California State University, Long Beach

CECS 491A - Software Development Project

Fall 2014

Team # 5

Project Title: Corral

Team Members:

Alex Bu

Nicholas Gallo

Joaquin Gonzalez

Nicholas Kwong

Israel Torres

Gyngai Ung

Brandon Whitney

Abstract

Introduction

With growing social networks, it becomes tedious to get on the same page as your friends when they are from all walks of life. Although you may have it in the back of your head when a friend may be free, you might not actually know if that person is available. Once you start pestering them with phone calls, text messages, and emails; it is hard to distinguish whether you are trying to hangout or have an emergency. If you start calling them when they are in a lecture or an important interview, you can embarrass them or even cost them a job. To avoid this confusion and cluttering each other's calendars, Corral (cor·ral) will help people get together at anytime with less hassle.

Goals

Corral will attempt to solve these problems by consolidating user's and their friends' calendars, thus finding the most appropriate time that everyone is free. Without having to re-enter individual events, Corral connects with Facebook and Google Calendar's, eliminating the need to maintain yet another calendar. By comparing the users' schedule and their friends' schedule, Corral can then inform the user if their friends are available at this moment in time or find out when their friends are available in the future. Along with this, Corral will also let users specify groups of friends, share if they are free via Facebook and Twitter, and notifying friends within Corral if they would like to get together.

Table of Contents

1. Introduction and Background

1.1. Statement of Problem Area

As friends coming from all walks of life, it becomes tedious to get on the same page and find free time to hang out together. Corral helps solve this problem by allowing user to find out which friends are available to hang out, quickly and with less hassle.

1.2. Previous and Current Work, Methods and Procedures

Previous and currently, people are using all sort of communication means, such text messaging, Facebook messaging, Google Hangout and the like, to get a hold of one another in order to find out the time to hang out together. There is a lot of hassle in this process because not all of your friends and family members are using the same application or they may not have the time to engage in text messaging or phone conversation just to find the time to hang out. Corral simplifies the process by pulling users calendar events (with user's permission) and figuring out when the users are available to hang out with friends or family without the hassle of text messaging, phone call or compromising on their privacy by revealing their calendar events to other people. Corral also allow the users to overwrite their calendar events by easily setting their own availability and the period of time they desire.

1.3. Background

With growing social networks, it becomes tedious to get on the same page as your friends as they come from all walks of life. Although you may have it in the back of your head when a friend may be free, you might not know when that person is actually available. It may not be inconvenient to call or text your friend all the time. To avoid this confusion and cluttering each other's calendars, Corral (cor·ral) helps people get together at anytime with less hassle.

1.4. Brief Project Description

Corral helps people get together with less hassle, by pulling user's calendar events and figuring out when the user is available so that their friends can ask him/her to hang out with. Corral also allows user to explicitly set availability for a fixed period of time, as desired.

1.5. Purpose of Project

The purpose of the app is to save user the time and avoiding the hassle of texting and calling each other in order to hang out together.

1.6. Team work assignment and accomplished

- Alex Bu
    - Implement Main UI page (Front end + JS)
    - Implement Friend request UI (Front end + JS)
    - Implement Nudges UI (Front end + JS)
    - Implement Friends list UI (Front end + JS)
    - Implement Set Mood UI (Front end + JS)
    - Implement Set Availability (Front end + JS)
    - Implement assigning friend(s) to favorite group UI (Front end + JS)
    - Implement Available Later View (Front end + JS)
    - Final report write-up

- Nicholas Gallo
    - Implement Corral user account Sign up (Back end)
    - Implement Log in and log out function for Corral user (Backd end)
    - Implement List all friends function (Back end)
    - Implement assigning friend(s) to favorite group (Back end)
    - Implement get nudges and set nudges function (Back end)
    - corral.readme.io online documentation
    - Final report write-up

- Joaquin Gonzalez
    - Implement notification services for nudges (Front end)
    - Create user account via Facebook (Back end and front end)
    - Final report write-up

- Nicholas Kwong
    - Invite friends by email address implementation (Front end)
    - Research Facebook Sharing feature
    - User input form validation (Front end)
    - Final report write-up

- Israel Torres
    - Establish technology stacks for the project
    - Create tables and set constraint in database
    - Provide server/hosting environment for API and MySQL database
    - Create user account via Google (Back end and front end)
    - Google and Facebook log in and log out function (Back end and front end)
    - Invite friends using Google Contacts (Front end)
    - Implement Email notification for inviting friend (Back end)
    - Final report write-up

- Gyngai Ung

    - Create and maintain database
    - UML diagram
    - Implement pull events from user's Google Calendar(s) (Back end and front end)
    - Implement get user's Google Contact for inviting friends to join Corral (Back end)
    - Implement set availability for a period of time (Back end)
    - Implement get nudges and set nudges function (Back end)
    - Modify get all available friends, taking into account the expiration of user's time availability
    - Final report write-up

- Brandon Whitney
    - Implement Corral user account Sign up (Back end)
    - Implement log in and log out function for Corral user (Back end)
    - Implement get all available friends in the first sprint
    - Implement invite friend function (Back end)
    - Implement accept friend request function (Back end)
    - Test case design and documentation
    - QA Testing between new builds
    - Create presentation for each sprint
    - Final report write-up

2. System Functional Specification

2.1. Functions Performed

- User Controller
  - create()
    - Create a user account based on user-provided email and password
  - login()
    - Log the user into the app, given the correct password.
  - logout()
    - Log the user out of the app.
  - getMyInfo()
    - Return the user's id, first name, last name, email address, mood, status, remaining time of the given status, and whether the user is Google user or Corral user.
  - checkUserExists()
    - Chec if a friend with given email address exists in the database.
  - addFriend(friendId)
    - Add friendship to the user profile with provided provided friendId.
  - acceptFriend(friendId)
    - Accept friendship request to the user profile with the provided friendId.
  - deleteFriend(friendId)
    - Delete friendship to the user profile with the provided firendId.
  - sendInvite()
    - For existing user, send in-app friend friend. For non-existing user, send email invitation via email queue.
  - getFriends()
    - Return the list of friends with the attribute id, first name, last name and favorite flag.
  - getFriendsCount()
    - Return the total number of friends
  - getRequests()
    - Return the list of friends who send request to the user, with the attribute id, first name, last name, email address and the request total count.
  - getAvailable()
    - Return the list of friends who are available now.
  - getAvailableFuture()
    - Return the list of friends who are available at a future, given the date and time.

    - checkAvailability()
        - Check the availability that the user set. Change the availability to be based on schedule when the set time expires.
    - setMood()
        - Set the mood of the user
    - setFavorite(friendId)
        - Given the friendId, set the friend to be a favorite friend.
    - getNudges()
        - Return the nudges from friends, with the attribute, id, first name, last name, the nudge message and the nudges total count.
    - setNudges()
        - Set the nudge to friend, given the friendId.
    - setTimeAvailability()
        - Set the availability time and status, given the number of minutes, from the time the function is evoked.

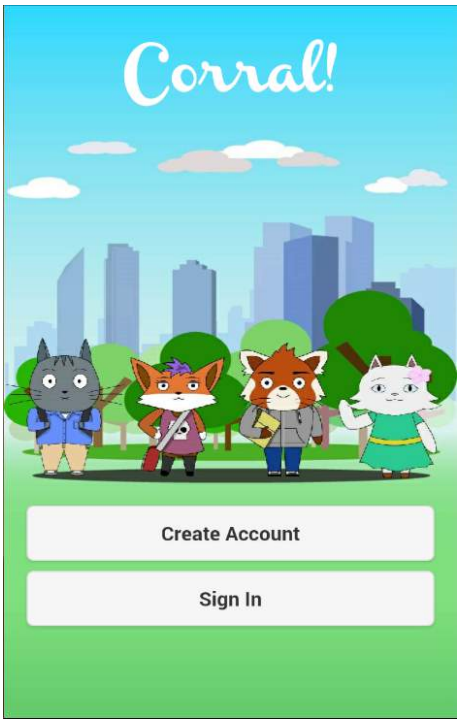
- Google Controller
    - create()
        - Create a user account based on user's gmail credential.
    - login()
        - Log the user into the app, given the user's gmail credential.
    - getProfile(accessToken)
        - Return google user profile based on Google API.
    - isValidToken(Id)
        - Check if the google access token is valid. If not valid, refresh the token.
    - refreshToken(Id)
        - Refresh the user google access token.
    - getCalendars()
        - Return the list of calendar from user google calendar.
    - confirmCalendar()
        - Save the user-selected calendars id and sync token to the database.
    - pullEvents()
        - Save the future events of the user-selected calendars into the database.
    - getContacts()
        - Save the email addresses, first name, last initial and the type of email address into the database.

2.2. User Interface Design

From the opening of the application (app), a user is presented with only two options; creating an account or signing in.

From here on out a friendly set of characters is established that will be seen in throughout the main interactions of the app.

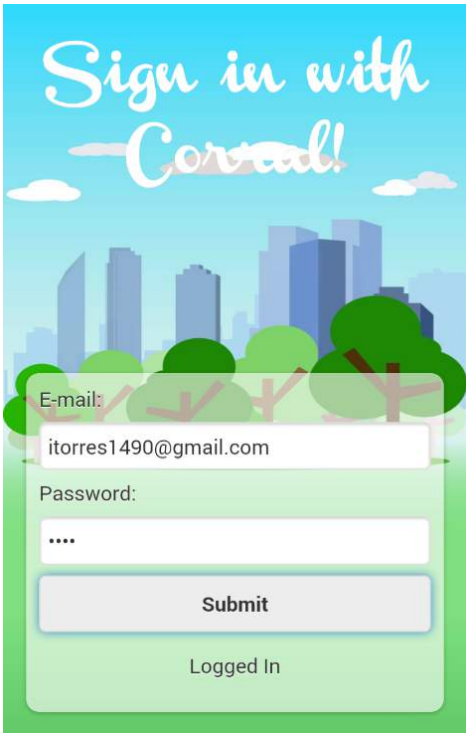
The two buttons seen here are of the little jQuery Mobile UI that remains in the app.



As mentioned before, some of the pages still use the default jQuery Mobile UI originally started the app with.

One of the main issues of when adding our new assets to the project was floating labels and input fields. One of the group members tackled this problem for the sign in page and the creating with Corral page.

There is user feedback provided here by giving form input errors, which are not seen on the right, and status messages, in this case being, “Logged In”.



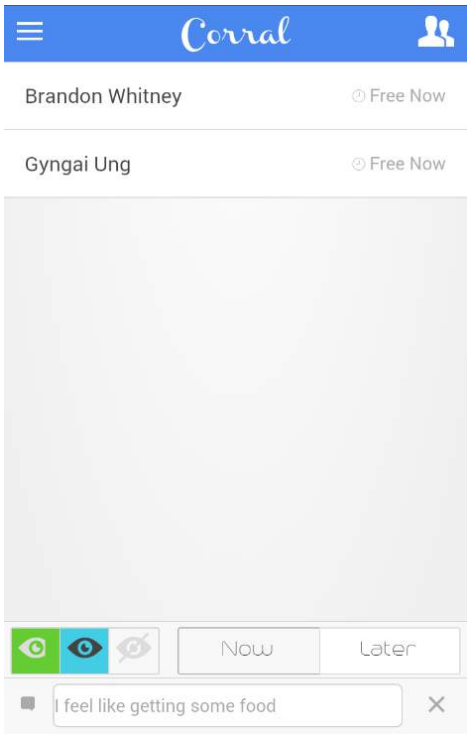
After signing in or creating an account, you can find the majority of the remaining styles of the app here.

Most all data coming forward is list based; either displaying friends, requests, nudges, and friend invites, attempting to keep these as clean as possible.

Swipe interactions coming from the Ionic Framework comes to play here, allowing for the displaying of menu items and lists such as the friends lists.

Instead of adding interactions moving the user away from the main page, the user can specify their mood, at the bottom of the screen to keep clutter down, and switch between viewing friends now or later.

Finally, the three most important buttons are found at the bottom of the app; free, invisibility, and schedule mode. These affect whether or not you appear on your friends availability lists.



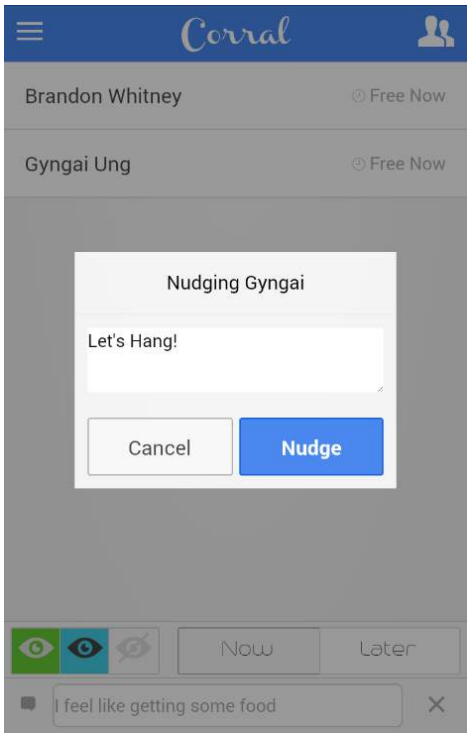
2.3. User Input Preview

All input of data on the main page follows a flat style with a popup as well as the mood input at the bottom of the page.

The colors used for accepting actions, in particular the blue shown to the right, is reminiscent of some of the baby blues that Google uses in some of their UI.

Less obvious actions that can be found are the x and pencil button to update or clear your mood, and the swipes in order to favorite or delete a friend.

The input that is inconsistent throughout each Android device is when the system displays its own date and time pickers, something that would be prefered to be theme, but its a limitation with using the Cordova native interactions.

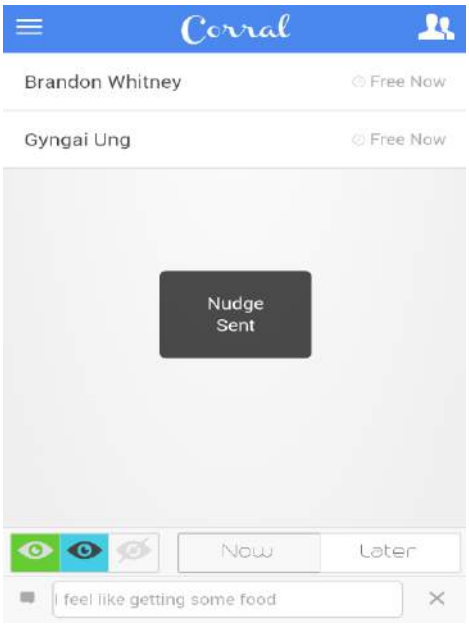


2.4. User Output Preview

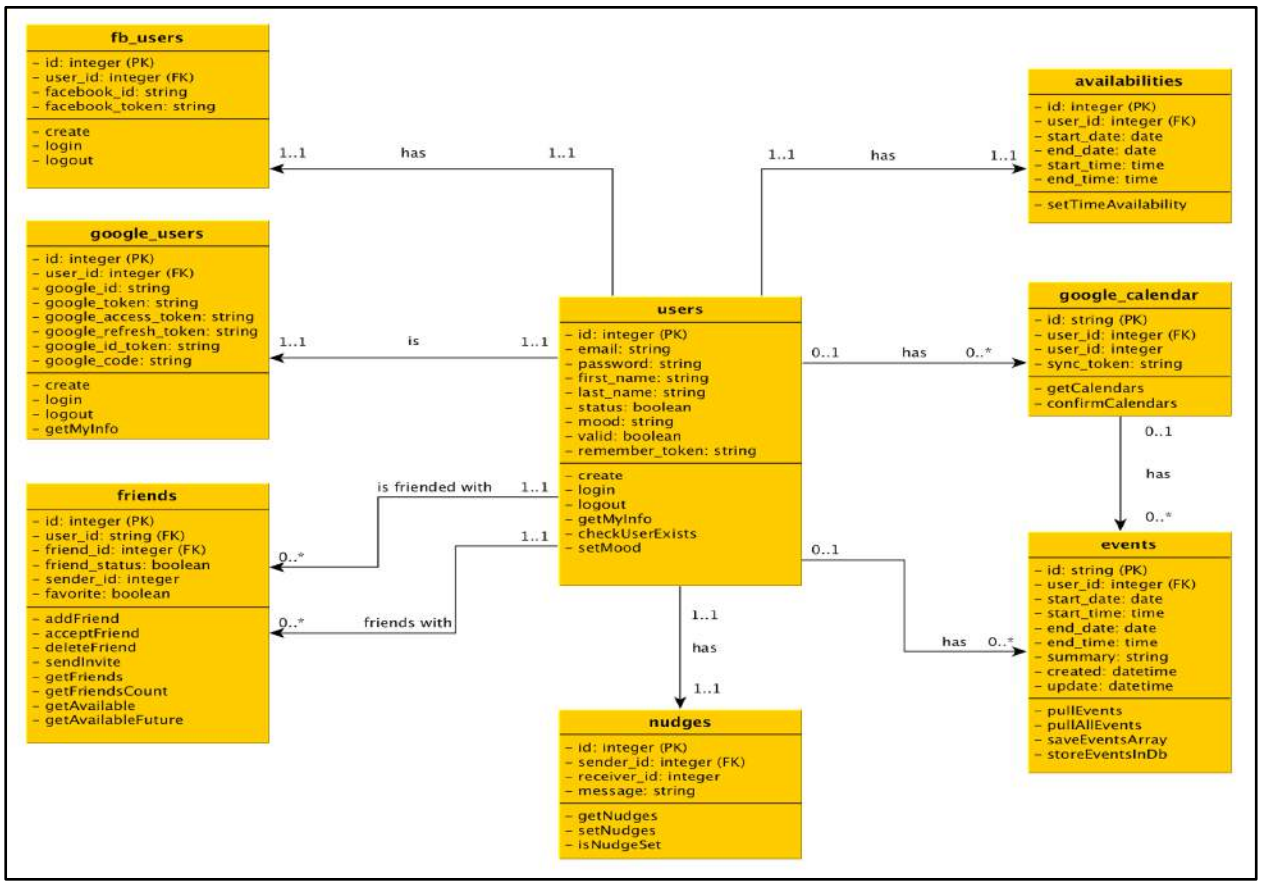
One of the most important things throughout the app is to inform the user of what is happening. Our best response to this is using what Google calls, “toast notifications”.

As seen on the right, its a success dialog of sorts but without any action needed as the notification dismisses itself after a few seconds.

The other confirmation of action that a user can see is when accepting a friend request or denying one, or setting a friend as a favorite. The list automatically refreshes itself to display the change of data, removing the need for a “toast notification” in those cases.



