Laravel 1 | Basis | Nog in ontwikkeling

Laravel CRUD - Game Collection

Module Laravel 1: Bouw je eigen Game Collection app

Leerdoelen

- Je kunt een Laravel-project installeren en lokaal starten.
- Je begrijpt hoe Laravel werkt met MVC: Model, View, Controller en Routes.
- Je kunt een database koppelen aan Laravel met een migratie en model.
- Je kunt CRUD-functionaliteit bouwen: Create, Read, Update en Delete.
- Je kunt Blade templates gebruiken om herhaling in HTML te voorkomen.
- Je kunt Git en GitHub gebruiken om je code netjes bij te houden.

Uitleg

In deze module bouw je stap voor stap een web-app waarmee je je game-collectie kunt beheren. De app heet **Game Collection**. Je kunt games toevoegen, bekijken, aanpassen en verwijderen.

Je gebruikt hiervoor **Laravel**, een PHP-framework. Een framework is een verzameling afspraken, mappen en hulpmiddelen waardoor je sneller en netter een webapplicatie kunt bouwen.

Daarnaast gebruik je **Git** en **GitHub**. Git houdt wijzigingen in je code bij. GitHub is de plek waar je je project online opslaat.

Wat moet je al kunnen?

- Basis HTML, CSS en PHP.
- Basiskennis van MVC, bijvoorbeeld uit eerdere Yii-lessen.
- Werken met VS Code.

- XAMPP starten en phpMyAdmin openen.

Les 1 - Laravel installeren

Module 1.1: Installatie, Composer en Git

🎯 Leerdoelen

- Je kunt uitleggen wat Composer doet.
- Je kunt Composer installeren en controleren.
- Je kunt een nieuw Laravel-project aanmaken.
- Je kunt de Laravel development server starten.
- Je kunt een Git repository initialiseren en publiceren naar GitHub.

📖 Uitleg

Je gaat een app bouwen waarmee je je game-collectie kunt bijhouden. Je kunt games toevoegen, bekijken, bewerken en verwijderen.

Laravel wordt geïnstalleerd via **Composer**. Composer is de package manager voor PHP. Een package manager downloadt automatisch alle bestanden (codebibliotheken) die jouw project nodig heeft.

Vergelijk het met een boodschappenlijst voor je project: Laravel zegt welke onderdelen nodig zijn, Composer haalt ze op en zet ze in de juiste map.

> Jij typt command Composer haalt packages op Laravel nieuw project composer create-project laravel/laravel games
> Composer downloadt Laravel en alle benodigde PHP-packages voor je project.

💻 Startercode

Controleer eerst of Composer al is geïnstalleerd. Open de terminal in VS Code en typ:

```
composer --version
```

Zie je een versienummer? Dan is Composer geïnstalleerd. Krijg je een foutmelding zoals `composer is not recognized`? Installeer Composer dan eerst via de officiële downloadpagina.

Download Composer hier:

https://getcomposer.org/download/

Gebruik op Windows meestal de **Composer-Setup.exe**. Laat de installer PHP zoeken in je XAMPP-map. Dat is meestal:

```
C:\xampp\php\php.exe
```

Sluit na de installatie VS Code en open VS Code opnieuw. Controleer daarna opnieuw:

```
composer --version
```

□□ Opdracht - Nieuw Laravel-project maken

1. Open VS Code.
2. Open een terminal.
3. Ga naar de map waarin je schoolprojecten staan.
4. Maak een nieuw Laravel-project aan:

```
composer create-project laravel/laravel games
cd games
code .
```

Start daarna de development server:

```
php artisan serve
```

Open in je browser:

```
http://localhost:8000
```

Als alles goed staat, zie je de Laravel-welkomstpagina.

□□ Uitleg

`php artisan serve` start een kleine lokale server. Die server is alleen bedoeld om tijdens het ontwikkelen je Laravel-app te testen.

Je stopt de server met **Ctrl + C** in de terminal.

□□ Opdracht - Git opzetten

1. Open in VS Code het tabblad **Source Control**.
2. Klik op **Initialize Repository**.
3. Stage alle bestanden met de plusknop bij **Changes**.
4. Gebruik als commit message: **Initieel Laravel project**.
5. Klik op **Commit**.
6. Klik op **Publish Branch**.
7. Kies **Publish to GitHub Public Repository**.

Let op: Laravel maakt automatisch een `.gitignore` bestand aan. Hierin staat welke bestanden Git **niet** moet uploaden, bijvoorbeeld tijdelijke bestanden of geheime instellingen zoals wachtwoorden etc.

