Robert Kim

**Argus: Monitoring and Assisting the Elderly Through Smart Home and GPS Tracking**

**Abstract**

*Currently, elderly people or those living with dementia or Alzheimer’s may have trouble in their everyday lives, and may need to be monitored. There is no existing technology that directly solves this problem. To solve this problem most efficiently, I am developing an Android app that can mitigate this problem by monitoring many aspects of a “Trackee’s” life, such as their location and status; and by assisting them with everyday tasks through technologies such as global positioning system (GPS) tracking.*

**Introduction**

The name “Argus” comes from a character in Greek mythology whose full name “Argus Panoptes” means “Argus the All-Seeing.” Argus, a character with over a thousand eyes, was appointed by the goddess Hera to watch over the cow Io.

Likewise, the elderly may need assistance in their everyday lives, and often do not have access to caretakers or someone else to watch over them. The purpose of this app, Argus, is to provide a “virtual caretaker” to monitor the activity of those who may need constant assistance. For example, people with Alzheimer’s tend to wander, so it is important to keep an eye on them. Additionally, the elderly may have trouble with everyday tasks which can be solved through an application with technologies such as global positioning system (GPS) tracking.

Argus attempts to solve this problem in a way that is intuitive for both the caretaker, referred to as the “Master” within the app, and the person being tracked, referred to as the “Trackee.” Argus helps the Master set boundaries for a Trackee to stay more informed on their whereabouts.

The Background section details the various technologies necessary for Argus to function properly, the Development / Techniques section details the standards for success along with the general functionality of Argus, the Materials section gives recommended hardware to write and run Argus properly, the Results / Discussion section details the current state of Argus and how it fares against the standards for success.

**Background**

Android is an operating system developed by Google primarily for mobile devices such as smartphones and tablets, running on over 2 billion devices globally (Ng, 2017). Android provides a variety of service, most notably, through Android applications (“apps”). All aspects of an app can be created through Android Studio, the official integrated development environment (IDE) for Android. These apps are primarily written in the Java programming language. Java is a language that can be compiled once and run on any platform that supports Java, having the motto “Write once, run anywhere.”

Android apps are composed primarily of Activities, Fragments, and Views. Activities are the most broad aspect of the user interface (UI) which the user can interact with. They generally take up the entire screen. Fragments are an optional subgroup that can be used within Activities to display aspects of the UI that may need to be reused in other areas of the app. Views are the smallest subgroup of the UI, which makes up a variety of different UI elements such as textboxes, known as TextViews.

A RecyclerView is a type of View that displays a list of elements. RecyclerViews are significant because they can scroll down through a variable list, which will change the RecyclerView accordingly. The RecyclerView reuses, or *recycles*, each row as it repopulates its data. Each row of a RecyclerView is a ViewHolder, which holds a single element of the list (Using the RecyclerView, n.d.).

There are multiple methods to determine a devices location through an Android app. The app can use a variety of location sources, including GPS-provided location, network-provided location, and cellular signal-provided location. GPS-provided location is optimal, but it uses the most battery and does not work as well indoors. Network and cellular signal-provided locations are less demanding on battery, but are less accurate.

There are multiple different application program interfaces (API’s) which can be used to manage these location providers. The two prominent API’s are the Android Location API (located at android.location.LocationListener) and the Google Play Services Location API (located at com.google.android.gms.location.LocationListener). The Android Location API is more customizable, but does not have many of the newer, intelligent features of the more updated Google Play Services Location API, which automatically manages location providers based on user preferences.

FusedLocationProvider is a tool from the Google Play Services Location API which can retrieve the last known location of a given device. FusedLocationProvider will automatically select a coarse or fine method of determining a user’s location while also maintaining higher battery efficiency. Additionally, there are two methods in the Google Play Services Location API to receive location updates of a given device, LocationCallback and LocationListener. While the LocationCallback is recommended for asynchronous tasks, the LocationListener has a onLocationChanged method which is equally as viable. The Google Play Services Location API LocationListener is more streamlined than the standard Android LocationListener, especially when used in a background Service. A background Service, or Service, is a class provided by Android which allows an app to run processes in the background, or while the app is not visible to the user.

A geofence is a virtual boundary around a geographic region. Ideally, a geofence has two states: a given point is either inside or outside of the geofence. There are multiple methods to determine if a given point is within the geofence or not.

Both the Google Play Services Location API and the Android Location API provide built in ways to create a geofence, but they are limited to circular shapes. There are a variety of methods to implement a more flexible way of creating geofences. For example, given a polygon with coordinates for each of the vertices, a ray casting algorithm can be used. Ray casting works by drawing a ray in any direction from the point in question and counting the number of intersections the polygon has with this ray. If there are an even number of intersections, the point is outside the polygon. There is also a built in method in the Google Maps Support library that isn’t mentioned in any of the guides, PolyUtil.containsLocation(LatLng point, List<LatLng>polygon, boolean geodesic) (Polylines and Polygons to Represent Routes and Areas, 2018). This opens up the door to more complex shapes, as long as the list of Latitude-Longitude pairs forms a proper polygon.

Firebase is a mobile and web app platform owned by Google which provides a variety of cloud-based services. Firebase provides the Firebase Database service which allows application data to be synchronized across devices and stored on the cloud. Because Firebase is built off of WebSockets, it can efficiently transmit data in realtime. WebSocket is a protocol that allows a server to transmit data to a client without first being requested by the client.

Firebase also provides an authentication service, Firebase Auth, which can help users log into an app or database with only client-side code. Each user is assigned a unique user id (UID) which can help partition the Firebase Database.

