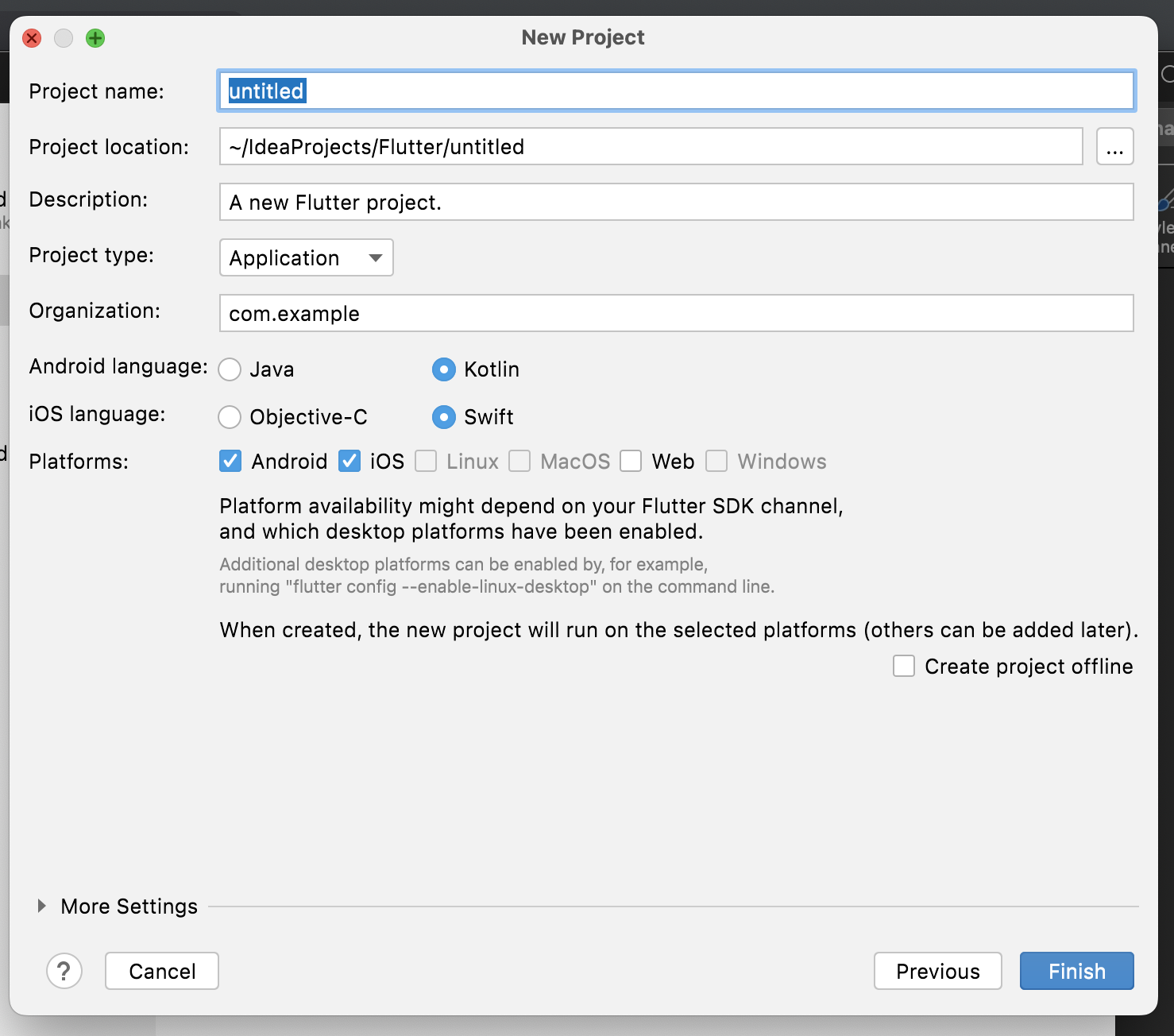
**ZELLESDK INTEGRATION FOR FLUTTER**

Flutter Application:

1. Create new flutter application with both Android and iOS platform enabled.
2. Select project type as Application and provide respective package name.
3. Select specific language support for both Android and iOS and click finish to create the project.



1. Once project has created navigate to lib folder and create dart class with respective widget to lauch Zelle.
2. To create a bridge between native and dart, First construct the channel. Use a Method channel with a single platform method that returns the data from dart to native and wise versa.
3. The client and host sides of a channel are connected through a channel name passed in the channel constructor.
4. All channel names used in single app must be unique; prefix the channel name with a unique ‘domain prefix’. Example: static const platform = MethodChannel(‘zellesdk.launch’);



1. Next, invoke a method on the method channel, specifying the concrete method to call using the String identifier **launchZelle**.
2. Pass the respective parameter to lauch zellesdk in key, value pair in invokeMethod.