Inleveren

1. Screenshot van de Laravel-welkomstpagina met de URL zichtbaar.
2. Screenshot van je GitHub repository.
3. Een korte uitleg in eigen woorden: wat doet `.gitignore`?

Les 2 - MVC in Laravel

Module 1.2: Model, View, Controller en Router

Leerdoelen

- Je kunt uitleggen wat MVC betekent.
- Je kunt benoemen waar Models, Views, Controllers en Routes staan in Laravel.
- Je begrijpt hoe een browserverzoek door Laravel heen loopt.

Uitleg

Laravel werkt volgens het **MVC-patroon**. MVC staat voor **Model, View, Controller**. Laravel gebruikt daarnaast ook een **Router**.

- **Model:** praat met de **database**. Voorbeeld: `Game.php`.

- **View:** bepaalt wat de gebruiker **ziet**. Voorbeeld: `index.blade.php`. Hierin staat ook de html / css, etc.
- **Controller:** haalt data op en stuurt die naar een view. Voorbeeld: `GameController.php`.
- **Router:** koppelt een URL aan een controllerfunctie. Voorbeeld: `routes/web.php`.

PLAATJE!!

bron_afbeelding_4.png

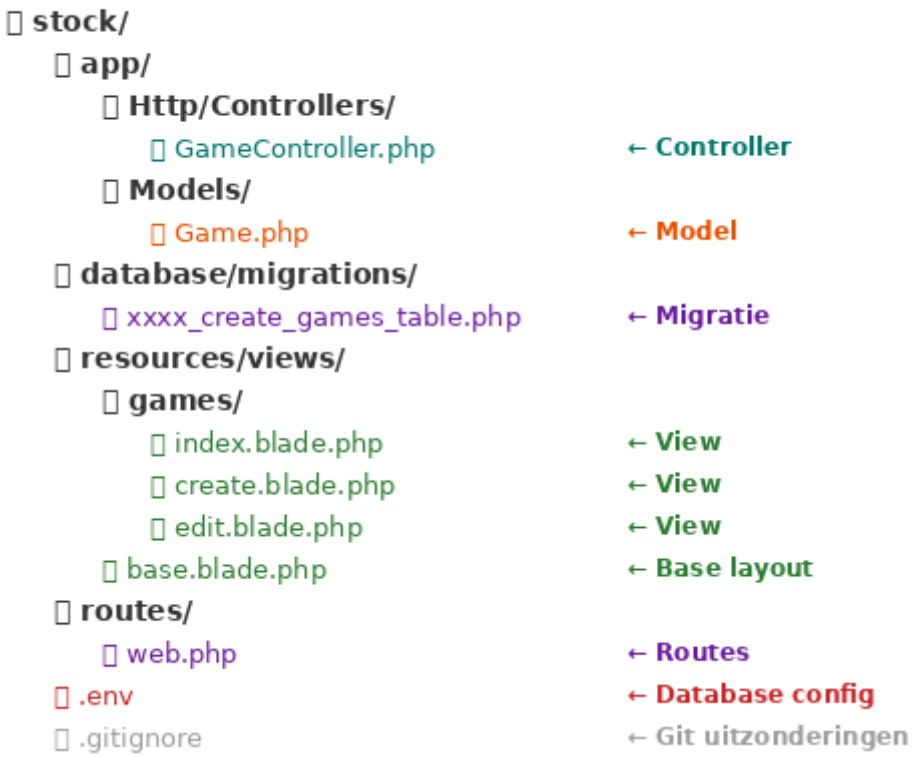
Uitleg

Een voorbeeld: de gebruiker gaat naar `/games`. Laravel zoekt in `routes/web.php` welke controllerfunctie bij die URL hoort. De controller haalt via het model games op uit de database en stuurt die naar een Blade-view. De view maakt daar HTML van voor de browser.

Startercode

```
routes/web.php                  // routes
app/Http/Controllers            // controllers
app/Models                      // models
resources/views                 // views
database/migrations             // database-aanpassingen
.env                            // lokale instellingen
```

Laravel projectstructuur



Opdracht - MVC herkennen

1. Open je Laravel-project in VS Code.
2. Zoek de mappen `routes`, `app/Models`, `app/Http/Controllers` en `resources/views`.
3. Maak in je eigen woorden een korte beschrijving van elk onderdeel.

