Link

Get connected

Final Report

Justin Richard

Version 1.0

December, 2016

Table of Contents

Recent Additions

Since the progress report was completed there have been a bunch of features added into the app. The following sections will outline what some of the more recent changes are and what they look like.

Active Link Sessions

The main core feature of the app now exists and is working well. The user interface can be seen in figure 1.0. The top portion of the screen is a Google Maps Interface. You can see your current location as a blue dot, and center the camera to yourself with the button in the top right of the map. The buttons in the bottom right will pull up directions in Google Maps to the selected person/pin, in this case it is the red Justin Richard. The bottom portion of the screen shows the members of the Link session, with each person showing their color on the map, their last update to the session, and their distance to your current location. Tapping on an item in the list moves the map to that person's location. In between the map and list of users is a progress bar which counts down the time until a new update is retrieved for the session. The length of this wait is set in the settings of the app.

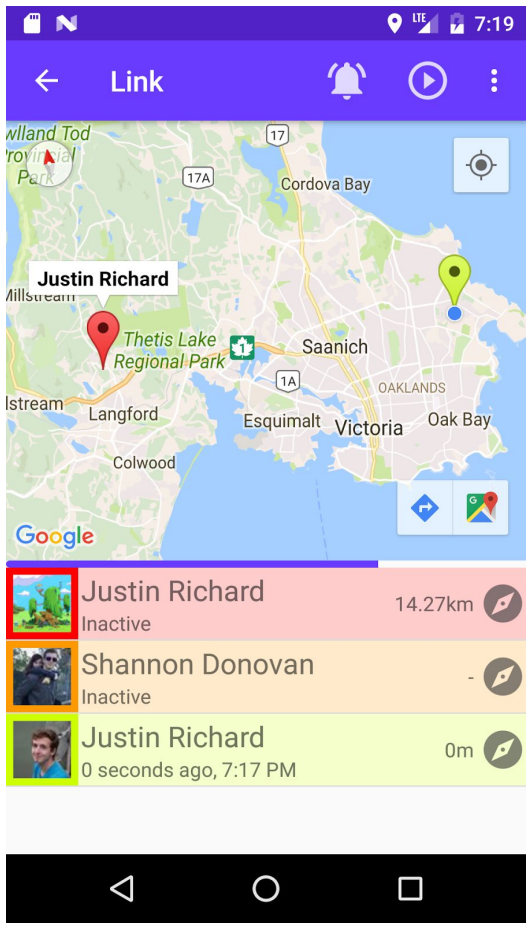


Figure 1.0 Active Link Session

Along the title bar at the top are a few extra functions to better the experience. The back arrow beside the title takes you back to the home screen. The first top right menu item which looks like a ringing bell is the notifications button, which sends a push notification to all users in the Link Session telling them that you wish to Link up. The notifications are in more detail in the next section. Next over is a pause/unpause button. This button allows you to temporarily stop receiving and sending updates to the Link session while you are looking at it, and will pause the countdown progress bar. Finally inside the drop down menu (Triple dots, top right) is an option to add more people to the Link session, since Link sessions are started between two users only and group sessions were desired.

Push Notifications

A key feature of the app is to be able to let your contacts know when you want to Link up with them. If the other person is not broadcasting their location then it is no use to you. Push notifications require the use of an external service to avoid having to use a polling approach with your app and staying awake. Instead, you create a service to run in the background silently and wait to receive messages from the operating system that get sent once a message is received. Google FireBase Cloud Messaging provides the cross-platform solution for no charge. When a notification is needed to be sent to the user, it is just a simple API call with their target users registration token (Received and stored when user initializes Firebase service by launching the app) and a message to send. If the user is online, the message is delivered to their device and handled by the service to create a notification. This is a really neat solution since the notifications arrive near instantly while still being efficient. Other application with notifications will use Firebase as well, meaning checking for updates checks for multiple apps at once saving resources.

Notifications appear differently depending on the operating system version. All that is set in the backend is an icon, color, message, and an action to be taken if clicked. In this case it would be to open the Link session being requested. You can see some sample notifications on two different firmware versions below in figure 2.

