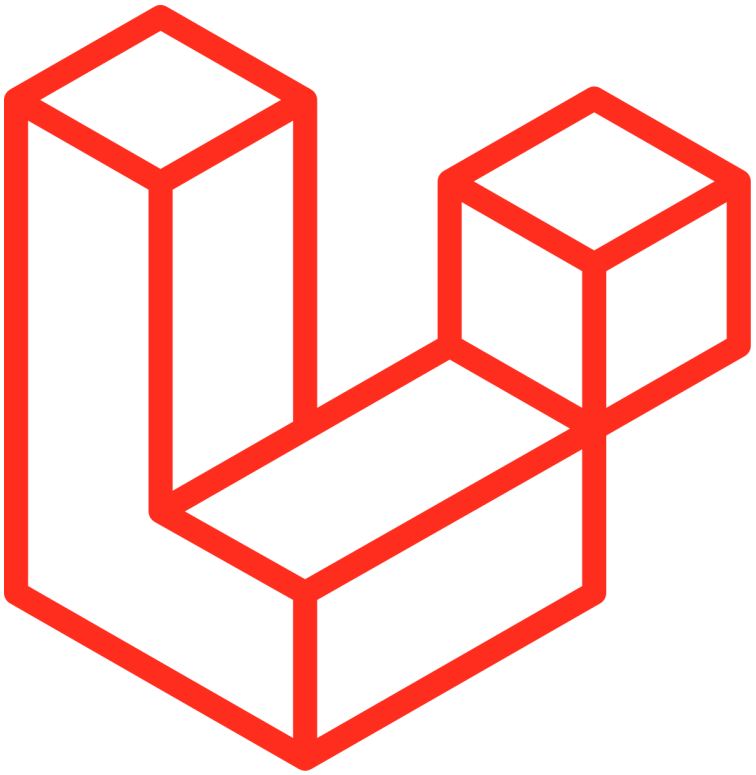
PROJECTE LARAVEL



Professor: Jordi Vega

Assignatura: M9 Programació de serveis i processos

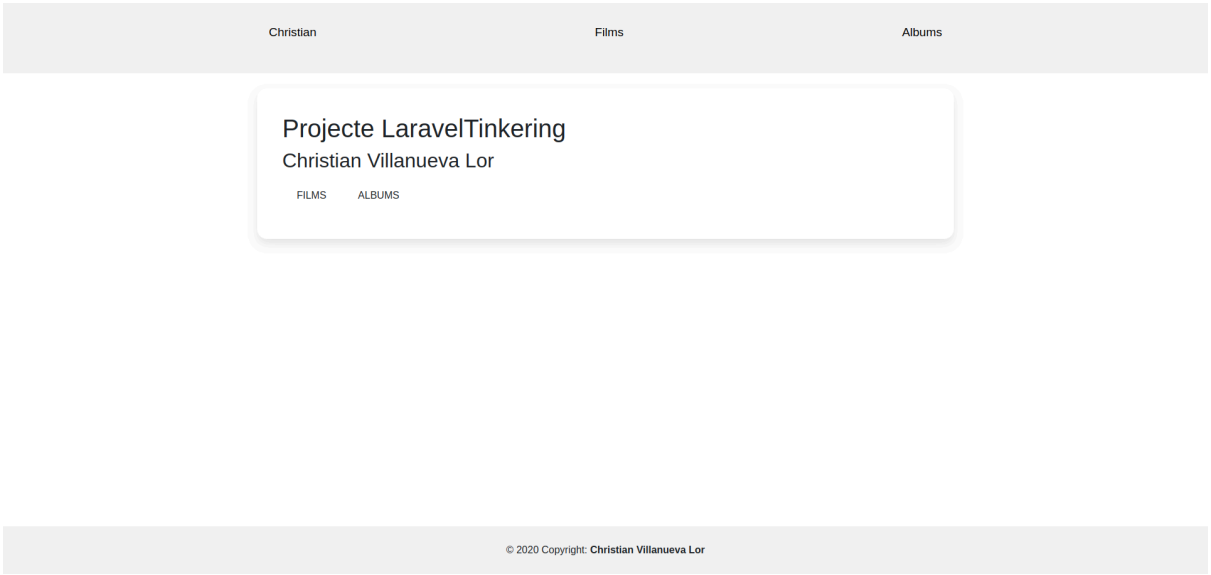
Nom: Christian

Cognoms: Villanueva Lor

Índex:

1. Creació de la *landing*

En accedir a la pàgina, ens trobem amb una landing page minimalista i fàcil de navegar. En aquesta, destaquen dos botons principals: un que redirigeix a la secció de pel·lícules i un altre a la d'àlbums, permetent un accés ràpid i directe a ambdues àrees. A més, a la part superior de la pàgina es mostra el títol del projecte, juntament amb el meu nom, per proporcionar context sobre la pàgina.



