iOS-ROS Interface

Bharat Jangir

Advisor: Dr. Zack Butler

Capstone Project Report

Summer 2016

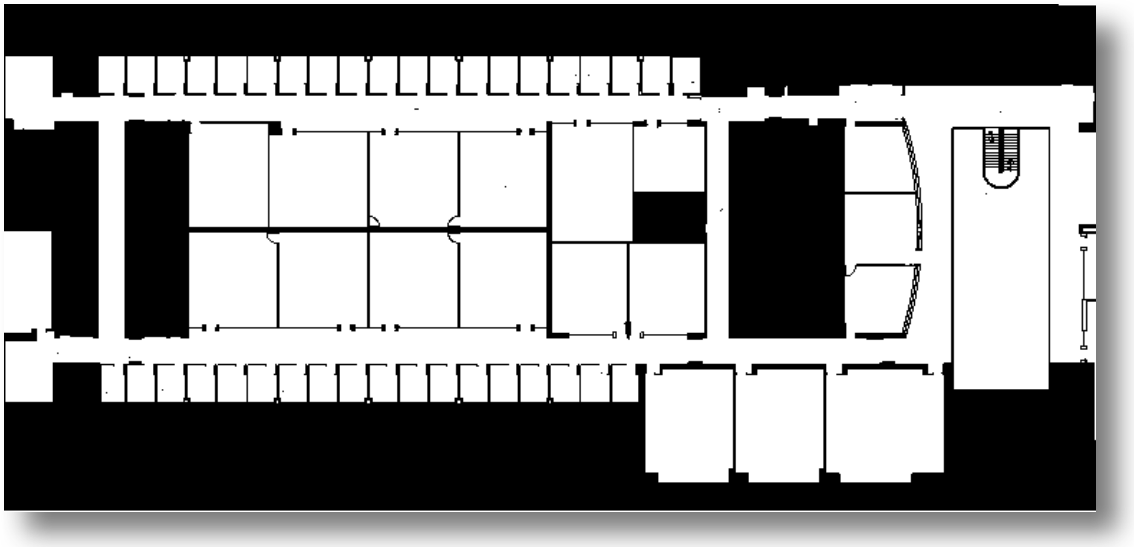
Table of Content

Background

The RIT Computer Science students may get access to work on the Corobots found in Room 3405. Below is a picture of a team of Corobots.

Corobot is a self-navigating robot that can interact with people both locally (through a mounted netbook or tablet) and remotely over the Web. You can read more about the project here https://www.cs.rit.edu/~robotlab/corobots/.

They have a laptop mounted on top that runs the ROS (Robot Operating System). The laptop is connected to two cameras on sides and a Kinect mounted below it. The two side cameras help the robots with localization in the Map. There are QR codes all around the CS floor that are being used by these Corobots by help of these side cameras to localize themselves.



The above is the third floor of the Golisano building CS Floor Map. All the development is done on this map as these Corobots are not made for outdoor environment and do not like stairs. The Microsoft Kinect has infrared inbuilt that helps in detecting obstacles and their distances. This primarily helps in obstacle avoidance, wall detection, etc.
The Corobots are pretty powerful Robots and can be used for development and deployment by the developers. There is a guide for launch instructions (Check Slack Guide) that lets developers launch the Corobots. Once the system is active it lets the web server vhost1.cs.rit.edu know that it is good to go. You can view the status of these Robots at vhost1.cs.rit.edu/status.php

To non-developers, there are limited means of communication with the Corobots. Once the robots are live, it would be more interactive for end users to engage in real time movements and habits of these robots. With so many applications that these Corobots can be used for, different apps can make the same robot behave differently.

Goals

In this project we are interested in creating an iOS app that lets user watch the stream of the Kinect of the Corobots, View status of Corobots and localize himself/herself as using QR code. The scope of this project is to develop a native iOS app that can stream video over the network, localize itself in the map and display feeds from the network server about the robots around them. Google firebase provides authentication services for the project.

Special Note

All special notes like username/password, etc. are attached to **Trello** and **Slack** updates. Make sure you check them for any updates about the project. All files of this project are uploaded on Slack.

Current System Configuration

Working on Corobot #6
Currently running ROS Groovy on Ubuntu 12.10
IOS Development version: 10
Language: Swift 3
Firebase version: 3

Getting started

Read ROS tutorial to understand some of the basics of the ROS system. You can follow this guide. http://wiki.ros.org/ROS/StartGuide
For best practices, use Slack and Trello for updates on the project. There is a file corobot_readme.pdf file with instructions on how to use these tools.

Proposed System

Robot Operating System (ROS) has lot of packages which are essentially building blocks of ROS system. The different packages deal with different task. Nodes are runtime processes. Different packages create different nodes for different tasks. There is a Master node which communicated with other nodes and manages them all.

The Architecture diagram below is used for this project. We are interested in the video feed from the Kinect, communicating with the webserver and watching movements of the Robots real-time in the Map. The vhost1.cs.rit.edu web server keeps tabs on all the Corobots. The iPhone interface will communicate with this web server to show overview of the activities of these robots.

The ROS node mjpeg_server will stream image topics from the Robots. This stream is captured in the app and live camera feed from the Kinect can then be viewed on the iPhone. The app is able to use the iPhone camera to decode RIT CS Floor QR Codes and localize itself in the CS Floor map. Firebase services provide authentication and real time location services of the user.

