DevConf

www.devconf.ru

[Larabeer] Мифы и реальность unit и не-unit тестирования в Laravel

Адель Файзрахманов

Unit test

Live coding

Unit test

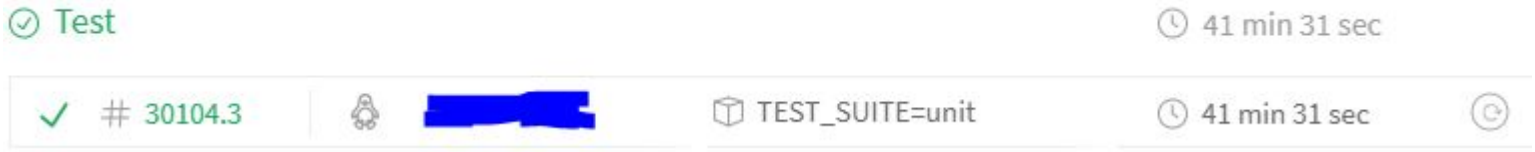
Быстрый

Не прерывать состояние потока

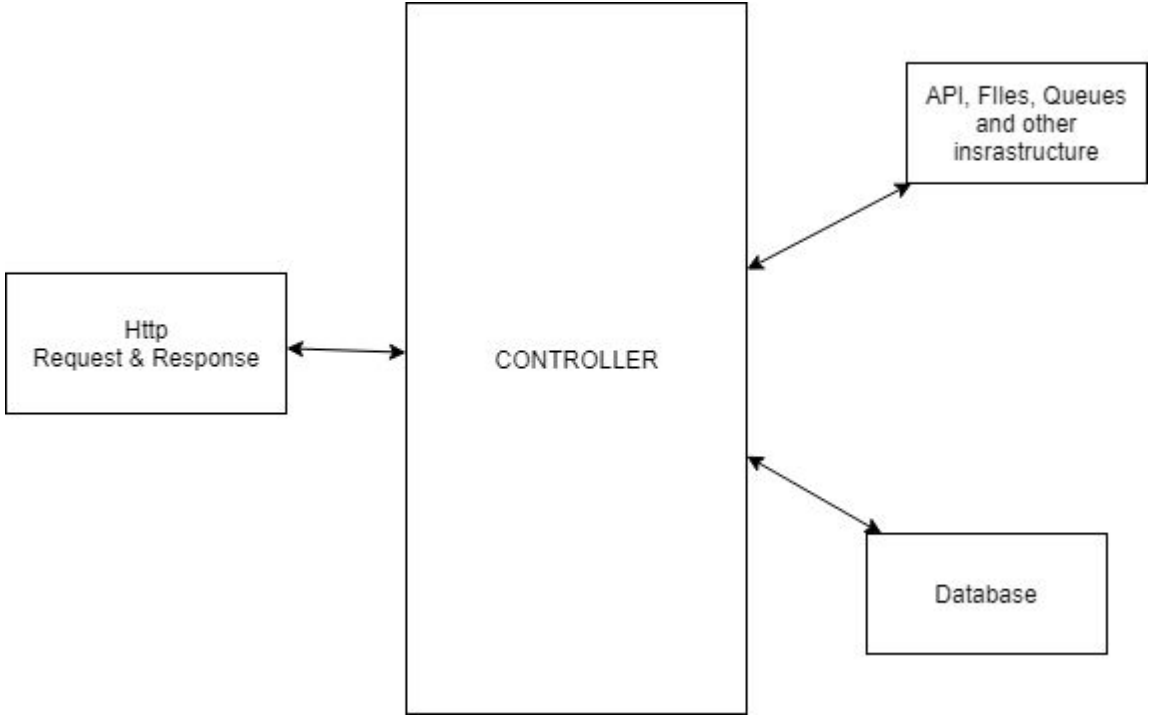
Unit test

Изолированный

Unit-тесты не должны быть долгими



Unit test контроллера



Типичный Laravel контроллер

- Берет данные из Http request и формирует Http response
- Дергает различную инфраструктуру(файлы, апишки)
- Дергает базу через Eloquent

