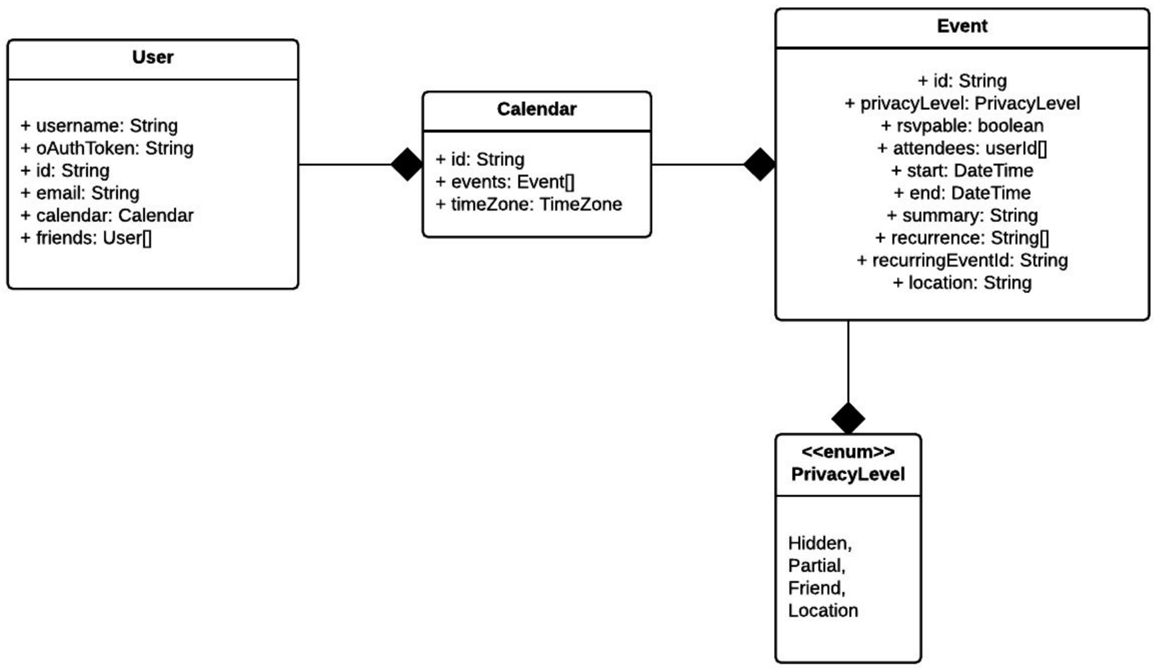


7.2 Database Models



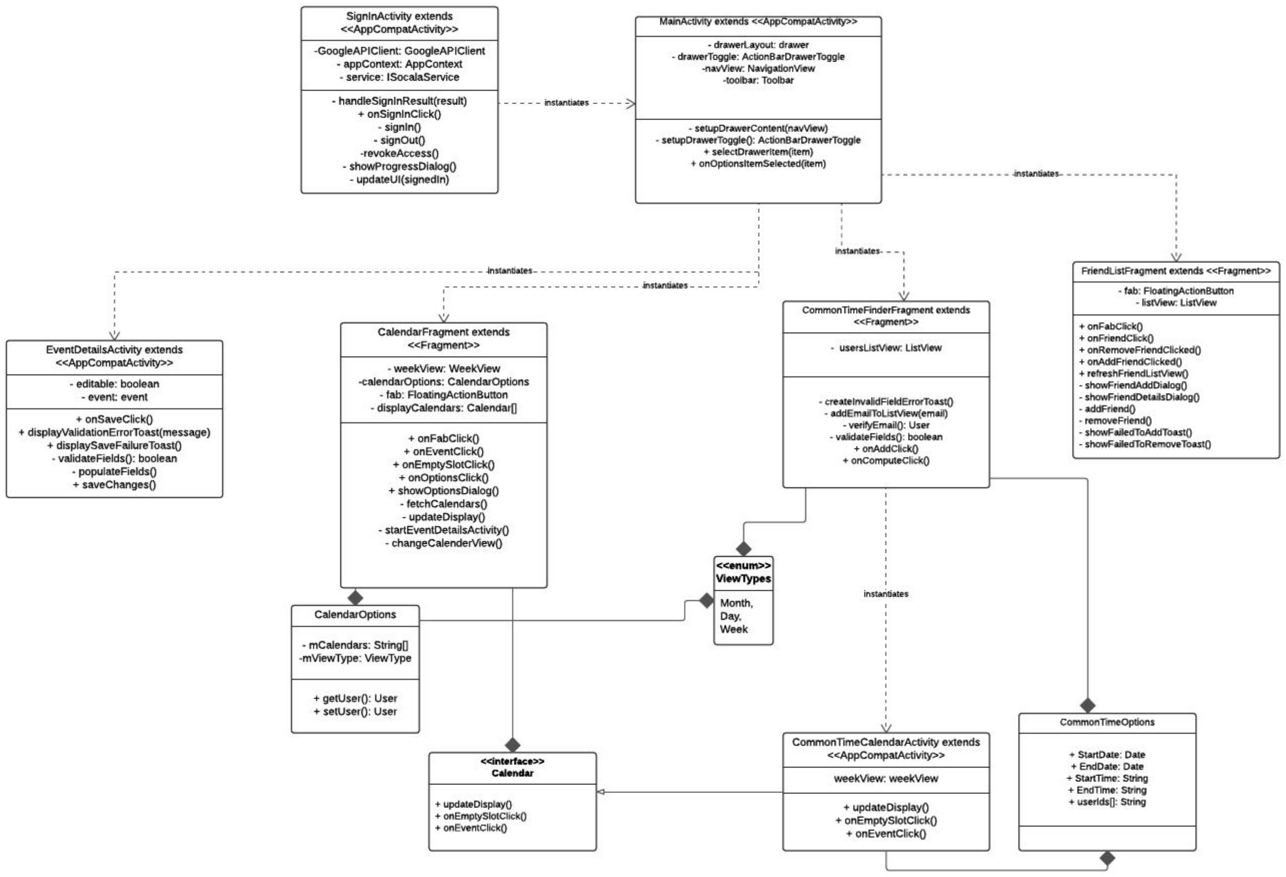
This class diagram represents our database models. These models showcase what will be stored in the database. The front-end models look very similar except for the events. An event can be implemented as a googleEvent; however, the front-end will not be aware of the underlying implementation. The back-end will fetch the relevant properties from the Google Calendar Event and insert it into the Event object to be then passed to the front-end.

