



Testing Laravel Applications

Backend Brownbag Session | 08.02.2023

Agenda

- Types of tests
- PhpStorm Configuration
- DatabaseTransactions && RefreshDatabase
- Data Providers
- UrlGenerator
- Testing Events
- Testing Notifications
- Testing Files
- Mocking behavior
- Parallel running
- Memory issues with GitHub Actions



Types of tests

- low level - called **unit tests** - which test small pieces of code
- higher level tests such as **feature and integration tests** - which check that different parts of your code work well together, and
- **browser tests** - which interact with your code the same way a user would.

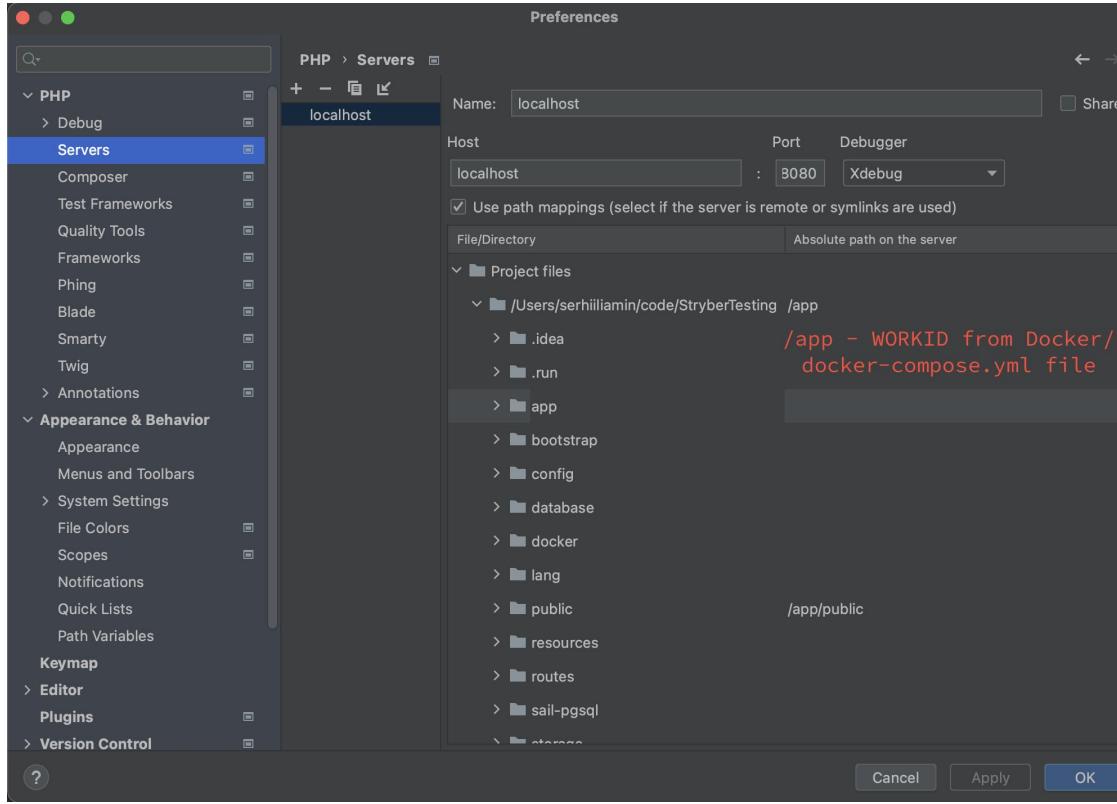
Each type of test focuses on the code differently. There are varied opinions on what each type of test should do and not do.

PhpStorm Configuration Pros:

- ◆ Running tests directly from PhpStorm
- ◆ Debugging code during test running
- ◆ Easy to filter passed/failsed tests
- ◆ Easy to wach output logs
- ◆ Possibility to use another database than described in **phpunit.xml**
- ◆ Save a lot of time during writing tests

PhpStorm Configuration

◆ Configure Xdebug



PhpStorm Configuration

◆ Configure remote debug

The screenshot shows the "Run/Debug Configuration Templates" dialog in PhpStorm. On the left, a sidebar lists various configuration templates, with "PHP Remote Debug" currently selected. The main area displays the configuration details for "PHP Remote Debug".

Configuration

- Filter debug connection by IDE key (?)
- Server:
- IDE key(session id): IDE KEY from Xdebug configuration

Pre-configuration

1. Install [Xdebug](#) or [Zend Debugger](#) on the Web Server.
Validate debugger configuration on the Web Server.
2. Install [browser toolbar](#) or bookmarklets.
3. Start "PHP Remote Debug" run-configuration.
4. Start debug session in browser with the toolbar or bookmarklets.

