RedPins
Design and Planning Document
03/03/2013
Version 2.0
Members: Andy Lee, Benjamin Le, Eric Cheong, Jerry Chen, Victor Chang

Rev 2.0 2013-03-19 -version 2.0, post-iteration 1
Rev 1.1 2013-03-15 -version 1.1
Rev 1.0 2013-03-03 -initial version

System Architecture

RedPins is essentially a front-end Android application that will run and communicate with a backend application server and database implemented with existing open source MVC architecture (specifically, Ruby on Rails deployed on Heroku). The 3 layers of our application are as follows:

- Front-end Android application
- Application server (Rails controllers and models)
- Backend database (PostgreSQL through Heroku)

We communicate from the front-end to the application server through API requests. The server responds back with JSON data. The application server communicates with the database using the ActiveRecord query interface, pre-built in Rails.

This is a simple use case diagram illustrating user interaction with our application:
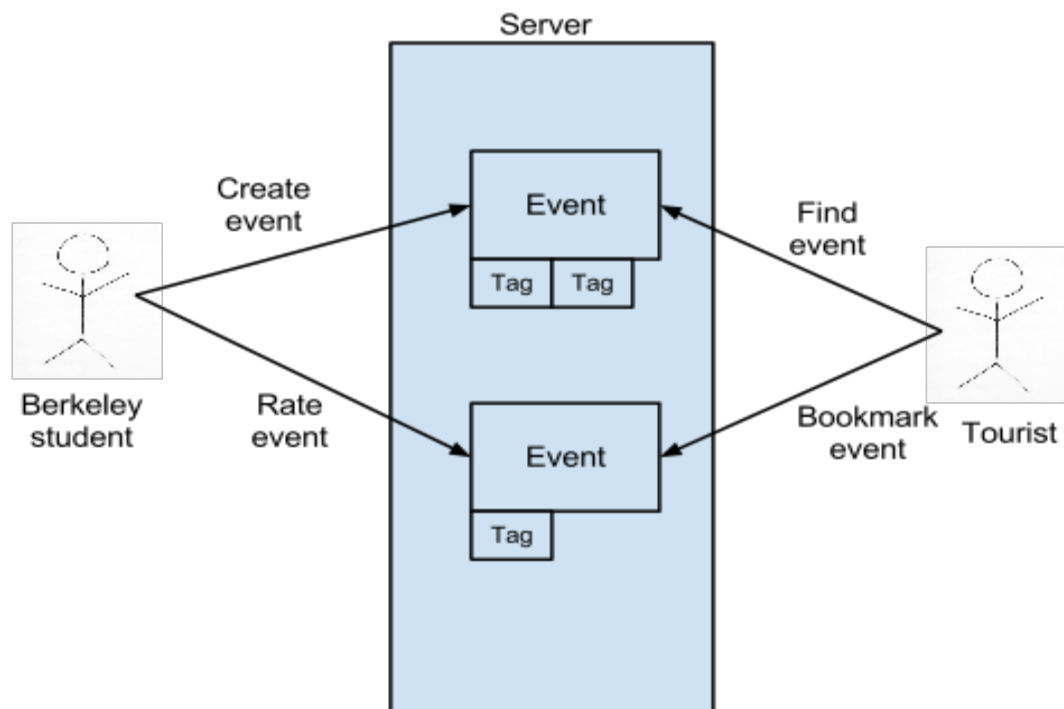
Design Details

Front-end Android Application Details:

The application will work with a variety of APIs to provide a full user experience. Using Facebook API, users will be able to login to the application through their Facebook accounts, where they can then search for events (and other users) that will be implemented through existing Android search interfaces and making API requests to our application server. Users will be able to see events using both Android ListView interfaces as well as map views through interacting with Google Maps API.

Application Server Details (Rails controllers and models):

Our Rails application server will process client requests and make queries to the backend database through the ActiveRecord library, which is an Object-relational mapping layer in Rails that provides a one-to-one mapping between a row in the relational table and an object instance of a class. This will allow for us to simply work with Ruby objects that will automatically query/modify the database. After retrieving the information from the backend database, the application server will utilize Rails controllers to send JSON objects back to the clients where we will have appropriate JSON parsing and handling logic in the front-end application level code.

