Campus Feed

Team 2

Anubhaw Arya, Matt Gotteiner, Daniel Green, Mayank Jethva,

Sean McCullough, Pranav Menon, Mengyao Wang

Purpose

This app will solve the problem of event fragmentation on campus by providing one central location to post events. The interface will be intuitive and will provide an easy way for campus organizers to promote their events and for students to stay informed about the events happening on campus.

Non-Functional Requirements

- **Availability**
    - We need our site to be online at all times, especially when major events are happening and large numbers of users are viewing events on the application. To solve this problem, we will use Amazon Elastic Instances and Amazon Content delivery networks to help scale our application to serve a large volume of requests. Since Amazon can auto scale our Elastic instances, we will not have to manually configure our servers to handle high traffic. Availability is critical to the function of our application. If there are very popular events happening on campus, and our servers cannot handle the high request volume, then our application will likely crash. With the solutions listed above, we will not face this problem during periods of high usage.
- **Usability**
    - We want our mobile application to have a simple and intuitive design to allow users to perform their tasks (adding, viewing, and RSPVing to events) quickly and easily. The interface should be clutter free and only provide the information the user really wants at the time. For example, when the user is browsing events, the most important information to the user is what the event is (does it interest them?) and when the event will be taking place. Extra information such as event

location, event description and event coordinator are secondary and available upon further request (clicking the event). Navigation throughout the app should also be intuitive and enable the user to get the information they want easily. By putting usability at the forefront of our design, we will enable users to bring the application into their daily lives with ease.

- **Performance**
    - We want our application to be as responsive as possible. Lag between user input and application response should be minimized as much as possible. To accomplish this we will start with the main page of the event only showing select popular events in each category. This will decrease the network usage of the application, http request response time, and application launch time. It will also load event details as a user views an event, and then cache the http responses for the event details. This will help reduce server load and allow the client to give quick responses to the user.
- **Security**
    - The server and client application should provide the best security it can to protect user information. Also, we do not want malicious users to spam our application with garbage events. To provide this non functional requirement, we will use OAUTH & HTTPS using Facebook Authentication to implement our login system. We will never store the actual passwords, we will only store user emails and OAUTH keys in encrypted form. In addition, to protect against spam we will implement a reporting system which will allow users to report events that they feel are spam. Using the user reports we will judge the event and follow the necessary actions such as deletion. We will also rate limit malicious users from trying to put

many garbage events in our database. We need security since users will stop using the application if any password and personal information gets leaked, and having many garbage events in our listing will hurt the integrity of our application.

Design Outline

Architecture Model: Client - Server model.

Using a client server model in our application will help us create a nice abstraction between client requests and the server/database. Using HTTPS requests, we will be able to send secure messages between the client and the server. This model also gives us a centralized way to exchange information between our database and clients. One major consideration in making this decision was the large number of clients that our application could possibly have. Another consideration was the greater ease of maintenance and security of such a model. We will be able to update, remove, and repair implementations in our server without affecting any implementations in our client .

We have four main components (our decisions for each component are listed below its title):

Server: Middleman between client and database

Our server will be implemented in an Representational State Transfer (REST) API. The REST API system will not only help make our servers easily scalable and perform operations quickly in high amounts traffic, it will also provide simple interfaces that our clients can use to interact with the server. We will be able to provide even better response times by caching messages sent from the server. Also, if we decide to increase the functionalities of our

application, we will be able to add features without having to greatly alter our client implementation.

In addition, the server will use the JSON data model for message handling, since we will be able to read and write messages quickly using the JSON key-value pair system. By using the JSON data model, we will also not have to write our own foolproof message specifications that the client and server would use to read and write. We will be using the apache Tomcat server framework and this will help us jump start development of our server message handling programs. With Apache Tomcat using java as the main framework language, our whole team will be able to understand the server programs, since everyone on our team is familiar with Java (especially more than with other server technologies such as PHP).

Scraper: An automated event information collector

One of the main components of our system is the event scraper. This is fundamental to our application because we believe we can outdo the competition by allowing users to post their events . The scraper will be written in JAVA and use the HTML parsing library called, JSOUP. It will repeatedly scrape events from Boiler-Link and export those events directly to our application database. The scraper will also be intelligent enough to check for events already listed in our database, and to be able to update details of existing events that are different between the database and Boiler-Link.

