

Alexander Ford 18361148
Andre Tertzakian 17682148
Neema Boutorabi 18694142
Omar Dzick 36579143

Emre Pekel 48488134
Konrad Izykowski 26506148
Luigi Sacco 21139143

Beacon - Design Document

INTRODUCTION

We are planning to build Beacon as an Android app that allows users to find events near their location in real-time, share events they are hosting with other Beacon users, comment and post photos on the page of an event they went to/are currently attending, and broadcast their events to a wider radius if they are a business account. This document will cover the high-level architecture and design of our system.

SYSTEM ARCHITECTURE AND RATIONALE

Beacon consists of the following major components:

1. Android application
2. Google Maps API
3. Firebase server

Android Studio

We are going to be working with Android Studio for our project as it is the most common IDE for android development, offering lots of built in functionality including device emulators and UI prototyping tools. Additionally, android studio can be used in tandem with gradle, a general purpose build tool that allows for easy integration of third party libraries. Android Studio also offers a tool for code refactoring, a standard code formatter and various testing frameworks such as JUnit4 and Espresso. Most of our team has experience using Android Studio, so we can begin our implementation faster.

Google Firebase

We are employing the use of Google's Firebase platform in order to handle multiple aspects and functionalities of our application. These include:

- User Authentication
 - Google
- Remote, Real-Time Database, To Store,
 - User Information
 - Event Information
- Storage of Media
 - Photos related to an Event
- Targeted Advertisements
 - Provide users with ads which are relevant to their search history

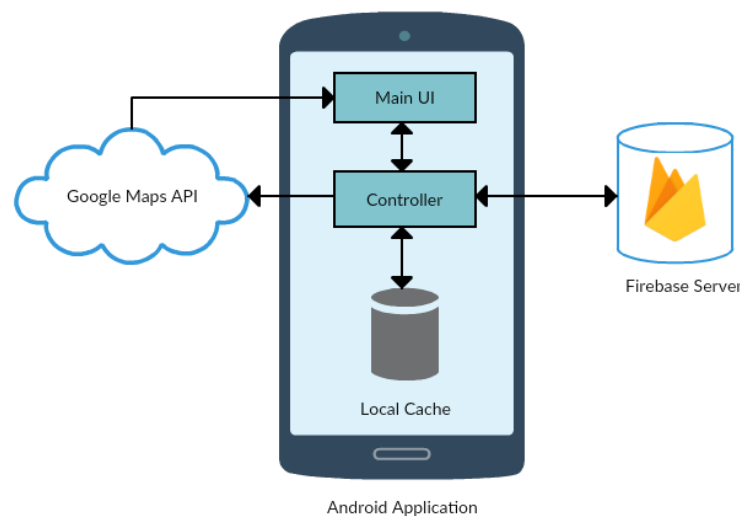
We chose to use Firebase instead of available alternatives because of the platform's simplicity. All of the features listed above are, for the most part, handled by Firebase. This allows us to focus more closely on the user's experience and making the app more usable, as opposed to spending time on the technical server-side aspects of the project.

Furthermore, Firebase offers offline services. This means that if a user is using Beacon while they have no internet connection, the functionality of the app will not be compromised. Any changes made by the user during their offline usage can be seamlessly merged with the existing, online data. This capability will increase the robustness of Beacon and improve the overall user experience.

Google Maps

We will be using the Google Maps API for the app's location-based features. Google Maps is by far the most used and robust map API. Google Maps will allow users to see locations that have been pinned by people hosting events in an intuitive UI. Since many users have already used Google Maps, it will make our application very intuitive for the average user. It will also provide directions to these locations for users who have Google Maps installed on their device.

