Beacon

A Personal Location-Broadcasting Application

Development Team:

Julian Clayton 100892373	Jason Bromfield 100934833
Nolan Hodge 100925862	Cameron McQuarrie 100770066

DELIVERABLE #3: BEACON ARCHITECTURE

BEACON ARCHITECTURE

PART ONE: COMMUNICATION BETWEEN CLIENTS

The architecture of Beacon can be represented by three main components:

- A Beacon Client
- A Google Firebase Server/Google Firebase Database
- A Beacon Node.js service Notification Service dispatching notifications and adding them to the database

These three components interact to allow users to communicate with each-other via the mobile network. The following base functional requirements are required by the network in order to achieve their goals:

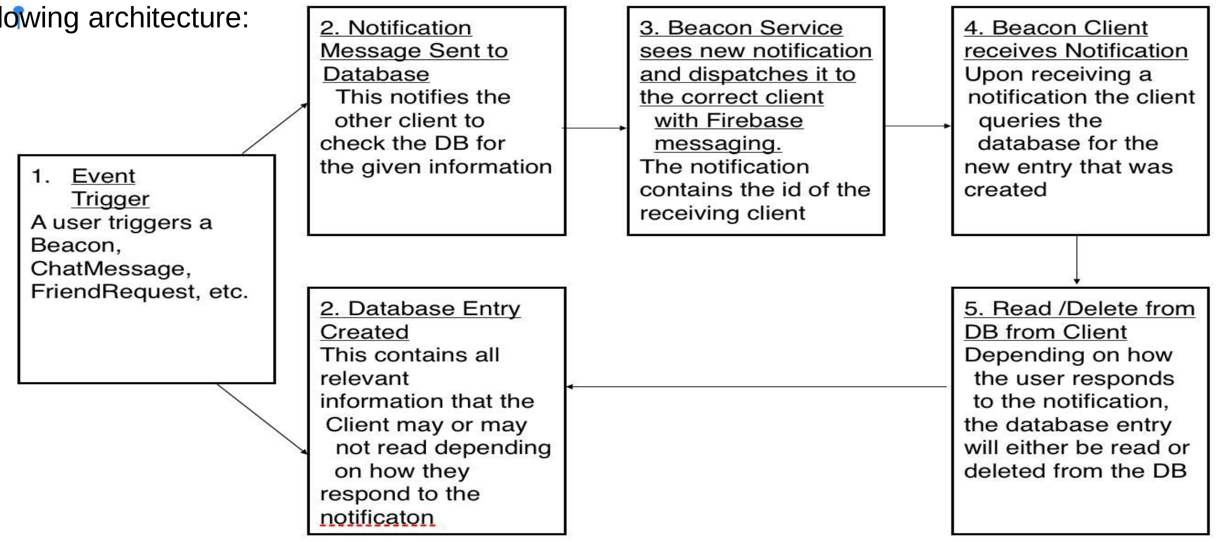
Beacon broadcasting: Users have the ability to send their location to one or many users via the internet.

Friends-List: Users can add their friends who have the Beacon application to a list of contacts. A friend request is sent via the network to another user. When this request is received the other user has the option to accept or reject the friend request. Upon accepting, a confirmation message is sent back to the user who sent the initial request.

Chat-Functionality: Users can send basic text messages to each other via the network.

Beacon Requests: Users can request other users send them a Beacon. If User A sends User B a Beacon Request, User B will be asked if they would like to send a Beacon to User A.