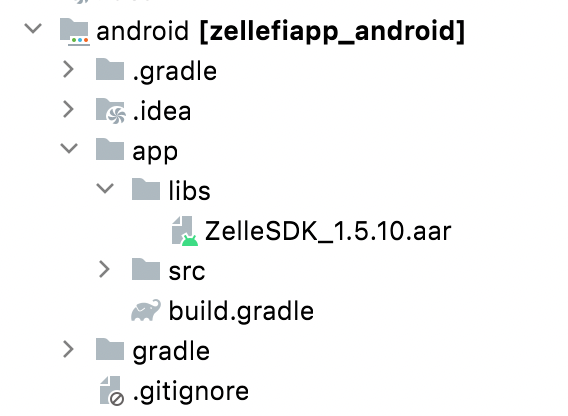
**final** String result = **await** *platform*.invokeMethod(**'launchZelle'**,{  
 **'applicationName'**: \_applicationNameController.**text**,  
 **'baseUrl'**:\_baseUrlController.**text**,  
 **"institutionId"**:\_institutionIdController.**text**,  
 **'product'**:\_productController.**text**,  
 **'ssoKey'**: \_ssoKeyController.**text**,  
 **'appData'**: pdData, *//Optional* **'parameter'**:map});

1. Use the returned result to update the user interface state.

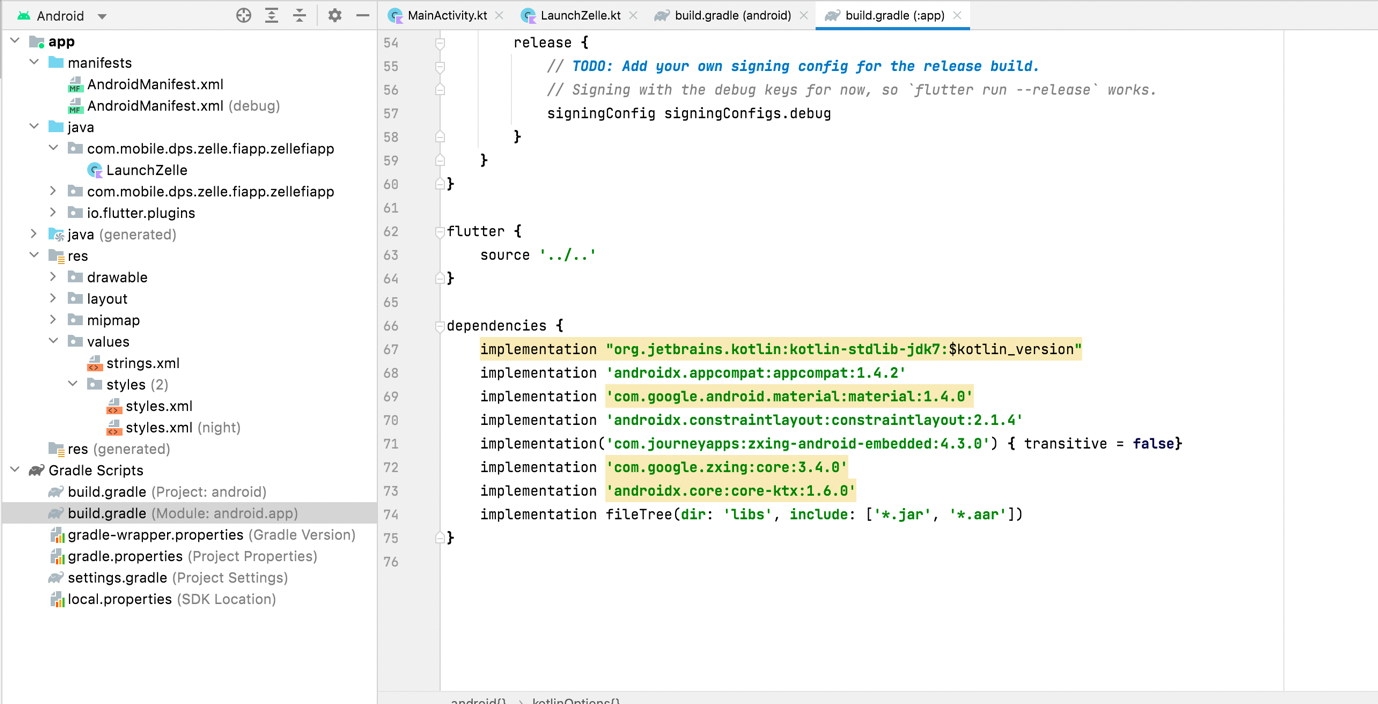


Android Platform:

1. Open Android platform inside android studio and create libs folder inside app.
2. Place the ZelleSDK.aar file inside the libs folder.



1. To add respective dependencies open build.gradle file and inside dependencies add required dependency given below.



Dependencies:

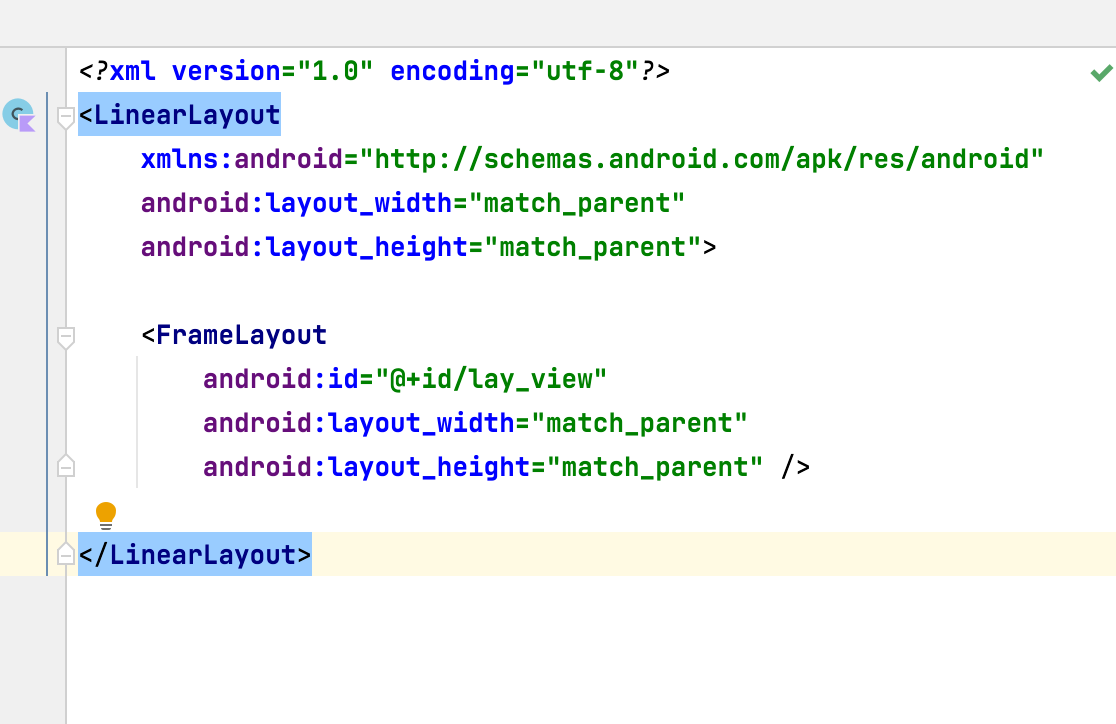
implementation **"org.jetbrains.kotlin:kotlin-stdlib-jdk7:**$kotlin\_version**"**implementation **'androidx.appcompat:appcompat:1.4.2'**implementation **'com.google.android.material:material:1.4.0'**implementation **'androidx.constraintlayout:constraintlayout:2.1.4'**implementation(**'com.journeyapps:zxing-android-embedded:4.3.0'**) **{** transitive = **false}**implementation **'com.google.zxing:core:3.4.0'**implementation **'androidx.core:core-ktx:1.6.0'**implementation fileTree(**dir**: **'libs'**, **include**: [**'\*.jar'**, **'\*.aar'**])

1. Open the file MainActivity.kt located in the **kotlin** folder in the Project view. Inside the configureFlutterEngine() method, create a MethodChannel and call setMethodCallHandler(). Make sure to use the same channel name as was used on the Flutter client side.
2. Create new activity to initialize and launch our ZelleSDK.
3. Get the arguments from the client side and pass that arguments to LaunchZelle.kt.

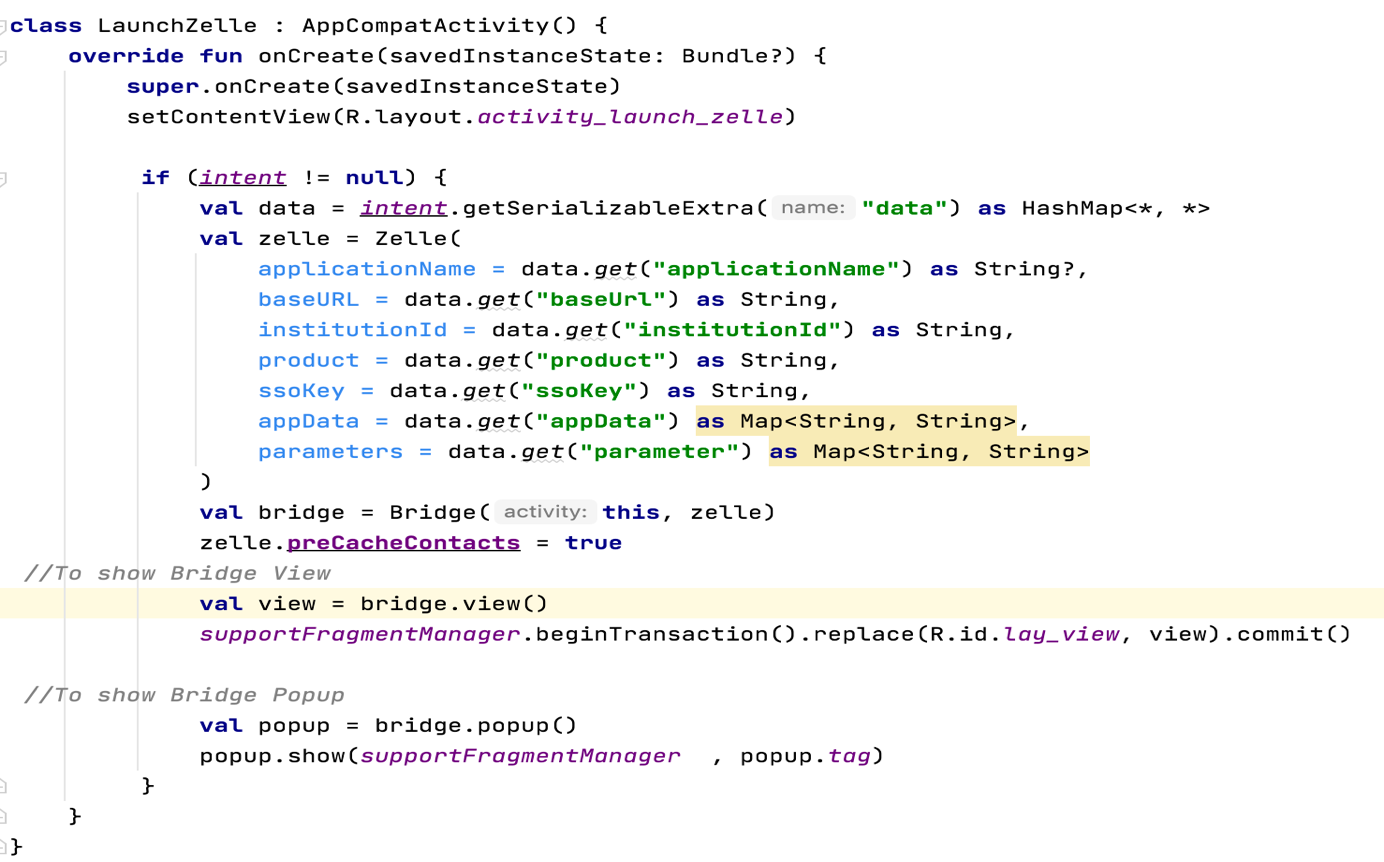
MethodChannel(flutterEngine.*dartExecutor*.*binaryMessenger*, **CHANNEL**).setMethodCallHandler **{** call, result **->  
 if** (call.**method** == **"launchZelle"**) {  
 *genericTag* = **this  
 val** hashMap = call.**arguments as** HashMap<\*,\*>  
 **val** intent = Intent(**this**, LaunchZelle::**class**.*java*)  
 intent.putExtra(**"data"**, hashMap)startActivity(intent)   
**}**



