

## TRABALHO

### Projeto ACR

Artur Pereira  
Nº 2040415

Curso Técnico Superior em Programação

**UNIDADE CURRICULAR:**

Aplicações Centradas em Redes

**DOCENTE:**

Michael Silva

**DATA:**

26 de Dezembro de 2020

# ESCOLA SUPERIOR DE TECNOLOGIAS E GESTÃO

# Índice

---

|                                  |    |
|----------------------------------|----|
| Índice.....                      | 2  |
| Introdução.....                  | 3  |
| Estrutura da base de dados ..... | 4  |
| Controladores e CRUD .....       | 5  |
| MovieController .....            | 5  |
| GenderController .....           | 9  |
| ContactController .....          | 10 |
| Middleware.....                  | 11 |
| Eventos .....                    | 12 |
| Rotas.....                       | 12 |
| Modelos.....                     | 13 |
| Migrações .....                  | 14 |
| Views e Front End .....          | 16 |
| Conclusão .....                  | 20 |

# Introdução

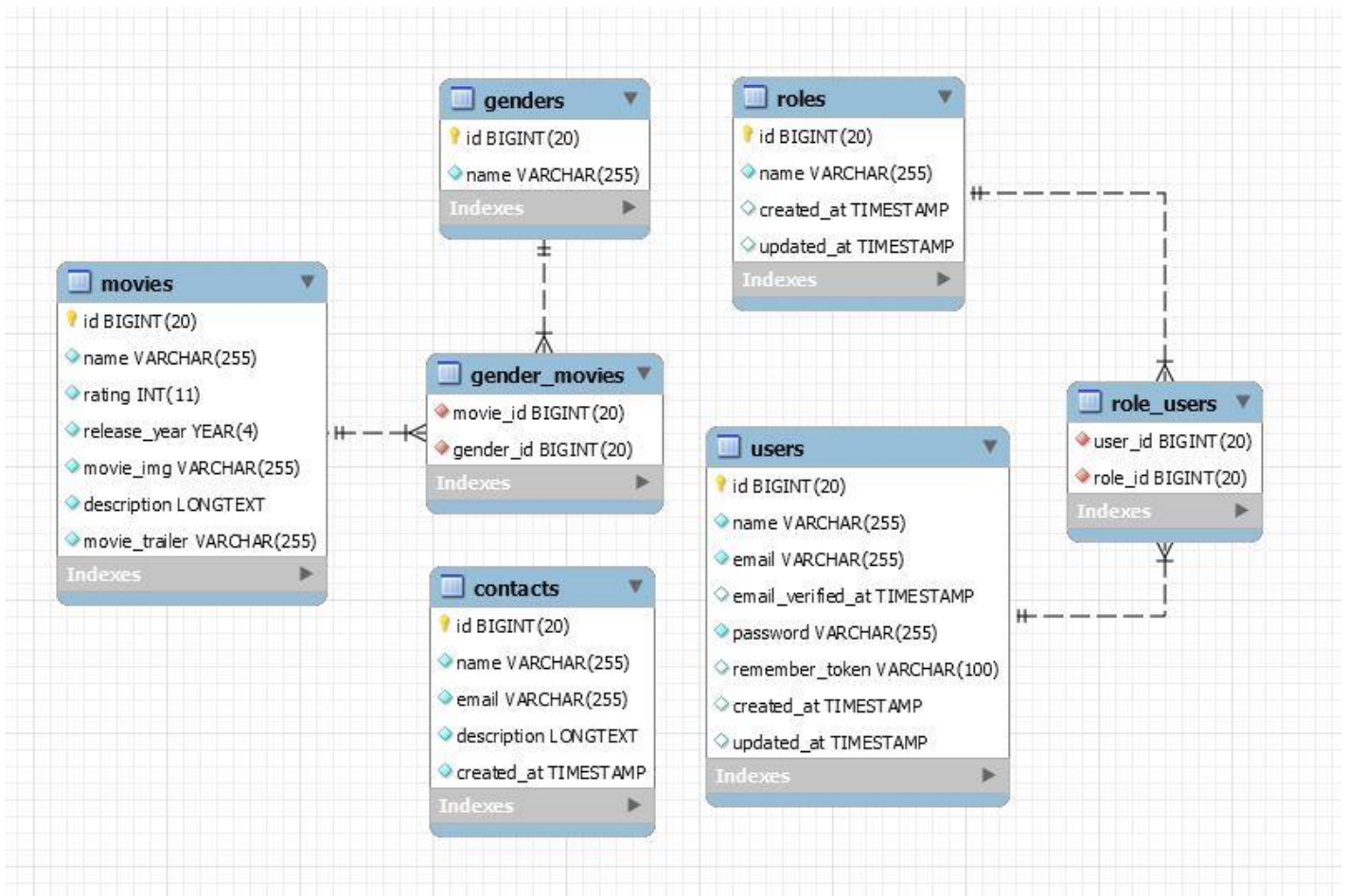
---

Este relatório é referente ao trabalho solicitado pelo professor Michael Silva. O trabalho tem como objetivo aplicar as principais técnicas adquiridas durante o 1º Semestre em Aplicações Centradas em Redes.

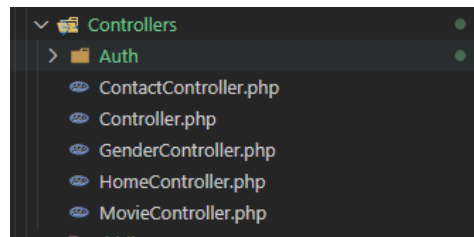
O Projeto tinha vários temas em que podíamos escolher, no entanto decidi escolher a criação de um Website em que permite aos utilizadores visualizarem filmes online, uma vez que no meu dia a dia costumo utilizar vários sites do mesmo género para visualizar todo o tipo de conteúdos.

