Wifu

Design Document

Team 6:

Michael Clayton

Billy King

Bakytzhan Kudebayev

Brandan Miller

Table of Contents

Purpose:

This document provides the main design aspects of the Wifil application. The document comprises important details of the application such as general priorities, a design outline, design issues and design details.

Wifil is an application for Android phones that allows for the location of Wi-Fi hotspots and crowd-sourced collection of public hotspot information.

General Priorities:

1) Battery Life

One of the top priorities of this application will be to maintain minimal battery consumption for the platform that will run this device. This priority has risen because if there is an over consumption of energy, then end users will abandon the Wifil application due to over taxation of the users device.

2) Performance

The application's performance will be a factor in whether or not the user will continue to use the application. If the application runs quickly and smoothly, then the end user will receive information quickly and thus giving the user a better experience with our application.

3) Usability

The key to getting our application used will mainly be driven by usability. Usability will have to be intuitive in order to keep the user patient with the application. The Usability will also have to be as descriptive as possible in order to allow the user to understand how to obtain information from the application.

4) Security

Security issues will be foreseen as minimal compared to the other general priorities, but if security fails it will lead to some major issues with the application. On the server side if the application is used in some manner to send or recieve requests of the server, then malicious activity can be used to render the application useless purdue wide. If the applications security fails to protect on the users device, then nonpublic information of the user may be stolen.

5) Accessibility / Portability

The application is design to be usable whenever Wi-Fi network information is desired. Therefore, it is important that the application be usable on while the user either portable, without wi-fi access, on the campus of purdue, or just simply lost.

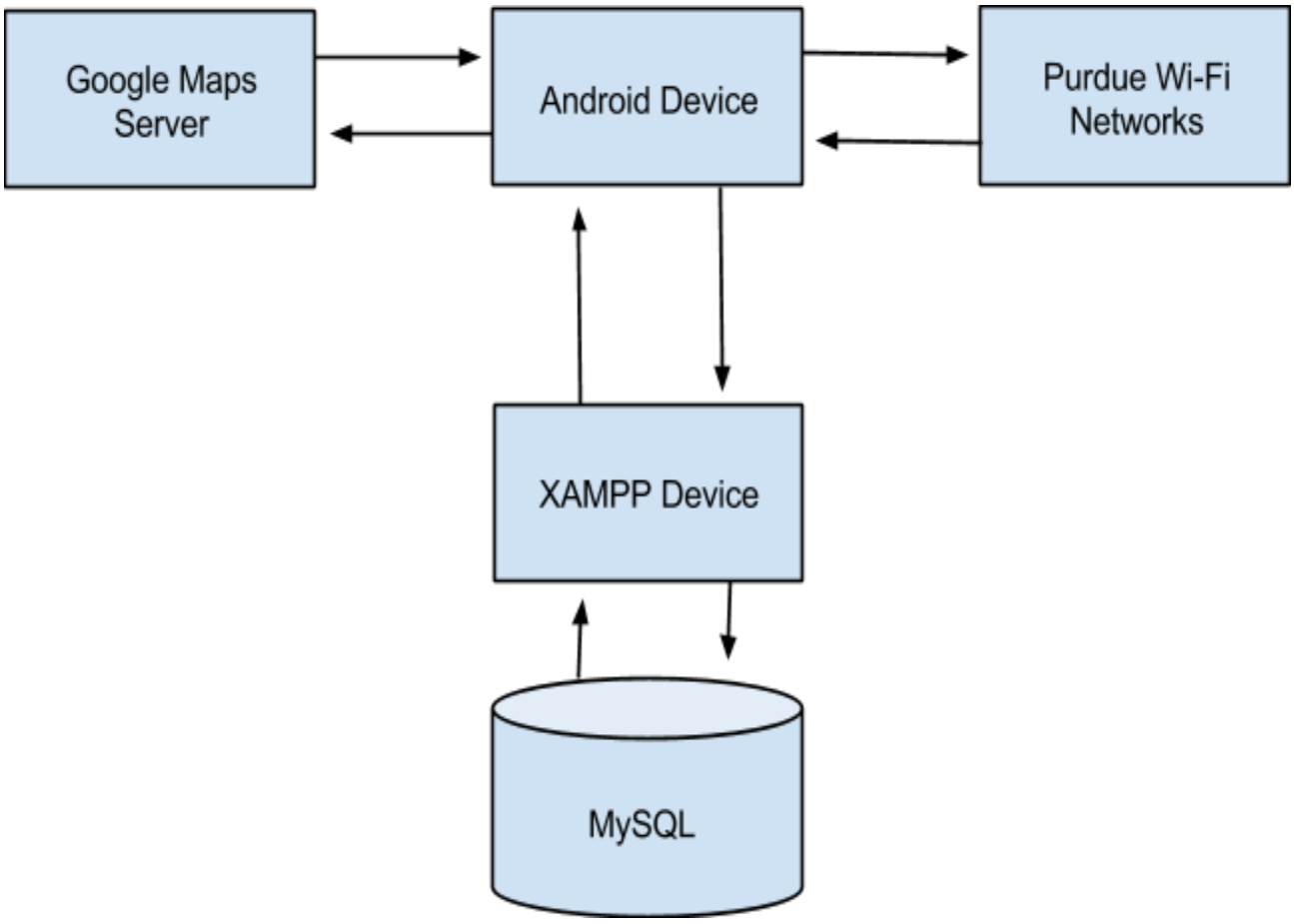
Design Outline:

Our application is composed of three interaction points where the android device running the application is the center start and end point.

If requested, the android device will locate local Purdue Wi-Fi Networks. If found and access is granted, the Purdue Wi-Fi Networks will return network information like location, model, and other miscellaneous information.

When a user interacts with the application in order to store or retrieve relevant information about Purdue Wi-Fi Networks, the android device will send a request to the XAMPP Device. The XAMPP Device will the process the request and will store or send relevant data from or to the device. The data of which will mainly consist of user or network information.

The Android device will interact with the Google Maps Server by sending requests for current position information along with Wi-Fi locations. The Google Maps Server will then return a map based on the user or wi-fi network.



Design Issues:

1. **Architecture Used:**
   Option 1: Client-Server
   Option 2: Service
   Result: A client-server architecture was selected for ease of implementation. Our system lends itself nicely to this architecture because we will have many Android phones (clients) which will need to be requesting data from our backend server.
2. **Client Design:**
   Option 1: Thin Client
   Option 2: Fat Client
   Result: A Thin Client architecture was selected because intense computations will be performed on the more-robust backend. Fat client is infeasible because the Android phone clients would not be able to perform the intense computations without suffering severe speed and power loss.
3. **Platform Used:**
   Option 1: Android and Web Service
   Option 2: iOS and Web Service
   Result: Android was selected for ease of development. There is no cost to develop for Android and development can be completed using Java. Google's libraries are nicer and our entire team won't have to learn Objective-C if we choose to develop for Android phones.
4. **Programming Languages:**
   Option 1: Java, PHP, NoSQL, JavaScript, HTML and CSS
   Option 2: Java, PHP, MySQL, JavaScript, HTML, and CSS
   Result: Java, PHP, MySQL, JavaScript, HTML, and CSS. MySQL was chosen over NoSQL because our design lends itself nicely to the relational database paradigm.
5. **User Registration:**
   Option 1: Custom Registration
   Option 2: Google Authentication
   Result: Google Authentication was selected in order to eliminate the need for the user to register for a separate account. If we would have chosen a custom registration system, then there would be complication such as the greater need to maintain the registration system and increase in the design time of our overall project.
6. **Cache Updating System:**
   Option 1: Based on time
   Option 2: Based on logging into the application.
   Option 3: Manual update
   Option 4: All
   Result: All options will be implemented in tandem. Option 1 and Option 2 will be used in order to keep an up to date cache for when the user ends up offline. Also, manual updates will be used in order allow low the user to update the whenever they might see the current cache as out of date.
7. **Offline Mode:**
   Option 1: Caching
   Option 2: No offline mode
   Result: A cache-based offline mode was selected in order to improve the practicality of the application. If the application did not work off-line, then the user could not find a Wi-Fi network in order to get back online. Excluding the offline mode would practically mean

