

Dział Pomocy Technicznej dotpay
ul. Wielicka 72, 30-552 Kraków
tel. +48 12 688 26 00
faks +48 12 688 26 49
e-mail: tech@dotpay.pl



Płatności elektroniczne **klasy e-biznes**

SDK (Android)

Wersja 1.2.0



SPIS TREŚCI

SPIS TREŚCI	2
WSTĘP	3
DOKUMENTY ZALĘŻNE	3
Rozpoczęcie pracy z biblioteką	4
USTAWIENIA PROJEKTU	4
USTAWIENIE JĘZYKA	4
WYBÓR SYSTEMU	5
OPCJE DODATKOWE	5
WERSJA SDK	5
Płatność	6
REJESTRACJA CALLBACKA POWROTU	6
INICJALIZACJA PŁATNOŚCI	6
ZAKOŃCZENIE PŁATNOŚCI	9
SZCZEGÓŁY PODSUMOWANIA	10
DOSTĘPNE WALUTY	10
ZMIANA STYLU PREZENTACJI	11
WŁASNA KONTROLKA WYBORU KANAŁU	15
WŁASNA KONTROLKA PODSUMOWANIA TRANSAKCJI	15
Obsługa kanałów specjalnych	16
PŁATNOŚĆ KARTĄ - 1CLICK	17
INICJALIZACJA PŁATNOŚCI 1CLICK	17
WYKORZYSTANIE WBUDOWANEJ KONTROLKI MENADŻER KART	18
REGULAMINY PRZY DODANIU KARTY	19
ZMIANA STYLU PREZENTACJI	19
WŁASNA KONTROLKA „MENEDŻER KART”	20
1. LISTA ZAREJESTROWANYCH KART	21
2. REJESTRACJA KARTY	21
3. WYREJESTROWANIE KARTY	21
4. USTAWIENIE KARTY DOMYŚLNEJ	22
Historia i status transakcji	23
WYKORZYSTANIE WBUDOWANEJ KONTROLKI	24
ZMIANA STYLU PREZENTACJI	24
WŁASNA KONTROLKA HISTORII	26

WSTĘP

Niniejszy dokument ma na celu opis sposobu wykorzystania biblioteki dla platformy Android, umożliwiającej Developerowi szybkie i wygodne umieszczenie w tworzonej przez siebie aplikacji mobilnej procesu płatności przeprowadzanego za pomocą systemu Dotpay.

Dzięki przeniesieniu możliwie dużej liczby kroków procesu ze strony www do aplikacji mobilnej, proces płatności jest znacznie wygodniejszy dla Użytkownika, a Developer zyskuje nad nim pełniejszą kontrolę.

W celu najlepszego wpisania elementów SDK w aplikację Kontrahenta, możliwa jest konfiguracja stylu prezentacji informacji (kolorystyka, czcionki).

Biblioteka została stworzona w języku Java. Wspierany jest system Android w wersji 4.1 (API:16, JELLY BEAN) i wyższej.

W dokumencie zastosowana następujące pojęcia i oznaczenia:

Kontrahent / Sprzedawca	Użytkownik serwisu Dotpay pobierający płatność lub właściciel aplikacji, na której rozpoczyna się proces płatności
Sklep	Sklep internetowy Kontrahenta, dla którego aplikacja mobilna jest frontendem
Użytkownik / Kupujący	Osoba dokonująca wpłaty na rzecz Kontrahenta za pośrednictwem aplikacji mobilnej
Developer	Programista, który tworzy aplikację mobilną dla kontrahenta

Dokumenty zależne

Instrukcja techniczna implementacji płatności – dokumentacja opisująca podstawowy proces płatności dla sklepów w Internecie, do pobrania z Panelu Sprzedawcy systemu Dotpay.

Rozpoczęcie pracy z biblioteką

Aby skorzystać z biblioteki, należy dodać ją do projektu oraz w odpowiedni sposób zainicjować. Szczegóły tych czynności opisane zostały w kolejnych podrozdziałach.

W celu ułatwienia procesu rozpoczęcia korzystania z biblioteki dostarczona została także aplikacja testowa, znajdująca się w podkatalogu `example` paczki z biblioteką.

Ustawienia projektu

1. Do projektu należy dodać plik biblioteki, znajdujący się w katalogu `lib` paczki.
2. Do projektu należy dodać biblioteki suportowe Androida:

```
compile 'com.android.support:appcompat-v7:25.3.1'
compile 'com.android.support:cardview-v7:25.3.1'
compile 'com.android.support:recyclerview-v7:25.3.1'
compile 'com.android.support:design:25.3.1'
```

3. Do projektu należy dołączyć biblioteki wymagane przez SDK. Należy dodać poniższe zależności do projektu:

```
compile 'org.jsoup:jsoup:1.7.2'
compile 'com.android.support:recyclerview-v7:25.3.1'
compile 'com.squareup.retrofit2:retrofit:2.0.2'
compile 'com.squareup.okhttp3:logging-interceptor:3.2.0'
compile 'com.mcxiaoke.volley:library:1.0.0'
compile 'com.squareup.picasso:picasso:2.5.2'
compile 'de.greenrobot:eventbus:2.2.0'
compile 'com.j256.ormlite:ormlite-core:4.47'
compile 'com.j256.ormlite:ormlite-android:4.47'
compile 'io.card:android-sdk:5.4.1'
```

Biblioteki te dołączone zostały także do aplikacji przykładowej.