Before launch

+ - ⚪ ▲ ▼

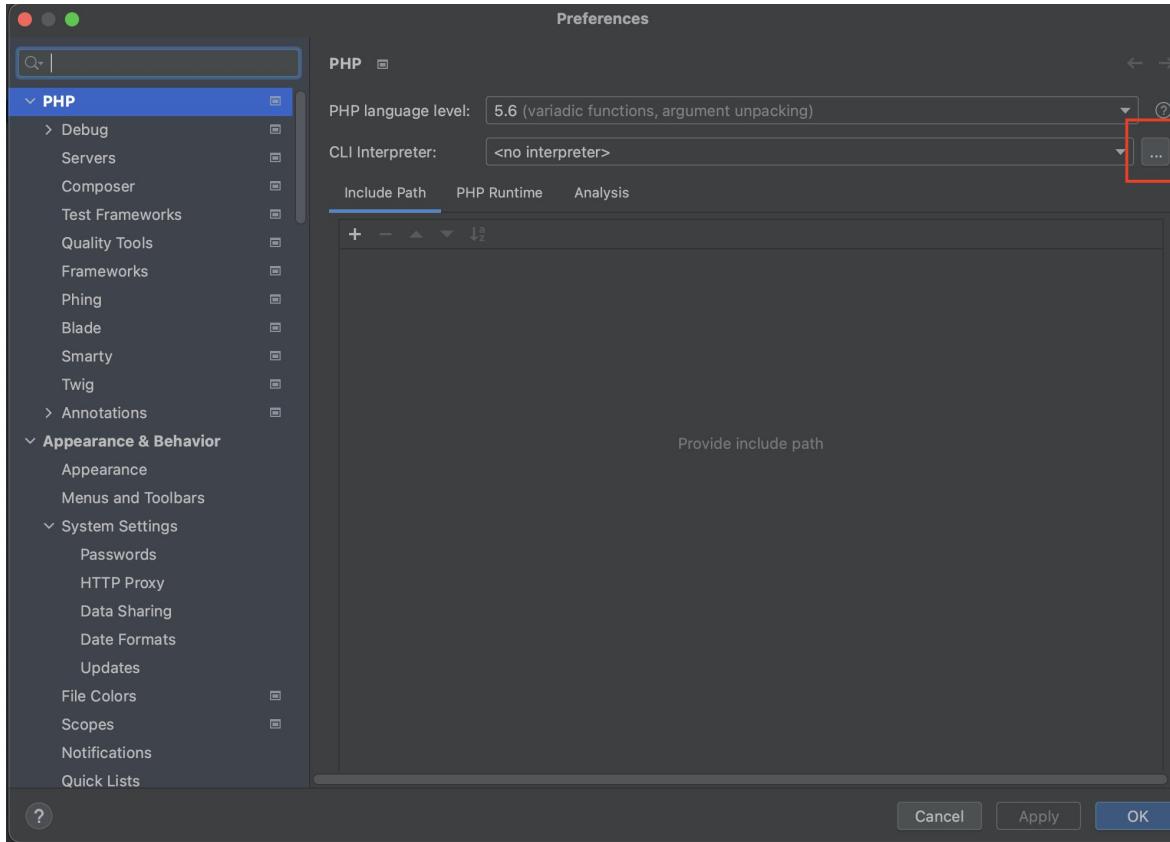
There are no tasks to run before launch

Show this page Activate tool window

Cancel Apply OK

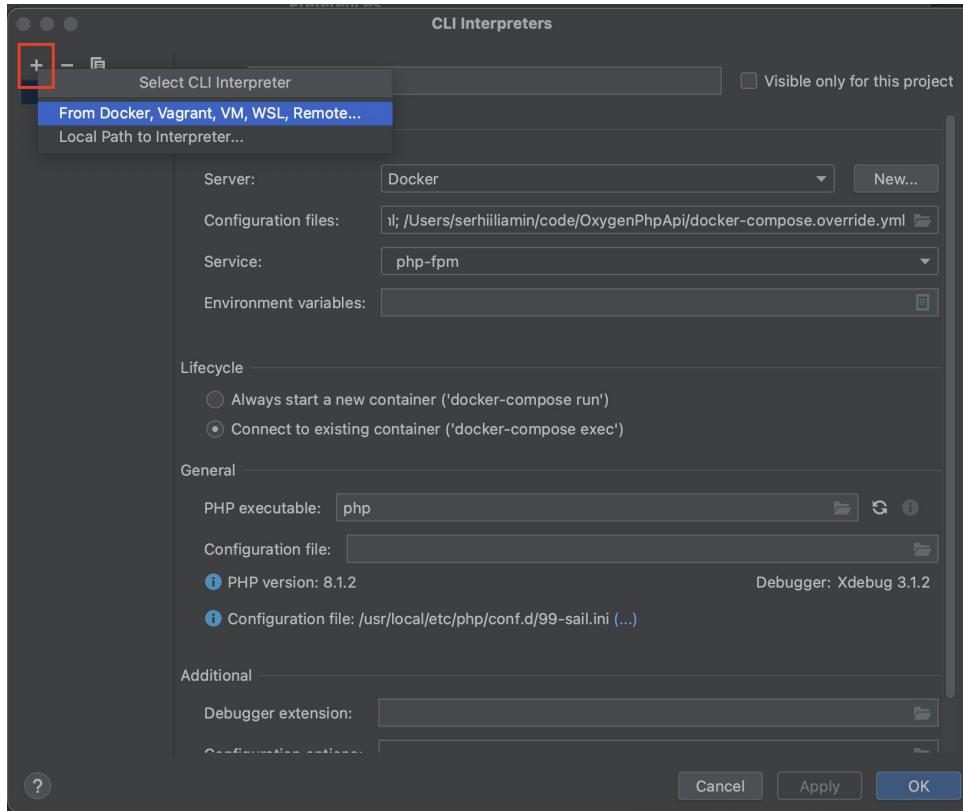
PhpStorm Configuration

◆ Configure CLI interpreter



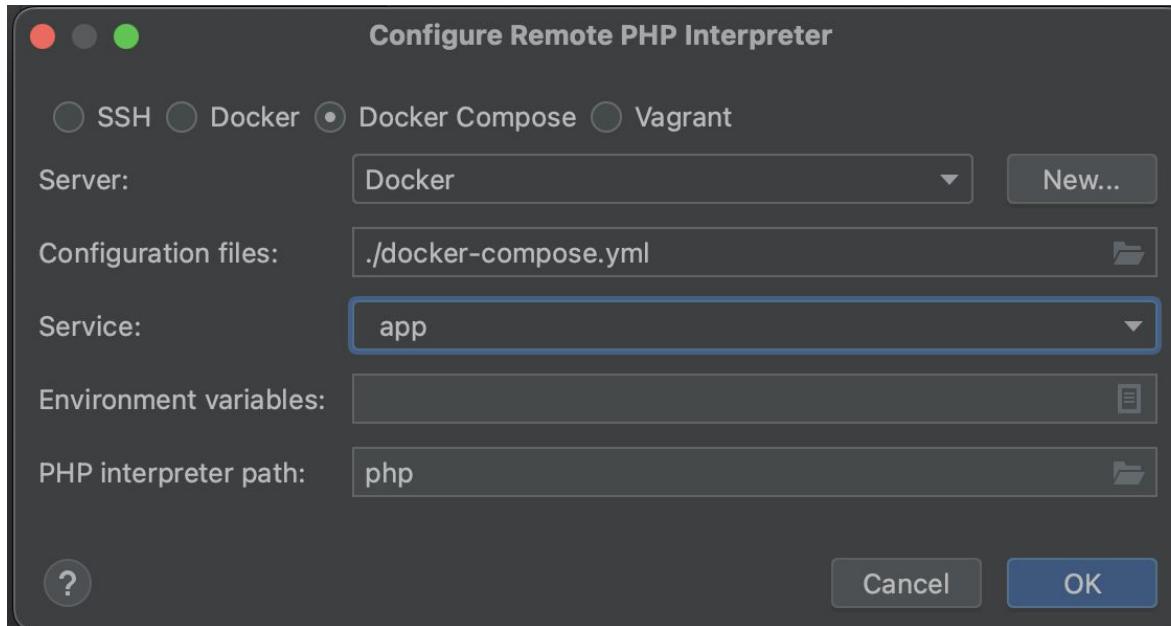
PhpStorm Configuration

◆ Configure CLI interpreter



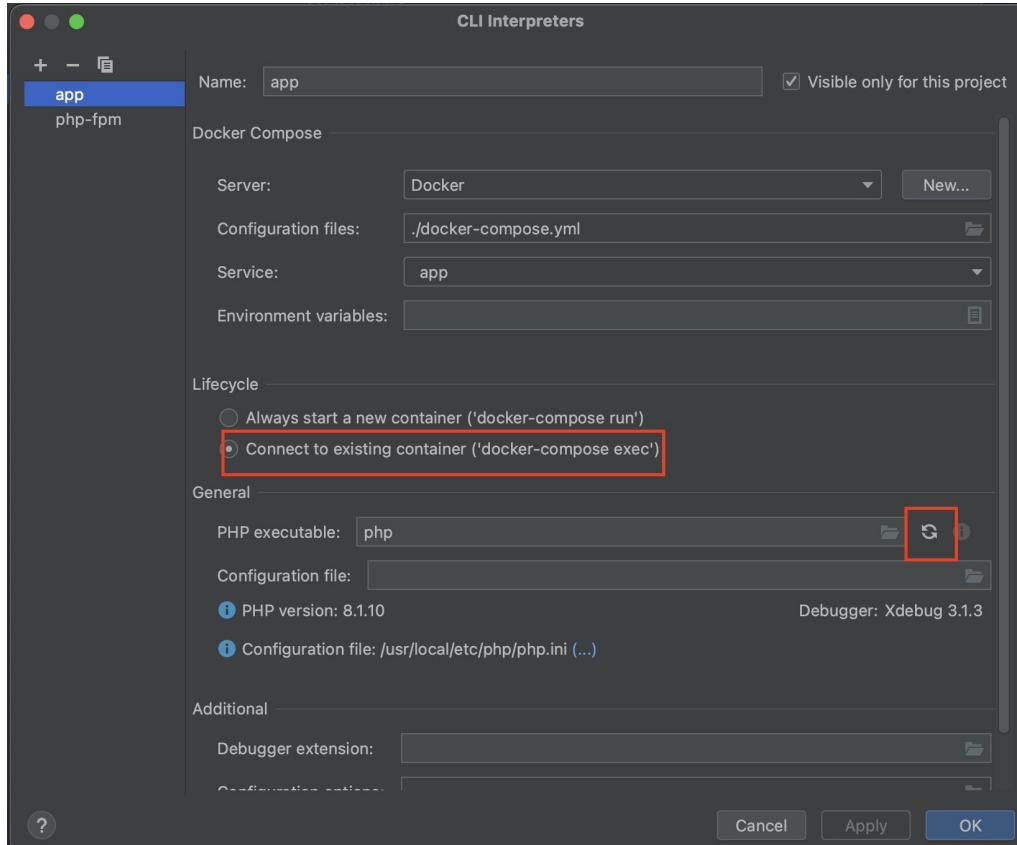
PhpStorm Configuration

- Configure CLI interpreter



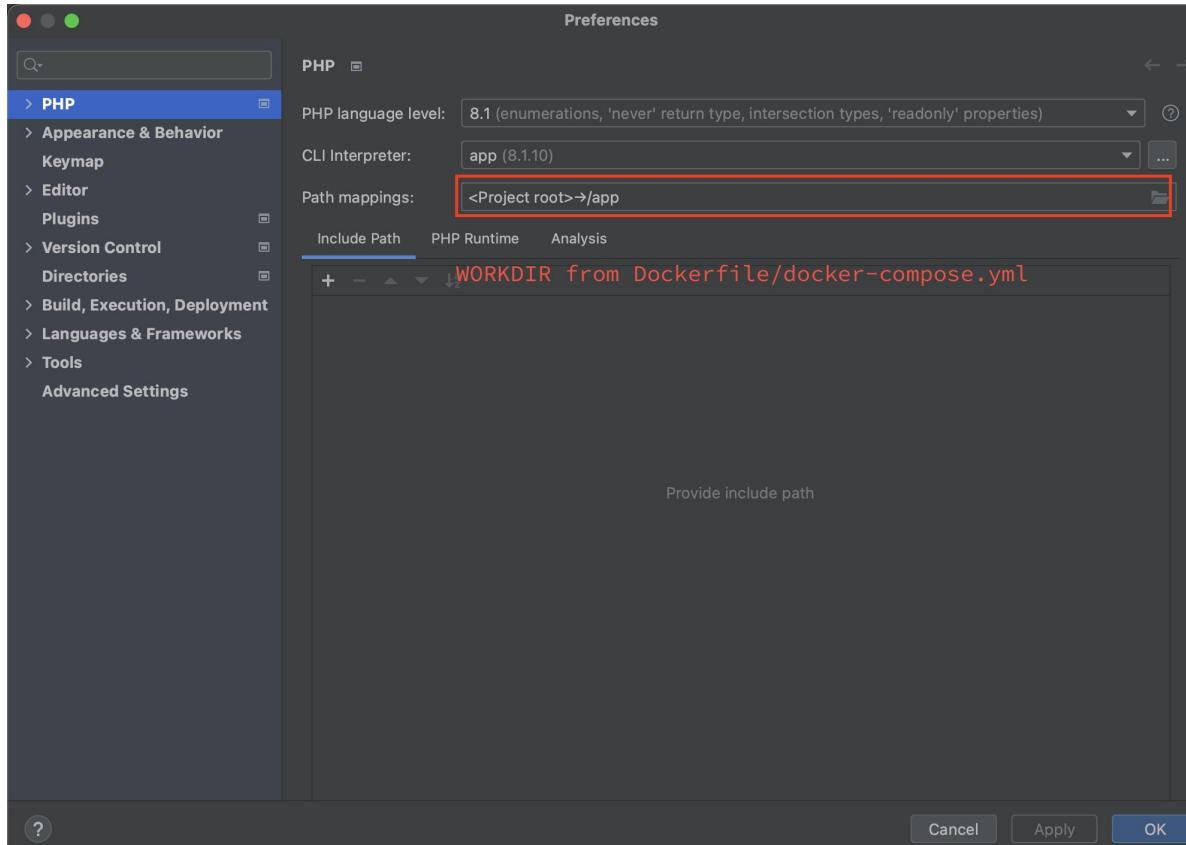
PhpStorm Configuration

◆ Configure CLI interpreter



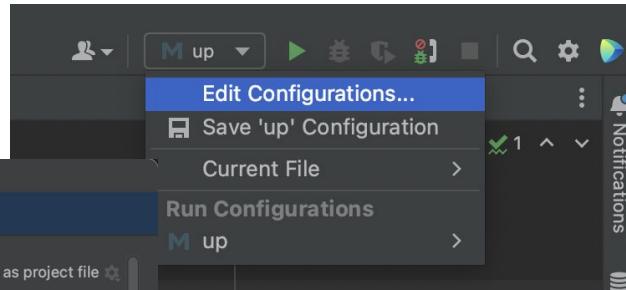
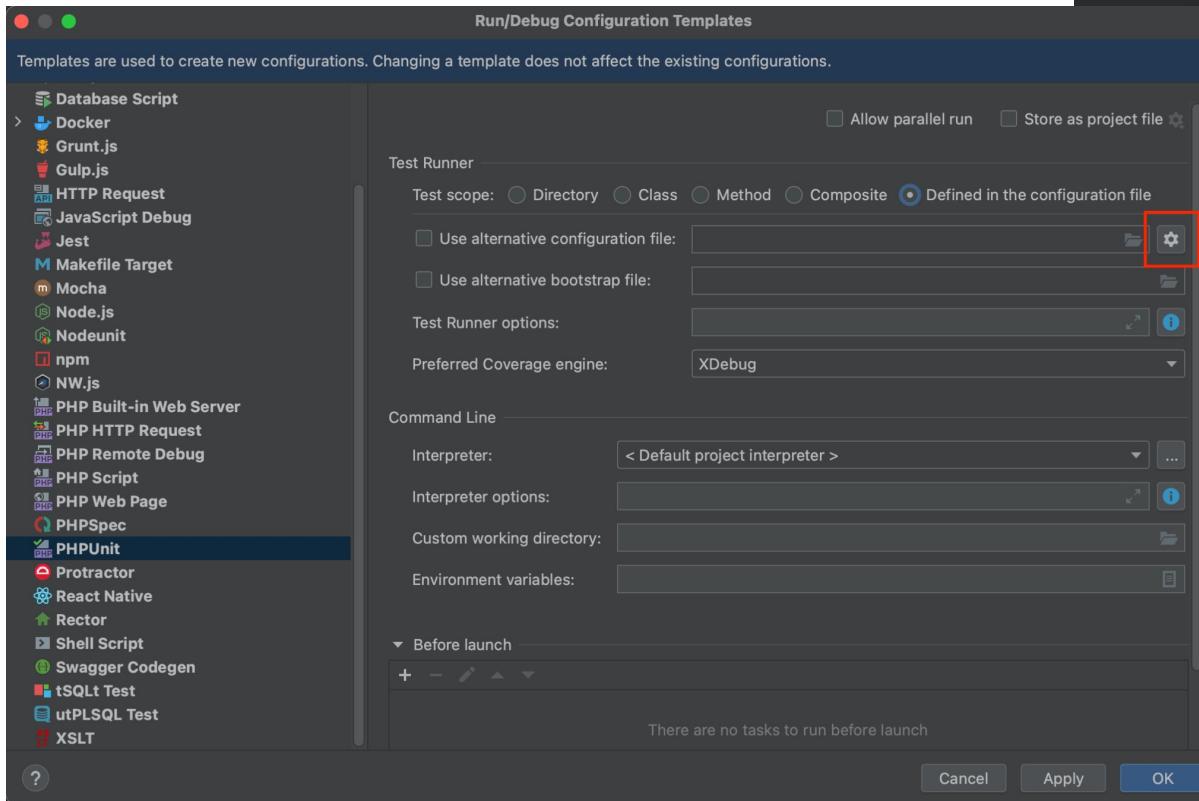
PhpStorm Configuration

