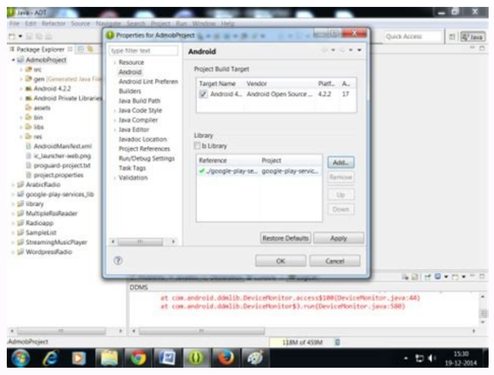
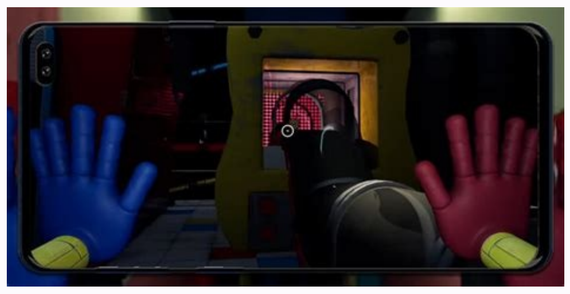
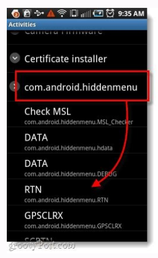
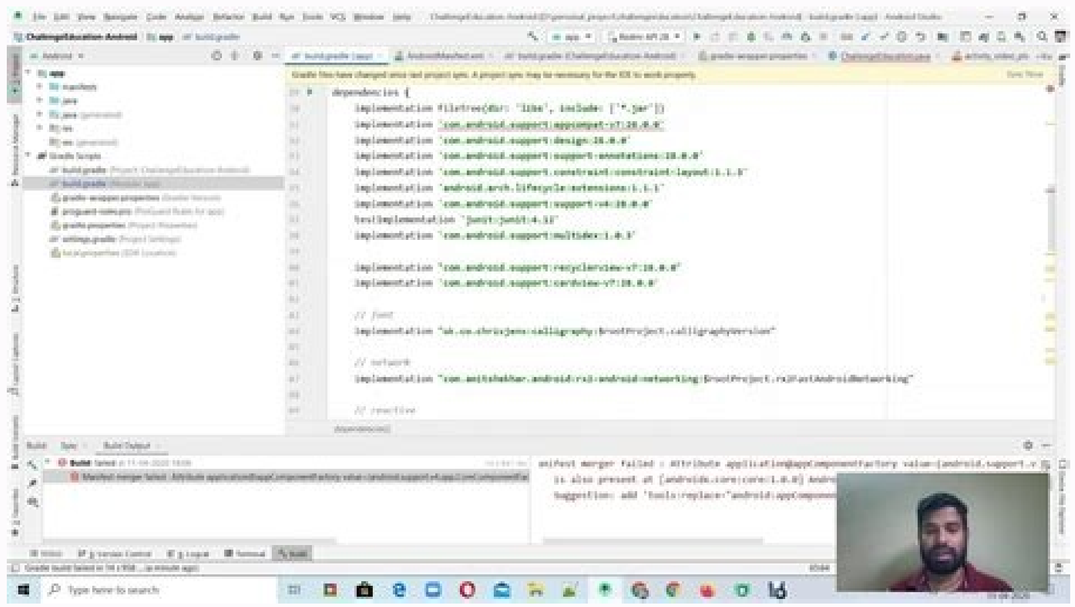
Adding admob to android app

Continue









How to install admob on android app. How to add admob ads to android app. How to add admob to my app.

