Written By Kolby Fenster

With Jim Ward

Project created in partnership with Josh Sloan

Senior Honors Thesis

University of Wyoming

Abstract

As smart devices become more and more popular among the general population, their use as a tool of communication is no longer their primary functionality. These devices are being used and treated as miniature computers that provide access to entertainment, email, and many other services. We as a group decided to focus on the location-tracking aspect that many smart devices currently provide in hopes of addressing the problem of visiting a location with a group, splitting up, and being unable to know where other group members are. Our solution was to create a free and easy to use mobile application that allows users to share their current location in real-time and send/receive notifications to other members in their group, while not draining a device's battery life.

The application is currently designed to run on smart devices running an Android operating system and was built using Android Studio. While other applications currently exist that have similar purposes, our application, LokkaL, was designed to include functionality that is not commonly found in free applications or has not been widely implemented. These features include knowing other members' battery life, notifying members when another group member has left a marked area, and an easy-to-use interface that allows users to easily create groups with one another. No longer will one have to worry about being unable to locate a group member when traveling to a new city, a concert/music festival, or even just a night downtown.

Acknowledgments

This application was created for my Senior Design class to present on Undergraduate Research Day in a two-person team consisting of Josh Sloan and myself. Jim Ward advised this project. Without Josh and Jim, many parts of this project would have not been possible. While this paper is my own work, their contributions should not be ignored as several parts of this project were designed and created by Josh and Jim provided excellent guidance throughout the project. Thank you Josh for your work and assistance on this project and Jim for being an excellent adviser and one of my favorite teachers.

Table Of Contents

List Of Figures

Purpose Of LokkaL

Visiting an amusement park with family, attending a concert with some friends, or going out for a night on the town with some buddies are all going to probably involve a lot of people, a lot of noise, and the chance that as a group, you are going to get split up. Upon splitting up, contacting each other and finding each other can prove to be extremely difficult due to being unable to effectively communicate and locate one another. Additional difficulties can arise in the instance that a member's phone dies or an individual has no idea where they actually are. This is where LokkaL strives to provide a solution to these problems by being an easy to use, group location-sharing mobile application where we allow users to easily broadcast and share their location with one another.

Designing LokkaL

One of the most important parts in the development process of software is the planning stage. The planning stage allows the development team to establish a detailed description for each component of the software, map how different components will interact, and create a roadmap of the software from the initial phases of development to the final product. When designing LokkaL, Josh and I aimed to create a descriptive and detailed design document in order to experience these benefits later on in the development stage.

Logical View

The first piece of the design document was to discuss the core functionality of the application and outline the various parts of this functionality. In order to do this, a logical view was created in order to show the general communication structure between multiple instances of the application. The app would simply connect to a server containing a database, upload it's information,

download any new information from the database, and display this new information of the screen for it's user. This view offered an easy understanding of how the app would interact, but was too broad to fully understand the core functionality of the app. A UML diagram was needed to outline the overall software architecture.

Software Architecture

The diagram that ended up being created showcases how a user would navigate through the app. A user would open the app and be presented with a login/register screen. At this screen, the user would be able to register a new account if they did not have one, or log in with an existing account. For registering the account, basic information such as a person's name, email, and password would be need to saved and stored in the database. However, before this information could be saved, validation on this information would need to occur. Checks for a valid email address, password, and all necessary information being filled out would need to be implemented in order to ensure an account could be successfully created and saved in the database. In the instance of the validation failing, the user would need to be prompted with an error message highlighting what was preventing the user from successfully creating an account. If the validation passed, a user would be directed back to the login/register screen. This logic (as well as other parts) can be

Figure 1 UML Diagram of LokkaL Software Architecture

Source: Taken From Our Design Document

visualized more clearly by examining Figure 1, the UML diagram from our design document that outlines the logic the app is supposed to follow.

On the login section of the app, a user would be required to enter an email and password in order to confirm an account existed for the individual. A query executed on the database would check to see if any account currently existed where the email and password match. An incorrect email/password would cause the query to return zero results and would notify the user that an invalid email or password was entered. If the email and password matched an account in the database, information relevant to that user would be returned and the user would be directed to the group screen.

From the group screen, the user's available functionality expands. A user will be able to add, accept, decline, and delete friends; and have the ability to start their own group, invite people or join a group. In order to add friends, a search function is used to query the database for fellow users and then an invite is sent. For accepting and declining friends, a query will pull all active friend requests the user has. On these requests, the user will have the ability to either accept or decline the friend request. Upon selecting an option, a query that updates the database will change the friend request to being accepted or declined. If the request is accepted, the users should now appear in each other's friends list.