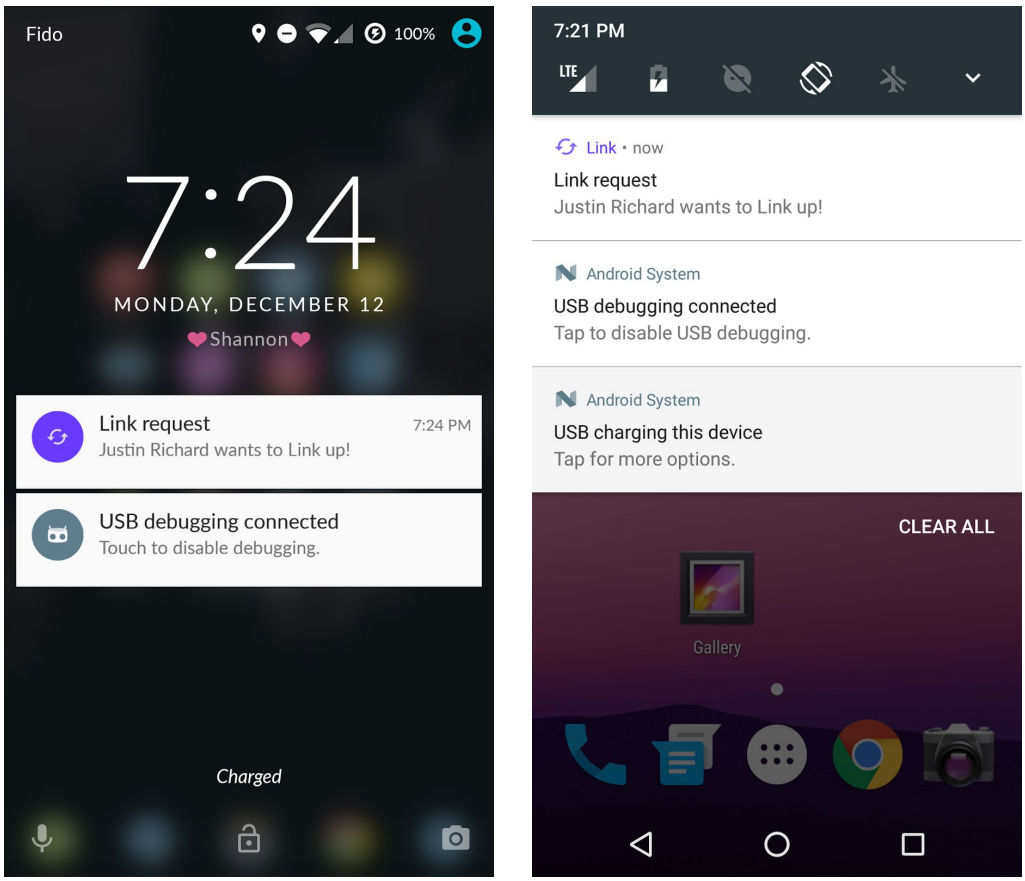


Figure 2.0 Notifications On Different Firmware Levels

Bugs and UI

The last set of changes since the progress report are less visually exciting and interesting at all. There were a wide range of bug fixes and UI tidy-ups to ensure a smoother user flow. There were bugs where after adding a contact the list of contacts would not automatically refresh, as I was having troubles creating a handle to the view in order to call an update on it. This sort of thing was left behind after initially adding the features since it technically worked anyways, I was more concerned with creating the original functionality.