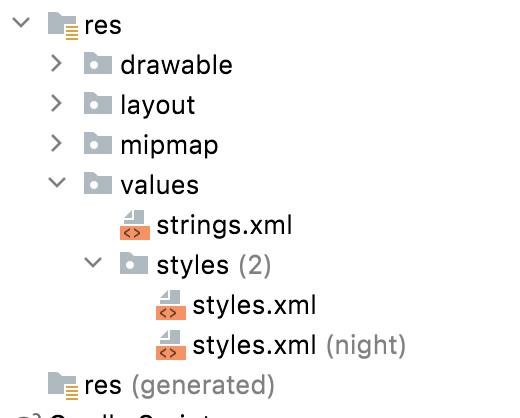
1. Open activity\_launch\_zelle xml file inside layout folder and create a framelayout to initialize zelle view.



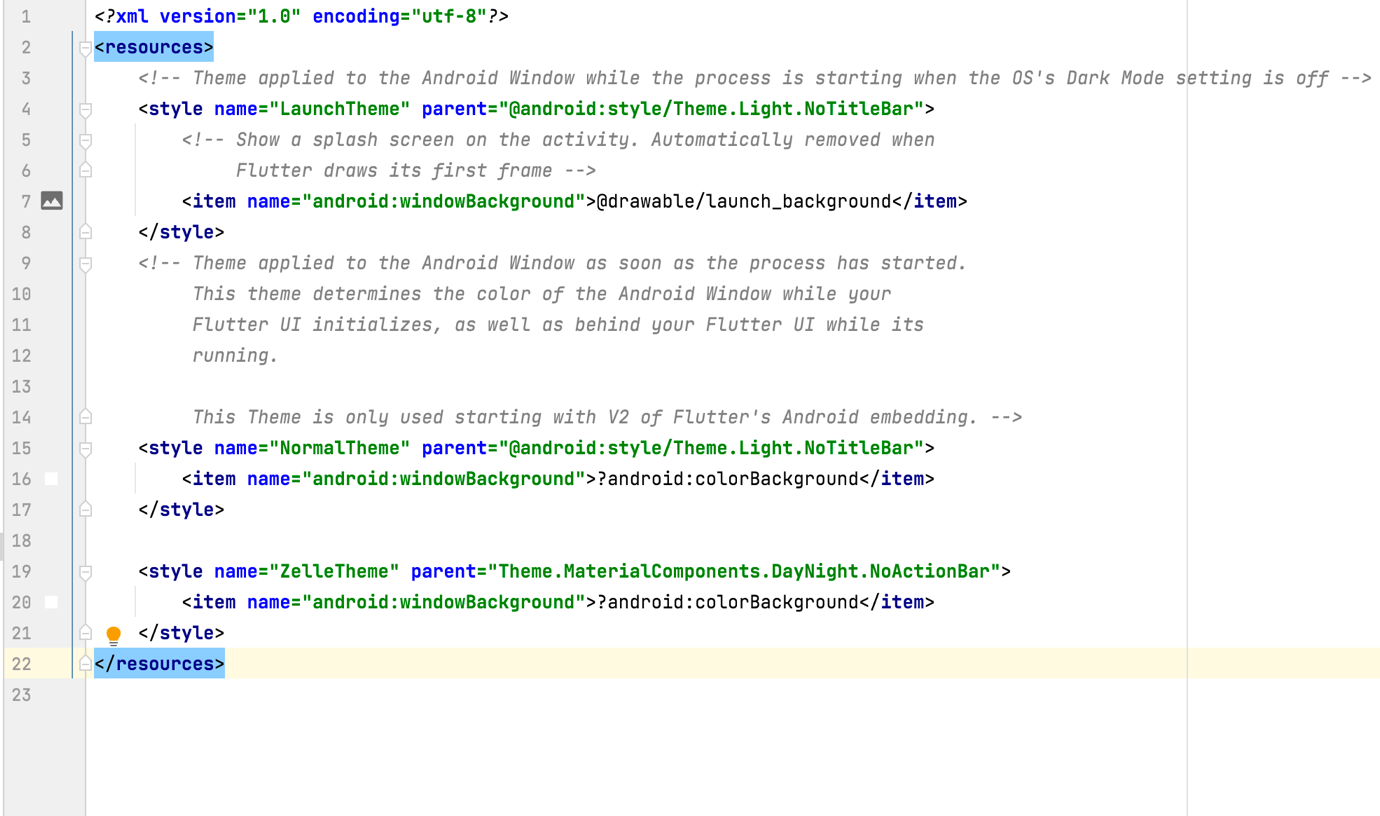
1. Open LauchZelle.kt file inside onCreateView function initialize Zelle, Bridge and BridgeView (or) BridgePopup with respective parameters.

