Cru Android App

by
Yuanxiang Dylan Sun

Computer Science Department
College of Engineering
California Polytechnic State University
March, 2018

Abstract

The Cru app is focused on connecting students and faculty to one of the many Cru ministries here in San Luis Obispo. The app is capable of connecting individuals to a specific community group and ministry teams, gathering information about upcoming Cru events and gatherings, and direct rideshare capabilities. The Cru Android App utilizes Java and other industry standard technologies to fully deliver a well rounded user experience and to be connected with Cru.

1. Introduction

Cru Central Coast is an organization focused on giving students and faculty in both Cal Poly and surrounding community colleges the opportunity to grow spiritually. Cru Central Coast partnered up with Dr. Janzen's 2015-16 Capstone class to develop a mobile application that provides a platform for students and faculty to connect with their respective Cru organizations on campus. The iOS application that came from that Capstone class has recently made an alpha release on the App Store. Since the release, the iOS application actually helped a few students to be connected and eventually lead them to the Lord. The Android application, on the other hand, has not yet been released to the Google Play Store.

After speaking with Kyle Fletcher, the Cru staff and customer that led the project and worked with Dr. Janzen's class, I discovered that the primary reason the Android application was not released was because there is a lack of synchronization between Android and iOS application features. After acknowledging the issue, I wanted to help contribute to this vast codebase to further closing the Android/iOS gap by developing new features for the Android application.

1.1 The Problem

With the application demand and increasing users in mind, it is necessary to have a well-designed access control for users with different permissions within the app.

The leadership hierarchy is designed to enhance the ministry experience and the Android application must also reflect that as well. Furthermore, OWASP Top 10, a security organization that produces industry standard for online security protocols, announced that access control is number five of the top ten security holes that exist in our online culture today.

The current implementation of the Android app lacks the capability to grant the specific user with additional actions to enhance the flow of the application. The current implementation forces a user, Kyle Fletcher, to update information and create users by hand, directly writing to the backend database. This process decreases productivity and increases the chances of mistakes. In the light of releasing this app on a big scale, the slightest misalignment can cause a big drop in productivity.

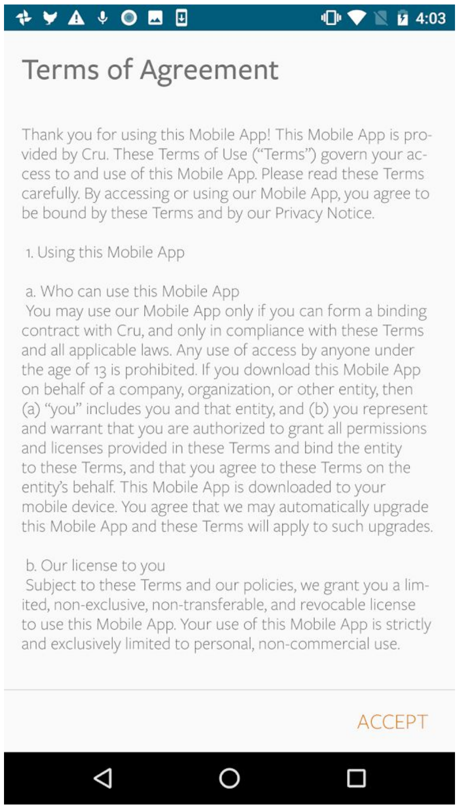
1.2 The Solution

The implementation of new features both provides a way for Kyle Fletcher and other Cru staffs to manage and create new users. On top of that, it provides a way for leaders within the respective ministries to edit their perspective groups. These features provided more access control, thus increased the Android application security. In addition to increases in security, Cru staffs now are able to spend less time on application organization and spend more time on leading students and faculty in specific ministries.

2. Application Overview

2.1 Term of Agreement Page

Due to legal purposes, Cru now demands students and faculties who uses this application must agree to a legal document prior to using this application. Figure 1 showcases what the user sees during the first download of the application. Utilizing Butterknife, I was able to bind textboxes as buttons to create an industry standard UI/UX design. This page will only be displayed when the app is first being used.



4:03

Terms of Agreement

Thank you for using this Mobile App! This Mobile App is provided by Cru. These Terms of Use ("Terms") govern your access to and use of this Mobile App. Please read these Terms carefully. By accessing or using our Mobile App, you agree to be bound by these Terms and by our Privacy Notice.

1. Using this Mobile App

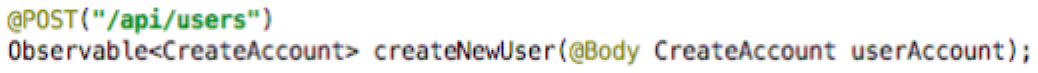
a. Who can use this Mobile App
You may use our Mobile App only if you can form a binding contract with Cru, and only in compliance with these Terms and all applicable laws. Any use of access by anyone under the age of 13 is prohibited. If you download this Mobile App on behalf of a company, organization, or other entity, then (a) "you" includes you and that entity, and (b) you represent and warrant that you are authorized to grant all permissions and licenses provided in these Terms and bind the entity to these Terms, and that you agree to these Terms on the entity's behalf. This Mobile App is downloaded to your mobile device. You agree that we may automatically upgrade this Mobile App and these Terms will apply to such upgrades.

b. Our license to you
Subject to these Terms and our policies, we grant you a limited, non-exclusive, non-transferable, and revocable license to use this Mobile App. Your use of this Mobile App is strictly and exclusively limited to personal, non-commercial use.