Clients: Receive information from the server and present it to users. Clients can also send information to the server (eg: creating an event)

The main client that we will be making is an Android client. We have done research on

which type of client we should make, and have decided on Android because of the vast majority of users of the platform, and the familiarity our team has with Java and the Android software development kit. When writing our client, we will put performance and maintenance at a high level on the hierarchy since a slow/unmaintainable mobile application will deter users. To do so, we will separate UI classes from background data operation classes. Having data operations in the main UI threads will slow down and even crash mobile applications. By having this separation, we will also be to keep our code organized. We will also write the android application to be a modular as possible. We will use modularity as a stronghold so that updating any class or function will impact outer classes minimally.

Database: Store and organize information received from scraper and clients and send this information to clients upon request (via the server)

Of the major kinds of databases available, our team has decided to use the relational database management system known as MySQL. We found that the relationship-based database model was very convenient to use, especially since it can easily be used to emulate the UMLs we've been designing along with our plans for Campus Feed. This convenience will allow us to create a more sensible and robust system as it'll help us prevent redundancies in structure and better flow in data. In addition to the convenience, MySQL has a large community built around it, which will be very helpful for when we need to create some of the more advanced queries.

Design Issues

- Event Display:
    - Options: Display all events in one list, display based on category, display based

on tags

We chose to display events based on categories and tags in order to only display events that interest the user. We do not want to display a large list of events to the user as many events may be irrelevant to the user.

- Platform
    - Options: Android, iOS, Windows Phone, Web

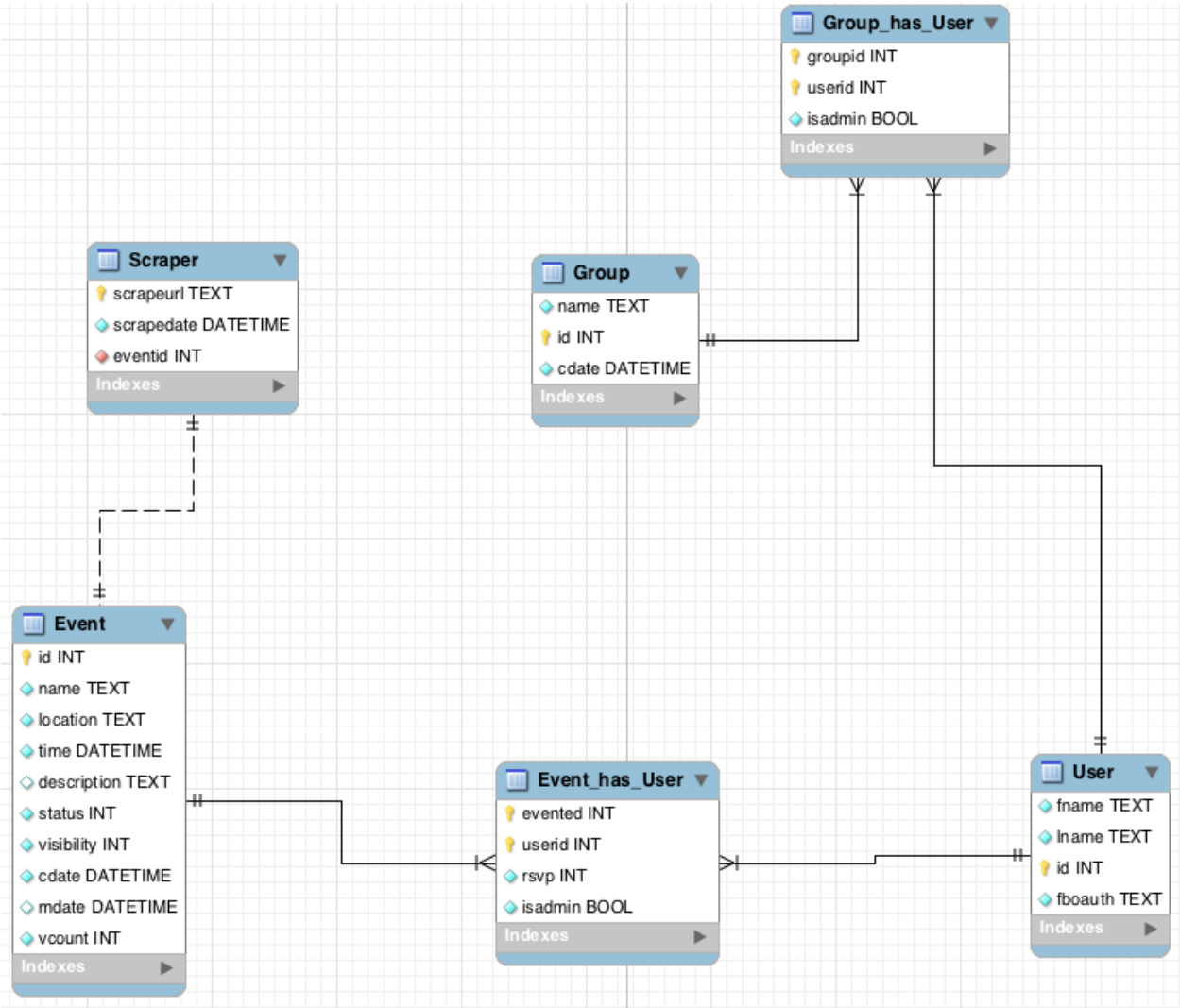
We chose to develop this app for Android since it was the platform we are all most familiar with. This will allow for faster development and a better quality product in the end.