Backend Database Details & ER Diagram:

The database layer will utilize the open source PostgreSQL relational database to act as a persistent data store. The ER diagram is shown below:
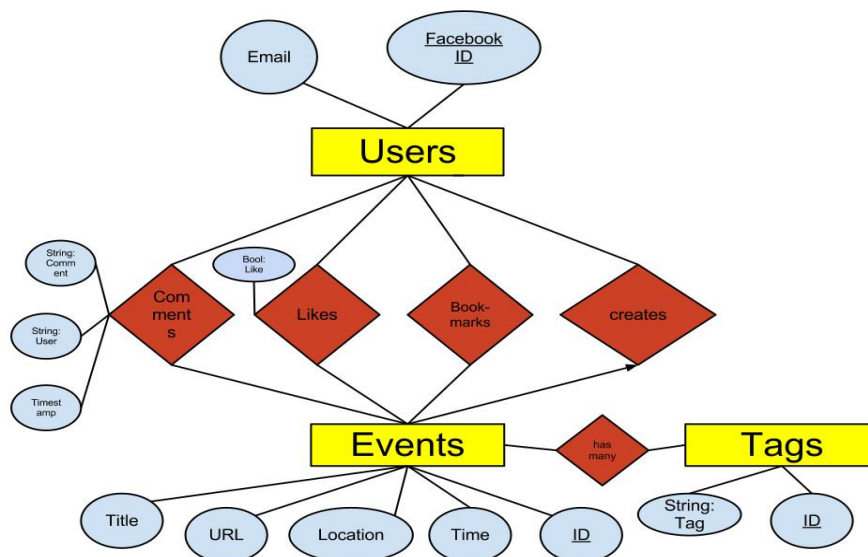
For iteration 1, we finished models for Users, Events, Likes, Bookmarks, and Comments, and we will be implementing Tags for iteration 2.

Invariants:
1. User
    a. Must have a valid Facebook account and email
2. Event
    a. Must have a creator/owner
    b. Must include a non-negative number of tags
3. Tags
    a. Must have exactly one creator/owner
4. Comments
    a. Must have exactly one creator/owner
5. Likes/Dislikes
    a. A user can like or dislike an event only once
6. Bookmarks
    a. One user can bookmark an event at most one time

Activity Details

Name: Main Page [1]
This is the main "home" page that the user sees upon logging into the app. The search bar will persist and stay constant across all views of the application.

Content	Component	Description
Search	Search Bar	Does the searching of events
Nearby	Button	Takes user to Listview[2] with nearby events
Subscriptions	Button	Takes user to the subscription page
Profile	Button	Takes user to the user's profile page
Bookmarks	Button	Takes user to Bookmark page

Name: List View [2]
The list view of the results of the user's search query, which the user can alternate with a Map View [3] as well.

Content	Component	Description
Home	Button	Takes user to Main page[1]
Search	Search Bar	Does the searching of events

Map	Button	Takes user to the Mapview[3]
Event List	Listview	Shows list of Events. Refer to Event right below
Event	Button	Takes user to Event Profile[4] page

Name: Map View [3]

Content	Component	Description
Home	Button	Takes user to Main page[1]
Search	Search Bar	Does the searching of events
List	Button	Takes user to Listview[2] with nearby events
Google Map	Map Fragment	Shows the Google Map
Event Profile	Pin/Popup	Takes user to Event Profile[4]

Name: Event Profile [4]
The page for events, containing information about events. Users can comment on the event, bookmark the event, and like and dislike the event itself.

Content	Component	Description		
Home	Button	Takes user to Main page[1]		
Event image	Image	Shows image of the event		
Event name	Text	Shows name of the event		
Event location	Text	Shows address/location of the event		
Event Time	Text	Shows date and time of the event		
Tags	Text	List of tags		
		Content	Component	Description
		Tags	Text	Text "Tags"
		Hashtag	Button	Takes user to Listview[2]

Like/Dislike Bar	Progress Bar	Show the ratio between the likes and the dislike for this event		
Like	Button	Increments counter for likes for the event in the database		
Dislike	Button	Increments counter for dislikes for the event in the database		
Comment	Listview	Content	Component	Description
		Username	Text	Username of user who commented
		Comment	Text	Description of the event

API calls:

New in Revision 2.0: every API call will now have a "session_token" parameter passed, so that our backend server can verify and authenticate the user's Facebook ID.