◆ Configure CLI interpreter



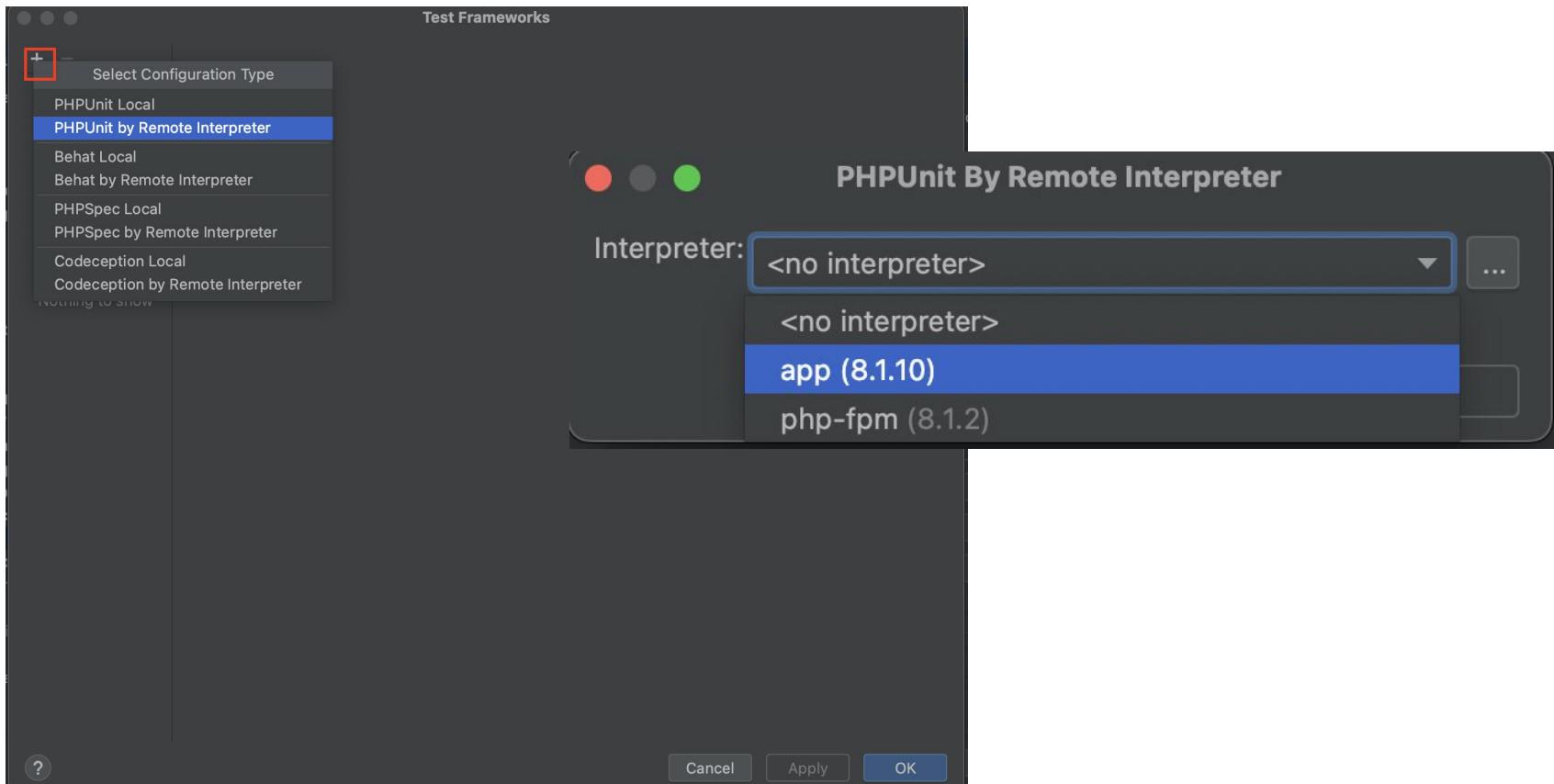
PhpStorm Configuration

◆ Configure PHPUnit



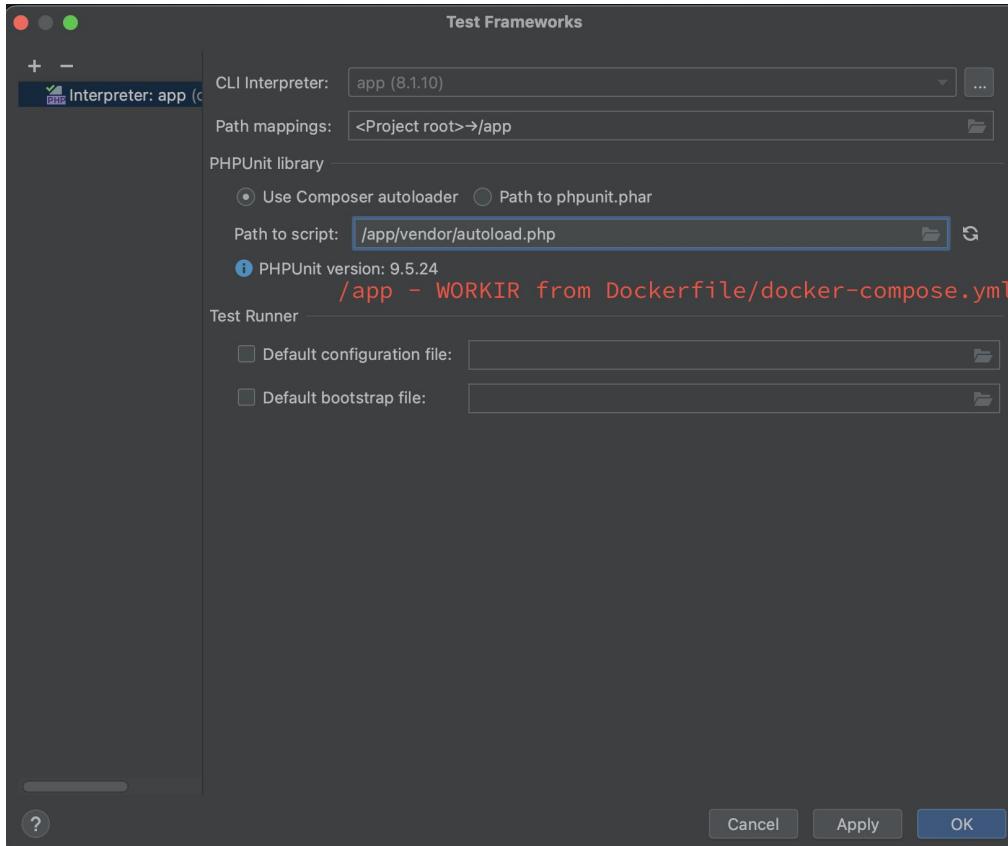
PhpStorm Configuration

◆ Configure PHPUnit



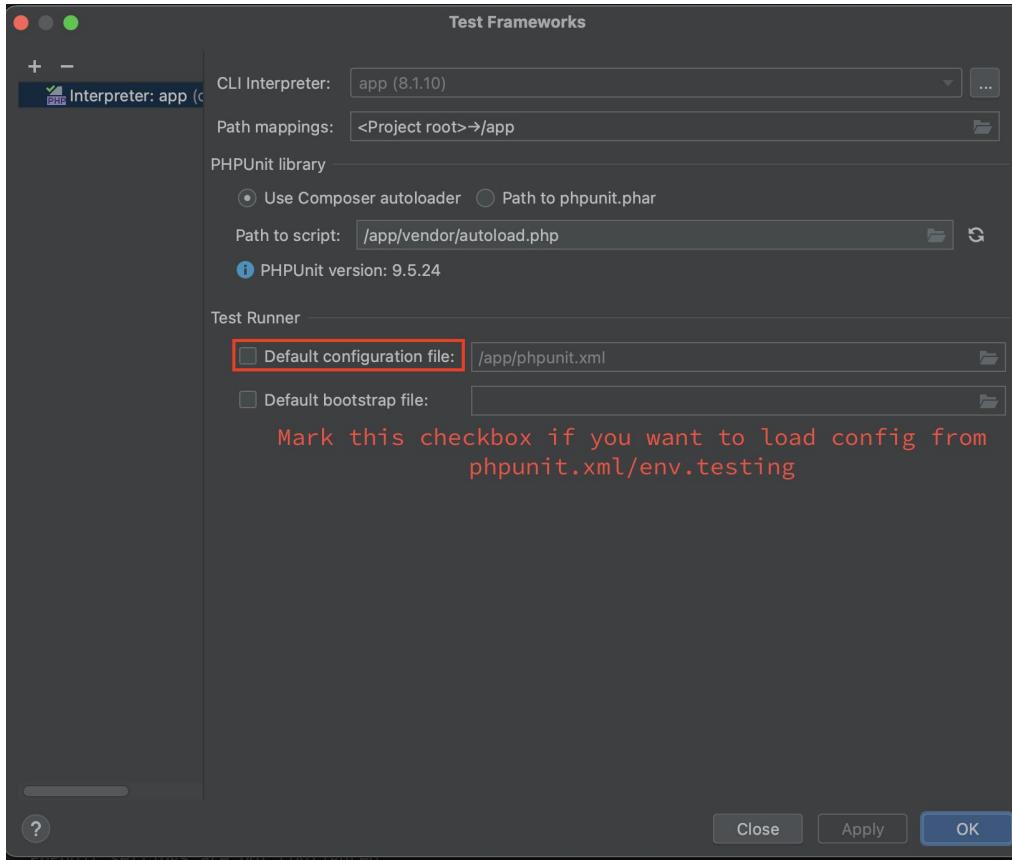
PhpStorm Configuration

◆ Configure PHPUnit



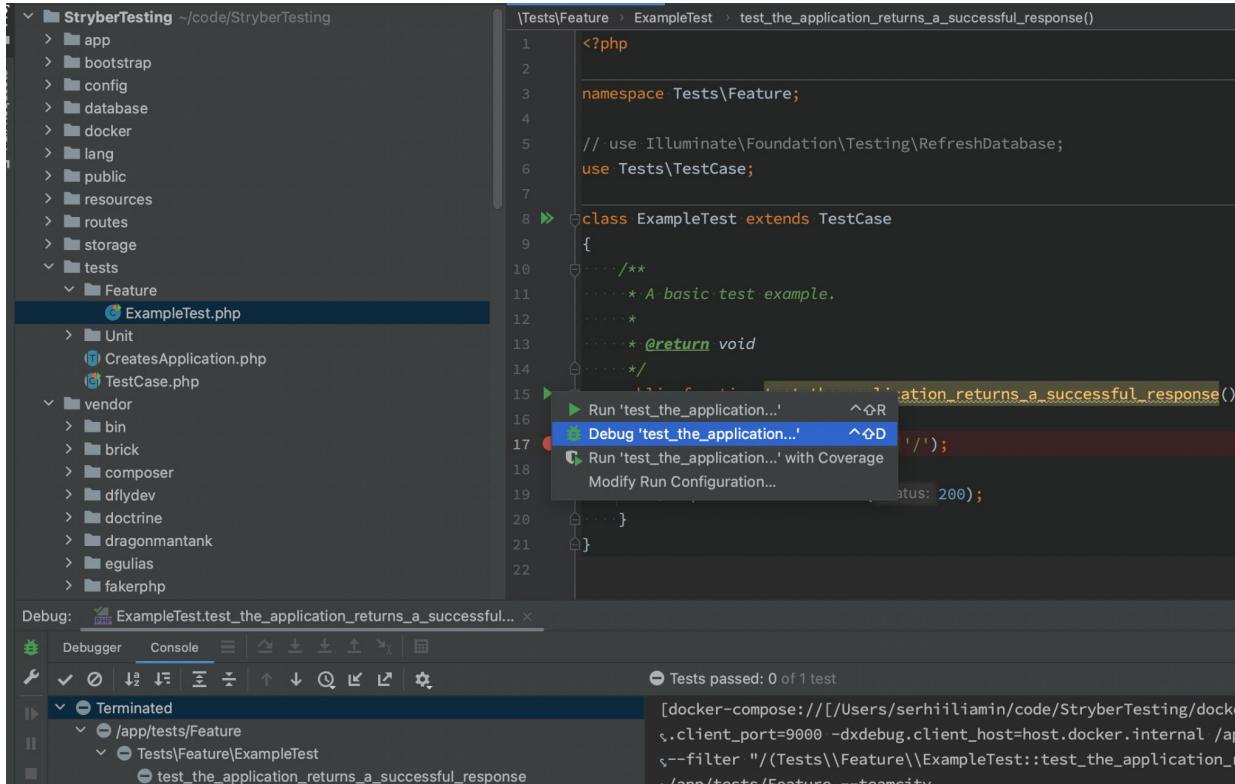
PhpStorm Configuration

◆ Configure PHPUnit



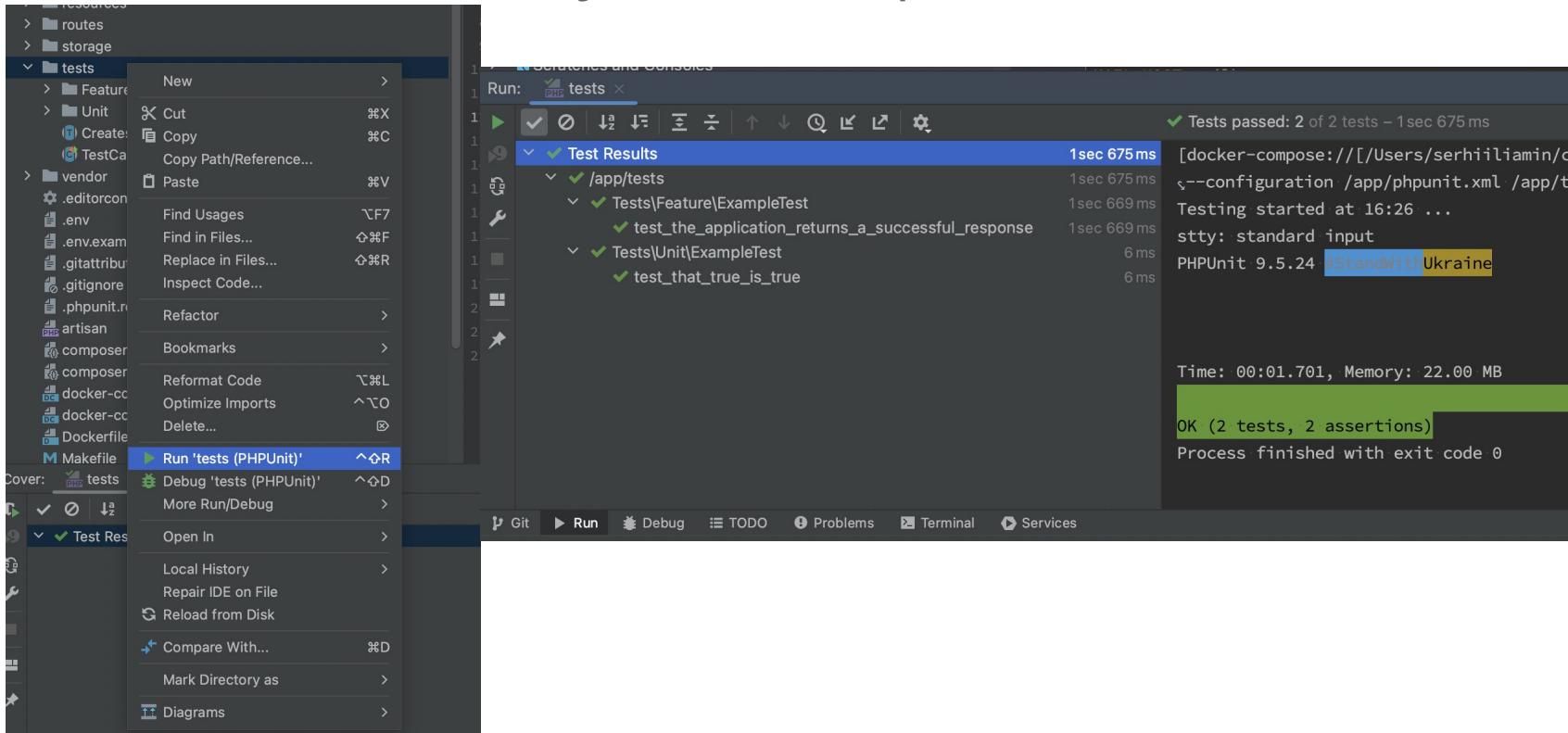
PhpStorm Configuration

- After it you'll be able to run/debug tests directly from PhpStorm



PhpStorm Configuration

- After it you'll be able to run/debug tests directly from PhpStorm



DatabaseTransactions && RefreshDatabase

Laravel provides the **RefreshDatabase** trait to reset the database after each test so that data from a previous test does not interfere with subsequent tests.

The **RefreshDatabase** trait uses the artisan command **migrate:fresh** to drop all the tables from the database and then execute the **migrate** command

DatabaseTransactions && RefreshDatabase

In large applications with hundreds of migrations,
the **migrate:fresh** could potentially slow down your
tests.



DatabaseTransactions && RefreshDatabase

DatabaseTransactions, on the other hand, expects your database to be properly **migrated before your tests start**.



Then, it wraps every test in a database transaction, which it rolls back at the end of each test. This means that, at the end of each test, your database will be returned to the exact same state it was in prior to the test.

Data Providers

Data Providers are a handy feature of PHPUnit which allows you to **run the same test with different inputs** and expected results.

A data provider method must be **public** and either **return an array of arrays** or an **object that implements the *Iterator* interface** and yields an array for each iteration step.

Data Providers

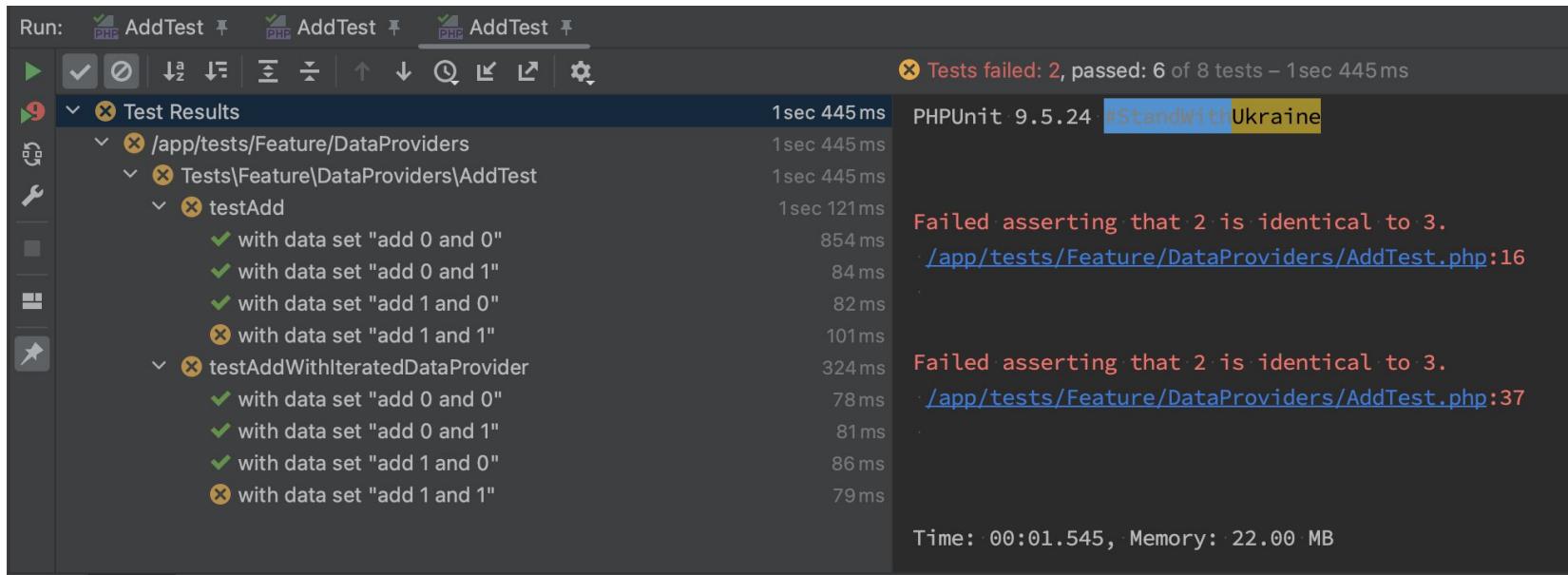
```
11  ...     /**
12  ...      * @dataProvider providerAddSuccess
13  ... */
14  public function testAdd(int $a, int $b, int $expected): void
15  {
16  ...     $this->assertSame($expected, $actual: $a + $b);
17  }
18
19  public function providerAddSuccess(): array
20  {
21  ...     return [
22  ...         /* [$firstArgument, $secondArgument, $expectedResult] */
23  ...         'add 0 and 0' => [0, 0, 0],
24  ...         'add 0 and 1' => [0, 1, 1],
25  ...         'add 1 and 0' => [1, 0, 1],
26  ...         'add 1 and 1' => [1, 1, 3]
27  ...     ];
28  }
```

Data Providers

```
32     ... /**
33      * @dataProvider providerAddSuccessIterated
34     */
35     public function testAddWithIteratedDataProvider(int $a, int $b, int $expected): void
36     {
37         $this->assertSame($expected, actual: $a + $b);
38     }
39
40     public function providerAddSuccessIterated(): iterable