Figure 1. Terms of Agreement

2.2 Create Account Page

As described before, the create account page is for new users to create an account within the Cru application so that they are able to seamlessly connect with their ministry of choice. Coming into this part of the feature, I did not have experience in reaching a REST API endpoint in Android. This has been a learning opportunity to use popular libraries such as RxJava and Retrofit to send a proper JSON object that is capable of reaching an endpoint. In this case, it is a POST request on the user's endpoint (as seen in figure 2). This took me a bit of time to understand because the complete process requires the application to reach to a “Provider” object that directs the connection. Furthermore, I had to create a special user data model (described in depth on page and in the figure) called “Create Account” to send an appropriate JSON format that the endpoint accepts



```
@POST("/api/users")
Observable<CreateAccount> createNewUser(@Body CreateAccount userAccount);
```

Figure 2. New User Endpoint

2.2a Navigating to Create Account

Coming into the home screen of the app, users can either press the navigation menu item in the Toolbar or slide their finger from left to right to pull out the navigation menu. The user then must select “Settings” to navigate to the settings menu. Within the settings menu, the user can see the “Create Account” option. By clicking on the option, the user is able to reach to the Android activity that allows users to enter personal information to create an account.

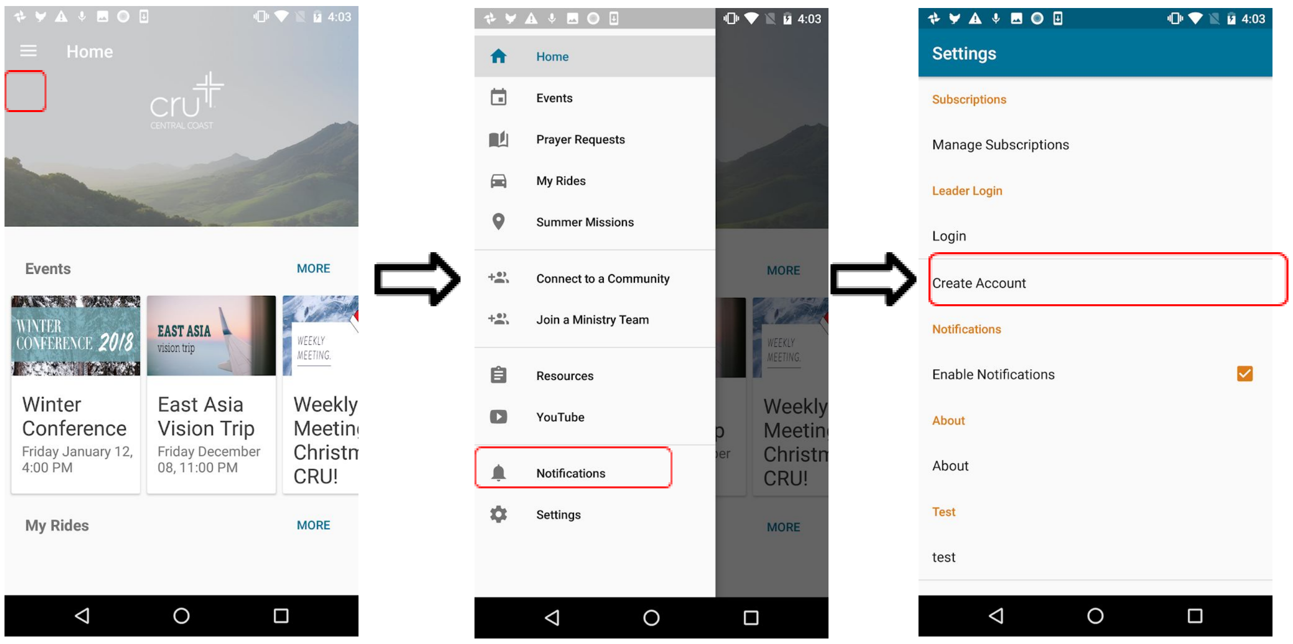


Figure 3. Create Account Navigation

2.2b Create Account Screen

When the user has reached to the Create Account screen, the application then prompts the user to enter their basic information such as phone number, email account, and password (Figure 4). By entering the correct and/or intended information, the user can

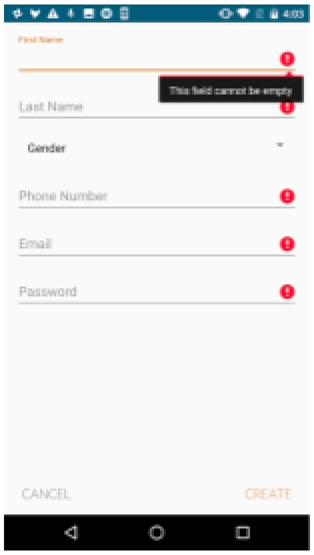


Figure 4. Create Account Page

Even though the input fields look similar, I was able to utilize pre-built Android input fields for the specific type of tasks. For example, while entering a phone number, the keyboard transformed into a regular phone pad and email field contains an “@” symbol. I personally believe that the smallest details do make or break the user experience within an app. Thus, the slightest improvement to allow user’s ease of access can maximize productivity. In figure 5 showcases different input fields.

