

Using Vite with Django in 2026

Isaac Bythewood · 2026-04-26 · 6 min read

webdev coding

```
VITE vite v6.3.1 building for production...
✓ 312 modules transformed.
analytics/static/ base.css 218.42 kB | gzip: 31.07 kB
analytics/static/ base.js 86.91 kB | gzip: 28.13 kB
analytics/static/ pages.js 12.48 kB | gzip: 4.62 kB
analytics/static/ properties.js 142.35 kB | gzip: 46.81 kB
analytics/static/ collector.js 3.42 kB | gzip: 1.58 kB
✓ built in 412 ms
watching for file changes...

DJANGO Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
April 26, 2026 - 14:02:18
Django version 5.1.4, using settings 'analytics.settings.development'
• Starting development server at http://0.0.0.0:8000/
[26/Apr/2026 14:02:21] "GET /properties/ HTTP/1.1" 200 31482
[26/Apr/2026 14:02:21] "GET /static/base.css HTTP/1.1" 200 218421
[26/Apr/2026 14:02:21] "GET /static/properties.js HTTP/1.1" 200 142353
```

I ran webpack on every Django project I had for years. In 2026 it's Vite, no wrapper package needed. Here's the whole setup.

I ran webpack on every Django project I had for years. The config grew, the build got slower, the plugin churn never stopped. In 2026 I use [Vite](#) and the integration with Django is so thin I don't bother with a wrapper package. The `vite.config.js` is a single file, Django sees regular static files, and `{% static %}` works the way it always has.

I use this on [analytics](#) and [status](#), so most of the snippets below are pulled straight out of those projects.

Why I moved off webpack

A few reasons:

- It's slow. Vite's dev server doesn't bundle in development, it serves source modules over native ESM. The Mews team [cut their builds 80% by switching to Rspack](#), and GitLab's [move to Rolldown-Vite took their builds from 2.5 minutes to 22 seconds](#).
- The config is verbose. A reasonable webpack setup wants `babel-loader`, `css-loader`, `style-loader`, `mini-css-extract-plugin`, `postcss-loader`, `sass-loader`, and `HtmlWebpackPlugin` before you've written a line of app code. Vite ships TypeScript, JSX, CSS, SCSS, and HMR built in. My `vite.config.js` is 30 lines.
- The energy is elsewhere. The [State of JavaScript 2025](#) survey put webpack at 86% usage and 14% retention. New work is going into Vite, esbuild, Rolldown, and Rspack.

- Webpack itself moves slowly. There's a [public 2026 roadmap](#) but no v6 release on the horizon. The Hacker News thread ["It's 2026 now. Is webpack 6.x going to happen?"](#) is a good read.

If you're on a giant webpack codebase and a full migration sounds like too much, [Rspack](#) is a Rust-based drop-in replacement that works with most webpack configs as-is. For a Django project I'd skip the middle step and go straight to Vite.

The Vite config

Each Django app has a `static_src/index.js` that imports its own SCSS and scripts. Vite takes those entry points and writes the bundle to a single `static/` directory inside the project's main app. This is `vite.config.js` straight from analytics:

```
import { resolve } from "path";
import { defineConfig } from "vite";

export default defineConfig({
  base: "/static/",
  build: {
    outDir: resolve(__dirname, "analytics/static"),
    emptyOutDir: true,
    rollupOptions: {
      input: {
        base: resolve(__dirname, "analytics/static_src/index.js"),
        pages: resolve(__dirname, "pages/static_src/index.js"),
        properties: resolve(__dirname, "properties/static_src/index.js"),
        collector: resolve(__dirname, "collector/static_src/index.js"),
      },
      output: {
        entryFileNames: "[name].js",
        assetFileNames: (assetInfo) => {
          if (/\. (png|jpg|gif|svg|webp)$/.test(assetInfo.name)) {
            return "images/[name][extname]";
          }
          return "[name][extname]";
        },
      },
    },
  },
  css: {
    preprocessorOptions: {
      scss: { quietDeps: true },
    },
  },
});
```

A few notes:

- `base: "/static/"` matches Django's `STATIC_URL`, so anything Vite writes into the bundle (image references, font URLs) resolves against the same path Django serves from.
- Each Django app gets its own entry under `rollupOptions.input`. Vite emits a `base.js`, `pages.js`, `properties.js`, and `collector.js` into the same output folder and templates load whichever they need.
- `entryFileNames: "[name].js"` keeps output names predictable. WhiteNoise hashes them at `collectstatic` time, so I don't need Vite to do it too.

