



AI & ML DRIVEN
MARKETING
AUTOMATION



Integration document

iOS Development

This document will help you integrate the applICE SDK in your iOS Projects.

Table of Contents

Setting up your app on //appice.io	4
Sign-up	4
Setup your App	4
Including applICE SDK in your project	7
Add applICE SDK to your app	7
Changes required in the Build Phase	7
Compile Sources	7
Link Binary With Libraries	8
Add ApplICE iOS SDK using CocoaPods	8
Changes to the AppDelegate.m file	9
Import applICE.h	9
Adding applICE project attributes	9
Initialize applICE	9
Handling Push Notifications	9
Enabling Push Notifications for your App	9
Create the CSR file	9
Developer Console Settings	9
Generate the APNS Certificate	9
Generate the certificate and key	9
1) for Development mode	9
Coding Custom Events	9
Define a Custom Event	9
Adding Attributes to the Custom Event	9
Examples	9
// Add a custom event for product viewed by a user along with attributes	9
Personalizing Notifications	9
Creating ICE-Tags	9
Using ICE-Tags in Notifications	9
Install App on your Android/iOS phone	9
Verify integration on applICE panel	9

Setting up your app on //appice.io

Sign-up

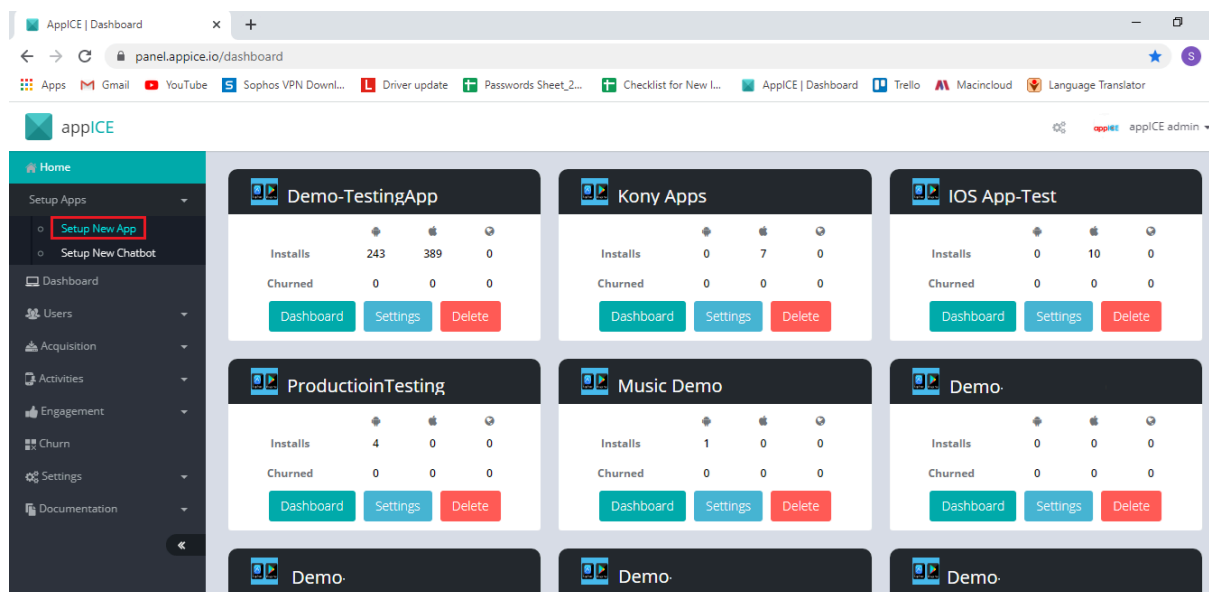
You need to sign-up and create an account with appice.io.

1. Visit <https://panel.appice.io/Signup> to Register
2. This brings you to the “Sign up” page
3. Provide your Name, Email address and your chosen password and create your account
4. Login using your username and password credentials once your account is approved

Setup your App

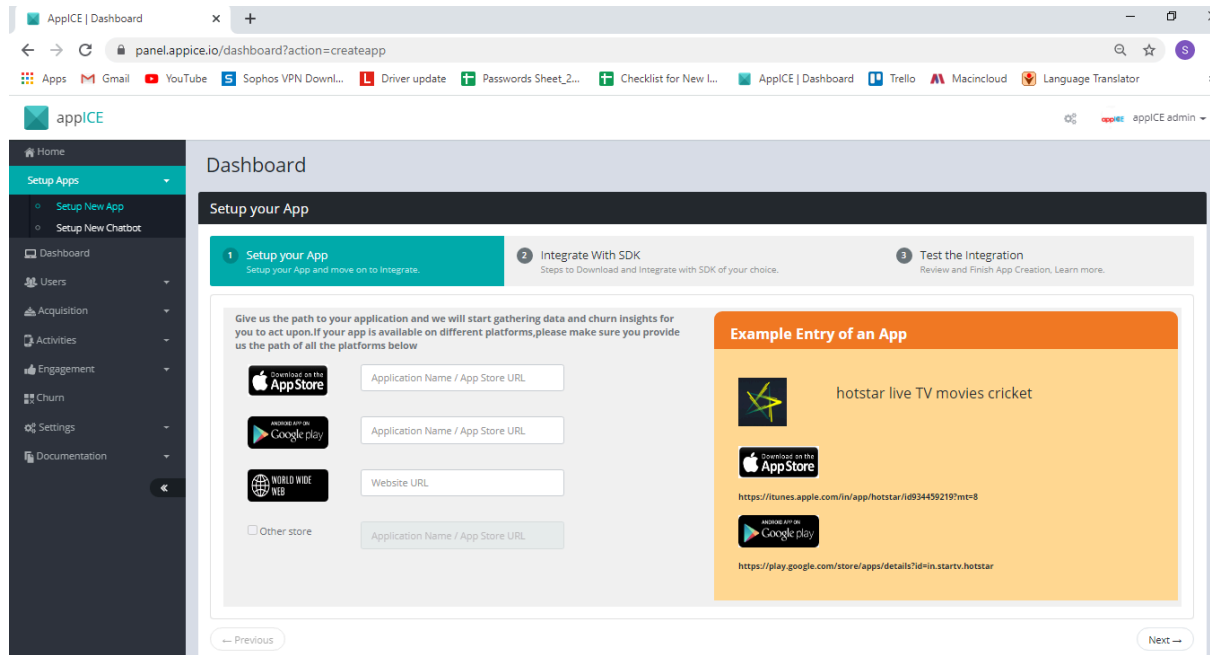
To start the process of integrating appICE in your app, you will first need to setup your app on the appICE dashboard.

1. Click on Setup New App in the left panel.



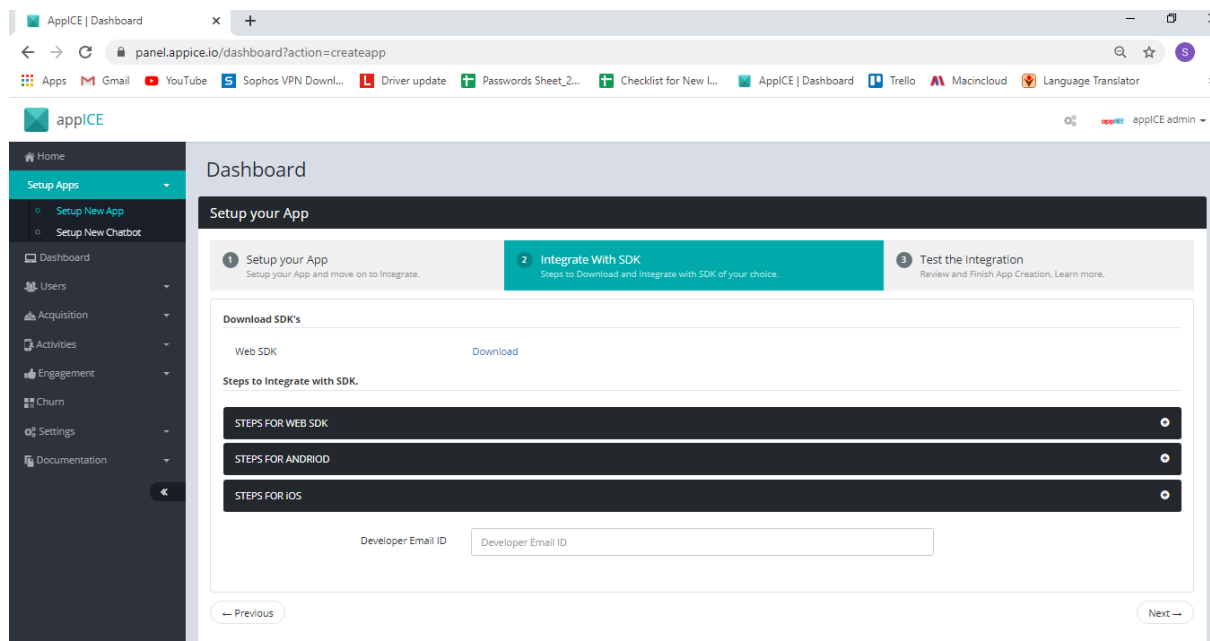
2. Provide the link to your mobile app on the Google Play Store as well as Apple store if you have both versions of your app.

Click Next to go to the next page.



Click Next to go to the next page.

- Now you have access to the SDK's for React Native including the instructions for integration. To make it easier, we have also provided the option to provide your developer's email address so that those instructions can be emailed to them directly.



