PayPal connect api integration to commercetools Frontend

Disclaimer

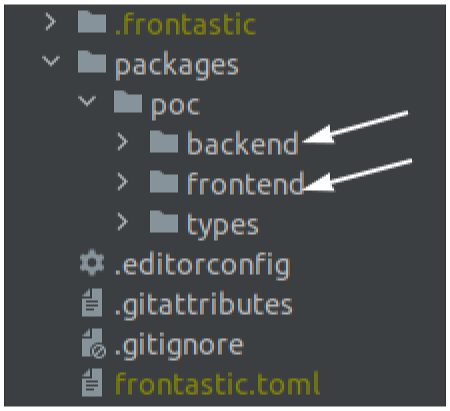
This project is based on the following SDK

- PayPal JavaScript SDK
- commercetools TypeScript SDK

and Frontastic commercetools Frontend with reference to commercetools HTTP API. PayPal connect api is also dedicated to be used together with PayPal commercetolls Connector. This guide only shows some use cases and highlights the features relevant the integration. For full official documentation please use the links already provided.

Project structure

For the details of the Frontastic commercetools Frontend project structure please check the commercetools frontend guide.
In brief, frontastic projects (in our case - poc for proof of concept) contain two folders relevant for integration - frontend and backend. For clarity when it goes about the folder, we will use names poc/frontend and poc/backend respectively.



Integration of the PayPal connect api involves both. poc/frontend, obviously, includes rendering of the components themselves. poc/backend is responsible for obtaining all the necessary data from the server and providing the data to the components.

Installation

It is only necessary to include paypal-commercetools-client to the poc/frontend package.json file, where the components are imported, and then use frontastic install as usual.

If your project is using a different frontend platform – the api can be installed from npm:

```
npm i paypal-commercetools-client
```

Integration

poc/frontend

All required payment methods can be imported directly from the installed package, for example:

```
import { PayPal, CardFields} from 'paypal-commercetools-client/dist/esm';
```

We set all the required parameters as the PayPalJson object, that can be submitted directly to a method (also added as a separate file without comments):

```
const PayPalJson = {
 getSettingsUrl: "URL/getPayPalSettings",
 createPaymentUrl: "URL/createPayment",
 createOrderUrl: "URL/createPayPalOrder",
 onApproveUrl: "URL/capturePayPalOrder",
 shippingMethodId: "", //Leave it empty if you do not use shippingMethodId or pass a valid ID
 requestHeader: {
  // this key and value object will be sent as header in the API calls
  KEY1: "VALUE1",
  KEY2: "VALUE2",
 },
 options: {
  clientId: "PAYPAL_CLIENT_ID",
  currency: "EUR",
 },
 fundingSource: "paypal",
 cartInformation: {
  account: {
   email: "CUSTOMER_EMAIL",
  },
  billing: {
   firstName: "CUSTOMER_FIRST_NAME",
   lastName: "CUSTOMER_LAST_NAME",
   streetName: "CUSTOMER_STREET_NAME",
   streetNumber: "CUSTOMER_STREET_NUMBER",
   city: "CUSTOMER_CITY",
   country: "CUSTOMER_COUNTRY_CODE",
   postalCode: "CUSTOMER_POSTAL_CODE",
  },
  shipping: {
   firstName: "CUSTOMER_FIRST_NAME",
   lastName: "CUSTOMER_LAST_NAME",
   streetName: "CUSTOMER_STREET_NAME",
   streetNumber: "CUSTOMER_STREET_NUMBER",
   city: "CUSTOMER_CITY",
   country: "CUSTOMER_COUNTRY_CODE",
   postalCode: "CUSTOMER_POSTAL_CODE",
  },
 },
 purchaseCallback: (result: any, options: any) => {
  // this function will be called after the payment is done
 },
 authenticateThreeDSOrderUrl: "URL/authenticateThreeDSOrder", //For CardFields and GooglePay component
 getClientTokenUrl: "URL/getClientToken", //For CardFields component
 getOrderUrl: "URL/getPayPalOrder", //Optional
 authorizeOrderUrl: "URL/authorizePayPalOrder", // For Vaulting
}
```

Finally, the component is rendered as follows:

```
PayPal: <PayPal {...PayPalJson} />,
```

In case if some error occurs, for example because wrong parameters are submitted – you can see the error description in the browser console.

A brief description of all the required parameters is already provided at npm page of the app. So, in this guide we will focus on actual use cases and in case of url used to access the api – how the api works in our case.

Use cases

PayPalMessages

PayPal messages component is used to demonstrate that the shop supports PayPal and it is recommended to place it not only at the checkout, but, for example, at the start page or at the header. This is the simplest use case, that does not involve creating the order.

It has an additional group of parameters: payPalMessagesParams, that are specific only for this component.

```
<PayPalMessages
  {...PayPalJson}
  {...payPalMessagesParams}
  options={{ ...PayPalJson.options, components: 'messages' }}
/>
```

payPalMessagesParams include

```
placement: 'product' | 'home' | 'category' | 'cart' | 'payment',
amount: number | string,
currency: 'EUR' | 'USD',
layout: 'text' | 'flex' | 'custom',
```

By default, we set the amount to 0 (then in the languages that support this option payment amount will not be shown), euro, text and product.