2.5. System Database/File Structure Preview



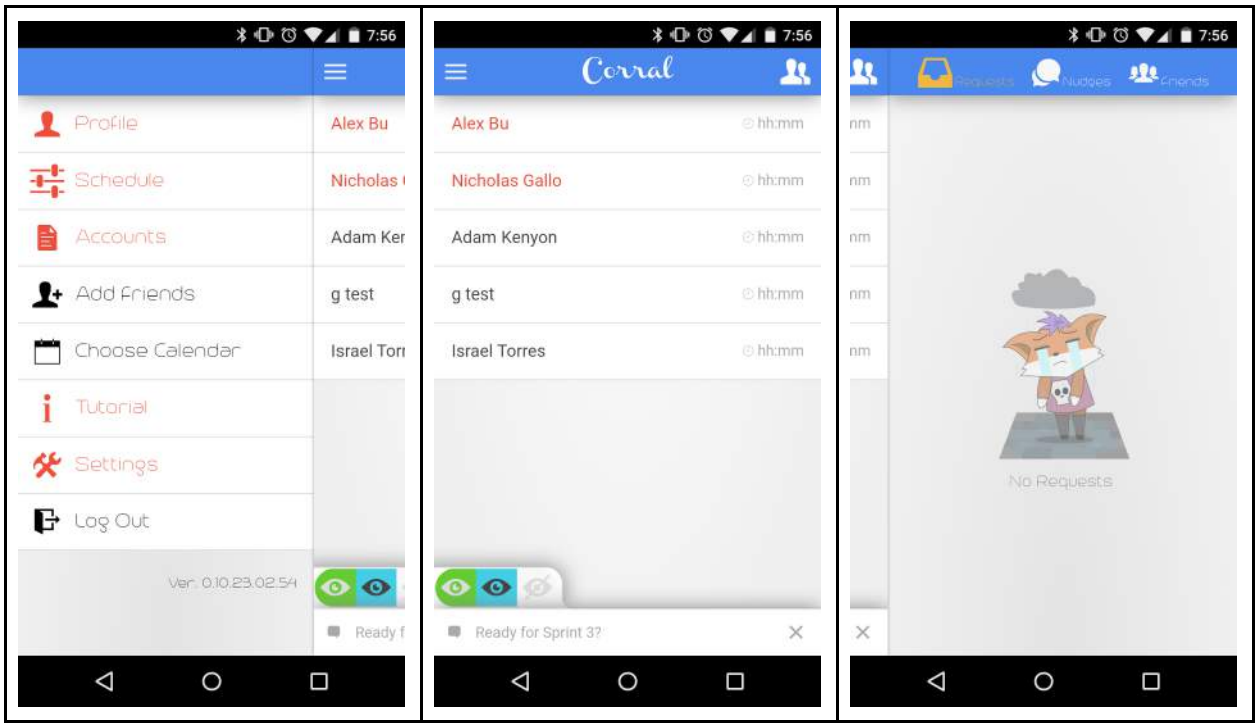
2.6. External and Internal Limitations and Restrictions

- Internal Limitations and Restrictions:
  - Hosting Server down
- External Limitations and Restrictions:
  - Google API down (Sign in, Calendars, and Contacts)
  - Google Mail Server down (invitation)
  - Facebook API down (Sign in, Events)
  - Wi-Fi and phone mobile data connection

2.7. User Interface Specification

Interface Metaphor Model

Using the User interface Metaphor Model that is used in Facebook mobile app and defined in jQuery Mobile UI framework and Ionic framework, Corral adopts the sliding left and right panels design (see *illustration below*) that provide access to all the functionalities of the app by simply sliding to the left or right or the home screen.



Interface Metaphor Model

User Screens/Dialog

See Sections 2.2-2.4

Report Formats/Sample Data

Not applicable

- Online Help Material

  There is currently no online support material for users, but there is a well documented back-end API listing found on http://corral.readme.io for front-end access.

- Error Conditions and System Messages

  Error messages are only displayed when a list refresh fails to fetch its data.

- Control Functions
  - Set availability:
    - Free mode (Forever): Sets user availability to Free for an indefinite amount of time.
    - Free mode (Time): Sets user availability to Free for a user set amount of time.
    - Schedule mode: Sets user availability to follow his/her schedule for an indefinite amount of time.
    - Invisibility mode (Forever): Sets user availability to Invisible for an indefinite amount of time.
    - Invisibility mode (Time): Sets user availability to Invisible for a user set amount of time.
  - Nudge Friend Now:
    - Nudge (Blank): Sends a blank message to the respective users' friend (overwrites previous nudge).
    - Nudge (Message): Sends a short message to the respective users' friend (overwrites previous nudge).
  - Favorite Friend:
    - Adds user to a special group which pushes that users' entry to the top of any list the user requests thereafter.
  - Unfavorite Friend:
    - Removes user from the favorite group, returning them to the original order of which he or she appeared.
  - Accept Request:
    - Adds the user associated with the request to the current user (and requesting users') friend list.
  - Deny Request:
    - Deletes the pending request from the current user.
  - Mark Nudge Seen:
    - Removes nudge from the receiver's nudge list.
  - Get friends now:
    - Shows all friends that are available at the current time (can be nudged and can view friends' mood).
  - Get friends later:
    - Shows all friends that are available at the inputted date and time.
  - Update Mood:

    - User can update his/her mood to express what he/she is eager to do.
- Clear Mood:
    - Removes current user mood.
- Import Calendars:
    - Uploads the users' respective calendars from Google to Corral's database.
- Add/Invite friend:
    - Sends either an email to the friend (if he/she isn't registered with Corral) or sends a friend request.
- Add/Invite from Phone Contacts:
    - Generates a list of friends from the users' phone contacts which can be invited or requested (See function Add/Invite friend).
- Add/Invite from Google Contacts:
    - Generates a list of friends from the users' Google contacts which can be invited or requested (See function Add/Invite friend).

3. System Performance Requirements

3.1. Efficiency

- Ease of Use
  - The app follows today's standards of interactivity. Consumers today expect an interface that responds to swiping left and right for menus and swiping down for updates to a feed.
  - Buttons for the menus are also implemented to help those not familiar with swiping to get to menus
  - Every action in the app includes some sort of visual feedback to alert the user that their action was processed

- Load Times and Query Times
  - The app runs at a decent speed, taking no longer than 5 seconds without visual feedback to the user. Most queries are finished within 2~5 seconds but longer ones (such as loading phone or google contacts) will return partial results as they go.

3.2. Reliability

- Issues with 3rd party APIs
  - the front end is written mostly in HTML and JavaScript using the Cordova API to port to mobile devices. Should the Cordova API change significantly the app may not function properly
  - The back end is written mostly in PHP using the Laravel Framework. Changes to Laravel may also lead to a loss of app functionality
  - Interacting with Google is reliant on their API and any changes to it or their privacy policy may lead to a loss of functionality
  - Interacting with Facebook is reliant on Facebook's privacy policy and changes to the OpenFB API that are being used.

- Error/Failure Detection and Recovery
  - When an error occurs in the app, the error is documented in log files stored on the server.
  - The app is designed to hide errors from the user and wait for another user interaction before proceeding. This way the app doesn't actually crash.

- Allowable/Acceptable Error/Failure Rate
  - With a decent internet connection, the app should only fail 1 in 300 cases at most. The app should be stable and run smoothly in all other cases.

3.3. Security

- Hardware Security
  Not applicable

- Software Security
    - Every page in the app runs a quick authentication check to make sure that the person accessing the page is authenticated and has a valid session.
    - User sessions are tracked on the phone and in the database until they physically press sign out or they clear their cache.
- Data Security
    - Every function in the backend that handles sensitive data checks for authentication from the requester before returning results.
    - All passwords and important authentication keys are hashed based on a random salt value in the database.
- Execution Security (user validation)
    - Most of the authentication employs heavy usage of the Laravel Auth library. It is how the app signs in, signs out, and verifies whether or not a user's session is still active.

3.4. Maintainability

- The costs for maintaining a server for Corral.
- Corral is written in HTML, CSS, JavaScript, and PHP.
- Each feature is encapsulated and the code is loosely coupled.
- The SQL Relational Database can easily be moved from MySQL to another SQL-based relational database like MariaDB.

3.5. Modifiability

- **Cordova**
    - Cordova is able to integrate with other JavaScript Libraries such as jQuery.
    - Cordova is able to integrate with Native Android Plugins which also for features that cannot be done with HTML or Javascript.
- **Laravel MVC Model**
    - Laravel allows for easy to add routes and functions to existing controllers.
    - Laravel just need to add new controllers for any new features.
    - Laravel uses JSON for transferring data, which makes it very modifiable.

3.6. Portability

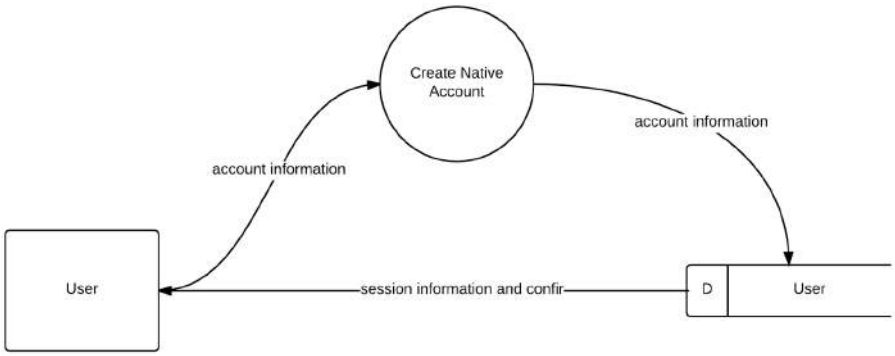
- Google Play Store Costs need to be accounted for if the application is put on the store for download.
- There are some device specific issues that prevents certain features, such as notifications, from working
- Cordova allows for ease of platform migration, much of the application written for Android can easily be exported for iOS and Windows Phone.
- Corral requires Android 3.5.1 as the minimum version.

