Welcome to knowledge-base

Pest

composer.json

```
{
    "require-dev": {
        "pestphp/pest": "^1.22",
        "pestphp/pest-plugin-laravel": "^1.3",
        "pestphp/pest-plugin-livewire": "^1.0",
        "phpstan/extension-installer": "^1.1",
        "phpstan/phpstan-deprecation-rules": "^1.0",
        "phpstan/phpstan-phpunit": "^1.0",
        "phpunit/php-code-coverage": "^9.2",
        "phpunit/phpunit": "^9.5"
    },
    "scripts": {
        "test:pest": "vendor/bin/pest --order-by default -d memory_limit=6144M",
        "test:pest-coverage": "php -dpcov.enabled=1 -dpcov.directory=. -dpcov.exclude='~vendor~' vendor/bin/pest -d
memory_limit=6144M --testdox --verbose --coverage --min=85",
    "test:unit": "vendor/bin/testbench package:test --no-coverage",
    "test:types": "vendor/bin/phpstan analyse",
    "test": [
        "@lint:fix",
        "@test:types",
        "@test:unit"
    ]
}
```

Issues verlinken

Sie können einen Pull-Request oder Branch mit einem Issue verknüpfen, um anzuzeigen, dass eine Behebung in Bearbeitung ist, und um das Issue automatisch zu schließen, wenn der Pull-Request oder Branch zusammengeführt wird.

Hinweis: Die speziellen Schlüsselwörter in einer Pull-Request-Beschreibung werden interpretiert, wenn die Pull-Request auf den *Default* - Branch des Repositorys abzielt. Wenn die Basis des PR jedoch *ein anderer Zweig* ist, werden diese Schlüsselwörter ignoriert, es werden keine Links erstellt und das Zusammenführen des PR hat keine Auswirkung auf die Ausgaben. **Wenn Sie einen Pull-Request über ein Schlüsselwort mit einem Issue verknüpfen möchten, muss sich der PR auf dem Standard-Branch befinden.**

Über verknüpfte Probleme und Pull-Requests

Sie können ein Problem manuell mit einer Pull-Anforderung verknüpfen oder ein unterstütztes Schlüsselwort in der Beschreibung der Pull-Anforderung verwenden.

Wenn Sie eine Pull-Anforderung mit dem Problem verknüpfen, auf das sich die Pull-Anforderung bezieht, können Mitbearbeiter sehen, dass jemand an dem Problem arbeitet.

Wenn Sie einen verknüpften Pull-Request mit dem Standard-Branch eines Repositorys zusammenführen, wird das verknüpfte Issue automatisch geschlossen. Weitere Informationen zum Standard-Zweig finden Sie unter „ Ändern des Standard-Zweigs “.

Verknüpfen einer Pull-Anforderung mit einem Problem mithilfe eines Schlüsselworts

Sie können eine Pull-Anforderung mit einem Problem verknüpfen, indem Sie ein unterstütztes Schlüsselwort in der Beschreibung der Pull-Anforderung oder in einer Commit-Nachricht verwenden. Die Pull-Anfrage **muss sich** im Standard-Branch befinden.

- nah dran
- schließt
- abgeschlossen
- Fix
- behebt
- Fest

- beschließen
- beschließt
- aufgelöst

Wenn Sie ein Schlüsselwort verwenden, um auf einen Pull-Request-Kommentar in einem anderen Pull-Request zu verweisen, werden die Pull-Requests verknüpft. Durch das Zusammenführen der referenzierenden Pull-Anforderung wird auch die referenzierte Pull-Anforderung geschlossen.

Die Syntax zum Schließen von Schlüsselwörtern hängt davon ab, ob sich das Problem im selben Repository wie die Pull-Anfrage befindet.

Verknüpftes Problem	**Syntax**	**Beispiel**
Ausgabe im selben Repository	*SCHLÜSSELWORT* # AUSGABE - *NUMMER*	`Closes #10`
Ausgabe in einem anderen Repository	*KEYWORD OWNER* / *REPOSITORY* # *ISSUE-NUMBER*	`Fixes octo-org/octo-repo#100`
Mehrere Probleme	Verwenden Sie für jedes Problem die vollständige Syntax	`Resolves #10, resolves #123, resolves octo-org/octo-repo#100`

Nur manuell verknüpfte Pull-Requests können manuell entkoppelt werden. Um die Verknüpfung eines Problems aufzuheben, das Sie mit einem Schlüsselwort verknüpft haben, müssen Sie die Pull-Request-Beschreibung bearbeiten, um das Schlüsselwort zu entfernen.