- User base
    - Options: College students only, Everyone

We chose to make this app open to everyone as there might be people who live near the college who would want to attend events on campus.

Design Details

Main Classes:



1. **Scraper**
    a. The scraper table will be used by the scraper to check if an event has been scraped and when it was last updated. This will allow the system to be fully automated in terms of what events to scrape and when to do so.
    b. scrapeurl
        i. The scraped event's original url

c. scrapedate
    i. The date is was last checked
d. eventid
    i. The id of the created event
    ii. 1 to 1 relation with the event table's id, which will link the scrape url with the corresponding event

2. Event

a. This table will hold all of an event's details
b. id
    i. The event's unique id
c. name
    i. The event's name
d. location
    i. The event's location
e. time
    i. The event's time
f. description
    i. The event's description
g. status
    i. The event's status
        1. Ongoing
        2. Delayed(
        3. Cancelled
h. visibility

        i. The event's visibility level
            1. Public
            2. Private
            3. Group
    i. cdate
        i. The event's creation date
    j. mdate
        i. The event's modification date
    k. vcount
        i. The event's view count (for popularity)

3. **Event_has_User**
    a. This table is an association table for the many to many relationship between a user and an event.
    b. eventid
        i. The event id
    c. userid
        i. The invited user id
    d. isadmin
        i. Tells us whether or not the user is the admin for this admin (for event editing purposes)
    e. rsvp
        i. Tells us the user's rsvp status
            1. Coming
            2. Maybe

3. Decline

4. **User**
    a. This table will contain all of a user's information
    b. id
        i. User's id
    c. fname
        i. User's first name
    d. lname
        i. User's last name
    e. fboauth
        i. User's OAuth key for Facebook (for logging in and out of the app)
5. **Group_has_User**
    a. This table is an association table between a group and users. This will help organize groups for mass invitations to events.
    b. groupid
        i. Group id
    c. userid
        i. User id
    d. isadmin
        i. Whether or not the user is an admin for the group
6. **Group**
    a. This table will hold all of the group's information
    b. name
        i. Group's name