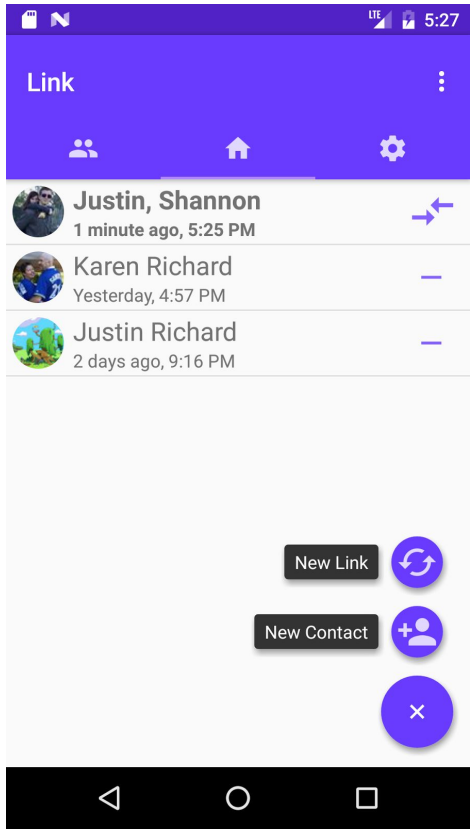


Figure 3.0 + Button In Homescreen Expanded

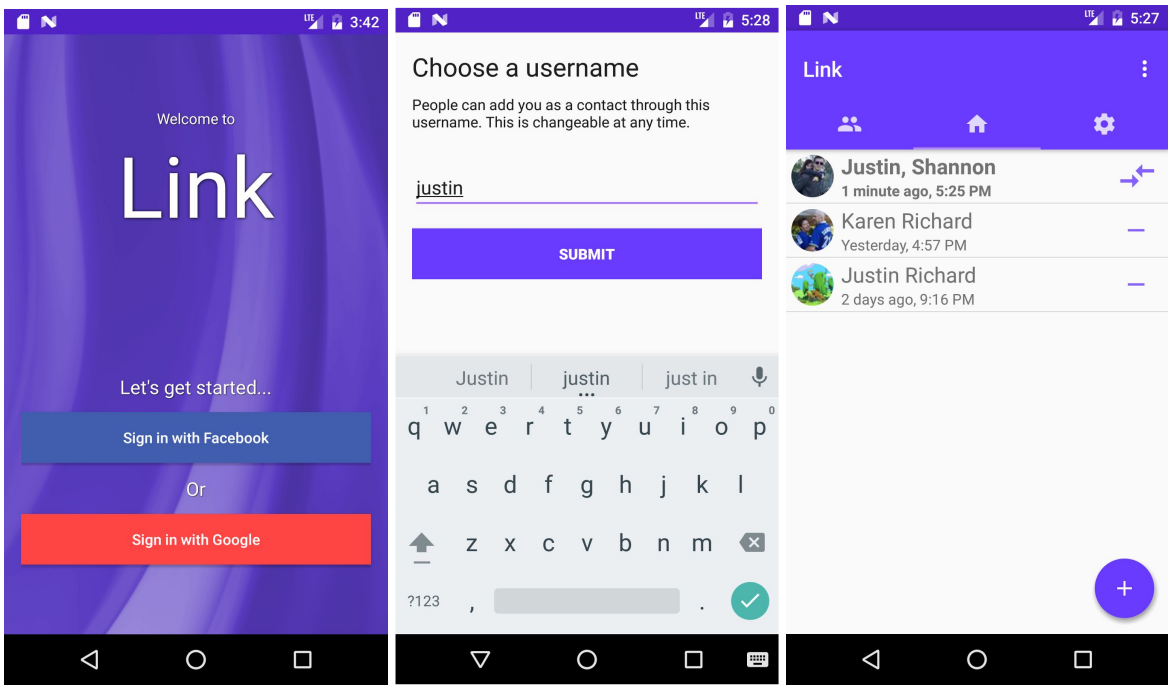
It was ensured that menu's would open smoothly and the functionality of each one was working as intended. A sample of the home screens plus floating action button can been to the right in figure 3.