Click Next to go to the next page.

- Now you are all ready to start receiving data from your mobile app once the developer completes the integration and publishes the app again on the app store.

AppICE | Dashboard

panel.appice.io/dashboard?action=createapp

Apps Gmail YouTube Sophos VPN Downl... Driver update Passwords Sheet_2... Checklist for New L... AppICE | Dashboard Trello Macincloud Language Translator

appICE

appICE admin

Home

Setup Apps

- Setup New App
- Setup New Chatbot

Dashboard

Users

Acquisition

Activities

Engagement

Churn

Settings

Documentation

Dashboard


Setup your App

1 Setup your App
Setup your App and move on to Integrate.

2 Integrate With SDK
Steps to Download and Integrate with SDK of your choice.

3 Test the Integration
Review and Finish App Creation. Learn more.

App Created!



test

Go to my Dashboard

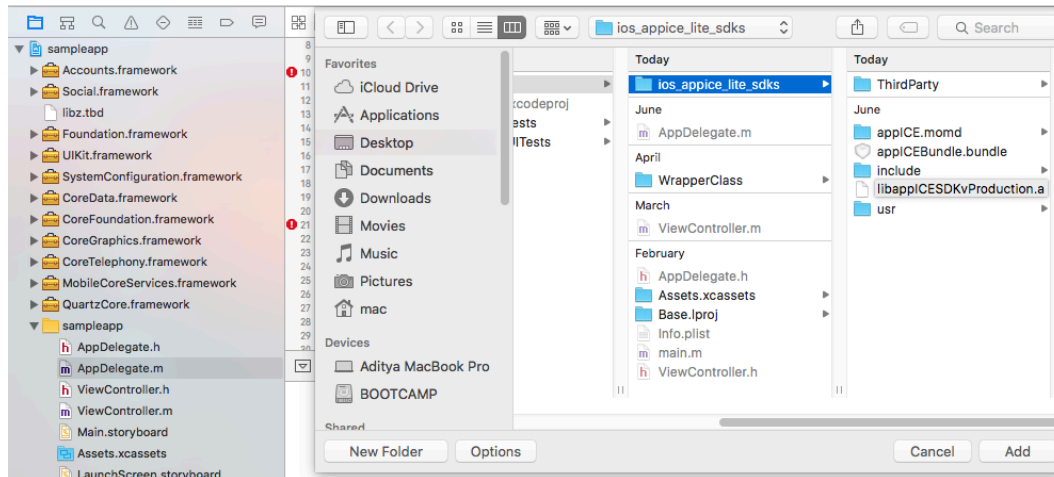
Previous

Next

Including applICE SDK in your project

Add applICE SDK to your app

You need to include the `ios_applICE_lite_sdks` folder in your application.



Note:

If you are already using `asi-http-request`, `SBJsonParser`, `Reachability` classes in your project, please don't include the Third Party Folder in your project.

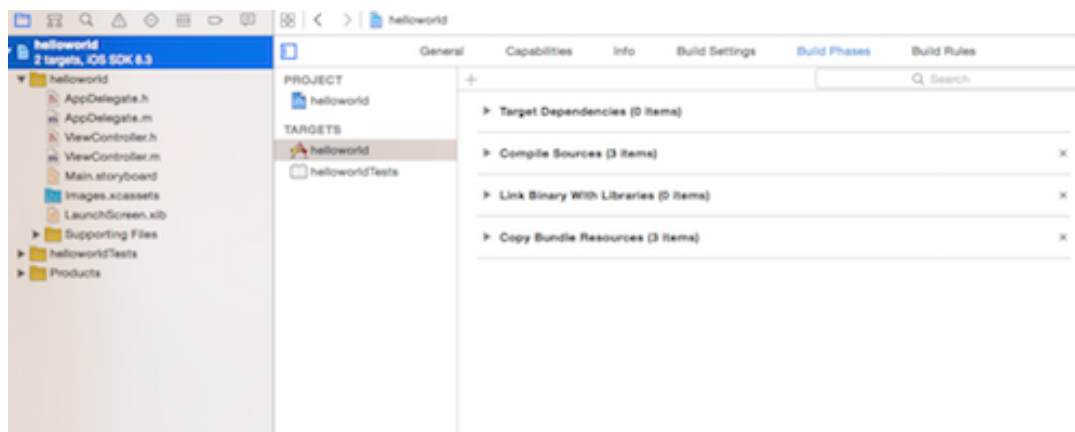
OR

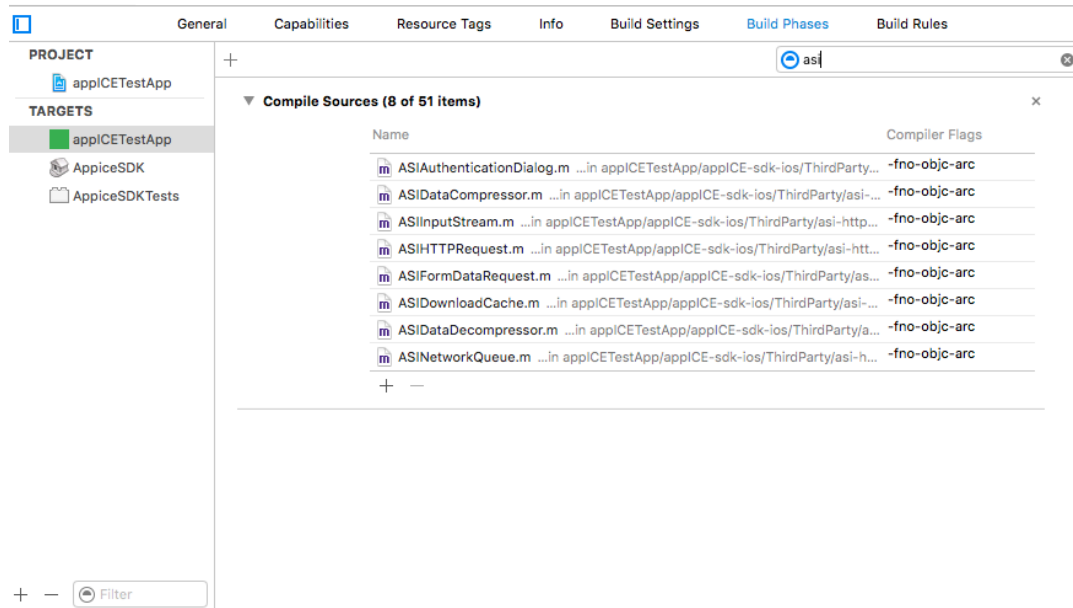
If you are not using `asi-http-request`, `SBJsonParser`, `Reachability` classes in your project, please include the Third Party Folder in your project.

Changes required in the Build Phase

Compile Sources

Add the flag `-fno-objc-arc` in compile sources of build phase option of your project for the above classes (`asi-http-request`, `SBJsonParser`, `Reachability`).



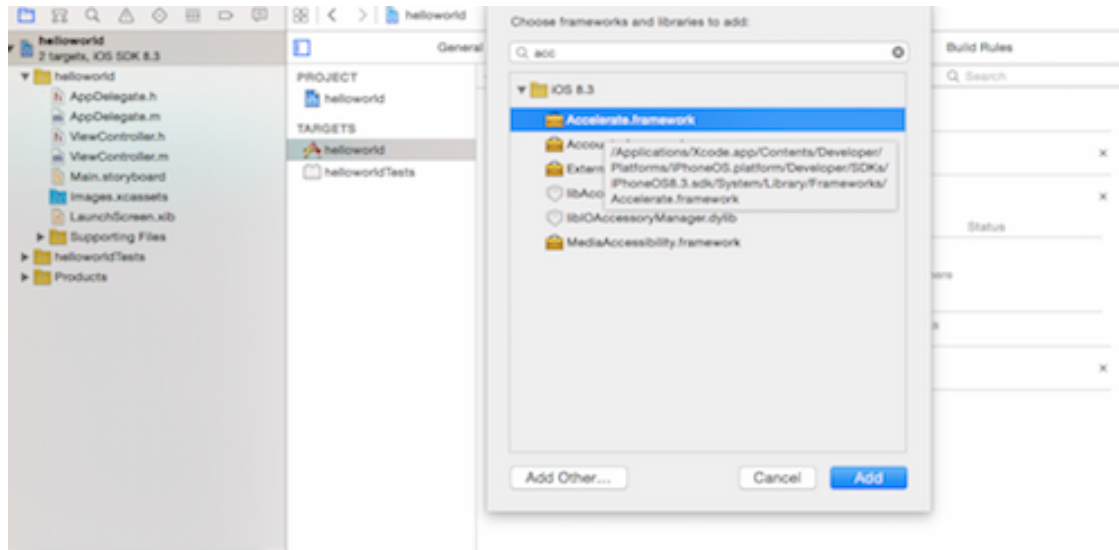


Link Binary With Libraries

Click on Link Binary With Libraries and add these frameworks.