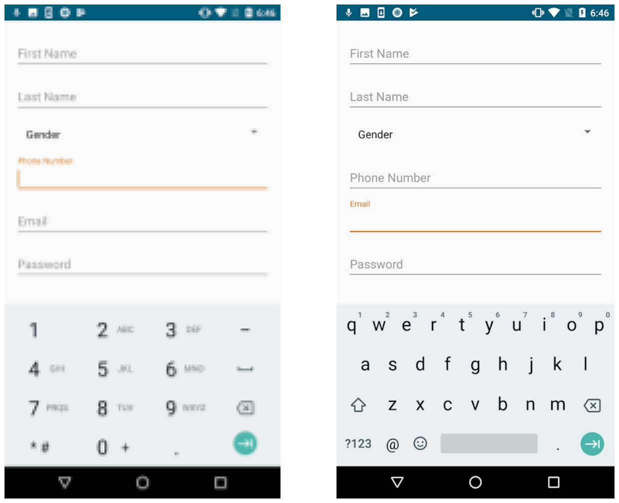
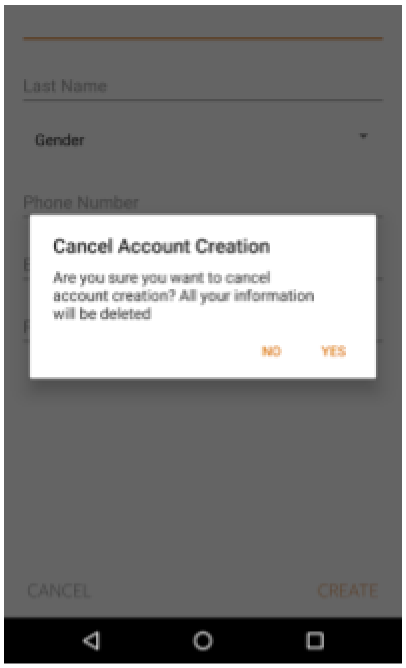


Figure 5. Different Input Types

2.2c Cancel Create Account

During the account creation process, the user is able to cancel the process at any time. When clicked on “Cancel” button, the user is prompted with a warning to make sure that he/she is meant to do what is about to happen. It was a difficult process in deciding what the two answer button should say. The common issue with these type of question is that the answer provided might be interpreted as the negate of the said answer. Instead of using ambiguous answers like “Confirm” or “Sure”, I decided that the simplest “Yes” and “No” gets straight to the point (as shown in figure 7).


Cancel Account Creation

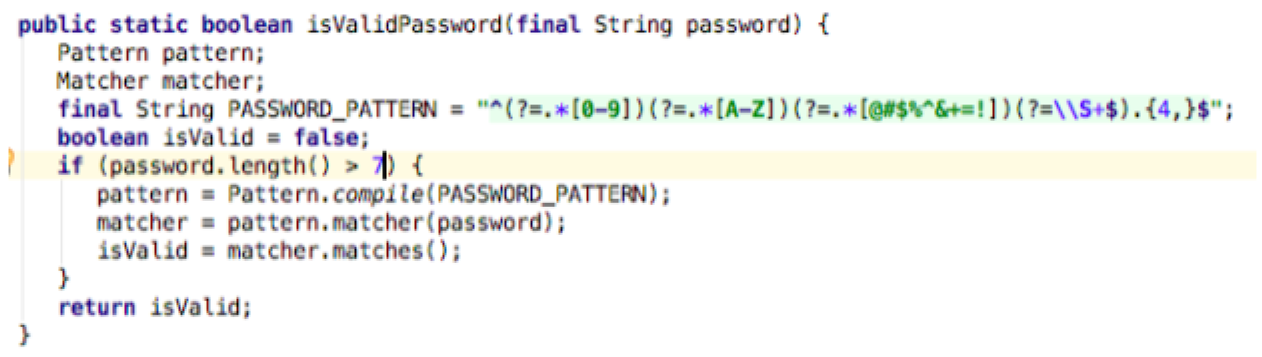
Are you sure you want to cancel account creation? All your information will be deleted

Figure 6. Cancel Account Creation

After the user presses “Yes”, the application will lead the user back to the settings screen. From there, the user is able to navigate to another area of the application. The app does not store any of the inputted information when the account creation process is canceled.

2.2d Misc in Create Account

Aligning the principles of enhancing the application security, the user passwords have to have some type of prefilter to check whether the user has satisfied certain password criteria. The common industry standard for a satisfactory password must be at least 8 characters long, includes a capital letter, lower case letter, and special letter. In figure 7, it showcased the specialty check that I've created. By using the Pattern and Matcher class, I was able to implement a string checking process that verifies the user input satisfies the said password requirement.



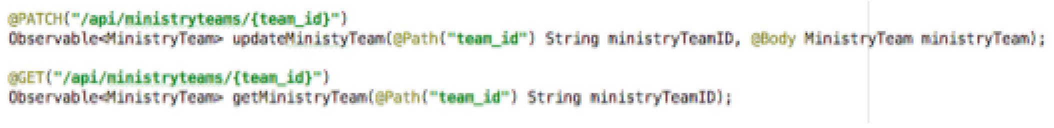
```java
public static boolean isValidPassword(final String password) {
    Pattern pattern;
    Matcher matcher;
    final String PASSWORD_PATTERN = "^(?=.*[0-9])(?=.*[A-Z])(?=.*[@#$%^&+=!])(?=\\S+$).{4,}$";
    boolean isValid = false;
    if (password.length() > 7) {
        pattern = Pattern.compile(PASSWORD_PATTERN);
        matcher = pattern.matcher(password);
        isValid = matcher.matches();
    }
    return isValid;
}
```

Figure 7. Password Verification Code.

