GA4のアプリログ検証のため、SwfitUIでネイティブアプリを作成してみた

2022年 2月 18日 (金)

こんにちは、データアナリストの左海です。

最近アサインされたプロジェクトにてアプリ/WebのGoogle Analytics 4(以下GA4)での計測を検討していましたが、私自身アプリもGA4も知見がなかったため、SwiftUIでネイティブアプリを作成しログ検証を行うことにしました。

※なお、Apple Developer Programを契約していないためiOS Simulatorを使用しています。

実行環境

- XcodeのVersion：13.2.1 (13C100)
- iOS Simulatorの名前：iPhone 13
- iOSのversion：iOS 15.2

最終的にできたもの

以下の通り、テストアプリを作成しGA4のDebugViewでscreen_viewイベント、カスタムイベントが意図したタイミングで飛んでいるか検証可能になりました。

atsuhiro sakai

mediba Creator × Engineer Blog

Design
Technology

Corporate
PR Blog

作業手順

以下の作業手順の通りに進めていきます。

1. SwiftUIでネイティブアプリの作成を行う
2. ネイティブアプリへFirebase SDKを追加する
3. GA4のDebugViewでログを検証する

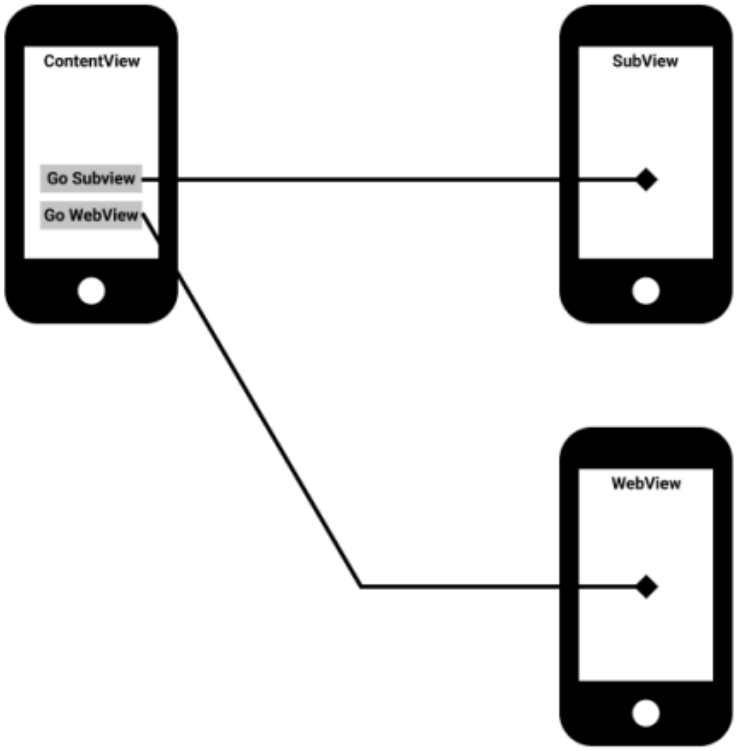
それでは、ネイティブアプリ作成の流れ、Firebase SDKの追加、GA4を用いたログ検証について説明していきます。

1.SwiftUIでネイティブアプリの作成を行う

画面遷移のイメージ

まずは、XcodeでFirebase SDK追加に必要なアプリの作成を行います。 ここでは、3つの画面を作成します。画面遷移は添付画像のようなイメージです。

- ContentView
- SubView
- MyWebView



ContentViewの作成

ContentViewから作成します。ContentViewでは、SubView、MyWebViewそれぞれに遷移するリンクを作成し、Buttonも画面右上部に配置します。加えて、イベント送信処理も追加します。

画面遷移は以下の通り NavigationLink を用います。 destination には遷移先のView名を、 Text() にはリンクに表示する文字列を記述します。

```
NavigationLink(destination: SubView()) {
  Text("Go SubView")
}
NavigationLink(destination: MySecondWebView(urlString: "https:
  Text("Go WebView")
}
```

screen_viewイベントはインスタンスメソッド onAppear で ContentViewが描画されたタイミングで送信します。 イベントパラメーターとして、 screen_name と screen_class を定義しておきま

