Prototype Study Report

|  |  |
| --- | --- |
| **FOODY Mobile Client Application (FMCA)** |  |
| Document Reference: | 1.0 |
| Project: | Foody Mobile Client Application (FMCA) |
| Document Title: | Prototype Study Report |
| Version: | 1.0 |
| Date: | 15 May 2017 |
| Author: | Sivasankari D/O Subramaniam  **Jason LU Wenshuo**  Vikram Jariwala |
| Reviewers: | Gandhimathi Somasundaram  **Naw Win Win Latt**  Virendra Yadav |
| **©** 2017  The information contained in this document is the property of ISS. The contents must not be copied in whole or in part for purposes other than which it has been supplied without the consent of ISS, or, if it has been furnished under contract to another party, as expressly authorised under that contract, then ISS shall not be liable for any errors or omissions. |  |
| ***Institute of Systems Science, 25 Heng Mui Keng Terrace,***  ***Singapore 119615*** | |

**Document Reference:** FMCA/TPS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Authorised by** | **Date** |
| 0.0.1 | 22/04/18 | GAO ZHIYU |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**FOODY Mobile Client Application (FMCA)**

**Prototype Survey Report**

**Distribution:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Department** | **Organisation** |
| GAO ZHIYU |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[Prototype Study Report Summary 5](#_Toc512191594)

[1 INTRODUCTION. 7](#_Toc512191595)

[2 Environment Setup 8](#_Toc512191596)

[2.1 Java8 8](#_Toc512191597)

[2.2 AWS 8](#_Toc512191598)

[2.3 Windows PC 8](#_Toc512191599)

[3 AWS Component 8](#_Toc512191600)

[3.1 steps and procedures for the Lambda and dynamoDB 8](#_Toc512191601)

[3.1.1 Download the sourcecode 8](#_Toc512191602)

[3.1.2 Building the Deployment File 8](#_Toc512191603)

[3.1.3 Creating the Dynamo Table 8](#_Toc512191604)

[3.1.4 Creating the Lambda Function 9](#_Toc512191605)

[3.1.5 Testing the Lambda Function 9](#_Toc512191606)

[3.2 Set up the Windows PC as the device to IOT core 10](#_Toc512191607)

[3.2.1 Go to IOT core, and click the onboarding 10](#_Toc512191608)

[3.2.2 Click the Get Started of Configuring a Device and it will route the screen below 11](#_Toc512191609)

[3.2.3 Click the Get start and it will route the page below 12](#_Toc512191610)

[3.2.4 choosing the correct OS/toolset and click the next 12](#_Toc512191611)

[3.2.5 Giving a name and click "next step" 12](#_Toc512191612)

[3.2.6 Click the windows to download the toolset, and next steps will become active 13](#_Toc512191613)

[3.2.7 After clicking next steps, it will route to the page with details text below 13](#_Toc512191614)

[3.2.8 To configure and test the device, perform the following steps. 13](#_Toc512191615)

[3.3 Final UI 14](#_Toc512191616)

[3.4 Conclusion of the prototype 15](#_Toc512191617)

[4 Location Retrieval Component 16](#_Toc512191618)

[4.1 Create a new react native project 16](#_Toc512191619)

[4.2 Location 18](#_Toc512191620)

[4.3 Conclusion 21](#_Toc512191621)

[5.1 Proposed Solution 22](#_Toc512191622)

[5.2 Prototyping using Proposed Solution Area 23](#_Toc512191623)

[5.3 Conclusion /Outcomes 31](#_Toc512191624)

[7 Facebook Authentication 36](#_Toc512191625)

[7.1 Conclusion 38](#_Toc512191626)

[8 Calling AWS lambda from react native 38](#_Toc512191627)

[8.1 Conclusion 40](#_Toc512191628)

# Prototype Study Report Summary

The below is the high-level summary of problem areas that were included to be addressed as part of prototyping and technology evaluation criteria

Table 1: Summary of the Prototype components

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Requirements**  **Category** | **Problem Statement** | **Technology Solution Proposed and Prototyped** | **Conclusion/Outcomes** |
| 1 | Functional | We need to store the structure data. As RDBMS is not a proper solution to us, so we decide to adopt the NoSQL approach | We adopt the AWS dynamoDB | DynamoDB provide functionality of NoSQL  Easier for us to do the clustering. No need to worry about the maintenance |
| 2 | Functional | We need to retrieve the information from device. | We adopt the IOT core. As we can publish and subscribe the message from topic. | MQTT Protocol Specification is a fit to purpose and easy to use solution for this problem. |
| 3 | Functional | When the msg is derived from the mqtt, we need to perform business actions | We will create the iot core rule for the msg received from the topic and perform the lambda function | Based on the research, IOT core can be used as the event source of lambda, we can create rules for how to route the messages |
| 4 | Functional | We need to process the msg received from mqtt, after the processing, we need to store in the database | We will adopt the lambda to perform the relevant the DB call | Based on the research, dynamoDB can be used the downstream of the DB call. |
| 5 | 5 | After we processed the data, we need to send back to mqtt for real time display | We need to adapt the security (private key / cert) to publish the message back to mqtt. |  |

# INTRODUCTION.

This prototype is to study the feasibility of using AWS related solution to build system for seris Application. A prototype using the AWS core/lambda/ec2/dynamoDB/ solution to be created to evaluate the complexity of the technology.

# Environment Setup

Following tools need to be installed in order to start the react native app development

## Java8

For the lambda, we adopt the Java 8 for the prototype.

## AWS

We need to have aws account. For the prototype, we are using the AWS free tier.

https://aws.amazon.com/free/

## Windows PC

We use the windows PC to act as the device to communicate to the IOT core

# AWS Component

# steps and procedures for the Lambda and dynamoDB

## Download the sourcecode

*https://github.com/gaozhiyu/SEProjectPOC.git*

## Building the Deployment File

To build the lambda application, we need to execute the following Maven command:

mvn clean package shade:shade

Lambda application will be compiled and packaged into a jar file under the target folder.

## Creating the Dynamo Table

Follow these steps to create the Dynamo table:

* Login to [AWS Account](https://aws.amazon.com/)
* Click “DynamoDB” that can be located under “All Services”
* This page will show already created DynamoDB tables (if any)
* Click “Create Table” button
* Provide “Table name” and “Primary Key” with its datatype as “Number”
* Click on “Create” button
* Table will be created

## Creating the Lambda Function

Follow these steps to create the Lambda function:

* Login to [AWS Account](https://aws.amazon.com/)
* Click “Lambda” that can be located under “All Services”
* This page will show already created Lambda Function (if any) or no lambda functions are created click on “Get Started Now”
* “Select blueprint” -> Select “Blank Function”
* “Configure triggers” -> Click “Next” button
* “Configure function”
  + “Name”: SavePerson
  + “Description”: Save Person to DDB
  + “Runtime”: Select “Java 8”
  + “Upload”: Click “Upload” button and select the jar file of lambda application
* “Handler”: com.baeldung.lambda.dynamodb.SavePersonHandler
* “Role”: Select “Create a custom role”
* A new window will pop and will allow configuring IAM role for lambda execution and we need to add the DynamoDB grants in it. Once done, click “Allow” button
* Click “Next” button
* “Review”: Review the configuration
* Click “Create function” button

## Testing the Lambda Function

Next step is to test the lambda function:

* Click the “Test” button
* The “Input test event” window will be shown. Here, we’ll provide the JSON input for our request:

|  |  |
| --- | --- |
|  |  |
|  | {    "id": 1,    "firstName": "John",    "lastName": "Doe",    "age": 30,    "address": "United States"  } |
|  |  |
|  |  |

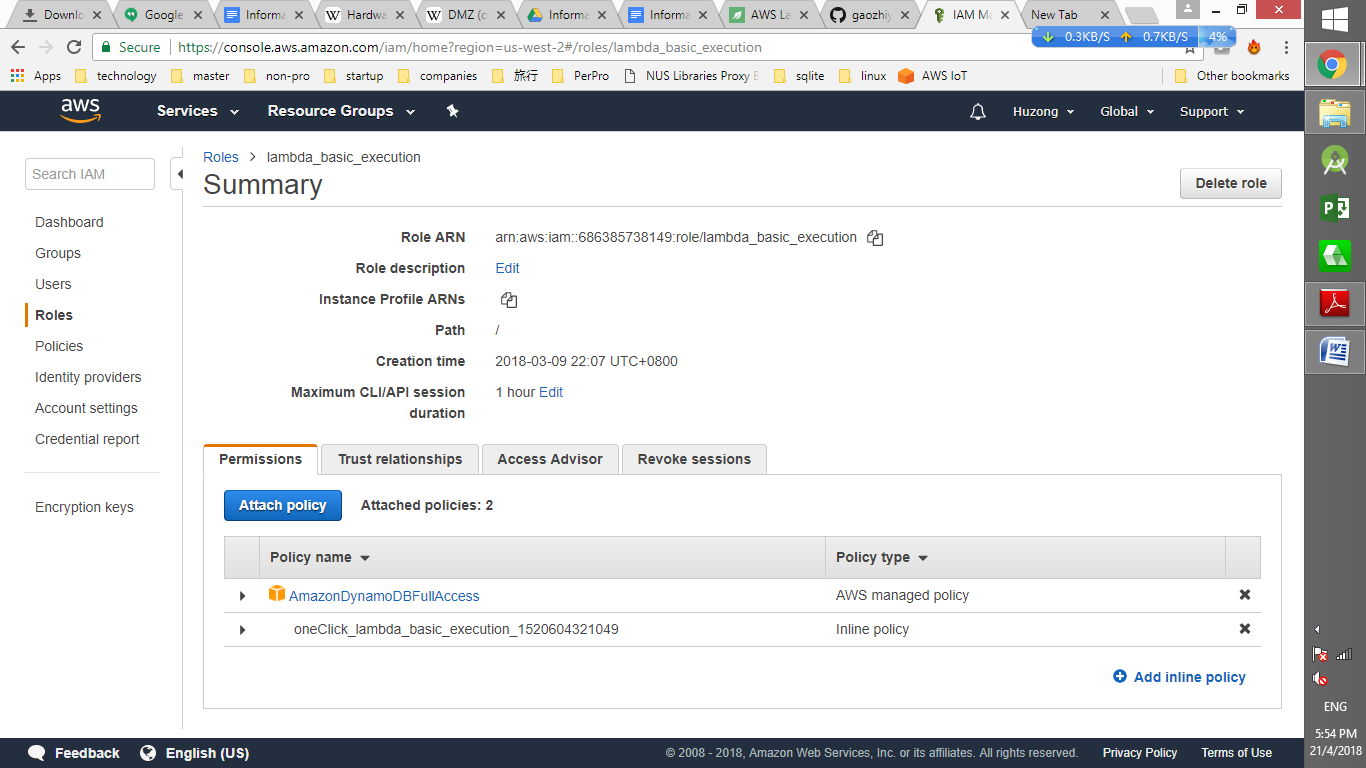
* Click “Save and test” or “Save” button
* The output can be seen on “Execution result” section:

|  |  |
| --- | --- |
|  | {    "message": "Saved Successfully!!!"  } |

* We also need to check in DynamoDB that the record is persisted:
  + Go to “DynamoDB” Management Console
  + Select the table “Person”
  + Select the “Items” tab
  + Here you can see the person’s details which were being passed in request to lambda application
* So the request is successfully processed by our lambda application

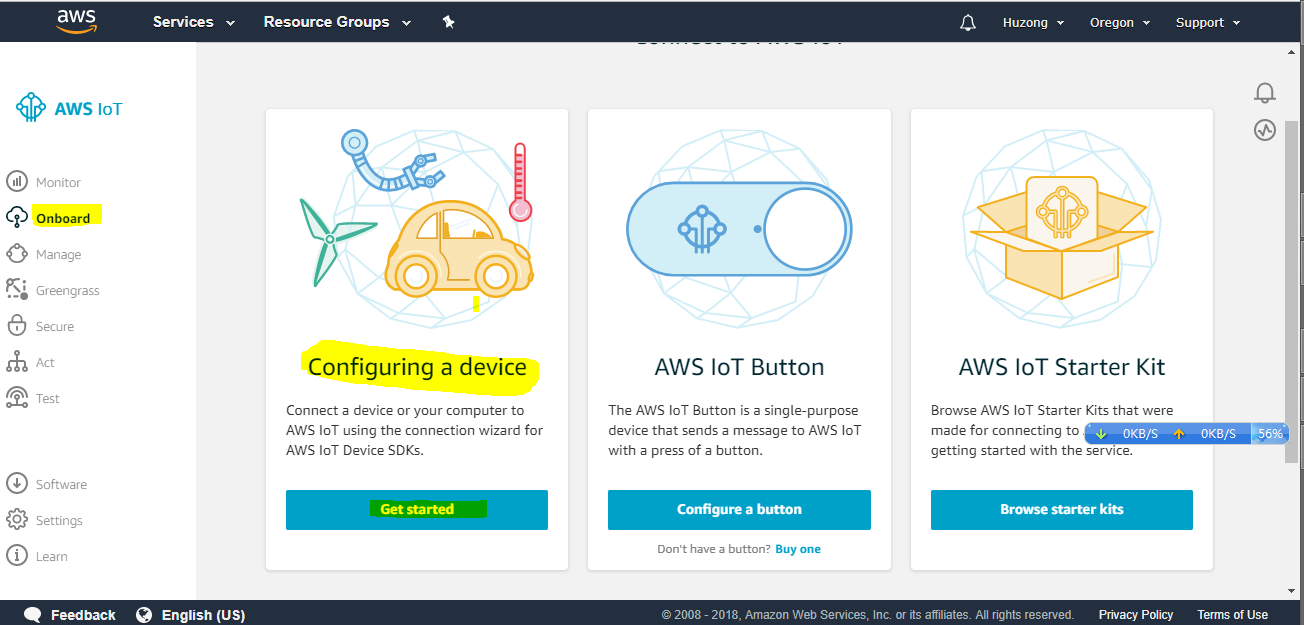
p.s:

The most tricky part is to allow the lambda to access the dynamodb function

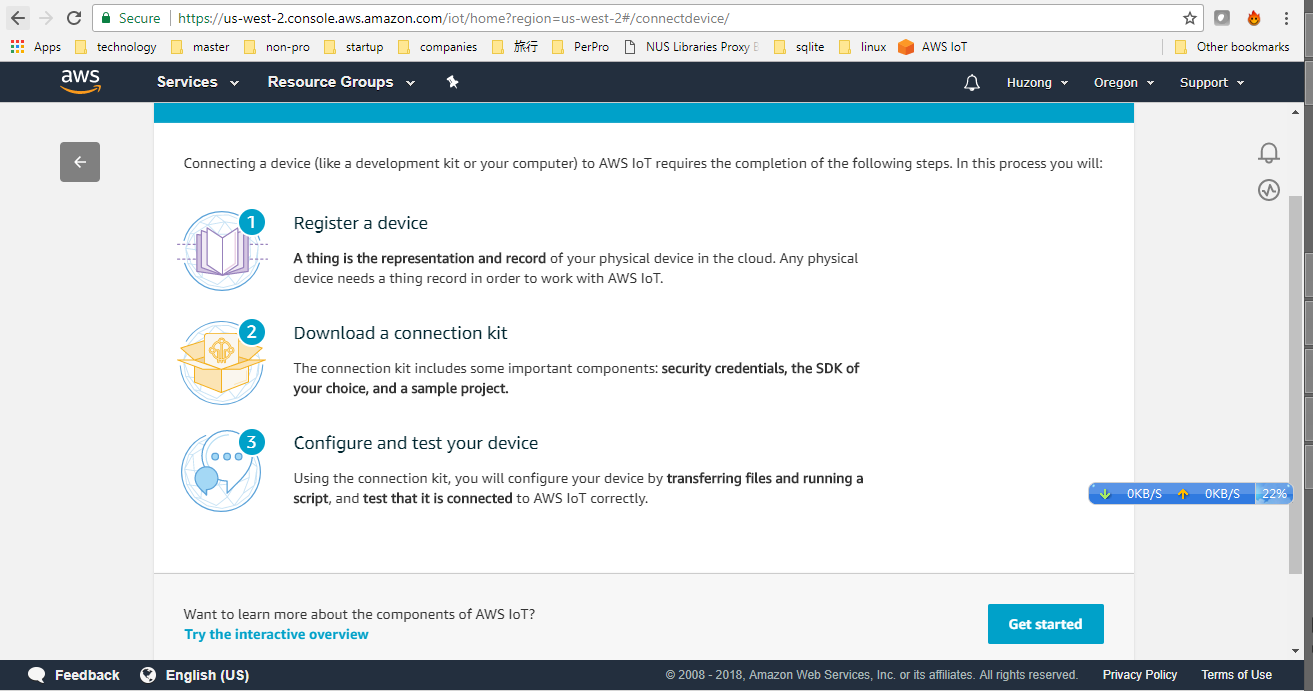


# Set up the Windows PC as the device to IOT core

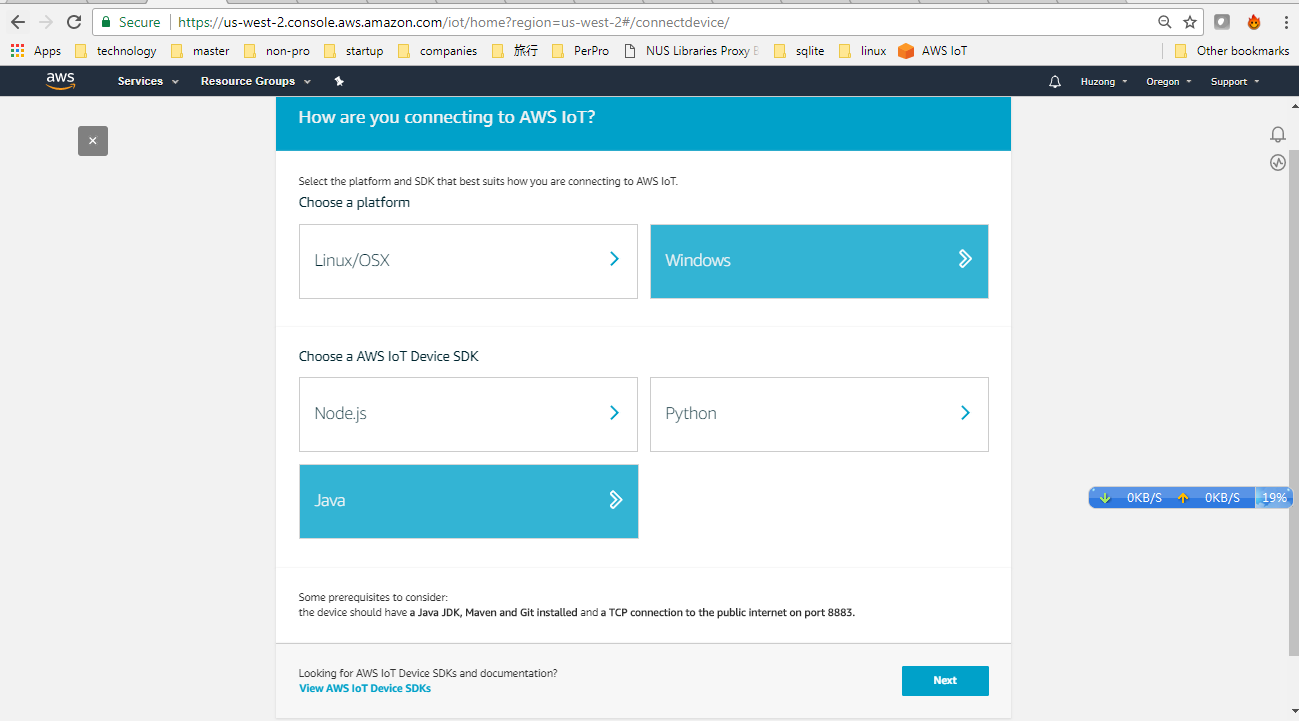
## Go to IOT core, and click the onboarding



