Haptik Webkit Flutter Integration Guide

Quick Start

Prerequisites

To Integrate Haptik SDK, you must obtain the following credentials to get started

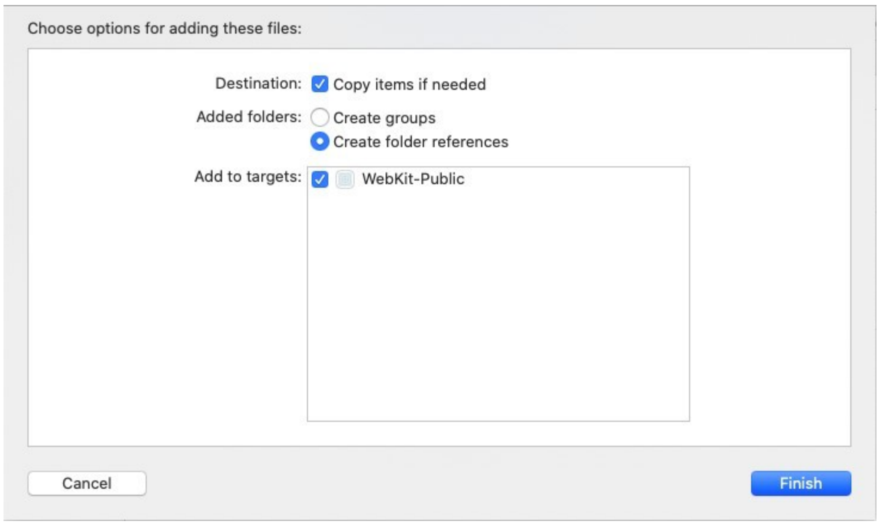
- Client ID: Unique Haptik identification number associated with your account
- Business ID: Unique business identification number
- Base URL: Link pointing to specific server environment (Different for Staging and Production)
- Auth Type: authentication type to be used for user registration
- Auth ID: authentication type to be used for user registration

Android – minSdkVersion 21, targetSdkVersion 30

Getting Started

1. **Create a Flutter Project**
    1. Create a new Flutter Project, if you have not already created one.
    2. Once created, add a dependency on the "haptik_sdk" package in your "pubspec.yaml" file. You can either do this manually by navigating to the file and adding the dependency, or you can enter this command in your terminal "flutter pub add haptik_sdk".
    3. Your project is now ready to integrate the Haptik Webkit.

2. **iOS Integration**
    1. **Install Haptik SDK** - For using the Haptik Webkit on iOS devices via Flutter, we will first have to install the Haptik SDK in our iOS files. To do this, you can open the iOS folder in Xcode.
        - **Manual Installation**
            1. Download the HPWebKit.zip file and unzip it.
            2. Drag and drop the downloaded XCFramework into your Xcode project.
            3. If the framework's symbols are unable to be loaded, navigate to the "General" pane of your target and find the "Frameworks, Libraries, and Embedded Content" dropdown. Switch HPWebKit.xcframework from "Do Not Embed" to "Embed and Sign"



Choose options for adding these files:

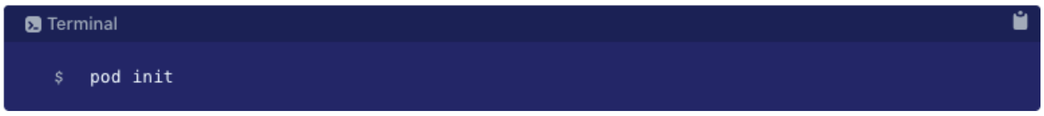
Destination: [✓ Copy items if needed]

Added folders: Create groups [✓ Create folder references]

Add to targets: [✓ WebKit-Public]

4. You will also need to install a HPWebKit dependency.

- **Auto Installation - CocoaPods**
  1. If you haven't already done so, install a recent version of CocoaPods.
  2. If you don't have an existing Podfile, run the following command to create one:

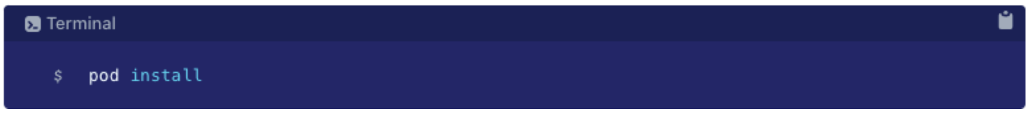


```
$ pod init
```

  3. Add this line to your Podfile:

```
use_frameworks!

target YourTargetName do pod 'HPWebKit'
end
```

  4. Run the following command:



```
$ pod install
```

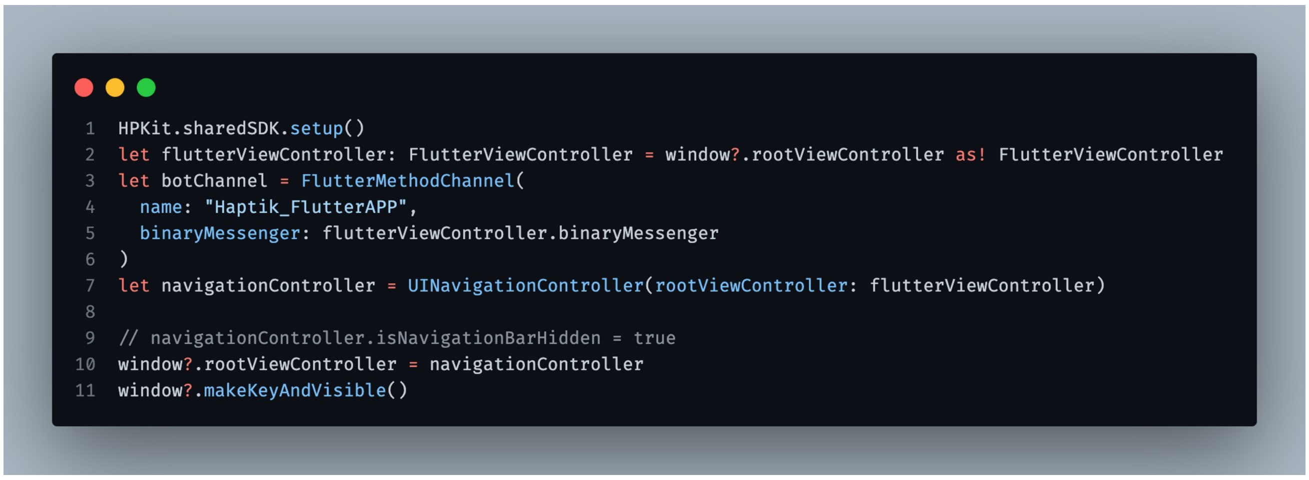
2. **Setup Credentials & Environment** - To define your credentials for the iOS build of your application we need to define them in "info.plist" file. Define a dictionary with the key – "HaptikLib" and add in the following credentials you were given after registering as a Haptik partner:
   - Client ID
   - Base URL (Different for Staging and Production)
   - Business ID

```
<key>HaptikLib</key>
<dict>
<key>baseUrl</key>
<string>insert_base_url</string>
<key>businessID</key>
<string>insert_busines_id</string>
<key>clientID</key>
<string>insert_client_id</string>
<key>runEnvironment</key>
<string>1</string>
</dict>
```

Note: While releasing the application on the App Store, only the Production URL (provided by a Haptik) should be used and the runEnvironment should be 0.

3. **Permissions** - HaptikLib requires some basic permissions which almost every application takes to function properly. Make sure to add them else iOS will assert the application when the SDK will try to present the prompt if the respective permission is not granted by the user.

4. **Initialize iOS SDK** – You need to initialize our Haptik SDK before you can use it. Head to the AppDelegate.swift file present in your Flutter Projects "iOS/Runner" directory.