4. Zainicjalizować SDK, dodając w metodzie `onCreate` klasy `Application` wywołanie :

```
AppSDK.initialize(this);
```

Ustawienie języka

Biblioteka, w obecnej wersji, obsługuje proces płatności w języku angielskim oraz polskim. Planowane jest rozszerzenie o kolejne języki, stąd zalecane jest wykonanie dynamicznej konfiguracji. Aby to zrobić, należy:

1. Pobrać listę dostępnych języków z `PaymentManager`:

```
PaymentManager.getInstance().getLanguages();
```

2. Dopasować najbardziej odpowiedni język
3. Ustawić wybrany język

```
PaymentManager.getInstance().setApplicationLanguage(bestLang.getLang());
```

Wybór systemu

Biblioteka ma możliwość komunikowania się zarówno z testowym, jak i produkcyjnym systemem Dotpay. Wybór systemu jest konieczny, bez niego SDK nie będzie pracowało poprawnie.

Aby ustawić system testowy, należy wykonać następującą instrukcję:

```
PaymentManager.getInstance().setApplicationVersion(Configuration.TEST_VERSION);
```

Analogicznie, w celu ustawienia system produkcyjnego należy wykonać:

```
PaymentManager.getInstance().setApplicationVersion(Configuration.RELEASE_VERSION);
```

Opcje dodatkowe

Wyłączenie zapamiętywania ostatnio wybranego przez Użytkownika kanału:

```
Configuration.loadLastSelectedChannel(false);
```

Wersja SDK

Zalecamy wyświetlenie w aplikacji końcowej wersji SDK, co pozwoli ułatwić proces diagnozowania problemów. Wersję SDK można pobrać wykonując metodę:

```
Settings.getSDKVersion();
```

Płatność

Całość procesu płatności polega na przekazaniu sterowania do klasy `PaymentManager` oraz oczekiwaniu na wywołanie zdarzeń błędu lub sukcesu zwracających sterowanie.

Biblioteka przeprowadzi Użytkownika przez proces wyboru kanału płatności, podawania/weryfikacji danych płatącego, wyboru opcji dodatkowych, akceptacji stosownych regulaminów, oraz samego dokonania płatności.

W wypadku płatności za rzeczywiste towary, informację o statusie płatności należy traktować jedynie informacyjnie, właściwy status transakcji zostanie dostarczony systemom backendowym zgodnie z Instrukcją techniczną implementacji płatności Dotpay.

W kolejnych podrozdziałach opisane są kroki, które należy wykonać, aby skorzystać z klasy `PaymentManager`, oraz opcje dodatkowe, pozwalające na dostosowanie procesu do własnych celów.

Rejestracja callbacka powrotu

Przygotowanie procesu płatności należy rozpocząć od przygotowania callbacków nasłuchujących sygnalizowanych zdarzeń końca płatności. W klasie, w której będzie inicjalizowana płatność należy zaimplementować interface `PaymentManagerCallback`:

```
public interface PaymentManagerCallback {  
    void onPaymentSuccess(PaymentEndedEventArgs paymentEndedEventArgs);  
    void onPaymentFailure(PaymentEndedEventArgs paymentEndedEventArgs);  
}
```

A następnie zarejestrować go w `PaymentManagerze`:

```
PaymentManager.getInstance().setPaymentManagerCallback(paymentManagerCallback);
```

Inicjalizacja płatności

Rozpoczęcie procesu płatności wykonujemy korzystając z metody `initialize PaymentManager`. Przyjmuje ona argumenty opisane w poniższej tabeli. Inicjalizacja ta musi się odbyć w ramach aktualnej aktywności Android.

PARAMETR	ZNACZENIE / OPIS
context	Typ: Context Kontekst, z którego ta metoda została wywołana
paymentInformation	Typ: PaymentInformation Obiekt przechowujący wszystkie parametry potrzebne do zapłaty

Konstruktor obiektu typu `PaymentInformation` przyjmuje argumenty zgodnie z poniższą tabelą:

PARAMETR	ZNACZENIE / OPIS
id	<u>Typ</u> : string ID konta w systemie Dotpay, na rzecz którego dokonywana jest płatność (ID konta Sprzedawcy)
amount	<u>Typ</u> : double Kwota transakcji
description	<u>Typ</u> : string Tytuł/opis płatności
currency	<u>Typ</u> : string <u>Domyślna wartość</u> : "PLN" Określenie w jakiej walucie podany jest parametr amount. Sposób na pobranie listę dostępnych walut opisany został w podrozdziale Dostępne waluty , poniżej.

