Reach - A Community App

CPE 462: Senior Project II
Advisor: Dr. Philip Nico
Date: 06/05/2019
By: Samuel Magaña

Department of Computer Science and Software Engineering
California Polytechnic State University
San Luis Obispo, CA
# Table of Contents

- Introduction ................................. 1
- Background .................................. 2
- Requirements .............................. 3
- System Overview ........................... 4
- User Interface Components ............... 5-9
- Description .................................. 10-15
- Evaluation .................................. 16-18
- Future Work ............................... 19
- Conclusion ................................. 19
- Works Cited ............................... 20
INTRODUCTION

"The greatness of a community is most accurately measured by the compassionate actions of its members." – Coretta Scott King

Communities are familiar threads that unite people through several shared attributes and interests. These commonalities are the core elements that link and bond us together. Many of us are part of multiple communities, moving in and out of them depending on our needs. These common threads allow us to support and advocate for each other when facing a common threat or difficult situation. Healthy and vibrant communities are fundamental to the operation of our society. These interactions within our communities define the way we as individuals interact with each other, and society at large. Being part of a community helps us push past our differences bringing forward issues that really matter to us.

The motivation for this application comes from a desire to increase community involvement and activism. Engaging members of a community can be difficult because each one is so different and plagued by its own set of issues. I know this firsthand, having worked and volunteered in my hometown that many of these circumstances are difficult and complex. This application will not change the world, nor does it attempt too because the real work that needs to be done to heal and repair a community starts with direct intervention and involvement. I think any application in this realm that makes claims like these is entirely false and is completely out of touch with its users. The intention is to serve as a platform for those looking to become more involved and involve others in their community.

The mobile application will be android based. A user’s primary options is twofold: discover events around them and post new events for others to see. If the user is trying to discover new events around them, they will have the option of both a Map View and List View each providing a detailed information of these events. If the user is trying to let others know about things going on in their community the user will have the option to create an event. Events in this case are broken down into two groups: community involvement activities like seminars or volunteer opportunities and other events like robberies or car crashes that inform the community of situations occurring around them. These distinctions will be made throughout the application. The application will rely on crowd sourced information so the greater the involvement the better and more up to date the application will be.
BACKGROUND

Currently, there are similar services that exist either standalone or as features hidden within other applications. The most notable of these applications on the market are NextDoor, Eventbrite and Facebook Events within the Facebook application. However, all these applications other than Eventbrite, which is simply an event management service, operate as social media sites. The application I designed takes a utilitarian approach like that of Google Maps or Yelp. Instead of trying to get the user to stay on the application for as long as possible, mostly to increase ad revenue, it’s supposed to be informative, so a user takes a quick glance to see what’s happening around them. Push notifications with a quick summary are provided to the user to limit their time spent on the application. This all occurs in real time providing the user with the most current information.

To accomplish this task, the application is responsible for retrieving a variable number of events from a database of previously logged events. The application is inherently set up to be data-driven, therefore results returned upon initial usage of the application will not be as representative as those returned once more events are input and stored into the database.

The goal is to get people out of their homes not stuck on their phones. The average user has 60-90 applications installed on their phone of which take up 2h 15m of their day already [1]. A core concept in this application is to provide wellness by increasing community involvement and promoting digital wellness by creating an effective interface that take a minimal amount of time out of the user’s day, while giving them all the information they need to be an informed member of their community.
I. Functional Requirements
The functional requirements of the Reach Application include:
1) Inputting user information to login to the system, which includes a username and password
   a) If not already a registered user, the user can opt to sign up for an account
      i) To sign up for an account, the user must provide email, and a password.
2) Discovering new events around them based on the user's current location, order from most recent to least recent.
3) Creating a new event which is associated to the users account and stored in the application’s database
   a) To log an event depending on the type the user must include a brief title of the event, a description, up to three photos if desired, a start date and time, type of event and a location marker that denotes the address of the event.
   b) Certain events depending on the type will receive a popup notification prioritized by the android system.
4) Notifications of new events occurring around them
   a) When within a half mile radius by default the user receives a title and a shortened description of the event.
   b) Certain events depending on the type will receive a popup notification prioritized by the android system.