Per configurar-ho, al fitxer web.php de la carpeta routes, he afegit la següent ruta de la següent manera:

```
Route::get( uri: '/', function () {
    return view( view: 'home');
});
```

1.1 Fitxer layout amb header i footer

El fitxer **`app.blade.php`** forma part de l'arquitectura de Laravel i es troba dins la carpeta **`views/layouts`**. Aquest fitxer s'utilitza com a plantilla base per a totes les pàgines de l'aplicació. La idea és definir una estructura comuna (com el capçalera, la barra de navegació i el peu de pàgina) que es pugui reutilitzar a través de diferents pàgines del lloc web. Així, en lloc de definir aquests elements en cada pàgina de manera individual, es crea una sola plantilla base que proporciona el disseny comú.

Les pàgines específiques del lloc web poden "heretar" aquesta plantilla base i afegir-hi el contingut únic per a cada vista, utilitzant la directiva Blade **`@yield('content')`**. Això permet que l'aplicació mantingui una estructura consistent i facilita el manteniment i la modificació del disseny de manera eficient, ja que qualsevol canvi en l'estructura base es reflectirà automàticament a totes les pàgines que utilitzin aquesta plantilla.

```
<body>
<nav>
    <a href="/">Christian</a>
    <a href="{{ route('films.index') }}">Films</a>
    <a href="{{ route('albums.index') }}">Albums</a>
</nav>

<div class="container mt-4 mb-4">
    @yield('content')
</div>

<footer class="text-center text-lg-start">
    <!-- Copyright -->
    <div class="text-center p-3">
        © 2020 Copyright:
        <strong>Christian Villanueva Lor</strong>
    </div>
    <!-- Copyright -->
</footer>

</body>
```

2. Pàgina de les pel·lícules

En la creació de la pàgina de pel·lícules, el primer pas ha estat crear el model **`Film`** mitjançant la comanda **`php artisan make:model Film -m`**. Aquesta comanda genera un fitxer de migració per a la base de dades a la carpeta **`database/migrations`**, on haurem d'indicar els camps que volem que tingui la taula corresponent.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    no usages
    public function up(): void
    {
        Schema::create( table: 'films', function (Blueprint $table) {
            $table->id();
            $table->string( column: 'title');
            $table->string( column: 'director');
            $table->integer( column: 'year');
        });
    }

    /**
     * Reverse the migrations.
     */
    no usages
    public function down(): void
    {
        Schema::dropIfExists( table: 'films');
    }
};

```

Un cop indicats els camps de la taula, executarem les migracions de la base de dades mitjançant la comanda **`php artisan migrate`**, la qual s'encarregarà d'aplicar els canvis i crear la taula a la base de dades.

```
alumne@christian:~/Documents/M09/LaravelTinkering/LaravelTinkering$ php artisan migrate

  INFO  Running migrations.

  2024_12_19_143613_create_films_table ..... 7.18ms DONE

alumne@christian:~/Documents/M09/LaravelTinkering/LaravelTinkering$
```

Seguidament, crearem un controlador per gestionar les rutes i les funcions corresponents, utilitzant la comanda **`php artisan make:controller BookController --resource`.** Aquesta comanda generarà un controlador amb totes les funcions bàsiques per al CRUD. Després, afegirem aquest controlador al fitxer de rutes **`web.php`** de la següent manera.