41     {
42         /* [$firstArgument, $secondArgument, $expectedResult] */
43         yield 'add 0 and 0' => [0, 0, 0];
44         yield 'add 0 and 1' => [0, 1, 1];
45         yield 'add 1 and 0' => [1, 0, 1];
46         yield 'add 1 and 1' => [1, 1, 3];
47     }

```

Data Providers



The screenshot shows the PHPStorm test runner interface with three runs of the same test labeled "AddTest". The middle run is selected, displaying the "Test Results" tree and a summary bar.

Test Results Summary:

- PHPUnit 9.5.24
- #StandWithUkraine
- Tests failed: 2, passed: 6 of 8 tests – 1sec 445 ms

Test Tree:

- /app/tests/Feature/DataProviders
- Tests\Feature\DataProviders\AddTest
 - testAdd
 - ✓ with data set "add 0 and 0"
 - ✓ with data set "add 0 and 1"
 - ✓ with data set "add 1 and 0"
 - ✗ with data set "add 1 and 1"
 - testAddWithIteratedDataProvider
 - ✓ with data set "add 0 and 0"
 - ✓ with data set "add 0 and 1"
 - ✓ with data set "add 1 and 0"
 - ✗ with data set "add 1 and 1"

Failed Assertions:

- Failed asserting that 2 is identical to 3.
[/app/tests/Feature/DataProviders/AddTest.php:16](#)
- Failed asserting that 2 is identical to 3.
[/app/tests/Feature/DataProviders/AddTest.php:37](#)

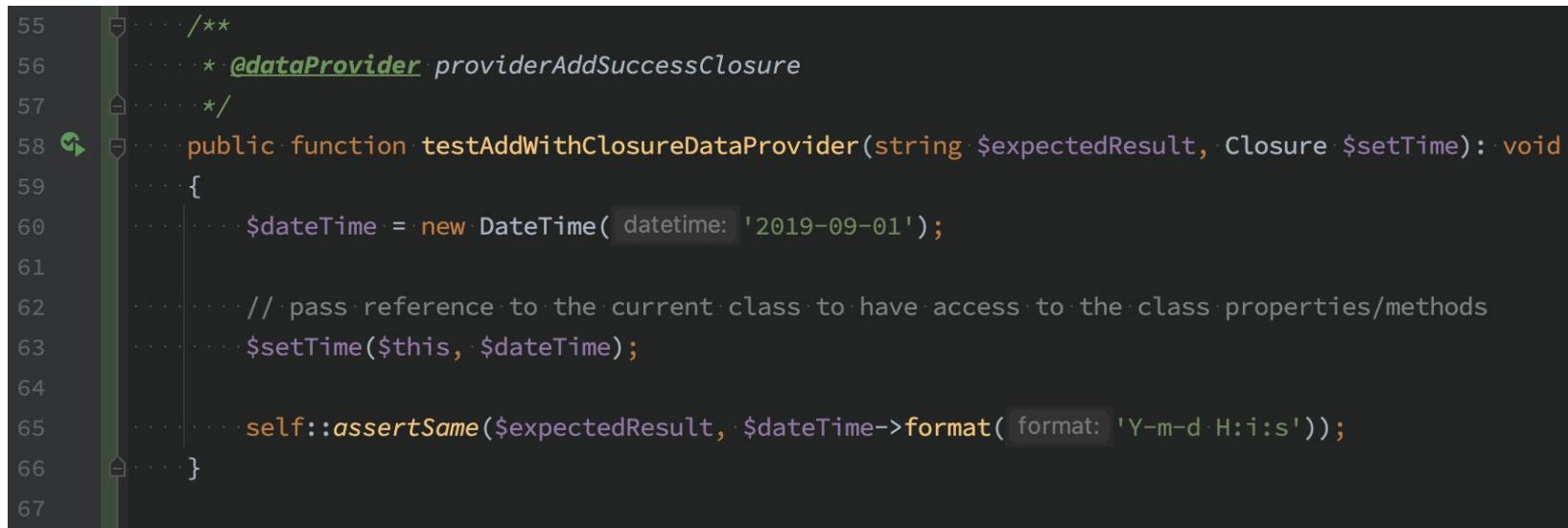
Performance Metrics:

- Time: 00:01.545
- Memory: 22.00 MB

Data Providers Tips:

- ◆ Always **name** the data providers.
- ◆ Add **type definitions**.
- ◆ You can use Closures in Data providers to delay evaluation.
- ◆ Use **yield** to simplify large nested arrays
- ◆ The **disadvantage** of data providers is that they are **evaluated before anything else** (to allow PHPUnit calculate the total number of tests). It means that they can't access anything initialized in `setUpBeforeClass()` or `setUp()`.

Data Providers and access to the Test methods/properties



```
55     /**
56      * @dataProvider providerAddSuccessClosure
57     */
58     public function testAddWithClosureDataProvider(string $expectedResult, Closure $setTime): void
59     {
60         $dateTime = new DateTime( datetime: '2019-09-01' );
61
62         // pass reference to the current class to have access to the class properties/methods
63         $setTime($this, $dateTime);
64
65         self::assertSame($expectedResult, $dateTime->format( format: 'Y-m-d H:i:s' ));
66     }
67 }
```

The screenshot shows a code editor window with a dark theme. On the left, there's a vertical toolbar with a hexagonal icon. The main area displays a PHP test method. The code uses annotations like `@dataProvider` and `@dataProvider` to define closures for test data. It includes imports for `DateTime` and `Closure`, and uses the `self::assertSame` method to assert equality between the expected result and the formatted date-time string.

