

PayU S.A. Tel. +48 61 630 60 05
182, Grunwaldzka Str. Email: pomoc@payu.pl
60-166 Poznań www.payu.pl
Poland



PayTouch

Android

v1.4.3

Mobile Payments for PayU platform

PayTouch Android Integration

The PayTouch library makes it easy to integrate mobile payments in the Android application. The library is available in form of a single JAR dependency. It provides both network communication, security and user interface for the payment process. It takes care of the payment method management, processing payment and payment authorization (CVV, 3D Secure, bank transfer on the Web, etc.)

The main use case is to accept single payment with following payment methods:

- Credit/Debit cards (Visa, Mastercard, Maestro)

Requirements

- Android 2.3.x or later
- Server-side OAuth2 token retrieval – described in a separate document

Importing the SDK

PayU SDK is available as a maven artifact. In order to use it in your build system follow steps below.

Add remote repository

Maven (pom.xml):

```
1. <repositories>
2.   <repository>
3.     <id>payu-mvn-repo</id>
4.     <url>https://raw.github.com/PayU/paytouch-android/mvn-repo</url>
5.   </repository>
6. </repositories>
```

Gradle (build.gradle):

```
1. repositories {
2.   maven { url "https://raw.github.com/PayU/paytouch-android/mvn-repo/" }
3.   mavenCentral()
4. }
```

Add required dependencies

Maven (pom.xml)

```
1. <dependencies>
2.   <!-- Other dependencies -->
3.   <dependency>
4.     <groupId>com.payu.android.sdk</groupId>
5.     <artifactId>payment-library-widget</artifactId>
6.     <version>1.4.3</version>
7.     <exclusions>
8.       <exclusion>
9.         <groupId>com.android.support</groupId>
10.        <artifactId>support-v4</artifcactId>
11.      </exclusion>
12.    </exclusion>
13.    <groupId>org.jetbrains</groupId>
14.    <artifactId>annotations</artifactId>
15.  </exclusion>
16. </exclusions>
17. </dependency>
18. <dependency>
19.   <groupId>com.payu.android.sdk</groupId>
20.   <artifactId>payment-library-full</artifactId>
21.   <version>1.4.3</version>
```

```
22.         <exclusions>
23.             <exclusion>
24.                 <groupId>com.android.support</groupId>
25.                 <artifactId>support-v4</artifactId>
26.             </exclusion>
27.             <exclusion>
28.                 <groupId>org.jetbrains</groupId>
29.                 <artifactId>annotations</artifactId>
30.             </exclusion>
31.         </exclusions>
32.
33.     </dependency>
34. </dependencies>
```

Gradle (build.gradle)

```
1.     dependencies {
2.         ...
3.         compile 'com.payu.android.sdk:payment-library-full:1.4.3'
4.         {
5.             exclude group: 'com.android.support', module: 'support-v4'
6.             exclude group: 'org.jetbrains', module: 'annotations'
7.         }
8.         compile 'com.payu.android.sdk:payment-library-widget:1.4.3'
9.         {
10.            exclude group: 'com.android.support', module: 'support-v4'
11.            exclude group: 'org.jetbrains', module: 'annotations'
12.        }
13.     }
```

Credentials & testing

Production mode

For production environment you will have to retrieve OAuth2 access token from the PayU backend service via your backend application. This process must be conducted by your backend since direct storage of your `client_id` and `client_secret` in the application is **prohibited**. The only request the application should do is “retrieve access token for currently logged user”.

Sandbox mode

Sandbox environment needs to be configured like production. This mode is created for test purpose so cards or bank account should not be charged. Additionally to production this mode lets you see some logs that helps with integrating PayTouch SDK.

Test mode with mocked network

For testing and integration purposes we provide a test mode, which later will be referred as local mode. Local mode is simplified and **does not** perform network connections to the PayU system. All data provided stays in the application and therefore **has no effect** on the production system. Test mode payment **will not invoke** payment status notification from PayU to your backend.

Localization

The SDK has support for Polish, English, Czech and German language. By default language selection is automatic with respect to the user's device interface language. It is possible to force the language if your application requires to do so.

Documentation

- This document, showing basic introduction & usage
- The sample application provided within SDK bundle
- JavaDoc – refer to `PaymentService` class for in-depth explanation
- Server-side integration documentation

Integration tutorial

AndroidManifest.xml

```
1. <!-- PayU SDK configuration -->
2. <uses-permission android:name="android.permission.INTERNET" />
3.
4. <activity android:name="com.payu.android.sdk.payment.ui.NewCardActivity" />
5. <activity android:name="com.payu.android.sdk.payment.ui.PaymentMethodListActivity" />
6. <activity
7.     android:name="com.payu.android.sdk.payment.ui.LocalCardCheckActivity"
8.     android:theme="@android:style/Theme.Translucent" />
9. <activity android:name="com.payu.android.sdk.payment.ui.LoginActivity" />
10. <activity
11.     android:name="com.payu.android.sdk.payment.ui.PaymentActivity"
12.     android:theme="@android:style/Theme.Translucent" />
13. <activity
14.     android:name="com.payu.android.sdk.payment.ui.AuthorizationActivity"
15.     android:theme="@android:style/Theme.Translucent" />
16. <activity android:name="com.payu.android.sdk.payment.ui.StrongAuthorizationActivity" />
17.
18. <service android:name="com.payu.android.sdk.payment.service.PaymentEntrypointService"
19.     android:exported="false"/>
```