Dodatkowo korzystając z setterów można dla obiektu typu `PaymentInformation` ustawić następujące parametry:

PARAMETR	ZNACZENIE / OPIS
senderInformation	<u>Typ</u> : Map<string, string> <u>Domyślna wartość</u> : null <u>Nazwa settera</u> : <code>setSenderInformation(Map<String, String> senderInformation)</code> Dodatkowe informacje o kupującym. Mapa z kluczami: "firstname" – imię; "lastname" – nazwisko; "email" – email; "phone" – telefon; "street" – ulica; "street_n1" – numer budynku; "street_n2" – numer mieszkania; "postcode" – kod pocztowy; "city" – miasto; "country" – kraj (3 literowe ISO3166). Wartości te nie są obowiązkowe. Zalecane jest podanie co najmniej imienia, nazwiska oraz adresu email. Dane te pozwolą na uzupełnienie formularza płatności. O dane brakujące SDK zapyta Użytkownika. Szczegółowe wyjaśnienie zawartości pól, ich znaczenie, znajduje się w Instrukcji technicznej implementacji płatności Dotpay.
additionalInformation	<u>Typ</u> : Map<string, string> <u>Domyślna wartość</u> : null

	<p><u>Nazwa settera:</u> <code>setAdditionalInformation (Map<String,String> additionalInformation)</code></p> <p>Dodatkowe parametry przekazywane podczas procesu płatności, zgodnie z szczegółami przekazanymi w dodatkowych dokumentacjach.</p>
control	<p><u>Typ:</u> string</p> <p><u>Domyślna wartość:</u> null</p> <p><u>Nazwa settera:</u> <code>setControl (String)</code></p> <p>Parametr identyfikujący daną płatność, przekazywany w potwierdzeniu płatności do Sklepu, potrzebny w celu dopasowania statusu płatności do właściwego zamówienia w Sklepie.</p> <p>Więcej informacji znajduje się w Instrukcji technicznej implementacji płatności Dotpay.</p> <p>Jeśli nie jest ustawiony, to jest generowany przez SDK.</p> <p><u>UWAGA</u></p> <p>W celu poprawnego działania historii płatności parametr ten powinien być unikalny dla każdej płatności.</p>
urlc	<p><u>Typ:</u> string</p> <p><u>Domyślna wartość:</u> null</p> <p><u>Nazwa settera:</u> <code>setUrlc (String)</code></p> <p>Adres URL Sklepu, do odbioru parametrów potwierdzających zrealizowanie lub odmowę realizacji transakcji.</p> <p>Więcej informacji znajduje się w Instrukcji technicznej implementacji płatności Dotpay.</p>

Przykład inicjalizacji płatności:

```
String description = "zamówienie 12345";
double amount = 123.45;
PaymentInformation paymentInformation = new PaymentInformation(merchant_Id, amount,
description, selectedCurrency);

Map<String, String> sender = new Map<String, String> {{ "firstname", "Jan"},
{"lastname", "Kowalski"}, {"email", "jan.kowalski@test.pl"}}
Map<String, String> additional = new Map<String, String> {{ "id1", "12345"},
{"amount1", "100"}, {"id2", "67890"}, {"amount2", "23.45"}}

paymentInformation.setSenderInformation(sender);
paymentInformation.setAdditionalInformation(additional);
PaymentManager.getInstance().initialize(ShopActivity.this, paymentInformation);
```

Strona | 9 / 26

Zakończenie płatności

Poprawne zakończenie procesu płatności sygnalizowane jest wywołaniem metody `onPaymentSuccess` tego interfejsu, błąd w procesie (zarówno podczas inicjalizacji parametrów, jak i późniejszym etapie) sygnalizowany jest wywołaniem `onPaymentFailure`. Zdarzenia te posiadają argument typu `PaymentEndedEventArgs` ze szczegółami:

PARAMETR	ZNACZENIE / OPIS
Result	<p><u>Typ:</u> <code>PaymentResult</code></p> <p>Informacje o płatności przekazane do procesu (kwota, waluta, control, id kanału)</p> <p>Status płatności znajduje się w polu <code>StateType</code></p> <p>Parametr pusty, jeśli przyczyną zakończenia procesu płatności był błąd.</p>
ErrorResult	<p><u>Typ:</u> <code>ProcessResult</code></p> <p>Określa przyczynę zakończenia procesu płatności. Wszystkie statusy inne niż „OK” oznaczają błąd, należy wyświetlić Użytkownikowi komunikat o problemie w realizacji płatności.</p>

Uwaga!!! Zakończenie procesu płatności z sukcesem nie oznacza, iż płatność została wykonana, a jedynie proces przebiegł bez błędów. Rezultat płatności zwrócony będzie w odpowiednim parametrze zdarzenia.