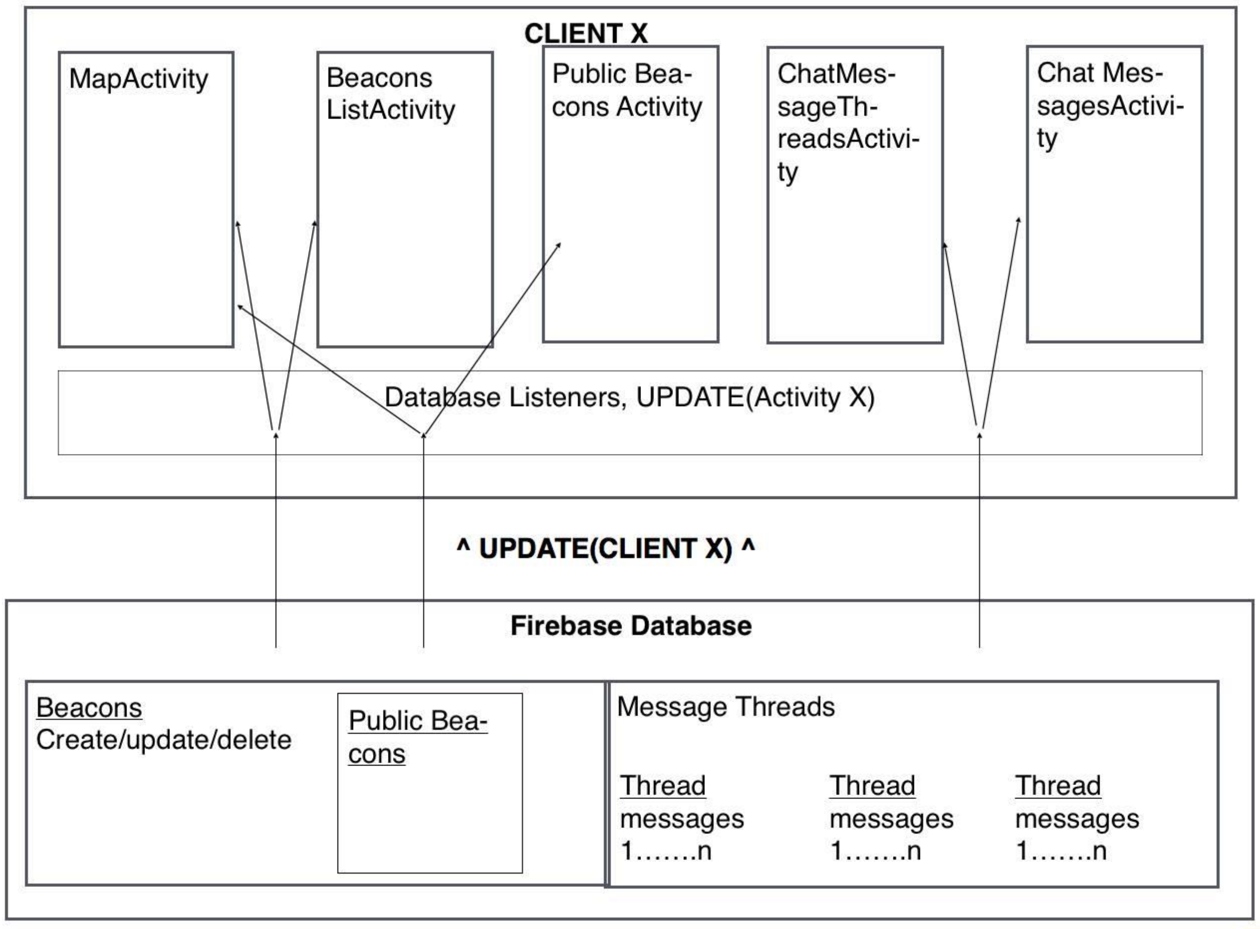
Public Beacons: Users can create beacons that anyone in their friend's list can see. These Beacons must be viewable by anyone with the application and must have their data accessible to all clients. The above functional requirements operate via messaging that behaves according to the following architecture:


1. Event Trigger
A user triggers a Beacon, ChatMessage, FriendRequest, etc.
2. Notification Message Sent to Database
This notifies the other client to check the DB for the given information
3. Beacon Service sees new notification and dispatches it to the correct client with Firebase messaging.
The notification contains the id of the receiving client
4. Beacon Client receives Notification
Upon receiving a notification the client queries the database for the new entry that was created
5. Read /Delete from DB from Client
Depending on how the user responds to the notification, the database entry will either be read or deleted from the DB
2. Database Entry Created
This contains all relevant information that the Client may or may not read depending on how they respond to the notificaton

USING THE OBSERVER PATTERN TO UPDATE CLIENTS AND THEIR ACTIVITIES

The Beacon Client's main goal is to display the Geolocations of a user's friends in a meaningful and intuitive manner. It achieves this by utilizing the Observer pattern. The Geolocations of a user's friends are stored in a map of Beacon objects that are updated via Database Listeners in the Client that query the database when notifications are received. When these Beacon objects are updated the GUI changes accordingly. The Beacons can then be viewed in a list, in the BeaconsListActivity, or on a map, in the MapActivity.