Starting payment method management & payment

```
1. import com.payu.android.sdk.payment.PaymentService;
2.
3. public class CartActivity extends Activity {
4.     protected void onCreate(Bundle savedInstanceState) {
5.         super.onCreate(savedInstanceState);
6.         setContentView(R.layout.activity_cart);
7.         /** Start payment method management */
8.         PaymentService.createInstance(this).startPaymentMethodChooser();
9.     }
10.
11.
12.     // set onClick on a button
13.     private void startPayment() {
14.         PaymentService.createInstance(this)
15.             .pay(new Order.Builder()
16.                 .withAmount(500) // 5 PLN
17.                 .withCurrency(Currency.PLN)
18.                 .withDescription("Example payment")
19.                 .build());
20.     }
21. }
```

Integration tutorial - full

Complete example of usage is provided as an example application. Here we present minimum set of configuration required.

AndroidManifest.xml

```
20. <!-- PayU SDK configuration -->
21. <uses-permission android:name="android.permission.INTERNET" />
22.
23. <activity android:name="com.payu.android.sdk.payment.ui.NewCardActivity" />
24. <activity android:name="com.payu.android.sdk.payment.ui.PaymentMethodListActivity" />
25. <activity
26.     android:name="com.payu.android.sdk.payment.ui.LocalCardCheckActivity"
27.     android:theme="@android:style/Theme.Translucent" />
28. <activity android:name="com.payu.android.sdk.payment.ui.LoginActivity" />
29. <activity
30.     android:name="com.payu.android.sdk.payment.ui.PaymentActivity"
31.     android:theme="@android:style/Theme.Translucent" />
32. <activity
33.     android:name="com.payu.android.sdk.payment.ui.AuthorizationActivity"
34.     android:theme="@android:style/Theme.Translucent" />
35. <activity android:name="com.payu.android.sdk.payment.ui.StrongAuthorizationActivity" />
36.
37. <service android:name="com.payu.android.sdk.payment.service.PaymentEntrypointService"
38.     android:exported="false"/>
```

Configuration file & instantiation

You will have to provide required configuration values via XML file placed in a `res/values` folder of your Android project.

Example configuration file:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <resources>
3.
4.     <string name="payu_language">auto</string> <!-- auto | Polish | English | Czech | German -->
5.     <string name="payu_environment">local</string> <!-- local | production | sandbox -->
6.     <string name="payu_token_service_class_full_qualified_name">com.example.service.YourTokenProvide
rService</string>
7.
8. </resources>
```

The main and only entry point from your perspective is the `PaymentService`. The configuration is validated in runtime, in case of any mistake an `IllegalConfigurationException` will be

thrown during `PaymentService` instantiation, containing detailed cause explanation. In order to instantiate the `PaymentService` class you should use following method:

1. `PaymentService.createInstance(android.content.Context);`

Token Provider Service

In order to use PayTouch SDK in a production environment you have to implement a class that extends `TokenProviderService`. Instance of this class will be created by PayTouch SDK, in order to obtain valid access token. For specific information refer to `TokenProviderService` in the JavaDoc.

In the example application you can find the implementation that may be used as a framework. Your implementation will be called by the SDK in a background thread, contact your server to obtain PayU access token for currently logged user. Example:

```
1. public class MerchantTokenProviderService extends TokenProviderService {
2.
3.     public MerchantTokenProviderService(Context context) {
4.         super(context);
5.     }
6.
7.     @Override
8.     public MerchantOAuthAccessToken provideAccessToken() throws ExternalRequestError {
9.
10.        try {
11.            // fetch access token from your servier
12.            // return ...
13.
14.            // or reports problems by throwing an exception
15.            // Report problems
16.            throw isNetworkProblem()
17.                ? new ExternalRequestError(ExternalErrorType.NETWORK_ERROR)
18.                : new ExternalRequestError(ExternalErrorType.SERVER_ERROR);
19.        }
20.    }
21. }
```

User logout notification

When user decides to logout from your application, PayU SDK **must be notified** about it, in order to clear sensitive user data. It is **critical to call** `PaymentService#notifyUserLogout()` after successful user logout to avoid unexpected SDK behaviour.


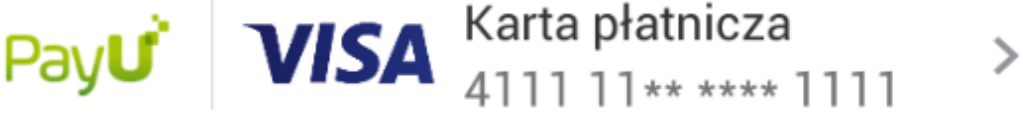
Payment method widget

PayTouch SDK provides complete payment method management. Your user is able to select or add payment method across different scenarios. In order to access PayU SDK and visualize selected payment method to your user use `PaymentMethodWidget` view in your layout.

your_cart_layout.xml

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:app="http://schemas.android.com/apk/res-auto"
3.     android:orientation="vertical"
4.     android:layout_width="match_parent"
5.     android:layout_height="wrap_content">
6.
7.     <!-- Your layout -->
8.
9.     <com.payu.android.sdk.payment.widget.PaymentMethodWidget
10.        android:id="@+id/payment_method_widget "
11.        android:layout_width="match_parent"
12.        android:layout_height="wrap_content"
13.        app:payuTheme="flat"
14.        android:layout_margin="@dimen/view_margin_small" />
15.
16.     <!-- Your layout -->
17.
18. </LinearLayout>
```