## Click the Get Started of Configuring a Device and it will route the screen below

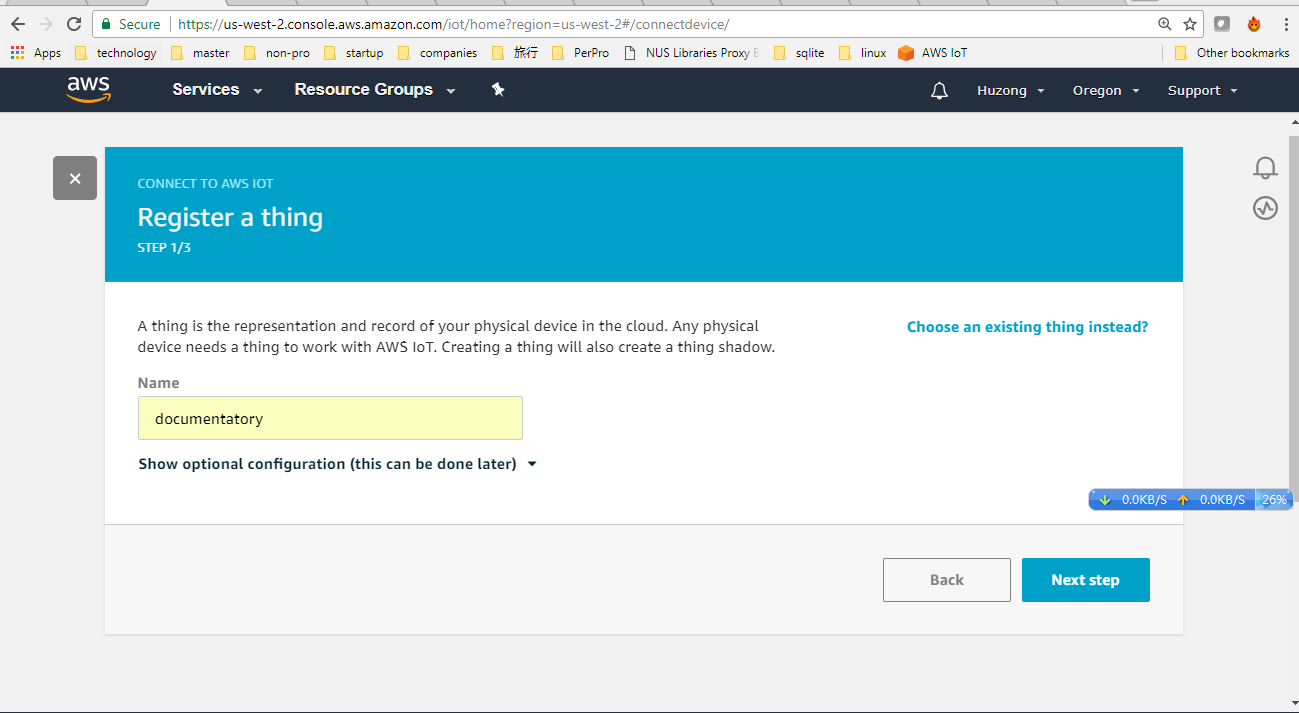


## Click the Get start and it will route the page below

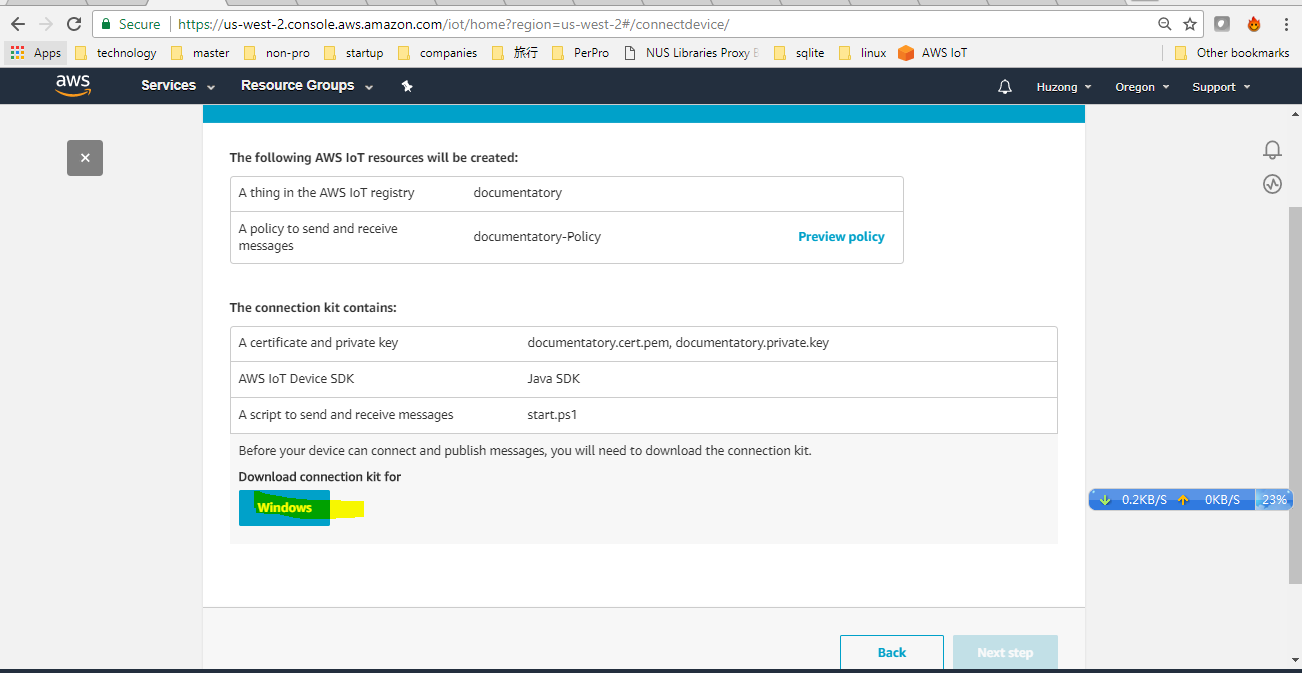


## choosing the correct OS/toolset and click the next

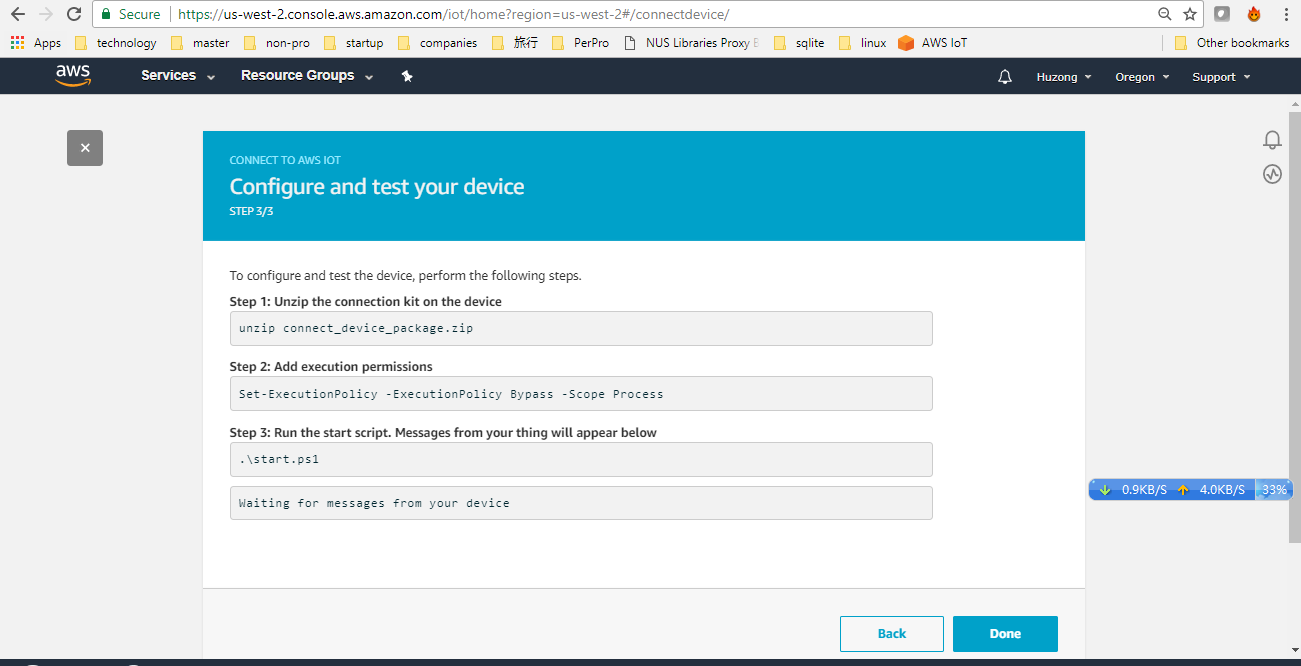
## Giving a name and click "next step"