Inleveren

1. Screenshot van je projectstructuur in VS Code.
2. Korte uitleg per onderdeel: Model, View, Controller en Router.

Les 3 - Database & migraties

Module 1.3: Database koppelen aan Laravel

Leerdoelen

- Je kunt een database maken in phpMyAdmin.

- Je kunt het `.env` bestand instellen voor MySQL.
- Je kunt een model met migratie maken.
- Je kunt een migratie uitvoeren.
- Je kunt uitleggen wat `$fillable` doet.

□□ Uitleg

Een database bewaart de gegevens van je app. In deze les maak je een database met de naam **games**. Laravel gebruikt een **migratie** om tabellen aan te maken.

Een migratie is een PHP-bestand waarin staat hoe een databasetabel eruit moet zien. Je kunt het zien als een bouwtekening voor je tabel.

□□ Opdracht - Database aanmaken

1. Start XAMPP.
2. Start **Apache** en **MySQL**.
3. Open phpMyAdmin:

```
http://localhost/phpmyadmin
```

4. Maak een nieuwe database met de naam **games**.

□□ Startercode

Open het bestand `.env` en pas de database-instellingen aan:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=games
DB_USERNAME=root
DB_PASSWORD=
```

□□ **Let op:** Bij een standaard XAMPP-installatie is de MySQL-gebruiker vaak `root` zonder wachtwoord.

⚠ **Let op:** Het `.env` bestand staat in `.gitignore` en wordt niet gecommit. Het kan wachtwoorden bevatten. Dus omdat het in het bestand .gitignore staat, wordt het niet op Github geplaatst.

Opdracht - Model en migratie maken

Voer dit commando uit in de terminal:

```
php artisan make:model Game --migration
```

Dit maakt twee bestanden:

- `app/Models/Game.php`
- een migratiebestand in `database/migrations`

Startercode

Open de migratie en pas de `up()` functie aan:

```
public function up()
{
    Schema::create('games', function (Blueprint $table) {
        $table->increments('id');
        $table->string('game_name');
        $table->string('platform');
        $table->string('genre');
        $table->decimal('rating', 3, 1);
        $table->timestamps();
    });
}
```

Voer daarna de migratie uit via je terminal. De tabel zoals die in het migratiebestand staat wordt aangemaakt in de database 'games'. De eerste keer dat je dit commando uitvoert, worden ook de standaard Laravel tabellen aangemaakt, zoals die van user.

```
php artisan migrate
```

Controleer in phpMyAdmin of de tabel **games** bestaat.

Startercode

Open `app/Models/Game.php` en zet dit erin:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Game extends Model
{
    use HasFactory;

    protected $fillable = [
        'game_name',
        'platform',
        'genre',
        'rating'
    ];

    protected $table = 'games';
}
```

`$fillable` bepaalt welke velden veilig via een formulier ingevuld mogen worden. `$table` koppelt dit model aan de tabel `games`.

Opdracht - Git commit

1. Open Source Control in VS Code.
2. Stage alle wijzigingen.
3. Commit met de message: **Database migratie en Game model**.
4. Klik op **Sync Changes**.

Inleveren

1. Screenshot van phpMyAdmin met de structuur van de tabel **games**.
2. Screenshot van je terminal na `php artisan migrate`.
3. Screenshot van je GitHub repository na de commit.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css">
    <title>Game Collection</title>
</head>
<body>
    <div class="container" style="margin:40px;">
        <h1 class="display-4">□□Game Collection</h1>
        <table class="table">
            <thead class="thead-dark">
                <tr>
                    <th>ID</th>
                    <th>Game</th>
                    <th>Platform</th>
                    <th>Rating</th>
                </tr>
            </thead>
            <tbody>
                @foreach($games as $game) // in deze loop worden alle rijen (records) gemaakt
die in de database zijn gevonden.
                    <tr>
                        <td>{{ $game->id }}</td>
                        <td>{{ $game->game_name }}</td>
                        <td>{{ $game->platform }}</td>
                        <td>{{ $game->rating }}/10</td>
                    </tr>
                @endforeach
            </tbody>
        </table>
    </div>
</body>
</html>
```

Les 4 - Read: games tonen

Module 1.4: Data uit de database tonen

□□ Leerdoelen

- Je kunt een resource controller maken.
- Je kunt data ophalen met een model.
- Je kunt data doorgeven aan een Blade-view.
- Je kunt een route maken naar een controllerfunctie.

□□ Uitleg

Read betekent: gegevens bekijken. In deze les toon je alle games uit de database in een tabel.

□□ Opdracht - Controller aanmaken