A similar pattern is used for the ChatMessages. When new chat-messages are added to the database, the correct users have their ChatMessage inbox updated via Database Listeners. The proper Android Activities are then notified to change; in this case, the ChatThreadsActivity and the ChatActivity. The ChatThreadsActivity, which shows each instance of a conversation between two Clients, must have its threads updated and the ChatActivity. The ChatActivity holds all the individual messages for a given thread, must have new individual messages displayed in the thread.
The following diagram illustrates this process:



USING SINGLETON PATTERN TO LOAD CLIENT DATA

Loading the correct client data is crucial to ensuring that each unique Beacon user sees their unique data when they log in.
These unique data include:
- The user's display name and picture.
- The user's friend-list.

- The Beacons that the user is following.
- The user's Beacons.
- The user's messages.

When a user logs into Beacon for the first time, a unique entry is created for that user in the database and they are assigned a unique user-id. On each subsequent login, this entry is reloaded from the database into a CurrentBeaconUser object. The CurrentBeaconUser object is a singleton that holds all the info needed to retrieve data relevant to the user that is currently logged in. The ability to load persistent data for multiple users is crucial for every functional requirement for this application. Using a singleton allows for crucial information to be retained while the application is running. Furthermore it allows for this information to be easily accessible throughout the Client's architecture.

USING THE PHONE'S HARDWARE AND EXTERNAL SERVICES TO SATISFY UNIQUE FUNCTIONAL REQUIREMENTS

Photo Sharing:

The Beacon client uses the phones built in camera, sd-card to satisfy the photo-sharing requirements.
The photo-sharing feature works in the following way:

- The CameraActivity is started from the MapsActivity;
- A picture is taken and saved in a static location on the phone's sd-card. This location can only store one photo at a time.
- When the photo is confirmed, it is uploaded to FirebaseStorage where it can be downloaded by other users. This is separate from the Firebase database.

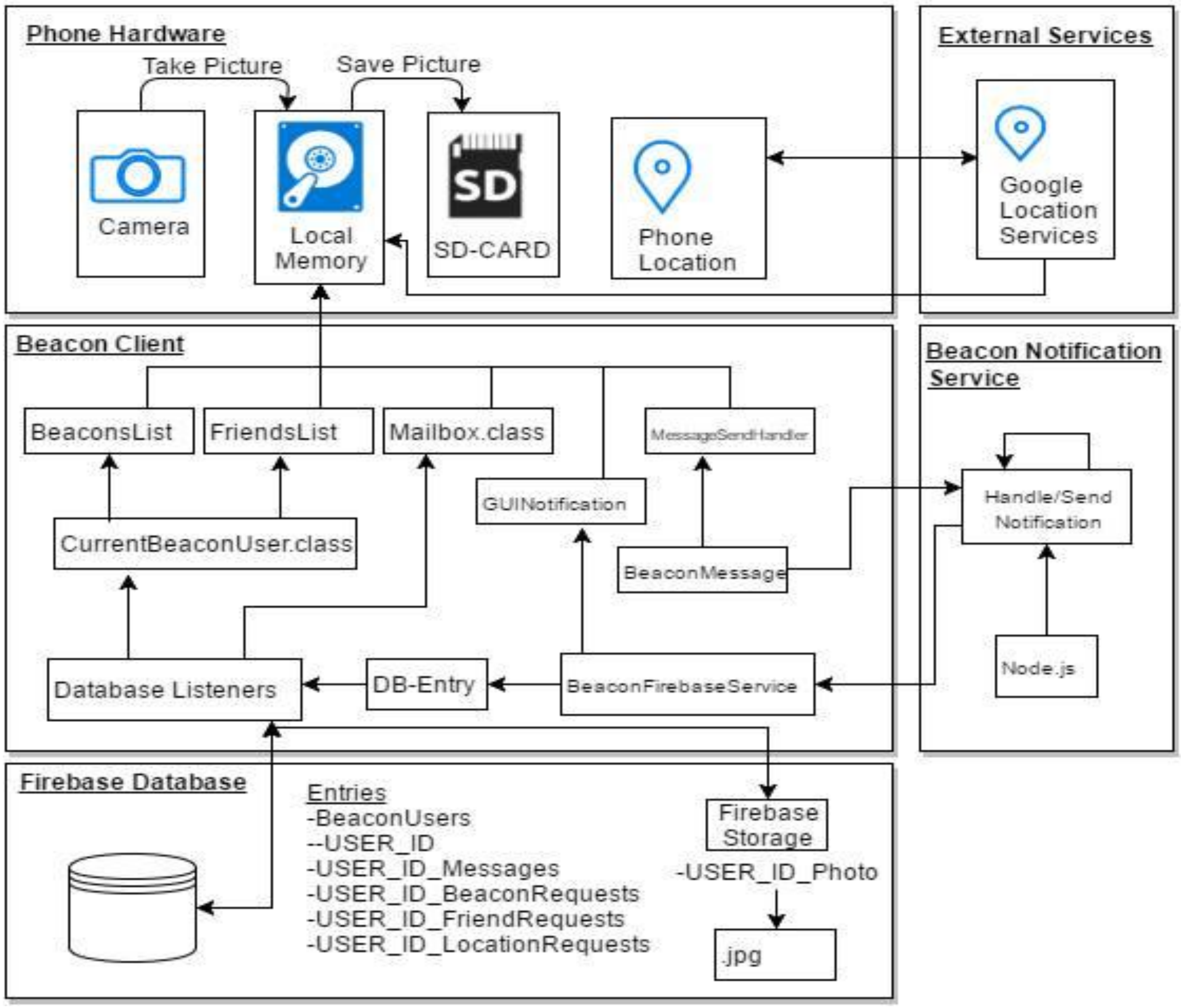
Location Tracking:

In order to satisfy the location tracking requirements, Beacon uses Google location services to determine the current location of a user. Additional location guidance functionality is invoked when using the compass to track a Beacon as Beacon uses the position and motion sensors that are built into most Android phones.

Authentication and Message Sending:

Beacon uses Google Firebase for Google and Facebook authentication. Furthermore, as outlined above, Beacon uses a Node.js service in conjunction with Google Firebase messaging to allow for communication between phones.

Beacon Overall Architecture Diagram



BEACON DESIGN

The class structure of Beacon is organized with each class assigned to one of the following sections:

USER

The User Class contains all classes that are relevant to a Beacon User. A Beacon User has an ID hash, friends-list, a list of Beacons that they follow and a list of Beacons that they have created.

All relevant data for the user who is currently logged into Beacon on a given phone is stored in the CurrentBeaconUser singleton class. The CurrentBeaconUser is a singleton because only one user can be logged into Beacon from any single phone at a time.

FIREBASE SERVICES

The Firebase Services Class encompasses all classes that interact with any of the Google Firebase Services. Three classes are in this category: the BeaconFirebaseMessagingService Class, the DatabaseManager Class, and the FirebaseUserService Class.

The BeaconFirebaseMessagingService interacts with Firebase Messaging and handles and displays notifications as they come in from the Server. The DatabaseManager creates, registers and handles all the DatabaseListeners and operations that actively perform CRUD operations on the Database. The FirebaseUserService is simply used on login to access the instance of the current user within Firebase.

DATABASE LISTENERS

As described above, the classes that are a part of the Database Listeners category activity create, read, update and destroy data within the database. There are currently 9 listeners, one for each type of message that a Client may receive from another user, i.e. BeaconRequest, LocationRequest, ChatMessage etc., as well as listeners that monitor when a Beacon user is updated. Furthermore, the Database Listeners load the current user’s Beacons, Friends and messages from the database into the device’s non-persistent memory when the application launches.

NETWORKING

The Networking section holds all the classes that are sent as messages to other users as well as classes that hold or send messages. The classes that are sent as messages are:

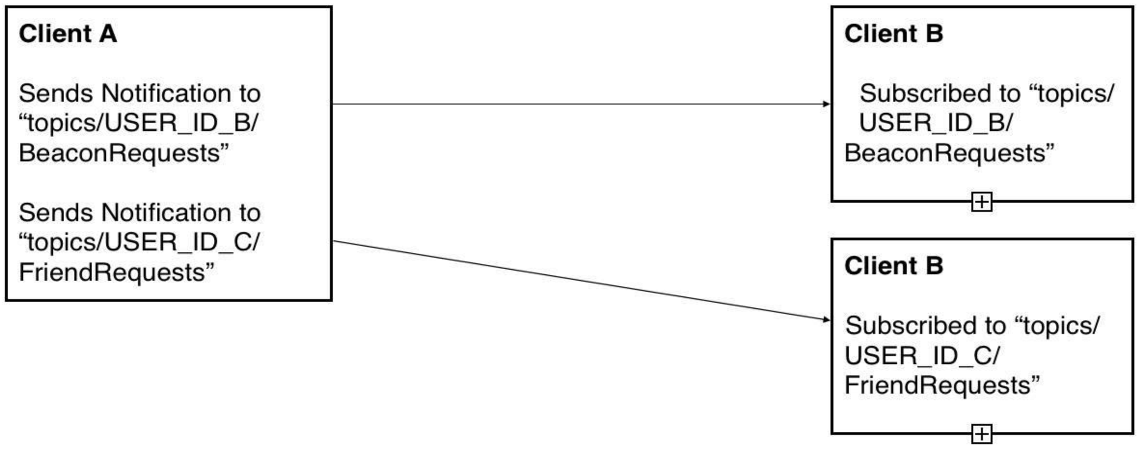
- BeaconInvitationMessage. This is the message that is sent when a user wants to send another user their location to track. When a user receives this message they are asked if they would like to track the user who sent the message’s Beacon.
- ChatMessage. This message is the basic instant-messaging message that is sent with text that a user has written to be sent to another user.
- FriendRequestMessage. Sent when one Beacon user wants another Beacon user to be on their friends list. Friends can send Beacons to each other.
- LocationRequestMessage. Sent when a user wants to know another user’s location; i.e, a user requests another to send them a Beacon.
- RegisterUserMessage. This is only sent once to the database when a user logs into Beacon for the very first time. A database entry is created for the user.

All these messages are sent to the Database and other Beacon Users via the MessageSenderHandler class. The MessageSenderHandler handles all *outgoing* messages and ensures that both a Notification Message is sent and a Database Entry is created. The PhotoSenderHandler is a similar class but only sends photos between Clients. Photos are not stored in the Firebase Database but are stored in Firebase Storage and are therefore handled in a different class.

The PublicBeacon handler holds and organizes all Public Beacons that are loaded from the database. Public Beacons can be seen by anyone and therefore are held in a completely separate class. This class updates whenever a new Public Beacon is sent to the database which subsequently updates the UI.

The final class that falls in this section is the SubscriptionHandler. This class is crucial to ensuring that a User only receives messages that are relevant to themselves. Moreover, this class ensures that a User does not receive messages intended for other Users or no messages at all. The Subscription Handler works by subscribing a user to certain Firebase “Topics”. When a notification is sent to a topic that a user is subscribed to, that user receives said notification.

The following diagram illustrates this:


Client A
Sends Notification to “topics/USER_ID_B/ BeaconRequests”
Sends Notification to “topics/USER_ID_C/ FriendRequests”
Client B
Subscribed to “topics/ USER_ID_B/ BeaconRequests”
Client B
Subscribed to “topics/ USER_ID_C/ FriendRequests”

The subscription handler allows for more messages to be easily added to a Client. All Messages are serialized from Java objects to JSON data in the database. Therefore, if a new type of Message is needed for communication between Clients, only the following steps need to be taken:

- Ensure that the client is subscribed to the appropriate topic in the SubscriptionHandler.
- Create the appropriate Java class that will be serialized to JSON in the database
- In the MessageSenderHandler, write the method that sends the Java Object to the database. This includes creating a DB entry and sending out the actual notification.

NOTIFICATION HANDLERS

The Notification Handlers store notification data as it comes in and destroy the data when it is no-longer needed. Currently these handlers can only handle one notification at a time as they come in but can be extended to allow multiple notifications if an array of notifications is implemented in each class instead of holding a single notification in each class. Separate classes were chosen for Notification Handlers as all messages are different de-serialized JSON objects and have different data values.

LOCATION MANAGEMENT

Location Services in Beacon are used to obtain a current users location, calculate the distance between users and find nearby locations. The following will outline finding a user's location and furthermore finding nearby locations.

The user's location is obtained using the locationService class. This class is a location listener that can be accessed from anywhere within the app. As an abstract class, so it can be used to do different things when it obtains the user's location. The onLocationChanged method in the location service is an abstract method that other classes need to have implementation for.
Since the app requires a user to be connected to the internet, it uses the phone's network provider. It is not as accurate as gps but it takes less time to activate.ion Management holds all classes that get and store the CurrentUser's Location and nearby locations. The LocationListener gets the users current location for the entire application and stores it in the MyLocationManager object.
The Place_JSON class is used to parse the JSON data received from a Google Place Search query into APlace objects that are viewable in the NearbyPlaces activity.