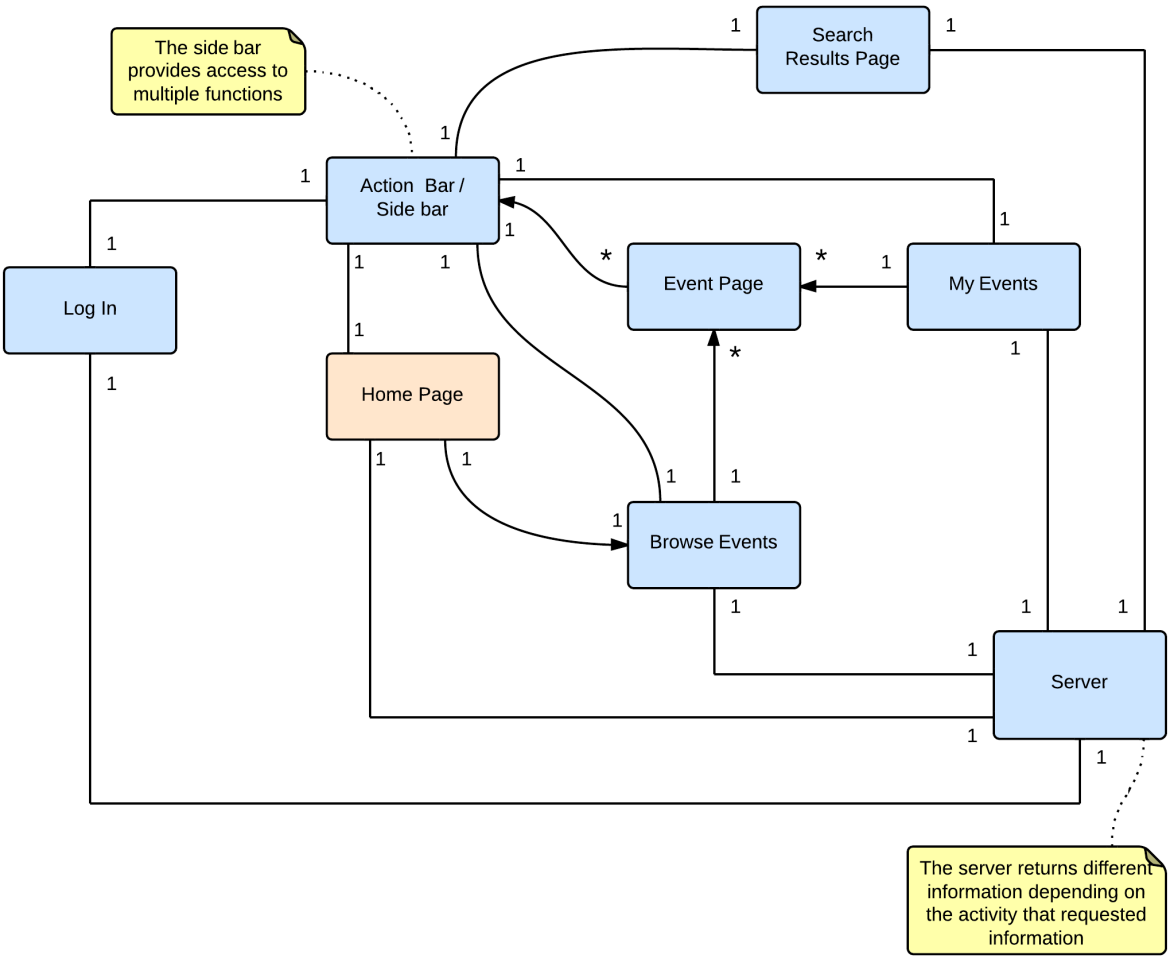
c. id

   i. Group's id

d. cdate

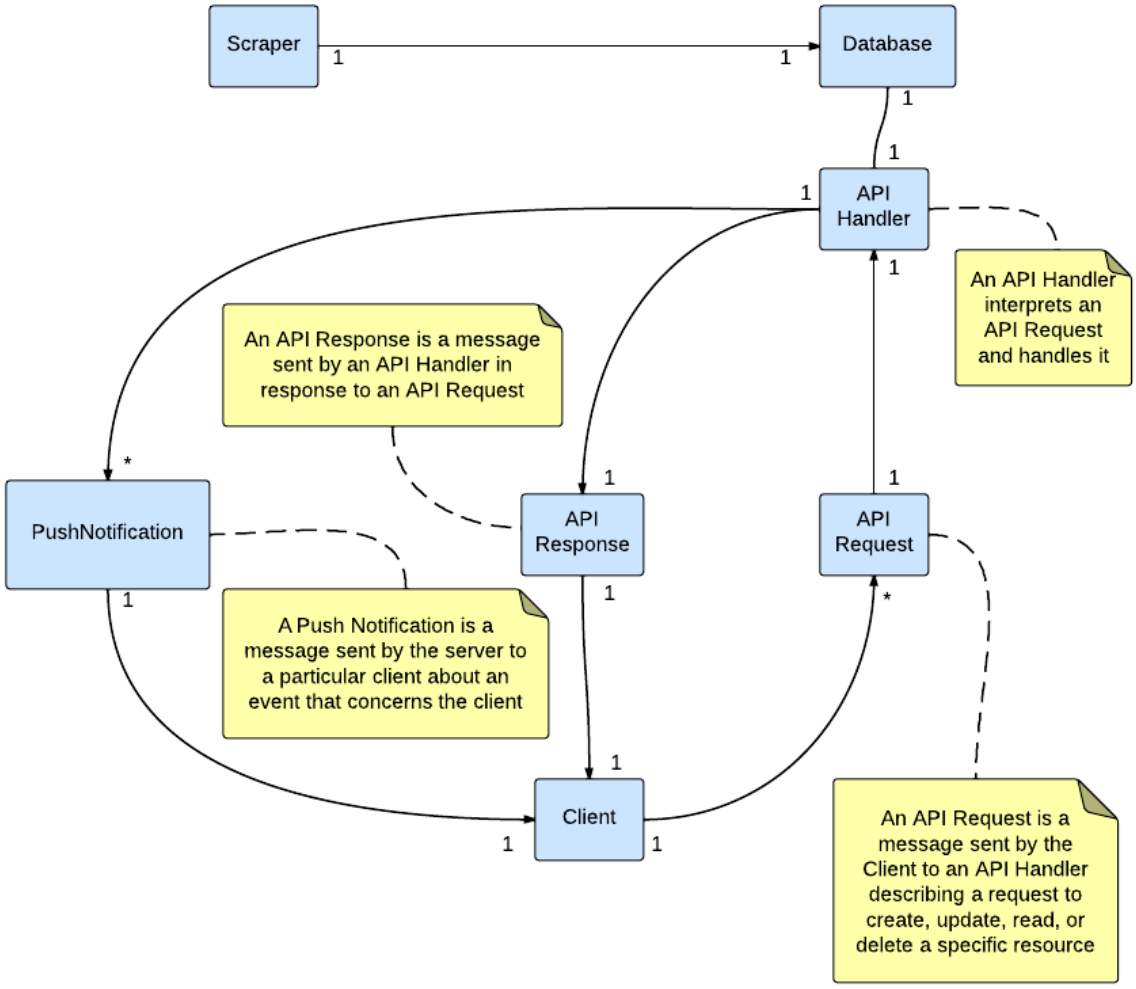
   i. The date the group was created

