Lab session 3: Multi-Screens Design using Flutter

Task-3 Adding AdMob ads to a Flutter app (Part II)

In this second part, we learn how to implement AdMob inline banner and native inline ads in a Flutter app using the Google Mobile Ads plugin for Flutter.

This documentation is designed to provide a clear and structured approach to the implementation. Each step is outlined in detail, ensuring that we can easily follow along and understand the process.

Set up your Flutter development environment

As seen in the Part I, to set up the Flutter development environment for this codelab, we will need to download two essential components: the Flutter SDK and an editor.

Ensure to have one of the following devices available for running the codelab:

- A physical Android or iOS device connected to your computer and set to Developer mode.
- The iOS simulator (requires installing Xcode tools).
- The Android Emulator (requires setup in Android Studio).
- A browser (Chrome is required for debugging).

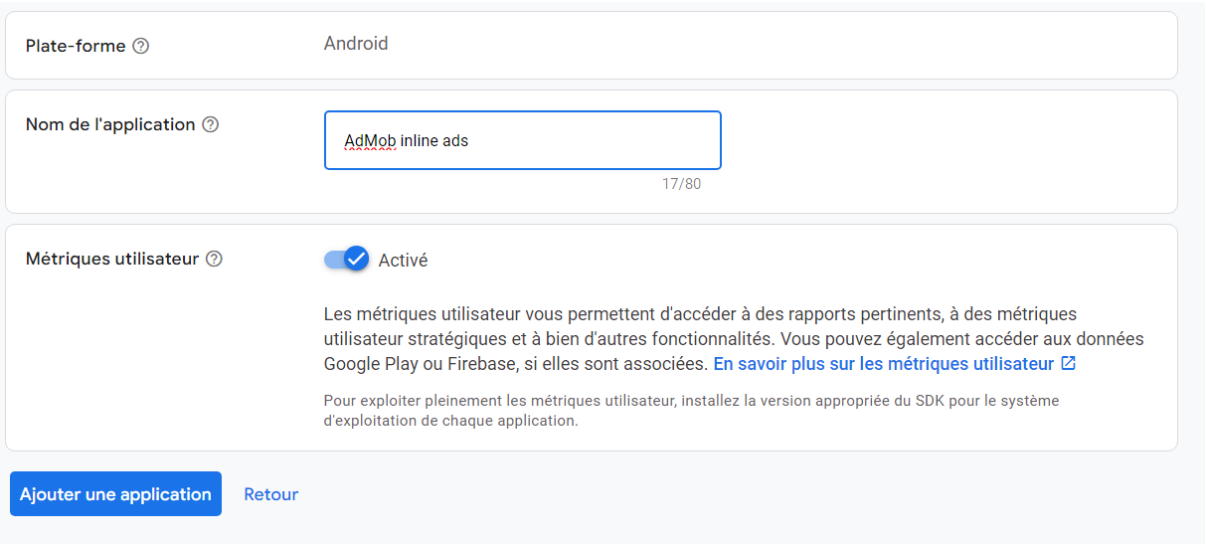
Once we have the necessary device or platform set up, we can proceed to download the code for the codelab.

Set up AdMob app and ad units

Set up for Android

Add an Android app

1. In the AdMob console, click ADD APP from the Apps menu.
2. When you're asked Have you published your app on Google Play or the App Store?, click NO.
3. Enter AdMob inline ads in the app name field, and select Android as the platform.



Enabling user metrics is not necessary to complete this codelab. However, we recommend that you do because it allows you to understand user behavior in more detail.

