

Klas / naam:

Module WDV-V en DBS-II

Webdevelopment en databases

Over dit document


Opleiding: Software Developer
Blok: C
Vak(ken): WEB
Weken 1 t/m 6

Versiebeheer

Versie	Datum	Auteur	Aanpassingen
1.0	22-07-2021	Mark Hoekveen Bart Roos	
2.0	10-07-2023	Steven van Rosendaal	Verwijzingen van XAMPP vervangen door Laragon, Gebroken links verwijderd of vervangen, Hulpkaarten aangepast

 **Theorieblok**

 **Opdracht**

 Een term: die je misschien nog niet kent

Inhoudsopgave

I. Over deze module	3
II. Studiewijzer	3
1. Mijn eerste framework	4
2. Laravel en databases	13
3. Controllers, routes en view	21
4. Relatieve database	30
5. CRUD in Laravel.....	33
6. Aggregates	43
III. Feedback-momenten	46
IV. Voor de docent	48
V. Kwaliteit en verbeteracties	48

I. Over deze module

Doelstelling en relevantie

Je gaat kennismaken met het Laravel-framework en enkele van de achterliggende principes. Die module draait om het aanpassen van bestaande applicaties. Je leert expliciet nog *niet* om een hele nieuwe applicatie te bouwen ('from scratch'). Tijdens het werken gebruik je de hulpkaarten van 4S.

Laravel is framework dat ontzettend veel gebruikt wordt bij webontwikkeling. Het is feitelijk de standaard wanneer je met PHP aan de slag gaat. Daarom moet je kennis hebben van Laravel voordat je op stage gaat.

Samenhang met andere onderdelen

Deze module bouwt voor op alle voorgaande WDV- en DBS-modules.

Materiaal- en middelenlijst

- Laragon
- Editor en browser

Feedback-momenten

- Week 3: het eerste feedbackmoment bestaat uit theoretische vragen.
- Week 6: voor het tweede feedbackmoment ga je de applicatie afmaken waar je ook bij de weekchecks ('inleveropdrachten') al aan hebt gewerkt.

Het is dus belangrijk om bij te blijven met de weekchecks, dat levert je een voordeel op bij het laatste feedbackmoment!

II. Studiewijzer

Week	Hoofdstuk / onderwerpen	Weekcheck	Bronnen
1	H1 – Mijn eerste framework	WDV-V-1: Scoutshop wordt 4Shop	https://laravel.com/
2	H2 – Laravel en databases	WDV-V-2: Kolommen toevoegen	
3	H3 – Controllers, routes en	<u>Feedbackmoment</u>	
4	views H4 – Relaties	WDV-V-4: Relatieve vragen	
5	H5 – CRUD in Laravel	WDV-V-5: Categorieën in 4Shop	
6	H6 – Aggregates	<u>Feedbackmoment</u>	

1. Mijn eerste framework

1. Werken met een framework

In het vorige blok heb je alle code bij WEB zelf moeten maken. Daar zijn we best ver mee gekomen, maar de grens was wel bereikt. Wanneer je een echte professionele webapplicatie wil bouwen, kun je eigenlijk niet zonder framework tegenwoordig. Er zijn talloze zaken qua beveiliging enzovoort waar je anders zelf rekening mee moet houden.

Je hebt al eerder gewerkt met een framework; in blok B heb je bij WIN gewerkt met een framework als WinForms of UWP. Je hoefde zelf geen code te schrijven om een scherm tevoorschijn te toveren. Al die code zat in het framework. Zelf maak je alleen code die de “logica” van de applicatie beschrijft.

Ook voor webdevelopment zijn er talloze frameworks (niet alleen voor PHP trouwens). Een aantal bekende zijn: Ruby on Rails, Django, Laravel, Symfony, Angular, React, enzovoort.

2. Het Laravel framework

Voor webontwikkeling zijn we gewend te werken met PHP. Het grootste en beste PHP-framework van dit moment is Laravel. Daar gaan we in dit blok dan ook mee werken.

Laravel is een zogenaamd “MVC”-framework (dat is een bepaalde manier van werken, en gaat bijvoorbeeld over het indelen van je bestanden). Veel andere frameworks volgen ook het MVC-principe. Stel dat je op stage dus niet met Laravel gaat werken maar met een ander framework, dan is de kans toch heel groot dat je al wat basiskennis hebt. De principes van een MVC-framework zijn altijd ongeveer gelijk.

3. Laravel versus blok B

Je zult even moeten wennen aan Laravel, maar houd altijd in je achterhoofd dat er ook veel *overeenkomsten* zijn met de werkwijze van blok B. Een paar overeenkomsten en verschillen op een rijtje:

Overeenkomsten	Verschillen
Ook Laravel werkt met controllers.	In Laravel gaat de controller niet alleen over het opslaan en aanpassen, maar óók over het tonen van gegevens.
Een controller gaat over één resource, en is ingedeeld in secties voor insert, update, etc.	De indeling van een controller bestaat niet uit if-statements, maar uit methodes.
Laravel heeft ook een centraal config-bestand dat niet gecommitt wordt.	Het config-bestand heet nu “.env” in plaats van “config.php”.
Request en response zijn belangrijke termen in Laravel, je kent deze uit het vorige blok nog.	
	In Laravel is er geen directe link tussen je mappenstructuur en de URL's. Er is een aparte “routes-file” waarin alle links zijn vastgelegd.

4. Opdracht: terugblik

Beantwoord voor jezelf de volgende vragen in een los document. Daarna worden ze klassikaal besproken. Gebruik eventueel het moduleboekje van WDV-III om terug te kijken. Ga nog niet op internet zoeken.

1. Wat is een controller, of wat doet een controller?
2. Wat is een resource? Geef een voorbeeld.
3. Wat staat er in een config-bestand?
4. Waarom wordt een config-bestand niet gecommitt (het is “ge-gitignored”)?
5. Wat is een request?
6. Wat is een response?

5. MVC

Laravel is dus een MVC-framework. MVC is een universeel principe in het programmeren en staat voor “Model-View-Controller”. Een applicatie opgebouwd volgens MVC bestaat globaal uit drie soorten files:

- **Models:** voor iedere resource is er een model, dit model is een laag tussen de database en de rest van de applicatie. In Laravel kan een model voor jou automatisch query's uitvoeren. Stel dat je alle spelers wil opvragen dan kun je zoiets doen met je ‘Player’-model:
`$players = Player::all();`
- **Views:** dit zijn simpelweg de pagina's met HTML en kleine stukjes PHP. Je kent dit uit blok B al, alleen hadden deze pagina's toen nog geen speciale naam. Het kan dus gaan om een index-pagina die alle items toont, maar ook de html van de formulieren voor maken en aanpassen.
- **Controllers:** de controllers zijn het centrale punt van je applicatie, ze verbinden alles aan elkaar. Hierin gebeurt veel van de logica (bijvoorbeeld validatie bij het aanmaken van een nieuw item).


Daarnaast hebben veel frameworks een manier om links / URL's vast te leggen:

- **Routes-file:** alle requests komen binnen in de routes-file. Hierin staat voor iedere mogelijke link hoe Laravel die request moet afhandelen. Bijvoorbeeld, deze regel legt vast dat een gebruiker die de link “http://localhost/tournaments/create” bezoekt, wordt doorgestuurd naar de TournamentController die de boel verder afhandelt middels zijn “create”-methode:


```
Route::get('/tournaments/create', [TournamentController::class, 'create'])
```

6. Opdracht: *an animated introduction tot MVC*

1. Kijk de volgende video; <https://laracasts.com/series/laravel-8-from-scratch/episodes/1>.
2. Maak de volgende zinnen af in je eigen woorden:
 - a. Een model is
 - b. Een view is
 - c. Een controller

 Laracasts: de video die je hebt gekeken is van “Laracasts”, een website met honderden video's over Laravel en webdevelopment in het algemeen. Over het algemeen een hele goede bron! Let wel op dat lang niet alle video's geschikt zijn voor bij deze module. Het is dus niet zinvol om zomaar wat video's te gaan kijken.

7. Onze ontwikkelomgeving voor Laravel

 Ontwikkelomgeving: je ontwikkelomgeving is de verzameling van alle programma's, tools, editors enzovoort die je nodig hebt om op jouw pc te kunnen ontwikkelen met een bepaalde taal of framework.

Om te kunnen ontwikkelen voor Laravel hebben we twee verschillende “packagemanagers” nodig. Een packagemanager kun je zien als “app-store” voor een programmeertaal. Misschien kun je uit blok A nog “pip” voor Python of uit blok B “NuGet” voor C#.

Voor Laravel gaan we werken met:

- Composer als packagemanager voor PHP.
- NPM als packagemanager voor front-end (HTML, CSS en JavaScript).

Dankzij programma's als composer en npm kunnen we heel makkelijk externe pakketten gebruiken in onze code. Al die code staat dan níet in jouw repository, maar wordt apart geïnstalleerd wanneer je een repo binnenhaalt.

8. Opdracht: ontwikkelomgeving installeren

Volg de instructies van hulpkaart 1 (‘installeren composer + npm’). Je vindt de hulpkaarten op Itslearning in de map Algemeen en SLB.

👉 9. Thema voor de komende lessen

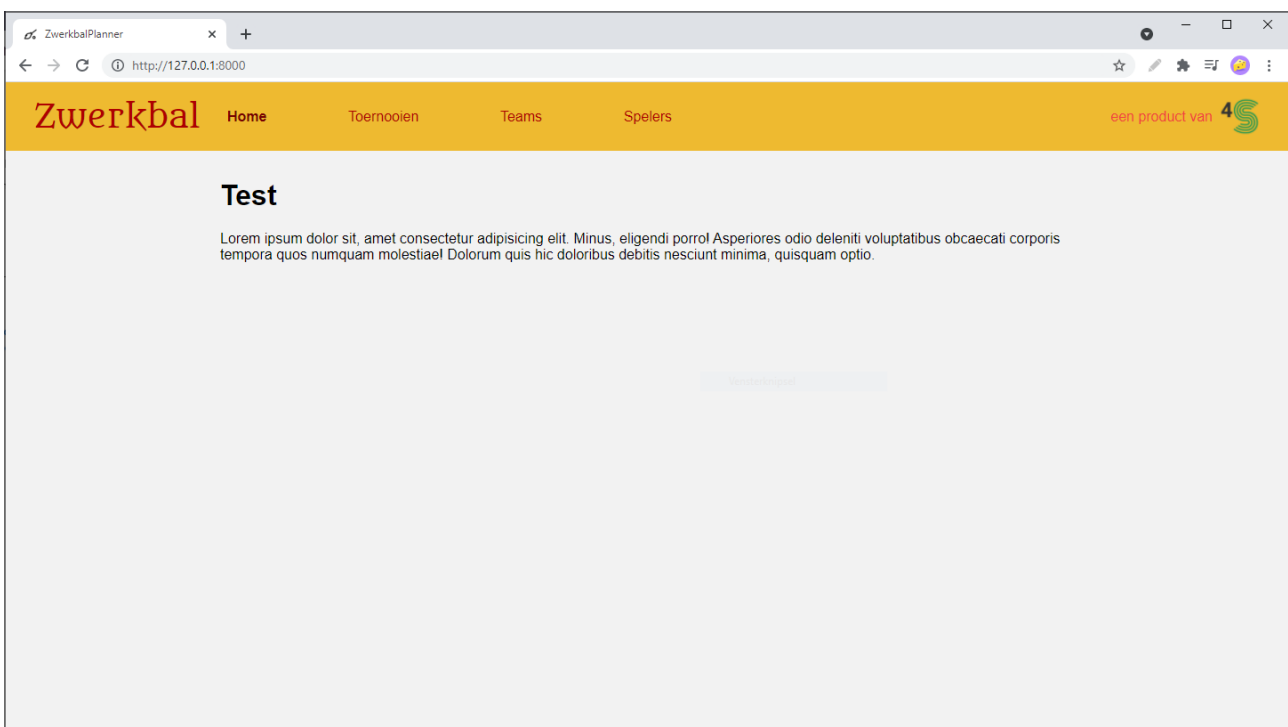
Bij 4S is momenteel een applicatie in ontwikkeling voor het plannen van Zwerkbalk-toernooien. Er is een kleine opzet gemaakt, maar wij gaan die applicatie in de komende weken uitbreiden.

Tegelijk maken we ook kennis met een eenvoudige webshop-applicatie. Hier komen ook een paar opdrachten over. Zo leer je snel je weg kennen in verschillende bestaande Laravel-apps.

👉 10. Opdracht: bestaande applicatie installeren

Nu we onze ontwikkelomgeving hebben opgezet, kunnen we de eerste Laravel-applicatie gaan installeren. We starten met de “4S_Zwerkbalk” applicatie. Op GitHub vind je al een repository met een bestaande Laravel-applicatie. Die repo gaan we forken en clonen, daarna gebruiken we composer om alle nodige packages te installeren.

1. De link naar de repo is: https://github.com/4S-NL/4S_Zwerkbalk.
2. Gebruik nu hulpkaart 2 ‘gebruik bestaand Laravel project’.
(We gaan later nog zelf testdata invoeren, sla stap 8 dus over)
3. Als het gelukt is, zie je ongeveer dit scherm in je browser:



👉 11. Opdracht: testdata invoeren

De zwerkbalk-applicatie werkt nog niet goed. Je kunt niet op een goede manier toernooien invoeren en bekijken. Daarom gaan we via phpMyAdmin zelf testdata invoeren.

- 🔍 Goede testdata is realistisch en gevarieerd: verzin dus namen die echt bij een sporttoernooi zouden kunnen passen. Let ook op dat je geen onprofessionele data gebruikt, dat staat natuurlijk niet goed. Variatie kun je vooral in de datums inbouwen: kies datums in het verleden, heden en de toekomst.

1. Vul minimaal 10 verschillende toernooien in via phpMyAdmin.
2. Kijk hoe dit er uit ziet in de applicatie op de ‘toernooien’-pagina.

👉 12. De opbouw van een Laravel-app

Een Laravel-applicatie heeft altijd dezelfde mappenstructuur. Tussen verschillende versies van Laravel kunnen kleine verschillen ontstaan. In versie 8, waar 4S_Zwerkbal ook op draait, is dit de globale structuur.

Let op, alleen de belangrijkste mappen staan hieronder:

app/Http/Controllers/ Controller.php TournamentController.php XyzController.php	Dit is basis-controller, deze file passen we niet aan. Per resource vind je hier verder één controller.
app/Models/ [†] User.php Xyz.php	In deze map vind je per resource een model.
database/migrations/ 2021....._create_xyx_table.php	Iedere migration beschrijft de structuur van één tabel.
public/ css/ img/	In de public-map kun je dingen als CSS en afbeeldingen kwijt.
resources/views/ xyz.blade.php ...	Hier vind je de HTML-kant van de applicatie. Meestal zijn hier nog submappen om de views te organiseren. Een view eindigt altijd op .blade.php. "Blade" is het Laravel onderdeel dat alles rondom views regelt.
routes/web.php	In de file routes-file ligt de link tussen URL's en controllers.

[†] In oudere versies van Laravel vind je de models niet in een aparte map maar gewoon in app/. Dit systeem gebruiken we nog bij de inleveropdrachten!

Alle andere mappen en files in de hoofdmap zijn (nog) niet relevant. We gaan daar niet mee werken, en je mag die ook niet aanpassen.

👉 13. Opdracht: de flow in een Laravel-app

Je kunt nu gaan naar <http://127.0.0.1:8000/tournaments>, en dan zie je een lijst van alle toernooien in de database. Beschrijf nu eens hoe dat werkt. Maak een los document waarin je screenshots plakt van alle stukjes code die hiermee te maken hebben.

Begin in de file routes/web.php en lokaliseer de regel die de GET-request voor /tournaments afvangt. Dat is je eerste screenshot. Die regel vertelt je ook meteen welke controller (en methode daarin) de volgende stap zijn. Volg zo het hele spoor tot je de HTML van deze pagina hebt gevonden. Tip: de HTML-kant van je app vind je dus in de views.

👉 14. Views en Blade

Views vormen dus de HTML-kant van je applicatie. Als je een *view* opent zul je daar ook bekende HTML-code vinden, met tussendoor wat stukken PHP. Naast alle overeenkomsten, kun je ook wat verschillen spotten met de aanpak uit blok B:

- De syntax `{{ $var }}` is kort voor `<?php echo $var; ?>`.
- De commando's die beginnen met een `@`, dit zijn zogenaamde "blade directives". Ook hierbij hoef je dus niet helemaal een inline php-tag te openen.

Deze schrijfwijzen zijn uniek voor Laravel. Het framework gebruikt de Blade-engine om deze speciale code om te zetten naar echte PHP. Het maakt ons leven als developer dus wat eenvoudiger, omdat het een hoop typewerk scheelt. Ook leest het een stuk makkelijker, als je bestaande code wil bekijken.