Android Activity Interactions:

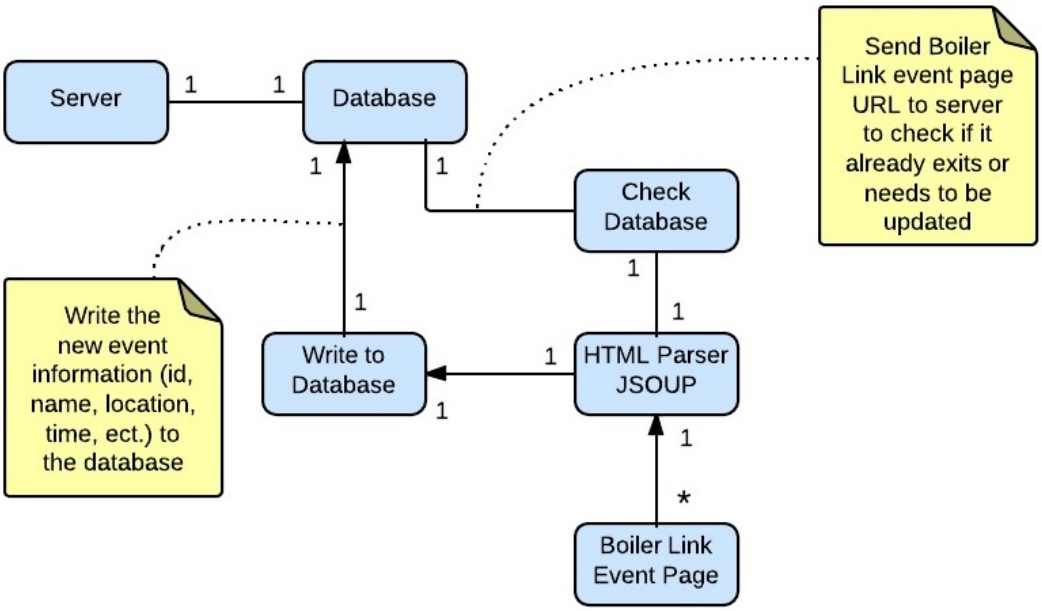


Android Activity Interactions
(Single User)