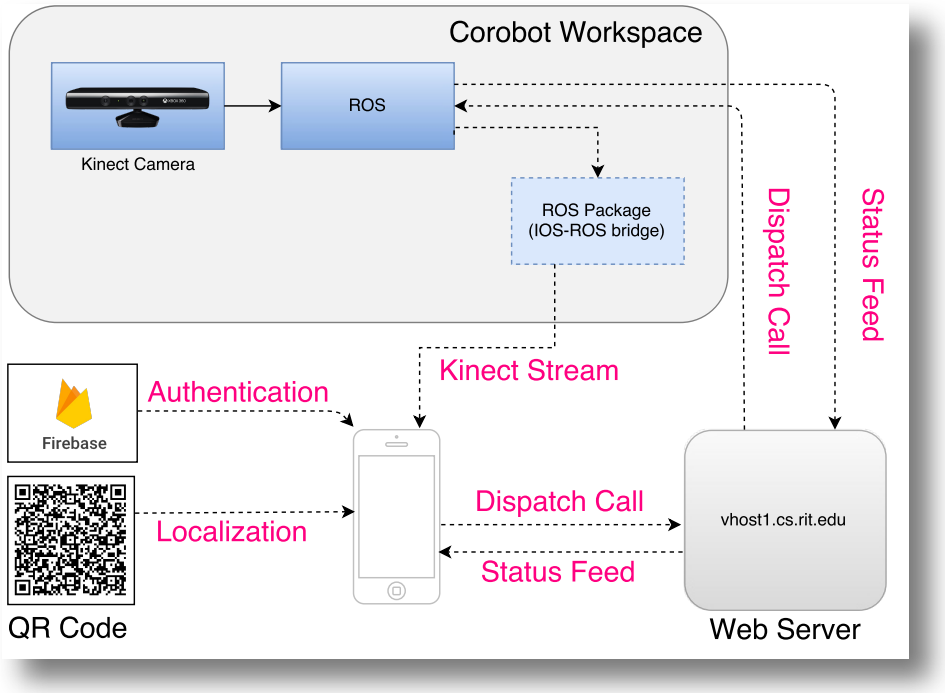


Figure 1: Architecture

Details about the approach

There are essentially three major components that complete the milestones. The first component is to get connected to the Vhost1 webserver which in turn connects to all the Corobots. The second one is to localize the user so that the user environment is relevant in the Corobot world. The third component is to have a map that plots all live Robots and are able to view and control the Robots on this map.

The first component was connection to webserver. Although there are many libraries for HTTP request, there were very few updated recently. Based on research, we found there are two major approaches that we could to take. One was WebKit, native iOS library to handle all the HTTP request or AlamoFire, elegant Swift HTTP networking library. The second one was very easy approach and handles lot of errors very elegantly. The Authentication services for AlamoFire hides lot of complexity and takes care of ATS (App Transport Security) itself. Unfortunately, it was still not updated for Swift 3. It did not compile with the Project and then had to take the first approach. All the code then was done using WebKit library. Authentication is really tricky with WebKit and since the vhost1 webserver has an invalid certificate, you have to change the ATS settings in order to communicate with it. I followed guide mentioned here - https://ste.vn/2015/06/10/configuring-app-transport-security-ios-9-osx-10-11/ to open communication with vhost1 webserver.

The second component was getting user location for localization on the CS Floor indoor map. The first try was using Firebase to locate the user on the floor. This route was unfruitful as the user location kept updating with error and was not reliable for our indoor environment setup. The QR codes are already mapped on the CS Floor which provide quick and easy solution for localization for the user on the CS Floor Map. There is a plethora of QR Code scanner libraries available for Swift. We were looking for libraries that were recent and compatible with Swift 3 and also grants license to use. The QR Code scanner library found at https://github.com/yannickl/QRCodeReader.swift (MIT license) was most recently updated and was compatible during our search with our code in Swift 3. Installation for its dependencies can be done via Cocoa Pods which is explained later in the iOS development section.

The third component is the Map. This map was influenced by the map found on the Corobots that display the Corobot's location in the Map and displays statistics about the current status of the Corobots. This enables end users to interact with the Corobots and view their real time status. The user can also view his location. Hence this provides users with no knowledge about Corobots to interact with them and get some information about it.

The authentication system of the Vhost1.cs.rit.edu handles RIT students' login credentials. But the purpose is to provide users with no knowledge of Corobots to create account and still view

some information about these Corobots. This is why the authentication system was decoupled from the Vhost1 web server as the users of the app may or may not be developers of the Corobots. Firebase provides location services for the user with pretty good precision but was not enough for indoor environments. We needed much superior accuracy which was done using QR Code scanner.

Details about the setup and installation

Apt-get

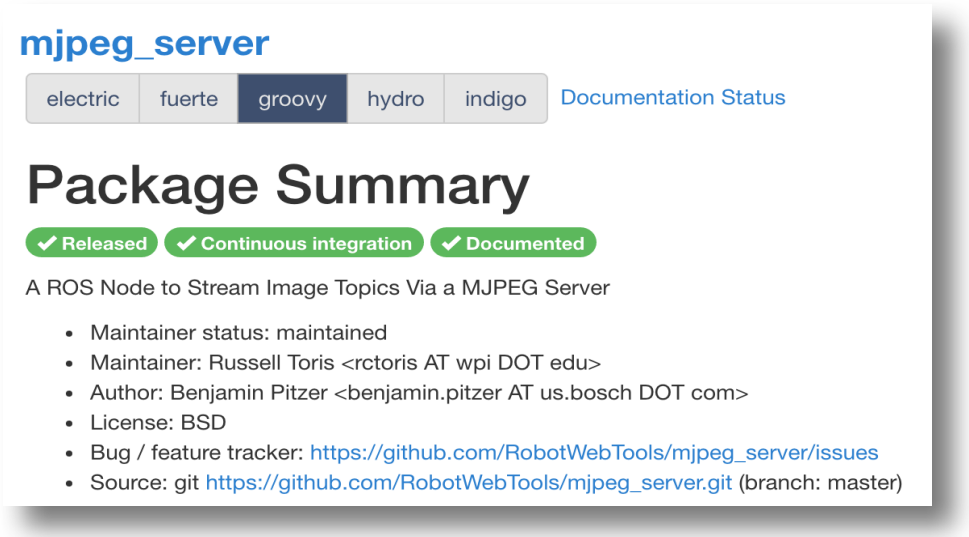
Our solution involves first installing a ROS package on Corobots using apt-get. First make sure you have apt-get, you can check it using

```
dpkg-query -l apt-get
```

If you do not have apt-get, install apt-get using *Ubuntu Software Center*.