Overigens werkt alle gewone PHP-code ook in Blade. Als je wil, zou je dus wel `<?php echo $var; ?>` kunnen schrijven om een variabele te echo'en.

Een uitgebreid voorbeeld van de werking van Blade:

```
@extends('layout')

@section('content')

    <h1>Toernooien</h1>

    <a href="{{ route('t.create') }}">+nieuw</a>

    <ul>
        @foreach($tournaments as $tournament)
            <li>{{ $tournament->name }}</li>
        @endforeach
    </ul>

@endsection
```

Deze view gebruikt de template 'layout.blade.php'.

In die template wordt deze sectie ingeladen op de plek voor 'content'.

Hier wordt een link naar de route `t.create` ge-echo't (dus: een link naar een formulier)

Een `foreach` in blade werkt exact hetzelfde als in 'gewone' PHP. Alleen zet je een `@` neer in plaats van een hele inline php-tag te openen.

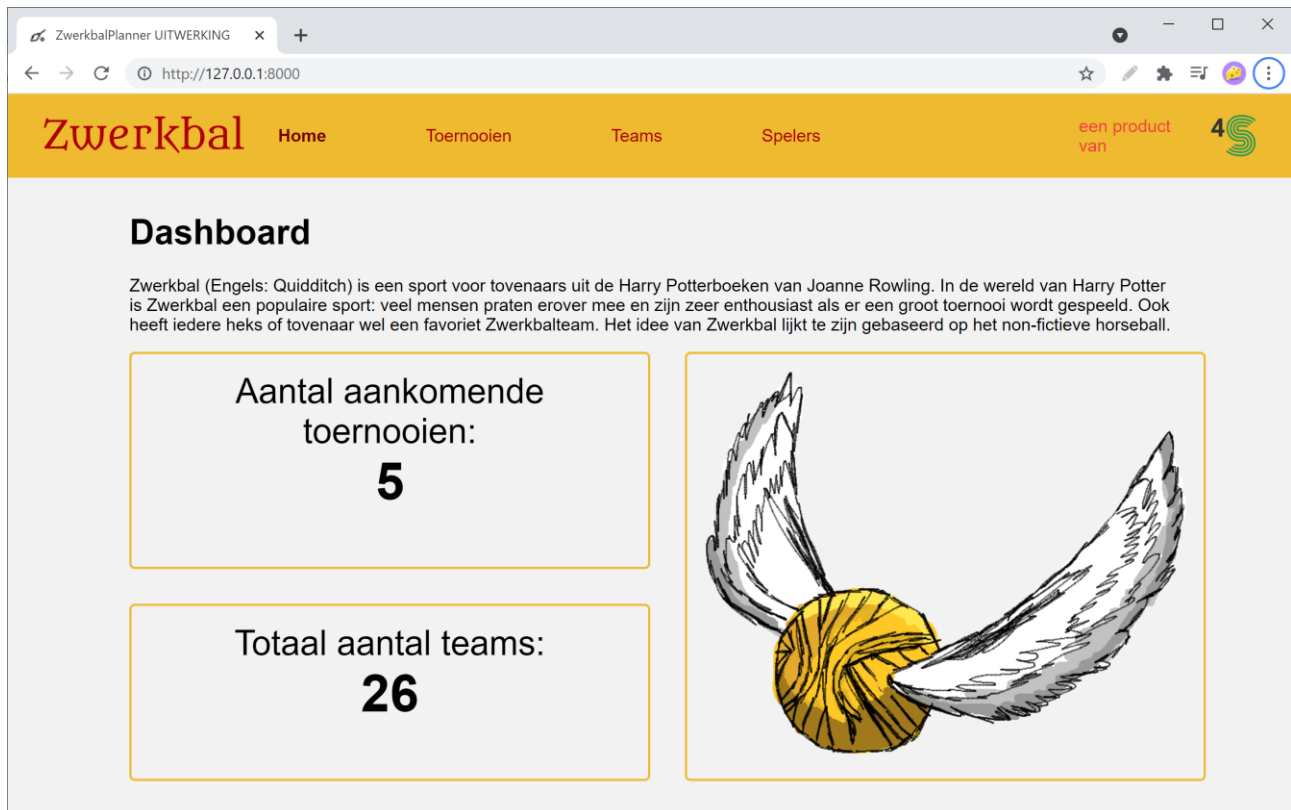
👉 15. Hulpkaart als naslagwerk

Bekijk hulpkaart nummer 3 ('Laravel: structuur en werking'). Houd deze ernaast bij de komende opdrachten.

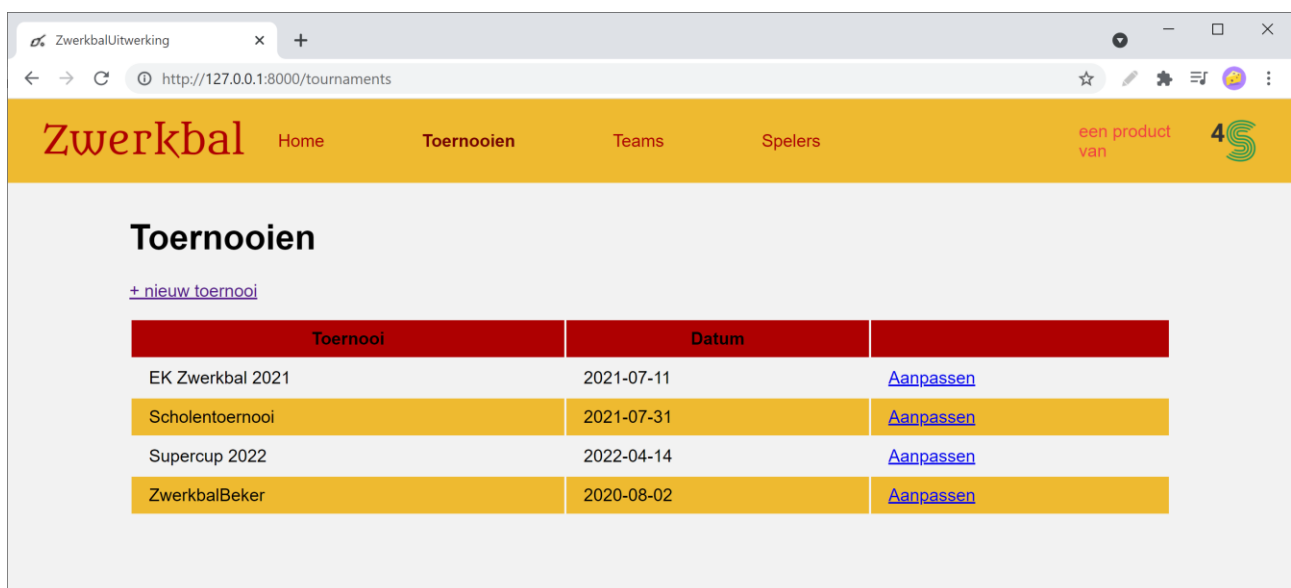
16. Opdracht: aanpassen 4S_Zwerkbal

Gebruik alle kennis die je hebt opgedaan om de volgende aanpassingen te maken aan de Zwerkbal-app. Houd eventueel hulpkaart 3 ernaast.

1. De homepage is eigenlijk nog leeg. Zorg dat je het onderstaande screenshot namaakt.
 - De tekst is afkomstig van <https://nl.wikipedia.org/wiki/Zwerkbal>.
 - De afbeelding is al aanwezig in de img-map.
 - De getallen zijn hardcoded. Schrijf nog géén PHP, maar zet letterlijk een '5' en '26' in je HTML.
 - Tip homepage: de volgorde waarop de dashboard-items in je HTML staan is van belang.



2. Maak de toernooien-pagina in orde, zoals het onderstaande screenshot. De nieuw-link heeft wat ruimte gekregen aan de onderkant, en de kolom voor 'Datum' is nu ook gevuld.



3. Op dit moment is het font voor alle lopende tekst Arial. Dat is wat saai. Een font dat mooier combineert met het “Zwerkbal”-font linksboven is Open Sans. Gebruik Google Fonts om Open Sans te integreren in de applicatie, en zorg ervoor dat alle tekst behalve linksboven voortaan Open Sans is. Tips:
- Haal minimaal twee stijlen van Open Sans binnen: Regular 400 en Bold 700.
 - In layout.blade.php zie je al hoe het bijzondere font voor het ‘zwerkbal’ logo wordt ingeladen.
 - Zoek in de CSS op Arial, dan weet je meteen waar je het nieuwe font moet instellen.

Datum
11-07-2021
31-07-2021
14-04-2022
02-08-2020

4. Bonusopdracht: probeer het datumformaat op de toernooien-pagina goed te krijgen (Nederlands formaat in plaats van internationaal). Gebruik deze bron:
- <https://laravel.com/docs/8.x/eloquent-mutators#date-casting>
 - <https://www.digitalocean.com/community/tutorials/easier-datetime-in-laravel-and-php-with-carbon>

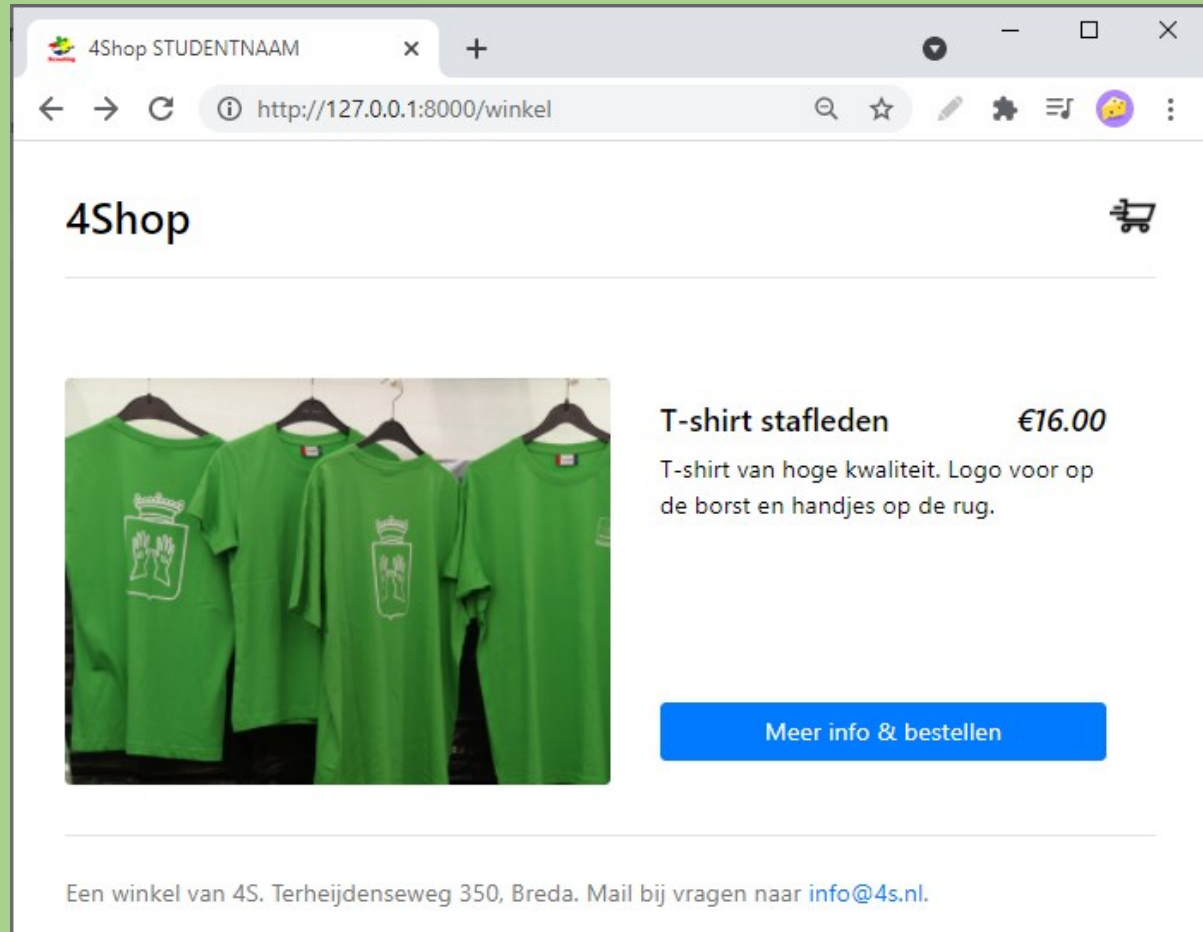


Weekcheck WDV-V-1: Scoutshop wordt 4Shop

Het bedrijf 4S wil graag een webshop opzetten om allerlei merchandise te verkopen, vooral gericht op hun eigen medewerkers die graag een T-shirt, trui, enzovoort met bedrijfslogo willen.

Men wil hiervoor een oud legacy-project verbouwen en opnieuw inzetten. Enkele jaren geleden heeft 4S een eenvoudige webwinkel gebouwd voor een vereniging. Die app zou ook prima voldoen voor de eigen webshop. De repo is al gekopieerd en klaargezet als '4Shop', maar aan de code is nog niets gewijzigd.

Voorbeeld van het eindresultaat:



1. Zorg eerst dat je de 4Shop-applicatie lokaal hebt draaien.
 - De repo: <https://github.com/4S-NL/4Shop>.
 - Gebruik hulpkaart 2 ('gebruik bestaand Laravel-project').
 - Bij stap 8: er een seeder aanwezig, dus kies optie 8b.
2. Maak de volgende aanpassingen in de 4Shop. Zie screenshot hierboven voor het eindresultaat.
 - a) Pas de header én de title-tag aan, zodat overal "4Shop" staat als titel van de website. In de title-tag moet je ook je eigen naam toevoegen. Tip: je moet hiervoor een de van de views in de 'layouts'-map aanpassen.
 - b) Pas de tekst in de footer aan. Verzin zelf een tijdelijk adres (let op: tijdelijke data hoort wel realistisch en professioneel zijn).
 - c) Zowel op de homepage als de detailpagina staat de prijs direct naast de titel van een product. Gebruik bijvoorbeeld flexbox om deze uit elkaar te zetten. Tip: als je de prijs en titel uit elkaar wil zetten, dan moet je dus van de *gezamenlijke parent* een flexbox maken.
 - d) De blauwe knop op de homepage hoort onderaan een product te staan. Tip: je kunt flexbox natuurlijk ook gebruiken in de verticale modus ('column').
3. Bonusopdrachten, deze zijn niet verplicht:
 - a) Plaats het logo van 4S ook in de header. Je vindt het logo in deze repo in de map Bronnen: <https://github.com/4S-NL/Organisatie>.
 - b) Pas het font van de header aan naar 'Libre Baskerville Regular'. Deze kun je vinden op Google Fonts.
 - c) Zoek uit hoe je de favicon kunt aanpassen (het icoontje dat je ziet in de tab van je browser). Probeer eens het 4S-logo (zie vorige link) om te zetten naar een favicon.

Inleveren

Precies één screenshot van de homepage (pagina met alle producten). Je hele browserscherm moet zichtbaar zijn. Let op dat in de <title>-tag je eigen naam staat. Plaats geen tekst bij het inleveren.

Deze regels helpen ons om iedereen snel van feedback te kunnen voorzien.

Herhalingsoefeningen

Deze oefeningen zijn niet verplicht, ze zijn bedoeld om extra te oefenen. Je kunt er ook later nog eens op terug komen, bijvoorbeeld als voorbereiding op een toets.

Men wil, om verwarring te voorkomen, graag wat informatie over verzendkosten enzovoort op de bestelpagina plaatsen. Er moeten twee blokjes met tekst naast elkaar komen. Het ene blokje gaat over verzendkosten, de andere over levertijden. Gebruik behalve de kopjes nog lorem-tekst. Ongeveer zo:

(Lijst met producten)

Verzendkosten	Levertijden
Lorem ipsum...	Dolor sit amet...

(Formulier voor gegevens)

1. Open de view orders/order (exact: resources/views/orders/order.blade.php).
2. Zoek uit waar je de informatie moet plaatsen, bijvoorbeeld door eerst een eenvoudige p-tag te plaatsen en te kijken in je browser wat het resultaat is.
3. Voeg dan ten slotte de twee blokken met tekst toe. Je zult flexbox nodig hebben om de blokken naast elkaar te zetten. Maak dus één parent met daarin twee div's.

2. Laravel en databases

Als eerste wordt in de klas aandacht besteed aan de vorige weekcheck.