## Click the windows to download the toolset, and next steps will become active



## After clicking next steps, it will route to the page with details text below



## To configure and test the device, perform the following steps.

**Step 1**: Unzip the connection kit on the device

unzip connect\_device\_package.zip

**Step 2**: Add execution permissions

Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope Process

**Step 3**: Run the start script. Messages from your thing will appear below

.\start.ps1

Waiting for messages from your device

p.s. *If you encounter any problem, you may use the git repository*

*https://github.com/aws/aws-iot-device-sdk-java*

## Final UI

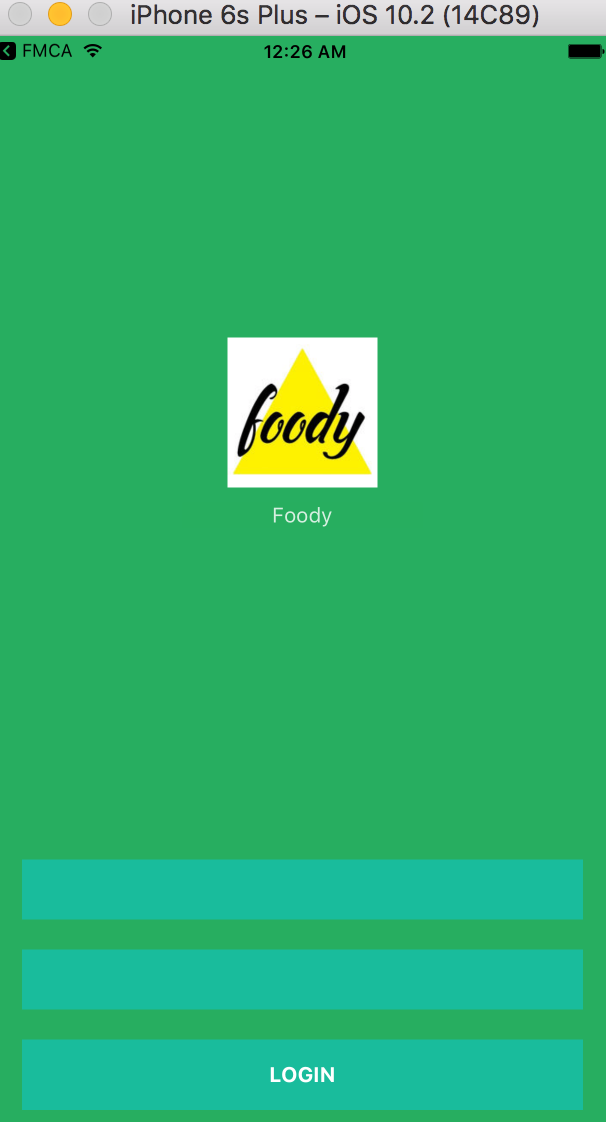


Figure : The final look of the prototype

# Conclusion of the prototype

Above prototype is done to evaluate the reactive native capabilities for our project modules. React native is good framework to create application for both android and iOS based application.

# Location Retrieval Component

## Create a new react native project

The following image shows the folder structure to retrieve the latitude and longitude of the user.

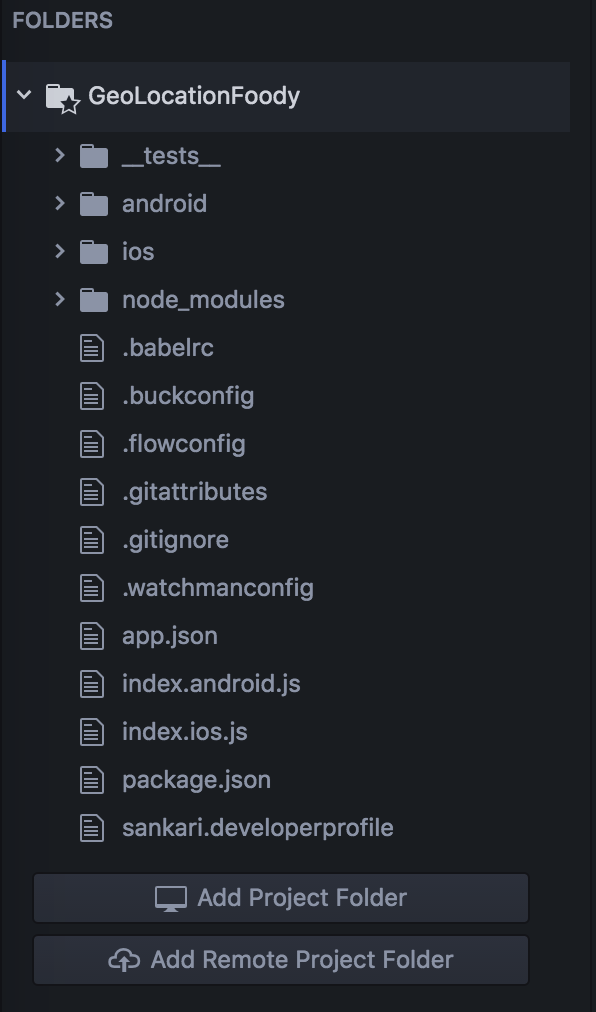


Figure 13: Initial folder structure

Run command react-native run-ios will stimulate the skeleton project in ios mode



Figure : Initial page when launching the prototype

## Location

index.ios.js file was edited with the following code snippet:



Figure 15: GeoLocation API in ios JavaScript file



Figure 16: GeoLocation API in ios JavaScript file

GeoLocation is an API provided by React Native. Using the API, we can get the latitude and longitude of the user.

The following screenshots show the popup to ask for user’s permission to retrieve the coordinates and display the same. To retrieve real time coordinates instead of predefined ones, we ran the stimulator using XCode IDE and connecting iPhone 7 Plus.

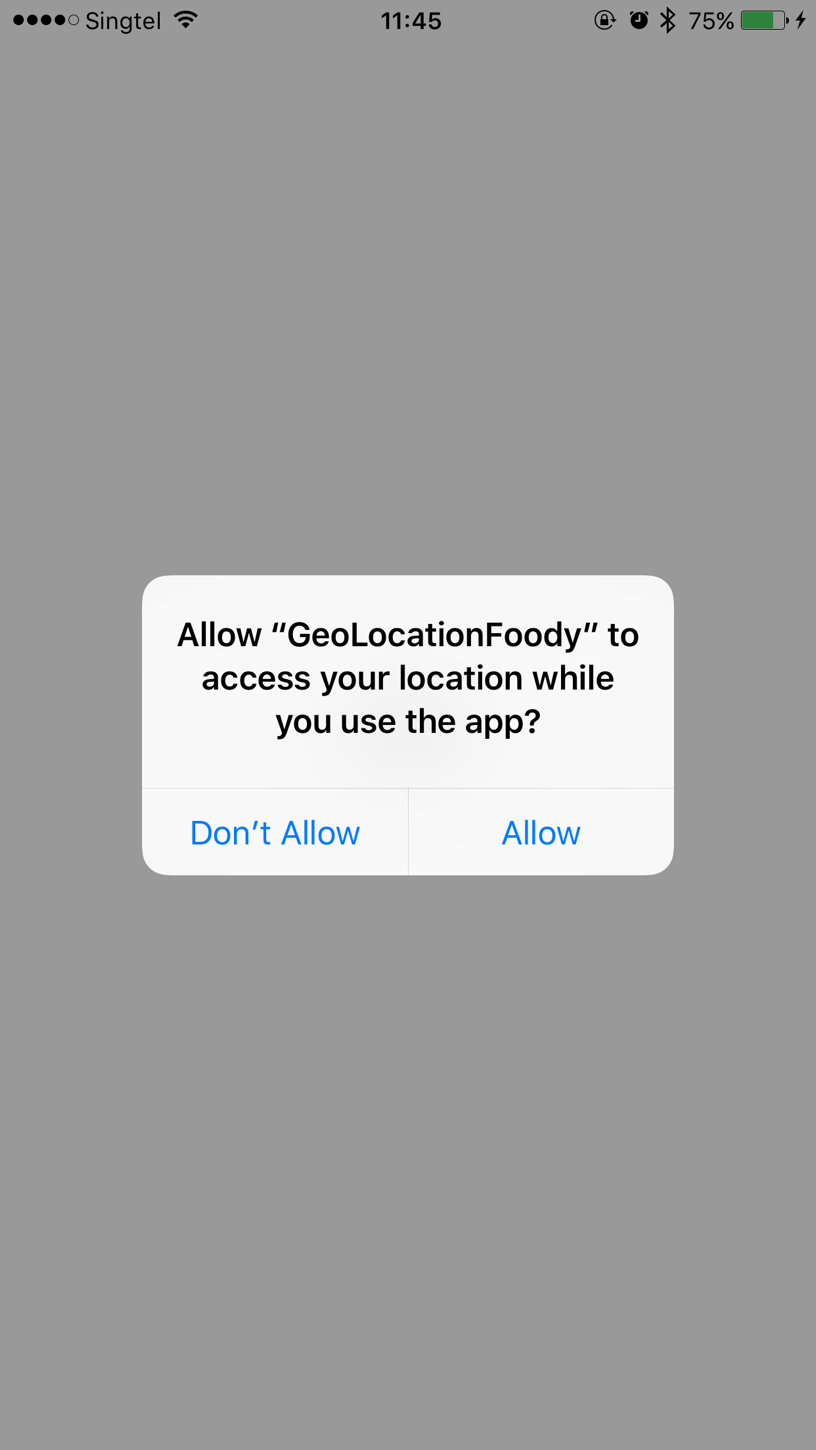


Figure 17: Application asking user for permission to use GPS on his/her mobile



Figure 18: Application displaying the Longitude and Latitude of user

## 4.3 Conclusion

Based on the prototype created above, we can retrieve the longitude and latitude of users. With these details, we can process the data and derive the address and location on a map.