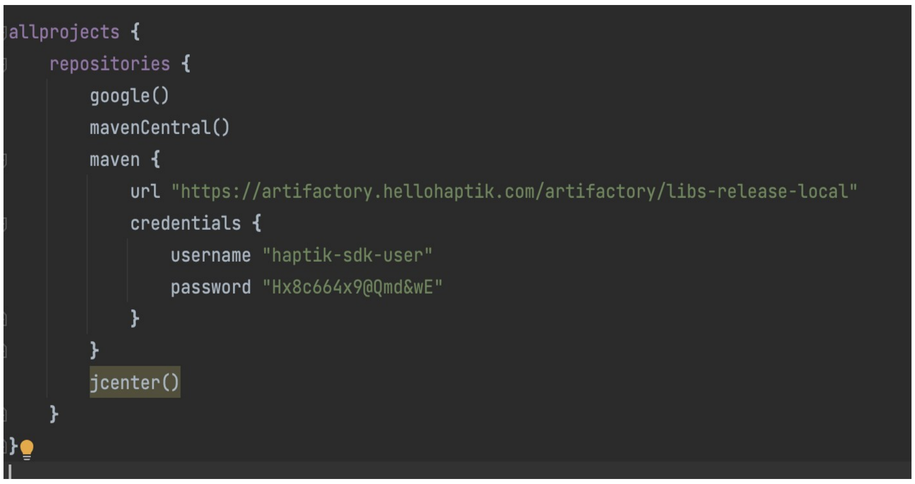
```swift
HPKit.sharedSDK.setup()
let flutterViewController: FlutterViewController = window?.rootViewController as! FlutterViewController
let botChannel = FlutterMethodChannel(
  name: "Haptik_FlutterAPP",
  binaryMessenger: flutterViewController.binaryMessenger
)
let navigationController = UINavigationController(rootViewController: flutterViewController)

// navigationController.isNavigationBarHidden = true
window?.rootViewController = navigationController
window?.makeKeyAndVisible()
```

Add the above code under the "GeneratedPluginRegistrant.register(with: self)" line.
Here, we are creating a FlutterViewController that will be used by our Flutter App show the Conversation on a new screen.
Then we are creating a MethodChannel, which our Flutter code will use to communicate with the native iOS code and execute some funcationality.
Finally, we are creating a UINavigationController which will be used to navigate to the FlutterViewController to display the conversation.

3. **Initialize Android SDK**

1. Install Haptik SDK - For using the Haptik Webkit on Android devices via Flutter, we will first have to install the Haptik SDK in our android files. To do this, you can open the android folder in Android Studio.

- Add the following lines of code to your project's root settings.gradle or build.gradle.



```groovy
allprojects {
    repositories {
        google()
        mavenCentral()
        maven {
            url "https://artifactory.hellohaptik.com/artifactory/libs-release-local"
            credentials {
                username "haptik-sdk-user"
                password "Hx8c664x9@Qmd&wE"
            }
        }
        jcenter()
    }
}
```

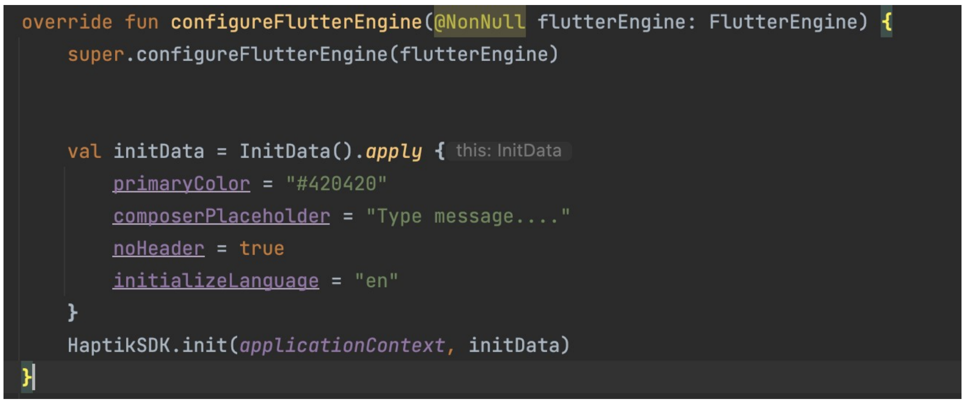
- Add the following to your app's **build.gradle** file:

```
dependencies {
    .
    .
    implementation 'ai.haptik.android.sdk:haptiklib-wrapper:1.7.1'
    .
    .
}
```