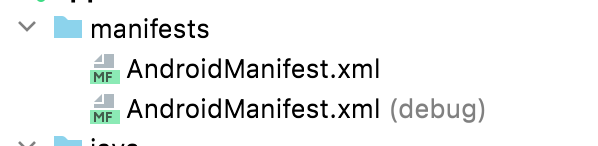


1. Create ZelleTheme inside Styles (or) Theme.xml file.



<**style name="ZelleTheme" parent="Theme.MaterialComponents.DayNight.NoActionBar"**>  
 <**item name="android:windowBackground"**>?android:colorBackground</**item**>  
</**style**>



1. Register the LaunchZelle.kt activity inside application manifest file with respective theme.



Note:

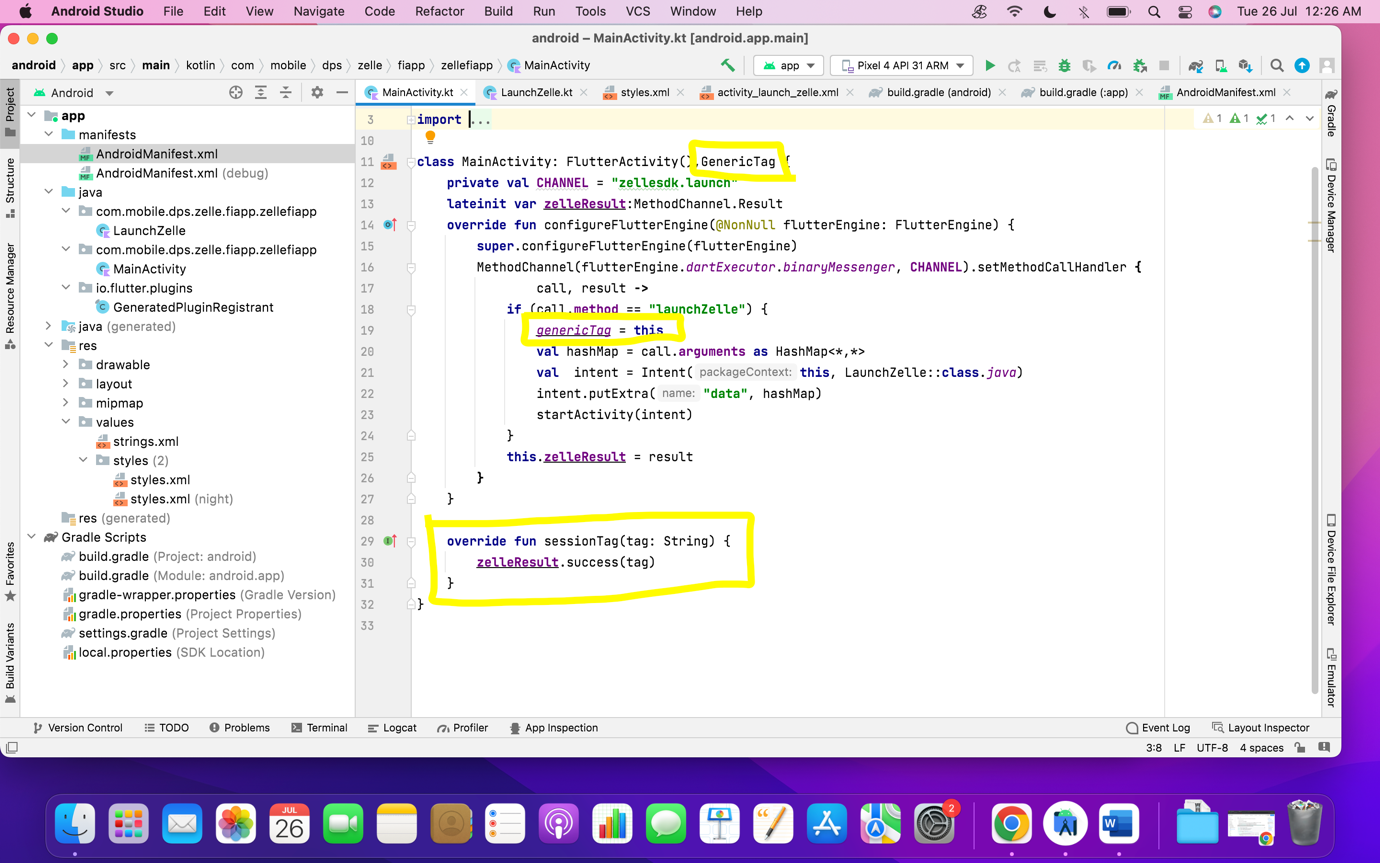
1. appData parameter passed in Zelle is an optional one. it is used to show the customized alert passed from FI (only for android not for iOS).