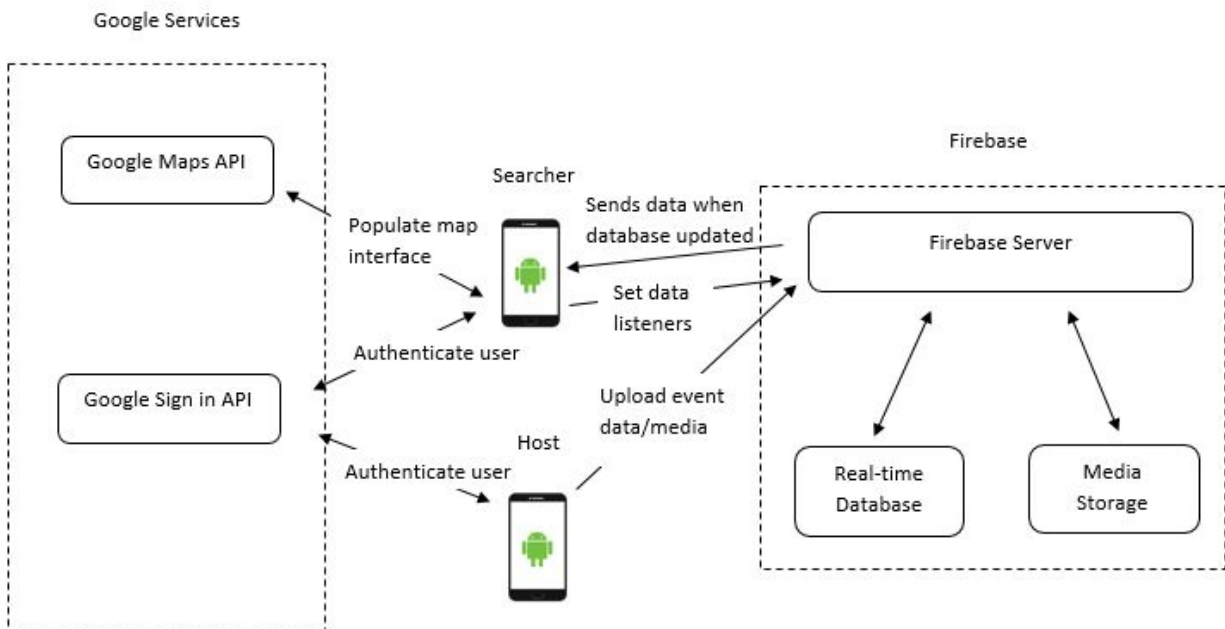
Model-View-Controller Architecture of the App



- The Main UI is the different pages/ views that the user can see. It sends information to the Controller telling it which button has been pressed and the Controller sends information back telling the UI which page to display.
- The Google Maps API interacts with both the Main UI and the Controller. The Controller sends locations for Beacons to be placed on the map. The map tells the Main UI to display the map with the updated pins.

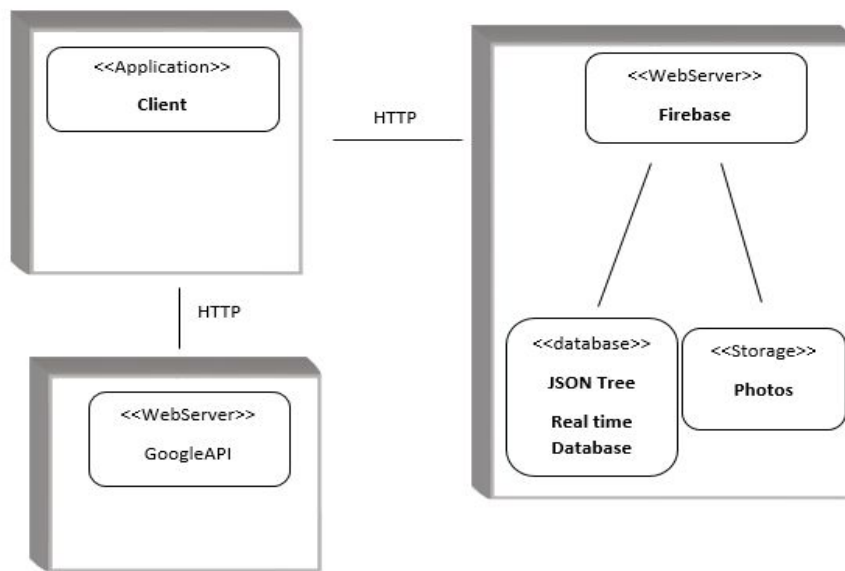
- The Local Cache stores user information, favourites, and a select number of closest nearby events. The Controller will access this data if there is no stable internet connection and new data can't be pulled from the server.
- The Firebase server will contain all of the event and user information in its database. That information within the Firebase database will be called upon and written to by the controller. Furthermore, the controller will contact the Firebase server in order to