Przykładowe metody obsługi zdarzeń:

```
private PaymentManagerCallback paymentManagerCallback = new PaymentManagerCallback() {  
    @Override  
    public void onPaymentSuccess(PaymentEndedEventArgs paymentEndedEventArgs) {  
        if(paymentEndedEventArgs.getStateType() == StateType.COMPLETED) {  
            // płatność zakończona pomyślnie  
        } else if(paymentEndedEventArgs.getStateType() == StateType.REJECTED) {  
            //płatność zakończona odmową  
        } else {  
            //płatność w trakcie realizacji  
        }  
    }  
    @Override  
    public void onPaymentFailure(PaymentEndedEventArgs paymentEndedEventArgs) {  
        //wewnętrzny błąd procesu płatności,  
    }  
};
```

Szczegóły podsumowania

Na ekranie podsumowania dodatkowo można wyświetlić szczegóły płatności: opis, status, kwotę. Szczegóły domyślnie są wyłączone.

W celu włączenia funkcjonalności należy wywołać metodę:

```
Configuration.setPaymentDetailsResultEnable(true);
```

Dostępne waluty

Listę aktualnie obsługiwanych przez Dotpay walut można pobrać wykonując odpowiednią metodę PaymentManagera:

```
PaymentManager.getInstance().getCurrencies()
```

Zmiana stylu prezentacji

Aby zmienić sposób prezentacji elementów kontrolek procesu płatności, należy wywołać odpowiednie settery singletona Configuration. Ustawienia należy wykonać przed zainicjalizowaniem parametrów PaymentManagera.

Strona | 11 / 26

Globalne ustawienia:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
ToolBarBackgroundColor	<u>Typ:</u> int Definiuje kolor tła toolbara
ToolBarTitleTextColor	<u>Typ:</u> int Definiuje kolor tekstu na toolbarze
ButtonTitleTextColor	<u>Typ:</u> int Definiuje kolor tekstu na przycisku
ButtonBackgroundColorResource	<u>Typ:</u> int Definiuje kolor tła przycisku

Przykład:

```
Configuration.setToolBarBackgroundColor(R.color.red);  
Configuration.setToolBarTitleTextColor(R.color.white);  
Configuration.setButtonTitleTextColor(R.color.white);  
Configuration.setButtonBackgroundColorResource(R.drawable.colorfulBtn);
```

Kontrolka wyboru kanałów:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
ChannelBackgroundColor	<u>Typ:</u> int Definiuje kolor tła całego widoku
ChannelBackgroundItemColor	<u>Typ:</u> int Definiuje kolor tła pojedynczego kafelka
ChannelBackgroundPressItemColor	<u>Typ:</u> int Definiuje kolor tła kafelka po kliknięciu
ChannelItemTextColor	<u>Typ:</u> int Definiuje kolor tekstu na kafelku
ChannelTextGravity	<u>Typ:</u> int Definiuje położenie tekstu na kaflu

Przykład:

```
Configuration.setChannelBackgroundColor(R.color.white);  
Configuration.setChannelBackgroundItemColor(R.color.gray);  
Configuration.setChannelBackgroundPressItemColor(R.color.green);  
Configuration.setChannelTextGravity(Gravity.CENTER_HORIZONTAL);  
Configuration.setChannelItemTextColor(R.color.black);
```

Kontrolka ulubionych kanałów płatności:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
FavoriteChannelBackgroundItemColor	<u>Typ:</u> int Definiuje kolor tła pojedynczego kafelka
FavoriteChannelBackgroundPressItemColor	<u>Typ:</u> int Definiuje kolor tła kafelka po kliknięciu
FavoriteChannelItemTextStyle	<u>Typ:</u> int Definiuje styl tekstu na kafelku

Przykład:

```
Configuration.setFavoriteChannelBackgroundItemColor(R.color.white);  
Configuration.setFavoriteChannelBackgroundPressItemColor(R.color.gray);
```

Strona | 13 / 26

Kontrolka zmiany metody płatności w formularzu płatności:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
PaymentFormChannelText	<u>Typ:</u> int Definiuje tekst w kontrolce zmiany kanału
PaymentFormChannelTextColor	<u>Typ:</u> int Definiuje kolor tekstu w kontrolce
PaymentFormChannelTextSize	<u>Typ:</u> int Definiuje rozmiar czcionki w kontrolce
PaymentFormChannelTextGravity	<u>Typ:</u> int Definiuje położenie tekstu w kontrolce
PaymentFormChannelTextAllCaps	<u>Typ:</u> boolean Ustawia duże litery w tekście
PaymentAmountTextColor	<u>Typ:</u> int Definiuje kolor tekstu dla kwoty
PaymentReceiverTextColor	<u>Typ:</u> int Definiuje kolor tekstu dla odbiorcy
PaymentDescriptionTextColor	<u>Typ:</u> int Definiuje kolor tekstu dla opisu