4. System Design Overview

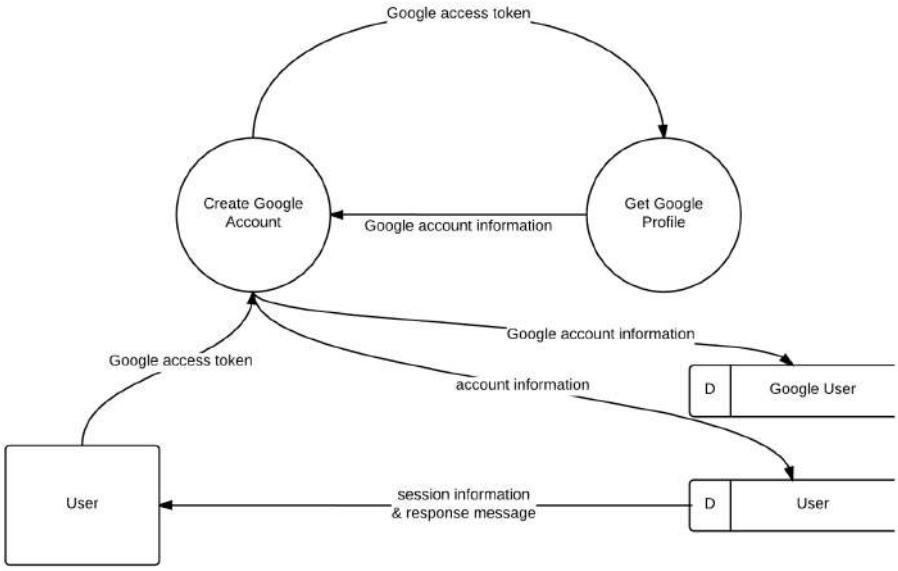
4.1. System Data Flow Diagrams

- User

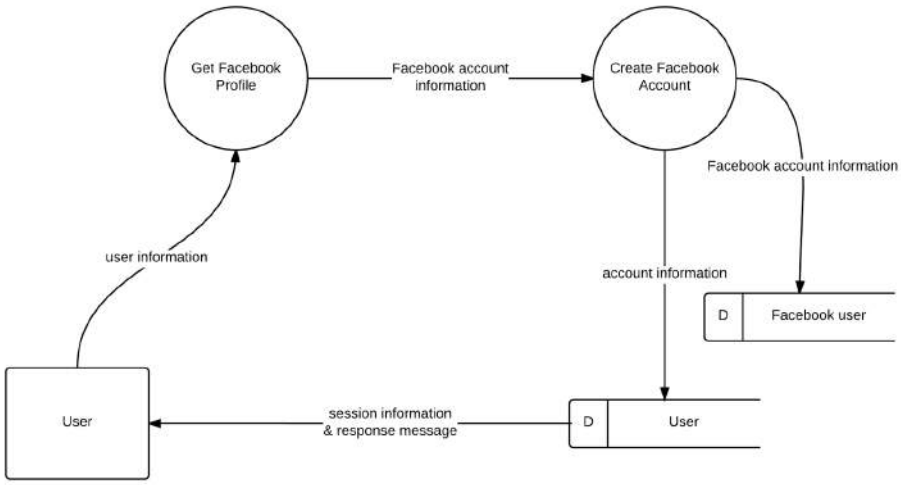
Account Creation - Native Account



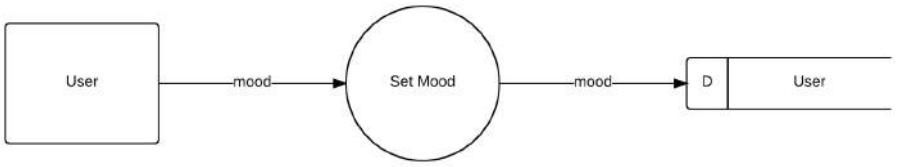
Account Creation - Google Account



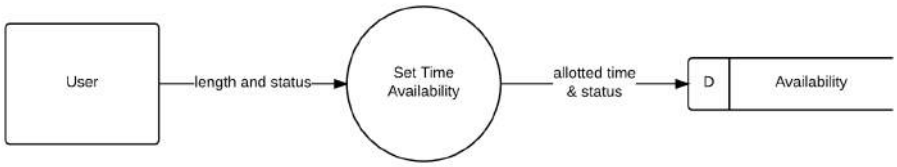
Account Creation - Facebook Account



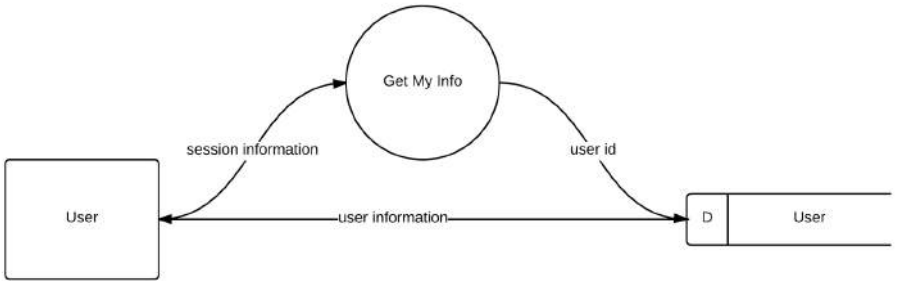
Set Mood



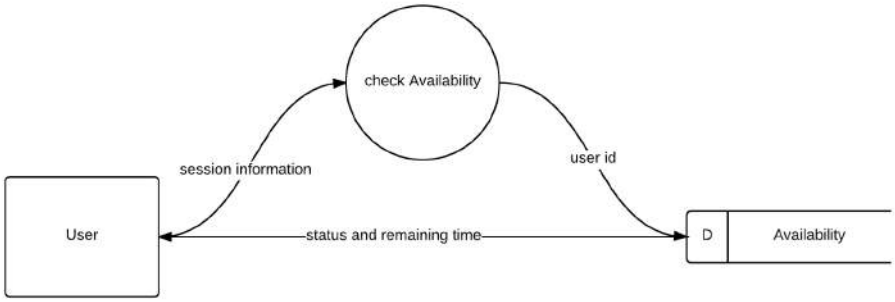
Set Availability



User Information

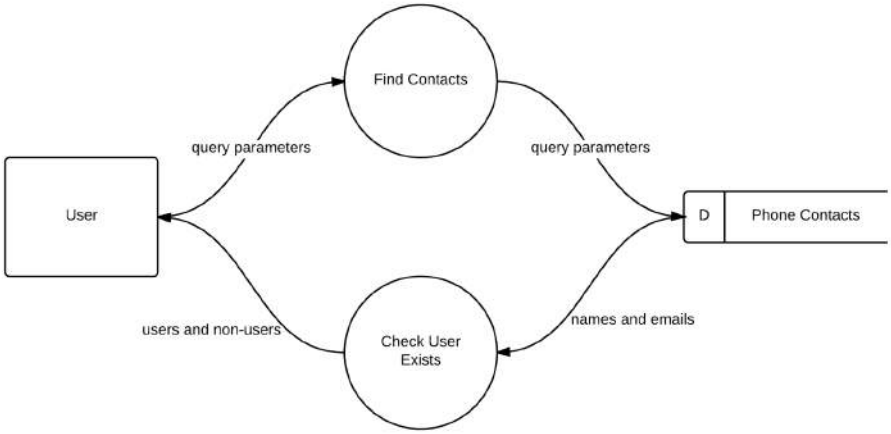


Check Availability

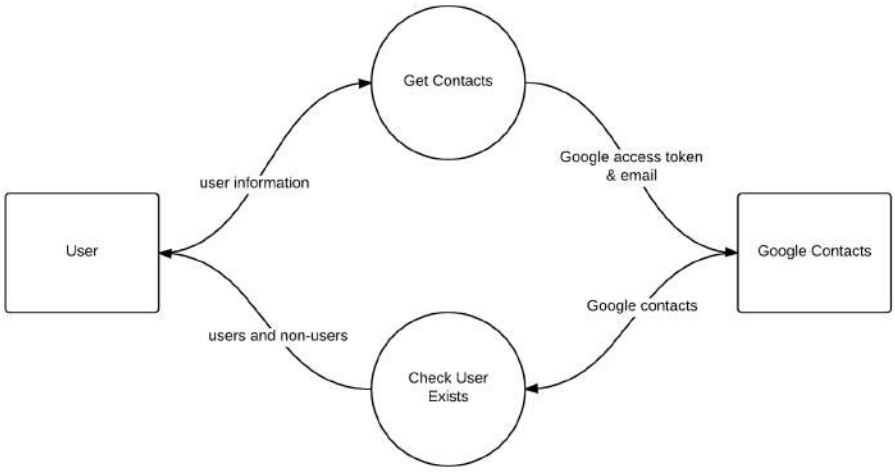


- Friend

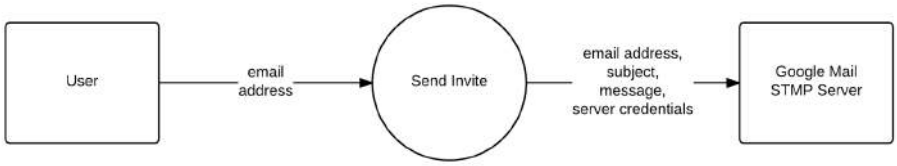
  Get Contacts - Phone Contacts



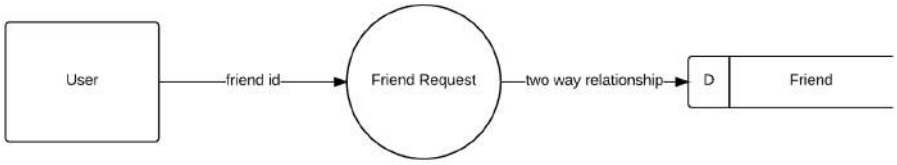
Get Contacts - Google Contacts



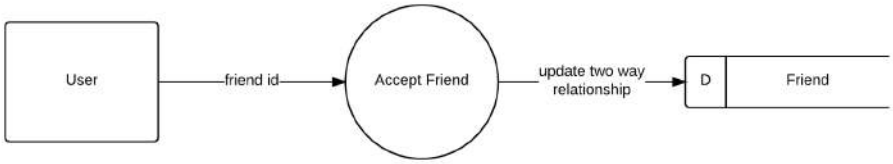
Friend Invite



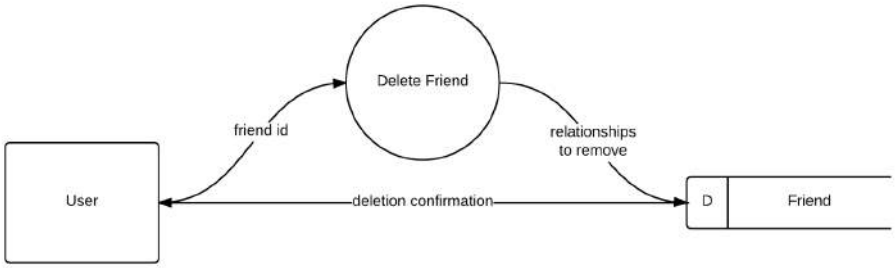
Friend Request



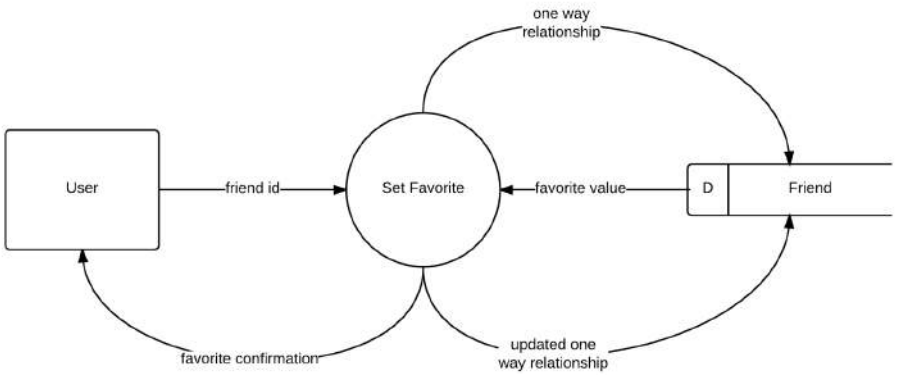
Accept Friend



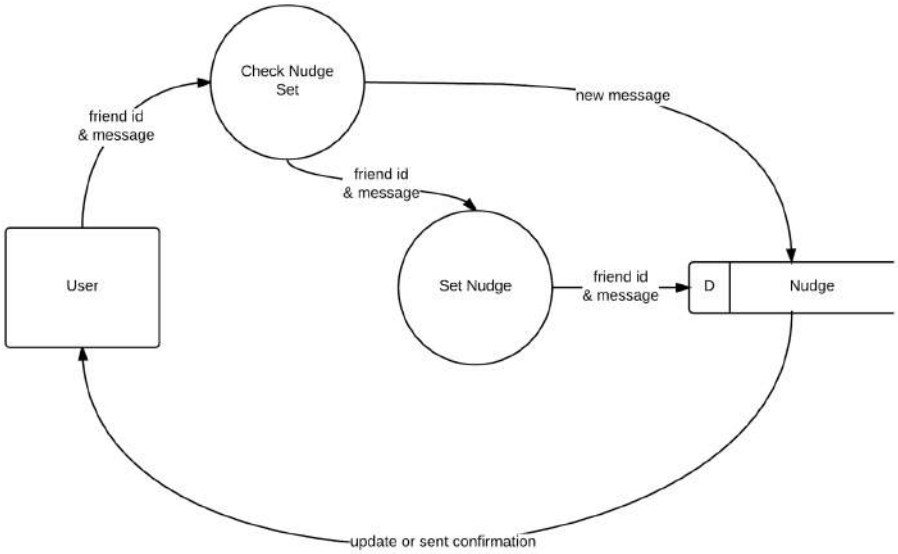
Delete Friend



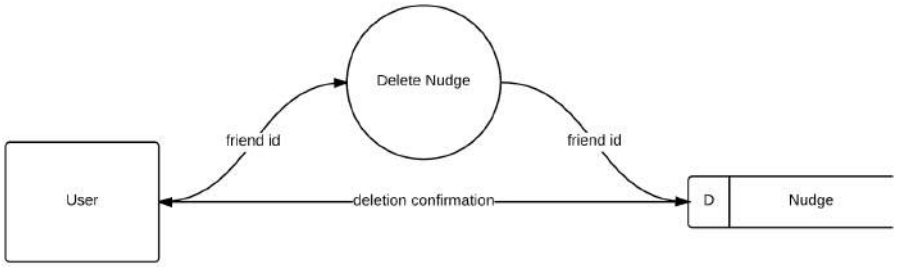
Mark Favorite Friend



Send Nudge

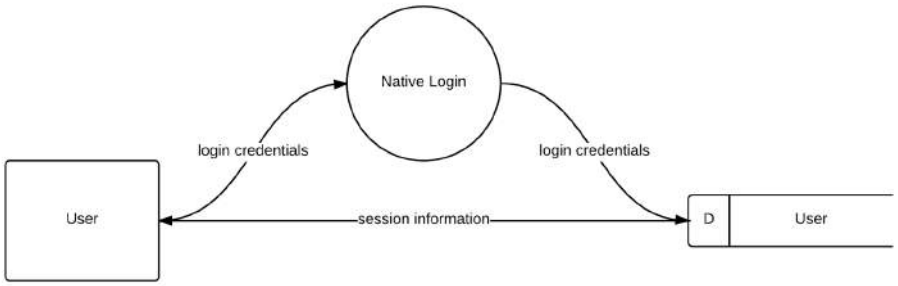


Delete Nudge

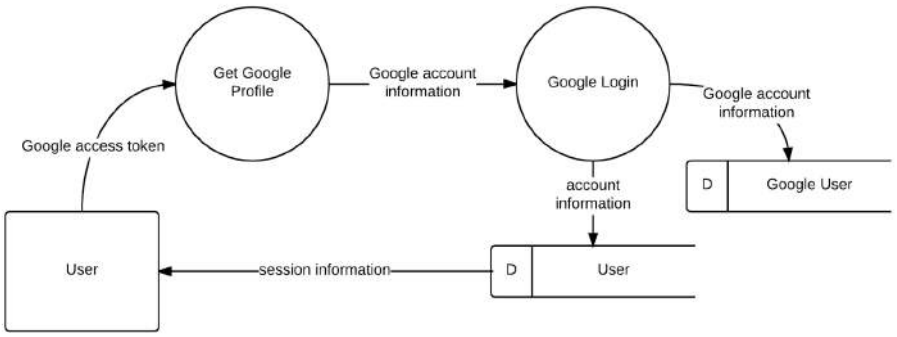


- Session

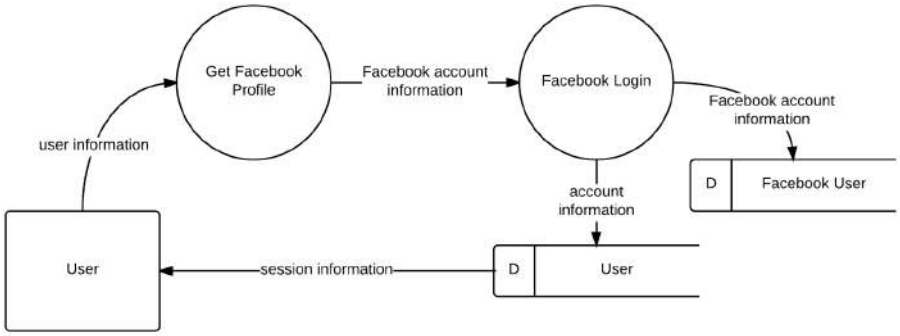
  Native Login



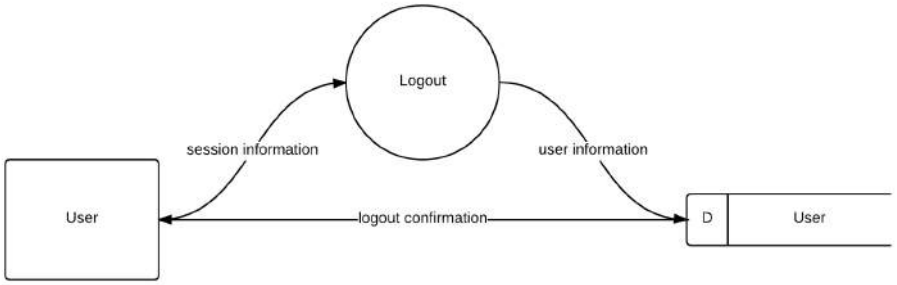
Google Login



Facebook Login

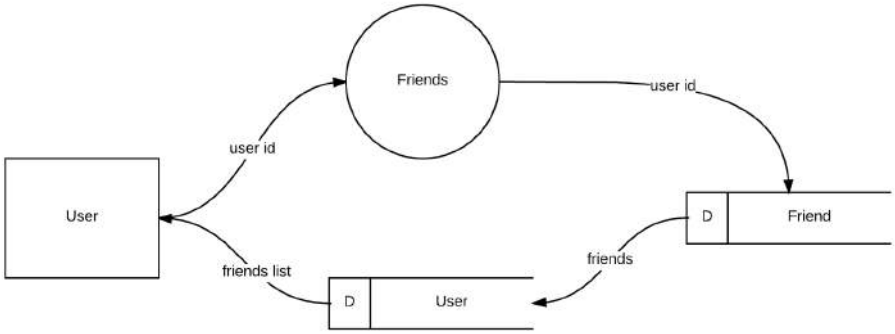


Logout