In regards to the group functionality, a query would first determine if the user is already in an existing group. If a user is not currently in a group, they will have the options to create or join a group. When creating a group, a user will set the group name and be able to invite users on their friends list. A query will send this information to the database and keep track of the invites sent to other users. After the group has been created, the group creator will have the option to disband the group. When

joining a group, a query will show all active group requests. The user will have the option to join or deny the request and an update query will notify the database of the option the user selected. After joining a group, the user will now see the location of all other group members on a map. A query will be constantly updating the database with the location of other users and another query will return this information back to each user. When a user leaves a group, this information will stop being processed and the user will now be able to join other groups. If a user is currently already in a group upon navigating to the group screen, they will have the same functionality as a group member.

Designing The Database

Once the core functionality had been mapped out, it was time to design the database. The challenge here was to decide on the information that would need to be collected from the user and how to store this information so it could be accessed in a meaningful way. We realized the easiest way to start this process was to define what we wanted to collect from a user when they registered an account. To register an account, a user would need to provide their first name, last name, a date of birth, an email, and a password. We made each one of these values a column in our "persons" table and added a unique person id field that would make it easy to distinguish individuals and accounts. The unique person id is also known as a primary key and simply means that no other row in the table can have the same person id. We then tied the person table to a friendship table that keeps track of the various friend requests and who is friends with whom. The columns for the friendship table were a unique

Table Name:	PersonModule_Friendship					
Column Name:	FriendshipID	LeftPersonID	RightPersonID	StartDate	EndDate	ResponseTypeID
Type:	int	int	int	DateTime	DateTime	int
Allow Nulls:					X	
Relationship:	Primary Key	Foreign Key	Foreign Key			

Figure 2 PersonModule_Friendship Table

Source: Taken From Our Design Document

friendship id (the primary key) to define unique requests, a left person id (the person who sent the request), a right person id (the person who received the request), a start date/time, an end date/time, and a response type id column as shown in Figure 2. The start and end date/times would keep track of when the request was sent and when the friendship ended (i.e. a person deletes an individual from their friends list). The response type id column would keep track of whether the friend request was pending, accepted, or denied. In regards to the left and right person ids, these columns would be populated using the unique ids found in the person table. This relationship is known as a foreign key relationship and means the data stored in these columns are primary keys for another table in the database. This also means that an account must first exist in the “accounts” table before its value can appear in either the left or right person id column.

Once these tables had been established, the next focus was on the tables pertaining to the group functionality. For a group, we decided the necessary information would be a group id (primary key), a group name, the group creator id (foreign key), a start date/time, and an end date/time. The group id will be used to identify groups, the group name serves as a simpler way to reference a group, the group creator id is the person id of the individual who created the group and the start/end date/times will keep track of when the group was officially created and closed. Then for the group member table, the necessary columns were determined to be a group member id (primary key), the group id (foreign key), a person id (foreign key), a start/end time, and a response type id column. The group member id would allow us to keep track of each group member, the group id tells us which group they belong to, the person id tells us who the person in the group is, the start/end times allow us to see when a group request was sent and when someone left the group, and the

accepted columns allows us to determine if the individual accepted or declined the group request.

Developing LokkaL

Starting With SQL Server

At the start of development, the plan was to use SQL Server 2017 to manage the database, with the server running on our laptops. This software was chosen because of the team's familiarity and pre-existing knowledge on how to use previous versions of SQL Server. Since purchasing a license would be too expensive for the team, we settled with using Microsoft SQL 2017 Express Edition. This is a free and simplified version of SQL Server 2017, but still allows databases to be created and accessed by applications. Once this was installed on each laptop, the database and tables were created.

The next step was to get the mobile application connected and communicating with the database. The first step in this process was downloading and installing a Java Database Connectivity (JDBC) driver to allow the communication between the server and app to occur. Once the JDBC driver was installed and added to the project, a connection class needed to be created. This connection class would be responsible for creating the connection between the server and the application. This connection class "would use the JDBC Driver and be passed a connection string, a string containing the location of the server, name of the database, login information, etc. necessary to connect to the server, and attempt to create a connection" (Milener and Guyer). Additionally, the connection needed to be implemented using an Asynchronous Task (Async Task). An Asynchronous Task is "a specific class that allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers"

("Android.os AsyncTask"). This is important because connecting to the server can be an action ran in the background of the app meaning the app can continue to run and operate while it attempts to connect to the server.

While this process seemed straightforward and Microsoft provided documentation with pseudocode on writing these classes, the development schedule ran into a major delay in attempting to connect the application to the server. The primary reason for the delay was attempting to debug the errors thrown by the mobile application when attempting to connect to the server. Since the mobile application is an outside source, the laptop with the server was going to have to allow SQL Server to allow outside connections. This meant disabling parts of the computer's firewall, opening specific ports to allow information requests to be made, and enabling various network protocols. It seemed after each change to the computer was made, another error would appear on why the connection attempt failed between the mobile application and the SQL Server. Eventually a point was reached where development could no longer continue until the connection issues were resolved. This lead to a discussion with Professor Ward where the decision to abandon using SQL Server was made.