PaymentInfoBackgroundColor	<u>Typ: int</u> Definiuje kolor tła kontrolki z danymi o płatności
----------------------------	---

Przykład:

```
Configuration.setPaymentFormChannelText(R.string.change_channel);  
Configuration.setPaymentFormChannelTextColor(R.color.green);  
Configuration.setPaymentFormChannelTextGravity(Gravity.RIGHT);  
Configuration.setPaymentFormChannelTextSize(20);  
Configuration.setPaymentFormChannelTextAllCaps(true);  
Configuration.setPaymentInfoBackgroundColor(R.color.dpsdk_green);
```

Możliwa jest także zmiana domyślnych kolorów statusów. Aby zmienić kolor statusu należy wykorzystać metodę setStatusColor:

```
Configuration.setStatusColor(StateType.COMPLETED, R.color.green);  
Configuration.setStatusColor(StateType.NEW, R.color.gray);  
Configuration.setStatusColor(StateType.PROCESSING, R.color.gray);  
Configuration.setStatusColor(StateType.REJECTED, R.color.red);  
Configuration.setStatusColor(StateType.PROCESSING_REALISATION, R.color.gray);  
Configuration.setStatusColor(StateType.PROCESSING_REALISATION_WAITING, R.color.gray);
```

Własna kontrolka wyboru kanału

Biblioteka umożliwia zastąpienie kontrolki wyboru kanału własną, bardziej dopasowaną do potrzeb Sklepu. Aby skorzystać z tej możliwości, należy:

Strona | 15 / 26

1. W `PaymentManager` zarejestrować zaimplementowaną własną aktywność wyboru kanału (w przykładzie jest to `CustomChannelList`), z podaniem bieżącego kontekstu:

```
PaymentManager.getInstance().registerCustomChannelComponent((MenuActivity.this, CustomChannelList.class);
```

2. Zainicjalizować płatność w `PaymentManager` (zgodnie z rozdziałem [Inicjalizacja płatności](#), powyżej)
3. Pobrać listę kanałów za pomocą jednej z metod:
4. `PaymentManager.getInstance().getChannels()` – zwraca wszystkie kanały
`PaymentManager.getInstance().getChannels(isOnline)` – zwraca kanały, które są obecnie online/offline
`PaymentManager.getInstance().getChannels(ids)` – zwraca kanały wg określonych ID
`PaymentManager.getInstance().getChannels(paymentTypes)` – zwraca kanały wg określonych typów
5. Ustawić zwróconą listę, jako źródła danych dla własnej kontrolki wyboru kanału
6. Po wyborze kanału przez Użytkownika wywołać odpowiednią metodę `PaymentManager`a:

```
PaymentManager.getInstance().initialPaymentForm(context, selectedChannel)
```

Własna kontrolka podsumowania transakcji

Biblioteka umożliwia zastąpienie kontrolki podsumowania płatności własną, bardziej dopasowaną do potrzeb Sklepu. Aby skorzystać z tej możliwości, należy:

1. Wywołać metodę `setPaymentResultEnabled(boolean paymentResultEnabled)` singletona `Configuration` przekazując wartość `false` jako argument. Powoduje ona wyłączenie ekranu podsumowania dostępnego w SDK.
2. Zakończenie procesu płatności sygnalizowane jest wywołaniem metody `onPaymentSuccess` interfejsu `PaymentManagerCallback` z przekazanym obiektem `PaymentResult`.
3. W celu odpytania serwera o status, należy wykorzystać metodę

```
getTransactionStatus(String id, String token, String number, String language)  
singletona PaymentManager, gdzie argumenty można pobrać z otrzymanego obiektu
```

```
paymentEndedEventArgs.getResult();
```

Przykład sprawdzenia statusu:

Strona | 16 / 26

```
@Override

public void onPaymentSuccess(PaymentEndedEventArgs paymentEndedEventArgs) {
    final PaymentResult paymentResult = paymentEndedEventArgs.getResult();
    new Thread(new Runnable() {
        @Override
        public void run() {
            try{

PaymentResultresult=PaymentManager.getInstance().getTransactionStatus(paymentResult.ge
tRecipientId(),paymentResult.getToken(),
paymentResult.getNumber(),paymentResult.getPaymentLanguage());

                //własny kod...

            }catch(OperationException e){

                //własny kod...

            }catch(NoConnectionException e){

                //własny kod...

            }
        }
    }).start();
}
```


Obsługa kanałów specjalnych

W rozdziale tym opisane zostaną dodatkowe funkcje związane ze specjalnymi kanałami płatniczymi.