1. Users
The model used to record email and Facebook ID of every user.

POST /users/login.json
POST /users/add.json

{ "facebook_id": ….
  "email": …..
  "session_token": ….
}

Facebook ID and email are fields to be obtained from the Facebook Graph User upon logging into facebook on our app. The `login` method verifies whether or not a user of the given facebook id exists or not in our database, and the `add` method creates a user with the facebook_id and the email as fields of the new user.. The return JSON data is in the form:

{ "errCode": ….
  }

Where if errCode = SUCCESS, the user exists or has been created. Any negative number error code signifies an error occurred and that the particular Facebook user does not exist. The `add` API call will only be used if the user has not logged in before

POST /users/bookmarkEvent.json

{ "facebook_id": ….
  "event_id": …..
  "session_token": ….
}

Facebook ID is to be obtained from the Facebook Graph User upon logging into facebook on our app. The event_id will be the id of the event we want to bookmark. Upon posting the data, our database should have a bookmark relationship between the user and the event specified.

{ "errCode": ….
}

Where if errCode = SUCCESS, the bookmark has been created. Any negative number error code signifies that an error occurred when attempting to create a bookmark to an event.

POST /users/likeEvent.json

{ "facebook_id": ….
  "event_id": …..
  "like": …...
  "session_token": ….
}

Facebook ID is to be obtained from the Facebook Graph User upon logging into facebook on our app. The event_id will be the id of the event we want to rate. The like field will be a boolean true or false depending on if we liked it or disliked an event. Upon posting the data, our database should register the like between the user and the event.

{ "errCode": ….
  "alreadyLikedEvent": …..
}

Where if errCode = SUCCESS, the like has been registered, and whether the user has ever liked the event before. Any negative number error code signifies that an error occurred when attempting to like/dislike the event.

POST /users/removeLike.json

```
{ "facebook_id": ….
  "event_id": …..
  "session_token": ….
}
```

Facebook ID is to be obtained from the Facebook Graph User upon logging into facebook on our app. The event_id will be the id of the event we want to remove our rating from. Upon posting the data, our database finds the like/dislike that the user previously rated for an event and removes it from the database.

```
{ "errCode": ….
}
```

Where if errCode = SUCCESS, the like has been removed. Any negative number error code signifies that an error occurred when attempting to remove the like between a user and an event.

POST /users/postComment.json

```
{ "facebook_id": ….
  "event_id": …..
  "session_token": ….
}
```

Facebook ID is to be obtained from the Facebook Graph User upon logging into facebook on our app. The event_id will be the id of the event we want to post a comment to.

```
{ "errCode": ….
}
```

Where if errCode = SUCCESS, the comment has been posted. Any negative number error code signifies that an error occurred.

POST /users/deleteEvent.json
POST /users/cancelEvent.json

```
{ "facebook_id": ….
  "event_id": …..
  "session_token": ….
}
```

Facebook ID is to be obtained from the Facebook Graph User upon logging into facebook on our app, and is used to check if the user is the owner of the event. The event_id will be the id of the

event we want to permanently delete or simply cancel.

```
{ "errCode": ….
}
```

Where if errCode = SUCCESS, the event has been deleted or cancelled (meaning it is still inside our database). Any negative number error code signifies that an error occurred.

POST /users/restoreEvent.json

```
{ "facebook_id": ….
  "event_id": …..
  "session_token": ….
}
```

Facebook ID is to be obtained from the Facebook Graph User upon logging into facebook on our app. The event_id will be the id of the event we want to restore.

```
{ "errCode": ….
}
```

Where if errCode = SUCCESS, the event has been restored from a "cancelled" state. Any negative number error code signifies that an error occurred.

2. Events

Used to list all the events created. This is "owned" by one User only, but can be bookmarked, liked, and commented on by many other users.

POST /events/add.json

```
{ "facebook_id": ….
  "Title": ...
  "URL": ...
  "Location": …
  "session_token": ….
}
```

Title, URL, and Location will be user input by the creator of the event. The return JSON data is in the form:

```
{ "errCode": ….
```

```
            }
```

Where if errCode = SUCCESS, the tag has been added successfully. Any negative number error code signifies that an error occurred when attempting to create an event.