In addition to password checks, I was able to implement a check for appropriate number counts in phone number input and email checks for valid domains. With these verification processes, the app now is capable of creating a user that is or close to the provided personal information. This mitigates Cru Staffs, like Kyle Fletcher, from updating a user by hand at a later time.

2.3 Edit Ministry Team

Ministry team is also a crucial aspect of Cru itself. Students and faculty are capable to serve the ministry through different ministry teams. In a sense, this is where the students and faculties find out opportunities to give back to the community that has invested in their spiritual growth. From what I’ve learned in Create Account page, I am now capable of hitting the REST API endpoints more smoothly since I know the general concept how the connections should be carried out using Retrofit. The two endpoints I must hit to successfully retrieve and update a ministry team is shown in figure 8. The update ministry team option will be evolved to allow leaders be able to update their respective images to showcase their team. Even though the backend has not yet developed to handle such process, it was a great learning opportunity to both retrieve and post photos from gallery and camera.



```
@PATCH("/api/ministryteams/{team_id}")
Observable<MinistryTeam> updateMinistyTeam(@Path("team_id") String ministryTeamID, @Body MinistryTeam ministryTeam);

@GET("/api/ministryteams/{team_id}")
Observable<MinistryTeam> getMinistryTeam(@Path("team_id") String ministryTeamID);
```

Figure 8. Ministry Team Endpoint

2.3a Navigating to Update Ministry Teams

Navigation is similar to navigating to the Create Account Page. The user first clicks on the menu icon in the toolbar on the top right of the home screen. A navigation menu will pop out and the user must click on “Join a Ministry Team.” By clicking that option, the application will lead the user to a list of available ministry teams that is hosted within Cru Central Coast. Figure 9 showcases the entire process of reaching to a ministry team.

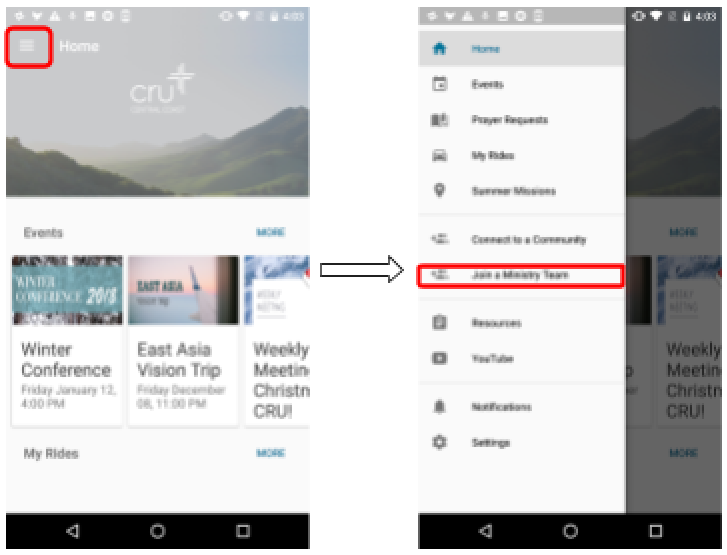
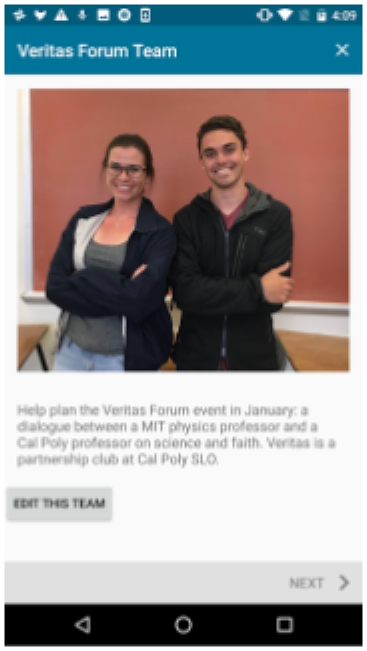


Figure 9. Navigate to Ministry Team

After the user sees a scrollable list of possible ministry teams, he/she is able to select the desired team to join. If he/she is a leader of that team and is logged into their Cru account within the application, the app will automatically recognize that and populate an “Edit this Team” button to allow the user to edit the current team they are on. This feature is only limited to a specific leader that leads the team. In each ministry team object, it contains a list of leaders with a unique ID number only to them. When logged in, each user also acquires a unique ID number. When the logged in user, the current user, has the same ID number as the leader, the button will automatically appear to them. The example is shown in figure 10.