```
php artisan make:controller GameController --resource
```

Open de controller (in de map \app\http\controllers) en voeg bovenaan toe:

```
use App\Models\Game;
```

Pas daarna de functie `index()` aan:

```
public function index()
{
    $games = Game::all();
    return view('games.index', compact('games'));
}
```

□□ Startercode

Maak de map `resources/views/games` en daarin het bestand `index.blade.php`:

`@foreach` en `@endforeach` vormen een loop in Blade. `{{ }}` drukt een variabele af en beveiligt automatisch tegen ongewenste HTML.

□□ Startercode

Voeg deze route toe in `routes/web.php`:

```
Route::get('games', [App\Http\Controllers\GameController::class, 'index']);
```

Opdracht - Testdata toevoegen

Voer deze SQL uit in phpMyAdmin via het tabblad **SQL**:

```
INSERT INTO `games` (`id`, `game_name`, `platform`, `genre`, `rating`, `created_at`,
`updated_at`) VALUES
(1, 'The Legend of Zelda: TotK', 'Switch', 'Adventure', 9.5, NULL, NULL),
(2, 'Elden Ring', 'PS5', 'RPG', 9.0, NULL, NULL),
(3, 'Minecraft', 'PC', 'Sandbox', 8.5, NULL, NULL),
(4, 'FIFA 25', 'PS5', 'Sports', 7.0, NULL, NULL),
(5, 'GTA V', 'PC', 'Action', 9.5, NULL, NULL);
```

Ga daarna naar:

```
http://localhost:8000/games
```

Opdracht - Git commit

1. Stage alle wijzigingen.
2. Commit met de message: **Read: controller, view en route**.
3. Klik op **Sync Changes**.

Opdracht - Oeps!! We zijn 'genre' vergeten in de view.

1. Voeg deze toe in het kopje van de tabel en als veld in de rij.
2. Daarna nogmaals comitten met een relevante message en syncen naar Github.

Inleveren

1. Screenshot van `localhost:8000/games` met de URL zichtbaar.
2. Screenshot van je controllerfunctie `index()`.
3. Screenshot van je GitHub repository na de commit.

Les 5 - Create: game toevoegen

Module 1.5: Formulier maken en data opslaan

Leerdoelen

- Je kunt een formulier maken in Blade.
- Je kunt formulierdata opslaan in de database.
- Je kunt server-side validatie toepassen.
- Je begrijpt waarom `@csrf` nodig is.

Uitleg

Create betekent: nieuwe gegevens toevoegen. Je maakt een formulier waarmee de gebruiker een nieuwe game kan opslaan.

Een formulier verstuurt data met `POST`. Laravel gebruikt `@csrf` om te controleren of het formulier echt van jouw website komt.

@csrf voegt in Laravel een beveiligingstoken toe aan een formulier, zodat Laravel kan controleren dat het verzoek echt van jouw website komt en niet van een kwaadwillende externe site.

Startercode

Voeg deze routes toe aan `routes/web.php`:

```
Route::get('games/create', [App\Http\Controllers\GameController::class, 'create']);
Route::post('games/store', [App\Http\Controllers\GameController::class, 'store']);
```

Voeg deze functies toe aan `GameController`:

```
public function create()
{
    return view('games.create');
}

public function store(Request $request)
{
    $request->validate([
        'game_name' => 'required',
        'platform' => 'required',
        'genre' => 'required',
        'rating' => 'required|numeric|min:0|max:10'
    ]);
```

```
    $game = new Game([
        'game_name' => $request->get('game_name'),
        'platform' => $request->get('platform'),
        'genre' => $request->get('genre'),
        'rating' => $request->get('rating')
    ]);

    $game->save();

    return redirect('/games')->with('success', 'Game added!');
}
```

Voeg bovenaan de controller toe als dat nog niet bestaat:

```
use Illuminate\Http\Request;
```

□□ Startercode

Voeg in `index.blade.php` boven de tabel deze knop toe:

```
<a href="/games/create" class="btn btn-success mb-3">□□Add Game</a>
```

□□ Startercode

Maak `resources/views/games/create.blade.php`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css">
    <title>Add Game</title>
</head>
<body>
    <div class="container" style="margin:40px;">
        <h1 class="display-4">□□Add Game</h1>
        <form method="post" action="/games/store">
            @csrf
```

```
            <div class="form-group">
                <label>Game Name *</label>
                <input type="text" class="form-control" name="game_name" />
            </div>
            <div class="form-group">
                <label>Platform *</label>
                <input type="text" class="form-control" name="platform" placeholder="PS5,
Xbox, PC, Switch..." />
            </div>
            <div class="form-group">
                <label>Genre *</label>
                <input type="text" class="form-control" name="genre" placeholder="Action, RPG,
Sports..." />
            </div>
            <div class="form-group">
                <label>Rating (0-10)</label>
                <input type="number" step="0.1" min="0" max="10" class="form-control"
name="rating" />
            </div>
            <button type="submit" class="btn btn-success">Add Game</button>
        </form>
    </div>
</body>
</html>
```

□□ Opdracht - Game toevoegen

1. Start de Laravel server.
2. Open `http://localhost:8000/games/create`.
3. Vul het formulier in.
4. Klik op **Add Game**.
5. Controleer of de nieuwe game in het overzicht staat.

□□ Opdracht - Git commit

1. Stage alle wijzigingen.
2. Commit met de message: **Create: formulier en store**.

3. Klik op **Sync Changes**.

Inleveren

1. Screenshot van `localhost:8000/games/create` met de URL zichtbaar.
2. Screenshot van het overzicht met de nieuw toegevoegde game.
3. Screenshot van je controllerfunctie `store()`.
4. Leg in eigen woorden uit waarom je om een game toe te voegen een route nodig hebt. Lever dit in via een PDF.

Les 6 - Update: game bewerken

Module 1.6: Bestaande data aanpassen

Leerdoelen

- Je kunt een edit-knop toevoegen aan een overzicht.
- Je kunt bestaande data ophalen met een id.
- Je kunt een formulier vullen met bestaande waarden.
- Je kunt gewijzigde data opslaan in de database.

Uitleg

Update betekent: bestaande gegevens aanpassen. Je maakt een edit-formulier waarin de huidige gegevens van een game al ingevuld zijn.

Startercode

Voeg in de tabel van `index.blade.php` een kolom **Edit** toe:

```
<th>Edit</th>
```

Voeg in de loop deze knop toe:

```
<td>
    <a href="/games/edit/{{ $game->id }}" class="btn btn-primary btn-sm">Edit</a>
```

```
</td>
```

Startercode

Voeg deze routes toe aan `routes/web.php`:

```
Route::get('games/edit/{id}', [App\Http\Controllers\GameController::class, 'edit']);
Route::post('games/update/{id}', [App\Http\Controllers\GameController::class, 'update']);
```

Voeg deze functies toe aan de controller:

```
public function edit($id)
{
    $game = Game::find($id);
    return view('games.edit', ['game' => $game]);
}

public function update(Request $request, $id)
{
    $request->validate([
        'game_name' => 'required',
        'platform' => 'required',
        'genre' => 'required',
        'rating' => 'required|numeric|min:0|max:10'
    ]);

    $game = Game::find($id);
    $game->game_name = $request->get('game_name');
    $game->platform = $request->get('platform');
    $game->genre = $request->get('genre');
    $game->rating = $request->get('rating');
    $game->save();

    return redirect('/games');
}
```