```
Route::resource('films', FilmController::class);
```

Un cop creat el controlador **`FilmController`**, es defineixen diverses funcions per a gestionar les operacions relacionades amb els films. A continuació, es descriu el que fa cada una de les funcions del controlador:

- **index()**: Aquesta funció recupera tots els films de la base de dades mitjançant **`Film::all()`** i els passa a la vista **`films.index`**. Aquesta vista mostrarà una llista de tots els films disponibles.
- **create()**: Aquesta funció retorna la vista **`films.create`**, que probablement conté un formulari per afegir un nou film.
- **store(Request $request)**: Aquesta funció s'encarrega de validar les dades del formulari que es fan servir per crear un nou film. Es valida que els camps **`title`**, **`director`** i **`year`** siguin obligatoris. Si la validació passa, es crea un nou registre de film a la base de dades amb **`Film::create($request->all())`**. Després redirigeix a la pàgina principal de films (**`films.index`**) amb un missatge d'èxit.
- **show(Film $film)**: Aquesta funció rep un model de film específic (el **`$film`** és passat automàticament per Laravel) i retorna la vista **`films.show`**, passant el film seleccionat a la vista. Aquesta vista es farà servir per mostrar els detalls d'un film específic.

- **edit(Film $film)**: Aquesta funció retorna la vista **`films.edit`**, que conté un formulari per editar les dades d'un film existent. Passa el model del film seleccionat a la vista per poder omplir el formulari amb les dades actuals.
- **update(Request $request, Film $film)**: Aquesta funció valida novament els camps **`title`**, **`director`** i **`year`** per garantir que es compleixin. Un cop validades les dades, s'actualitzen els camps del model **`film`** amb **`update()`**. Després redirigeix a la pàgina de llista de films (**`films.index`**) amb un missatge d'èxit.
- **destroy(Film $film)**: Aquesta funció elimina un film de la base de dades amb **`delete()`**. Un cop eliminat, redirigeix a la pàgina principal de films (**`films.index`**) amb un missatge d'èxit.

Aquest conjunt de funcions ofereix una gestió completa de CRUD (crear, llegir, actualitzar, eliminar) per als films dins de l'aplicació.

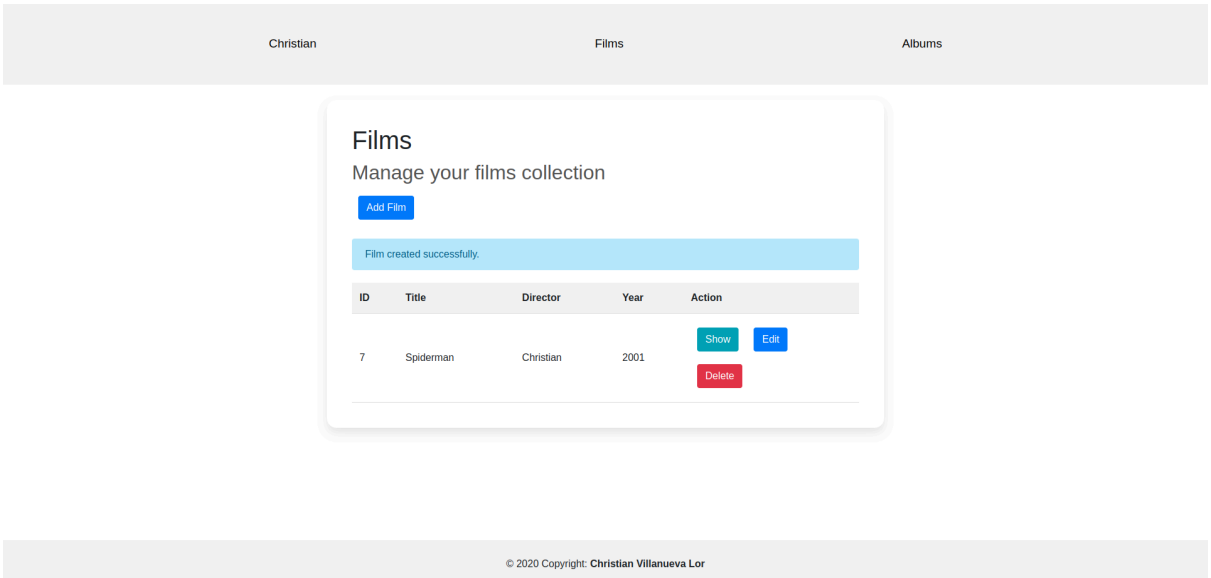
Finalment, dins la carpeta **`views`** que es troba a la carpeta **`resources`**, crearem una subcarpeta per a les pàgines de les pel·lícules. En aquesta subcarpeta, crearem quatre fitxers: un per a la pàgina on es llistaran totes les pel·lícules, un altre per a la creació de noves pel·lícules, un altre per a l'edició de les pel·lícules, i finalment, un altre per a veure els detalls d'una pel·lícula específica.