Strona | 17 / 26

Płatność kartą - 1Click

Funkcjonalność 1Click pozwala na szybkie przeprowadzenie płatności zapamiętaną w systemie kartą płatniczą/kredytową. Podstawowe dane karty pamiętane są po stronie systemu płatniczego Dotpay.

Usługa ta, o ile dozwolona dla Sklepu po stronie systemu Dotpay, jest domyślnie w SDK włączona. Na skorzystanie z tej funkcjonalności musi także wyrazić zgodę Użytkownik (w trakcie wypełniania formularza płatności).

W celu wyłączenia funkcjonalności należy wywołać następującą metodę:

```
PaymentManager.getInstance().setOneClickEnabled(false);
```

UWAGA

Po wyłączeniu powyższych opcji nie są usuwane dane zapamiętanej wcześniej karty. Aby je usunąć należy wywołać metody opisane poniżej.

Inicjalizacja płatności 1Click

Proces płatności należy rozpocząć wykorzystując metodę `oneClickPayment` `PaymentManager`. Argumenty wymagane zostały opisane w rozdziale [Inicjalizacja płatności](#).

Dla potrzeb dewelopera w metodzie zostały przygotowane specjalne wyjątki, które należy obsługiwać. Lista wyjątków znajduje się poniżej w tabeli:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
<code>OneClickUnableException</code>	Wyjątek informuje o wyłączonej usłudze 1Click
<code>NotFoundPaymentCardException</code>	Wyjątek informuje o braku zarejestrowanych kart
<code>NotFoundDefaultPaymentCardException</code>	Wyjątek informuje o braku ustawionej domyślnej karty Sposób ustawienia domyślnej karty został opisany w osobnym podrozdziale.

Przykład inicjalizacji płatności:

```
String description = "zamówienie 12345";
double amount = 123.45;
PaymentInformation paymentInformation = new PaymentInformation(merchant_Id, amount,
description, selectedCurrency);

Map<String, String> sender = new Map<String, String> {"firstname", "Jan"},
{"lastname", "Kowalski"}, {"email", "jan.kowalski@test.pl"}
Map<String, String> additional = new Map<String, String> {"id1", "12345"},
{"amount1", "100"}, {"id2", "67890"}, {"amount2", "23.45"}

paymentInformation.setSenderInformation(sender);
paymentInformation.setAdditionalInformation(additional);
try {
    PaymentManager.getInstance().oneClickPayment(this, paymentInformation);
} catch (NotFoundPaymentCardException e) {

    // własny kod ...

} catch (NotFoundDefaultPaymentCardException e) {

    // własny kod ...

} catch (OneClickUnableException e) {

    // własny kod ...

}
}
```

Wykorzystanie wbudowanej kontrolki Menadżer kart

Menadżer wyświetla listę zarejestrowanych kart oraz posiada przycisk, który umożliwia dodanie nowej karty.

UWAGA: jeżeli usługa jest wyłączona, przycisk nie jest wyświetlany.

Aby skorzystać z wbudowanej kontrolki „Menadżer kart” należy:

1. W xml odpowiedniego layout, przygotować kontrolkę `FrameLayout`.
2. Następnie należy stworzyć instancję `PaymentCardManagerFragment` przez statyczną metodę `newInstance` którą należy przekazać do `FragmentManager`-a. Metoda `newInstance` przyjmuje argumenty zgodne z poniższą tabelą:

PARAMETR	ZNACZENIE / OPIS
merchant_id	Typ: string

	ID konta w systemie Dotpay, na rzecz którego dokonywana jest płatność (ID konta Sprzedawcy)
currency	<u>Typ</u> : string waluta w której jest wykonywana płatność Sposób pobrania listy dostępnych walut został opisany w podrozdziale Dostępne waluty .
language	<u>Typ</u> : string język w jakim jest wykonywana płatność

Przykład inicjalizacji „Menedżera kart”:

```
private void initPaymentCardManagerFragment() {  
    FragmentManager fm = getSupportFragmentManager();  
    FragmentTransaction ft = fm.beginTransaction();  
    Fragment fragment = PaymentCardManagerFragment.newInstance(merchant_id,  
currency, language);  
    ft.replace(R.id.fragment_container, fragment, null).commit();  
}
```

Regulaminy przy dodaniu karty

Aby zarejestrować nową kartę użytkownik podczas dodawania musi zaakceptować dwa regulaminy. Dostępne są dwie metody, które umożliwiają zmianę nazwy sklepu oraz przekierowanie do regulaminu sklepu wystawionego pod wskazanym adresem URL. Aby to wykonać należy wywołać odpowiednie settery singletona `Configuration`.

Dostępne metody:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
MerchantName	<u>Typ</u> : String Definiuje nazwę sklepu
MerchantPolicyUrl	<u>Typ</u> : String Przekierowuje klienta do regulaminu sklepu umieszczonego pod wskazanym URL

Zmiana stylu prezentacji

Aby zmienić sposób prezentacji elementów kontrolki „Menedżer kart” należy wywołać odpowiednie settery singletona `Configuration`. Ustawienia należy wykonać przed zainicjowaniem layoutu zawierającego kontrolkę „Menedżer kart”.

