Carma iOS Integration Guide.

Rev pA8.

Contents

About this document

This document describes how to integrate Carma with Apple's Push Notification Service (APNS) for iOS.

Change notes:

pA1	first version
pA2	changed the URL and payload layout for device registration.
pA3	Updated example code
pA4	updated authentication methodology
pA5	new authentication method for Carma and APNS
pA6	Added info on supplying apptype in registration
pA7	Added sandbox setting for testing through xcode.
pA8	updated with correct URL for registration.

How Apple's push notification service works

Apple's system considers 3 parties:

- The client application
- Apple's push notification service
- a 3rd party server, in this case Carma.

When a user installs a push-enabled application and starts it for the first time, it ask the user for permission of receiving push notifications. Once the user agrees, as soon as you decide to send your users a notification, a succession of actions will be triggered:

- Carma sends the push notification to Apple's push notification servers.
- These servers relay the message to all your registered mobile applications.
- Messages are queued and stored for devices that are offline.
- As soon as a device comes back online, APNS servers relay the queued message.
- The messages are received and presented according to the platform-specific implementation.

Getting started

There are four tasks must be performed in order to send and receive a push notification:

1. **Apple Configuration**. The app must be configured properly and registered with the Apple Push Notification Service (APNS) to receive push notifications upon every start-up.
2. **Carma Registration.** The app is registered in Carma using the appName, App id and either signing certificate or development and production certificate.
3. **Push Send in Carma**. Carma must send a push notification to APNS directed to one or more specific devices. This process is not covered in this document. Please check http://expertise.carmamarketinghub.com/edit-content/edit-email-or-app-push-content/create-apppush/ for more information.
4. **Consume Push in iOS.** The app must receive the push notification; it can then perform tasks or handle user actions using callbacks in the application delegate.

Apple Configuration.

Before you begin, you will need to have an iOS app to which you want to send push notifications. This push notifications tutorial assumes that Xcode 8.3.2 or later is used as the IDE.

The process of sending push notification is described in the xcode online manual here:

http://help.apple.com/xcode/mac/current/#/dev11b059073

Enabling the Push Notification Service

The first step is to change the App ID. In Xcode, go to **App Settings -> General** and change Bundle Identifier to something unique. Next, you need to create an App ID in your developer account that has the push notification entitlement enabled. Luckily, Xcode has a simple way to do this. Go to **App Settings -> Capabilities** and flip the switch for Push Notifications to On.
Behind the scenes, this creates the App ID in your member center if it does not exist already, and then adds the push notifications entitlement to it. If any issues occur, or if you prefer, manually create the App ID or add the push notifications entitlement in the member center by using the + or Edit buttons.

Carma Registration

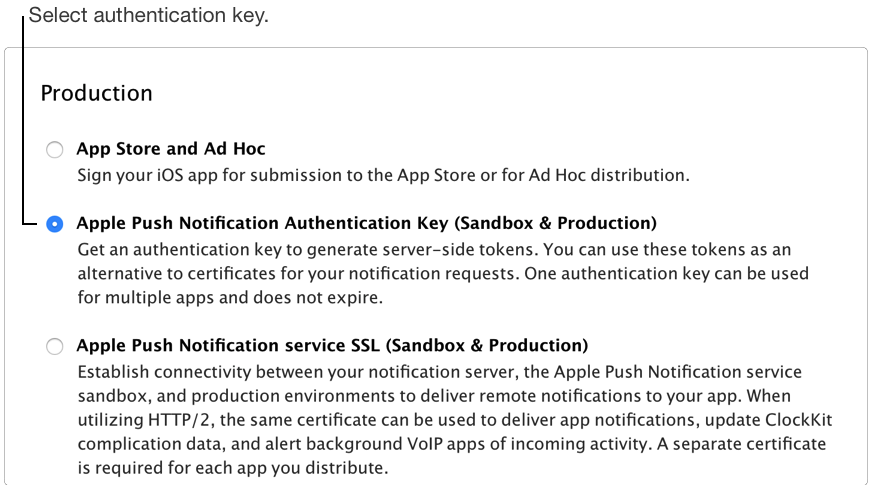
Before Carma can send notifications to your app, the app needs to be registered in Carma together with the certificate for that app. Apple has two methods for signing push requests, either with two certificates, *Develop* and *Production* or with a single *Signing Certificate* that doesn't expire.