Dynamic View And Description



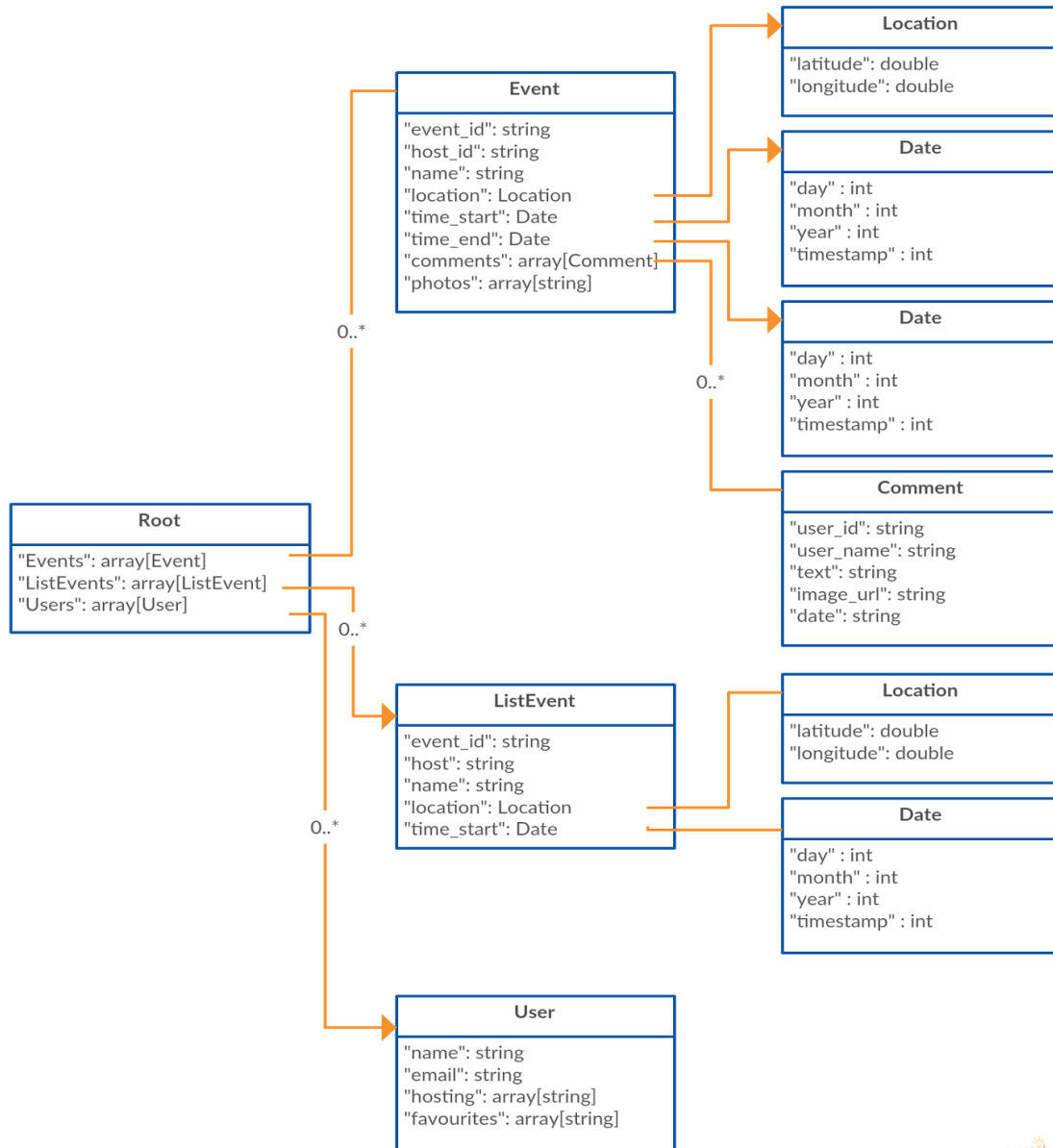
The above image highlights the key interactions between the various components and services that make up our application. In particular, the image identifies two android phones, one acting as a searcher and one acting as an event host. Upon launching the application, both the searcher and host initially sign in using the “Google Sign in API”. The host can then create a new event at a nearby location, this data will be pushed to the firebase server and stored in the Real-time Database and/or Media Storage. The searcher will set listeners for database change events and will pull event data based on his/her search range from the firebase server. The searcher also utilizes the “Google Maps API” to populate a google maps object with markers corresponding to nearby events. Note: Every user can assume the role of a searcher or a host.