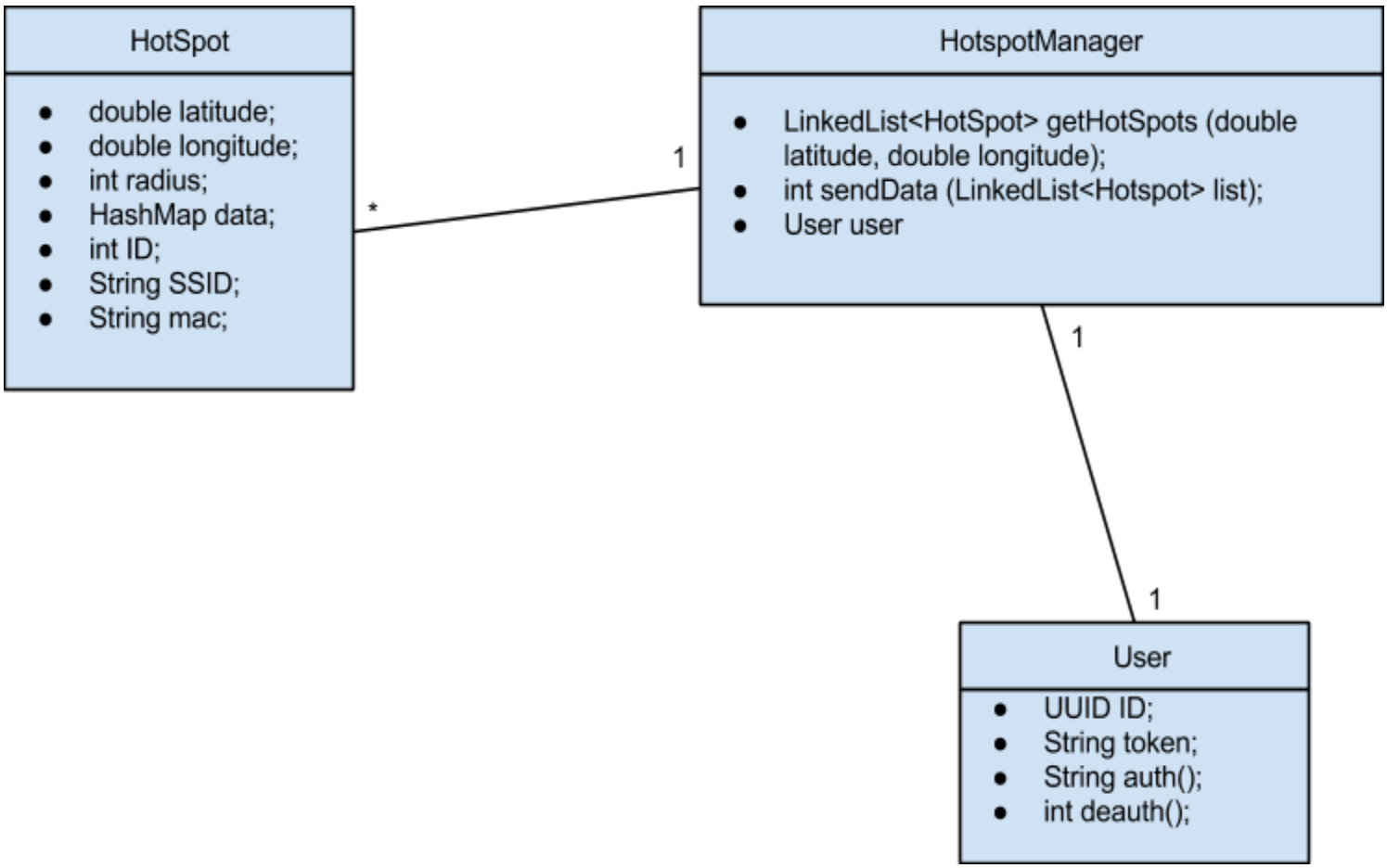
death for our application, due to lack of use.