II. Non-Functional Requirements
The non-functional requirements of the Reach application include:

1) The application should support daily usage
2) The database should support large amounts of information
3) The application should be intuitive and easy to use.
4) The system should be a mobile application
SYSTEM OVERVIEW

As a high-level overview, the mobile application will be deployed on the Android platform and is hosted on Google’s Cloud services. The frontend component of the application is written using XML designed in accordance with Google’s Material Design layouts and concepts, for best and modern design practices [2]. The backend processes use a variety of Google Cloud platform services for database management, user authentication and mapping.
USER INTERFACE COMPONENTS

User Login / Sign up

Home Screen
List View for Events

Fri, Jun 07 at 02:11 PM
Fri, Jun 07 at 03:11 PM
Kennedy Library, San Luis Obispo, CA 93405, USA

List View for Alerts

Sun, Jun 02 at 12:00 PM
Recreation Center, San Luis Obispo, CA 93405, USA
Was walking to my car with my backpack in my hand. Someone came running and grab it from my hand. It looks like the one in the pic
More info about an Event  

Choose an item to Create/Report
DESCRIPTION

Frontend Setup

The front-end development of most android applications starts with XML based layout files that behave very similarly to HTML in web pages. These layouts allow for a distinction between the UI and the code that implements them. When initially creating the home page, I chose to use the Navigation Drawer activity. This activity along with a multitude of others are included by default in Android Studio. They are a great way to deploy large chunks of functionality very quickly without having to worry too much about the initial UI setup. Prior to starting this project, I had a beginner to amateur familiarity with android development, having only created one small application in a summer internship my first year, so using this default layout with the skeleton backend code to implement these UI elements created a solid foundation for development.

The advantage of starting from a default activity is that they contain the latest Google design guidelines that implement UI/UX elements which are modern and run much more efficiently on the latest versions of Android. They also include the best coding practices for backend development. I used my Samsung Galaxy S8 to develop this application, which is running Android Pie API 28, the latest version of android. Choosing early on which version of Android you are developing for is important in determining what libraries you can use throughout your implementation. It’s no secret that the Android OS is extremely fragmented with only 10 percent of phones supporting the latest version of android that was released almost a year ago [3]. With this in mind, I chose to develop for android API 26 and up which means my application is only compatible with 21.6% of android phones on the market right now [4]. In the end, I felt that having an application that looks modern and provides an enjoyable user experience was a higher priority than one that was compatible with more devices. I believed I could take advantage of currently UI/UX elements and practices to make my application more effective which could lead to an even greater adoption.

Layout and Design Elements

The android support libraries offer different types of UI layouts depending on the requirements of the design. I found that using a constraint layout was the easiest way of building forms like those needed to capture or display user generated input. This is because different UI elements can be applied to the layout before having to define their constraints to the container and other UI elements. For UI elements like the map fragments, using a relative layout was the best method for ensuring a layout built upon elements of other layouts stayed relative to their own. Each XML layout file can be
Implementation of floating action buttons was a key design element I wanted to include in the development of my application. This design element provides for a better user experience and simpler UI. In much of Google’s design guidelines less is more, users prefer a relatively simple yet expansive interface. Avoiding the tedious work of having to implement these buttons and animations myself, I found the Floating Action Button Speed Dial on GitHub to be exactly what I was looking for [5]. This library was very plug and play with the best documentation and ability to customize actions. This library works by building XML layout files that are relative to the constraints the element is placed in. Unlike some other libraries, only a single element needs to be added to the layout, additional actions must be added programmatically. This meant that most of the work had to be done in the backend.

The process of getting these different UI elements to work together to create the desired user experience was probably the most time-consuming aspect of this project. I found it easier to storyboard an entire user workflow like creating a new event was the most effective way. It also saved a lot of time from having to go back and rework these elements later. Android studio has great capabilities in helping you debug these XML files but, the documentation from Google on how to choose the correct layouts for your specific needs was lacking and many tutorials simply do not explain.

**Backend Setup**

The backend implementation of my application was all done in Java. While Google officially endorses Kotlin as their programming language of choice for Android development, I found existing tutorials and support libraries to be scarce and lacking. Getting bogged down in learning new syntax is not where I wanted to allocate my time and resources. Because Java is an object-oriented programming language, layouts can be built and reused for single or multiple classes, so time was saved from having to replicate code throughout the project. Android develops the backend implementation of a layout through the instance of ‘Activities’ which are just classes that contain methods specific to android functionality like a binding that builds the object in a specific view. Activities can call other activities and pass information to them using an intent. Using a series of intents is how the application progresses normally through a typical user workflow depending on the user input. This is how a user can progress through the workflow for creating an activity with each intent passing information to the next activity.
The way that android differentiates classes from activities is through the android manifest. The android manifest is another XML used to declare which classes are activities and permissions needed from the device for different services like location data or internet access.

Fragments

Another large portion of the backend implementation was the use of fragment classes. These activities are self-contained and often included as an element of an overarching activity. Deciding to use a fragment class in my opinion was a good choice because these self-contained activities can be used without having to worry about its relation to the main activity. This level of abstraction like that of an API, allows for easier implementation of these features. In my case, I used a fragment class for creating the map activity that allows the user to choose a location on the map by tapping it and another fragment which grabs user input from the input fields. Fragments were also used in creating the map and list view. Interfaces are often required to communicate between fragments and the main activity. This is particularly useful because these interfaces allow for the functions passing data to be redefined for the specific activity implementing them.