Sie können auch schließende Schlüsselwörter in einer Commit-Nachricht verwenden. Das Problem wird geschlossen, wenn Sie den Commit in den Standard-Branch zusammenführen, aber die Pull-Anforderung, die den Commit enthält, wird nicht als verknüpfte Pull-Anforderung aufgeführt.

Jeder mit Schreibberechtigungen für ein Repository kann eine Pull-Anforderung manuell mit einem Vorgang über die Seitenleiste für Pull-Anforderungen verknüpfen.

Sie können bis zu zehn Issues manuell mit jeder Pull-Anfrage verknüpfen. Das Issue und die Pull-Anforderung müssen sich im selben Repository befinden.

1. Navigieren Sie auf GitHub.com zur Hauptseite des Repositorys.
2. Klicken Sie unter Ihrem Repository-Namen auf **Pull-Anforderungen** .

   Auswahl der Registerkarte „Probleme und Pull-Requests“.
3. Klicken Sie in der Liste der Pull-Requests auf den Pull-Request, den Sie mit einem Issue verknüpfen möchten.
4. Klicken Sie in der rechten Seitenleiste im Abschnitt "Entwicklung" auf .

5. Klicken Sie auf das Problem, das Sie mit der Pull-Anforderung verknüpfen möchten. Drop-down zum Link-Problem

Jeder mit Schreibberechtigungen für ein Repository kann eine Pull-Anfrage manuell verknüpfen oder von der Issue-Seitenleiste zu einem Issue verzweigen.

Sie können bis zu zehn Issues manuell mit jeder Pull-Anfrage verknüpfen. Das Problem kann sich in einem anderen Repository befinden als der verknüpfte Pull-Request oder Branch. Ihr zuletzt ausgewähltes Repository wird gespeichert

1. Navigieren Sie auf GitHub.com zur Hauptseite des Repositorys.
2. Klicken Sie unter Ihrem Repository-Namen auf **Issues** .

   Registerkarte "Probleme".
3. Klicken Sie in der Liste der Issues auf das Issue, mit dem Sie eine Pull-Anfrage oder einen Branch verknüpfen möchten.
4. Klicken Sie in der rechten Seitenleiste auf **Entwicklung** . Entwicklungsmenü in der rechten Seitenleiste
5. Klicken Sie auf das Repository mit der Pull-Anfrage oder dem Zweig, den Sie mit dem Problem verknüpfen möchten. Drop-down, um das Repository auszuwählen
6. Klicken Sie auf den Pull-Request oder Branch, den Sie mit dem Issue verknüpfen möchten. Drop-down, um Pull-Request oder Branch zu verknüpfen
7. Klicken Sie auf **Anwenden** . Anwenden

Weiterlesen

- " Automatisch verlinkte Verweise und URLs "

Git

Force push git branches

```
git push --force origin main
git push -f origin main
git push origin +main
```

- Original tweet by Stefan Judis

Github key anpassen bzw. bereinigen

```
composer update --lock
```

```
composer update --lock s
```

Arr::helper

Throw an exception

```php
<?php

$config = ['.....'];
$apiKey = Arr::get($config, 'api_key', fn () => throw new Exeption('your message here'));

// or
$apiKey = $config['api_key'] ?? throw new Exeption('your message here'));
```

- Original tweet by Steve Bauman

Collection

sort

```php
<?php

$exampleEntries = [
    'my example exa' => 'value_4',
    'my example' => 'value_1',
    'my example ex' => 'value_3',
    'my example e' => 'value_2',
];

$result = collect($exampleEntries)->sort()->toArray();

$result = [
    "my example" => "value_1",
    "my example e" => "value_2",
    "my example ex" => "value_3",
    "my example exa" => "value_4",
]
```

sortKeys

```php
<?php

$exampleEntries = [
    'key_1' => 'value_1',
    'key_3' => 'value_3',
    'key_2' => 'value_2',
    'key_7' => 'value_7',
    'key_4' => 'value_4',
    'key_6' => 'value_6',
    'key_5' => 'value_5',
];

$result = collect($exampleEntries)->sortKeys()->toArray();

$result = [
    "key_1" => "value_1",
    "key_2" => "value_2",
    "key_3" => "value_3",
    "key_4" => "value_4",
    "key_5" => "value_5",
    "key_6" => "value_6",
    "key_7" => "value_7",
]
```

Eloquent

Touch eloquent method

```php
<?php

$user = User::find(1);

// instead of
$user->update(['subscribed_at' => now()]);

// use
$user->touch('subscribed_at');
```