A ideia por trás do website consiste em que o administrador introduz os filmes e os mesmos vão para uma base de dados onde vão ficar guardados, apenas os utilizadores que se encontram registados no site podem visualizar os seus conteúdos.

# Estrutura da base de dados



# Controladores e CRUD



## MovieController

Neste controlador encontrasse as funcionalidades mais importantes do website como por exemplo algumas das operações CRUD em que vai permitir ao utilizador visualizar todo o conteúdo disponível.

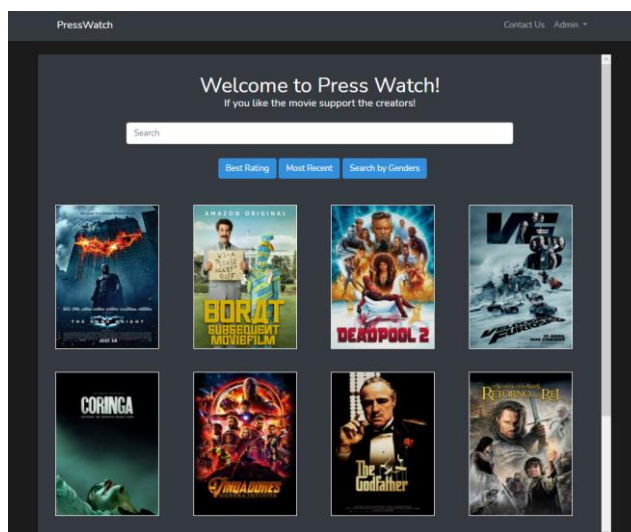
➔ Em baixo vai constar uma breve descrição do que as funções fazem.

```
public function index()
{
    if (request()->has('rating')) {
        //order by rating
        $movies = Movie::orderBy('rating','desc')->get();
    }elseif (request()->has('recent')) {
        //show most recent
        $movies = Movie::orderBy('release_year','desc')->get();
    } else {
        // showAllMovies
        // $movies = Movie::all();
        $movies = DB::table('movies')->simplePaginate(10);
    }

    $data = session('Message');
    $showUp=false;
    if (strlen($data) > 1) {
        $showUp = true;
    }

    return view('mainPage',['movies' => $movies, 'showUp' => $showUp]);
}
```

A função em cima representada vai procurar na base de dados os filmes que estão presentes na base de dados e enviar para a view principal onde apresenta todos os resultados, também foi introduzido queries de pesquisa de forma a que o utilizador caso pretenda possa facilitar a procura de informação.



```
public function create()
{
    $genders = Gender::all();
    return view('createMovie',['genders' => $genders]);
}
```

Vai redirecionar o utilizador para a pagina onde é realiza a inserção de novos conteúdos no site, a querie acima representada vai a base de dados pesquisar todos os géneros dos filmes existentes e envia para a view.

```
public function validationRules(Request $request)
{
    $validated = $request->validate([
        'name' => 'required|max:255',
        'rating' => 'required|integer|min:0|max:10',
        'release_year' => 'required|integer|min:1900|max:2080',
        'movie_img' => 'required|image|mimes:jpeg,png,jpg,gif|max:2048',
        'description' => 'required',
        'movie_trailer' => 'required|max:255',
        'gender' => 'exists:genders,id'
    ]);
}
```

Foi criada uma função para a validação dos dados inseridos na base de dados em que será utilizada na função store onde vai armazenar os contuedos enviados.

```
public function store(Request $request)
{
    $validation = new MovieController;
    $validation->validationRules($request);

    $movie_img = "";
    if ($request->has('movie_img')) {
        $image = $request->file('movie_img');
        $iname='prod'.'_'.time();
        $folder = 'movies/';
        $fileName=$iname.'.' . $image->getClientOriginalExtension();
        $filePath= $folder . $fileName;

        $image->storeAs($folder,$fileName,'public');
        $movie_img = "/storage/".$filePath;
    }

    $newMovie = new Movie(); //instancia do modelo para saber onde grava as informações
    $newMovie->name = $request->name;
    $newMovie->rating = $request->rating;
    $newMovie->release_year = $request->release_year;
    $newMovie->movie_trailer = $request->movie_trailer;
    $newMovie->description = $request->description;
    $newMovie->movie_img = $movie_img;
    $newMovie->save();

    $newGender = new GenderMovie();
    $newGender->gender_id = $request->gender;
    $newGender->movie_id = $newMovie['id']; //vai buscar o ID do novo e insire na tabela de muitos para muitos
    $newGender->save();

    return redirect('/movie')->with('Message','Movie has been added to the DB');
}
```

Esta função vai receber a informação introduzida na view e envia para a base de dados toda a informação introduzida através do form.