Mjpeg Server

This is a package found on ROS http://wiki.ros.org/mjpeg_server. For the video streaming from the Kinect we are going to use this package. Description from the above link is displayed below.



Check if your machine has this node installed, if it doesn't follow the steps below. Install mjpeg_server by using instructions below in terminal. This will help us setup stream Image Topics of Kinect feed over the network.

sudo apt-get install ros-groovy-mjpeg-server

For development purposes, use ports between 7000 and 8000. I randomly chose 7555. *You can then launch it by*

rosrun mjpeg_server mjpeg_server _port:=7555

You can now *view* the kinect feed in browser by opening the following URL. This is the same link used by the app to view the stream from the Kinect. For this project we are using the raw image_topic. You can replace the IMAGE_TOPIC below by "?topic=/**camera/rgb/image_raw"**

http://localhost:7555/stream?topic=/IMAGE_TOPIC

There are different parameters you can add specify what kind of stream you want.

```
http://localhost:8080/stream?topic=/IMAGE_TOPIC?param1=value1?param2=value2?p
aram3=value3
```

The different parameters are as follows:

width (integer, default: original width): This is used to specify the width of the frame.

height (integer, default: original height): This is used to specify the height of the frame.

quality (integer, default: 90): This Parameter is used to specify the quality of the JPEG images. This can be changed to change bandwidth of the stream.

invert (none, default:): This rotates the image 180 degrees.

iOS Development

At the time of this project development Apple had released preview of the new Swift 3 and iOS 10; available only to Apple paid developers. Therefore, when the iOS 10 is released to the public it should be compatible with it.

As for the setup of the project, it was a standard setup, where each new file was a "Swift" file.

All of the dependencies can be installed via Cocoa pods. Cocoa Pods is dependency manager for the Swift. Cocoa Pods can be installed via default ruby on OSX. Use the following command in terminal.

```
$ sudo gem install cocoapods
```

Install the following dependencies for this Project using the instruction on this page. The first one installs Firebase dependencies for authentication and second one installs QR Code Scanner libraries.

https://cocoapods.org/pods/Firebase
https://cocoapods.org/pods/QRCodeReader.swift

All apps in iOS are build app using MVC design pattern. The following explanations is an overview of the code. Let's check the view controllers first. All the following explanation is done based on the code used.

SplashScreenViewController

This is the first screen of the app. This screen provides options for Login or Register.
This is connected to the Firebase Server. The setup of the screen is done using **setupView()** function. There is a function **createButtons()** to create the buttons and the events of the user pressing these buttons is handled by **signInButtonPressed()** and **registerButtonPressed().**

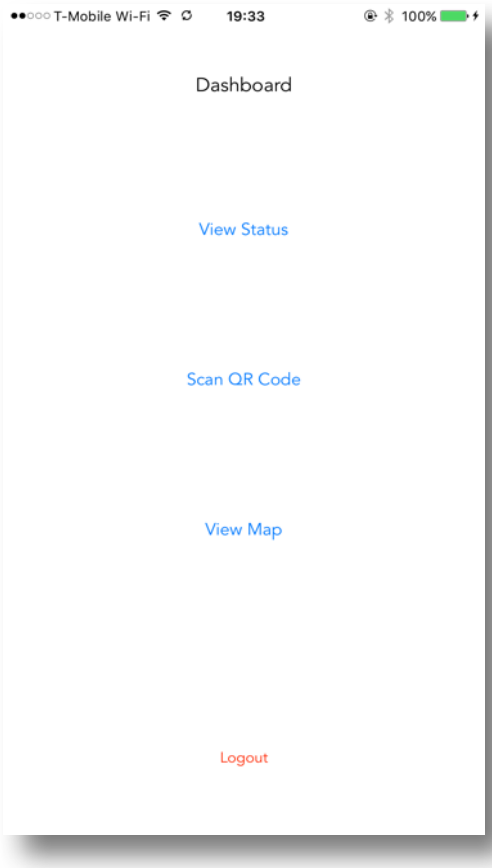
There is a video playing in the background using AVFoundation library. The video is added in the **setupView()** function and there is a **stopVideo()** function to stop the video from playing in the background when user goes to other screen.

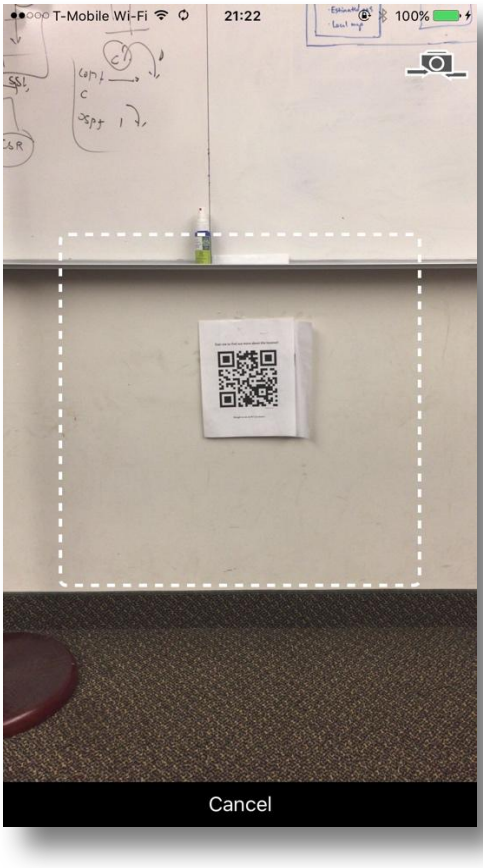
SignUpViewController

The above screen is also linked to this controller. This is mainly linked to Google Firebase Authentication service. User can register using any email ID. If they use RIT Email Credentials to register and log in, then it will automatically log them on the vhost webserver. Otherwise they have will be prompted with another log in screen stopping them from using the vhost webserver.

