Konto Cloud E-Wallet Platform

Android SDK Integration

Publication date:
July 17, 2017

ContoWorks GmbH

Data classification: NDA Confidential

Document identifier: 2A9ABA29-734D-44F8-A666-26043C3D1A45

Contents

1. Add KontoCloud SDK to your app

To integrate the KontoCloud SDK libraries into your own project, you need to perform a few basic tasks to prepare your project. This tutorial assumes your IDE for application development is Android Studio.

First, copy SDK files (*kontocloud-sdk.aar*, *kontocloud-apiclient.aar, kontocloud-uicomponents.aar*) to the module "*libs*" folder (usually the *app/libs/)*.

Then, in your module Gradle file (usually the *app/build.gradle*), add the dependencies for the KontoCloud SDK

```
android {
    // ...
}


repositories {
    flatDir {
        dirs 'libs'
    }
}

dependencies {
    // ...

    // KontoCloud SDK dependencies
    compile 'net.danlew:android.joda:2.9.7'
    compile 'com.squareup.retrofit2:retrofit:2.1.0'
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'
    compile 'com.samskivert:jmustache:1.13'

    // KontoCloud SDK modules
    compile(name:'kontocloud-apiclient', ext:'aar')
    compile(name:'kontocloud-uicomponents', ext: 'aar')
    compile(name:'kontocloud-sdk', ext:'aar')
}
```

Add the *INTERNET* permission to the AndroidManifest file (usually the *app\src\main\AndroidManifest.xml*)

```
<manifest xmlns:android...>
    <!-- ... -->
    <uses-permission android:name="android.permission.INTERNET"/>
    <!-- ... -->
    <application>
        <!-- ... -->
    </application>
</manifest>
```

2. KontoCloud SDK

2.1. Payment Form

2.1.1. Request authorization token

In order to show Payment Form your app should request an authorization token from the KontoCloud API by calling following API method depending on payment form mode:

Payment Form Mode	**API method**
REGISTRATION	Init Add Stored Payment Option
PAYMENT	Init Authorize

In case of your app is using KontoCloud SDK API Services see Init add stored payment option.

1.1.2. Initialize and render Payment Form

Add the com.contoworks.kontocloud.uicomponents.widget.PaymentForm view to the layout.

```
<com.contoworks.kontocloud.uicomponents.widget.PaymentForm
    android:id="@+id/payment_form"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

In addition, assuming xmlns:app="http://schemas.android.com/apk/res-auto is declared as namespace at the top of your layout, you can also define custom attributes app:mode and app:paymentProviderMode.

Once you have added a PaymentForm widget to layout, obtain a handle to the object, initialize, attach callbacks, and call render method.

```
PaymentForm paymentForm = (PaymentForm) findViewById(R.id.payment_form);

// Initialize payment form
paymentForm.getOptions().setMode(PaymentFormMode.REGISTRATION);

paymentForm.getOptions().setApiUrl(Constants.API_URL);
paymentForm.getOptions().setPaymentProviderMode(PaymentProviderMode.TEST);

// Attach callbacks
paymentForm.setOnSubmitCallback(new PaymentForm.OnSubmitCallback() {
    @Override
    public void onBeforeSubmit(OnBeforeSubmitArgs args) {
        // For example, disable Submit button and show loading indicator.
    }

    @Override
    public void onSuccess(String paymentOptionCode, String
authorizationToken) {
        // Complete registration/payment process (see 2.1.3)
    }

    @Override
    public void onError(String errorMessage) {
        // Handle error. For example, close payment form and show Toast.
    }
});

// Render payment form
paymentForm.render(paymentOptionCode, authorizationToken);


// Initialize save button
saveButton = (Button) getView().findViewById(R.id.save_button);
saveButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Submit payment form when user clicks Save button
        paymentForm.submit();
    }
});
```

1.1.2.1. Screen orientation change support

In case your app supports screen orientation change, we recommend to declare that your activity handles the screen configuration change itself, which will prevent the payment form from clearing input data.

Edit the appropriate *<activity>* element in your AndroidManifest (usually the *app\src\main\AndroidManifest.xml*) file to include the *android:configChanges* attribute with the value “keyboardHidden|orientation|screenSize”.