Nearby places are obtained from sending an http request to Google's Places API.
After obtaining the user's current location, a URL is created with the user's latitude and longitude as its parameters. The response is a JSON object that is parsed to obtain the data for each place.
The place's name, address, and coordinates are obtained and are used to create place objects used by the app. A listview is populated with the information for each place, along with its distance from the user's current location. The distance is calculated by taking the user's latitude and longitude, the place's latitude and longitude, and calling the getDistanceFromLocation() method.
When a user clicks on an item in the list, they can either set up a beacon there, or start tracking the place's location.

GRAPHICAL USER INTERFACE (GUI)

This class contains all of the Android Activities for the application. A brief description of the Activities is as follows:

Login Activity. The Login Activity allows a user to login using their Google or Facebook account.

Maps Activity. The Maps Activity displays all the User's Beacons and the Beacons they are following on the map in the correct locations. The activity also launches the Android Camera activity to allow for a user to take a picture.

FriendList Activity: The FriendList Activity displays all the friends of the current Beacon User.

UserSearch Activity: The UserSearch Activity allows for a user to search for other users and add them as a friend.

ChatMessageThreads Activity: The ChatMessageThreads Activity displays each individual chat conversation between 2 beacon users. Clicking on the conversation displays the Chat thread.

Chat Activity: The Chat Activity displays Chat messages between two users.

Public Beacons Activity: The Public Beacons Activity Displays a list of all the users who have broadcasted Public Beacons.

My Beacon Activity: The My Beacon Activity displays all the Beacons that the Current User has broadcasted.

Beacons List Activity: The Beacons List Activity displays all the Beacons that the Current User is following.

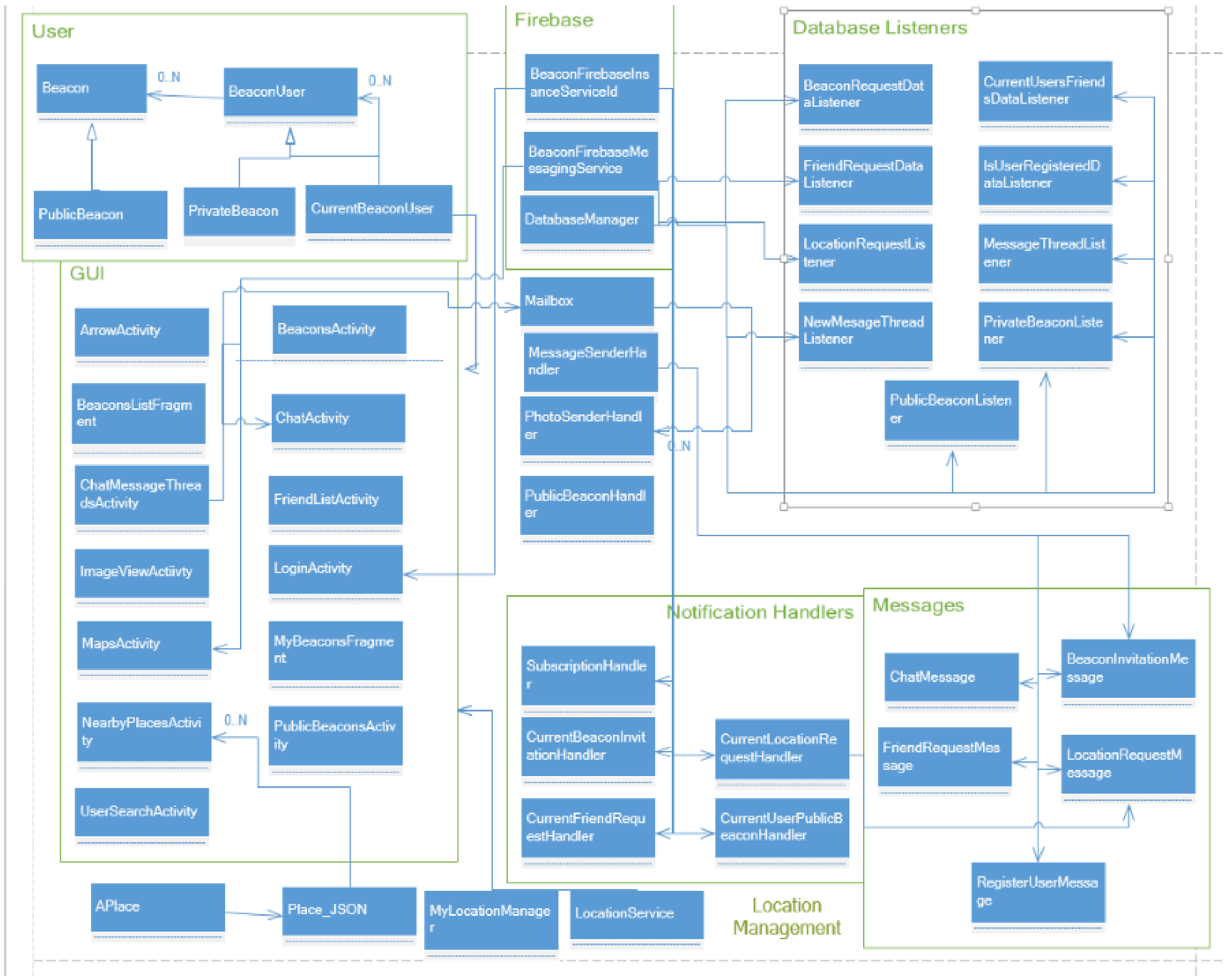
Arrow Activity: The Arrow Activity displays a location-guiding arrow that uses the Google geo-location services to guide a user to a Beacon.