In this code lab, you'll implement an AdMob banner and built-in AdMob native ads in Flutter. What You Will Create This code lab will help you implement AdMob's built-in banner and native AdMob's built-in ads in Flutter using the Google Mobile Ads Plugin for Flutter. If you encounter any issues while working with this code lab (code errors, grammatical errors, unclear language, etc.), please report a problem using the Report a bug link in the lower left corner of the code lab. What you'll learn How to set up the Google Mobile Ads Flutter plugin How to implement built-in banner and rewarded ads in Flutter What you'll need Android Studio 4.1 or later Xcode 12 or later (for iOS development) Beginner Intermediate Experienced Beginner Intermediate Stated that Flutter is a cross-platform package SDK, you need to add apps and ad units to AdMob for both Android and iOS. Note. If you'd like to follow the lab's code instead of creating a new app and ad units yourself, go to Using a test AdMob app and ad units. Android Setup To set up Android, you'll need to add an Android app and create ad units. Adding an Android app. In the AdMob console, under the Apps menu, click ADD APPLICATION. When asked if you published your app on Google Play or the App Store?, click NO. In the app name field, enter AdMob embedded ads and select Android as your platform. Enabling custom metrics is not necessary to complete this experimental code work. However, we recommend that you do so as it allows you to understand user behavior in more detail. Click ADD to complete the process. Creating ad units To add ad units, in the AdMob console, in the Apps section, select Embedded AdMob ads. Click the Ad Blocks menu. Banner Click ADD ADVERTISER. Choose Banner as the format. Enter android-inline-banner in the Ad unit name field. Click CREATE ADVERTISER to complete the process. ADD ADVERTISER custom click. Choose Native Advanced as the format. Enter android-inline-native in the Ad unit name field. Click CREATE AD UNIT.terminate the process. Important! When building and testing apps, remember to use test ads instead of live production ads. Failure to do so may result in suspension of your account. For more information on how Mobile Ads SDK test ads work, see Testing Ads. It usually takes a few hours for a new ad unit to start serving ads. To test your ads immediately, use the test app IDs and ad unit IDs in the Android App ID/Ad Unit ID and iOS App ID/Ad Unit ID tables. iOS Setup iOS setup requires adding an iOS app and creating ad units. Add an iOS app. From the Apps menu in the AdMob console, click ADD APP. To the question Have you published your app on Google Play or the App Store? click NO. In the App Name field, enter included AdMob ads and select iOS as the platform. You do not need to enable custom metrics to complete this code experiment. However, we recommend doing this because it allows for a more detailed understanding of user behavior. Click ADD to complete the process. Creating ad units To add ad units, select AdMob Embedded Ads from the Applications menu in the AdMob console. Click on the Ad Units menu. Banner Click on ADD ADVERTISER. Select Banner as the format. Enter ios-inline-banner in the Ad Unit Name field. Click CREATE ADVERTISER to complete the process. Custom Click on ADD ADVERTISER. Select Native Advanced as the format. Enter ios-inline-native in the Ad Unit Name field. Click CREATE ADVERTISER to complete the process. Important! When building and testing apps, remember to use test ads instead of live production ads. Failure to do so may result in suspension of your account. For more information on how Mobile Ads SDK test ads work, see Testing Ads. It usually takes a few hours for a new ad unit to start serving ads. To test your ads immediately, use the Test App ID and Ad Unit ID listed in this table. Optional: Use a test app and AdMob ad units. If you want to follow the lab codeYou can use the AdMob App ID and Ad Unit ID in the tables below to create new apps and ad units yourself. Android App ID/Ad Unit ID App ID/Ad Unit ID AdMob App ID ca-app-pub-3940256099942544~3347511713 Banner ca-app-pub-3940256099942544/6300978111 iOS Native978111 ca-app-pub-App-3949025 Native 3940256099942544/6300978111 native iOS978111 94517/925 Ad Unit ID App ID/Ad Unit ID ADMOB CA-APP-Pub-39402560999942544 1458002511 pub, see developer documentation for Android test ads and test ads. Flutter uses plugins to provide access to a wide range of platform-specific services. Plugins allow you to access services and APIs on any platform. The google_mobile_ads plugin supports loading and displaying banner, print, paid and native ads using the AdMob API. Because Flutter has a cross-platform SDK, the