

# Moviewexpert

---

## Página web con Laravel

**Alicia de la Cruz Peris – Sheila Bravo Sánchez**

**Curso DAW**

MovieExpert es una interfaz web de cine donde podrás ver información sobre las películas de estreno y dispondrá de una comunidad donde se realizarán concursos de películas, cortometrajes,... que suban los usuarios.

## Indice

1.	Introducción .....	2
1.1.	Descripción del Proyecto.....	2
2.	Análisis de Requisitos .....	3
2.1.	Objetivos del Proyecto .....	3
2.2.	Tecnologías Utilizadas .....	3
3.	Desarrollo del proyecto.....	4
3.1.	Base de datos .....	4
3.1.1.	Modelo Entidad Relación .....	4
3.1.2.	Modelo Relacional.....	4
3.2.	Base de datos .....	5
3.2.1.	Diagrama Casos de Usos .....	5
3.2.2.	DESCRIPCIÓN DE MODULOS .....	6
4.	Diagrama de Gantt .....	7
5.	Guía de estilos de MoviExpert .....	7
6.	Desarrollo de la Página Web. ....	14
6.1.	Instalación Laravel.....	14
6.2.	Elaboración de la página web .....	17
7.	Conclusión .....	55
8.	Propuestas de ampliación o mejoras.....	55
9.	Webgrafía .....	55

## **1. Introducción**

El Proyecto de Fin de Curso del Ciclo Formativo de Grado Superior Desarrollo de Aplicaciones Web cuyo título es MovieExpert, es una interfaz web de cine donde podrás ver información sobre las películas de estreno y dispondrá de una comunidad donde se realizaran. Este proyecto será desarrollado por Alicia de la Cruz Peris y Sheila Bravo Sánchez alumnas del centro IES Augustóbriga en la promoción 2015/2017.

### **1.1. Descripción del Proyecto**

MovieExpert es una interfaz web de cine donde podrás ver información sobre las películas de estreno y dispondrá de una comunidad donde se realizarán concursos de películas, cortometrajes,.. que suban los usuarios.

Es una aplicación donde se actualizarán las películas de estreno con su descripción resumida, director, guionistas, actores principales... y donde se podrá ampliar la información de cada película más detalladamente y con comentarios de usuarios acerca de la película. También habrá una sección de concurso de cortometrajes, donde los usuarios podrán inscribirse al concurso si entran dentro del plazo, subiendo algún proyecto de cortometraje, o también podrán valorar el resto de cortometrajes, y darles una puntuación y añadir comentarios. El cortometraje que mayor puntuación tenga será el ganador.

El perfil de usuario se introducirá la experiencia laboral y la categoría (sí es director, cámara, guionista, director de fotografía, etc.) del usuario así como enlaces a sus proyectos externos a la aplicación, si sus proyectos están dentro de la aplicación se le podrá vincular a dichos proyectos.

Habrá un chat/foro que podrá iniciar un usuario que inicie un proyecto de cortometraje para poder interactuar con otros usuarios que estén interesados en participar en el proyecto, así este podrá ponerse en contacto y participar juntos en la creación del cortometraje cada uno participando según su especialidad, si uno está interesado y es cámara, será el cámara, otro será el guionista, y así hasta formar el equipo completo y al terminar podrán inscribirse al concurso.  
Sólo podrán inscribirse al concurso los usuarios que estén registrados y logueados en la aplicación, así como iniciar un chat.

## **2. Análisis de Requisitos**

### **2.1. Objetivos del Proyecto**

Crear una interfaz web (frontend y backend) relacionada con el cine. Donde se podrá buscar películas y ver su información, tráiler, ranking,... Esta página también dispondrá de una comunidad para usuarios registrados donde podrás conocer a personal técnico cinematográfico y poder desarrollar un proyecto cinematográfico juntos. Y se realizarán concursos de películas, cortometrajes,...