Deployment Diagram - Static View



Data

Our application's data model is as follows:



Event info is stored in two places in the database. A **ListEvent** object holds all the data necessary for determining an event's proximity and displaying it on the map or in list view. An **Event** object holds full event information, including comments, photos, and end time. This allows minimal data to be downloaded when looking for nearby events. Flattening out data in this fashion causes the Beacon to transition from screen to screen faster and lowers overall data usage.

DETAILED DESIGN

UML

A detailed UML diagram of Beacon software system can be found on Beacon's repository at:

<https://github.com/lsaccoz/Beacon/blob/master/docs/Beacon%20-%20UML.jpeg>

Below is short legend describing the UML:

- Colours
 - Light Blue - Activity Classes
 - Light Green - Google API classes
 - Neon Green - Data Container classes
 - Yellow - Base Activity classes
 - Purple - Utility classes
 - Orange - Fragment classes
 - Dark Blue - All other classes
- Lines
 - Solid with Arrow - Extends
 - Dotted with Arrow - Implements
 - Solid with Diamond - Associated (i.e. the class where the terminates has an instance of the originating class as a class variable)

There are a few implements design patterns which are clearly visible from the diagram. For example, the *DatabaseUtil* class has a reference to itself, as well as the activity classes which have connections to the Firebase Database. This is because the *DatabaseUtil* employs the Singleton design pattern. We decided to use this pattern for the connections to the database as we believed that there should only be a singular point of connection to the database. In this way, we avoided having a complex web of database connections and references scattered throughout the project. The singleton pattern we used was implemented with lazy initialization, so that the database is not called upon until required.

Another design pattern we used was Private Class Data pattern. In all of the data container classes, such as the *User* and *Event* classes, we implemented this design pattern in order to reduce the exposure of their held attributes by limited their visibility.

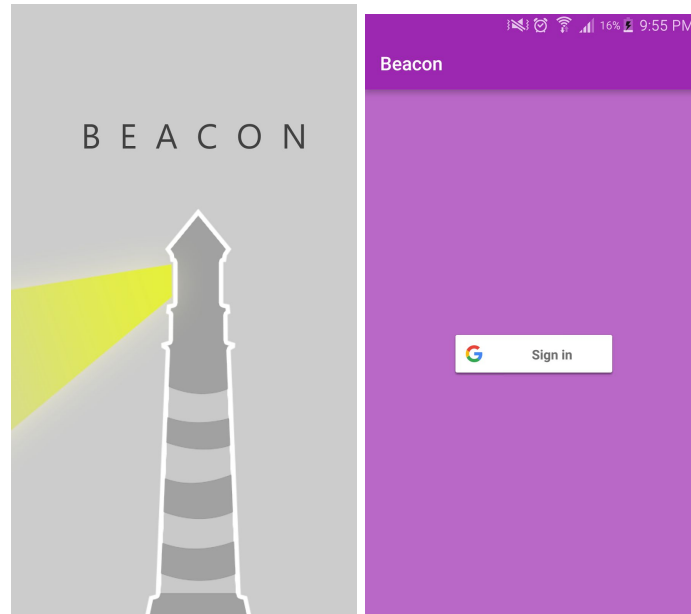
Furthermore, we implemented the Adapter design pattern for all of the activities and/or fragments within the project. For example, the *EventListAdapter* class helps to map the data contained within events to the listview of either the Favourites or the List View fragments.