Startercode

Maak `resources/views/games/edit.blade.php`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css">
    <title>Edit Game</title>
</head>
<body>
    <div class="container" style="margin:40px;">
        <h1 class="display-4">✏ Edit Game</h1>
        <form method="post" action="/games/update/{{ $game->id }}">
            @csrf
            <div class="form-group">
                <label>Game Name *</label>
                <input type="text" class="form-control" name="game_name" value="{{ $game-
>game_name }}" />
            </div>
            <div class="form-group">
                <label>Platform *</label>
                <input type="text" class="form-control" name="platform" value="{{ $game-
>platform }}" />
            </div>
            <div class="form-group">
                <label>Genre *</label>
                <input type="text" class="form-control" name="genre" value="{{ $game->genre
}}" />
            </div>
            <div class="form-group">
                <label>Rating (0-10)</label>
                <input type="number" step="0.1" min="0" max="10" class="form-control"
name="rating" value="{{ $game->rating }}" />
            </div>
            <button type="submit" class="btn btn-primary">Update</button>
        </form>
    </div>
</body>
</html>
```

Het verschil met create: de velden zijn al ingevuld via `value="{{ $game->... }}"`.

□□ Opdracht - Game bewerken

1. Open het overzicht.
2. Klik bij een game op **Edit**.
3. Pas bijvoorbeeld de rating aan.
4. Klik op **Update**.
5. Controleer of de wijziging zichtbaar is in het overzicht.

□□ Opdracht - Git commit

1. Stage alle wijzigingen.
2. Commit met de message: **Update: edit form en update functie**.
3. Klik op **Sync Changes**.

□□ Inleveren

1. Screenshot van `localhost:8000/games/edit/1` met de URL zichtbaar.
2. Screenshot van het overzicht na het aanpassen van een game.
3. Screenshot van je controllerfunctie `update()`.
4. Leg in eigen woorden uit welke stappen Laravel neemt bij het bewerken van een game. Dus op volgorde. (url -> route -> etc. etc.) Lever dit in als pdf.

Les 7 - Delete: game verwijderen

Module 1.7: Data veilig verwijderen

□□ Leerdoelen

- Je kunt een delete-knop toevoegen aan een overzicht.
- Je kunt een record verwijderen uit de database.
- Je begrijpt waarom verwijderen via een formulier veiliger is dan via een link.

Uitleg

Delete betekent: gegevens verwijderen. Omdat verwijderen een gevaarlijke actie is, doe je dit niet via een gewone link. Een link gebruikt meestal `GET`. Verwijderen moet via `POST` met een `@csrf` token.

Daarom maken we een formulier met een knop. De gebruiker krijgt ook eerst een bevestigingsvraag.

Startercode

Voeg in de tabel van `index.blade.php` een kolom **Delete** toe:

```
<th>Delete</th>
```

Voeg in de loop deze delete-knop toe:

```
<td>
    <form action="/games/destroy/{{ $game->id }}" method="post">
        @csrf
        <button onclick="return confirm('Weet je het zeker?')" class="btn btn-danger btn-sm"
type="submit">Delete</button>
    </form>
</td>
```

Startercode

Voeg deze route toe aan `routes/web.php`:

```
Route::post('games/destroy/{id}', [App\Http\Controllers\GameController::class, 'destroy']);
```

Voeg deze functie toe aan de controller:

```
public function destroy($id)
{
    $game = Game::find($id);
    $game->delete();
    return redirect('/games');
}
```

Opdracht - Game verwijderen

1. Open het overzicht.
2. Klik bij een game op **Delete**.
3. Bevestig de vraag.
4. Controleer of de game verdwenen is.

□□ Opdracht - Git commit

1. Stage alle wijzigingen.
2. Commit met de message: **Delete functionaliteit**.
3. Klik op **Sync Changes**.

□□ Inleveren

1. Screenshot van het overzicht met Edit- en Delete-knoppen en de URL zichtbaar.
2. Screenshot van het overzicht nadat je een game hebt verwijderd.
3. Screenshot van je controllerfunctie `destroy()`.

Les 8 - Blade templates optimaliseren

Module 1.8: Herhaling verminderen met een base template

□□ Leerdoelen

- Je kunt uitleggen waarom dubbele HTML-code onhandig is.
- Je kunt een base template maken met `@yield`.
- Je kunt views laten erven van een template met `@extends`.
- Je kunt een gemiddelde rating berekenen in een Blade-view.

□□ Uitleg

Je hebt nu meerdere views met dezelfde HTML-structuur. Als je bijvoorbeeld Bootstrap of een menu wilt aanpassen, moet dat in meerdere bestanden. Dat is foutgevoelig.

De oplossing is een **base template**. Daarin zet je de gedeelde HTML. De losse pagina's vullen alleen hun eigen titel en inhoud in.

□□ Startercode

Maak `resources/views/base.blade.php`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css">
    <title>Game Collection</title>
</head>
<body>
    <div class="container" style="margin:40px;">
        <h1 class="display-4">@yield('title')</h1>
        @yield('content')
    </div>
</body>
</html>
```

`@yield('content')` is een plekhouder. Child-views vullen die plekhouder later in.