- Original tweet by Oussama Sid

Validation

Test-Route

```
<?php
use Illuminate\Support\Facades\Route;

Route::get('/test-route', function () {
    // return your example
})
```

withoutMiddleware

```
use App\Http\Middleware\FirstMiddleware;
use App\Http\Middleware\SecondMiddleware;
use Illuminate\Support\Facades\Route;

Route::prefix('my-prefix')
    ->middleware([
        FirstMiddleware::class,
        SecondMiddleware::class,
    ])
    ->group(function () {
        Route::get('my-route', function () {
            // Uses first & second middleware
        });
        Route::withoutMiddleware(FirstMiddleware::class)
            ->get('route-without-first-middleware', function () {
                // Uses second middleware
            });
        Route::withoutMiddleware(SecondMiddleware::class)
            ->get('route-without-second-middleware', function () {
                // Uses first middleware
            });
        Route::withoutMiddleware([
            FirstMiddleware::class,
            SecondMiddleware::class,
        ])->get('route-without-middleware', function () {
            // Uses no middleware
        }
    );
});
```

with Parameters

```
<?php
namespace App\Http\Middleware;
```

```php
use Closure;

class EnsureUserHasRole
{
    public function handle($request, Closure $next, $role)
    {
        if (! $request->user()->hasRole($role)) {
            // Redirect...
        }

        return $next($request);

    }
}
```

```php
<?php
use Illuminate\Support\Facades\Route;

Route::put('/post/{id}', function ($id) {
    //
})->middleware('role:editor');
```

Tests

expectException and expectExceptionMessage

```php
<?php
use Application\User\Queries\ListUserQuery;
use Illuminate\Http\Request;
use Spatie\QueryBuilder\Exceptions\InvalidFilterQuery;
use Tests\TestCase;

class ListUserQueryTest extends TestCase
{
    /** @test */
    public function it_throws_an_exception_when_the_key_for_filtering_is_not_supported(): void
    {
        $this->expectException(InvalidFilterQuery::class);
        $this->expectExceptionMessage('Requested filter(s) `key_not_supported` are not allowed. Allowed filter(s) are `id, email, nickname`.');

        $request = new Request(['filter' => ['key_not_supported' => 'value is irrelevant']]);

        new ListMyModelQuery($request);
    }
}
```

it_uses_the_right_query_filters

```php
<?php
    /** @test */
    public function it_uses_the_right_query_filters(): void
    {
        $this->assertQueryFilterEquals(
            ListMyModelQuery::class,
            [
                AllowedFilter::exact('id'),
                AllowedFilter::exact('handle'),
                AllowedFilter::partial('name', 'display_name'),
            ]
        );
    }
```

assertThrows | it_throws_an_error_if_model_doesnt_exist

```php
<?php
    /** @test */
    public function it_throws_an_error_if_model_doesnt_exist(): void
    {
```

```
        $className = User::class;

        $this->assertThrows(
            fn () => User::findOrFail(0),
            ModelNotFoundException::class,
            "No query results for model [{$className}] 1"
        );
    }
```

it_uses_the_right_query_class

```
<?php
    /** @test */
    public function it_uses_the_right_query_class(): void
    {
        $query = $this->mock(
            ListMyModelQuery::class,
            fn (MockInterface $mock) => $mock->shouldReceive('simplePaginate')
                ->once()
                ->andReturn(MyModelFactory::new()->create()->simplePaginate()),
        );

        $controller = new ViewMyModelListController();
        $controller($query, new Request());

        $this->assertControllerUsesClass(
            ViewMyModelListController::class,
            ListMyModelQuery::class
        );
    }
```

it_uses_the_right_collection

```
<?php
    /** @test */
    public function it_uses_the_right_collection(): void
    {
        $this->assertControllerReturns(
            ViewMyModelListController::class,
            MyModelCollection::class
        );
    }
```

it_uses_the_right_middleware

```
<?php
    public function middlewares(): array
    {
        return [
```

```
            'group1' => ['group1_sub1', 'group1_sub2'],
            'group2' => ['group2_sub1', 'group2_sub2'],
            MyMiddleware::class => [MyMiddleware::class],
        ];
    }

    /**
     * @test
     *
     * @dataProvider middlewares
     */
    public function it_uses_the_right_middleware($middleware)
    {
        $this->assertControllerUsesMiddleware(
            ViewMyModelListController::class,
            $middleware,
        );
    }
```

