Transferuj Android Mobile Library

Biblioteka mobilna przygotowana dla systemu Android.

Konfiguracja projektu

- Biblioteka jest zgodna z API >= 16. Jest to minimalna wspierana wersja, którą należy ustawić w projekcie.
- W folderze "app", znajdującym się w głównym katalogu projektu, należy dodać nowy folder o nazwie "libs".
- Do folderu "libs" należy przekopiować bibliotekę "tpay-android-library.aar".
- W pliku "build.gradle" (należącym do głównego modułu projektu, w przykładowym projekcie jest to "app") należy dodać:

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

oraz w sekcji "dependencies":

```
dependencies {
    compile(name:'tpay-android-library', ext:'aar')
    compile 'com.squareup.retrofit2:retrofit:2.3.0'
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'
}
```

- Ostatnim etapem integracji jest sychronizacja projektu z dodaną biblioteką poprzez naciśnięcie przycisku "Sync Project with Gradle Files" (zielona ikona w górnej części interfejsu Android Studio).
- Biblioteka jest zależna od pakietów "com.squareup.retrofit2:retrofit:2.3.0" i "com.squareup.retrofit2:converter-gson:2.1.0" których użycie powinno być zadeklarowane w pliku *build.gradle* jak pokazano powyżej.

Sposób użycia biblioteki w projekcie - płatności poprzez mobilną stronę WWW

Poniżej opisano przykładowy sposób rozszerzenia klasy Activity.

- Po poprawnym skonfigurowaniu projektu rozpocznij od deklaracji zmiennej:

```
private TpayPayment.Builder paymentBuilder = null;
```

- W metodzie onSaveInstanceState dodaj następujący fragment kodu:

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putParcelable(
        TpayActivity.EXTRA_TPAY_PAYMENT,
        paymentBuilder);
}
```

- W metodzie onCreate dodaj:

```
if (savedInstanceState == null) {
    paymentBuilder = new TpayPayment.Builder()
    // ustawienia parametrów
} else {
    paymentBuilder = savedInstanceState
.getParcelable(TpayActivity.EXTRA_TPAY_PAYMENT);
}
```

- Po utworzeniu obiektu reprezentującego płatność należy ustawić wymagane paramatry zgodnie z
  dokumentacją znajdującą się na stronie transferuj.pl (dokumentacja). Przykładowe ustawienia parametrów:

```
if (savedInstanceState == null) {
    paymentBuilder = new TpayPayment.Builder()
        .setId("twoje_id")
        .setAmount("kwota_transakcji")
        .setCrc("crc")
        .setSecurityCode("kod_bezpieczeństwa")
        .setDescription("opis_transakcji")
        .setClientEmail("email_klienta")
        .setClientName("imie_nazwisko_klienta");
} else {
    paymentBuilder = savedInstanceState
        .getParcelable(TpayActivity.EXTRA_TPAY_PAYMENT);
}
```

- Zamiast podawania parametrów “security code” i “crc”, można podać parametr “md5 code”, który wygenerować można zgodnie z dokumentacją):

```
paymentBuilder = new TpayPayment.Builder()
        .setMd5Code("wygenerowany_kod_MD5")
```

- Można również ustawić gotowy, wygenerowany wcześniej link. Konfiguracja obiektu reprezentującego płatność wygląda wtedy następująco:

```
if (savedInstanceState == null) {
    paymentBuilder = new TpayPayment.Builder()
                .setPaymentLink("wygenerowany_link_platnosci");
} else {
    paymentBuilder = savedInstanceState.getParcelable(TpayActivity.EXTRA_TPAY_PAYMENT);
}
```

- Aby rozpocząć proces płatności, wywołujemy aktywność oczekującą na rezultat poprzez dodanie poniższego fragmentu kodu w odpowiednim miejscu (np. kod może być wywoływany po naciśnięciu przycisku):

```
final Intent payIntent = new Intent(MainActivity.this,
    TpayActivity.class);
final TpayPayment tpayPayment = paymentBuilder.create();
payIntent.putExtra(TpayActivity.EXTRA_TPAY_PAYMENT, tpayPayment);

startActivityForResult(payIntent,
TpayActivity.TPAY_PAYMENT_REQUEST);
```

Po wywołaniu aktywności, aplikacja pokaże widok WebView, dzięki któremu można będzie przejść przez cały proces płatności.

- Ostatni krok to dodanie poniższego kodu, obsługującego informacje zwrotne z wywołanej w poprzednim korku aktywności, w metodzie onActivityResult:

```
switch (requestCode) {
    case TpayActivity.TPAY_PAYMENT_REQUEST:
        if (resultCode == RESULT_OK) {
            // Transakcja poprawna. Poczekaj na powiadomienie.
        } else {
            // Użytkownik anulował transakcję lub wystąpił błąd.
        }
    default:
        super.onActivityResult(requestCode, resultCode, data);
}
```

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, wykorzystując do tego przygotowany builder:

```
TpayBlikTransaction transaction = new TpayBlikTransactionBuilder()
    .setApiPassword("haslo_api")
    .setId("twoje_id")
    .setAmount("kwota_transakcji")
    .setCrc("kod_crc")
    .setSecurityCode("kod_bezpieczenstwa")
    .setDescription("opis_transakcji")
    .setClientEmail("email_klienta")
    .setClientName("imie_nazwisko_klienta")
    .addBlikAlias("alias_blik", "etykieta", "klucz_aplikacji")
    .build();
```

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 String
- label: etykieta aliasu, pole opcjonalne, typ String
- key: numer aplikacji, pole opcjonalne, typ String.

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 przygotowanie intencji za pomocą statycznej metody *createIntent()* klasy TpayBlikDefaultActivity:

```
Intent intent = TpayBlikDefaultActivity.createIntent(activity,
                    transaction,
                    key,
                    viewType);
```

Metoda *createIntent()* przyjmuje parametry:

- activity - referencja do aktywności, z której przechodzić będziemy do domyślnego widoku płatności
- transaction - obiekt transakcji stworzony w kroku 1.
- key - unikalny ciąg dostępu, wygenerowany w Panelu Odbiorcy Płatności w zakładce Ustawienia->API
- viewType: jedna z wartości TpayBlikDefaultActivity.BlikViewType.REGISTERED_ALIAS, TpayBlikDefaultActivity.BlikViewType.UNREGISTERED_ALIAS, TpayBlikDefaultActivity.BlikViewType.ONLY_BLIK.

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

- TpayBlikDefaultActivity.BlikViewType.ONLY_BLIK - pokazuje widok pozwalający dokonać jedynie transakcji BLIK, bez możliwości rejestracji aliasu (dodawanie aliasu do obiektu transakcji nie jest wtedy konieczne)
- TpayBlikDefaultActivity.BlikViewType.UNREGISTERED_ALIAS - pokazuje widok pozwalający dokonać transakcji BLIK z możliwością wyrażenia chęci rejestracji aliasu
- TpayBlikDefaultActivity.BlikViewType.REGISTERED_ALIAS - pokazuje widok płatności dla zarejestrowanego aliasu.

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

Następnie należy wywołać aktywność oczekującą na rezultat zgodnie z przygotowaną intencją:

```
startActivityForResult(intent,
    TpayBlikDefaultActivity.BLIK_ACTIVITY_REQUEST);
```

Ostatni krok to dodanie poniższego kodu, obsługującego informacje zwrotne z wywołanej w poprzednim korku aktywności, w metodzie onActivityResult:

```
    if (requestCode == TpayBlikDefaultActivity.BLIK_ACTIVITY_REQUEST) {
        switch (resultCode) {
            case TpayBlikDefaultActivity.BLIK_RESULT_SUCCESS:
                 // Transakcja poprawna.
                 // Klient powinien zatwierdzić płatność
                 // w aplikacji mobilnej banku.
                 // Poczekaj na powiadomienie.
```

```
            break;

        case TpayBlikDefaultActivity.BLIK_RESULT_ERROR:
            // Wystąpił błąd o podanym kodzie błędu.
            // Więcej w dokumentacji API.
            break;

        case TpayBlikDefaultActivity.BLIK_RESULT_FAILURE:
            // Wystąpił błąd typu Throwable,
            // np. brak połączenia internetowego.
            break;
    }
}
```

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 = new TpayBlikTransactionBuilder()
      .setApiPassword("haslo_api")
      .setId("twoje_id")
      .setAmount("kwota_transakcji")
      .setCrc("kod_crc")
      .setSecurityCode("kod_bezpieczenstwa")
      .setDescription("opis_transakcji")
      .setClientEmail("email_klienta")
      .setClientName("imie_nazwisko_klienta")
      .setBlikCode("6_cyfrowy_kod_blik")
      .addBlikAlias("alias_blik", "etykieta", "klucz_aplikacji")
      .build();
```

Szczegółowy opis w sekcji *Użycie domyślnych widoków*.

Następnie należy skorzystać z klienta pozwalającego na wysłanie transakcji oraz obsłużyć odpowiedzi:

```
TpayClient.getInstance().postBlikTransaction(key, transaction,
    new TpayBlikCallback<TPayBlikResponse>() {
        @Override
        public void onResponseSuccess(TPayBlikResponse response) {
            // Sukces oznacza jedynie,
            // że transakcja została wysłana poprawnie.
```

```
            // Rezultat zależny od otrzymanej odpowiedzi.
        }

        @Override
        public void onResponseError(ResponseBody errorBody) {
            // Błąd w trakcie wysyłania transakcji.
        }

        @Override
        public void onResponseFailure(Throwable t) {
           // Błąd w trakcie wysyłania transakcji typu Throwable.
           // Np. brak połączenia z internetem.
        }
    });
```

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)