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

dotpay®

Płatności elektroniczne klasy e-biznes

SDK (iOS)

Wersja 1.2.0

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 8.0 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. Do projektu należy dodać framework Dotpay, znajdujący się w podkatalogu `lib` paczki (w zakładce `Build Phases` wybrać sekcję `Embed Frameworks`)
2. Do projektu należy dodać także systemowy framework JavaScriptCore.
3. W każdym miejscu, w którym będziemy korzystać z elementów SDK należy zaimportować plik nagłówkowy:

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

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 ID konta w systemie Dotpay, na rzecz, którego dokonywana jest płatność (ID konta Sprzedawcy)
amount	Typ: NSDecimalNumber Kwota transakcji
textDescription	Typ: NSString Tytuł/opis płatności

currency	Typ: DPCurrency Domyślna wartość: „PLN” Określenie w jakiej walucie podany jest parametr amount. Sposób na pobranie listę dostępnych walut opisany został w podrozdziale *Dostępne waluty*, poniżej.
senderInformation	Typ: NSDictionary Domyślna wartość: nil Dodatkowe informacje o kupującym. Słownik 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	Typ: NSDictionary Domyślna wartość: nil Dodatkowe parametry przekazywane podczas procesu płatności, zgodnie z szczegółami przekazanymi w dodatkowych dokumentacjach.
control	Typ: NSString Domyślna wartość: nil 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: nil Adres URL Sklepu, do odbioru parametrów potwierdzających zrealizowanie lub odmowę realizacji transakcji. 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];
[self presentViewController:viewController animated:YES completion:nil];
```

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.

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]
```