**CS 4720 - S17 - Final Project Proposal**

Device Name: Blastoise Platform: Android

Name: Zachary Skemp Computing ID: zrs2sb

Name: Apoorva Arunkumar Computing ID: aa3vs

App Name: Safe Nights

Project Description:

Our app, Safe Nights, provides dual functionality as a mechanism of keeping one safe when going out for a night of drinking as well as a personal tracking system for long-term alcohol consumption and money spending. Staying safe while participating in activities that causes loss of physical and mental capabilities can be a huge challenge. By creating an app that allows people to set a safe place at which they would like to end their night, users can have a "safety net" that alerts friends/family if for a number of reasons end up in a place they didn't necessarily want to be. Alcoholism is also a large problem in the United States. By providing a system that allows users to self-track their alcohol consumption after their safe night out, people can gain a greater insight into their alcohol activities and the effect it has on their lives long-term.

What we proposed to do is create an app that will do the following:

- The system shall track a user's location over the course of a night out drinking;
- The system shall send data by email to specified contacts if user is not where they want to be (previously specified by user);
- The system shall allow a user to store list of locations;
- The system shall allow a user to self-report their alcohol consumption for a specific date;
- The system shall graphically display the user's alcohol consumption and money spent on alcohol over time;
- The system shall allow a user to see their geolocation history from their previous night out (if for some strange reason they cannot remember); and,

We planned to incorporate the following features:

- GPS - The system will track a user's location throughout the night every 10-15 minutes
- Accelerometer/Shake - The device's acceleration will be used as a measurement of when a user is "done" for the night (passed out/fallen asleep/cannot move/etc.)
- Consume a self-built webservice - We will store the user's contacts, list of saved locations, and geo-locations over the course of a night in a relational database for a

web-service built in Django. The entries for how much a user drinks each night will also be stored in a relational database to record drinking habits over time.

- Open shared activity / features - Contacts specified by user will be sent an email specifying the user's start/end of a safe night, or the user's last location if the user's phone battery dies/app shuts down. Contacts will also be sent a notification if the user's accelerometer does not trigger for a 40 min period and the user is not where they previously specified they would like to end up.

Updated Final Wireframe Description:

Our wireframe shows the basic layout that we built for the application. After the splash screen appears, we have a login page where a user can log in, or choose to navigate to a signup page. If a user signs up, they are redirected to the login page, where upon successful authentication, they are directed to our set of 4 fragments. The default fragment is the Get Started screen, where a user can pick a final destination from his/her saved locations, or search for a new one instead. They also must enter in a contact's information (name, email) before hitting "Start Night", which starts a service that tracks the user's night out.

If a user switches the layout to the Add Drinks screen, they are met with a giant button to set the date they would like to enter data for, a SeekBar for entering the amount of money they spent, and a few number pickers for the drinks they consumed. Upon hitting submit, they are redirected to the History screen, which shows them a graph of their money and alcohol consumption for that month. This can be toggled to see previous months to see how he/she has progressed.

The final screen is a map screen (using Google Maps), where users can see their plotted points from their past (or currently ongoing) night. This screen also has a set of tabs in a list view along the left-hand side of the screen, each of which has a timestamp of the moment the location data was uploaded. Users can click on the timestamps or zoom in on the map to see where they were at any particular time or location.

Note that the images for the wireframe are appended to this documentation.

Platform Justification:

Additionally, there are a large number of open-source Android packages available for anyone to use, which was a huge benefit to us. A large portion of our UI for navigation, number-pickers, seek-bars, and other widgets all came from libraries that we were able to find online. Because of the large amount of users and open-source libraries Android has, it was also relatively easy to find multiple ways of going about each task, which was helpful since this project was fairly open-ended. Also, if we were to deploy this application (which we intend to eventually), Android has a very large share of the market, as around 90% of mobile apps are built on the platform.

In terms of real world benefits, both of us were fairly experienced in Java and the development environment for Android is fairly easy to use, so the ability to conduct rapid prototyping was a big help here.

