

Testing

Testing

Anonymous · 08/10/2026

Testing

There are two useful levels: sending requests through the **real middleware stack**, and testing a **single middleware** in isolation. The `derafu/http` repository does both, in `tests/src/HttpTest.php` and `tests/src/Middleware/`.

Testing the Whole Stack

Build the same stack the container would build, with the real classes, and hand it a request. Only the renderer needs a stub when your routes return data instead of templates.

```
use Derafu\Http\Factory\ProblemFactory;
use Derafu\Http\Factory\RequestFactory;
use Derafu\Http\Factory\SafeThrowableFactory;
use Derafu\Http\Middleware\DispatcherMiddleware;
use Derafu\Http\Middleware\RequestFactoryMiddleware;
use Derafu\Http\Middleware\ResponseNormalizerMiddleware;
use Derafu\Http\Middleware RouterMiddleware;
use Derafu\Http\Service\Dispatcher;
use Derafu\Http\Service\ProblemHandler;
use Derafu\Http\Service\RequestHandler;
use Derafu\Renderer\Contract\RendererInterface;
use Derafu\Routing\Parser\DynamicParser;
use Derafu\Routing\Parser\StaticParser;
use Derafu\Routing\Router;
use Invoker\Invoker;
use Nyholm\Psr7\ServerRequest;
use PHPUnit\Framework\Attributes\CoversNothing;
use PHPUnit\Framework\TestCase;
use Psr\Http\Message\ResponseInterface as PsrResponseInterface;
use Symfony\Component\DependencyInjection\ParameterBag\ParameterBag;

#[CoversNothing]
class ApiTest extends TestCase
{
    private function send(string $path): PsrResponseInterface
    {
        $params = new ParameterBag([
```

```

        'kernel.environment' => 'test',
        'kernel.debug' => false,
        'kernel.context' => [],
        'kernel.project_dir' => sys_get_temp_dir(),
    ]);

    $router = new Router([new StaticParser(), new DynamicParser()], [
        'hello' => [
            'path' => '/api/hello',
            'handler' => fn () => ['status' => 'ok'],
        ],
    ]);

    $dispatcher = new Dispatcher(
        new Invoker(),
        $this->createStub(RendererInterface::class)
    );

    // A new handler for each request: it can not be reused.
    $handler = new RequestHandler(
        new ProblemFactory($params, new SafeThrowableFactory($params)),
        new ProblemHandler($router, $dispatcher),
        new RequestFactoryMiddleware(new RequestFactory(), $params),
        new RouterMiddleware($router),
        new DispatcherMiddleware($dispatcher),
        new ResponseNormalizerMiddleware(),
    );

    return $handler->handle(new ServerRequest(
        'GET',
        'http://localhost' . $path,
        ['Accept' => 'application/json'],
        null,
        '1.1',
        [
            'SERVER_NAME' => 'localhost',
            'SERVER_PORT' => '80',
            'REQUEST_METHOD' => 'GET',
            'HTTP_HOST' => 'localhost',
        ]
    ));
}

public function testHello(): void
{
    $response = $this->send('/api/hello');

    $this->assertSame(200, $response->getStatusCode());
    $this->assertSame(
        ['status' => 'ok'],
        json_decode((string) $response->getBody(), true)
    );
}

```

```

    );
}

public function testUnknownPath(): void
{
    $response = $this->send('/api/nope');

    $this->assertSame(404, $response->getStatusCode());
}
}

```

Errors are part of the stack, so you can assert them the same way: a route whose handler throws gives you the response a client would get, with its status, headers and `detail`. See [Error Handling](#).

What the setup needs

Most mistakes when building this by hand come from these points:

- **A new RequestHandler for each request.** It keeps its position in the chain, so a second request with the same instance finds the chain already consumed.
- **Server parameters.** `RouterMiddleware` builds the request context from them. Without `SERVER_PORT` it emits a PHP warning, and with `failOnWarning` on that fails the test.
- **The kernel parameters.** The factories read `kernel.environment`, `kernel.debug`, `kernel.context` and `kernel.project_dir` from the `ParameterBag`. A missing key throws a `ParameterNotFoundException` from inside the error handling, which hides the real error.
- **An Accept header.** Without it the client prefers HTML, and an error would be rendered as an HTML page through the router instead of as JSON. Ask for JSON, or use a path under `/api`.
- **Real ProblemHandler and ProblemFactory.** They are what turns exceptions into responses. Stubbing them makes errors in your own routes harder to see.

Coverage metadata

These tests run many classes on purpose. If your PHPUnit configuration has `requireCoverageMetadata` and `beStrictAboutCoverageMetadata`, mark them with `#[CoversNothing]`, otherwise PHPUnit reports them as risky.

Testing a Middleware in Isolation

Give it a request and a handler that is either a mock (to check whether the chain continued) or a stub that returns a known response.

```

use PHPUnit\Framework\Attributes\CoversClass;
use PHPUnit\Framework\TestCase;
use Psr\Http\Server\RequestHandlerInterface;

```

```
#[CoversClass(MyMiddleware::class)]
class MyMiddlewareTest extends TestCase
{
    public function testItShortCircuits(): void
    {
        $handler = $this->createMock(RequestHandlerInterface::class);
        $handler->expects($this->never())->method('handle');

        $response = (new MyMiddleware())->process($request, $handler);

        $this->assertSame(204, $response->getStatusCode());
    }
}
```

Two details that matter for middlewares that read the path:

- **Set the path as is.** Some PSR-7 implementations normalize the URI. To test how a middleware reacts to a hostile path like `../secret.css`, set it explicitly:

```
$request = $request->withUri($request->getUri()->withPath($path));
```

- **Mind symlinked temporary directories.** On macOS `sys_get_temp_dir()` is under `/var`, which is a symlink to `/private/var`. When a test compares real paths, resolve the directory with `realpath()` first.

Tests That Find Bugs

A test that exposes a bug must fail, and stop the suite, until the bug is fixed. Do not exclude it by group, skip it or weaken it: that hides the problem, and the next change builds on top of it.

Last updated on 08/10/2026