Dział Pomocy Technicznej Dotpay
ul. Wielicka 28B, 30-552 Kraków
tel. +48 12 688 26 00
e-mail: tech@dotpay.pl

Płatności elektroniczne klasy e-biznes

# SDK (iOS)

Wersja 1.4.x

## SPIS TREŚCI

## WSTĘP

Niniejszy dokument ma na celu opis sposobu wykorzystania biblioteki dla platformy iOS, 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 Objective-C. Wspierany jest system iOS w wersji 9.0 i wyższej.

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, 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. Aby dodać bibliotekę DotPaySDK należy użyć popularnego menedżera zależności CocoaPods, Gdy mamy już zainicjowany plik podfile, należy dopisać do niego:

   ```
   pod 'DotPaySDK', '~> 1.4.15'
   ```

2. W każdym miejscu, w którym będziemy korzystać z elementów SDK należy zaimportować plik nagłówkowy:

   ```
   #import <DotPaySDK/DotPaySDK.h>
   ```

3. Jeśli chcemy, aby prawidłowo były obsługiwane płatności z aplikacją PeoPay (dla Banku Pekao S.A.), w pliku plist aplikacji głównej należy dodać obsługę odpowiedniej query scheme:

   ```
   <key>LSApplicationQueriesSchemes</key>
   <array>
   <string>peopay</string>
   </array>
   ```

## 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 singletona DotPay:

   ```
   [[DotPay sharedInstance] getLanguageListWithCompletion:^(NSArray
   *languageList, NSError *error){}];
   ```

2. Dopasować najbardziej odpowiedni język

3. Ustawić wybrany język

   ```
   [DotPay sharedInstance].defaultLanguage = bestLanguage;
   ```

## Wybór systemu

Biblioteka ma możliwość komunikowania się zarówno z testowym, jak i produkcyjnym systemem Dotpay. Domyślnie biblioteka ustawiona jest na system produkcyjny.

Aby ustawić system testowy, należy wykonać następującą instrukcję:

```
[DotPay sharedInstance].debugMode = YES;
```

Analogicznie, w celu ustawienia system produkcyjnego należy wykonać:

```
[DotPay sharedInstance].debugMode = NO;
```

## Wersja SDK

Zalecamy wyświetlenie w aplikacji końcowej wersji SDK, co pozwoli ułatwić proces diagnozowania problemów. Wersję SDK można pobrać z odpowiedniej właściwości singletona DotPay:

```
[DotPay sharedInstance].sdkShortVersionString
```

## Usunięcie niepotrzebnych architektur z Dynamic Libraries w Xcode

SDK zostało przygotowane pod wszystkie architektury, aby można było korzystać z niego na urządzeniach i symulatorze. W przypadku budowania aplikacji z użyciem SDK do App Store niepotrzebne architektury muszą zostać usunięte z projektu. Aby usunąć architektury po kroku embed frameworks należy dodać „run script":

```
APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
    FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist"
CFBundleExecutable)
    FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
    echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"

    EXTRACTED_ARCHS=()

    for ARCH in $ARCHS
    do
        echo "Extracting $ARCH from $FRAMEWORK_EXECUTABLE_NAME"
        lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o
"$FRAMEWORK_EXECUTABLE_PATH-$ARCH"
        EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
    done

    echo "Merging extracted architectures: ${ARCHS}"
    lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
    rm "${EXTRACTED_ARCHS[@]}"

    echo "Replacing original executable with thinned version"
    rm "$FRAMEWORK_EXECUTABLE_PATH"
    mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"

done
```

## Płatność

Całość procesu płatności polega na wyświetleniu kontrolera `DPDotPayViewController`, oraz oczekiwaniu na wywołanie metody delegata zwracającego 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 kontrolerów płatności, oraz opcje dodatkowe, pozwalające na dostosowanie procesu do własnych celów.