The Signing Certificate is the preferred method, and the develop/production certificate method will probably be deprecated soon so we will only cover the token method in this document.

Generate a universal provider token signing key

You can now use token-based authentication based on a signing certificate to communicate with the APNs instead of multiple certificates that are time limited. You can get one authentication key to sign tokens for multiple apps. The authentication key doesn't expire but can be revoked.

1. In your developer account, go to Certificates, Identifiers & Profiles, and under Certificates, select All or APNs Auth Key.
2. Click the Add button (+) in the upper-right corner.
3. Under Production, select the "Apple Push Notification Authentication Key (Sandbox & Production)" checkbox, and click Continue.



4. Click Download.

Important: Save a back up of your key in a secure place. It will not be presented again and cannot be retrieved later.

Team and Key Id

You will also have to retrieve you Team Id and your Key Id.

You will find the *Team Id* by login onto your iOS developer account and clicking Membership. https://developer.apple.com/account/#/membership/

The *Team Id* has the following form: M4DK7XT1245

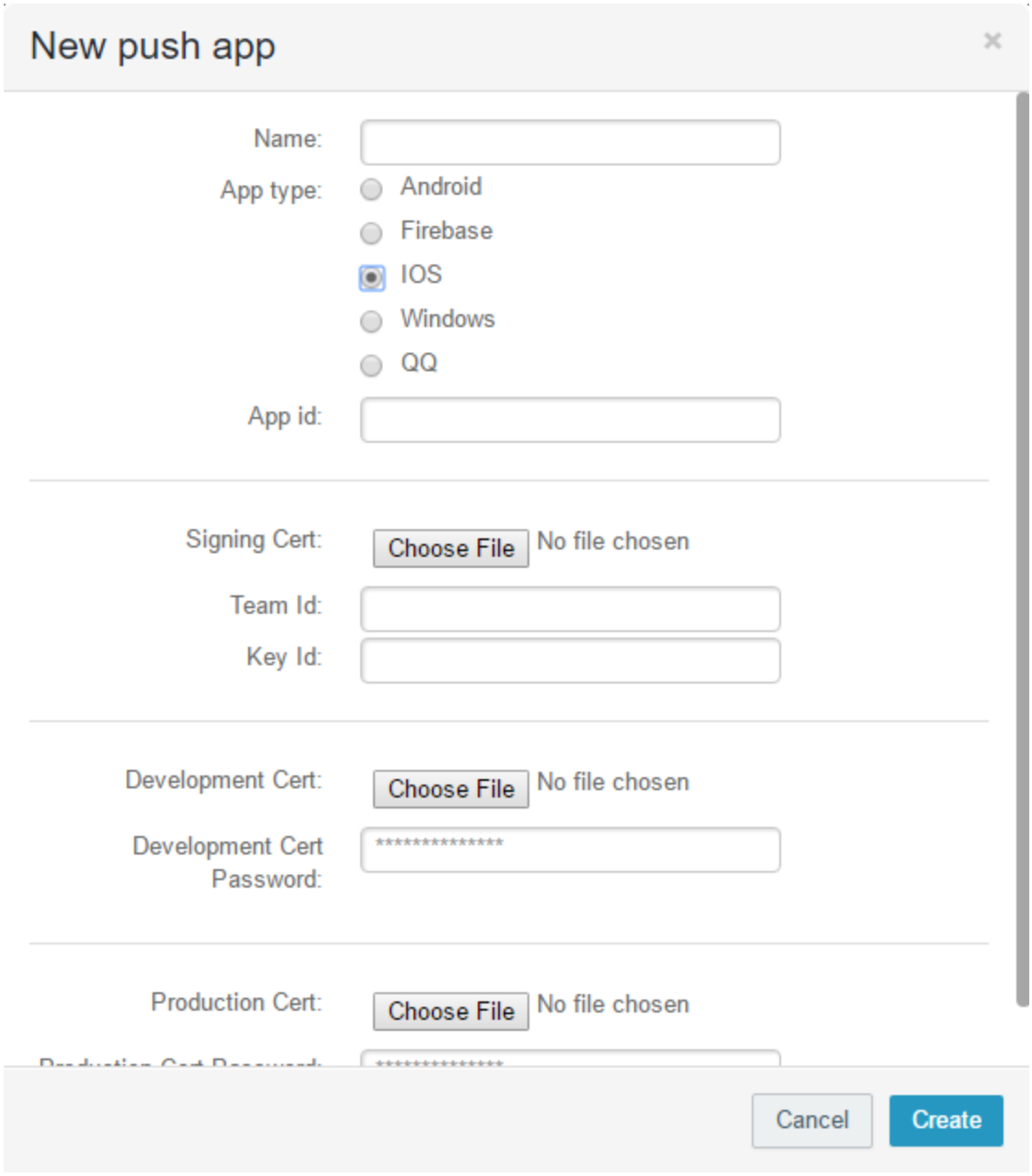
The *Key Id* is found in the APNS Auth Key section:
https://developer.apple.com/account/ios/certificate/key