```
<manifest xmlns:android...>
    <!-- ... -->
    <application>
        <!-- ... -->
        <activity
            android:name=".AddPaymentOptionActivity"
            android:configChanges="keyboardHidden|orientation|screenSize"
            android:label="Register Payment Option" />
    </application>
</manifest>
```

1.1.1.2. UI Customization

The Payment Form uses styles to customize UI. A style is a collection of properties that specify the look and format for a View or window. A style can specify properties such as font size, font color, etc. A style is defined in an XML resource that is separate from the XML that specifies the layout.

The payment form provides default styles.

- *Widget.PaymentForm.Label*
- *Widget.PaymentForm.EditText*
- *Widget.PaymentForm.ValidatorHint*

The payment form provides list of attributes you can customize.

Attribute Name	**Default Value**	**Description**
paymentFormFontName	sans-serif	The font name of the whole payment form.
paymentFormShowCVVHint	false	If set to true then the credit card form will display a hint on where the CVV is located.
paymentFormLabelStyle	*Widget.PaymentForm.Label*	The style of a label.
paymentFormEditTextStyle	*Widget.PaymentForm.EditText*	The default style of an edit text.
paymentFormCardNumberEditTextStyle	paymentFormEditTextStyle	The style of the card number edit text.
paymentFormExpiryDateEditTextStyle	paymentFormEditTextStyle	The style of the expiry date edit text.

paymentFormCardHolderEditTextStyle	paymentFormEditTextStyle	The style of the card holder edit text.
paymentFormCvvEditTextStyle	paymentFormEditTextStyle	The style of the cvv edit text.
paymentFormAccountHolderEditTextStyle	paymentFormEditTextStyle	The style of the account holder edit text.
paymentFormIbanEditTextStyle	paymentFormEditTextStyle	The style of the IBAN edit text.
paymentFormValidationHintStyle	*Widget.PaymentForm.ValidatorHint*	The style of a validation hint.

To override predefined style, add a new style inherited from the default one and override some attribute values in your styles XML file (usually the /res/values/styles.xml).

```
<resources>
    <!-- ... -->


    <style name="CustomPaymentForm.EditText"
parent="Widget.PaymentForm.EditText">
        <item name="borderBottomSize">1dp</item>
        <item name="borderBottomColor">@color/colorPrimary</item>
        <item name="isPlaceholderVisible">false</item>
    </style>
</resources>
```

Then, apply new custom style in the application theme using corresponding theme attribute.

```
<resources>
    <!-- ... -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- ... -->

        <item
name="paymentFormEditTextStyle">@style/CustomPaymentForm.EditText</item>
    </style>
</resources>
```

Here is the complete example of the UI customization.

```
<resources>
    <!-- Application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- ... -->


        <item name="paymentFormFontName">Times New Roman</item>
        <item
name="paymentFormLabelStyle">@style/CustomPaymentForm.Label</item>
        <item
name="paymentFormEditTextStyle">@style/CustomPaymentForm.EditText</item>
        <item
name="paymentFormExpiryDateEditTextStyle">@style/CustomPaymentForm.EditText.
ExpiryDate</item>
        <item
name="paymentFormValidationHintStyle">@style/CustomPaymentForm.ValidatorHint
</item>
    </style>


    <style name="CustomPaymentForm.Label" parent="Widget.PaymentForm.Label">
        <item name="android:textSize">12sp</item>
        <item name="android:textStyle">bold</item>
    </style>
    <style name="CustomPaymentForm.EditText"
parent="Widget.PaymentForm.EditText">
        <item name="borderBottomSize">1dp</item>
        <item name="borderBottomColor">@color/colorPrimary</item>
        <item name="isPlaceholderVisible">false</item>
    </style>
    <style name="CustomPaymentForm.EditText.ExpiryDate"
parent="CustomPaymentForm.EditText">
        <item name="isPlaceholderVisible">true</item>
    </style>
    <style name="CustomPaymentForm.ValidatorHint"
parent="Widget.PaymentForm.ValidatorHint">
        <item name="android:textColor">@color/colorAccent</item>
        <item name="android:textStyle">italic</item>
    </style>
</resources>
```

1.1.1.3. Payment form wizard

The payment form can be shown in a wizard format. I.e. you can set one specific field visible at the current step, validate it, and then show the next one.

Example of usage

```
PaymentForm paymentForm = (PaymentForm)findViewById(R.id.payment_form);
PaymentForm.FormElement activeElement = PaymentForm.FormElement.cardNumber;

// Set specific payment form field visible
paymentForm.setElementVisible(PaymentForm.FormElement.all, false);
paymentForm.setElementVisible(activeElement, true);

// Specify validation callback
paymentForm.setOnValidationCallback(new PaymentForm.OnValidationCallback() {
    @Override
    public void onValidateElement(PaymentForm.FormElement element, boolean
isValid) {
        if (isValid) {
// go to next wizard step
        }
    }
});