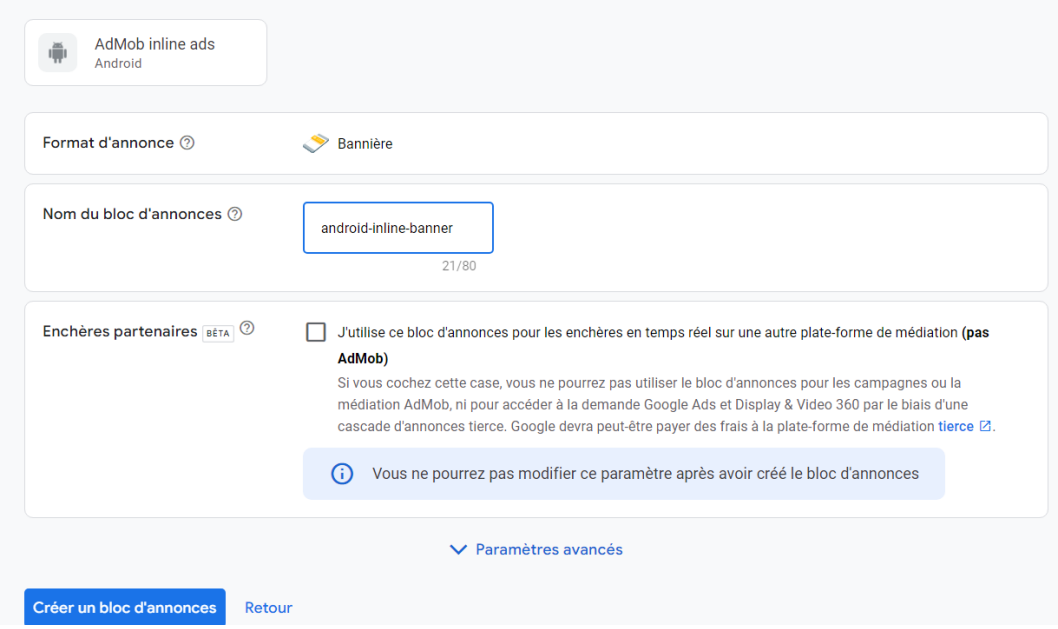
4. Click ADD to complete the process.

Create ad units

1. Select AdMob inline ads app from Apps menu in the AdMob console.
2. Click the Ad units menu.

Banner

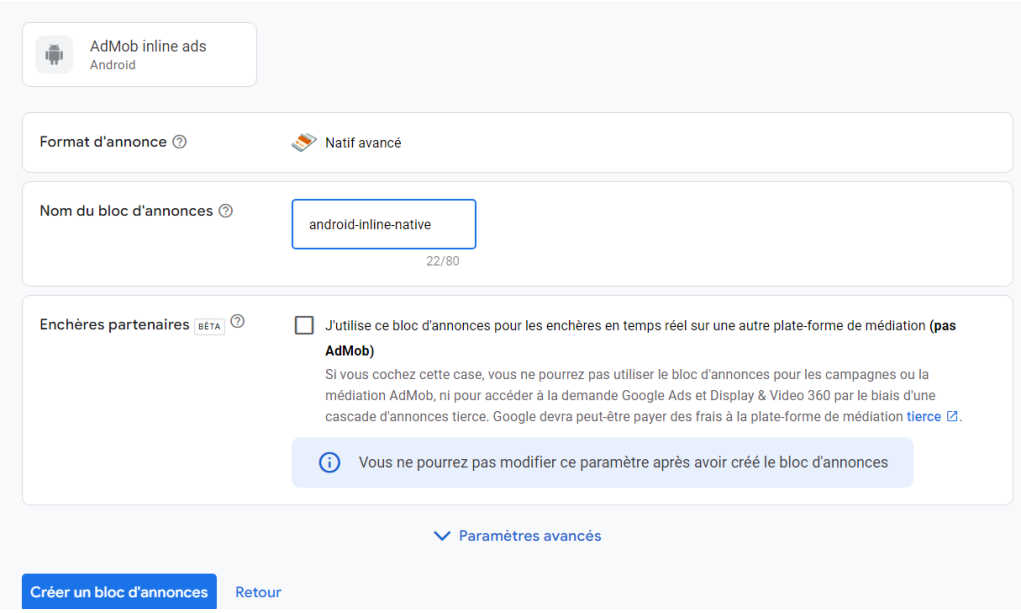
1. Click ADD AD UNIT.
2. Select Banner as the format.
3. Enter android-inline-banner in the Ad unit name field.



4. Click CREATE AD UNIT to complete the process.

Native

1. Click ADD AD UNIT.
2. Select Native advanced as the format.
3. Enter android-inline-native in the Ad unit name field.



4. Click CREATE AD UNIT to complete the process.

Set up for iOS

To set up for iOS, we need to add an iOS app and create ad units. To do so, we just follow the same process as for android but use ios-inline-… for ad units name.