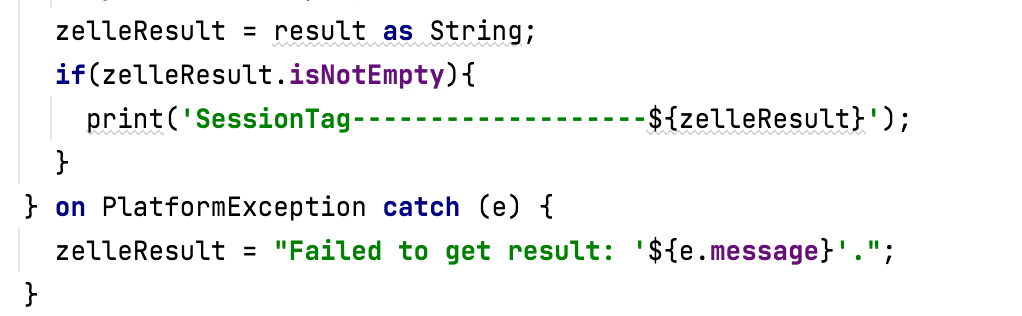
Default values:

“pd\_title” => “We would like to access your phone contacts”,

“pd\_message” => “We only sync phone numbers and email address from your contact list to help you add and pay a new ewcipient in zelle®.

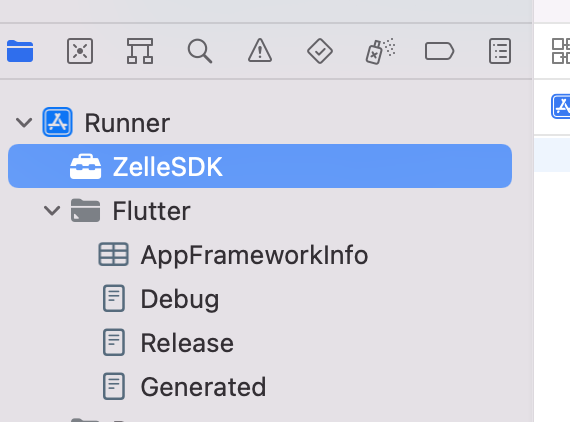
1. ZelleSDK launches from MinimumSDK version 24.
2. After zelle session is closed. To get the session tag value to client side initialize genericTag and implement GenericTag to get the tag value from ZelleSDK.



1. Inside sessionTag override method get the data from Zelle and send result back to client side.

iOS Platform:

1. Open iOS platform inside Xcode and create new ViewController to launch Zelle.
2. Place the ZelleSDK XCframework file inside the Runner.