// Validate visible payment form element when some button tapped
// If the last visible element validated then submit payment form
paymentForm.validateElement(activeElement);
if (activeElement == PaymentForm.FormElement.cvv) {
    paymentForm.submit();
}
```

1.1.1.4. Accept the Direct Debit Authorization

To add Direct Debit authorization support implement the onBeforeSubmit handler of the *PaymentForm.OnSubmitCallback*

1. Check if Bank Account is selected and direct debit authorization is not accepted yet.
2. Prevent form from being submitted.
3. Show confirmation dialog.
4. Re-submit form if the user presses the positive button.

See full example below

```
private boolean isDirectDebitAuthorizationAccepted = false;

@Override
public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);

    // PaymentForm initialization

    paymentForm.setOnSubmitCallback(new PaymentForm.OnSubmitCallback() {
        @Override
        public void onBeforeSubmit(OnBeforeSubmitArgs args) {
            // Prevent form from being submitted if Bank Account is selected
and direct debit authorization is not accepted yet
            if(!isDirectDebitAuthorizationAccepted &&
args.getPaymentOptionCode().equals(Lookups.PaymentOptionCode.BankAccount)){
                // Preventing form from being submitted
                args.preventSubmit();

                // Extracting user input
                String accountHolder = args.getData().get("AccountHolder");
                String iban = args.getData().get("Iban");

                // Constructing confirmation dialog
                new AlertDialog.Builder(getActivity())
                        .setTitle("Accept the direct debit authorization")
                        .setMessage(String.format("Please accept the direct
debit authorization below to let us automatically pull funds from your bank
account.\n\n" +
                                "Account Holder: %s\n" +
                                "IBAN: %s", accountHolder, iban))
                        .setPositiveButton("Accept", new
DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int
which) {
                                // Re-submit form after direct debit
authorization is accepted
                                isDirectDebitAuthorizationAccepted = true;
                                paymentForm.submit();
                            }
                        })
                        .setNegativeButton("Cancel", null)
                        .create()
                        .show();

                return;
            }

            // Disable Submit button and show loading indicator
        }

        // ...
    });
}
```

1.1.3. Complete registration/payment process

To complete registration/payment process your app should call the following methods of the KontoCloud API.

Payment Form Mode	**API methods**
REGISTRATION	Complete Add Stored Payment Option
PAYMENT	Complete Authorize Capture

In case of your app is using KontoCloud SDK API Services see Complete add stored payment option.

1.1.4. More examples

1.1.4.1. Common use case for any payment provider

```
PaymentForm paymentForm = (PaymentForm) findViewById(R.id.payment_form);

// Set the appropriate form mode. PaymentFormMode.REGISTRATION for register payment
option or PaymentFormMode.PAYMENT for making a payment
paymentForm.getOptions().setMode(PaymentFormMode.PAYMENT);


// Set the appropriate form provider mode.
paymentForm.getOptions().setPaymentProviderMode(PaymentProviderMode.TEST);

// Pass the same redirect URL as used in the API method "Init Authorize". This option
should be set for PaymentOS, CyberSource, CyberSource with TokenEX payment providers.
paymentForm.getOptions().setRedirectUrl(redirectUrl_Cyber);

// Set KontoCloud API URL for payment form
paymentForm.getOptions().setApiUrl(Constants.API_URL);

// Set current payment provider.
paymentForm.getOptions().setPaymentProvider(currentPaymentProvider);

// Attach callbacks
paymentForm.setOnSubmitCallback(new PaymentForm.OnSubmitCallback() {
    @Override
    public void onSuccess(String paymentOptionCode, String authorizationToken, String
providerResponse) {
        // Complete registration/payment process (see 2.1.3)
    }
    @Override
    public void onError(String errorMessage) {
        // Handle error. For example, close payment form and show Toast.
    }

    @Override
    public void onCancel() {
        // Handle cancellation from paypal
    }
});

// Render payment form with specified payment options
paymentForm.render(paymentOptionCodes, authorizationToken);
```

2.1.4.1 PayPal direct integration

Pass redirect URL and cancel URL to the Payment Form and also set the specified payment provider

```
PaymentForm paymentForm = (PaymentForm) findViewById(R.id.payment_form);

// Initialize payment form
paymentForm.getOptions().setMode(PaymentFormMode.PAYMENT);
paymentForm.getOptions().setPaymentProviderMode(PaymentProviderMode.TEST);


// Set KontoCloud API URL for payment form
paymentForm.getOptions().setApiUrl(Constants.API_URL);

// Set the appropriate payment provider
paymentForm.getOptions().setPaymentProvider(Lookups.PaymentOptionProvider.PayPal);

// Attach callbacks
paymentForm.setOnSubmitCallback(new PaymentForm.OnSubmitCallback() {
    @Override
    public void onSuccess(String paymentOptionCode, String authorizationToken, String
payerID) {
        // Complete registration/payment process (see 2.1.3)
    }

    @Override
    public void onCancel() {
        // Handle cancellation from paypal
    }

    @Override
    public void onError(String errorMessage) {
        // Handle error. For example, close payment form and show Toast.
    }
});

// Render payment form
paymentForm.render(Lookups.PaymentOptionCode.PayPal, authorizationToken);
```

1.2. API Services

1.2.1. Initialization

To initialize KontoCloud SDK create an instance of *ApiProvider* with specific parameters (*ApiProviderOptions*).