Add the Google Mobile Ads Flutter plugin

Add the Google Mobile Ads plugin as a dependency

1. To access the AdMob APIs from the AdMob inline ads project, add google_mobile_ads as a dependency to the pubspec.yaml file located at the root of the project.

```
! pubspec.yaml
24  flutter:
25    sdk: flutter
26  # TODO: Add google_mobile_ads as a dependency
27  google_mobile_ads: ^1.2.0
```

2. Click Pub get to install the plugin in the AdMob inline ads project.

Update AndroidManifest.xml (Android)

1. Open the android/app/src/main/AndroidManifest.xml file in Android Studio.
2. Add the AdMob app ID by adding a <meta-data> tag with the name com.google.android.gms.ads.APPLICATION_ID. For example, if the AdMob app ID is ca-app-pub-3940256099942544~3347511713, then you need to add the following lines to the AndroidManifest.xml file.

```
android > app > src > main > AndroidManifest.xml
38
39      <!-- TODO: Add AdMob app ID -->
40      <meta-data
41          android:name="com.google.android.gms.ads.APPLICATION_ID"
42          android:value="ca-app-pub-3940256099942544~3347511713"/>
```

Update Info.plist (iOS)

1. Open the ios/Runner/Info.plist file in Android Studio.
2. Add a GADApplicationIdentifier key with the string value of your AdMob app ID. For example, if your AdMob app ID is ca-app-pub-3940256099942544~1458002511, then you need to add the following lines to the Info.plist file.

```
ios > Runner > Info.plist
31  <key>GADApplicationIdentifier</key>
32  <string>ca-app-pub-3940256099942544~1458002511</string>
```

Add a helper class for ads

1. Create a new file named ad_helper.dart under the lib directory.
2. Implement the AdHelper class, which provides an AdMob app ID and ad unit IDs for Android and iOS.

```
lib > ad_helper.dart > AdHelper
1   import 'dart:io';
2
3   class AdHelper {
4     static String get bannerAdUnitId {
5       if (Platform.isAndroid) {
6         return 'ca-app-pub-3940256099942544/6300978111';
7       } else if (Platform.isIOS) {
8         return 'ca-app-pub-3940256099942544/2934735716';
9       }
10      throw UnsupportedError("Unsupported platform");
11    }
12
13    static String get nativeAdUnitId {
14      if (Platform.isAndroid) {
15        return 'ca-app-pub-3940256099942544/2247696110';
16      } else if (Platform.isIOS) {
17        return 'ca-app-pub-3940256099942544/3986624511';
18      }
19      throw UnsupportedError("Unsupported platform");
20    }
21  }
```

Make sure that you replace the AdMob app ID (ca-app-pub-xxxxxx~yyyyy) and the ad unit ID (ca-app-pub-xxxxxxx/yyyyyyyy) with the IDs created in the previous step.

Initialize the Google Mobile Ads SDK

Before loading ads, we need to initialize the Google Mobile Ads SDK.

Open the lib/home_page.dart file, and modify _initGoogleMobileAds() to initialize the SDK before the home page is loaded.

```
home_page.dart > HomePage

// TODO: Import google_mobile_ads.dart
import 'package:google_mobile_ads/google_mobile_ads.dart';
```

```
home_page.dart > HomePage
    ); // Scaffold
  }

  Future<InitializationStatus> _initGoogleMobileAds() {
    // TODO: Initialize Google Mobile Ads SDK
    return MobileAds.instance.initialize();
  }
```

Note that you need to change the return type of the _initGoogleMobileAds() method from Future<dynamic> to Future<InitializationStatus> to get the SDK initialization result after it completes.

Add a banner ad

1. Open the lib/banner_inline_page.dart file.
2. Import ad_helper.dart and google_mobile_ads.dart by adding the following lines:

```
lib > banner_inline_page.dart > ...
14
15  // TODO: Import ad_helper.dart
16  import 'package:admob_inline_ads_in_flutter/ad_helper.dart';
17
18  import 'package:admob_inline_ads_in_flutter/destination.dart';
19  // TODO: Import google_mobile_ads.dart
20  import 'package:google_mobile_ads/google_mobile_ads.dart';
```

3. In the _BannerInlinePageState class, add the following members and methods for a banner ad.

```
lib > banner_inline_page.dart > _BannerInlinePageState > _getDestinationItemIndex
34  }
35
36  class _BannerInlinePageState extends State<BannerInlinePage> {
37    // TODO: Add _kAdIndex
38    static final _kAdIndex = 4;
39
40    // TODO: Add a banner ad instance
41    BannerAd? _ad;
```

```
lib > banner_inline_page.dart > _BannerInlinePageState > _getDestinationItemIndex
91  // TODO: Add _getDestinationItemIndex()
92  int _getDestinationItemIndex(int rawIndex) {
93      if (rawIndex >= _kAdIndex && _ad != null) {
94        return rawIndex - 1;
95      }
96      return rawIndex;
97    }
98
```

Note that _kAdIndex indicates the index where a banner ad will be displayed, and it's used to calculate the item index from the _getDestinationItemIndex() method.