Each fragment in the activity is managed by the fragment manager. The fragment manager allows for a fragment to be created and its current state to be saved when switching between them. I used this to my advantage when implementing the map and list view, where I was able to save the current state of each instance. This made switching between each view faster because the activity does not need to recreate the fragment each time.

Google Firestore Database

Google firestore is a NoSQL cloud-based document database. Document Oriented databases are inherently a subclass of key-store databases and follow concepts of modern object-oriented programming. Using firestore provided me with many advantages from real time syncing, database user authentication read writes, to the ability of creating convenient and dynamic database schemas. The document-based schema was particularly useful for containing all information for a post in a single document. This saved me the time of having to create a traditional relational database schema with multiple tables, while still providing me with the ability to create traditional SQL queries. Below is what the schema for a user generated event document looks in firestore.
Integrating firestore was quite simple because the API provides functionality to map an object to a document by dynamically creating the fields from the object in the database. This also means that when querying for a document the fields can be mapped back to the object itself which was very helpful for passing information between functions and activities. Another huge advantage of using firebase was the ability to add a snapshot listener which caches a current snapshot of the collection and checks for any changes when its updated. The granularity of knowing if a document was added, deleted or updated allowed real time updates to the UI elements like the maps and push notifications when a new event was added.

There were a few drawbacks of using firestore that required careful consideration during implementation. The biggest issue being that the Firestore API is fully asynchronous, calls to the database like a new query or adding a document return immediately and a
Task object is created to execute the request. This was an issue when activities require data from these calls to complete their layouts causing null pointer errors. One workaround I found was to create an async task which executes the given query or upload on another thread then returns to a completion method before returning to the main activity thread that called it. This worked for some of the activities that did not require the data to be returned immediately like adding a marker to the main activity map. In the instances where the data needed to be available before an activity could continue, I used a semaphore in the async activity to block the thread where the async task was called from forcing the async task to finish before returning to the main thread.

**Google Cloud Storage**

Uploading images related to user events was done through Google’s cloud storage. When an image is successfully uploaded by an upload task a download URL is returned. This URL is stored in the document along with other information about the event. Downloading these images was done using Picasso. Picasso is an android support library used for downloading and caching images [6]. Using Picasso was convenient in that it deals with all networking issues related to downloading and caches images allowing them to load much quicker. Another advantage of using Picasso was that once an image was downloaded it return it as an image view object which meant I could pass it directly to a view pager object. While this is still possible regardless of the support library having the functionality made the implementation much simpler.

**View Pager, Recycle View**

View Pager is a layout manager used to flip through pages of data like a slide show [7]. They contain arrays of other types of view objects. I found them to be a perfect solution for slide showing one or multiple images depending on how many images were uploaded. Using the view pager object in conjunction with Picasso required an adapter, which extends the view pager class itself to support Picasso. Using the adapter, I can pass a string array of download URLs and let the adapter instantiate each item in the view pager once it finishes downloading. The view pager is the container used for displaying all images, so the adapter was used for just about every activity. Also, using the view pager provided another important element that added to the user experience that I was trying to create.

The Recycler View was the layout manager that I used to implement the list view option. This part of the implementation took the longest because it used multiple elements in the project from the view pager implementation, async task functions to query data, to designing the layout for each UI component. Despite some of the challenges, I think this feature greatly enhances the user experience providing different representations and
views of the data. Like the view pager, the recycler view requires an adapter to map items in the view to the UI components. This is because each item in the list is its own object, so the recycler view instantiates the adapter for each item.

**Google Firestore Authentication**

The user sign-up and sign-in process is derived from a default login activity included in android studio. The reason I decided to use this over designing my own login process was because it uses Google’s Firestore authentication process. This authentication method works in conjunction with the read write rules of the database so only users signed in can create and view events. The image above shows what the existing user table looks like along with the unique user id which is used to identify the user creating a new event. This method also works well to identify new users, returning a flag if the email does not exist in the table.

**Google Maps, Google Places API**

In the home page of the application and when creating a new events maps are used to display and set locations. Along with the markers on the maps denoting different locations, selecting any of the bottom action bar buttons filters these markers depending on the activity. There is also a search bar at the top that allows you to set the map focus to different locations. To produce the maps and search for locations, Google Maps and Places API was used. The Google Maps API provided the functionality to set locations on the maps as well as customize markers. The Google Places API provided functionality that worked to move user around the map more efficiently by allowing them to search for a place directing the map focus around it. Also, using Google Maps when creating a new event, the user is provided the closest address to the marker allowing them to fine tune the exact location of the event.
EVALUATION

In order to assess the project implementation, I will go through the requirements I had set for this project and its features. Whether my implementation was a success or not, I will further discuss and elaborate on the steps I took to arrive at this current state of my project.