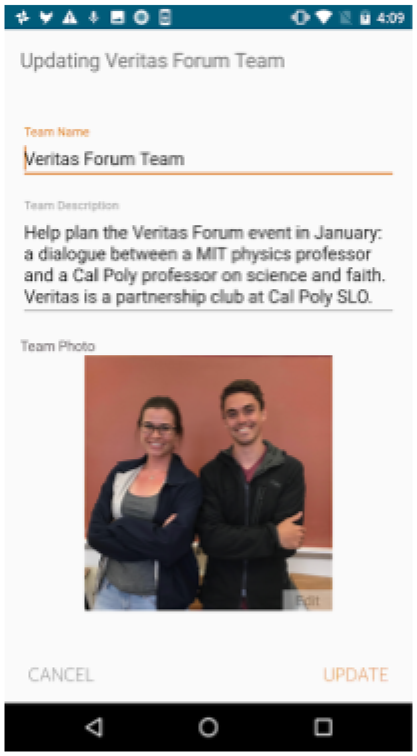


Help plan the Veritas Forum event in January: a dialogue between a MIT physics professor and a Cal Poly professor on science and faith. Veritas is a partnership club at Cal Poly SLO.

Figure 10. Edit This Team Button

2.3b Editing Ministry Team

When the leader presses “Edit This Team”, the app takes the user to update the desired ministry team (Figure 11). In the light of previous examples, the Veritas Forum Team will be our example for this paper. While it is similar input fields as Create Account, I was able to incorporate the ability add a photo from both gallery and utilize the camera to capture the photo to be uploaded (figure 12).



Updating Veritas Forum Team

Team Name: Veritas Forum Team

Team Description: Help plan the Veritas Forum event in January: a dialogue between a MIT physics professor and a Cal Poly professor on science and faith. Veritas is a partnership club at Cal Poly SLO.

Figure 11. Edit Ministry Team

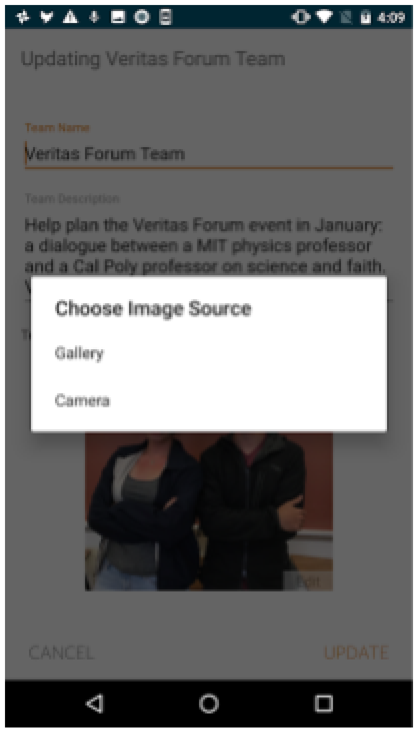
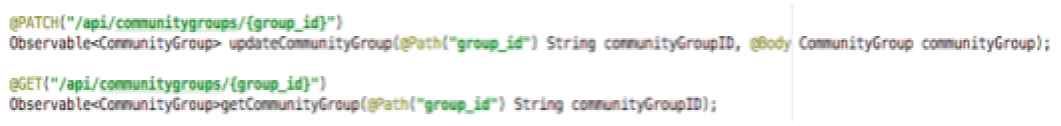


Figure 12. Image Source

The storing and retrieving images from gallery and camera has been an interesting learning opportunity because of its complexity due to memory allocation and data parsing between sessions.

2.4 Edit Community Group

Just like ministry team, community group is also an important way for students and faculty to be connected within a community.



```
@PATCH("/api/communitygroups/{group_id}")
Observable<CommunityGroup> updateCommunityGroup(@Path("group_id") String communityGroupID, @Body CommunityGroup communityGroup);

@GET("/api/communitygroups/{group_id}")
Observable<CommunityGroup>getCommunityGroup(@Path("group_id") String communityGroupID);
```

Figure 13. Community Group Endpoint

2.4a Navigating to Community Group

Similar to navigating to ministry teams, community groups can be found by going into the navigation menu and into the "Connected to a Community" section of the app. As shown in figure 14.

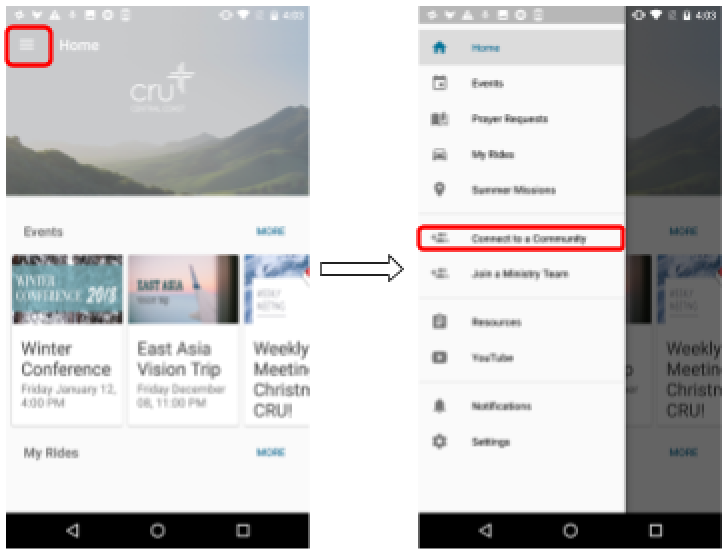


Figure 14. Navigate to Community Group

Utilizing same concept, I was able to dynamically create a button for leaders to click on when they've selected their own community group. As shown in figure 15, leaders must be logged in to be granted the rights to update their community group details.

