**Ryan Carson**

**Dyson Fraser**

**Tyrel Froese**

**COMP 4910: Computing Science Project**

**Project Report**

Table of Contents

[Abstract 4](#_Toc37602230)

[Requirements 6](#_Toc37602231)

[Initial Requirements 6](#_Toc37602232)

[Week 3 6](#_Toc37602233)

[Week 4 6](#_Toc37602234)

[Week 5 7](#_Toc37602235)

[Week 6 7](#_Toc37602236)

[Week 10 7](#_Toc37602237)

[Analysis 8](#_Toc37602238)

[Initial Analysis 8](#_Toc37602239)

[Week 3 8](#_Toc37602240)

[Week 4 9](#_Toc37602241)

[Week 5 9](#_Toc37602242)

[Week 6 9](#_Toc37602243)

[Week 7 10](#_Toc37602244)

[Week 10 10](#_Toc37602245)

[UI Design 11](#_Toc37602246)

[Initial UI Design 11](#_Toc37602247)

[Second Iteration 12](#_Toc37602248)

[Final Iteration 13](#_Toc37602249)

[System Design / Architecture 16](#_Toc37602250)

[Implementation 17](#_Toc37602251)

[Building the project and running the app 17](#_Toc37602252)

[Android 17](#_Toc37602253)

[Running the Debug Version 17](#_Toc37602254)

[Building the Release Version 17](#_Toc37602255)

[iOS build 18](#_Toc37602256)

[Relevant Project Files & Folders 18](#_Toc37602257)

[App.js 18](#_Toc37602258)

[VirtualCommunitiesList.js 18](#_Toc37602259)

[package.json & node\_modules 18](#_Toc37602260)

[ios & android 18](#_Toc37602261)

[Testing 19](#_Toc37602262)

[Appendix 20](#_Toc37602263)

[Client Documentation 20](#_Toc37602264)

[Installing the Deep Map app on Android 20](#_Toc37602265)

[Adding a marker to the Deep Map app 26](#_Toc37602266)

Abstract

The idea of a cross-platform mobile application is a very appealing product idea for many different organizations and can seem quite simple for a non-technical client on the surface. However, when starting to look into the problem and analyzing what it would take to create a mobile app for multiple different platforms with the correct systems in place for a client to manage the content that would be on the app, it is apparent that the scope of developing such a system can become much larger than expected. This is the experience we found when developing the Deep Map Mobile App.

Our project for the Computing Science Project course this semester was the Deep Map Mobile App. Proposed by the Accessibility Experiential Learning Coordinator Jennifer Mei at TRU, this project was to create a mobile app that would replicate the functionality currently present on the [Deep Map Website](https://access.trubox.ca/). Our motivation for working on this project was that our team was already familiar and comfortable with Mobile App Development from some courses taken at TRU for android development, and were interested in what it would be like to develop an cross-platform app for an actual business client , as cross-platform development is a highly sought-after skill in the industry.

Two main challenges to solve this problem arose with this project. The first problem is how we could develop a cross-platform app that would be up to professional standards on both iOS and Android in the limited development time of this project. We were all familiar with developing for Android, and if we had simply stuck with the Android side this would be a simple problem to solve. Knowing that we had to create a product up to our client’s expectations not only for Android, but also for iOS meant that we could either split our team into two halves and develop this application natively on both platforms, effectively doubling our implementation time, or look to an alternative solution. This led us to look into Cross-Platform Frameworks, and subsequently to the React Native framework. Developed by Facebook, this platform is still being constantly worked on but seemed like the industry standard when it came to Cross-Platform Frameworks. While this platform was foreign to us at the beginning of the project, leading to a steep learning curve, we felt that this would be the best approach for us to develop our skillset in software development and save time developing different products at the same time.

The next challenge we had to solve was the challenge of how our client would actually manage the content of the app she was wanting to be made. While we considered creating a custom version of the app for our client to manage the content, we felt the most user-friendly way to achieve this was to create a new webpage solely for the purpose of managing this app. We believed that it would be a better user experience to use a simple user interface rather that manipulate a database directly. To achieve this, we developed a WordPress site specifically made for managing the database content for our app.

This Project Report aims to provide depth into the technical and non-technical requirements we were faced with in this project and the analysis and decisions we made to meet the needs of these requirements. It also serves to present a chronological view of the UI design of the app and the system architecture of the entire software system, from the app to the database and to the webpage created for our client to manage the database. Finally, it serves to provide references to video demonstrations to validate the testing of the app for all features in its feature set.

Requirements

For this project, the requirements were rather simple from a client deliverable perspective. She simply wanted an app to do all of the functionality available currently on the DeepMap Website. While the deliverables seemed easy, the requirement to have the app present on both Android and iOS presented some unique challenges in terms of technical requirements. In addition, we also had to consider how our client would manage the content on the app itself.