google_mobile_ads plugin is applicable to both iOS and Android devices. Therefore, if you add a plugin to a Flutter app, it will be used by both the Android and iOS versions of the AdMob app. Add the Google Ads Mobile plugin as a dependency. To access the AdMob API from the AdMob Inline Ads project, add google_mobile_ads as a dependency to the pubspec.yaml file at the root of the project. pubspec.yaml ... dependencies: flutter: sdk: flutter google_fonts: ^0.3.9 # TODO: add google_mobile_ads as a dependency google_mobile_ads: ^1.2.0 ... Click Pub to install the plugin to your AdMob embedded ads project. Open android/app/src/main/AndroidManifest.xml in Android Studio. Add your AdMob App ID by adding a tag named com.google.android.gms.ads.APPLICATION_ID. For example, if your AdMob App ID is ca-app-pub-3940256099942544~3347511713, you need to add the following lines to the AndroidManifest.xml file. AndroidManifest.xml ... ... Open the ios/Runner/Info.plist file in Android Studio. Add the GADApplicationIdentifier key with the string value of your AdMob application ID. For example, if your AdMob App ID is ca-app-pub-3940256099942544~1458002511, you need to add the following lines to your Info.plist file. ... GADApplicationIdentifier ca-app-pub-3940256099942544~1458002511 ... Create a new file called ad_helper.dart in the lib directory. Next, implement the AdHelper class, which provides the AdMob app ID and ad unit IDs for Android and iOS. Be sure to replace the AdMob App ID (ca-app-pub-xxxxxx~yyyyy) and Ad Unit ID (ca-app-pub-xxxxxxx/yyyyyyyy) with the IDs you created in the previous step. ad_helper.dart import 'dart:io'; class AdHelper { static string get bannerAdUnitId { if (Platform.isAndroid) { return " Create Class ListTileNativeAdFactory In the project pane, right-click on the com.codelab.flutter.admobinlineads package and select New > Java Class. Enter ListTileNativeAdFactory as the name and select Class from the list. When the New Class dialog box appears, leave everything blank and click OK. You will see that the ListTileNativeAdFactory class is created in the com.codelab.flutter.admobinlineads package. To deployclass as follows. Notice that the class implements the createNativeAd() method in the GoogleMobileAdsPlugin.NativeAdFactory interface. The factory class is responsible for creating a view object to render the native display. As you can see from the code, the factory class creates a UnifiedNativeAdView object and populates it with a NativeAd object. ListTileNativeAdFactory.java // TODO: implementation of ListTileNativeAdFactory class package com.codelab.flutter.admobinlineads; import com.google.android.gms.ads.nativead.NativeAd; import com.google.android.gms.ads.nativead.NativeAdView; import android.content.Context; import android.view.LayoutInflater; import android.view.View; import android.widget.ImageView; import android.widget.TextView; import java.util.Map; import io.flutter.plugins.googlemobileads.GoogleMobileAdsPlugin; ListTileNativeAdFactory class implements GoogleMobileAdsPlugin.NativeAdFactory { private final Context context; ListTileNativeAdFactory(Context context) { this.context = context; } @Override public NativeAdView createNativeAd(NativeAd nativeAd, Map customOptions) { NativeAdView nativeAdView = (NativeAdView) LayoutInflater.from(context) .inflate(R.layout.list_tile_native_ad, null); TextView attributionViewSmall = nativeAdView .findViewById(R.id.tv_list_tile_native_ad_attribution_small); TextView attributionViewLarge = nativeAdView .findViewById(R.id.tv_list_tile_native_ad_attribution_large); ImageView iconView = nativeAdView.findViewById(R.id.iv_list_tile_native_ad_icon); NativeAd.Image Icon = nativeAd.getIcon(); if (icon != null) {attributionViewSmall.setVisibility(View.VISIBLE); attributionViewLarge.setVisibility(View.INVISIBLE); iconView.setImageDrawable(icon.getDrawable()); } else { attributionViewSmall.setVisibility(View.INVISIBLE); attributionViewLarge.setVisibility(View.VISIBLE); } nativeAdView.setIconView(iconView); TextView headlineView = nativeAdView.findViewById(R.id.tv_list_tile_native_ad_headline);nativeAdView.setHeadlineView(headlineView); TextView bodyView = nativeAdView.findViewById(R.id.tv_list_tile_native_ad_body); bodyView.setText(nativeAd.getBody()); bodyView.setVisibility(nativeAd.getBody() != null ? View.VISIBLE : View.INVISIBLE); nativeAdView.setBodyView(bodyView); nativeAdView. setNativeAd(nativeAdView); return native ad view; } } Register Class ListTileNativeAdFactory An instance of NativeAdFactory must be registered with GoogleMobileAdsPlugin to be used by Flutter. Open the MainActivity.java file and override the configureFlutterEngine() and cleanUpFlutterEngine() methods. Register the ListTileNativeAdFactory