```
final DefaultAccessTokenProvider accessTokenProvider = new
DefaultAccessTokenProvider();

// Configure API provider
ApiProviderOptions apiProviderOptions = new ApiProviderOptions() {{
    setUrl("{api url}");
    setProgramCode("{program code}");
    setAccessTokenProvider(accessTokenProvider);
}};

ApiProvider apiProvider = new ApiProvider(context, apiProviderOptions);
```

1.1.2. Call anonymous API method

To call API method get an instance of API service using *ApiProvider*.

List of all API services described *on page 21*.

```
// Getting instance of API client
UserApi userApi = apiProvider.getUserApi();

// Call API method
CheckUserUniquenessResponse checkUserUniquenessResponse =
userApi.checkUserUniqueness("{new user id}", new
CheckUserUniquenessRequest()).execute().body();
Boolean isUserFound = checkUserUniquenessResponse.getFound();
```

1.1.3. Authentication

To add authentication support to your app

- Implement your own *AccessTokenProvider*
- Use *DefaultAccessTokenProvider* included in the KontoCloud SDK

Then, pass an instance of *AccessTokenProvider* to the *ApiProviderOptions* during *Initialization*.

API service requests *AccessTokenProvider* for an access token each time an API method is called.

Access token and refresh token (*AccessTokenResponse*) can be obtained through the

- AuthenticationApi.getAccessToken using user id and password
- AuthenticationApi.getRefreshToken using refresh token

Access tokens have a limited lifetime determined by the specified session timeout of the KontoCloud API endpoint. If an application uses an expired access token, a HTTP error 401 is returned.

To obtain a new access token without the user id and password (when access token is expiring or expired) use refresh token.

Refresh tokens are subject to strict storage requirements to ensure that they are not leaked.

Detailed authentication process described *below*.

1.1.3.1. Authenticate user

Call authenticationApi.getAccessToken(grant_type, user_id, password) to retrieve *AccessTokenResponse*.

Then, configure *AccessTokenProvider* to return access token.

```
// Getting instance of authentication API service
AuthenticationApi authenticationApi = apiProvider.getAuthenticationApi();

// Retrieving access token
AccessTokenResponse accessTokenResponse =
authenticationApi.getAccessToken("password", "{user id}", "{user
password}").execute().body();

// Authenticating user
accessTokenProvider.setAccessToken(accessTokenResponse.getAccessToken());

// Store token details
String refreshToken = accessTokenResponse.getRefreshToken();
LocalDateTime accessTokenExpireDate =
LocalDateTime.now().plusSeconds(accessTokenResponse.getExpiresIn());
```

1.1.1.2. Refresh user access token

Call authenticationApi.getRefreshToken(grant_type, refresh_token) to retrieve new *AccessTokenResponse*.

Then, configure *AccessTokenProvider* to return new access token.

```
// Getting instance of authentication API service
AuthenticationApi authenticationApi = apiProvider.getAuthenticationApi();


// Assume that the user is already authenticated

boolean isTokenExpired = LocalDateTime.now().isAfter(accessTokenExpireDate);
if(isTokenExpired){
    // Retrieving refresh token
    AccessTokenResponse refreshTokenResponse =
authenticationApi.getRefreshToken("refresh_token",
refreshToken).execute().body();

    // Refreshing user token
    String accessToken = refreshTokenResponse.getAccessToken();
    accessTokenProvider.setAccessToken(accessToken);

    // Refresh stored token details
    refreshToken = refreshTokenResponse.getRefreshToken();
    accessTokenExpireDate =
LocalDateTime.now().plusSeconds(refreshTokenResponse.getExpiresIn());
}
```

1.1.4. More examples

1.1.4.1. Find account number

```
// Getting instance of API service
AccountApi accountApi = apiProvider.getAccountApi();

// Assume that the user is already authenticated

// Getting account number
FindUserAccountResponse response = accountApi.findUserAccount(new
FindUserAccountRequest("{user id}")).execute().body();
String accountNumber = response.getAccno();
```

1.1.4.2.

1.1.1.2. Get account information

```
// Getting instance of API service
AccountApi accountApi = apiProvider.getAccountApi();

