Guide

Complete Usage Guide

Anonymous · 09/09/2026

Complete Usage Guide

Derafu Routing is a flexible PHP routing library that uses a parser-based architecture. Instead of having a monolithic router, it separates routing logic into specialized parsers, each handling different types of routes.

Installation

```
composer require derafu/routing
```

Basic Concepts

Parser Architecture

The routing system is built around specialized parsers:

1. **StaticParser**: Handles exact route matches.
2. **DynamicParser**: Processes routes with parameters.
3. **FileSystemParser**: Maps URLs to physical files.

Each parser implements the `ParserInterface`:

```
interface ParserInterface {
    public function parse(string $uri, array $routes): ?RouteMatchInterface;
    public function supports(RouteInterface $route): bool;
}
```

Route Types

Static Routes

The simplest form of routing, handled by `StaticParser`:

```
$router = new Router([new StaticParser()]);
$router->addRoute('/about', 'PagesController::action', name: 'about');
$router->addRoute('/contact', 'ContactController::show', name: 'contact');
```

Dynamic Routes

Handled by `DynamicParser`, supporting various parameter types:

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

// Basic parameter.
$router->addRoute('/users/{id}', 'UserController::show', name: 'user.show');

// Validation with regular expressions.
$router->addRoute('/users/{id:\d+}', 'UserController::show', name: 'user.show');

// Optional parameters.
$router->addRoute('/blog/{year?}', 'BlogController::index', name: 'blog.index');

// Multiple parameters.
$router->addRoute('/blog/{year}/{month?}', 'BlogController::archive', name:
'blog.archive');

// Complex patterns.
$router->addRoute('/users/{username:[a-z0-9_-]+}', 'UserController::profile', name:
'user.profile');
```

Filesystem Routes

The `FileSystemParser` maps URLs to actual files:

```
$parser = new FileSystemParser(
    directories: [__DIR__ . '/pages'],
    extensions: ['.html.twig', '.md']
);
$router->addParser($parser);
```

Directory structure:

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

Using the Router

Basic Configuration

```
use Derafu\Routing\Router;
use Derafu\Routing\Parser\StaticParser;
use Derafu\Routing\Parser\DynamicParser;

$router = new Router([
    new StaticParser(),
    new DynamicParser(),
]);
```

Adding Routes

```
// String handler (Controller@action).
$router->addRoute('/users', 'UserController::index', name: 'users.index');

// Closure handler.
$router->addRoute('/api/data', function($params) {
    return ['data' => 'value'];
}, name: 'api.data');

// Array handler.
$router->addRoute('/blog', [
    'controller' => 'BlogController',
    'action' => 'list'
], name: 'blog.index');

// Routes with name and parameters.
$router->addRoute(
    route: '/users/{id}',
    handler: 'UserController::show',
    name: 'user.show',
    parameters: ['active' => true]
);
```

Route Matching

```
try {
    $match = $router->match('/users/123');
    // $match->getHandler(): Returns the route handler.
    // $match->getParameters(): Returns the route parameters.
    // $match->getName(): Returns the route name if defined.
} catch (RouteNotFoundException $e) {
    // Handle 404.
}
```

URL Generation

The router allows generating URLs from named routes:

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

// Generate relative URLs.
$url = $router->generate('user.show', ['id' => 123]); // /myapp/users/123
$url = $router->generate('blog.archive', [
    'year' => '2024',
    'month' => '03'
]); // /myapp/blog/2024/03

// Generate URL without optional parameter.
$url = $router->generate('blog.archive', [
    'year' => '2024'
]); // /myapp/blog/2024

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

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

Available reference types are:

- `ABSOLUTE_PATH`: Absolute path from root (default).
- `ABSOLUTE_URL`: Complete URL with scheme and host.
- `NETWORK_PATH`: URL without scheme (useful for resources that work on both HTTP and HTTPS).

The Dispatcher

The dispatcher handles the execution of matching routes:

```
$dispatcher = new Dispatcher([
    'md' => function($file, $params) {
        // Render markdown file.
        return parseMarkdown(file_get_contents($file));
    },
```

```
    'twig' => function($file, $params) {
        // Render Twig template.
        return $twig->render($file, $params);
    }
]);

$result = $dispatcher->dispatch($match);
```

Note: This is a very basic *dispatcher*, you should implement your own.

Advanced Usage

Custom Parser Example

```
class RegexParser implements ParserInterface
{
    public function parse(string $uri, array $routes): ?RouteMatchInterface
    {
        foreach ($routes as $route) {
            if (!$this->supports($route)) {
                continue;
            }

            // Custom regex matching logic
            if (preg_match($route->getPath(), $uri, $matches)) {
                return new RouteMatch(
                    $route->getHandler(),
                    $matches
                );
            }
        }
        return null;
    }

    public function supports(RouteInterface $route): bool
    {
        // Define what routes this parser can handle
        return str_starts_with($route->getPath(), '#');
    }
}
```

Best Practices

1. **Parser Order**: Add parsers in order of specificity.
    - StaticParser first (faster, more specific).

    - DynamicParser next.
    - FileSystemParser last (more flexible but slower).

2. **Route Organization**: Group related routes.

```
// User management
$router->addRoute('/users', 'UserController::index', name: 'users.index');
$router->addRoute('/users/{id}', 'UserController::show', name: 'users.show');

// Blog system
$router->addRoute('/blog', 'BlogController::index', name: 'blog.index');
$router->addRoute('/blog/{slug}', 'BlogController::show', name: 'blog.show');
```

3. **Parameter Validation**: Use regex constraints for better security.

```
// Ensure ID is numeric.
$router->addRoute('/users/{id:\d+}', 'UserController::show', name: 'users.show');

// Validate username format.
$router->addRoute('/users/{username:[a-z0-9_-]+}', 'UserController::profile',
name: 'users.profile');
```

4. **Error Handling**: Always wrap matches in try-catch.

```
try {
    $match = $router->match($uri);
    $result = $dispatcher->dispatch($match);
} catch (RouteNotFoundException $e) {
    // Handle 404.
} catch (DispatcherException $e) {
    // Handle dispatcher errors.
}
```

5. **URL Generation**: Always use route names instead of hardcoded URLs.

```
// Bad
$url = '/users/' . $id;

// Good
$url = $router->generate('users.show', ['id' => $id]);
```

6. **Request Context**: Configure context if absolute URLs are needed.

```
$router->setContext(new RequestContext(
```

```
    baseUrl: '/myapp',
    scheme: 'https',
    host: 'example.com',
    httpPort: 80,
    httpsPort: 443
));
```

Last updated on 09/09/2026