Insert a new movie

Movie Name

Rating

Release Year

Escolher ficheiro

Nenhum ficheiro selecionado

Movie Trailer Link

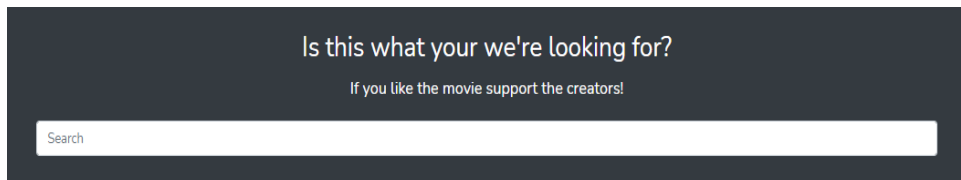
Description

Select Gender

Submit

```
public function search()
{
    $search_name = $_GET['query'];
    $movie_search = Movie::where('name', 'LIKE', '%'.$search_name.'%')->get();
    // dd($movie_search);
    return view('search', ['movie_search' => $movie_search]);
}
```

Permite ao utilizador utilizar a caixa de pesquisa que está disponível na view, faz uma query a base de dados com o nome introduzido e depois redireciona esses resultados para a view onde os mesmos são apresentados.



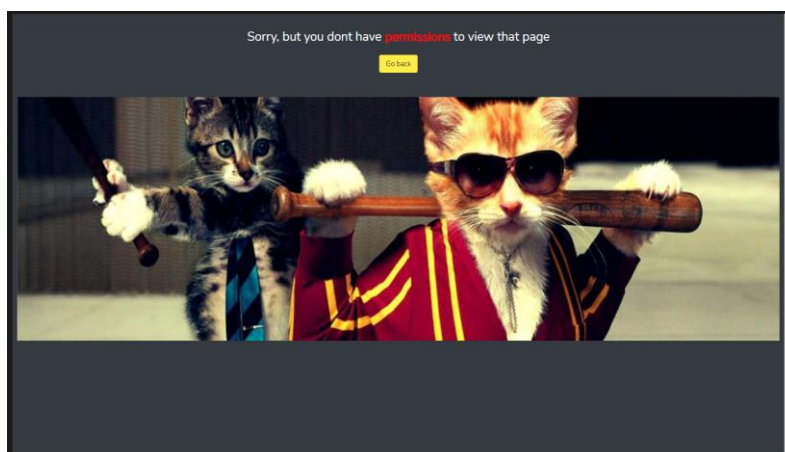
De forma a validar se o utilizador tem ou não permissões para visualizar a informação transmitida foi utilizado uma função disponível do laravel em que utiliza o middleware que criei para verificar se é um admin, também utilizei o middleware mesmo do laravel para validar se o utilizador está logado.

```
class MovieController extends Controller
{
    public function __construct()
    {
        $this->middleware(CheckAdmin::class, ['except' => ['index', 'show', 'noPermissions', 'search']]);
        $this->middleware('auth', ['except' => ['index', 'orderByRating']]);
    }
}
```

Isto faz com que todos os metodos que estam dentro do controller passem pelo middleware, apenas os que estam marcados como exceção é que estam “imunes” a serem verificados.

```
public function noPermissions()
{
    return view('noPermission');
}
```

A função vai fazer com que o utilizador caso não tenha permissões seja redirecionado para uma página que foi criada.

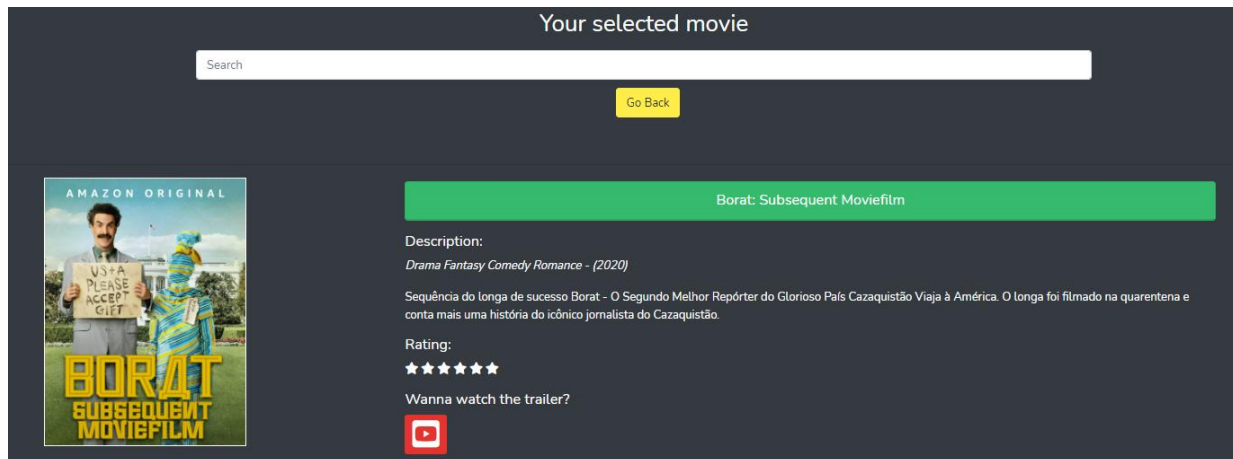


```

public function show($id)
{
    $movie = Movie::findOrFail($id);
    $movieGenders = Movie::findOrFail($id)->genres->pluck('name');
    return view('movieDetails',['movie' => $movie,'movieGenders' => $movieGenders]);
}

```

Esta função faz com que o utilizador ao carregar num filme que está presente na pagina inicial recebe o id desse filme como parametro, depois pesquisa através da querie o filme e o seu genero e envia para a view para o utilizador poder ver mais detalhes sobre o filme.