The *Key Id* has the following form: F5HAL12345

Setup App in Carma:

Click on **A*ccount Setting*** in the left hand menu and then choose *Push Apps* under *Manage*

Click **Create new,** check the iOS radio button and you will see the following dialog where you enter the information



Fill in the fields with name, appId and attach the Signing certificate file that you produced earlier. You will not have to enter Development or production cert if you use a Signing certificate.

Carma is now ready to start sending notifications to your application.

Registering the token in Carma

Before you start coding you will need to find four pieces of data that are used in all interactions with Carma.

The first is the BundleId i.e. `com.yourcompany.yourappname (appName)`

To find the other three you will need to logon to Carma. The data is:

- REST URL (`restUrl`)
- Carma API Key. (`carmaAccessToken`)
- Customer ID (`customerId`)

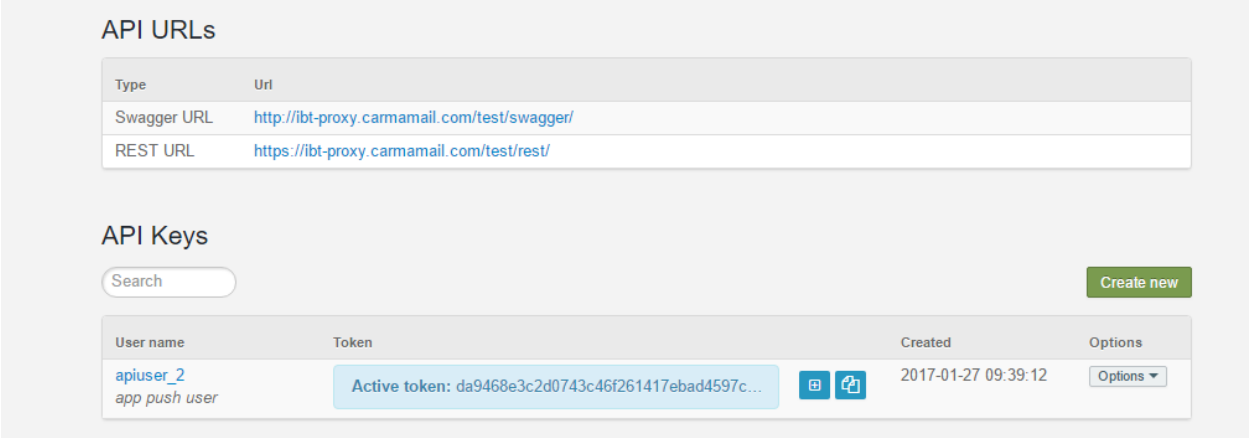
The URL will then be created like this:

```
let urlString = "\(restUrl)/\(customerId)/apps/\(appName)/devices"
let headers: HTTPHeaders = [
        "Accept": "application/json",
        "Content-Type": "application/json",
        "X-Carma-Authentication-Token": "\(carmaAccessToken)"
```

The first two can be found in Carma under Account Settings-> Carma API

(https://web-ibt-test.carmamail.com/carma/sv-se/carmaapi)

If you haven't created an API key before, click the *Create New* button to create one now.