4. In the initState() method, create and load a BannerAd for the 320x50 banner (AdSize.banner).

```
lib > banner_inline_page.dart > _BannerInlinePageState > initState
    super.initState();

    // TODO: Load a banner ad
    BannerAd(
    adUnitId: AdHelper.bannerAdUnitId,
    size: AdSize.banner,
    request: AdRequest(),
    listener: BannerAdListener(
      onAdLoaded: (ad) {
        setState(() {
          _ad = ad as BannerAd;
        });
      },
      onAdFailedToLoad: (ad, error) {
        // Releases an ad resource when it fails to load
        ad.dispose();
        print('Ad load failed (code=${error.code} message=${error.message})');
      },
    ), // BannerAdListener
  ).load(); // BannerAd
```

Note that an ad event listener is configured to update the UI (setState()) when an ad is loaded.

5. Modify the build() method to display a banner ad when available.

```
lib > banner_inline_page.dart > _BannerInlinePageState > build
          // TODO: Render a banner ad
          if (_ad != null && index == _kAdIndex) {
            return Container(
              width: _ad!.size.width.toDouble(),
              height: 72.0,
              alignment: Alignment.center,
              child: AdWidget(ad: _ad!),
            ); // Container
          } else {
```

6. Update itemCount, to count a banner ad entry, and update itemBuilder, to render a banner ad at the ad index (_kAdIndex) when the ad is loaded.

```
lib > banner_inline_page.dart > _BannerInlinePageState > build
        body: ListView.builder(
          // TODO: Adjust itemCount based on the ad load state
          itemCount: widget.entries.length + (_ad != null ? 1 : 0),
          itemBuilder: (context, index) {
```

7. Update the code to use the _getDestinationItemIndex() method to retrieve an index for the content item.

```
lib > banner_inline_page.dart > _BannerInlinePageState > build
            child: AdWidget(ad: _ad!),
          ); // Container
          } else {
            // TODO: Get adjusted item index from _getDestinationItemIndex()
            final item = widget.entries[_getDestinationItemIndex(index)];
```

8. Release the resource associated with the BannerAd object by calling the BannerAd.dispose() method in the dispose() callback method.

```
lib > banner_inline_page.dart > _BannerInlinePageState > dispose
110   void dispose() {
111     // TODO: Dispose a BannerAd object
112     _ad?.dispose();
```

9. Run the project, and click the Banner inline ad button from the home page. After an ad is loaded, you'll see a banner ad in the middle of the list.

Add a native ad

Native ads are presented to users using UI components that are native to the platform (for example, View on Android or UIView on iOS).

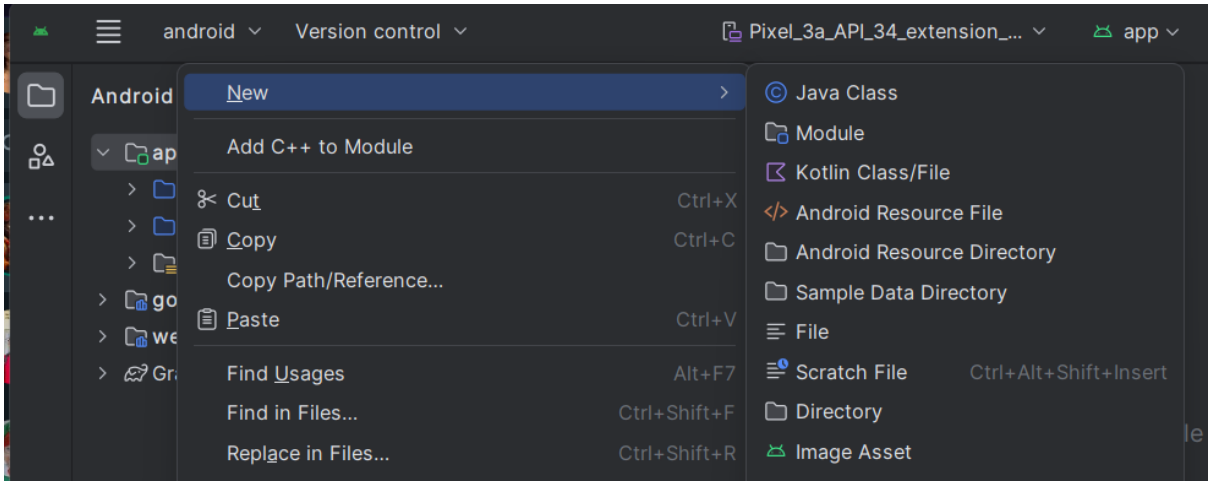
However, it isn't possible to create native UI components directly by using Flutter widgets. So, we have to implement a NativeAdFactory for each platform, which is used to build a platform-specific native ad view (NativeAdView on Android and GADNativeAdView on iOS) from a native ad object (NativeAd on Android and GADNativeAd on iOS).