1. **MQTT Evaluation:**

Secure and Reliable Communication channel between the Vendor and Customer for facilitating ordering. During the ordering process, the customer will initiate the submission of order via the Mobile application and vendor communicate or respond to the order via Vendors Web Application, since these are 2 separate applications running different platforms and technologies there is need to standardize the communication and message exchange between these 2 applications.

The below are the quality attributes pertain to the message exchange

* **Scalability**: Well defined standard to establish common technical vocabulary. When there are multiple Customers and multiple vendors, each communication needs to be unique yet based on common standard and provide loose coupling between the two applications.
* **Reliability**: The communication should be reliable and near real time to match the real-life communication as much as possible and without causing loss of any message
* **Security**: The communication should work in secured manner to avoid tempering of messages that can potentially result in Order ambiguity and loss of business.
* **Performance**: The message exchange has to be as light weight as possible to enable optimum performance over the wire transfers of messages

## 5.1 Proposed Solution

In order to address above requirement a Message broker based Queueing is proposed using MQTT based product and this section discuss about the consideration points for the same.

The below are key consideration criteria to use message broker and reduce the amount of coding required to meet above requirement.

* Message brokers natively support Publish and Subscribe style pattern implementation

This enables Lose coupling of Publisher and Subscriber.

* Message brokers provide features like Reliability via the Quality of Service Attributes (QAS)
* Scalability and Security can also be implemented easily as most messaging brokers support this via out of the box configurations.

There are many such message brokers availableand every Message Broker product implements a particular Specification, set of protocols, set of data formats, messaging styles and implementation patterns.

MQTT is one of specification which is implemented in products like HIVEMQ.

**MQTT as message broker solution – (MQ Telemetry Transport** or **Message Queue Telemetry Transport)**

The below are key consideration criteria to use MQTT based message broker as its has

* Support for low Bandwidth connection in order to facilitate light weight message packets based open standards like XML or JSON.
* Support for REST API Based solution is also available to facilitate features via REST based design and overall fit into project Architecture design.
* Well defined interface and proven and mature in IOT and instant messaging solutions e.g. Facebook Messenger

**For more information on MQTT refer to** [**http://mqtt.org/faq**](http://mqtt.org/faq)

**MQTT is also an OASIS Standard for more information refer to** [**http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf**](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf)

**Understanding Proposed Solution and its usage.**

This section will discuss at a high-level capability of MQTT and choosing a Tool or Product that implements MQTT

## 5.2 Prototyping using Proposed Solution Area

For the purpose of the prototype, we will use the HIVE MQ Public MQTT Server to test,

This comes with a free online MQTT Server/Broker and a client which uses web socket based connection to check

Step 1 -We start the POC using the free online public MQTT Server – This will mimic the MQTT Server in AWS cloud

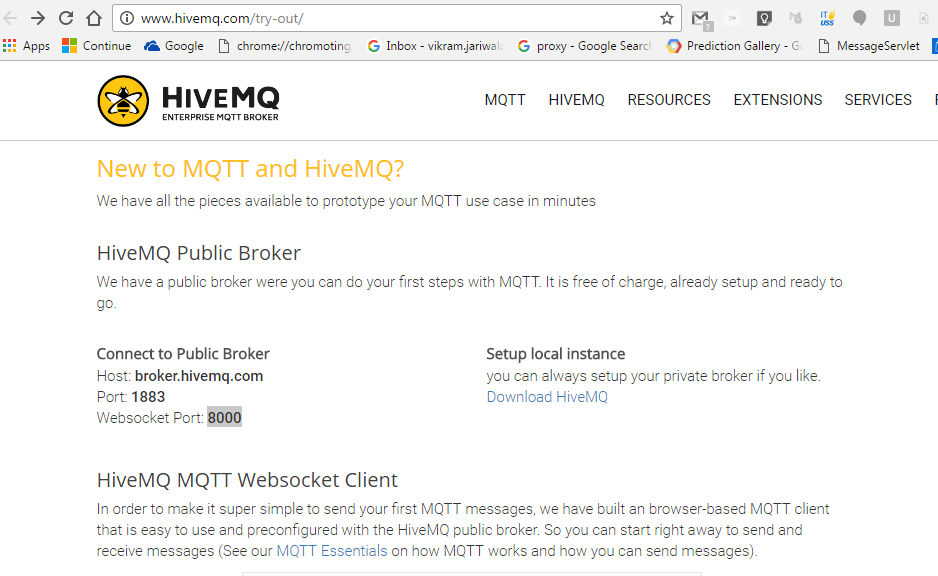


Figure 19: MQTT free server page

Step 2 – Click the Try WebSocket Client which is Java script based free MQTT Client

This will mimic Vendor Application

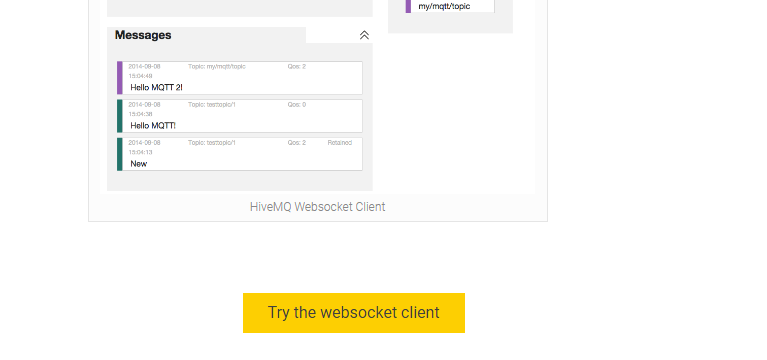


Figure 20: Picture showing the 'Try the websocket client' button

Step 3 – We will use MQTTLens – client which is a chrome plugin to mimic the Foody Mobile Client Application