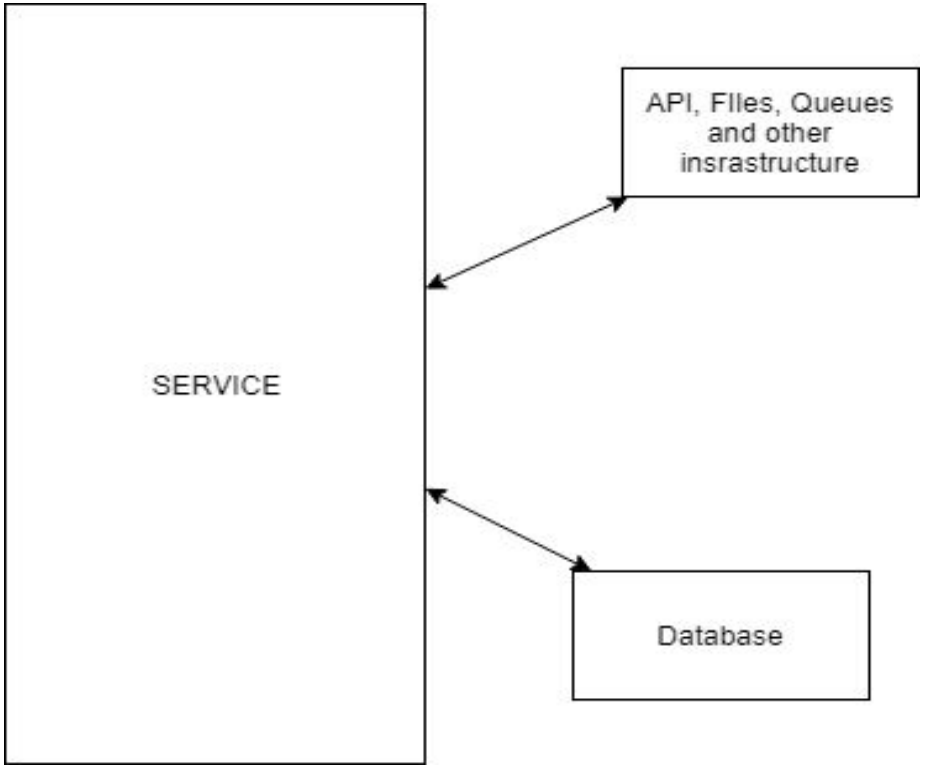
Тестить контроллеры - дело неблагодарное

Лучше вынести логику из него, тем самым исключив Http логику из тестов

Сервисные классы

или экшены, UseCase, Command, CommandHandler

Сервисные классы



Laravel testing

```php
class PollService
{
    public function create(PollCreateDto $dto)
    {
        ...
        \DB::transaction(function () use ($dto, $poll) {
            $poll->save();
            ...
        });
        \Event::dispatch(new PollCreated($poll->id));
    }
}
```

Laravel testing

```php
class PollServiceTest extends TestCase
{
    public function testCreate()
    {
        \Event::fake();

        $postService = new PollService();
        $postService->create(new PollCreateDto('test title', ['option1', 'option2']));

        \Event::assertDispatched(PollCreated::class);
    }
}
```

Application layer

```
class PollServiceTest extends Illuminate\Foundation\Testing
\TestCase
{
    ...
}
```

Это создаст приложение Laravel для каждого теста… со всеми сервис-провайдерами и т.д.

Laravel testing

```
class PollService
{
    public function create(PollCreateDto $dto)
    {
        ...
        \DB::transaction(function () use ($dto, $poll) {
            $poll->save();
            ...
        });
        \NewFacade::call();
    }
}
```

Laravel testing

Вызов будет реальным, хоть и из тестового окружения.

Но мне рассказывали случаи, когда случайно генерились тысячи денежных транзакций из-за прогона тестов.