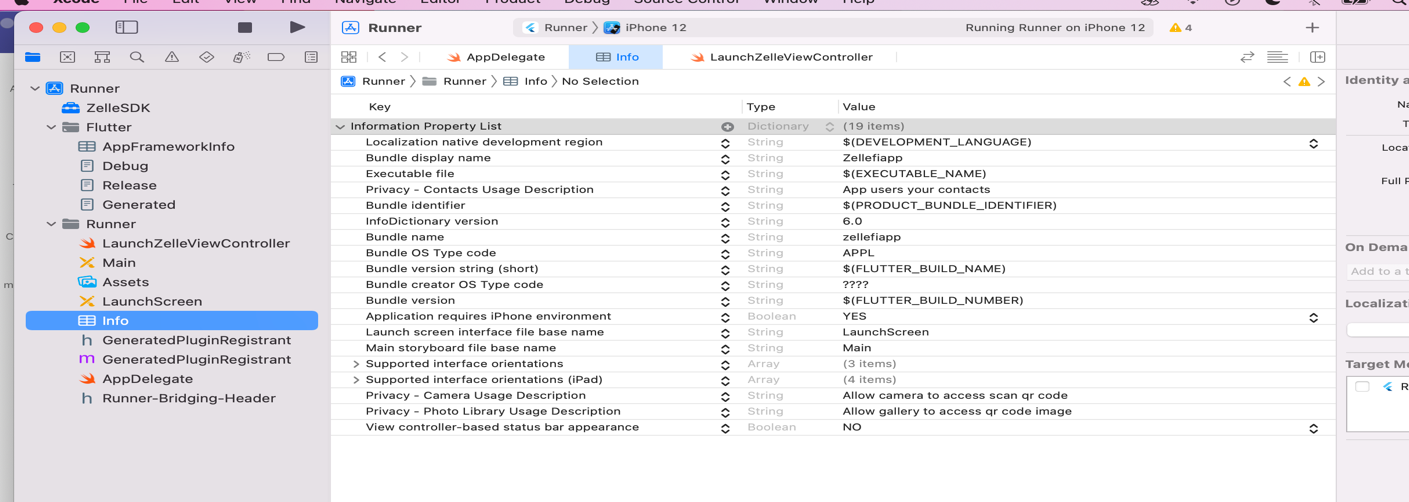


1. Add the ZelleSDK required permissions in Info.plist file

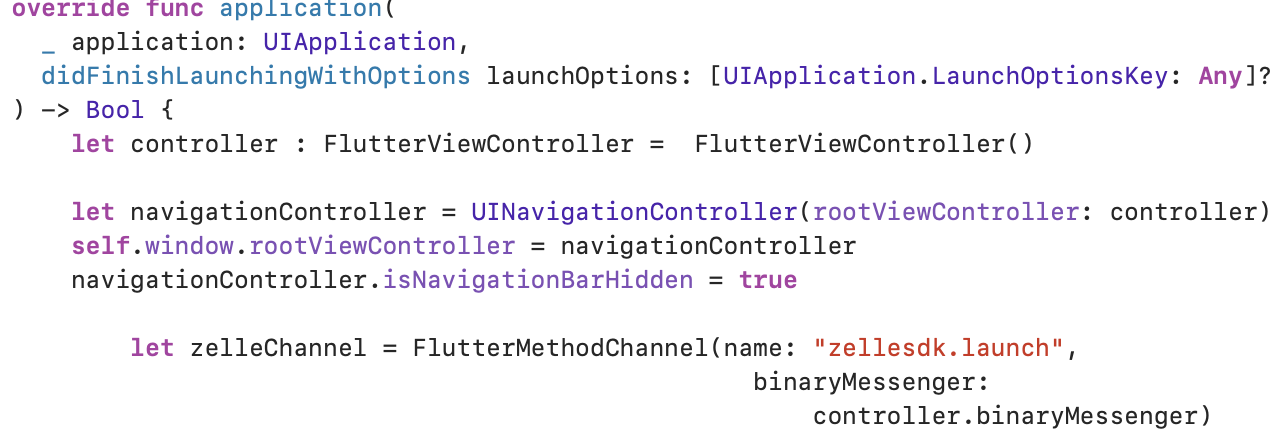
Example: 1) Contact permission

2) Camera permission

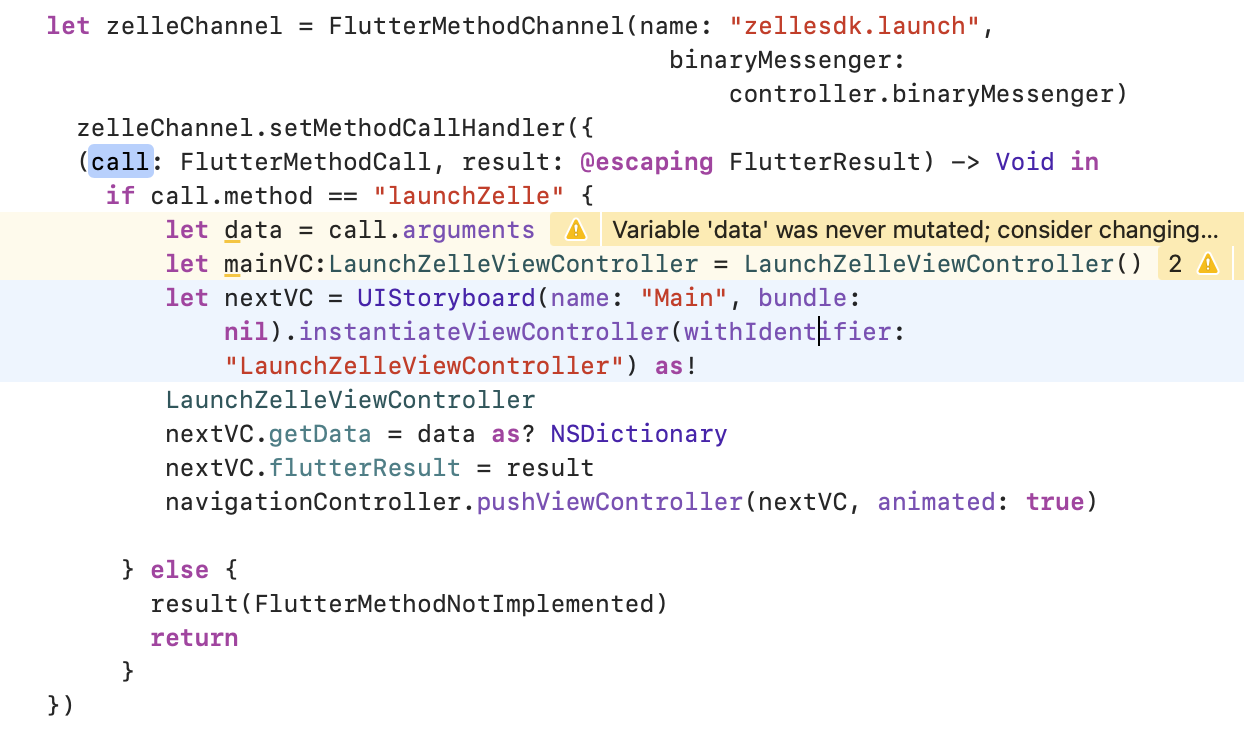
3) Gallery Permission



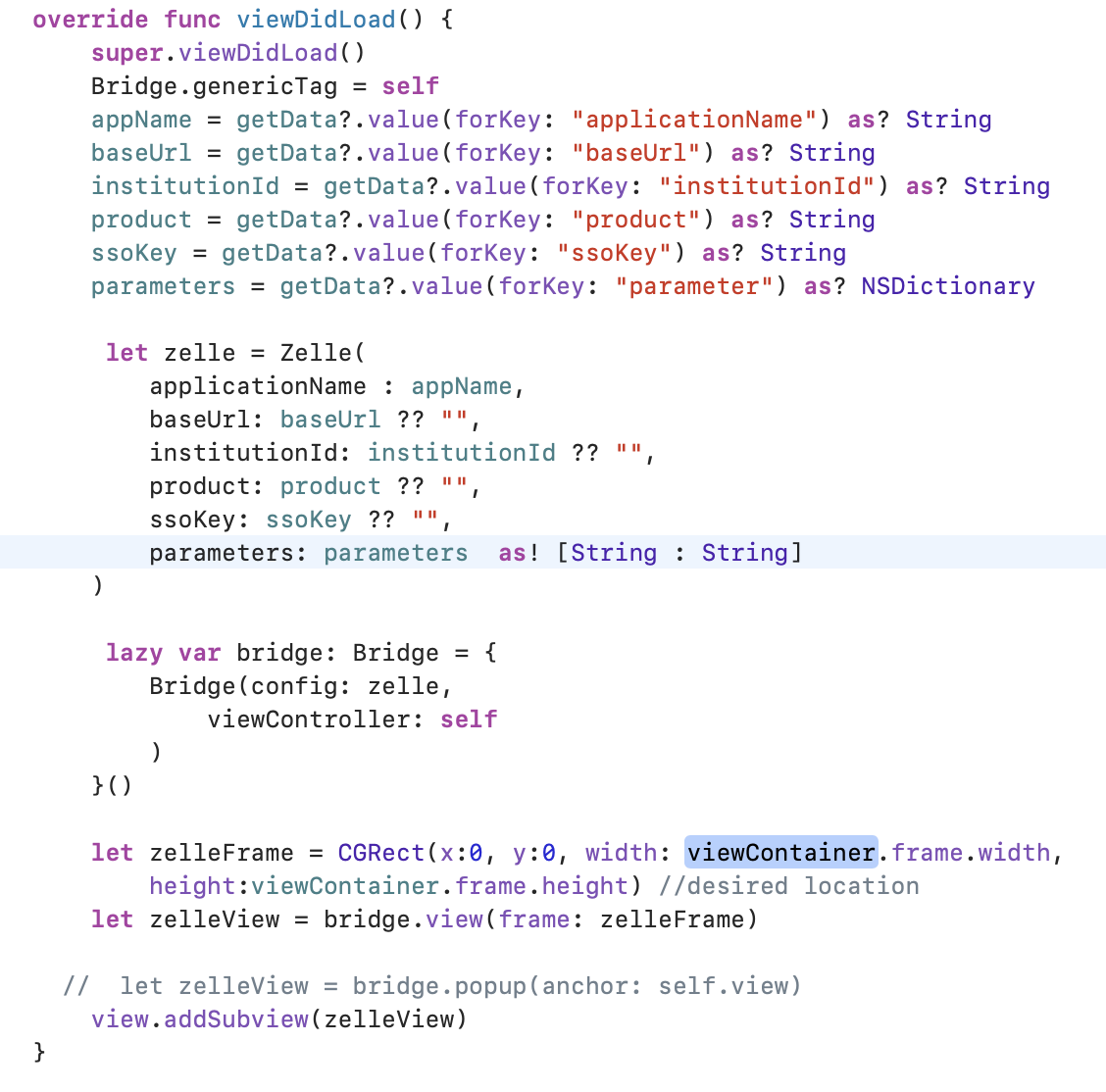
1. Open the file AppDelegate.swift located under **Runner > Runner** in the Project navigator.
2. Override the application:didFinishLaunchingWithOptions: function and create a FlutterMethodChannel tied to the channel name zellesdk.launch.



1. Get the arguments from the client side and pass that arguments to created ViewController inside FlutterMethodChannel in AppDelegate.