- a) libz.dylib
- b) MobileCoreServices.framework
- c) CoreGraphics.framework
- d) CFNetwork.framework (**mark it as optional**)
- e) Foundation.framework (**mark it as optional**)
- f) UIKit.framework (**mark it as optional**)
- g) SystemConfiguration.framework
- h) CoreTelephony.framework
- i) CoreData.framework
- j) Security.framework
- k) Social.framework
- l) Accounts.framework
- m) ImageIO.framework

n) AssetsLibrary.framework



(OR) Add AppICE iOS SDK using CocoaPods

Step 1: Open Podfile

1. Open your terminal and navigate to your project directory.
2. Open the Podfile using the following command:
open -e Podfile

Step 2: Add AppICE iOS SDK Dependency

use_frameworks! :linkage => :static (for xcode 12 and above)

pod 'AppICE-IOS-SDK'(for specific version pod 'AppICE-IOS-SDK' , '1.7.59')

Step 3: Install Pods

Save the Podfile and run the following command to install the specified version of the AppICE iOS SDK and its dependencies.

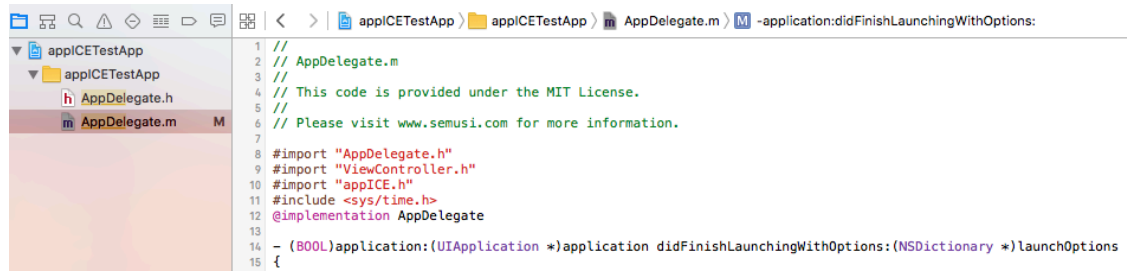
pod install

Changes to the appDelegate.m file

Import appICE.h

Import the appICE.h file in the beginning of your appDelegate.m file

#import "appICE.h"



```

1 //
2 // AppDelegate.m
3 // This code is provided under the MIT License.
4 // Please visit www.semusi.com for more information.
5
6 #import "AppDelegate.h"
7 #import "ViewController.h"
8 #import "appICE.h"
9 #include <sys/time.h>
10 @implementation AppDelegate
11
12 - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
13 {
14
15

```

Adding appICE project attributes

To ensure that the data is added to the right application you created on the appICE dashboard, you need to add the following highlighted block in the **didFinishLaunchingWithOptions** function in AppDelegate.m file

```

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

```

Objective C:

```

NSArray* nameArr = [NSArray arrayWithObjects: @"assets/isrgrootx1.der",nil];
[appICE setupKeys:@"YOUR_APP_KEY" withapiKey:@"YOUR_API_KEY"
withappId:@"YOUR_APP_ID" otherSdkdeviceId:@"OtherAnalytics_distinctID"
region:nil baseUrl:nil certificates:nameArr];

```

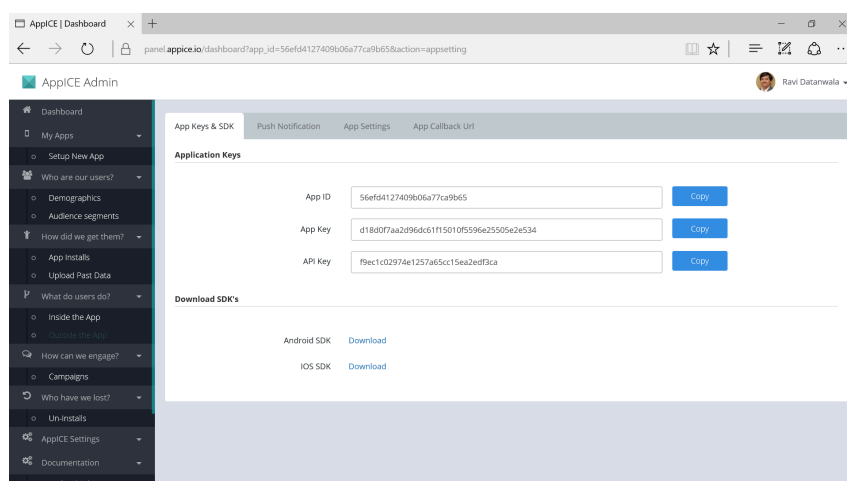
Swift:

```

let nameArr = ["assets/isrgrootx1.der"]
appICE.setupKeys("YOUR_APP_KEY", withapiKey: "YOUR_API_KEY",
withappId: "YOUR_APP_ID", otherSdkdeviceId: "OtherAnalytics_distinctID", region:
"", baseUrl: "", certificates: nameArr)
}

```

Please replace **YOUR_APP_KEY**, **YOUR_API_KEY** and **YOUR_APP_ID** with the values from your appICE Dashboard Settings section (shown below)



If you are not using any other analytics tool in your project, please replace **OtherAnalytics_distinctID** with "" (empty string)

If you are using any other analytics tool, please replace **OtherAnalytics_distinctID** with the variable name which represents the users unique ID.

Initialize appICE

Add the following line just below the section where you have provided the appICE attributes to initialize appICE functionality.

Objective C:

```
[[appICE sharedInstance] startContext];
```

Swift:

```
appICE.sharedInstance().startContext()
```

Handling Push Notifications

To handle push notification, please add the following functions (In case you already have these functions in your project, just add the highlighted yellow blocks in the respective functions)

```
- (void)application:(UIApplication*)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken{
```

Objective C:

```
[appICE setTokenInPushNotification:deviceToken];
```

Swift:

```
appICE.setTokenInPushNotification(deviceToken)
```

```
}
```

```
- (void)application:(UIApplication*)application
didFailToRegisterForRemoteNotificationsWithError:(NSError*)error{
```

```
    NSLog(@"Failed to get token, error: %@", error);
```

```
}
```

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

Objective C:

```
NSDictionary *userInfo = response.notification.request.content.userInfo;
```

```
    BOOL appceServer = [[appICE sharedInstance] isAppICENotification:userInfo];
```

```
    if(appceServer)
```

```
{
    [[appICE sharedInstance] handleClickOnPush:userInfo OpenDeepLink:YES];
    completionHandler();
}
```

Swift:

```
let appIceServer = appICE.sharedInstance().isAppICENotification(userInfo)

if appIceServer {
    appICE.sharedInstance().handleClick(onPush: userInfo, openDeepLink: true)
    completionHandler()
}
}
```

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

Objective C:

```
BOOL appIceServer = [[appICE sharedInstance] isAppICENotification:userInfo];
if(appIceServer)
{
    [[appICE sharedInstance] pushNotificationReceived:userInfo];
}
}
```

Swift:

```
let appIceServer = appICE.sharedInstance().isAppICENotification(userInfo)
if appIceServer {
    appICE.sharedInstance().pushNotificationReceived(userInfo)
}
}
```

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

Objective C:

```
BOOL appIceServer = [[appICE sharedInstance] isAppICENotification:userInfo];
if(appIceServer)
{
    [[appICE sharedInstance] pushNotificationReceived:notification];
}
```

```
}
```

Swift:

```
if let userInfo = notification.userInfo {
    let appIceServer = appICE.sharedInstance().isAppICENotification(userInfo)
    if appIceServer {
        appICE.sharedInstance().pushNotificationReceived(notification)
    }
}
}
```

```
- (void)application:(UIApplication *)application handleActionWithIdentifier:(NSString *)identifier forLocalNotification:(UILocalNotification *)notification completionHandler:(void (^)(void))completionHandler {
```

Objective C:

```
if ([identifier isEqualToString:@"ACTION_ONE"] || [identifier isEqualToString:@"ACTION_TWO"])
{
    //normalizing the data into dictionary
    NSDictionary *userInfo = [((UILocalNotification *) notification) userInfo];
    BOOL appIceServer = [[appICE sharedInstance] isAppICENotification:userInfo];
    if(appIceServer)
    {
        [[appICE sharedInstance] handleClickOnPush:userInfo OpenDeepLink:YES];
    }
}
if (completionHandler) {
    completionHandler();
}
```

Swift:

```
if let identifier = identifier {
    if identifier == "ACTION_ONE" || identifier == "ACTION_TWO" {
        let appIceServer =
appICE.sharedInstance().isAppICENotification(userInfo)
        if appIceServer {
```

```

applICE.sharedInstance().handleClick(onPush: userInfo, openDeepLink:
true)
    }
    }
    }
    }

    completionHandler()
}
}
}

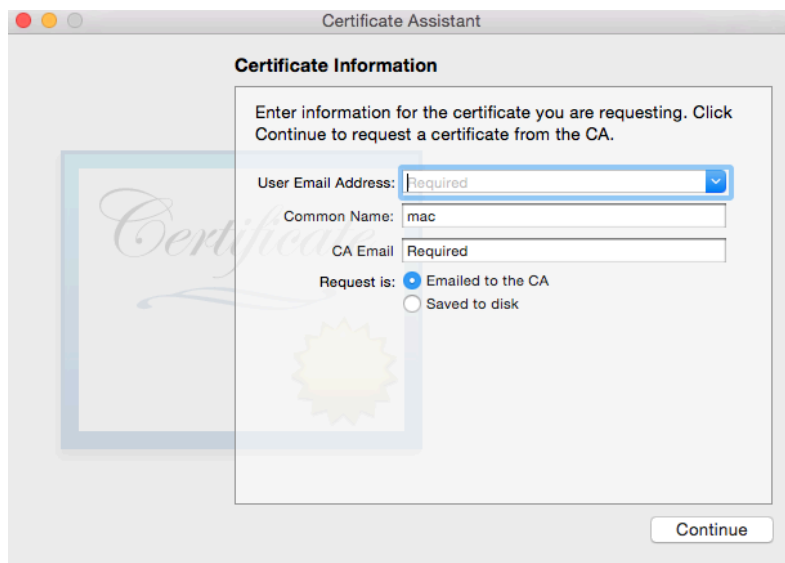
```

Enabling Push Notifications for your App

Create the CSR file

Open Keychain Access on your Mac (it is in Applications/Utilities) and choose the menu option **Request a Certificate from a Certificate Authority...** Also make sure no private key is selected in the main Keychain Access window.