1. Migrations om je database te beheren

Het kan soms lastig zijn om een applicatie te integreren met een database. Databases zijn immers bedoeld als algemene dataopslag, en de meeste programmeertalen spreken geen SQL. Web-frameworks hebben dan ook een laag tussen de applicatie en de database nodig; deze noemen ze met een ingewikkelde term een Object-Relational Mapper (ORM). Ook Laravel heeft dit. En dat zorgt ervoor dat we niet meer direct SQL queries hoeven te schrijven, maar in plaats daarvan de ingebouwde functies van Laravel kunnen gebruiken om dingen met onze data te doen. Het kan in sommige gevallen nog nodig zijn dat we wel voor bepaalde ingewikkelde problemen handmatig de database in moeten duiken, dus het is wel fijn dat we toch weten hoe het moet.

Om het mogelijk te maken dat Laravel goed kan communiceren met je database maken we gebruik van de 'migrations' feature van Laravel. Deze feature heet migrations omdat het ook de mogelijkheid heeft om te migreren (veranderen) van de ene database-status naar de andere database-status. Bijvoorbeeld als er op gegeven moment nieuwe tabellen worden toegevoegd aan een al bestaande database. De migrations worden echter ook gebruikt om de database in eerste instantie op te zetten; je zou kunnen zeggen dat je "migreert" van geen tabellen naar wel tabellen.

Deze migrations zitten in de database/migrations map; per tabel een migration-bestand. We kunnen in die map zien dat de 4Shop en Zwerkbal-app al enkele migrations beschikbaar heeft.

```
database > migrations > 2019_05_13_074359_create_products_table.php > CreateProductsTable.php
1  <?php
2
3  use Illuminate\Support\Facades\Schema;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Database\Migrations\Migration;
6
7  class CreateProductsTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('products', function (Blueprint $table) {
17             $table->increments('id');
18             $table->string('title');
19             $table->text('description')->nullable();
20             $table->string('image')->nullable();
21             $table->decimal('price', 8, 2);
22             $table->boolean('leiding');
23             $table->boolean('active')->default(true);
24             $table->timestamps();
25         });
26     }
```

2. Opdracht: migrations uitpluizen

Maak de migrations map van de 4Shop open en kijk naar de migration die eindigt op create_orders_table.php In de up() functie zien we dat er een aanroep is naar een Schema::create() die vervolgens aan de slag gaat met een \$table-object. De lijst met aanroepen naar verschillende functies op \$table maken hier de verschillende kolommen aan van de orders-tabel:

```
$table->increments('id');
$table->integer('opening_id')->unsigned();
$table->string('slug')->nullable();
$table->string('name');
$table->string('email');
$table->string('speltak');
$table->decimal('amount', 8, 2)->nullable();
$table->string('mollie_id')->nullable();
$table->boolean('payed')->default(false);
$table->timestamps();
```

De namen van deze functies kunnen je al wel bekend voorkomen. \$table->increments("id") is bijvoorbeeld equivalent aan het aanmaken van een integer met het AUTO INCREMENTING attribuut in SQL. Vul de volgende lijst aan met de overige functie-aanroepen die je nog meer herkent, en wat het equivalent is in SQL. Als je een functie niet herkent kan je het ook proberen te beredeneren.

<u>Laravel</u>	<u>MySQL</u>
increments	AUTO_INCREMENT
integer	INT
string	

3. Migrations in de documentatie

In dit moduleboekje en op de hulpkaarten vind je alle basisinformatie over de werking van Laravel. Maar regelmatig zul je ook de documentatie van Laravel moeten raadplegen. Laravel wordt geroemd om de duidelijke en volledige *docs* op hun website. Dat zal ook één van de redenen zijn dat Laravel zo groot is geworden!

De documentatie vind je op <https://laravel.com/> onder knop 'Documentation'. Als je bijvoorbeeld over migrations wil lezen, dan klik je in het linkse menu op 'Database' en daarna 'Migrations'.

4. Opdracht: start-tijd van een toernooi

Naast de datum waarop een toernooi plaatsvindt, willen we ook de starttijd gaan opslaan:

Toernooi	Datum	Start-tijd	
EK Zwerkbal 2021	02-08-2021	20:00:00	Aanpassen
Zwerkbalcup	19-11-2020	11:00:00	Aanpassen
Scholentoernooi2022	19-08-2022		Aanpassen
ZwerkbalBeker	19-07-2021	01:00:00	Aanpassen

- Maak een nieuwe kolom aan in de migration van de *tournaments*-tabel.
 - Naam van de kolom: `start_time`
 - Type: zoek in de Laravel docs een kolomtype dat geschikt is om enkel en alleen een tijd in op te slaan. Kijk in het artikel over migrations onder de subkop 'Available Column Types'.
 - Let op: maak de kolom nullable. Kijk in het docs-artikel over migrations onder de subkop 'Column Modifiers'.
- Run in de terminal het commando `php artisan migrate`. Is er nu een migratie uitgevoerd?
- Run het commando `php artisan migrate:fresh`. Kijk in je database, wat is er veranderd?

5. Het principe van migrations en seeds

Het eigenlijke idee van migrations is als volgt: als je een nieuwe kolom wil toevoegen aan de database, maak je daarvoor een nieuwe migration-file. Maar in deze vroege ontwikkelfase is dat eigenlijk niet te doen. Je zou misschien tientallen migrations krijgen omdat we steeds de database een klein beetje aanpassen. Daarom moeten we steeds een 'fresh' migratie doen, dat betekent dat de database helemaal opnieuw wordt opgebouwd. Enig nadeel: de testdata is nu ook allemaal weg.

Maar let op: bij een applicatie die 'in productie' is en dus online staat, ga je natuurlijk nooit een `migrate:fresh` doen. Bij een app in productie zal Laravel je ook waarschuwen, of je het echt zeker weet.

Testdata en seeds

Omdat een `migrate:fresh` onze database helemaal opnieuw bouwt, is ook de testdata weg. De oplossing daarvoor vinden we in 'seeds'. Dat zijn kleine scripts die onze database automatisch kunnen vullen met testdata. Deze seeds kun je direct na een migratie laten uitvoeren. Het volledige commando is dan:

```
php artisan migrate:fresh --seed
```

6. Opdracht: start-tijd van een toernooi

We werken verder aan de aanpassing van opdracht 4: het toevoegen van een start-tijd.

4. Voeg een seeder toe aan jouw applicatie:

- Download uit Itslearning de file "DatabaseSeeder.php".
- Open in je project de file `database/seeds/DatabaseSeeder.php`.
- Vervang de inhoud van jouw eigen `DatabaseSeeder.php` door de gedownload file.

5. Bouw je database opnieuw op met het commando:

```
php artisan migrate:fresh --seed
```

- Controleer of het gelukt is; open je applicatie en kijk of je vier toernooien ziet staan.
- Kreeg je een foutmelding bij de seeder? Lees de foutmelding goed, vooral het eerste deel.
- Let op dat je de "time"-kolom echt nullable hebt gemaakt in de migration. De seeder stelt namelijk niet voor *ieder* toernooi een tijd in.

6. Zorg ervoor dat we de start-tijd nu ook kunnen zien op de toernooien-pagina:

- Open de view in `resources/views/tournaments/index.blade.php`.
- Bekijk hoe bijvoorbeeld op regel 25 de naam van een toernooi wordt weergegeven.
- Op dezelfde manier kun je de tijd tonen.
- Let op dat je ná r19 ook een `<th>` toevoegt.

Je applicatie moet er nu ongeveer zo uitzien als het screenshot in opdracht 4. Voor de volledigheid: het aanpassen van de formulieren, zodat je daar ook de datum en tijd kunt invullen komt in het volgende hoofdstuk.

7. Nieuwe migrations maken

Als we een hele nieuwe tabel nodig hebben, moeten we ook een nieuwe migration opzetten. In een Laravel applicatie schrijf je zelf je migrations (er zijn ook frameworks die dat automatisch doen, vooral voor andere programmeertalen). Je kan wel een startpunt laten opzetten met het artisan-commando "make:migration".

Als we bijvoorbeeld een nieuwe tabel genaamd 'monkeys' willen toevoegen (om wat voor reden dan ook) kunnen we `php artisan make:migration create_monkeys_table` doen. Deze maakt dan een bestand met de tijd, datum en de naam, bijvoorbeeld `2021_04_01_120155_create_monkeys_table.php` als we op 1 april 2021 om 12:01:55 dit commando uit zouden voeren.

Laravel haalt hier meteen uit wat de naam van de tabel is die je wilt maken en zet dan in ieder geval de volgende dingen klaar in je nieuwe migration:

Voorbeeld van een migration:

```
class CreateMonkeysTable extends Migration
{
    public function up()
    {
        Schema::create('monkeys', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('monkeys');
    }
}
```

Deze `up()` en `down()` functies omschrijven hoe Laravel de tabel moet aanmaken en ook weer moet verwijderen. Het verwijderen is standaard heel eenvoudig; gewoon een `DROP TABLE` statement. Bij de `up` moet er geschreven worden hoe de tabel moet worden aangemaakt. Feitelijk is dit de “Laravel manier” van het schrijven van een `CREATE TABLE` statement, zoals eerst phpMyAdmin dat voor ons deed.

We hebben eerder al gezien dat de namen van die Laravel-functies corresponderen met bepaalde kernwoorden uit SQL. In een nieuwe migration maakt Laravel standaard een `bigInteger` `id` aan, met de `auto-increment` property. Dat zit allemaal in het commando `$table->id()`. Ook wordt `timestamps()` aangeroepen, die maakt twee kolommen aan: `created_at` en `updated_at`, die automatisch bijhouden wanneer een rij is aangemaakt en bewerkt.

Als we bijvoorbeeld ook de naam van onze monkey’s willen bijhouden kunnen we dat doen in een string (`VARCHAR` in MySQL). We voegen dan na het `id` een regel toe: `$table->string('name');`.

Voor een lijst van overige functies die we kunnen gebruiken voor het aanmaken van kolommen, kun je kijken in de documentatie van Laravel: <https://laravel.com/docs/9.x/migrations#available-column-types>

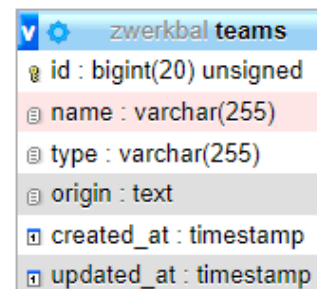
8. Opdracht: migration voor teams-tabel

In de zwerkbal-app willen we ook een overzicht van teams bijhouden. Daarvoor moet eerst een tabel gemaakt worden en gevuld met testdata.

1. Voor een nieuwe tabel hebben we een nieuwe migration nodig. We gebruiken het `make`-commando om de basis van een migration klaar te zetten. Voer uit in de terminal:
`php artisan make:migration create_teams_table`

2. Open de gemaakte migration (map: `database/migrations`). Bouw een migration voor de tabel uit het diagram hiernaast. Tips:

- Kolommen `id`, `created_at` en `updated_at` zijn al aanwezig, die worden gemaakt met `$table->id()` en `$table->timestamps()`.
- Een `VARCHAR`-kolom maak je in een migration door te gebruiken: `$table->string(...)`.
- Maak van ‘`origin`’ een *nullable*-kolom.
- Bekijk eventueel de docs op <https://laravel.com/> of gebruik andere migrations als voorbeeld.



zwerkbal teams	
id	: bigint(20) unsigned
name	: varchar(255)
type	: varchar(255)
origin	: text
created_at	: timestamp
updated_at	: timestamp

3. Voer je nieuwe migration uit met het commando `php artisan migrate`. Je hoeft geen 'fresh' migratie te doen, omdat je echt een nieuwe file hebt gemaakt. De rest van de database blijft dus intact. Als je hierna nog iets wijzigt in een migration, moet je wél weer `migrate:fresh` gebruiken (eventueel `--seed`).
4. Open phpMyAdmin, ga naar de teams-tabel en voer voorbeelddata in (realistisch en professioneel).

9. Convention over configuration

Laravel volgt het principe van "convention over configuration". Dat betekent dat Laravel liever een conventie hanteert, in plaats van expliciet dingen in te stellen. Bijvoorbeeld: als je een model maakt dat `Team.php` heet, dan zal Laravel automatisch zoeken naar een tabel die "teams" heet. Want de conventie is dat je model dezelfde naam heeft als de tabel, maar dan in enkelvoud. Soms is dat even wennen, veel dingen zijn *impliciet*: een ervaren Laravel-developer weet dat het zo zit, maar het staat nergens beschreven.

Voordeel voor startende developers

Maar er is ook een groot voordeel: met een beetje logisch nadenken en 'afkijken' kun je heel snel dingen bouwen in Laravel. Bijvoorbeeld: onze zwerkbball-app heeft al een pagina die al toernooien toont. Het is vrij eenvoudig om dan ook een overzicht van teams te bouwen. Je kunt veel dingen kopiëren en ervanuit gaan dat je simpel steeds "tournament" moet vervangen door "team". Natuurlijk moet je wel blijven nadenken wat je doet. De regel is: kopiëren – plakken – nadenken.

10. Opdracht: teams toevoegen

In de zwerkbball-app willen we ook een overzicht van teams bijhouden. De tabel hiervoor is gemaakt, nu gaan we de 'index'-pagina bouwen. Deze lijkt sterk op de toernooien-pagina.

Een vooruitblik op het eindresultaat:

Team	Soort	Herkomst
Zwadderich	School	Zweinstein
Nationaal Belgisch Zwerkbballteam	Country	België
Griffioendor	School	Zweinstein
Zwerkbball Zwabbers	Commercial	

Pak hulpkaart 3 ('Laravel: structuur en werking') erbij. Gebruik de flowchart op de voorkant.

1. Als we een *request* doen naar <http://127.0.0.1:8000/teams>, dan zal je applicatie in de routes-file kijken of je hebt vastgelegd hoe deze GET-request naar /teams moet worden afgehandeld.
 - a. Open de routes-file (`routes/web.php`).
 - b. Op welke regel wordt deze *request* afgehandeld?
 - c. Naar welke controller wordt de *request* doorgestuurd, en welke methode wordt aangeroepen?

2. De volgende stap in het proces is de controller. Open de controller die genoemd is in de routes-file (map: `app/Http/Controllers/`).
 - a. Als je de link voor teams opent in je browser, dan krijg je een foutmelding. Waar zie je dat gebeuren in de controller? Verwijder deze regel alvast.
 - b. Onze code voor teams moet precies hetzelfde doen als die voor de tournaments. Open de `TournamentController` en bekijk de `index`-methode.
 - c. Kopieer de `index`-methode uit de `TournamentController` naar de `TeamController`. Uiteraard pas je de code zo aan dat overal 'team' staat in plaats van 'tournament'. Let bij het aanpassen op of je wel / geen hoofdletter moet gebruiken (kopiëren – plakken – nadenken).

```
public function index()
{
    $teams = Team::all();
    return view('teams/index')
        ->with('teams', $teams);
}
```

3. De `TeamController` zal als eerste proberen om via het `Team`-model alle teams op te halen. Controleer of het `Team`-model bestaat. Kijk in de map `app/Models/`. De file zelf mag vrijwel leeg zijn, omdat het alle functionaliteit overneemt uit het basis-model (zie het stukje "extends Model" op r8).
4. Tenslotte gaat de controller een *response* retourneren. Hiervoor wordt een view gebruikt. De view is al gemaakt, maar nog leeg. Open de view (kijk eventueel op de hulpkaart waar je de views kunt vinden).
5. Test je code: type een paar woorden in de code van de view. Klik nu in de browser op de link naar Teams. Je ziet een witte pagina, met enkel jouw net geschreven tekst. Zie je nog steeds een foutmelding? Controleer dan of je de `index`-methode van de `TeamController` juist hebt aangepast.
6. In het screenshot kon je al zien dat de teams-pagina vrijwel identiek is aan de toernooien-pagina. Ook voor de view gaan we dus kopiëren – plakken – nadenken.
 - a. Open de view `tournaments/index(.blade.php)`.
 - b. Kopieer de inhoud naar `teams/index`.
 - c. Pas de pagina aan, zodat alles precies klopt met het screenshot bovenaan de opdracht.
 - d. Tip: vergeet niet om in de bovenste section ('nav') de active-klasse op een andere link te zetten.
 - e. Tip: op de teams-pagina hebben we nog geen links voor nieuw en/of aanpassen.
 - f. Voeg intussen voorbeelddata in via `phpMyAdmin`, anders kun je natuurlijk niet goed testen of de view ook werkt in de browser.
7. Bonusopdracht, niet verplicht: maak seeders om standaard een aantal teams aan te maken. Werk in de file `database/seeders/DatabaseSeeder.php`. Bestudeer eerst hoe de bestaande seeders werken.

Herhalingsoefeningen - Deze oefeningen zijn niet verplicht, ze zijn bedoeld om extra te oefenen.