// Assume that the user is already authenticated

// Getting account information for authenticated user
GetAccountInformationResponse response =
accountApi.getAccountInformation("{account number}", new
GetAccountInformationRequest(Lookups.AccountNumberType.KontoCloud)).execute(
).body();
String firstName = response.getFirstName();
String lastName = response.getLastName();
```

1.1.1.3.

1.1.1.3. Get payment information

```
// Getting instance of API service
PaymentApi paymentApi = apiProvider.getPaymentApi();

// Getting payment information by unique reference
GetPaymentInformationResponse response =
paymentApi.getPaymentInformation("{unique reference}", new
GetPaymentInformationRequest()).execute().body();
String customerFullName = response.getCustomerFullName();
```

1.1.1.4.

1.1.1.4. Send password reset code

```
// Getting instance of API service
UserApi userApi = apiProvider.getUserApi();

userApi.sendPasswordResetCode("{user id}", new
SendPasswordResetCodeRequest()).execute();
```

1.1.1.5.

1.1.1.5. Init add stored payment option

```
// Getting instance of API service
AccountApi accountApi = getApiProvider().getAccountApi();

InitAddStoredPaymentOptionRequest request = new
InitAddStoredPaymentOptionRequest(Lookups.AccountNumberType.KontoCloud,
paymentOptionCode);

InitAddStoredPaymentOptionResponse response =
accountApi.initAddStoredPaymentOption(accountNumber,
request).execute().body();


String authorizationToken = response.getAuthorizationToken();
```

1.1.1.6.

1.1.1.6. Complete add stored payment option

```
// Getting instance of API service
AccountApi accountApi = getApiProvider().getAccountApi();

CompleteAddStoredPaymentOptionRequest request = new
CompleteAddStoredPaymentOptionRequest();
request.setAccnoType(Lookups.AccountNumberType.KontoCloud);
request.setPaymentOptionCode(paymentOptionCode);
request.setAuthorizationToken(authorizationToken);
request.setUseDifferentBillingAddress(false);

CompleteAddStoredPaymentOptionResponse response =
accountApi.completeAddStoredPaymentOption(appContext.getAccountNumber(),
completeAddStoredPaymentOptionRequest).execute().body();

String storedPaymentOptionReference = response.getStoredPaymentOptionReference();
```

1.1.1.7.

1.1.5. Push notifications

KontoCloud push notifications are delivered to the device using Firebase Cloud Messaging.

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages.

1.1.5.1. Add Firebase and FCM SDK to your app

To add Firebase to your app you'll need a Firebase project and a Firebase configuration file for your app.

1. Create a Firebase project in the Firebase console, if you don't already have one. If you already have an existing Google project associated with your mobile app, click Import Google Project. Otherwise, click Create New Project.
2. Click Add Firebase to your Android app and follow the setup steps. If you're importing an existing Google project, this may happen automatically and you can just download the config file.
3. When prompted, enter your app's package name. It's important to enter the package name your app is using; this can only be set when you add an app to your Firebase project.
4. At the end, you'll download a *google-services.json* file.

5. If you haven't done so already, copy this into your project's module folder, typically *app/*.
6. Please communicate the Firebase server key to Customer Support or your Administrator to be set up in the E-Wallet Platform back office for the corresponding program. The server key is available in the Cloud Messaging tab of the Firebase console Settings pane.

To integrate the Firebase libraries into one of your own projects, you need to perform a few basic tasks to prepare your Android Studio project. You may have already done this as part of adding Firebase to your app.

First, add rules to your root-level *build.gradle* file, to include the google-services plugin:

```
buildscript {
    // ...
    dependencies {
        // ...
        classpath 'com.google.gms:google-services:3.0.0'
    }
}
```

Then, in your module Gradle file (usually the *app/build.gradle*), add the apply plugin line at the bottom of the file to enable the Gradle plugin:

```
apply plugin: 'com.android.application'

android {
  // ...
}

dependencies {
  // ...
  compile 'com.google.firebase:firebase-core:9.8.0'
  compile 'com.google.firebase:firebase-messaging:9.8.0'
}

// ADD THIS AT THE BOTTOM
apply plugin: 'com.google.gms.google-services'
```

Learn more about Firebase
https://firebase.google.com/docs/android/setup

1.1.1.2. Register account device

To start receiving messages you should register the Firebase token from the KontoCloud API.

```
// Retrieve the current registration token
String firebaseToken = FirebaseInstanceId.getInstance().getToken();

// Get instance of API client
AccountApi accountApi = apiProvider.getAccountApi();

// Assume that the user is already authenticated

// Call register account device API method to associate the device for
notifications with an account.
accountApi.registerAccountDevice(accountNumber, new
RegisterAccountDeviceRequest(Lookups.AccountNumberType.KontoCloud,
Lookups.DeviceType.Android, firebaseToken)).execute();
```

1.1.1.3. Unregister account device

To stop receiving messages you should unregister the Firebase token from the KontoCloud API.