- Haptik SDK takes in required credentials through your app's strings.xml file. Add base_url, business_id, and client_id in your strings.xml:

```
<resources>
    .
    .
    <string name="haptik_base_url">PROVIDED_BASE_URL</string>
    <string name="haptik_client_id">PROVIDED_CLIENT_ID</string>
    <string name="haptik_business_id">PROVIDED_BUSINESS_ID</string>
</resources>
```

- The SDK must be initialized in your MainActivity.kt before using it. Add this code to initialise it. InitData can be modified as per needs:



```
override fun configureFlutterEngine(@NonNull flutterEngine: FlutterEngine) {
    super.configureFlutterEngine(flutterEngine)


    val initData = InitData().apply { this: InitData
        primaryColor = "#420420"
        composerPlaceholder = "Type message...."
        noHeader = true
        initializeLanguage = "en"
    }
    HaptikSDK.init(applicationContext, initData)
}
```

These are the steps needed to initialize the SDK on both iOS and Android.

4. There are two ways to register a user.

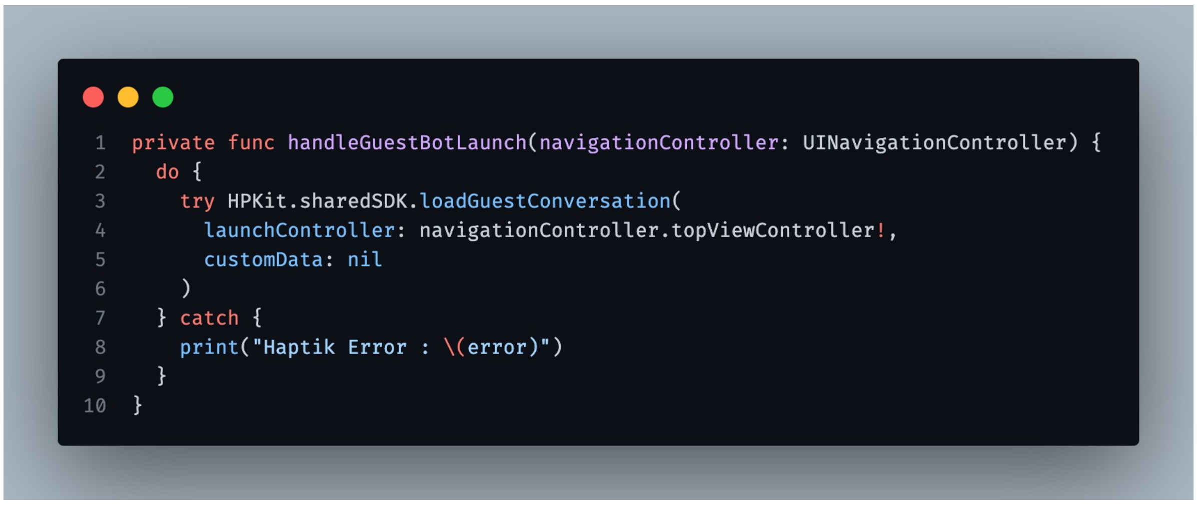
- **Guest**
- **Custom Client Sign Up**

For you to take the user to the conversation screen where the user interacts/chats with the bot/agent, you need to provide user details to SDK

