Registering a web-based payment app

- Configuring a payment app with a web app manifest
  - Registering a service worker just-in-time (JIT)
- Debugging a web-based payment app
  - Developing on a local server
  - Port forwarding a local server
  - Remote debugging a website on Android Chrome from desktop DevTools
  - Payment Handler event logging
- Next steps

Web-based payment apps are Progressive Web Apps (PWA) and run on top of service workers. The service worker in a payment app plays an important role as it captures payment requests from a merchant, launches the payment app, and mediates the communication with the merchant.

To configure a web-based payment app, you need to register available payment methods, and a service worker. You can configure your web-based payment app declaratively with a web app manifest.

Configuring a payment app with a web app manifest

To configure your web-based payment app declaratively, serve a web app manifest.

The following properties in the web app manifest are relevant for web-based payment apps:

- `name`
- `icons`
- `serviceworker`

- `src`
- `scope`
- `use_cache`

Check out Setting up a payment method to make sure your payment method manifest points to your web app manifest properly.

Registering a service worker just-in-time (JIT)

Service workers are usually registered using JavaScript, but they can also be automatically registered by the browser when the user chooses to pay with a web-based payment app on a merchant website. This is called *just-in-time (JIT) registration*.

The JIT registration requires only serving the web app manifest and no additional coding. If you've already configured your web app manifest and are serving it properly, you should be all set. The browser will handle the rest.

While a single payment method identifier can support multiple payment apps, JIT registration happens only when the payment method manifest points to a single payment app.

Debugging a web-based payment app

When developing a web-based payment app frontend, you'll probably jump between merchant context and payment app context. The following debugging tips will help your developing experience on Chrome.

Developing on a local server

Which server do you use for development? Many developers tend to use localhost or a company-internal server environment which can be challenging because powerful features in the browser tend to require a secure environment (HTTPS) and a valid certificate. The Payment Request API and the Payment Handler API are no exception and localhosts or company-internal servers usually don't come with a valid certificate.

The good news is some browsers, including Chrome, exempt certificates for `http://localhost` by default. Also in Chrome, you can exempt the certificate requirement by launching a Chrome

instance. For example, to exempt the requirement from `http://*.corp.company.com`, use the following flags:

- `--ignore-certificate-errors`
- `--unsafely-treat-insecure-origin-as-secure=http://*.corp.company.com`

macOS

```
/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --ignore-certificate-errors --unsafely-treat-insecure-origin-as-secure=http://*.corp.company.com
```

Windows

```
chrome.exe --ignore-certificate-errors --unsafely-treat-insecure-origin-as-secure=http://*.corp.company.com
```

Learn more about running Chrome with a runtime flag at Run Chromium with flags.

Port forwarding a local server

You can port forward the local web server to an Android device using Chrome's DevTools and test how it works from a mobile browser. To learn how to do it, check out Access Local Servers.

Remote debugging a website on Android Chrome from desktop DevTools

You can also debug Android Chrome on desktop DevTools. To learn how to do it, check out Get Started with Remote Debugging Android Devices.

Payment Handler event logging

DevTools can display Payment Handler API events for easier local development. Open DevTools on the merchant context and go to the "Payment Handler" section under the **Application** pane. Check "Show events from other domains" and click the "Record" button to start capturing events sent to the service worker that handles payments.

A screenshot of Payment Handler event logging.

Payment Handler event logging.

Next steps

The next step is to learn how the service worker can orchestrate a payment transaction at runtime.

- Orchestrating payment transactions with a service worker

□□□ #3
□ □□□ □□□ 2026-10-31 23:08:22 UTC
□ cocobook-sync □□□ 2026-11-04 17:39:45 UTC