```
// Retrieve the current registration token
String firebaseToken = FirebaseInstanceId.getInstance().getToken();

// Get instance of API client
AccountApi accountApi = apiProvider.getAccountApi();

// Assume that the user is already authenticated

// Call unregister account device API method to remove the device from
notifications associated with an account.
accountApi.unregisterAccountDevice(accountNumber, new
UnregisterAccountDeviceRequest(Lookups.AccountNumberType.KontoCloud,
Lookups.DeviceType.Android, firebaseToken)).execute();
```

1.1.1.4. Receive messages

To receive messages, use a service that extends FirebaseMessagingService. Your service should override the onMessageReceived callback.

To use FirebaseMessagingService, you need to add the following in your app manifest:

```
<service
    android:name=".SampleFirebaseMessagingService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>
```

By overriding the method FirebaseMessagingService.onMessageReceived, you can show notifications on device:

```
public class SampleFirebaseMessagingService extends FirebaseMessagingService
{
    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        // Check if message contains a data payload.
        if (remoteMessage.getData().size() > 0) {
            Map<String, String> data = remoteMessage.getData();

            // Extract message parameters
            String accountNumber = data.get("accno");
            String notificationMessageCode =
data.get("notificationMessageCode");
            String amount = data.get("amount");
            String currencyCode = data.get("currencyCode");
            String ownerName = data.get("ownerName");

            // Construct notification message
            String notificationMessage = String.format("%1$s received
notification of type %2$s. amount: %3$s %4$s, owner: %5$s.", accountNumber,
notificationMessageCode, amount, currencyCode, ownerName);