8. **Caching System:**
   Option 1: Cache based on locality
   Option 2: Cache based on most frequently used Wi-Fi networks
   Result: Caching based on locality was chosen as it is likely to provide more information relevant to the user.

Design Details:

Class Diagram:

Class diagram below represents the important classes for the application. It also show how each class is associated with other classes.



Class description:

HotSpot:

This class includes the details of the Wi-Fi routers that our application will require. A double Longitude and a double Latitude to be used to request the location with google maps which both need to be doubles for location accuracy. An Integer Radius to build a “Wi-Fi” strength radius, this number does not need to be as precise as the Coordinates so we will only use an integer. We will use a hashmap to store data about the Wi-Fi router that will not be needed to be sent anywhere and will only be used locally on the users phone (such as the name of the wifi spot given by the user, the quality of the signal, ect. ect). We will give the Wi-Fi spot an ID number to better parse for our data base to have better maintainability with in said database. Finally, we will include two types of strings for mac addresses and SSIDs so that we can tell determine the type of Wi-Fi router we are dealing with.

HotSpotManager:

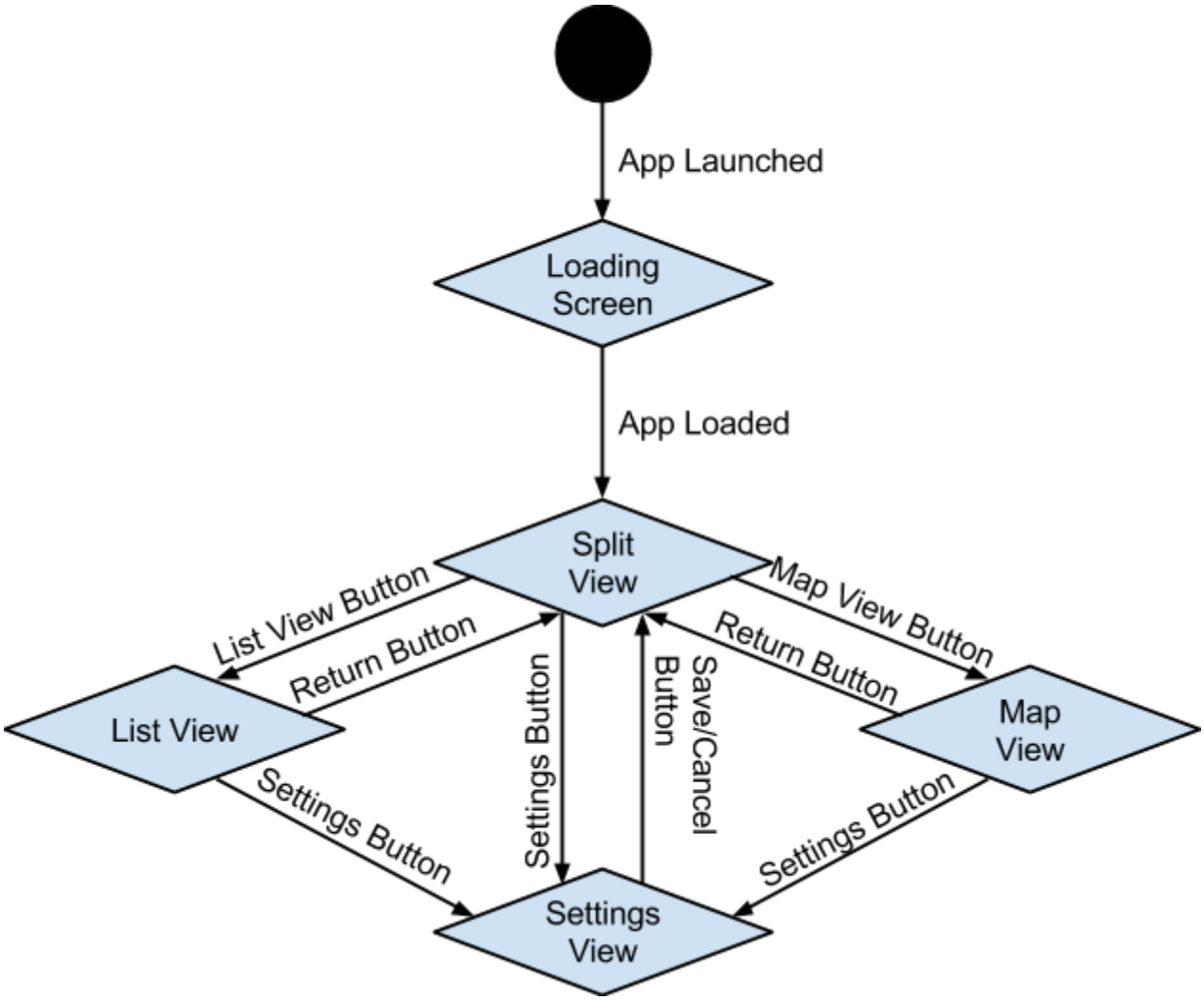
This class includes the functionality to place onto the map the locations of all the hotspots including their details. We will be using a linked list of the hotspot class variables with mainly the hot spots coordinates in order to keep a list of all possible hot spots. we will have a send data function which will then send the linked list of hotspots to send information to google maps in order to build a map containing all the hotspot locations. Finally, we will have a user that will exist to have a tailored list of information about wifi spots