Major Screens:

- Login/Create Account: Contains the UI and logic for registering or logging in to our app. Users have a name, email, username, and password. This functionality is achieved through SharedPreferences.
- Get Started: The first screen users will see after logging in, and is where users can state that they are about to go out for a night of fun. Users can search and select where they expect to end up at the end of the night, and list an emergency contact with his/her email. Upon clicking “Start”, this starts a service which tracks the user’s location every 10 minutes and uploads it to our web-service. After 2am, if a user isn’t where they intended on going, haven’t “Stopped Their Night”, and hasn’t moved for 40 minutes, then a message is sent to their emergency contact. A message is also sent to this contact with the user’s last known location if their battery level runs too low (<10%) or if their app suddenly crashes.
- Add Drinks: A page where users can select what date they went out on and input in how much money they spent along with how much they drank. These results are stored in our database and used for tracking their drinking/spending progress.
- History: A page that contains 2 graphs, one for how much the user has spent that month, and one containing their approximate levels of intoxication for each time they went out that month. Users can toggle the month to see their past history and see their progress.
- Last Night: A Google Maps UI page that plots all the location points from where the user was last night. Along the left side of the screen is a timeline with the recorded times from the night, and users can zoom in or select the times along the screen to see where they were at each point in the night.

Optional Features:

1. Build/Consume a web-server (including all server-side code): 20 points
    a. The GitHub for the server is https://github.com/zskemp/SafeNights_Web.
    b. The live URL is https://gentle-badlands-54918.herokuapp.com/
    c. This is a website in Django/Python that was written by us to handle adding/receiving data for our users, along with send emails to emergency contacts in the application.
    d. The SafeNightsAPIClient and SafeNightsAPIInterface classes are the two classes we made to help push/pull data, and in each of the classes for our main features we utilize the API to push/pull data.
2. GPS/Location Awareness: 15 points
    a. Our app uses GPS by storing user-locations every 10 minutes when they state that they are going out for a night of fun. This runs as a service in the background for as long as the say they are out on their night (see TrackingActivity.java). Location is primarily used to test whether a user has reached the location they

expected to reach at the end of their night, using Geocoding and GPS coordinates stored.

b. The stored GPS locations are also used in our Last Night feature, where users can see a plotted set of points on a map of all of their GPS locations from their last night out.
c. In terms of testing, this service runs its check every 10 minutes after “Start Night” occurs, so to test you may either move from location to location in 10 minute intervals, or simply change the Timer in TrackingActivity.java to run much more frequently, perhaps every 5-10 seconds. The email feature will be discussed below, but the user’s last known location is also sent in the email messages sent as well. Simply destroy the app after hitting “Start Night”, and you should get an email with information on the user’s last known location. (*NOTE: if the timer is changed to run too quickly, the logic for ending a night in the correct place will not work correctly*)

3. Open Shared Activity (email): 10 points
    a. After a user indicates they have started their night, an email can be sent in 3 possible ways. If a user’s app is running low on battery (<10%), an email will be sent to their emergency contact’s email stating their last known location and where they were intending to go. An email will also be sent if a user’s app happens to crash or destroy, without them previously hitting “Stop Night” manually. This email will also send their contact their last known location, and where they intended to go.
    b. The last case an email will be sent is a bit more complicated. This email triggers if a user has not moved physically (Shake detection, explained below), and has remained in the same GPS location for the last 40 minutes. This action only triggers between 2am and 6am, as we are assuming users could potentially be stationary voluntarily before 2am. If all of these checks occur, then an email is sent to the emergency contact with their last known location and intended destination.
    c. To test this, you may want to remove the time constraint (2am-6am) in TrackingActivity, although the other two email condition checks are fairly easy to simulate. (*See NOTE above about time though. It will still work, you just might have to wait a bit longer than expected*)
4. Device Shake: 10 points
    a. Device shake came into play with our main condition for sending a notification to the user’s emergency contact. We assumed that it would be possible for users to remain at a certain destination past 2am, so we needed a way to check if they were moving about, or dangerously passed out.
    b. To fix this, we used a device shake in our condition loop. The TrackingActivity service uses the Accelerometer to check if a user has physically moved around recently, and if they have not, this information is included in our email check. However, if it is past 2am and they stay at the same location, but are still moving