You should now see the following window:



Enter your email address here. you use the same email address that you used to sign up for the iOS Developer Program.

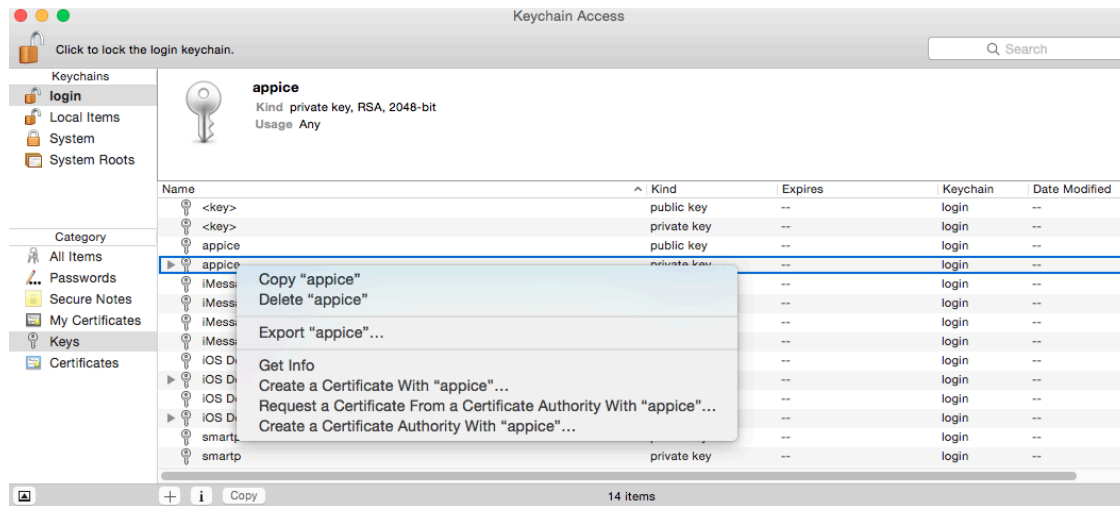
Enter “applICE” for Common Name. You can type anything you want here, but choose something descriptive. This allows us to easily find the private key later.

Check **Saved to disk** and click **Continue**.

Save the file as “applICE.certSigningRequest”.

Confirm the same by going to the Keys section of Keychain Access and you will see that a new private key has appeared in your keychain.

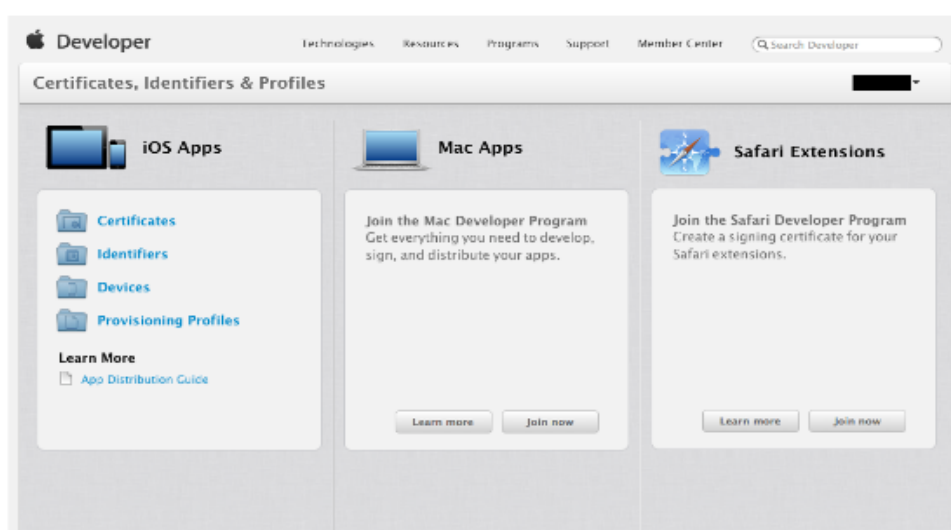
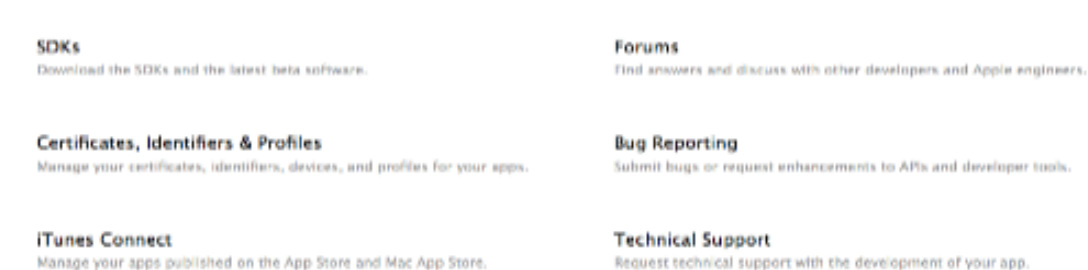
Right click on the newly created private key and choose Export.



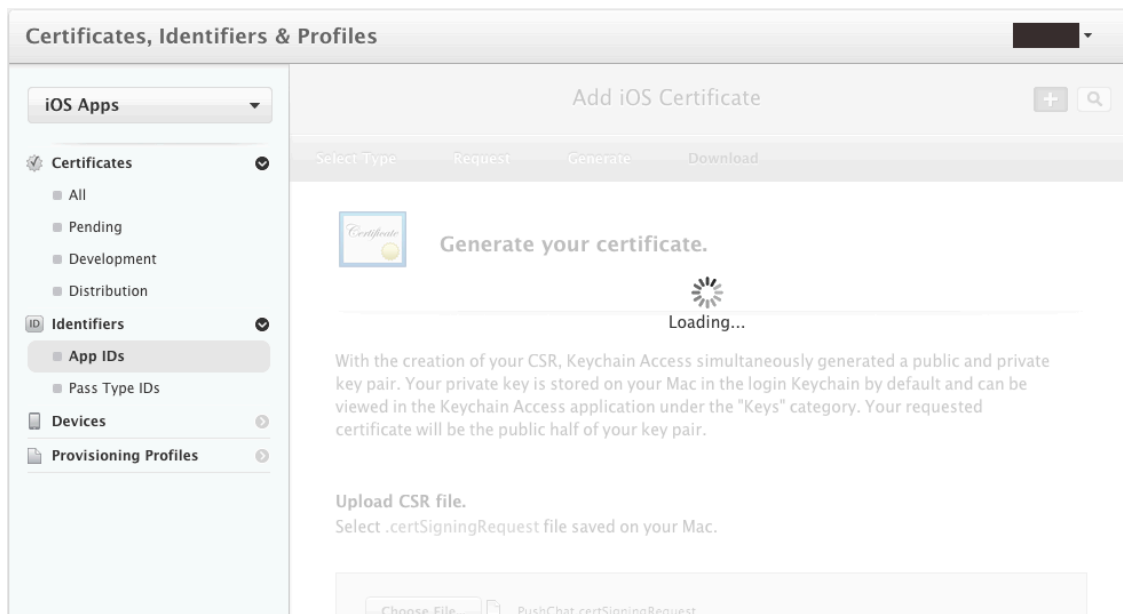
Save the private key as **appicEKey.p12** and enter passphrase.

Developer Console Settings

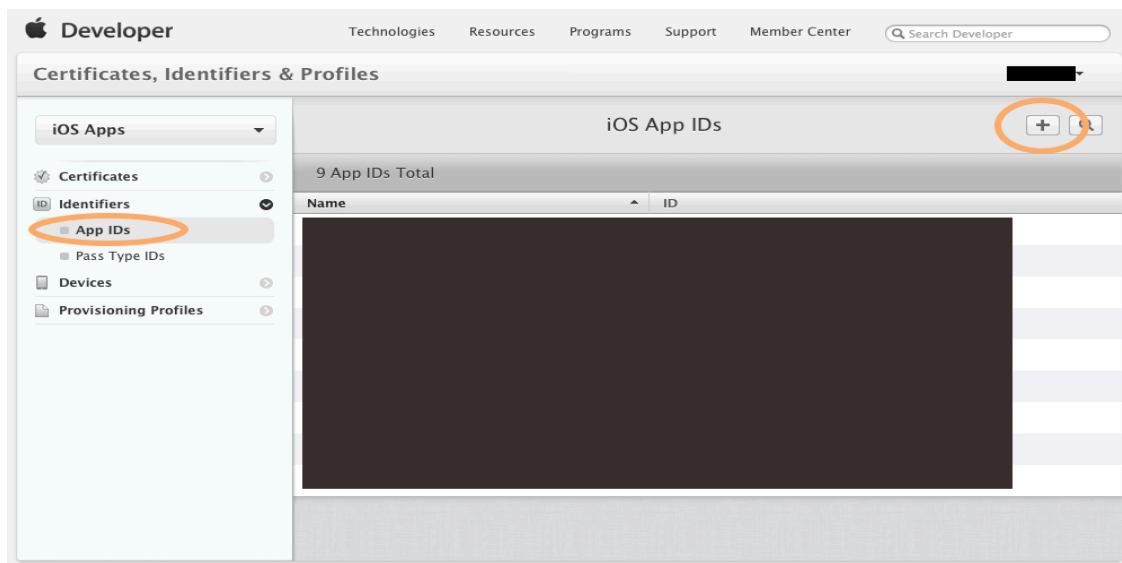
Now you have to login on member center of developer portal for creation the certificate and provision profile. You will be presented with the following screen. Now click on certificate, identifier & Profiles.