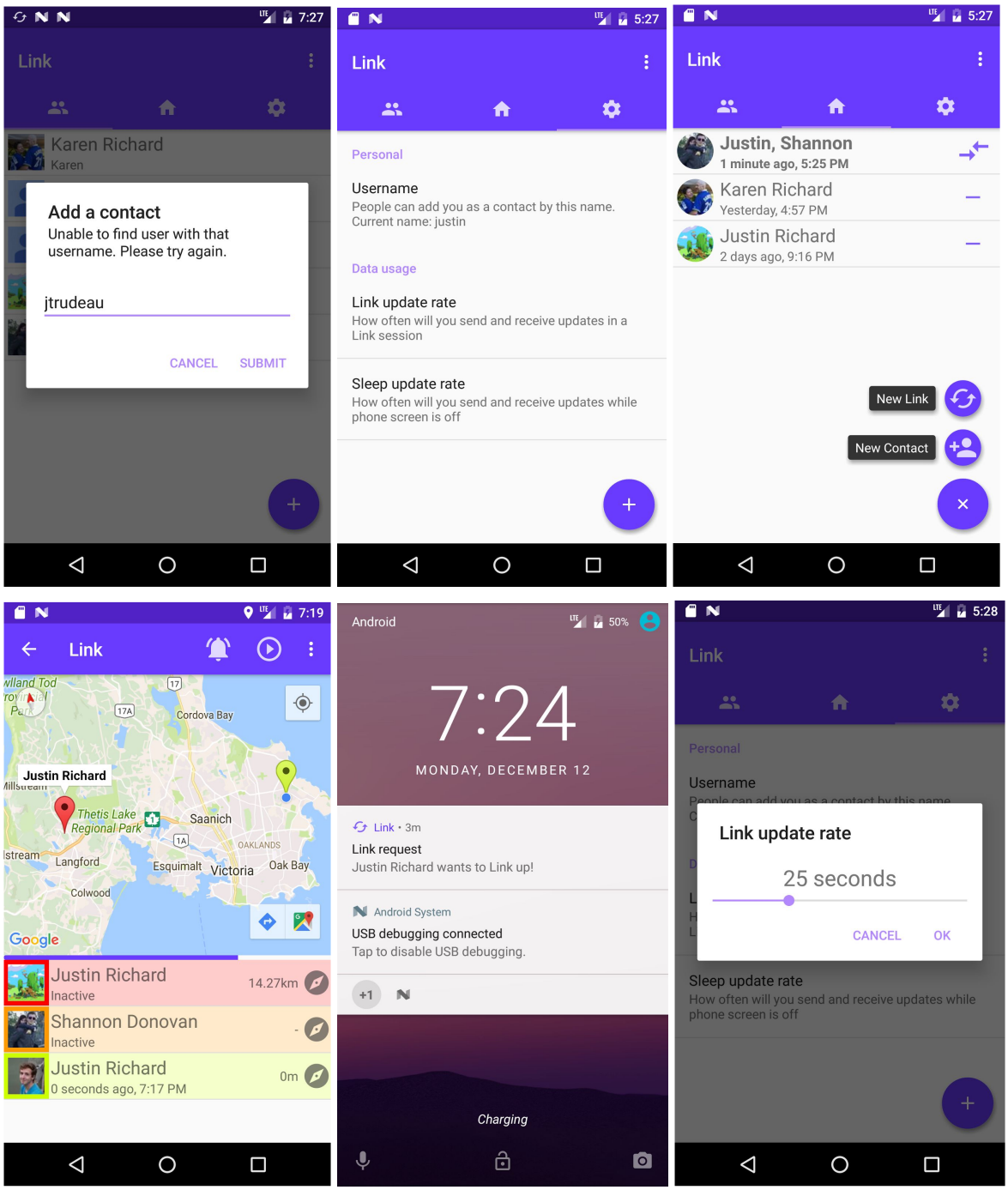
It has been a bit of a challenge to figure out and remove most of the crashes. There is a crazy amount of things that require exception handling, even where you would least expect it. Many portions of the application will be working, then one thing will act in a weird way and set of a chain of events that somewhere leads to accessing a null object and then another exception is thrown, crashing the problem. One specific point of difficulty was with permissions. Permissions are something that allow your app to do something more than the default basic actions, and take information from your users device or allow it to do something invasive. Users must allow permissions before the action it wants to complete can be done. In this case, getting the user's location requires permissions. This had added difficulty since at API revision 23, a fairly recent build of Android changed how permissions work, so now you have to handle permissions for older devices as well as newer devices. Older devices asked for all permissions at once when installing the app, but the newer versions require you to prompt the user for the individual permission when you actually want to use it. After hours of playing with it, I finally had it prompting for permission correctly. Unfortunately with a small range of devices to test on it's hard to tell if it will function correctly as it has been hit or miss, sometimes the app will crash but then restart and function correctly.

Final App

An application is best demonstrated through using it, or seeing a video of the action. The application is not ready to be published live on the Google Play Store at the moment as I do not feel it is ready for public use. A file has been attached to the submission folder called Link.apk. An APK is the file extension of an Android app, just like an exe is a windows executable. Before you can install it on your phone you will need to make sure that third-party apps are allowed on your device. Go to *Menu* > *Settings* > *Security* > and check *Unknown Sources* to allow your phone to install apps from sources other than the Google Play Store. Transfer the APK file to your phone, then open the file with any file explorer application and the installer will launch.