PayPal

Technically <PayPal /> component is used to render most of the payment methods, just the displayed components list changes. Here we will focus on rendering the following set of components: PayPal buy now button, PayPal pay later button and benefits message for pay later feature. We will also mention the flow for the vaulted account. Here we show the example for the cart checkout page of our demo website to show address changing on the PayPal side workflow as well.

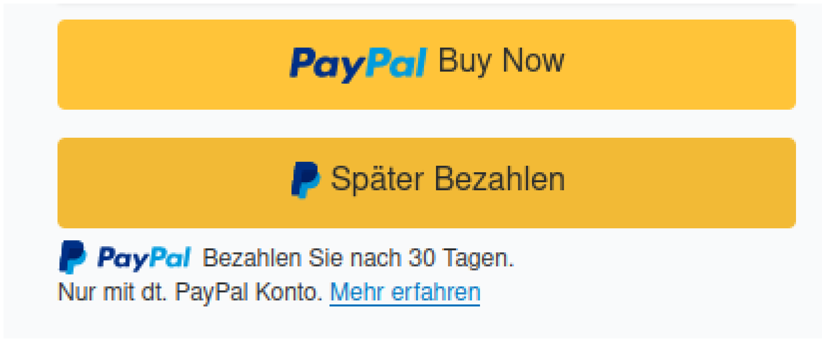
The component is displayed with the following additional options:

```
commit: false,
enableFunding: acceptPayLater ? 'paylater' : '',
disableFunding: ['sepa', 'card'],
components: 'messages,buttons',
```

commit set to false will force the purchase to be completed on the merchant side instead of PayPal. In this workflow it should be use together with

```
onApproveRedirectionUrl
```

that determines at which page the checkout actually happens. acceptPaylater is the setting that comes from getSettingsUrl (already mentioned in PayPalMessages). The set of enabled and disabled funding options provide the PayPal payment buttons, and messages listed in components show the pay later benefits.



PayPal Bezahlen Sie nach 30 Tagen.
Nur mit dt. PayPal Konto. Mehr erfahren

In cart checkout, where address (and shipping) is set in advance and can't be changed at the PayPal side. Here we can accept the address set at the PayPal side, by using the

`onShippingChange`

callback, that accepts the address data and actions (see PayPal documentation for details). In our case the callback performs the following actions:

1. Find the shipping option matching to the new address, if there is no matching option – return reject action.
2. Update cart – set the new address
3. Update cart – set the new shipping method (please note – it is impossible to set shipping method at commercetools backend side if the address is empty).
4. Call the `updateOrderOnShippingChange` endpoint at the poc/backend to update the payment (due to a new delivery price).
5. Return resolve on success

The workflow of the payment method is the following:

1. getSetting endpoint is called in advance before rendering the component, to set the necessary parameters, such as if the website accepts pay later.
2. createPayment is called on rendering the button
3. createPayPalOrder endpoint is called on click on the button and PayPal payment floating window opens.
4. In the PayPal floating window if relevant onShippingChange callback is trigerred
5. In the PayPal floating window after order is approved redirection to onApproveRedirectionUrl is triggered
6. After order is confirmed in the shop capturePayPalOrder enpoint is called.
7. purchaceCallback is called to trigger the checkout and redirect to thank you page.

poc/backend

All the methods build at the backend are shown in detail at the github repository and utilize the commercetools TypeScript SDK. Here we will only mention which HTTP API endpoints are called to get the data and how the data are formatted for further processing. Additionally, we would like to mention, that for clarity we separated the methods to SettingsController and PayPalController.

Url endpoint	List of performed calls
getSettings	Get CustomObject by container and key where the container is the connector name (for us – "paypal-commercetools-connector") and the key is "settings". The response already contains all the necessary data for further processing.

createPayment	1. Update Cart according to the request data, for successful processing of any payment email, shipping address, billing address, line items and shipping method id should be sufficient.
	2. Check if relevant PayPal payment already exists, if not – add payment to cart with proper amountPlanned and the following paymentMethod data:
	`paymentMethodInfo: {`
	`  paymentInterface: "PayPal",`
	`  method: "",`
	`},`
	`paymentStatus: {`
	`  interfaceCode: "init",`
	`},`
	We also use in payment status the frontastic feature
	`interfaceText: payment.debug,`
	to add a custom message to the payment.
	3. Create proper response using payment id, payment version and cart data.

getClientToken	Set custom type and set custom field for payment. Payment Id and version should come in the request. For the type set:
	`type: {`
	`  key: 'paypal-payment-type',`
	`  typeId: 'type',`
	`},`
	for the field:
	`getClientTokenRequest: JSON.stringify({}),`
	Success response includes the requested token.

createPayPalOrder	1. Get cart and latest payment for this cart from commercetools backend. From it extract line items id and

	quantity, that are required for creating the order at the PayPal side. 2. Use the set custom field endpoint with action name “'createPayPalOrderRequest” and value formatted according to the target payment method.

capturePayPalOrder	1. Necessary data (cart, paymentId, orderId, which is PayPalOrderId) are fetched or obtained from the request body. If orderId is not provided in the request – it is obtained from custom fields of the payment.
	2. Payment is updated with an action:
	`{`
	`  action: 'setCustomField',`
	`  name: 'capturePayPalOrderRequest',`
	`  value: JSON.stringify({}),`
	`},`
	3. Cart is refetched.