**Workshop Unittest Laravel**

Deze oefening laat je unittests creëren in Laravel. Het is zoveel mogelijk gelijk aan de vorige workshop in WPF.

Achter de bullets staan de handelingen die je moet uitvoeren.

* Maak een Laravel project aan geheten “unittestLaravel”
* Voeg een klasse toe aan het project met de naam Bankrekening.php
* Vul dit bestand met :

class Bankrekening extends Model

{

private float $saldo = 0.0;

public function GetSaldo() : float

{

return $this->saldo;

}

public bool $bevroren = false;

public function \_\_construct($amount)

{

$this->saldo = $amount;

}

public function Opnemen (float $amount) : void

{

if ($this->bevroren) return;

$this->saldo += $amount;

}

public function Storten(float $amount) : void

{

$this->saldo += $amount;

}

}

Dit is de klasse voor een bankrekening, waarin Balance staat voor saldo. Het saldo wordt opgehoogd door creditering en verlaagd door debitering. Als de rekening geblokkeerd is (frozen), dan mag het saldo niet verlaagd worden.

Maak nu al unittests voor deze klasse. Dat kan gewoon, ook al is er nog geen applicatie gebouwd.

* Voeg een unittest toe met het commando  
  php artisan make:test BankrekeningTest --unit
* Verwijder ExampleTest.php in de mappen tests\Feature en tests\Unit
* In BankrekeningTest.php in de map tests\Unit kun je unittesten maken. Plaats in dit bestand:

<?php

namespace Tests\Unit;

use PHPUnit\Framework\TestCase;

use App\Models\Bankrekening;

class BankrekeningTest extends TestCase

{

public function test\_Storten\_Geldig\_Bedrag()

{

$rekening = new Bankrekening(11.99); // Arrange

$rekening->Storten(4.55); // Act

$this->assertEqualsWithDelta(16.54, $rekening->GetSaldo(), 0.001,

"storten niet gelukt"); // Assert

}

public function test\_Opnemen\_Geldig\_Bedrag()

{

$rekening = new Bankrekening(11.99); // Arrange

$rekening->Opnemen(4.55); // Act

$this->assertEqualsWithDelta(7.44, $rekening->GetSaldo(), 0.001,

"opnemen niet gelukt "); // Assert

}

}

In de act-regel voer je de te testen methode uit, in dit geval een storting respectievelijk opname met een bedrag van 4,55. Deze methoden worden uitgevoerd op een object van de klasse Bankrekening. In de arrange-regel maak je het object aan met een startbedrag van 11,99. In de assert-regel controleer je de uitkomst. Het saldo moet uitkomen op 16,54 respectievelijk 4,55. De methode die je hier toepast is: assertsEqualsWithDelta(expected, actual, precision, error message). De precisie geeft aan hoeveel het mag afwijken vanwege de afwijking die je kunt hebben wanneer je werkt met gebroken getallen.

Voor meer over asserts, zie: <https://phpunit.readthedocs.io/en/9.5/assertions.html#assertequalswithdelta>

* Voer de unttesten uit de het commando “php artisan test” .

Het resultaat moet zijn dat er een test faalt met de mededeling:

• Tests\Unit\BankrekeningTest > opnemen geldig bedrag

opnemen niet gelukt

Failed asserting that 16.54 matches expected 7.44.

Zoals in de volgende afbeelding:



* Blijkbaar is óf de unittest zelf fout óf de methode Opnemen in de klasse Bankrekening. Verbeter deze fout en test opnieuw.
* Voeg 2 unittesten toe: test\_Storten\_Negatief en test\_Opnemen\_negatief, waarin de methoden Storten en Opnemen worden getest met een negatieve waarde.

public function test\_Storten\_Negatief()

{

$rekening = new Bankrekening(11.99); // Arrange

$rekening->Storten(-4.55); // Act

$this->assertEqualsWithDelta(11.99, $rekening->GetSaldo(), 0.001,

" Negatieve waarde in storting"); // Assert

}

public function test\_Opnemen\_negatief()

