Testing

- PHPUnit
- Playwright
  - Layout
  - CI
  - Running locally
  - Two things the setup handles for you
  - Writing tests
- Add-on smoke tests in CI
  - Add-ons come from their release zips, not from source
  - Getting them into wp-env
  - The blocker is a token, not the tooling
  - What the specs should assert
  - The other direction

GiveWP has two test suites. They cover different failure modes and neither replaces the other.

Suite	Runs	Command	CI
PHPUnit	PHP, in-process, against the WordPress test library	`composer test`	`.github/workflows/wordpress.yml`
Playwright	A real browser against a real WordPress install	`npm run test:e2e`	`.github/workflows/tests-e2e.yml`

PHPUnit

The bulk of the coverage. Models, repositories, actions, REST routes, migrations — anything whose behavior can be asserted without a browser belongs here, because it runs in seconds and fails with a stack trace pointing at the line.

Setup and conventions are in tests/README.md.

Add-ons run this same suite against core: an add-on's `tests/bootstrap.php` requires GiveWP's autoloader from the sibling directory and hands its main plugin file to `Give\Tests\Framework\Addons\Bootstrap`, which loads the add-on on `muplugins_loaded` and then defers to core's `tests/bootstrap.php`. In CI the shared `impress-org/givewp-github-actions/.github/workflows/addon-tests.yml` workflow checks out the GiveWP branch named by its `givewp_branch` input, so an add-on can be tested against `develop` before core ships.

Playwright

E2E tests live in `tests/e2e/` and run against a WordPress install started by `wp-env`.

This suite is written for CI. It exists to catch what PHPUnit structurally cannot see: assets that fail to enqueue, React apps that fail to mount, and v3 REST requests that error only once a real browser makes them with a real nonce. Running it locally is supported and useful while writing a spec, but the environment it is designed against is the clean install the workflow builds.

Keep the specs thin. A browser test that asserts something a PHPUnit test could have asserted is a slow test with a worse failure message.

Layout

One spec file per screen, named for the screen, holding everything that screen is covered for: `donor-dashboard.spec.ts`, `reports.spec.ts`, `tools-logs.spec.ts`, `tools-migrations.spec.ts`.

New coverage goes in the file for the screen it exercises. Grouping by the kind of assertion instead — every screen's mount check in one file, every screen's REST check in another — puts one screen's coverage in as many files as there are kinds of assertion, so nothing tells you what a screen is actually covered for, and each new kind adds a file that re-lists every screen. `test.describe` already groups within a file, which is where that grouping belongs.

`admin-pages.spec.ts` is the deliberate exception. It is a breadth-first smoke suite — each admin screen that mounts a React app, one assertion each — and its job is to be pointed at environments core does not control, which is what the add-on plan below builds on. Screens with a spec file of their own are not repeated in it.

Shared helpers live in `tests/e2e/utils/`. `utils/rest.ts` records the REST calls a page makes and asserts it reached the route it owns with nothing failing. Assert on that traffic rather than on rendered records: a wrong route, a missing nonce, or a response shape the client no longer unwraps all produce a page that builds and mounts cleanly and then shows nothing.

CI

`.github/workflows/tests-e2e.yml` installs Composer and npm dependencies, runs `npm run build`, starts wp-env on the port in `.wp-env.json`, and runs the suite. Failure artifacts (screenshot, video, trace) upload on failure. Open a trace with `npx playwright show-trace <path to trace.zip>`.

Running locally

```
# First time only
npm install
composer install
npx playwright install chromium

npm run build       # the admin screens under test are React apps served from build/
npm run env:start
npm run test:e2e
```

`npm run test:e2e:headed` opens a visible Chrome window with each action slowed to 800ms. `npm run test:e2e:ui` opens Playwright's time-travel viewer, which lets you step through a run and see a screenshot at every action.

If your wp-env is not on the port in `.wp-env.json` — a `.wp-env.override.json`, a TLS proxy in front of it, `--auto-port` picking something else — point the run at it:

```
WP_BASE_URL=https://give.wpenv.net:9443 npm run test:e2e
```