す。 同様の処理をSubView、MyWebViewでも記述していきます。

```
.onAppear() {
  Analytics.logEvent(AnalyticsEventScreenView,
    parameters: [AnalyticsParameterScreenName: "\(ContentView.
    AnalyticsParameterScreenClass: "\(ContentView.self)"]
  )
}
```

次に、画面右上部のButtonをタップした際に送信するカスタムイベント button_tap を追加します。イベントパラメータとして button_name を定義します。今回送信するイベント送信処理の記述は以上です。

```
Button(action: {
  let button_name = "hoge"
  Analytics.logEvent("button_tap",
    parameters: ["button_name" : button_name as NSObject,]
  )
}
```


SubViewの作成

ContentViewの遷移先であるSubViewの作成を行います。ここではダミーの遷移先を設定しておきました。

MyWebViewの作成

ContentViewの遷移先であるMyWebView画面の作成を行います。MyWebView画面は私が作成したWebサイトを表示します。WebサイトはFirebase Hostingで公開しています。

これでアプリの作成は終了です。以下の通り、問題なくビルドすることができました。

app

atsuhiro sakai

mediba Creator × Engineer Blog

Design
Technology

Corporate PR Blog

2.アプリへFirebase SDKを追加する

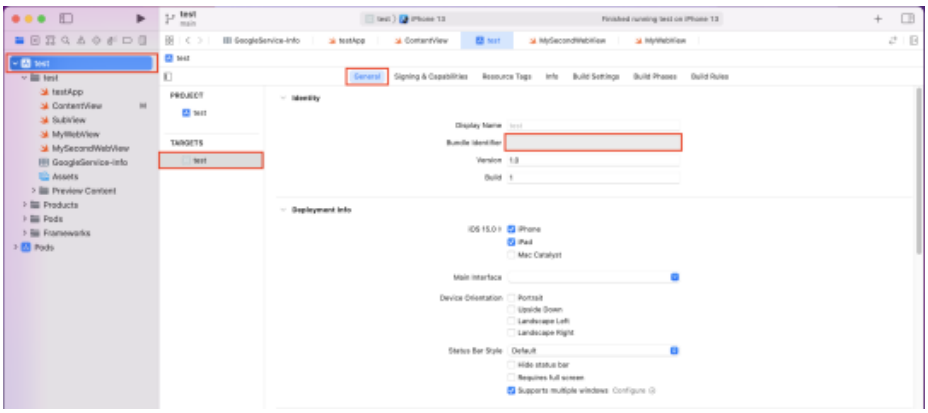
アプリに接続するFirebase プロジェクトを新規作成する

作成したアプリにFirebase SDKを追加します。まずは、Firebase コンソールでアプリに接続するための Firebase プロジェクトを新規作成します。

作成したアプリをFirebaseに登録する

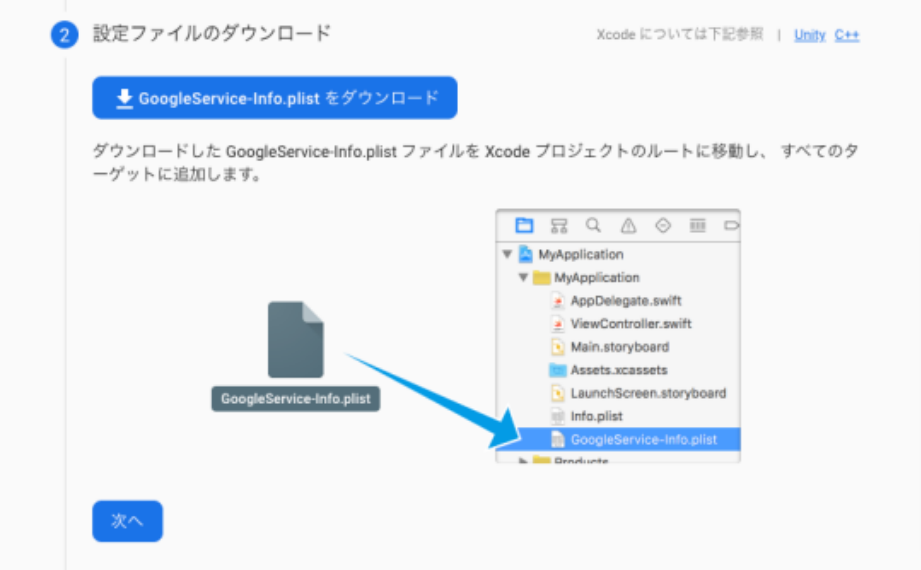
プロジェクト概要 > iOS+ アイコン > 設定ワークフローに進みます。すると、Apple バンドルIDの入力を求められます。