2.1 Llista de pel·lícules

En aquesta pàgina, es mostra una llista de films en una taula i permet realitzar accions com veure, editar i eliminar cada pel·lícula. Primer, utilitza la directiva **`@extends('layouts.app')`** per incloure una plantilla base que probablement conté la infraestructura comuna del lloc (com el capçalera i peu de pàgina). A continuació, la secció de contingut es defineix amb **`@section('content')`**, on s'inclou el codi HTML específic d'aquesta pàgina.

Es mostra un missatge d'èxit si la sessió conté un missatge amb la clau 'success', que es visualitza en una alerta. Després, a la taula es fa un bucle amb **`@foreach`** per mostrar tots els films, iterant sobre la variable **`$films`**. Cada pel·lícula es presenta amb el seu ID, títol, director i any. Al costat de cada pel·lícula, hi ha tres botons: un per veure els detalls del film, un altre per editar-lo i un últim per eliminar-lo. El botó de "Show" enllaça a la vista de detalls, el botó "Edit" enllaça a la pàgina d'edició i el botó "Delete" elimina el film, enviant una petició **DELETE** a la ruta corresponent.

Finalment, s'apliquen estils CSS per millorar la visualització de la taula i dels botons, incloent un desplaçament per la taula si hi ha massa elements i efectes per als botons. Aquest codi implementa una interfície d'usuari senzilla però funcional per gestionar una col·lecció de films en una aplicació Laravel.

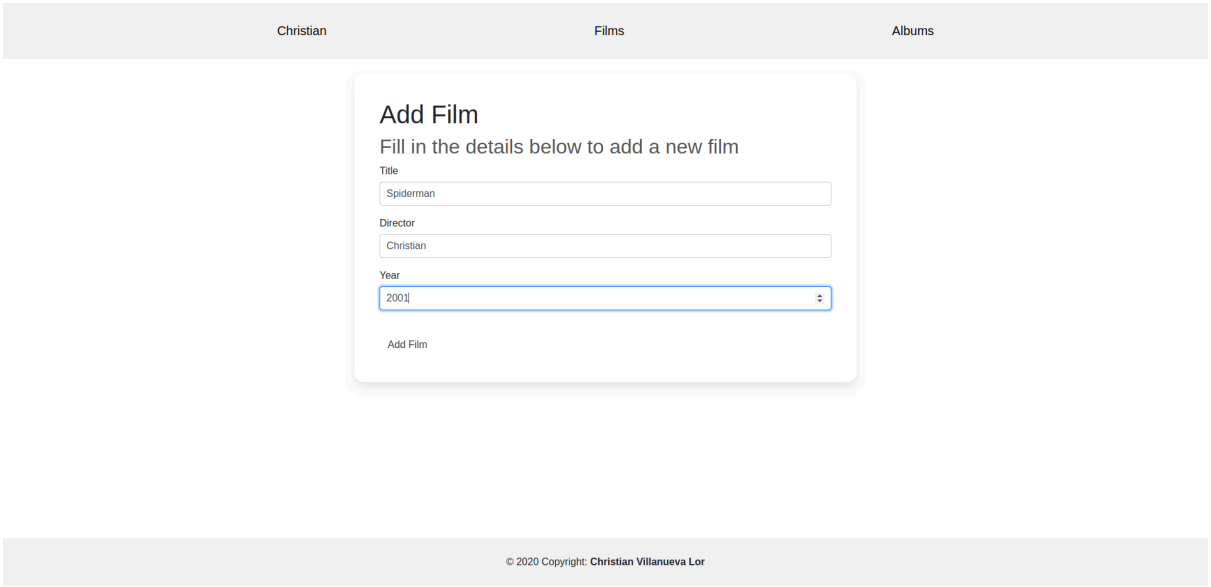


2.2 Creació d’una nova pel·lícula

Aquesta pàgina s’utilitza per mostrar un formulari per afegir una nova pel·lícula. Utilitza la plantilla base **`layouts.app`** mitjançant la directiva **`@extends('layouts.app')`**, el que permet heretar la seva estructura comuna. A l'interior de la secció **`@section('content')`**, s'afegeix el contingut específic d'aquesta vista, que inclou un formulari per inserir els detalls de la pel·lícula.

El formulari utilitza la ruta **`{{ route('films.store') }}`** per enviar les dades mitjançant el mètode **`POST`** al controlador de films, específicament a la funció **`store`**. Aquesta ruta està associada a una acció que s'encarrega de validar i desar la nova pel·lícula a la base de dades.

Els camps del formulari permeten introduir el títol, director i any de la pel·lícula, i cadascun està marcat amb l'atribut **`required`**, la qual cosa obliga a l'usuari a omplir-los abans de poder enviar el formulari. Finalment, hi ha un botó de submissió per afegir la pel·lícula.

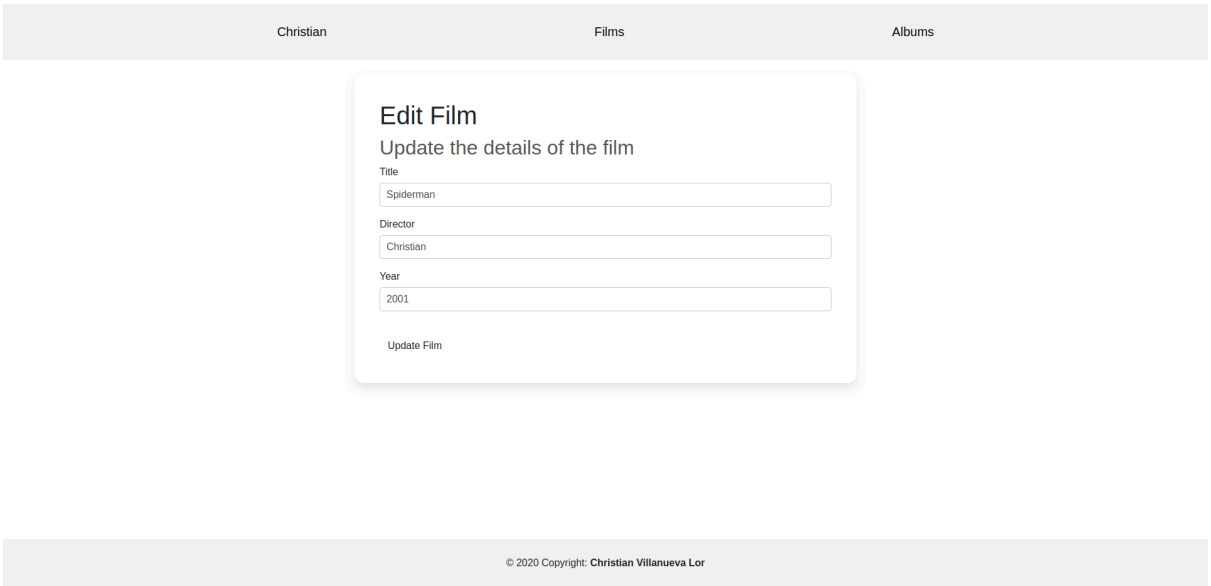