User information is then stored into Firebase database. User can login later using these credentials.

The notable functions in this controller are **addUserToFirebaseDatabase()** which adds new users automatically to the firebase database and **login()** which verifies with the firebase database if the user is registered or not and deal with other exceptions.





DashboardViewController

It provides 3 options: View Status, Scan QR code, View Map. This is also the main controller for QR Code Scanner implementation. The dependencies for the QR code was installed above. To import the library. Just use **import QRCodeReader**

You can find more information about it here:
https://github.com/yannickl/QRCodeReader.swift

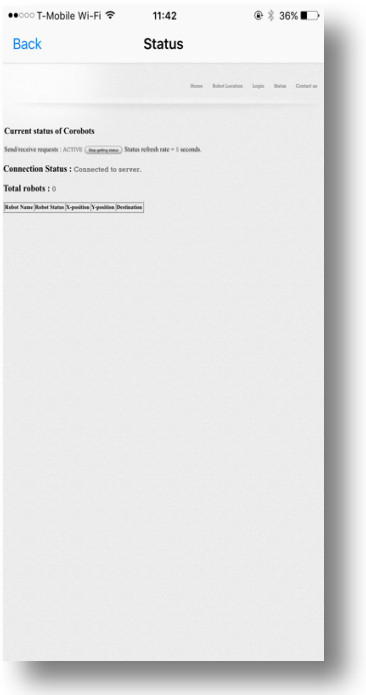
First option on this screen **ViewStatus** connects to the **StatusViewController.** This controller is explained later.

The second option is to scan the QR Code. This is to localize the user in the map. First the csv file that contains all the locations "**barcodePoints.csv**" is converted to an array. This helps us get the location of the user in order to locate in the map. The scanQRCode function accesses

camera on your phone to scan QR Codes. Once user scans the QR code, it receives a URL which is then searched in this array. Once a hit is found, the X and Y coordinates are sent to the the **FloorMapViewController**. Then using the function **viewMap**() it displays the users location on the Golisano building 3rd floor map.

ViewMap connects to the **FloorMapViewController**. This is controller is explained later.

User also has an option of **logout**. User stays "*logged in*" even if the user closes the app. The app saves the credentials. If user chooses to log out it will then prompt the user for credentials next time.

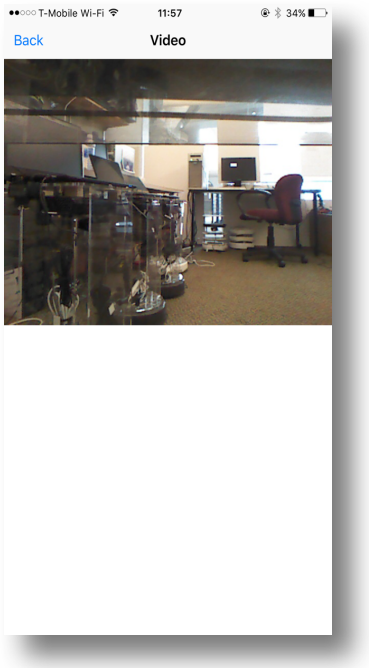


StatusViewController

We attempt to make a connection here to the Vhost1 server. If the user has logged in using the RIT credentials it will automatically log him into the webserver otherwise it will prompt them to enter the credentials again.

Essentially the viewDidLoad() function connects to the login page of the Vhost1 and request Firebase for username and password of the currently logged in user. The username password is passed to the vhost1 webserver. If RIT authentication grants access the user is logged in. Otherwise it will prompt the user to login again.

The authentication is done using WebKit's authentication system. This controller connects to the vhost1.cs.rit.edu server in order to login and dispatch Corobots. Logging in to the server grants user access to dispatch the Corobots. It also provides the option "Show Video" when the robot has been dispatched. This "Show Video" button connects to the KinectVideoViewController.