Go to the certificate in the sidebar and click the + button and upload the CSR file.



Now download the development certificate and double click on it with show in finder.



Now, you are going to make a new App ID. Each push app

needs its own unique ID because push notifications are sent to a specific application. (You cannot use a wildcard ID.) Go to **App IDs** in the sidebar and click the + button.

Fill the following details:

App ID Description: "YOUR_APP_NAME"

App Services: Check the Push Notifications Checkbox

Explicit App ID: com.domainname.appname

It is probably best if you choose your own Bundle Identifier here – com.domainname.appname. You will need to set this same bundle ID in your Xcode project.

After you're done filling all the details press the **Continue** button. You will be asked to verify the details of the app id, if everything seems okay click **Submit**

Hurray! You have successfully registered a new App ID.

In a few moments, you will generate the SSL certificate that your push server uses to make a secure connection to APNS. This certificate is linked with your App ID. Your server can only send push notifications to that particular app, not to any other apps. After you have made the App ID, it shows up like this in the list, Select the **appICE** app ID from the list. and see this screen view.

App Group	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Associated Domains	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Data Protection	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Game Center	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
HealthKit	<input type="radio"/> Disabled	<input type="radio"/> Disabled
HomeKit	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Wireless Accessory Configuration	<input type="radio"/> Disabled	<input type="radio"/> Disabled
iCloud	<input type="radio"/> Disabled	<input type="radio"/> Disabled
In-App Purchase	<input checked="" type="radio"/> Enabled	<input checked="" type="radio"/> Enabled
Inter-App Audio	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Apple Pay	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Passbook	<input type="radio"/> Disabled	<input type="radio"/> Disabled
Push Notifications	<input type="radio"/> Disabled	<input type="radio"/> Disabled
VPN Configuration & Control	<input type="radio"/> Disabled	<input type="radio"/> Disabled

Edit

Generate the APNS Certificate

1) In development mode

Click on the **Edit** button to configure these settings, Scroll down to the Push Notifications section and select the **Create Certificate** button in the **Development SSL Certificate** section.




Push Notifications
● Enabled




Apple Push Notification service SSL Certificates

To configure push notifications for this iOS App ID, a Client SSL Certificate that allows your notification server to connect to the Apple Push Notification Service is required. Each iOS App ID requires its own Client SSL Certificate. Manage and generate your certificates below.


Development SSL Certificate


Create certificate to use for this App ID.

Create Certificate...


Production SSL Certificate

Create certificate to use for this App ID.

Create Certificate...


Development SSL Certificate

Name: Apple Development iOS Push Services: XXXXXXXXXX
Type: APNs Development iOS
Expires: Apr 12, 2014

Revoke

Download

Create an additional certificate to use for this App ID.

Create Certificate...

The first thing it asks you is to generate a Certificate Signing Request. You already did that, so click **Continue**. In the next step you upload the CSR. Choose the CSR file that you generated earlier and click **Generate**. It takes a few seconds to generate the SSL certificate. Click **Continue** when it's done.

Now click **Download** to get the certificate it is named `aps_development.cer`

Making a PEM File, So now you have three files:

- 1) The CSR
- 2) The private key as a p12 file (`applICEKey.p12`)
- 3) The SSL certificate, `aps_development.cer`

Store these three files in a safe place or Desktop.

- 2) In Distribution mode or for live application

Now select the **Create Certificate** button in the **Production SSL Certificate** section and upload the CSR. Choose the CSR file that you generated earlier and click **Generate**. It takes a few seconds to generate the SSL certificate. Click **Continue** when it done.

Now click **Download** to get the certificate it is named `aps.cer`

Making a PEM File, So now you have three files:

- 1) The CSR
- 2) The private key as a p12 file (`appleKey.p12`)
- 3) The SSL certificate, `aps.cer`

So please also store these three files in a safe place or Desktop.

Generate the certificate and key

You are going to use the command-line OpenSSL tools for this. Open a Terminal and execute the following steps. Go to the folder where you downloaded the files, in my case the Desktop:

1) for Development mode

```
cd ~/Desktop/
```

```
openssl x509 -in aps_development.cer -inform der -out appleCert.pem
```

Convert the private key `.p12` file into a `.pem` file:

```
openssl pkcs12 -nocerts -out appleKey.pem -in appleKey.p12
```

Enter Import Password:

MAC verified OK

Enter PEM pass phrase:

Verifying - Enter PEM pass phrase:

You first need to enter the passphrase for the `.p12` file so that openssl can read it. Then you need to enter a new passphrase that will be used to encrypt the PEM file. Again for this tutorial I used `apple` as the PEM passphrase. You should choose something more secure.

Note: if you don't enter a PEM passphrase, openssl will not give an error message but the generated `.pem` file will not have the private key in it.

Finally, combine the certificate and key into a single `.pem` file so you need to run the command separately

```
cat appleCert.pem appleKey.pem > ck.pem
```

Execute the following command:

```
telnet gateway.sandbox.push.apple.com 2195
```

```
Trying 17.172.232.226...
```

Connected to gateway.sandbox.push-apple.com.akadns.net.

Escape character is '^['.

This tries to make a regular, unencrypted, connection to the APNS server. If you see the above response, then your Mac can reach APNS. Press Ctrl+C to close the connection. If you get an error message, then make sure your firewall allows outgoing connections on port 2195.

Lets try connecting again, this time using our SSL certificate and private key to set up a secure connection:

```
openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert applICECert.pem  
-key applICEKey.pem
```

Enter pass phrase for applICEKey.pem:

You should see a whole bunch of output. If the connection is successful, you should be able to type a few characters. When you press enter, the server should disconnect. If there was a problem establishing the connection, openssl will give you an error message but you may have to scroll up through the output to find it.

2) For production mode

```
cd ~/Desktop/
```

```
openssl x509 -in aps.cer -inform der -out applICECert.pem
```

Convert the private key 痴 .p12 file into a .pem file:

```
openssl pkcs12 -nocerts -out applICEKey.pem -in applICEKey.p12
```

Enter Import Password:

MAC verified OK

Enter PEM pass phrase:

Verifying - Enter PEM pass phrase:

You first need to enter the passphrase for the .p12 file so that openssl can read it. Then you need to enter a new passphrase that will be used to encrypt the PEM file. Again for this tutorial I used apple as the PEM passphrase. You should choose something more secure.

Note: if you don't enter a PEM passphrase, openssl will not give an error message but the generated .pem file will not have the private key in it.

Finally, combine the certificate and key into a single .pem file so you need to run the command separately

```
cat applICECert.pem applICEKey.pem > ck.pem
```

Execute the following command:

```
telnet gateway.sandbox.push.apple.com 2195
```

Trying 17.172.232.226...

Connected to gateway.sandbox.push-apple.com.akadns.net.

Escape character is '^['.

This tries to make a regular, unencrypted, connection to the APNS server. If you see the above response, then your Mac can reach APNS. Press Ctrl+C to close the connection. If you get an error message, then make sure your firewall allows outgoing connections on port 2195.

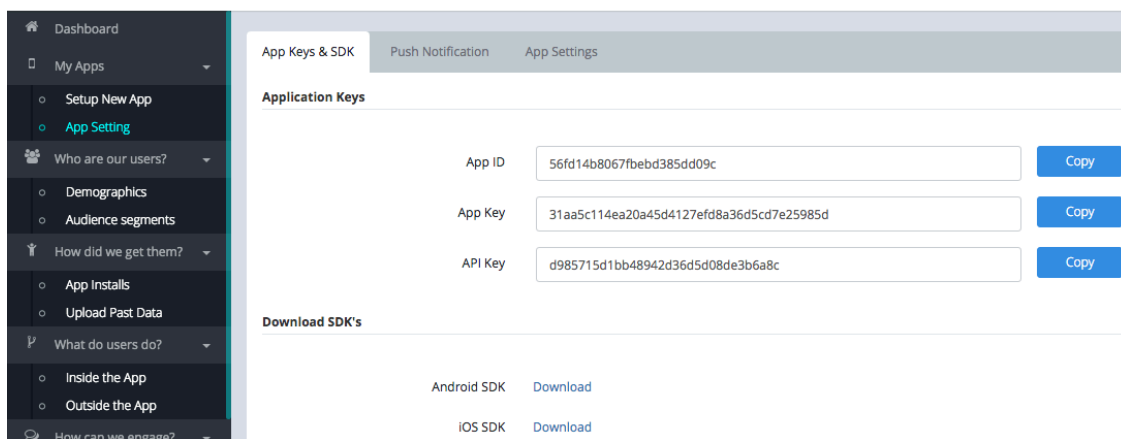
Lets try connecting again, this time using our SSL certificate and private key to set up a secure connection:

```
openssl s_client -connect gateway.sandbox.push.apple.com:2195 -cert appICECert.pem -key appICEKey.pem
```

Enter pass phrase for appICEKey.pem:

You should see a whole bunch of output. If the connection is successful, you should be able to type a few characters. When you press enter, the server should disconnect. If there was a problem establishing the connection, openssl will give you an error message but you may have to scroll up through the output to find it.