Laravel testing

В базу все равно идут запросы

Тесты будут медленными

Application layer

```
class CreateQuestionController
{
    public function create(
        HttpCreateQuestionRequest  $request,
        CreateQuestionCommand  $command): RedirectResponse
    {
        $questionHash  = $command->execute(\Auth::user(), $request);

        return redirect()->route('questionCreated', $questionHash);
    }
```

Application layer

```
class CreateQuestionCommand extends BaseQuestionCommand
{
    public function __construct(
            ConnectionInterface $connection,
            ImageGenerateService $imageGenerateService,
            WriteS3Service $s3Service,
            Dispatcher $dispatcher,
            NaturalLanguageService $languageService)
    { ... }

    public function execute(User $author, CreateQuestionRequest $request): string
    { ...
```

Application layer

```
$repositoryMock = $this->createMock(PollRepository::class);
$repositoryMock->method('save')
    ->with($this->callback(function(Poll $poll) {
        return $poll->title == 'test title';
    }));
$repositoryMock->expects($this->at(2))
    ->method('saveOption');
```

```php
public function testCreatePoll()
{
    $eventFake = new EventFake(
        $this->createMock(Dispatcher::class));

    $postService = new PollService($eventFake);

    $postService->create(new PollCreateDto(
        'test title',
        ['option1', 'option2']));

    $eventFake->assertDispatched(PollCreated::class);
}
```

Application layer

```
    public function testCreatePoll()
    {
        $eventFake = new EventFake(
            $this->createMock(Dispatcher::class));
        $repositoryMock = $this->createMock(PollRepository::class);
        $repositoryMock->method('save')
            ->with($this->callback(function(Poll $poll) {
                return $poll->title == 'test title';
            }));
        $repositoryMock->expects($this->at(2))
            ->method('saveOption');
        $postService = new PollService(
            new FakeConnection(), $repositoryMock, $eventFake);

        $postService->create(new PollCreateDto(
            'test title',
            ['option1', 'option2']));
        $eventFake->assertDispatched(PollCreated::class);
    }
}
```

Отрывать от базы сложно

Надо убрать все вызовы Eloquent методов из кода сервисов и поместить в “репозиторий”

“Repository”

```
interface PollRepository {
    public function getById($id);
    public function save(Poll $poll);
    public function saveOption(PollOption $pollOption);
}
```

“Repository”

```
class EloquentPollRepository implements PollRepository {

    public function getById($id) {
        return Poll::findOrFail($id);
    }

    public function save(Poll $poll) {
        $poll->save();
    }
}
```

“Repository”

```
class PollService
{
    public function create(PollCreateDto $dto)
    {
        ...
        $this->connection->t ransaction(function () use ($dto, $poll) {
            $this->repository ->save($poll);
            ...
        });
        $this->dispatcher-> dispatch(new PollCreated($poll->id));
    }
}
```

Application layer

```
$repositoryMock = $this->createMock(PollRepository::class);
$repositoryMock->method('save')
    ->with($this->callback(function(Poll $poll) {
      return $poll->title == 'test title';
     }));
 $repositoryMock->expects($this->at(2))
     ->method('saveOption');
```

