Schnelle Integration-Tests

mit BrowserKit und Laravel-Tech

Ronny Hartenstein, Axilaris GmbH, @rhflow_de, 2018

Integrationstests

- Behat via Selenium
  -> steuert echte Browser-Instanz
- Laravel Dusk via ChromeDriver
- führt echte Requests via HTTP durch
  -> Netzlatenz,
  jedes Mal PHP-Bootstrapping
- eher langsam

```
# language: de
Funktionalität: Einschalttest
Prüft ob Startseite erreichbar ist

Szenario: Startseite vorhanden
    Angenommen ich bin auf "/"
    Dann sollte ich "WILLKOMMEN" sehen
```

schnellere I-Tests

- via Headless Browser, z.B. Goutte
- führt immernoch echte Requests durch
- aber keine Browserinstanz mehr
- schon schneller
- (Behat hat Goutte Schnittstelle)

Unit Tests

- direkt auf Klassen angewendet
- wieselflink
- einmal PHP Bootstrapping
- kein Request-Response
- kein Tests im Response

"Missing Link" zw. schnellen Unit-Tests und langsamen Integrations-Tests?

Lösung: Feature Tests

so in **Laravel**

```
public function
testRedirectionToLoginWithoutCurrentUser()
{
    $this->from(route('login'))
        ->get(route('home'))
        ->assertStatus(302)
        ->assertRedirect(route('login'));
}


public function testRouteHomeIsDashboard()
{
    $user = factory(User::class)->create();

    $this->actingAs($user)
        ->get(route('home'))
        ->assertSee('Dashboard');
}
```

- meine Legacy-App hat kein Laravel!
- dann direkt Symfony BrowserKit mit eigenen Erweiterungen nutzen
- übertragen in eigene Legacy
- erstmal ohne schicke Laravel API

```
public function testHomepage() {
    $this->client->request('GET', '/');
    $response = $this->client->getResponse();
    $this->assertEquals(200, $response->getStatus());
    $this->assertNotFalse(strpos((string) $response, 'WILLKOMMEN'))
}
```

- dann noch in die eigene `FeatureTestCase` Basisklasse von Laravel gemopst..
- mit schicker API

```
public function testHomepage() {
    $this->get('/')
        ->assertStatus(200)
        ->assertSee('WILLKOMMEN');
}

public function testLoginDirekt() {
    $this->post('/nutzer/nutzer.php', [
        'login' => 'ronny',
        'passwort' => '***'
    ])
        ->assertSessionHas('ID_USER')
        ->assertSessionHas('USERDATA')
        ->assertSee('Ihr Aktivitätsstatus');
}
```

- BrowserKit bietet Symfony DomCrawler unterstützt durch CssSelector
- u.a. für Formular-Steuerung

```
public function testLoginViaSeitenleiste()
{
    $this->get('/');
    $form = $this->client->getCrawler()
        ->filter('form[name=”form_login”]')->form();
    $form['login'] = 'ronny';
    $form['passwort'] = '***';
    $this->client->submit($form);
    $this->client->getResponse()   // TestResponse
        ->assertSessionHas('ID_USER')
        ->assertSessionHas('USERDATA')
        ->assertSee('Ihr Aktivitätsstatus');
}
```

- Hier die für meine Belange spezifische `FeatureTestCase` Schnittstelle

```
abstract class FeatureTestCase extends TestCase
{
    public function withHeaders(array $headers)
    public function withHeader(string $name, string $value)
    public function actingAs(string $login)
    public function from(string $url)
    public function get($uri, array $headers = [])
    public function getJson($uri, array $headers = [])
    public function post($uri, array $data = [], array $headers = [])
    public function postJson($uri, array $data = [], array $headers = [])
    public function json($method, $uri, array $data = [], array $headers = [])
    public function withSession(array $data)
    public function session(array $data)
    public function flushSession()
}
```

- get/post etc. sprechen mit dem BrowserKitClient
- gekürzte Implementierung für eine absolut alte Legacy-App
- ohne PSR-7 (HTTP Messages),
- ohne PSR-11 (Container),
- ohne PSR-15 (HTTP Handlers)

```php
class BrowserKitClient extends BaseClient {
    protected function doRequest($request)
    {
        $uri = $request->getUri();
        $url = parse_url($uri);
        $_SERVER['SERVER_NAME'] = $url['host'];
        $serverparams = $request->getServer();
        foreach ($serverparams as $k => $v) $_SERVER[$k] = $v;

        // $_GET und $_POST befüllen
        if (!empty($url['query'])) // .. $_GET[];
        if ($request->getMethod() === 'POST') // .. $_POST[]

        // absolutes Legacy, je App speziell implementiert
        ob_start();
        include PUBLIC_ROOT . '/router.php';
        $content = ob_get_contents();
        ob_end_clean();

        // $_GET und $_POST leeren
        $status = 200; // hier kann man noch steuern
        $headers = ['Content-Type' => 'text/html'];

        return new TestResponse($content, $status, $headers);
    }
```

- zum Schluss: **TestResponse**
- inspiriert durch Laravels Erweiterungen

```
use \Symfony\Component\BrowserKit\Response;

class TestResponse extends Response {
    public function isSuccessful()
    public function assertSuccessful()
    public function assertStatus($status)
    public function assertSee($value)
    public function assertSeeText($value)
    public function assertDontSee($value)
    public function assertDontSeeText($value)
    public function assertJson(array $data, $strict = false)
    public function decodeResponseJson()
    public function json()
    public function assertSessionHas($key, $value = null)
    public function assertSessionHasAll(array $bindings)
    public function assertSessionMissing($key)
}
```

Recap

– – –

- Integrations-Tests via Browser sind langsam (auch ChromeDriver)
- Unit-Tests sind (sollten) verflixt schnell
- Feature-Tests sind Mischung aus beiden
- Test über Pseudo-Request-Response mit Analyse des Responses und Crawling (Form-Submit)
- aber **eine** PHP-Instanz = kein Boot, schnell wie ein PHP-Funktionsaufruf
- aber: $_GET, $_POST, $_SERVER, $_SESSION je “Request” leeren und füllen
- Process-Isolation von PHP-Unit ist keine Lösung!
- es bleiben in Legacy-Apps Seiteneffekte durch globale Vars etc.
- btw: Behat und Mink kann an BrowserKit angebunden werden

Frohe Ostern!

Ronny Hartenstein
Axilaris GmbH
@rhflow_de

Fragen!?

Testet jemand so?

Wie testet ihr alternativ?

Gibt es versteckte Pitholes?