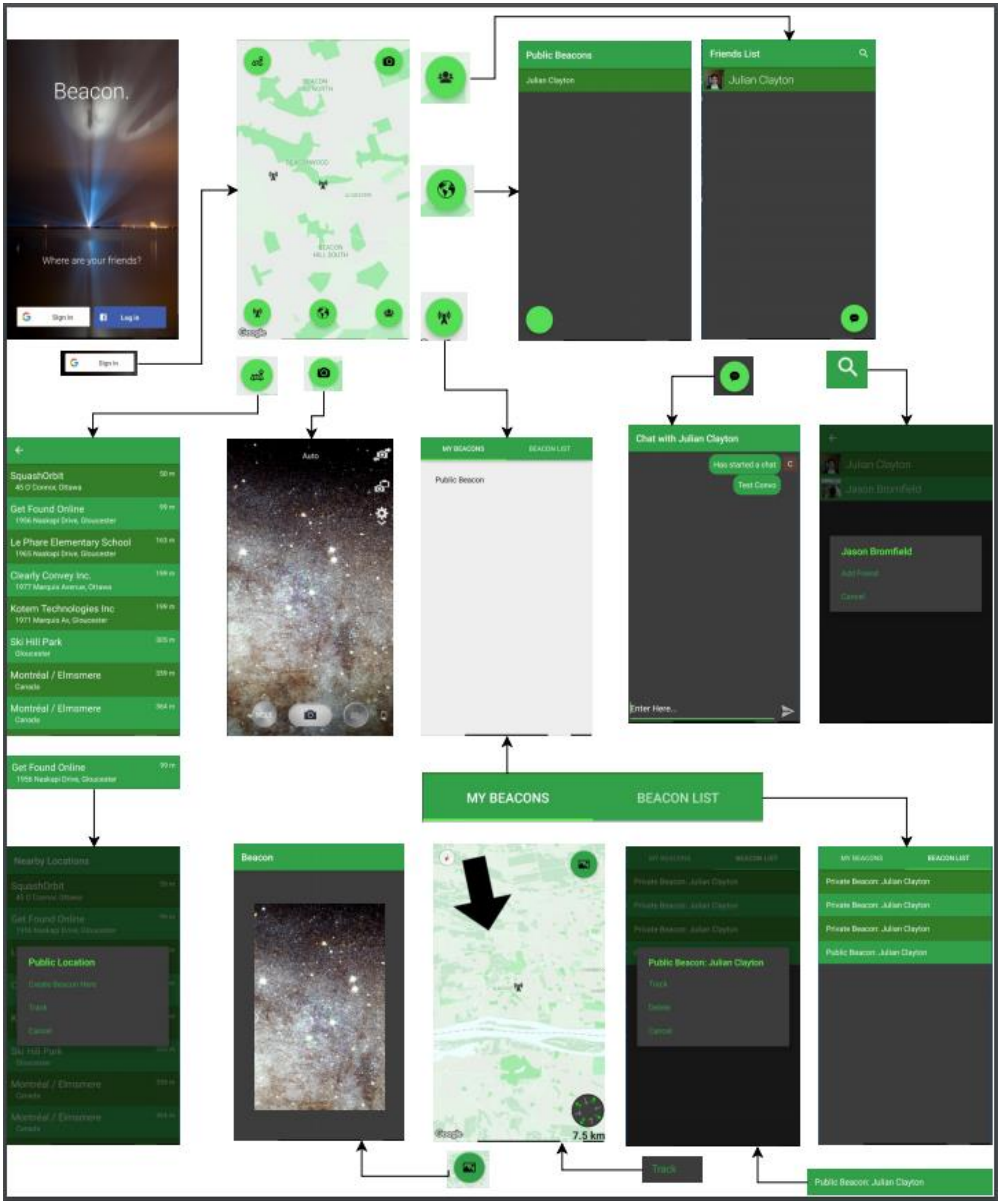
Image View Activity: The Image View Activity allows a user to view the image attached to a given Beacon. A diagram of these activities is presented on the following page.