Implement NativeAdFactory for Android (Java)

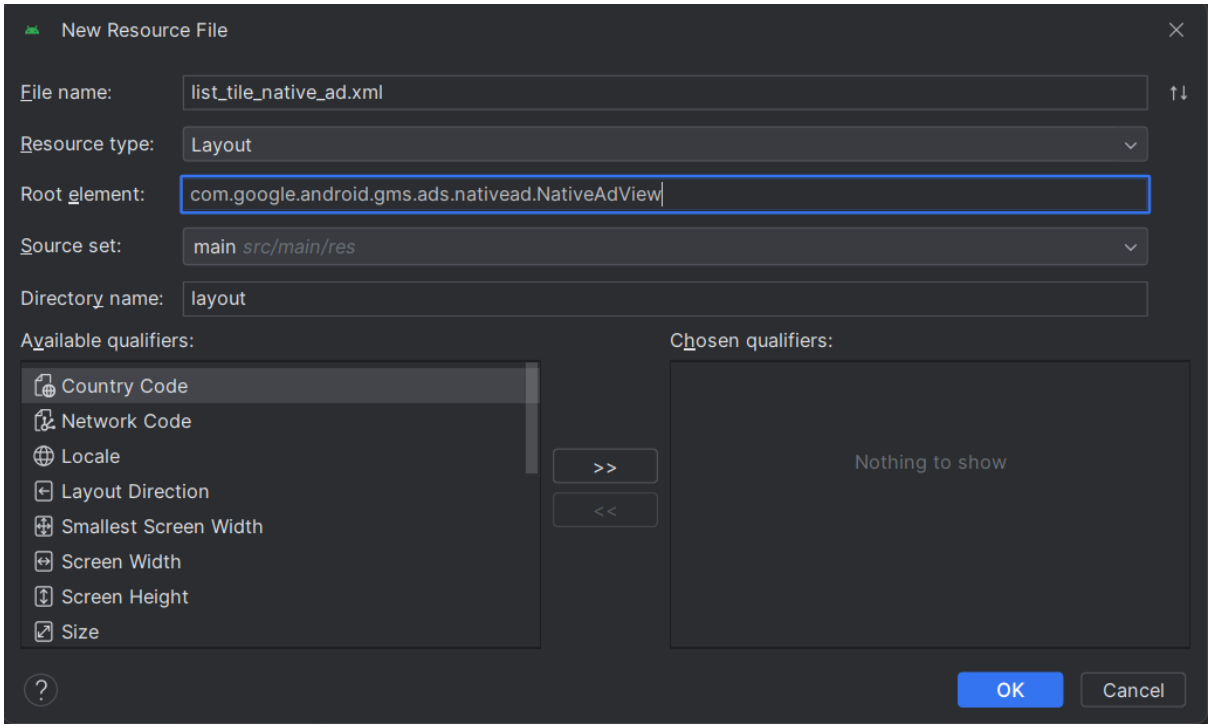
1. Open the android/build.gradle file (or any file under the android folder), and click Open for Editing in Android Studio to open an Android project.
   If you're asked to select a window to open a new project, click New Window to make the Flutter project remain open while you're working on the Android project.