7.3 Front-End Models



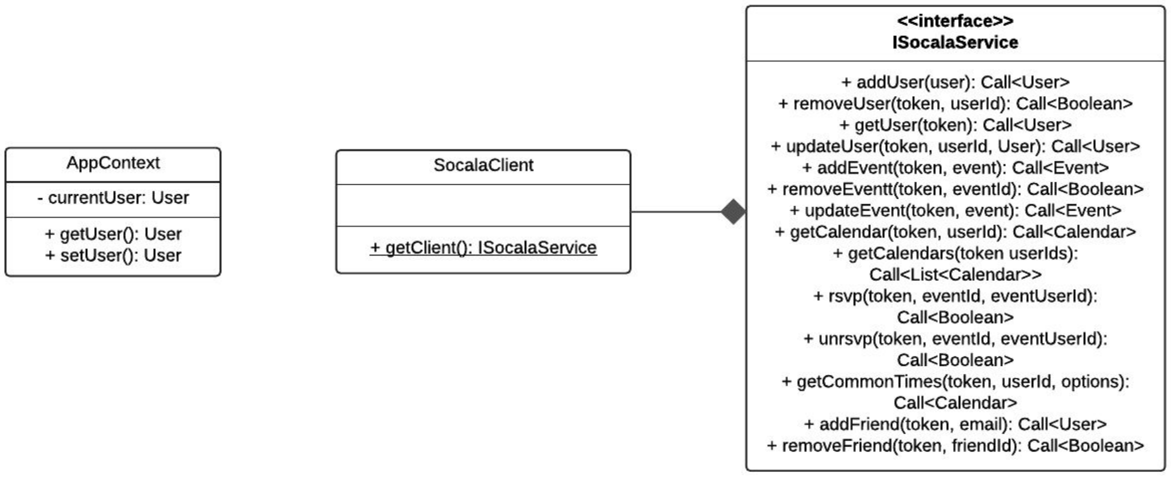
The front-end model is very similar to the back-end models. However, the difference is that the relevant properties from the underlying client implementation (Google Calendar in this case) are extracted and put onto these models. This allows the front-end to work without knowledge of which underlying calendar client implementation is being used.