class with a unique string identifier (listTile) in the configureFlutterEngine() method. MainActivity.java public class MainActivity extends FlutterActivity { @Override public void configureFlutterEngine(@NonNull FlutterEngine flutterEngine) { super.configureFlutterEngine(flutterEngine); // TODO: Register ListTileNativeAdFactory GoogleMobileAdsPlugin.registerNativeAdFactory(flutterEngine, "listTile", new ListTileNativeAdFactory(getContext())); } ... } Unregister all instances of NativeAdFactory during cleanup. Unregister the ListTileNativeAdFactory class using the cleanUpFlutterEngine() method. MainActivity.java public class MainActivity extends FlutterActivity { ... @Override public void cleanUpFlutterEngine(@NonNull FlutterEngine flutterEngine) { super.cleanUpFlutterEngine(flutterEngine); // TODO: Unregister ListTileNativeAdFactory GoogleMobileAdsPlugin.unregisterNativeAdFactory(flutterEngine, "listTile"); } } You can now use the ListTileNativeAdFactory class to render native ads on Android. Deploy NativeAdFactory for Android (Kotlin) Open the android/build.gradle file (or any file in the Android folder) and click Open for editing in Android Studio to open the Android project. When prompted to choose a window to open a new project, click New Window to create Flutterstay open while working on an android project. Create your own screen layout With an Android project open in the Android Studio project panel, right-click the application and select New > Android Resource File from the context menu. In the New Resource File dialog, enter list_tile_native_ad.xml as the filename. Select "Layout" as the resource type and enter com.google.android.gms.ads.nativead.NativeAdView as the root element. Click OK to create a new layout file. Implement the ad layout as follows. Remember that the layout must match the visual design of the user interface of the platform it is designed for. list_tile_native_ad.xml Create ListTileNativeAdFactory class In the project pane, right-click the com.codelab.flutter.admobinlineads package and select New > File/Class Kotlin Enter ListTileNativeAdFactory as Nam en and select the list from class. You will see that the ListTileNativeAdFactory class is created in the com.codelab.flutter.admobinlineads package. Implement the ListTileNativeAdFactory class as follows. Notice that the class implements the createNativeAd() method in the GoogleMobileAdsPlugin.NativeAdFactory interface. The factory class is responsible for creating a view object to render the native display. As you can see from the code, the factory class creates a NativeAdView and fills it with a NativeAd object. ListTileNativeAdFactory.kt // TODO: Implement ListTileNativeAdFactory class package com.codelab.flutter.admobinlineads import android.content.Context import android.view.LayoutInflater import android.view.View import android.widget.ImageView import android.widget.comText import android .widget.com android. .google.android.gms.ads.nativead.NativeAd import com.google.android.gms.ads.nativead.NativeAdView import io.flutter.plugins.googlemobileads.GoogleMobileAdsPlugin class ListTileNativeAdFactory(val context: context):{ override fun createNativeAd(nativeAd: NativeAd, customOptions: MutableMap?): NativeAdView { val nativeAdView = LayoutInflater.from(context).inflate(R.layout.list_tile_native_ad, null) as NativeAdView with(nativeAdView) { val attributionViewSmall = findViewById(R.id.tv_list_tile_native_ad_attribution_small) val attributionViewLarge = findViewById(R.id.tv_list_tile_native_ad_attribution_large) val iconView = findViewById(R.id.iv) = nu list_tile_native_ad_icon) !SmallAd ll attributionView. Visibility = View.VISIBLE attributionViewLarge.visibility = View.INVISIBLE iconView.setImageDrawable(icon.drawable) } else { attributionViewSmall.visibility = View.INVISIBLE attributionViewLarge.visibility = View.VISIBLE } this.iconView = iconView val headlineView = findViewById (R.id.tv_list_tile_native_ad_headline) headlineView.text = nativeAd.headline this.headlineView = headlineView val bodyView = findViewById(R.i d.tv_list_tile_native_ad_body) with(bodyView) { text = nativeAd.body visibility = if (nativeAd. body.isNotEmpty()) View.VISIBLE else View.INVISIBLE } this.bodyView = bodyView setNativeAd(nativeAd) } return nativeAdView } } Class registration ListTileNativeAdFactory The NativeAdFactory instance must be registered with the GoogleMobileAdsPlugin plugin before using it on the Flutter side can be. Open the MainActivity.kt file and replace the configureFlutterEngine() and cleanUpFlutterEngine() methods. Register the ListTileNativeAdFactory class with a unique string identifier (listTile) in the configureFlutterEngine() method. MainActivity.kt class