Create a native ad layout

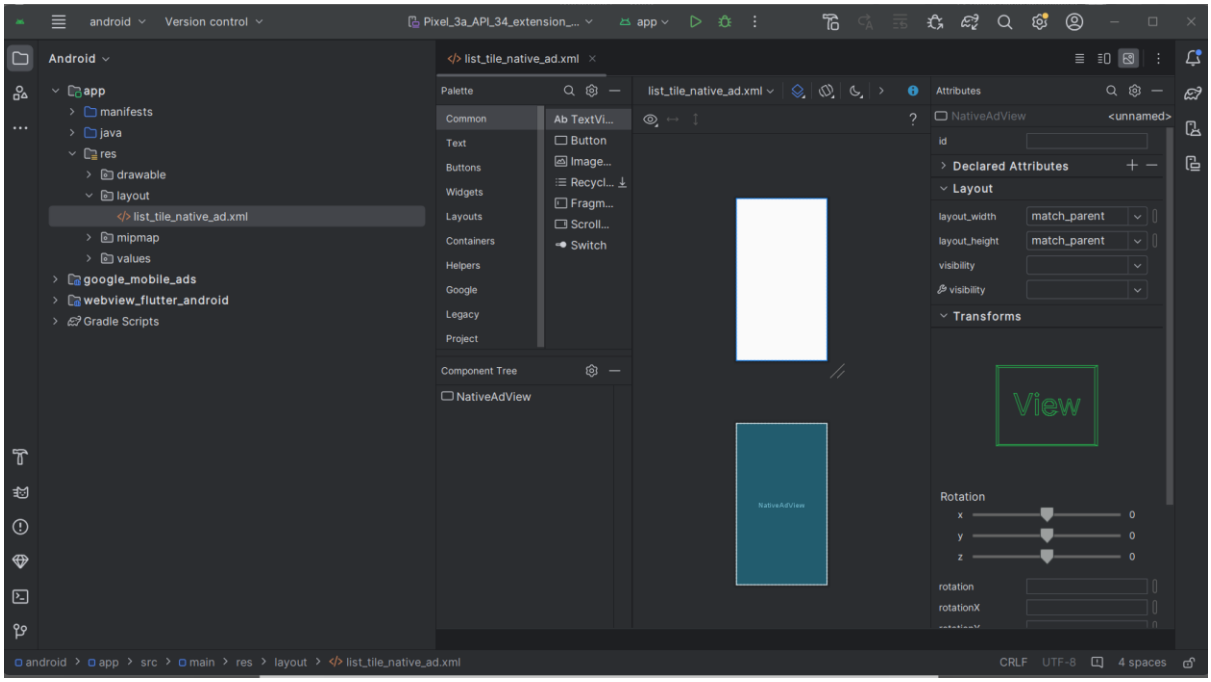
1. With the Android project opened, right-click app from the project pane in Android Studio, and select New > Android Resource File from the context menu.



2. In the New Resource File dialog, enter list_tile_native_ad.xml as the file name.
3. Select Layout as the resource type, and enter com.google.android.gms.ads.nativead.NativeAdView as a root element.



4. Click OK to create a new layout file.



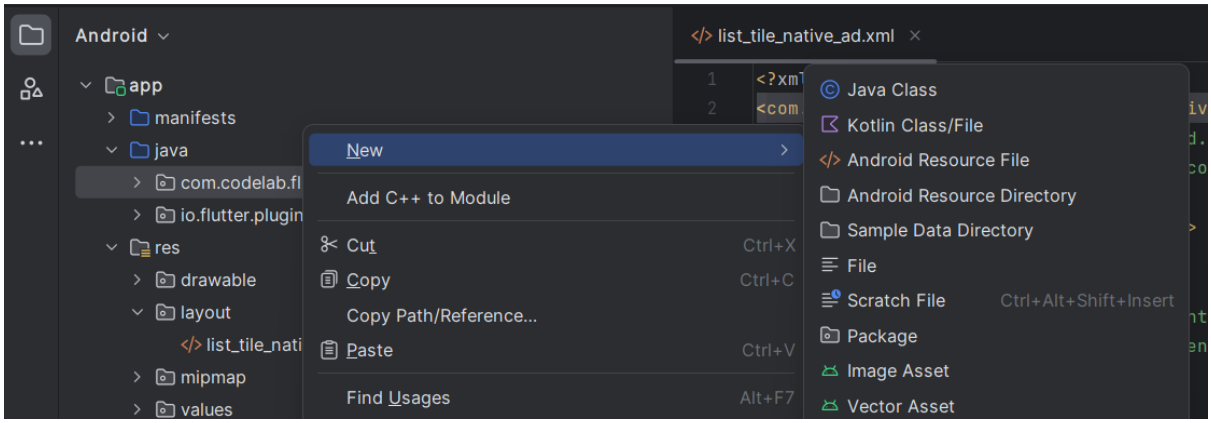
5. Implement the ad layout as follows.

```xml
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.gms.ads.nativead.NativeAdView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/tv_list_tile_native_ad_attribution_small"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#F19938"
            android:text="Ad"
            android:textColor="#FFFFFF"
            android:textSize="12sp" />

        <ImageView
            android:id="@+id/iv_list_tile_native_ad_icon"
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:layout_gravity="center_vertical"
            android:layout_marginStart="16dp"
            android:layout_marginLeft="16dp"
            android:scaleType="fitXY"
```

Note that the layout should match the visual design of the user experience for the platform it's intended for.