The customer ID is found on the bottom of the main menu in Carma.

The usage of this information is described in a chapter below.

Consume Push in iOS

The three steps of Push Notifications registrations

There are three steps to register for push notifications.

1. You must obtain the user's permission to show any kind of notification, after which you can register for remote notifications.
2. If all goes well, the system will then provide you with a device token, which you can think of as an "address" to this device.
3. The application needs to register the device token with Carma's servers.

Obtaining User Permissions in Swift 3 for iOS.

How to obtain user Permissions depends on which version of iOS you are developing for.iOS 10 and later uses a notification center and earlier versions uses a more basic method.

In your AppDelegate.swift class first import the notification center

```
import UserNotifications
```

Then have AppDelegate implement the UNUserNotificationCenterDelegate protocol:

```
class AppDelegate: /*Some other protocols I am extending...*/,
UNUserNotificationCenterDelegate {
```

To register the app for push, declare the following function:

```
func registerForPushNotifications(application: UIApplication) {

    if #available(iOS 10.0, *){
        UNUserNotificationCenter.currentNotificationCenter().delegate = self

UNUserNotificationCenter.currentNotificationCenter().requestAuthorizationWithOptions(
[.Badge, .Sound, .Alert], completionHandler: {(granted, error) in
            if (granted)
            {
                UIApplication.sharedApplication().registerForRemoteNotifications()
            }
            else{
                //Do stuff if unsuccessful...
            }
        })
    }

    else{ //If user is not on iOS 10 use the old methods we've been using
        let notificationSettings = UIUserNotificationSettings(
            forTypes: [.Badge, .Sound, .Alert], categories: nil)
        application.registerUserNotificationSettings(notificationSettings)

    }

}
```

Then call this function from didFinishLaunchingWithOptions

```
func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
...
    registerForPushNotifications(application)
...
}
```

Receive Token from APNS

If all goes well you should receive a token from APNS by implementing the following method in AppDelegate:

```
func application(_ application: UIApplication,
didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {

      // remove white spaces from token.
        let tokenString = deviceToken.reduce("", {$0 + String(format: "%02X", $1)})
        … register token with Carma.
    }
```

The registration might fail, so you should also implement the following app delegate function for this condition:

```
func application(application: UIApplication,
didFailToRegisterForRemoteNotificationsWithError error: NSError) {
 print("Failed to register:", error)
 …
}
```

Store token and extra information in Carma.

You communicate with Carma using standard REST calls with a JSON payload. As iOS have no built-in mechanism for either REST or JSON, you may choose any implementation for doing this. To provide an example, this document uses the **AlamoFire** for REST calls and **SwiftlyJSON** for the JSON implementation

Sandbox vs Production

Apple has two environments for sending notifications; *Sandbox (* or *Develop)* and *Production*.

Sandbox is used when you are debugging your App in XCODE using a device connected to your Mac.

Production is used when the app is compiled and uploaded to itunes connect. **NOTE!** Even if you are using testflight, this counts as production.

This means that a token that has been received and registered in Carma in a sandbox environment will be invalid for the app in the production environment and vice versa. So if you send a notification to *production*, but the token is registered in *sandbox*, the token/device will be marked as invalid.

This could be really bad when you go live and someone happens to send a sendout using the sandbox.

To fix this you should add a field to the device registration payload that tells Carma if you are debugging or not. You do this with

`"sandboxToken":"true|false"`

To tell if you are using the debugger or not you can use the following Swift 3 code:

```
   func isDevelopmentEnvironment() -> Bool {
        guard let filePath = Bundle.main.path(forResource: "embedded",
ofType:"mobileprovision") else {
            return false
        }
        do {
            let url = URL(fileURLWithPath: filePath)
            let data = try Data(contentsOf: url)
            guard let string = String(data: data, encoding: .ascii) else {
                return false
            }
            if
string.contains("<key>aps-environment</key>\n\t\t<string>development</string>
") {
                return true
            }
        } catch {}
        return false
}
```

NOTE! The sandboxToken field defaults to false if you do not add it explicitly.