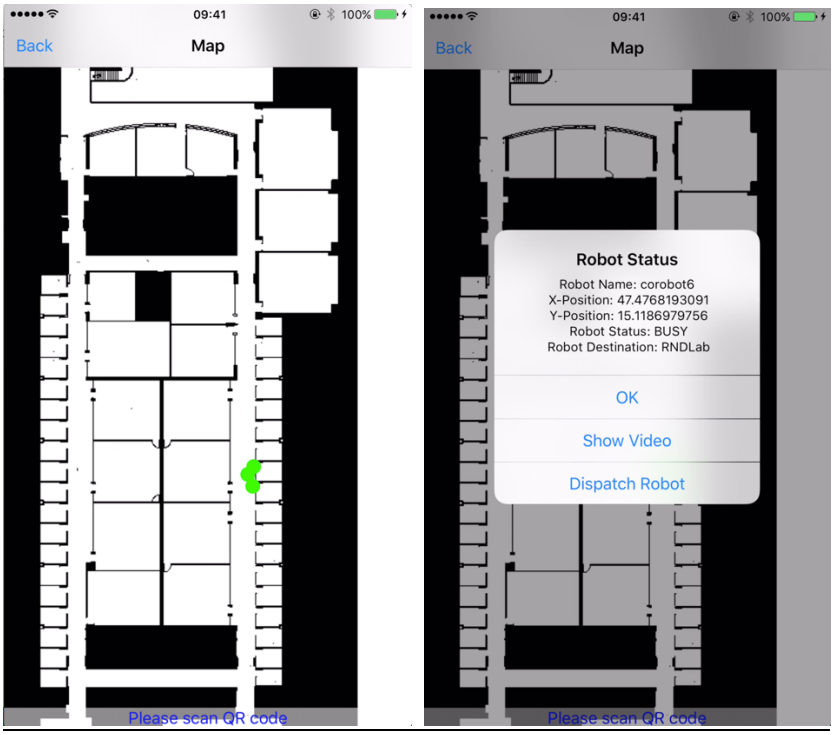


KinectVideoViewController

This controller shows the live video stream of the robot. It connects to the localhost server of the robot where the video is streaming and shows live video. The video is being hosted by the MJPEG server which can be accessed at URL. For #Corobot 6, below is the the URL for viewing the Corobot's Kinect feed when it is live.

http://129.21.106.63:7555/stream?topic=/camera/rgb/image_raw

We create a NSURL session for continuous streaming and keep refreshing it. We create a browser view using WebKit which displays the URL Information. For different Robots this link needs to be updated.



FloorMapViewController

This controller shows Golisano third floor map and also shows user's and corobots(if any live) locations on the map. User location is shown in red. Robot location is shown in green. User can see the robot moving on the map. It also shows the coordinates and destination and status of robot when user taps on the robot's location. It also shows the coordinates and user's location name when user taps on his/her location.

We took the image used by the Corobots for the CS floor map. We make this image a UIImage. Set the properties of this UIImage for displaying the image vertically and scale it to the screen. Then we overlay information on top of it. We create dynamic buttons on top of this image along with their pop-up's. Whenever we have the user location, we update it using the function **plotUserLocation().**

The web server status page update the web page using a JSON object. This JSON object tells us the current status of Robots. We convert this JSON object to an array using the function **parseRobotLocations()**. This array is iterated and then plotted using the function **plotRobotLocationsOnMap().** There is also a button action that's applied on these dynamically created buttons. This is handled by the function **buttonActionForRobots()**

Models

ThirdFloorLocation - Model class to store properties xCoordinate, yCoordinate, locationName for third floor of Golisano building.
RobotStatus - Model class to store status of a robot. It has properties like xCoordinate, yCoordinate, destination, status. We get the status of the robot as json object from the link https://vhost1.cs.rit.edu/cgi-bin/jsonOutput.php

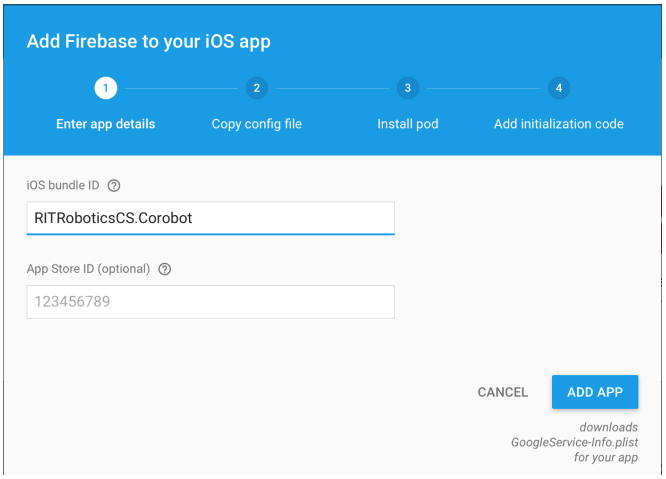
---

Firebase

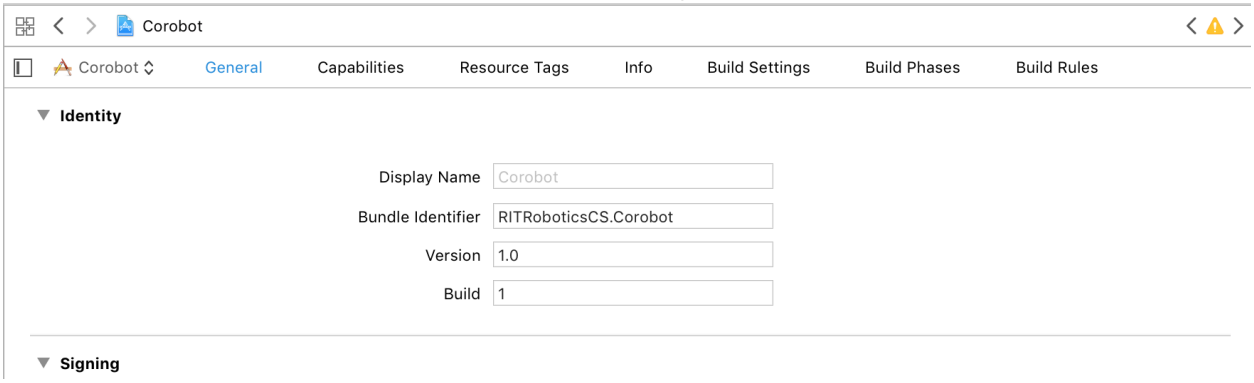
Firebase is a backend cloud service provider for mobile apps.
Development of Firebase is free but if more users sign up then you can pay for premium service. It provides up to 100 free live connections at any moment, which is enough in my opinion.
All of the Corobot development on Firebase is done using the username xxxx.corobot@gmail.com (Check Trello). Sign in using the credentials to follow instructions below.
If it is a not a new development and is a similar project but just a different platform, skip the "Creating a new Project" part below.