□□ Startercode

Pas `index.blade.php` aan:

```
@extends('base')

@section('title', '□□Game Collection')

@section('content')
    <a href="/games/create" class="btn btn-success mb-3">Add Game</a>

    <table class="table">
        <!-- tabelinhoud blijft hetzelfde -->
    </table>
@endsection
```

Pas `create.blade.php` en `edit.blade.php` op dezelfde manier aan:

- `index.blade.php`: □□ Game Collection
- `create.blade.php`: □□ Add Game
- `edit.blade.php`: ✏ Edit Game

□□ Startercode

Voeg onderaan de tabel een gemiddelde rating toe. Zet vóór de loop:

```
@php( $sum = 0 )
```

Zet in de loop:

```
@php( $sum += $game->rating )
```

Zet na de loop:

```
<tr>
    <td colspan="4"><strong>Gemiddelde rating:</strong></td>
    <td><strong>{{ count($games) > 0 ? number_format($sum / count($games), 1) : 0
}}/10</strong></td>
    <td></td>
</tr>
```

□□ Opdracht - Views optimaliseren

1. Maak `base.blade.php`.
2. Pas `index.blade.php` aan met `@extends` en `@section`.
3. Pas `create.blade.php` aan met `@extends` en `@section`.
4. Pas `edit.blade.php` aan met `@extends` en `@section`.
5. Voeg de gemiddelde rating toe aan de indexpagina.

□□ Opdracht - Git commit

1. Stage alle wijzigingen.

2. Commit met de message: **Blade templates geoptimaliseerd**.
3. Klik op **Sync Changes**.

□□ Inleveren

1. Screenshot van de indexpagina met gemiddelde rating en URL zichtbaar.
2. Screenshot van de createpagina.
3. Screenshot van de editpagina.
4. Het bestand `base.blade.php`.
5. Het bestand `create.blade.php` in de `@extends`-versie.
6. Screenshot van je GitHub repository met je commits.