MainActivity: FlutterActivity() { override fun configureFlutterEngine(flutterEngine: FlutterEngine) { super.configureFlutterEngine(flutterEngine) // TODO: register ListTileNativeAdFactory GoogleMobileAdsPlugin.registerNativeAdFactory(flutterEngine, "listTile", ListTileNativeAdFactory(context)) } .. . } Any NativeAdFactoryneed to unregister during the cleanup process. Unregister the ListTileNativeAdFactory class with the cleanUpFlutterEngine() method. MainActivity.kt class MainActivity: FlutterActivity() { ... fun clearing overrides Use the ListTileNativeAdFactory class to display native ads on Android. Implementation of NativeAdFactory for iOS (Objective-C) If your project uses Swift instead of Objective-C, see Implementing NativeAdFactory for iOS (Swift) below. Open ios/podfile (or any file in the ios folder) and click "Open iOS Module" in Xcode to open the iOS project. Preparing the Native Ad Layout The native ad asset layout requires a custom view (*.xib). This code lab uses a readiness view to cut down on your efforts. Note. For more information about creating a custom view for native ads, see the code examples section of the developer documentation. With the iOS project open in Xcode, make sure ListTileNativeAdView.xib is included in the Runner project. Creating the ListTileNativeAdFactory Class In Project Navigator, right-click the runner group and select New File to create a header file for the new class. In the template dialog, select the header file and name it ListTileNativeAdFactory. After creating the ListTileNativeAdFactory.h file, define the ListNativeAdFactory class as follows:

ListTileNativeAdFactory: NSObject @end #endif /* ListTileNativeAdFactory_h */// Create an Objective-C lib by choosing New File in the Runner group. In the next dialogue ListTileNativeAdFactory in the File box and select Empty File as the lib type. When you click Next, you will be asked to select a folder to create the new file. Leave everything as it is and click Create. Implement the ListTileNativeFactory class as follows. Note that the FLTNativeAdFactory class implements the createNativeAd() method in the protocol. The factory class is responsible for creating a view object to display native ads. As you can see from the code, the factory class creates a GADNativeAdView object and populates it with a GADNativeAd object. ListTileNativeAdFactory.m // TODO: Import ListTileNativeAdFactory.h // TODO: ListTileNativeAdFactory implementation @implementation ListTileNativeAdFactory - (GADNativeAdView *)createNativeAdFactory:(CreateNativeAdFactory)createNativeAdFactory *)nativeAd customOptions:(NSDictionary *)customOptions{ return [[NSBundle mainBundle] loadNibNamed:@"ListTileNativeAdView" owner:nil options:nil].firstObject; ((UILabel *)nativeAdView.headlineView).text = nativeAd.headline; ((UILabel *)nativeAdView.bodyView).text = nativeAd.body; nativeAdView.bodyView.hidden = nativeAd.body? NOT THAT: ((UIImageView *)nativeAdView.iconView).image = nativeAd.icon.image; nativeAdView.iconView.hidden = nativeAdView.icon? NOT THAT; nativeAdView.callToActionView.userInteractionEnabled = NO; nativeAdvertising.nativeAdvertising = nativeAdvertising; reciprocal local advertising; } @end Register the ListTileNativeAdFactory class so that the "AppDelegate.h" #import "GeneratedPluginRegistrant.h" // TODO: Import ListTileNativeAdFactory.h #import "ListTileNativeAdFactory.h" @implementation AppDelegate — (BOOL) application:(UIApplication*)finishLaunchingWithOptions:(NSDictionary *)launchOptions { [GeneratedPluginRegistrant registerWithRegistry:self]; // TODO: Register ListTileNativeAdFactory ListTileNativeAdFactory *listTileFactory = [[ListTileNativeAdFactory alloc] init]; [FLTGoogleMobileAdsPlugin registerNativeAdFactory:self factoryId:@"listTile" nativeAdFactory:listTileFactory]; // Overrides the customization point when the application starts. return [super app: didFinishLaunching WithOptions:launchOptions app]; } @end You can now render native ads on iOS using ListTileNativeAdFactory. NativeAdFactory implementation for iOS (Swift) Open the ios/Podfile (or any file in the ios folder) and click Open iOS Module in Xcode to open the iOS project. Preparing a native ad layout To layout a native ad asset, you need a custom display (*.xib). This code lab uses a predefined view to minimize effort. Note: For more information on creating a custom native ads view, see the developer documentation in the sample code section. With the iOS project open in Xcode, check that ListTileNativeAdView.xib is present in the Runner project. Create the ListTileNativeAdFactory class In the project navigator, right-click on the runner group and select New File to create a header file for the new class. In the