Creating a new Project

In case you need to create a new project you can click "Create new project". This will display the following.



The iOS bundle ID can be found in Xcode (Click on project name>General)

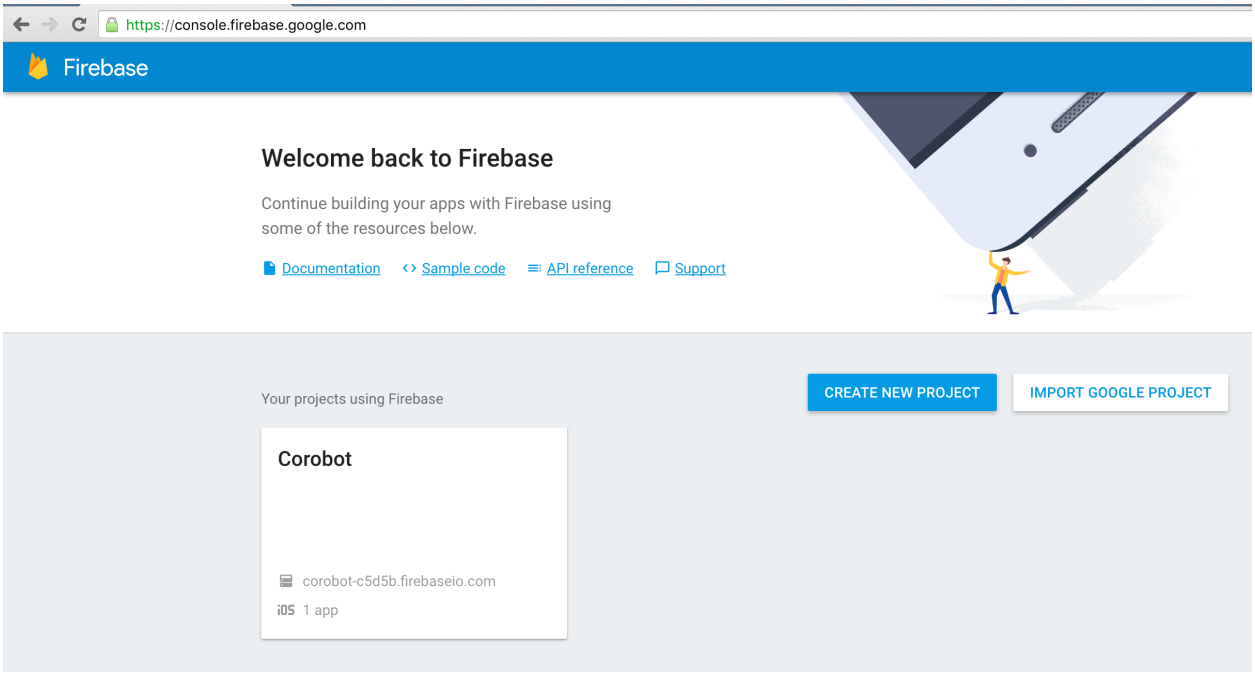


Follow the rest of the steps. They are pretty self-explanatory.

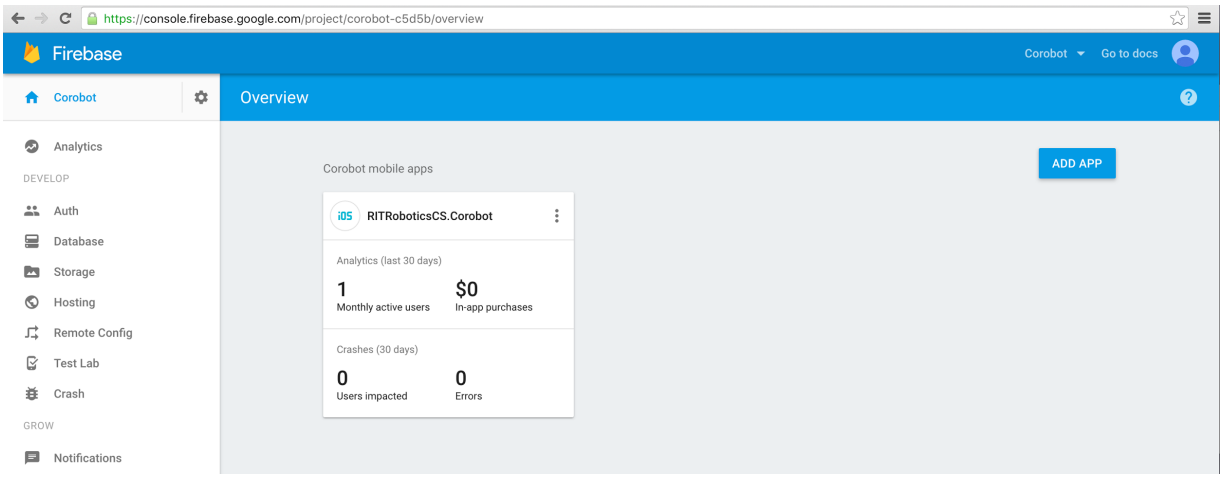
Existing Project

One Firebase can handle many projects and is also cross-platform. Therefore, each individual app is a project which would contain the "iOS" as well as "android" version of the code. This project is done with use of username "xxxx.corobot@gmail.com" to login to Firebase. Check the Trello page for password.