User:

The user class contains information of the current user of the application on a specific device. We will maintain whom the user is within our databases with the users ID that we will maintain with a UUID ID. We will use an the int auth function to obtain the authentication of a user based on their ID. We will then hold whether or not the user has authentication for our application within a String token which we will use to allow/deny access to certain aspects of the application. The User class will have two functions. First is User auth which will return a string as to whether or not the user has access to the application based on their ID. Second, int deauth will de-authorize a user's ID number.

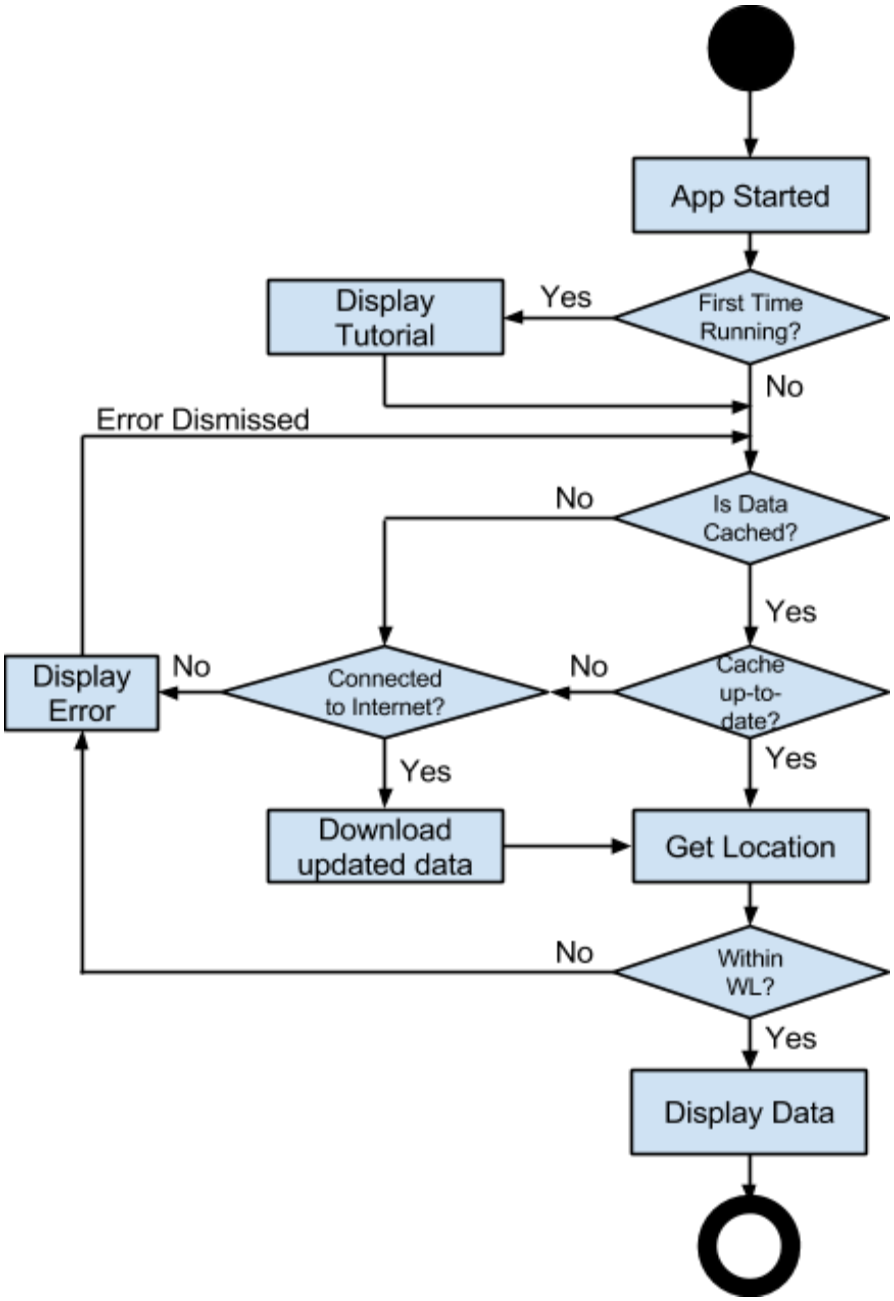
Activity Diagrams:

State Diagrams:



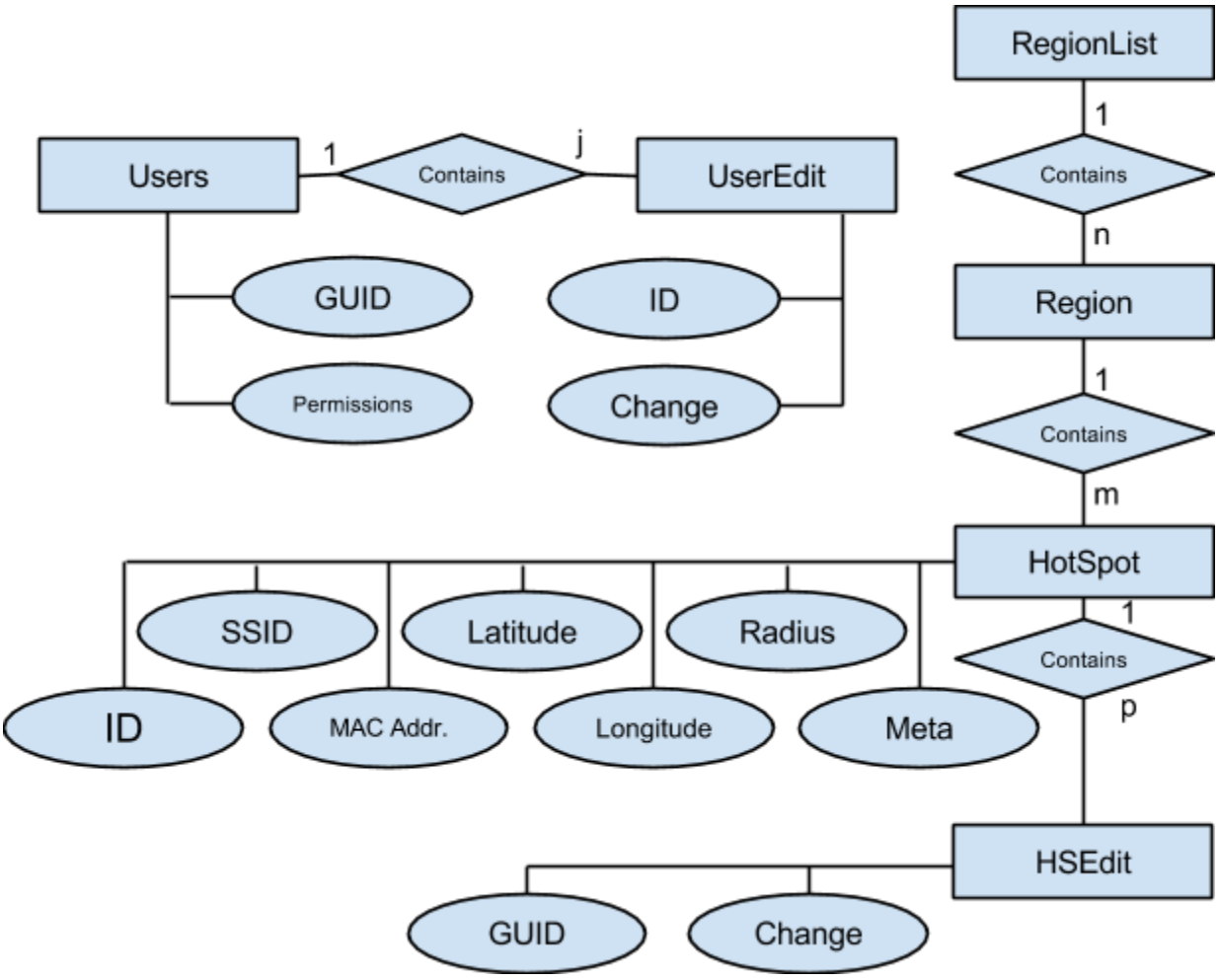
View Navigation

Sequence Diagram:



App Startup Sequence

Entity Relationship Diagram:



Database Structure

UI Mockups:

Android Client UI

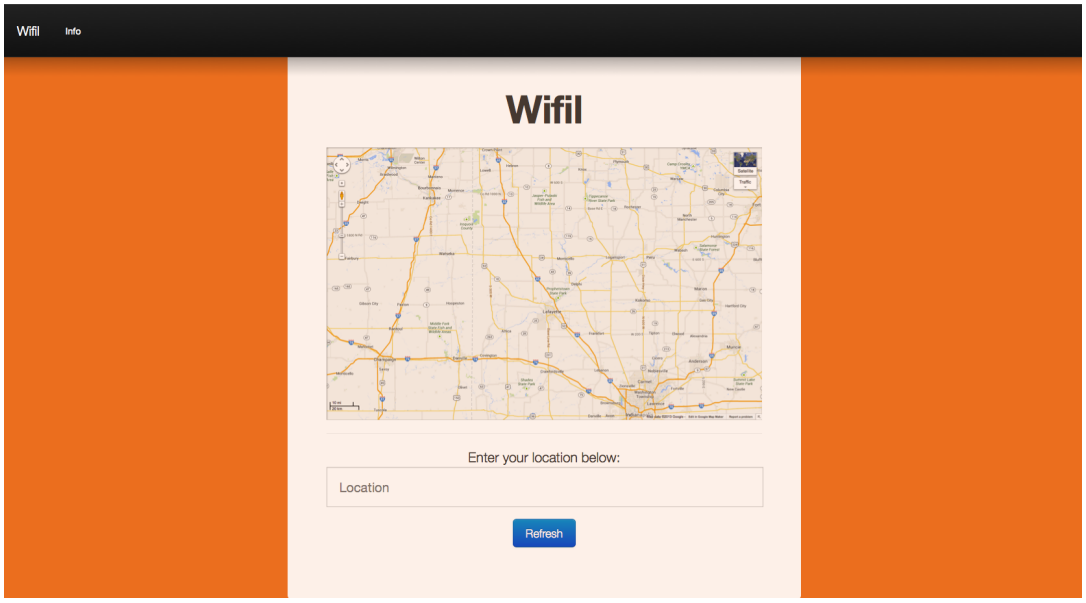
1. The **splash screen** is simply an image of Wifil's logo being displayed while the application loads upon each startup. Initial Wi-Fi and location data will be loaded here before the application fully starts.