GUI

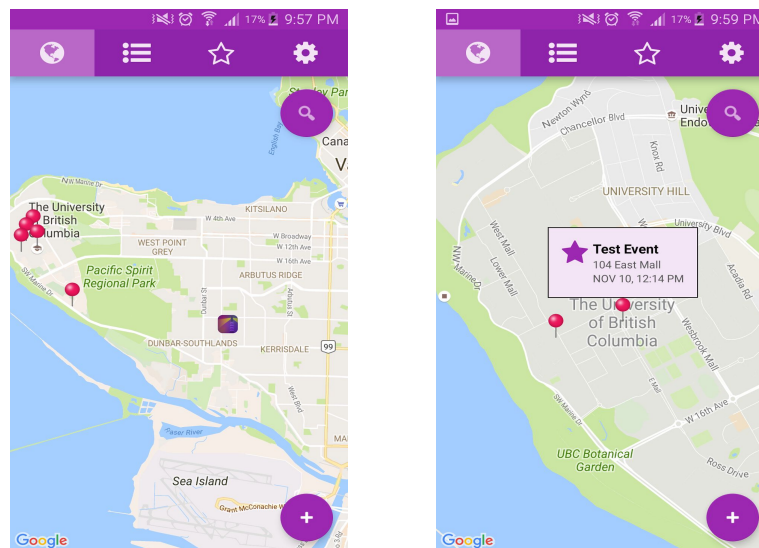
The UI is based on Google's material design. It will consist of an interactive map, and a list view where nearby Beacons will be displayed. Every Beacon will be a link to an event page, where information, comments, and pictures of the events can be viewed.

Login View

After clicking the Beacon icon, a splash-screen will display while the app is loading. Upon opening the application, the users will then be presented with a Google sign in page



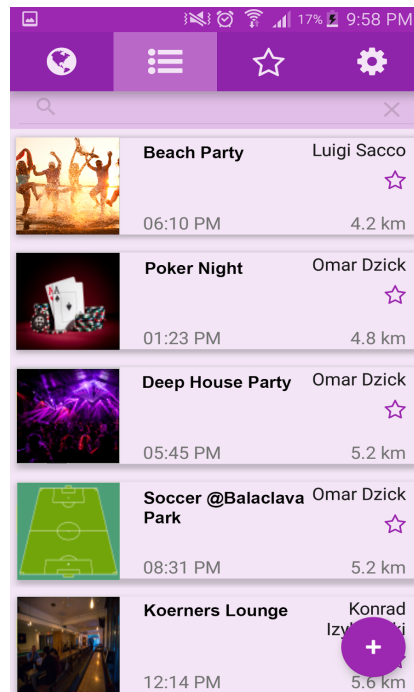
Map View



After users go through the login view, the main page is displayed. This page contains the map view - an interactive map that shows the Beacon pins that are nearby the user's location.

List View

Users can also choose to see Beacon pins in the list view, which sorts events by name, location (i.e. distance), or time; and shows said basic information.

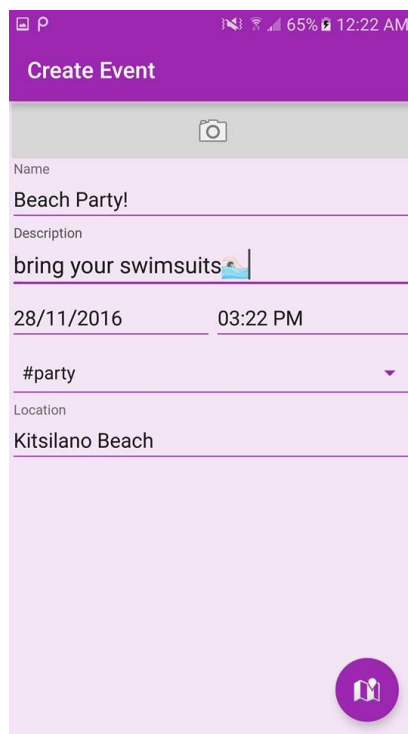


Event Page

The event page will display all of the information that relates to an event: location, title, short description, comments feed (includes text and photos)