Carma Authentication

You authenticate with carma by setting a request header in the rest call:

```
"X-Carma-Authentication-Token": "thetoken"
```

you also need to set the *Content-Type* and *Accept* headers to "*application/json"*

Carma device token registration

Once a device has registered with APNS successfully, the didRegisterForRemoteNotificationsWithDeviceToken method in the app delegate will be called with the token.

This token needs to be registered in Carma along with the iOS bundle Id (called appId in Carma) and an originalid of you choosing. The originalid is an identifier of the user in you system. It could be an email address or a primary key of the user in your database.

NOTE! You can have many tokens registered for the same originalId, i.e. a user can use the app in an iPad and an iPhone at the same time and have a notification sent to both.

This is done by issuing a PUT request to:

https://<serverUrl>/rest/<**customerid**>/apps/**<appid>**/pushdevices

with a json payload containing at the minimum:

- deviceToken
- originalId

NOTE! Even if they are the same both values must be provided.

A minimal version of the payload would look like this:

```
{
 "originalId": "abcabc123123",
 "deviceInfo": [{"devicetoken": "abc123abc123kmlkmlkml",
                 "sandboxToken": "false" }]
}
```

NOTE! If you use the same appId for apps of different platforms e.g. com.company.name for both iOS and Android, you must add the appType to the token.
Valid values for appType are:

- 1 : iOS
- 2: Android
- 3: Windows
- 6: Firebase

The payload will then be:

```
{
 "originalId": "abcabc123123",
```

```
 "deviceInfo": [{"devicetoken": "abc123abc123kmlkmlkml", "appType": "1",
“sandboxToken”: "false"}]
}
```

In this payload you can also add other information about the user that should be transferred to Carma. Below is the full list of available properties.

```
{
 "listId": 1000000005,
 "country": "Sweden",
 "originalId": "abcabc123123",
 "firstName": "lars",
 "lastName": "hansson",
 "middleName": null,
 "emailAddress": "lars@somwhere.se",
 "title": null,
 "dateOfBirth": null,
 "city": null,
 "zipcode": null,
 "sex": null,
 "mobileNumber": null,
 "properties": {
   "food": "meat",
   "drink": "tea"
 }
,
 "deviceInfo": [
   {
     "devicetoken": "abc123abc123kmlkmlkml",
     "manufacturer": "apple",
     "model": "iphone 6",
     "osVersion": "9.2",
     "country": "Sweden",
     "dateOfInvalidation": null,
     "invalidationType": 0
   }
 ]
}
```

The same struct is used to return info on the device, but it is then appended with all devices and information that is connected to the contact.

The following code can be used for the registration.

```
   class func registerDeviceWithCarma(deviceInfo:String,
                                      firstName:String,
                                      lastName:String,
                                      mobileNumber:String,
                                      emailAddress:String,
                                      originalId:String,
                                      beverage:String,
                                      food:String)-> Bool{

     let urlString = "\( restUrl)/\( customerId)/apps/\( appName)/pushdevices"
     let headers: HTTPHeaders = [
```

```
            "Accept": "application/json",
            "Content-Type": "application/json",
            "X-Carma-Authentication-Token": "\( carmaAccessToken)"

        ]
        userData.deviceInfo["manufacturer"] = "apple"
        let deviceInfo:[String:String] = deviceInfo
        let deviceInfoArray = [deviceInfo]

        let namedProperties:[String:String] =
      ["food":food!,"beverage":beverage!]

        let json:Parameters  = ["firstName":firstName!,
                        "lastName": lastName!,
                       "mobileNumber":mobileNumber!,
                       "emailAddress":emailAddress!,
                       "originalId":originalId!,
                       "deviceInfo":deviceInfoArray,
                       "properties":namedProperties]

        Alamofire.request(urlString, method: .put, parameters:json, encoding:
JSONEncoding.default, headers: headers)
            .validate(contentType: ["application/json","text/html"])
            .responseJSON { (response:DataResponse<Any>) in

            switch(response.result) {
            case .success(_):
                if let data = response.result.value{
                    print(data)
                    let retJson = JSON(data)
                    // do something with the data.
                }
                break

            case .failure(_):
                print(response.result.error!)
                break

            }
        }

        return true
    }
```

Invalidation of a devicetoken.

