Docs

HTTP Project

Derafu HTTP

Anonymous · 21/08/2026 · #php

Table of Contents

HTTP Project

Derafu HTTP

Anonymous · 21/08/2026 · #php

Derafu HTTP

Last updated on 21/08/2026

Introduction

Standard-Compliant HTTP Library with Extended Features

Anonymous · 21/08/2026

Standard-Compliant HTTP Library with Extended Features

A PSR and RFC compliant HTTP library that provides elegant request/response handling, content negotiation and problem details for PHP applications.

Why Derafu\Http?

Simple, but not limited, HTTP Handling

Most HTTP libraries either do too much or too little. Derafu\Http provides:

- **Extended Request/Response**: Smart content type negotiation and safe data access.
- **Content Negotiation**: Intelligent format detection and response transformation.
- **Problem Details**: RFC 7807 implementation for structured error handling.
- **PSR Compliance**: Built on PSR-7 and PSR-15 standards.
- **Middleware Architecture**: Flexible request/response processing pipeline.

Key Features

- **Smart Request Handling**: Safe access to query, post, and JSON data.
- **Intelligent Responses**: Automatic content negotiation and format transformation.
- **Structured Errors**: Complete RFC 7807 Problem Details implementation.
- **Modular Design**: Core HTTP functionality with middleware support.
- **Type Safety**: Use of enums for HTTP status codes and content types.
- **Middleware Pipeline**: PSR-15 compliant middleware chain for request processing.

Installation

```
composer require derafu/http
```

Basic Usage

public/index.php

```
use Derafu\Http\Kernel;
use Derafu\Kernel\Environment;

require_once dirname(__DIR__) . '/app/bootstrap.php';

return fn (array $context): Kernel => new Kernel(new Environment(
    $context['APP_ENV'],
    (bool) $context['APP_DEBUG'],
    $context
));
```

Middleware Configuration

The library uses PSR-15 middlewares for request processing. Configure your middleware stack in the services file, for example `services.yaml`:

```
# Core middlewares in required order.
Psr\Http\Server\RequestHandlerInterface:
    class: Derafu\Http\Service\RequestHandler
    public: true
    arguments:
        $middlewares:
            - '@Derafu\Http\Middleware\RequestFactoryMiddleware'
            - '@Derafu\Http\Middleware\RouterMiddleware'
            - '@Derafu\Http\Middleware\DispatcherMiddleware'
            - '@Derafu\Http\Middleware\ResponseNormalizerMiddleware'

# Register individual middlewares.
Derafu\Http\Middleware\RequestFactoryMiddleware: ~
Derafu\Http\Middleware\RouterMiddleware: ~
Derafu\Http\Middleware\DispatcherMiddleware: ~
Derafu\Http\Middleware\ResponseNormalizerMiddleware: ~
```

Core Middlewares

The library includes four essential middlewares that must be configured in order:

1. **RequestFactoryMiddleware**: Converts PSR-7 requests to Derafu requests.
2. **RouterMiddleware**: Handles URL routing and route matching.
3. **DispatcherMiddleware**: Executes route handlers (controllers, closures or templates).
4. **ResponseNormalizerMiddleware**: Ensures PSR-7 compliant responses.

Custom Middlewares

Create custom middlewares by implementing PSR-15's `MiddlewareInterface`:

```
class CustomMiddleware implements MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ): ResponseInterface {
        // Process request.
        $response = $handler->handle($request);
        // Process response.
        return $response;
    }
}
```

Working with Requests

```
// Safe access to request data.
$id = $request->query('id', 0);
$data = $request->json();
$file = $request->file('document');

// Content negotiation.
$format = $request->getPreferredFormat();
```

Creating Responses

```
// Automatic content negotiation.
// Returns JSON for API requests in `/api` paths
return ['status' => 'ok'];

// Explicit responses.
return (new Response())->asJson($data)->withHttpStatus(HttpStatus::CREATED);

// Redirect.
return (new Response())->redirect('https://www.example.com');
```

Error Handling with Problem Details

Any exception will be treated as "Problem Detail" (RFC 7807). To have control over all fields, exceptions must implement `HttpExceptionInterface`.

```
{
    "type": "about:blank",
```

```
    "title": "Not Found",
    "status": 404,
    "detail": "No route found for \"/api/mustFail\".",
    "instance": "/api/mustFail",
    "extensions": {
        "timestamp": "2025-02-20T22:04:25+00:00",
        "environment": "dev",
        "debug": true,
        "context": [],
        "throwable": null
    }
}
```

Integration with Other Packages

Derafu\Http is designed to work with other Derafu packages:

Required

- **derafu/kernel**: The core of the application, with dependency injection.
- **derafu/renderer**: For template rendering, with dependency on **derafu/twig**.
- **derafu/routing**: For URL routing, with autodiscovery of templates.
- **derafu/translation**: For I18n, with a simple translator that supports ICU.