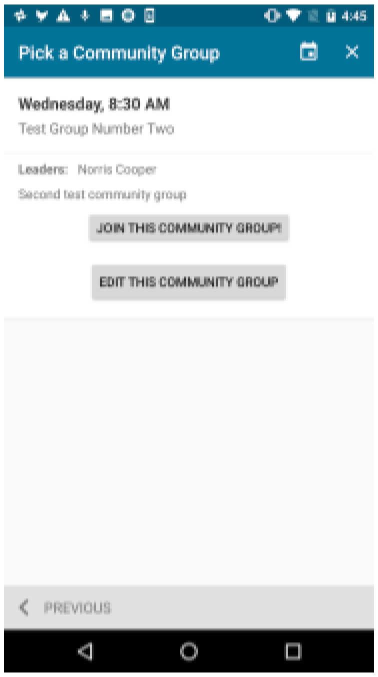
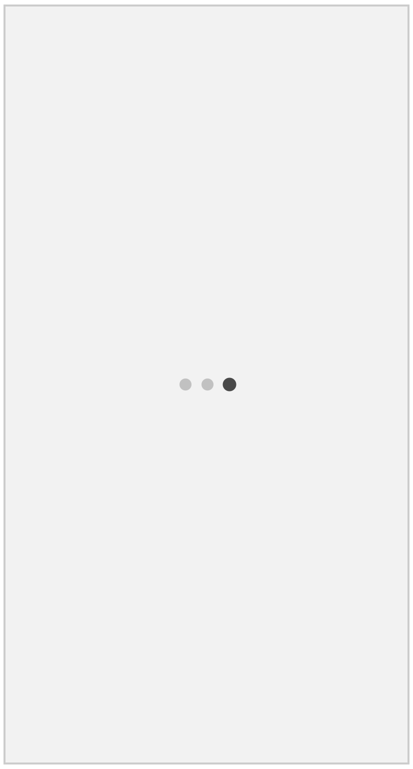


Figure 15. Edit Community Group

2.4b Editing Community Group

Within the edit community group fragment, the leader has the ability to update the group name, description, group type for student's year rank, and a specific meeting day. Cru promotes a consistent meeting time so it will not interfere with student's schedules. As we all know, sometimes things come up in life and we either have to cancel or reschedule community group meeting times. The updating community gives that option for leaders to reschedule community group times and it will send out notifications for students who have joined that specific community group. Figure 15 showcases this option.



3. Architecture and Tech Deck Highlights

3.1 Android

This project is mainly coded in Java with mix of xml and utilizing Retrofit to send and receive JSON objects to the backend. I have designed the project to where any editable information is capable of utilizing my "Update Information" activity. This activity then is capable of spawning different types of fragments depending on where in the app is requesting a update information fragment to be spawned. I believe this design can simplify the app's burden and align the application towards industry standard.

3.2 Architecture to Retrieve Data (Retrofit)

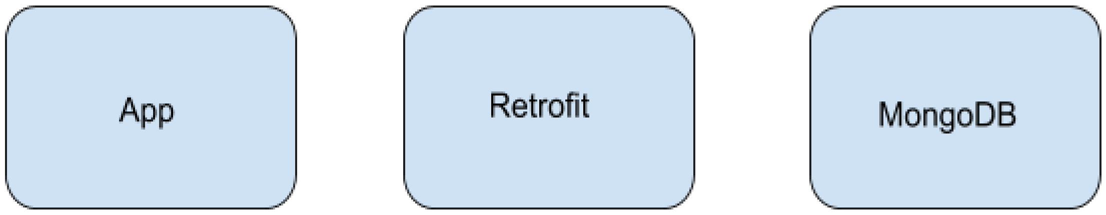


Figure 16. Architecture

As mentioned before, this app utilizes Retrofit's type-safe HTTP client to reach to the Cru backend. Retrofit is an open source HTTP client for Android developed by Square. Retrofit allows developers to consume JSON data easily by parsing it into POJOs while all REST API requests can be executed. Utilizing this technology, I was able to create POJOs and using Retrofit to send out specific requests to the Cru backend so that information can be updated. Figure 16 describes the general flow of data, whether it's a GET request or a POST request, all data flows similarly through the architecture.

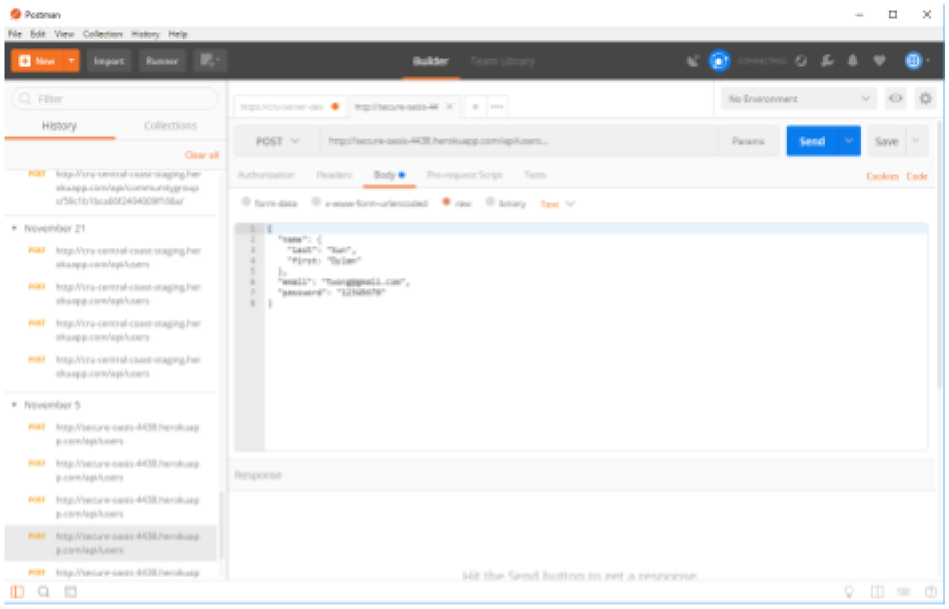
In addition to this powerful tool, by utilizing different paths in the API, the application is capable of retrieving more detail specific data depending on what the app provides.

