Tpay iOS Mobile Library

Biblioteka mobilna przygotowana dla systemu iOS.

Konfiguracja

Biblioteka wspiera iOS w wersji 9.0 lub nowszej. W środowisku Xcode należy dołączyć do projektu TpaySDK.framework.

- Dokumentacja Apple https://developer.apple.com/library/ios/recipes/xcode_help-project_editor/Articles/AddingaLibrarytoaTarget.html
- Biblioteka zależy od frameworków UIKit oraz Foundation, dołączonych wraz z TpaySDK.

Budowa

TpayPayment - model płatności.

TpayViewController - kontroler widoku płatności.

TpayPaymentDelegate - delegat dostarczający informację zwrotną o statusie transakcji.

Framework wykorzystuje parametry *mReturnUrl* oraz *mReturnErrorUrl* do ustalenia statusu transakcji. W przypadku braku uzupełnienia zostaną nadane im domyślne wartości.

Sposób użycia

Poniżej opisano możliwy sposób implementacji w projekcie przy użyciu Storyboard.

- Po poprawnym skonfigurowaniu projektu utwórz w *Storyboard*zie pusty *ViewController* nazwijmy go *PaymentViewController*.
- Utwórz segue oraz nazwij *Identifier* do kontrolera np. *TpayPaymentInnerSegue*.
- W utworzonym *ViewController*ze ustaw *Custom Class* na *TpayViewController*.
- Następnie w pliku *PaymentViewController.h* dołącz nagłówek:

```
#import <TpaySDK/TpayViewController.h>
```

- Pozostając w tym samym pliku zadeklaruj implementację protokołu *TpayPaymentDelegate* w kontrolerze *TpayViewController* (możesz również dokonać tego w wewnętrzym interfejsie).

```
@interface PaymentViewController : UIViewController <TpayPaymentDelegate>
```

- Przejdź do implementacji kontrolera *PaymentViewController.m*.
- Utwórz w nim płatność i ustaw wymagane parametry zgodnie z dokumentacją.

```
self.payment = [TpayPayment new];

self.payment.mId = @"twoje_id";
self.payment.mAmount = @"kwota_transakcji";
self.payment.mDescription = @"opis_transakcji";
self.payment.mClientEmail = @"email_klienta";
self.payment.mClientName = @"imie_nazwisko_klienta";
self.payment.md5 = @"obliczony_md5";

// W przypadku braku obliczonego MD5:

self.payment.mCrc = @"crc";
self.payment.mSecurityCode = @"twój_kod"
```

Można również ustawić gotowy, wygenerowany wcześniej link i wtedy konfiguracja obiektu reprezentującego płatność wygląda jak poniżej:

```
self.payment.mPaymentLink = @"wygenerowany_link_płatności";
```

- Aby rozpocząc proces płatności należy utworzyć *TpayViewController*. W chwili wywołania wykonywany jest segue, który trzeba nazwać w *Storyboard*zie i przechwycić, w nim należy przekazać naszą płatność do wewnętrznego kontrolera oraz ustawić delegata dla zdarzeń.

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    NSString *segueName = segue.identifier;
    if ([segueName isEqualToString: @"TpayPaymentInnerSegue"]) {
        TpayViewController *childViewController = (TpayViewController *) [segue destinationViewCor
        childViewController.payment = self.payment;
        childViewController.delegate = self;
    }
}
```

Należy się upewnić, że została ona wcześniej odpowiednio utworzona.

- Należy też dodać metody infromujące o przebiegu transakcji.

```
- (void)tpayDidSucceedWithPayment:(TpayPayment *)payment

- (void)tpayDidFailedWithPayment:(TpayPayment *)payment
```

- Jeżeli używamy metod zachowywania stanu aplikacji. Należy pamietąć o implementacji odpowiednich metod. Obiekt TpayPayment może być kodowany i dekodowany za pomocą klasy *NSCoder*.