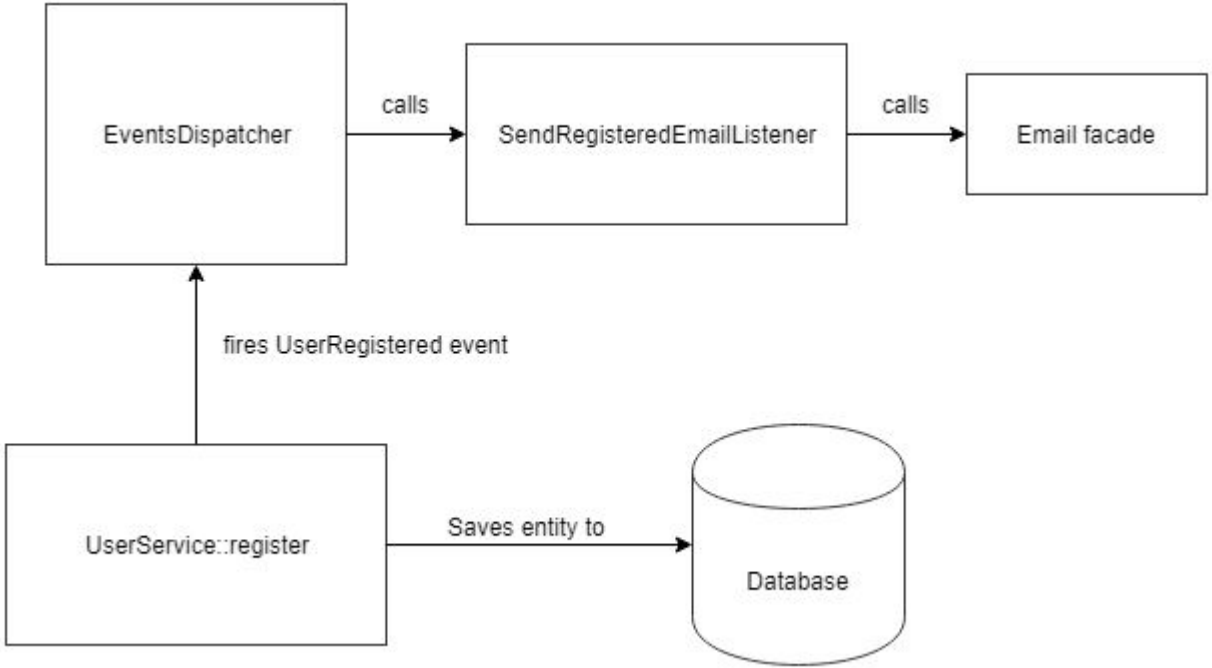
Польза таких тестов

Польза <<< Время на них потраченное(и которое будет тратится регулярно)

Laravel testing

Юнит-тестирование в Laravel вещь очень непопулярная по причинам, которые я описывал тут.

Поэтому тесты в основном интеграционные или функциональные.


EventsDispatcher
calls
SendRegisteredEmailListener
calls
Email facade
fires UserRegistered event
UserService::register
Saves entity to
Database

Интеграционное тестирование

```php
public function testOrderShipping()
{
    Mail::fake();

    // Perform order shipping...

    Mail::assertSent(OrderShipped::class, function ($mail) use ($order) {
        return $mail->order->id === $order->id;
    });
```

Интеграционное тестирование

Чтобы написать тест, нужно знать полностью как внутри работает все.

Какой эвент вызовется

Какое письмо отправится

Чего-нибудь незначащее поменял в коде - тест упал

Интеграционное тестирование в Laravel

Тестирование методом белого ящика

А оно очень хрупкое

Функциональное тестирование

руками тестировщик тестирует приложение

скрипт гоняет браузер сам(Selenium)