### Przygotowanie delegata powrotu

Przygotowanie procesu płatności należy rozpocząć od przygotowania implementacji delegata `DPDotPayViewControllerDelegate`:

```
-(void) dotpayViewController: (DPDotPayViewController *)viewController
didFinishPaymentWithSummary: (DPPaymentSummary *)paymentSummary;

-(void) dotpayViewController: (DPDotPayViewController *)viewController
didFailToFinishPaymentWithError: (NSError *)error
```

### Inicjalizacja płatności

Aby rozpocząć płatność, należy przygotować jej opis. W tym celu należy utworzyć instancję klasy `DPPaymentInfo`, oraz wypełnić ją wymaganymi danymi, opisanymi w tabeli poniżej:

| PARAMETR | ZNACZENIE / OPIS |
|---|---|
| merchantID | Typ: NSString<br>ID konta w systemie Dotpay, na rzecz, którego dokonywana jest płatność (ID konta Sprzedawcy) |
| amount | Typ: NSDecimalNumber<br>Kwota transakcji |
| textDescription | Typ: NSString<br>Tytuł/opis płatności |

| | |
|---|---|
| currency | Typ: DPCurrency<br><br>Domyślna wartość: „PLN”<br><br>Określenie w jakiej walucie podany jest parametr amount. Sposób na pobranie listę dostępnych walut opisany został w podrozdziale *Dostępne waluty*, poniżej. |
| senderInformation | Typ: NSDictionary<br><br>Domyślna wartość: nil<br><br>Dodatkowe informacje o kupującym. Słownik z kluczami:<br><br>“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).<br><br>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.<br><br>Szczegółowe wyjaśnienie zawartości pól, ich znaczenie, znajduje się w *Instrukcji technicznej implementacji płatności Dotpay*. |
| additionalInformation | Typ: NSDictionary<br><br>Domyślna wartość: nil<br><br>Dodatkowe parametry przekazywane podczas procesu płatności, zgodnie z szczegółami przekazanymi w dodatkowych dokumentacjach. |
| control | Typ: NSString<br><br>Domyślna wartość: nil<br><br>Parametr identyfikujący daną płatność, przekazywany w potwierdzeniu płatności do Sklepu, potrzebny w celu dopasowania statusu płatności do właściwego zamówienia w Sklepie.<br><br>Więcej informacji znajduje się w *Instrukcji technicznej implementacji płatności Dotpay*.<br><br>Jeśli nie jest ustawiony, to jest generowany przez SDK.<br><br>UWAGA<br><br>W celu poprawnego działania historii płatności parametr ten powinien być unikalny dla każdej płatności. |
| urlc | Typ: string |

| | |
|---|---|
| | Domyślna wartość: nil<br><br>Adres URL Sklepu, do odbioru parametrów potwierdzających zrealizowanie lub odmowę realizacji transakcji.<br><br>Więcej informacji znajduje się w *Instrukcji technicznej implementacji płatności Dotpay*. |

Przykład:

```
DPPaymentInfo *info = [[DPPaymentInfo alloc] init];
info.currency = [DPCurrency currencyWithCode:@"PLN"];
info.amount = [NSDecimalNumber decimalNumberWithString:@"100.0"];
info.merchantID = @"10000";
info.textDescription = @"Some description";
info.senderInformation = @{@"firstname": @"Jan", @"surname": @"Kowalski",
@"email": @"jan.kowalski@test.pl"};
info.additionalInformation = @{@"id1": @"12345", @"amount1": @"40", @"id1":
@"67890", @"amount2": @"60"};
```

Kolejnym krokiem jest pobranie listy kanałów. Wykonuje się to za pomocą jednej z poniższych metod, podając na wejściu dodatkowo szczegóły płatności:

`[[DotPay sharedInstance] getChannelListForPaymentInfo:info withCompletion:^(NSArray *channelList, DPPaymentDetails *paymentDetails, NSError *error) { }]` – pobiera wszystkie kanały

`[[DotPay sharedInstance] getChannelListForPaymentInfo:info online:online withCompletion:^(NSArray *channelList, DPPaymentDetails *paymentDetails, NSError *error) { }]` – pobiera tylko kanały, które są obecnie online/offline

`[[DotPay sharedInstance] getChannelListForPaymentInfo:info withIds:ids withCompletion:^(NSArray *channelList, DPPaymentDetails *paymentDetails, NSError *error) { }]` – pobiera tylko wskazane kanały, wg identyfikatorów

`[[DotPay sharedInstance] getChannelListForPaymentInfo:info group:group withCompletion:^(NSArray *channelList, DPPaymentDetails *paymentDetails, NSError *error) { }]` - pobiera tylko kanały wskazanego typu