Now login into the Applice Dashboard through <https://panel.applice.io/> , and go to My Apps and click on settings of the application and see this Wizard...



Now click on Push Notification as shown below and Upload the APNS Certificate and APNS Key for the Push notification for development and production mode separately that you have already downloaded and saved on Desktop. Click on **Save Files**.

App Keys & SDK
Push Notification
App Settings

To be able to send Push Notifications, we need to link your account with the appropriate certificate and keys. Make sure that you do specify the correct certificate keys for your production environment as well as your test/staging/development environment.

Development
Production

Apple Push Notification
Google Cloud Messaging (GCM)

Apple Push Notification Service(APNS) – Development

APNS Certificate

APNS Certificate

Upload

Password of APNS Certificate

password

APNS Key

APNS Key

Upload

Password of APNS Key

password

Save Files

Coding Custom Events

You can add Custom events in your iOS apps to capture user actions along with associated data attributes.

Define a Custom Event

```
(void)recordEvent:(NSString *)key count:(int)count;
```

Objective C:

```
[[applCE sharedInstance]recordEvent:@"Event_Sign_In" count:1];
```

Swift:

```
applCE.sharedInstance().recordEvent("Event_Sign_In", count: 1)
```

The above code snippet creates a custom event called 'Event_Sign_In' in your Dashboard.

Adding Attributes to the Custom Event

You can pass data attributes along with the Event for granularity:

```
- (void)recordEvent:(NSString *)key segmentation:(NSDictionary *)segmentation
count:(int)count;
```

Native Code Sample

// Add a custom event for product viewed by a user along with attributes

Objective C:

```
NSDictionary *dict = @{@"user_id" : @"John@applCE.io", @"value":@"Tech"};
[[applCE sharedInstance] recordEvent:@"Product_viewed" segmentation:dict
count:1];
```

Swift:

```
let dict: [String: Any] = ["user_id": "John@appICE.io", "value": "Tech"]
appICE.sharedInstance().recordEvent("Product_viewed", segmentation: dict, count:
1)wq
```

Hybrid Code Sample

iOS AppICE Hybrid Bridge:

Register the WebView in ViewController.m

```
func addWebview() {
    let appIceInterface: AppICEJSInterface = AppICEJSInterface()
    self.webView = WKWebView (frame: self.view.frame)
    self.webView.configuration.userContentController.add(appIceInterface,
name: "appice")
    //using dynamic path
    self.webView.load(NSURLRequest(url:NSURL(string:"https://websdk.semusi.c
om/?isWebView=1")! as URL)as URLRequest) //sample websdk Url

    self.view.addSubview(self.webView)
}
```

In Objective C

```
NSString *path = @"https://websdk.semusi.com/?isWebView=1";
NSURL *url = [NSURL URLWithString:path];

NSURLRequest *request = [NSURLRequest requestWithURL:url];

// Initialize the Webview and add the AppICEJSInterface as a script message
handler

AppICEJSInterface *appIceInterface = [[AppICEJSInterface alloc] init];
self.webView = [[WKWebView alloc] initWithFrame:self.view.frame];
[self.webView.configuration.userContentController
addScriptMessageHandler:appIceInterface name:@"appice"];
self.webView.frame = CGRectMake(0, _txt_field.frame.origin.y, 400, 200);
[self.webView loadRequest:request];
[self.view addSubview:self.webView];
```

WebApp should have AppICE WebSDK integrated internally.

So when the user clicks on a button inside the website and if the web team is using recordEvent (from WebSDK) on that button then the event will be captured in the same session.

Web app Example:

Function call :

```
function recordAppICEEvent() {  
  var key = document.getElementById("recordKey").value;  
  var segmentKey = document.getElementById("SegmentK").value;  
  var segmentValue = document.getElementById("SegmentV").value;  
  var props = {  
    segmentKey: segmentValue  
  };  
  recordEvent(key,props)  
}
```

```
function setAppICEUserId() {  
  var userId = document.getElementById("userId").value;  
  const userIdArray = [  
    userId  
  ];  
  setUserId(userIdArray);  
}
```

```
function setAppICEUserDetail() {  
  var name = document.getElementById("name").value;  
  var phone = document.getElementById("phone").value;  
  var email = document.getElementById("email").value;  
  var age = document.getElementById("age").value;  
  var dob = parseInt(document.getElementById("dob").value);  
  var education = document.getElementById("education").value;  
  var gender = document.getElementById("gender").value;
```



```

var employed = document.getElementById("employed").value;
var employmentType = document.getElementById("employmentType").value;
var married = document.getElementById("married").value;
var userDetails = {
    "n": name,
    "p": phone,
    "e": email,
    "a": age,
    "d": dob,
    "edt": education,
    "g": gender,
    "em": employed,
    "emt": employmentType,
    "m": married
};
setUser(userDetails)
}

```

```

/**
 * This function retrieves the custom key and value from the input fields,
 * determines the appropriate data type for the custom value, and then
 * calls the setCustomVariable function with the correct data type.
 *
 * We are getting customValue from a text-field (HTML), which means the input
 * is always a string initially. Depending on the actual content of the
 * customValue, we need to convert it to the appropriate data type (integer,
 * float, boolean, or leave it as a string) before passing it to setCustomVariable.
 */

```

```

function setAppICECustomVariable() {
    var customKey = document.getElementById("customKey").value;
    var customValue = document.getElementById("customValue").value;
    var parsedValue;
    if (!isNaN(customValue)) {
        if (customValue.includes('.')) {
            parsedValue = parseFloat(customValue); // float or double
        } else {
            parsedValue = parseInt(customValue); // int or long
        }
    }
}

```

```
    } else if (customValue.toLowerCase() === 'true' || customValue.toLowerCase() ===  
'false') {  
        parsedValue = customValue.toLowerCase() === 'true';  
    } else {  
        parsedValue = customValue;  
    }  
    setCustomVariable(customKey, parsedValue);  
}
```

Fetching Campaigns

This method retrieves campaigns based on a specific type (such as "NATIVE", "INAPP", or "PUSH") using the `getCampaigns` method. This allows you to get detailed campaign

information data based on the type of campaign you are targeting.

Define a Campaign Fetch

```
- (NSMutableArray<Campaign *> *)getCampaigns:(NSString *)type;
```

Objective C:

```
NSString *campaignType = @"NATIVE";

NSMutableArray<Campaign *> *campaigns = [[applICE sharedInstance]
getCampaigns:campaignType];

for (Campaign *campaign in campaigns) {

    NSLog(@"Campaign ID: %@", campaign.campId);

    NSLog(@"Action Type: %@", campaign.actionType);

    NSLog(@"Action URL: %@", campaign.actionUrl);

    NSLog(@"Custom Data: %@", campaign.customData);

}
```

Swift:

```
let campaignType = "NATIVE"

let campaigns = applICE.sharedInstance().getCampaigns(campaignType)

for campaign in campaigns {

    print("Campaign ID: \(campaign.campId)")

    print("Action Type: \(campaign.actionType)")

    print("Action URL: \(campaign.actionUrl)")

    print("Custom Data: \(campaign.customData)")

}
```

Fetching Campaign by ID

This method retrieves a specific campaign by its unique campaign ID using the `getCampaignById` method. This allows you to retrieve detailed information about a particular campaign based on its `campId`.

Define a Campaign Fetch by ID

```
- (Campaign *)getCampaignById:(NSString *)campId;
```

Objective C:

```
NSString *campaignId = @"66e051f99c6223d36db0cfc3";

Campaign *campaign = [[applICE sharedInstance] getCampaignById:campaignId];

if (campaign) {

    NSLog(@"Campaign ID: %@", campaign.campId);

    NSLog(@"Action Type: %@", campaign.actionType);

    NSLog(@"Action URL: %@", campaign.actionUrl);

    NSLog(@"Custom Data: %@", campaign.customData);

} else {

    NSLog(@"No campaign found for ID: %@", campaignId);
}
```

Swift:

```
let campaignId = "66e051xxxxxxxxxf4"

if let campaign = applICE.sharedInstance().getCampaignById(campaignId) {

    print("Campaign ID: \(campaign.campId)")

    print("Action Type: \(campaign.actionType)")

    print("Action URL: \(campaign.actionUrl)")

    print("Custom Data: \(campaign.customData)")

} else {

    print("No campaign found for ID: \(campaignId)")

}
```

Find Index.js from below link:

<https://webcdn.appice.io/sdk/index.js>

Personalizing Notifications

appICE allows you to personalize your notifications (push/in-app) to ensure that your users can feel more engaged in the app. In order to enable personalization in your campaigns, you would be making use of “ICE-Tags”

Creating ICE-Tags

You can use your own application/CRM data and pass that to the appICE services to be used in push/in-app notifications. Ideally this would be done as soon as the app starts and you can set the values in these ICE-Tags.