POST /events/search.json

```
{  "query": ….
    "facebook_id": ….
    "session_token": ….
}
```

"query" will be input by the user, will be used to find events most associated with this term, and "facebook_id" is used to check if the user owns each of the returned events. This method searches for events given a search term. The return JSON data is in the form:

```
{ "errCode": ….
    "events":  [    {              "location": …..
                                    "latitude": ….
                                    "longitude": …..
                                    "title": …..
                                    "url" : ….
                                    "start_time": …...
                                    "end_time": …...
                                    "canceled": …..
                                    "owner": ….
                        }
                        ……..
              ]

    }
```

Where if errCode = SUCCESS, the search has been successful. Any negative number error code signifies that an error occurred. "Events" returns a JSON array of events that match the query. Location signifies the string name of the event location, latitude and longitude refer to the GPS coordinates of the event. Title, url, start_time and end_time are self-explanatory attributes of the event. Canceled is a boolean indicating whether or not the event is cancelled, and "owner" is a boolean referring to whether or not the requesting user is the creator of the event.

POST /events/getRatings.json

```
{ "event_id": ….
```

```
    "session_token": ….
}
```

searchTerm will be input by the user, will be used to find events most associated with this term. This method searches for events given a search term. The return JSON data is in the form:

```
{ "errCode": ….
  "likes":.....
  "dislikes":......
}
```

Where if errCode = SUCCESS, the ratings query was successful. Any negative number error code signifies that an error occurred when attempting grab the event ratings. "Likes" and "dislikes" contains ratings information about the event.

POST /events/getComments.json

```
{ "event_id": ….
  "session_token": ….
}
```

searchTerm will be input by the user, will be used to find events most associated with this term. This method searches for events given a search term. The return JSON data is in the form:

```
{
        "errCode": ….
        "comments": [    {    "facebook_id": …..
                              "created_at": ….
                              "firstname": …..
                              "lastname": …..
                              "comment" : ….

                         }
                         …..
                    ]
}
```

Where if errCode = SUCCESS, the comments query was successful. Any negative number error code signifies that an error occurred. "Comments" contains a JSON array of comments.

POST /events/getEvent.json

```
{
```

```
        "event_id": ….
        "facebook_id": …...
        "session_token": ….
}
```

Event_id is the event we want to look up, and facebook_id is for identifying whether or not the user is the owner of the event.The return JSON data is in the form:

```
{
        "errCode": ….
        "event":    {    "location": …..
                         "latitude": ….
                         "longitude": …..
                         "title": …..
                         "url" : ….
                         "start_time": …...
                         "end_time": …...
                         "canceled": …..
                         "owner": ….
                    }

}
```

Where if errCode = SUCCESS, the event query was successful. Any negative number error code signifies that an error occurred. Location signifies the string name of the event location, latitude and longitude refer to the GPS coordinates of the event. Title, url, start_time and end_time are self-explanatory attributes of the event. Canceled is a boolean indicating whether or not the event is cancelled, and "owner" is a boolean referring to whether or not the requesting user is the creator of the event.

3. Tags
Used for alternative method of searching for Events.

POST /tags/add.json

```
{ "tagName": ….
   "event_id": …..
   "session_token": ….
}
```

tagName will be retrieved from the tag string that was typed in, and the event_id will be the ID of event that the tagging was done. This method creates a tag with name and event_id. The return

JSON data is in the form:

```
{ "errCode": ….
}
```

Where if errCode = SUCCESS, the tag has been added successfully. Any negative number error code signifies that an error occurred when attempting add a tag.

POST /tags/searchByTag.json

```
{ "tagName": ….
  "session_token": ….
}
```

tagName will be retrieved from the tag string that was typed in. This method searches for the events with given tag. The return JSON data is in the form:

```
{ "errCode": ….
  "searchResults": ...
}
```

Where if errCode = SUCCESS, the tag has been added successfully. Any negative number error code signifies that an error occurred when attempting to search for events associated with given tag.
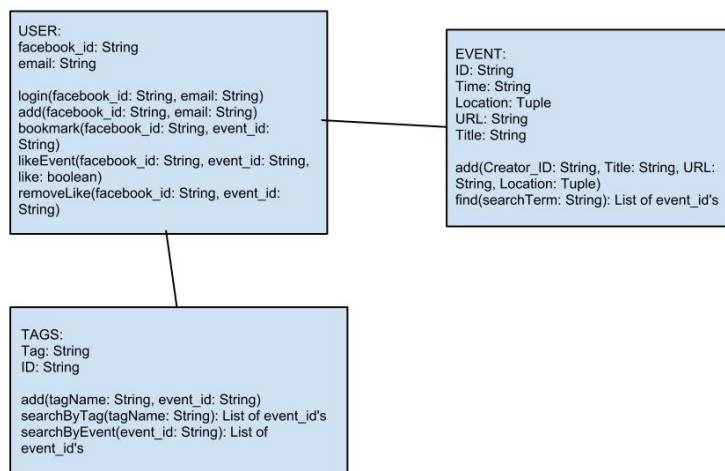
POST /tags/searchByEvent.json

```
{ "event_id": …..
  "session_token": ….
}
```