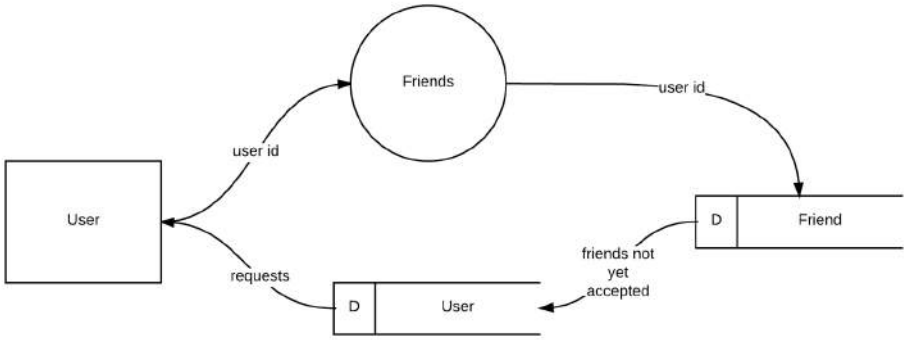


- Lists

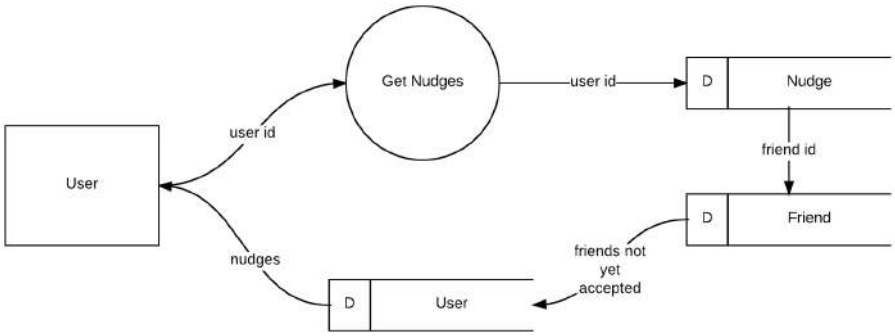
Friends



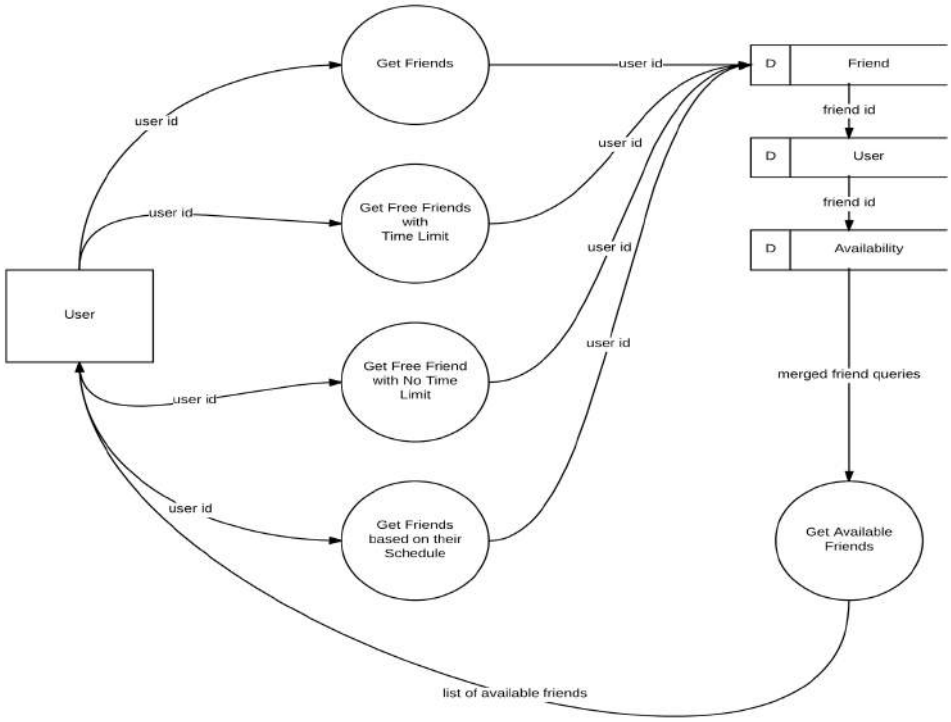
Requests



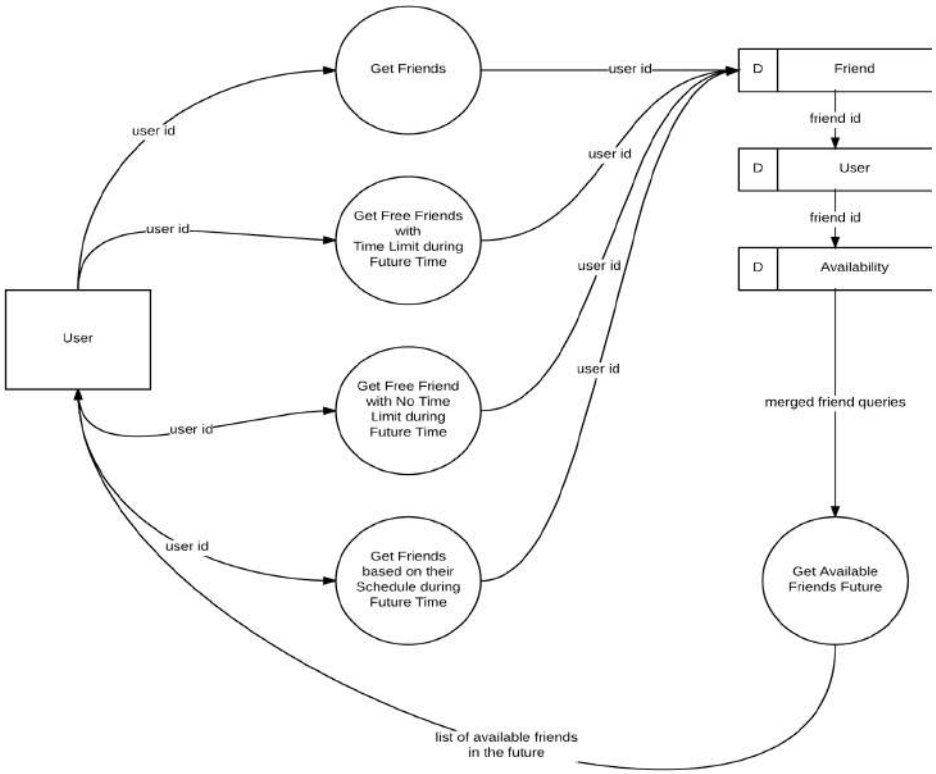
Nudges



Available Friends Now

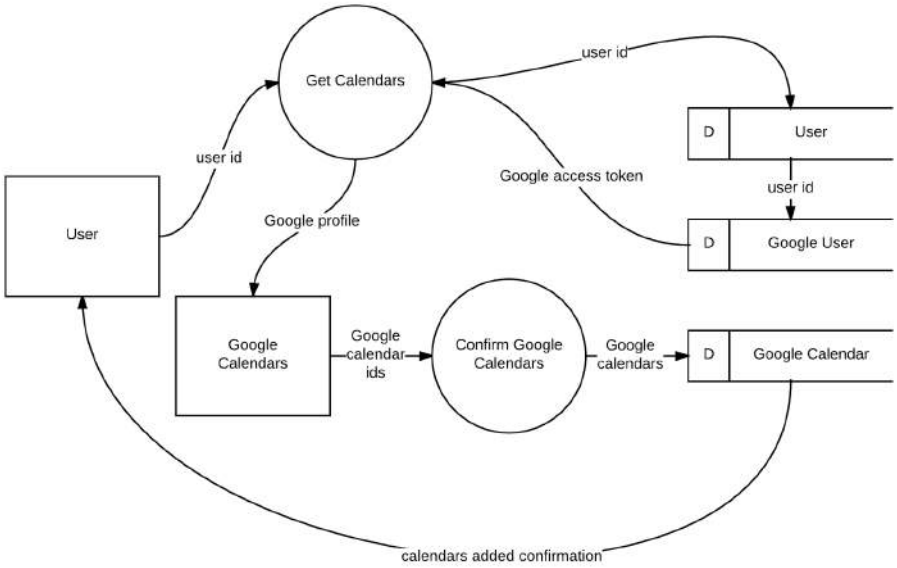


Available Friends Future

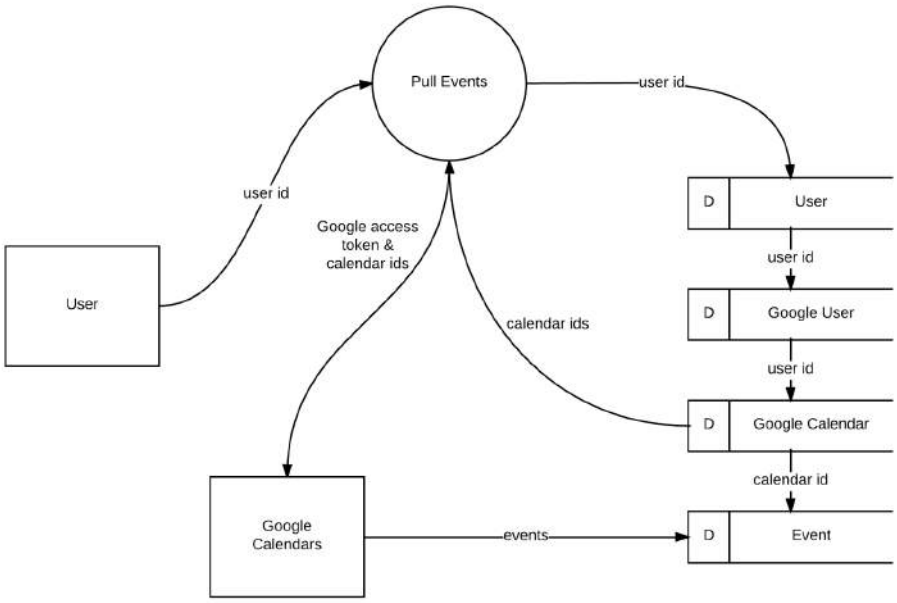


- Schedule

Get Calendars



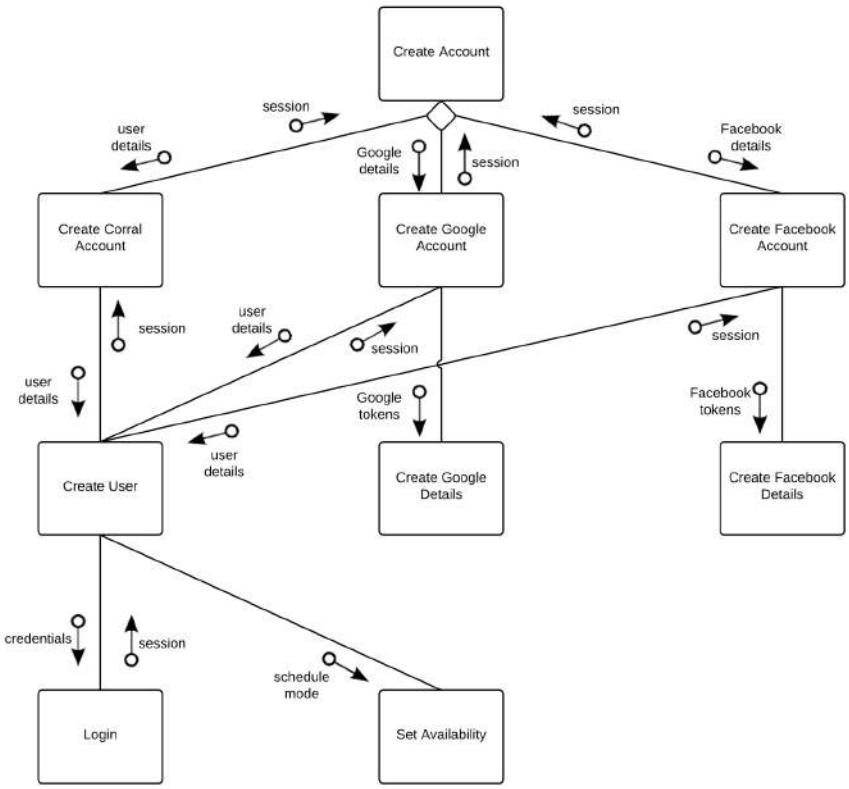
Get Events



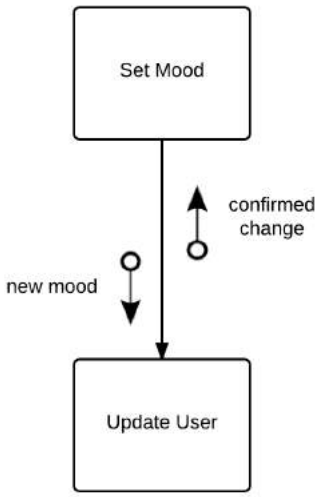
4.2. System Structure Charts

- User

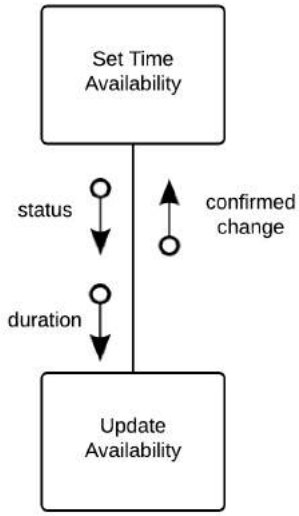
Account Creation



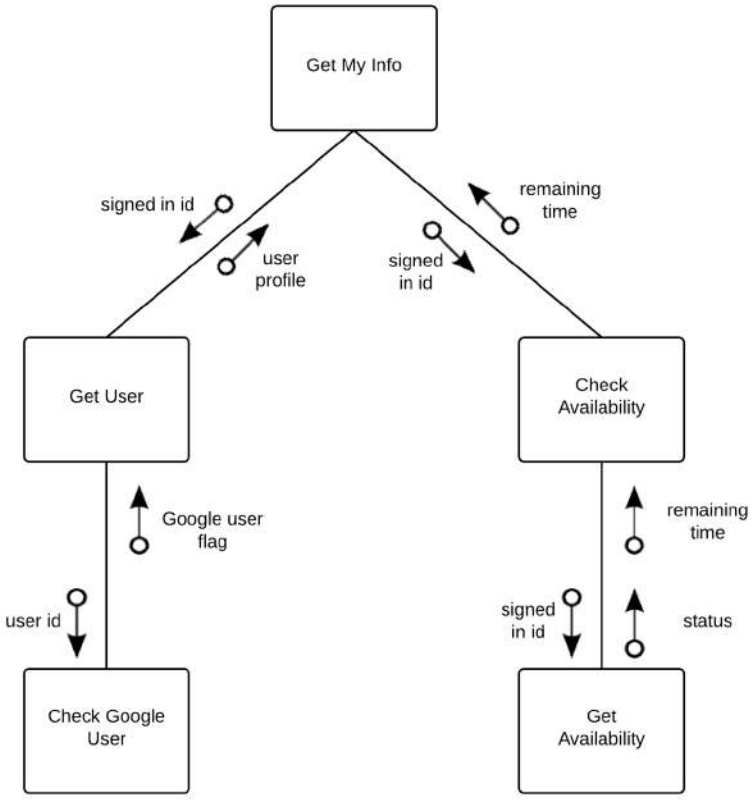
Set Mood



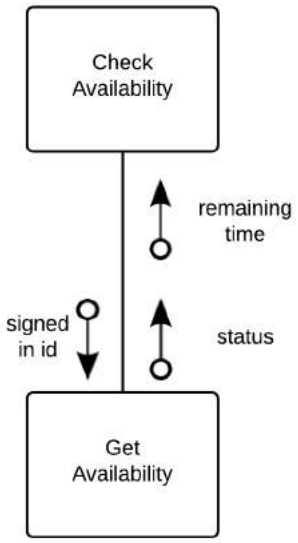
Set Availability



User Information

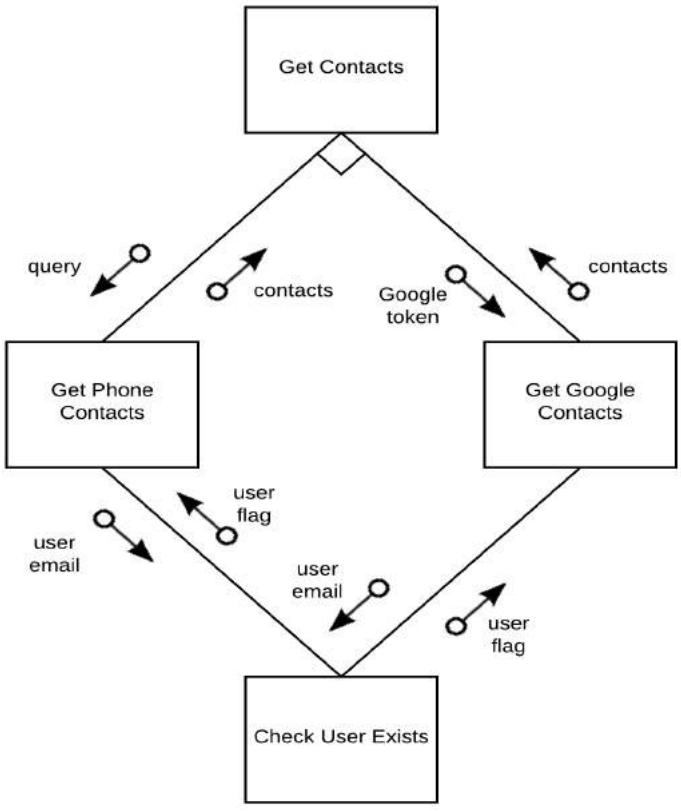


Check Availability

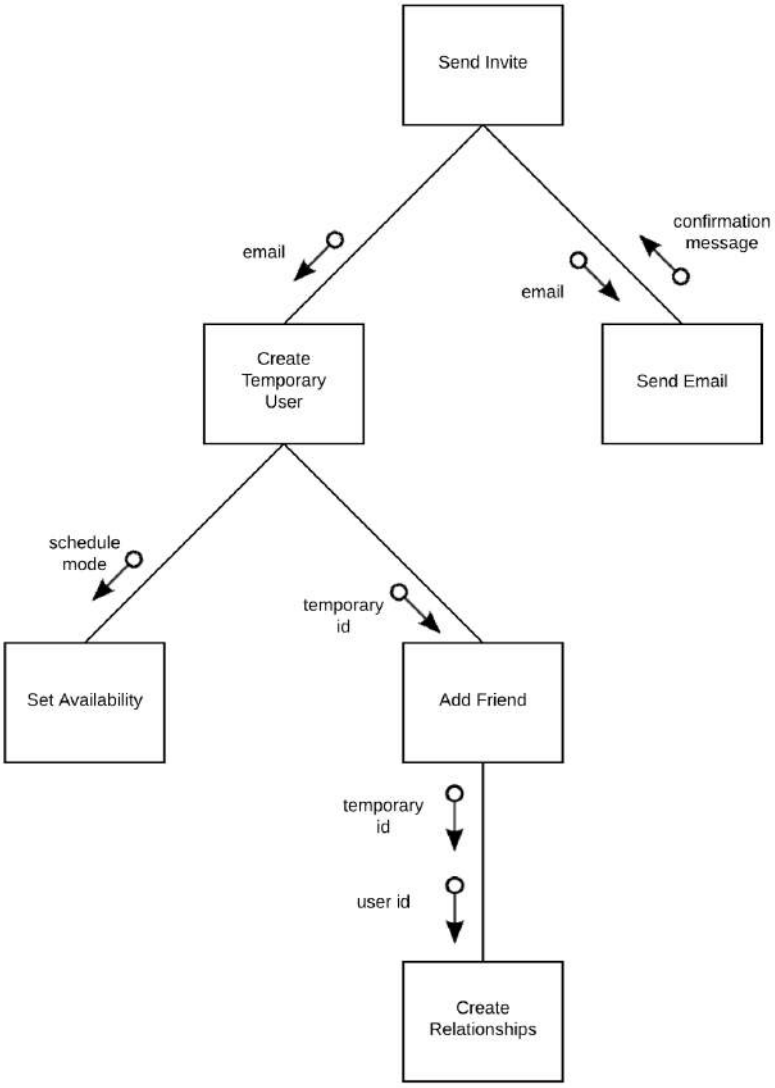


- Friend

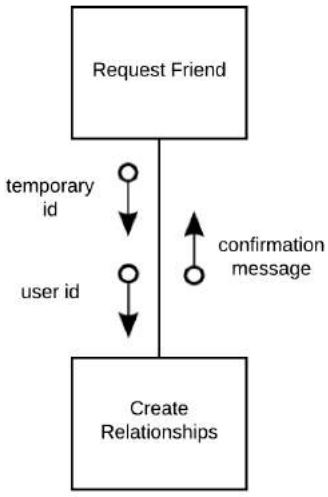
  Get Contacts



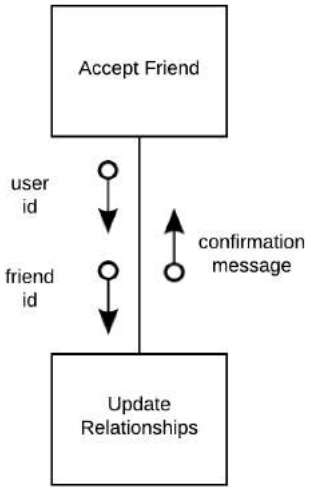
Friend Invite



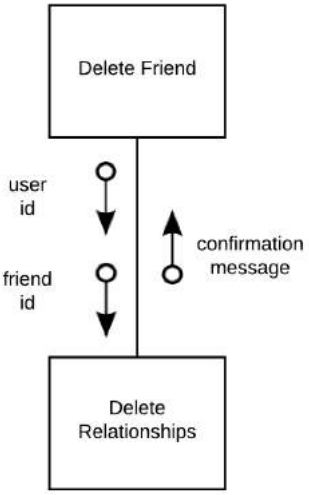
Friend Request



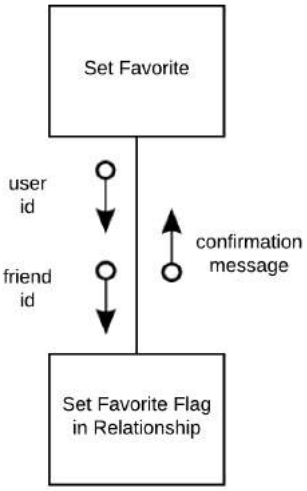
Accept Friend



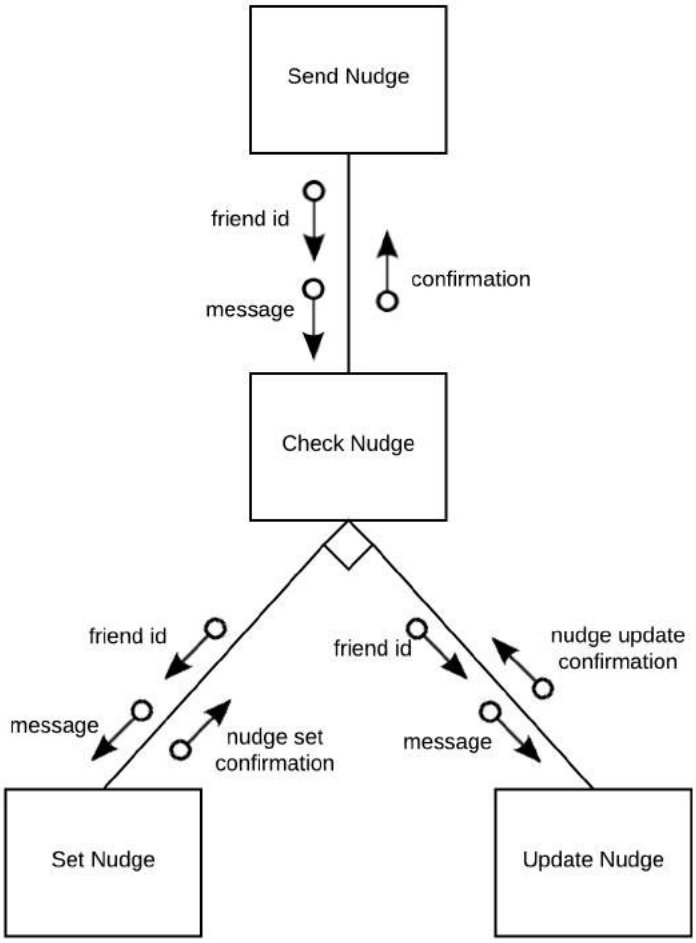
Delete Friend



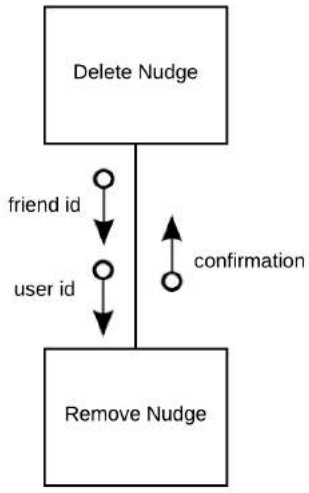
Mark Favorite Friend



Send Nudge

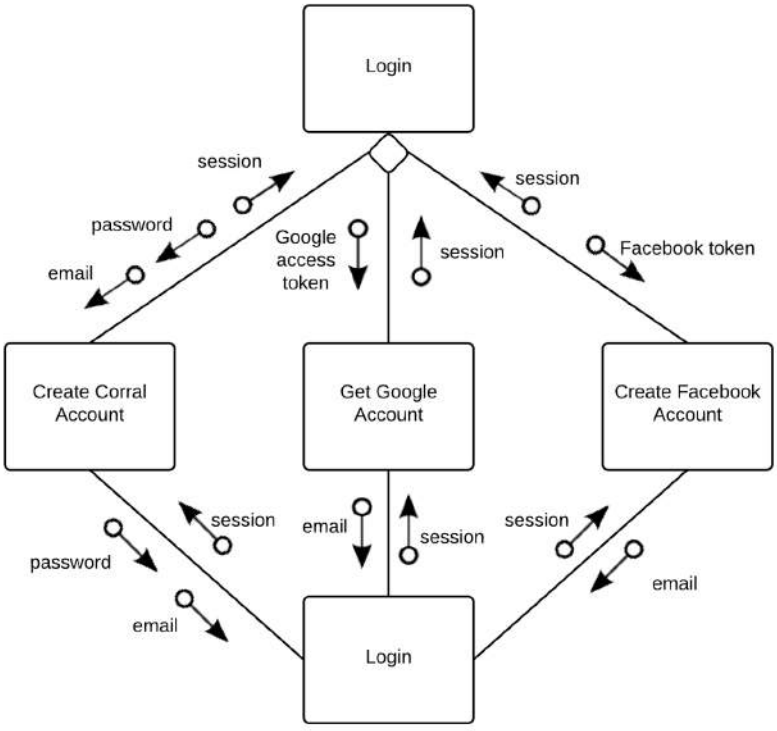

Send Nudge
friend id
message
confirmation
Check Nudge
friend id
message
nudge set
confirmation
friend id
message
nudge update
confirmation
Set Nudge
Update Nudge