// set custom variable for any Bool object

[[appICE sharedInstance]setCustomVariableBool:@"set_flag" value:YES];

// set custom variable for any int32 object

[[appICE sharedInstance]setCustomVariableint32:@"id" value: 123];

// set custom variable for any int64 object

[[appICE sharedInstance]setCustomVariableint64:@"type_id" value:12345678];

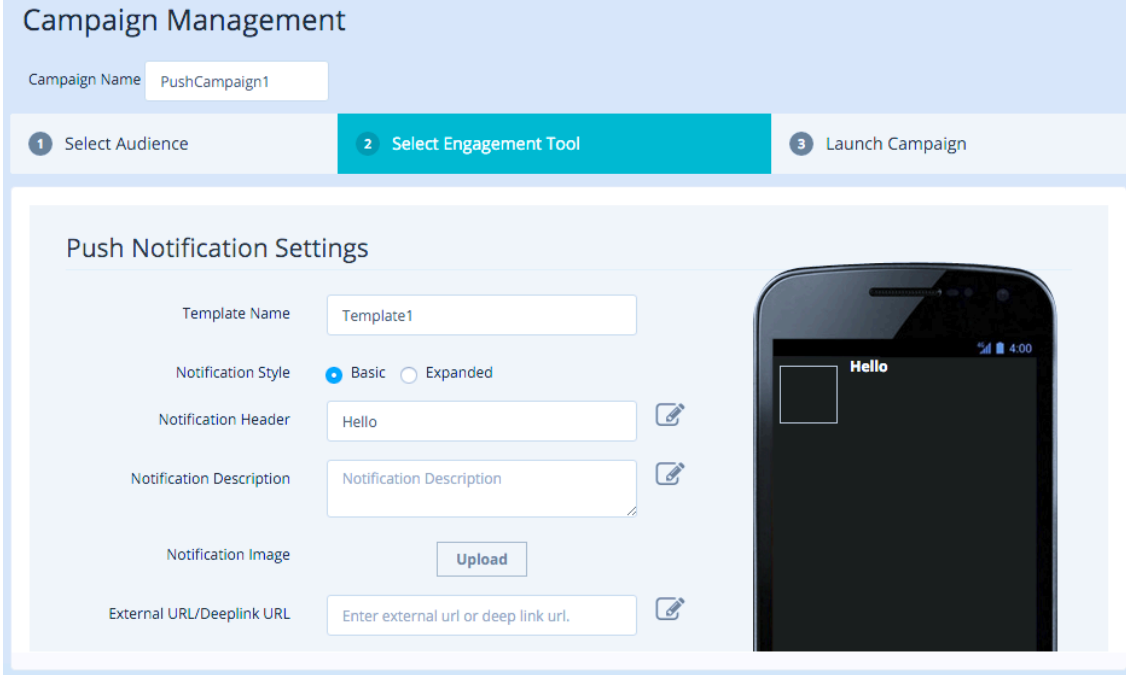
// set custom variable for any string object

[[appICE sharedInstance]setCustomVariableString:@"name" value:@"appICE"];

Using ICE-Tags in Notifications

Once you have custom data setup. You can use them for setting up tokens for campaigns. To personalize campaign contents.

Click on the edit buttons next to ‘Notification Header’ OR ‘Notification Description’ to add ICE-Tags in the content.



The screenshot shows the 'Campaign Management' interface. At the top, there's a 'Campaign Name' field with the value 'PushCampaign1'. Below this is a progress bar with three steps: '1 Select Audience', '2 Select Engagement Tool' (which is highlighted in blue), and '3 Launch Campaign'. The main content area is titled 'Push Notification Settings' and contains several form fields: 'Template Name' (value: Template1), 'Notification Style' (radio buttons for 'Basic' and 'Expanded', with 'Basic' selected), 'Notification Header' (value: Hello), 'Notification Description' (value: Notification Description), 'Notification Image' (with an 'Upload' button), and 'External URL/Deeplink URL' (placeholder: Enter external url or deep link url.). To the right of these fields is a preview of a smartphone screen displaying a notification with the text 'Hello' and a small square icon.

Selecting edit buttons next to 'Notification Header' OR 'Notification Description' etc. Helps you to easily add up tokens in the text content wherever current cursor is in selected text field.

And

In Push Notification campaign you can set only Notification Description in basic style of Push notification which will show in PushNotification content , you don't need to add image in Push notification because in iOS the application icon will show instead of image.

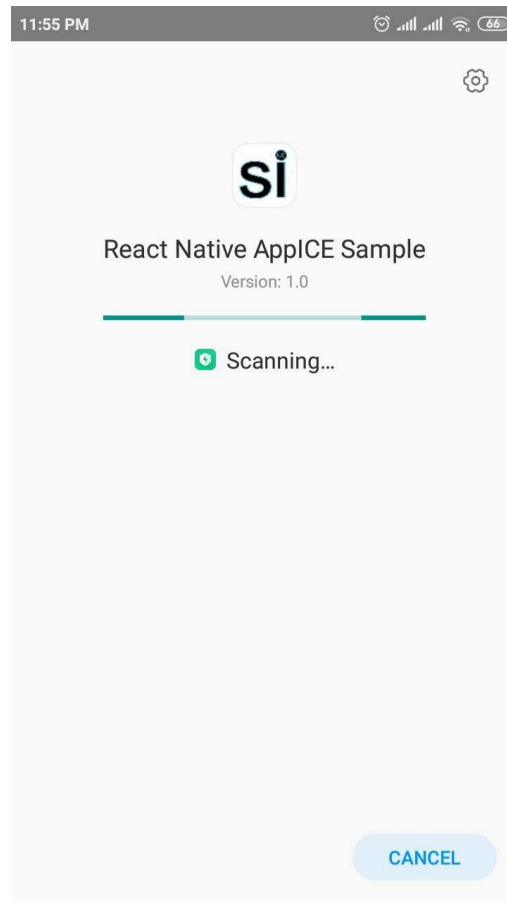
You can select any field name to insert in text content.

Field Name is the custom variable sent via sdk to server.

Default Value is the value which will be replaced against token if FieldName doesn't exist on any phone in app.

Install App on your iOS phone

1. Download & install app on your iOS phone.



2. Launch/Open/Initialize app. It shows three functions - "Record Event", "Custom Variable" & "UserId".

EventName

Key

value

Record Event

Custom EventName

Custom Value

Custom variable

UserId

UserId

3. For Record Event, put 3 values:

EventName

Key

Value

and click on the 'Record Event' button.

Test

KeyName

Demo

Record Event

Custom EventName

Custom Value

Custom variable

UserId

UserId

4. For Custom Variable, put 2 values:

Custom EventName

Custom Value

and click on the 'Custom Variable' button.

EventName

Key

value

Record Event

CustomTest

CustomVal

Custom variable

UserId

UserId

5. For UserId, put 1 value

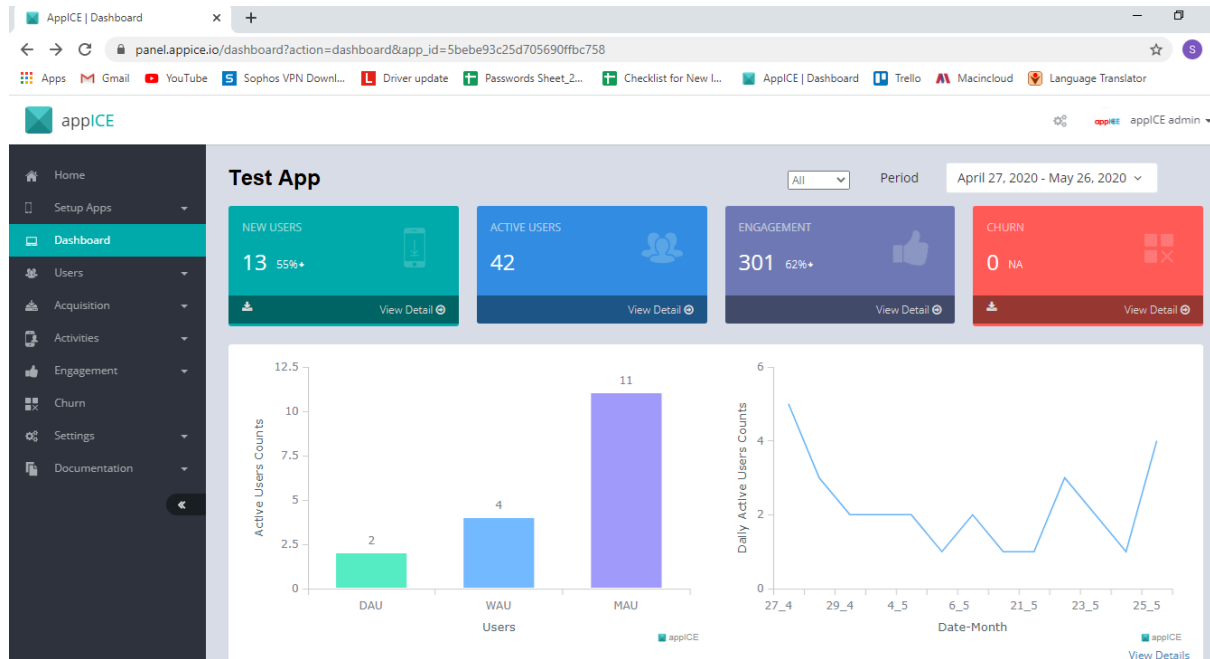
UserId

and click on the 'UserId' button.