Some screenshots of the app are spread throughout the progress report and this design document, but I have condensed them here for ease of viewing. They display a variety of some of the possible actions you take within the app.





Reflections

After completing a large project like this it can be beneficial to look back and take in everything one has learned during the time. It can also be interesting to look back at how you strayed from the original plan, and why. In the following two sections I will go over just that.

Comparison To Original Design

While the original designs user interfaces were pretty close to what I ended up with, there are quite a few changes from the actual requirements to the final product. Most of the changes are due to a lack of time which is a pretty realistic occurrence in the world of software development. It's good to be exposed to the difficulties of predicting how long work will take since it’s a large problem for software developers. The main feature left out would be finding friends from your social media platform that also use the app. The core backend was also changed, I will go more in depth into what services got changed in the Amazon Web Services section below. Overall I am pleased with the final product compared to the original design as I believe it captures the idea very well.

What I learned

Throughout this project I have gotten to learn a handful of things, so many that I can’t even remember and list them. Not to mention, many of the skills and experiences cannot be labeled easily which makes it tricky to discuss, but regardless I have gone ahead and narrowed down what I’ve learned into several categories which are in the subsections below.

Android Studio

Android Studio is one of two commonly used IDE’s for Android development, the other being Eclipse. Android Studio is built on top of a popular IDE, IntelliJ IDEA which I had actually used years ago for some Java programming I believe so it interesting to see it again. It was really interesting getting to learn this tool as it is a very powerful IDE. I of course already had experience with normal IDE’s for Java and C development, but Android Studio has a ton of cool debug features being able to debug your application on emulators or real devices and being able to filter through console messages very easily. The emulator for Android Studio, called AVD for Android Virtual Device is extremely powerful letting you basically run a phone inside of your computer with complete functionality and control over sensors and GPS locations etc. Android Studio also has a lot of neat updating functions to streamline updates, as well as Gradle integration simplifying builds and including libraries. I found the code completion and suggestions to be some of the best I have experienced in any IDE which was very helpful.

Activities and Fragments

Activities and Fragments are the bread and butter of an Android app. They essentially hold the content for your app and are kind of like the different ‘screens’. To move between actions in an app, you generally would just swap to a different activity passing data between. Fragments are essentially a mini-activity, used for a portion of the interface in an activity. Fragments are always embedded in an activity, and often many fragments are combined into a single activity for a multi-pane UI. Fragments can also be reused between different activities which is handy to avoid code duplication. Activities and Fragments both have their own lifecycle which must be looked after. They both have a list of methods such as OnAttach, OnCreate, OnCreateView, OnStart, OnResume, OnPause, OnStop, OnDestroy

etc. one for each possible state of the activity/fragment. Each of these functions should be overriden to provide the appropriate action when that part of the lifecycle is reached. I had to research each of these states and learn what should happen in the method in order to make my components function correctly.

For the Link app, there are five activities, a splash activity, a sign-in activity, the username selection activity, the main activity, and the Link activity. Inside the main activity is a tabbed layout which holds three fragments for the contacts, list of links and the settings.

Views and UI

The method Android uses for creating their user interfaces I thought was pretty interesting as it was something I had not seen before, and was quite different from web development UI's which I have experience with. All user interface elements are composed of View and ViewGroup objects. Anything that draws anything on the screen that a user can interact with or see is a view. So if you're adding a label, it's a TextView, or an image is an ImageView. ViewGroup objects are simply objects that hold other View objects and control the layout of them. There are some built in ones, such as Linear layouts, Relative layouts or Frame layouts, but you can of course build your own custom View and ViewGroup objects. ViewGroup objects can be nested together, as seen in figure 4. This turned out to be really neat, from inside the code you can access any UI element by calling a function findViewById and supplying the Id of a view, then you can simply cast the result to whatever kind of view you're trying to receive and then you have the instance of the object and can manipulate it however you need. Most of the view objects share a bunch of common methods and properties you can use. While tricky at first, handling the UI was something new and refreshing.