around, we are making the assumption that they are alright, and intended to be there.
    c. The accelerometer/shake logic here is also included in the TrackingActivity.java file.
5. Data Storage using Shared Preferences: 10 points
    a. This was utilized for user accounts for logging in and logging out. When a user logs in, we store a unique ID for them along with their password so that the user does not have to keep logging in.
    b. Additionally, this was used for certain key variables throughout the application, such as a unique identifier number for a user's night out, and their saved locations. This code can be found in SignIn/SignUp

Total Potential Optional Points: 65

Testing Methodologies

Testing was completed in various ways. For GPS testing, we utilized a few different methods of testing, including logging data for our current and intended user locations. We lowered and desensitized our service requirements so that it would run every 5 seconds, so we essentially kept a list of GPS coordinates and locations for various locations nearby our house and UVA, and walked around logging and checking to make sure that the data being transmitted was valid and correct. GPS sensitivity was more or less checked through trial and error to help gauge how accurate/sensitive we needed to make our GPS readings.

Email testing was completed using a dummy email account and done through Django, so we were able to sandbox that feature and conduct testing until we confirmed that feature was working properly.

For the rest of the features, we utilized incremental programming and modular programming, slowly testing each feature we developed and confirming it worked. We wrote down some use cases and expected results, and followed through with those in our application to test whether we got our expected results.

In terms of UI, we conducted extensive testing on our application simply by using it a lot and getting our fellow students to use the application as well to find bugs.

Usage

Add Drinks and History can easily be tested aside from the rest of the features, as all that requires is manually entering in data for various dates, and seeing those results visualized on a graph.

Get Started along with the TrackingActivity service is the core of our application, but we kept in the realistic boundaries for submission. This means location is pulled every 10 minutes, and messages are only sent past 2am and before 6am. However, this code is commented and

can easily be changed if you don't wish to walk around in 10 minute increments, and then not move for 40 minutes. However, the other two message triggers (battery and app-crash) can be mimicked fairly easily.