# Initial Requirements

* Develop a mobile app with the same features as the [Deep Map Website](https://access.trubox.ca/)
  + Integrated map of locations
  + Filter for different resources to appear on the map

# Week 3

* Possibility of Community Feature system, using Database and User Authentication & Management features
  + Added after the base prototype of the app looks polished and functional to the client’s expectations
  + Considered as a "nice to have" feature, but not a necessary requirement
* Distributing through both iOS and Android
  + Android is simple and easy to distribute, Ryan has gone through the process of publishing on the Google Play Store before
  + iOS has an overhead cost of $99 USD per year, client will follow up with her contacts at TRU about the costs and process of distributing the app through TRU's channels
* Develop for Android and iOS using React Native
  + Developing using a cross-platform tool would be a great learning experience
  + Open to alternatives depending on how the development of the first iteration goes

# Week 4

* App needs to scale for tablets and larger phones
* Need to make sure the app will run on iOS early in development

# Week 5

* Cloud-based database required for client to add, edit and manage the points on the map
  + Decided to try Google Firebase, as we have some experience with it already and integrates nicely with Mobile and Web platforms
* App should have scrolling tabs, rather than having buttons on the bottom of the screen to go to separate pages
* App should use a native map within the app, rather than using a WebView displaying content already on the DeepMap Site

# Week 6

* Can distribute app on app stores with our own accounts, doesn't have to go through official TRU channels
* App should have a tab for "Virtual Communities", online resources without a physical location (ie. a Facebook group)
* We should be providing an easy-to-use webpage hosted on a website that the client can manage the data from the database

# Week 10

* Map markers should have a pop-out page, where a marker can be tapped on again to show a page that consists of all relevant information for that marker

Analysis

The analysis of the problem gave us many different approaches we could take. Since this product was already available on a webpage, we could have easily took that webpage and just made it display within a mobile app. We decided against this approach, opting to create our own mobile application not only for the flexibility and iteration that we could do with the final product, but also so that we could gain experience as developers to solve a common problem in the industry right now which is how we can create a cross-platform mobile app that maintains a consistent look and feel. We also had a few options to approach the solution to content management. We opted to create a webpage for our client, as that would allow her to manage the database on any device. While this did expand the scope of the project as a whole, it gave us a better glimpse into what true system design looks like in practice.

# Initial Analysis

* Best Mobile development platform for us would be React Native
  + Cross-platform support
  + Exports to native projects, where native development can take place if necessary
  + An alternative to this would be to develop both apps simultaneously for both platforms, but the overhead work to do so might take more time that simply learning a cross-platform environment such as React Native

# Week 3

* Community features would require some additional requirements
  + Would require a user authentication system and database support
* How to deal with content management from the client
  + Creating a separate version of the app with additional access to add / update content
  + Creating a simple webpage that can access a Database
* Distributing the app to users
  + Easy for Google Play Store
    - only $25 USD one-time fee
    - Less restrictive
  + Difficult for iOS
    - $100 USD yearly fee
    - more restrictive with content

# Week 4

* App should scale on multiple different devices and screen sizes
  + Decided to start working on the iOS app early in development and continuously test both versions of the app whenever changes were made
  + This decision ultimately lead to our decision to divide the work of the three of us into 3 different platforms, one on Android, one on iOS and one focused on the client webpage using Javascript

# Week 5

* Our client presented us with a contact from a recent trip with some app development experience that we could contact. After discussing as a group, we decided against incorporating him into our project, instead leveraging him as a resource if we needed. We had already determined our technical requirements and didn't want to include someone else that might have a different vision and scope of the final project than our own, especially considering our limited development time for this project.
* Decided to use the Cloud-based Database Google Firebase for content management, as we have some experience with it already and integrates nicely with Mobile and Web platforms

# Week 6

* Decided it would be best to change the User Interface of the app from buttons at the bottom to take you to a new screen to scrolling tabs
* Switched our Map screen implementation from an iFrame that just displays the current DeepMap site to a MapView in React Native that takes advantage of the Google Maps API. Google Maps Services had to be enabled on both platforms for this to work, but it was a necessary improvement that allows us to integrate our database with the map for easy content management for our client

# Week 7

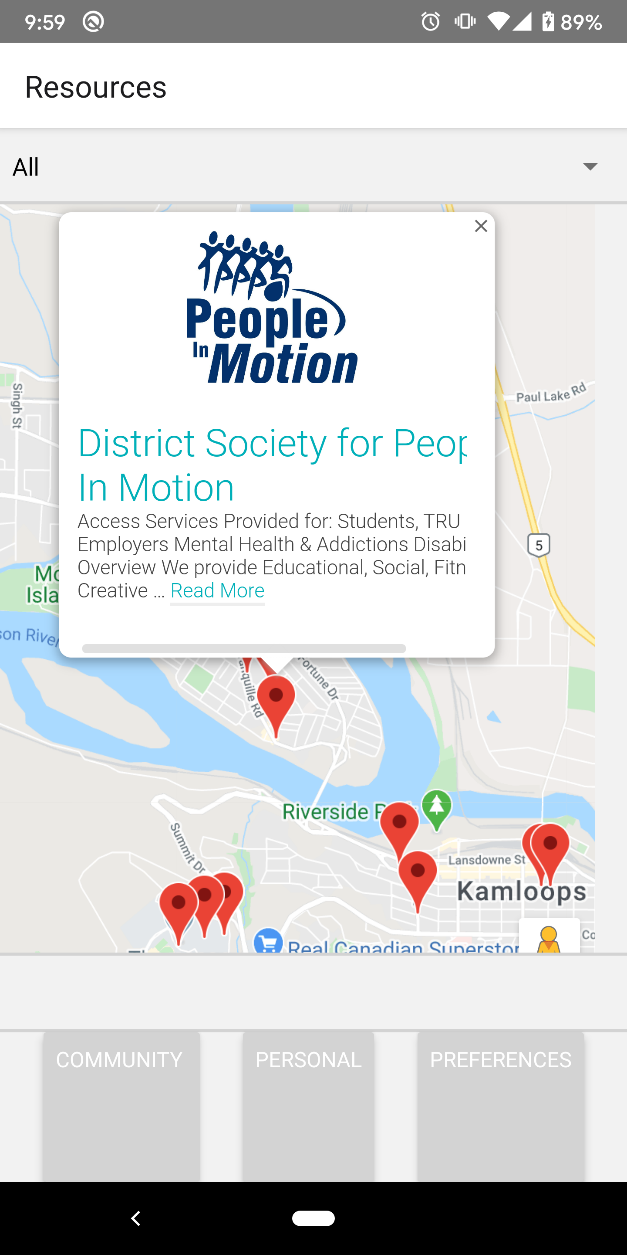
* Added a section on the map for Virtual locations, as some community resources might not have a physical location (eg. a Facebook group, website)
* Created a webpage on Wordpress using our Firebase database and javascript to allow our client to easily edit
  + An alternative to this that we considered was to create a separate version of the app made specifically for our client that would allow her to edit the map, but felt it would be more user-friendly and take less development time to simply create this as a webpage