Create the ListTileNativeAdFactory class

1. In the Project pane, right-click the com.codelab.flutter.admobinlineads package, and select New > Java Class.



2. Enter ListTileNativeAdFactory as the name, and select Class from the list.

3. After the New Class dialog appears, leave everything empty, and click OK.
4. Implement the ListTileNativeAdFactory class as follows.

```
list_tile_native_ad.xml    ListTileNativeAdFactory.java
1   package com.codelab.flutter.admobinlineads;
2
3   import com.google.android.gms.ads.nativead.NativeAd;
4   import com.google.android.gms.ads.nativead.NativeAdView;
5
6   import android.content.Context;
7   import android.view.LayoutInflater;
8   import android.view.View;
9   import android.widget.ImageView;
10  import android.widget.TextView;
11
12  import java.util.Map;
13
14  import io.flutter.plugins.googlemobileads.GoogleMobileAdsPlugin;
15
    no usages
16  class ListTileNativeAdFactory implements GoogleMobileAdsPlugin.NativeAdFactory {
17
```

Note that the class implements the createNativeAd() method in the GoogleMobileAdsPlugin.NativeAdFactory interface.

The factory class is responsible for creating a view object for rendering a native ad. As you can see from the code, the factory class creates a UnifiedNativeAdView and populates it with a NativeAd object.

Register the ListTileNativeAdFactory class

An instance of a NativeAdFactory should be registered to the GoogleMobileAdsPlugin before it can be used from the Flutter side.

1. Open the MainActivity.java file, and override the configureFlutterEngine() method and the cleanUpFlutterEngine() method.
2. Register the ListTileNativeAdFactory class with a unique string ID (listTile) in the configureFlutterEngine() method.

```
</> list_tile_native_ad.xml    ⓒ ListTileNativeAdFactory.java    ⓒ MainActivity.java ×

27
28
29 public class MainActivity extends FlutterActivity {
       9 usages
30     @Override
31     public void configureFlutterEngine(@NonNull FlutterEngine flutterEngine) {
32         super.configureFlutterEngine(flutterEngine);
33
34     // TODO: Register the ListTileNativeAdFactory
35         GoogleMobileAdsPlugin.registerNativeAdFactory(flutterEngine, factoryId: "listTile",
36                 new ListTileNativeAdFactory(getContext()));
```

3. Every NativeAdFactory instance should be unregistered during the cleanup process. Unregister the ListTileNativeAdFactory class in the cleanUpFlutterEngine() method.

```
</> list_tile_native_ad.xml    ⓒ ListTileNativeAdFactory.java    ⓒ MainActivity.java ×

36                 new ListTileNativeAdFactory(getContext()));
37     }
38
       3 usages
39     @Override
40     public void cleanUpFlutterEngine(@NonNull FlutterEngine flutterEngine) {
41         super.cleanUpFlutterEngine(flutterEngine);
42
43     // TODO: Unregister the ListTileNativeAdFactory
44         GoogleMobileAdsPlugin.unregisterNativeAdFactory(flutterEngine, factoryId: "listTile");
45     }
46 }
47
```

Now we are ready to use the ListTileNativeAdFactory class to render native ads on Android.