- `emptyOutDir: true` wipes the output between builds. Vite refuses to clean a directory outside its project root unless you set this, so without it you get a warning every build and stale files pile up.

Django settings

I don't use `django-vite` or any other integration package. Django doesn't need to know Vite exists, it sees a static directory full of pre-built files and serves them.

```
# settings.py

STATIC_URL = "static/"
STATICFILES_STORAGE = "whitenoise.storage.CompressedManifestStaticFilesStorage"
STATICFILES_DIRS = (BASE_DIR / "analytics/static",)
STATIC_ROOT = BASE_DIR / "static"

MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "whitenoise.middleware.WhiteNoiseMiddleware",
    # ...
]
```

`STATICFILES_DIRS` points at Vite's output directory so `collectstatic` and the dev server pick it up. `CompressedManifestStaticFilesStorage` hashes every file during `collectstatic` and rewrites references in CSS to point at the hashed names. Don't append `?v=...` query strings to `{% static %}`. WhiteNoise expects to control the URLs and the manifest will get out of sync.

Templates stay boring:

```
<link href="{% static 'base.css' %}" rel="stylesheet">
<script type="module" src="{% static 'base.js' %}"></script>
```

In dev that resolves to `/static/base.css`. In production WhiteNoise rewrites it to `/static/base.abc123.css` automatically.

Running Django and Vite together

Vite has a watch mode that rebuilds on every change. I run it next to Django's runserver from a Makefile:

```
.PHONY: run runserver vite

run: install
    ${MAKE} -j2 runserver vite

runserver:
    uv run python manage.py runserver 0.0.0.0:8000

vite:
    bun run dev
```

`make -j2` runs both targets in parallel in the same terminal. The `dev` script in `package.json` is just `vite build --watch`:

```
{
  "scripts": {
    "dev": "vite build --watch",
    "build": "vite build"
  }
}
```

I don't use Vite's dev server. It's great for SPAs but on a Jinja-rendered page HMR doesn't really do anything, and you've added an extra port and an integration layer for no reason.

`vite build --watch` writes plain files to disk that look identical to what production serves. Hard refresh in the browser, same as webpack always was.

If you really want HMR, [django-vite](#) is the package most people reach for and it works well. I just don't think it's worth the extra dependency for a multi-page app.

Per-app static_src

For the analytics dashboard the entry point looks like:

```
// properties/static_src/index.js
import './scripts/property_graphs.js';
import './scripts/property_map.js';
import './scripts/property_date_select.js';
import './scripts/property_filters.js';

import './styles/print.scss';
```

And in the dashboard template:

```
{% block extra_js %}
<script type="module" src="{% static 'properties.js' %}"></script>
{% endblock %}
```

New JS for an app goes in that app's `static_src/` and shows up as `<app-name>.js` in the static directory. No webpack chunks config, no `splitChunks`, no Babel preset to keep current.

Production

The Dockerfile runs `bun run build` once during the image build, then `collectstatic`. After that there's no Node or Vite at runtime, just Gunicorn serving Django and WhiteNoise serving the hashed files.

```
RUN bun install --frozen-lockfile
RUN bun run build
RUN uv run python manage.py collectstatic --noinput
```

Total build time across `bun install`, `vite build`, and `collectstatic` is under 30 seconds on these projects. Webpack used to take three or four minutes.

Sources

Honza Hrubý's ["Goodbye Webpack, Hello Rspack"](#) is the case for the drop-in path if a full migration isn't realistic. Ratchapol Thaworn's ["Migrating from Webpack to Vite"](#) and HK Lee's ["Vite vs. Webpack in 2026"](#) go further on the Vite side. The [Vite docs](#) and the [django-vite](#)

[README](#) are the references I keep open when I'm actually wiring it up. Saas Pegasus has a [longer Django + Vite walkthrough](#) too if you want one with React and Tailwind on top.

Webpack served me well for a long time. I just don't have a reason to keep using it.