it_returns_the_right_structure | Collection

```
<?php
    /** @test */
    public function it_returns_the_right_structure(): void
    {
        MyModelFactory::new()->create();

        $response = $this->createResource(MyModelCollection::class, MyModel::simplePaginate());

        $response->assertJsonStructureExact([
            'data',
            'links',
            'meta',
        ]);

        $this->assertEquals(MyModelResource::class, MyModelCollection::make([])->collects);
    }
```

it_returns_the_right_structure | Resource

```
<?php
/** @test */
    public function it_returns_the_right_structure(): void
    {
        $response = $this->createResource(
            MyModelResource::class,
            MyModelFactory::new()->create()
        );

        $response->assertJsonStructureExact([
            'id',
```

```
            'my_column_one',
            'my_column_two',
            'created_at',
            'updated_at',
        ]);

        $response->assertJson(
            fn (AssertableJson $json) => $json
                ->whereAllType([
                    'id' => ['integer'],
                    'my_column_one' => ['string'],
                    'my_column_two' => [null, 'string'],
                    'created_at' => ['string'],
                    'updated_at' => ['string'],
                ])
                ->etc()
        );
    }
```

examples_for_mock_actions

```
<?php
    /** @test */
    public function examples_for_mock_actions(): void
    {
        $myAction = $this->spy(MyActionn::class);
        $myAction->shouldReceive('execute')
            ->once()
            ->withArgs(fn ($arg) => $arg === 'foo')
            ->andReturn('bar');

        // or
        $spy = $this->spy(MyActionn::class);
        $spy->shouldReceive('execute')
            ->once()
            ->with(MyExampleModel::class, MyExampleData::class)
            ->andReturn('bar');
    }
```

Event::fake

Event::assertDispatched

```
<?php
/** @test */
    public function event_assert_dispatched(): void
    {
        Event::fake();
        // or
        Event::fake([
```

```
            ExampleCreated::class,
        ]);

        $myExample = MyExampleFactory::new()->create();

        // Assert a event was dispatched...
        Event::assertDispatched(ExampleCreated::class);

        // Assert a event was dispatched...
        Event::assertDispatched(ExampleCreated::class, function ($event) use ($myExample) {
            return $event->myExample->is($myExample);
        });
    }
```

Event::assertNotDispatched

```
<?php
/** @test */
    public function event_assert_not_dispatched(): void
    {
        Event::fake();
        // or
        Event::fake([
            ExampleCreated::class,
        ]);

        $myExample = MyExampleFactory::new()->create();

        // Assert a event was not dispatched...
        Event::assertNotDispatched(ExampleCreated::class);
    }
```

Event::assertDispatchedTimes

```
<?php
/** @test */
    public function event_assert_dispatched_times(): void
    {
        Event::fake();
        // or
        Event::fake([
            ExampleCreated::class,
        ]);

        $myExample = MyExampleFactory::new()->create();

        // Assert a event was dispatched exactly n times...
        Event::assertDispatchedTimes(ExampleCreated::class, 1);

        // Assert a event was dispatched at least n times...
        Event::assertDispatchedTimes(ExampleCreated::class, 1, '>=');

        // Assert a event was dispatched at most n times...
```

```
        Event::assertDispatchedTimes(ExampleCreated::class, 1, '<=');

        // Assert a event was dispatched exactly n times with the given callback...
        Event::assertDispatchedTimes(ExampleCreated::class, 1, function ($event) use ($myExample) {
            return $event->myExample->is($myExample);
        });

        // Assert a event was dispatched at least n times with the given callback...
        Event::assertDispatchedTimes(ExampleCreated::class, 1, '>=', function ($event) use ($myExample) {
            return $event->myExample->is($myExample);
        });

        // Assert a event was dispatched at most n times with the given callback...
        Event::assertDispatchedTimes(ExampleCreated::class, 1, '<=', function ($event) use ($myExample) {
            return $event->myExample->is($myExample);
        });
    }
```

Event::assertListening

```
<?php
/** @test */
    public function event_assert_listening(): void
    {
        Event::fake();
        // or
        Event::fake([
            ExampleCreated::class,
        ]);

        $myExample = MyExampleFactory::new()->create();

        // assert that a listener is listening to a given event
        Event::assertListening(ExampleCreated::class, ExampleListener::class);

        // assert that a listener is not listening to a given event
        Event::assertNotListening(ExampleCreated::class, ExampleListener::class);

        // assert that a listener is listening to a given event with the given callback
        Event::assertListening(ExampleCreated::class, function ($event) use ($myExample) {
            return $event->myExample->is($myExample);
        });
    }
```