2. The **first time login** page, intuitively, will be shown if it is the first time the user has logged into the application. This page will display a dialogue which will guide the user through the features of the application.

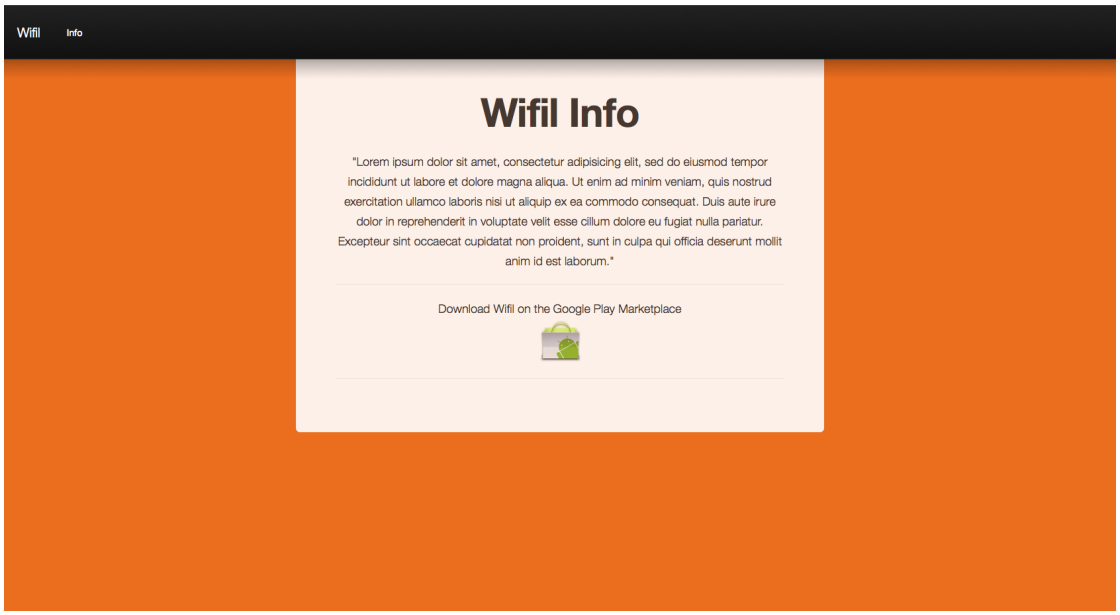
3. The **main page** will contain a Google maps view where the user may view, add, or edit Wi-Fi location pins. It will include a List View where users can view more in depth information about the Wi-Fi locations nearby. Selecting a Listview item will provide the user with more data and choices about each Wi-Fi location.

4. The **settings page** is, intuitively, where all of the applications settings will be managed. These settings will be persisted and will not need to be set each time the application is run.

Web Portal UI



1. The **Home Page** will be a viewer, implementing Google Maps, to show local Wi-Fi hotspots. There will be an input box for getting the location desired. The refresh button will update the Google Maps view accordingly based on the input location to show any new or updated Wi-Fi locations.



2. The **Info page** will simply be a page full of text describing the basic functionality of the application and how to use it. It will also contain a link to download the Android client application.