Guest: SDK creates a new user, without any specific details

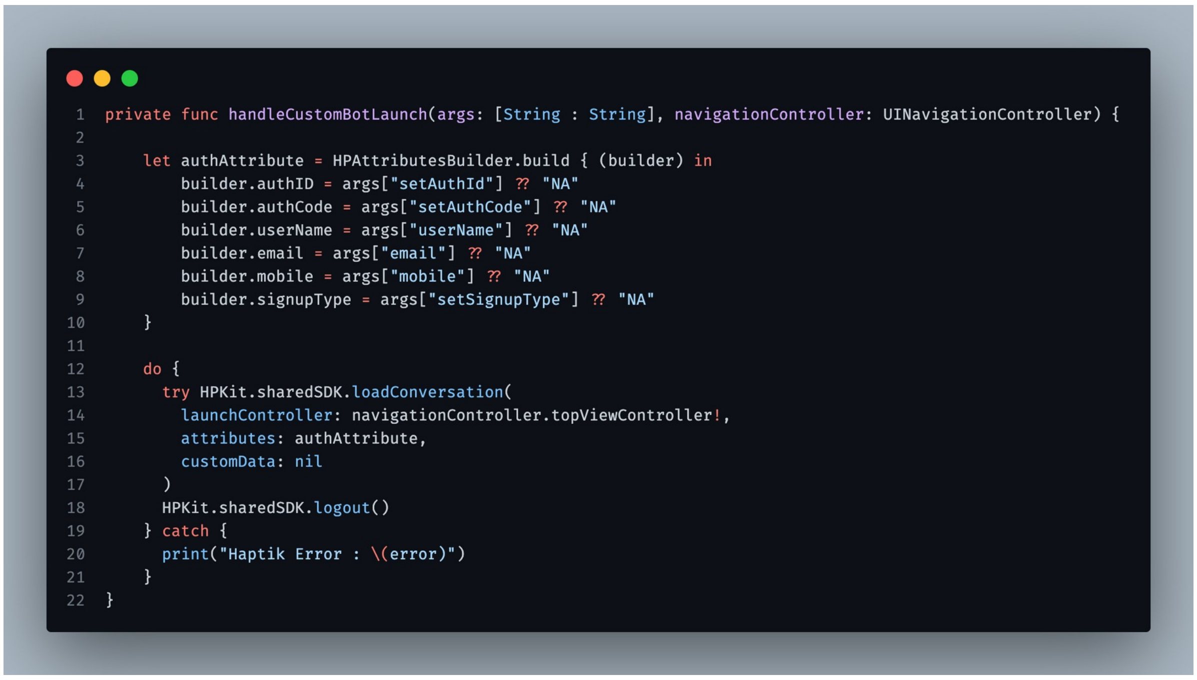
Custom: For cases when the client/parent application already has a signup flow in place and wants to link the same user to the SDK, we support Custom Signups.

We will create 2 functions that will initiate the respective ways to register a user.



```swift
private func handleGuestBotLaunch(navigationController: UINavigationController) {
  do {
    try HPKit.sharedSDK.loadGuestConversation(
      launchController: navigationController.topViewController!,
      customData: nil
    )
  } catch {
    print("Haptik Error : \(error)")
  }
}
```

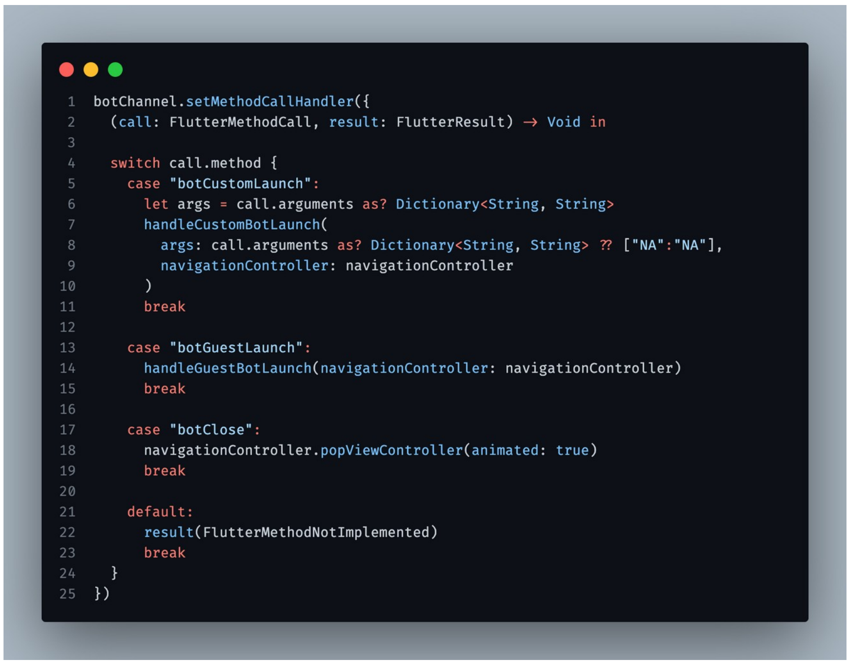
Guest Conversation (iOS)



```swift
private func handleCustomBotLaunch(args: [String : String], navigationController: UINavigationController) {

    let authAttribute = HPAttributesBuilder.build { (builder) in
        builder.authID = args["setAuthId"] ?? "NA"
        builder.authCode = args["setAuthCode"] ?? "NA"
        builder.userName = args["userName"] ?? "NA"
        builder.email = args["email"] ?? "NA"
        builder.mobile = args["mobile"] ?? "NA"
        builder.signupType = args["setSignupType"] ?? "NA"
    }

    do {
      try HPKit.sharedSDK.loadConversation(
        launchController: navigationController.topViewController!,
        attributes: authAttribute,
        customData: nil
      )
      HPKit.sharedSDK.logout()
    } catch {
      print("Haptik Error : \(error)")
    }
}
```