Event::assertNotListening

```
<?php
/** @test */
    public function event_assert_not_listening(): void
    {
        Event::fake();
        // or
        Event::fake([
```

```
            ExampleCreated::class,
        ]);

        $myExample = MyExampleFactory::new()->create();

        // assert that a listener is not listening to a given event
        Event::assertNotListening(ExampleCreated::class, ExampleListener::class);
    }
```

Queue::fake

Queue::assertPushed

```
/** @test */
    public function examples_for_queue_fakes(): void
    {
        Queue::fake();

        // or
        Queue::fake([
            ExampleJob::class,
        ]);

        // queue a job
        ExampleJob::dispatch();

        // assert that a job was pushed...
        Queue::assertPushed(ExampleJob::class);

        // assert that a job was pushed a given number of times...
        Queue::assertPushed(ExampleJob::class, 1);

        // assert that a job was pushed with a given payload...
        Queue::assertPushed(ExampleJob::class, function ($job) {
            return $job->example == 'example';
        });
    }
```

Queue::assertNothingPushed

```
/** @test */
    public function queue_assert_nothing_pushed(): void
    {
        Queue::fake();

        // or
        Queue::fake([
            ExampleJob::class,
        ]);

        // queue a job
        ExampleJob::dispatch();
```

```
        // assert that no jobs were pushed...
        Queue::assertNothingPushed();

        // or
        Queue::assertNothingPushed(ExampleJob::class);
    }
```

Queue::assertPushedOn

```
/** @test */
    public function queue_assert_pushed_on(): void
    {
        Queue::fake();

        // or
        Queue::fake([
            ExampleJob::class,
        ]);

        // queue a job
        ExampleJob::dispatch();

        // assert that a job was pushed on a given queue...
        Queue::assertPushedOn('queue-name', ExampleJob::class);

        // assert that a job was pushed a given number of times on a given queue...
        Queue::assertPushedOn('queue-name', ExampleJob::class, 1);

        // assert that a job was pushed with a given payload on a given queue...
        Queue::assertPushedOn('queue-name', ExampleJob::class, function ($job) {
            return $job->example == 'example';
        });
    }
```

Queue::assertPushedWithChain

```
/** @test */
    public function queue_assert_pushed_with_chain(): void
    {
        Queue::fake();

        // or
        Queue::fake([
            ExampleJob::class,
        ]);

        // queue a job
        ExampleJob::dispatch();

        // assert that a job was pushed with a given chain...
        Queue::assertPushedWithChain(ExampleJob::class, [
            new AnotherJob,
```

```
            new YetAnotherJob,
        ]);

        // assert that a job was pushed with a given chain on a given queue...
        Queue::assertPushedWithChain('queue-name', ExampleJob::class, [
            new AnotherJob,
            new YetAnotherJob,
        ]);

        // assert that a job was pushed with a given chain a given number of times...
        Queue::assertPushedWithChain(ExampleJob::class, [
            new AnotherJob,
            new YetAnotherJob,
        ], 1);

        // assert that a job was pushed with a given chain a given number of times on a given queue...
        Queue::assertPushedWithChain('queue-name', ExampleJob::class, [
            new AnotherJob,
            new YetAnotherJob,
        ], 1);
    }
```

Queue::assertPushedWithoutChain

```
/** @test */
    public function queue_assert_pushed_without_chain(): void
    {
        Queue::fake();

        // or
        Queue::fake([
            ExampleJob::class,
        ]);

        // queue a job
        ExampleJob::dispatch();

        // assert that a job was pushed without a given chain...
        Queue::assertPushedWithoutChain(ExampleJob::class);

        // assert that a job was pushed without a given chain on a given queue...
        Queue::assertPushedWithoutChain('queue-name', ExampleJob::class);

        // assert that a job was pushed without a given chain a given number of times...
        Queue::assertPushedWithoutChain(ExampleJob::class, 1);

        // assert that a job was pushed without a given chain a given number of times on a given queue...
        Queue::assertPushedWithoutChain('queue-name', ExampleJob::class, 1);
    }
```

Queue::assertPushedWithCallback

```
/** @test */
    public function queue_assert_pushed_with_callback(): void
    {
        Queue::fake();

        // or
        Queue::fake([
            ExampleJob::class,
        ]);

        // queue a job
        ExampleJob::dispatch();

        // assert that a job was pushed with a given callback...
        Queue::assertPushedWithCallback(ExampleJob::class, function ($job) {
            return $job->example == 'example';
        });

        // assert that a job was pushed with a given callback on a given queue...
        Queue::assertPushedWithCallback('queue-name', ExampleJob::class, function ($job) {
            return $job->example == 'example';
        });

        // assert that a job was pushed with a given callback a given number of times...
        Queue::assertPushedWithCallback(ExampleJob::class, function ($job) {
            return $job->example == 'example';
        }, 1);

        // assert that a job was pushed with a given callback a given number of times on a given queue...
        Queue::assertPushedWithCallback('queue-name', ExampleJob::class, function ($job) {
            return $job->example == 'example';
        }, 1);
    }
```