            // Build and show notification
            NotificationCompat.Builder notificationBuilder =
                    new NotificationCompat.Builder(this)
                            .setSmallIcon(R.drawable.notification_icon)
                            .setContentTitle("Notification title")
                            .setContentText(notificationMessage);
            NotificationManagerCompat notificationManager =
NotificationManagerCompat.from(this);
            notificationManager.notify(0 /* ID of notification */,
notificationBuilder.build());
        }
    }
}
```

Learn more about Firebase Cloud Messaging
https://firebase.google.com/docs/cloud-messaging/

1.1.1.4.1. RemoteMessage data parameters

Field	**Description**
accno	Recipient account number. For example you can show notifications for logged in user only (notification accno should be equal to current user account number)
uniqueReference	Unique transaction reference assigned by KontoCloud. Can be useful to determine if the transaction is initiated on the device or not. You may want to hide notifications for transactions initiated on the device
notificationMessageCode	Message code Can be useful for constructing localized messages. List of all supported notification message codes described *below*
amount	Message amount
currencyCode	Message currency code
ownerName	Message owner name

1.1.1.4.2.

1.1.1.1.2. Notification message codes

Push notifications do not contain ready to use localized message string.

To construct a localized message, use the notificationMessageCode parameter of the RemoteMessage object.

Here is a list of all supported notification message codes:

Notification Message Code	**Example Message Template**
LDACCT-Credit	Funds loaded successfully: {amount} {currencyCode}.
ULDACCT-Debit	Funds withdrawn successfully: {amount} {currencyCode}.
KC-CPTR-Debit	You've paid {amount} {currencyCode} to {ownerName}.
KC-RFND-Credit	You received a refund of {amount} {currencyCode} from {ownerName}.
KC-RFND-Debit	You sent a chargeback of {amount} {currencyCode} to {ownerName}.
KC-CHBK-Debit	You sent a chargeback of {amount} {currencyCode} to {ownerName}.
TFRAMNT-Credit	You received {amount} {currencyCode} from {ownerName}.
TFRAMNT-Debit	You sent {amount} {currencyCode} to {ownerName}.
AJTBAL-Credit	Your balance was adjusted by +{amount} {currencyCode}.
AJTBAL-Debit	Your balance was adjusted by -{amount} {currencyCode}.

3. Class reference

3.1. Payment Form

3.1.1. PaymentForm

com.contoworks.kontocloud.uicomponents.widget.PaymentForm

Member	**Type**	**Description**
setOptions(*PaymentFormOptions* options)	void	Sets payment form options.
getOptions()	*PaymentFormOptions*	Gets payment form options.
setOnSubmitCallback(*PaymentForm.OnSubmitCallback* callback)	void	Attaches payment form submit callbacks.
render(String[] paymentOptionCodes, String authorizationToken)	void	Renders payment form for specified parameters. Supported payment option codes: "*VISA*" – Visa "*MSTRCRD*" – MasterCard If the *paymentOptionCodes* parameter contains more than one credit card code (e.g. { "VISA", "MSTRCRD" }) then payment form detects the payment option automatically based on the first four digits entered in the credit card number field.
render(String paymentOptionCode, String authorizationToken)	void	Renders payment form for specified parameters. Supported paymentOptionCode values: "*BNKACCT*" – Bank Account "*VISA*" – Visa "*MSTRCRD*" – MasterCard "*PAYPAL*" – PayPal "*PAYU*" – PayU
submit()	void	Validates and submits the payment form to the server. If the validation fails, the submit does not perform. Supported for payment options: Bank Account Visa MasterCard

setElementVisible(PaymentForm.FormElement element, boolean visible)	void	Set specific form field visibility. Supported fields: “all” – to set all elements visible/invisible “cardNumber” – card number field “cardHolder” – card holder field “expiry” – card expiration field “cvv” – cvv field
validateElement(PaymentForm.FormElement element)	void	Validate specific form field.
setOnValidationCallback(OnValidationCallback onValidationCallback)	void	Attaches payment form field validation callback.

1.1.2. PaymentForm.OnSubmitCallback

com.contoworks.kontocloud.uicomponents.widget.PaymentForm.OnSubmitCallback

Member	**Type**	**Description**
onBeforeSubmit(OnBeforeSubmitArgs args)	void	Invoked when a payment form has been validated and is going to be submitted.
onSuccess(String paymentOptionCode, String authorizationToken)	void	Invoked when a payment form has been successfully submitted.
onSuccess(String paymentOptionCode, String authorizationToken, String additionalParam)	void	Invoked when a payment form has been successfully submitted. Used in PayPal direct integration to get additional parameter – payerID. Used in CyberSource to get paymentProviderResponse.
onCancel()	void	Used in paypal direct integration to cancel current form.
onError(String errorMessage)	void	Invoked when there has been an error during a payment form submission.

1.1.3. PaymentForm.OnValidationCallback

com.contoworks.kontocloud.uicomponents.widget.PaymentForm.OnValidationCallback

Member	**Type**	**Description**

onValidateElement(PaymentForm.FormElement element, boolean isValid)	void	Invoked when a specific payment form field has been validated.

1.1.4. PaymentForm.OnSubmitCallback.OnBeforeSubmitArgs

com.contoworks.kontocloud.uicomponents.widget.PaymentForm.OnSubmitCallback.OnBeforeSubmitArgs

Member	**Type**	**Description**
getPaymentOptionCode()	String	Payment Form payment option code.
getData()	Map<String, String>	Payment Form additional user input data. Use this method to extract user input programmatically. Supported keys for different payment options: *1.* Bank Account*:* ● *AccountHolder* ● *Iban* Example: *args.getData().get("AccountHolder");*
preventSubmit()	void	Prevents form from being submitted.

1.1.5. PaymentFormMode