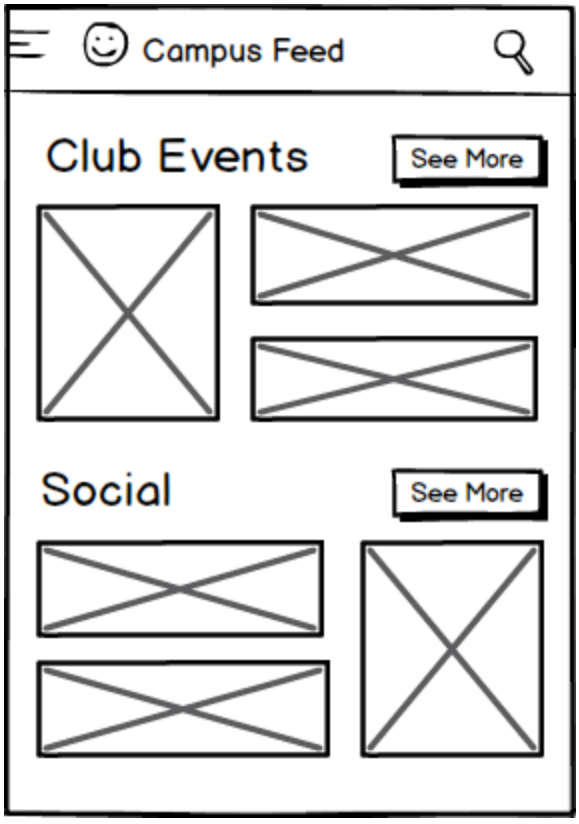
Server Interactions:



Scraper Interactions:



UI Mockups



Home Page

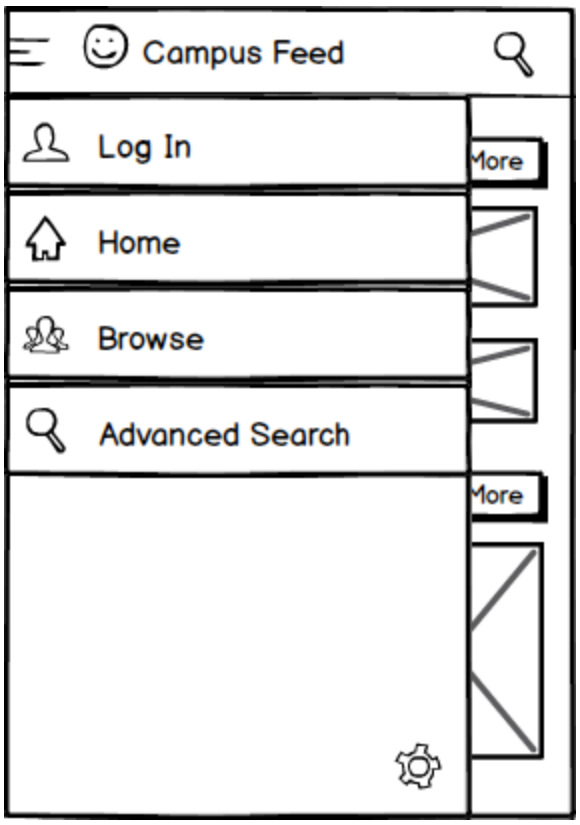
This is the main screen for the app. It serves as a central overview of the app and start location for the user. There will be several categories of events, showing the top events in each category (determined based on popularity and proximity to current date). Clicking an event will bring up the Event page Clicking the “See More” button will open the Browse screen to the tab of the given category.