2.3 Edició d’una pel·lícula

Aquesta pàgina serveix per editar els detalls d'una pel·lícula existent en una aplicació Laravel. Similar a la vista anterior, s'utilitza la plantilla base **`layouts.app`** mitjançant la directiva **`@extends('layouts.app')`** i es defineix el contingut específic dins de **`@section('content')`.**

El formulari per editar la pel·lícula utilitza la ruta **`{{ route('films.update', $film->id) }}`**, que apunta a la ruta associada a la funció **`update`** del controlador de pel·lícules. Aquesta ruta inclou l'ID de la pel·lícula (**`$film->id`**) per identificar de manera única la pel·lícula que s'ha de modificar.

S'utilitza la directriu **`@method('PUT')`** per especificar que el mètode de la petició HTTP serà **`PUT`**, ja que estem actualitzant una entrada existent a la base de dades. Els camps del formulari contenen els valors actuals de la pel·lícula, com el títol, el director i l'any, que es mostren mitjançant **`{{ $film->title }}`**, **`{{ $film->director }}`**, i **`{{ $film->year }}`** respectivament. Aquests valors es poden editar i, després de la submissió, es desa la pel·lícula actualitzada a la base de dades.

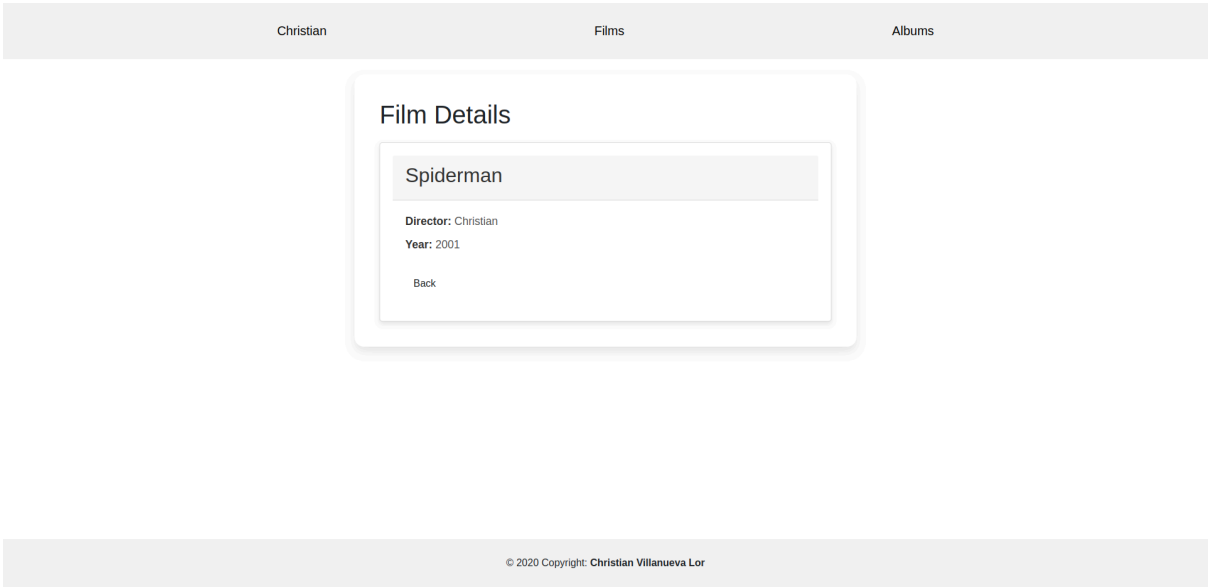


2.4 Detalls d'una pel·lícula

Aquesta pagina serveix per mostrar els detalls d'una pel·lícula a l'aplicació Laravel. Al principi, s'estableix la plantilla base mitjançant **`@extends('layouts.app')`** i s'inclou el contingut dins de la secció **`@section('content')`**.

La vista presenta una estructura bàsica amb un títol i una descripció dels detalls de la pel·lícula. El títol de la pel·lícula es mostra mitjançant **`{{ $film->title }}`**, mentre que el director i l'any es mostren utilitzant **`{{ $film->director }}`** i **`{{ $film->year }}`** respectivament. Aquests valors provenen de la variable **`$film`** que s'ha passat al formulari des del controlador. Aquestes dades es mostren dins de la secció **`card-body`** per garantir un disseny net i visualment atractiu.

S'usa també un botó "Back" que enllaça a la ruta **`films.index`**, la qual retorna a la vista principal on es mostren totes les pel·lícules. Així, l'usuari pot veure els detalls d'una pel·lícula i tornar a la llista de pel·lícules amb facilitat.



3. Pàgina dels albums

En la creació de la pàgina de pel·lícules, el primer pas ha estat crear el model **`Album`** mitjançant la comanda **`php artisan make:model Album -m`**. Aquesta comanda genera un fitxer de migració per a la base de dades a la carpeta **`database/migrations`**, on haurem d'indicar els camps que volem que tingui la taula corresponent.