1. Van een toernooi willen we ook nog bijhouden wat de einddatum is:
 - a. Voeg een DATE-kolom toe aan de migration, om de einddatum in op te slaan. Deze kolom moet nullable zijn, want soms is nog niet bekend wat de einddatum zal zijn.
 - b. Voer je migrations en seeds opnieuw uit (`php artisan migrate:fresh --seed`).
 - c. Zorg dat er testdata in de einddatum-kolom zit, eventueel via `phpMyAdmin`.
 - d. Pas de view in `resources/views/tournaments/index.blade.php` aan zodat de einddatum ook getoond wordt op de toernooien-pagina.
2. We willen een tabel bijhouden met scholen waar men zwerkbal kan leren. Maak een migration voor een 'schools'-tabel met de kolommen `id`, `name`, `location` (en timestamps). Voer de migration uit en vul de tabel met testdata ([zie eventueel hier](#)).
3. Maak een pagina om de lijst met scholen te bekijken. Gebruik de flowchart uit hulpkaart 3. Gebruik het commando `php artisan make:model School -c -r` om een model en controller te maken. Voeg handmatig een regel aan de routes-file toe, en maak handmatig een file in `resources/views`.

Weekcheck WDV-V-2: Kolommen toevoegen

In de 4Shop ontbreken nog twee belangrijke functies, waar we nu voorbereidingen voor gaan treffen:

- We willen graag de mogelijkheid hebben om een actie-prijs in te stellen. Hiervoor willen we het zo doen dat er een instelbaar kortingspercentage wordt bijgevoegd aan elk product.
- Van een order moet het mogelijk zijn om te geven of deze is bezorgd of niet (net als we nu al bijhouden of een order is betaald).

1. Open je lokale 4Shop in Visual Studio Code, en start de applicatie met **php artisan serve**. Let op: als je de zwerkbal-app nog had draaien, moet je die nu eerst afsluiten (ctrl + c in de terminal). Anders kan het zijn dat je niet de 4Shop krijgt te zien in de browser, maar de zwerkbal-app.
2. Open de migration voor de products-tabel. Voeg een kolom toe van type decimal (<https://laravel.com/docs/8.x/migrations#column-method-decimal>):
 - Naam: discount
 - Precision: 4
 - Scale: 1
3. Geef deze kolom een default-waarde van 0 (<https://laravel.com/docs/8.x/migrations#column-modifiers>).
4. Doe een 'fresh' migratie met **php artisan migrate:fresh --seed**. Voer bij minstens één product als testdata korting in via phpMyAdmin (bijvoorbeeld: 25.0 om zodoende 25% korting te geven).
5. Nu willen we zorgen dat overall in de applicatie de juiste prijs wordt getoond. Daarvoor plaats je de volgende functie in de file `app/Models/Product.php`. Deze functie overschrijft de prijs die wordt opgevraagd door de een nieuwe prijs, maar dan minus korting. Kijk in je browser of je het effect ziet.

```
public function getPriceAttribute($value)
{
    $discount = $value * ($this->discount / 100); //Korting in euro's
    $final_price = $value - $discount;           //Haal korting af van prijs
    return number_format($final_price, 2);      //Zorg altijd voor 2 decimalen
}
```

6. Het moet voor de bezoeker duidelijk zijn dat er korting van toepassing is. Open de view `products/index.blade.php`. Alléén als de waarde voor `$product->discount` groter is dan 0, toon je deze tekst onder de beschrijving. Let ook op de styling:



Bonusopdracht: toon in de rode tekst ook de originele prijs ("Nu 12.5% korting! Originele prijs: €29.00"). Zie voor inspiratie <https://stackoverflow.com/questions/34181119/>.

7. Tenslotte doen we een beetje voorbereiding voor het bijhouden of een order geleverd is:
 - a. Open de migration van de orders-tabel. Voeg een kolom toe van het type boolean met als naam 'delivered' en default-waarde false. Dit is precies hetzelfde als de kolom 'payed'.
 - b. Doe een 'fresh' migratie met `php artisan migrate:fresh --seed`.
 - c. Open de view in (resources/views/)admin/orders.blade.php. Op r32 t/m 34 kun je zien hoe de kolom 'payed' wordt weergegeven. Doe exact hetzelfde voor 'delivered'.
 - d. Plaats een of meer bestellingen om te testen (het betalen is uitgeschakeld, geen zorgen).
 - e. Ga naar het admin-gedeelte. Zie de README.md voor info over inloggen. Je zou nu ongeveer dit moeten zien:

Inleveren

Precies één screenshot met daarop twee zichtbare schermen:

- Links: phpMyAdmin scherm. Je hebt ergens in de app op de tab "SQL" geklikt en precies deze query uitgevoerd: `SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, COLUMN_TYPE FROM INFORMATION_SCHEMA.COLUMNS WHERE COLUMN_NAME IN('delivered', 'discount')`
- Rechts: de homepage van je app, waarop minstens één product met korting zichtbaar is.

Let op dat in de <title>-tag je eigen naam staat. Plaats geen tekst bij het inleveren. Deze regels helpen ons om iedereen snel van feedback te kunnen voorzien.

Voorbeeld ingeleverd screenshot:

3. Controllers, routes en view

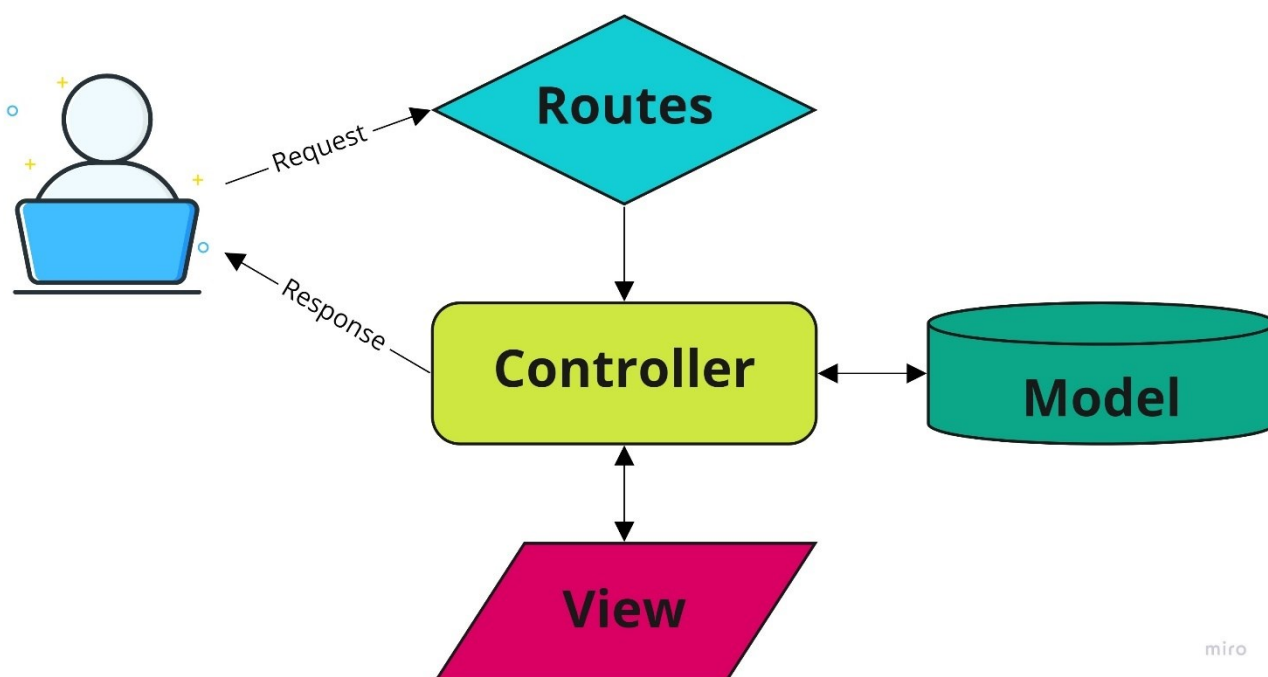
Als eerste wordt in de klas aandacht besteed aan de vorige weekcheck.

1. Requests, MVC en CRUD

In de afgelopen twee hoofdstukken heb je kennigemaakt met de basis van Laravel. Je hebt geleerd hoe het MVC-principe in elkaar steekt en je weet hoe je eenvoudige wijzigingen kunt maken aan de database en de HTML. In dit hoofdstuk gaan we meer in detail kijken naar routes en controllers, zodat je uiteindelijk met formulieren kunt gaan werken.

Hoe was het ook alweer?

Iedere *request* komt binnen in de routes-file. Een gebruiker die een item wil aanpassen zal twee requests doen; eerst het formulier opvragen en vervolgens het formulier versturen. De controller handelt beide requests af en gebruikt daarbij een model en/of een view. Daarna stuurt de controller een response, bijvoorbeeld het opgevraagde formulier als HTML-pagina. De response kan overigens ook een *redirect* zijn naar bijvoorbeeld de index.



CRUD in Laravel

Uit WDV-III weet je dat er vier soorten acties zijn op één resource: Create, Read, Update en Delete. In Laravel is de controller verantwoordelijk voor al die acties. Bijvoorbeeld ook het tonen van het aanpas-formulier (dus niet alleen het opslaan ervan). Alle acties hun eigen standaard-naam.

Dat maakt dat routes en controllers in Laravel zeven standaard-acties kennen:

	Omschrijving	Soort request	URL van route	Methode in controller	Naam van route
Create	Toon nieuw-formulier	GET	/photos/create	create()	photos.create
	Opslaan nieuw item	POST	/photos	store()	photos.store
Read	Toon alle items	GET	/photos	index()	photos.index
	Toon één item	GET	/photos/{photo}	show()	photos.show
Update	Toon edit-formulier	GET	/photos/{photo}/edit	edit()	photos.edit
	Opslaan aangepast item	PUT	/photos/{photo}	update()	photos.update
Delete	Verwijderen item	DELETE	/photos/{photo}	destroy()	photos.destroy

Dus als we straks een toernooi willen maken doet onze gebruiker in totaal twee requests:

- Klik op "+ nieuw": GET-request naar /tournaments/create, afgehandeld door create()-methode.
- Verstuur formulier: POST-request naar /tournaments, afgehandeld door de store()-methode.

2. Opdracht: invullen

Maak de volgende zinnen af, net zoals de zinnen hierboven:

1. Als onze gebruiker alle toernooien wil bekijken kost dat in totaal één request:

- Klik op "toernooien": _____-request naar / _____ afgehandeld door de _____-methode.

2. Als onze gebruiker een toernooi wil aanpassen gebeuren er twee requests:

- Klik op "aanpassen": _____-request naar / _____ afgehandeld door de _____-methode.
- Klik op "opslaan": _____-request naar / _____ afgehandeld door de _____-methode.

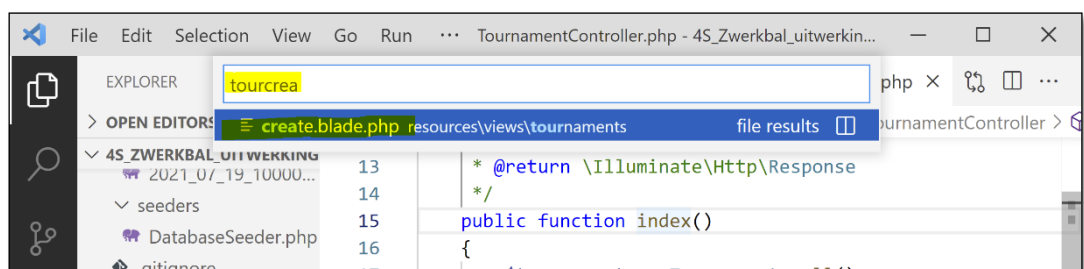
3. Als onze gebruiker een toernooi wil verwijderen kost dat in totaal één request:

- Klik op "verwijderen": _____-request naar / _____ afgehandeld door de _____-methode.

3. Opdracht: zoeken en tekenen

Er is al een formulier voor het maken van een toernooi, maar dit bevat alleen nog het veld voor "naam". Intussen hebben we aan de toernooien-tabel ook een datum en start-tijd toegevoegd. We moeten dus de code rondom het maken van een toernooi updaten. We gaan eerste onderzoeken hoe alles werkt.


1. Als we op "+nieuw" klikken, wordt de create()-methode van de TournamentController ingeschakeld.
 - a. Op welke regel in de file routes/web.php kun je dat zien?
 - b. Deze methode opent een view en retourneert deze. Welke view?
2. Open de genoemde view. Tip: je hoeft niet altijd te zoeken in de mappenstructuur. Als je "ctrl + p" drukt opent het "Go to file" menu. Type (delen van) de bestandsnaam die je wil openen en druk enter om te bevestigen.



3. Kijk in de form-tag waar het formulier in deze view naartoe wordt gestuurd. Je ziet als het goed is de naam van een route staan. Zoek deze route op in de routes-file. Welke methode in welke controller roept deze route aan?
4. Open de genoemde methode. Wat retourneert deze methode uiteindelijk?
5. Schets hieronder het hele proces. Vanaf het klikken op de link tot de laatste response. Welke stappen vallen daartussen?



4. Opdracht: een toernooi maken

1. Open de view tournaments/create, hierin vind je het formulier om een toernooi te maken.
2.  Link naar een view: één van de conventies van Laravel is het weglaten van delen van de locatie van een view. Als we zeggen “open de view tournaments/create” dan zeg je eigenlijk “open de file **resources/views/tournaments/create.blade.php**”. De twee dikgedrukte stukken zijn voor *iedere* view hetzelfde, en dus laten we die weg. Laravel vult het zelf ook aan als dat nodig is.
3. Bekijk eerste hoe het veld voor toernooi-naam nu is gemaakt. Voeg dan twee velden toe. Het is conventie om de velden hetzelfde te noemen als de database-kolom waar ze uiteindelijk bij horen. Let ook op dat je de labels en id's aanpast:
 - a. Een input[type=date] met als naam 'date'.
 - b. Een input[type=time] met als naam 'start_time'.
3. Open de TournamentController en zoek de store()-methode. Vervang de bestaande r40 t/m 42 door het onderstaande blokje. Dit is de Laravel-manier om snel en eenvoudig datavalidatie te doen (denk even terug aan WDV-III waar je handmatig if-statements moest schrijven om dit te doen).

```
$request->validate([
    'name' => 'required',
    'date' => 'required|date_format:Y-m-d',
    'start_time' => 'required|date_format:H:i'
]);
```


- a. Als de validatie niet slaagt, stuurt Laravel je terug naar het formulier. Maar je ziet de errors niet vanzelf. Daarvoor moet je ergens in de view van het formulier ongeveer deze code toevoegen:

```
@if($errors->any())
    <ul class="errors">
        @foreach($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>
@endif
```

- b. Test de validatie (en of je de errors ziet) door ongeldige waarden in te voeren en op te slaan.

4. Terug naar de TournamentController. Rond regel 46 zou je dit blokje code moeten tegenkomen. Het maakt een nieuw toernooi aan, stelt de naam in en slaat dan op. Pas de laatste regel met save() zorgt ervoor de het toernooi echt in de database komt!

```
$tournament = new Tournament();
$tournament->name = $request->name;
$tournament->save();
```

- a. Breid deze code uit (kopieer de middelste regel) zodat ook de date en start_time vanuit de request worden opgeslagen in de \$tournament.
- b. Test het toevoegen door enkele toernooien aan te maken met geldige waarden. Komen ze in de lijst erbij?

5. Opdracht: een toernooi aanpassen

Voer dezelfde aanpassingen door voor het aanpassen van een toernooi. Tips hierbij:

- Werk in de view tournaments/edit.
- Let op dat je hier ook de value van iedere input moet invullen. We gaan immers een bestaand toernooi aanpassen. Je hoeft niet zelf een query te schrijven. Laravel zorgt er automagisch voor dat je kunt beschikken over een \$tournament.
- De value van het date-veld werkt niet altijd zoals verwacht. Probeer eventueel {{ \$tournament->date->format('Y-m-d') }}. Hiermee forceer je de datum in het juiste formaat.
- Pas vervolgens de update()-methode in de TournamentController aan. De validatieregels zijn hetzelfde als bij de store(). Je hoeft niet alles uit je hoofd te doen, maar onthoud: kopiëren – plakken – **nadenken**.

6. Opdracht: teams maken

Voor het maken van een nieuw team is nog helemaal geen code aanwezig. Je gaat het hele stuk van router, naar controller naar view zelf opzetten. Gebruik eerste de tabel op p20 op deze vragen aan te vullen;

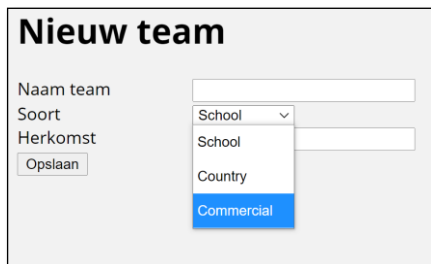
Als de gebruiker een team wil maken gebeuren zijn er twee requests nodig:

- Na klik op “+ nieuw team”: een _____-request naar / _____
afgehandeld door de _____-methode.
- Na klik op “opslaan”: een _____-request naar / _____
afgehandeld door de _____-methode.