template dialog, select the Swift file and name it ListTileNativeAdFactory. After creating the ListTileNativeAdFactory.swift file, implement the ListNativeAdFactory class. Note that the class implements the createNativeAd() method in the FLTNativeAdFactory protocol. The factory class is responsible for creating the view object to render the native view. As you can see from the code, the factory class creates a GADNativeAdView object and populates it with a GADNativeAd object. ListTileNativeAdFactory.swift // TODO: import google_mobile_ads import google_mobile_ads // TODO: ListTileNativeAdFactory implementation ListTileNativeAdFactory class implementation ListTileNativeAdFactory : FLTNativeAdFactory { func nibView = Bundle.main.loadNibNamed("ListTileNativeAdView", owner: nil, options: nil)! First let nativeAdView = nibView as ! GADNativeAdView (nativeAdView.headlineView as! UILabel).text = nativeAd.body nativeAdView.bodyView?.isHidden = nativeAd.body == nil UIImageView) " , before it can be used from a Flutter page. Open the AppDelegate.m file and register the ListTileNativeAdFactory with a unique string identifier (listTile) by calling the FLTGoogleMobileAdsPlugin.registerNativeAdFactory() method. AppDelegate.swift import Flutter // TODO: Import google_mobile_ads import google_mobile_ads @UIApplicationMain @ google obj< class AppDelegate: FlutterAppDelegate { overrride func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions. .register(with: self) // TODO: Register ListTileNativeAdFactory let listTileFactory = ListTileNativeAdFactory(FLTGoogleMobileAdsPlugin.registerNativeAdFactory(self, factoryId: "listTile", nativeAdFactory : listTileFactory) return app, didinch readyFactory} using ListTileNativeAdFactory to render iOS native ads. Open lib/native_inline_page.dart. Then import the google_mobile_ads.dart files by adding the following lines: native_inline_page.dart ... // TODO: Import ad_helper.dart import// TODO: import google_mobile_ads.dart import 'package:google_mobile_ads/google_mobile_ads.dart'; The NativeInlinePage class extends StatefulWidget { ... } Add the following elements and methods to your own display in the _NativeInlinePageState class. Note that _kAdIndex specifies the index at which the banner ad appears and is used to calculate the item's index using the _getDestinationItemIndex() method. native_inline_page.dart class _NativeInlinePageState extends State { // TODO: add _kAdIndex static final _kAdIndex = 4; // TODO: add _getDestinationItemIndex(int _getDestinationItemIndex(intrawIndex) { if (rawIndex >= _kAdIndex && _ad != null) { return rawIndex - 1; } return the raw index; } ... } In the initState() method, create and load a NativeAd that uses the ListTileNativeAdFactory to create the native ad view. Note that it uses the same factory ID (listTile) that was used to register the factory with the plugin. native_inline_page.dart @override void initState() { super.initState(); // TASKS: Instantiate NativeAd _ad = NativeAd(adUnitId: AdHelper.nativeAdUnitId, factoryId: 'listTile', Request: AdRequest(), Listener: NativeAdListener(onAdLoaded: (ad, error) { // Discard of resource if loading fails ad.dispose(); print("Failed to load ad {code=${error.code} message=${ Error message }}"); },); ad.load(); } Change the build() method to display the banner, if available. Update itemCount to count the banner ed entry and update itemBuilder to display the banner ad in the ad index (_kAdIndex) when the ad is loaded. Update the code to use the _getDestinationItemIndex() method to get the index of the content item. native_inline_page.dart @override Widget Builder (BuildContext context) { return Scaffold(... body: ListView.builder(// TODO: set itemCount based on display loading state itemCount: widget.entries.length + (_ad != null ? 0): 0), itemBuilder: (context, index) { // TODO: Display ad banners if (_ad != null && index == _kAdIndex) { return Container(height: 72.0, alignment: Alignment.center, child: AdWidget(ad : _ad)advertisement)),); } else { // TODO: Get custom item index with _getDestinationItemIndex()(final item = widget.entries[_getDestinationItemIndex(index)]; return ListTile(...); } },); } Release the resource associated with the NativeAd object by calling the NativeAd.dispose() method in the distribution() callback method. native_inline_page.dart @override void deletep() { // TODO: Delete NativeAd object _ad?.dispose(); super.delete(); } That's all! Launch the project and click the Native Inline Ad button on the home page. After the ad has loaded, you will see the native ad in the middle of the list. You have completed the code lab. You can find the completed code for this code lab in the complete or complete_kotlin_swift folder. folder.