Hemos elegido este proyecto porque creemos necesario un apoyo en el sector cinematográfico para aquellos profesionales que necesiten desarrollar sus ideas, y no tienen los contactos suficientes para llevarlas a cabo. Además se incluyen concursos y críticas para impulsar y darse a conocer sus ideas de forma profesional con otras personas del sector. Creemos que esta aplicación puede servir para iniciarlos profesionalmente en el cine, televisión, web series.

### **2.2. Tecnologías Utilizadas**

Lenguaje:

- HTML
- CSS
- PHP
- JavaScript

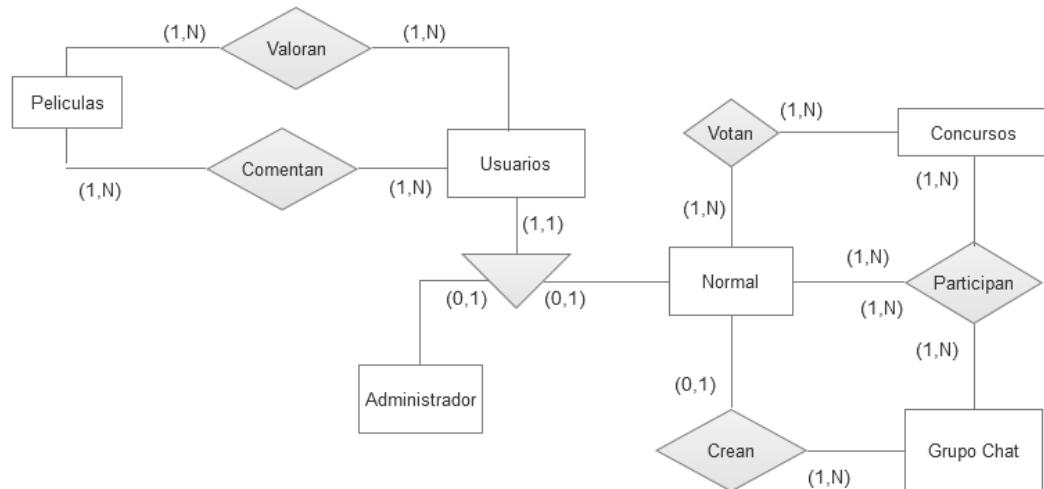
Herramientas:

- Bootstrap
- JQuery
- Framework: Laravel

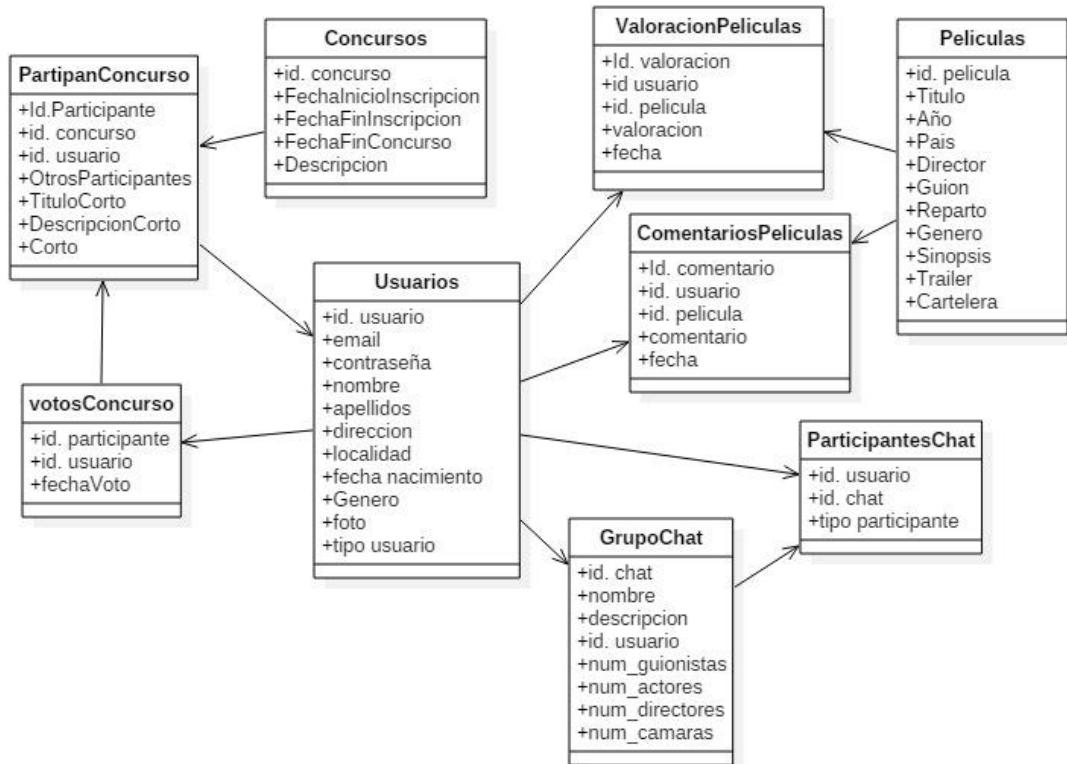
### 3. Desarrollo del proyecto

#### 3.1. Base de datos

##### 3.1.1. Modelo Entidad Relación

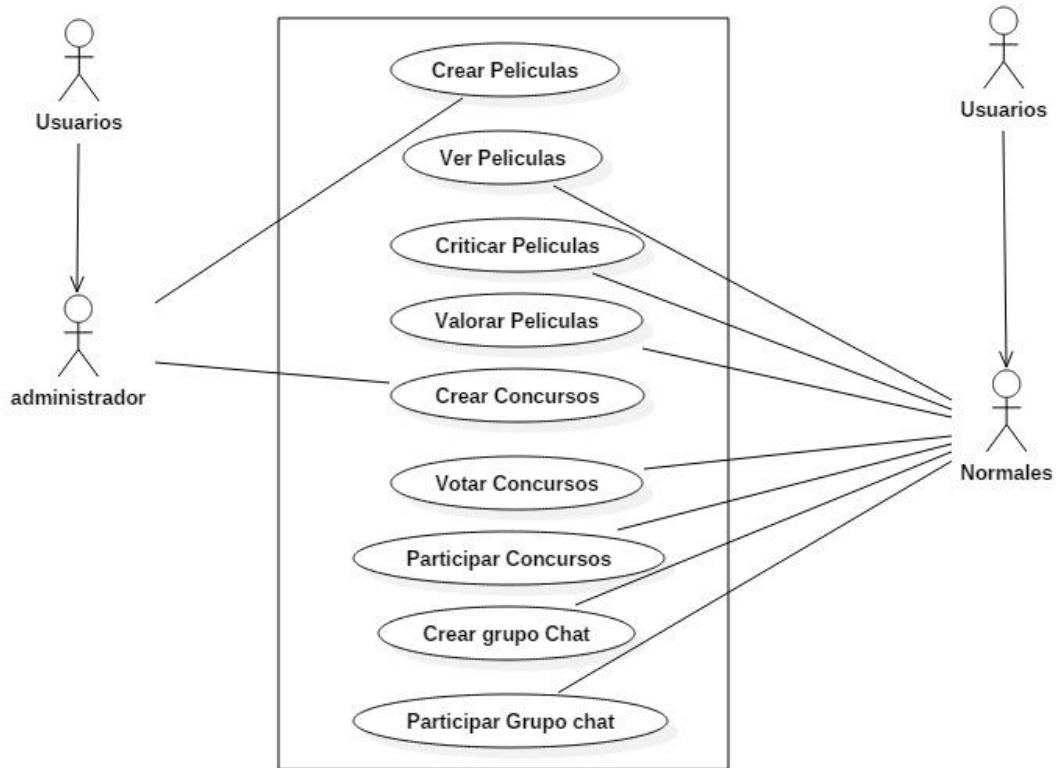


##### 3.1.2. Modelo Relacional



## 3.2. Base de datos

### 3.2.1. Diagrama Casos de Usos

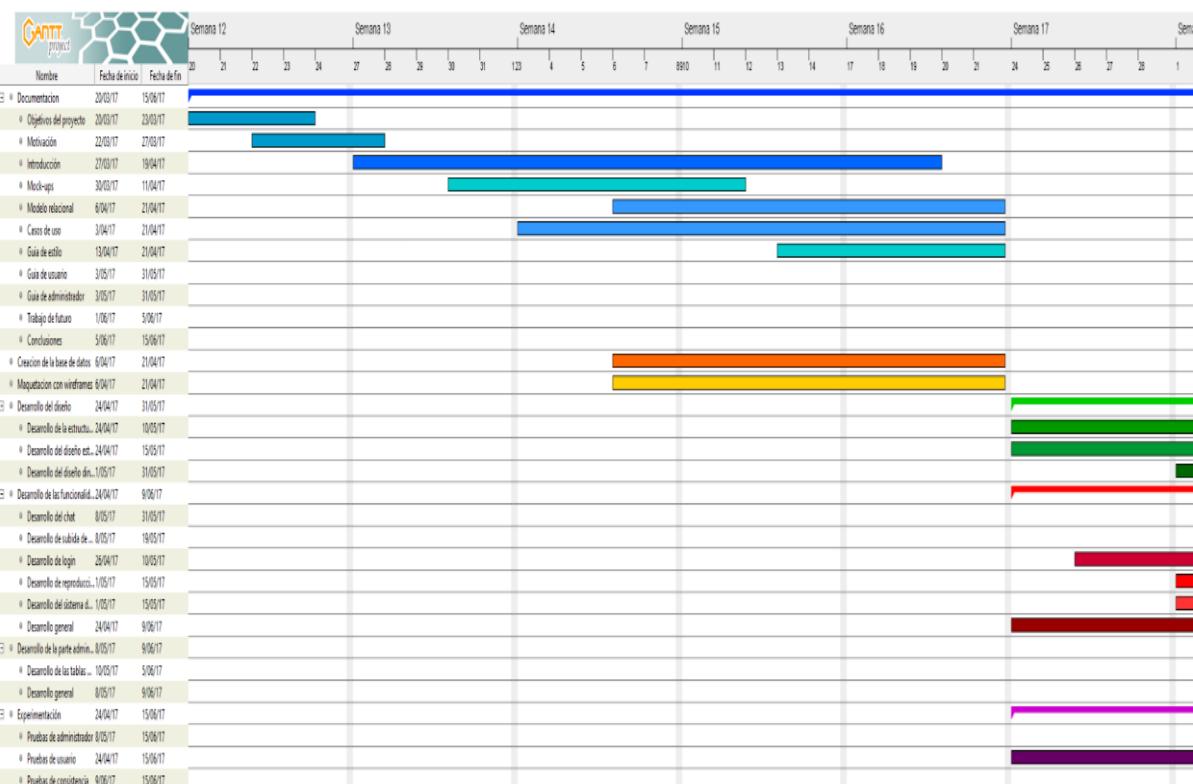


### 3.2.2. DESCRIPCIÓN DE MODULOS

- **Frontend:** Sección de la aplicación web con funcionalidad limitada para el uso de los usuarios normales o usuarios anónimos que puedan visitar la página. Solo se ofrecerá información sobre las películas para los usuarios anónimos o no registrados, mientras que a los usuarios normales o registrados además de ver la información de las películas podrán criticarlas y valorarlas. También tendrán acceso a otras dos secciones dentro de la página, a la sección de concursos y a la sección del chat.
- **Backend:** Sección de la aplicación web con funcionalidad completa de administrador para manipular toda la información de la base de datos, gestionar usuarios, películas, concursos,...
- **Módulos MVC:** Estos módulos separan los datos y la lógica de una aplicación de la interfaz de usuario y el modulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.
  - Modelos: Gestiona todos los accesos a la información con la que opera el sistema, tanto consultas como actualizaciones, implementando los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación. Envía a la vista aquella parte de la información en cada momento se le solicite para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al ‘modelo’ a través del ‘controlador’.
    - **Nuestros Modelos**
    - **Nuestros Modelos**
  - Controlador: Responde a eventos e invoca peticiones al ‘modelo’ cuando se le hace alguna solicitud. También tiene que enviar comandos a su ‘vista’ asociada si se solicita un cambio en la forma en que se presenta el ‘modelo’. El controlador hace de intermediario entre la ‘vista’ y el ‘modelo’.
    - **Nuestros Controladores**
    - **Nuestros Controladores**
  - Vistas: Presenta el ‘modelo’ en un formato adecuado para interactuar, normalmente la interfaz de usuario.
    - **Nuestras Vistas**
    - **Nuestras Vistas**