Optional

- **derafu/markdown**: For renderig templates in markdown format.

Suggested

- **derafu/data-processor**: For data processing: format, cast, sanitize and validate.

Last updated on 21/08/2026

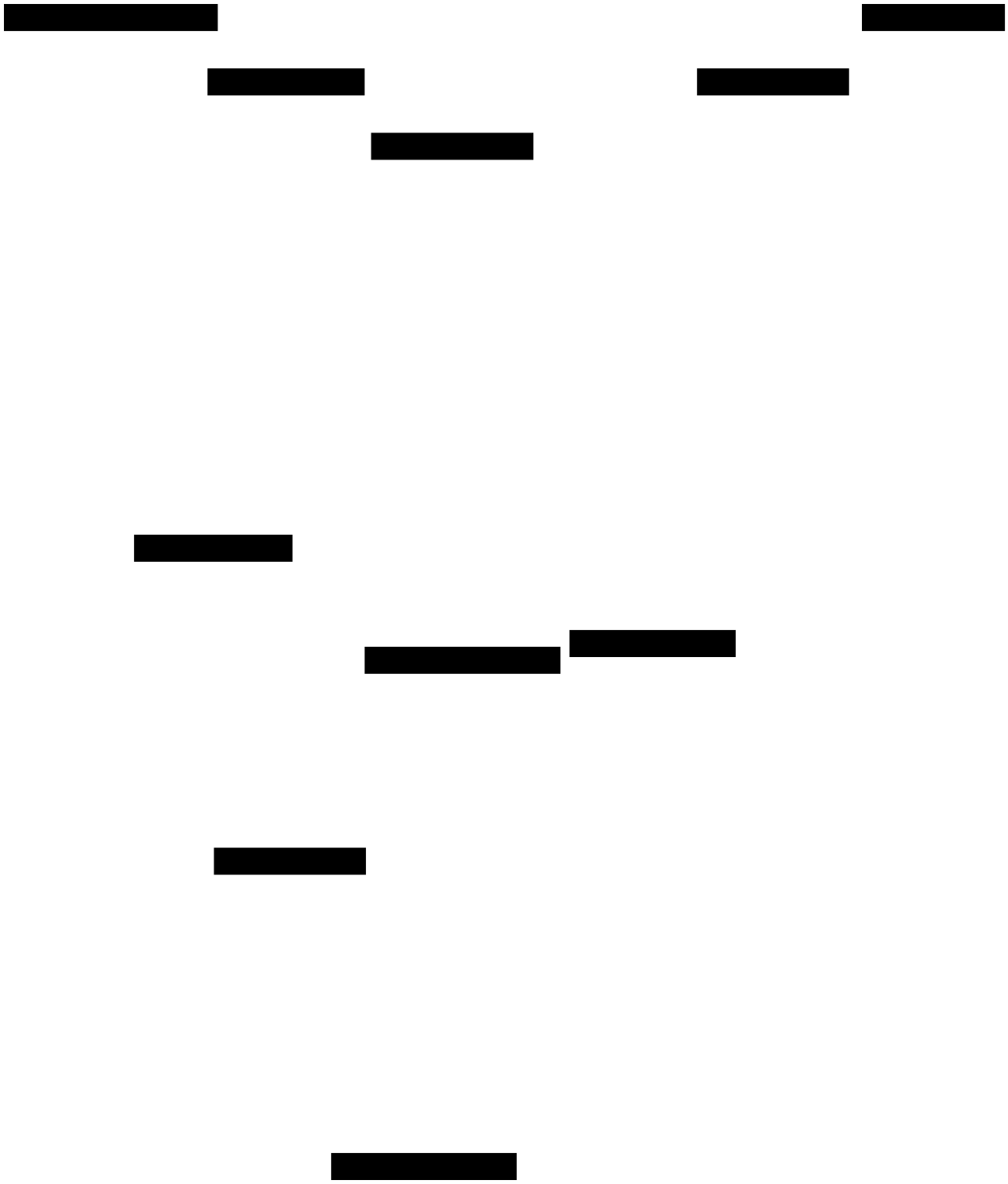
HTTP Flow

Flow - Detailed Explanation

Anonymous · 21/08/2026

Flow - Detailed Explanation

The Derafu HTTP component implements a clean, modular approach to handling HTTP requests in PHP applications. The flow diagram illustrates how an HTTP request travels through the system, from the initial client request to the final response delivery.