ナビゲーターエリアの上のプロジェクト名 > TARGETS のプロジェクト名 > General タブでApple バンドルIDを確認できます。



設定ファイルのダウンロード

ダウンロードした GoogleService-Info.plist ファイルをXcode プロジェクトのルートに移動し、すべてのターゲットに追加します。



Firebase SDKを追加する

Firebase SDKはいくつかインストール方法がありますが、今回はCocoaPodsを利用します。

Podfileを作成する

プロジェクトディレクトリのルートから次のコマンドで、Podfileを作成します。

```
pod init
```

PodfileにFirebaseAnalyticsを追加する

アプリで使用する'Firebase/Analytics'をPodfileに追記します。

```
# Add the Firebase pod for Google Analytics
pod 'Firebase/Analytics'
# For Analytics without IDFA collection capability, use this p
# Add the pods for any other Firebase products you want to use
# pod 'Firebase/Auth'
# pod 'Firebase/Firestore'
```

Podをインストールする

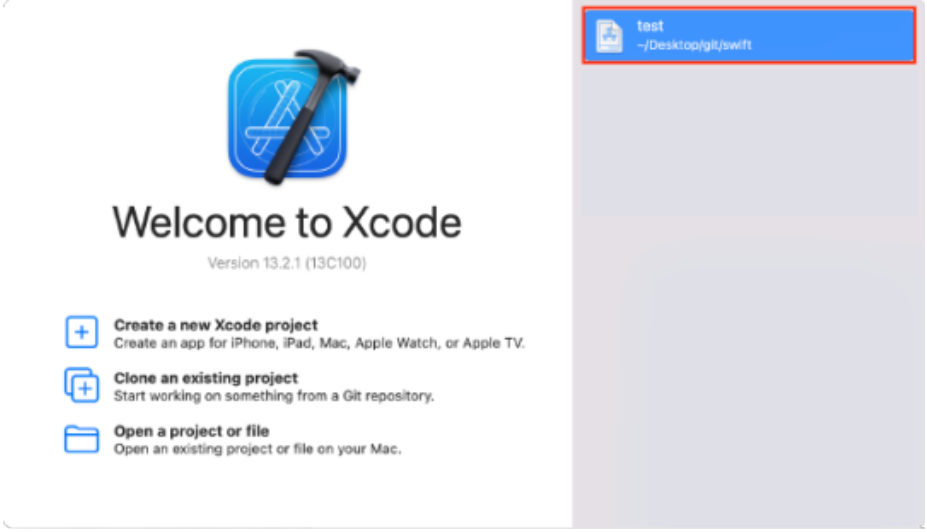
pod install を実行します。ただし、M1 Macだと上手くインストールできないことがあるようです。 私の環境下ではターミナルのアーキテクチャに依存していたので以下のページを参考に、タ

ーミナル > 右クリックで「情報を見る」 > 「Rosettaを使用して開く」 に事前にチェックを入れます。そうすることで、問題なくPodがインストールできました。

【参考】M1 Macでpod installを実行する


```
pod install
```

Podインストールが完了すると、 .xcworkspace ファイルが作成されます。以降は .xcworkspace ファイルからXcodeプロジェクトを開きます。