1. Open de file routes/web.php en maak twee nieuwe regels voor de bovenstaande requests. Uiteindelijk moet het blokje van de teams in je routes-file er dus zo uitzien:

```
Route::get('/teams', [TeamController::class, 'index'])->name('teams.index');
Route::get('/teams/create', [TeamController::class, 'create'])->name('teams.create');
Route::post('/teams', [TeamController::class, 'store'])->name('teams.store');
```

2. Een GET-request naar /teams/create zou het formulier moeten tonen. Deze request wordt afgehandeld door de create()-methode in de TeamController. Plaats in deze methode en regel code om de juiste view te retourneren (de view zelf bestaat nog niet).
3. Maak nu ook de view teams/create.blade.php aan. Neem eventueel de structuur over uit bijvoorbeeld tournaments/create. Let wel op: kopiëren – plakken – nadenken. Vergeet bijvoorbeeld niet om de “active”-klasse in de nav-sectie op de juiste link te zetten. Uiteindelijk moet je formulier er zo uitzien:



4. Wanneer het formulier wordt verstuurd, kom je dus terecht in de store()-methode. Bouw deze als volgt;
 - a. Als eerste de validatie, de code is als volgt:

```
$request->validate([
    'name' => 'required',
    'type' => 'required'
]);
```
 - b. Dan maak je een nieuw team aan: `$team = new Team();`
 - c. En vervolgens neem je name, type en origin over van de request naar het team:
`$team->name = $request->name;`
 - d. Dan sla je het team op: `$team->save();`
 - e. En tenslotte retourneer je een redirect naar de route teams.index.
5. Test je implementatie;
 - a. Lukt het om nieuwe teams te maken?
 - b. Zie je alle gegevens ook verschijnen in de lijst / tabel?
 - c. Krijg je een validatie-error als je bijvoorbeeld de naam leeglaat?

7. Les 2: feedbackmoment

Feedbackmoment FC12-WEBw03: theoretische vragen

Eind week 3 of begin week 4 staat het feedbackmoment op de planning. Je krijgt theoretische vragen over alle stof tot aan dit blokje. Je ontvangt instructies van de docent.

8. 'Custom' controllers

Niet iedere controller past in het verhaal van CRUD en de zeven standaard-acties uit de tabel die je eerder zag. Denk bijvoorbeeld aan de HomeController van de zwerkbal-app. Deze heeft ook niet alle methodes die je in andere controllers vindt. De conventie binnen Laravel is: een controller gaat over alle zeven acties van een resource óf over andere dingen.

Dus: stel dat we een lijst willen maken van aankomende toernooien. Dan valt dat niet direct onder een van de zeven standaard-acties. Daar maken we dus een aparte controller voor, bijvoorbeeld de UpcomingTournamentController.

Maken van een controller

Er zijn twee manieren om een controller via het artisan 'make'-commando:

- Een standaard-controller maak je tegelijk met het model. De toevoegingen `-c -r` betekenen: "geef mij ook een controller, en maak het een resourcecontroller (dus met de zeven standaard-acties):"
 - `php artisan make:model Team -c -r`
- Een custom-controller maken is eigenlijk eenvoudiger. Het volgende commando levert een vrijwel lege controller op, waarin je zelf de methodes moet maken:
 - `php artisan make:controller UpcomingTournamentController`

🔗 Benamingen: wees niet te kort in je naamgeving, het kan beter duidelijk zijn dan kort. Je hoeft je tegenwoordig toch geen zorgen te maken over een letter meer-of-minder qua opslagruimte en bandbreedte. Kies dus liever een beschrijvende naam zoals "UpcomingTournamentController", en niet een naam die te ver is afgekort zoals `U_ToursController`. Zulke cryptische namen geven later voor jezelf (en collega's!) alleen maar onduidelijkheid

9. Namespaces in PHP en de 'use' directive

Wanneer je een custom-controller maakt, krijg je minder code automatisch voorgeschoteld. Zie het voorbeeld hieronder, en let vooral op regel 5. Je ziet in de TeamController rechts een belangrijke regel staan. Zonder die regel kunnen we het Team-model niet gebruiken om bijvoorbeeld alle teams op te halen.

SomeCustomController.php	TeamController.php
<pre>app > Http > Controllers > SomeCustomController.php > ... 1 <?php 2 3 namespace App\Http\Controllers; 4 5 6 use Illuminate\Http\Request; 7 8 class SomeCustomController extends Controller 9 { 10 // 11 } 12</pre>	<pre>app > Http > Controllers > TeamController.php > TeamController > store 1 <?php 2 3 namespace App\Http\Controllers; 4 5 use App\Models\Team; 6 use Illuminate\Http\Request; 7 8 class TeamController extends Controller 9 { 10 /** 11 * Display a listing of the resource. 12</pre>

Wat gebeurt hier nu precies?

In Laravel is je code verdeeld over veel verschillende files (vooral models, views en controllers). Toch hoef je nergens een 'require_once' te gebruiken om andere php-files in te laden. Laravel regelt dat allemaal automatisch voor je (of eigenlijk: composer, onze package manager, regelt dat).

Dus als jij schrijft `$teams = Team::all()`, dan gaat Laravel opzoek naar de class `Team`. Het systeem zoekt dan in de eigen namespace. Je ziet hierboven op r3 dat de controllers in een bepaalde namespace zitten. Je kunt dat vergelijken met een soort mappenstructuur.

Maar in die namespace `App\Http\Controllers` ga je het `Team`-model natuurlijk niet vinden. Daarom moet je Laravel met een “use” toch vertellen waar jouw model staat. De namespaces in Laravel komen exact overeen met de mappenstructuur, dus dat is makkelijk te onthouden.

? Error – class `XYZ` not found: deze foutmelding komt vaak voor bij beginnende Laravel-developers. In vrijwel alle gevallen ben je dan vergeten om bovenaan de file een “use” neer te zetten.

👉 10. ‘Custom’ query’s uitvoeren

Als het goed is heb je nog geen letter SQL geschreven in deze module. Dat doet het model allemaal voor ons. Als je alle toernooien wil hebben, doe je simpelweg `Tournament::all()`. Maar wat nu als je complexere query's wil doen? Bijvoorbeeld “alle toernooien waar de startdatum ligt na vandaag”.

Ook dit kan het model voor ons regelen. Vrijwel ieder SQL-commando kun je als methode aanroepen op een model. Dit leest al bijna als een SQL-query:

```
$tournaments = Tournament::where('date', '>', '2021-07-19')->orderBy('date')->get();
```

Get of first

Een custom-query kan op twee manieren eindigen:

- Je verwacht een lijst van items: `->get()`, vergelijkbaar met `fetchAll`, hier kun je `foreach` op doen
- Je verwacht precies één resultaat: `->first()`, vergelijkbaar met `fetch`, je kunt nu `$item` gebruiken

Vandaag in PHP

Overigens kijkt de bovenstaande query natuurlijk altijd naar dezelfde datum. Als je precies de huidige datum wil, gebruik je de functie `date("Y-m-d")`, die geeft de datum van vandaag in het formaat *year-month-day*.

```
$tournaments = Tournament::where('date', '>', date("Y-m-d"))->orderBy('date')->get();
```

👉 11. Limit in SQL

Soms wil je het aantal resultaten van een query beperken. Dat komt vaak voor als je gevraagd wordt om bijvoorbeeld een “top 3” te maken. De truc bij zo’n vraag is: sorteer op de juiste kolom, en limiteer dan het aantal resultaten. Bijvoorbeeld “geef de drie duurste producten” zou de volgende query kunnen opleveren:

```
SELECT * FROM products ORDER BY price DESC LIMIT 3
```

Vertaald naar de Laravel-manier krijg je dan zoiets:

```
$top3 = Product::orderBy('price', 'desc')->limit(3)->get();
```

Of stel dat je de allergeedkoopste order wil ophalen, dus echt maar één resultaat:

```
$cheapest = Order::where('payed', '=', 1)->orderBy('amount')->limit(1)->first();
```

Bedenk hierbij: als je niet opgeeft hoe de query moet sorteren, dan is het automatisch ASCending, dus oplopend, dus met de goedkoopste bovenaan. Als je dat limiteert naar één resultaat heb je dus de goedkoopste order te pakken.

12. Opdracht: aankomend toernooi

We gaan een pagina bouwen waarop je het eerstvolgende toernooi kunt zien. Het eindresultaat moet er ongeveer zo uit zien, denk dus ook aan een beetje styling:



1. **Bedenk eerst hoe je deze informatie vraag kunt aanpakken.** We hebben een tabel met toernooien en daar staat een datum bij. Maar als je gewoon sorteert op datum dan heb je bovenaan de lijst óf het allereerste toernooi ooit (dus ook in het verleden) óf het allerlaatste toernooi (ver in de toekomst).

Vul de volgende zin aan: *ik wil alle toernooien ophalen die voldaan aan de voorwaarde*

en die lijst wordt gesorteerd op de kolom _____ in de volgorde ASC / DESC.

Vervolgens limiteer ik dat tot één resultaat, en eindig de code dus met get() / first().

2. **Kijk nog eens naar de flowchart op hulpkart 3 om te zien wat je allemaal moet bouwen. Vul aan:**
Als we naar de link /upcoming gaan dan ontstaat een _____-request die wordt afgehandeld door de controller: _____. Deze controller krijgt één methode genaamd show(). Daarin gebruiken we het bestaande model

_____ om de query op te bouwen. Tenslotte geeft de controller een view terug, die moeten we nog maken. Een logische map en bestandsnaam zou zijn:

_____.

3. Maak een nieuwe custom-controller aan:
`php artisan make:controller UpcomingTournamentController`
4. Voeg nu een regel toe aan de file `routes/web.php`. Vergeet niet om bovenaan een use-directive toe te voegen dat verwijst naar de nieuw aangemaakte controller.
5. Maak in de controller een “`public function show()`”. Voeg bovenaan een use-directive om te verwijzen naar het juiste model.
6. Schrijf een query die je informatievraag beantwoord (één eerstvolgend toernooi). Gebruik de informatie uit paragrafen 10 en 11 en combineer die kennis om tot de juiste query te komen.
7. Maak een view aan (handmatig, hiervoor is geen make-commando). Wees éérst tevreden als je gewoon de gegevens op het scherm krijgt zonder styling. Ga daarna de pagina stylen zoals het voorbeeld. Tips:
 - Gebruik flexbox. Beide vakjes hebben dus één gezamenlijke *parent*.
 - Het linkse deel is 75% breed en heeft aan de rechterkant een marge van 5%.
 - Het rechtse deel heeft geen ingestelde breedte, maar krijgt vanzelf wat er over is.
 - Voor de twee vakjes: recycle eventueel de klasse die je op de homepage ook hebt gebruikt. De styling qua rand en font-size is namelijk precies hetzelfde.

Herhalingsoefeningen - Deze oefeningen zijn niet verplicht, ze zijn bedoeld om extra te oefenen.

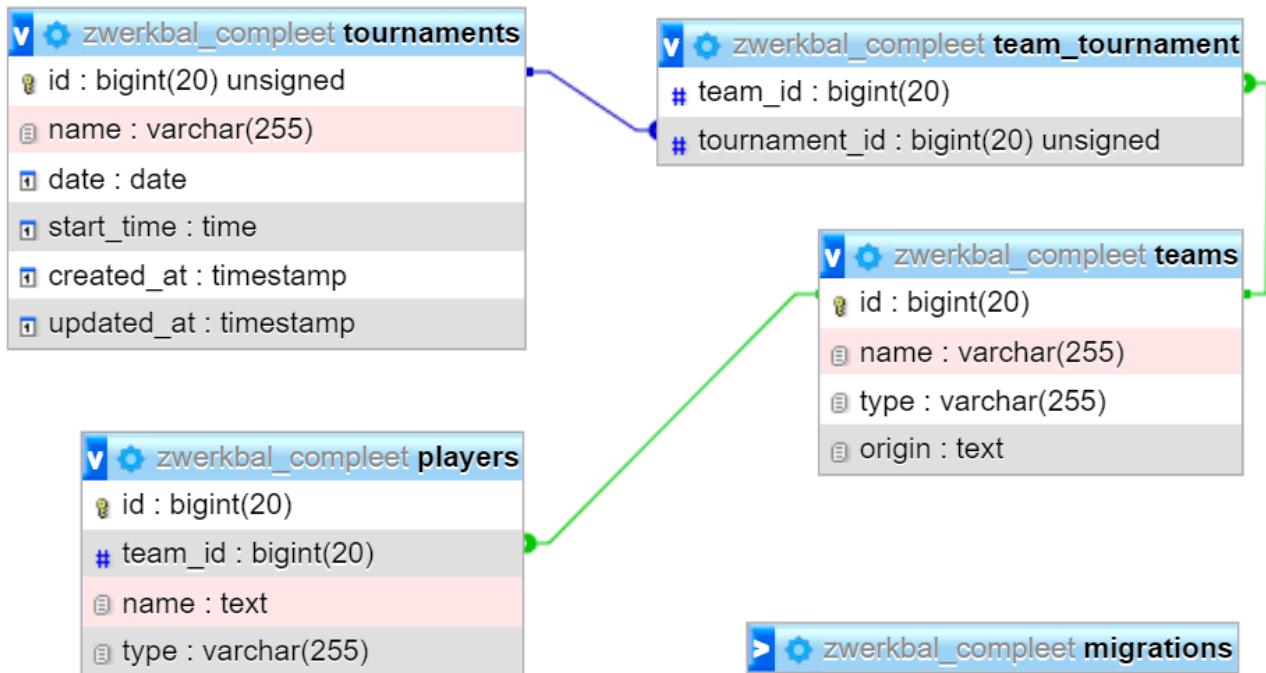
1. Zorg ervoor dat we teams ook kunnen aanpassen:
 - a. Voeg twee regels toe in de routes-file, zie de tabel op p20.
 - b. Zorg dat de `edit()`-methode in de controller de juiste view retourneert. Je moet hierbij wel even de bestaande `$team` meegeven. Zie de `TournamentController` voor hoe dat werkt.
 - c. Maak de view aan en bouw het formulier. Het form verstuurt naar de `update()`-methode.
 - d. Bouw de `update()`-methode in de controller. Validatieregels zijn hetzelfde als bij `store()`.
 - e. Plaats tenslotte een link naar de edit-pagina van ieder team in de lijst van teams. Zie eventueel de view `tournaments/index` voor een voorbeeld.

4. Relationale database

Docent: voor dit hoofdstuk staat één les. De LessonUp is *leading*, zie de gedeelde map op LessonUp.

1. ERD als database-diagram

Als we aan de slag gaan met databases kan het snel ingewikkeld worden. Het is dus handig om een schematische weergave te hebben van alle tabellen. Daarnaast kan een schema ons ook helpen om een bestaande database te begrijpen. Voor dit doel worden “Entity-Relationship Diagrammen” gebruikt. Het laat de relatie tussen verschillende tabellen (*entities*) zien in een schematisch overzicht. We korten Entity-Relationship Diagram vaak af tot ERD.



In het ERD kunnen we in één oogopslag meteen een gevoel krijgen voor de structuur van onze database. We zien de tabellen staan voor tournaments, teams en spelers. We zien ook de pijlen tussen die tabellen staan; dat zijn de ‘*Relationships*’.

Specifiek kunnen we bijvoorbeeld zien dat een player een team_id heeft. Vanaf dat team_id gaat er een pijl naar het id van teams. Dit geeft aan dat hier een koppeling tussen zit. Als we bijvoorbeeld een speler hebben die als team_id 2 heeft, moet er een corresponderend team zijn wat als id 2 heeft. Die speler is dan lid van dat team. We zeggen dat team_id een *foreign key* is naar id van de teams tabel.

2. Foreign Keys

We zijn al bekend met het concept van een *Primary Key*; een kolom die voor elke rij in de tabel uniek identificeerbaar is. In de praktijk maken wij hier altijd een integer van die automatisch ophoogt (AUTO_INCREMENT) en deze noemen we altijd id.

Omdat een primary key dus een rij uit een tabel uniek identificeerbaar maakt kunnen we ook vanuit een tabel refereren naar een andere tabel door gebruik te maken van dat unieke identificatienummer. We noemen dit een foreign key.

