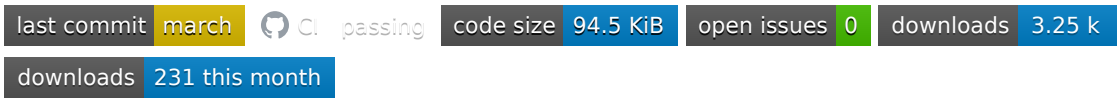
Introduction

Elegant PHP Router with Plugin Architecture

Anonymous · 09/09/2026

Elegant PHP Router with Plugin Architecture



A lightweight, extensible PHP routing library that combines simplicity with power through its parser-based architecture.

Features

- Plugin architecture with swappable parsers.
- Multiple routing strategies (static, dynamic, filesystem).
- Easy to extend with custom parsers.
- Built-in filesystem routing for static sites.
- Support for different content types (.md, .twig, etc.).
- Clean separation of concerns.
- Lightweight with zero dependencies.
- ⚡ Fast pattern matching.
- Comprehensive test coverage.
- URL generation for named routes.

Why Derafu\Routing?

Unlike traditional monolithic routers, Derafu\Routing uses a unique parser-based architecture that offers several advantages:

- **Modularity**: Each routing strategy is encapsulated in its own parser.
- **Flexibility**: Easy to add new routing patterns without modifying existing code.
- **Clarity**: Clear separation between route matching and request handling.
- **Extensibility**: Add custom parsers for specific routing needs.
- **Predictability**: Each parser has a single responsibility.
- **Performance**: Only load the parsers you need.

Installation

Install via Composer:

```
composer require derafu/routing
```

Basic Usage

```
use Derafu\Routing\Router;
use Derafu\Routing\Dispatcher;
use Derafu\Routing\Parser\StaticParser;
use Derafu\Routing\Parser\FileSystemParser;

// Create and configure router.
$router = new Router();
$router->addParser(new StaticParser());
$router->addParser(new FileSystemParser([__DIR__ . '/pages']));

// Add routes.
$router->addRoute('/', 'HomeController::index', name: 'home');
$router->addDirectory(__DIR__ . '/pages');

// Create and configure dispatcher.
$dispatcher = new Dispatcher([
    'md' => fn ($file, $params) => renderMarkdown($file),
    'twig' => fn($file, $params) => renderTwig($file, $params),
]);

// Handle request.
try {
    $route = $router->match();
    echo $dispatcher->dispatch($route);
} catch (RouterException $e) {
    // Handle error.
}
```

Available Parsers

StaticParser

Handles exact route matches:

```
$router->addRoute('/about', 'PagesController::about', name: 'about');
```

DynamicParser

Supports parameters and patterns:

```
$router->addRoute('/users/{id:\d+}', 'UserController::show', name: 'user.show');
$router->addRoute('/blog/{year}/{slug}', 'BlogController::post', name: 'blog.post');
```

FileSystemParser

Maps URLs to files in directories:

```
$router->addDirectory(__DIR__ . '/pages');
// Examples:
// /about maps to /pages/about.md
// /contact maps to /pages/contact.html.twig
```

URL Generation

Generate URLs for named routes:

```
// Set request context (needed for absolute URLs).
$router->setContext(new RequestContext(
    baseUrl: '/myapp',
    scheme: 'https',
    host: 'example.com'
));

// Generate URLs.
$url = $router->generate('user.show', ['id' => 123]); // /myapp/users/123
$url = $router->generate('blog.post', [
    'year' => '2024',
    'slug' => 'hello-world'
]); // /myapp/blog/2024/hello-world

// Generate absolute URL.
$url = $router->generate('about', [], UrlReferenceType::ABSOLUTE_URL);
// https://example.com/myapp/about
```

Creating Custom Parsers

Implement your own routing strategy by creating a parser:

```
class CustomParser implements ParserInterface
{
```

```
    public function parse(string $uri, array $routes): ?RouteMatch
    {
        // Your custom routing logic.
    }

    public function supports(Route $route): bool
    {
        // Define what routes this parser can handle.
    }
}

$router->addParser(new CustomParser());
```

File-based Routing Example

Perfect for static sites:

```
pages/
├── about.md
├── contact.html.twig
└── blog/
    ├── post-1.md
    └── post-2.md
```

URLs are automatically mapped to files:

- `/about` → `pages/about.md`
- `/contact` → `pages/contact.html.twig`
- `/blog/post-1` → `pages/blog/post-1.md`

Last updated on 09/09/2026