Data Providers and access to the Test methods/properties

```
67
68     ... // Data Provider with closure
69     public function providerAddSuccessClosure(): iterable
70     {
71         yield 'midnight' => [
72             '2019-09-01 00:00:00',
73             function (AddTest $self, DateTime $date): void {
74                 // do something with current test instance
75                 // $self->user->id
76
77                 $date->setTime( hour: 0, minute: 0, second: 0);
78             },
79         ];
80     }
}
```

UrlGenerator

```
Route::get( uri: '/some-endpoint/{id}', action: 'App\Http\Actions\Some\SomeAction')
    ...->where( name: 'id', expression: '[0-9]+')
    ...->name( name: 'some-endpoint')
;
```

UrlGenerator allow us to get API endpoint URL based on it's name/action

UrlGenerator

```
11  final class SomeEndpointTest extends TestCase
12  {
13      public function testSomeEndpointSuccessResponse(): void
14      {
15          $number = $this->faker->randomNumber( nbDigits: 3);
16
17          $urlByAction = $this->urlGenerator->action( action: SomeAction::class, [
18              'id' -> $number
19          ]);
20
21          $urlByName = $this->urlGenerator->route( name: 'some-endpoint', [
22              'id' -> $number
23          ]);
24
25          $this->assertEquals($urlByAction, $urlByName);
26
27          $this->getJson($urlByAction)
28              ->assertOk()
29              ->assertExactJson([
30                  JsonResource::$wrap -> [
31                      $number
32                  ],
33              ]);
34
35      }
36  }
```

Authentication what to check:

- ◆ Always check that API endpoint isn't accessible fro unauthorized users
- ◆ Authentication flow (log in/log out)
- ◆ HTTP **forbidden** code **403**



Authentication

```
10 trait WithAuthentication
11 {
12     final protected function authAsUser(?User $user = null): User
13     {
14         if (empty($user)) {
15             $user = User::factory()->create();
16         }
17
18         $this->actingAs($user, guard: 'sanctum');
19
20         return $user;
21     }
22
23     protected function logout(): void
24     {
25         /** @uses \Tests\TestCase::setUp() */
26         Auth::guard( name: 'sanctum')->logout();
27     }
28 }
```

Authentication

```
11
12 ⌘ ⌘ > abstract class TestCase extends BaseTestCase
13 {
14     ...use CreatesApplication;
15     ...use DatabaseTransactions;
16     ...use WithFaker;
17     ...// use RefreshDatabase;
18
19     ...protected UrlGenerator $urlGenerator;
20
21 ⌘ ⌘ ↑ ⚡ protected function setUp(): void
22 {
23     ...parent::setUp();
24
25     ...$this->urlGenerator = $this->app->make( abstract: UrlGenerator::class);
26
27     ...// To be able to make user unauthorized
28     ...RequestGuard::macro( name: 'logout', function () {
29         ...$this->user = null;
30     });
31 }
32 }
33 }
```

Testing Events what to check:

- ◆ Event is dispatched with parameters that we expect
- ◆ Listeners are listening for dispatched event
- ◆ Listeners are handling dispatched event in a proper way
- ◆ Listenres were/weren't queued

Testing Events

```
1 <?php
2
3 namespace App\Events;
4
5 use Illuminate\Foundation\Events\Dispatchable;
6 use Illuminate\Queue\SerializesModels;
7
8 final class ApiEndpointCalledEvent
9 {
10     use Dispatchable, SerializesModels;
11
12     /**
13      * Create a new event instance.
14      *
15      * @return void
16     */
17     public function __construct(private string $endpointName)
18     {
19     }
20
21     public function getEndpointName(): string
22     {
23         return $this->endpointName;
24     }
25 }
```

```
1 <?php
2
3 namespace App\Listeners;
4
5 use App\Events\ApiEndpointCalledEvent;
6 use Illuminate\Contracts\Queue\ShouldQueue;
7 use Psr\Log\LoggerInterface;
8
9 final class ListenToApiEndpointCalled implements ShouldQueue
10 {
11     /**
12      * Create the event listener.
13      *
14      * @return void
15     */
16     public function __construct(private LoggerInterface $logger)
17     {
18     }
19
20     /**
21      * Handle the event.
22      *
23      * @param \App\Events\ApiEndpointCalledEvent $event
24      *
25      * @return void
26     */
27     public function handle(ApiEndpointCalledEvent $event): void
28     {
29         $this->logger->info($event->getEndpointName() . ' Endpoint called');
30     }
31 }
```

Testing Events

```
1  <?php
2
3  declare(strict_types=1);
4
5  namespace App\Http\Actions\ActionWithEvent;
6
7  use ...
8
9
10 class EventAction extends Controller
11 {
12     public function __construct(
13         private Dispatcher $eventDispatcher
14     ) {
15     }
16
17     public function __invoke(): JsonResource
18     {
19         // event(new ApiEndpointCalledEvent(Route::currentRouteName()));
20         // ApiEndpointCalledEvent::dispatch(Route::currentRouteName());
21         $this->eventDispatcher->dispatch(
22             new ApiEndpointCalledEvent(Route::currentRouteName())
23         );
24
25         return new JsonResource(['ok']);
26     }
27
28 }
```

Testing Events

```
18  public function testEventWasDispatched(): void
19  {
20      $dispatcher = Event::fake([
21          ApiEndpointCalledEvent::class,
22      ]);
23
24      $routeName = 'create-event';
25      $url = $this->urlGenerator->route($routeName);
26      $this->getJson($url)->assertOk();
27
28      $dispatcher->assertDispatched( event: ApiEndpointCalledEvent::class );
29
30      $dispatcher->assertDispatched(function (ApiEndpointCalledEvent $event) use ($routeName) {
31          return $event->getEndpointName() === $routeName;
32      });
33  }
```

Testing Events

```
48 ► ┌··· public function testListenerIsListening(): void
49 └··· {
50   ┌··· $dispatcher = Event::fake([
51   ···     ApiEndpointCalledEvent::class,
52   ··· ]);
53
54   ···     $routeName = 'create-event';
55   ···     $url = $this->urlGenerator->route($routeName);
56   ···     $this->getJson($url)->assertOk();
57
58   ┌···     $dispatcher->assertListening(
59   ···         expectedEvent: ApiEndpointCalledEvent::class,
60   ···         [ListenToApiEndpointCalled::class, 'handle']
61   ··· );
62 }
```

Testing Events

```
51 ► 51    public function testEventListenerWasQueued(): void
52     52    {
53         53        Queue::fake();
54
55         55        $routeName = 'create-event';
56         56        $url = $this->urlGenerator->route($routeName);
57         57        $this->getJson($url)->assertOk();
58
59         59        // assertNotPushed if listener doesn't implement Illuminate\Contracts\Queue\ShouldQueue
60         60        Queue::assertPushed(
61             61            job: CallQueuedListener::class,
62             62            function ($listener) {
63                 63                 return $listener->class === ListenToApiEndpointCalled::class;
64             64         }
65             65     );
66         66     }
```

Testing Events

```
68     public function testEventListenerWork(): void
69     {
70         /** @var MockInterface|LoggerInterface $listener */
71         $logger = $this->partialMock( abstract: LoggerInterface::class);
72
73         $routeName = 'create-event';
74
75         $logger
76             ->shouldReceive( ...methodNames: 'info')
77             ->with( ...args: $routeName . ' Endpoint called')
78             ->once()
79         ;
80
81         $url = $this->urlGenerator->route($routeName);
82         $response = $this->getJson($url);
83         $response->assertOk();
84     }
85 }
```

Testing Events

```
121  public function testEventListenerWorkOtherWay(): void
122  {
123      /** @var MockInterface|LoggerInterface $listener */
124      $logger = $this->partialMock( abstract: LoggerInterface::class);
125
126      $routeName = 'create-event';
127
128      $logger
129          ->shouldReceive( ...methodNames: 'info')
130          ->with( ...args: $routeName . ' Endpoint called')
131          ->once()
132      ;
133
134      (new ListenToApiEndpointCalled($logger))->handle(new ApiEndpointCalledEvent($routeName));
135  }
136 }
```

Exceptions what to check:

- ◆ Response HTTP code
- ◆ Error response message

For custom API exceptions:

- ◆ Error response code
- ◆ Error response fallback_message
- ◆ Error response payload

Checking exceptions

```
11  class ExceptionAction extends Controller
12  {
13      public function __invoke(): JsonResource
14      {
15          throw new SomeException();
16      }
17  }
```

Checking exceptions

```
8  final class SomeException extends DomainException implements ApiException
9  {
10     ...
11     public const CODE = 'exception.code.for.frontend';
12
13     public function getExceptionCode(): string
14     {
15         return self::CODE;
16     }
17
18     public function getFallbackMessage(): string
19     {
20         return ___('key: exceptions.group.message');
21     }
22
23     public function getPayload(): array
24     {
25         return [];
26     }
27
28     public function getHttpCode(): int
29     {
30         return Response::HTTP_UNPROCESSABLE_ENTITY;
31     }

```

Checking exceptions



```
40     /**
41      * Register the exception handling callbacks for the application.
42      *
43      * @return void
44      */
45     public function register()
46     {
47         $this->reportable(function (Throwable $e) {
48             // ...
49         });
50         $this->renderable(function (ApiException $e, $request) {
51             return (new ApiExceptionResource($e))->toResponse($request);
52         });
53     }

```

Checking exceptions

```
10  /** @mixin ApiException */
11  final class ApiExceptionResource extends JsonResource
12  {
13  ↗    public static $wrap = 'error';
14
15  ↗    public const KEY_CODE = 'code';
16  ↗    public const KEY_FALLBACK_MESSAGE = 'fallback_message';
17  ↗    public const KEY_PAYLOAD = 'payload';
18
19  ↗    public function toArray($request): array
20  {
21  ↗        return [
22  ↗            self::KEY_CODE => $this->getExceptionCode(),
23  ↗            self::KEY_FALLBACK_MESSAGE => $this->getFallbackMessage(),
24  ↗            self::KEY_PAYLOAD => $this->getPayload(),
25  ↗        ];
26  ↗    }
27
28  ↗    public function withResponse($request, $response): void
29  {
30  ↗        $response->setStatusCode($this->getHttpCode());
31  ↗    }
32  ↗}
```

Checking exceptions

```
13 ⚡ final class ExceptionsTest extends TestCase
14 {
15     public function testExceptionIsThrown(): void
16     {
17         $url = $this->urlGenerator->route( name: 'make-exception');
18
19         $response = $this->getJson($url);
20
21         // Exceptions that don't inherit from App\Exceptions\ApiException\ApiException::class
22         // $this->assertStringContainsString(
23         //     ___('exceptions.group.message'),
24         //     $response->json('message')
25     );
26
27         // Exceptions that inherit from App\Exceptions\ApiException\ApiException::class
28         $response
29             ->assertStatus( status: Response::HTTP_UNPROCESSABLE_ENTITY)
30             ->assertJson( fn(AssertableJson $json) =>
31                 $json->has(
32                     'error',
33                     fn(AssertableJson $json) => $json
34                     ->where(
35                         key: ApiExceptionResource::KEY_CODE,
36                         expected: SomeException::CODE
37                     )
38                     ->where( key: ApiExceptionResource::KEY_FALLBACK_MESSAGE, ___( key: 'exceptions.group.message' ))
39                     ->where( key: ApiExceptionResource::KEY_PAYLOAD, [] )
40                     ->etc()
41                 )
42             );
43     }
44 }
```

Validation what to check:

- ◆ Response HTTP code
- ◆ All required parameters
- ◆ Parameters values
(min/max/before/after/in/exists/etc)
- ◆ Custom validation rules messages
- ◆ Validation messages with custom attributes

Testing Validation errors

```
11  /**
12  * @property-read string $pay_day
13  * @property-read string $invoice_cutoff_days
14  * @property-read string $IBAN
15  * @property-read string $some_field
16  */
17  final class ValidationActionRequest extends ApiRequest
18  {
19      public const KEY_PAY_DAY = 'pay_day';
20      public const KEY_INVOICE_CUTOFF_DAYS = 'invoice_cutoff_days';
21      public const KEY_IBAN = 'IBAN';
22      public const KEY_SOME_FIELD = 'some_field';
23
24  ⚡  function rules(): array
25  {
26      return [
27          self::KEY_PAY_DAY => [
28              'required',
29              'integer',
30              'min:1',
31              'max:31',
32          ],
33          self::KEY_INVOICE_CUTOFF_DAYS => [
34              'required',
35              'integer',
36              'min:' . User::MIN_CUTOFF_DAYS,
37              'max:' . User::MAX_CUTOFF_DAYS,
38              new InvoiceCutoffDaysConsideringPayDay((int) $this->invoice_cutoff_days, (int) $this->pay_day),
39          ],
40          self::KEY_IBAN => [
41              'required',
42              'string',
43          ],
44          self::KEY_SOME_FIELD => [
45              'required',
46              'string',
47          ],
48      ];
49  }
50 }
```