Les 9 - Mini Challenge

Module 1.9: Zelf een show-pagina maken

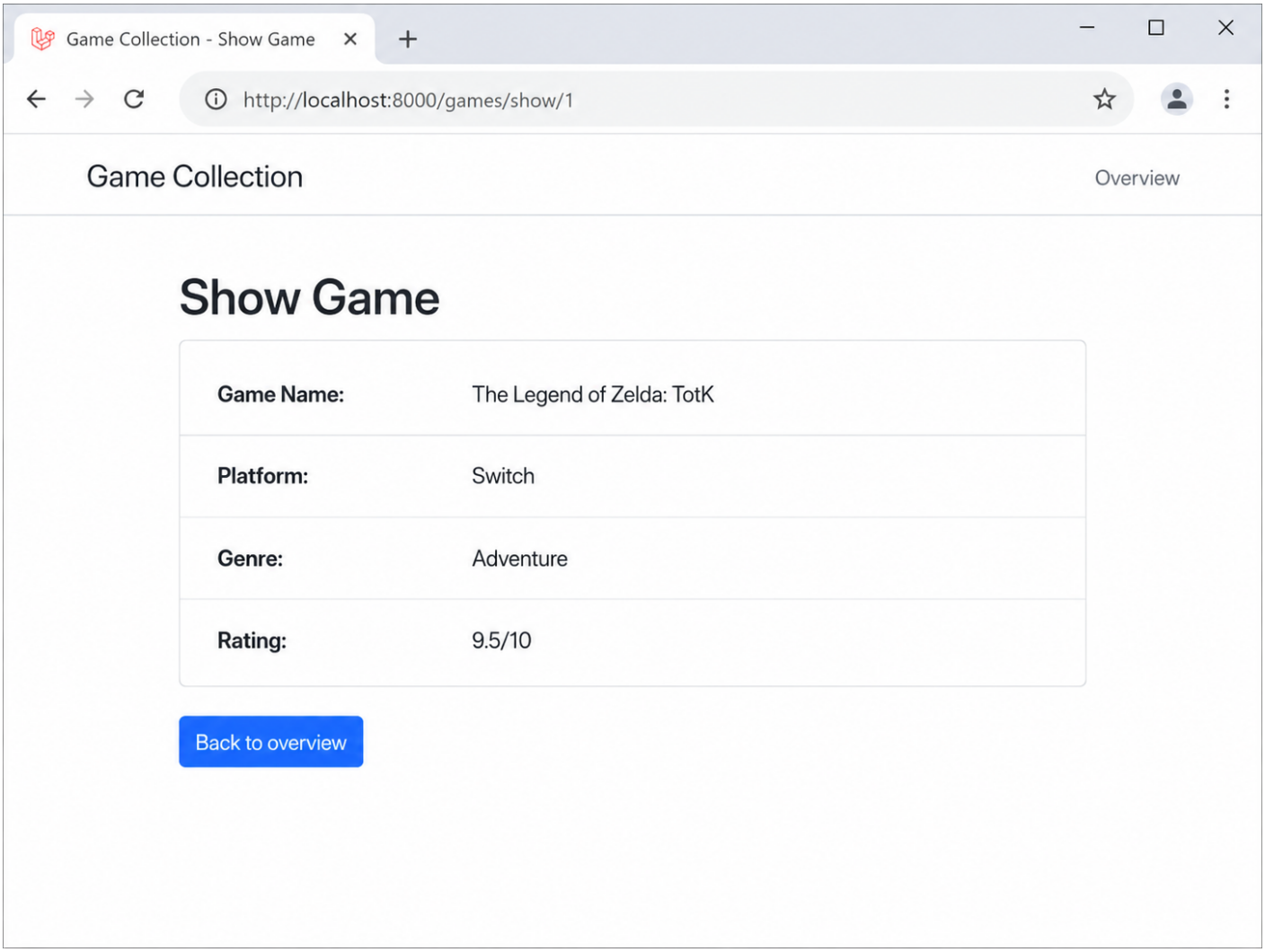
□□ Leerdoelen

- Je kunt zelf een route maken naar een detailpagina.
- Je kunt zelf een controllerfunctie maken voor één game.
- Je kunt zelf een Blade-view maken om één game te tonen.
- Je kunt één record ophalen uit de database met een id.
- Je kunt een Show-knop toevoegen aan het index-overzicht.

□□ Uitleg

In deze mini challenge ga je zelf een **show-pagina** maken voor één game.

Tot nu toe kun je games toevoegen, bekijken in een overzicht, aanpassen en verwijderen. Nu ga je ervoor zorgen dat een gebruiker op een knop kan klikken om de details van één specifieke game te bekijken.



Je maakt deze opdracht zoveel mogelijk zelf. Gebruik de eerdere lessen als voorbeeld.

Opdracht - Show Game maken

1. Maak een nieuwe route voor het tonen van één game.
2. Maak een nieuwe controllerfunctie genaamd `show()`.
3. Haal in deze functie één game op met het juiste `id`.
4. Stuur deze game door naar een nieuwe view.
5. Maak een nieuwe Blade-view voor de detailpagina.
6. Toon op de detailpagina minimaal de volgende gegevens:
   - Game name
   - Platform
   - Genre
   - Rating

7. Voeg in het overzicht een knop **Show** toe naast **Edit** en **Delete**.

PLAATJE!!

□□ Tips

- Kijk goed naar hoe de **Edit**-knop is gemaakt in `index.blade.php`.
- Kijk bij de routes hoe `games/edit/{id}` werkt.
- In de controller kun je spieken bij de functie `edit($id)`.
- Voor het ophalen van één game kun je weer `Game::find($id)` gebruiken.
- De nieuwe view lijkt waarschijnlijk op `edit.blade.php`, maar je gebruikt geen formulier.
- Denk eraan dat je de juiste game moet meegeven aan de view.
- Gebruik de base template met `@extends('base')` en `@section`.

□□ Denk zelf na

Je krijgt deze keer geen volledige startercode. Probeer zelf te bedenken welke code waar moet komen.

Denk aan deze onderdelen:

1. Welke route heb je nodig?
2. Welke controllerfunctie hoort bij die route?
3. Hoe haal je de juiste game op uit de database?
4. Hoe stuur je de game door naar de view?
5. Hoe noem je de nieuwe view?
6. Hoe ziet de Show-knop in het overzicht eruit?

□□ Opdracht - Testen

1. Start de Laravel server.
2. Open het overzicht via `http://localhost:8000/games`.
3. Klik bij een game op **Show**.

4. Controleer of je op een detailpagina komt.
5. Controleer of de juiste gegevens van die game zichtbaar zijn.
6. Test dit bij minimaal twee verschillende games.

Opdracht - Git commit

1. Stage alle wijzigingen.
2. Commit met de message: **Mini Challenge: show game pagina**.
3. Klik op **Sync Changes**.

Inleveren

1. Screenshot van het overzicht met de **Show**-knop zichtbaar.
2. Screenshot van de detailpagina van één game met de URL zichtbaar.
3. Screenshot van je route in `routes/web.php`.
4. Screenshot van je controllerfunctie `show()`.
5. Screenshot van je nieuwe Blade-view.
6. Screenshot van je GitHub repository met de laatste commit.
7. Korte uitleg in eigen woorden: welke stappen neemt Laravel vanaf het klikken op de Show-knop tot het tonen van de detailpagina?

Revision #4
Created 2026-06-09 10:25:19 UTC by Aron
Updated 2026-06-09 10:35:46 UTC by Aron