3.3 Butterknife

Butterknife is another powerful tool to be used in Android development to inject views into Android components. Utilizing annotation process, Butterknife bindings are capable of task like “@OnClick(R.id.anID) to add an onClickListener to a specific view. This is my first time using this awesome library and it has definitely made my code a lot cleaner and clearer. Just by using “@OnClick”, I was able to break down different buttons in different methods instead of having everything grouped together in a general Android method.

3.4 Postman

Postman is HTTP client for testing web services. Utilizing this technology, I was able to successfully testing what data within the JSON file are needed when I try to hit an endpoint. In figure 17, I was trying to run a POST request to create a new account. At time it was adjusting little things to test whether the JSON object was accepted properly before I coded it out within the app. This is crucial for development because this process saves a lot of time for developer to test out the actual endpoint.



POST http://secure-oasis-4438.herokuapp.com/api/users...

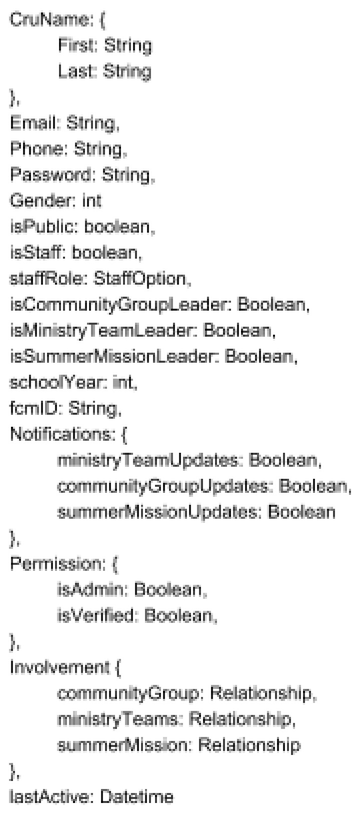
```
{
  "name": {
    "last": "Sun",
    "first": "Dylan"
  },
  "email": "[illegible]",
  "password": "12345678"
}
```

Figure 17. Postman

3.5 Data Models

3.5a User Model

The user model, seen in figure 19, is what a Cru User looks like. It contains a CruName, email as string, gender as int, password as int. More importantly, it contains the crucial boolean values that will determine the relationship between the users and their respective leadership roles. Furthermore, it provides different clearances depending on what rank they are within the Cru staff hierarchy.

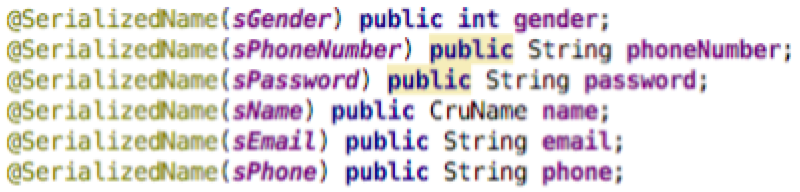


```
CruName: {
        First: String
        Last: String
},
Email: String,
Phone: String,
Password: String,
Gender: int
isPublic: boolean,
isStaff: boolean,
staffRole: StaffOption,
isCommunityGroupLeader: Boolean,
isMinistryTeamLeader: Boolean,
isSummerMissionLeader: Boolean,
schoolYear: int,
fcmID: String,
Notifications: {
        ministryTeamUpdates: Boolean,
        communityGroupUpdates: Boolean,
        summerMissionUpdates: Boolean
},
Permission: {
        isAdmin: Boolean,
        isVerified: Boolean,
},
Involvement {
        communityGroup: Relationship,
        ministryTeams: Relationship,
        summerMission: Relationship
},
lastActive: Datetime
```

Figure 18. User Data Model

3.5b Create Account Model

For this data model, I had to create a special model that satisfies the Create Account endpoint. The model that renders the JSON object includes CruName, gender, phone number, email, and password. It was pretty difficult to come up with a model that the endpoint accepted. Utilizing Postman, I was able to send out countless request trying what exactly the endpoint required so that I may be able to adapt the JSON object as a data model. One of the minor yet funny mistakes that I have encountered in creating this model was that I made a minor mistake on the request body types that were being sent to Postman. I was supposed to send out JSON objects to the endpoint, but for the longest time, I was sending plaintexts. I was able to find out the correct model but because of the wrong body type, I still received an error back from the database. Figure 19 showcases the correct and working model to be used when creating an account.