Two things the setup handles for you

- `@wordpress/e2e-test-utils-playwright` reads `WP_BASE_URL` from the environment once at module load and otherwise defaults to `http://localhost:8889`, wp-env's tests environment. The `baseURL` passed to `RequestUtils.setup()` never reaches it, so REST API discovery would go to 8889 regardless of what the rest of the suite targets. `tests/e2e/environment.ts` writes the resolved URL back into `process.env` to keep the two in agreement.
- `RequestUtils` builds its own request context with no way to pass `ignoreHTTPSErrors`, so a self-signed local certificate fails the login before any spec runs — while the browser, which does get `ignoreHTTPSErrors`, would have been fine. `tests/e2e/global-setup.ts` relaxes Node's certificate check for local HTTPS runs only; CI talks to wp-env over plain HTTP with verification on.
- A WordPress install answering on the expected port is not necessarily the one under test. If another wp-env environment holds the port, every spec would run green against an unrelated site. `global-setup.ts` checks that the REST API exposes the `givewp/v3` namespace and fails with that explanation if it does not. Seeing that error means `npm run`

`env:start` picked a different port — `env:start` passes `--auto-port` so it can step aside from a busy one — and the run needs `WP_BASE_URL`.

Writing tests

Specs use `@wordpress/e2e-test-utils-playwright`, which supplies the `admin`, `page`, and `requestUtils` fixtures. `tests/e2e/global-setup.ts` logs in once and saves the session to `artifacts/storage-states/admin.json`, so no spec pays for a login.

Assertions should not depend on the site's data. CI runs against a fresh install; a developer runs against whatever their wp-env site accumulated. Assert on structure — a root element, a page heading, a form field — not on a row count or an empty state, unless the test seeds that state itself through `requestUtils.rest()`.

Add-on smoke tests in CI

Not wired up yet. This section is the design, not a description of something that runs.

Add-ons come from their release zips, not from source

Every add-on publishes a single `<slug>.zip` asset on its GitHub release — verified on `give-recurring` 2.19.1, `give-fee-recovery` 2.3.7, `give-form-field-manager` 3.2.1, and `give-data-generator` 1.0.0. The zip is what pup built and what customers install: `vendor/` is already there, assets are already compiled, and everything is wrapped in a `<slug>/` directory, so it unzips straight into a plugins directory.

That removes the per-add-on `composer install` and `npm ci && npm run build` a source checkout would need, and it tests the artifact people actually run rather than a working tree.

```
- name: Download the latest add-on releases
  run: |
    mkdir -p addons
    for slug in give-recurring give-fee-recovery; do
      gh release download --repo "impress-org/$slug" --pattern "$slug.zip" --dir addons
      unzip -q "addons/$slug.zip" -d addons
    done
  env:
```

```
    GH_TOKEN: ${{ secrets.ADDON_READ_TOKEN }}
```

Getting them into wp-env

`.wp-env.override.json` is gitignored and outranks `.wp-env.json`, so the job writes one naming what it just unpacked. Paths are relative to the config file.

```
- name: Compose a wp-env config that includes the add-ons
  run: |
    cat > .wp-env.override.json <<'JSON'
    {
        "plugins": [".", "./addons/give-recurring", "./addons/give-fee-recovery"],
        "port": 8888,
        "testsEnvironment": false
    }
    JSON
```

`addons/` is gitignored, so the same two steps work locally when you want to reproduce a CI failure.

The blocker is a token, not the tooling

`give-recurring`, `give-fee-recovery`, and `give-form-field-manager` are private. Core's `GITHUB_TOKEN` is scoped to `impress-org/give` alone, so it cannot read their releases. `.github/workflows/zip.yml` notes that no bot token is wired into this repository, and `gh secret list --repo impress-org/give` is still empty. A fine-grained PAT or GitHub App installation token with `contents: read` on the add-on repositories is the first step.

`give-data-generator` is public, so the whole mechanism can be proven end to end with the default `GITHUB_TOKEN` before any secret exists.

What the specs should assert

A smoke test that only checks core still works with an add-on active catches most of what breaks: a fatal on activation, a JavaScript error that stops a list table mounting, an asset that 404s. The existing specs in `tests/e2e/admin-pages.spec.ts` already do all three, which is why that file stays breadth-first — pointing it at an environment with add-ons loaded is most of the value, before a single add-on-specific spec is written.

The other direction

Running this from each add-on's own repository needs no new secrets at all: the add-on's `GITHUB_TOKEN` checks out the add-on, and GiveWP core is public so the same token checks that out too. That mirrors the PHPUnit setup already in place — a reusable `addon-e2e-tests.yml` beside `addon-tests.yml` in `impress-org/givewp-github-actions`, taking the same `givewp_branch` input.

It catches an add-on change breaking against core. It does not catch a core change breaking an add-on, which is the direction that needs the token above.

□□□ #1
□ □□□ □□□ 2026-08-29 22:50:57 UTC
□ □□□ □□□ 2026-08-29 22:50:57 UTC