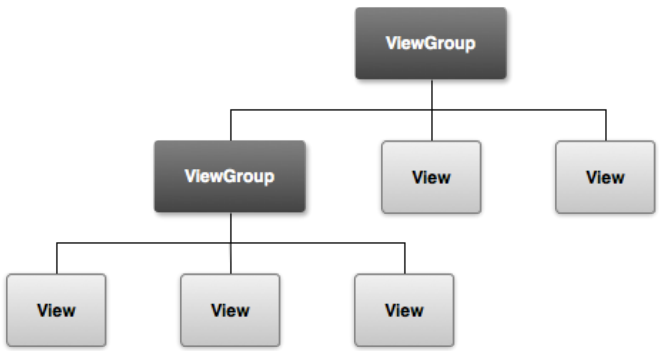


Figure 4.0 Illustration of a View Hierarchy

Asynchronous Programming

Up until now, I have not been exposed to very much multi-threaded or asynchronous programming where the performance is critical. I have worked with some threading in C and some callbacks in JavaScript, but they were both non critical. I had some experience using Golang for distributed systems, but that was using multiple different systems operating independently, so it was still pretty different. One thing that kind of surprised me but I found really neat, is that if you complete any sort of networking operation on the main thread that

an exception is thrown, crashing your app. Any network operation must be completed on a separate thread. Change UI elements is often done on another thread as well, as it can cause the main thread to skip or lag causing elements to redraw poorly and have choppy interaction. I ran into an issue where I wanted open an activity, but I want to display some data from the database on it, but I can't simply wait until I receive the data to finish drawing the activity. This requires the use of AsyncTasks which are separate runnable threads you can call which have two important functions you override, doInBackground and onPostExecute. The doInBackground method is where you complete the network activity and processing, and the onPostExecute is used to manipulate the UI based on the results from the doInBackground. The main thread should display some sort of loading screen while it waits for the AsyncTask to display the results. Since this app uses lots of network resources and should react accordingly, I got very accustomed to adding asynchronous tasks.

Amazon Web Services

AWS is the backbones for the apps functionality. A variety of AWS services were originally planned to be used in the original Link design document, but later on the plan was changed and some of the services were scrapped. The AWS services I ended up playing with are discussed below.

API Gateway + Lambda

The original intent was for the app to make API calls to my own API created with API Gateway. The API Gateway will expose an API I can call from anywhere, and with it I can call Lambda functions. Lambda functions are runnable chunks of code stored on the cloud. I can pass inputs into the API gateway, which passes them into Lambda, which then can store the results in DynamoDB or return values. This seemed like a great idea, except I realized later I could just do the computation directly on the users device and store the result straight into DynamoDB. I can save myself money by cutting out the extra services and make the user do the basic storing of data. The only issue is that if I am developing the app for iOS I need to reprogram the logic for those computations opposed to just calling the API, but in the end since I am just developing the Android version of the app this is more efficient overall.

DynamoDB

I had heard of DynamoDB previously, but hadn't been able to play with it until now. DynamoDB is a fully managed non-relational database service with fast performance and high scalability which made it desirable to use. By using DynamoDB I avoided having to manage my own database and having to worry about expanding it in the future and dealing with sharding issues. I previously have just used relational databases, so it was interesting to learn about the NoSQL database style, and how the data is handled. It began with "What! There's no joins?" to "Okay thats pretty neat" fairly quickly. The one complaint I have about DynamoDB is the awkward web interface, it is very clunky to try and view and edit values from within it. Other then that caveat, DynamoDB is super useful and the schema-less style was refreshing.

DynamoDB is accessed through the Amazon Web Services Android SDK in the app to retrieve and store data. A helper object was created which has a long list of methods to obtain or set different groups of data. The helper object class removes clutter from other areas of the code, allowing for database actions to be completed with one simple line such as db.GetUserFromUserId(UserId).

Cognito

Amazon Cognito is a service that allows you to create your own user pools and authenticate them using federated identity providers. This meant I did not have to build a custom authentication system into my app, and I could require the users to sign in through Facebook or Google and a Cognito user would be created for that person. This gave me a unique userId for the user which I could then keep and use as their identifier. Cognito is very powerful and helped avoid having to set up the federated identities myself and dealing with the access and refresh tokens and the pains that come along with them. The app just checks to make sure the token provided is still valid, then considers them as a logged in user with the corresponding userId. If a token is not valid, the user is pushed out to the Login activity.