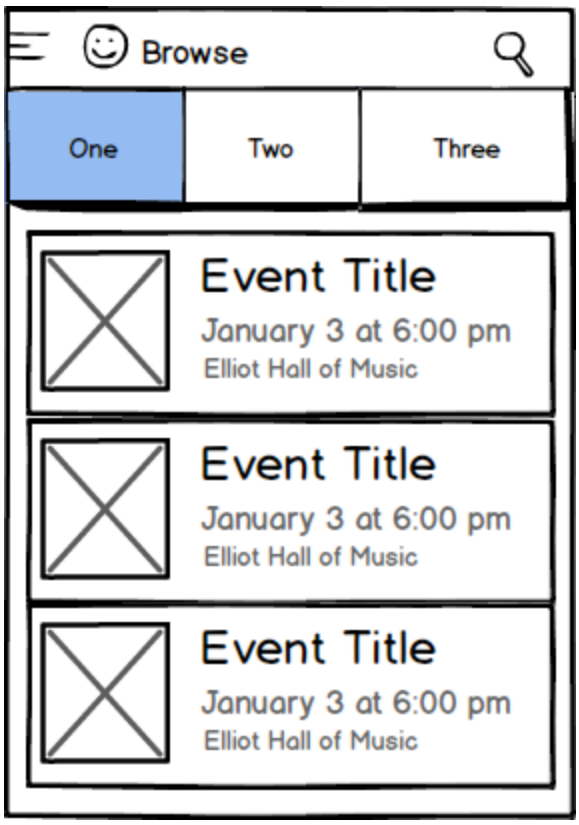
Sidebar

The sidebar will be always accessible via the action bar. It presents an overarching navigation for the app. When logged out, the user has the option to log in. When logged in, a “My Events” option appears after “Home.”

Advanced Search opens up the Advanced Search Options page which will then launch the search and navigate to the search results screen.

The gear in the lower right corner opens the Settings page which will have customization options and the logout option.





Browse

Accessed via the sidebar or the “See More” button of a specific event category. The tabs represent yet to be decided event categories. There will be more than three. Swiping to either side will navigate between the tabs. They will also be clickable.

Clicking an event navigates to the Event Page.

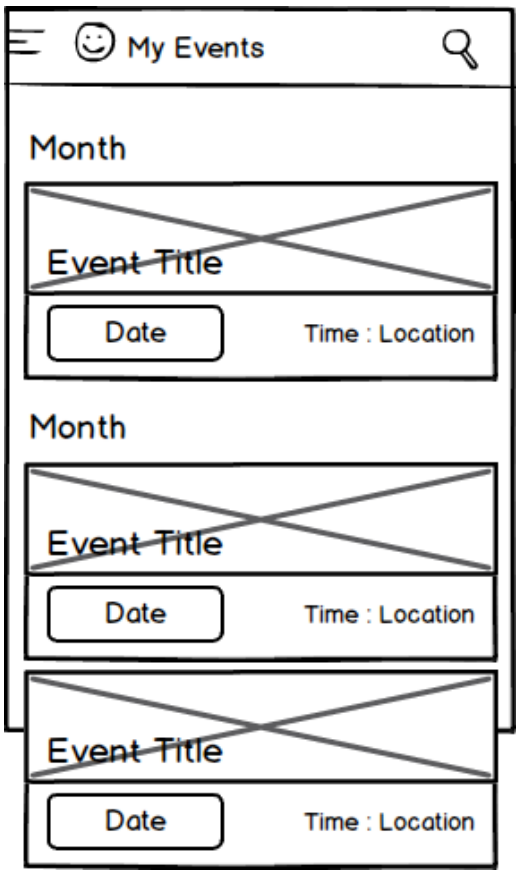
This view is infinitely scrollable.

My Events

Only available if a user is logged in.

Displays a list of the events for which the user has selected “Going” or “Maybe Going.” The image will be the event image uploaded by the event creator. Clicking the event navigates to the Event Page.

Events coming off the page indicate that the Event cards are large and that the view is scrollable.



Event Page

Displays all the information about the event and allows the user to indicate an RSVP status. They can also choose to invite others to this event.

If the Event description doesn't fit entirely in the information box, it will be cut off. Clicking "See More" expands the box to include the full description.

The "Event Coordinator" displays information about the user who created the event.