com.contoworks.kontocloud.uicomponents.PaymentFormMode

Member	**Type**	**Description**
REGISTRATION	int	Payment option registration mode. Use this mode to add a stored payment option to an e-wallet account.
PAYMENT	int	Payment mode. Use this mode to do a payment to an e-wallet account.

1.1.6. FormElement

com.contoworks.kontocloud.uicomponents.widget.PaymentForm.FormElement

Member	**Type**	**Description**

all	String	Used to set all payment form elements visible/unvisible
cardNumber	String	Payment form card number field
cardholder	String	Payment form card holder field
expiry	String	Payment form card expiration field
cvv	String	Payment form card cvv field

1.1.7. PaymentOptionProvider

com.contoworks.kontocloud.apiclient.Lookups.PaymentOptionProvider

Member	**Type**	**Description**
Payon	String	Payon – payment option provider. Supported payment options: VISA, MSTRCRD, BNKACCT, PAYPAL
PaymentOS	String	PaymentOS - payment option provider. Supported payment options: PAYU
PayPal	String	PayPal – payment option provider. Supported payment options: PAYPAL
CyberSource	String	CyberSource – payment option provider. Supported payment options: VISA, MSTRCRD, AMEX, MSTRO, DISCOVER
CyberSourceWithTokenEx	String	CyberSourceWithTokenEx – payment option provider. Supported payment options: VISA, MSTRCRD, AMEX, MSTRO, DISCOVER
VestaWithTokenEx	String	VestaWithTokenEx – payment option provider. Supported payment options: VISA, MSTRCRD, AMEX, MSTRO, DISCOVER
Sepa	String	Sepa - payment option provider. Supported payment options: BNKACCT
PayonWithPCIProxy	String	PayonWithPCIProxy – payment option provider. Supported payment options: VISA, MSTRCRD, AMEX, MSTRO

1.1.8. PaymentFormOptions

com.contoworks.kontocloud.uicomponents.widget.PaymentFormOptions

Member	**Type**	**Description**
setMode(*PaymentFormMode* mode)	void	Sets payment form mode.
setPaymentProviderMode(*PaymentProviderMode* mode)	void	Sets payment provider mode.
setShowStorePaymentMethod(Boolean value)	void	Set this value to true to add a checkbox to the payment form to let the customer decide whether or not to store the card details. Supported in the PAYMENT mode only.
setRedirectUrl(String redirectUrl)	void	The same redirect URL as used in the API method "Init Authorize"
setApiUrl(String apiUrl)	void	KontoCloud API URL
setPaymentProvider(String paymentProvider)	void	Sets payment provider. Supported values: Lookups.PaymentOptionProvider.Payon, Lookups.PaymentOptionProvider.PayPal. If is not set the default value will be Lookups.PaymentOptionProvider.Payon.

1.1.9. PaymentProviderMode

com.contoworks.kontocloud.uicomponents.PaymentProviderMode

Member	**Type**	**Description**
TEST	int	Payment provider test mode. Use this mode for development and testing.
LIVE	int	Payment provider live mode. Use this mode for production.

1.1.10. BuildConfig

com.contoworks.kontocloud.uicomponents.BuildConfig

Member	**Type**	**Description**

VERSION_NAME	String	Current Payment Form version

1.1.11. Styles

This section contains the list of available styles, attributes supported by them and their predefined values.

1.1.11.1.Widget.PaymentForm.Label

Attribute Name	**Type**	**Value**
android:textSize	dimension	12sp
android:textColor	color	#B3B3B3
android:textStyle	flag	normal

1.1.11.2.

1.1.1.2. Widget.PaymentForm.EditText

Attribute Name	**Type**	**Value**
android:textSize	dimension	14sp
android:textColor	color	#343434
android:textStyle	flag	normal
borderLeftColor	color	#919191
borderLeftSize	dimension	0dp
borderTopColor	color	#919191
borderTopSize	dimension	0dp
borderRightColor	color	#919191
borderRightSize	dimension	0dp
borderBottomColor	color	#919191
borderBottomSize	dimension	1dp
isPlaceholderVisible	boolean	true

1.1.1.3.

1.1.1.3. Widget.PaymentForm.ValidatorHint

Attribute Name	**Type**	**Value**
android:textSize	dimension	12sp
android:textColor	color	#D50000
android:textStyle	flag	normal

1.1.1.4.

1.2. API Services

1.2.1. AccessTokenResponse

com.contoworks.kontocloud.apiclient.api.model.AccessTokenResponse

Member	**Type**	**Description**
accessToken	String	Access token. Provide this token with *AccessTokenProvider* to authenticate requests. See *Authenticate* user
expiresIn	Integer	Access token expires in seconds.
refreshToken	String	Refresh token. Special kind of token that is used to authenticate a user without them needing to re-authenticate. See *Refresh user access token*

1.1.2. AccessTokenProvider

com.contoworks.kontocloud.sdk.AccessTokenProvider

Member	**Type**	**Description**
getAccessToken()	String	Returns access token for request. If accessToken is null – API requests will be anonymous.

1.1.3. ApiProvider

com.contoworks.kontocloud.sdk.ApiProvider

Member	Type	Description
getAccountApi()	AccountApi	Returns account API service
getAuthenticationApi()	AuthenticationApi	Returns authentication API service
getPaymentApi()	PaymentApi	Returns payment API service
getSettingsApi()	SettingsApi	Returns settings API service
getUserApi()	UserApi	Returns user API service
getPaypalTrackingID(Context)	String	Returns riskID which is used for AuthorizeRequest in paypal direct integration to execute payment with saved payment option

1.1.4. ApiProviderOptions

com.contoworks.kontocloud.sdk.ApiProviderOptions

Member	Type	Description
accessTokenProvider	*AccessTokenProvider*	An instance of the *AccessTokenProvider*. If accessTokenProvider is null all requests are anonymous. Default implementation: *DefaultAccessTokenProvider*
partnerReferencePrefix	String	Partner reference prefix for all requests.
programCode	String	Program code
url	String	KontoCloud API URL.

1.1.5. DefaultAccessTokenProvider

com.contoworks.kontocloud.sdk.DefaultAccessTokenProvider

Member	Type	Description
getAccessToken()	String	Returns current access token.
setAccessToken(String accessToken)	void	Sets current access token.

1.1.6. BuildConfig

com.contoworks.kontocloud.sdk.BuildConfig

Member	**Type**	**Description**
VERSION_NAME	String	Current SDK version