```
- (void)encodeRestorableStateWithCoder:(NSCoder *)coder {
    [coder encodeObject:self.payment forKey:kExtraPayment];
    [super encodeRestorableStateWithCoder:coder];
}

- (void)decodeRestorableStateWithCoder:(NSCoder *)coder {
    self.payment = [coder decodeObjectForKey:kExtraPayment];
    [super decodeRestorableStateWithCoder:coder];
}
```

Sposób użycia biblioteki w projekcie - płatności BLIK oraz BLIK OneClick

Użycie domyślnych widoków

Biblioteka pozwala na szybkie użycie płatności BLIK oraz BLIK One Click za pomocą gotowych, domyślnych widoków płatności.

W pierwszym kroku należy stworzyć obiekt reprezentujący transakcję BLIK::

```
    TpayBlikTransaction *transaction = [TpayBlikTransaction new];
    transaction.mApiPassword = @"haslo_api";
    transaction.mId = @"twoje_id";
    transaction.mAmount = @"kwota_transakcji";
    transaction.mCrc = @"kod_crc";
    transaction.mSecurityCode = @"kod_bezpieczeństwa";
    transaction.mDescription = @"opis_transakcji";
    transaction.mClientEmail = @"email_klienta";
    transaction.mClientName = @"imie_nazwisko_klienta";
    [transaction addBlikAlias:@"alias_blik" withLabel:@"etykieta"
    andKey:@"klucz_aplikacji"];
```

Hasło do api (parametr *api_password*) jest polem obowiązkowym - w dokumentacji API na stronie 2. można znaleźć więcej szczegółów. Pozostałe parametry opisane są w dokumentacji ogólnej.

Zamiast podawania parametrów *security code* i *crc*, można podać parametr *md5 code*, który wygenerować można zgodnie z dokumentacją.

W przypadku transakcji BLIK bez możliwości rejestracji aliasu (czyli bez możliwości skorzystania z One Click) dodanie aliasu BLIK jest opcjonalne. W przypadku transakcji dla zarejestrowanego aliasu, bądź chęci rejestracji aliasu należy podać przynajmniej jeden alias za pomocą metody *addBlikAlias()*.

Metoda *addBlikAlias()* przyjmuje parametry:

- alias: pole obowiązkowe, typ NString
- label: etykieta aliasu, pole opcjonalne, typ NString
- key: numer aplikacji, pole opcjonalne, typ NString.

Więcej informacji na temat poszczególnych parametrów zawarto w dokumentacji API na stronie 7.

Jeden alias BLIK może być zarejestrowany do wielu aplikacji bankowych, co powoduje niejednoznaczność aliasu - domyślny widok płatności obsługuje tę sytuację wyświetlając stosowny widok wyboru.

Kolejnym krokiem, pozwalającym na wyświetlenie domyślnego widoku płatności, jest inicjalizacja storyboardu zawierającego kontroler widoku płatności, następnie jego zainicjowanie oraz przekazanie odpowiednich parametrów:

```
   UIStoryboard *storyboard = [UIStoryboard storyboardWithName:@"TpayBlikStoryboard"
 bundle: [NSBundle bundleWithIdentifier:@"pl.transferuj.TpaySDK"]];
    TpayBlikTransactionViewController *blikDefaultVC = (TpayBlikTransactionViewController *)[story
    blikDefaultVC.blikDelegate = delegate;
    blikDefaultVC.blikTransaction = transaction;
    blikDefaultVC.key = @"apiKey";
    blikDefaultVC.viewType = viewType;
```

Przekazywane parametry:

- blikDelegate - delegat dostarczający informację zwrotną z API
- blikTransaction - obiekt transakcji stworzony w kroku 1.
- key - unikalny ciąg dostępu, wygenerowany w Panelu Odbiorcy Płatności w zakładce Ustawienia->API
- viewType: jedna z wartości kRegisteredAlias, kUnregisteredAlias, kOnlyBlik.

Typ widoku, który powiniśmy wybrać zależny jest od typu transakcji, którą chcemy przeprowadzić:

- kOnlyBlik - pokazuje widok pozwalający dokonać jedynie transakcji BLIK, bez możliwości rejestracji aliasu (dodawanie aliasu do obiektu transakcji nie jest wtedy konieczne)
- kUnregisteredAlias - pokazuje widok pozwalający dokonać transakcji BLIK z możliwością wyrażenia chęci rejestracji aliasu
- kRegisteredAlias - pokazuje widok płatności dla zarejestrowanego aliasu.

Ponadto dostępny jest również typ kNonUniqueAlias, używany wewnątrz biblioteki do obsługi sytuacji niejednoznaczego aliasu BLIK.

Następnie należy zaprezentować kontroler widoku płatności:

```
  [self.navigationController pushViewController:blikDefaultVC animated:YES];
```

Ostatni krok to rozszerzenie klasy, która obsługiwać będzie odpowiedź zwrotną z API o obsługę protokołu TpayBlikTransactionDelegate.

Przykład: Jeśli zaprezentowaliśmy kontroler widoku płatności z kontrolera MyViewController, to powinniśmy ustawić pole:

```
 blikDefaultVC.blikDelegate = self;
```

natomiast klasa MyViewController powinna rozszerzać protokół TpayBlikTransactionDelegate:

```
@interface MyViewController () <TpayBlikTransactionDelegate>
```

Implementacja kontrolera MyViewController powinna zawierać metody:

```
- (void) tpayDidSucceedWithBlikTransaction:(TpayBlikTransaction *)transaction
andResponse: (id)responseObject {
   // Transakcja poprawna.
   // Klient powinien zatwierdzić płatność
   // w aplikacji mobilnej banku.
   // Poczekaj na powiadomienie.
}

- (void) tpayDidFailedWithBlikTransaction:(TpayBlikTransaction *)transaction
andResponse: (id)responseObject {
  // Wystąpił błąd.
  // Odpowiedź jest klasy NSDictionary, jeśli przyszedł błąd z API.
  // Odpowiedź jest klasy NSError, jeśli jest to inny błąd,
  // np. brak połączenia z internetem.
  // Więcej w dokumentacji API.
}
```

Samodzielna obsługa płatności BLIK i BLIK One Click

Biblioteka zawiera metody pozwalające na obsługę płatności bez wykorzystania domyślnych widoków.

Należy stworzyć obiekt reprezentujący transakcję BLIK:

```
TpayBlikTransaction *transaction = [TpayBlikTransaction new];
  transaction.mApiPassword = @"haslo_api";
  transaction.mId = @"twoje_id";
  transaction.mAmount = @"kwota_transakcji";
  transaction.mCrc = @"kod_crc";
  transaction.mSecurityCode = @"kod_bezpieczeństwa";
  transaction.mDescription = @"opis_transakcji";
  transaction.mClientEmail = @"email_klienta";
  transaction.mClientName = @"imie_nazwisko_klienta";
  transaction.mBlikCode = "6_cyfrowy_kod_blik";
  [transaction addBlikAlias:@"alias_blik" withLabel:@"etykieta"
  andKey:@"klucz_aplikacji"];
```

Szczegółowy opis w sekcji *Użycie domyślnych widoków*.

Następnie należy skorzystać z klienta pozwalającego na wysłanie transakcji oraz przekazać delegata, który obsłuży odpowiedź zwrotną z API (zgodnie z ostatnim punktem sekcji *Użycie domyślnych widoków*):

```
TpayApiClient *client = [TpayApiClient new];
    client.delegate = delegate;
    [client payWithBlikTransaction:transaction withKey:@"apiKey"];
```

Szczegóły związane z odpowiedziami API oraz kodami błędów znajdują się w dokumentacji API na stronach 6-13.

Historia zmian

Wersja 1.0 (Czerwiec 2015)
Wersja 2.0 (Maj 2017)
Wersja 3.0 (Lipiec 2017)