Custom Client Conversation (iOS)

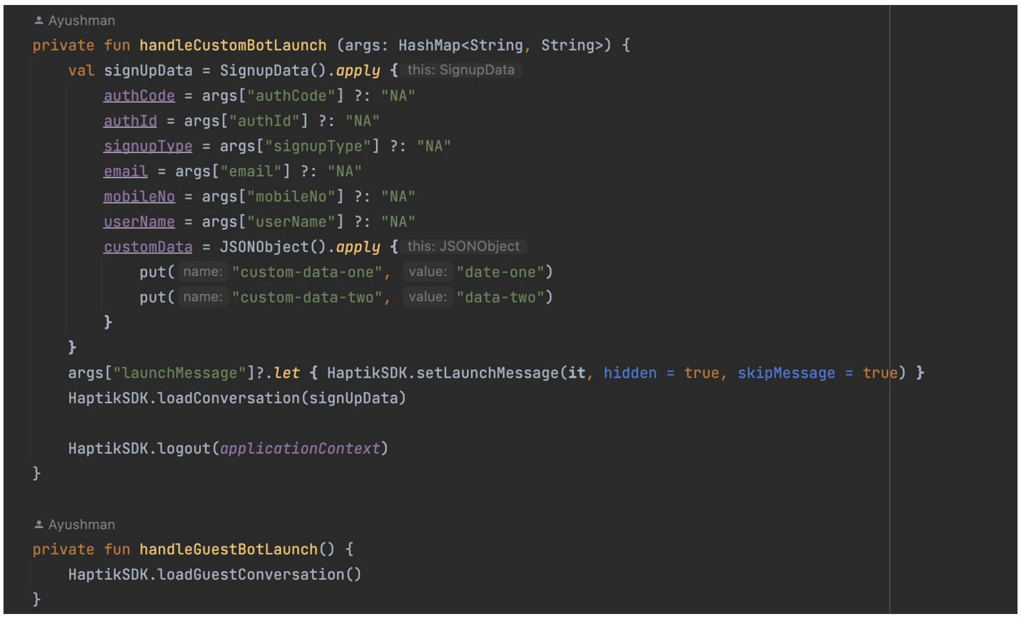
To use these functions we need to handle our MethodChannel Calls. We will do this using setMethodCallHandler.



```swift
botChannel.setMethodCallHandler({
  (call: FlutterMethodCall, result: FlutterResult) -> Void in

  switch call.method {
    case "botCustomLaunch":
      let args = call.arguments as? Dictionary<String, String>
      handleCustomBotLaunch(
        args: call.arguments as? Dictionary<String, String> ?? ["NA":"NA"],
        navigationController: navigationController
      )
      break

    case "botGuestLaunch":
      handleGuestBotLaunch(navigationController: navigationController)
      break

    case "botClose":
      navigationController.popViewController(animated: true)
      break

    default:
      result(FlutterMethodNotImplemented)
      break
  }
})
```

