Project Structure

Project Structure

Anonymous · 09/09/2026

Project Structure

Derafu Foundation provides a standardized directory structure for your projects. This document explains the purpose of each directory and key files.

Root Directories

`.github/`

Contains GitHub-specific configurations:

- `workflows/ci.yml`: Continuous Integration workflow that runs tests and code quality checks.
- `workflows/cd.yml`: Continuous Deployment workflow for automatically deploying. By default, it is configured to deploy to GitHub Pages. Deployment is disabled by default, just remove the `false &&` in the file to enable it.

`app/`

Application bootstrap files:

- `bootstrap.php`: The main bootstrap file that initializes the application runtime.

`assets/`

Frontend assets organized by type:

- `css/`: CSS stylesheets.
- `js/`: JavaScript files.
- `img/`: Images and other media files.

`config/`

Configuration files:

- `routes.yaml`: Route definitions for your application.
- `services.yaml`: Service container configuration (dependency injection).

`public/`

Web server document root:

- `index.php`: Application entry point with Front Controller.
- `static/`: Compiled assets (generated by the build process).

Note: If your application doesn't need a Front Controller, by default, it's used only for documentation.

`src/`

Application source code. Organize your PHP classes here according to their namespace. By convention, but not enforced out of Derafu ORG, use:

- `Abstract/`: Abstract classes, with `Abstract` prefix.
- `Contract/`: Interfaces for your application, with `Interface` suffix.
- `Controller/`: Controller classes, with `Controller` suffix.
- `Service/`: Service classes.

`templates/`

Template files for rendering HTML:

- `components/`: Reusable template components (header, footer, etc.).
- `layouts/`: Layout templates that define page structure.
- `pages/`: Page-specific templates.
- `base.html.twig`: Base template that defines the HTML structure for all layouts.
- `error.html.twig`: Error page template.
- `html.html.twig`: HTML wrapper template, used when rendering a markdown o php template.

`tests/`

Test files:

- `fixtures/`: Test fixtures and sample data.
- `src/`: Tests for your source code. You can organize your tests by tests suits, mirroring the structure of the `src/` directory, by features or any other criteria (but not enforced out of Derafu ORG).

`var/`

Temporary files and caches:

- `cache/`: Cache files.
- `logs/`: Log files.
- `tmp/`: Temporary files.

Key Files

Configuration Files

- `.gitignore`: Specifies files that Git should ignore.
- `LICENSE`: MIT license file by default.
- `package.json`: NPM configuration for frontend assets.
- `php-cs-fixer.php`: PHP CS Fixer configuration.
- `phpstan.neon`: PHPStan configuration.
- `phpunit.xml`: PHPUnit configuration.
- `vite.config.js`: Vite build configuration.

Application Files

- `public/index.php`: Main entry point that bootstraps the application.
- `app/bootstrap.php`: Initializes the runtime environment.

File Copying Mechanism

The `Installer` class in `Installer.php` is responsible for copying foundation files to new projects during Composer installation. This class:

1. Defines a list of files to copy in the `FILES` constant.
2. Copies each file from the foundation package to the project directory.
3. Creates necessary directories if they don't exist.
4. Handles file overwrite rules (some files are configured to be overwritten in updates, others not).

Files marked with `true` in the `FILES` constant will be overwritten if they already exist:

```
private const FILES = [
    // Files that will not be overwritten if they exist.
    '.github/workflows/cd.yml',
    '.github/workflows/ci.yml',
    // Files that will be overwritten even if they exist.
    'app/bootstrap.php' => true,
    // ...
];
```

Note: The idea behind overwriting files is to make it easier to update the foundation. But, you can disable it by removing the `Derafu\\Foundation\\Installer::copyFiles` from the `post-install-cmd` and `post-update-cmd` scripts in your `composer.json` file.

Extending the Structure

When creating a new project based on this foundation, you should:

1. Keep the existing directory structure.
2. Add your own directories as needed.
3. Follow PSR-4 autoloading standards for your PHP classes.
4. Place tests in the corresponding structure within the `tests/` directory.

The structure is designed to be flexible while providing a consistent organization pattern across projects.

Last updated on 09/09/2026