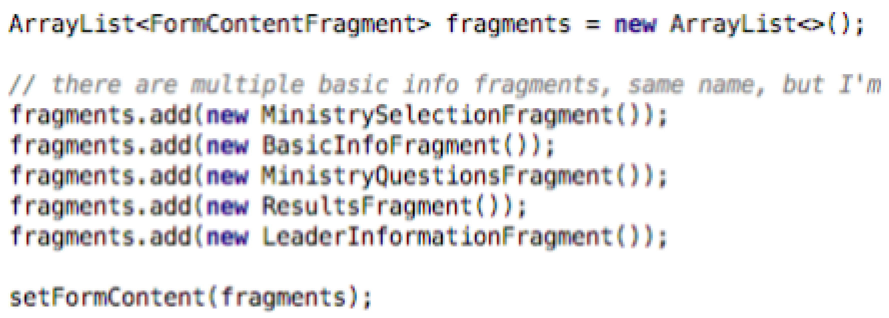
```
@SerializedName(sGender) public int gender;
@SerializedName(sPhoneNumber) public String phoneNumber;
@SerializedName(sPassword) public String password;
@SerializedName(sName) public CruName name;
@SerializedName(sEmail) public String email;
@SerializedName(sPhone) public String phone;
```

Figure 19. Creat Account Model

4. Issue Faced

4.1 Legacy Code

Since this project started back in 2015, some of the code that I had to read through was at least two years old. It came with many difficulties in terms of understanding the code itself, figuring out the design pattern, and lack of error handling or edge case checking process. My project is mainly building on top the existing architecture and working with legacy code brought extra confusion during my development time. Even though this experience was difficult, I was able to pick up some crucial skills that will improve my personal career as a Software Engineer. Through this process, I was able to pick up the intention of the old code and actively update it and/or made improvements to it. This experience will be discussed in a later section. Overall, working with legacy code opened my eyes to other design patterns. For example, in figure 20, the legacy code calls for different fragments being put into one array list to be projected one after another. I thought it was a unique way to present multiple fragments and passing data between each fragment.



```
ArrayList<FormContentFragment> fragments = new ArrayList<>();

// there are multiple basic info fragments, same name, but I'm
fragments.add(new MinistrySelectionFragment());
fragments.add(new BasicInfoFragment());
fragments.add(new MinistryQuestionsFragment());
fragments.add(new ResultsFragment());
fragments.add(new LeaderInformationFragment());

setFormContent(fragments);
```

Figure 20. Fragment Arraylist

4.2 Broken Backend in Database

The current backend database has been in development since 2015 as well. The advances in backend development have allowed more seamless connections and also a lot more data manipulation combination during requests. With the updated models, it has caused a lot of issues in the legacy code. One of the said areas that had affected my project development is within the community group area. The misalignment caused the app to crash every time when I try to find a community group. What made it worse was the log files only reported a very ambiguous error. Fortunately, utilizing Android Profiler, the core issue has quickly surfaced and the issue was fixed right away.

4.3 Customer and Advisor

I think the biggest issue that I've faced was determining what to work on that can satisfy both the customer, Cru, and senior project requirements. The Cru application has matured enough over the years for an alpha release. With that intention, the customer was trying to aim for fixing existing issues instead of creating new features. At the same time, the senior project has to meet a certain threshold of advancement on the app itself. I felt like I was caught in the middle between two different intentions for this project. At times, it was frustrating where I want to satisfy both parties but I do not have the capacity or much say in either matter.

5. Lesson Learned

The path to reach what I've accomplished was not the easiest one. There were times where I had to reach out to the original students that worked on this project. In the end, it was ultimately a learning experience for me to become a better software engineer.

5.1 Understanding Legacy Code

Since my project is built on legacy code, it was a learning experience for me to debug and dive into how the code base functioned. There were many times where variable names were just "i" or "x" and did not give a description of it should be. Utilizing the powerful debug build on Android Studio, I was able to discover the data that was being passed around within the codebase and successfully (with few bumps in the road) implemented my features. The consistency of pushing myself to understand the code rather than asking for explanations will help me in the future as a developer. In the workforce, not a lot of people will have the time nor patience to sit and explain the code to me. This experience allowed me to witness my potential on diving into the unknown.

5.2 Try and Fail Better

One of the major learning point while developing for this project is understanding the idea that we should try and fail but fail better the next time. The acceptance for failure opened up room for me to try different approaches to problems. Knowing that each failure will lead to a working product allowed me to come up with interesting and unique approaches to a single problem. With this knowledge going into the workforce, I believe I am able to put the Cal Poly slogan “Learn by Doing” to the real test. Each attempt is a way to learn about what works and does not work.

6. Conclusion

The experience on developing features on Cru Android App has been a wild ride for my last year at CalPoly. I’ve gained immense knowledge in Android development and also acquired more experience in software engineering as a whole. Utilizing industry standard technology that I did not typically use for my personal projects has expanded my view on how powerful and versatile Android development can be. Furthermore, I was able to work as a P.M. with Kyle Fletcher to find out what exactly he wants out of this project. At the same time, I can maximize my output as a developer and learn how to truly understand customer’s wants and needs.

In the future, I can definitely see this app takes off and serve as a tool for Cru across all campuses in the US and enable students to connect with a ministry that they feel comfortable in. This application does have the potential to have an industry standard functionality but I believe it will have to take a few years of a full development team to reach that goal.