**Development / Techniques**

This project will be considered a success if it can 1. Determine a trackee’s location from their device and update it on Firebase in realtime, 2. Retrieve that location and display it on a master’s device, and 3. Give notifications to the master device if the trackee leaves a specific region.

Argus first launches into an EmailPasswordActivity where a FirebaseAuth object is instantiated. If a user has logged into the device previously, it will automatically log them into their account. Otherwise, they will be prompted to log-in manually with the signInWithEmailAndPassword method, or create a new account with the createUserWithEmailAndPassword method. When creating a new account, a user will be given the option to create an account as a Trackee or a Master account, both of which have separate Activities.

As a Master account, the user will be then sent to the MainActivity, which uses a BottomNavigationView as its layout. This provides options for the Master to see a list of their Trackees with the time of their last location update, a map of their Trackees and their locations, and a settings page. These separate tabs are all brought to view with Fragments and a FragmentManager.

The TrackeesFragment is the first tab of the application. The main contents of this Fragment is a RecyclerView, which is used to hold information on each of the Trackees associated with a Master account. Each item of the RecyclerView is a LinearLayout holding a TextView for the name, a TextView for the time and date of the last location update, a Button to edit the Trackee’s name and boundaries, and a Button to remove the Trackee from the account. Trackee information is retrieved from the Firebase Database and stored into TrackeeModel objects, which store the Trackee’s name, UID, last known location, time of the last location update. This conversion is done with the TrackeeAdapter, which also attaches an ValueEventListener to the item using the Trackee’s UID, to update it in realtime as the Trackee’s location changes.

When the “Edit” button is selected in the TrackeesFragment, an AlertDialog.Builder displays the BoundariesFragment. The BoundariesFragment contains several EditTexts regarding the name and boundaries of the selected Trackee which are populated with data retrieved from Firebase. On submission of the AlertDialog.Builder, Firebase is updated with the data from the EditTexts.

In the next tab, a CustomMapFragment populates the View. The CustomMapFragment uses a GoogleMap from the Google Maps Support library to display a map. Similar to the TrackeeAdapter in the TrackeesFragment class, it retrieves location data for each of the Trackees in realtime, updating the map with MapMarkers using the Trackees position labeled with their respective names.

Another service, MasterService, retrieves data in the same way as TrackeesFragment and CustomMapFragment, but it is implemented in a background Service. Using the PolyUtil.containsLocation method, it determines whether or not the Trackee is within the specified boundaries. If the Trackee is outside of the boundaries, and it wasn’t already outside at the last check, it will send a notification through a NotificationManager.

The final Fragment is the SettingsFragment. This Fragment is a simple Fragment which allows the user to update their email and password. Email can be updated with the given EditText and the FirebaseAuth changeEmail method, which is called when the “Change Email” Button is pressed. When the “Change Password” Button is pressed, an AlertDialog.Builder displays the ChangePasswordDialog. This Dialog has EditTexts for “Old Password,” “New Password,” and “Confirm New Password.” When the Dialog is submitted, the FirebaseAuth changePassword method is called with the new password if the two passwords match.

On a designated Trackee device, the MainActivity is very different. There are only two tabs in the BottomNavigationView, one of which directs the user to the same SettingsFragment. The other just shows information about the Master.

The actual functionality of the app for a Trackee is housed in the LocationService, which is called from the onCreate method of the TrackeeMainActivity. This Service updates the Firebase Database with the Trackee’s location in realtime, using the FusedLocationProvider and the onLocationChanged method of the LocationListener using the Google Play Services Location API.

**Materials**

* Two Google Pixels, or other GPS-enabled Android (API 27) devices.
* USB Type-C USB-A cable
* Windows computer with Android Studio installed

**Results / Discussion**

Argus fulfills its basic purpose; it can retrieve the location of a Trackee device and transmit it to a Master device in real time, notifying them when it leaves a given boundary. However, there are a multitude of features which can be improved on. For example, the method I use to determine if a Trackee is within a boundary supports an infinite number of points to form the boundary polygon, but Argus only supports four at the moment. Additionally, while the user interface is intuitive with the BottomNavigationBar, there are other ways to display the app, something that may be useful when considering larger screen sizes.

Most significantly, there is no easy way to add Trackees to a Master account; it is currently done manually through the online Firebase user interface. This will be the next feature to be implemented, perhaps by displaying a temporary key on the Trackee’s device which must be inputted into the Master device.

**References**

Google Play Services and Runtime Permissions. (2017, March 21). Retrieved May 18, 2018, from <https://developers.google.com/android/guides/permissions>

Create a Notification. (2018, May 8). Retrieved May 18, 2018, from <https://developer.android.com/training/notify-user/build-notification>

Maps SDK for Android. (2018, May 10). Retrieved May 18, 2018, from <https://developers.google.com/maps/documentation/android-sdk/intro>

Ng, A. (2017, May 17). Google's Android powers more than 2 billion devices. Retrieved May 18, 2018, from <http://www.cnet.com/news/google-boasts-2-billion-active-android-devices/>

Polylines and Polygons to Represent Routes and Areas. (2018, May 10). Retrieved May 18, 2018, from <https://developers.google.com/maps/documentation/android-sdk/polygon-tutorial>

Structure Your Database. (2018, May 14). Retrieved May 18, 2018, from <https://firebase.google.com/docs/database/android/read-and-write>

Using the RecyclerView. (n.d.). Retrieved May 18, 2018, from https://guides.codepath.com/android/Using-the-RecyclerView