Delete Nudge

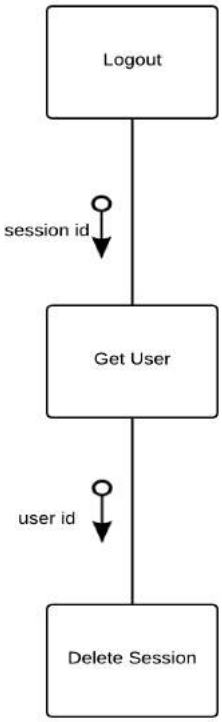


- Session

  Login

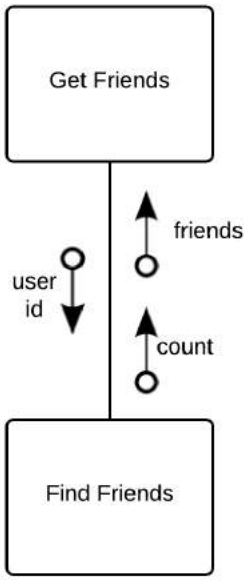


Logout

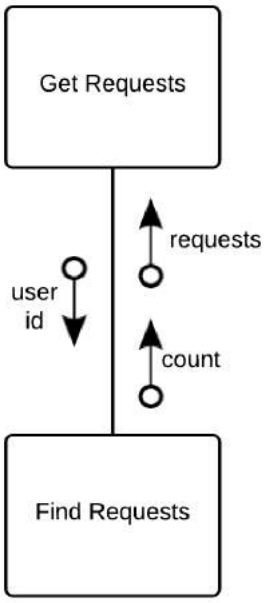


- Lists

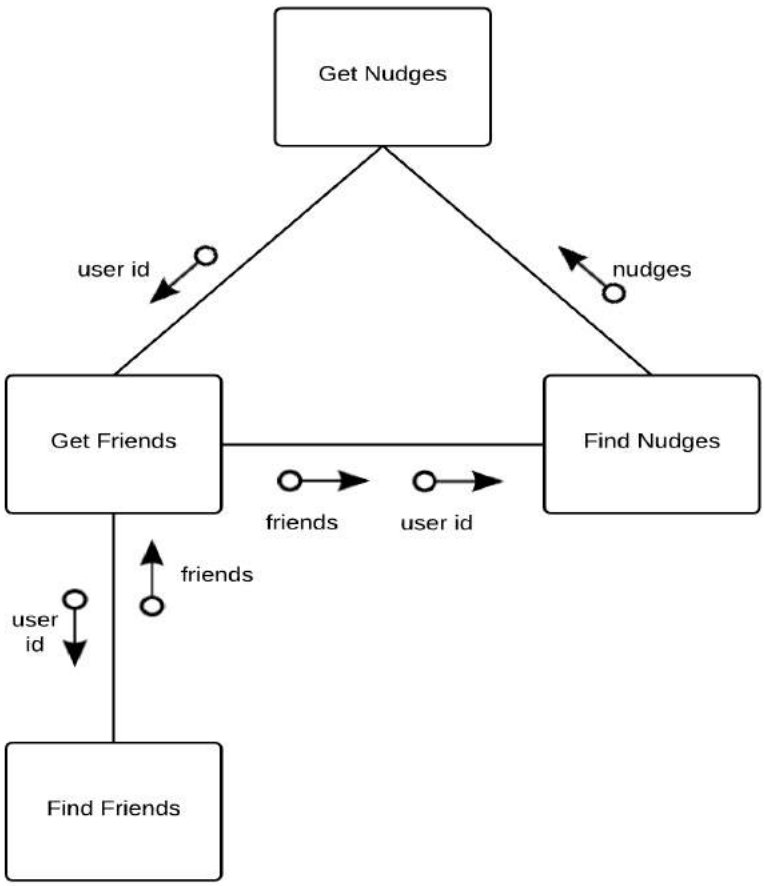
  Friends



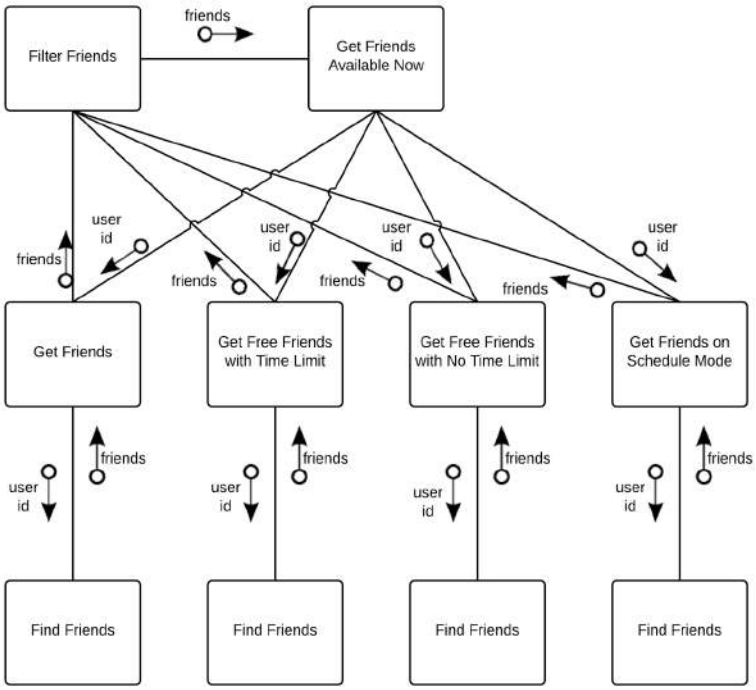
Friend Requests



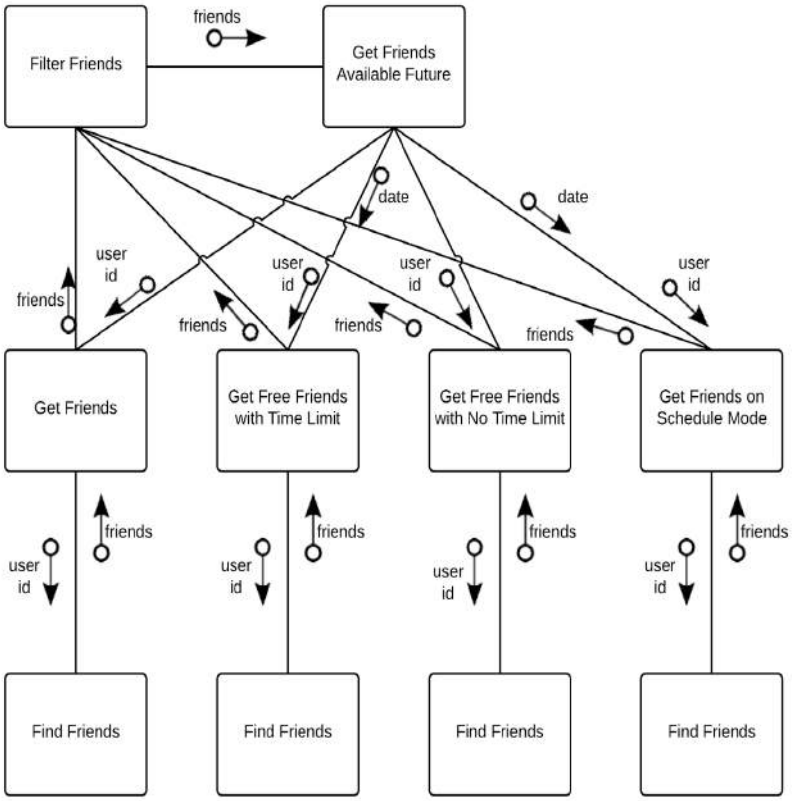
Nudges



Available Friends Now

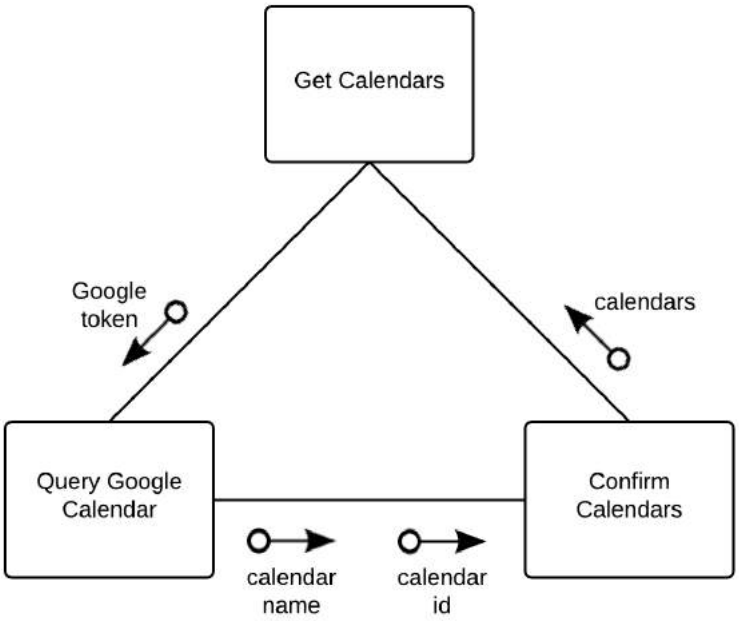


Available Friends Future

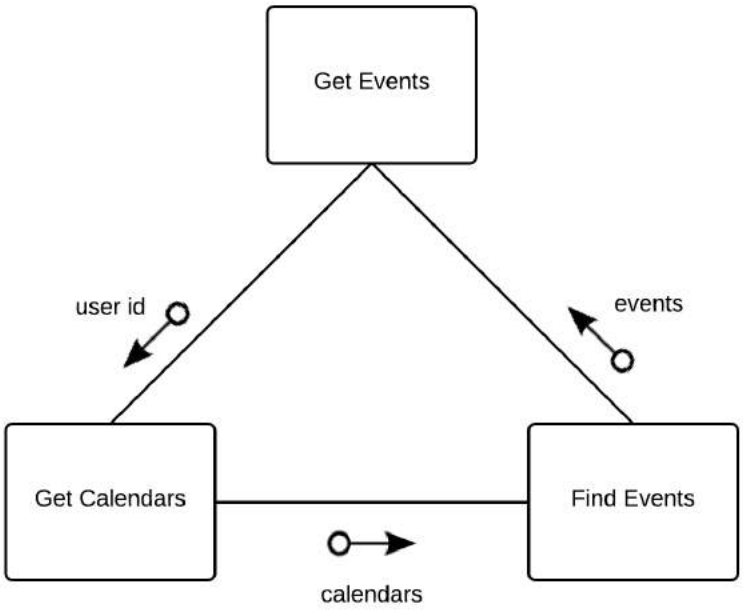


- Schedule

  Get Calendars



  Get Events



4.3. System Data Dictionary

Availability				
Element	Type	Length	Description	Example
id	Integer	11	Primary key	1
user_id	Integer	11	Foreign key from user table	1
start_time	Time	N/A	Start time of an availability period	08:00:00 13:30:00
end_time	Time	N/A	End time of an availability period	08:00:00 13:30:00
start_date	Date	N/A	Start date of an availability period	2014-12-01
end_date	Date	N/A	End date of an availability period	2014-12-01
status	Integer	11	Availability status of the user. 0 - not available, 1 - based on schedule, 2 - free	1
created_at	DateTime	N/A	Record creation date and time stamp	2014-10-17 12:08:14
updated_at	DateTime	N/A	Record last update date and time stamp	2014-10-17 12:08:14

Event				
Element	Type	Length	Description	Example
id	Integer	11	Primary key	1
user_id	Integer	11	Foreign key from user table	1
calendar_id	Varchar	variable	ID	1
start_date	Date	N/A	Start date of an availability period	2014-12-01
start_time	Time	N/A	Start time of an event	08:00:00 13:30:00
end_date	Date	N/A	End date of an availability	2014-12-01

			period	
end_time	Time	N/A	Start end time of an event	08:00:00 13:30:00
summary	Varchar	255	Name of an event	Birthday Party
created	DateTime	N/A	Record creation date on Google's server	2014-10-17 12:08:14
updated	DateTime	N/A	Record last update date on Google's server	2014-10-17 12:08:14
created_at	DateTime	N/A	Record creation date and time stamp	2014-10-17 12:08:14
updated_at	DateTime	N/A	Record last update date and time stamp	2014-10-17 12:08:14

Facebook				
Element	Type	Length	Description	Example
id	Integer	11	Primary key	1
user_id	Integer	11	Foreign key from user table	1
facebook_id	Varchar	variable	User ID on Facebook	gvdfbg213
facebook_token	Varchar	variable	Token that allows Corral to access information from a user's Facebook Account	21231sdfsd231
created_at	DateTime	N/A	Record creation date and time stamp	2014-10-17 12:08:14
updated_at	DateTime	N/A	Record last update date and time stamp	2014-10-17 12:08:14

Friend				
Element	Type	Length	Description	Example
id	Integer	11	Primary key	1
user_id	Integer	11	Foreign key from user table	1
friend_id	Integer	11	Foreign key from user table	1

friend_status	Integer	1	Approved friendship	1
sender_id	Integer	11	Person who sent friend request	11
favorite	Integer	1	Record that a user is favorited	1
created_at	DateTime	N/A	Record creation date and time stamp	2014-10-17 12:08:14
updated_at	DateTime	N/A	Record last update date and time stamp	2014-10-17 12:08:14

Google Calendar				
Element	Type	Length	Description	Example
id	Integer	11	Primary key	1
user_id	Integer	11	Foreign key from user table	1
sync_token	Varchar	255	Token that detects if a calendar state has changed	afhsdfhs4683
created_at	DateTime	N/A	Record creation date and time stamp	2014-10-17 12:08:14
updated_at	DateTime	N/A	Record last update date and time stamp	2014-10-17 12:08:14

Google User				
Element	Type	Length	Description	Example
id	Integer	11	Primary key	1
user_id	Integer	11	Foreign key from user table	1
google_id	Varchar	255	User ID on Google	gvdfbg213
google_access_token	Varchar	255	Token that allows Corral to access information from a user’s Google Account	21231sdfsd231
google_refresh_token	Varchar	255	A token to replace the Google Access Token	sfsdfsdf465415

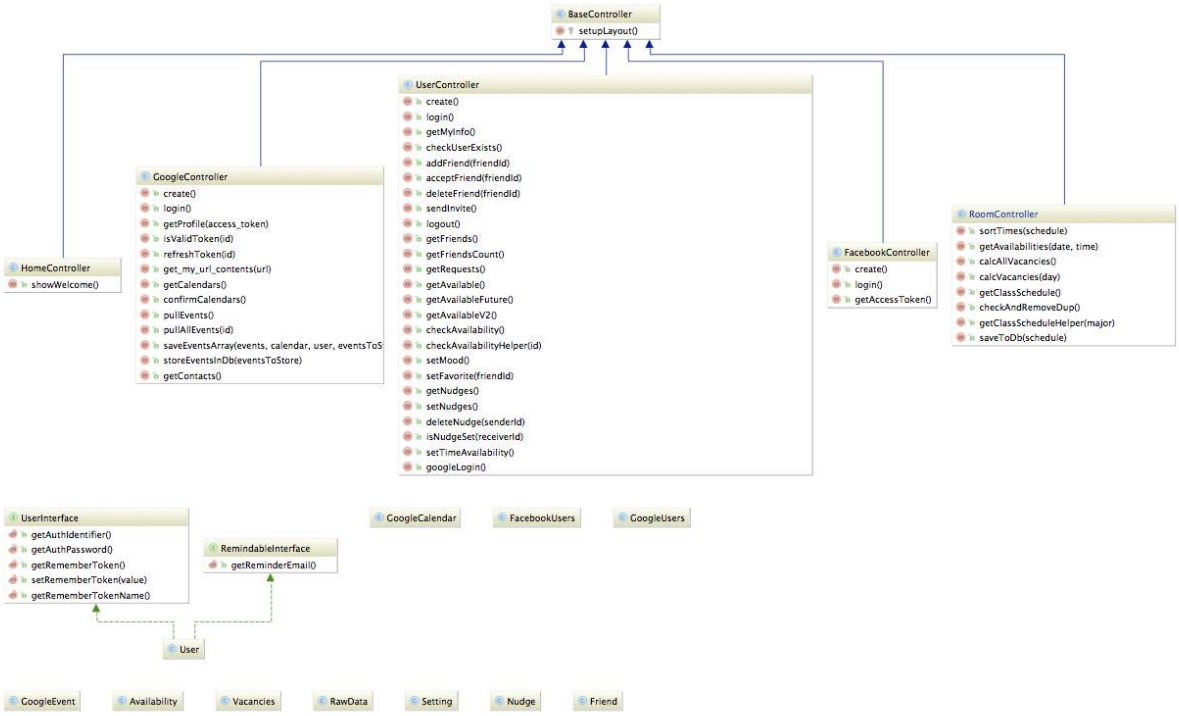
created_at	DateTime	N/A	Record creation date and time stamp	2014-10-17 12:08:14
updated_at	DateTime	N/A	Record last update date and time stamp	2014-10-17 12:08:14

Nudge				
Element	Type	Length	Description	Example
id	Integer	11	Primary key	1
sender_id	Integer	11	Foreign key from user table	1
receiver_id	Integer	11	Foreign key from user table	1
message	Varchar	50	Text associated with a nudge	You have been nudged
created_at	DateTime	N/A	Record creation date and time stamp	2014-10-17 12:08:14
updated_at	DateTime	N/A	Record last update date and time stamp	2014-10-17 12:08:14

Setting				
Element	Type	Length	Description	Example
id	Integer	11	Primary key	1
source	Varchar	50	String that stores external APIs being used	Google
client_id	Varchar	255	Public ID needed to make calls to APIs	dfgsh456
client_secret	Varchar	255	Private ID needed to make calls to APIs	nb1231321e
created_at	DateTime	N/A	Record creation date and time stamp	2014-10-17 12:08:14
updated_at	DateTime	N/A	Record last update date and time stamp	2014-10-17 12:08:14

User				
Element	Type	Length	Description	Example
id	Integer	11	Primary key	1
first_name	Varchar	100	First name of the user	John
last_name	Varchar	100	Last name of the user	Doe
email	Varchar	100	E-mail the user uses to sign up and identify themselves	example@email.com
mood	Varchar	50	Text to describe a user's mood	I am busy
remember_token	Varchar	255	Token for keeping a user signed in even after exiting the app	gyesbh54654
valid	Tinyint	1	Record that a user is valid	1
password	Varchar	255	Hashed password for a registered user	#!@RFe4huZ4
created_at	DateTime	N/A	Record creation date and time stamp	2014-10-17 12:08:14
updated_at	DateTime	N/A	Record last update date and time stamp	2014-10-17 12:08:14

4.4. System Internal Data Structure Preview

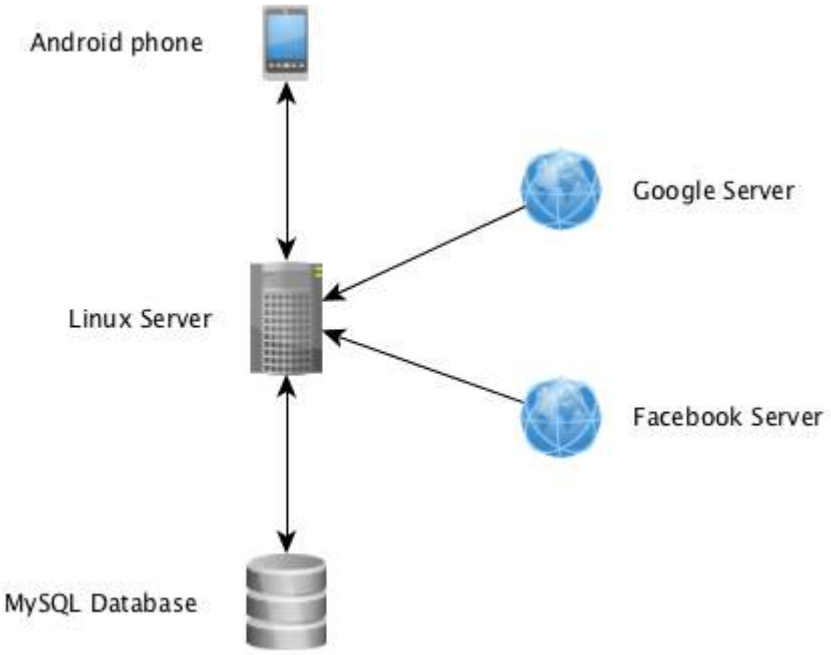


4.5. Description of System Operation

Corral is developed using the open source PHP web application framework, designed for the development of model-view-controller (MVC) web applications. Laravel provides RESTful Routing, which allows for separation of the logic behind serving HTTP GET and POST requests for Corral app APIs.

Laravel also provides Eloquent Object-Relational Mapping (ORM that helps enforce constraints to the relationships between database objects, which protects database relational integrity.

4.6. Equipment Configuration



The core of the app lives on a shared hosting account running Linux CentOS. Along with that is a dedicated MySQL database that is allocated to a partitioned server. For the front-end of the app, the code lives in a WebView wrapper in native Android Java code, running on Android phones that support versions 3.5.1 and above. And external to the app is the server's that belong to Google and Facebook of which pulls specific data from to enhance the app.

4.7. Implementation Languages

- PHP: Hypertext Preprocessor (PHP)

  With PHP's general purpose use, the back-end API can be programmed fairly quickly, allowing to iterate over old code when needed, also allowing for new controllers to be added on the fly. It was also a language that roughly half of the group was already familiar with.
- My Structured Query Language (MySQL)

  MySQL was the first choice because the of the familiarity most of the group had in CECS 323: Database Fundamentals. Going this route instead of going with the more trendy databases like Not Only SQL (NoSQL), came about because there was no way to abstract the objects to be their own encapsulating entities, and there clearly is a need to have established relationships between users and their pertaining objects.

- HyperText Markup Language (HTML)

Using Cordova allowed the writing of all the front-end code to reside on the phone with HTML. Everyone on the time was already familiar with it, and since it is only a markup language, learning new HTML 5 technologies were non-trivial. On top of that, as long as the HTML is written correctly, all the code can be viewed the same on all platforms, allowing for quick iterations of the front-end layout.

- JavaScript (JS)

  Along with the HTML, Cordova also allowed the writing of native JS that could be run without the use of an emulator, allowing for quick debugging and fine tuning of the code. Cordova did also introduce device specific libraries, but the main purpose for choosing Cordova and JS, was that it is a simple language that did not require compiling and could be tested very rapidly.