# Week 10

* To account for the filter we were planning to add to the app, we modified the database and the client webpage to allow for location categories. A location in the database now has a subtree of categories with True or False values to indicate whether it is part of a community or not.

UI Design

# Initial UI Design



For our first prototype, we decided to revolve our UI elements around the main focus of the app, which is the Deep Map itself. In order to save on development time and get a first iteration to our client as soon as possible, we opted to simply display the map of the DeepMap webpage as a WebView, simply displaying the Deep Map as it currently is on the DeepMap Webpage. While not entirely functional at this point in the project, we also included tabs at the bottom of the page for future pages to be added, such as a Community and Preferences page. We also included a filter at the top of the map, as one of our requirements is to allow for filtering by different communities.

# Second Iteration

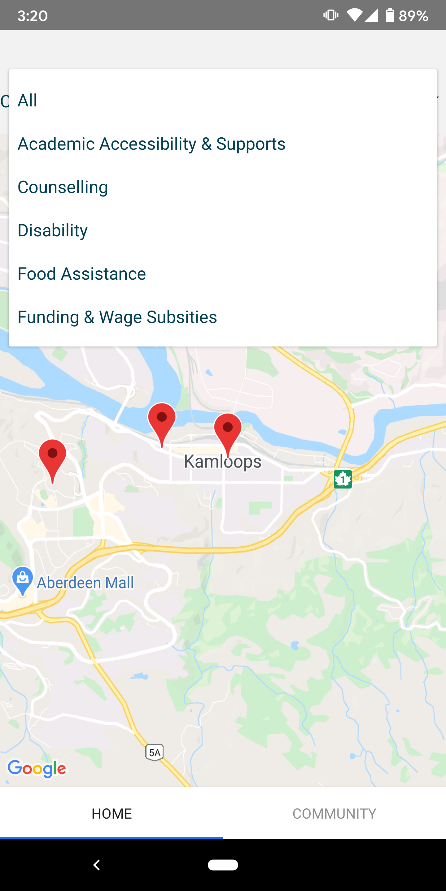


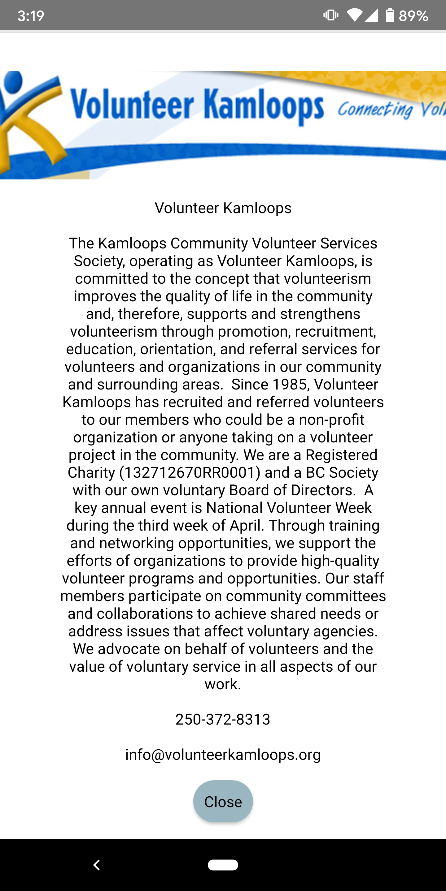
In our second iteration, we implemented the MapView within React Native. This view takes advantage of the Google Maps API. We chose to create our own MapView rather than continue using a WebView to the DeepMap Webpage, as this would allow us to manipulate the markers on the map using our own database and do any further edits to map outside of using the Webpage.

