Kernel Project

Lightweight Kernel Implementation with DI Container

Anonymous · 21/08/2026 · #php

Lightweight Kernel Implementation with DI Container

A lightweight kernel implementation with a dependency injection container, inspired by Symfony but with a minimal approach.

Overview

Derafu Kernel provides a clean, flexible foundation for PHP applications with minimal dependencies. It offers:

- A lightweight kernel implementation with DI container.
- Environment-aware configuration management.
- Support for PHP and YAML configuration files.
- Container caching for improved performance.
- Built-in support for multiple environments (dev, prod, test, etc.).

Installation

```
composer require derafu/kernel
```

Basic Usage

Create a Kernel

The simplest way to use the kernel is to create an instance with a specific environment:

```
use Derafu\Kernel\MicroKernel;

// Create a kernel with 'dev' environment and debug mode enabled.
```

```
$kernel = new MicroKernel('dev', true);

// Boot the kernel to initialize the container.
$kernel->boot();
```

Custom Kernel Implementation

You can extend the `MicroKernel` class to customize its behavior:

```
use Derafu\Kernel\MicroKernel;
use Symfony\Component\DependencyInjection\Loader\Configurator\ContainerConfigurator;

class AppKernel extends MicroKernel
{
    // Override the default configuration files.
    protected const CONFIG_FILES = [
        'services.php' => 'php',
        'routes.php' => 'routes',
        'parameters.yaml' => 'yaml',
    ];

    // Add additional container configuration.
    protected function configure(ContainerConfigurator $configurator): void
    {
        $services = $configurator->services();

        // Register application services.
        $services->set('app.service', MyService::class)
            ->public()
            ->args(['param1', 'param2']);
    }
}
```

Environment Configuration

The kernel uses an environment object to determine settings and directories:

```
use Derafu\Kernel\Environment;

// Create an environment with custom settings.
$environment = new Environment(
    'prod',                 // Environment name.
    false,                  // Debug mode.
    ['custom' => 'value']   // Context variables.
);

// Create a kernel with the custom environment.
$kernel = new MicroKernel($environment);
```

Configuration

Directory Structure

The kernel expects a `config` directory for configurations. Any other structure is free.

Service Configuration

Define services in `config/services.php`:

```
use Symfony\Component\DependencyInjection\Loader\Configurator\ContainerConfigurator;

return static function (ContainerConfigurator $configurator) {
    $services = $configurator->services();

    // Auto-configure and autowire services by default.
    $services->defaults()
        ->autowire()
        ->autoconfigure();

    // Register a single service.
    $services->set('app.service', AppService::class)
        ->public();

    // Register multiple services from namespace.
    $services->load('App\\Service\\', '../src/Service/*')
        ->public();
};
```

Route Configuration

Define routes in `config/routes.php` or `config/routes.yaml`:

```
// routes.php
return [
    'home' => [
        'path' => '/',
        'controller' => 'App\\Controller\\HomeController::index',
    ],
    'blog_show' => [
        'path' => '/blog/{slug}',
        'controller' => 'App\\Controller\\BlogController::show',
        'parameters' => [
            'requirements' => [
                'slug' => '[a-z0-9-]+',
```

```
            ],
        ],
    ],
];
```

Or in YAML:

```yaml
# routes.yaml
home:
    path: /
    controller: App\Controller\HomeController::index

blog_show:
    path: /blog/{slug}
    controller: App\Controller\BlogController::show
    parameters:
        requirements:
            slug: '[a-z0-9-]+'
```

Environment Types

The kernel supports multiple environment types through constants in `EnvironmentInterface`:

- `LOCAL`: Local development environment.
- `DEVELOPMENT`: Development environment.
- `TEST`: Testing environment.
- `STAGING`: Staging environment.
- `QUALITY_ASSURANCE`: QA environment.
- `PREPRODUCTION`: Pre-production environment.
- `PRODUCTION`: Production environment.

Last updated on 21/08/2026