{

$rekening = new Bankrekening(11.99); // Arrange

$rekening->Opnemen(-4.55); // Act

$this->assertEqualsWithDelta(11.99, $rekening->GetSaldo(), 0.001,

" Negatieve waarde in opname "); // Assert

}

* Deze beide testen falen doordat de methoden in de klasse Bankrekening niet controleren op negatieve bedragen. Verbeter deze fout in de klasse Bankrekening.
* Maak nog 2 unittesten erbij, namelijk:  
  - test\_Storten\_Bevroren\_Rekening  
  - test\_Opnemen\_Bevroren\_Rekening  
  Waarin je Bevroren in het arrange-gedeelte op true zet en de storting en opname met correcte waarden test. In het geval van opname zou het saldo niet aangepast mogen zijn in en het geval van storting zou het saldo correct opgehoogd moeten zijn.
* Als laatste kun je nog de applicatie bouwen. Met een geteste klasse Bankaccount zou dat moeten lukken. Je kunt de volgende view en controller gebruiken, maar je kunt het zelf vast veel beter.

bankrekening.blade.php

<!doctype html>

<html lang="nl">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css">

<title>Bank</title>

<style>

button {width:100%; margin-top:20px;}

.container {width:20%; min-width:300px;}

h1 {margin-top:40px; margin-bottom:20px;}

p, input {font-size:150%;}

</style>

</head>

<body>

<div class="container">

<h1>Bankrekening</h1>

<p>Saldo: {{ $saldo }}</p>

<form action="bevroren" method="post">

@csrf

<input type="checkbox" onclick="submit()"

@if ($bevroren) {{ "checked" }} @endif> bevroren

</form>

<h1>Bankoverschrijving</h1>

<form method="post">

@csrf

<div class="form-group">

<label>Bedrag: </label><br/>

<input type="text" pattern="[0-9]+(\.[0-9]{0,2})?" name="bedrag"

style="width:100%"/>

<button class="btn btn-primary"

type="sumbit" formaction="opnemen">Opnemen</button>

<button class="btn btn-primary"

type="sumbit" formaction="storten">Storten</button>

</div>

</form>

</div>

</body>

</html>

web.php

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\BankrekeningController;

Route::get('/', [BankrekeningController::class, 'index']);

Route::post('bevroren', [BankrekeningController::class, 'bevroren']);

Route::post('opnemen', [BankrekeningController::class, 'opnemen']);

Route::post('storten', [BankrekeningController::class, 'storten']);

BankAccountController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\BankAccount;

class BankAccountController extends Controller

{

public function index()

{

$account = new BankAccount(0.00);

session(['Balance' => $account->GetBalance()]);

session(['Frozen' => false]);

return view('bankaccount',

array('balance'=>$account->GetBalance(), 'frozen'=>$account->frozen));

}

public function frozen()

{

$account = new BankAccount(session('Balance'));

$account->frozen = !session('Frozen');

session(['Balance' => $account->GetBalance()]);

session(['Frozen' => $account->frozen]);

return view('bankaccount',

array('balance'=>$account->GetBalance(), 'frozen'=>$account->frozen));

}

public function debit(Request $request)

{

$account = new BankAccount(session('Balance'));

$account->frozen = session('Frozen');

$amount = $request->input('amount');

$account->Debit($amount);

session(['Balance' => $account->GetBalance()]);

session(['Frozen' => $account->frozen]);

return view('bankaccount',

array('balance'=>$account->GetBalance(), 'frozen'=>$account->frozen));

}

public function credit(Request $request)

{

$account = new BankAccount(session('Balance'));

$account->frozen = session('Frozen');

$amount = $request->input('amount');

$account->Credit($amount);

session(['Balance' => $account->GetBalance()]);

session(['Frozen' => $account->frozen]);

return view('bankaccount',

array('balance'=>$account->GetBalance(), 'frozen'=>$account->frozen));

}

}