4.8. Required Support Software

- File Transfer Protocol (FTP) Client
  - Filezilla
  - Cyberduck
  - Coda
- Text Editors
  - Sublime Text
  - Coda
  - Notepad++
  - PHPStorm
- MySQL Database Manager
  - Sequel Pro
  - Coda
- Git: Version Control System
  - SourceTree
  - Command Line
- Cordova Application Packaging
  - Cordova Command-line interface (Cli)

5. System Data Structure Specifications

5.1. Other User Input Specification

Sign-In

Native Account

- E-mail
  - Source: User
  - Input Device: Phone Keyboard
  - Data Format: String
  - Example: “example@email.com”
- Password
  - Source: User
  - Input Device: Phone Keyboard
  - Data Format: String
  - Example: “password123”

Google Account

- E-mail
  - Source: User
  - Input Device: Phone Keyboard
  - Data Format: String
  - Example: “example@email.com”
- Password
  - Source: User
  - Input Device: Phone Keyboard
  - Data Format: String
  - Example: “password123”

Facebook Account

- E-mail
  - Source: User
  - Input Device: Phone Keyboard
  - Data Format: String
  - Example: “example@email.com”
- Password
  - Source: User
  - Input Device: Phone Keyboard
  - Data Format: String
  - Example: “password123”

Create Account

Native Account

- First Name
  - Source: User
  - Input Device: Phone Keyboard
  - Data Format: String
  - Example: “John”

- Last Name
    - Source: User
    - Input Device: Phone Keyboard
    - Data Format: String
    - Example: “Doe”
- E-mail
    - Source: User
    - Input Device: Phone Keyboard
    - Data Format: String
    - Example: “example@email.com”
- Password
    - Source: User
    - Input Device: Phone Keyboard
    - Data Format: String
    - Example: “password123”
- Verified Password
    - Source: User
    - Input Device: Phone Keyboard
    - Data Format: String
    - Example: “password123”

Google Account

- E-mail
    - Source: User
    - Input Device: Phone Keyboard
    - Data Format: String
    - Example: “example@email.com”
- Password
    - Source: User
    - Input Device: Phone Keyboard
    - Data Format: String
    - Example: “password123”

Facebook Account

- E-mail
    - Source: User
    - Input Device: Phone Keyboard
    - Data Format: String
    - Example: “example@email.com”
- Password
    - Source: User
    - Input Device: Phone Keyboard
    - Data Format: String
    - Example: “password123”

Friend Requests

E-mail Search

- E-mail
  - Source: User
  - Input Device: Phone Keyboard
  - Data Format: String
  - Example: “example@email.com”
- Phone Contacts
  - Source: Phone Address Book
  - Input Device: Phone
  - Data Format: JSON
  - Example:

```
{
      name: {
            givenName: “Israel”
            familyName: “Flores”
      }
      email: “israel@flores.com”
}
```

- Google Contacts
  - Source: Google Account
  - Input Device: Phone
  - Data Format: JSON
  - Example:

```
{
      firstName: “Test”,
      lastInitial: “T”,
      email: “testg1@yahoo.com”,
      type: “home”
}
```

Availability

Free Mode

- Set free mode
  - Source: User
  - Input Device: Phone's touch screen
  - Data Format: Integer
  - Example: 1
- Time in free mode
  - Source: User
  - Input Device: Phone's touch screen
  - Data Format: Integer
  - Example: 90

Invisible Mode

- Set invisible mode

        - Source: User
        - Input Device: Phone's touch screen
        - Data Format: Integer
        - Example: 2
    - Time in invisible mode
        - Source: User
        - Input Device: Phone's touch screen
        - Data Format: Integer
        - Example: 90

Schedule Mode

- Set schedule mode
    - Source: User
    - Input Device: Phone's touch screen
    - Data Format: Integer
    - Example: 0

Calendars

- Google Calendar import
    - Source: Google Account
    - Input Device: Phone's touch screen
    - Data Format: JSON
    - Example:

```
{
      id: “4324jj@group.com”,
      user_id: “101”,
      name: “Mom’s Calendar”
}
```

- Google Event import
    - Source: Google
    - Input Device: Phone's touch screen
    - Data Format: JSON
    - Example:

```
{
      calendar_id : "",
      created : "2014-10-27 21:21:35",
      updated : "2014-10-27 21:41:32",
      summary : "Meeting",
      start_date : "2014-11-14",
      start_time: "08:00:00",
      end_date : "2014-11-14",
      end_time : "18:00:00"
}
```

Mood

Set Mood

- Set mood text

    - Source: User
    - Input Device: Phone's keyboard
    - Data Format: String
    - Example: “Busy”

Nudges

- Send Nudge
    - Friend ID
    - Source: User
    - Input Device: Phone's touch screen
    - Data Format: Integer
    - Example: 81
- Nudge text (optional)
    - Source: User
    - Input Device: Phone's keyboard
    - Data Format: String
    - Example: “You have been nudged”
- Delete Nudge
    - Friend ID
    - Source: User
    - Input Device: Phone's touch screen
    - Data Format: Integer
    - Example: 81

Favorites

- Set Friend as Favorite
    - Friend ID
    - Source: User
    - Input Device: Phone's touch screen
    - Data Format: Integer
    - Example: 81
- Remove Friend as Favorite
    - Friend ID
    - Source: User
    - Input Device: Phone's touch screen
    - Data Format: Integer
    - Example: 81

5.2. Other User Output Specification

Availability

- Availability List
    - Friend's Name
    - Source: Database
    - Output Device: Phone screen
    - Data Format: String
    - Example: “John Doe”

        - Meaning: The name of the friend that is available to hang out.
    - Available Time
        - Source: Database
        - Output Device: Phone screen
        - Data Format: Time
        - Example: 01:45
        - Meaning: How much time that person has before they are busy
    - Mood Text
        - Source: Database
        - Output Device: Phone screen
        - Data Format: String
        - Example: “I am busy”
        - Meaning: Flavor text that user may input as their current mood or status.

Requests

- Friend Request List
    - Requester's Name
    - Source: A separate user
    - Output Device: Phone Screen
    - Data Format: String
    - Example: “Jane Doe”
    - Meaning: The name of the person requesting to be friends.

Nudges

- Nudge's List
    - Nudger's Name
    - Source: A separate user
    - Output Device: Phone Screen
    - Data Format: String
    - Example: “Jane Doe”
    - Meaning: The name of the person who nudged the user.
- Nudge Text(Optional)
    - Source: A separate user
    - Output Device: Phone Screen
    - Data Format: String
    - Example: “You have been nudged”
    - Meaning: Optional text that may accompany a nudge

Friends

- Friend List
    - Friend's Name
    - Source: Database
    - Output Device: Phone screen
    - Data Format: String

        - Example: “John Doe”
        - Meaning: The name of a person that a user is friends with.
- Favorite Friends
    - Source: Database
    - Output Device: Phone screen
    - Data Format: Integer
    - Example: 1
    - Meaning: Designation is a friend is on a user's favorite list.
- Friend Invite
    - User Search
        - Current Corral User
            - Name
            - Source: Database
            - Output Device: Phone screen
            - Data Format: String
            - Example: “Jane D”
            - Meaning: When searched, this name will designate that the user is a current Corral user.
        - Non-Existent User
            - E-mail
            - Source: User
            - Output Device: Phone screen
            - Data Format: String
            - Example: “example@email.com”
            - Meaning: When searched, this email will designate that the user is not currently signed up with Corral.
        - Phone Contacts
            - Name
            - Source: Phone Address Book
            - Output Device: Phone screen
            - Data Format: String
            - Example: “Jane ”
            - Meaning: When searched, this name will designate that the user is a current Corral user.

            - E-mail
            - Source: Phone Address Book
            - Output Device: Phone screen
            - Data Format: String
            - Example: “example@email.com”

        - Meaning: When searched, this email will designate that the user is not currently signed up with Corral.
    - Google Contacts
        - Name
        - Source: Google Account
        - Output Device: Phone screen
        - Data Format: String
        - Example: “Jane ”
        - Meaning: When searched, this name will designate that the user is a current Corral user.

        - E-mail
        - Source: Google Account
        - Output Device: Phone screen
        - Data Format: String
        - Example: “example@email.com”
        - Meaning: When searched, this email will designate that the user is not currently signed up with Corral.

Calendar

- Google Calendar
    - Calendar Name
    - Source: Google
    - Input Device: Phone Screen
    - Data Format: JSON
    - Example:

```
{
      name: “Fun Calendar”
}
```

    - Meaning: Will list the available Google calendars to import into Corral.

5.3. System Database/File Structure Specification

- Identification of Database/Files

The database used by the Corral app is a MySQL relational database.

- (Sub)systems Accessing the Database

Access to the database is not done in native SQL anywhere on the server. Instead, the database is controlled using PHP methods. In turn the PHP methods can be accessed via REST services from the client app. Each REST service encapsulates a specific function that to be executed on the database. For

example, the database needs to be accessed to add a new user. The flow for adding a new user is as follows:

- User inputs the required registration information on the client app.
- The client app invokes a REST service on the server, sending it all the user registration information.
- The REST service invokes a PHP method to add the user to the database.

In the example above, the client app invoked an update operation on the database. Access to the database is always done in this format, where the client app triggers a command on the database whether that be a create command, read command, update command or delete command.

Logical File Structure

The Corral app has both a server component as well as a client component. The files that make up the app can be found on both the server and the client.

The server contains the following folders at the root of the file system:

- Config - Contains Application configuration files, such as the following:
  - app.php
    Contains settings such as the app name as well as the locations to find the other files of the server.
  - database.php
    Contains settings to find the database configuration file and declares mySQL as the specified Database Management System.
- Local - Stores E-Mail Configuration files, such as
  - mail.php
    Stores credentials for e-mails and points to a GMail SMTP server to power the Corral invite system.
- Testing - Files for testing the PHP modules of the server.
- Controllers - The main logic of the app. Each file here defines one major idea in the app, as follows:
  - UserController.php
    Most of the app logic is in this file. It defines all interactions between users.
  - FacebookController.php
    This file contains the logic for all interactions with Facebook, such as Facebook login and Facebook sharing.
  - GoogleController.php
    This file contains the logic for all interactions with Google, such as Google Login, Google+ sharing, and Google Calendar integration.
- Models – Model Classes uses to interface between PHP and the database.
  - Availability.php

    - FacebookUsers.php
    - Friends.php
    - GoogleCalendar.php
    - GoogleEvent.php
    - GoogleUsers.php
    - Nudge.php
    - User.php

The server also contains files that are simply at the root of the file system and not in any folder. These files are as follows:

- Corral.sql
  This file contains the MySQL needed to create and manage the database.
- Routes.php
  This file points calls from the client application to REST services on the server to the PHP classes that handle each call.

The client application uses the file structure of other Cordova Applications, which is as follows at the root:

- Platforms – This folder contains Platform-specific code to allow the Cordova application to be deployed on multiple platforms. Each platform has a subfolder in this folder, and all code specific to that platform in contained there.

- Plugins – Stores files for each of the plugins installed in Cordova. In the case of Corral, the OpenFB plugin and the Local Notification Plugin are stored here.

- WWW – This folder stores the HTML and JavaScript files that make up the User Interface and logic of the application. It's contents are as follows:

    - HTML files – These files define the layout of the Cordova application
    - JS folder – Contains JavaScript files that define the logic for the application, as well as JavaScript libraries such as Ionic for responsiveness.
    - IMG folder – Contains image files for the various images used in the application such as the application icon, backgrounds and splash screens.
    - CSS folder – Contains CSS files that define the style of the HTML pages and the application overall.
    - Fonts folder – Contains custom font definitions to allow custom fonts to be displayed properly.

- Physical File Structure
  As stated above, the files that make up the Corral application are split between the client and the server. For the files on the server, storage is provided on a hard drive shared with other users of the server. For the files on the client, they are on the user's device, which is usually flash storage. Management of the disks on the server is in the control of the operating system, which is Linux on the server and Android (Linux) on the client.

- Database Management Subsystems Used
  The database in the Corral application is managed by MySQL. Its structure and file usage is defined completely by MySQL, and not by the server.

5.4. System Internal Data Structure Specification

- Identification of Data Structures: MySQL Relational Database
  Tables
  - Users
    Main table storing the identification information for a user of the app such as name, e-mail and password as well as app-specific information such as mood and an identification number. This table is related to all the other tables in the database.

  - Google Users
    Supplemental table to users that stores extra information relating to the Google account of that user. This information provides access to information and services on that user's Google account such as calendars and Google+ sharing.

  - Facebook Users
    Supplemental table to users that stores extra information relating to the Facebook account of that user. This information provides access to information and services on that user's Facebook account such as Facebook sharing.

  - Friends
    Table storing the relationship between two users that allows sharing of event information between users in the client app.

  - Nudges
    Table storing messages sent from one user to another using the client app.

  - Availabilities
    Table storing the availability of the user.

- Google Calendar
  Table storing the identifying information for the user in order to access their Google Calendar information.

- Events
  Table that stores the events saved to the Google Calendar of the user. This information is imported from Google.

Modules Accessing Structures

All the modules in use in the Corral application make use of the database in some way. Most of the modules read data and perform actions based upon the data they read, but some create and update data found in the database. Below is a summary of each module and the interactions it has with the database.

- OpenFB
  The OpenFB module takes care of all calls to the Facebook API. As a result, it generates information that must be stored on the database such as an authentication token that allows for new calls to the Facebook API. The OpenFB module also consumes data from the database, requiring and authentication token and a user's e-mail address for making calls to the Facebook API. Since the OpenFB module is completely inside the client application, so it must interact with the database and the server using calls to REST services.

- Google API
  Unlike the Facebook API, the Google API is easier to integrate into a hybrid client application, so a third-party module was not needed to make calls to the Google API. Also unlike the Facebook API, the Google API is called from both the client application and the server, to implement features such as calendar import, which required data to be saved in the database. Similar to Facebook, the Google API also generates an authentication token that needs to be saved to the database for future calls to the API, along with the user's e-mail address. Calls to the API also require pulling the token and e-mail address of the user from the database. If the Google API is called from the server, then the database is accessed indirectly using PHP classes. If instead the Google API is accessed from the client application, the database is accessed through REST services, similar to the OpenFB module.

- Ionic Framework
  The Ionic Framework is another of the Corral application modules that is completely contained in the client application. As a result, all access to the database from the Ionic Framework is done through the use of calls to

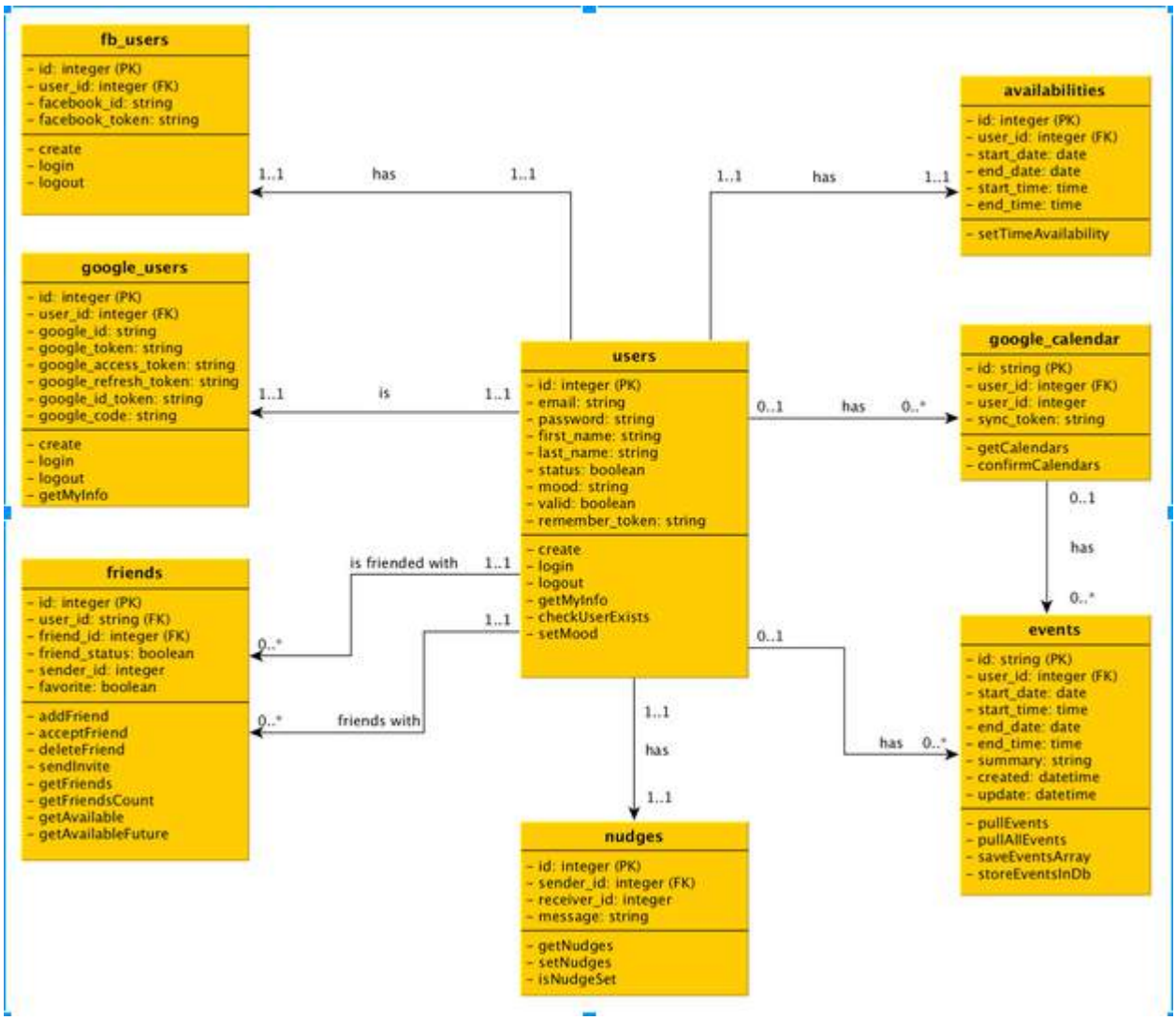
REST services. Ionic controls much of the interactions of the user with the client application, so Ionic consumes more information from the database then it generates. Most of the data consumed from the database comes from the Users and Friends tables, as it is what is used to determine availability between users and to show user information. Ionic also generates information to be stored in the database, such as nudges.

- Laravel
  Laravel is completely contained in the server, similar to Ionic being contained in the client application. So, all access to the database is done through the use of PHP classes. Since Laravel also receives and controls all REST service calls from the client application, it accesses the database anytime a request comes in from the client application.

- Cordova
  Similar to Laravel, Cordova controls all aspects of the client application, including modules installed to the client application like OpenFB. So, all access to the database from Cordova is done through REST services. Also like Laravel, Cordova adds information to the database or consumes data from the database whenever OpenFB and the Ionic Framework do.

Logical Structure of Data

Since all of the data in the application is stored in a relational database, its structure depends entirely on the relationships between the tables that the data is organized into. Furthermore, all data stored in the application database revolves around the Users table. Each of the other tables in the database relates in some way or another to the Users table. The following are the relationships in the database, along with a UML diagram representing the relationships.



- Users/Availabilities
  This relationship is one to one in both directions. A user can have only have availability at any given time (either available or not available). The Availabilities table simply stores how long that availability persists. Any availability is associated with only one user since availability is a trait unique to a specific user.

- Users/Google Calendar
  In this relationship, a calendar can be associated with no users and at most one user. A user can have no Google calendars or up to many calendars.

- Google Calendar/Events
  This relationship is the only one that does not relate directly to the users table. A calendar can have no events or up to many events, and an event

does not have to be associated to a calendar (allowing for the possibility of user-created events) but can be associated with at most one calendar.

- Users/Events
  A user can have no events or many events, and an event can exist alone or be associated with at most one user.

- Users/Nudges
  This relationship is also one to one in both directions. So, a user can have one and only one nudge at any given time from another user. Similarly, a nudge can be associated with only one user (since the association is made only to the sending user).

- Users/Users – Friends Table
  This relationship is a self-relationship. Since the relationship is many to many (A user can have many friends), there is a separate table (Friends) that stores information about each specific relationship. A relationship need not exist between two users (indicating that they are not friends).

- Users/Google/Facebook Users
  This is two relationships, but the relationship is the same. A user can be a Google or Facebook user only once (in other words, they can only have one account with either Facebook or Google).

6. Module Design specifications

Module:	OpenFB
Module Functional Specification	
Functions Performed:	OpenFB allows you to login to Facebook and execute any Facebook Graph API request.
Module Interface Specifications:	OpenFB is a micro-library that lets you integrate your JavaScript applications with Facebook. It works for both browser-based and Cordova/PhoneGap apps.
Module Limitations and Restrictions:	Not full-fledged, less out-of-the-box features. Integration not as tight, for example no native dialogs.
Module Operational Specification	
Locally Declared Data Specifications:	Access Tokens, email, name, profile information, friends
Algorithm Specification:	1. Log in using Facebook username and password 2. Store access token 3. Access profile information, such as name and friends 4. Execute necessary Facebook Graph API requests
Description of Module Operation:	The Facebook Plugin is still the best technical solution to integrate your Cordova app with Facebook because it provides a tighter integration (using native dialogs, etc). However, if you are looking for a lightweight and easy-to-set-up solution with no dependencies, or if you are targeting mobile platforms for which an implementation of the plugin is not available, you may find this library useful as well.

Module:	Google API
Module Functional Specification	
Functions Performed:	Authenticate Google users, access Google calendars
Module Interface Specifications:	Inputs include Google+ username and password. Outputs include name, friends, and calendar events.
Module Limitations and Restrictions:	Can’t use if Google servers are down.
Module Operational	

Specification	
Locally Declared Data Specifications:	Access tokens, email, names, calendar
Algorithm Specification:	1. Log in using Google+ username and password 2. Store access token 3. Access profile information, such as name, friends, and calendar events 4. Share about the app on Google+ profile
Description of Module Operation:	This module allows Google+ user authentication by communicating with Google servers. It also allows the app to access user information that is necessary for the app, such as name, friends, and calendar events.