A função que vou demonstrar em baixo basicamente é funciona como a função store so que vai permitir ao admin alterar a informação de um determinado filme, caso o mesmo pretenda.

```

public function update(Request $request,$id)
{
    $movieEdit = Movie::findOrFail($id);
    if ($request->movie_img == NULL) { ...
    } else { ...
    }

    if ($request->name == NULL) { ...
    } else { ...
    }

    if ($request->rating == NULL) { ...
    } else { ...
    }

    if ($request->release_year == NULL) { ...
    } else { ...
    }

    if ($request->movie_trailer == NULL) { ...
    } else { ...
    }

    if ($request->description == NULL) { ...
    } else { ...
    }

    $movieEdit->save();
    //cria um novo genero para o filme (isto é se o filme tiver muitos generos)
    $newGender = new GenderMovie(); //instancia o genero
    if ($request->gender == "Select Gender") {
        return redirect('/')->with('Message','Movie has been updated in the DB');
    } else {
        $validated = $request->validate(['gender' => 'exists:genres,id']);
        $newGender->gender_id = $request->gender;
        $newGender->movie_id = $movieEdit['id'];
    }
    $newGender->save();
    return redirect('/')->with('Message','Movie has been updated in the DB');
}

```



Depois de gravar as alterações envia para a pagina principal com uma mensagem a indicar como foi sucedida a operação.

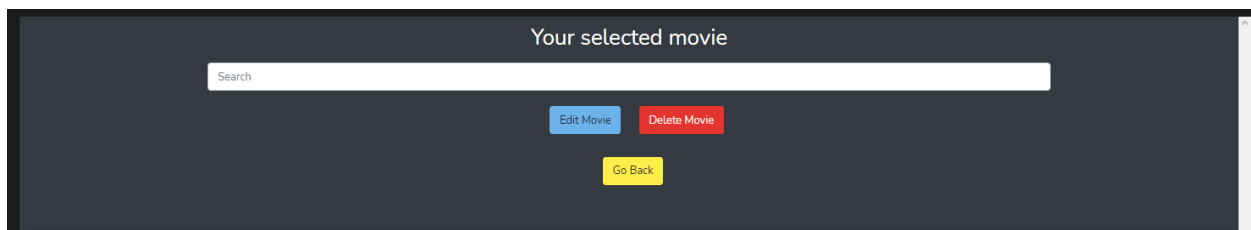
E por fim neste controlador temos a função destroy em que vai apagar a informação que está na base de dados e apaga em simultaneo a imagem do filme que estava guardada.

```
public function destroy($id)
{
    $movieRemove = Movie::findOrFail($id);

    $fileFullName = $movieRemove['movie_img'];
    $filename= str_replace('/storage/movies/', '', $fileFullName);

    Storage::delete('/public/movies/' . $filename);
    $movieRemove->delete();
    return redirect('/');
}
```

Botões que estão disponíveis quando está como admin de forma a alterar ou apagar os conteúdos.



## GenderController

Foi criada uma relação de muitos para muitos entre os generos e os filmes, uma vez que um filme pode ter muitos generos e muitos generos podem fazer parte de varios filmes.

Primeiro criei tres tabelas na base de dados.

Tabela “movies” -> Guarda o nome dos roles disponiveis para utilização

Tabela “genders” -> Guarda o nome de todos os utilizadores e outras informações.

Tabela “gender\_ movies” -> Vai guardar a chave estrangeira de ambas as tabelas (“movies” e “genders”) em que neste são os Ids.

Depois nos modelos foi criada uma relação entre ambas as tabelas.

```
class Gender extends Model
{
    use HasFactory;

    public function movies()
    {
        return $this->belongsToMany(Movie::class, 'gender_movies');
    }
}

class Movie extends Model
{
    use HasFactory;
    public $timestamps = false;
    protected $guarded = [];

    public function genders()
    {
        return $this->belongsToMany(Gender::class, 'gender_movies');
    }
}
```

```

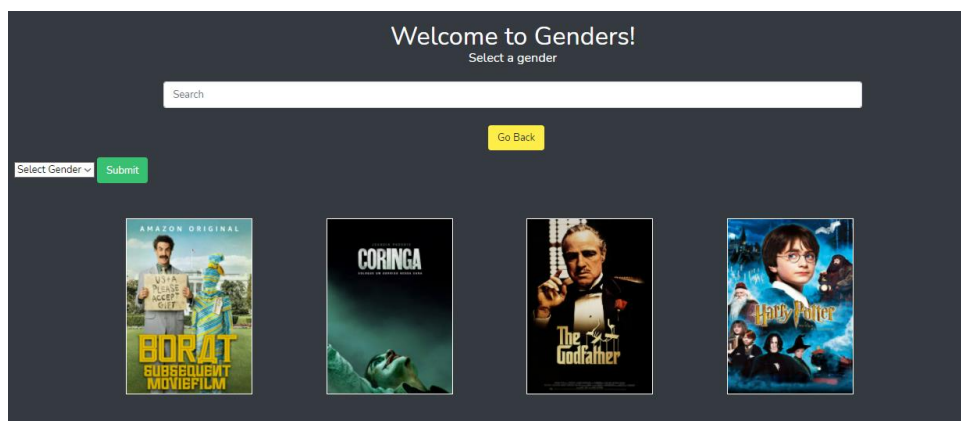
class GenderController extends Controller
{
    public function genderMenu(Request $request)
    {
        $genders = Gender::with('movies')->get(); // vai buscar os generos
        $generoEscolhido = $request->gender; //recebe o filme que o user está a procurar

        $movies = DB::table('gender_movies') //inner join do pivo com as outras duas
        ->join('movies', 'gender_movies.movie_id', '=', 'movies.id')
        ->join('genders', 'gender_movies.gender_id', '=', 'genders.id')
        ->select('movies.*') // vai buscar todos os campos do movies como no sql
        ->where('genders.id','=', "$generoEscolhido")
        ->get();

        $showUp=false;
        if ($request->submit && count($movies) == 0) {
            session()->flash('erroGen','That gender is empty');
            $showUp=true;
            return view('genderSearch',['genders' => $genders,'movies' => $movies,'showUp' => $showUp]);
        }
        return view('genderSearch',['genders' => $genders,'movies' => $movies,'showUp' => $showUp]);
    }
}

```

Este controlador é utilizado para a pesquisa por genero, inicialmente vai procurar os generos que existem na base de dados para depois o utilizador poder escolher, de forma a que posso ver quais os filmes e o seu tipo de genero fiz 2 inner joins de forma a juntar os filmes e os generos com a sua tabela pivo(gender\_ movies) e depois envio os resultados para a view.