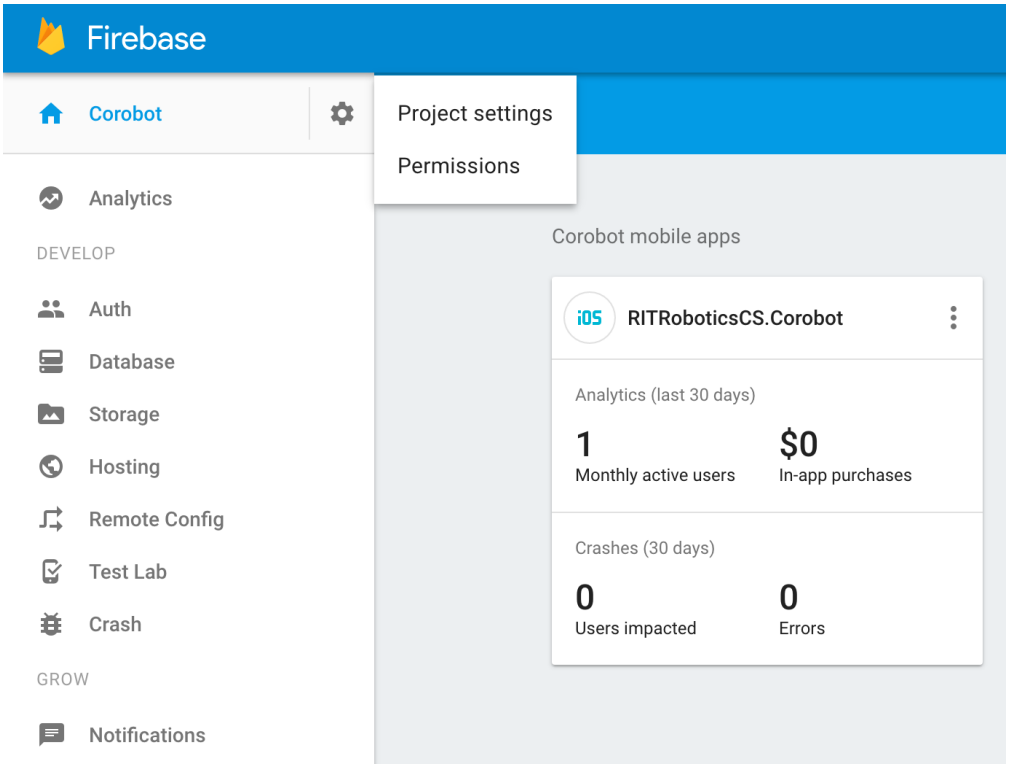
On the Console of the firebase, there should be project named **Corobot**



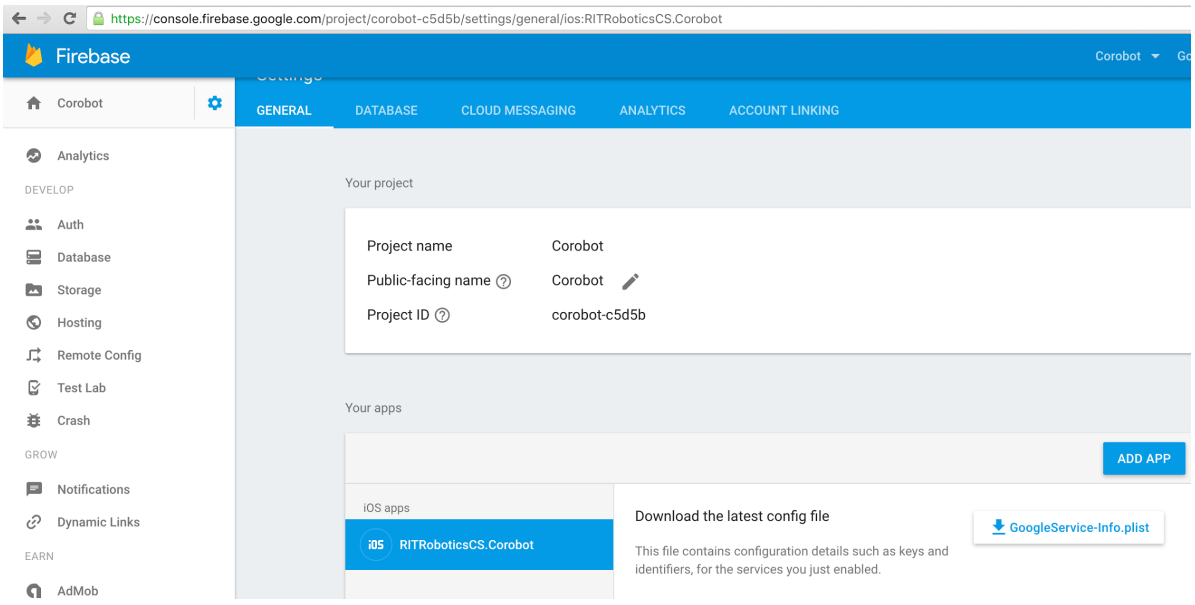
When you click on the project Corobot, you can see **overview** of the app. This project is iOS development but let's say you are developing android app for the same project, can use this same project in firebase. Because the logic for backend remains the same. Below you can see the Corobot overview page shows one mobile app associated. That is this project. You can add your app here (upper right side) and get started.



If you wish to download the config file for the project. You can click the settings icon and then Project settings.