Module:	Ionic Framework
Module Functional Specification	
Functions Performed:	Increases fluidity of Javascript on the phone.
Module Interface Specifications:	Uses Javascript to make applications more interactive.
Module Limitations and Restrictions:	Since it is targeted for cross-platform development, it will not be able to do as much as native development.
Module Operational Specification	
Locally Declared Data Specifications:	N/A
Algorithm Specification:	N/A
Description of Module Operation:	Open source front-end SDK for developing hybrid mobile apps with HTML5.

Module:	Laravel
Module Functional Specification	
Functions Performed:	Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, and caching.
Module Interface	Laravel uses PHP. It can receive and output Javascript calls

Specifications:	and calls to the MySQL database.
Module Limitations and Restrictions:	Since Laravel is used for the backend of the app, it is dependent on the frontend and the database.
Module Operational Specification	
Locally Declared Data Specifications:	N/A
Algorithm Specification:	1. Receive a message from the frontend. 2. Accesses the database. 3. Retrieves the desired data. 4. Sends the desired data back to the frontend.
Description of Module Operation:	Laravel attempts to combine the very best of other web frameworks, including frameworks implemented in other languages, such as Ruby on Rails, ASP.NET MVC, and Sinatra.

Module:	Cordova
Module Functional Specification	
Functions Performed:	Apache Cordova is a set of device APIs that allow a mobile app developer to access native device function such as the camera or accelerometer from JavaScript.
Module Interface Specifications:	Cordova provides a set of uniform JavaScript libraries that can be invoked, with device-specific native backing code for those JavaScript libraries.
Module Limitations and Restrictions:	Since it is targeted for cross-platform development, it will not be able to do as much as native development.
Module Operational Specification	
Locally Declared Data Specifications:	N/A
Algorithm Specification:	N/A
Description of Module Operation:	Combined with a UI framework such as jQuery Mobile or Dojo Mobile or Sencha Touch, this allows a smartphone app to be developed with just HTML, CSS, and JavaScript.

Module:	Database
Module Functional Specification	
Functions Performed:	Stores all the necessary data. Creates and finds relationships between the data.
Module Interface Specifications:	The database is created using MySQL. It is accessed by the app using PHP calls to the database. The database then outputs the desired data, which is retrieved using PHP.
Module Limitations and Restrictions:	The database cannot be accessed if the servers are down.
Module Operational Specification	
Locally Declared Data Specifications:	email, name, friends, availability
Algorithm Specification:	1. A new account is created, so an email and password are sent to the database. 2. The database stores the email and password. 3. A message confirming the successful account creation is sent back.
Description of Module Operation:	The database stores all relevant data for the app to function properly.

7. System Verification

7.1. Items/Functions defined in Test Suite

- Account Creation
    - Creating an Account Without Social Media (In-App)
    - Creating an Account using a Google Account
    - Creating an Account using a Facebook Account
- Logging In
    - With an In-App Account
    - With a Google Account
    - With a Facebook Account
- Sending Friend Requests
    - Sending a Single Friend Request to
        - Friends With an Account in Corral
        - Friends Without an Account in Corral
    - Sending Friend Requests through Phone Contacts
        - Friends With an Account in Corral
        - Friends Without an Account in Corral
    - Sending Friend Requests through Google Contacts
        - Friends With an Account in Corral
        - Friends Without an Account in Corral
- Accepting Friend Requests
- Denying Friend Requests
- Deleting Friends
- Setting Availability
    - Set Free Mode
        - Infinite Duration
        - Limited Duration
    - Set Schedule Mode
    - Set Invisible Mode
        - Infinite Duration
        - Limited Duration
- Checking Available Friends
    - Current Available Friends
    - All Friends Available at a Specific Day and Time
- Importing Google Calendar
- Setting Favorite Friends
- Sending a Nudge
- Acknowledging a Nudge
- Receiving an Android Notification for a Nudge
- Setting Mood Message
- Sharing to Facebook

7.2. Description of Test Cases

The test cases are designed to follow the following format

Test ID	(type of test)_(sprint number)_(requirement number)
Title	(descriptive title of what we are testing)
Priority	(low,med,high) (how much does this feature we are testing impact our app's usability)
Test Last Updated	(date of when you last edited this test)
Last Run Date	(date of the last time you ran this test)
Description	(The purpose of this test, more detailed than the title usually)
Pre-Conditions	(what needs to be set-up before testing. Do we need 2 accounts already? Does the account need to have friends already? Does it have to be a gmail or facebook account? Etc...)
Dependencies	(things like "the server needs to be running" or "google isn't down for maintenance"
Steps	(the individual steps you need to take to run the test, try and be specific)
Expected Behavior	(the result after every step. There should be an equal number of these as there are steps)
Post-Conditions	(what is the status of the app when testing is finished, what was changed in the database or are things now friends that weren't before, etc..._
Last Test Result:	(did the last run of this test PASS or did it FAIL)
Notes:	(any comments or concerns or potentially information on where the test failed and why iy might have failed)

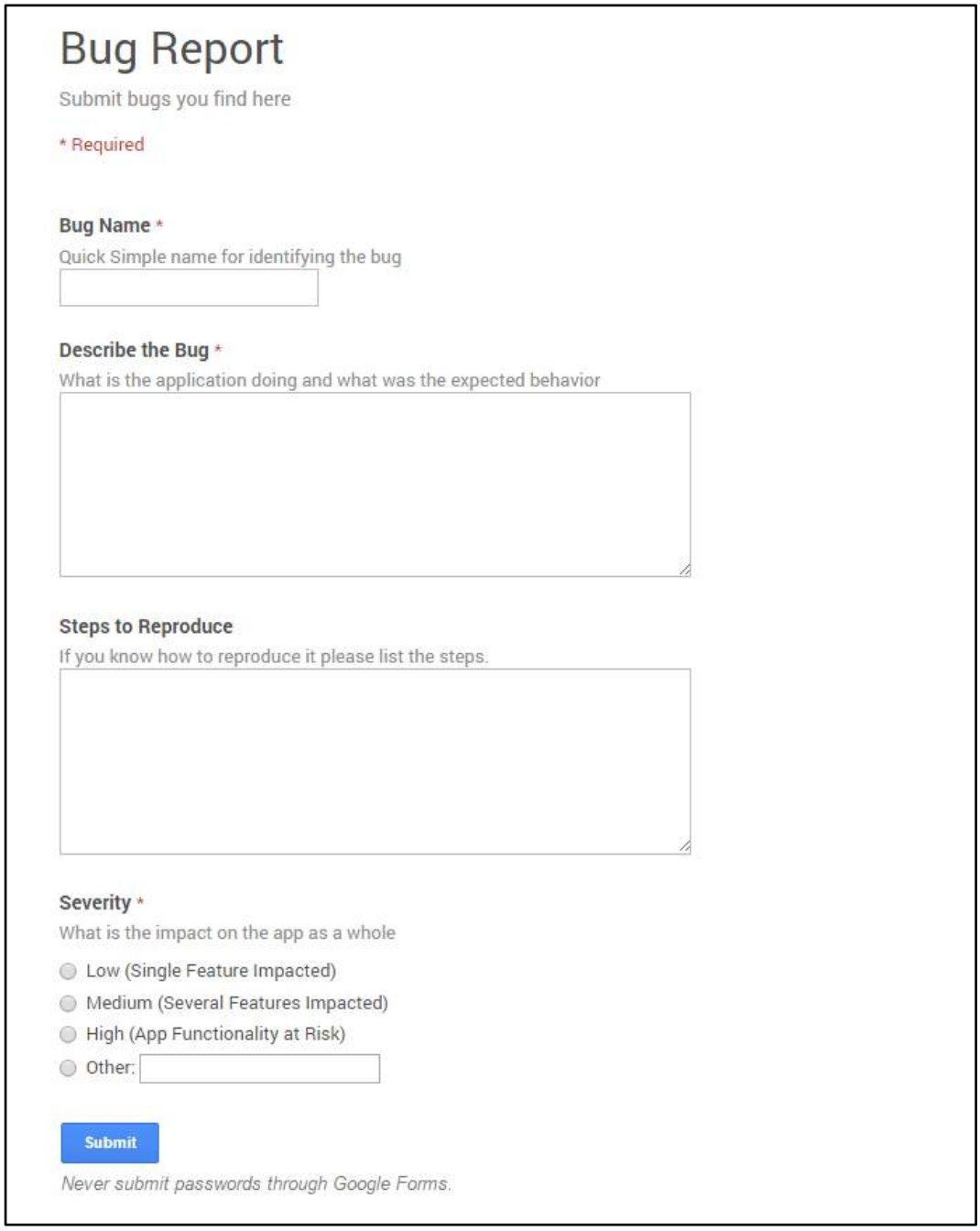
7.3. Justification of Test Cases

- Regression Testing
  - Each sprint all test cases are run from the previous sprints to check that recent changes have not broken old features. Performing these tests makes sure that there is a solid user experience and that application flow is correct.
- Specification-Based Testing
  - New test cases are written each sprint to define how to test new features. These test cases are written according to the proposal's defined user stories.

7.4. Test Run Procedures and Logging Test Results

Test ID	(type of test)_(sprint number)_(requirement number)
Title	(descriptive title of what we are testing)
Priority	(low,med,high) (how much does this feature we are testing impact our app's usability)
Test Last Updated	(date of when you last edited this test)
Last Run Date	(date of the last time you ran this test)
Description	(The purpose of this test, more detailed than the title usually)
Pre-Conditions	(what needs to be set-up before testing. Do we need 2 accounts already? Does the account need to have friends already? Does it have to be a gmail or facebook account? Etc...)
Dependencies	(things like "the server needs to be running" or "google isn't down for maintenance"
Steps	(the individual steps you need to take to run the test, try and be specific)
Expected Behavior	(the result after every step. There should be an equal number of these as there are steps)
Post-Conditions	(what is the status of the app when testing is finished, what was changed in the database or are things now friends that weren't before, etc..._
Last Test Result:	(did the last run of this test PASS or did it FAIL)
Notes:	(any comments or concerns or potentially information on where the test failed and why i might have failed)

- These fields are used to keep track of test results. When test cases are run, the word document is opened and fields are modified in order to keep track of when the last test was run and what the result of that test was.
- When a bug is encountered, a Google form is used to submit the details of the bug



Bug Report

Submit bugs you find here

* Required

Bug Name *

Quick Simple name for identifying the bug

Describe the Bug *

What is the application doing and what was the expected behavior

Steps to Reproduce

If you know how to reproduce it please list the steps.

Severity *

What is the impact on the app as a whole

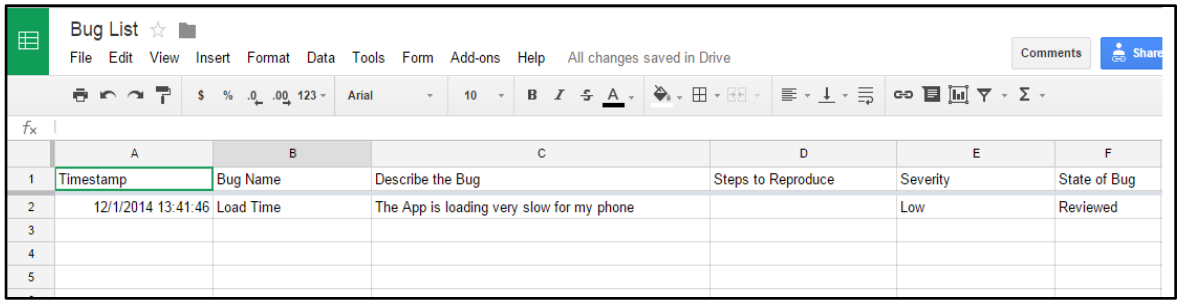
- Low (Single Feature Impacted)
- Medium (Several Features Impacted)
- High (App Functionality at Risk)
- Other:

Submit

Never submit passwords through Google Forms.

- The results of this form are stored in a spreadsheet on Google. From there state of the bug is tracked as well as the information provided by the form. When a bug

has been reviewed, fixed, and verified, it is removed from the list.



	A	B	C	D	E	F
1	Timestamp	Bug Name	Describe the Bug	Steps to Reproduce	Severity	State of Bug
2	12/1/2014 13:41:46	Load Time	The App is loading very slow for my phone		Low	Reviewed
3						
4						
5						

8. Conclusions

8.1. Summary

The goal of the project is helping people get together at anytime with less hassle, by consolidating their calendars and their friends' calendars.
By the end of the semester, all the features were implemented as described in the project proposal. Some of the features that help fulfill the original goal of this project are allowing user to sign up using Google Account, pulling user's calendar events from Google Calendar, allowing user to manually overwrite his/her availability status, and effectively figuring out which of user's friends are available to get together.

Along with completing the project according to the requirements as detail in the project proposal, the team members have also advanced their skills in mobile app and web service development. An API was developed using the Laravel MVC framework; backend technologies were learned such as PHP and RESTful web services, and incorporate MySQL database designs that were previously acquired in CECS 323; social network APIs are incorporated from Facebook and Google to eliminate redundant user data entry; and mobile application development on Android platform was learned as well.

8.2. Problems Encountered and Solved

- Google+ Sharing
  - Although the Google+ API provides writing privileges to a user's profile, the API endpoints aren't fully flushed out to allow actually pushing of a Moment (Google's name for a post object) to the user's wall. For the end of these sprint and the overall project, the team has spoken with the professor to leave out the functionality completely, as user's that have created Facebook accounts with the app can still share their availability on their wall.
- Slow latency with jQuery mobile interactions
  - With the early discovery of a low latency response of using jQuery Mobile UI, one of the members decided to take a look at other mobile frameworks, finally landing with the Ionic Framework which helps create hybrid mobile apps. This vastly improved responsiveness that was needed on the main page of the app, while still using jQuery Mobile UI on the least visited pages.

8.3. Suggestions for Better Approaches to Problem/Project

- Setup the whole infrastructure to follow a true service-oriented application (SOA), since the current configuration has a major failure point, in that all the API layer and database layer resides on one shared hosted server.
- Setup the same virtual development environment for all team members to eliminate any potential problems that may occur due to variations in development environment and operation environments.

8.4. Suggestions for Future Extensions to Project

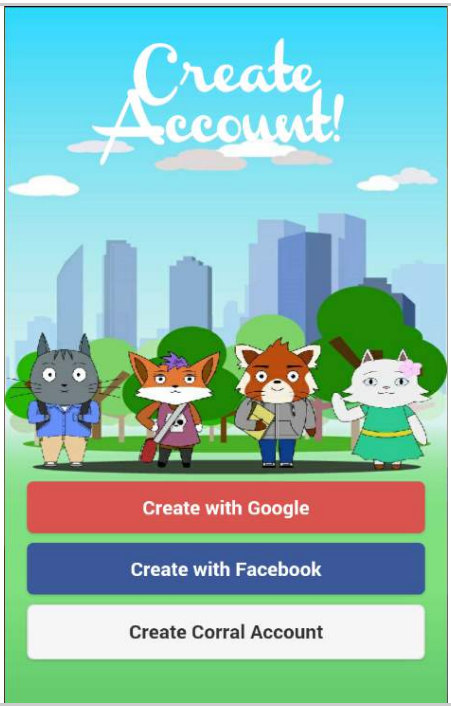
- Allow users to set their weekly schedule in their profile
- Allow users to sign up with the email requirement, using phone number as alternative.
- Allow users to add friends using a phone number or names

Corral User Manual

I. Creating an Account

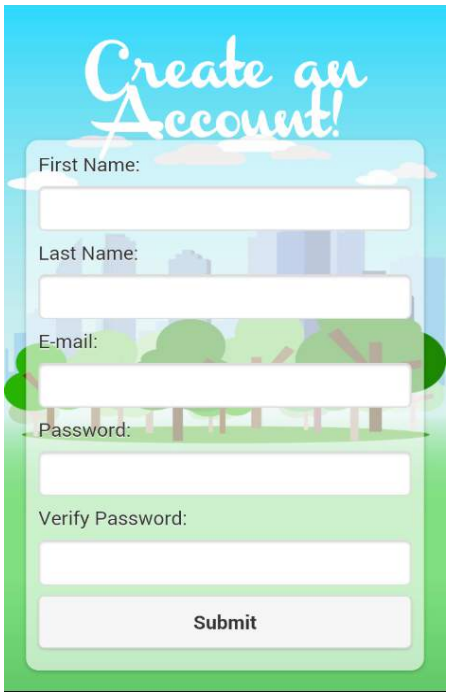
A. Google or Facebook

1. Press Create Account
2. Select Either Google or Facebook
3. Log in to your Google or Facebook account
4. Accept the permissions for the app
5. You will automatically be logged in to your new account



B. Create an Account without Social Media (In-App)

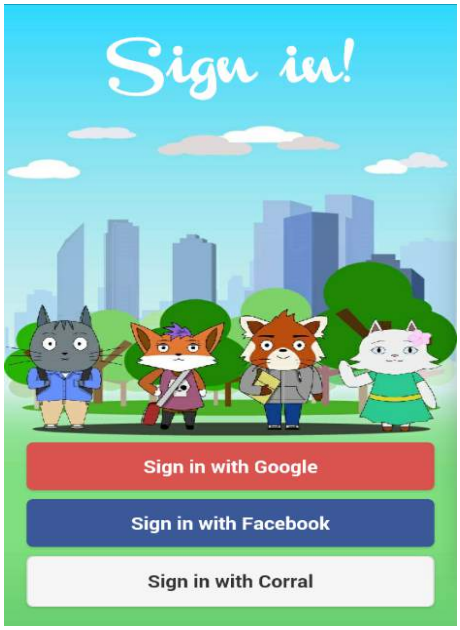
1. Press Create Account
2. Select “Create Corral Account”
3. Fill in all the fields in the form
4. Press Submit
5. You will automatically be logged in to your new account



II. Logging In

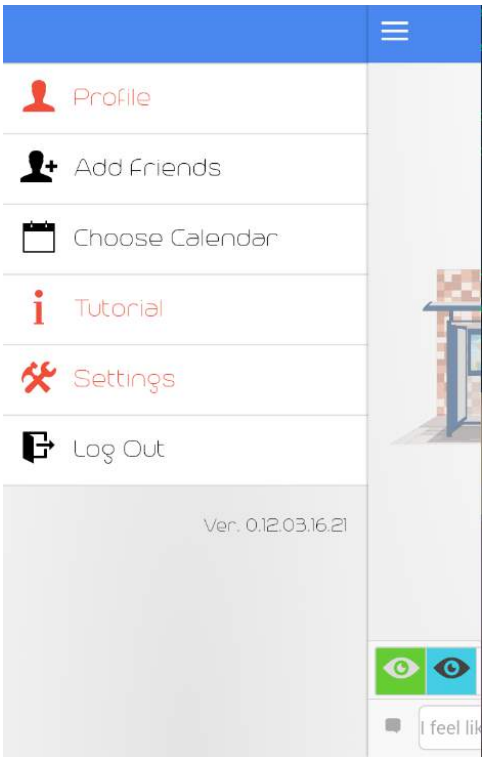
1. Press Sign In
2. Select the type of account you created
3. Enter your credentials to Sign In

Note: for Google and Facebook these are the credentials for your Google and Facebook account



III. Logging Out

1. From the main Availability page of the app, access the menu by swiping from left to right or by tapping the icon in the top left corner.
2. Select Log Out from the menu in order to logout.