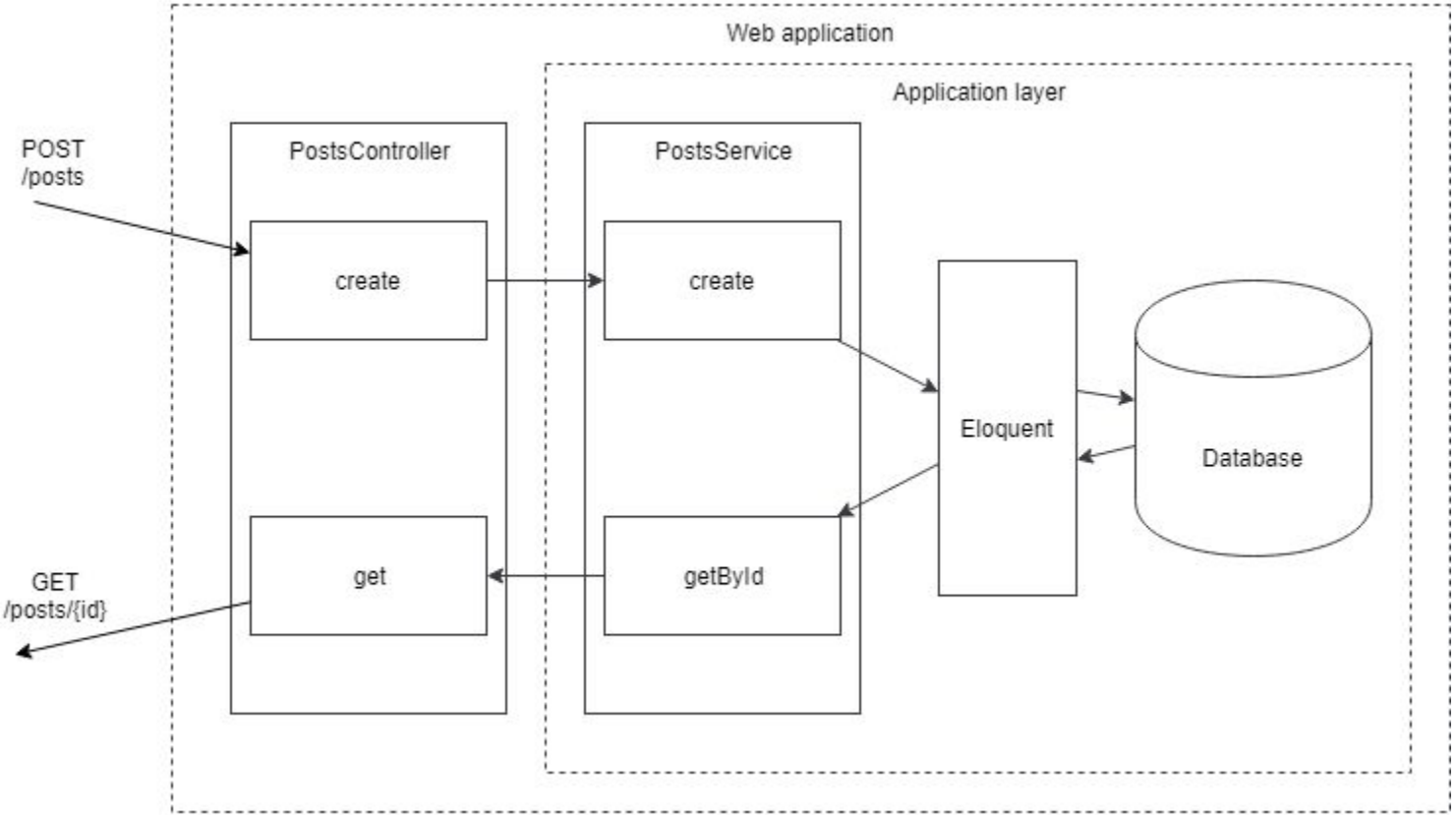
скрипт вызывает API методы

Тест API

```php
public function testBasicExample()
 {
     $response = $this->json('POST', '/user', ['name' => 'Sally']);

     $response
         ->assertStatus(201)
         ->assertJson([
             'created' => true,
         ]);
 }
```

Тест API

```
public function testBasicExample()
 {
     $response = $this->json('POST', '/user', ['name' => 'Sally']);

     // Check Response status

     $this->assertDatabaseHas('users', [
          'email' => 'sally@example.com'
     ]);
```


Web application
Application layer
POST
/posts
PostsController
PostsService
create
create
Eloquent
Database
get
getById
GET
/posts/{id}