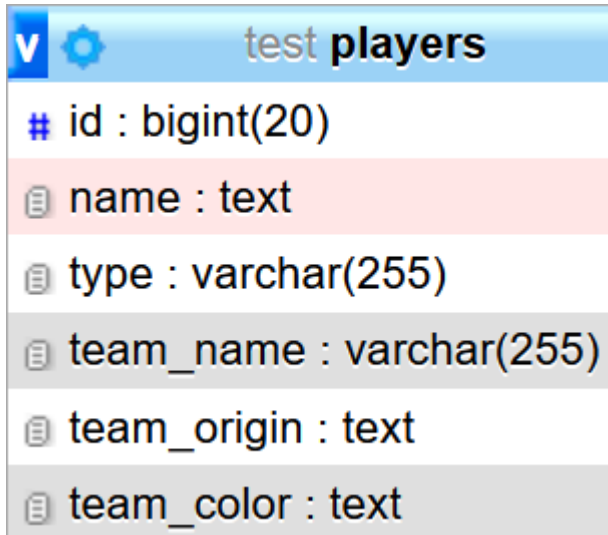
? Foreign key betekent letterlijk vertaald “vreemde sleutel”. Dit is om aan te geven dat het dus een “key” is die vreemd is aan de tabel, met andere woorden komt deze dus uit een andere tabel.

Maar waarom slaan we niet gewoon als eigenschap van een speler op in welk team hij zit? We hebben een filmpje wat hierover gaat. In dit filmpje gaat het over een tabel van een todo-app, waarbij informatie moet worden opgeslagen over de eigenaar van een ticket. Kijk het volgende filmpje:

<https://youtu.be/AhziVGAYKs8>

Om te zien waarom dit ook voor ons handig kan zijn kunnen we ons indenken dat we bij de spelers meteen het team zouden opslaan. In dat geval willen we dezelfde informatie bijhouden per team, maar dan in de tabel van de speler zelf. Zo'n tabel heeft dan dus alle informatie over de speler, én alle informatie over het team waar de speler in zit. Op dezelfde manier als in het voorbeeld uit het filmpje zouden we dan overbodig extra informatie opslaan die we niet nodig hebben.

Zo'n samengevoegde tabel zou er zo uit kunnen zien:



test players	
#	id : bigint(20)
	name : text
	type : varchar(255)
	team_name : varchar(255)
	team_origin : text
	team_color : text

Hier slaan we dus voor elke speler op wat hun teamnaam is, maar omdat we ook extra informatie over teams willen hebben slaan we ook de herkomst en de teamkleur op.

Meteen kunnen we twee problemen constateren:

- We slaan onnodig data dubbel op. Twee spelers in hetzelfde team moeten allebei dezelfde teamnaam, herkomst, en kleur opslaan.
- Als er ooit een aanpassing komt aan een team (bijvoorbeeld als een team een andere kleur gaat dragen) moet die aanpassing op heel veel verschillende plekken gebeuren.

Beide problemen lossen we op door gebruik te maken van een foreign key. In plaats van het opslaan van het team bij de speler slaan we de teams op in een aparte tabel, waar we naar kunnen verwijzen vanuit de players tabel. We krijgen dan weer de situatie zoals in het eerdere ERD.

3. ON DELETE

Een foreign key wijst dus naar de primary key van een andere tabel. Dat wil zeggen, er wordt een koppeling gemaakt die intact moet zijn. Als een speler een foreign key heeft naar een team is het vanzelfsprekend dat dat team ook moet bestaan. Dit concept heet "data-integriteit". Je wilt nooit dat het voorkomt dat je een foreign key hebt die niet correspondeert met een rij in de tabel waar je naar wijst.

Om problemen met data-integriteit te voorkomen kunnen we bij een foreign key aangeven wat er moet gebeuren voor een rij in je tabel als de rij waar je naar verwijst met een foreign key wordt verwijderd.

Bijvoorbeeld: wat moet er gebeuren met een speler als het team waar hij in zit wordt verwijderd? Elke speler heeft immers een team nodig (de kolom is niet nullable).

SQL biedt een aantal manieren om dit op te lossen:

- Cascade
- Set null
- No action

Bij een cascade (*Engels: waternival*) wordt, als de rij waar je naar verwijst wordt verwijderd, ook de rij die ernaar verwijst verwijderd. Bijvoorbeeld: als het team wordt verwijderd, worden ook alle spelers uit de database gehaald.

Bij een set null wordt geprobeerd de foreign key op null te zetten. Dit kan natuurlijk alleen als de foreign key nullable is.

Bij een no action wordt er niets gedaan. Dit betekent dat het niet mogelijk is een team te verwijderen als er nog spelers in zitten, je krijgt dan een foutmelding.

In de meeste gevallen wil je waarschijnlijk de cascade gebruiken.

👉 4. Foreign key toevoegen in je migration

Een foreign key is een normale kolom die dient als verwijzing naar een andere tabel. Zodoende moeten we dus eerst een kolom aanmaken die gaat dienen als onze foreign key. Dit doen we net als normaal. We werken door met het voorbeeld waar we een speler aan een team koppelen:

In de spelers-migration:

```
$table->unsignedBigInteger('team_id');
```

Nu hebben we een kolom genaamd team_id in onze spelerstabel zitten. Nu moeten we nog een extra commando uitvoeren om deze kolom om te zetten naar een foreign key:

In de spelers-migration:

```
$table->foreign('team_id')->references('id')->on('teams');
```

Deze laatste regel gaan we een klein beetje uit elkaar halen. De aanroep naar foreign geeft aan van welke kolom we een foreign key gaan maken. Dan geeft references aan wat de kolom gaat zijn waar we naar gaan wijzen (in dit geval heet die kolom id). Uiteindelijk geeft on aan welke tabel we naartoe verwijzen.

Omdat we op het gebied van on delete niets hebben aangegeven is dit dus een "on delete no action". Dit zou het voor ons niet mogelijk maken om een team te verwijderen zolang er nog spelers in zitten. Het is heel goed mogelijk dat dat de bedoeling is natuurlijk; in dat geval moeten we eerst elke speler toekennen aan een ander team voordat we het team kunnen verwijderen.

Mochten we de "nucleaire" optie willen kiezen dan kunnen we de foreign key ook definiëren met een cascade. Dat doen we door een aanroep naar onDelete achteraan bij te voegen:

In de spelers-migration:

```
$table->foreign('team_id')->references('id')->on('teams')->onDelete('cascade');
```

Voor meer informatie over foreign key constraints in Laravel kun je hier kijken:

<https://laravel.com/docs/9.x/migrations#foreign-key-constraints>

Let op: de shorthand notatie met foreignId zoals je in de docs tegenkomt, is pas in Laravel 7.0 geïntroduceerd. Dit werkt dus niet in de 4Shop-applicatie, want die draait op Laravel 5.8. Je kunt in alle versies van Laravel de manier gebruiken die hierboven staat uitgelegd (eerst een kolom maken, daarna pas aanwijzen als foreign key).

Weekcheck WDV-V-4: Relationele vragen

Maak de vragen op Itslearning behorend bij deze weekcheck.

5. CRUD in Laravel

Als eerste wordt in de klas aandacht besteed aan de vorige weekcheck.

1. Terugblik: relaties tussen tabellen

In het vorige hoofdstuk heb je geleerd dat tabellen gerelateerd kunnen zijn aan elkaar. Vrijwel iedere applicatie die je gebruikt werkt op die manier. Je kunt dat vaak wel afleiden aan de interface. Bijvoorbeeld:

- Hlearning: één vak heeft meerdere studenten, en een student heeft meerdere vakken (*veel-op-veel*).
- Teams: een team heeft meerdere kanalen, maar een kanaal hoort altijd bij één team (*een-op-veel*).

Noteer zelf nog minstens drie van deze zinnen. Zoek de relaties in de interface van jouw favoriete apps. Het helpt als het een applicatie is die je goed kent en veel gebruikt:

○

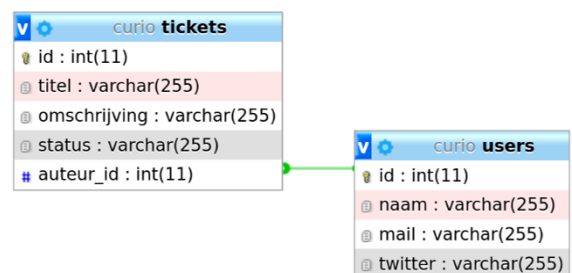
○

○

2. Relaties gaan twee kanten op

We gaan ons voorlopig alleen bezighouden met één-op-veel relaties (die zijn eenvoudiger dan veel-op-veel). Je hebt daarover een aantal dingen geleerd:

- Een relatie is in de database vastgelegd met een *foreign key*. Deze is aanwezig aan de “veel” kant.
- Een relatie gaat twee kanten op, in dit voorbeeld:
 - a) Eén user heeft meerdere tickets.
 - b) Een ticket heeft maar één user.
- Omdat het ticket de “veel” kant is, heeft die een *foreign key* `auteur_id` naar de tabel `users`.
- Je kunt in een ERD aan de uiteindes van het lijntje ook zien wat de een/veel kant is.



👉 3. Relaties uitleggen aan je model

We blijven bij het voorbeeld hierboven. Het doel is dat we uiteindelijk heel eenvoudig van een user alle tickets kunnen opvragen. Ook weer zonder zelf een query te schrijven:

```
$user = User::find(1);
$tickets = $user->tickets;
var_dump($tickets);           // Geeft een lijst van alle tickets van gebruiker nr. 1
```

Of andersom, we hebben we een ticket en willen niet alleen het auteur_id maar juist de naam hebben:

```
echo $ticket->user->naam;     // Zoekt uit de database alle gegevens van de maker
                             // van het ticket. Daarom kunnen wij de naam echo'en
```

Je hebt in het vorige hoofdstuk geleerd hoe je een relatie kunt vastleggen in een migration. Het staat dan goed in de database, maar Laravel weet nog niets van de relatie. Voordat we het bovenstaande kunnen doen, moeten we nog aan onze models uitleggen hoe de relatie in elkaar steekt.

Relatie definiëren in een model

Voeg aan het model een “public function” toe met de naam van de relatie. Let goed op het gebruik van enkelvoud / meervoud! Een User heeft meerdere tickets, maar een Ticket slechts één user.

```
class User extends Model
{
    use HasFactory;

    public function tickets()
    {
        return $this->hasMany(Ticket::class);
    }
}
```

```
class Ticket extends Model
{
    use HasFactory;

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

In die nieuwe functie leg je vast wat voor soort relatie het is, en welk model aan de andere kant hoort. Meer informatie vind je in de docs onder de kopjes “One To Many” en “One To Many (Inverse)”. Zie: <https://laravel.com/docs/9.x/eloquent-relationships>.

🔗 **Versies:** let op dat er kleine verschillen zijn met oudere versies van Laravel. Bij de 4Shop wordt Laravel 5.8 gebruikt. In de documentatie kun je rechtsboven kiezen voor het juiste versienummer om het verschil te zien.

Relatie gebruiken

Dankzij de “magie” van Laravel kun je nu al `$user->tickets` gebruiken (zonder haakjes dus – dat heet voluit een *dynamic property* – je hebt een functie gemaakt die je eigenlijk als variabele benaderd). Dit kun je ook toepassen in je views, bijvoorbeeld:

Detailpagina van gebruiker met al zijn tickets:

```
Hier zijn alle tickets van {{ $user->naam }}:
@foreach($user->tickets as $ticket)

    <h3>Ticket nummer {{ $ticket->id }}</h3>
    <p>{{ $ticket->omschrijving }}</p>

@endforeach
```

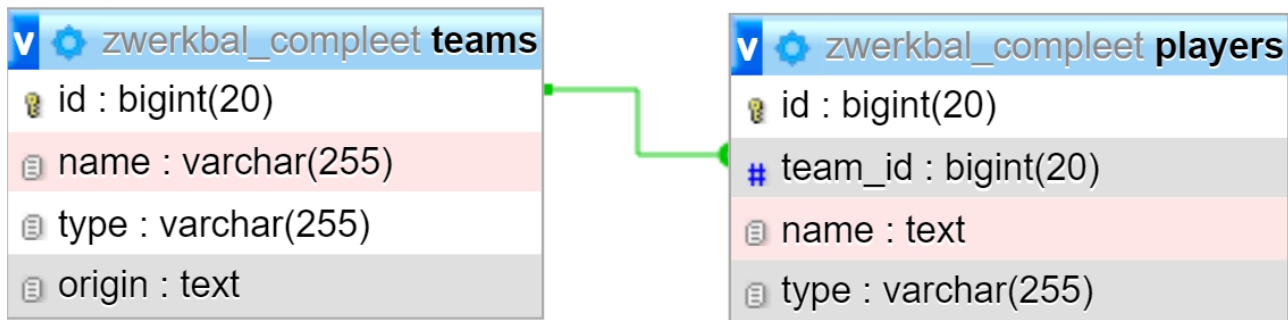
Detailpagina ticket met ook naam van maker:

```
<h1>Ticket {{ $ticket->id }}</h1>
<p>Gemaakt door: {{ $ticket->user->name }}</p>
<p>{{ $ticket->omschrijving }}</p>
```

4. Opdracht: spelers in de zwerkbal-app

In onze zwerkbal-app kunnen we nu toernooien en teams bijhouden. Met de spelers hebben we niet-voor-niets gewacht, hier is immers sprake van een relatie:

Eén team heeft meerdere spelers, maar een speler heeft slechts één team.



1. Bekijk de afbeelding hierboven en omcirkel / onderstreep de volgende dingen, zet ook de letter erbij:
 - a. *Foreign key*
 - b. *Primary key* (2x)
 - c. De relatie
 - d. De veel-kant (het symbool daarvoor)
2. Eerst gaan we database in orde maken:
 - a. Maak een nieuwe migration: `php artisan make:migration create_players_table`
 - b. Bouw de players-tabel op volgens het bovenstaande ERD. Let op: `team_id` moet unsigned zijn. Alle andere info qua datatypes haal je uit het ERD. Kijk eventueel in de docs onder het kopje Available Column Types (<https://laravel.com/docs/9.x/migrations>).
 - c. Nu moet in de migration ook de relatie worden opgebouwd. Voeg na de laatste kolom een regel om de relatie aan te geven. Gebruik deze zin om de juiste code te schrijven (zie ook H4):
"De *foreign key* `team_id` refereert aan de kolom `id` op de tabel `teams`. Bij verwijderen moet dat *cascaden* naar de spelers."
 - d. Test je migration. Je hebt een nieuwe file gemaakt, dus `php artisan migrate` is voldoende. Mocht je migration niet werken, dan moet je na het aanpassen wél een '`migrate:fresh`' doen.
3. Maak een model aan voor Player, en geef ons ook direct een controller met alle resource-methodes:
`php artisan make:model Player -c -r`
4. Een collega-developer heeft de seeder uitgebreid, zodat er ook test-spelers worden aangemaakt. Omdat jullie niet in dezelfde repository werken, heeft de collega een patch-file voor je gemaakt.
 - a. Download de file `seeder.patch` uit Itslearning en plaats in de hoofdmap van je zwerkbal-app.
 - b. Gebruik dit commando om de patch toe te passen: `git apply seeder.patch`
 - c. Bouw je hele database opnieuw op en draai meteen de nieuwe seeders:
`php artisan migrate:fresh --seed`

? Patch-files: werken met patches is een vrij oude manier om wijzigingen te delen met anderen. Het komt in het bedrijfsleven nauwelijks meer voor, omdat meestal alle developers verbonden zijn met de centrale repo op GitHub. In onze lessen is dat vaak niet zo, daarom werken we op school soms toch met patch-files.

5. We willen nu wel resultaat zien van onze inspanningen, dus we gaan een snelle schermdump van alle spelers maken:

- Maak een nieuwe routing-regel in `routes/web.php`: een GET-request naar `/teams` wordt afgehandeld door de `index()`-methode van de `PlayerController`. Vergeet niet om bovenaan een `use` te zetten voor deze controller.
- Open de `index()`-methode van de `PlayerController`. Voeg deze regel toe (als je een lijst van data retourneert, zal Laravel die als JSON op je scherm dumpen):
`return Player::all();`
- Open je applicatie en klik op Spelers, je zou nu een dump van alle spelers moeten zien. Gelukt? Goed bezig!