# Final Iteration

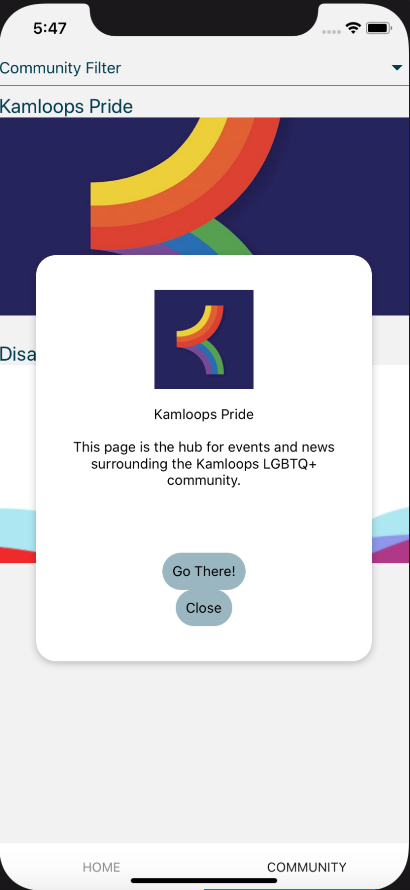
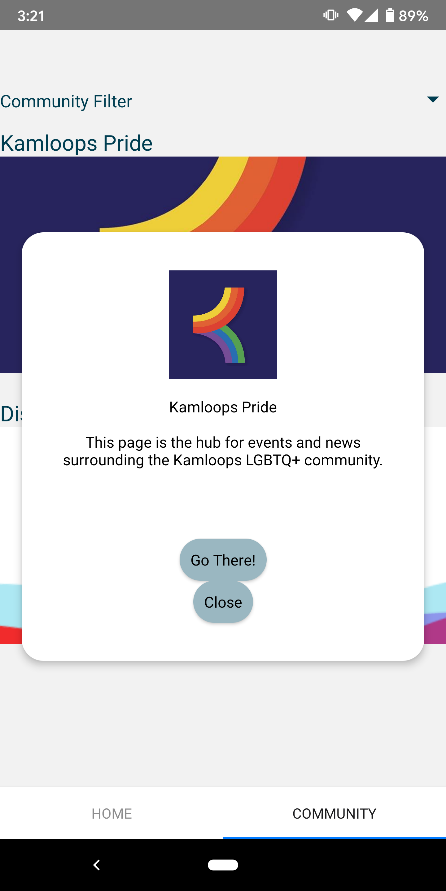












Our Final Iteration of the app further polishes the look and feel of the app, taking advantage of scrolling tabs at the bottom of the screen. It also uses a Community tab to show Virtual Locations, community resources that don't have a specific location such as Facebook groups and websites. In addition, we created a Modal Pop-out page that displays a larger, more detailed view of each individual marker on the map. This modal is available on the Map as well as the Virtual Communities page.

# System Design / Architecture

Our system design / architecture is composed of three main parts. The first, and most obvious part of this architecture is the app itself. This part of the system simply displays the content of the system to the end users, and is the only user-facing part of the system.

The second part of our architecture is our Firebase database itself. This NoSQL database is a simple database that has a unique child for each marker on the map. Under each marker, the database stores all relevant information for that marker, including the title, description, email, phone number and website of the location. In addition, it also includes a subtree of booleans, indicating what community each marker is a part of.

Finally, the last piece of this architecture is the client webpage. This webpage allows our client to directly manipulate the database in a very user-friendly interface. This hides the backend technical database from our client and ensures the integrity of the database, as the client webpage is very well structured to not "break" the underlying structure of the data in the database.

Implementation

# Building the project and running the app

In order to run the DeepMap project, you first have to setup React Native on your local machine. The instructions on how to do so are located on the [React Native Website](https://reactnative.dev/docs/environment-setup). For this project, we built the project using the React Native CLI Quickstart instructions, so the project has not been tested with the Expo environment.

## Android

### Running the Debug Version

1. Open a command line and navigate to the project folder
2. run npm i
3. adb uninstall com.deepmap (only necessary if a previous version is already installed)
4. npx react-native run-android

Note: this will often fail when trying to build the debug app. In our experience, you sometimes have to run npx react-native run-android up to 5 times for it to successfully build. An alternative to this would be to simply run the android folder within Android Studio.

### Building the Release Version

1. Move the **gradle.properties** file with the keystore info to DeepMap\android
2. Move the **my-upload-key.keystore** to DeepMap\android\app
3. Open a command line and navigate to the android folder of the project (DeepMap\android)
4. Run gradlew bundleRelease
   * Note: the [Official Documentation](https://reactnative.dev/docs/signed-apk-android) advises to use ./gradlew bundleRelease. In our experience, this does not work on a Windows Environment
5. Navigate back to DeepMap and run npx react-native run-android --variant=release
6. The release apk will be located at DeepMap\android\app\build\outputs\apk\release

## iOS build

You have have some issues using the command npx react-native run-ios

If you get errors when building

1. Make sure to have Cocoapods installed
2. Navigate to the IOS Folder in your Terminal
3. Use the command pod install
4. Try building the app again

# Relevant Project Files & Folders

Many of the files in the project directory are generated by React Native, but there are some important files that must be worked on in order to modify the project.

## App.js

This file is the main file that loads everything in the project upon opening. It is similar to the Driver class or Index file of a typical program or webpage. This file includes all of the import statements relevant to the project, the HomeScreen class including all the UI elements that appear when the app loads for the first time, the integration for the Firebase database and the Style Sheet that includes all the styling necessary for the UI elements.

## VirtualCommunitiesList.js

This is the view for the Virtual Communities that is imported in the **App.js** file. It populates a list of virtual communities on the second tab of the app.