Using Chrome, we will invoke the MQTTlens

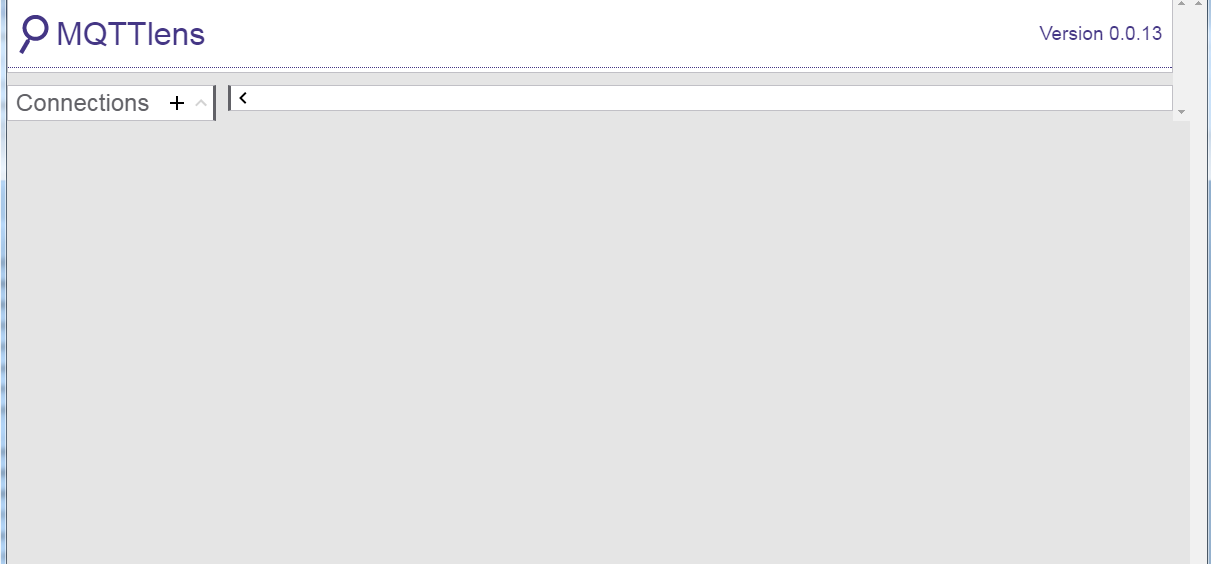


Figure 21: MQTTlens page

Step 4 : Connect to the MQTT broker

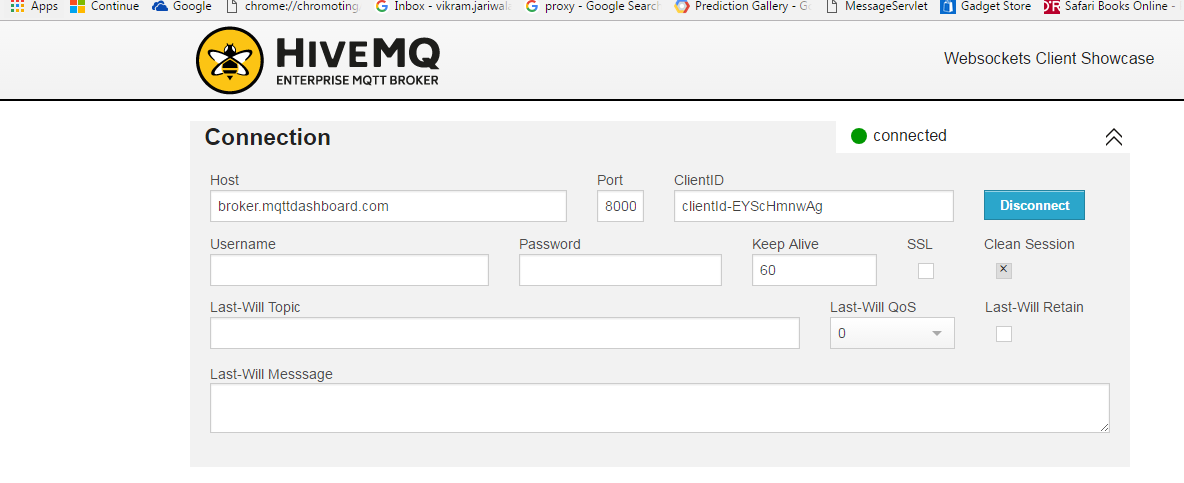


Figure 22: MQTT broker connection

Step 5: Create a topic i.e. testtopic/Aperia/ChinesefoodStal

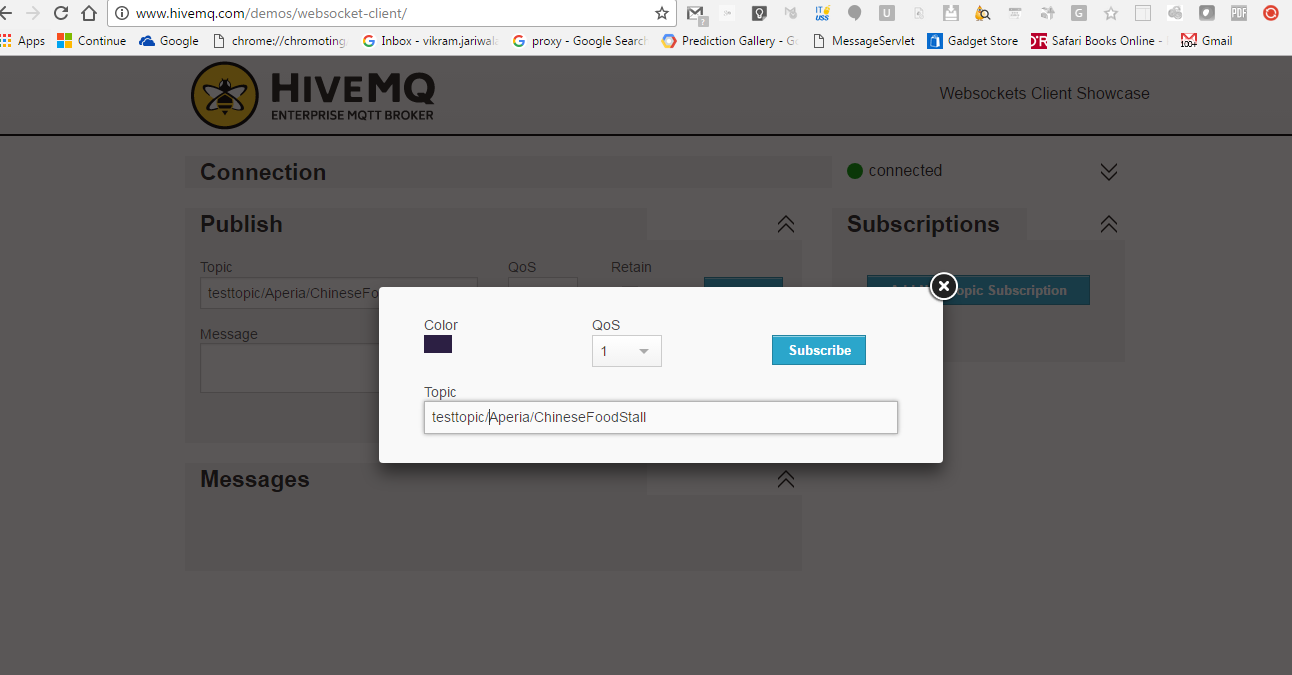


Figure 23: Creation of Topic

Step 6 : Subscribe to the topic in MQTTLens

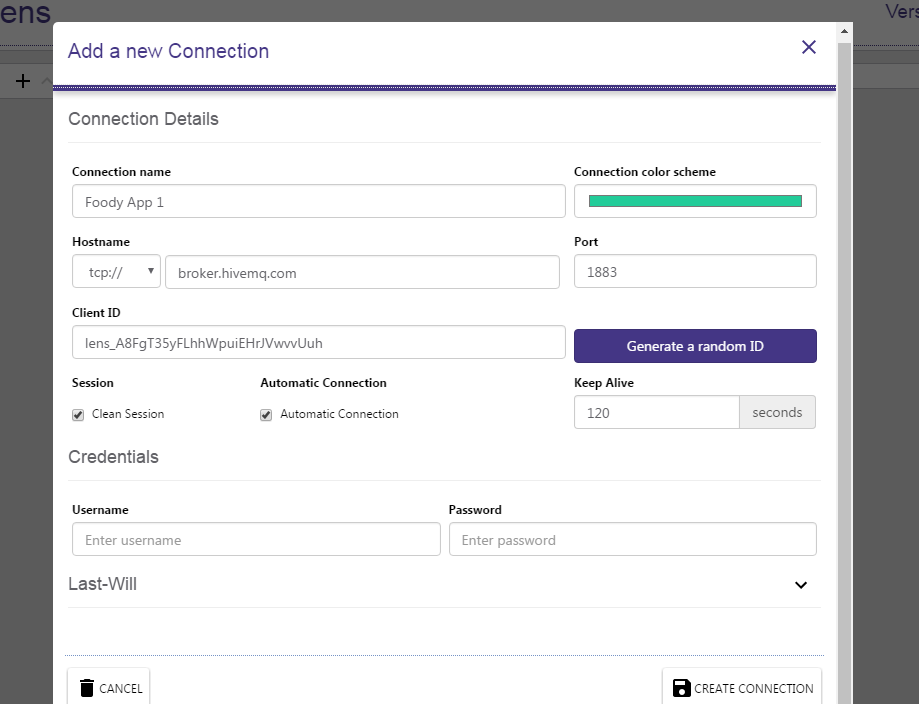


Figure 24: Subscribing to Topic

Step 7: ensure the MQTT lens is connected to topic

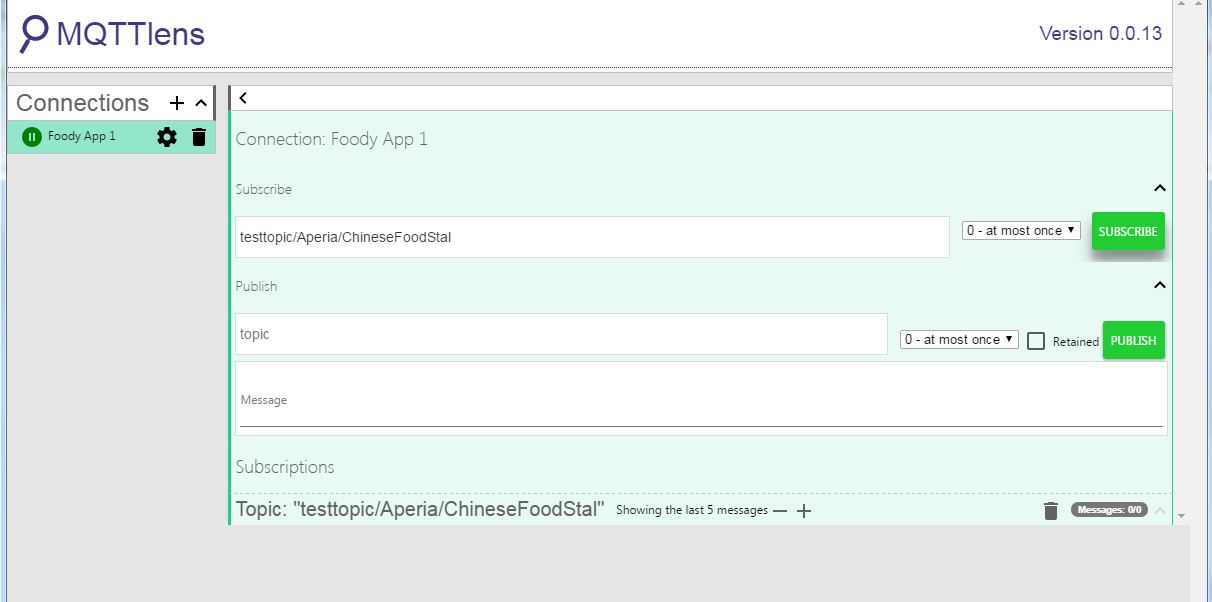


Figure 25: Topic connection

Step 8: Publish message on to the topic to mimic an order

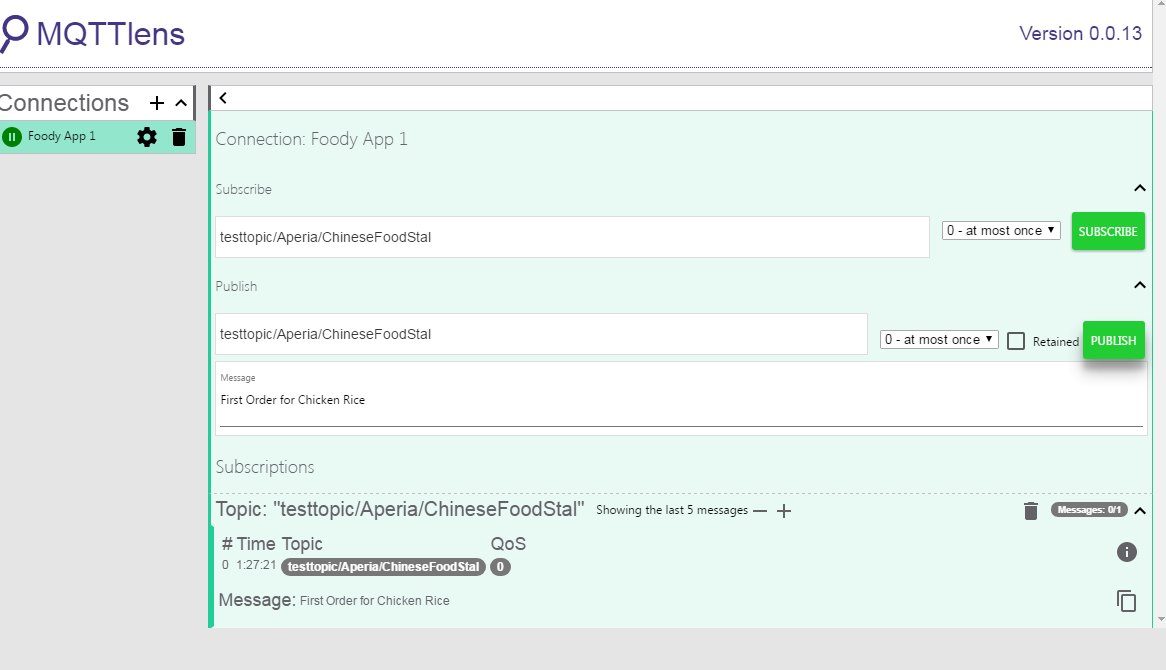


Figure 26: Publishing message on Topic

Step 9: Check the Submitted message from Foody Mobile App in the vendor module on the webSocket Client

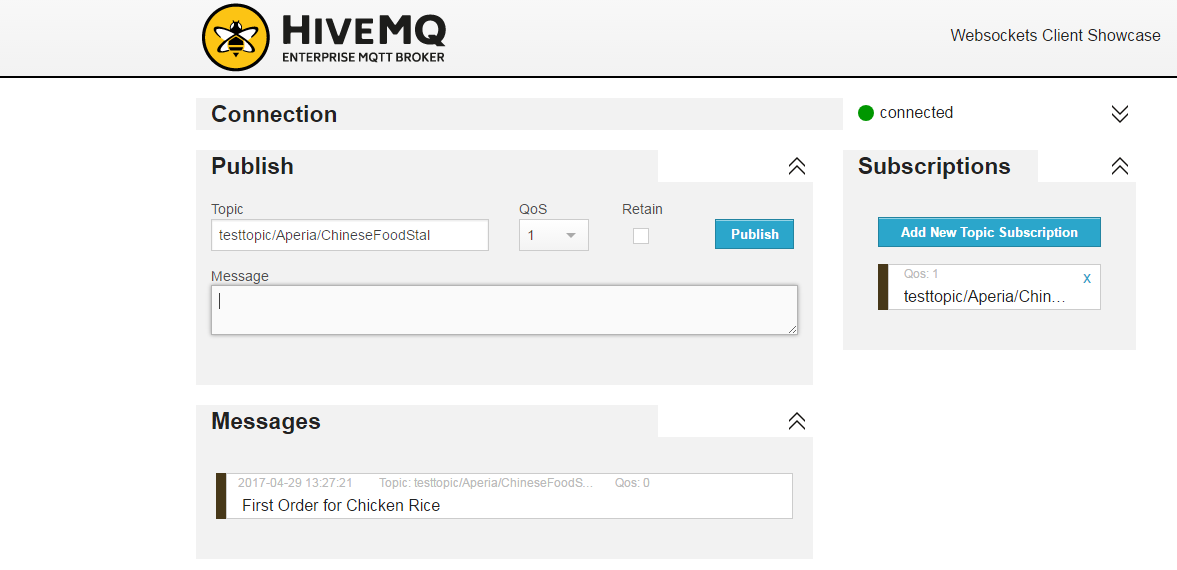


Figure 27: Submitted Message

Step 10: Respond to the Message with confirmation via publishing a message via the web socket client

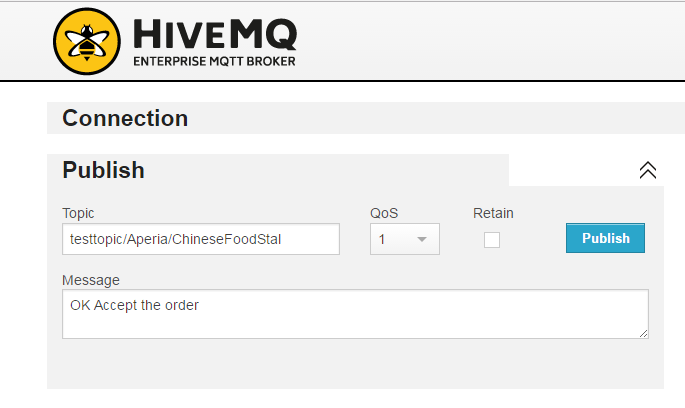


Figure 28: Response to the Message

Step 11: Check the message of acceptance is show in the MQTTLens and two-way communication is established.

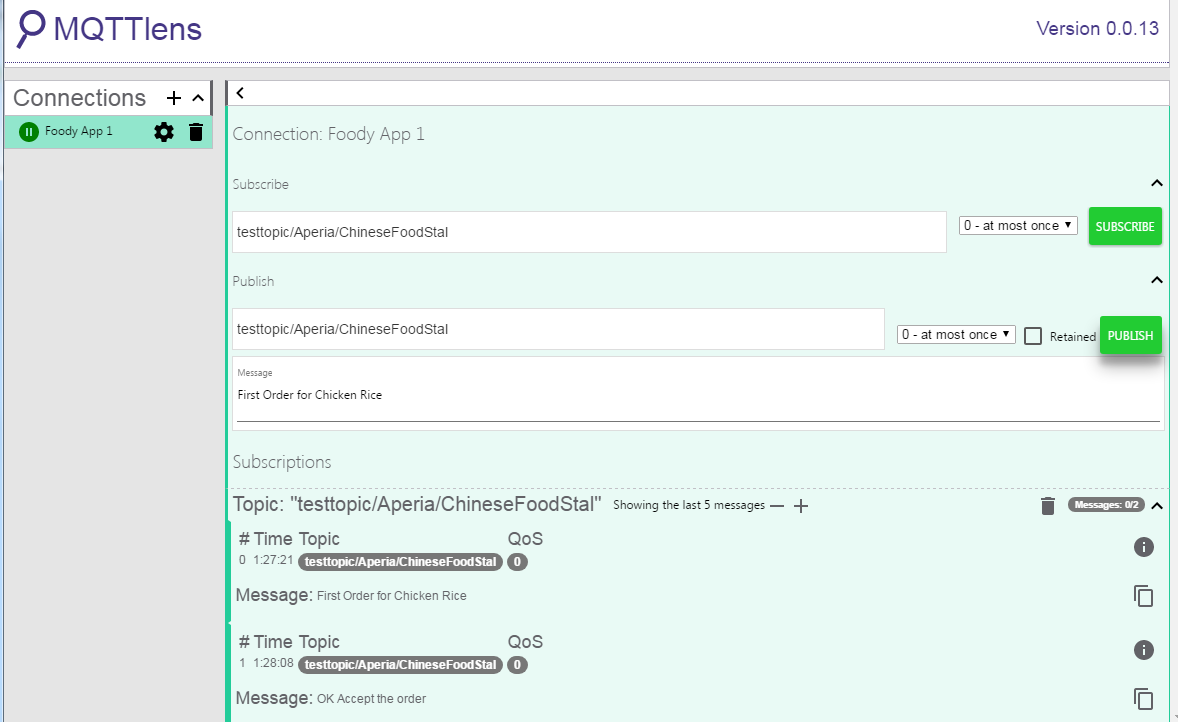


Figure 29: Messages summary