```

[
  {
    "id": 1,
    "team_id": 1,
    "name": "Ludo Bagman",
    "type": "chaser",
    "created_at": "2021-07-21T12:25:04.000000Z",
    "updated_at": "2021-07-21T12:25:04.000000Z"
  },
  {
    "id": 2,
    "team_id": 3,
    "name": "Argus Filch",
    "type": "keeper",
    "created_at": "2021-07-21T12:25:04.000000Z",
    "updated_at": "2021-07-21T12:25:04.000000Z"
  },
  {
    "id": 3,
    "team_id": 2,
    "name": "Marvolo Gaunt",
    "type": "beater",
    "created_at": "2021-07-21T12:25:04.000000Z",
    "updated_at": "2021-07-21T12:25:04.000000Z"
  },
  {
    "id": 4,
    "team_id": 1,
    "name": "Rubeus Hagrid",
    "type": "beater",
    "created_at": "2021-07-21T12:25:04.000000Z",
    "updated_at": "2021-07-21T12:25:04.000000Z"
  },
  {
    "id": 5,
    "team_id": 2,
    "name": "Lee Jordan",
    "type": "seeker",
    "created_at": "2021-07-21T12:25:04.000000Z",
    "updated_at": "2021-07-21T12:25:04.000000Z"
  }
]

```

? **JSON en leesbaarheid:** Laravel kan heel makkelijk data dumpen in JSON-formaat. Dat is fijn om te debuggen, allen het leest niet zo prettig. Er zijn verschillende browser-extensies in omloop om de JSON op je scherm leesbaar te formatteren. In het screenshot zie je de [JSON Formatter](#) uit de Chrome Web Store.

5. Opdracht: een team heeft spelers

We werken verder aan de zwerkbal-app. De spelers zijn toegevoegd (althans, de basis ervan), en nu stappen we terug naar de teams-pagina. Bij het overzicht moet een extra kolom komen met daarin een lijst van spelers in het team (inclusief hun type):

Team	Soort	Herkomst	Spelers
German National Quidditch team	Country	Duitsland	Ludo Bagman (chaser) Rubeus Hagrid (beater) Newt Scamander (seeker) Alicia Spinnet (chaser) Garrick Ollivander (chaser) Luna Lovegood (keeper)
Hufflepuff	School	Zweinstein	Horace Slughorn (chaser) Gregory Goyle (keeper) Colin Creevey (seeker)
4S Quidditch Team	Commercial		Argus Filch (keeper) Neville Longbottom (chaser) Susan Bones (chaser) Hannah Abbott (chaser)

- Om van een team alle spelers op te vragen, moeten we eerst aan het Team-model uitleggen wat de relatie is met het Player-model, en ook andersom. Kijk nogmaals naar het onderstaande voorbeeld en houd in je achterhoofd: een team heeft meerdere spelers, maar een speler heeft slechts één team. Voeg nu de relaties toe aan de Team- en Player-models.

```
class User extends Model
{
    use HasFactory;

    public function tickets()
    {
        return $this->hasMany(Ticket::class);
    }
}
```

```
class Ticket extends Model
{
    use HasFactory;

    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

- Open de view team/index.
 - Voeg een <th> toe met als inhoud "Spelers".
 - Voeg een <td> toe waarin je foreach't over \$team->players.

Deze foreach staat dus binnen de bestaande foreach waarin je de lijst van teams afgaat. Dat is ook logisch: er is een lijst van teams, en ieder team op zich heeft weer een lijst van spelers.

- Echo de naam en het type van iedere speler, zie vorige screenshot.

Bonus: groepeer de lijst van spelers op type. Zie het voorbeeld. Gebruik sortBy() en groupBy() op de zogenaamde "collectie" \$team->players. Zie ook de documentatie:

Team	Soort	Herkomst	Spelers
German National Quidditch team	Country	Duitsland	Beaters: Rubeus Hagrid Chasers: Ludo Bagman Alicia Spinnet Garrick Ollivander Keepers: Luna Lovegood Seekers: Newt Scamander

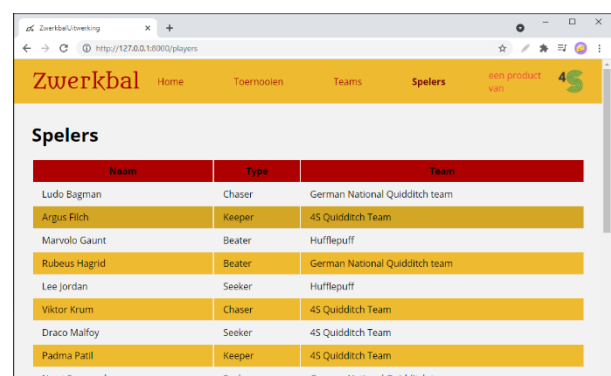
<https://laravel.com/docs/9.x/collections#available-methods>.

6. Opdracht: index van spelers

Terug naar de spelers-pagina. In plaats van een JSON-dump van alle spelers, moet er nu een nette tabel komen zoals op de andere index-pagina's. Maak voorlopig drie kolommen: naam, type en team.

- Pas eerst de PlayerController aan, zodat de index()-functie niet de lijst van spelers retourneert. In plaats daarvan sla je die lijst op een variabele, om vervolgens de view players/index terug te geven (die view krijgt de lijst met spelers mee).
- Maak een nieuwe folder 'players' in resources/views, en daarin een nieuw file index.blade.php.
- Deze view is vrijwel gelijk aan de andere index-views. Zie het kleine screenshot hierboven. Maar let op, de regel is nog steeds: kopiëren – plakken – **nadenken**.

Tip: voor de teamnaam kun je doen \$player->team->name.



Naam	Type	Team
Ludo Bagman	Chaser	German National Quidditch team
Argus Filch	Keeper	45 Quidditch Team
Marvolo Gaunt	Beater	Hufflepuff
Rubeus Hagrid	Beater	German National Quidditch team
Lee Jordan	Seeker	Hufflepuff
Viktor Krum	Chaser	45 Quidditch Team
Draco Malfoy	Seeker	45 Quidditch Team
Padma Patil	Keeper	45 Quidditch Team
Newt Scamander	Seeker	German National Quidditch team

7. Maken en aanpassen van items die een relatie hebben

Voor de één-kant van de relatie geldt: bij het maken of aanpassen van een item moet je kiezen bij welk bovenliggend item deze hoort. Meer concreet:

- Als je een ticket aanmaakt, moet je kiezen bij welke user deze hoort.
- Als je een product maakt, dan geef je aan bij welke categorie die hoort.
- Als je een speler maakt, moet je aangeven in welke team hij/zij zit.

Meestal gebruiken we daarvoor een `<select>` ('dropdown'). Dat betekent dat het create- en/of edit-formulier van een ticket moet weten welke users er zijn. De TicketController moet dit doorgeven bij het laden van de view van het formulier.

If-statement bij aanpassen

Bij het aanpassen van een ticket (bijvoorbeeld) komt nog iets om de hoek kijken: zodra je het edit-formulier opent, moet natuurlijk de huidige user al zijn geselecteerd in de dropdown. Je kunt dat bereiken met een if-statement in de `<option>`-tag, die voor de juiste gebruiker het attribuut "selected" zal echo'en.



Let goed op waar de tag begint en eindigt, dat is hier **geel** gemarkeerd. De if zit echt in de option-tag:

```
<select name="users">
  @foreach($users as $user)
    <option value="{{ $user->id }}"
      @if($user->id == $ticket->user_id) selected @endif
    >{{ $user->name }}</option>
  @endforeach
</select>
```

8. Opdracht: spelers aanmaken

Nu moeten we nog nieuwe spelers kunnen maken. We gaan het standaard-proces langs:

1. **Route:** maak een regel in de routes-file die zegt "een GET-request naar `/players/create` wordt afgehandeld door de create-methode van de PlayerController".
2. **Controller:** pas de create-methode van de PlayerController aan. Let op: bij het maken van een speler moet je het team selecteren. De controller gaat dus éérst alle teams ophalen. Die lijst wordt dan meegeven bij het inladen van de view `players/create`.

Tip: waarschijnlijk moet je bovenaan een use toevoegen, omdat je nu een model gaat gebruiken dat nog niet bekend was bij deze controller (model `Team` in de `PlayerController`).

3. **View:** maak de view `players/create` aan. Deze lijkt uiteraard erg op de andere create-views. Maar bedenk nog altijd: kopiëren – plakken – **nadenken**. Maak drie velden in je formulier:
 - Naam – vrije invoer.
 - Type – select met vier vaste options (chaser, keeper, beater, seeker).
 - Team – een select waarvan de options worden gebouwd met foreach (zie par. 7 als voorbeeld).

Het formulier verstuur je als POST-request naar `/players`. En dan begint de cyclus weer opnieuw:

4. **Route:** maak een regel in de routes-file die zegt "een POST-request naar `/players` wordt afgehandeld door de store-methode van de PlayerController".
5. **Controller:** pas de store-methode van de PlayerController aan. Maak een nieuwe Player maakt, zet de gegevens uit de request daarin en sla tenslotte de `$player` op.

Tip: je moet het veld `$player->team_id` instellen, maar waarschijnlijk heet die in de `$request` net anders. De naam in `$request` is precies gelijk aan de naam die je de `<select>` hebt gegeven.

6. **Response:** deze methode retourneert zelf geen view, maar een redirect naar `/players`.

7. **Link naar de pagina:** voeg tenslotte in de view `players/index` een link toe naar `/players/create`. Doe dit op dezelfde manier als op de toernooien- en teams-pagina's.

9. Opdracht: spelers aanpassen

Docent en klas bespreken eerst hoe dit hoofdstuk wordt afgemaakt. De weekcheck is een behoorlijke opdracht. Afhankelijk van de overgebleven tijd kiest men één van deze opties:

- Iedereen maakt opdracht 9, en begint daarna pas aan de weekcheck.
- Opdracht 9 is een bonusopdracht voor studenten die al zover zijn.
- Niemand maakt opdracht 9, start direct aan de weekcheck.

De opdracht

Tenslotte willen we een speler kunnen aanpassen. Veel stappen in deze opdracht lijken op het maken van een speler, daarom geven we de stappen alleen in hoofdlijnen:

1. **Routes**, maak twee regels in de routes-file:
 - a. GET-request naar `/player/{player}/edit` wordt afgehandeld door edit-methode in `PlayerController`.
 - b. PUT-request naar `/player/{player}` wordt afgehandeld door update-methode in `PlayerController`.
2. **Controller**, pas de twee methodes aan:
 - a. De edit-methode moet de view `players/edit` teruggeven. Deze view krijgt twee dingen mee: een lijst van alle teams én de huidige player die aangepast moet worden.
 - b. De update-methode verwerkt de aanpassing en redirect dan naar `/players`.
3. **Views**: bouw in de view `players/edit` een formulier om de speler aan te passen. Zie ook paragraaf 7 over het if-statement in een `<option>`-tag.

Herhalingsoefeningen - Deze oefeningen zijn niet verplicht, ze zijn bedoeld om extra te oefenen.

Dit hoofdstuk kent geen aparte herhalingsoefening. Bespreek op opdracht 9 verplicht is, anders kun je die gebruiken als extra oefening. Ook kent de weekcheck helemaal onderaan twee bonusopdrachten.

Weekcheck WDV-V-5: Categorieën in 4Shop

Deze weekcheck draait om de klantvraag “we willen graag producten kunnen indelen in categorieën, zoals *kleding*, *gadgets*, *overig*, *enzovoort*”. We gaan een deel van dat systeem bouwen in het admin-gedeelte. In een latere opdracht ga je de categorieën ook in openbare winkelpagina inbouwen.

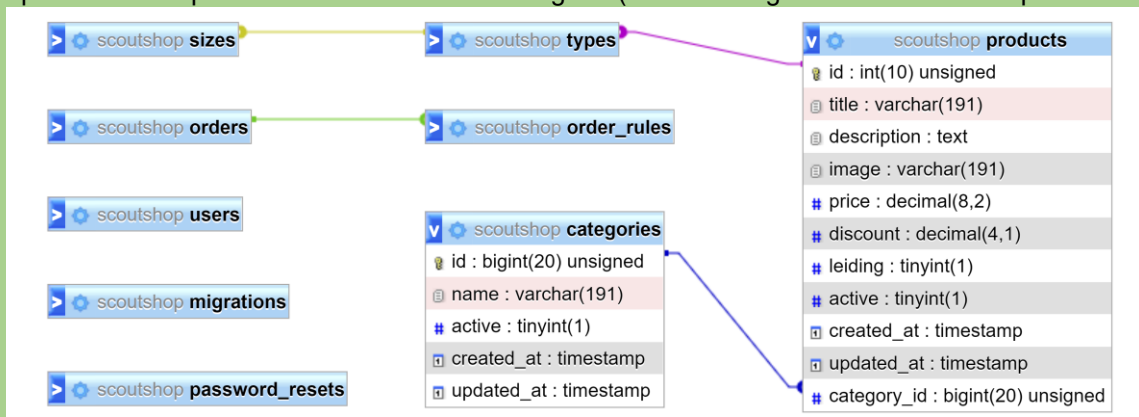
Kijk eerst naar deze korte demo van het eindresultaat: <https://youtu.be/ar8QuXBK7DM>.

1. Het admin-gedeelte is niet helemaal standaard qua opzet. Daarom heeft een meer ervaren collega alvast wat dingen voor je klaargezet. Je ontvangt die opzet in de vorm van een “patch-file”, hierdoor kun je met Git in één keer allerlei aanpassingen doen in je app.
 - a. Download de patch-file uit de weekcheck in Itslearning.
 - b. Zet ‘m in de hoofdmap van je 4Shop.
 - c. Open je terminal in diezelfde map, en run het commando:
`git apply categories.patch`

Deze patch heeft de volgende dingen gedaan:

- Past seeder aan om daar categorieën in te krijgen.
- Maakt Category-model en controller (let wel: in de admin-map).
- Past routes-file aan, je hebt nu alle nodige routes voor de categorieën.
- Maakt alvast een view in `admin/categories/index`.

2. Nu moet je de *categories*-tabel gaan opzetten én vanuit de *products*-tabel moeten we een relatie opzetten. Ieder product heeft immers een categorie (en een categorie heeft meerdere producten).



- a. Maak eerst een migration voor de *categories*-tabel. Negeer nog de relatie. De kolom `name` kun je aanmaken als string, de kolom `active` als boolean met default `true`.
- b. Nu moeten we de *products*-tabel aanpassen om hier de `category_id` in te voegen en de relatie te definiëren. Maar de *products*-tabel wordt gemaakt vóór de *categories* (dat gaat op volgorde van bestandsnaam). Daarom maken we een losse migration die de *products*-tabel gaat aanpassen nádat de *categories*-tabel is gemaakt. Het commando is: `php artisan make:migration add_category_id --table=products`
- c. Geef je nieuwe migration `add_category_id` de volgende inhoud:

```
Schema::table('products', function (Blueprint $table) {
    $table->bigInteger('category_id')->unsigned()->nullable();
    $table->foreign('category_id')
        ->references('id')->on('categories');
});
```

3. Test je werk tot nu toe:
 - a. Bouw je database opnieuw op met `php artisan migrate:fresh --seed`.
 - b. Ga naar <http://127.0.0.1:8000/admin/categories>.
 - c. Je zou nu een dump van de twee gemaakte categorieën moeten zien.

4. Leg de relatie ook vast in de models:

- Zie ook de docs van deze Laravel-versie, en het kopje onder deze link ("inverse"): <https://laravel.com/docs/5.8/eloquent-relationships#one-to-many>
- Category.php "has many" App\Product.
- Product.php "belongs to" App\Category.

5. We gaan het overzicht van categorieën maken. De route en controller zijn al aanwezig (dankzij de patch). Alleen de view moet worden aangepast. Open de view admin/categories/index.

Gebruik de bestaande view admin/products/index als voorbeeld, om dit resultaat te krijgen:

Categorieën

[Nieuwe categorie toevoegen](#)

Naam	
Kleding	Aanpassen
Overige	Aanpassen

Onthoud dat alle routes al zijn gemaakt door de patch. Als je een link wil naar de nieuwe-categorie-pagina kun je simpelweg gebruiken: `{{ route('admin.categories.create') }}`, enzovoort.

6. Dan het aanpassen van een categorie. Pas in de Admin/CategoryController eerst de edit()-methode aan zodat er een view wordt geretourneerd. Maak dan die view aan. Toon op de edit-pagina ook een lijst van producten in die categorie.

Categorie aanpassen

Naam

Producten

- Vest
- Polo
- Test
- Testproduct

- Je ziet hierboven een eenvoudige flexbox. Je moet dus wat CSS schrijven.
- Voor de producten kun je *lopen* over `$category->products`.

Pas nu ook de update()-methode van de Admin/CategoryController aan. Je mag de validatie even overslaan. Zorg dat de nieuwe naam wordt opgeslagen en *redirect* dan naar de route `admin.categories.index`.