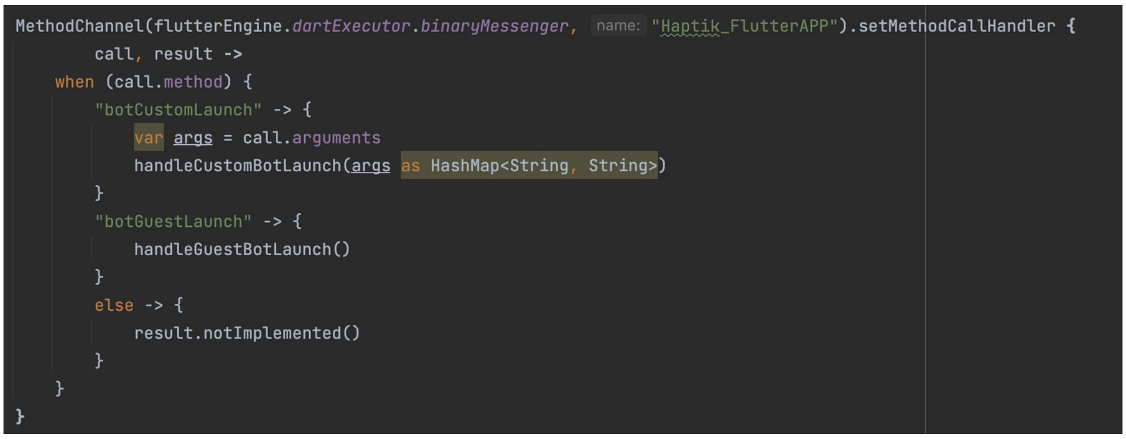
Add this section of code, under the “window?.makeKeyAndVisible()” line in your iOS/Runner/AppDelegate.swift file.

Similarly, for android we make a methodChannel and create 2 functions to handle the bot launches:



```kotlin
Ayushman
private fun handleCustomBotLaunch (args: HashMap<String, String>) {
    val signUpData = SignupData().apply { this: SignupData
        authCode = args["authCode"] ?: "NA"
        authId = args["authId"] ?: "NA"
        signupType = args["signupType"] ?: "NA"
        email = args["email"] ?: "NA"
        mobileNo = args["mobileNo"] ?: "NA"
        userName = args["userName"] ?: "NA"
        customData = JSONObject().apply { this: JSONObject
            put( name: "custom-data-one",  value: "date-one")
            put( name: "custom-data-two",  value: "data-two")
        }
    }
    args["launchMessage"]?.let { HaptikSDK.setLaunchMessage(it, hidden = true, skipMessage = true) }
    HaptikSDK.loadConversation(signUpData)

    HaptikSDK.logout(applicationContext)
}

Ayushman
private fun handleGuestBotLaunch() {
    HaptikSDK.loadGuestConversation()
}
```

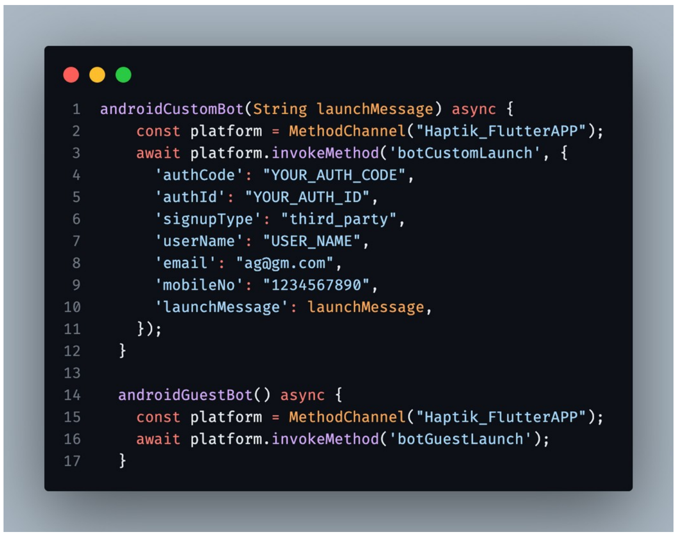
Refer to [this] to view the more attributes and refer to [this] for customization options.
To use these functions, we have to implement the methodChannel on the android native side as well.



```
MethodChannel(flutterEngine.dartExecutor.binaryMessenger, name: "Haptik_FlutterAPP").setMethodCallHandler {
        call, result ->
    when (call.method) {
        "botCustomLaunch" -> {
            var args = call.arguments
            handleCustomBotLaunch(args as HashMap<String, String>)
        }
        "botGuestLaunch" -> {
            handleGuestBotLaunch()
        }
        else -> {
            result.notImplemented()
        }
    }
}
```