7.4 Android App Activity and Fragments Diagram



Above is the main class diagram for our android app. The SignInActivity uses the Google API to handle sign in and retrieving a user's oauth token. The mainActivity has a fragmentManager which allows swapping out the current fragment being displayed. All communication between the fragments happens through the mainActivity. Fragments which need to do special work that is independent from the other fragments spawn activities to do these tasks. For example, the Calendar Fragment spawns an Event Details Activity to view/create/update properties on an event.

7.5 App Services and Contexts Diagram


AppContext
- currentUser: User
+ getUser(): User
+ setUser(): User
SocalaClient
+ getClient(): ISocalaService
<<interface>>
ISocalaService
+ addUser(user): Call<User>
+ removeUser(token, userId): Call<Boolean>
+ getUser(token): Call<User>
+ updateUser(token, userId, User): Call<User>
+ addEvent(token, event): Call<Event>
+ removeEventt(token, eventId): Call<Boolean>
+ updateEvent(token, event): Call<Event>
+ getCalendar(token, userId): Call<Calendar>
+ getCalendars(token userIds): Call<List<Calendar>>
+ rsvp(token, eventId, eventUserId): Call<Boolean>
+ unrsvp(token, eventId, eventUserId): Call<Boolean>
+ getCommonTimes(token, userId, options): Call<Calendar>
+ addFriend(token, email): Call<User>
+ removeFriend(token, friendId): Call<Boolean>

AppContext is a singleton used to keep the user data consistent between activities and fragments. The signInActivity sets the user on the appContext after handling the sign in flow. After the signIn flow, it is guaranteed that a user is present on the instance. The ISocalaService is the interface used to talk with our backend REST API. Every function described in this service returns an object wrapped in a Call class. Call is a class that has onSuccess and onFailure handlers. The SocalaClient is a class with a static function getClient that creates an instance of ISocalaService and returns it. This is done using the popular Retrofit library that converts interfaces into working HttpClients.

7.6 Socala REST API Endpoints

Add User:
POST /user/
Body: User
Response: User

Get User:
GET /user/
Headers:
    Authorization: OAuthToken
Response: User

Delete User:
DELETE /user/{id}
Headers:
    Authorization: OAuthToken
Response: Boolean

Update User:
PUT /user/{id}
Headers:
    Authorization: OAuthToken
Body: User
Response: Boolean

Add Event
POST /event/
Headers:
    Authorization: OAuthToken
Body: Event
Response: Event

Update Event
Put /event/{id}
Headers:
    Authorization: OAuthToken
Body: Event
Response: Event

Delete Event
DELETE /event/{id}
Headers:
    Authorization: OAuthToken
Response: Boolean

Get Calendar
GET /calendar/{userId}
Headers:
    Authorization: OAuthToken
Response: Calendar

Get Calendars
POST /calendars/
Headers:
    Authorization: OAuthToken
Body: userId[]
Response: Calendar

RSVP
GET /user/{id}/rsvp/{eventId}
Headers:
    Authorization: OAuthToken
Response: Boolean

UNRSVP
GET /user/{id}/unrsvp/{eventId}
Headers:
    Authorization: OAuthToken
Response: Boolean

Get Common Time
GET /events/commonTimes
Headers:
    Authorization: OAuthToken
Body: CommonTimeFinderOptions
Response: Calendar

Add Friend
GET /user/friend/add?email
Headers:
    Authorization: OAuthToken
Response: User

Remove Friend
GET /user/friend/remove?email
Headers:
    Authorization: OAuthToken
Response: Boolean