## ContactController

Este controlador é utilizado caso o utilizador pretenda mandar uma mensagem ao administrador, ao ir a pagina de contacto o utilizador tem de preencher um formulario com o seu nome,email e o assunto, depois ao carregar enviar, o backend automaticamente envia a informação para a base de dados e fica gravado, caso o admin queira visualizar a informação pode ver diretamente através da BD e se pretender também pode apagar pela BD.

```

class ContactController extends Controller
{
    public function contact()
    {
        return view('contactPage');
    }

    public function validationRules(Request $request)
    {
        $validated = $request->validate([
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'email', 'max:255'],
            'description' => ['required', 'max:255']
        ]);
    }

    public function contactMessage(Request $request)
    {
        $validation = new ContactController();
        $validation->validationRules($request);

        $contactMessage = new Contact();
        $contactMessage->name = $request->name;
        $contactMessage->email = $request->email;
        $contactMessage->description = $request->description;
        $contactMessage->update(['updated_at' => now()]);
        // dd($request);
        $contactMessage->save();

        return redirect('/movie')->with('Message','Your message has been sent!');
    }
}

```

## Middleware

Criei uma classe chamada CheckAdmin de forma a poder validar se é o admin que está logado o que se for o caso depois vai permitir acesso a informação que o utilizador normal não tem acesso, depois apenas usei o middleware default do laravel para verificar se o user está logado.

Primeiro criei tres tabelas na base de dados.

Tabela “roles” -> Guarda o nome dos roles disponiveis para utilização

Tabela “users” -> Guarda o nome de todos os utilizadores e outras informações.

Tabela “role\_users” -> Vai guardar a chave estrangeira de ambas as tabelas (“roles” e “user”) em que neste caso são os Ids.

Depois nos modelos foi criada uma relação entre ambas as tabelas.

```

class Role extends Model
{
    use HasFactory;
    public function users()
    {
        return $this->belongsToMany(User::class, 'role_users');
    }
}

public function roles()
{
    return $this->belongsToMany(Role::class, 'role_users');
}

```

E depois na classe é apenas necessario chamar a sua relação como podem verificar em baixo.

```

class CheckAdmin
{
    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure $next
     * @return mixed
     */
    public function handle(Request $request, Closure $next)
    {
        $userRole = Auth::user()->roles->pluck('name');
        if (!$userRole->contains('Admin')) {
            return redirect('/noPermission');
        } else {
            return $next($request);
        }
    }
}

```

Utilizei a função `pluck()` do laravel em que automaticamente procura no array o que é designado neste caso 'name'. Se verificar que o utilizador não tem o role de admin ele automaticamente vai redirecionar para a pagina onde indica que não tem permissão e caso o mesmo tenha permissão ele permite continuar.

## Eventos

Uma vez que sempre que um utilizador era criado ele automaticamente não ficava com qualquer tipo de role associado eu decidi criar um evento no laravel de forma a que sempre que o utilizador seja criado o evento dispara e automaticamente associa o role de utilizador a esse novo utilizador.

```

class UserCreatedEvent
{
    use Dispatchable, InteractsWithSockets, SerializesModels;
    public $user;

    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct(User $user)
    {
        $this->user = $user;
    }
}

class SetDefaultRoleListener
{
    /**
     * Create the event listener.
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Handle the event.
     *
     * @param UserCreatedEvent $event
     * @return void
     */
    public function handle(UserCreatedEvent $event)
    {
        dd($event->user);
        //
        $role = Role::where('name', 'User')->firstOrFail();
        $event->user->roles()->attach($role->id);
    }
}

```

Na classe `SetDefaultRoleListener` ele depois procura o role pelo nome através do modelo e em baixo associa esse role ao ID do novo utilizador.

## Rotas

Na definição das rotas apenas permiti que o utilizador que não tivesse conta pudesse ver quais os filmes disponíveis, para ver o resto da informação é necessário ter conta ou criar uma nova conta. No recurso `MovieController` as permissões foram definidas dentro do próprio recurso como podemos ver quando expliquei os controladores.

```

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

Route::get('/noPermission', [MovieController::class, 'noPermissions'])->name('noPermissions');
Route::get('movie/search', [MovieController::class, 'search'])->name('search');

Route::get('movie/contact', [ContactController::class, 'contact'])->name('contact');
Route::post('movie/contact', [ContactController::class, 'contactMessage']);

Route::resource('movie', MovieController::class);

Route::get('/genders', [GenderController::class, 'genderMenu'])->name('genders')->middleware('auth');
Route::get('/home', [HomeController::class, 'index'])->name('home')->middleware('auth');

Auth::routes();