Event id will be retrieved by searching the Event's name in the Event table then retrieving the id of it. This method searches for tags associated with Event of given event_id. The return JSON data is in the form:

```
{ "errCode": ….
  "searchResults": ...
}
```

Where if errCode = SUCCESS, the tag has been added successfully. Any negative number error code signifies that an error occurred when attempting to search for list of tags given an event.

Here is a class diagram (iteration 1) showing the relationships between our models and their various fields and methods:

```
USER:
facebook_id: String
email: String

login(facebook_id: String, email: String)
add(facebook_id: String, email: String)
bookmark(facebook_id: String, event_id:
String)
likeEvent(facebook_id: String, event_id: String,
like: boolean)
removeLike(facebook_id: String, event_id:
String)
```

```
EVENT:
ID: String
Time: String
Location: Tuple
URL: String
Title: String

add(Creator_ID: String, Title: String, URL:
String, Location: Tuple)
find(searchTerm: String): List of event_id's
```

```
TAGS:
Tag: String
ID: String

add(tagName: String, event_id: String)
searchByTag(tagName: String): List of event_id's
searchByEvent(event_id: String): List of
event_id's
```

Implementation Plan

Our implementation plan is geared toward building the core functionality of the app in the first iteration (adding events, searching for the events, seeing the events displayed in proper fashion, and Facebook login) and adding on features from there. This is represented by the first 4 user stories. Stories marked with "(DONE I1)" means they were finished in iteration 1, and the remaining stories are ones that will be completed in iteration 2, except for story #11, which is planned for iteration 3.

User Stories
1. Users should be able to search for an event using the search bar (3 points)
   a. Building the search bar UI on the Android App **(DONE I1)**
   b. Building the backend that takes a search query and returns a list of the most relevant results **(DONE I1, but search algorithm will be improved for I2)**
   c. Implementing functionality for searching not only by event title, but also by location
      i. Difficulty: 1 point
      ii. Dependencies: 1a
2. Adding Event to our Database (2 points)
   a. Allowing User to choose a location of an event using the Google Maps API
      i. Difficulty: 1 point
      ii. Dependencies: N/A
   b. Building a simple form users can fill out to add an event
      i. Difficulty: 0.5 points
      ii. Dependencies: 2c
   c. Implementing the backend to add the new event to our database **(DONE I1)**
3. User should be able to log into the app with Facebook (2 points)
   a. Add Facebook SDK code that allows user to log into facebook so we can use Facebook sessions to identify users **(DONE I1)**
   b. Building backend that allows us to create and "login" into user accounts **(DONE**

I1)

4. Switching between ListView and MapView (2 Points)
    a. Use Google Maps API to get the map and pin locations **(DONE I1)**
    b. Create ListView using the data returned by the search query **(DONE I1)**
5. User should be able rate events (2 points)
    a. Build the UI that allows users to thumbs up or thumbs down event.
        i. Difficulty: 1 point
        ii. Dependencies: 2c, 5c
    b. Build UI to show current number of likes and dislikes **(DONE I1)**
    c. Incorporate backend that registers user likes and dislikes **(DONE I1)**
6. Adding comments to an event (2 points)
    a. Build UI form that allows user to comment on an event and submit the comment **(DONE I1)**
    b. Build UI to show the comments for an event **(DONE I1, but still needs bug fixes for I2)**
    c. Incorporate backend that saves comments in the database **(DONE I1)**
7. Deleting comments from an event
    a. Building UI to allow user to delete their comments from an event
        i. Difficulty: 1 point
        ii. Dependencies: 6b
    b. Update the backend to delete comments from the database
        i. Difficulty: 1 point
        ii. Dependencies: N/A
8. User should be able to bookmark an event and subsequently remove that bookmark (2 points)
    a. Add a "Bookmark" button to every event
        i. Difficulty: 0.5 points
        ii. Dependencies: 2c, 7b
    b. Incorporate backend to save bookmarks for a user **(DONE I1)**
    c. Add a "Remove Bookmark" button for an event, if it was bookmarked by the user
        i. Difficulty: 0.5 points
        ii. Dependencies: 2c
    d. Incorporate backend to remove bookmarks for a user
        i. Difficulty: 0.5 points
        ii. Dependencies: 2c
9. User should be able to cancel events (1 point)
    a. Add a "cancel" and "resume" button to every event
        i. Difficulty: 0.25 points
        ii. Dependencies: 2c, 8b
    b. Incorporate backend to set an event canceled as T or F in the database and inform users who bookmarked event that it was canceled **(DONE I1)**
10. User should be able to delete an event entirely (1 point)