Testing Validation errors

```
1 <?php
2
3 namespace App\Rules;
4
5 use Illuminate\Contracts\Validation\Rule;
6
7 final class InvoiceCutoffDaysConsideringPayDay implements Rule
8 {
9     public function __construct(
10         private int $invoiceCutoffDays,
11         private int $payDay,
12     ) {
13     }
14
15     public function passes($attribute, $value): bool
16     {
17         return $this->invoiceCutoffDays <= $this->payDay;
18     }
19
20     public function message(): string
21     {
22         return __(
23             key: 'validation.custom.wrong_invoice_cutoff_days',
24             pay_day' => $this->payDay,
25         );
26     }
27 }
```

Testing Validation errors

```
156     'custom' => [
157         'attribute-name' => [
158             'rule-name' => 'custom-message',
159         ],
160         'wrong_invoice_cutoff_days' => [
161             ValidationActionRequest::KEY_INVOICE_CUTOFF_DAYS
162             . " should be less or equal than :pay_day if "
163             . "
164             ValidationActionRequest::KEY_PAY_DAY
165             . "
166             . ' equals :pay_day'
167             ,
168         ],
169     ],
170     /*
171      |
172      | Custom Validation Attributes
173      |
174      | The following language lines are used to swap our attribute placeholder
175      | with something more reader friendly such as "E-Mail Address" instead
176      | of "email". This simply helps us make our message more expressive.
177      |
178      |
179     */
180     'attributes' => [
181         'IBAN' => 'IBAN',
182     ],
183 ];
```

Testing Validation errors

```
14  ↵    public function testValidationErrorIsThrown(): void
15  {
16      $payDay = $this->faker->numberBetween( int1: 1, int2: 30);
17      $invoiceCutoffDays = $payDay + 1;
18
19      $response = $this->getJson(
20          $this->urlGenerator->action( action: ValidationAction::class,
21              ValidationActionRequest::KEY_PAY_DAY => $payDay,
22              ValidationActionRequest::KEY_INVOICE_CUTOFF_DAYS => $invoiceCutoffDays,
23
24              // Without required field
25              ValidationActionRequest::KEY_IBAN => $this->faker->iban,
26              ValidationActionRequest::KEY_SOME_FIELD => $this->faker->name,
27          ])
28      )->assertUnprocessable();
29
30      $errors = $response->json( key: 'errors');
31  
```

Testing Validation errors

```
32     // =====
33     // Check custom validation message
34
35     $this->assertArrayHasKey( key: ValidationActionRequest::KEY_INVOICE_CUTOFF_DAYS, $errors);
36     $this->assertContains(
37        __(
38             key: 'validation.custom.wrong_invoice_cutoff_days',
39             [
40                 'pay_day' => $payDay
41             ]
42         ),
43         $response->json( key: 'errors.' . ValidationActionRequest::KEY_INVOICE_CUTOFF_DAYS)
44     );
45 
```

Testing Validation errors

```
46     // =====
47     // Check required field validation error with custom validation attribute
48
49     /** @var Translator $translator */
50     $translator = app( abstract: Translator::class);
51
52     $requiredMissingFields = [
53         ValidationActionRequest::KEY_IBAN,
54         ValidationActionRequest::KEY_SOME_FIELD
55     ];
56
57     foreach ($requiredMissingFields as $requiredMissingField) {
58         $this->assertArrayHasKey($requiredMissingField, $errors);
59         $this->assertContains(
60             [
61                 [
62                     [
63                         [
64                             [
65                                 [
66                                     [
67                                         [
68                                             [
69                                                 [
70                                                     [
71                                                         [
72                                                             [
73
74             ],
75         }
76     }
77 }
```



Notifications what to check:

- ◆ Notification was send
- ◆ Notification recipient
- ◆ Notification channels
- ◆ Notification `toArray()` parameters
- ◆ Notification mail attributes(subject/template)
- ◆ Notification mail content
- ◆ Notification was/wasn't queued

Testing Notifications

```
14  final class ApiEndpointCalledNotification extends Notification implements ShouldQueue
15  {
16      ...use Queueable;
17
18      ...public const SUBJECT = 'API endpoint called';
19      ...public const TEMPLATE = 'emails.api-endpoint-called';
20
21      ...public function __construct(
22          ...private string $route
23      ) {
24          ...$this->queue = QUEUE_NOTIFICATIONS;
25      }
26
27      ...public function via(User $notifiable): array
28      {
29          ...return [MailChannel::class];
30      }
31
32      ...public function toMail(User $notifiable): MailMessage
33      {
34          ...return (new MailMessage())
35          ...    ->subject( subject: self::SUBJECT)
36          ...    ->markdown( view: self::TEMPLATE, $this->toArray($notifiable))
37          ...;
38      }
39
40      ...public function toArray(User $notifiable): array
41      {
42          ...return [
43              ...'route' => $this->route,
44          ];
45      }
46  }
```

Testing Notifications

```
15  final class NotificationAction extends Controller
16  {
17      public function __invoke(): JsonResource
18      {
19          try {
20              /** @var Notifiable $user */
21              $user = User::query()->latest()->firstOrFail();
22          } catch (ModelNotFoundException $e) {
23              return new JsonResource([
24                  'no active users'
25              ]);
26          }
27
28          try {
29              // In the real life application it's better to create event and listener for sending email
30              $user->notify(new ApiEndpointCalledNotification(Route::currentRouteName()));
31          } catch (\Throwable $t) {
32              return new JsonResource([
33                  $t->getMessage()
34              ]);
35          }
36
37          return new JsonResource([
38              'Notification send'
39          ]);
40      }
41  }
```

Testing Notifications

```
18     public function testNotificationWasSend(): void
19     {
20         $notification = Notification::fake();
21         $user = User::factory()->create();
22
23         $routeName = 'make-notification';
24         $this->getJson(
25             [
26                 $this->urlGenerator->route($routeName)
27             ]
28         )->assertOk();
29
30         $notification->assertSentTo(
31             [
32                 $user,
33                 notification: ApiEndpointCalledNotification::class,
34                 function ($notification, $channels, $notifiable) use ($user, $routeName) {
35                     // Check channels
36                     $this->assertContains($needle: MailChannel::class, $channels);
37
38                     // Check the data that is passed to the view
39                     $notificationArray = $notification->toArray($user);
40                     $this->assertEquals($routeName, $notificationArray['route']);
41
42                     // Check mail attributes
43                     $mailNotification = (object)$notification->toMail($user);
44                     $this->assertEquals(expected: ApiEndpointCalledNotification::SUBJECT, $mailNotification->subject);
45                     $this->assertEquals(expected: ApiEndpointCalledNotification::TEMPLATE, $mailNotification->markdown);
46                     $this->assertEquals($notificationArray, $mailNotification->viewData);
47
48                 }
49             ]
50         );
51     }
```

Testing Notifications

```
50  public function testNotificationWasQueued(): void
51  {
52      Queue::fake();
53
54      $user = User::factory()->create();
55
56      $this->getJson(
57          $this->urlGenerator->action( action: NotificationAction::class)
58      )->assertOk();
59
60      // assertNotPushed if notification doesn't implement Illuminate\Contracts\Queue\ShouldQueue
61      Queue::assertPushed(
62          job: SendQueuedNotifications::class,
63          function ($job, $queue, $data) use ($user) {
64              $this->assertEquals( expected: ApiEndpointCalledNotification::class, get_class($job->notification));
65              $this->assertEquals( expected: QUEUE_NOTIFICATIONS, $queue);
66              $this->assertContains($user->id, $job->notifiables->pluck('id'));
67
68              return true;
69          }
70      );
71  }
72 }
```

Files what to check:

Download:

- ◆ File name
- ◆ File content



Upload:

- ◆ File validation(size/mime/dimentions/etc)

Testing Files(Upload)

```
11  final class UploadFileAction extends Controller
12  {
13      public function __invoke(UploadFileRequest $request): JsonResource
14      {
15          if (!$request->image) {
16              // remove file from DB...
17              // Storage::delete($user->file)
18
19              return new JsonResource([
20                  'removed'
21              ]);
22          }
23
24
25          $path = Storage::putFile(
26              [
27                  'path' => \App\Infrastructure\Storage::KEY_STORAGE_PATH,
28                  'file' => $request->image
29              ]
30          );
31
32          return new JsonResource([
33              [
34                  'path' => $path
35              ]
36          ]);
37      }
38  }
```

Testing Files(Upload)

```
11  /**
12  * @property-read File|UploadedFile|string|null $image
13  */
14  final class UploadFileRequest extends ApiRequest
15  {
16      public const KEY_IMAGE = 'image';
17
18      public const VALIDATION_MIN_WIDTH = 100;
19      public const VALIDATION_MAX_WIDTH = 2000;
20
21      public const VALIDATION_MIN_HEIGHT = 100;
22      public const VALIDATION_MAX_HEIGHT = 2000;
23
24      public const VALIDATION_MAX_SIZE = 2048;
25
26      public const VALIDATION_ALLOWED_MIMES = ['jpg', 'png', 'jpeg', 'gif', 'svg'];
27
28  ⚡  public function rules(): array
29  {
30      return [
31          self::KEY_IMAGE => [
32              'sometimes',
33              'nullable',
34              'image',
35              'mimes:' . implode( separator: ',', array: self::VALIDATION_ALLOWED_MIMES),
36              'max:' . self::VALIDATION_MAX_SIZE,
37              'dimensions:min_width=' . self::VALIDATION_MIN_WIDTH . ',min_height=' . self::VALIDATION_MIN_HEIGHT
38              . ',max_width=' . self::VALIDATION_MAX_WIDTH . ',max_height=' . self::VALIDATION_MAX_HEIGHT,
39          ],
40      ];
41  }
42 }
```

Testing Files(Upload)

```
15     public function testUploadImageSuccess(): void
16     {
17         $storage = Storage::fake();
18
19         // GD library required
20         $fakeImage = UploadedFile::fake()->image(
21             name: 'photo1.jpg',
22             $this->faker->numberBetween(
23                 int1: UploadFileRequest::VALIDATION_MIN_WIDTH,
24                 int2: UploadFileRequest::VALIDATION_MAX_WIDTH
25             ),
26             $this->faker->numberBetween(
27                 int1: UploadFileRequest::VALIDATION_MIN_HEIGHT,
28                 int2: UploadFileRequest::VALIDATION_MAX_HEIGHT
29             ),
30             );
31
32         $response = $this->postJson($this->urlGenerator->route( name: 'upload-file'), [
33             UploadFileRequest::KEY_IMAGE => $fakeImage
34         ]);
35
36         $response->assertOk();
37         $storage->assertExists($response->json( key: 'data.0'));
38     }
```

Testing Files(Upload)

The screenshot shows a code editor with a dark theme. On the left, there's a sidebar icon resembling a hexagon with internal dots. The main area displays a PHP test file with syntax highlighting. A yellow arrow points to line 43, which defines a test function. The code uses the `Storage::fake()` method to create a fake file and then posts it to a route named 'upload-file'. The response is asserted to have a status of `HTTP_UNPROCESSABLE_ENTITY`. The error message is checked to contain a key 'validation.dimensions' with an attribute pointing to the `UploadFileRequest::KEY_IMAGE` constant. A tooltip or callout box highlights the `UploadFileRequest::KEY_IMAGE` constant, which is defined in another part of the code.

```
40     /**
41      * @dataProvider providerWrongDimensions
42     */
43     final public function testUploadImageDimensionErrors(int $width, int $height): void
44     {
45         Storage::fake();
46
47         $fakeImage = UploadedFile::fake()->image(
48             name: 'photo1.jpg',
49             $width,
50             $height,
51         );
52
53         $response = $this->postJson($this->urlGenerator->route( name: 'upload-file' ), [
54             UploadFileRequest::KEY_IMAGE => $fakeImage
55         ]);
56
57         $response->assertStatus( status: Response::HTTP_UNPROCESSABLE_ENTITY );
58
59         $this->assertContains(
60             __( key: 'validation.dimensions', ['attribute' => UploadFileRequest::KEY_IMAGE] ),
61             $response->json( key: 'errors.' . UploadFileRequest::KEY_IMAGE )
62         );
63     }
64
65     final public function providerWrongDimensions(): iterable
66     {
67         yield 'less_than_min' => [
68             UploadFileRequest::VALIDATION_MIN_WIDTH - 1,
69             UploadFileRequest::VALIDATION_MIN_HEIGHT - 1
70         ];
71
72         yield 'greater_than_max' => [
73             UploadFileRequest::VALIDATION_MAX_WIDTH + 1,
74             UploadFileRequest::VALIDATION_MAX_HEIGHT + 1
75         ];
76     }
77 }
```

