Dział Pomocy Technicznej Dotpay
ul. Wielicka 28B, 30-552 Kraków
tel. +48 61 60 061 76
e-mail: tech@dotpay.pl

dotpay®

Płatności elektroniczne klasy e-biznes

SDK (Android)

Wersja 1.4.x

SPIS TREŚCI

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. Ze względu na ograniczenia systemu Android związane z obsługiwanymi wersjami algorytmów kryptograficznych (TLS 1.2), w pełni prawidłowa praca na domyślnie skonfigurowanym urządzeniu możliwa jest od wersji 4.4.4 (API: 20). Przy próbie płatności przy braku wsparcia dla TLS 1.2 płatność nie powiedzie się, sterowanie zostanie zwrócone do aplikacji wraz z informacją o niepowodzeniu procesu.

W dokumencie zastosowano 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 SDK, należy:

1. Dodać odpowiednie sekcje w plik konfiguracji projektu gradle:

```
// Project build.gradle
allprojects {
    repositories {
        maven { url 'https://github.com/dotpay/Mobile-SDK-Android/raw/master/'
}
    }
    flatDir {
        dirs 'libs'
    }
}
// Module build.gradle
dependencies {

    // needed to compile Visa Checkout .aar, see paragraph below
    implementation fileTree(include: ['*.jar', '*.aar'], dir: 'libs')

    // Standard Dotpay SDK version
    implementation('pl.mobiltek.paymentsmobile:dotpay:1.4.18@aar') {
        transitive = true
    }
}
```

2. W pliku AndroidManifest.xml dodać:

```
<application
        ...
        tools:replace="android:allowBackup"
        ...
```

3. Dołączyć ręcznie bibliotekę Visa Checkout, gdyż nie jest ona niestety dystrybuowana w formie zależności gradle. Właściwa wersja biblioteki dla SDK 1.4.18 to 6.6.1, jest do pobrania z podkatalogu `visa_checkout_sdk` z repozytorium SDK Dotpay. Bibliotekę należy dołączyć do katalogi `libs` projektu.

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 PaymentManagera:

```
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 Użytkowania kanału:

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

Zablokowanie możliwości robienia screenów:

```
PaymentManager.getInstance().enableSecureFlagOnWindows(true)
```

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 płacącego, 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 `PaymentManagera`, 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	Typ: string ID konta w systemie Dotpay, na rzecz którego dokonywana jest płatność (ID konta Sprzedawcy)
amount	Typ: double Kwota transakcji
description	Typ: string Tytuł/opis płatności
currency	Typ: string Domyślna wartość: "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	Typ: Map<string, string> Domyślna wartość: null Nazwa settera: `setSenderInformation(Map<String, String> senderInformation)` Dodatkowe informacje o kupującym. Mapa z kluczami: "firstname" – imię; "lastname" – nazwisko; "email" – email; "phone" – telefon; „phone_verified" - informacja o zweryfikowaniu numeru telefonu; "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	Typ: Map<string, string> Domyślna wartość: null

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

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);
```

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

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.getPaymentResult().getStateType() == StateType.COMPLETED) {

        // płatność zakończona pomyślnie

}else if(paymentEndedEventArgs.getPaymentResult().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.

Globalne ustawienia:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
ToolBarBackgroundColor	Typ: int Definiuje kolor tła toolbara
ToolBarTitleTextColor	Typ: int Definiuje kolor tekstu na toolbarze
ButtonTitleTextColor	Typ: int Definiuje kolor tekstu na przycisku
ButtonBackgroundColorResource	Typ: 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	Typ: int Definiuje kolor tła całego widoku
ChannelBackgroundItemColor	Typ: int Definiuje kolor tła pojedynczego kafelka
ChannelBackgroundPressItemColor	Typ: int Definiuje kolor tła kafelka po kliknięciu
ChannelItemTextColor	Typ: int Definiuje kolor tekstu na kafelku
ChannelTextGravity	Typ: 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);
```

Kontorolka ulubionych kanałów płatności:

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

Przykład:

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

Kontrolka zmiany metody płatności w formularzu płatności:

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

PaymentInfoBackgroundColor	Typ: int 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:

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:

   ```
   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
   ```

4. Ustawić zwróconą listę, jako źródła danych dla własnej kontrolki wyboru kanału

5. Po wyborze kanału przez Użytkownika wywołać odpowiednią metodę PaymentManagera:

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

Filtrowanie kanałów

Jeśli w aplikacji realizujemy sprzedaż różnego rodzaju towarów/usług, gdzie każda z nich charakteryzuje się innym oczekiwanym zestawem kanałów płatniczych (np. jedna z nich wymaga szybkiego czasu finalizacji płatności), aby nie było potrzeby samodzielnego tworzenia customowej kontrolki wyboru kanałów, istnieje możliwość utworzenia prostego callbacka, za pomocą którego ofdiltrujemy płatności w danym kontekście.

Aby to zrobić, należy zaimplementować interfejs ChannelFilterCallback:

```
public interface ChannelFilterCallback {
      List<Channel> filter(List<Channel> channels);
    }
```

I zarejestrować go w PaymentManagerze:

```
PaymentManager.getInstance().setChannelFilterCallback(channelFilterCallback);
```

Przykład callbacka filtrującego kanały wg grup:

```
ChannelFilterCallback channelFilterCallback = new ChannelFilterCallback() {
@Override
public List<Channel> filter(List<Channel> channels) {
    List<Channel> filteredChannels = new LinkedList<>();
        for (Channel channel : channels) {
        if (channelGroups.contains(channel.group)) {
        filteredChannels.add(channel);
        }
        }
        return filteredChannels;
        }

        };
```

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:


```
@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.

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 obsłużyć. Lista wyjątków znajduje się poniższej w tabeli:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
OneClickUnableException	Wyjątek informuje o wyłączonej usłudze 1Click
NotFoundPaymentCardException	Wyjątek informuje o braku zarejestrowanych kart
NotFoundDefaultPaymentCardException	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 podsiada przycisk, który umożliwia dodanie nowej karty.

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

Aby skorzystać z wbudowanej kontrolki „Menedżer 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	Typ: string waluta w której jest wykonywana płatność Sposób pobrania listy dostępnych walut został opisany w podrozdziale *Dostępne waluty*.
language	Typ: string język w jakim jest wykonywana płatność
firstname	Typ: string Imię posiadacza karty
lastname	Typ: string Nazwisko posiadacza karty
email	Typ: string Email posiadacza karty

Przykład inicjalizacji „Menedżera kart”:

```
private void initPaymentCardManagerFragment() {

        FragmentManager fm = getSupportFragmentManager();
        FragmentTransaction ft = fm.beginTransaction();
        Fragment fragment = PaymentCardManagerFragment.newInstance(merchant_id,
currency, language, firstName, lastName, email);
        ft.replace(R.id.fragment_container, fragment, null).commit();

        }
```

Jeśli nie znamy danych posiadacza karty, można skorzystać z przeciążonej metody, która nie wymaga tych danych.

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	Typ: String Definiuje nazwę sklepu

MerchantPolicyUrl	Typ: 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	Typ: int Definiuje kolor tła całego widoku
PaymentCardManagerBackgroundItemColor	Typ: int Definiuje kolor tła pojedynczego kafelka
PaymentCardManagerBackgroundPressItemColor	Typ: int Definiuje kolor tła kafelka po kliknięciu
PaymentCardManagerTextStyle	Typ: int Definiuje styl tekstu na kafelku
PaymentCardManagerDefaultMarkColor	Typ: int Definiuje kolor ikony zaznaczenia (działa od API 17+)

Kontrolka formularza dodania karty:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
PaymentCardManagerFormBackgroundColor	Typ: int Definiuje kolor tła całego widoku
PaymentCardManagerFormLabelStyle	Typ: 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.

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 *`PaymentCardInfo`*

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 poniższej w tabeli:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
PaymentOperationException	Wyjątek przesłany ze strony serwerowej, zwierający 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 objekt `PaymentCardInfo`.

Lista wyjątków dla metody znajduje się poniższej 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 ...
            }
        });
```

Dane karty dostarczane z zewnątrz

Domyślnie referencje do kart zapamiętywane są wewnątrz SDK, stąd funkcjonalność jest dostępna bez żadnych dodatkowych prac koniecznych czy to po stronie aplikacji mobilnej, czy też backendu, z którym aplikacja może współpracować. Efektem ubocznym takiego podejścia jest brak możliwości przenoszenia zarejestrowanych kart pomiędzy urządzeniami, czy też po reinstalacji aplikacji.

Jeśli jednak aplikacja współpracuje z własnym backendem, jednoznacznie identyfikując Użytkownika (jest to warunek konieczny), referencje do kart można przechowywać w backendzie, co pozwoli współdzielić je (np. jeśli konto użytkownika jest współdzielone pomiędzy serwis WWW oraz aplikację mobilną).

Szczegółowy opis mechanizmów zapamiętywania kart znajduje się w Instrukcji technicznej implementacji płatności Dotpay.

Aby włączyć obsługę tak zapamiętywanych kart, należy przed inicjalizacją płatności dodać do `PaymentManager`-a listę kart wraz ze wskazaniem identyfikatora `credit_card_customer_id`:

```
setExternalCreditCards(List<PaymentCardInfo> creditCards, String creditCardCustomerId)
```

gdzie `creditCards` jest listą obiektów `PaymentCardInfo` zawierającą komplet informacji o karcie, a `creditCardCustomerId` jest unikalnym identyfikatorem użytkownika, w kontekście którego rejestrowane były karty. Dodatkowo, istnieje przeciążona metoda o analogicznej nazwie, przyjmująca dodatkowy parametr pozwalający wyłączyć widoczność kart zapamiętanych wewnątrz SDK (jeśli obecnie z takich kart aplikacja korzysta, bądź też jeśli musi pracować zarówno w kontekście bez zalogowanego Użytkownika/z zalogowanym Użytkownikiem)

Obiekt `PaymentCardInfo` ma następujące właściwości:

WŁAŚCIWOŚĆ	ZNACZENIE / OPIS
maskedNumber	Typ: string Zamaskowany numer karty
cardId	Typ: string Identyfikator karty (znaczenie zgodnie z dokumentacją podstawową Dotpay)
name	Typ: string Nazwa karty prezentowana Użytkownikowi
codeName	Typ: string Typ karty (znaczenie zgodne z dokumentacją podstawową Dotpay)
logo	Typ: string Link do logotypu karty (zgodnie z dokumentacją podstawową Dotpay, link jest zwracany razem z danymi zarejestrowanej karty)

Przykład włączenia kart z zewnątrz:

```
PaymentCardInfo externalCard = new PaymentCardInfo(maskedNumber, cardId, name,
codeName, logo);
List<PaymentCardInfo> externalCreditCards = new ArrayList();
externalCreditCards.add(externalCard);
PaymentManager.getInstance().setExternalCreditCards(externalCreditCards,
creditCardCustomerId);
```

UWAGA: metoda ustawiająca karty z zewnątrz nadpisuje wszystkie karty dodane wcześniej oznaczone jako zewnętrzne, można ją wołać wielokrotnie, aby aktualizować listę kart.

Jeśli chcemy zdjąć kontekst użytkownika, usunąć z pamięci SDK dodane karty tak, aby ponownie działać jedynie w kontekście kart pamiętanych w SDK, należy wywołać metodę `PaymentManager`-a `disableExternalCards()`

```
PaymentManager.getInstance().disableExternalCards()
```

1. Rejestracja kart w kontekście zewnętrznego Użytkownika

Inicjalizując SDK danymi kart z zewnątrz, należy zwrócić uwagę, iż wszystkie karty rejestrowane z poziomu SDK będą rejestrowane z podanym `credit_card_customer_id` (zarówno te przez Managera Kart, jak i te rejestrowane podczas płatności).

Jeśli Użytkownik nie posiada jeszcze żadnej zarejestrowanej karty, ale chcemy, aby nowe karty rejestrowane były w jego kontekście, metodę `setExternalCreditCards` należy zawołać z podaniem pustej listy kart.

Wszystkie tak rejestrowane karty są automatycznie dodane do SDK, jak karty dostarczone z zewnątrz (w kontekście wskazanego credit_card_customer_id). Informację o fakcie zarejestrowania nowej karty można uzyskać na poziomie backendu. Gdy tylko aplikacja uzyska tę informację, należy ponownie ustawić na `PaymentManager` aktualną listę kart zewnętrznych.

W celu uzyskania informacji o samym fakcie rejestracji, należy zaimplementować interfejs:

```
public interface ExternalCardRegisteredCallback {
    void onCardRegistered(PaymentCardInfo creditCard);
}
```

oraz zarejestrować go w `PaymentManager`:

```
PaymentManager.getInstance().setExternalCardRegisteredCallback
    (externalCardRegisteredCallback);
```

Gdy tylko nowa karta zostanie zarejestrowana, aplikacja zostanie o tym poinformowana, może więc zwrócić się do backendu w celu odświeżenia listy kart.

2. Zmiany kart domyślnych

Karty dostarczane z zewnątrz, tak samo jak karty pamiętane w SDK, mogą być oznaczane jako „domyślne" na potrzeby funkcjonalności 1Click (zgodnie z rozdziałem Ustawienie karty domyślnej). Należy zwrócić uwagę, iż karta domyślna jest innym pojęciem, niż karta ostatnio użyta przy płatności standardowej.

Aby uzyskać informację, iż karta domyślna została na SDK zmieniona, należy zaimplementować intefejs:

```
public interface DefaultCardChangedCallback {
    void onDefaultCardChanged(PaymentCardInfo newDefaultCard);
}
```

oraz zarejestrować go w `PaymentManager`:

```
PaymentManager.getInstance().setDefaultCardChangedCallback(defaultCardChangedCallback
).
```

Gdy zmieniona zostanie karta domyślna w SDK, aplikacja zostanie o tym poinformowana, możliwe będzie więc przekazanie tej informacji do backendu.

Maskowanie pola z CVV

Jeśli SDK jest wykorzystywane w aplikacji, która często jest wykorzystywana przez Użytkowników w warunkach ograniczających możliwość zachowania odpowiedniego poziomu poufności wprowadzanych danych (np. zakup biletu komunikacji miejskiej już po wejściu do zatłoczonego pojazdu), można skorzystać z opcji maskowania pola na CVV tak, aby dać Użytkownikowi większe poczucie bezpieczeństwa.

Aby to zrobić, należy wywołać metodę:

```
PaymentManager.getInstance().setCvvConcealed(true);
```

Należy jednak zwrócić uwagę, iż wartość ta jest tylko jednym z czynników zapewniających bezpieczeństwo płatności kartą, stąd nie ma zalecenia, aby pole to było tak zabezpieczone w każdej aplikacji, pamiętając, iż błędne wprowadzenie tego pola prowadzi do nieudanej płatności, a wielokrotna nieudana płatność może prowadzić do zablokowania karty do płatności internetowych.

Płatność z pominięciem formularza płatniczego.

Na wypadek konieczności wyjęcia specjalnego kanału płatniczego do koszyka aplikacji mobilnej, powstał mechanizm umożliwiający wywołanie płatności ze wskazaniem kanału, a przez to pominięciem jego wyboru, oraz pominięciem formularza płatniczego.

Aby wykonać taką płatność, należy wykorzystać metodę `payment` na `PaymentManager`, której parametry opisane zostały w rozdziale Inicjalizacja płatności. Dodatkowym parametrem jest identyfikator kanału.

Parametry dodatkowe, specyficzne dla kanału, należy przekazać jako elementy słownika `additionalInformation` w obiekcie klasy `PaymentInformation`.

Masterpass Champion Wallet

Portfel Masterpass Champion Wallet umożliwia szybkie i wygodne skorzystanie z funkcjonalności płatności Masterpass, gdy w trakcie płatności takiego portfela nie posiadamy. Rejestracja do Masterpass Champion Walleta jest wygodna, do przeprowadzenia całkowicie online oraz umożliwia płatność typu „1click". Po jednokrotnym zarejestrowaniu/zalogowaniu do portfela, kolejne płatności prowadzone są z podaniem minimalnej liczby informacji.

Aby skorzystać z funkcjonalności, po uzgodnieniach biznesowych, wykonywana jest odpowiednia konfiguracja konta Sklepu, co automatcznie daje dostęp do tej wersji portfela.

Uwaga: aby skorzystać z tej funkcjonalności, aplikacje mobilne wykorzystujące SDK muszą mieć dedykowane konto Sklepu dotpay.

Dodatkowe parametry inicjalizacji płatności

Aby usprawnić proces wykorzystania portfela przez Płacącego, zalecane jest przekazanie parametru `phone` we właściwości `senderInformation` obiektu `PaymentInformation`, z pomocą którego inicjowana jest instancja SDK.

Dodatkowo, jeśli przekazany zostanie parametr `phone_verified` o wartości „true", którego przekazaniem aplikacja mobilna jednoznacznie potwierdza, iż numer telefonu przekazany do SDK jest w pełni zweryfikowany (np. kodem jednorazowym SMS), uproszczony zostanie proces rejestracji.

Autoryzacja płatności

Portfel Masterpass Champion Wallet został stworzony po to, aby płatności były możliwie proste. Aby ograniczyć ryzyko fraudów, zalecane jest wykonanie autoryzacji płatności po stronie aplikacji. Aby to zrobić, należy zaimplementować interfejs:

```
public interface MasterpassPaymentCallback {
    void onPaymentShouldCreateAuthorization (Context context);
    void onPaymentShouldAuthorizeUser (Context context);
}
```

oraz zarejestrować go w `PaymentManager`:

```
PaymentManager.getInstance().setMasterpassPaymentCallback(masterpassPaymentCallback);
```

Zwrotnie na klasie `PaymentManager` wystawione są następujące metody:

```
public void authorizationCreateResult(boolean isAuthorizationCreated);

public void authorizeUserResult (boolean isUserAuthorize, Activity activity);
```

Metoda `onPaymentShouldCreateAuthorization` zostanie wywołana w trakcie logowania się (bądź rejestracji) użytkownika do portfela, i umożliwia zdefiniowanie/zainicjowanie mechanizmów autoryzacji w aplikacji mobilnej (np. zdefiniowanie PINu do aplikacji). Po zakończeniu tego procesu należy wywołać metodę `authorizationCreateResult` `PaymentManager`'a przekazując status tworzenia autoryzacji. W wypadku pozytywnego, SDK będzie kontynuowało proces rejestracji do portfela, w wypadku negatywnego przerwie ten proces.

Metoda `onPaymentShouldAuthorizeUser` zostanie wywołana każdorazowo, gdy zalogowany Płacący będzie chciał wykonać kolejną płatność powiązanym portfelem. W ramach jej przebiegu powinniśmy wykonać autoryzację (np. weryfikację PINu), a po zakończeniu wywołać metodę `authorizeUserResult` `PaymentManager`'a przekazując jej status. Potwierdzenie autoryzacji pozwoli wykonać płatność, informacja o nieprawidłowej autoryzacji przerwie ją.

Metoda zwrotna wymaga dodatkowego parametru, czyli bieżącej Activity, która umożliwi właściwe odniesienie się do kontekstu Android.

Przykładowy kod implementujący metody delegata, który tylko oddaje kontrolę zwrotnie, mógł by wyglądać tak:

```
        private MasterpassPaymentCallback masterpassPaymentCallback = new
MasterpassPaymentCallback() {

            @Override

            public void onPaymentShouldCreateAuthorization(Context context) {

PaymentManager.getInstance().authorizationCreateResult(true);

            }

            @Override

            public void onPaymentShouldAuthorizeUser(Context context) {

    //create activity if required
```

```
    ...
PaymentManager.getInstance().authorizeUserWithResult(true,CurrentActivity.this);

        };
```


Google Pay

Domyślna wersja biblioteki SDK ukrywa obsługę płatności Google Pay, nawet jeśli są one dostępne w konfiguracji sklepu i dostępne przy płatności webowej, gdyż bez spełnienia dodatkowych wymogów metoda ta nie jest w stanie funkcjonować poprawnie w aplikacji mobilnej.

Alternatywna wersja biblioteki SDK, zawiera w sobie natywną obsługę SDK Google Pay, aby z niej skorzystać, należy w pliku gradle zależność z SDK Dotpay podmienić na:

```
implementation('pl.mobiltek.paymentsmobile:dotpay-googlepay:1.4.18@aar') {
    transitive = true
}
```

Umożliwienie produkcyjnego wykorzystania tej biblioteki wymaga jednak zgłoszenia aplikacji do Google.

Szczegółowy opis całości uwarunkowań znajduje się w dokumentacji dostępnej pod linkiem:

https://www.dotpay.pl/developer/doc/google-pay/

W rozdziale 5 wspomnianej dokumentacji, znajduje się odniesienie do checklisty Google, którą należy przejść, aby poprosić o zgodę na przejście na produkcję.

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	Typ: boolean Pozwala ukryć nagłówek
HistoryTitleText	Typ: int Definiuje tekst wyświetlany w nagłówku
HistoryTitleStyle	Typ: int Definiuje styl TextView z tytułem
HistoryDividerColor	Typ: int Definiuje kolor separatora pod tytułem

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

Przykład:

```
Configuration.setHistoryTitleVisibility(true);
Configuration.setHistoryTitleText(R.string.HistoryTitle);
Configuration.setHistoryTitleStyle(R.style.HistoryTitleStyl);
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().loadTransactios();
```

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 poniższej w tabeli:

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