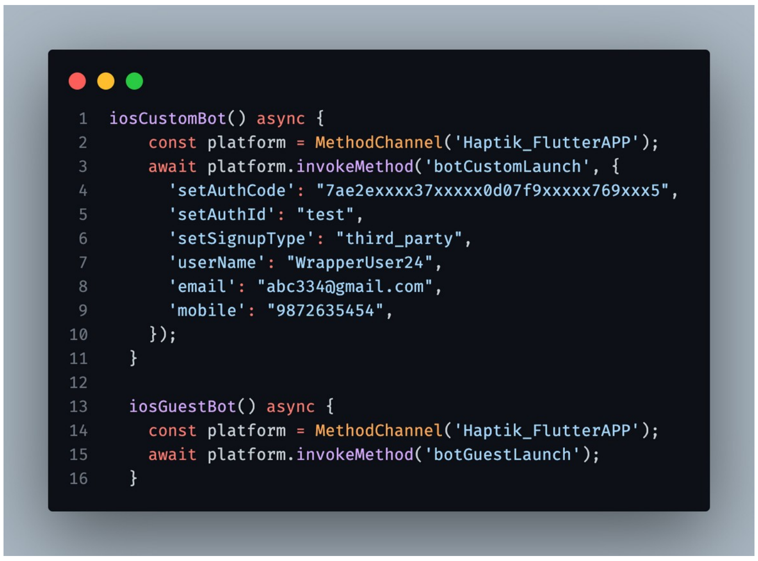
Add this method channel under the init function of the HaptikSDK in the MainActivity.kt file.

To launch the bot on Android, we create functions invoking the respective MethodChannel call.



```
androidCustomBot(String launchMessage) async {
    const platform = MethodChannel("Haptik_FlutterAPP");
    await platform.invokeMethod('botCustomLaunch', {
      'authCode': "YOUR_AUTH_CODE",
      'authId': "YOUR_AUTH_ID",
      'signupType': "third_party",
      'userName': "USER_NAME",
      'email': "ag@gm.com",
      'mobileNo': "1234567890",
      'launchMessage': launchMessage,
    });
  }

  androidGuestBot() async {
    const platform = MethodChannel("Haptik_FlutterAPP");
    await platform.invokeMethod('botGuestLaunch');
  }
```

Similarly, to launch our bot on iOS, we create their functions invoking the respective MethodChannel call.



```
iosCustomBot() async {
    const platform = MethodChannel('Haptik_FlutterAPP');
    await platform.invokeMethod('botCustomLaunch', {
      'setAuthCode': "7ae2exxxx37xxxxx0d07f9xxxxx769xxx5",
      'setAuthId': "test",
      'setSignupType': "third_party",
      'userName': "WrapperUser24",
      'email': "abc334@gmail.com",
      'mobile': "9872635454",
    });
  }

  iosGuestBot() async {
    const platform = MethodChannel('Haptik_FlutterAPP');
    await platform.invokeMethod('botGuestLaunch');
  }
```

Here, we simply initialise our platform MethodChannel and invoke the methods we created mentioned in this documentation above. For sending **custom data in iOS**, please refer to this.

These function calls can be added into any onTap() or any onPressed() of a Button in flutter and it should work as expected.
Refer to the sample project to understand the functionality better in terms of code.