Testing Files(Upload)

```
77 ► 77     final public function testUploadImageSizeErrors(): void
78     {
79         Storage::fake();
80
81         $fakeImage = UploadedFile::fake()->create(
82             [
83                 'name' => 'photo1.jpg',
84                 'kilobytes' => UploadFileRequest::VALIDATION_MAX_SIZE + 1
85             ]
86         );
87
88         $response = $this->postJson($this->urlGenerator->route([
89             'name' => 'upload-file'
90         ]),
91         [
92             'key' => [
93                 'attribute' => UploadFileRequest::KEY_IMAGE,
94                 'max' => UploadFileRequest::VALIDATION_MAX_SIZE
95             ],
96             'key' => [
97                 'errors.' . UploadFileRequest::KEY_IMAGE
98             ]
99         ]
100     );
101
102     $response->assertStatus(Response::HTTP_UNPROCESSABLE_ENTITY);
103     $this->assertContains(
104         [
105             [
106                 'key' => [
107                     'validation.max.file' => [
108                         [
109                             'attribute' => UploadFileRequest::KEY_IMAGE,
110                             'max' => UploadFileRequest::VALIDATION_MAX_SIZE
111                         ]
112                     ]
113                 ]
114             ]
115         ],
116         $response->json('errors.' . UploadFileRequest::KEY_IMAGE)
117     );
118 }
```

Testing Files(Upload)

```
100 ► 100 final public function testUploadImageMimeErrors(): void
101 {
102     Storage::fake();
103
104     $fakeImage = UploadedFile::fake()->create(
105         name: 'photo1.txt',
106         $this->faker->numberBetween( int1: 1, int2: UploadFileRequest::VALIDATION_MAX_SIZE),
107     );
108
109     $response = $this->postJson($this->urlGenerator->route( name: 'upload-file'), [
110         UploadFileRequest::KEY_IMAGE => $fakeImage
111     ]);
112
113     $response->assertStatus( status: Response::HTTP_UNPROCESSABLE_ENTITY);
114     $this->assertContains(
115         __( key: 'validation.mimes', [
116             'attribute' => UploadFileRequest::KEY_IMAGE,
117             'values' => implode( separator: ', ', array: UploadFileRequest::VALIDATION_ALLOWED_MIMES)
118         ]),
119         $response->json( key: 'errors.' . UploadFileRequest::KEY_IMAGE)
120     );
121 }
```

Testing Files(Download)

```
7  class DownloadFileAction
8  {
9      public const FILENAME = 'test';
10
11     public function __invoke(): StreamedResponse
12     {
13         return response()->streamDownload(function () {
14             echo $this->generateFile();
15         }, [name: self::FILENAME . '.csv']);
16     }
17
18     private function generateFile(): string
19     {
20         $maxMemoryMb = 1024 * 1024 * 3;
21         $handle = fopen( filename: 'php://temp/maxmemory:' . $maxMemoryMb, mode: 'w' );
22         $fields = [
23             'column1' => 'value1',
24             'column2' => 'value2',
25         ];
26
27         // Header
28         fputcsv($handle, array_keys($fields));
29
30         // Body
31         fputcsv($handle, $fields);
32
33         rewind($handle);
34         $csv = stream_get_contents($handle);
35
36         fclose($handle);
37
38         return $csv;
39     }
40 }
```

Testing Files(Download)

```
12  public function testDownloadSuccess(): void
13  {
14      $this->getJson($this->urlGenerator->route('name: 'download-file'))
15          ->assertOk()
16          ->assertDownload('filename: DownloadFileAction::FILENAME' . '.csv')
17      ;
18  }
19 }
```

Testing Files(Download)

```
20     public function testFileContent(): void
21     {
22         $content = $this->getJson($this->urlGenerator->route(['name' => 'download-file']))->streamedContent();
23         $content = explode(PHP_EOL, $content);
24
25         // last line is empty
26         array_pop(&array: $content);
27
28         $header = null;
29         foreach ($content as $key => $line) {
30             if ($key === 0){
31                 $header = $line;
32             } else {
33                 $data = array_combine(str_getcsv($header), str_getcsv($line));
34                 $this->debug([
35                     '$data' => $data,
36                     '$header' => $header
37                 ]);
38
39                 // Your file content checking logic goes here:
40                 $this->assertEquals(
41                     expected: 'value1',
42                     $data['column1']
43                 );
44
45                 $this->assertEquals(
46                     expected: 'value2',
47                     $data['column2']
48                 );
49             }
50         }
51     }
```

Response structures what to check:

- ◆ Each field inside response



Testing response structures

```
17     Route::middleware(['auth:sanctum'])->group(function () {
18         /** @uses App\Http\Actions\ActionWithResourceResponse\ResourceAction */
19         Route::get('uri: /user/{with_group?}', [action: 'App\Http\Actions\ActionWithResourceResponse\ResourceAction'])
20             ->name( name: 'user-info')
21         ...
22     });
23
24
25 final class ResourceAction extends Controller
26 {
27     public function __invoke(Authenticatable $user, $withGroup = null): UserResource|UserWithGroupResource
28     {
29         assert( assertion: $user instanceof User);
30
31         return (bool)$withGroup ? new UserWithGroupResource($user) : new UserResource($user);
32     }
33 }
```

Testing response structures

```
9  /** @mixin \App\Models\User */
10 final class UserWithGroupResource extends JsonResource
11 {
12     public const KEY_FIRSTNAME = User::COLUMN_FIRSTNAME;
13     public const KEY_LASTNAME = User::COLUMN_LASTNAME;
14     public const KEY_USERNAME = User::COLUMN_USERNAME;
15     public const KEY_EMAIL = User::COLUMN_EMAIL;
16     public const KEY_IMAGE = User::COLUMN_IMAGE;
17     public const KEY_GROUP = User::RELATION_GROUP;
18
19     public function toArray($request): array
20     {
21         return [
22             self::KEY_FIRSTNAME => $this->firstname,
23             self::KEY_LASTNAME => $this->lastname,
24             self::KEY_USERNAME => $this->username,
25             self::KEY_EMAIL => $this->email,
26             self::KEY_IMAGE => $this->image,
27             self::KEY_GROUP => new GroupResource($this->group),
28         ];
29     }
30 }
```

Testing response structures

```
8  /★ @mixin \App\Models\Group */
9  final class GroupResource extends JsonResource
10 {
11     public const KEY_ID = Group::COLUMN_ID;
12     public const KEY_NAME = Group::COLUMN_NAME;
13     public const KEY_CODE = Group::COLUMN_CODE;
14
15     public function toArray($request): array
16     {
17         return [
18             self::KEY_ID => $this->id,
19             self::KEY_NAME => $this->name,
20             self::KEY_CODE => $this->code,
21         ];
22     }
23 }
```

Testing response structures

```
7 ⚡ abstract class Structure
8 {
9 ⚡     abstract public function getStructure(): array;
10
11    final public function getCollectionStructure(): array
12    {
13        return ['*' => $this->getStructure()];
14    }
15}
```

Testing response structures

```
12     final class UserWithGroupStructure extends Structure
13     {
14         public function getStructure(): array
15         {
16             return [
17                 UserResource::KEY_FIRSTNAME,
18                 UserResource::KEY_LASTNAME,
19                 UserResource::KEY_USERNAME,
20                 UserResource::KEY_EMAIL,
21                 UserResource::KEY_IMAGE,
22                 UserWithGroupResource::KEY_GROUP => (new GroupStructure())->getStructure()
23             ];
24         }
25     }
```

Testing response structures

```
14 final class ResponseStructureTest extends TestCase
15 {
16     use WithAuthentication;
17
18     public function testResponseStructure(): void
19     {
20         $user = User::factory()->create();
21         $this->authAsUser($user);
22
23         $groupShouldBeInResponse = $this->faker->boolean();
24         $response = $this->getJson($this->urlGenerator->route( name: 'user-info', [
25             'with_group' => $groupShouldBeInResponse
26         ]));
27         $response->assertOk();
28
29         $structure = $groupShouldBeInResponse
30             ? new UserWithGroupStructure()
31             : new UserStructure()
32         ;
33
34         $response->assertJsonStructure([
35             /** @see Structure::getCollectionStructure() for collections **/
36             JsonResource::$wrap => $structure->getStructure()
37         ]);
38     }
39 }
```

Interacting With Time

```
11  final class TimeAction extends Controller
12  {
13      public const INVASION_DATE = '2022-02-24';
14
15      public const RESPONSE_DEFAULT = '❤️';
16      public const RESPONSE_AFTER_INVASION = 'Russian 🇷🇺 warship go f***ck yourself!';
17
18      public function __invoke(): JsonResource
19      {
20          $response = TimeAction::RESPONSE_DEFAULT;
21
22          $currentTime = Carbon::now();
23          $invasionDate = Carbon::createFromFormat('Y-m-d', time: TimeAction::INVASION_DATE);
24
25          if ($currentTime->gt($invasionDate)) {
26              $response = TimeAction::RESPONSE_AFTER_INVASION;
27          }
28
29          return new JsonResource([$response]);
30      }
31  }
```

Interacting With Time

```
12  final class TimeMachineTest extends TestCase
13  {
14      public function testMessageBasedOnCurrentDate(): void
15      {
16          $currentDate = $this->faker->dateTimeBetween( startDate: '-2 years', endDate: '+2 years')->format( format: 'Y-m-d');
17
18          $now = Carbon::createFromFormat( format: 'Y-m-d', $currentDate)
19          ->setTime( hour: 0, minute: 0)
20          ;
21          $this->travelTo($now);
22
23          $expectedResponse = ($currentDate < TimeAction::INVASION_DATE)
24              ? TimeAction::RESPONSE_DEFAULT
25              : TimeAction::RESPONSE_AFTER_INVASION
26          ;
27
28          $url = $this->urlGenerator->action( action: TimeAction::class);
29
30          $this->getJson($url)
31          ->assertOk()
32          ->assertExactJson([
33              JsonResource::$wrap => [
34                  $expectedResponse
35              ],
36              []
37          );
38
39          $this->debug([$currentDate, $expectedResponse]);
40      }
41  }
```

Parallel running

```
> php artisan test --parallel
```

Laravel automatically handles creating and migrating a test database for each parallel process that is running your tests.

The test databases will be suffixed with a process token which is unique per process.

Code coverage

```
> XDEBUG_MODE=coverage php artisan test --parallel --coverage
```

phpunit.xml



```
15 <coverage processUncoveredFiles="true">
16   <include>
17     <directory suffix=".php">./app</directory>
18   </include>
19   <report>
20     <html outputDirectory="tests/Coverage/html"/>
21   </report>
22 </coverage>
```

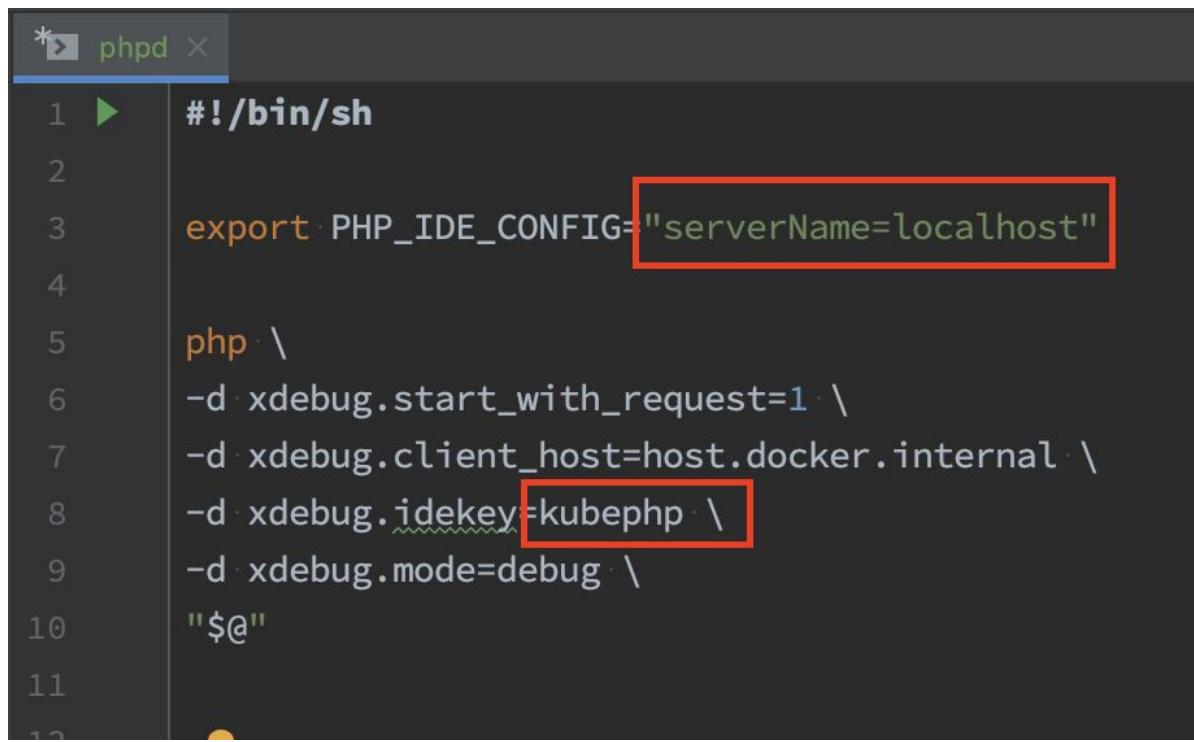
Mocking behavior

```
13  final class HttpRequestAction extends Controller
14  {
15      public const API_URL = 'https://cat-fact.herokuapp.com/facts';
16      public const API_HTTP_METHOD = 'GET';
17
18      public function __construct(
19          private ClientInterface $client,
20          private LoggerInterface $logger
21      ) {
22          ...
23      }
24
25      public function __invoke(): JsonResource
26      {
27          try {
28              $facts = $this->client->request(HttpRequestAction::API_HTTP_METHOD, HttpRequestAction::API_URL);
29          } catch (GuzzleException $e) {
30              $this->logger->error('Request failed', ['error' => $e]);
31
32              throw $e;
33          }
34
35          return new JsonResource(json_decode($facts->getBody()->getContents()));
36      }
37  }
```

