RedPins
Design and Planning Document
03/03/2013
Version 1.0
Members: Andy Lee, Benjamin Le, Eric Cheong, Jerry Chen, Victor Chang

Rev 1.0 2013-03-03 -initial version

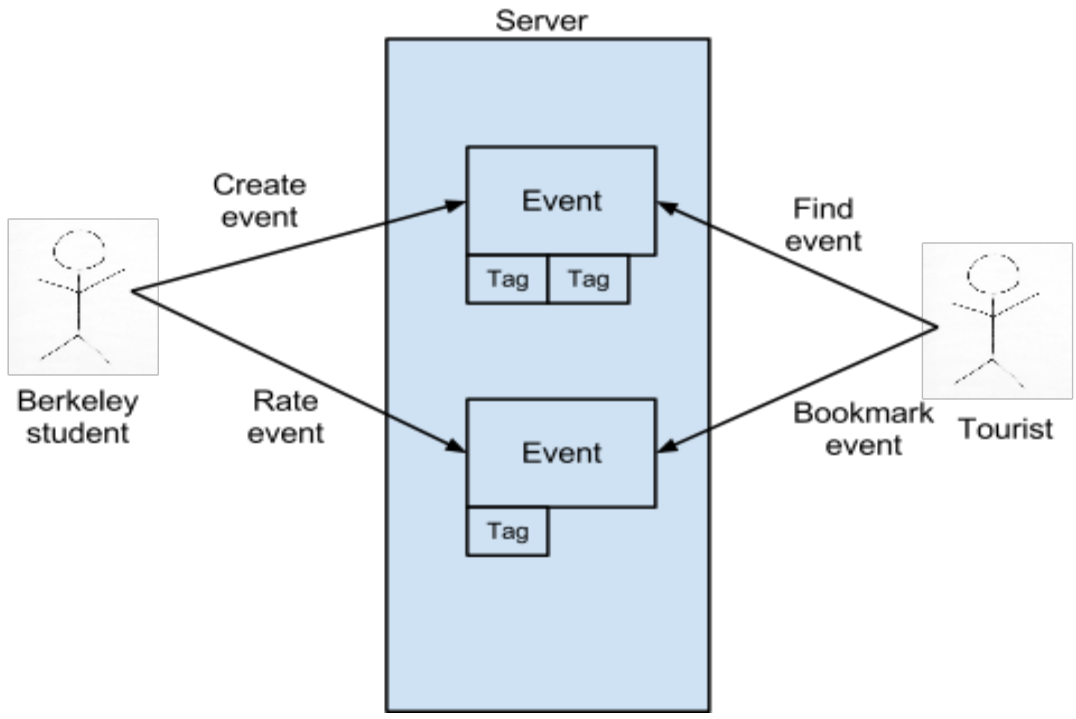
System Architecture

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, most likely 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:

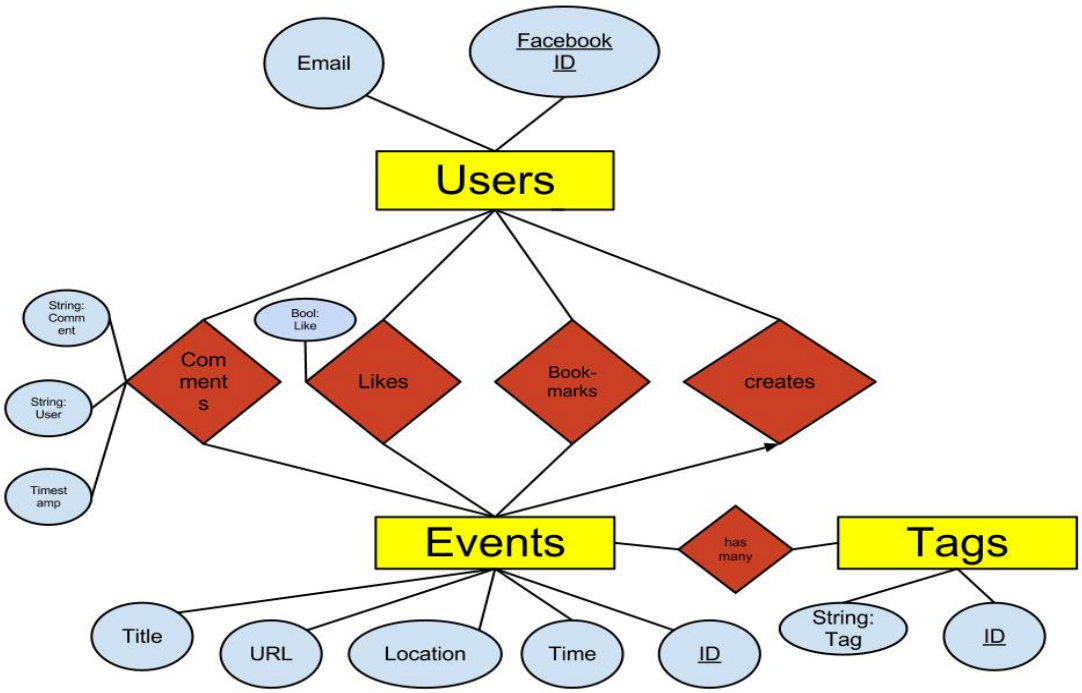
The application will work with a variety of APIs to provide a full user experience. Using Facebook API, users will be able to login to the application through their Facebook accounts, where they can then search for events (and other users) that will be implemented through existing Android search interfaces 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 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:



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
History	Button	Takes user to History page
Hashtags	Tab	searches by hashtags
Users	Tab	searches by username

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	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

<table>
<tr><td></td><td></td><td><table>
<tr><td>Tags</td><td>Text</td><td>Text “Tags”</td></tr>
<tr><td>Hashtag</td><td>Button</td><td>Takes user to Listview[2]</td></tr>
</table></td></tr>
<tr><td>Like/Dislike Bar</td><td>Progress Bar</td><td>Show the ratio between the likes and the dislike for this event</td></tr>
<tr><td>Like</td><td>Button</td><td>Increments counter for likes for the event in the database</td></tr>
<tr><td>Dislike</td><td>Button</td><td>Increments counter for dislikes for the event in the database</td></tr>
<tr><td>Comment</td><td>Listview</td><td><table>
<tr><td>Content</td><td>Component</td><td>Description</td></tr>
<tr><td>Username</td><td>Text</td><td>Username of user who commented</td></tr>
<tr><td>Comment</td><td>Text</td><td>Description of the event</td></tr>
</table></td></tr>
</table>

API calls:

1. Users

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

POST /users/login.json

{ “facebook_id”: ….
“email”: …..
}

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

{ “errCode”: ….
}

Where if errCode = SUCCESS, the user exists. Any negative number error code signifies

an error occurred and that the particular Facebook user does not exist.

POST /users/add.json

{ “facebook_id”: ….
“email”: …..
}

Facebook ID and email are fields to be obtained from the Facebook Graph User upon logging into facebook on our app. This 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 has been created. Any negative number error code signifies that an error occurred when attempting to create a new user. This API call will only be used if the user has not logged in before.

POST /users/bookmark.json

{ “facebook_id”: ….
“event_id”: …..
}

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”: ……
}

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”: ….
}

Where if errCode = SUCCESS, the like has been registered. 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”: …..
}

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.

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

{ “Creator_ID”: ….
“Title”: ...
“URL”: ...
“Location”: ...
}

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/find.json

{ “searchTerm”: ….
}

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”: ….
“searchResult”
}

Where if errCode = SUCCESS, the search has been successful. Any negative number error code signifies that an error occurred when attempting add a tag. searchResult contains list of event_id’s that match the searchTerm.

