**Stripe Payment Gateway Integration: Flutter**

* Used [flutter\_stripe](https://pub.dev/packages/flutter_stripe)

#### Android

This plugin requires several changes to be able to work on Android devices. Please make sure you follow all these steps:

1. Use Android 5.0 (API level 21) and above.
2. Use Kotlin version 1.8.0 and above: [example](https://github.com/flutter-stripe/flutter_stripe/blob/main/example/android/settings.gradle#L22)
3. Requires Android Gradle plugin 8 and higher
4. Using a descendant of Theme.AppCompat for your activity: [example](https://github.com/flutter-stripe/flutter_stripe/blob/main/example/android/app/src/main/res/values/styles.xml#L15), [example night theme](https://github.com/flutter-stripe/flutter_stripe/blob/main/example/android/app/src/main/res/values-night/styles.xml#L16)
5. Using an up-to-date Android gradle build tools version: [example](https://github.com/flutter-stripe/flutter_stripe/blob/main/example/android/build.gradle#L9) and an up-to-date gradle version accordingly: [example](https://github.com/flutter-stripe/flutter_stripe/blob/main/example/android/gradle/wrapper/gradle-wrapper.properties#L6)
6. Using FlutterFragmentActivity instead of FlutterActivity in MainActivity.kt: [example](https://github.com/flutter-stripe/flutter_stripe/blob/79b201a2e9b827196d6a97bb41e1d0e526632a5a/example/android/app/src/main/kotlin/com/flutter/stripe/example/MainActivity.kt#L6)
7. Add the following rules to your proguard-rules.pro file: [example](https://github.com/flutter-stripe/flutter_stripe/blob/master/example/android/app/proguard-rules.pro)

-----------------

### Steps to Obtain Stripe Publishable Key for Testing:

1. **Create a Stripe Account**:
   * Go to Stripe Signup Page and create a free Stripe account.
   * If you already have an account, log in to the Stripe Dashboard.
2. **Enable Test Mode**:
   * Once logged in, switch to **Test Mode**.
     + In the Stripe Dashboard, you will see a toggle for "Test mode" (usually at the top-right corner of the page).
3. **Get Your API Keys**:
   * Navigate to **Developers** → **API keys**.
   * In Test Mode:
     + You’ll see the **Publishable Key** (e.g., pk\_test\_\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*).
     + You’ll also see the **Secret Key** (e.g., sk\_test\_\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*).
   * Copy the **Test Publishable Key** and use it in your Flutter app.
4. **Add the Key to Your App**:
   * Update your Stripe configuration in Flutter with the **Test Publishable Key**. For example:

dart

Copy code

import 'package:flutter\_stripe/flutter\_stripe.dart';

void main() {

Stripe.publishableKey = "pk\_test\_XXXXXXXXXXXXXXXXXXXXXXXX";

runApp(MyApp());

}

1. **Testing Payments**:
   * Use Stripe's test credit card numbers to simulate payments.
     + For example, use the card number 4242 4242 4242 4242 with any valid expiration date (e.g., 12/34) and CVC (e.g., 123).

### Dummy Test API Key (Example):

You **must obtain your own key** for development, but a typical **Test Publishable Key** format looks like this:

Copy code

pk\_test\_51HxxxxxL87Yjxxxxxxxxxxxxxxxxxxxxx

### Why Use Test Mode?

* **Test Mode** keys allow you to simulate payments without processing real transactions.
* Stripe provides sandbox data for testing purposes, so you don’t need to worry about real money or fraud detection during development.

**dangerouslyUpdateCardDetails (Why not to use)**

There is a significant **problem** with using the dangerouslyUpdateCardDetails method in your Flutter app, especially if you are collecting raw credit or debit card information from the user and sending it to your own server.

Here’s a breakdown of the issue and why this can result in **app rejection** from both the **Apple App Store** and **Google Play Store**:

### 1. **Violating PCI DSS Compliance**

* **PCI DSS** (Payment Card Industry Data Security Standard) is a set of security standards required for handling credit and debit card data.
* By collecting and sending sensitive card details to your server, **you are responsible for PCI compliance**, which is extremely complex and costly.
* Stripe is a PCI-compliant service provider and offers tools like **Stripe Elements** or **Stripe's Payment Intents API** to ensure card data never touches your server.
* Using dangerouslyUpdateCardDetails circumvents Stripe's security model and exposes raw card data to your app and server, violating PCI DSS compliance.

### 2. **Apple App Store Guidelines**

Apple’s App Store Review Guidelines explicitly state that apps **must not directly handle credit card data**. The relevant guideline is:

**Guideline 5.2.1** - Apps must use **secure payment mechanisms** provided by the platform or payment services (like Stripe) to handle credit card transactions. Apps that collect or store credit card information outside secure APIs can be rejected.

Apple encourages developers to use secure solutions such as **Apple Pay**, **third-party PCI-compliant SDKs**, or web-based Stripe Checkout.

### 3. **Google Play Store Policies**

Google Play also has strict policies about financial transactions. According to their guidelines:

* Apps must use **secure and compliant methods** to process payments.
* Directly handling sensitive user information, like raw card details, outside secure SDKs (e.g., Stripe’s official integration) **violates Play Store’s data security policies**.
* Using non-compliant payment methods or insecure practices can result in app rejection or removal.

### 4. **Security Risks**

The method dangerouslyUpdateCardDetails itself comes with a warning from Stripe because it bypasses Stripe’s standard security measures. By directly handling sensitive card information, you expose your app and server to:

* Data breaches or hacks.
* Accidental leaks of cardholder data.
* Legal consequences if card data is compromised.
* Loss of trust from your users.

### Recommendation

Instead of using dangerouslyUpdateCardDetails, use Stripe’s **official and secure methods** for handling payments in Flutter:

1. **Stripe Elements/PaymentSheet**: Use Stripe’s PaymentSheet for collecting payment details securely.
   * This way, card data is securely tokenized and processed without ever touching your app server.
2. **Stripe Payment Intents API**: Use Payment Intents for the entire flow, ensuring PCI compliance.
3. **Apple Pay** and **Google Pay**: Integrate these payment methods as they are natively secure and trusted.

### Conclusion

If you continue to use dangerouslyUpdateCardDetails and send card details to your server, your app is likely to get **rejected** from both the Apple App Store and Google Play Store. Additionally, you may face **PCI compliance violations** and security risks.

To prevent this, switch to Stripe’s official methods like **PaymentSheet** or **Payment Intents API** to ensure security, compliance, and app approval.