## 5.3 Conclusion /Outcomes

MQTT is an easy to use and scalable message broker which can be used in Foody application to establish secure, reliable and scalable and near real time communication between foody Mobile Client and Vendor Application

The Design of the queues will require discussion and alignment with Vendor Application and vendor will have to be made owner of the queues.

The QOS Attributes were not explored in the prototype due to lack of time ,however the basic option QOS=1 is suitable , however for the advance scenarios a further study or discussion with other QOS attributes i.e. .0,1,2 should be done to select the most suitable.

1. **Firebase Could Messaging**

Step 1. Create a developer account and signing certificate in Apple

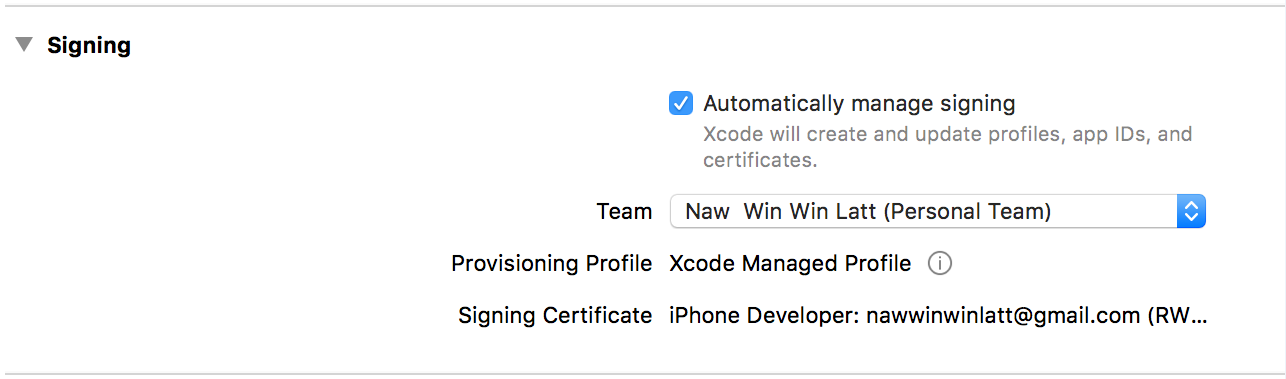


Figure 30: Creation of developer account

1. Create signing certificate in Keychain App
2. Create Push Notification Certificate in Apple Developer portal by using signing certificate
3. Download and install this certificate
4. Export this Push Notification Certificate into P12 file format

Step 2. Create a project in Firebase

1. Sign in with google account to <http://firebase.google.com>
2. Create new project
3. From project app settings, upload the certificate created in step 1.