`PaymentMethodWidget` uses optional XML attribute `payuTheme` to specify theme. It defaults to `flat`.

payuTheme	Preview
flat	
classic	

`PaymentMethodWidget#isPaymentMethodPresent` allows to check if payment method is already selected in order to continue payment process. Example code:

```
1. private void startPayment() {  
2.     PaymentMethodWidget paymentMethodWidget =  
3.         (PaymentMethodWidget) findViewById(R.id.payment_method_widget);  
4.  
5.     if (paymentMethodWidget.isPaymentMethodPresent()) {  
6.         // Payment code goes here  
7.     }  
8. }
```

EventBus

Overview

In order to receive feedback from all your requests you have to register to an event bus.

Available events are:

- `PaymentSuccessEvent`
- `PaymentFailedEvent`
- `PresentSelectedPaymentMethod`

- `AbsentSelectedPaymentMethod`

Please refer to Javadoc for specific explanation.

Registration

Preferably you should create new `PaymentEventBus` instance using the default constructor and register in `Activity.onResume()` and then unregister in `Activity.onPause()` method as in the example below.

```
1. import com.payu.android.sdk.payment.PaymentEventBus;
2. import com.payu.android.sdk.payment.PaymentService;
3.
4. public class CartActivity extends Activity {
5.
6.     private PaymentService mPaymentService;
7.
8.     private PaymentEventBus mPaymentEventBus = new PaymentEventBus();
9.
10.    protected void onPause() {
11.        mPaymentEventBus.unregister(this);
12.        super.onPause();
13.    }
14.
15.    protected void onResume() {
16.        super.onResume();
17.        mPaymentEventBus.register(this);
18.    }
19. }
```

Event subscription

Event subscription is done by convention. In the object passed to registration method you have to implement a public handler method in order to receive events. Method should begin with `public void onPaymentProcessEvent...` and then should specify a Thread on which event will be received.

Available modes are:

- `MainThread` - Subscriber will be called in Android's main thread
- `BackgroundThread` - Subscriber will be called in a background thread.

An example of subscriber method:

```
1. public void onPaymentProcessEventMainThread(PaymentSuccessEvent event) {  
2.     // Success payment handling code goes here  
3. }
```

Payment

An entity used to settle payment request is an `Order`. For your convenience we provide fluent API for creating Orders. Please refer to `Order.Builder`.

Payment is the last point of the process. When you submit the `Order` the user will be directed through the payment process. In the best case scenario the user will see progress dialog and the payment will be finished with no interaction. However keep in mind that the user may be asked to confirm payment using some verification method (CVV, 3DS, etc.). All the UI is provided by PayU and you are not required to do any action. Please refer to `Order.Builder` to learn how to build `Order` instances.

In SDK when using `local` environment if you set payment to be less than 100 PLN there will be thrown `GENERIC_ERROR`. This will let you test 'error' path.

Note: Every single payment must be confirmed server side as described [here](#).

Example payment:

```
1. private void startPayment() {  
2.     mPaymentService.pay(new Order.Builder()  
3.         // Notifications are sent in JSON format using POST method  
4.         .withNotifyUrl("http://notify.me/notify-endpoint")  
5.         // Payment amount defined in a fractional part of defined Currency  
6.         .withAmount(500)  
7.         // Currency instance  
8.         .withCurrency(Currency.PLN)  
9.         // Order identifier assigned by the Seller  
10.        .withExtOrderId(UUID.randomUUID().toString())  
11.        // Order description that will be displayed to the user  
12.        .withDescription("Example payment")  
13.        .build());  
14. }
```

Status retrieval:

```
1. public void onPaymentProcessEventMainThread(PaymentSuccessEvent event) {  
2.     // Payment succeeded, confirm status  
3. }  
4.  
5. public void onPaymentProcessEventMainThread(PaymentFailedEvent event) {  
6.     // Payment failed or canceled, check event for details  
7. }
```

ProGuard

In case you are using ProGuard, following lines have to be added to ProGuard configuration file:

```
1. # PayU SDK  
2. -keepclassmembers class ** {  
3.     public void onPaymentProcessEvent* (**);  
4. }  
5. -keep class com.payu.android.sdk.** { *; }  
6. -keep,allowoptimization class * extends com.payu.android.sdk.payment.TokenProviderService  
7. -keepclassmembers class * extends com.payu.android.sdk.payment.TokenProviderService {  
8.     public <init>(android.content.Context);  
9. }
```