Mobile Hub

After playing with all of the other services, I finally ran into Mobile Hub. The hub is a place to get people started quickly with developing an app. You pick and choose which features you want and then customize and implement them. You could very well do this yourself by using the individual services, but this was a nice way to add cohesion bringing them together into one place. Inside the Mobile Hub I selected I was using user sign-in (Cognito) and NoSQL Database (DynamoDB) and it created a user pool and DynamoDB instance for me. The Hub then provides you with a small code package which is customized to your project. This code package imports the appropriate libraries and gives you some methods to call to initialize each of the services you selected. In my case it added objects for each table in the DynamoDB instance, and classes which contained methods for authentication. There are also a few code examples for basic things such as “How to retrieve the user's current identity” etc, but I found following those gave me troubles as it would not work with errors I could not understand without understanding how the rest of library worked. It was helpful to go to the individual services documentation to gain an understanding of the SDK.

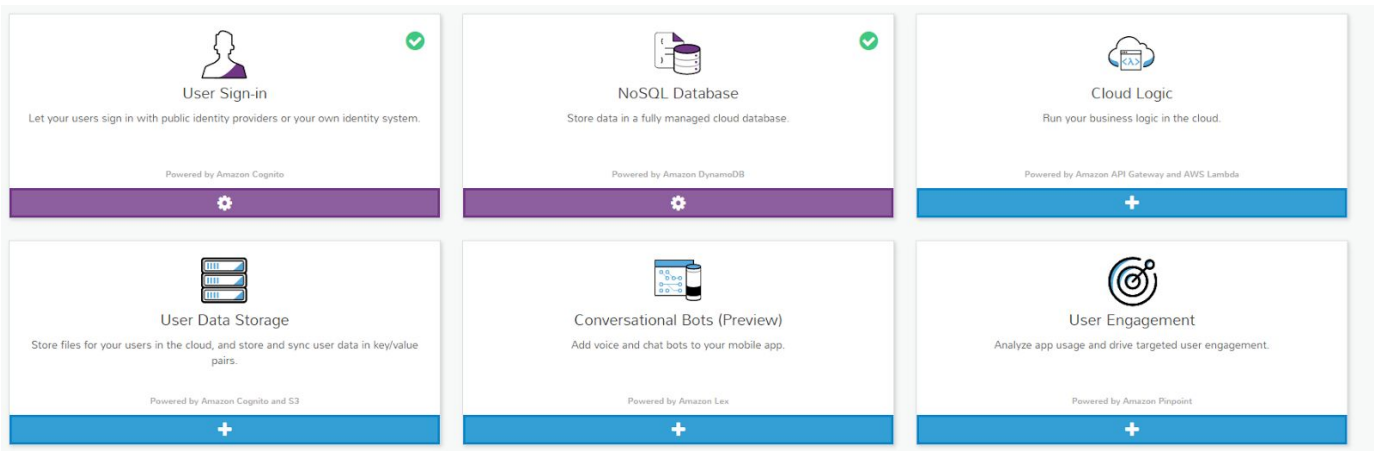


Figure 4.1 Mobile Hub Console - Services Screen

Google APIs

The Google API's seemed interesting but unfortunately I did not get to play with them all too much. Most of the work inside the Google API's was enabling their various API's to be accessed through the app. The API's limit calls to come from your specific app only, so I had to provide SHA1 signatures as well as the package name of my app to create a project. The app mainly uses the Google+ API for user authentication, and the Google Maps API for the maps interfaces. Both of these API's are provided free of charge. The actual API calls are completed inside the Google SDK and exposed with methods, so the dirty work is largely hidden.

Google Firebase

Before this project I hadn't heard of Google Firebase at all, although it is relatively new (2012). Firebase is described as a "mobile platform that helps you quickly develop high-quality apps, grow your user base, and earn more money. Firebase is made up of complementary features that you can mix-and-match to fit your needs." and they are more than correct, they have a ton of cool features to really take your app to the next level. For this project though, I am only using it for the cloud messaging feature to deliver notifications to users.