## package.json & node\_modules

The **package.json** file includes a list of the necessary dependencies needed for the React Native project. When the command npm i is run, it looks to this file to create everything necessary in the **node\_modules** folder.

## ios & android

The **ios** and **android** folders hold the project files for the iOS and Android projects respectively. They were automatically generated when the React Native project was created and change content based on and change as the React Native project is edited and rebuilt. These folders can be directly imported into the xCode and Android Studio IDEs, allowing for easy testing if already familiar with these IDEs or further implementation specific to either platform.

Testing

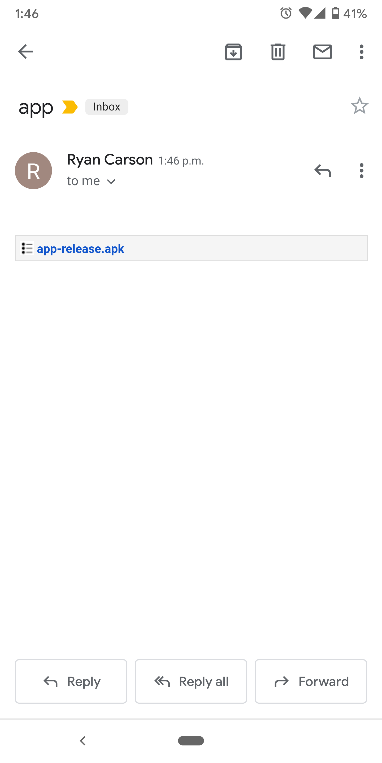
For this project, our testing plan was quite simple. Unlike a complex product such as a machine learning product, our final product was rather simple, so we simply focused on making sure our app and the client website remained stable and reliable under different use cases. To illustrate the different use cases and feature set working under different conditions, we have included two demonstration videos to our Project Binder on our GitHub page. These demonstration videos are located at <https://github.com/djryancarson/DeepMap/tree/master/Project%20Binder/5.%20Testing>. While there is a bit of lag with the demonstration video for the app, this is caused by the software used to capture the footage and is not reflective of the final product.

Appendix

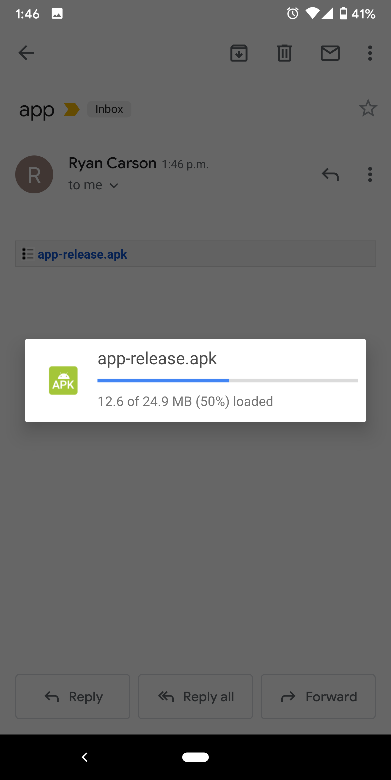
# Client Documentation

## Installing the Deep Map app on Android

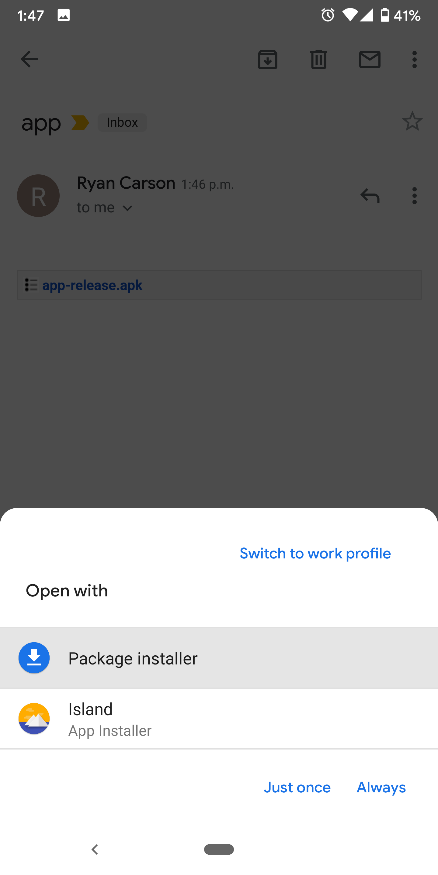
**Step 1.** Open the app-release.apk file from the email on your phone.



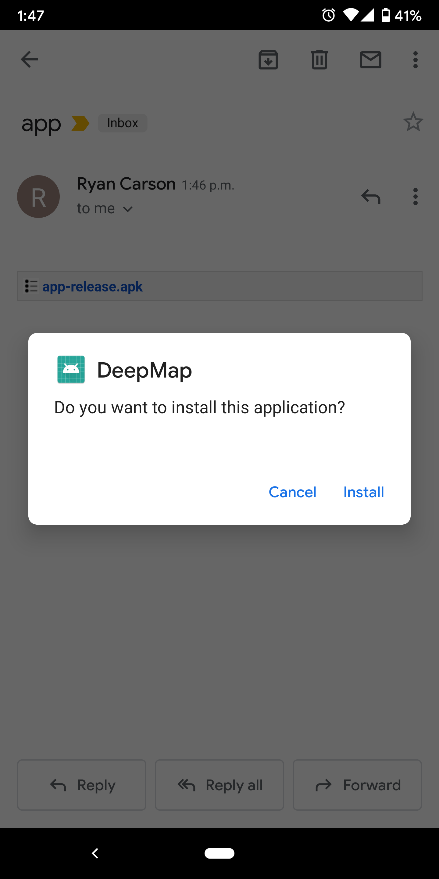
**Step 2.** Allow the app to load.