1) Inputting user information to login to the system, which includes an email and password
   a) If not already a registered user, the user can opt to sign up for an account
      i) To sign up for an account, the user must provide email, and a password.

One of the first requirements I had set out to complete once I finished developing most of the core features of the application was a login and sign-up page. Most of my time was spent trying to integrate different APIs and working in the android environment to create the features I thought would amount to a minimally viable product. Overall, this functionality is working, and a user must sign-in back in when exiting the application. Also, a user sign-in is directly tied to the ability to access the data in the database by the user id.

I would say this requirement was implemented but not to the extent I had envisioned. When a user is signing up, I wanted to create a process where basic information about the user could be collected and preferences like a minimum distance for receiving notifications about new events could be set. Currently, the implementation has default preferences like a minimum notification distance of half a mile and no information about a user is collected or used other than the user ID when creating new events. I think allowing the user to have these options would create a much more personalized experience which is something I believe is important.

2) Discovering new events around them based on the user's current location, order from most recent to least recent.

The application allows the user to receive push notifications for events within half a mile of their current location and whenever the application is opened a query is performed using the current date and time removing old events. Providing the user with both a map and list view of events within their immediate location allows them to discover and sort through different types of events. Not to mention that the user can also click either the marker on the map or the description of the list view to see the full details of the event.
This level of customization allows the user to get more information on events that interest them.

I think this feature and the subset of others that fall within this requirement were adequately implemented in the application. The user does have some ability to filter through different types of events, which I think provides an efficient and pleasant way of sorting through all this data, increasing the likelihood that a user would continue to use the application. I also think that the overall design of the application provides a modern and efficient user experience, an underlying element of this requirement.

3) Creating a new event which is associated to the users account and stored in the application’s database
   a) To log an event depending on the type the user must include a brief title of the event, a description, up to three photos if desired, a start date and time, type of event and a location marker that denotes the address of the event.

The user can provide as many details as they choose in order to find or avoid the event being described. Having the view pager so the user can easily switch between setting the location and providing details of the event, another UI element that I think improves the efficiency and experience of the user. Focusing on a user workflow that quickly allows them to create an event was the inspiration for the design. I would find it very inefficient if the user had to return anywhere in this workflow to change something and previously entered information was lost in the process. Also, including the address bar at the bottom on the map location input removed some of the ambiguity of just tapping the map and assuming it’s on the correct address.

The implementation of this requirement was met in all aspects and provides the user with many options to report information about an event quickly and easily. The only improvement that I could see with this process is including a Google Places search bar at the top of the map so the user can more easily navigate to different locations that are not in their immediate location. Currently, the process is a bit cumbersome because the user must manually guide themselves to the area otherwise.

4) Notifications of new events occurring around them
   a) When within a half mile radius by default the user receives a title and a shortened description of the event.
   b) Certain events depending on the type will receive a popup notification prioritized by the android system.
The last requirement and probably the most useful is the ability to receive push notifications for all users once a new event has been created. This functionality is implemented using the Google firestore snapshot listener which queries the dataset and caches it is checking for any updates to the current version. The user can receive push notifications whenever an event within a half mile of them is added but, no priority to certain events is given. Each notification provides the title and description of the event allowing them to take a quick glance. Overall, the implementation met the basic requirements and works well to notify users despite the current issues in play.
FUTURE WORK

There are many features I had thought up during the process of implementation that I had hoped to eventually add to my application. I think this is where better planning in the form of more storyboarding would have come in handy. But I also know given the time constraints and the entire scope of the project that only certain features could be implemented. One of the biggest aspects that I think could use more refinement is the UI in general. I also wanted to implement a polling feature in the application which I had originally thought of in my initial implementation but, given the constraints was not something I could implement to the extent I envisioned. This polling feature would use geofences to allow users in a certain area to take a poll. I found that while doing this was possible, unless some backend algorithm was created to analyze this data it seemed pointless.

While it was difficult to pick up speed on development early on, substantial progress was eventually made. Developing this application in a team, I believe, will produce the best result. It would also help squash some of the bug I found after the implementation. Overall, new ideas and layouts from different perspective would greatly benefit the application’s development.

CONCLUSION

The challenges faced in designing and implementing this application taught me a lot about android development. Coming up with innovative solutions to these challenges whether that was using a semaphore or binging back some good old recursion, I realized how adept I have become at problem solving. It was genuinely a joy coming up with solutions to these different challenges. More than that coming in with only limited experience and developing what I believe is almost a full-scale application was an accomplishment on its own that I am proud of.
WORKS CITED