```

```
$ php artisan route:list
```

| Domain | Method    | URI                | Name          | Action  | Middleware                                    |
|--------|-----------|--------------------|---------------|---|---|
|        | GET HEAD  | /                  |               | App\Http\Controllers\MovieController@index              | web   |
|        | GET HEAD  | api/user           |               | Closure   | api<br>auth:api                               |
|        | GET HEAD  | genders            | genders       | App\Http\Controllers\GenderController@genderMenu        | web<br>auth                                   |
|        | GET HEAD  | home               | home          | App\Http\Controllers\HomeController@index               | web<br>auth                                   |
|        | POST      | login              |               | App\Http\Controllers\Auth\LoginController@login         | web   |
|        | GET HEAD  | login              | login         | App\Http\Controllers\Auth\LoginController@showLoginForm | guest<br>web                                  |
|        | POST      | logout             | logout        | App\Http\Controllers\Auth\LoginController@logout        | guest<br>web                                  |
|        | GET HEAD  | movie              | movie.index   | App\Http\Controllers\MovieController@index              | web   |
|        | POST      | movie              | movie.store   | App\Http\Controllers\MovieController@store              | web<br>App\Http\Middleware\CheckAdmin         |
|        | GET HEAD  | movie/contact      | contact       | App\Http\Controllers\ContactController@contact          | auth<br>web                                   |
|        | POST      | movie/contact      |               | App\Http\Controllers\ContactController@contactMessage   | web   |
|        | GET HEAD  | movie/create       | movie.create  | App\Http\Controllers\MovieController@create             | web<br>App\Http\Middleware\CheckAdmin         |
|        | GET HEAD  | movie/search       | search        | App\Http\Controllers\MovieController@search             | auth<br>web                                   |
|        | DELETE    | movie/{movie}      | movie.destroy | App\Http\Controllers\MovieController@destroy            | auth<br>web<br>App\Http\Middleware\CheckAdmin |
|        | PUT PATCH | movie/{movie}      | movie.update  | App\Http\Controllers\MovieController@update             | auth<br>web<br>App\Http\Middleware\CheckAdmin |
|        | GET HEAD  | movie/{movie}      | movie.show    | App\Http\Controllers\MovieController@show               | auth<br>web                                   |
|        | GET HEAD  | movie/{movie}/edit | movie.edit    | App\Http\Controllers\MovieController@edit               | auth<br>web<br>App\Http\Middleware\CheckAdmin |
|        | GET HEAD  | noPermission       | noPermissions | App\Http\Controllers\MovieController@noPermissions      | auth<br>web                                   |

Aqui podemos ver um pouco melhor o que cada classe faz as suas acções e o seu middleware.

## Modelos

| Models  |   |
|---|---|
|  Contact.php     | U |
|  Gender.php      | U |
|  GenderMovie.php | U |
|  Movie.php       | U |
|  Role.php        | U |
|  RoleUser.php    | U |
|  User.php        | U |

Os modelos é o que permite acesso as tabelas que foram utilizadas.

Nos modelos Gender, Contact, User, Movie consta também as relações de muitos para muitos já explicadas na parte dos controladores e autenticação, as unicas alterações feitas nos controladores foi a retirada dos timestamps uma vez que não eram necessários.

# Migrações

```
2020_12_14_205650_contacts.php U

class Contacts extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('contacts', function (Blueprint $table) {
            $table->bigIncrements('id')->unique();
            $table->string('name');
            $table->string('email');
            $table->longText('description');
            $table->timestamp('created_at')->default(DB::raw('CURRENT_TIMESTAMP'));
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('contacts');
    }
}
```

- ➔ Criei o id como único, os restantes campos estão como string uma vez que vão receber texto, e na description coloquei como longText uma vez que se fosse string normal estava mais limitado a nível do tamanho do texto que podia receber, depois criei os timestamps com a query para a data atual, assim sempre que é criado um registo ele automaticamente introduz a data atual.

```
2020_12_14_205650_genders.php U

class Genders extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('genders', function (Blueprint $table) {
            $table->bigIncrements('id')->unique();
            $table->string('name');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('genders');
    }
}
```

- ➔ Novamente Id como único e uma vez que recebe o nome do género ficou como string

```

class Movies extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('movies', function (Blueprint $table) {
            $table->bigIncrements('id')->unique();
            $table->string('name');
            $table->integer('rating');
            $table->year('release_year');
            $table->string('movie_img');
            $table->longText('description');
            $table->string('movie_trailer');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('movies');
    }
}

```

- ➔ O id sempre como único para não existir repetições, depois o rating ficou como inteiro, o ano decidi colocar apenas como ano de forma a ter apenas 4 numeros , o resto uma vez que são texto que vai receber ficaram definidos como string, com a exceção da descrição que ficou como texto longo pra que tenha mais espaço.

```

class CreateGenderMoviesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('gender_movies', function (Blueprint $table) {
            $table->unsignedBigInteger('movie_id');
            $table->unsignedBigInteger('gender_id');

            $table->foreign('movie_id')
                ->references('id')
                ->on('movies')
                ->onDelete('cascade');

            $table->foreign('gender_id')
                ->references('id')
                ->on('genders')
                ->onDelete('cascade');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('gender_movies');
    }
}

```

- ➔ Nesta migração é onde vamos buscar as FK das outras tabelas, esta é uma tabela pivo em que recebeu os ids das outras tabelas e é a tabela que origina de uma relação de muitos para muitos.

