# Integrate card payments with JS SDK for direct merchants

SDK Legacy ADVANCED Last updated: September 6th 2023, @ 12:43:42 pm



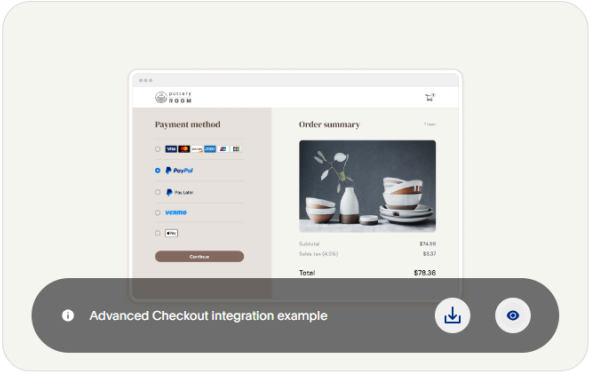
**Important:** This version of the JavaScript SDK integration guide for direct merchants is a legacy integration. Use **version 2** for new integrations.

## How it works

Advanced Checkout lets you offer the PayPal payment button and custom credit and debit card fields. This guide covers integrating the PayPal button, credit, and debit card payments. You can find more ways to extend your integration in our **customization doc**.

Once you integrate advanced Checkout, you can also offer options like Pay Later, Venmo, and alternative payment methods with some additional configuration.

This integration guide follows the code in this **GitHub sample**.




## Which integration to use

There are 2 versions of card payment integrations for direct merchants using the JavaScript SDK:

- **Recommended:** Version 2 uses the PayPal-hosted `CardFields` component to accept and save cards without handling card information. PayPal handles all security and compliance issues associated with processing cards. The `CardFields` component is the recommended integration method and receives ongoing enhancements and improvements.
- Version 1 is a legacy integration that uses the `HostedFields` component. This integration is no longer under active development and won't receive further updates.

Other elements of the JavaScript SDK integration for direct merchants remain the same.

GitHub Codespaces are cloud-based developer containers that are ready right away. Use GitHub Codespaces to code and test your PayPal integration.

Open in Codespaces

## Integration video

Watch our video tutorial for this integration:

### 2. Verify package.json

A `package.json` file has a list of the packages and version numbers needed for your app. You can share your `package.json` file with other developers so they can use the same settings as you.

The following snippet is an example of a `package.json` file for a PayPal integration. Compare this sample to the `package.json` in your project:

```
{
  "name": "@paypalcorp/advanced-integration",
  "version": "1.0.0",
  "description": "",
  "main": "YOUR-SERVER-NAME.js",
  "type": "module",
  "scripts": {
    "test": "echo "Error: no test specified" && exit 1",
    "start": "node YOUR-SERVER-NAME.js"
  },
  "author": "",
  "license": "Apache-2.0",
  "dependencies": {
    "dotenv": "^16.0.0",
    "ejs": "^3.1.6",
    "express": "^4.17.3",
    "node-fetch": "^3.2.1"
  }
}
```

- Pass the name of your server file using `main` by replacing the default `YOUR-SERVER-NAME.js` on lines 5 and 9.
- Use `scripts.test` and `scripts.start` to customize how your app starts up.

If you're having trouble with your app, try reinstalling your local library and package files using `npm install`.

If you're getting the node error below, you need to include `"type": "module"` in your `package.json` file. This line isn't automatically added when `package.json` is created:

`Warning: To load an ES module, set "type": "module" in the package.json or use the .mjs extension (Use `node --trace-warnings ...` to show where the warning was created)`

See line 6 of the sample `package.json` file for an example.

### 3. Set up .env

A `.env` file is a line-delimited text file that sets your local working environment variables. Use this `.env` file to securely pass the client ID and app secret for your app.

This code shows an example `.env` file. Rename `.env.example` to `.env` and replace the `PAYPAL_CLIENT_ID` and `PAYPAL_CLIENT_SECRET` with values from your app:

> **Note:** View your `PAYPAL_CLIENT_ID` and `PAYPAL_CLIENT_SECRET` in your **PayPal Developer Dashboard** under "Apps & Credentials".

```
PAYPAL_CLIENT_ID="YOUR_CLIENT_ID_GOES_HERE"
PAYPAL_CLIENT_SECRET="YOUR_SECRET_GOES_HERE"
```

## 2 Integrate back end

This section explains how to set up your back end to integrate advanced Checkout payments.

### Back-end process

1. Your app creates an order on the back-end by making a call to the **Create order** endpoint of the Orders v2 API.
2. Your app makes a call to the **Capture payment** endpoint of the Orders v2 API to move the money when the buyer confirms the order.

### Back-end code

The following example uses 2 files, `/server/server.js` and `/server/views/checkout.ejs`, to show how to set up the back end to integrate with advanced payments.

**/server/server.js**

The `/server/server.js` file provides methods and functions for making the API calls and handling errors, and creates the endpoints that your app uses to:

- Generate an access token to use PayPal APIs.
- Create an order.
- Capture payment for the order.
- Simulate errors for negative testing.

You'll need to save the `server.js` file in a folder named `/server`.

This is a breakdown of the `/server/server.js` code sample.

**Declare imports**

This section of code imports the dependencies for your Node.js application:

```
1  import express from "express";
2  import fetch from "node-fetch";
3  import "dotenv/config";
```

- Line 1 imports the Express.js framework to get started with creating a web server.
- Line 2 imports a `node-fetch` dependency needed to make `http` requests to PayPal REST APIs.
- Line 3 imports the dotenv library needed for loading sensitive data from environment variables, such as a PayPal client secret.