7. Voeg in de lijst met producten (/admin/products) ook de categorie toe in de tabel. Maak een nieuwe `<th>` en verderop `<td>`. De categorie vind je via `$product->category->name`. Zie de video aan het begin voor een voorbeeld.
8. Bij het maken van een nieuw product willen we ook de categorie instellen. Hiervoor moet het create-formulier van een product eerst kennis hebben van alle categorieën.
- a) Open de Admin/ProductController en zoek de create()-methode.
 - b) Haal hier alle categorieën op en geef deze mee aan de view met `->with()`. Vergeet niet om bovenaan de controller `use App\Category;` te zetten, als je dat model wil gebruiken.
 - c) In de view admin/products/create beschik je nu over alle categorieën. Voeg een select toe om de categorie te kiezen. Gebruik een foreach om alle opties te echo'en:

```
<option value="{{ $category->id }}">{{ $category->name }}</option>
```
 - d) Ga terug naar de Admin/ProductController en zoek de store()-methode. Zorg dat de categorie ook wordt opgeslagen. Let op: je moet deze in `$product->category_id` zetten, maar het veld dat je uit de `$request` haalt heet wellicht anders.

Bonusopdracht 1: zorg dat je ook bij het aanpassen van een product de categorie kunt kiezen.

Bonusopdracht 2: zorg dat we ook nieuwe categorieën kunnen maken.

Inleveren

Maak de volgende screenshots en lever die in. Screenshot altijd je hele browserscherm en zorg dat je eigen naam in de <title>-tag staat. Pas hiertoe eventueel de view layouts/admin aan.

- In de browser: de edit-pagina van één categorie waarbij je rechts de producten ziet staan.
- Code: create() en store()-methodes van de Admin/ProductController (één screenshot).

6. Aggregates

Als eerste wordt in de klas aandacht besteed aan de vorige weekcheck.

1. Terugblik Querybuilder

We weten al dat we onze modellen kunnen gebruiken om informatie uit de database op te halen:

- Met `get()` of simpelweg `all()` kunnen we alle resultaten ophalen.
- Met een `where()` kun je deze resultaten filteren.
- Je kunt die resultaten sorteren en beperken met `orderBy()` en `limit()`.
- Met zoiets als `$category->products` kun je tenslotte alle gerelateerde models ophalen.

Geef voor de volgende vragen een stuk "Laravel-query" wat het juiste resultaat zal leveren:

1. Alle teams:

_____ `Team::all();` _____

2. Alle spelers van team 2:

3. Alle toernooien, meest recente eerst:

4. De drie meest recente toernooien:

2. Zoeken met LIKE

Een variatie op een normale where is een where waar een 'like' in voorkomt. Zo zoeken we niet op een exacte match, maar op een string die lijkt op iets wat we aangeven. Een voorbeeld:

```
Player::where('name', 'like', 'A%')->get();
```

Met deze query vinden we alle spelers wiens naam begint met een A. Het procent-teken (%) dient hier als "wildcard". Het betekent: 0 of meer karakters. In dit geval vragen we dus om alle spelers wiens naam is als een A met karakters erachter.

Er is nog een karakter wat we kunnen gebruiken als wildcard en dat is de underscore (_). De underscore is precies 1 karakter. Zo zouden we de volgende query kunnen doen:

```
Member::where('firstname', 'like', 'Mar_')->get();
```

In dit geval krijgen we alle members terug wiens voornaam Mark, Marc, Marq, Mars, Maro, etc. is. Let op dat we dus in dit geval niet Martijn zouden vinden, omdat de underscore maar één karakter is. Als we `Mar%` hadden gedaan had dat wel gewerkt.

Als je een team zoekt dat "iets met Zweinstein in de naam heeft" kun je ook aan beide kant een % plaatsen:

```
Team::where('name', 'like', '%zweinstein%')->get();
```

3. Aggregate functions

Een aggregate function is een functie die meerdere resultaten samenvoegt in één antwoord. Aggregate (Nederlands: aggregaat) betekent zoiets als samengevoegd of bijeengevoegd.

Veelvoorkomende *aggregate* functions zijn SUM, COUNT, MIN, MAX, AVG. Al deze functies tellen iets van het resultaat bij elkaar op tot één uiteindelijk resultaat. Zo kunnen we de AVG (average) functie bijvoorbeeld gebruiken om de gemiddelde leeftijd van een groep te bepalen. We hebben dan van meerdere resultaten de leeftijden *samengevoegd*.

Laravel beschikt over methodes die het voor ons makkelijk maken om deze aggregate functies te gebruiken. We bespreken een aantal voorbeelden:

```
$num_players_team = Player::where('team_id', '=', 2)->count();
```

Hoeveel spelers zijn er in team 2?

```
$average_price = Product::avg('price');
```

Wat is het gemiddelde van de kolom prijs van alle producten?

```
$highest_order_price = Order::max('total');
```

Wat is de hoogste waarde die voorkomt in de kolom total van alle orders?

```
$order_sum_total = Product::where('order_id', '=', 341)->sum('price');
```

Wat is de totaalprijs van alle producten in bestelling 341?

4. Opdracht: aanpassen zwerkbal-app

Nu je enkele aggregate functies kent kun je de twee getallen op de homepage dynamisch laten bepalen. Die staan nu als het goed is "hard-coded" op 5 en 26 in je HTML.

1. Vraag het aantal teams op en het aantal aankomende toernooien (toernooien waarvan de datum in de toekomst ligt) in de HomeController.
2. Geef deze variabelen door aan de view zoals je dat geleerd hebt met de with functie.
3. Haal de variabelen in de view op en vervang de hardcoded getallen met de berekende getallen.

5. Afronding module

Docent en klas bespreken de afronding van deze module. In de laatste les staat het feedbackmoment op het programma. Is daartoe nog voorbereiding nog? Werk eventueel ook nog aan de bonusopdracht hieronder.

Bonusopdracht: zoeken naar spelers

Deze opdracht biedt extra oefening, maar kan ook uitdagend zijn. Het is meer verdieping dan herhaling.

De lijst met spelers kan erg lang worden. Het zou mooi zijn om daar een zoekvakje boven te zetten. Aanwijzingen voor het ontwikkelen hiervan:

- In de view players/index krijg je bovenaan een formuliertje.
- Dit formulier verstuurt een GET-request naar players/search, die wordt afgehandeld door de search-methode van de PlayerController.
- Deze methode gebruikt een "where like"-query om alle spelers te zoeken waarbij de ingevoerde tekst in hun naam zit. De tekst moet overal in hun naam kunnen zitten.
- Tip: zorg eerst dat je gewoon de lijst met spelers dumpst op het scherm (return \$players vanuit je controller) om snel te kunnen werken.
- Tenslotte retourneert de search-methode gewoon de view players/index, maar met een andere lijst.

Feedbackmoment FC16-WEBw06: basistoets Laravel

Voor deze “toets” ga je de 4Shop helemaal afmaken. Je vindt bij het feedbackmoment op Itslearning het officiële programma van eisen van de 4Shop en/of je krijgt die op papier. Aan veel van de eisen heb je afgelopen weken al gewerkt, dus streep eerst alles door wat al aanwezig is in jouw applicatie.

Daarna houd je een todo-lijst over waar je t/m zondagavond van deze week aan kunt werken. Zorg dat je de les gebruikt om alle belangrijke vragen nog te stellen.

Het is niet de bedoeling om samen te werken aan deze opdracht. Doe je dat toch, dan krijg je feedback die niet op jou van toepassing is. Je docent kan een valse indruk krijgen van wat je wel / niet beheerst, en daardoor de verkeerde ondersteuning bieden.

Een laatste technische tip

Eén van de eisen is dat je de originele prijs van een item moet tonen. Maar als je opdrachten hebt gevolgd, zal `$product->price` altijd de prijs minus korting teruggeven. In het antwoord op deze vraag vind je de oplossing: <https://stackoverflow.com/questions/34181119/>.

Inleveren: automatische test

Let op: je opdracht wordt semi-automatisch nagekeken. Besteed aandacht aan het inleveren op de juiste manier! Als je niet de regels volgt bij inleveren, zal je opdracht niet worden nagekeken.

Zodra je klaar bent, moet je het commando `php artisan test:run` uitvoeren. Een script zal opstarten, dat een aantal automatische tests gaat uitvoeren op jouw code. Je ziet het resultaat hiervan in de terminal. Je kunt dit commando ook al eerder gebruiken, om te kijken of je goed op weg bent.

Het commando `test:run` genereert ook twee files in de hoofdmap: `test_output.txt` en `test_key.txt`. Die moet je uiteindelijk inleveren.

Inleveren: instructies

Let op: aan de inhoud en/of naam van deze `test_*` files mag je niets veranderen. Als je dat toch doet, dan kan je applicatie niet worden nagekeken. Bovendien zijn er verschillende zichtbare en onzichtbare checks ingebouwd, om te controleren of je de juiste files inlevert (en ook je eigen files). Verander je iets, dan breken die checks en stopt het nakijkproces.

Algemene tip: verwijder de `vendor`-map voor je gaat zippen. Deze bevat alle packages die je met composer hebt geïnstalleerd. Die zijn niet nodig bij het nakijken, maar nemen wel véél ruimte in.

Je moet exact drie dingen inleveren:

- ✓ Een zip van je code met de naam `repo.zip`.
- ✓ De losse file `test_output.txt` (dus niet in de zip)
- ✓ De losse file `test_key.txt` (dus niet in de zip)

Voldoe je niet exact aan deze voorwaarden, dan kunnen we niet nakijken. Wat moet je dus niet doen?

- ✗ Een rar-bestand inleveren.
- ✗ De `test_output` en `test_key` in je zip-file stoppen.
- ✗ De naam veranderen zoals `test_output_StudentNaam.txt`.
- ✗ Je hoeft je naam niet op te nemen in de bestandsnaam, omdat Itslearning al weet wie je bent.

III. Feedback-momenten

Feedback-moment FC12-WEBw03

Week: 3
Vorm: Theoretische toets
Vorbereiden: Hoofdstukken 1 t/m 3 goed doornemen.
Meer info: Je krijgt theorievragen over de behandelde stof t/m hoofdstuk 3.

Feedback-moment FC16-WEBw06

Week: 6
Vorm: Inleveropdracht
Vorbereiden: Zorg dat je alle opdrachten in de les hebt gemaakt.
Meer info: Het doel van dit feedbackmoment is: je hebt de 'scoutshop' aangepast zodat deze exact voldoet aan het gegeven programma van eisen.

Leeruitkomsten

Aspect	Voldaan
13.03	Vanuit een gegeven ERD of databaseontwerp kun je een database daadwerkelijk opbouwen.
ERD lezen	Je kunt een eenvoudig ERD aflezen en gebruiken om de database te bouwen.
WDV.19	Je zet models en migrations op om met de database te werken.
Begrip migrations	Je begrijpt dat je niet meer direct de database aanpast, maar daarvoor migrations gebruikt, zodat je de databasestructuur ook kunt committen en delen via git.
Migrations	Je schrijft migrations en voert deze uit om de database te bouwen.
DBS.07	Je begrijpt de principes van een relationele database.
Primary key	Elke tabel die je aanmaakt heeft een primary key. Deze heet id en is een auto incrementing integer.
Foreign key	Je kan met behulp van een foreign key een link leggen tussen twee tabellen in een database.
Relaties en ERD	Je snapt hoe een relationele database in elkaar zit en kan deze structuur afleiden uit een ERD.
DBS.08	Je kunt een <i>select</i> -query gebruiken met een LIKE-statement.
Zoeken	Met behulp van een WHERE ... LIKE kun je een zoek-query maken, die alle rijen teruggeeft waarin de zoekterm voorkomt.
DBS.09	Je kunt functies gebruiken in je query ('aggregates').
MIN / MAX	Je gebruikt de MIN- of MAX-functie om het grootste of kleinste getal uit een kolom te halen. Eventueel in combinatie met WHERE.
AVG / SUM	Je gebruikt de AVG of SUM-functie om het gemiddelde of totaal te berekenen van een kolom. Eventueel in combinatie met WHERE.
DBS.10	Je kunt LIMIT gebruiken (bijvoorbeeld i.c.m. ORDER BY om een ranking te presenteren).
LIMIT	Je voegt LIMIT toe aan een query om een exact aantal rijen te selecteren.
Ranking	In combinatie met ORDER BY kun je een top X of ranking selecteren.
WDV.16	Je kunt werken met een package-manager om externe frameworks en/of packages te installeren en updaten.
Composer	Je hebt composer geïnstalleerd, je kunt met composer <i>packages</i> installeren.

WDV.17	Je kunt werken met een MVC-framework, je volgt de bijbehorende conventies en je kunt de documentatie lezen.	
Begrip MVC	Je hebt enige kennis van MVC, je weet hoe zo'n framework werkt.	
Laravel	Je gebruikt het Laravel-framework.	
Volgen conventies	Je werkt op de aanbevolen manier (mappenstructuur, conventies, etc.)	
Gebruik docs	Je zoekt waar nodig de juiste werkwijze op in de docs.	
Ontwikkelomgeving	Je kunt je lokale ontwikkelomgeving opzetten (.env en <i>php artisan serve</i>)	
WDV.18	Je zet <i>routes</i> op om de link te leggen tussen URL's en de bestanden/mappenstructuur van het framework.	
Begrip routes-file	Je begrijpt dat er bij een framework geen directe link meer is tussen de mappenstructuur en de URL's. Die link leg je handmatig in de routes-file.	
WDV.20	Je zet views op om data te presenteren.	
Conventies	De views staan op de juiste plek en hebben logische namen.	
Blade	Je gebruikt blade-directives (@...) om je views op te bouwen.	
Lay-outs	Met @section bouw je één of meer template-views, met @extends gebruik je die in alle andere views (dus niet iedere view is een volledige html-pagina).	
WDV.13	Je schrijft valide HTML en de applicatie heeft een professionele en zakelijke uitstraling door de inzet van technieken als flexbox en grid.	
HTML / CSS	Zowel CSS als HTML zijn valide, correct en opgeruimd.	
Uitstraling	De applicatie heeft een eenvoudige maar zakelijke uitstraling	
Flexbox	Waar nodig / zinnig pas je CSS 'flexbox' toe. Je kent de basis van de techniek.	
Grid	Waar nodig / zinnig pas je CSS 'grid' toe. Je kent de basis van de techniek.	
WDV.09	Je kunt data zowel op de front-end als de back-end valideren.	
Front-end	Je kunt input's etc. voorzien van attributen om een eerste validatie te doen.	
Backend-end	In de controller-acties voeg je validatiecode toe. Als validatie niet slaagt, stuur je de gebruiker terug naar het formulier met een errormelding.	

En daarnaast wordt aandacht besteed aan de leeruitkomsten van het V-model:

Aspect	Voldaan
13.05	Tijdens het programmeren gebruik je de gestelde <u>conventies</u> .
13.08	Je kunt zelfstandig je code <u>debuggen</u> ; je leest foutmeldingen nauwkeurig en je gaat systematisch te werk om een fout op te sporen. Waar nodig schakel je hulp in.
13.09	Je kent de <u>documentatie</u> van de programmeertaal waar je mee werkt en je kunt deze raadplegen; je leest nauwkeurig en analytisch. Eventueel gebruik je ook de bijbehorende community om antwoorden te zoeken of vragen te stellen.

IV. Voor de docent

Oppervlakkige kennis

Het idee van deze module is expliciet *niet* dat studenten al een heel diep begrip hebben van Laravel. Hierna volgt nog LAR-I waarin het gaat over het opzetten van een applicatie *from scratch*. Deze module is slechts de introductie.

Inleveropdrachten

De weekchecks zijn zo geschreven dat studenten meestal precies één screenshot inleveren. Daardoor kun je heel vlug door de opdrachten heen klikken in Itslearning. Eventueel geschikt om zo 'whole class feedback' te verzamelen.

Het maakt wel dat studenten niet altijd ál hun werk inleveren. Maak eventueel aan de groep duidelijk dat het inleveren echt maar een check is. Het maken van de volledige opdrachten doen ze uiteindelijk als oefening voor de toets en hun stage.

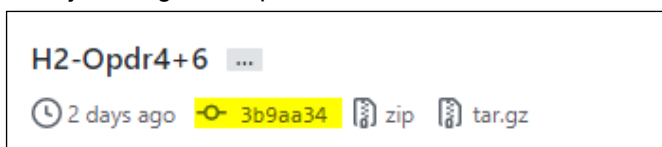
Nakijken feedbackmoment

Bij het laatste feedbackmoment leveren studenten het resultaat van hun unittests in. In de docentenmap op Itslearning staat een tool om deze te controleren.

Uitwerkingen

- Zwerkbal: https://github.com/curio-lesmateriaal/4S_Zwerkbal_uitwerking/tags
- 4Shop: https://github.com/curio-lesmateriaal/4Shop_uitwerking/tags

Klik bij een tag even op de commit-hash om snel de aanpassingen te zien:



V. Kwaliteit en verbeteracties

Versie	Aanpassingen voor opstart	Uitvoer	Punten na afloop
1.*	Nvt – eerste uitvoer	2021-nj	

curric