Switching To REST Services

The plan was to switch to using REST Services to communicate information between the database and the mobile application. REST "stands for Representational State Transfer and uses the HTTP protocol to make calls between machines" which means a user only has to make HTTP requests to manipulate the data in the database (Ward). With the help of Professor Ward, the team created a web server that would now be used to communicate with the database. The database and web server would be hosted through the University of Wyoming's Computer Science

department. The one limitation of hosting it through the department was that users needed to be connected to the University of Wyoming's network in order to access the database through the mobile application. The database would be accessed using MariaDB outside of the mobile application and would be a MySQL database. Switching over to a new database environment required the table creation scripts to be recreated due to the slight syntactical differences between T-SQL and MySQL.

Once the tables were created, PHP scripts were built that would handle the various insert, update, delete, and query functions that the mobile application would need to function. No one on the team knew how to code using PHP so a few days were spent learning the basics of this language. These scripts would be passed parameters from the mobile application, which would indicate what information needed to be returned or altered. These requests were called using Async Tasks so that communication with the webserver could be handled in the background and any delays in communication would go unnoticed by the user.

Registration/Signing In

The first part of the mobile app to work was the registration of an account and the signing in part. When a user opens LokkaL on their phone, they will be presented with the login screen. A user has two options, sign in or register an account. Upon clicking the register account button, a dialog box will appear prompting the user to enter the necessary information to create an account. EditTexts (Android's version of a textbox) are used to obtain the first name, last name, email, and password from the user, while a date picker is used to obtain the user's date of birth. Validation was added onto this dialog box to ensure each section has been filled out correctly and that the password confirmation EditText matches the value in the password EditText. Once the validation

Figure 3 Login Screen

Source: Taken From LokkaL

tests are passed, an insert script is called to insert the information into the database. A user will be prompted with a message stating the account has been registered and can now sign in with said account.

In order for the user to sign in, they need to enter a valid email and password into the appropriate EditTexts on the login screen, as seen in Figure 3. Once a user has hit the sign in button, a query searches for any account with that email and password entered. If the query is successful and an individual is found, they are "signed in" and taken to the next screen where they can now add friends, create groups, etc. If no record is found, the user is informed that an invalid email/password was entered.

Managing Friends

After successfully signing in, a user now has the option to send friend requests and accept/delete friend requests from other users. Opening the navigation drawer from the main home screen allows the user to access these drawer options. To send a friend request, a user simply enters the email address of the individual they would like to send a friend request to. To accept or decline a friend request, a user swipes right on the friend request to accept or swipes left to decline. A user can view their friends through the Friends drawer option where they can invite friends to a group by swiping right on their friend's name and delete friends by swiping left.

Managing Groups

Additionally, a user will now have the option to manage groups after a successful sign in. This functionality is very similar to the friends and friend request functionality. By opening the navigation drawer, a user has the ability to create a group and invite friends to their group or they can manage their group requests by swiping right on a group request to join the group or swiping left to decline the group request. Once an individual has accepted a group request or started their own group, their location will be sent to the database and the app will now start populating the map with the location of every group member in their group. As seen in Figure 4, the map places a marker at each individual's location and by tapping on the marker; the name of the individual is displayed. One unique feature about using the Google Maps API to keep track of location is the ability to tap on a marker and send the coordinates to the Google Maps mobile application. By doing this, a user could get directions from their current location to the marker that they had tapped on, which would be extremely beneficial when attempting to navigate to a friend that you are unsure on how to reach. To see all members in a group, a user could go to the Group Management drawer and view a list of the group members. A group member's battery life would appear