Key Components

Client

The external entity (browser, API consumer, etc.) that initiates the HTTP request and receives the response.

Runtime

The application's entry point and execution environment. It serves as the orchestrator of the entire HTTP flow, responsible for:

- Bootstrapping the application.
- Creating the PSR-7 request object from the HTTP request.
- Initializing the kernel.
- Delegating request processing.
- Sending the response back to the client.

Kernel

The core of the application, responsible for:

- Building and configuring the dependency injection container.
- Loading application configurations.
- Managing the application lifecycle.
- Coordinating the request handling process.

The Kernel implements a micro-kernel architecture that keeps the core small and efficient while pushing most functionality to handlers and middleware.

Request Handler

Processes the HTTP request and produces a response. Handlers:

- Receive the request from the kernel.
- Apply application logic.
- Generate an appropriate response.
- May use services from the container to fulfill the request.

Request

A PSR-7 compliant ServerRequest object that represents the HTTP request. It encapsulates:

- HTTP method.
- URI and query parameters.

- Headers.
- Body content.
- Server and environment variables.

Response

A PSR-7 compliant Response object that represents the HTTP response. It includes:

- Status code.
- Headers.
- Response body.

Container

The dependency injection container that:

- Manages service instantiation and configuration.
- Provides service dependencies throughout the application.
- Implements the PSR-11 container interface.
- Supports autowiring for simplified service definition.

The HTTP Request/Response Flow

1. **Client Sends HTTP Request** The flow begins when a client makes an HTTP request to the application.

2. **Runtime Creates Request Object** The Runtime transforms the raw HTTP request into a PSR-7 compliant Request object, preparing it for processing.

3. **Runtime Initializes Kernel** The Kernel is initialized with the necessary configurations to process the current request.

4. **Kernel Builds Container** The Kernel builds and configures the dependency injection container, making all application services available.

5. **Runtime Delegates Request Processing** The Runtime passes the Request object to the Kernel for processing.

6. **Kernel Delegates to Handler** The Kernel identifies the appropriate Request Handler based on routing information and delegates the request processing.

7. **Handler Generates Response** The Handler applies business logic, interacts with the application services as needed, and generates a Response object.

8. **Response Returned to Runtime** The generated Response is returned through the call chain back to the Runtime.

9. **Runtime Sends Response to Client** The Runtime outputs the Response to the client, completing the HTTP cycle.

Last updated on 21/08/2026

Design Principles

Design Principles

Anonymous · 21/08/2026

Design Principles

The Derafu HTTP component provides a lightweight, standards-compliant approach to HTTP request handling. By following established PHP interoperability standards (PSRs) and sound architectural principles, it enables the development of robust, maintainable web applications with minimal overhead.

The Derafu HTTP is designed with several key principles in mind:

PSR Compliance

- Implements PSR-7 for HTTP messages.
- Follows PSR-11 for container interoperability.
- Supports PSR-15 for middleware.

Separation of Concerns

Each component has a single, well-defined responsibility:

- Runtime manages the application lifecycle.
- Kernel coordinates processing.
- Handlers implement business logic.
- Container manages dependencies.

Flexibility

The design allows for:

- Custom request handlers.
- Middleware integration.
- Extensible container configuration.
- Multiple runtime environments.

Testability

The clear separation of components and dependency injection make unit testing straightforward:

- Mock the container for handler tests.
- Create test requests easily.
- Validate responses without invoking the full stack.

Implementation Guidelines

When working with the Derafu HTTP component:

1. **Define Request Handlers** for different routes or endpoints.
2. **Configure the Container** with your application services.
3. **Set up your routes** to map URLs to handlers.
4. **Extend the base classes** as needed for custom functionality.

The architecture is designed to be minimal yet powerful, allowing developers to focus on the business logic rather than HTTP processing details.

Last updated on 21/08/2026