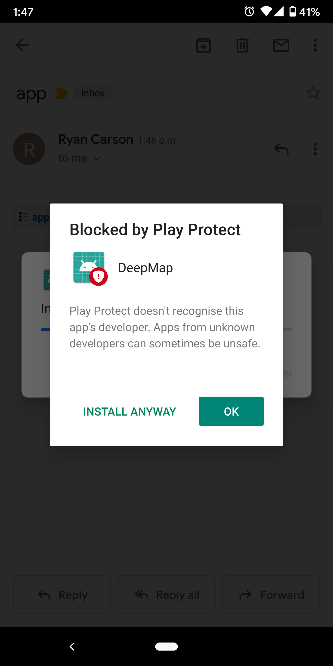
**Step 3 (if prompted).** Choose the Package Installer.



**Step 4.** Select **Install**.

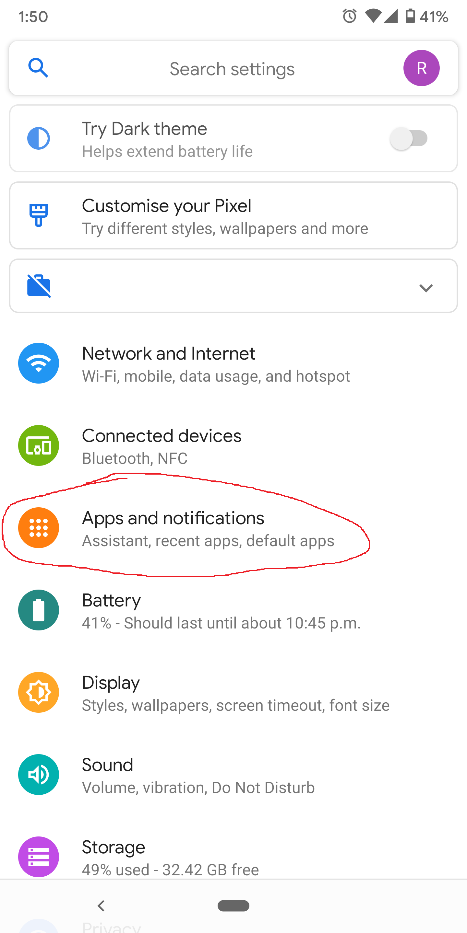


**Step 5.** Select **INSTALL ANYWAY**.

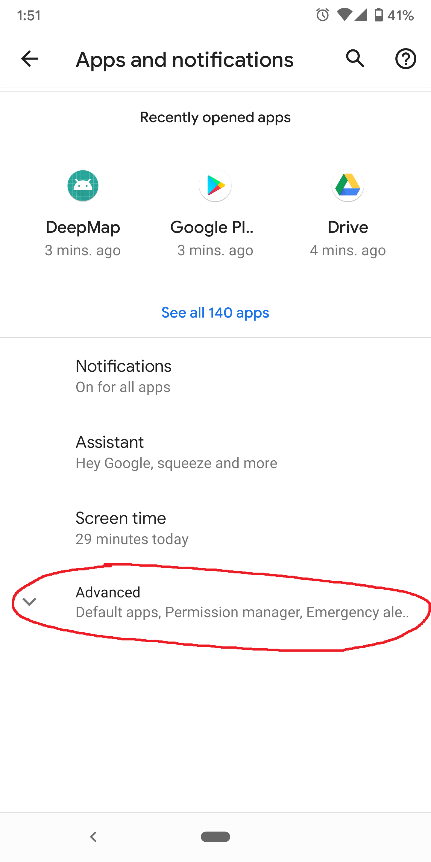


**Note**: You may be asked to enable your email program to allow you to install APK files. If so, follow these steps:

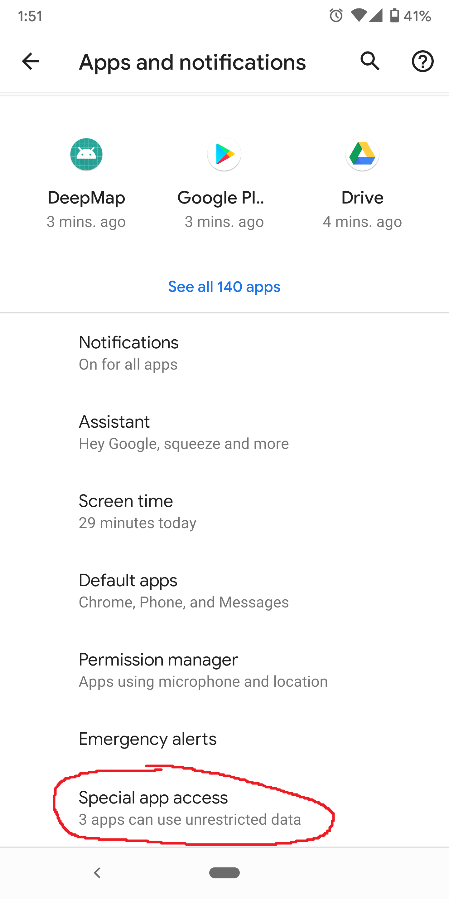
**Step 1.** Select **Apps and notifications.**