Kontrolka listy kart:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
PaymentCardManagerBackgroundColor	<u>Typ:</u> int Definiuje kolor tła całego widoku
PaymentCardManagerBackgroundItemColor	<u>Typ:</u> int Definiuje kolor tła pojedynczego kafelka
PaymentCardManagerBackgroundPressItemColor	<u>Typ:</u> int Definiuje kolor tła kafelka po kliknięciu
PaymentCardManagerTextStyle	<u>Typ:</u> int Definiuje styl tekstu na kafelku
PaymentCardManagerDefaultMarkColor	<u>Typ:</u> int Definiuje kolor ikony zaznaczenia (działa od API 17+)

Kontrolka formularza dodania karty:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
PaymentCardManagerFormBackgroundColor	<u>Typ:</u> int Definiuje kolor tła całego widoku
PaymentCardManagerFormLabelStyle	<u>Typ:</u> int Definiuje styl tekstu etykiet w kontrolce

Przykład:

```
Configuration.setPaymentCardManagerBackgroundColor(R.color.gray);  
Configuration.setPaymentCardManagerTextStyle(R.style.CardManagerTextStyle);  
Configuration.setPaymentCardManagerDefaultMarkColor(R.color.red);  
Configuration.setPaymentCardManagerFormBackgroundColor(R.color.gray);  
Configuration.setPaymentCardManagerFormLabelStyle(R.style.CardManagerLabelStyle);
```

Własna kontrolka „Menedżer Kart”

Aby lepiej dopasować prezentowane dane do specyfiki sklepu, można stworzyć własną kontrolkę „Menedżer Kart”, pobierając z biblioteki jedynie dane.

1. Lista zarejestrowanych kart

W celu pobrania informacji o liście zapisanych kart należy wywołać odpowiednią metodę:

`PaymentManager.getInstance().getPaymentCardList()` - metoda zwraca listę obiektów `PaymentCardInfo`, które reprezentują kartę płatniczą.

Strona | 21 / 26

2. Rejestracja karty

Aby zarejestrować nową kartę należy użyć metody:

`PaymentManager.getInstance().registerPaymentCard()` która przyjmuje argumenty zgodne z poniższą tabelą:

PARAMETR	ZNACZENIE / OPIS
context	Typ: Context Kontekst, z którego ta metoda została wywołana
merchantId	Typ: string ID konta w systemie Dotpay (ID konta Sprzedawcy)
email	Typ: string Email użytkownika karty potrzeby do rejestracji
paymentCardData	Typ: PaymentCardData Obiekt zawierający informacje o karcie płatniczej
cardRegisteredCallback	Typ: CardRegisteredCallback Callback nasłuchujący sygnalizowanie zdarzeń końca rejestracji. Poprawnie zarejestrowana karta zwraca obiekt <code>PaymentCardInfo</code>

3. Wyrejestrowanie karty

W celu wyrejestrowania karty należy wywołać metodę:

`PaymentManager.getInstance().unregisterCardData(paymentCardId)` – jego argument jest dostępny w obiekcie `PaymentCardInfo.getPaymentCardId()`. Lista wyjątków dla metody znajduje się poniżej w tabeli:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
<code>PaymentOperationException</code>	Wyjątek przesłany ze strony serwerowej, zawierający opis zdarzenia.

NoConnectionException

Wyjątek oznacza problemy z komunikacją sieciową.

4. Ustawienie karty domyślnej

Aby dokonać płatności metodą 1Click należy ustawić kartę domyślną za pomocą metody:

`PaymentManager.getInstance().setDefaultPaymentCard(paymentCardId)` - jego argument jest dostępny w obiekcie `PaymentCardInfo.getPaymentCardId()`.

Aby sprawdzić jaka karta aktualnie jest ustawiona jako domyślna karta, należy wywołać metodę:

`PaymentManager.getInstance().getDefaultPaymentCard()` - która zwraca obiekt `PaymentCardInfo`.

Lista wyjątków dla metody znajduje się poniżej w tabeli:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
NotFoundDefaultPaymentCardException	Wyjątek oznacza brak zapisanej karty domyślnej.