```php
use ...

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    no usages
    public function up(): void
    {
        Schema::create( table: 'albums', function (Blueprint $table) {
            $table->id();
            $table->string( column: 'title');
            $table->string( column: 'singer');
            $table->integer( column: 'songs');
            $table->integer( column: 'year');
        });
    }

    /**
     * Reverse the migrations.
     */
    no usages
    public function down(): void
    {
        Schema::dropIfExists( table: 'albums');
    }
};
```

Un cop indicats els camps de la taula, executarem les migracions de la base de dades mitjançant la comanda **`php artisan migrate`**, la qual s'encarregarà d'aplicar els canvis i crear la taula a la base de dades.

```
alumne@christian:~/Documents/M09/LaravelTinkering/LaravelTinkering$ php artisan migrate

  INFO  Running migrations.

  2024_12_19_143613_create_films_table ..... 7.18ms DONE

alumne@christian:~/Documents/M09/LaravelTinkering/LaravelTinkering$
```

Seguidament, crearem un controlador per gestionar les rutes i les funcions corresponents, utilitzant la comanda **`php artisan make:controller AlbumController --resource`.** Aquesta comanda generarà un controlador amb totes les funcions bàsiques per al CRUD. Després, afegirem aquest controlador al fitxer de rutes **`web.php`** de la següent manera.