Integrate the native ad with Flutter widgets

1. Open lib/native_inline_page.dart file. Then, import ad_helper.dart and google_mobile_ads.dart by adding the following lines:

```
lib > native_inline_page.dart > ...
14
15  // TODO: Import ad_helper.dart
16  import 'package:admob_inline_ads_in_flutter/ad_helper.dart';
17
18  import 'package:admob_inline_ads_in_flutter/destination.dart';
19  // TODO: Import google_mobile_ads.dart
20  import 'package:google_mobile_ads/google_mobile_ads.dart';
21
```

2. In _NativeInlinePageState class, add the following members and methods for a native ad.

```dart
lib > native_inline_page.dart > _NativeInlinePageState
}

class _NativeInlinePageState extends State<NativeInlinePage> {
  // TODO: Add _kAdIndex
  static final _kAdIndex = 4;

  // TODO: Add a native ad instance
  NativeAd? _ad;
```

```dart
lib > native_inline_page.dart > _NativeInlinePageState > _getDestinationItemIndex
// TODO: Add _getDestinationItemIndex()
int _getDestinationItemIndex(int rawIndex) {
    if (rawIndex >= _kAdIndex && _ad != null) {
      return rawIndex - 1;
    }
    return rawIndex;
  }
```

Note that _kAdIndex indicates the index where a banner ad will be displayed, and it's used to calculate the item index from the _getDestinationItemIndex() method.

3. In the initState() method, create and load a NativeAd that uses ListTileNativeAdFactory to generate a native ad view.

```dart
lib > native_inline_page.dart > _NativeInlinePageState > initState

  @override
  void initState() {
    super.initState();

    // TODO: Load a native ad
    _ad = NativeAd(
      adUnitId: AdHelper.nativeAdUnitId,
      factoryId: 'listTile',
      request: AdRequest(),
      listener: NativeAdListener(
        onAdLoaded: (ad) {
          setState(() {
            _ad = ad as NativeAd;
          });
        },
        onAdFailedToLoad: (ad, error) {
          // Releases an ad resource when it fails to load
          ad.dispose();
          print('Ad load failed (code=${error.code} message=${error.message})');
          },
      ), // NativeAdListener
    ); // NativeAd

  _ad.load();
```

Note that the same factory ID (listTile) used to register the factory to the plugin is used.

4. Modify the build() method to display a banner ad when available.

```
lib > native_inline_page.dart > _NativeInlinePageState > build
77          itemCount: widget.entries.length,
78          itemBuilder: (context, index) {
79            // TODO: Render a native ad
80            if (_ad != null && index == _kAdIndex) {
81            return Container(
82              height: 72.0,
83              alignment: Alignment.center,
84              child: AdWidget(ad: _ad!),
85            ); // Container
86          } else {
```

5. Update itemCount, to count a banner ad entry, and update the itemBuilder, to render a banner ad at the ad index (_kAdIndex) when the ad is loaded.

```
lib > native_inline_page.dart > _NativeInlinePageState > build
73          title: const Text('AdMob Native Inline Ad'),
74        ), // AppBar
75        body: ListView.builder(
76          // TODO: Adjust itemCount based on the ad load state
77          itemCount: widget.entries.length + (_ad != null ? 1 : 0),
78          itemBuilder: (context, index) {
```

6. Update the code to use the _getDestinationItemIndex() method to retrieve an index for the content item.

```
lib > native_inline_page.dart > _NativeInlinePageState > build
86          } else {
87            // TODO: Get adjusted item index from _getDestinationItemIndex()
88            final item = widget.entries[_getDestinationItemIndex(index)];
```

7. Release the resource associated with the NativeAd object by calling NativeAd.dispose() method in the dispose() callback method.

```
lib > native_inline_page.dart > _NativeInlinePageState > dispose
109
110     @override
111     void dispose() {
112       // TODO: Dispose a NativeAd object
113       _ad?.dispose();
114
115       super.dispose();
116     }
```

8. Run the project, and click the Native inline ad button from the home page. After an ad is loaded, you'll see a native ad in the middle of the list.

Testing Ads

During our testing process, we encountered challenges due to version incompatibility between Gradle and the Android Gradle Plugin. Unfortunately, our attempts to resolve this issue resulted in the emergence of further errors leading to us give up the testings.

Conclusion

In this tutorial, we continued our exploration of implementing ads in our mobile app, following Part I of the tutorial. We focused on two new types of ad units that require more extensive code modifications, particularly the native advanced ad unit. Although we encountered difficulties in debugging the errors this time, it is worth noting that version incompatibility is often a common cause of issues. The tutorial was published in 2022, and since then, updates have introduced significant changes to functionality.

Nonetheless, throughout the tutorial, we gained a solid understanding of the basic process of adding inline ads. Armed with this knowledge, we can confidently implement ads in our personal projects.

References

https://codelabs.developers.google.com/codelabs/admob-inline-ads-in-flutter#8

https://apps.admob.com/v2/apps/5884492276/adunits/list