Mocking behavior

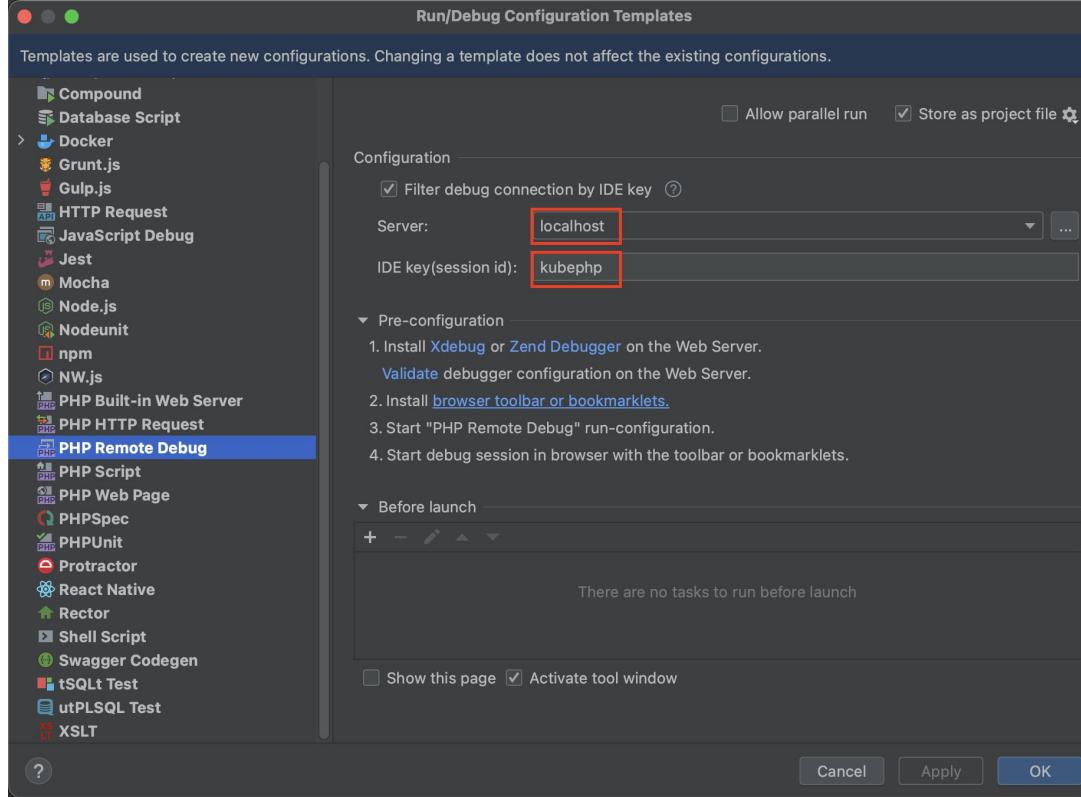
```
17  public function testMakingHttpRequestSuccess()
18  {
19      $url = $this->urlGenerator->action( action: HttpRequestAction::class);
20
21      $mockedData = '[{"status":{"verified":true,"sentCount":1},"_id":"58e008800aac31001185ed07","user
22
23      /**
24          * Also @see TestCase::partialMock
25          * You may use the partialMock method when you only need to mock a few methods of an object.
26          * The methods that are not mocked will be executed normally
27          *
28          * Also @see TestCase::createPartialMock
29          * Also @see Mockery::mock
30          * Also @see MockInterface::makePartial
31          */
32
33      $this->mock( abstract: ClientInterface::class, function (MockInterface $mock) use ($mockedData) {
34          $mock
35              ->shouldReceive( ...methodNames: 'request')
36              ->with( ...args: HttpRequestAction::API_HTTP_METHOD, HttpRequestAction::API_URL)
37              ->once()
38              ->andReturn(
39                  new Response(
40                      $status = 200,
41                      $headers = [],
42                      $mockedData
43                  )
44              );
45      });
46
47      $this->getJson($url)
48          ->assertOk()
49          ->assertJson([
50              JsonResource::$wrap => [
51                  json_decode($mockedData, associative: true)[0]
52              ],
53              ],
54          );
55  }
```

Bonus: Debugging CLI

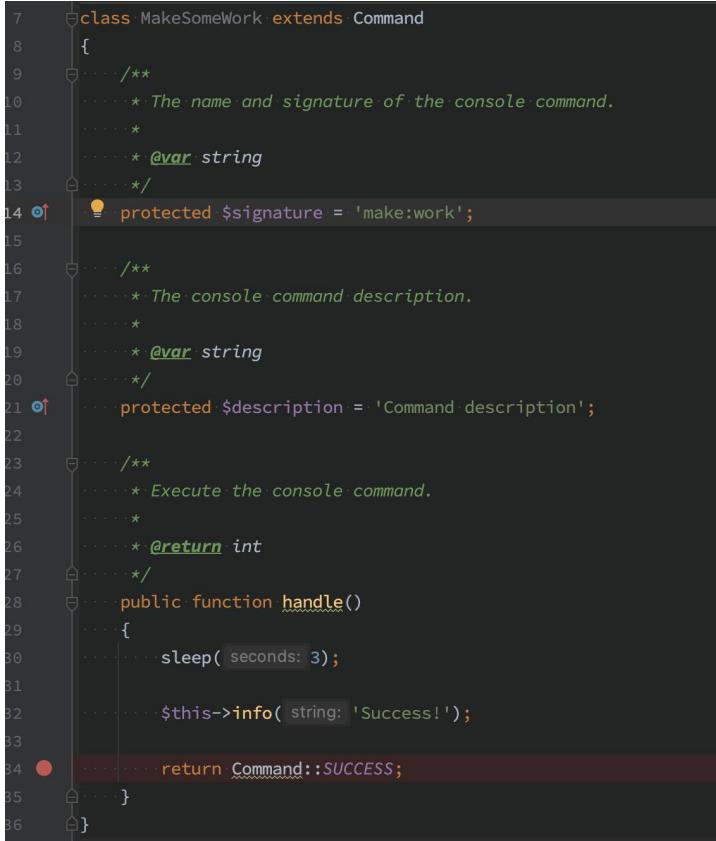


```
* ➜ phpd ✘
1 ► #!/bin/sh
2
3 export PHP_IDE_CONFIG="serverName=localhost"
4
5 php \ 
6 -d xdebug.start_with_request=1 \
7 -d xdebug.client_host=host.docker.internal \
8 -d xdebug.idekey=kubephp \
9 -d xdebug.mode=debug \
10 "$@"
11
12
```

Bonus: Debugging CLI

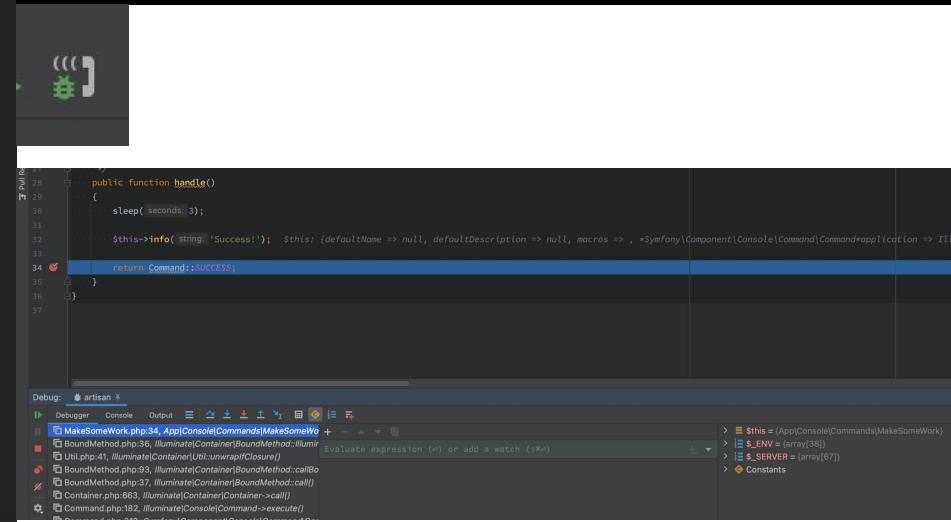


Bonus: Debugging CLI



```
7 class MakeSomeWork extends Command
8 {
9     /**
10      * The name and signature of the console command.
11      *
12      * @var string
13      */
14     protected $signature = 'make:work';
15
16     /**
17      * The console command description.
18      *
19      * @var string
20      */
21     protected $description = 'Command description';
22
23     /**
24      * Execute the console command.
25      *
26      * @return int
27      */
28     public function handle()
29     {
30         sleep( seconds: 3 );
31
32         $this->info( string: 'Success!' );
33
34         return Command::SUCCESS;
35     }
36 }
```

```
/app $ php artisan make:work
Success!
/app $ ./php artisan make:work
```



```
28
29
30     sleep( seconds: 3 );
31
32     $this->info( string: 'Success!' );
33
34     return Command::SUCCESS;
35
36 }
```

Debug: artisan

- MakeSomeWork.php:34, App\Console\Commands\MakeSomeWork +
- BoundMethod.php:36, Illuminate\Container\BoundMethod::Illuminate\Container\{closure}
- Util.php:41, Illuminate\Container\Util::unwrap(Closure)
- BoundMethod.php:93, Illuminate\Container\BoundMethod::call()
- BoundMethod.php:37, Illuminate\Container\BoundMethod::call()
- Container.php:693, Illuminate\Container\Container->call()
- Command.php:182, Illuminate\Console\Command->execute()
- Command.php:149, Illuminate\Console\Command->handle()

\$this = /App\Console\Commands\MakeSomeWork

\$ENV = (array:38)

\$SERVER = (array:47)

Constants

Memory issues with GitHub Actions

- If you will face memory issues with GitHub Actions during running tests just split execution into multiple steps:

```
> php artisan test --env=testing tests/Feature/Admin
```

```
> php artisan test --env=testing tests/Feature/Auth
```

...

Tips:

- ◆ Don't use the Laravel App when you don't need to: If your tests extend from **Tests\TestCase** which uses the **CreatesApplication** trait, a new Laravel App is built for each test which takes a bit of time. This includes initialising the service-providers, detecting routes, and more.

If you don't need Laravel's functionality in a test, extend from **PHPUnit\Framework\TestCase** instead.

Useful links:

- ◆ <https://laravel.com/docs/9.x/testing>
- ◆ <https://blog.martinhujer.cz/how-to-use-data-providers-in-phpunit/>
- ◆ <https://www.code-distortion.net/books/fast-test-databases/>
- ◆ <https://mayahi.net/laravel/make-refresh-database-trait-much-faster/>
- ◆ <https://prinsfrank.nl/2022/09/20/How-to-write-decoupled-unit-tests-in-Laravel>
- ◆ <https://laravelpackage.com/>
- ◆ <https://devinthewild.com/blog/2022-05-31/testing-that-event-listeners-are-dispatched-in-laravel>
- ◆ <https://dev.to/macelux/how-to-test-mail-and-database-notifications-in-laravel-8-178o>
- ◆ <https://ralphjsmit.com/test-laravel-notification-e-mail-contents>



Thank you !

Authored By Liamin Serhii