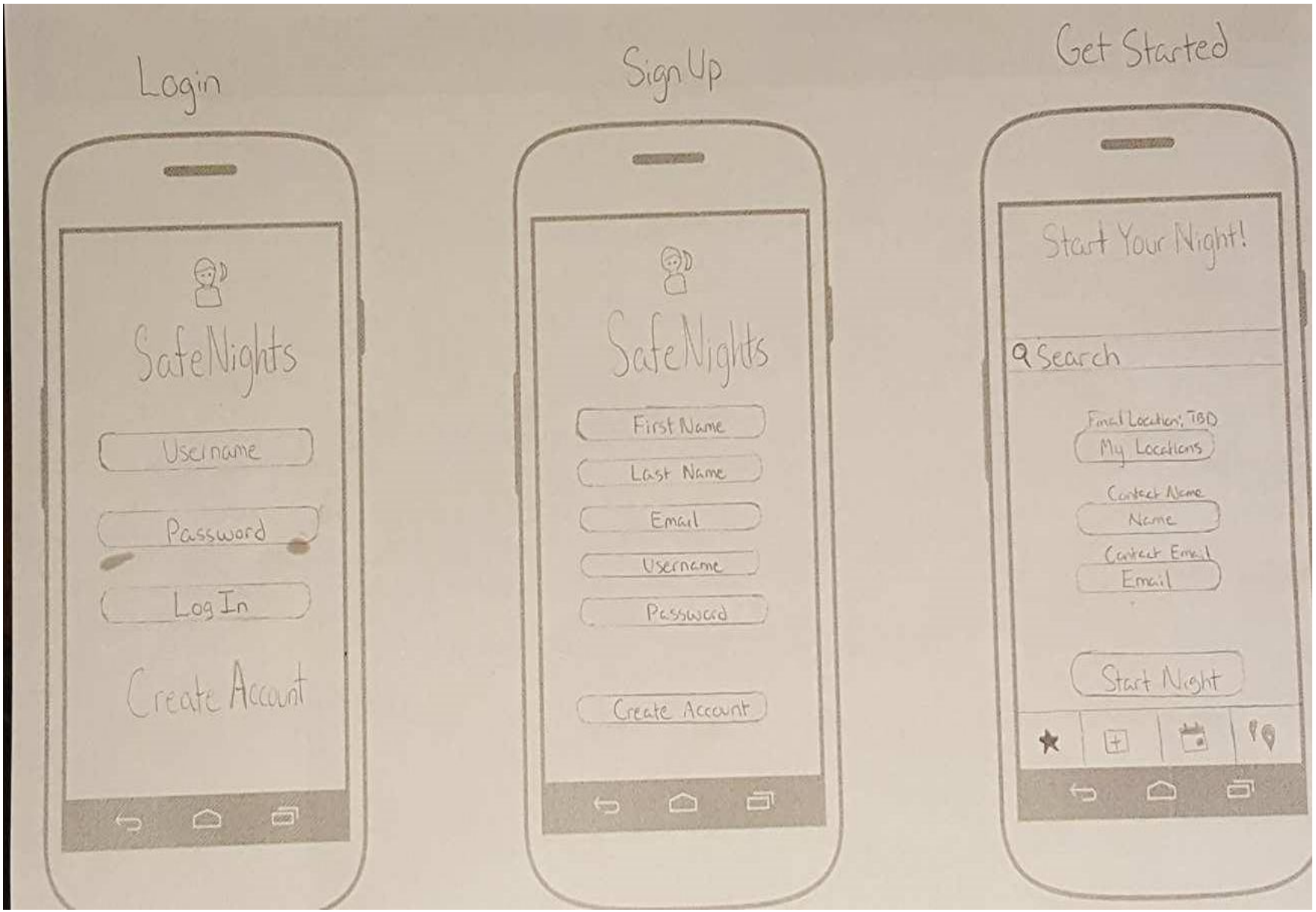
For accounts, you can create a new account if you wish, or use a stored account we have with username: "zrs" and password: **"**1234".