Queue::assertPushedWithoutCallback

```
/** @test */
    public function queue_assert_pushed_without_callback(): void
    {
        Queue::fake();

        // or
        Queue::fake([
            ExampleJob::class,
        ]);

        // queue a job
        ExampleJob::dispatch();

        // assert that a job was pushed without a given callback...
        Queue::assertPushedWithoutCallback(ExampleJob::class);

        // assert that a job was pushed without a given callback on a given queue...
        Queue::assertPushedWithoutCallback('queue-name', ExampleJob::class);
```

```
        // assert that a job was pushed without a given callback a given number of times...
        Queue::assertPushedWithoutCallback(ExampleJob::class, 1);

        // assert that a job was pushed without a given callback a given number of times on a given queue...
        Queue::assertPushedWithoutCallback('queue-name', ExampleJob::class, 1);
    }
```

Queue::assertPushedAfterResponse

```
/** @test */
    public function queue_assert_pushed_after_response(): void
    {
        Queue::fake();

        // or
        Queue::fake([
            ExampleJob::class,
        ]);

        // queue a job
        ExampleJob::dispatch();

        // assert that a job was pushed after (and not before) a given job...
        Queue::assertPushedAfterResponse(ExampleJob::class, AnotherJob::class);

        // assert that a job was pushed after (and not before) a given job on a given queue...
        Queue::assertPushedAfterResponse(ExampleJob::class, AnotherJob::class, 'queue-name');
    }
```

Validation

prepareForValidation

```php
<?php
use Illuminate\Validation\Rule;
use Smake\Common\Http\Requests\Request;

class EditUserRequest extends Request
{
    protected function prepareForValidation()
    {
        parent::prepareForValidation();

        $this->merge([
            'nickname' => $this->input('nickname', '') . ' with Love',
        ]);
    }
}
```

Rule::unique with ignore method on update

Adding a record:

```php
<?php

use Illuminate\Validation\Rule;
use Smake\Common\Http\Requests\Request;

class AddUserRequest extends Request
{
    public function rules()
    {
        return [
            'email' => [
                'required',
                'email:strict',
                'max:255',
                Rule::unique('users', 'email'),
            ],
        ];
    }
}
```

Update an existing record:

```php
<?php
```

```php
use Illuminate\Validation\Rule;
use Smake\Common\Http\Requests\Request;

class EditUserRequest extends Request
{
    public function rules()
    {
        return [
            'email' => [
                'required',
                'email:strict',
                'max:255',
                Rule::unique('users', 'email')->ignore($this->route('userId')),
            ],
        ];
    }
}
```

passedValidation

```php
<?php
use Illuminate\Validation\Rule;
use Smake\Common\Http\Requests\Request;

class EditUserRequest extends Request
{
    protected function passedValidation()
    {
        $this->merge([
            'nickname' => $this->input('nickname', '') . '❤️',
        ]);

        $this->getValidatorInstance()->setData(
            collect($this->input())
                ->only(collect($this->rules())->keys())
                ->all()
        );
    }
}
```

Migrations

Unique columns

Laravel Example 1:

```
<?php

DB::update("
   ALTER TABLE tournament_league_game_days
   ADD COLUMN game_schedule_day_unique varchar (512)
   GENERATED ALWAYS AS
   (
      CONCAT(
         CONCAT(day, '#', game_schedule_id),
         '#',
         IF(deleted_at IS NULL, '-',  deleted_at)
      )
   ) VIRTUAL;

");

DB::update("
   CREATE UNIQUE INDEX game_schedule_day_unique ON tournament_league_game_days
(game_schedule_day_unique);
");
```

Laravel Example 2:

```
<?php

Schema::table('tournament_league_game_days', function (Blueprint $table) {
   $table->string('game_schedule_day_unique')
      ->virtualAs(
         DB::raw(
            "CONCAT(
               CONCAT(day, '#', game_schedule_id),
               '#',
               IF(deleted_at IS NULL, '-',  deleted_at)
            )"
         )
      );
});

Schema::table('tournament_league_game_days', function (Blueprint $table) {
   $table->unique(['game_schedule_day_unique'], 'game_schedule_day_unique_index');
});
```