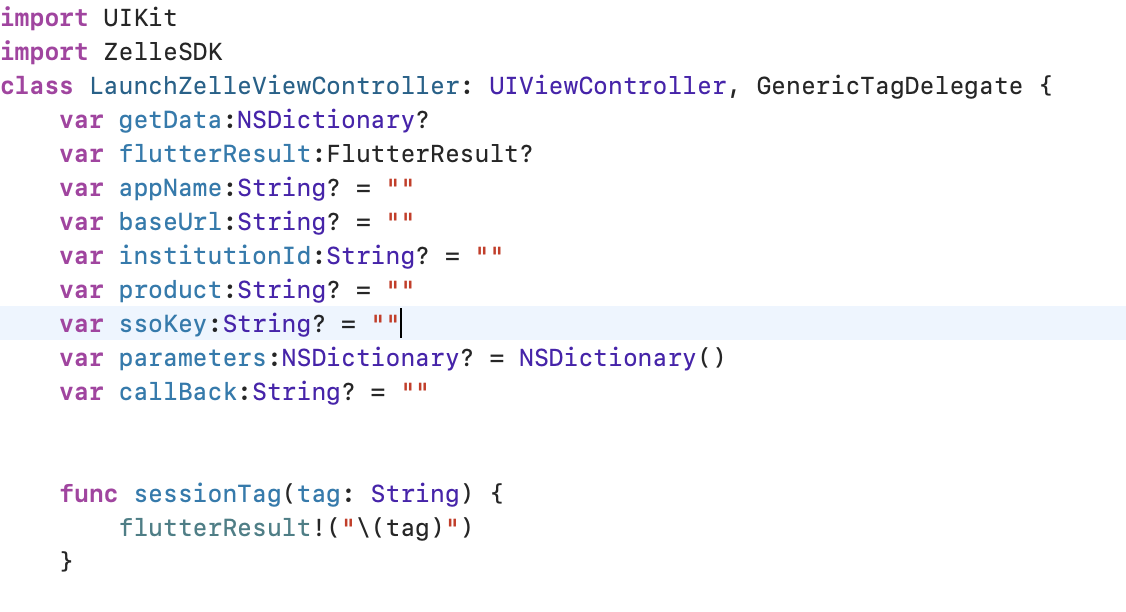
1. Now open created ViewController, Inside viewDidLoad() function initialize Zelle, Bridge, and BridgeView (or) BridgePopup to launch Zelle.



1. Pass the required variables to the Zelle Class which is passed from the client side.

Note:

1. XCframework requires minimum version iOS - 13.
2. After zelle session is closed. To get the session tag value to client side initialize genericTag and implement GenericTagDelegate to get the tag value from ZelleSDK.



1. Inside sessionTag override method get the data from Zelle and send result back to client side.
2. Before sending data back to client side initialize the FlutterResult variable created in ViewController from AppDelegate.