****

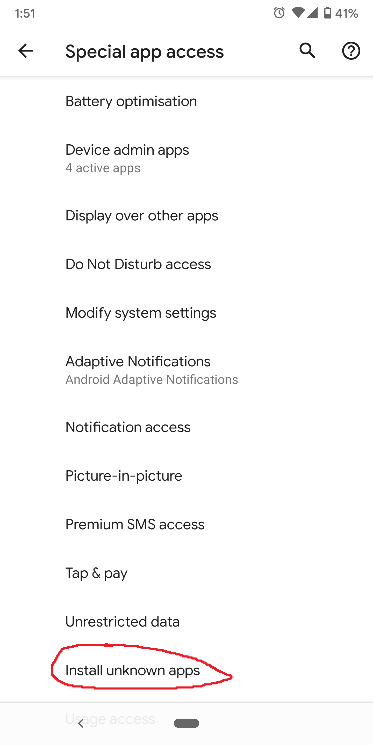
**Step 2.** Select **Advanced**.



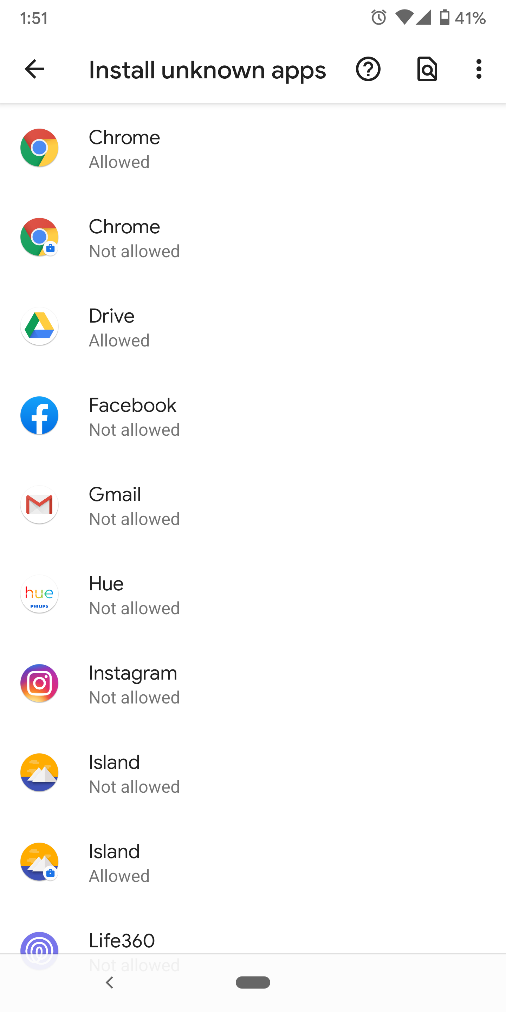
**Step 3.** Select **Special app access**.



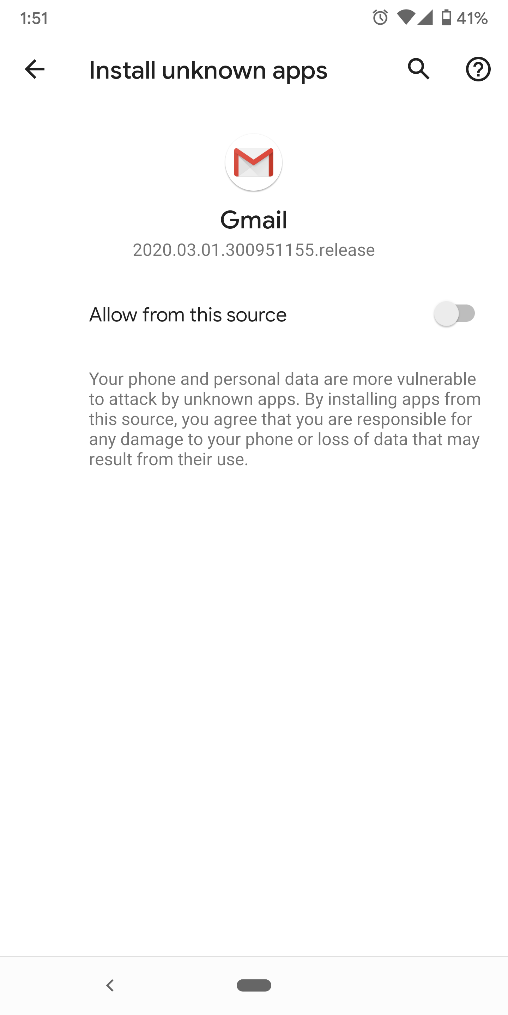
**Step 4.** Select **Install unknown apps.**

****

**Step 5.** Select your email app you are trying to open the file on (eg. Gmail).



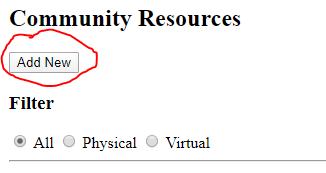
**Step 6.** Select the option **Allow from this source**.