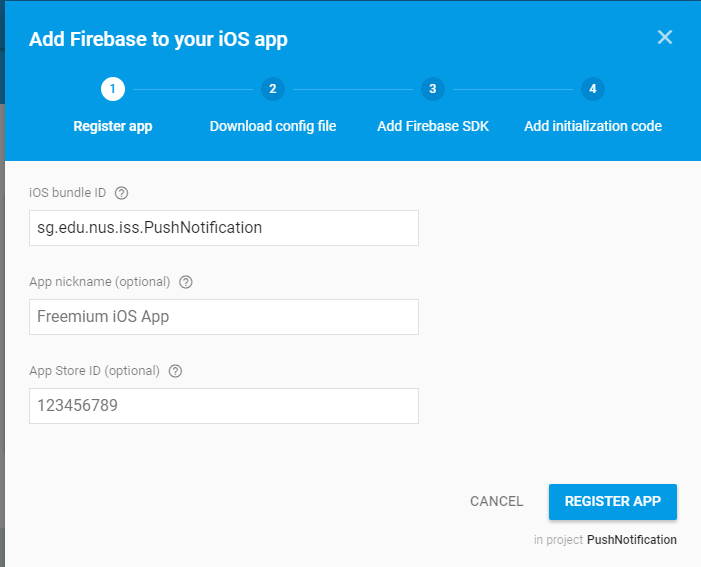


Figure 31: Creation of Firebase for iOS application

Step 3. Import SDK to Project

1. Download SDK from <https://dl.google.com/firebase/sdk/ios/3_17_0/Firebase-3.17.0.zip>
2. Import SDK of Firebase Analytics and Firebase Messaging
3. Download GoogleService-Info.plist and copy into the project

Step 4. Initialisation SDK to AppDelegate files

#import <UIKit/UIKit.h>

@import UserNotifications;

@interface AppDelegate : UIResponder <UIApplicationDelegate, UNUserNotificationCenterDelegate>

@property (nonatomic, strong) UIWindow \*window;

@end

/\*\*

\* Copyright (c) 2015-present, Facebook, Inc.

\* All rights reserved.

\*

\* This source code is licensed under the BSD-style license found in the

\* LICENSE file in the root directory of this source tree. An additional grant

\* of patent rights can be found in the PATENTS file in the same directory.

\*/

#import "AppDelegate.h"

#import <React/RCTBundleURLProvider.h>

#import <React/RCTRootView.h>

#import "RNFIRMessaging.h"

@implementation AppDelegate

- (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions

{

NSURL \*jsCodeLocation;

jsCodeLocation = [[RCTBundleURLProvider sharedSettings] [jsBundleURLForBundleRoot:@"index.ios](mailto:jsBundleURLForBundleRoot:@%22index.ios)" fallbackResource:nil];

RCTRootView \*rootView = [[RCTRootView alloc] initWithBundleURL:jsCodeLocation

moduleName:@"ChatApp"

initialProperties:nil

launchOptions:launchOptions];

rootView.backgroundColor = [[UIColor alloc] initWithRed:1.0f green:1.0f blue:1.0f alpha:1];

self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];

UIViewController \*rootViewController = [UIViewController new];

rootViewController.view = rootView;

self.window.rootViewController = rootViewController;

[self.window makeKeyAndVisible];

[FIRApp configure];

[[UNUserNotificationCenter currentNotificationCenter] setDelegate:self];

return YES;

}

- (void)userNotificationCenter:(UNUserNotificationCenter \*)center willPresentNotification:(UNNotification \*)notification withCompletionHandler:(void (^)(UNNotificationPresentationOptions))completionHandler

{

[RNFIRMessaging willPresentNotification:notification withCompletionHandler:completionHandler];

}

- (void)userNotificationCenter:(UNUserNotificationCenter \*)center didReceiveNotificationResponse:(UNNotificationResponse \*)response withCompletionHandler:(void (^)())completionHandler

{

[RNFIRMessaging didReceiveNotificationResponse:response withCompletionHandler:completionHandler];

}

//You can skip this method if you don't want to use local notification

-(void)application:(UIApplication \*)application didReceiveLocalNotification:(UILocalNotification \*)notification {

[RNFIRMessaging didReceiveLocalNotification:notification];

}

- (void)application:(UIApplication \*)application didReceiveRemoteNotification:(nonnull NSDictionary \*)userInfo fetchCompletionHandler:(nonnull void (^)(UIBackgroundFetchResult))completionHandler{

[RNFIRMessaging didReceiveRemoteNotification:userInfo fetchCompletionHandler:completionHandler];

}

@end

Step 5. Listen Push Notification

// src/actions/index.js

import FCM, { FCMEvent,

NotificationType,

WillPresentNotificationResult,

RemoteNotificationResult } from 'react-native-fcm';

import { Platform } from 'react-native';

// ...

const startChatting = function (dispatch) {

dispatch(userAuthorized());

dispatch(fetchMessages());

FCM.requestPermissions();

FCM.getFCMToken()

.then(token => {

console.log(token)

});

FCM.subscribeToTopic('secret-chatroom');

FCM.on(FCMEvent.Notification, async (notif) => {

console.log(notif);

if (Platform.OS === 'ios') {

switch (notif.\_notificationType) {

case NotificationType.Remote:

notif.finish(RemoteNotificationResult.NewData); //other types available: RemoteNotificationResult.NewData, RemoteNotificationResult.ResultFailed

break;

case NotificationType.NotificationResponse:

notif.finish();

break;

case NotificationType.WillPresent:

notif.finish(WillPresentNotificationResult.All); //other types available: WillPresentNotificationResult.None

break;

}

}

});

FCM.on(FCMEvent.RefreshToken, token => {

console.log(token);

});

}

# 7 Facebook Authentication

With Facebook SDK integrated into React Native, we can use Facebook Login button object which is in the SDK for Authentication.



Figure 32: FB login button example code snippet

From the code snippet above, we can see that Login Interface is handled by Facebook SDK.

With React Native’s built in API, user’s friend list can be retrieved for FMCA.

From <https://developers.facebook.com/docs/graph-api/reference/friend-list/>, reference, we can see that a user’s friend list can be retrieved using an API call as shown below from the same webpage.

/\* make the API call \*/

FB.api(

"/{friend-list-id}",

function (response) {

if (response && !response.error) {

/\* handle the result \*/

}

}

);

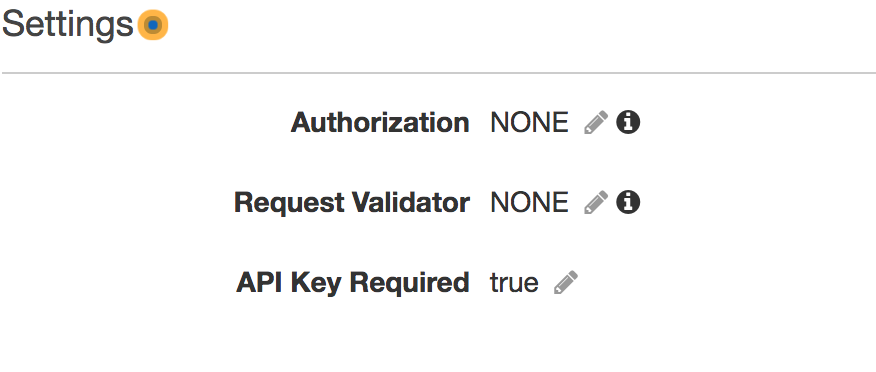
## 7.1 Conclusion

We can use the Facebook SDK for integrating FB authentication

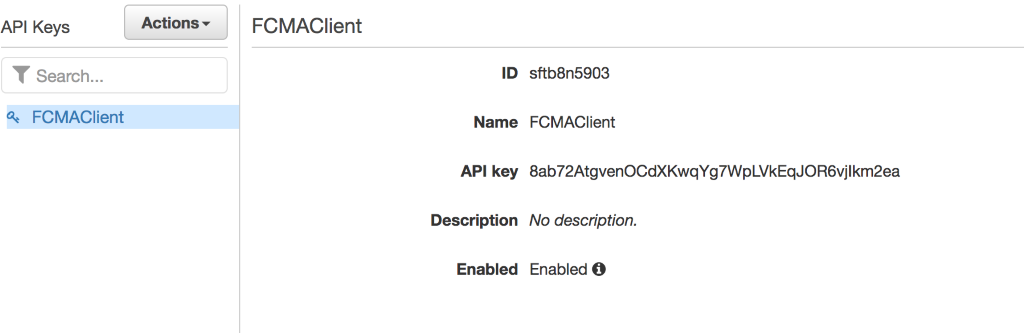
# 8 Calling AWS lambda from react native

AWS lambda function will be exposed via APIGetway as RESTFul service. By using react native built in fetch function with AWS api key will be able to call the service successfully.

API gateway will be configured to validation API Key:



Generate API Key



React native code. Uses fetch method to make http call, include the x-api-key generated above in every http request



## 8.1 Conclusion

Calling lambda function via APIGetway has no differernt than calling other http service. As long as the APIGetway is API Key enabled and correct API key is included in http request header will do.