Create Event Page

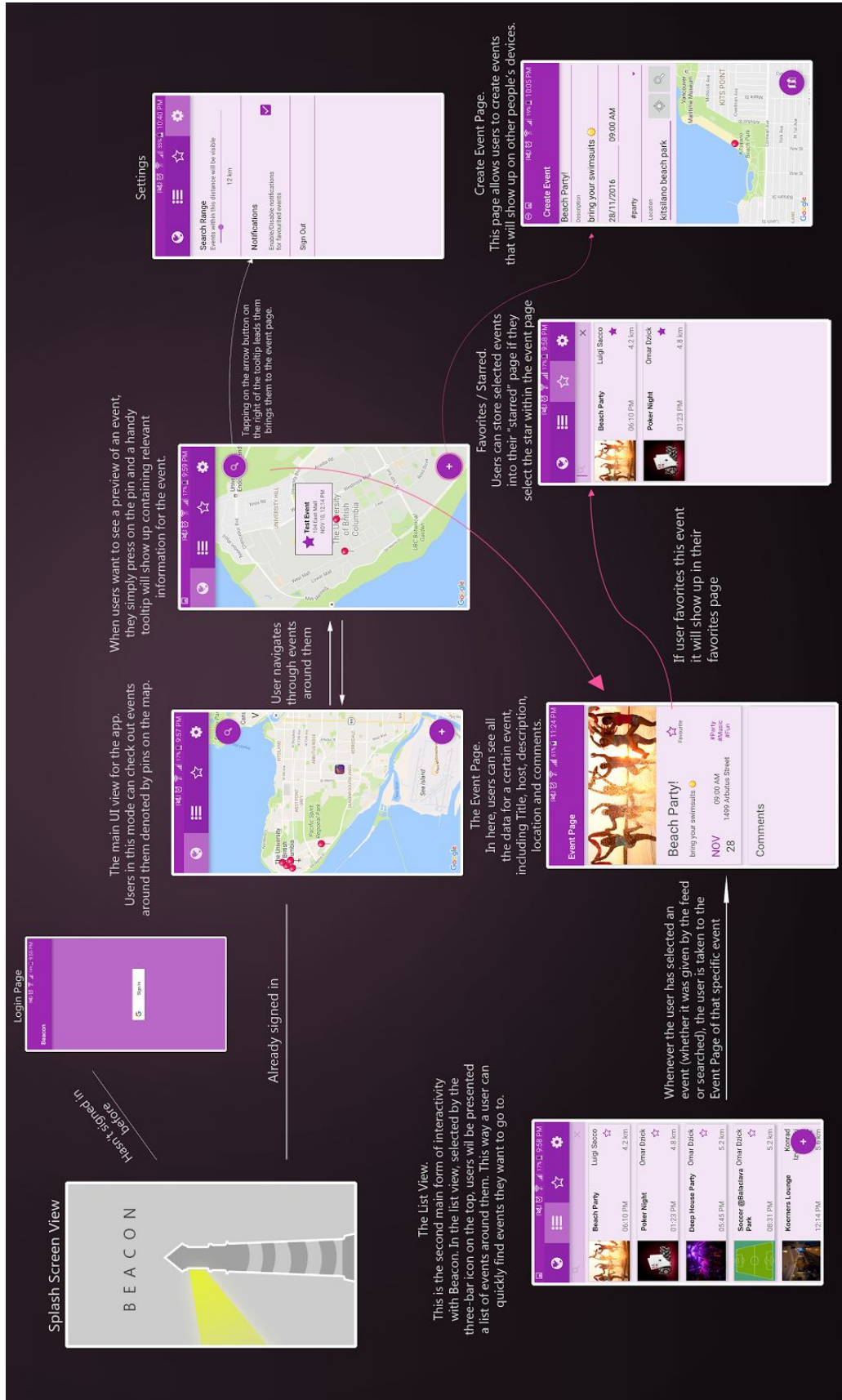
The create event page is where hosts can set their event information, location, and upload their event photo. There are name and description fields, that take in standard text inputs. There is a date and time field that uses google date and time pickers to populate the field. There is an location field, that opens up an interactive map where a user can select their event location by searching, dragging a pin, or setting it to their current location.