a. Add a "delete" button to every event and prompt user with "Are you sure?"
    i. Difficulty: 0.25 points
    ii. Dependencies: 2c, 9b
b. Incorporate backend to remove event from the database entirely and inform users who bookmarked event that it has been deleted. **(DONE I1)**
11. User should be able to add pictures to an event
    a. Implement an "Add picture" button along with a properly sized thumbnail/photo of the event on the event page
        i. Difficulty: 1.5 points
        ii. Dependencies: N/A
    b. Save the pictures to the event on the backend
        i. Difficulty: 1.5 points
        ii. Dependencies: 11a

External Libraries

Facebook SDK: For allowing users to login and use the application, as well as authentication purposes.
Google Maps API: For providing our app's interactive map view.
paperclip: A Ruby gem that will be used as an easy file attachment library for Active Record. This will be used for attaching pictures with events.
sunspot: A Ruby gem that will provide Apache Solr powered full text search for our Ruby objects. This will most likely be integrated with our searching functionality.
simplecov: A Ruby gem that will provide a code coverage analysis tool for us to use. We will check to make sure that our tests will run every line of code at least once in our code base.
geokit: A Ruby gem that will provide geocoding and distance calculations for our app. This will allow us to find events that are in close proximity to the user.

Group Task Assignments (Iteration 2)
Andy - 2a, 2b
Benjamin - 1b, 1c
Eric - 6b, 7a, 7b
Jerry - 8a, 9a, 10a
Victor - 5a, 7b

Risk Assessment
In general, there are a number of risks associated with our application:
1. Users may add the same events over and over again, creating spam events. Malicious users may even add fake events that do not exist in real life.
    a. We will mitigate this through implementing some sort of event flagging function, where if enough users flag a spam event it will be set up for automatic deletion.
2. Deciding event editing permissions also presents a risk for our design. If we allow any

user to be able to edit events, then potentially users can tamper with events and falsify information.

    a. The solution to this is to simply only give event editing permissions to the users who created/own the event. Though this does present different problems when the user may not be able to edit events in a timely manner, the associated risk is considerably smaller than letting any user edit an event.

3. Users may abuse the system by creating multiple accounts and rating/reviewing their own events to create false reliability.

    a. Our application design for logging in users only using Facebook API prevents the users from abusing this system. Allowing users to make their own accounts is something that we will most likely accomplish for the next iteration (3)

4. There is no authentication for this iteration, meaning anyone who can make the API calls to our server can provide someone else's facebook id and post, delete, etc. events for that person

    a. We would start requiring each POST request to include a session token that we would use for user authentication. This will be completed by the end of iteration 2.

Testing Plan

Overall: We will be using some form of automatic testing for our models and controllers and will be manually testing our UI on our Android devices.

1. Unit Testing
   a. We will be using the behavior-driven development framework Rspec in order to test our models, since Rspec has good integration with our backend Rails implementation.
   b. We plan on writing unit tests for all of our models and associations, including User, Event, and Tag entities as well as Likes, Bookmarks, and Comments relations. Users and Events will be unit-tested first as they are the backbones of our system and without ensuring the correctness of those models, testability of the other system components will be jeopardized.

2. Functional Testing
   a. We will be using Rspec as well for our functional testing purposes, to verify that our system implementation matches our original designs.
   b. We will stub model objects when necessary.
   c. We will be testing all the API methods that are used directly by our front end Android application.

3. UI testing
   a. We plan on testing the User Interface thoroughly, through actual Android devices as well as virtual Android emulators. For iteration 2, we will additionally work with `uiautomator` and `uiautomatorviewer`, two tools that are provided by the Android SDK for automated, functional UI testing for our testing needs.
   b. Most of the UI testing will only involve making sure that buttons work correctly and

that pages display properly, with all the various API components being able to be shown on the Android screens.