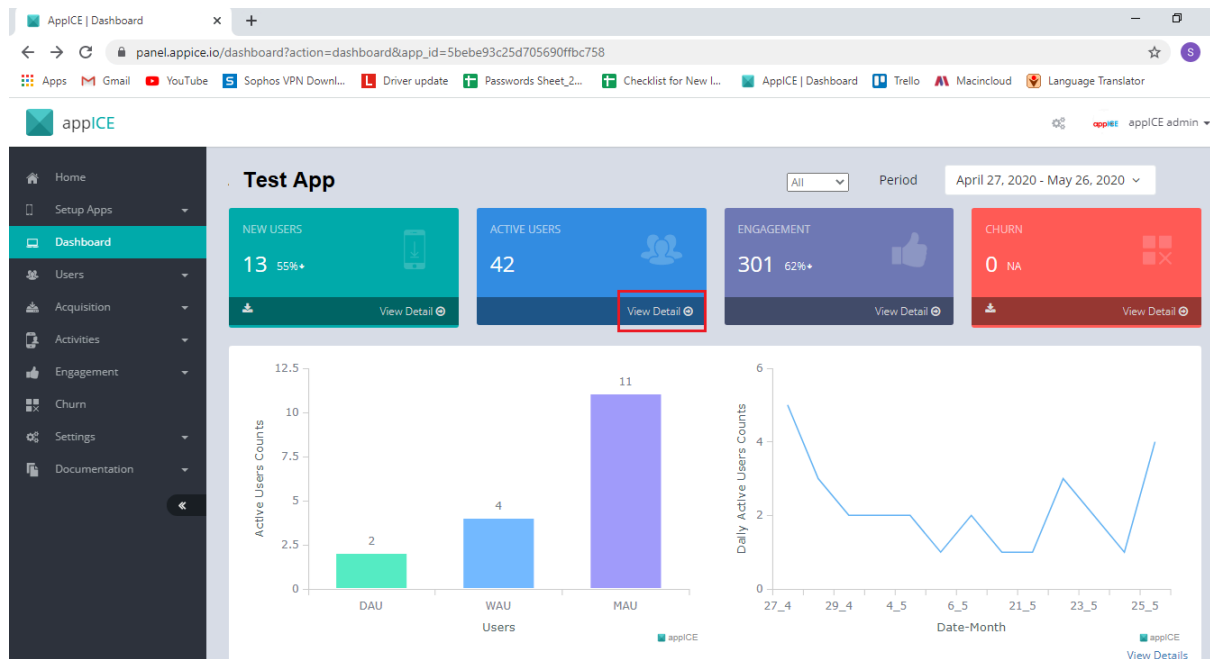
EventName	Key	value
Record Event		
Custom EventName	Custom Value	
Custom variable		
Dummy123		
UserId		

Verify integration on appICE panel

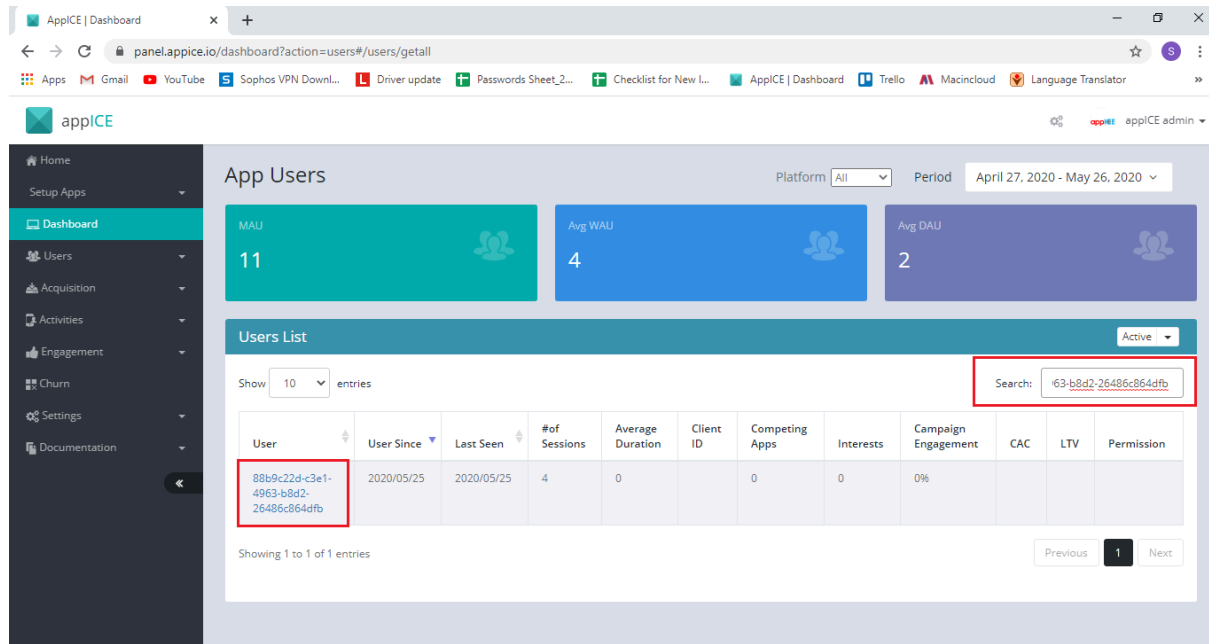
1. Login to panel.appice.io to see these values. Click on your app and go to its dashboard.



2. Click on 'Active Users' ☐ 'View Detail' to see details of your phone app.



3. Clicking on 'View Detail' takes to App Users page. Search your device Id in the Search box to search for your phone Ads Id.



App Users

Platform: All | Period: April 27, 2020 - May 26, 2020

MAU: 11 | Avg WAU: 4 | Avg DAU: 2

Users List (Active)

Show 10 entries

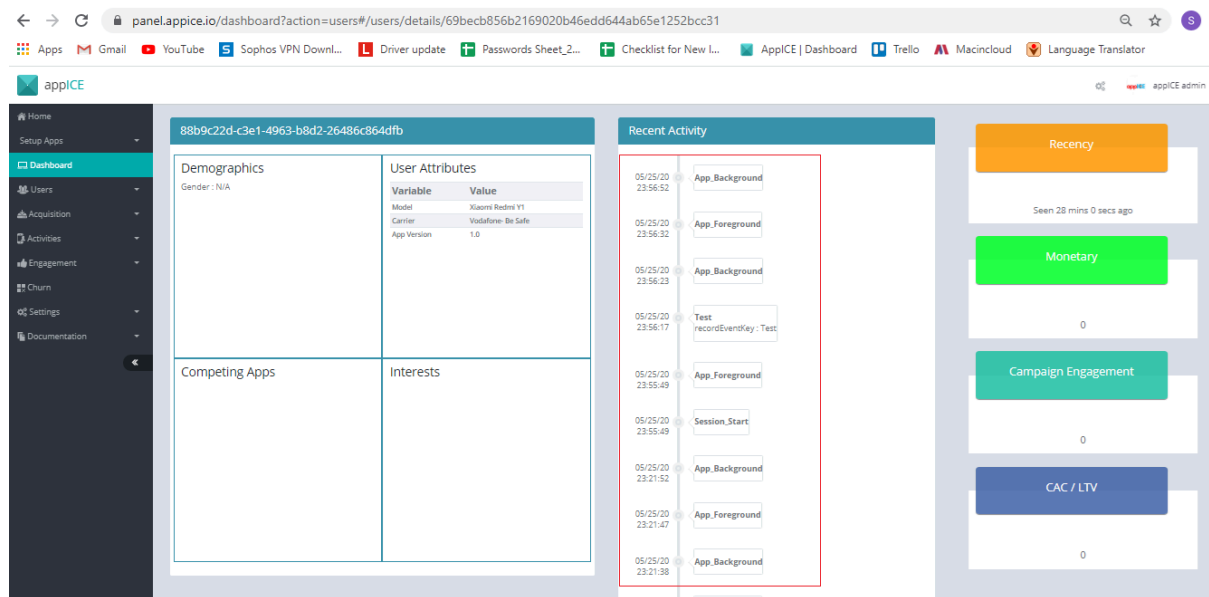
Search: 63-b8d2-26486c864dfb

User	User Since	Last Seen	# of Sessions	Average Duration	Client ID	Competing Apps	Interests	Campaign Engagement	CAC	LTV	Permission
88b9c22d-c3e1-4963-b8d2-26486c864dfb	2020/05/25	2020/05/25	4	0		0	0	0%			

Showing 1 to 1 of 1 entries

Previous 1 Next

4. Click on the searched Ad Id to see details:



88b9c22d-c3e1-4963-b8d2-26486c864dfb

Demographics
Gender: N/A

User Attributes

Variable	Value
Model	Xiaomi Redmi Y1
Carrier	Vodafone- Be Safe
App Version	1.0

Competing Apps

Interests

Recent Activity

Timestamp	Event
05/25/20 23:56:52	App_Background
05/25/20 23:56:32	App_Foreground
05/25/20 23:56:23	App_Background
05/25/20 23:56:17	Test recordEventKey: Test
05/25/20 23:55:49	App_Foreground
05/25/20 23:55:49	Session_Start
05/25/20 23:21:52	App_Background
05/25/20 23:21:47	App_Foreground
05/25/20 23:21:38	App_Background

Recency
Seen 28 mins 0 secs ago

Monetary
0

Campaign Engagement
0

CAC / LTV
0