2020\_12\_22\_144234\_create\_role\_users\_tabl...

```
class CreateRoleUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('role_users', function (Blueprint $table) {
            $table->unsignedBigInteger('user_id');
            $table->unsignedBigInteger('role_id');

            $table->foreign('user_id')
                ->references('id')
                ->on('users')
                ->onDelete('cascade');

            $table->foreign('role_id')
                ->references('id')
                ->on('roles')
                ->onDelete('cascade');
        });
    }

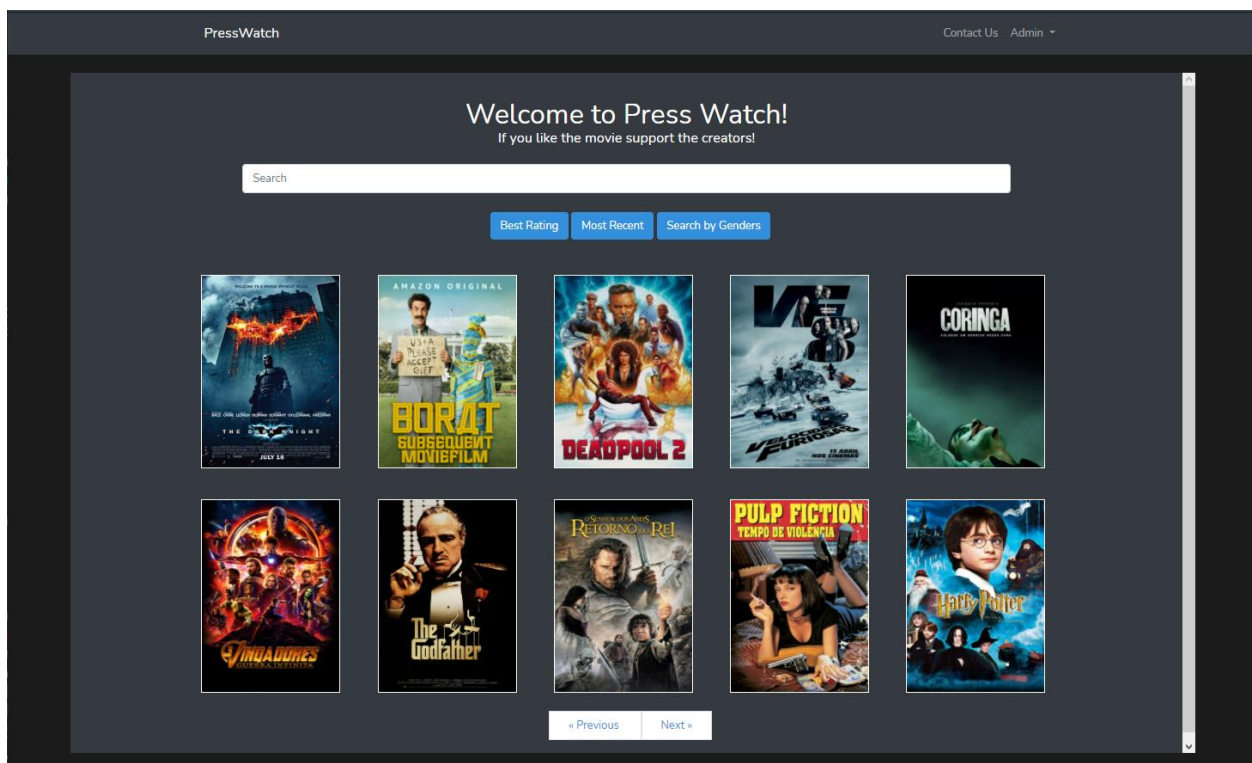
    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('role_user');
    }
}
```

➔ É exatamente a mesma situação que a migração anterior cria as FK.

## Views e Front End

A pagina principal:

mainPage.blade.php





A pagina de criação:

createMovie.blade.php

PressWatch

Contact Us Admin

Insert a new movie

Movie Name

Rating

Release Year

Choose File

No file chosen

Movie Trailer Link

Description

Select Gender

Submit

A pagina de edição:

editMovie.blade.php

PressWatch

Contact Us Admin

Your are currently editing

teste

Movie Name

Rating

Release Year

Choose File

No file chosen

Movie Trailer Link

Description

Select Gender

Submit

A pagina de pesquisa:

search.blade.php

PressWatch


Contact Us Admin

Is this what your we're looking for?

If you like the movie support the creators!

Search

Go Back



A pagina de contacto:

contactPage.blade.php

U

PressWatch

Contact Us Admin

CONTACT US

Wanna be able to upload? Send us a message!

Your Name

Name

Your Email

Email@example.com

Why do you want to contact us?

Enter your message

Are you sure?

Submit

CONTACT DETAILS


+351 965 400 55

siterandom.com

Artur\_pereira

SOCIAL

Our CEO



Alfredo Esdrubal

A pagina dos detalhes:

movieDetails.blade.php

U

PressWatch

Contact Us Admin

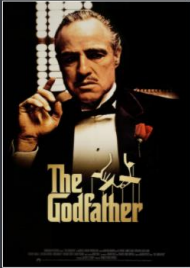
Your selected movie

Search

Edit Movie

Delete Movie

Go Back



O Padrinho

Description:


Action Drama - (1972)

O patriarca de uma dinastia de crime organizado transfere o controlo do crime do seu império clandestino para seu filho relutante.