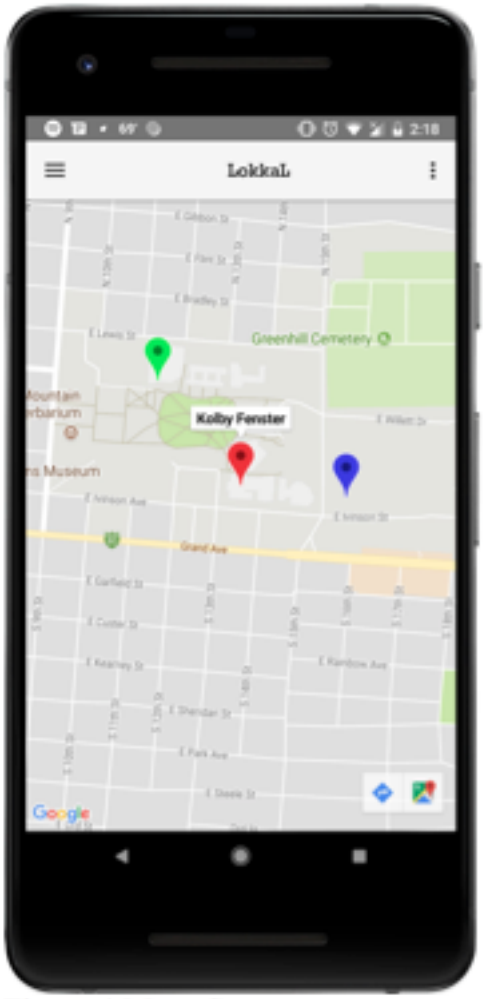

Figure 4 Maps Screen

Source: Taken From LokkaL

next to their name to inform other members if a phone is about to die. A member would also be able to leave a group from this screen.

Analysis of Project

In our original design of this project, we hoped to accomplish three goals in regards to the app and its functionality in this order of importance:

1. A mobile application that would allow a user to share their location with multiple individuals
2. Implement geofencing, messaging, and some other unique features to separate our app from existing apps in the market place
3. Develop the app for the iOS operating system and release the app to the Google Play and Apple App stores.

After the development of our application, we were only able to successfully complete one of our goals and partially complete another. We built a functioning mobile application where users could create account, add friends, create groups, and share their locations with one another and were able to implement the unique feature of being able to see another individual's battery life. The rest of our goals we were unable to fully complete or even attempt. Our finished project ended up being a proof-of-concept group location-sharing mobile application with little validation. However with the failures of not completing all of our goals, we have a variety of future work that is planned in furthering developing LokkaL.

Future Work

Our future work includes a variety of tasks that will take LokkaL from a proof-of-concept mobile application to an app that will be readily available on the Google Play and Android App stores. Our

first plan is to implement geofencing into our application. Geofencing is this idea where a user can define a specific point, create a perimeter around this point, and receive notifications when someone enters, exits, or is inside the perimeter. This would be extremely useful in our application as group members could easily be notified when a group member has strayed to far from a group or the group is becoming too separated. Additionally, we would like to increase the amount of account personalization that a user has by implementing profile pictures, recommended friends, etc. Profile pictures would be extremely useful as a current drawback with using the Google Maps API to track location is that information can only be displayed for one marker at a time. Profile pictures would allow us to replace these markers with actual images of the individual that way group members always have a visual representation of the individual they are seeing on the map. Both of these ideas should be possible and easy to implement if the backend infrastructure of the application and database can be moved to using Google Firebase. Google Firebase is a cloud based service for app developers that provides a variety of unique tools to aid the development of an app. Firebase can allow users of our app to easily send messages to one another and to multiple people at once, which could come in very handy when implementing the idea of geofencing. Additionally, Firebase supports sign-ins from Facebook, Google Plus, and a variety of other social media sites that could easily aid in the aspects of account personalization. For developers, we would have the ability to easily send messages to our entire user base, implement real-time databases, and have access to analytical tools to assist in understanding our user base, what our app is being used for, and what new features we should focus on implementing. Lastly, we hope to develop LokkaL for the iOS operating system to broaden our audience and allow almost anyone owning a mobile device to be able to use LokkaL. By

implementing the previously mentioned work, we easily see LokkaL going from a proof-of-concept mobile application to a fully functional mobile application that can be extremely successful.

Conclusion

The mobile app industry is continuously growing due to the functionality of smart devices being expanded from simply communication to ideas previously not thought of. Location tracking is apart of this functionality that can provide unique benefits to users, however; implementing location tracking requires a lot of work to be beneficial to users. It took Josh and myself nearly eight months to design, create, and implement a database and mobile application that took advantage of location tracking and sharing this information with a group. While it was a full project, us, as a team, fell short of completing all of our goals and understand why mobile applications can takes months or even years to develop with fully staffed teams. That being said, this project has taught both of us how to implement REST Services, design a mobile application and database that are easily scalable, and the effort required to see an idea transfer from a piece of paper to a fully functional mobile application. These skills will prove invaluable in our future careers as computer scientists and this project has allowed us to experience the highs and lows of designing and developing software. Regardless of the failures we experienced, this project has been a success in terms of completing our degree and gaining real-world experience as we prepare for the next step in our lives.

References

"Android.os.AsyncTask." *Android Developers*, developer.android.com/reference/android/os/AsyncTask.html.

Milener, Gene, and Craig Guyer. "Step 3: Proof of Concept Connecting to SQL Using Java." *Microsoft Docs*, Microsoft, 19 Jan. 2017, docs.microsoft.com/enus/sql/connect/jdbc/step-3-proof-of-concept-connecting-to-sql-using-java?view=ssdt-18vs2017.

Ward, Jim. "REST services" 13 November 2017, PowerPoint file.