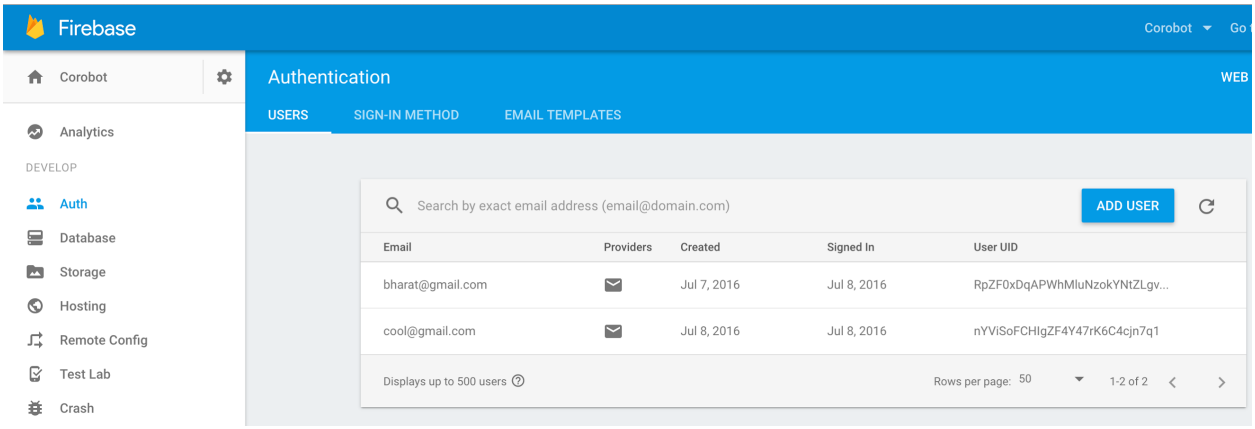


The **settings page** below contains the info-plist file critical for development with Firebase and iOS. It also has settings for Database and other features. This is needed for our project.

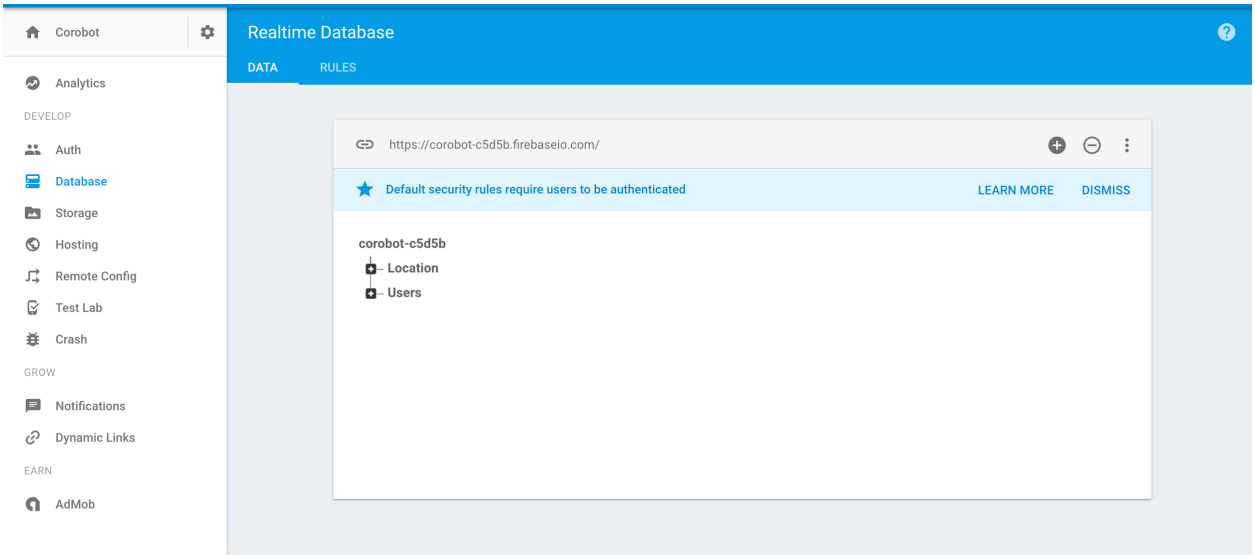


We are using real time database, Geo-fire, authentication and analytics for this project from the collection of services they offer.

The Auth Table below shows all the users registered with the app.



The database below currently contains the users and their locations.



Other components are not connected but could be used in the future for other developments.

Experiments

The first option "view status" is linked to the vhost1.cs.rit.edu web server. Once you log in to the app, you have to log in to the vhost1.cs.rit.edu from inside the app if the user email address is not an RIT email address. The network system is pretty robust. I ran the system for about two days and the robots still shows up on the map. The app did not crash throughout the test.

The second component to the app is the QR Code detection. The QR Code is able to scan all locations in the map and user gets localization on the map. The QR Code fails if it receives anything that is not in the **waypoint.csv** file and then the app crashes. Otherwise it keeps updating the user location.

The third component is the Map. The map is updated every 1 second for some update. The map contains the green sign for the Robots and shows the update for user location (X, Y coordinates) in the bottom of the screen. If user has not scanned it shows in red to scan the QR Code. Once it recognizes the location, the user location is in updated in red and shows the location of the user in the label at the bottom of the screen.

Firebase location services using GPS failed to provide services (to update the user location in the map) because of accuracy required for the indoor environment. The QR Code is quick and accurate localization option for our setup.

Conclusion

The app is able to complete all the functionalities mentioned above. You are able to scan the Robot at any location on the CS floor and is able to detect it. The user is successfully able to communicate with the Web Server and the Corobots connected to it. The app is able to handle any change in location of user or the robot themselves.

Future Idea

The app could be used for various purposes. There are many Robots that need a native IOS app for the ROS system. This would work well for any indoor location with a map.
This app could be used to be the front end for the new users so they get familiar with functionalities and cools things it can do. You can create something as basic as controller for kids to move around the robot (there is already code for it on GitHub!) to something like attaching the iLauncher (has a nerf gun attached) to let the user control where he wants to shoot. Also maybe with the advent of all the new image object detection API services, you

could create some processing in the front end for the user. Or there could be some fun game with all of this.
The Vhost1 server could also be deployed to Firebase server.

References

http://wiki.ros.org/mjpeg_server

https://developer.apple.com/reference/webkit

https://firebase.googleblog.com

https://vhost1.cs.rit.edu/status.php

https://vhost1.cs.rit.edu/location.php

https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CLLocation_Class/

https://github.com/yannickl/QRCodeReader.swift