Пример плохого теста

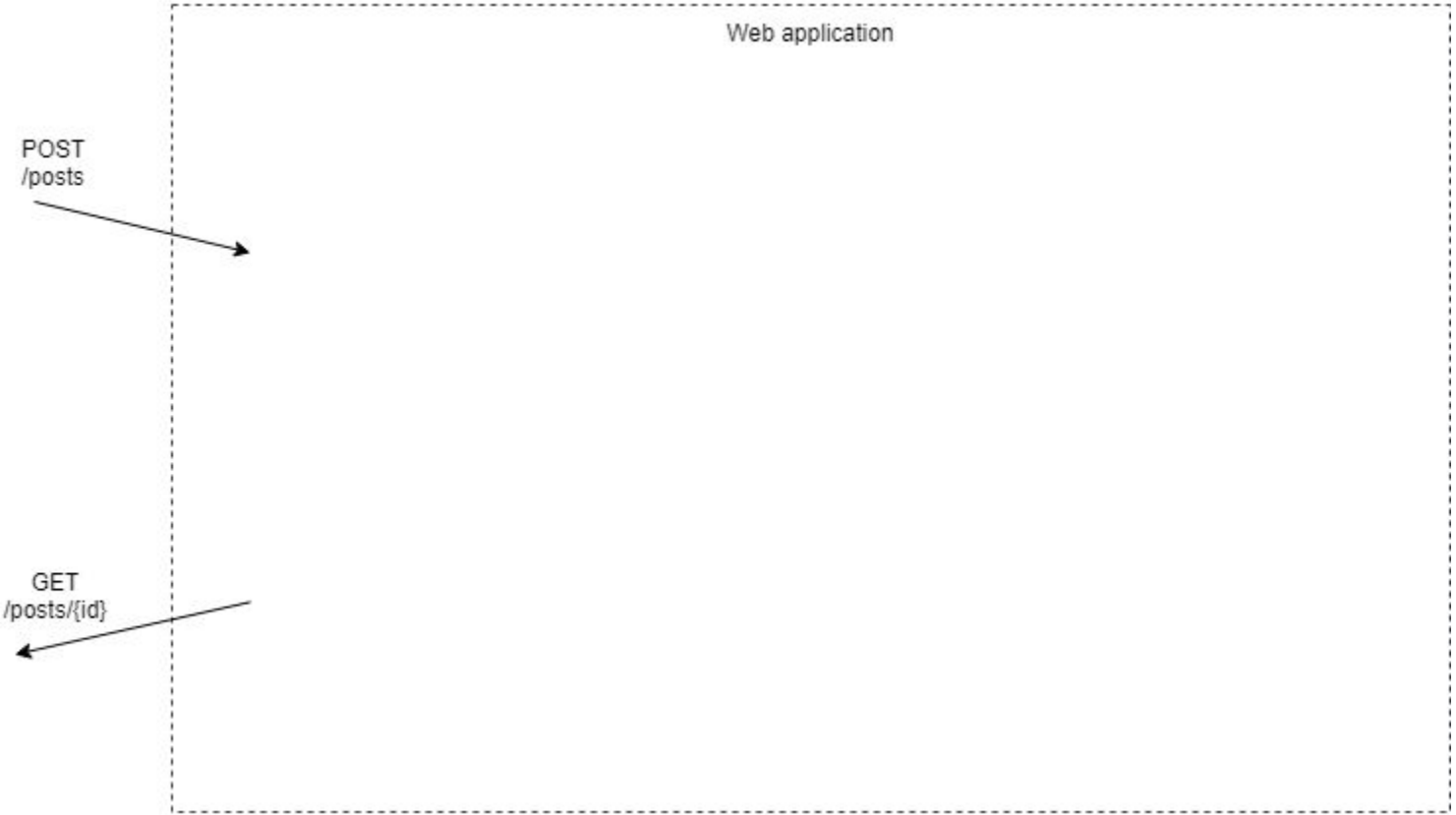
```
public function testDelete()
{
    $response = $this->deleteJson('/posts/1');

    $response->assertOk();

    $this->assertDatabaseMissing('posts', ['id' => 1]);
}
```

Пример плохого теста

```
class Post
{
    use SoftDeletes;

}
```

Пример плохого теста

```
public function testDelete()
{
    $response = $this->deleteJson('/posts/1');

    $response->assertOk();

    $this->assertSoftDeleted('posts', ['id' => 1]);
}
```


Web application
POST
/posts
GET
/posts/{id}

```php
public function testCreate()
{
    $response = $this->postJson('/api/posts',
        ['title' => 'Post test title']);

    $response->assertOk()->assertJsonStructure([ 'id']);

    $checkResponse = $this->getJson(
        '/api/posts/' . $response->getData()->id);

    $checkResponse->assertOk()
        ->assertJson([ 'title' => 'Post test title',]);
}
```

```php
public function testDelete()
{
    // Создание поста с $postId

    $this->getJson('/api/posts/' . $postId)
         ->assertOk();

    $this->jsonDelete('/posts/1')
         ->assertOk();

    $this->getJson('/api/posts/' . $postId)
         ->assertStatus(404);
}
```

Функциональный тест

Работа с приложением как с черным ящиком

Laravel

Если удобно писать какой-то функционал с юнит-тестами - обязательно писать.

Eloquent неудобно юит-тестить. Doctrine?

А функционал тестить руками :)

А потом писать нормальные функциональные тесты