The screenshot shows the 'Create Event' form on a mobile app. The form has a purple header with the title 'Create Event' and a camera icon. Below the header, there are several input fields: 'Name' with the text 'Beach Party!', 'Description' with the text 'bring your swimsuits' and a beach icon, 'Date' with the text '28/11/2016', 'Time' with the text '03:22 PM', 'Hashtag' with the text '#party' and a dropdown arrow, and 'Location' with the text 'Kitsilano Beach'. At the bottom right, there is a purple circular button with a white icon of two people.



DESIGN FLOWCHART



VALIDATION

We are continuously seeking opinions from potential users as a way of validation. Below, are assessments on how well our design is meeting potential user's demands. Additionally, there are some examples of changes to Beacon that were driven by our design reviews:

Search For an Event

Searching for events in Beacon must be fast and intuitive. Our design offers several avenues by which users can find an event they are interested in, namely, a map view, list view, and searching. Below are descriptions of our design choices and how they meet our user's demands:

- The design of the map view allows for an easy and fun way for users to search for events around them. The map view is one of the more unique design elements of Beacon. Google maps is by far the most widely use map API, so the average user will be quite comfortable using it within Beacon. Tapping on event pins opens a small info window that gives key information about the event before viewing the entire page.
- Some users found the map interface too cumbersome or cluttered for finding an event (especially if they already know exactly the event page they want to find). In the list view, events are sorted by distance from the user (closest events nearby). There is also a search bar, that will show events that match a user query. These addition mechanisms for finding an event, make the process of finding events much more robust, and ultimately make the app much more usable.
- We found that once a user has found an event they want to attend, the final step in the process was often unclear. Most events that are found will take place at some time in the future, and so it's likely that user would forget about the event entirely. Favoriting an event allows users to easily separate events that they are interested in from all other events. Notifications reminders are also sent to users before an event starts. This design for favoriting gives users an easy built in way to create reminders and sort the events they are interested in.

Event Pages

Event pages are what users will view to entice them into going to events. Therefore, event information must be displayed concisely and in an appealing format.

- Photo uploading was added so that event pages are both more visually appealing, but also have more customizability to help host's events stand out.

- A common feature potential users demanded was an easy way by which to interact with the event host and other attendees within the app. The solution was a comments section for each page, that any nearby user could post to. Some alterations made to comments are:
 - Users preferred that their pictures be displayed large and next to their username and the first line of the comment, instead of them being displayed small and next to just the username.
 - Users deemed a character limit between 2 and 140 appropriate for the purposes of this app. Moreover, one client suggested that users should not be allowed to send a comment consisting of whitespace only.

Create an Event

Since all events in Beacon are created by users, the create event page must be fast, intuitive, and robust enough to make the process of adding an event extremely simple. If users find the process of creating an event frustrating or confusing our application will not be widely adopted. Some design choices and rationales that considered these user demands are as follows:

- Users often want to create events at (or very near) their current location. Therefore the default location for all events was set to a user's current location.
- Interactive time and date pickers were added to make entering fields quick and simple. The time pickers are set so that it is impossible for a user to enter in an invalid time or date (ie a time in the past, or a start time after an end time).
- Requiring hosts to enter in a full address to set their event location was found to be too limiting, as many users want to create events at small businesses or landmarks whose addresses were not well known. As a solution, a separate page for selecting the location of a created event was implemented. This page contains an interactive map and a Google autocomplete search bar, which allows hosts to enter in any location indexed by Google maps as their event's location.
- The location selection page also allows you to drag and drop your event's pin on a google map. This was found to be a particularly useful design choice for users who wanted to set their event's location somewhere near their current location. Instead of having to enter in an address, they could simply move a pin from their current location on the map.