Na koniec asynchronicznie prezentujemy kontroler na głównym wątku, wykonując następujące operacje:

a. Zainicjalizowanie kontrolera płatności z podaniem listy kanałów oraz szczegółów płatności
b. Opcjonalnym wyłączeniem funkcji użycia ostatnio użytego kanału (domyślnie opcja ta jest włączona)
c. Ustawienie delegata zwrotnego
d. Wyświetlenie kontrolera

Przykład:

```
dispatch_async(dispatch_get_main_queue(), ^{

    DPDotPayViewController *dotPayViewController = [[DPDotPayViewController
alloc] initWithPaymentChannelList:channelList paymentDetails:paymentDetails];
    dotPayViewController.useLastChannelSelection = NO;
    dotPayViewController.paymentControllerDelegate = self;

    [self presentViewController:dotPayViewController animated:YES
completion:NULL];

        });
}];
```

## Zakończenie płatności

Poprawne zakończenie procesu płatności sygnalizowane jest odpowiedniej metody delegata.

W wypadku pozytywnego zakończenia procesu płatności, wywoływana jest metoda `dotpayViewController:didFinishPaymentWithStatus:`, której argumentem będzie obiekt klasy `DPPaymentSummary`, zawierający szczegóły wykonanej płatności.

Jeśli proces zakończy się błędem, wywołana będzie metoda `dotpayViewController:didFailToFinishPaymentWithError:`, której argumentem będzie opis błędu.

**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:

```
-(void) dotpayViewController: (DPDotPayViewController *)viewController
didFinishPaymentWithSummary: (DPPaymentSummary *)paymentSummary {
        if([paymentSummary.status isEqualToString: kDPPaymentSummaryStatusCompleted])
{
            // płatność zakończona pomyślnie
        } else if ([paymentSummary.status isEqualToString:
kDPPaymentSummaryStatusRejected]) {
            // płatność zakończona odmową
        } else {
            // płatność w trakcie realizacji
        }
    }
-(void) dotpayViewController: (DPDotPayViewController *)viewController
didFailToFinishPaymentWithError: (NSError *)error {
          // 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:

```
[[DotPay sharedInstance] setShowDetailsOnSummaryScreen:YES];
```

## Dostępne waluty

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

```
[[DotPay sharedInstance] getCurrencyListWithCompletion:^(NSArray *currencyList,
NSError *error){}];
```

## Zmiana stylu prezentacji

Zmiana sposób prezentacji elementów kontrolek procesu płatności wykonywana jest z wykorzystaniem frameworka `UIAppearance`.

Korzystając z metody `appearanceWhenContainedIn` kontrolek systemowych można ograniczać zmiany w prezentacji jedynie do poszczególnych kontrolerów płatności:

- `DPPaymentChannelsViewController` – kontroler wyboru kanałów
- `DPPaymentFormViewController` – kontroler formularza. Jako iż ten kontroler nie jest dostępny dla Developera, do jego klasy należy się odnieść przez `NSClassFromString`

  `NSClassFromString(@”DPPaymentFormViewController”)`
- `DPPaymentSummaryViewController` – kontroler strony statusu potwierdzenia

Przykłady zmian stylu:

1. Zmiana koloru tła całego widoku `DPPaymentFormViewController` na czerwony:

```
[[UIView appearanceWhenContainedIn: NSClassFromString(@"DPPaymentFormViewController"),
nil] setBackgroundColor:[UIColor redColor]];
```

2. Zmiana koloru aktywnych przycisków znajdujących się w kontrolerze `DPDotPayViewController` na myCustomColor:

```
[[UIBarButtonItem appearanceWhenContainedIn:[DPDotPayViewController class], nil]
setTintColor:myCustomColor];
```

3. Zmiana koloru tekstu literałów w kontrolerze `DPDotPayViewController` na `myCustomColor`:

```
[[UILabel appearanceWhenContainedIn:[DPDotPayViewController class], nil]
setTextColor:myCustomColor];
```

Możliwa jest także zmiana domyślnych kolorów statusów. Można to wykonać ustawiając odpowiednią tablicę singletona `DPStyleManager`:

```
[DPStyleManager sharedInstance].colorsForSummaryStatus = @{
        kDPPaymentSummaryStatusNew  : [UIColor greyColor],
        kDPPaymentSummaryStatusProcessing :[UIColor greyColor],
        kDPPaymentSummaryStatusRejected  : [UIColor redColor],
        kDPPaymentSummaryStatusCompleted  : [UIColor greenColor],
        kDPPaymentSummaryStatusProcessingRealizationWaiting : [UIColor greyColor],
        kDPPaymentSummaryStatusProcessingRealization : [UIColor greyColor]
    };
```

## Zmiana stylu prezentacji ekranu podsumowania płatności

Korzystając z poniższych metod dla ekranu podsumowania płatności można ustawić kolor tła, czcionki tytułu, kolor i czcionkę przycisku:

- ustawienie koloru przycisku płatności `(domyślnie jest #ab191e)`

```
@property (nonatomic, strong) UIColor *paymentButtonColor;
- (void)setPaymentButtonColor:(UIColor * _Nonnull)paymentButtonColor;
```

- ustawienie koloru tekstu na przycisku płatności `(domyślnie jest white)`

```
@property (nonatomic, strong) UIColor *paymentButtonFontColor;
- (void)setPaymentButtonFontColor:(UIColor * _Nonnull)paymentButtonFontColor;
```

- ustawienie koloru nagłówka ekranu `(domyślnie jest #ab191e)`

```
@property (nonatomic, strong) UIColor *paymentHeaderDetailsColor;
- (void)setPaymentHeaderDetailsColor:(UIColor * _Nonnull)paymentHeaderDetailsColor;
```

- ustawienie koloru tekstu na nagłówku ekranu `(domyślnie jest white)`

```
@property (nonatomic, strong) UIColor *paymentHeaderDetailsFontColor;
- (void)setPaymentHeaderDetailsFontColor:(UIColor * _Nonnull)paymentHeaderDetailsFontColor;
```

## Własna kontrolka wyboru kanału

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

1. Przygotować obiekt klasy `DPPaymentInfo`, wypełnić go danymi (zgodnie z rozdziałem *Inicjalizacja płatności*, powyżej)
2. Pobrać listę kanałów, (z której można np. usunąć nieinteresujące nas pozycje), wraz z pobraniem dodatkowych szczegółów płatności:

```
[[DotPay sharedInstance] getChannelListForPaymentInfo:info withCompletion:^(NSArray
*channelList, DPPaymentDetails *paymentDetails, NSError *error) { }
```

3. Na koniec asynchronicznie prezentujemy kontroler na głównym wątku, wykonując następujące operacje:
    a. Zainicjalizowanie własnego kontrolera płatności z podaniem listy kanałów oraz szczegółów płatności
    b. Zainicjalizowanie kontrolera płatności `DPDotPayViewController` ze wskazaniem na własny kontroler wyboru kanału
    c. Ustawienie delegata zwrotnego
    d. Wyświetlenie kontrolera płatności Dotpay

Przykład:

```
dispatch_async(dispatch_get_main_queue(), ^{
MyCustomChannelViewController *myChannelViewController =
[[MyCustomChannelViewController alloc] initWithPaymentChannelList:channelList ];
DPDotPayViewController *paymentViewController = [[DPDotPayViewController alloc]
initWithPaymentChannelSelectionController: myChannelViewController
paymentDetails:paymentDetails];
paymentViewController.paymentControllerDelegate = self;
[self presentViewController:paymentViewController animated:YES completion:NULL];
        });
    }];
```

## Własna kontrolka podsumowania transakcji

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

1. W delegacie `DPDotPayViewControllerDelegate` wyłączenie ekranu podsumowania płatności
2. Zainicjalizowanie kontrolera podsumowania transakcji, który implementuje protokół `DPDotPayViewControllerDelegate` i metodę

```
- (BOOL)dotpayViewController:(DPDotPayViewController *)dotpayViewController
shouldShowViewControllerForPaymentSummary:(DPPaymentSummary *)paymentSummary;
```

3. Ustawienie delegata zwrotnego
4. Wyświetlenie kontrolera podsumowania transakcji

Przykład:

```
- (BOOL)dotpayViewController:(DPDotPayViewController *)dotpayViewController
shouldShowViewControllerForPaymentSummary:(DPPaymentSummary *)paymentSummary
    {
        return NO;
    }
(void)dotpayViewController:(DPDotPayViewController *)dotpayViewController
didFinishPaymentWithSummary:(DPPaymentSummary *)paymentSummary {
    MySummaryViewController *controller = [[MySummaryViewController alloc]
initWithPaymentSummary:paymentSummary];
    dispatch_async(dispatch_get_main_queue(), ^{
[self presentViewController:controller animated:YES completion ^{}];
    });
}
```

## 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:

```
[DPOneClickManager sharedInstance].enabled = NO;
```

UWAGA

Po wyłączeniu powyższej opcji nie są usuwane dane zapamiętanej wcześniej karty. Aby je usunąć należy wywołać metody opisane poniżej. Dodatkowym obowiązkiem Developera jest umieszczenie w aplikacji funkcji pozwalającej zlecić usunięcie zapamiętanych danych karty.

## Metody dostępne dla developera

Aby zarejestrować nową kartę należy zainicjalizować obiekt `DPOneClickPaymentCardInfo` i wywołać metodę:

```
[[DPOneClickManager sharedInstance] registerPaymentCardWithCardInfo:cardInfo
withCompletion:^(DPOneClickPaymentCard *response, NSError *error)
```

Aby dowiedzieć się, która karta jest kartą domyślną należy wywołać metodę:

```
[[DPOneClickManager sharedInstance] defaultOneClickPaymentCard];
```

Aby oznaczyć kartę, jako domyślną należy wywołać metodę:

```
[[DPOneClickManager sharedInstance]setDefaultOneClickPaymentCardWithId:(NSString
*)cardId];
```

Aby dowiedzieć się, jakie karty są zapamiętane, można wywołać metodę:

```
[[DPOneClickManager sharedInstance] paymentCards];
```

Aby usunąć kartę płatniczą należy wywołać:

```
[DPOneClickManager sharedInstance] unregisterPaymentCardWithId:<#(NSString *)#>
withCompletion:<#^(NSDictionary *response, NSError *error)completion#>];
```

Aby usunąć wszystkie zapisane karty płatnicze wywołując:

```
[[DPOneClickManager sharedInstance] clearData];
```

Wywołanie tej metody zleci usunięcie wszystkich kart w systemie Dotpay, a także usunie dane przechowywane lokalnie.

## Zarządzanie kartami

Dodatkową funkcją biblioteki jest menadżer kart. W menadżerze na liście widoczne są zapamiętane karty. Z poziomu zarządzania istnieje możliwość dodania, oznaczenia domyślnej lub usunięcia karty.

## Wykorzystanie wbudowanej kontrolki

Aby skorzystać z wbudowanej kontrolki zarządzania kartami, należy zainicjować i wyświetlić kontroler `DPOneClickPaymentCardManagerViewController`:

```
NSString *merchantId = #yourMerchantId
NSString *currency = #yourcurrencycode //(eg. @"PLN")
DPOneClickPaymentCardManagerViewController *viewController =
[[DPOneClickPaymentCardManagerViewController alloc] initWithMerchantId:merchantId
currency:currency firtsName:firstName lastName:lastName email:email];
[self presentViewController:viewController animated:YES completion:nil];
```

Jeśli nie chcesz podawać danych osobowych posiadacza karty, wprowadź NULL w miejsce parametrów lub skorzystaj z uproszczonego konstruktora, który ich nie wymaga.

## Własna kontrolka menadżera kart

Aby lepiej dopasować prezentowane dane do specyfiki Sklepu, można stworzyć własną kontrolkę zarządzania kartami, korzystając z metod udostępnionych przez bibliotekę.

## Płatność kartą bez wypełnienia formularza

Biblioteka umożliwia wykonanie płatności kartą za pomocą jednego kliknięcia, bez potrzeby wyświetlania formularza podsumowania dla płatności.

UWAGA

W czasie wykonywania płatności istnieje prawdopodobieństwo dodatkowej weryfikacji karty, niezależnej od SDK.

Aby wykonać płatność należy zainicjalizować obiekt `DPPaymentInfo` oraz wywołać metodę:

```
[[DotPay sharedInstance] oneClickPaymentWithPaymentInfo:paymentInfo paymentCard:nil
withCompletion:^(DPPaymentSummary *_Nullable jsonResponse, NSError * _Nonnull error)
```

UWAGA

Jeżeli parametr `paymentCard:nil` wówczas płatność zostanie zrealizowana z ustawionej domyślnej karty.

## 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 w odpowiedni sposób zainicjalizować `DPOneClickManager`:

```
[[DPOneClickManager sharedInstance]configureWithExternalCards:( NSArray <
DPOneClickExternalPaymentCard *> *)externalCards forCustomerId:( NSString
*)customerId];
```

Gdzie `customerId` jest odpowiednikiem parametru `credit_card_customer_id` opisywanego w głównej dokumentacji integracji technicznej, a `externalCards` listą obiektów `DPOneClickExternalPaymentCard` zawierającą komplet informacji o karcie. Znaczenie poszczególnych właściwości:

| PARAMETR | ZNACZENIE / OPIS |
|---|---|
| maskedNumber | Typ: NSString<br>Zamaskowany numer karty |
| creditCardId | Typ: NSString<br>Identyfikator karty (znaczenie zgodnie z dokumentacją podstawową Dotpay) |

| | |
|---|---|
| brandName | Typ: NSString<br>Nazwa karty prezentowana Użytkownikowi |
| brandCodename | Typ: NSString<br>Typ karty (znaczenie zgodne z dokumentacją podstawową Dotpay) |
| logoUrlPath | Typ: NSString<br>Link do logotypu karty (zgodnie z dokumentacją podstawową Dotpay, link jest zwracany razem z danymi zarejestrowanej karty) |

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:

```
[[DPOneClickManager sharedInstance] clearExternalCardsData]
```

Dodatkowo, istnieje możliwość wyłączenia widoczności 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). Aby wyłączyć użycie takich kart, wystarczy ustawić następującą właściwość:

```
[DPOneClickManager sharedInstance].useInternalCards = NO;
```

## 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 `configureWithExternalCards` 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 zainicjalizować `DPOneClickManager` aktualną listę kart zewnętrznych.

## Uzyskanie informacji o zmianach wywołanych działaniami Użytkownika

Aby uzyskać informacje o wpływie działań Użytkownika na bazę kart podawanych z zewnątrz (o fakcie rejestracji, zmianie karty domyślnej), można zaimplementować delegata `DPOneClickManagerDelegate:`

```
- (void) oneClickManager:( DPOneClickManager *)manager didRegisterNewCreditCard:(
DPOneClickCardRegistrationData *)cardRegistrationData;
```

```
- (void) oneClickManager:( DPOneClickManager *)manager
didChangeDefaultOneClickPaymentCard:( DPOneClickPaymentCard *)newDefaultCard;
```

I ustawić referencję do niego:

```
[DPOneClickManager sharedInstance].delegate = self;
```

Gdy tylko nowa karta zostanie zarejestrowana, aplikacja zostanie o tym poinformowana poprzez wywołanie metody `didRegisterNewCreditCard` zaimplementowanego delegate. Gdy tylko otrzymamy tę informację, aplikacja może zwrócić się do backendu w celu odświeżenia listy kart.

Gdy zmieni się karta używana jako domyślna (należy zwrócić uwagę, iż karta domyślna jest innym pojęciem, niż karta ostatnio użyta przy płatności standardowej), wywołana zostanie metoda `didChangeDefaultOneClickPaymentCard` zaimplementowanego delegata.

## 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:

```
[DPOneClickManager sharedInstance].treatsCvvAsPassword = YES;
```

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 (np. Visa Checkout), powstał mechanizm umożliwiający wywołanie płatności ze wskazaniem kanału, a przez to pominięciem jego wyboru, oraz pominięciem formularza płatniczego.

Aby wykonać taką płatność, należy zainicjować obiekt `DPPaymentInfo` oraz wywołać metodę:

```
[[DotPay sharedInstance] fastPaymentWithPaymentInfo:paymentInfo paymentChannelId:
channelId showingDefaultSummary:defaultSummary withCompletion:^(DPPaymentSummary
*_Nullable jsonResponse, NSError * _Nullable error))completion;
```

Parametr określający identyfikator kanału jest obowiązkowy. Parametry dodatkowe, specyficzne dla kanału, należy przekazać jako elementy słownika `additionalInformation` w obiekcie `DPPaymentInfo`.

## 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 „YES", 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 rozpocząć od przygotowania implementacji delegata `DPMasterpassPaymentDelegate:`

```
-(void) masterpassPaymentShouldCreateAuthorizationOn:(UIViewController
<DPMasterpassAuthorization> *)viewController;
- (void)masterpassPaymentShouldAuthorizeUserPaymentOn:(UIViewController
<DPMasterpassAuthorization> *)viewController;
```

I ustawić go na głównym kontrolerze SDK, np.:

```
dotPayViewController.masterpassPaymentDelegate = self;
```

Zwrotnie, przekazany w parametrach wskazanych metod UIViewKontroler implementuje metody delegata `DPMasterpassAuthorization`:

```
- (void)didCreateAuthorization:(BOOL)successful andError:(NSError *)error;
- (void)didAuthorizeUser:(BOOL)successful andError:(NSError *)error;
```

których należy użyć do zwrotnego przekazania kontroli.

Metoda `masterpassPaymentShouldCreateAuthorizationOn` 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 `didCreateAuthorization` delegata zwrotnego przekazując status tworzenia autoryzacji. W wypadku pozytywnego, SDK będzie kontynuowało proces rejestracji do portfela, w wypadku negatywnego przerwie ten proces.

Metoda `masterpassPaymentShouldAuthorizeUserPaymentOn` zostanie wywołana każdorazowo, gdy zalogowany Płacący będzie chciał wykonać kolejną płatność powiązanym portfelem. W ramach jej przebiegu powinniśmy wykonać autoryzację (np. weryfikację PINu), a w wypadku powodzenia wywołać metodę `didAuthorizeUser` delegata zwrotnego przekazując jej status. Potwierdzenie autoryzacji pozwoli wykonać płatność, informacja o nieprawidłowej autoryzacji przerwie ją.

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

```
-(void) masterpassPaymentShouldCreateAuthorizationOn:(UIViewController
<DPMasterpassAuthorization> *)viewController {

        [viewController didCreateAuthorization:YES andError:nil];

    }

-(void)  masterpassPaymentShouldAuthorizeUserPaymentOn:  (UIViewController
<DPMasterpassAuthorization> *)viewController {

        [viewController didAuthorizeUser:YES andError:nil];

}
```

## 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 zainicjować i wyświetlić kontroler `DPPaymentHistoryController`:

```
DPPaymentHistoryController *historyController = [[DPPaymentHistoryController alloc]
initWithPaymentHistory:nil];
[self presentViewController:historyController animated:YES completion:NULL];
```

## Zmiana stylu prezentacji

Zmiana sposób prezentacji elementów kontrolek procesu płatności wykonywana jest z wykorzystaniem frameworka `UIAppearance`. Szczegóły opisane zostały w rozdziale *Zmiana stylu prezentacji*, dotyczącym kontrolerów płatności.

Kontroler historii, dla którego można wprowadzić ograniczenia zmiany prezentacji to `DPPaymentHistoryController`.

Przykład:

```
[[UIView appearanceWhenContainedIn: [DPPaymentHistoryController class], nil]
setBackgroundColor:[UIColor redColor]];
```

## Własna kontrolka historii

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

Pobranie danych historii (tablica obiektów klasy `DPPaymentSummary`) dostępna jest we właściwości `paymentHistory` singletona `DPPaymentHistory`:

```
NSArray *paymentSummaries = [DPPaymentHistory sharedInstance].paymentHistory;
```

Dodatkowo, dla transakcji zaprezentowanych we własnej historii można:

1. Usunąć pojedynczą pozycję:

```
[[DPPaymentHistory sharedInstance] removePaymentSummary: paymentToRemove];
```

2. Sprawdzić aktualny status płatności dla pozycji z historii podczas jej wyświetlania, jednocześnie odświeżając wyświetlone informacje:

   a. We własnym widoku implementujemy metody obserwera `DPPaymentHistoryObserver`, które będą odświeżały widok:

```
- (void)paymentHistory:(DPPaymentHistory *)paymentHistory
didUpdatePaymentSummary:(DPPaymentSummary *)paymentSummary;
- (void)paymentHistory:(DPPaymentHistory *)paymentHistory
didFailedToUpdatePaymentSummary:(DPPaymentSummary *)paymentSummary
withError:(NSError *)error;
```

   b. Rejestrujemy się na informacje o zmianach:

```
[[DPPaymentHistory sharedInstance] registerObserver: self]
```

   c. Wywołujemy metodę aktualizującą status wybranej pozycji:

```
[[DPPaymentHistory sharedInstance] updatePaymentSummary: paymentToUpdate];
```

   d. Przy opuszczeniu własnego kontrolera historii, wyrejestrowujemy się z obserwowania informacji o zmianach:

```
[[DPPaymentHistory sharedInstance] unRegisterObserver: self]
```