```
Route::resource('albums', AlbumController::class);
```

Un cop creat el controlador **`AlbumController`**, es defineixen diverses funcions per gestionar les operacions relacionades amb els àlbums. A continuació, es descriu el que fa cada una de les funcions del controlador:

- **`index()`**: Aquesta funció recupera tots els àlbums de la base de dades mitjançant **`Album::all()`** i els passa a la vista **`albums.index`**. Aquesta vista mostrarà una llista de tots els àlbums disponibles.
- **`create()`**: Aquesta funció retorna la vista **`albums.create`**, que probablement conté un formulari per afegir un nou àlbum.
- **`store(Request $request)`**: Aquesta funció s'encarrega de validar les dades del formulari que es fan servir per crear un nou àlbum. Es valida que els camps **`title`**, **`singer`**, **`year`** i **`songs`** siguin obligatoris. Si la validació passa, es crea un nou registre d'àlbum a la base de dades amb **`Album::create($request->all())`**. Després redirigeix a la pàgina principal d'àlbums (**`albums.index`**) amb un missatge d'èxit.
- **`show(Album $album)`**: Aquesta funció rep un model d'àlbum específic (el **`$album`** és passat automàticament per Laravel) i retorna la vista **`albums.show`,** passant l'àlbum seleccionat a la vista. Aquesta vista es farà servir per mostrar els detalls d'un àlbum específic.

- **`edit(Album $album):`** Aquesta funció retorna la vista **`albums.edit`**, que conté un formulari per editar les dades d'un àlbum existent. Passa el model de l'àlbum seleccionat a la vista per poder omplir el formulari amb les dades actuals.
- **`update(Request $request, Album $album):`** Aquesta funció valida novament els camps **`title`**, **`singer`**, **`year`** i **`songs`** per garantir que es compleixin. Un cop validades les dades, s'actualitzen els camps del model àlbum amb **`update()`**. Després redirigeix a la pàgina de llista d'àlbums (**`albums.index`**) amb un missatge d'èxit.
- **`destroy(Album $album)`**: Aquesta funció elimina un àlbum de la base de dades amb **`delete().`** Un cop eliminat, redirigeix a la pàgina principal d'àlbums (**`albums.index`**) amb un missatge d'èxit.

Aquest conjunt de funcions ofereix una gestió completa de CRUD (crear, llegir, actualitzar, eliminar) per als àlbums dins de l'aplicació.

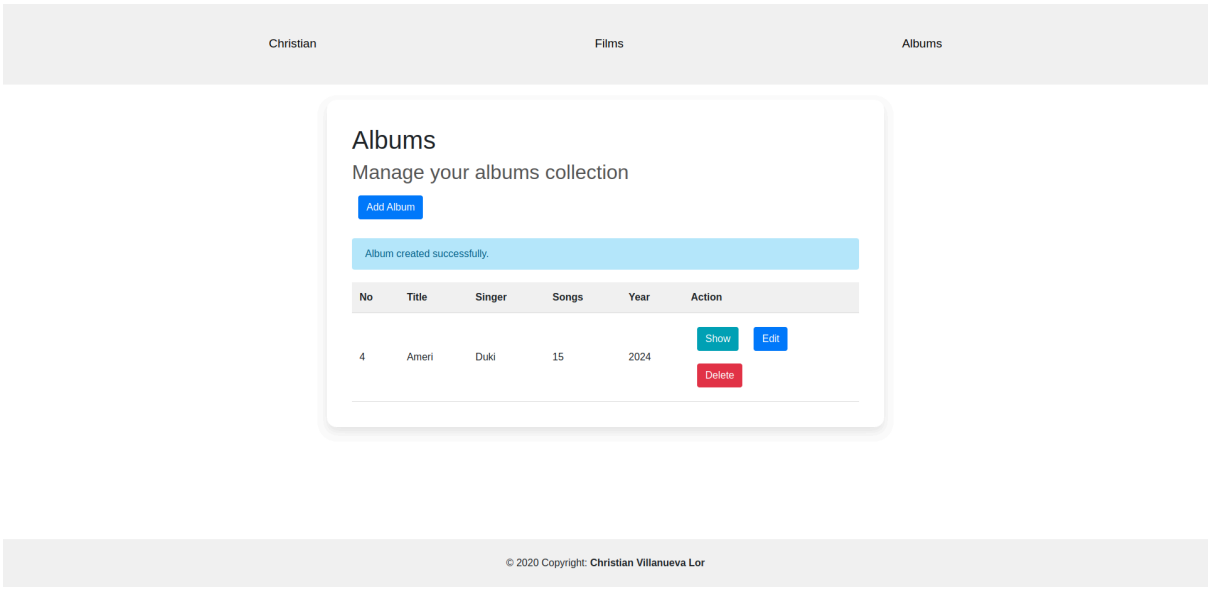
Finalment, dins la carpeta **`views`** que es troba a la carpeta **`resources`**, crearem una subcarpeta per a les pàgines dels àlbums. En aquesta subcarpeta, crearem quatre fitxers: un per a la pàgina on es llistaran tots els àlbums, un altre per a la creació de nous àlbums, un altre per a l'edició dels àlbums, i finalment, un altre per a veure els detalls d'un àlbum específic.

3.1 Llista de pel·lícules

Aquesta pàgina serveix per mostrar una llista d'àlbums. El principal procés PHP es realitza mitjançant les directrius de Blade, el sistema de plantilles de Laravel.

Primer, la vista carrega un conjunt d'àlbums des del controlador, utilitzant una variable **`$albums`** que es passa des del controlador a través de la funció **`compact('albums')`**. Aquesta variable conté tots els àlbums que s'han recuperat de la base de dades. Cada element de la llista es representa en una fila de la taula, amb informació com el títol, el cantant, les cançons i l'any del l'àlbum.