Getting started with Firebase was interesting, because I was able to start by importing my project over from the Google APIs. When I signed into Firebase I was prompted to migrate the project over, meaning it knew my apps information such as package name, SHA1 fingerprints etc. Once inside, I was able to send notifications through the online console to any registered Firebase instance Id. The instance Ids are unique identifiers obtained when the user launches the app, so you can target a single user with the messages. You can also target groups of users, or all users of the app, but that was not relevant for this app. See figure 4.2 for an example of the web console sending cloud messages.

The next issue was figuring out how to send notifications from inside the app opposed to through the web interface. It turns out you cannot do this directly, as the clients should not be able to make calls to Firebase as that would expose your API key. I was surprised there was no Google service existing specifically for this as it would be super important in any app requiring lots of notifications. Since I needed my own app server to send requests to the FCM (Firebase Cloud Messaging, formerly GCM for Google Cloud Messaging) I decided to use my AWS EC2 personal instance I already had running, and simply created and added a PHP script I could call from my app to send the request. Now from the app it's just a simple GET request to my certain URL with parameters to specify the target Firebase instance Id, message, and the Link Id that the notification is for.

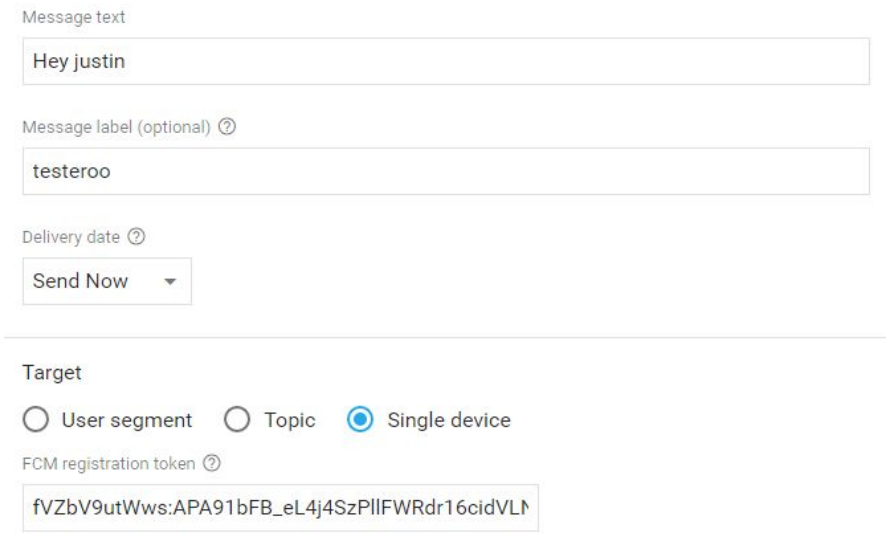


Figure 4.2 Firebase Cloud Console Notification Send

What's Next

If I decide to continue with this project, the next step would be preparing the application for the Google Play Store. This would involve optimization to ensure that the app is efficient for users, as well as testing on some sample users. One major problem I have is that before releasing the app I would want the app to be complete on iOS. A large portion of people have iPhones, so excluding them is not a great decision. If someone started to use my app, began to like it and then realized that their friend can't download it they would not be very happy. The problem with developing on iOS is that there is a developers fee of 100$, as well as the need for a Mac computer to develop on. Thanks to the unnecessary gateway from Apple, I am not very likely to proceed with developing it, leaving my app off of the store. I am happy to keep the app running as a smaller localized thing to share between my friends, and see how it goes from there. If the demand comes and we find the app useful enough then I may make the jump.

Conclusion

Overall I am very pleased with how this project went. I am very thankful for the opportunity to do this is a directed study, and greatly thank Yvonne Coady for being my supervisor on this. Without having that opportunity I wouldn't be able to justify the amount of time I put into this taking away from my other courses. I learned a wealth of information that will be very helpful when app development comes along my way in the future, and this is something that is very beneficial to have on a resume. It is unfortunate that not all students are able to learn this kind of material since it does not fit into the curriculum, but I am grateful for the chance to do so. I am happy with the state of my application and look forward to playing with it between my friends and carrying the knowledge forward in life.