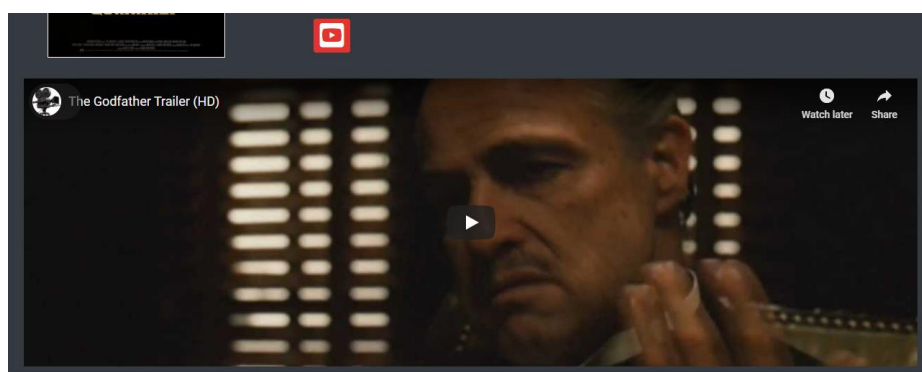
Rating:

★★★★★★★★

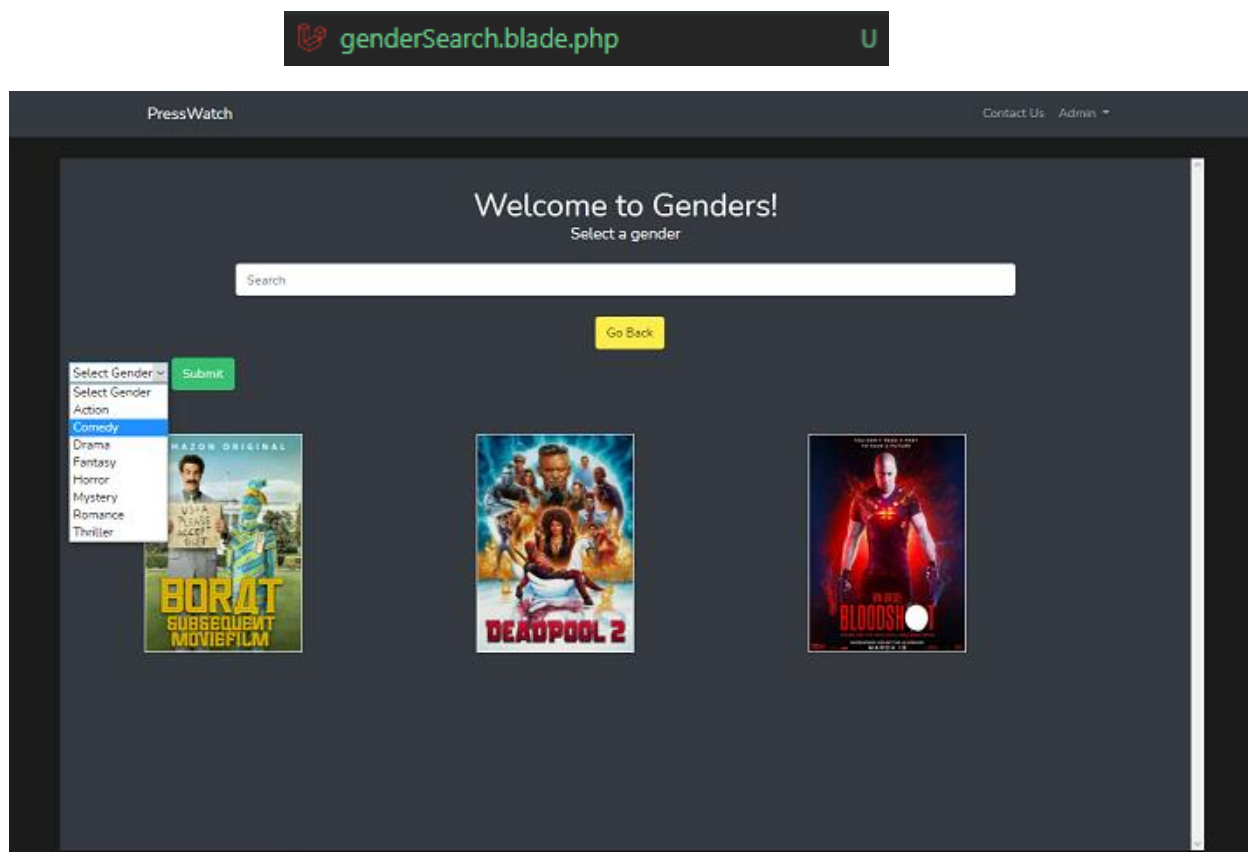
Wanna watch the trailer?



➔ Se carregar no botao de youtube vai mostrar o player com o trailer



A pagina de procura por genero:



# Conclusão

---

Após a conclusão deste trabalho verifico que de forma a ser eficiente é necessário primeiro planejar a base de dados e as classes a utilizar de forma a ser eficiente, uma das vantagens que consegui verificar com o laravel é que a informação é abundante, basta pesquisar e tentar compreender a informação que conseguirmos visualizar na internet, inicialmente estava com algumas dificuldades no entanto depois de me acostumar mais ao framework consegui resolver todo o tipo de problemas e bugs que foram aparecendo.

Finalizo este relatório com a certeza de ter aumentado os meus conhecimentos e até de ter expandido os mesmos, uma vez que a grande parte do trabalho foi realizada sem utilizar as fichas que foram providenciadas na aula, isto foi uma questão de preferência uma vez que prefiro pesquisar a informação online e ajustar ao que necessesito em vez de estar simplesmente a copiar código, tenho desde já a agradecer ao professor Michael Silva por todos os conhecimentos que me transmitiu durante a realização deste trabalho e durante todo o semestre.