En cas d'haver-hi un missatge de sessió (**`Session::get('success')`**), es mostrarà una alerta amb el missatge. A la taula es permet realitzar diverses accions: mostrar els detalls, editar i eliminar un àlbum. A través de la direcció **`route`**, es generen les rutes adequades per cada acció. Per eliminar un àlbum, s'utilitza un formulari amb el mètode **`DELETE`**, seguit d'un botó de tipus **`submit`** per realitzar l'eliminació en el servidor. El controlador s'encarregarà de gestionar aquesta petició, i un cop processada, redirigirà l'usuari a la llista d'àlbums.

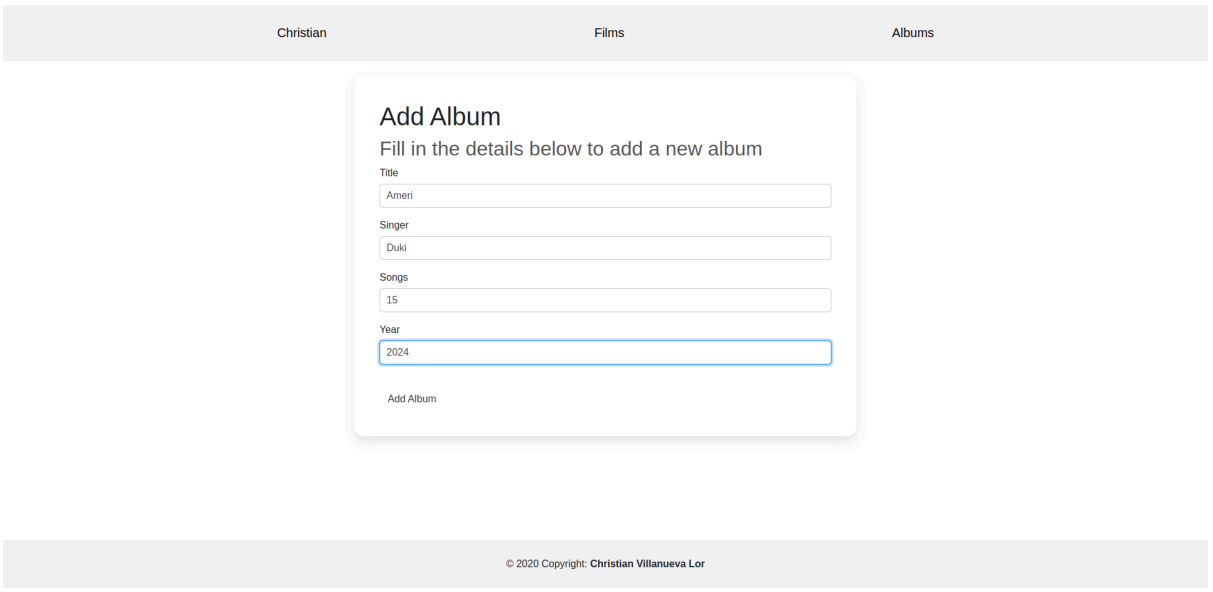


3.2 Creació d’una nova pel·lícula

Aquesta pàgina serveix per afegir un nou àlbum a la base de dades. La vista utilitza el sistema de plantilles Blade per generar el formulari i enviar-lo al servidor. En el formulari, es recullen els detalls de l'àlbum, com el títol, el cantant, el nombre de cançons i l'any de l'àlbum.

Quan l'usuari omple els camps i envia el formulari, les dades es processen mitjançant la ruta definida com **`albums.store`**, que s'encarrega de gestionar la creació del nou àlbum al controlador associat. A cada input se li aplica la validació **`required`**, assegurant-se que l'usuari ompli tots els camps abans d'enviar el formulari.

El formulari utilitza el mètode **`POST`** per enviar les dades al servidor, i després de completar la creació de l'àlbum, el controlador redirigeix l'usuari a una altra pàgina, com la llista d'àlbums, mostrant un missatge d'èxit.

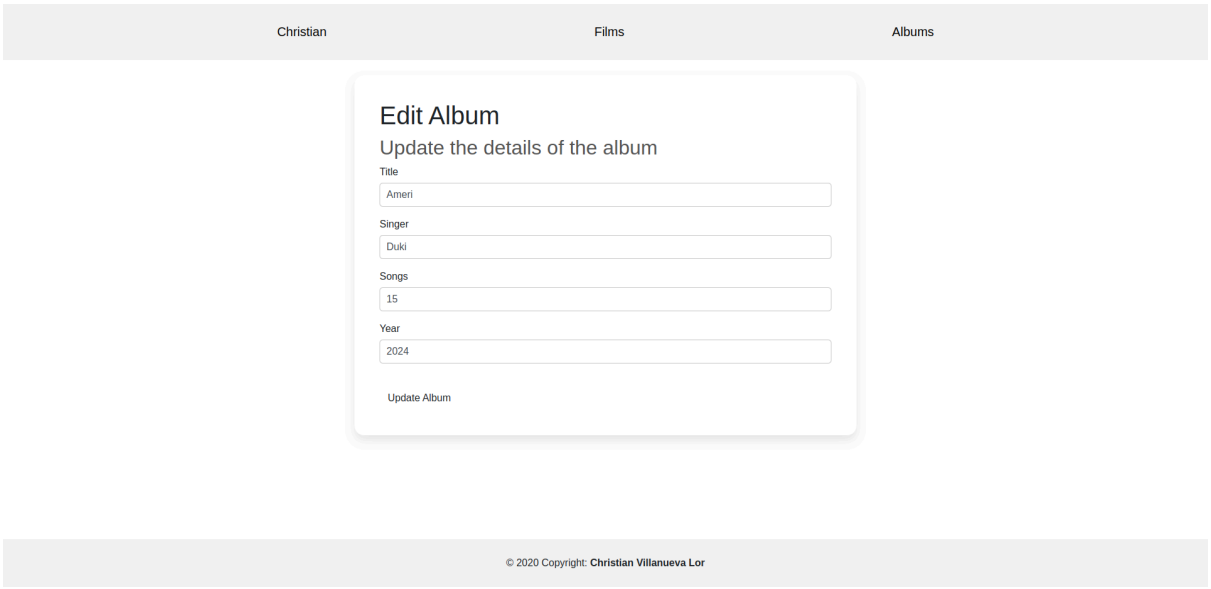


3.3 Edició d’una pel·lícula

Aquesta pàgina serveix per editar un àlbum existent. Com a part de l'aplicació, la vista permet actualitzar les dades d'un àlbum mitjançant un formulari HTML. El formulari es crea amb els valors actuals de l'àlbum, els quals es passen des del controlador. Així, l'usuari pot modificar el títol, el cantant, el nombre de cançons i l'any de l'àlbum.

El formulari s'envia mitjançant el mètode **PUT** per actualitzar les dades del model en la base de dades. A l'atribut **`action`** de l'etiqueta **`<form>`**, es defineix la ruta per actualitzar l'àlbum, que fa servir el mètode **`albums.update`**, juntament amb l'ID de l'àlbum a editar. A més, es fa servir el token CSRF per garantir la seguretat en la submissió del formulari.

Quan s'envia el formulari, els valors actualitzats es processen en el controlador, i si tot és correcte, es redirigeix l'usuari a la vista corresponent amb els canvis aplicats.



3.4 Detalls d'una pel·lícula

Aquest codi correspon a una vista de Laravel per mostrar els detalls d'una pel·lícula. En aquesta vista, es presenta informació específica sobre un film, com el títol, el director i l'any de l'estrena, que són passats al fitxer des del controlador mitjançant la variable **`$film`**.

La vista utilitza una estructura de disseny basada en una targeta (card) per mostrar la informació de manera atractiva i organitzada. La targeta es divideix en dues parts: un "header" per al títol del film, i un "body" per als detalls (director i any). A més, es crea un enllaç que permet retornar a la llista de pel·lícules (**`films.index`**) amb un botó de retorn.

El disseny de la pàgina està optimitzat per ser net i senzill, amb colors suaus i marges adequats per a una bona llegibilitat. Quan l'usuari desitja veure més pel·lícules, pot fer-ho fàcilment clicant en el botó de "Back", que redirigeix a la vista principal de la llista de films.