- Original tweet by Tobias_Petry.sql

Kommandos

mkdocs help

```
mkdocs --help
```

mkdocs new

```
mkdocs new
```

mkdocs build

```
mkdocs build
```

mkdocs serve

```
mkdocs serve
```

mkdocs serve --livereload

```
mkdocs serve --livereload
```

Install

Install python3

Weitere Information siehe Python3.de

Install mkdocs

```
pip3 install mkdocs
```

Pip Commands

pip3

```
pip3 --version
```

Plugins

mkdocs-glightbox

```
pip3 install mkdocs-glightbox
```

mkdocs-minify-plugin

```
pip3 install mkdocs-minify-plugin
```

mkdocs-static-i18n

```
pip3 install mkdocs-static-i18n
pip3 install mkdocs-static-i18n --use-pep517
```

mkdocs-admonition

```
pip3 install mkdocs-admonition
```

mkdocs-pdf-export-plugin

```
pip3 install mkdocs-pdf-export-plugin
```

Python3

Install python3

```
python3 -m ensurepip --upgrade
```

python3 --version

```
python3 --version
```

Templates

```
pip3 install mkdocs-material
```

Kommandos

pest: composer test:pest-coverage

Siehe hier: Pest.de

pest: Datei Testen

```
composer test:pest-coverage Tests\Pfad\Zur\Datei\DateiOhneEndung
```

php

Try catch finally

Beim Ausführen unserer Skripte treten Fehler und Ausnahmen auf, das gehört zur Natur der Sache. Jedoch ist es nötig, Fehler voneinander zu unterscheiden und entsprechend zu behandeln. Dafür können wir in PHP die Schlüsselwörter **try**, **catch** und **finally** verwenden.

1. Ein einfacher PHP try catch Block

Fehler und Ausnahmen können wir behandeln, wenn der Code, der diese produziert, in einem **try** Block geschrieben wird. Mit dem Schlüsselwort **catch** können wir diese dann abfangen.

```
<?php
try {
   # Tu was
} catch (Exception $e) {
   # Kümmere dich um Ausnahmen
}
```

1.1 Fehler und Ausnahmen

Ein Fehler tritt zum Beispiel auf, wenn du eine Funktion aufrufst, die nie definiert wurde.

```
<?php undefinierte\_funktion(); ?\>

<?php undefinierte\_funktion(); ?>

<?php
<?php undefinierte_funktion(); ?>
```

Output:

```
Fatal error: Uncaught Error: Call to undefined function undefinierte\_funktion() in C:\\xampp\\htdocs\\codecitrus\
\try\_catch.php:1 Stack trace: #0 {main} thrown in C:\\xampp\\htdocs\\codecitrus\\try\_catch.php on line 1

Fatal error: Uncaught Error: Call to undefined function undefinierte\_funktion() in C:\\xampp\\htdocs\\codecitrus\
\try\_catch.php:1 Stack trace: #0 {main} thrown in C:\\xampp\\htdocs\\codecitrus\\try\_catch.php on line 1

Fatal error: Uncaught Error: Call to undefined function undefinierte_funktion() in C:
\xampp\htdocs\codecitrus\try_catch.php:1 Stack trace: #0 {main} thrown in C:
\xampp\htdocs\codecitrus\try_catch.php on line 1
```

Um Code zu definieren, der nur ausgeführt wird, wenn ein Fehler (**Error**) auftritt, kannst du **try** und **catch** anwenden.

```php
<?php
try {
    undefinierte_funktion();
    echo 'Es ist kein Fehler aufgetreten';
} catch (Error $e) {
    echo 'Ein Fehler ist aufgetreten' . '<br>';
    echo $e->getMessage();
}
```

Output:

```
Ein Fehler ist aufgetreten
Call to undefined function undefinierte_funktion()
Ein Fehler ist aufgetreten Call to undefined function undefinierte\_funktion()
```

Wenn stattdessen eine Ausnahme (**Exception**) auftritt, musst du das im **catch** Block so angeben, um diese zu behandeln.

```php
<?php
function ausnahme() {
    throw new Exception('Eine Ausnahme ist aufgetreten');
}
try {
    ausnahme();
} catch (Exception $e) {
    echo $e->getMessage();
}
```

Output:

```
Eine Ausnahme ist aufgetreten
```

1.2 Beide abfangen mit Throwable

Die vorigen Erklärungen haben sowohl Exceptions als auch Errors behandelt. Um beide abzufangen, könntest du zwei **catch** Blöcke hintereinander definieren.

```php
<?php
try {
    undefinierte_funktion();
} catch (Exception $e) {
    echo 'Exception';
} catch (Error $e) {
    echo 'Error';
}
```