## Adding a marker to the Deep Map app

**Step 1.** Go to the [Deep Map Database Webpage](http://www.djryan.ca/DeepMap/home.php)

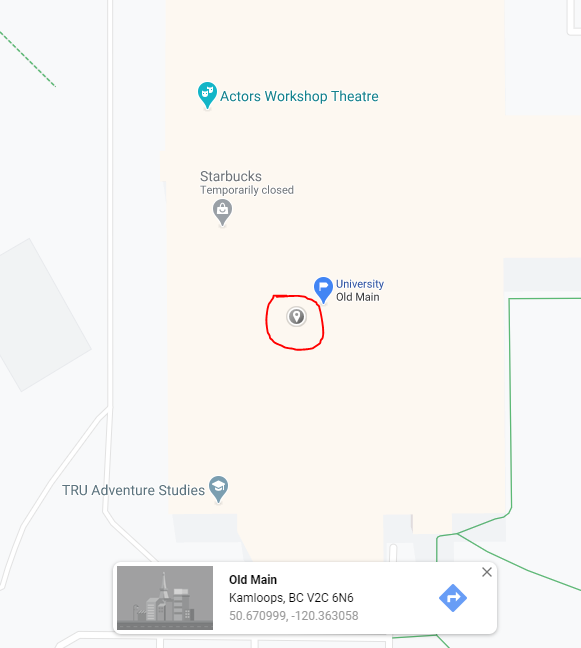
**Step 2.** Select Add New



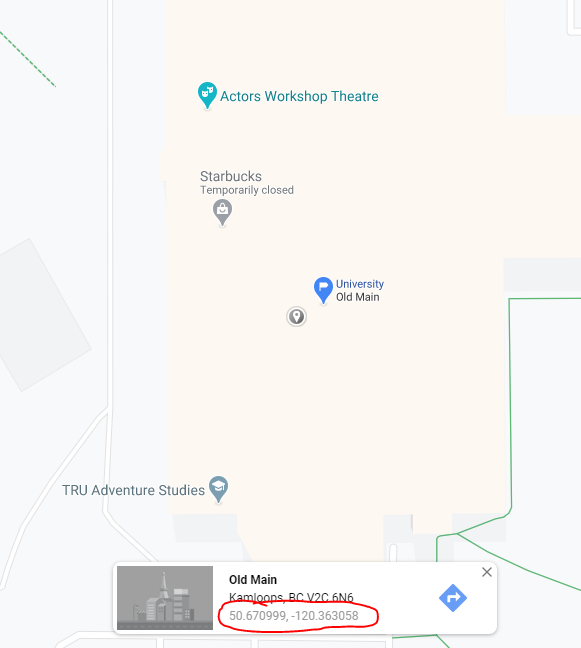
**Step 3.** Go to [Google Maps](https://www.google.ca/maps/)

**Step 4.** Click on the point you would like to add

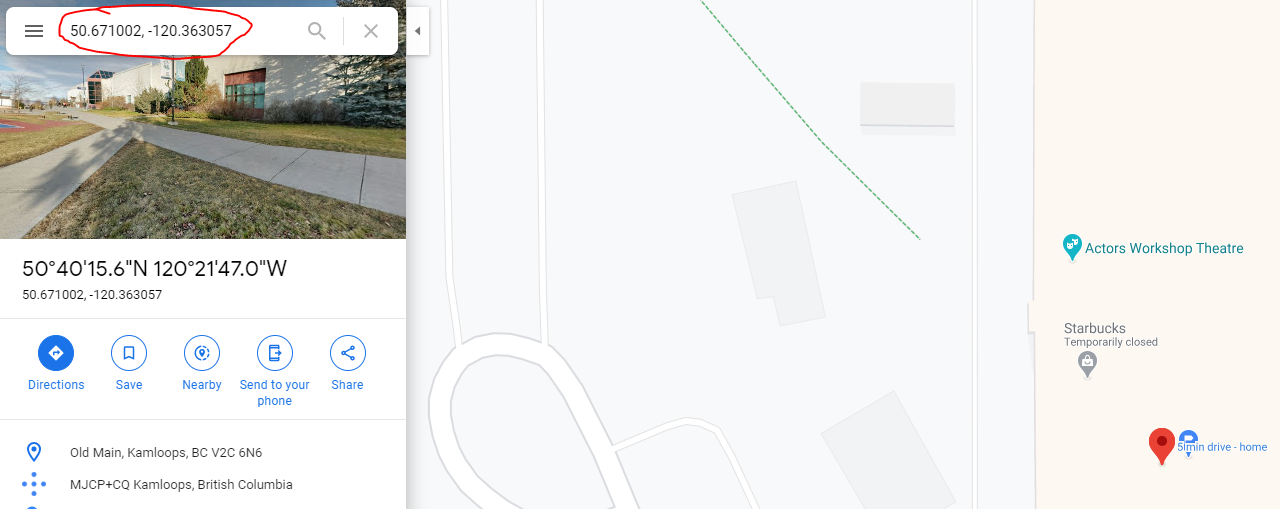
**NOTE:** Do not click on an actual marker on Google Maps, click on an empty space instead



**Step 5.** Click on the **Coordinates**



**Step 6.** Copy the Coordinates at the top of the screen



**Step 7.** Paste these coordinates into the Location field along with any other relevant information you’d like to add and click **Submit**