初期化コードを追加する

アプリ起動時に Firebase を接続するには、Xcodeプロジェクトの@mainアトリビューションの指定の場所に指定のコードを追加します。


```
import SwiftUI
import Firebase //追記
@main
struct testApp: App {
  var body: some Scene {
    WindowGroup {
    ContentView()
    }
  }
  init() {
   FirebaseApp.configure() //追記
  }
}
```

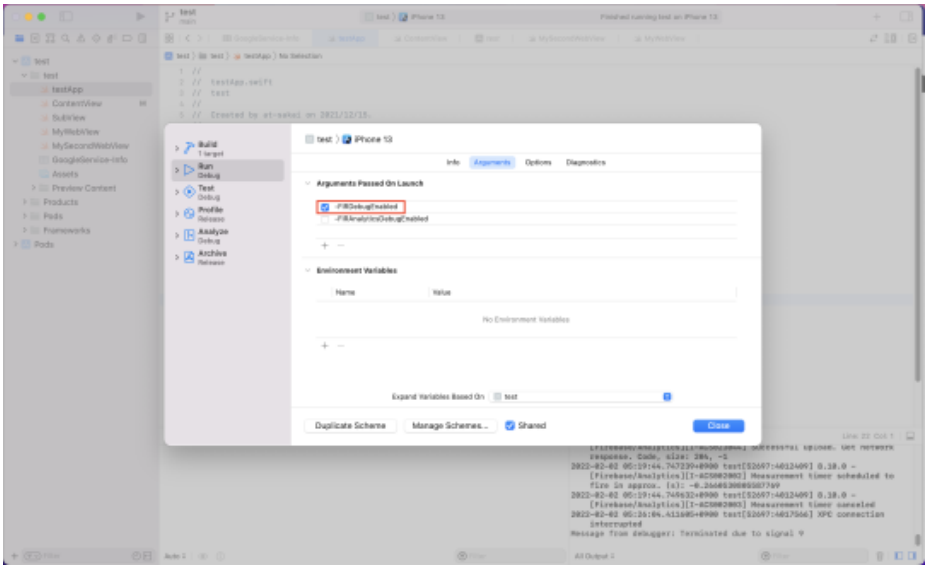
これで、Firebase SDKの追加は完了です。

3.GA4のDebugViewでログ検証を行う

最後に、GA4でログ検証を行います。早速ログ検証といきたいところですが、事前にいくつか設定を行う必要があります。

Debug Modeを有効にする

Xcodeを開き、メニューバー > Product > Scheme > Edit Scheme > Run > ArgumentsのArguments Passed On Launchに -FIRDebugEnabled を追加することで、Debug Modeを有効にします。