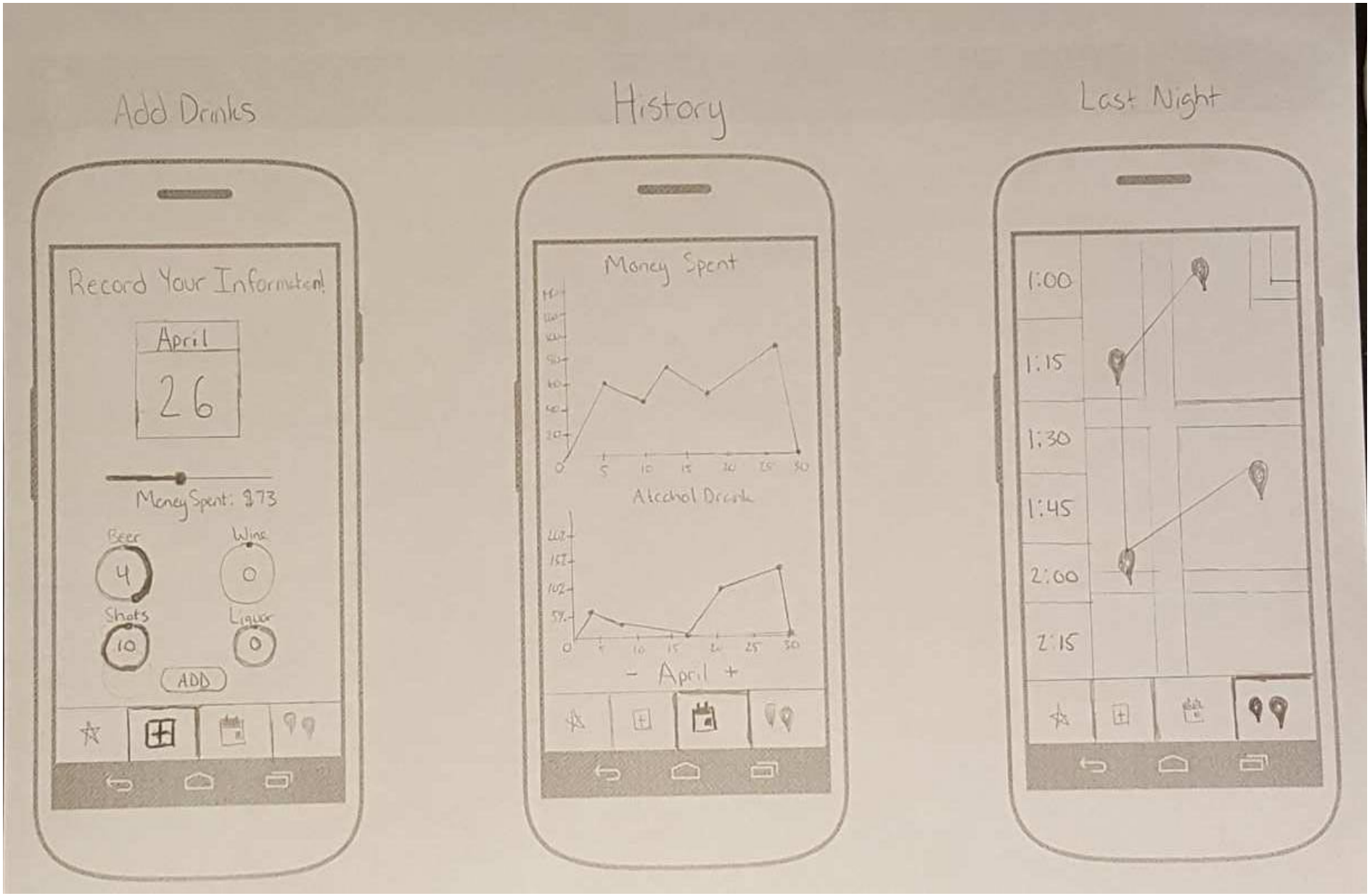
Lessons Learned

This was a very thorough project, and we learned a lot. One of the biggest things we learned is that in terms of UI, it is very important to design an application before starting, and refine that throughout the course of development. Even if someone has an application that contains a good idea, if it has poor design or UI, no one will use the application.

In terms of coding practices, we did most of the backend first, and then worked to fix the frontend and UI. Looking back on it, this was helpful because we did more of the hard work (arguably) first, but it was hard to match certain methods and logic to UI when revamping the UI at the end of the project. For example, we first made four activities for our four features. We then decided a couple days before submission that tabbed layout like most popular apps we looked at (Spotify, Facebook, Instagram, etc.) would be much better for our app: it would enable us to take users right into functionality instead of giving a useless main menu, decrease "click-depth" and simply looked nicer. To implement this correctly we then had to change all our activities into fragments. This sounded like a fairly simple task, but proved to be a rigorous and frustrating endeavor. For this reason, we may look into some research or practices that condone doing the front-end interface first, and then coding the backend afterwards.

From a real software job perspective, working in a team to produce a well-functioning application takes time, patience, and more patience. None of the code included was particularly algorithmic or novel, but mobile programming takes quite a bit of perseverance to be successful. Handling working on different features at different times takes good communication and teamwork, which is essential unless you want to resolve 500 line merge conflicts.


Login
SafeNights
Username
Password
Log In
Create Account
Sign Up
SafeNights
First Name
Last Name
Email
Username
Password
Create Account
Get Started
Start Your Night!
Search
Final Location: TBD
My Locations
Contact Name
Name
Contact Email
Email
Start Night


Add Drinks
Record Your Information!
April
26
Money Spent: $73
Beer
4
Wine
0
Shots
10
Liquor
0
ADD
History
Money Spent
Alcohol Drank
- April +
Last Night
1:00
1:15
1:30
1:45
2:00
2:15