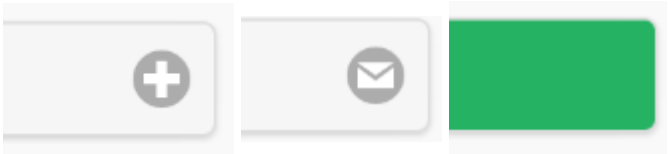


IV. Adding Friends

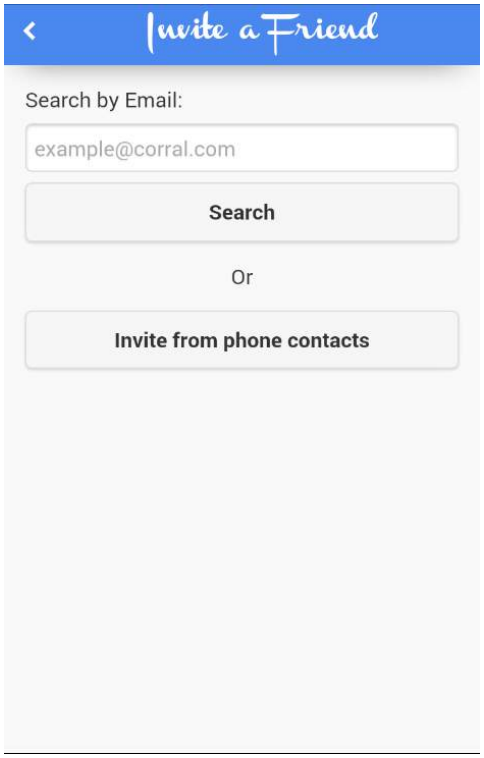
A. Adding a Single Friend

1. From the main Availability page of the app, access the menu by swiping from left to right or by tapping the icon in the top left corner.
2. From the menu, select “Add Friends”
3. Enter the E-email address of the friend you want to invite. (This should be the E-mail address they used to create the account with.)
4. Press Search, the name of your friend or the e-mail address you entered should appear below as a button
5. Tapping on this button will send your friend a request.

Note: Depending on the icon, the type of request sent will be different. A plus symbol means that e-mail is a registered user and will send them an In-App request



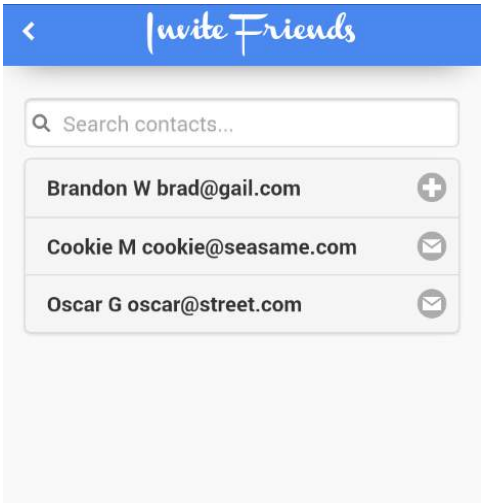
A Mail symbol means that E-mail is not a registered user, and will send them an email invite to start using the app along with the request.
If the button is Green, they are already your friend.



B. Adding Friends From Your Google or Phone Contacts

1. From the main Availability page of the app, access the menu by swiping from left to right or by tapping the icon in the top left corner.
2. From the menu, select “Add Friends”
3. Select whether you want to invite from your phone contacts or your Google contacts.
4. A list of all of your contacts will begin to populate, use the search bar to help you find a specific contact.
5. Tap the icon of anyone you wish to add as a friend.

Note: The list filters out all e-mails that are set as Work E-mails. To add those e-mails, please use “Adding a Single Friend”.



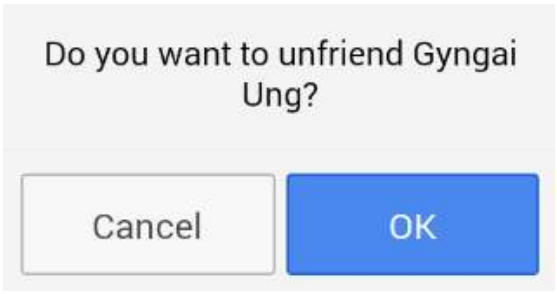
V. Managing Friends and Requests

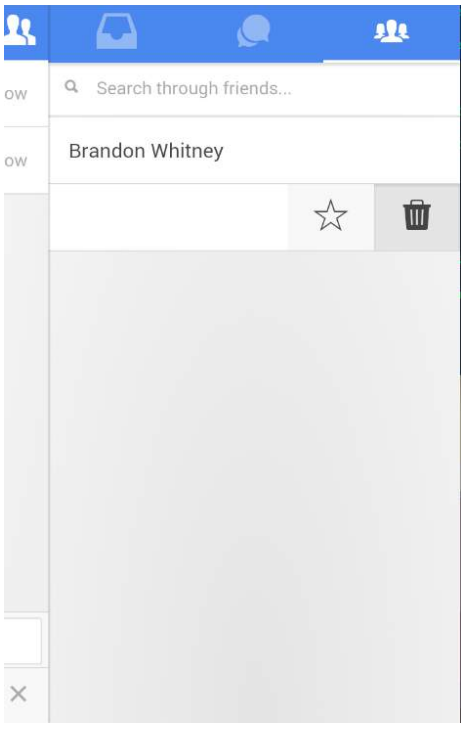
A. Deny or Accept Requests

1. From the main Availability page of the app, access your friend requests by swiping from right to left or by tapping the icon in the top right corner.
2. Check your pending requests by tapping the Inbox icon (shown here with a red 1 next to it).
3. Press the red X to deny the friend request
4. Press the green checkmark to accept the friend request
5. If you accepted the request, your new friend will appear in the friends tap denoted by the group icon (top right corner in this image).

B. Deleting Friends

1. From the main Availability page of the app, access your friends list by swiping from right to left or by tapping the icon in the top right corner.
2. Check your friends list by tapping the group icon (in the top right corner of this image).
3. Swipe from right to left on a friend to bring up the options for that friend.
4. Select the trashcan icon to delete that friend
5. Press OK on the pop-up prompt to confirm the deletion.





C. Setting Favorite Friends

1. From the main Availability page of the app, access your friends list by swiping from right to left or by tapping the icon in the top right corner.
2. Check your friends list by tapping the group icon (in the top right corner of this image).
3. Swipe from right to left on a friend to bring up the options for that friend.
4. Select the star to add that friend to your favorites

Note: Setting a friend to favorite means that friend will appear at the top of your availability list along with other favorite friends. Their name will appear in red to signify they are set as a favorite friend.

VI. Checking Available Friends

A. Current Available Friends

1. On the main Availability page, make sure that "Now" is selected. If not, select it.
2. Pull down on the top of the list to refresh the page.
3. The page will reload with all your friends that are currently available.

Notes:

- The clock symbol to the right of each name dictates how much longer that friend is free based on how long they set themselves to available.
- This feature also works if your friend has set themselves to "Schedule Mode" and has imported their Google Calendar events into the app.
- If they have set themselves to free infinitely or are in Schedule mode without having imported a Google Calendar, they will appear as "Free Now"

B. All Friends Available at a Specific Day and Time

1. On the main Availability page, select Later at the bottom of the page.
2. Tap the date field to set the date of the day you want to check.
3. Tap the time field to set the time of the day you want to set.
4. Pull down on the top of the list to refresh and check availability of all your friends for the date and time you specified.

Note: On some devices, taping the date and time fields will pull up a date and time picker. If a date and time picker doesn't pop up follow the following format:
Date: YYYY-MM-DD
Military Time (24hr clock): HH:MM:SS

Note 2: As of 12/09/2014 there is a known bug in Android Version 5.0 (Lollipop) where the time picker doesn't give you an option to set the time. This is an OS level bug so there is nothing that can be done to fix it.

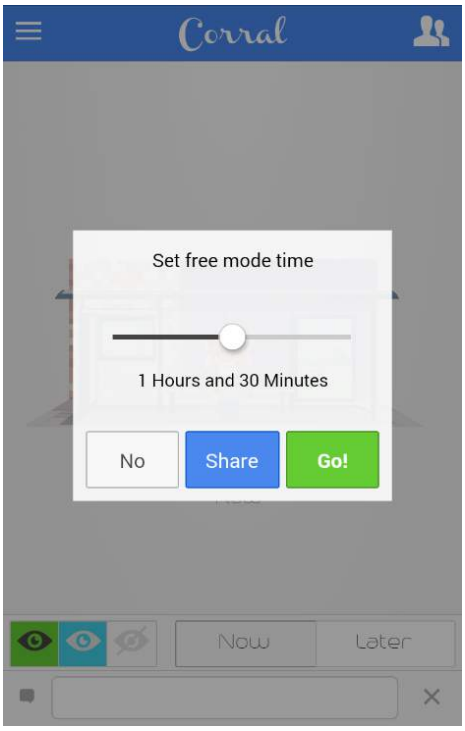
VII. Setting Availability

A. Set Free Mode

1. On the main Availability page, press the green eye icon on the bottom left of the screen.
2. Select how long you want to be available for. You can set from 15 minutes to 3 hours or to be free forever (until you change your availability manually).
3. Press Go to finalize your availability.

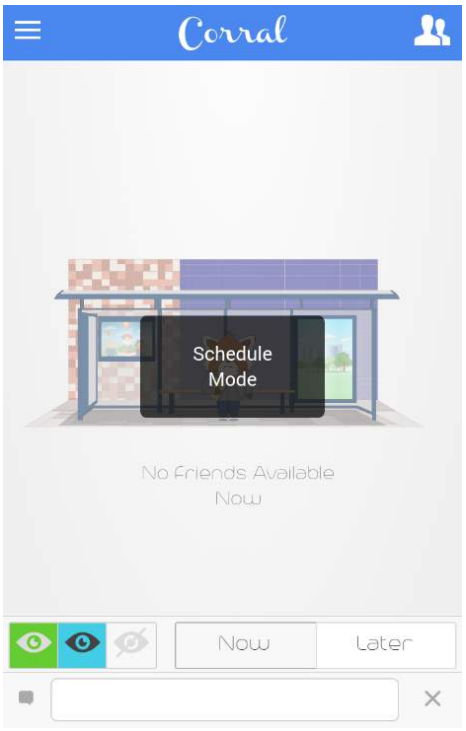
Note: To set yourself available forever, drag the slider to the far left.

Note 2: You can check how much longer you are set to available at any time by tapping the green eye icon again. If the message simply states "Free Mode" you are currently set available forever.



B. Set Schedule Mode

1. On the main Availability page, press the blue eye icon on the bottom left of the screen.

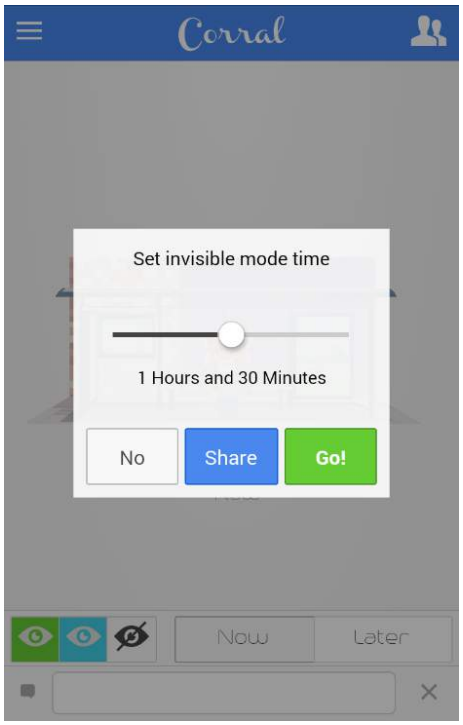


C. Set Invisible Mode

1. On the main Availability page, press the white eye icon on the bottom left of the screen.
2. Select how long you want to be invisible for. You can set from 15 minutes to 3 hours or to be invisible forever (until you change your availability manually).
3. Press Go to finalize your setting.

Note: To set yourself invisible forever, drag the slider to the far left.

Note 2: You can check how much longer you are set to invisible at any time by tapping the white eye icon again. If the message simply states “Invisible Mode” you are currently set invisible forever.



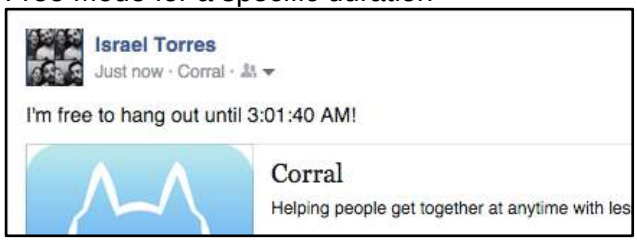
VIII. Sharing to Facebook

IMPORTANT: This feature is only available to Corral users that originally signed up with their Facebook account.

1. From the main Availability page of the app, press the green eye icon or the white eye icon to begin setting your availability.
2. When you are ready to share and set your availability, press Share instead of Go!
3. The system will send a default text post with a link to start using the app, letting your friends know your status change.

Note: The message generate depends on what you set

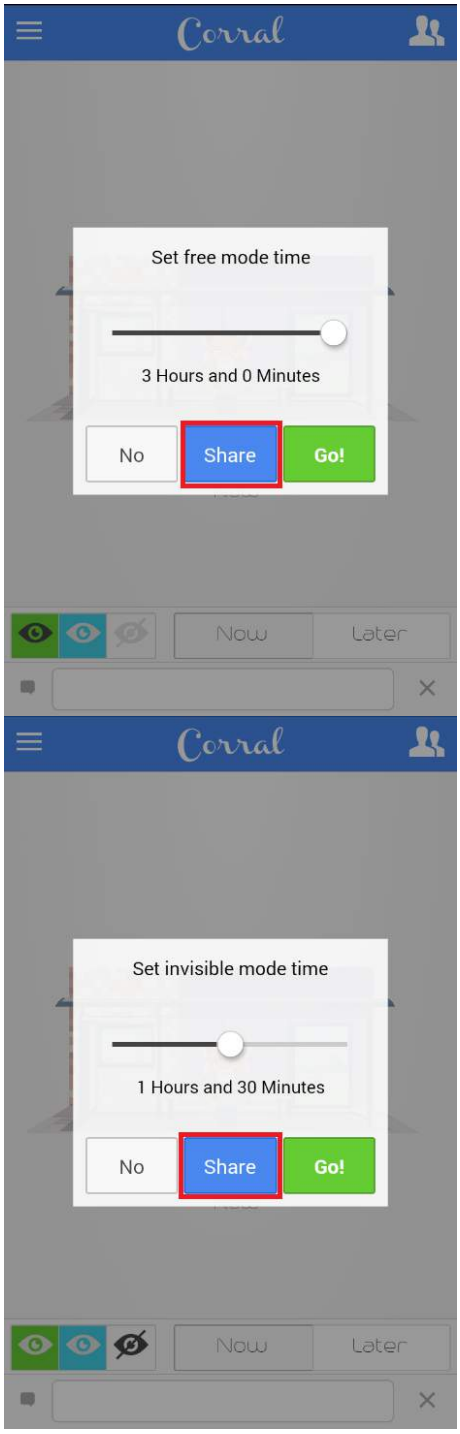
- Free Mode for a specific duration



- Free Mode forever

- Invisible for a specific duration

- Invisible forever

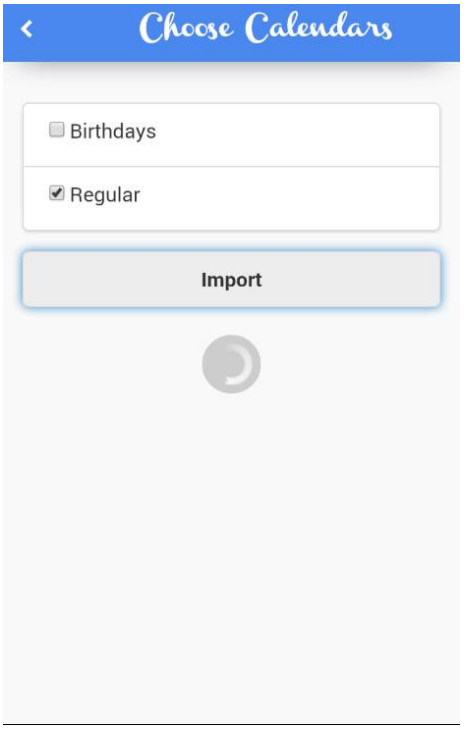


IX. Importing Google Calendar

IMPORTANT: This feature is only available to Corral users that originally signed up with their Google account.

1. From the main Availability page of the app, access the menu by swiping from left to right or by tapping the icon in the top left corner.
2. From the menu, select “Choose Calendar”
3. Select which of your Google calendars you wish to import events from.
4. Select Import to import your events from your selected calendars.

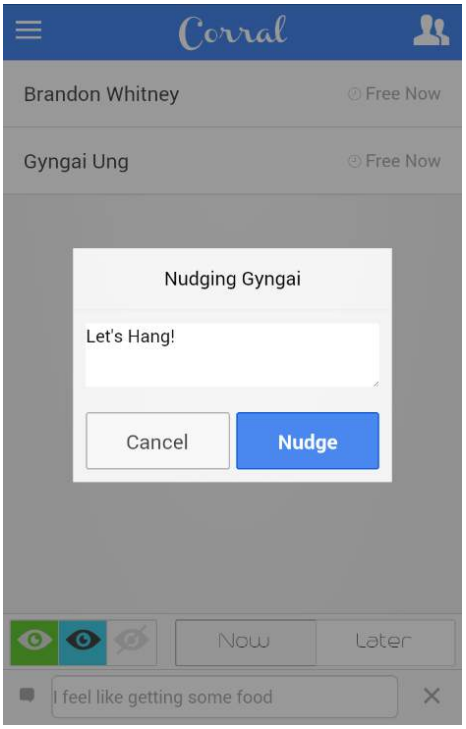
Note: This only pulls your events for the next 30 days. You will need to re-import your events after 30 days in order to update your availability. You can also re-import at any time to update your availability if you change your calendar.



X. Sending a Nudge

1. From the main Availability page, make sure “Now” is selected at the bottom of the screen and tap one of your currently available friends you wish to nudge.
2. Press the green nudge button that appears next to your friend’s name. 
3. You can enter custom text to your nudge if you wish, or just send a blank nudge.
4. When you are ready to send the nudge, press “Nudge”

Note: You must be set to “Now” because you can only nudge a friend that is currently available. You can not nudge someone who is not on your availability list (someone set to invisible or who is busy based on their schedule).



XI. Checking and Acknowledging a Nudge

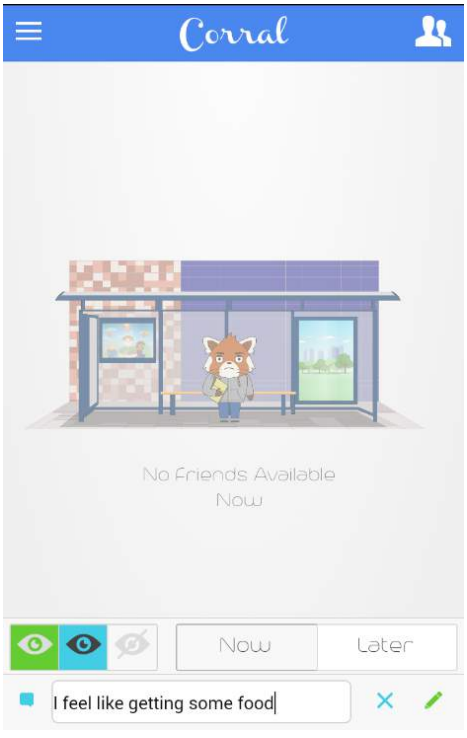
1. From the main Availability page of the app, access your pending nudges by swiping from right to left or by tapping the icon in the top right corner.
2. Check your nudges by tapping the speech bubbles (as shown with a red 1 next to it).
3. To acknowledge that you've seen the nudge and remove it from the list, tap the eye icon.

Note: If you leave the app running in the background of your device, you will receive a notification when you get a nudge from one of your friends. If you want to leave the app running in the background, use the Home button on your device to exit the app without logging out.

XII. Setting Mood Message

1. From the main Availability page, your current Mood message can be seen at the bottom of the screen underneath the eye icons.
2. Tap your current mood message to begin changing it.
3. When you are done changing it, press the green pen icon to finalize it.
4. To cancel your change, press the blue X icon.

Note: To clear your mood message and set it blank, press the grey X icon that is always visible whenever not editing your mood message.



XIII. Checking Friend's Mood Message

1. From the main Availability page, simply tap on any currently available friend to see their Mood message.
2. If the green Nudge icon is visible but you cannot see your friend's Mood message, your friend doesn't have a Mood message set.

Note: you can only check the Mood message of your currently available friends. The point of the Mood message is to quickly let people know how you're feeling while you are free so they know whether or not to Nudge you.