If you want to remove a registration of a devicetoken for an OriginalId you can issue a PUT request to :

`https://<serverUrl>/rest/<`**`customerid`**`>/apps/`**`<appid>`**`/pushdevices/invalidation`

with a json payload containing at the minimum:

- deviceToken
- originalId,

```
{
   "deviceInfo":[{"devicetoken":
   "abc123abc123kmlkmlkml_2","invalidationType": "1"}],
   "originalId":"abcabc123123"
}
```

NOTE! Even if the originalId and devicetoken are the same, both values must be provided.

Retrieving information about a contact

If you want to retrieve the information connected to a contact you can issue a GET request to

`https://<serverUrl>/rest/<`**`customerid`**`>/apps/`**`<appid>`**`/pushdevices/`**`<originalId>`**

You will receive the same struct as when you register a device. It will contain a list of all devices registered on the originalId. If you have imported information by other means i.e, via scheduledimport on the list, this data will also retreived.

Handling Push Notifications

A compost push message contains the following fields:

- alert - Message to be the line shown next to the icon in the notification
- info - Message to be the line shown under the first in the notification
- url - the url pointing to the Carma rich content of the push
- pushreadurl - call this URL to mark the push as read in Carma. ***NOTE!*** Use this if you are not using carma rich content.
- badge - optional field to show badge number on app icon
- sound -optional field to play sound on reception of push
- category -optional field used for push actions
- source - will always by “carma”. This should be used to distinguish between push messages that comes from other systems than carma. A game for example might have high-score system in place that will notify a player
- any arbitrary field that you have included

It’s important to remember that it’s up to the application developer to decide how these fields should be used. Our iOS client is now registered with APNS, and our server can begin sending messages to devices running our client using the provided app Id and APNS certificate

What Happens When You Receive a Push Notification?

When your app receives a push notification, a method in UIApplicationDelegate is called.

The notification needs to be handled differently depending on what state your app is in when it’s received:

- If your app wasn’t running and the user launches it by tapping the push notification, the push notification is passed to your app in the launchOptions of application(_:didFinishLaunchingWithOptions:).
- If your app was running and in the foreground, the push notification will not be shown, but application(_:didReceiveRemoteNotification:) will be called immediately.
- If your app was running or suspended in the background and the user brings it to the foreground by tapping the push notification, application(_:didReceiveRemoteNotification:) will be called.

Here is a minimal implementation of didReceiveRemoteNotification:

```
    func application(_ application: UIApplication, didReceiveRemoteNotification
data: [AnyHashable : Any]) {
        // Print notification payload data
        print("Push notification received: \(data)")
        let dict:NSDictionary = data as NSDictionary
        let url: String? = data["url"] as? String
        let pushreadurl: String? = data["pushreadurl "] as? String
      …
    }
```

Carma Rich Content

The rich content of a push message is not sent in the push itself, but is located at server side where the url in message is pointing. Content is usually a page of html that is generated for the particular device, user by a template of the server side. It could however be XML or other text-based formats as defined by template.

The most common way to handle content is to simply open an UIWebView and pointing it to the url, making it display the page. The developer could however choose to handle it in different ways.

NOTE! If you don't want to fetch the rich content you should mark the push as opened in carma by issuing a GET request to the *pushreadurl.* No credentials is needed for this request.

Testing and troubleshooting

Once you have done integration, perform the following tests.

- Install the application on a new device
- Notice that the number of recipients increases for each time you do this. If not, check that the id's you supply are correct. Check also that the appid you gave when registering the application is the same as the package given in the top of your manifest.
- If the list does grow, test a simple push while the application is running. If a notification does not show up after a few minutes, make sure that you have given the correct AppId when registering the app in Carma.
- Once the above works correct, verify that the notifications show up when sending a push even though the application is not running.
- Remember that you must send push as develop or production even if you are using a singing certificate. APNS uses different endpoint depending on whether the app is live on the app store or if it is in development.
- If you do not receive any notifications while developing in XCODE, makes sure that you have registered your device with field sandboxToken = true.