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

POST /tags/add.json

{ “tagName”: ….
“event_id”: …..
}

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”: ….
}

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

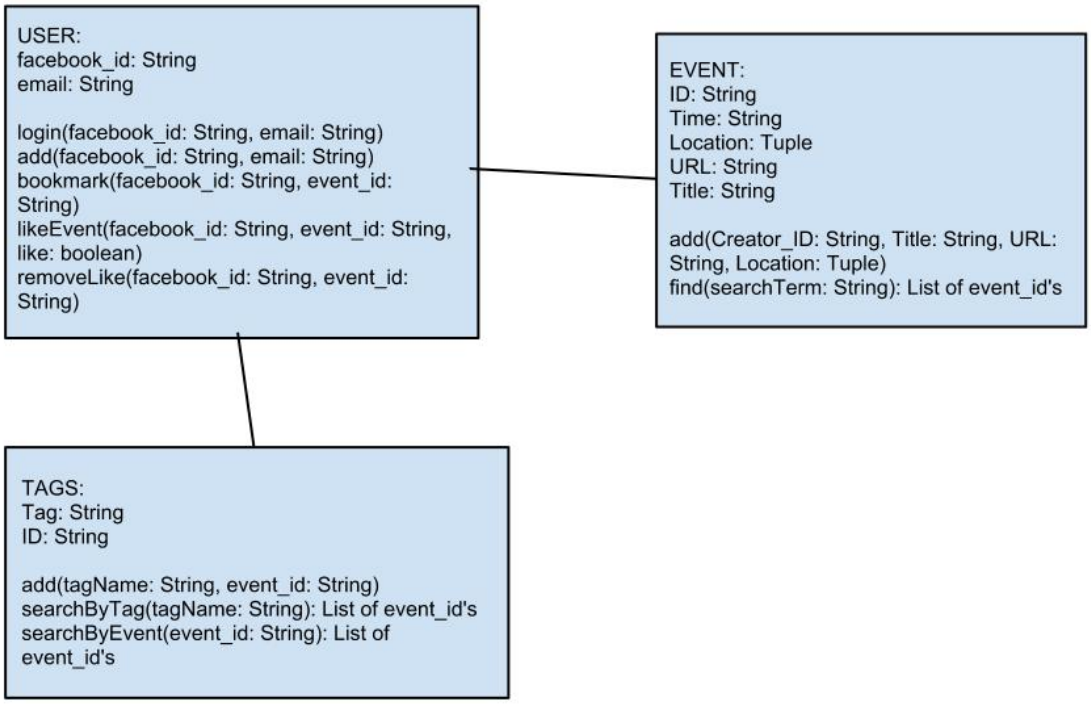
{ “event_id”: …..
}

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 showing the relationships between our models and their various fields and methods:



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.

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
        i. Difficulty: 1 point
        ii. Dependencies: 1b
    b. Building the backend that takes a search query and returns a list of the most relevant results
        i. Difficulty: 2 points
        ii. Dependencies: 2a, 2b, 2c
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
        i. Difficulty: 0.5 points
        ii. Dependencies: N/A
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
        i. Difficulty: 1 point
        ii. Dependencies: N/A
    b. Building backend that allows us to create and “login” into user accounts
        i. Difficulty: 1 point
        ii. Dependencies: N/A
4. Switching between ListView and MapView (2 Points)
    a. Use Google Maps API to get the map and pin locations
        i. Difficulty: 1 point
        ii. Dependencies: 1a, 1b
    b. Create ListView using the data returned by the search query
        i. Difficulty: 1 point
        ii. Dependencies: 1a, 1b
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
        i. Difficulty: 0.5 point
        ii. Dependencies: 2c, 5c
    c. Incorporate backend that registers user likes and dislikes
        i. Difficulty: 0.5 points
        ii. Dependencies: 2c
6. Adding comments to an event (2 points)
    a. Build UI form that allows user to comment on an event and submit the comment
        i. Difficulty: 0.5 points
        ii. Dependencies: 2c, 6c
    b. Build UI to show the comments for an event
        i. Difficulty: 1 point
        ii. Dependencies: 2c, 6c
    c. Incorporate backend that saves comments in the database
        i. Difficulty: 0.5 points
        ii. Dependencies: 2c
7. User should be able to bookmark an event (1 point)
    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
        i. Difficulty: 0.5 points
        ii. Dependencies: 2c
8. 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
        i. Difficulty: 0.75 points
        ii. Dependencies: 2c
9. 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.
        i. Difficulty: 0.75 points
        ii. Dependencies: 2c

<u>Risk Assessment</u>

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.
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 “token” that is the session token for logging in after authenticating with Facebook.

Group Task Assignments (only includes first iteration tasks)
Andy - 2a, 2b, 2c
Benjamin - 3a, 3b
Eric - 4a, 4b
Jerry - 1b
Victor - 1a, 1b

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 integration.
    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 manually, through actual Android devices as well as virtual Android emulators. This is because finding and learning a framework for UI testing is probably more work than it is worth.
    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.