Output:

```
Error
```

Eine weitere Möglichkeit ist, **Throwable** mit **catch** abzufangen. Throwable ist das Interface, das jede PHP-Klasse implementieren muss, soll sie mit **throw** aufgerufen werden. Für das nächste Beispiel wird die Funktion *ausnahme*, die wir vorhin definiert haben, erneut genutzt.

```
<?php
try {
   undefinierte_funktion();
} catch (Throwable $t) {
   echo 'Gotta catch \'em all!';
}
// Gotta catch 'em all!
try {
   ausnahme();
} catch (Throwable $t) {
   echo 'Schnapp sie dir alle!';
}
// Schnapp sie dir alle!
```

2. finally um den Block zu beenden

Mit dem Schlüsselwort **finally** kannst du deinem Block weiteren Code hinzufügen. **finally** ermöglicht es dir, Code zu schreiben, der ausgeführt wird egal ob ein Fehler auftritt oder nicht.

```
<?php
function finally_beispiel_1() {
   try {
      throw new Exception();
   } catch (Exception $e) {
      echo "Exception!" . '<br>';
   } finally {
      echo "Finally!";
   }
}
finally_beispiel_1();
```

Output:

```
Exception!
Finally!
```

2.1 Vorsicht mit return!

return kann bei Verwendung von **finally** zu unerwarteten Ergebnissen führen.

```
<?php
function finally_beispiel_2() {
   try {
      throw new Exception();
      return 1;
   } catch (Exception $e) {
      echo "Exception!" . '<br>';
```

```
        return 2;
    } finally {
        echo "Finally!" . '<br>';
        return 3;
    }
}
echo finally_beispiel_2();
```

Output:

```
Exception!
Finally!
3
```

Im vorigen Beispiel wurde der Wert von **return** aus dem **catch** Block einfach mit dem aus **finally** überschrieben. Dasselbe würde passieren, wenn **return** bereits einen Wert aus **try** empfangen hätte.

3. **Fazit** – PHP try & catch

Mit **try**, **catch** und **finally** stellt uns PHP mächtige Werkzeuge zur Verfügung, um Ausnahmefälle und Fehler zu behandeln. Damit können wir Logs erstellen, alternative Routinen ausführen oder einfach unser Skript stoppen. Vorsicht ist jedoch geboten, wenn wir unseren Code in solchen Blöcken mit **return** verwenden, da dies zu unerwarteten Ergebnissen führen kann.

- [Original Artikel von Patrick](https://codegree.de/php-try-catch/

When does PHP call __destruct()?

In PHP,

`__construct()` is called while creating an object and `__destruct()` is called while the object is being removed from memory. Using this knowledge, we can create more fluent APIs as demonstrated by Freek Van der Herten in this short video.

Now let's see when PHP calls

`__destruct()` exactly.

An object is removed from memory if you explicitly remove it:

```
<?php
$object = new Object();

unset($object); // __destruct will be called immediately.

$object = null; // __destruct will be called immediately.
```

It's also called when the scope where the object live is about to be terminated, for example at the end of a controller method:

```php
<?php
function store(Request $request)
{
  $object = new Object();

  User::create(...);

  // __destruct will be called here.

  return view('welcome');
}
```

Even if we're within a long running process, queued job for example, __destruct will be called before the end of the handle method:

```php
<?php
function handle()
{
  $object = new Object();

  User::create(...);

  // __destruct will be called here.
}
```

It'll also be called when the script is being terminated:

```php
<?php
function handle()
{
  $object = new Object();

  User::create(...);

  // __destruct will be called here.

  exit(1);
}
```

- Original article by Mohamed Said

Kommandos

phpstan: Der tägliche Wahnsinn

```
vendor/bin/phpstan clear-result-cache --memory-limit=5G
vendor/bin/phpstan analyse --ansi --memory-limit=5G
vendor/bin/phpstan --generate-baseline --memory-limit=5G
```

Kommandos

Valet: Erstellt V-Hosts Links: htttp://www.meine-lokale-webseite.test und htttps://www.meine-lokale-webseite.test

```
valet link --secure meine-lokale-webseite
```

Valet: Löscht die V-Hosts Links: htttp://www.meine-lokale-webseite.test und htttps://www.meine-lokale-webseite.test

```
valet unlink --secure meine-lokale-webseite
```

Valet: Alle registrierten Valet-Links anzeigen

```
valet links
```

Valet: Listet alle verfügbaren Valet Kommandos auf

```
valet -h
valet --help
```