Przykład wywołania powyższych operacji:

```
PaymentManager.getInstance().registerPaymentCard(ShopActivity.this, merchantId, email,
paymentCardData, new CardRegisteredCallback() {
    @Override
    public void onSuccess(final PaymentCardInfo paymentCardInfo) {

PaymentManager.getInstance().setDefaultPaymentCard(paymentCardInfo.getCredit_card_id()
);
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {

PaymentManager.getInstance().unregisterCardData(paymentCardInfo.getCredit_card_id());
                } catch (PaymentOperationException e) {
                    // własny kod ...
                } catch (NoConnectionException e) {
                    // własny kod ...
                }
            }
        }).start();
    }
    @Override
    public void onFailure(ErrorCode errorCode) {
        // własny kod ...
    }
});
```

Historia i status transakcji

Dodatkową funkcją biblioteki jest zapamiętywanie, i umożliwienie zaprezentowania historii płatności wykonywanych za pomocą SDK. W historii tej widoczne są także transakcje powiązane, np. później wykonane zwroty. Dostępne są także dodatkowe operacje na danych z historii.

Wykorzystanie wbudowanej kontrolki

Aby skorzystać z wbudowanej kontrolki historii, należy:

1. Kontrolkę historii płatności umieścić w pliku xml wybranego layoutu, który będzie odpowiedzialny za wyświetlanie historii, np.:

```
<fragment
```

```
    android:name="pl.mobiltek.paymentsmobile.dotpay.fragment.TransactionHistoryFragment"
    android:id="@+id/transactionHistory"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Zmiana stylu prezentacji

Aby zmienić sposób prezentacji elementów kontrolki historii, należy wywołać odpowiednie settery singletona `Configuration`. Ustawienia należy wykonać przed zainicjowaniem layoutu zawierającego kontrolkę historii.

Kontrolka wyboru kanałów:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
HistoryTitleVisibility	<u>Typ:</u> boolean Pozwala ukryć nagłówek
HistoryTitleText	<u>Typ:</u> int Definiuje tekst wyświetlany w nagłówku
HistoryTitleStyle	<u>Typ:</u> int Definiuje styl TextView z tytułem
HistoryDividerColor	<u>Typ:</u> int Definiuje kolor separatora pod tytułem

HistoryBackgroundColor	<u>Typ:</u> int Definiuje kolor tła całego widoku
setHistoryDateTextStyle	<u>Typ:</u> int Definiuje styl TextView dla elementu data
setHistoryAmountTextStyle	<u>Typ:</u> int Definiuje styl TextView dla elementu kwota
HistoryDescriptionTextStyle	<u>Typ:</u> int Definiuje styl TextView dla elementu opis
HistoryBackgroundItemColor	<u>Typ:</u> int Definiuje kolor tła całego elementu
HistoryDetailsHeaderTitleTextStyle	<u>Typ:</u> int Definiuje styl TextView dla nagłówka w oknie szczegółów
HistoryDetailDividerColor	<u>Typ:</u> int Definiuje kolor tła separatora nagłówka w oknie szczegółów
HistoryDetailsTitleTextStyle	<u>Typ:</u> int Definiuje styl TextView podtytułów szczegółów
HistoryDetailsValueTextStyle	<u>Typ:</u> int Definiuje styl TextView wartość szczegółów

Przykład:

Strona | 26 / 26

```
Configuration.setHistoryTitleVisibility(true);
Configuration.setHistoryTitleText(R.string.HistoryTitle);
Configuration.setHistoryTitleStyle(R.style.HistoryTitleStyle);
Configuration.setHistoryDividerColor(R.color.black);
Configuration.setHistoryBackgroundColor(R.color.gray);
Configuration.setHistoryDateTextStyle(R.style.HistoryDateStyle);
Configuration.setHistoryAmountTextStyle(R.style.HistoryAmountStyle);
Configuration.setHistoryDescriptionTextStyle(R.style.HistoryDescriptionStyle);
Configuration.setHistoryBackgroundItemColor(R.color.white);
Configuration.setHistoryDetailsHeaderTitleTextStyle(R.style.HistoryTitleStyle);
Configuration.setHistoryDetailDividerColor(R.color.black);
Configuration.setHistoryDetailsTitleTextStyle(R.style.HistoryDetailsTitleStyle);
Configuration.setHistoryDetailsValueTextStyle(R.style.HistoryDetailsValueStyle);
```

Własna kontrolka historii

Aby lepiej dopasować prezentowane dane do specyfiki Sklepu, można stworzyć własną kontrolkę historii, pobierając z biblioteki jedynie dane.

W celu pobrania informacji o liście transakcji należy wywołać odpowiednią metodę PaymentManagera:

```
PaymentManager.getInstance().loadTransactions();
```

Dodatkowo, dla transakcji zaprezentowanych we własnej historii można:

1. Usunąć pojedynczą pozycję:

```
PaymentManager.getInstance().deleteTransaction(paymentResult);
```

2. Sprawdzanie aktualnego statusu płatności dla pozycji z historii:

```
PaymentManager.getInstance().getTransactionStatus(paymentResult);
```

Lista wyjątków dla metody znajduje się poniżej w tabeli:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
PaymentOperationException	Wyjątek przesłany ze strony serwerowej, zawierający opis zdarzenia.
NoConnectionException	Wyjątek oznacza problemy z komunikacją sieciową.