**Set up port and server**

This section of code collects the environment variables, sets up the port to run your server, and sets the base sandbox URL:

```
5  const { PAYPAL_CLIENT_ID, PAYPAL_CLIENT_SECRET, PORT = 8888 } = process.env;
6  const base = "https://api-m.sandbox.paypal.com";
```

Lines 7-10 start the express Node.js web application framework:

```
7  const app = express();
8  app.set("view engine", "ejs");
9  app.set("views", "./server/views");
10 app.use(express.static("client"));
```

Line 8 declares that the rendering engine uses Embedded JavaScript templates.

**Generate client token**

A client token uniquely identifies your payer. You need a client token for the payer so your app can use the hosted card fields. The following code sample creates a client token for you by making a `POST` call to the `/v1/identity/generate-token` endpoint:

```
42 /**
43  * Generate a client token for rendering the hosted card fields.
44  * See https://developer.paypal.com/docs/checkout/advanced/sdk/v1/#link-integratebackend
45  */
46 const generateClientToken = async () => {
47   const accessToken = await generateAccessToken();
48   const url = `${base}/v1/identity/generate-token`;
49   const response = await fetch(url, {
50     method: "POST",
51     headers: {
52       Authorization: `Bearer ${accessToken}`,
53       "Accept-Language": "en_US",
54       "Content-Type": "application/json",
55     },
56   });
57
58   return handleResponse(response);
59 };
```

This code runs your app on `localhost`, using the port `8888` that you set up in line 5 of the API call:

```
176 app.listen(PORT, () => {
177   console.log(`Node server listening at http://localhost:${PORT}/`);
178 });
```

**Render checkout page**

Set up your app's checkout page to load this section of code when it's rendered by your server:

```
140 // render checkout page with client id & unique client token
141 app.get("/", async (req, res) => {
142   try {
```

- Line 65 passes information from the `cart` into the `createOrder` function.
- Line 72 checks that you have the correct access token.
- Line 73 sets up the `url` for the API call.
- Lines 74-84 pass the `intent` and `purchase_units` to an object called `payload`. This gets passed to the API call.
- Lines 86-98 create an order by sending a `POST` request to the Create orders endpoint of the Orders v2 API, using `accessToken`, `url`, and the `payload` from lines 74-84.
- Line 97 converts the `payload` information into JSON format and passes it into the `body` section of the request.

**Negative testing for createOrder()**

This section of the `createOrder()` function provides 3 negative testing opportunities behind content tags:

- `MISSING_REQUIRED_PARAMETER`
- `PERMISSION_DENIED`
- `INTERNAL_SERVER_ERROR`

Remove the `//` from the beginning of a line to simulate that error. the `createOrder()` call will pass that line as a `PayPal-Mock-Response` header in the `POST` request to the Orders v2 API:

```
90 // Uncomment one of these to force an error for negative testing (in sandbox mode only). Documentation:
91 // https://developer.paypal.com/tools/sandbox/negative-testing/request-headers/
92 // "PayPal-Mock-Response": '{"mock_application_codes": "MISSING_REQUIRED_PARAMETER"}'
93 // "PayPal-Mock-Response": '{"mock_application_codes": "PERMISSION_DENIED"}'
94 // "PayPal-Mock-Response": '{"mock_application_codes": "INTERNAL_SERVER_ERROR"}'
```

**Run captureOrder()**

This section of code defines the `captureOrder()` function. Call the function to capture the order and move money from the payer's payment method to the seller:

```
103 /**
104  * Capture payment for the created order to complete the transaction.
105  * See https://developer.paypal.com/docs/api/orders/v2/#orders_capture
106  */
107 const captureOrder = async (orderID) => {
108   const accessToken = await generateAccessToken();
109   const url = `${base}/v2/checkout/orders/${orderID}/capture`;
110
111   const response = await fetch(url, {
112     method: "POST",
113     headers: {
114       "Content-Type": "application/json",
115       Authorization: `Bearer ${accessToken}`,
116       // Uncomment one of these to force an error for negative testing (in sandbox mode only). Documentation:
117       // https://developer.paypal.com/tools/sandbox/negative-testing/request-headers/
118       // "PayPal-Mock-Response": '{"mock_application_codes": "INSTRUMENT_DECLINED"}'
119       // "PayPal-Mock-Response": '{"mock_application_codes": "TRANSACTION_REFUSED"}'
120       // "PayPal-Mock-Response": '{"mock_application_codes": "INTERNAL_SERVER_ERROR"}'
121     },
122   });
123
124   return handleResponse(response);
125 };
```

- Line 107 passes the `orderID` from the request parameters into the `createOrder` function.
- Line 108 checks that you have the correct access token.
- Line 109 sets up the `url` for the API call, which includes the `orderID`.
- Lines 111-122 capture an order by sending a `POST` request to the Capture payment endpoint of the Orders v2 API, using `accessToken` and `url`.

See `/client/app.js` in the next section for more details on capturing payments from the front-end.

**Negative testing for captureOrder()**

This section of the `captureOrder()` function provides 3 negative testing opportunities behind content tags:

- `INSTRUMENT_DECLINED`
- `TRANSACTION_REFUSED`
- `INTERNAL_SERVER_ERROR`

Remove the `//` from the beginning of a line to simulate that error. The `captureOrder()` call will pass that line as a `PayPal-Mock-Response` header in the `POST` request to the Capture payment endpoint of the Orders v2 API:

```
116 // Uncomment one of these to force an error for negative testing (in sandbox mode only). Documentation:
117 // https://developer.paypal.com/tools/sandbox/negative-testing/request-headers/
118 // "PayPal-Mock-Response": '{"mock_application_codes": "INSTRUMENT_DECLINED"}'
119 // "PayPal-Mock-Response": '{"mock_application_codes": "TRANSACTION_REFUSED"}'
```