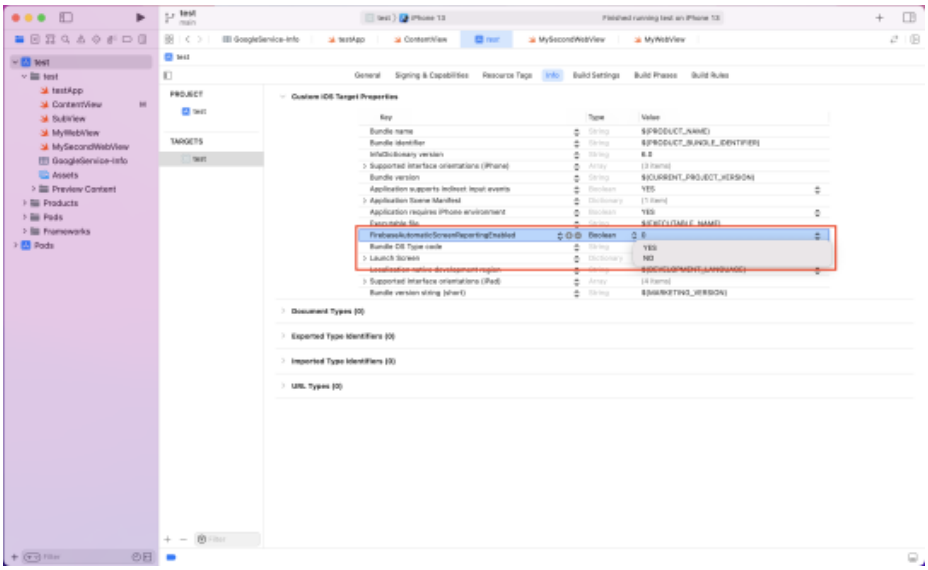


Firebaseの自動ログ取得をオフにして手動ログ取得を設定する

今回SwiftUIとの相性かもしれませんが、自動ログ取得では画面遷移でscreen_viewイベントが飛んだり飛ばなかったりしていたので、手動でログ取得をする実装にしています。

Info.plist ファイルに以下を追加することで自動ログ収集をオフにします。

- Key：FirebaseAutomaticScreenReportingEnabled
- Value：NO(Boolean)



デバック時にXcodeのReport Navigatorで以下のようなログが確認できれば、自動ログ収集がオフになっています。

Analytics screen reporting is disabled. UIViewController transitions will not be logged.

ログ検証

Debug Viewでscreen_viewイベント、buttun_tapイベントが計測されているのが分かります。問題なくログが飛んでいます、検証完了です。

screen_view

atsuhiro sakai

00:27

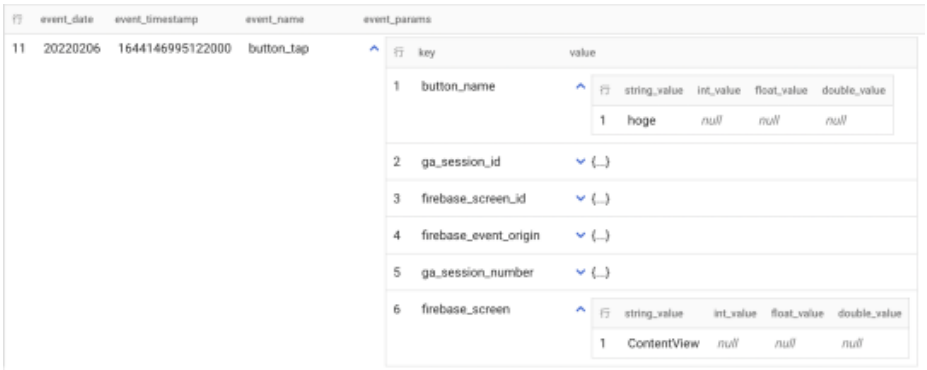
button_tap

atsuhiro sakai

00:32

BigQuery連携

BigQueryに連携してエクスポートされているログを確認しました。添付画像のカスタムイベントbutton_tapのログは実装した通り、button_name に設定した値 hoge が格納されています。



	event_date	event_timestamp	event_name	event_params
11	20220206	1644146995122000	button_tap	

	key	value
1	button_name	
2	ga_session_id	{...}
3	firebase_screen_id	{...}
4	firebase_event_origin	{...}
5	ga_session_number	{...}
6	firebase_screen	

	string_value	int_value	float_value	double_value
1	hoge	null	null	null

	string_value	int_value	float_value	double_value
1	ContentView	null	null	null

おわりに

今回無事にアプリを完成させてログ検証を行うことができました。私自身プログラミング未経験だったため、取り組む前は分からないことだらけでした。具体的には、プログラミングの基礎がなく、エラーの解決方法が分かりませんでした。他にもターミナルやGitHubで何ができるのかすら知りませんでした。

しかし、Udemyで学習したり、分からないことを一つ一つ検索エンジンで調べることで問題解決することができました。アプリ制作の場面はエラーの連続でしたが、エラーを解決した瞬間が最も楽しかったです。

そして、何を取り組むにしても「やればできる」と自信がつくようになりました。これからもさまざまなことにチャンレンジしていきます。

どなたかの参考になれば幸いです。