Expansion:

The most likely ways that Beacon will expand from here is mostly in GUI improvements as the functionality is all laid out. Such improvements include different colours for different types of Beacons, photos viewable from map etc. Perhaps the next big improvement would be to allow users to have a Beacon profile instead of using a Google or a Facebook account.

Below: Beacon Class Diagram




Beacon.
Where are your friends?
Sign in
Log in
Sign in
Public Beacons
Julian Clayton
Friends List
Julian Clayton
SquashOrbit
45 O Connor, Ottawa
50 m
Get Found Online
1956 Naskapi Drive, Gloucester
99 m
Le Phare Elementary School
1965 Naskapi Drive, Gloucester
163 m
Clearly Convey Inc.
1977 Marquis Avenue, Ottawa
199 m
Kotem Technologies Inc
1971 Marquis Av, Gloucester
199 m
Ski Hill Park
Gloucester
305 m
Montréal / Elmsmere
Canada
359 m
Montréal / Elmsmere
Canada
364 m
Auto
MY BEACONS
BEACON LIST
Public Beacon
Chat with Julian Clayton
Has started a chat
Test Convo
Enter Here...
Julian Clayton
Jason Bromfield
Jason Bromfield
Add Friend
Cancel
Get Found Online
1956 Naskapi Drive, Gloucester
99 m
MY BEACONS
BEACON LIST
Nearby Locations
Public Location
Create Beacon Here
Track
Cancel
Beacon
7.5 km
Private Beacon: Julian Clayton
Private Beacon: Julian Clayton
Private Beacon: Julian Clayton
Public Beacon: Julian Clayton
Track
Delete
Cancel
Private Beacon: Julian Clayton
Private Beacon: Julian Clayton
Private Beacon: Julian Clayton
Public Beacon: Julian Clayton
Track
Public Beacon: Julian Clayton

PROJECT DUTIES BY GROUP MEMBER

JULIAN CLAYTON

1. Networking/Messaging. Messaging for Beacon encompasses all communication needed between clients. This include Beacons, Beacon Requests, Chat Messages and Friend Requests. Furthermore, this included writing and maintaining the BeaconNotificationService which is a Node.js service.

2. Database Management. Database Management is required for the above messaging requirement and ensuring that each unique user had their unique data stored in persistent memory.

JASON BROMFIELD

1. Location Services. This includes using Google Location services to determine the Geolocation of a client as well as the triangulation algorithm that is used by the ArrowActivity to calculate how far a user is from a beacon.

2. Nearby locations. The ability to create Beacons by looking at a list of locations that are near to the user.

3. Friend Search. The ability to query all existing Beacon users and send Friend Request to them

NOLAN HODGE

1. Arrow Activity. Using the built-in motion-sensors to guide the user in the right direction

2. General UI design. Choosing images and colors for the activities.

CAMERON MACQUARRIE

1. Overall Activity flow and Design. This includes designing all the user-facing elements in the activities to be organized in a way that is intuitive to the user.

2. Design planning and project management. Diagrams for application design and ensuring requirements were met.