## 4. Diagrama de Gantt

El diagrama de Gantt es una herramienta para **planificar y programar tareas** a lo largo de un período determinado. Gracias a una fácil y cómoda visualización de las acciones previstas, permite realizar el seguimiento y control del progreso de cada una de las etapas de un proyecto y, además, reproduce gráficamente las tareas, su duración y secuencia, además del calendario general del proyecto.



## 5. Guía de estilos de MoviExpert

### Introducción

Una guía de estilo es un documento que establece el diseño a nivel de formato de una página web. En ella se establecen las directrices comunes para el estilo de textos e imágenes, el uso de colores y fuentes, las variaciones de logotipo que se pueden utilizar, etc.

Es un documento que sirve para que los desarrolladores implicados en diversas partes de la página web, puedan seguir delante de forma que sus desarrollos sean consistentes entre sí. Por lo tanto, es muy importante en proyectos en los que va a participar más de una persona.

Los puntos que se tratarán en esta guía de estilo serán los siguientes:

## **1. Objetivos de este documento**

En este punto se tratará cual es el objetivo del documento, se definirá el concepto de MoviExpert para entender a fondo el contexto de la página web y el diseño.

## **2. Usabilidad**

En este apartado se definirá a qué tipo de público está dirigida la página y que facilidades debe dar al usuario.

## **3. Accesibilidad**

La accesibilidad de página web se refiere al uso de ésta por parte de personas discapacitadas principalmente.

## **4. Tipografía y texto**

Se refiere al tipo y tamaño de letra, en toda la página, definiendo los títulos, subtítulos, descripciones, formulario, cabecera, pie, etc. Además también incluye el formato del texto, el espaciado etc.

## **5. Paleta de colores**

Aquí se definirá la gama de colores presente en la página web, los colores que deben tener los títulos, las imágenes, la letra, las secciones, etc.

## **6. Cabecera y pie**

Se define el estilo y diseño de la cabecera y pie de página.

## **7. Logo y fotos**

Se indica el logo a utilizar y el tipo de imágenes que se deberán usar, indicando tamaños, dimensiones, etc.

## **8. Iconografía**

Se mostrarán la gama de iconos que utiliza la web.

## **9. Elementos de interfaz**

Se indican el tipo de botones a utilizar, el diseño de elementos gráficos, etc.

Una vez vista cual es la estructura del documento, se puede comenzar a desglosar correspondientemente.

### **• Objetivos de este documento**

Con este documento se pretende hacer una guía para los desarrolladores que deseen realizar mejoras a la página web, puedan hacerlo sin crear inconsistencias. La página web que se va a desarrollar tiene por nombre “MoviExpert”. Es una página de estrenos de cine. Por lo tanto, la página web debe tener un aspecto sencillo e informativo, donde lo importante es el contenido. Porque los usuarios quieren información acerca de las películas, las opiniones de otros usuarios para decidir si ver la película o no. Aparte es una página que sirve para que gente del sector audiovisual, que no se conocen con anterioridad, puedan conocerse y realizar proyectos juntos,

participando de esta forma, en concursos internos de la página. El estilo y el diseño juegan una parte muy importante a la hora de mostrar una interfaz “user-friendly” o amigable.

- **Usabilidad**

La utilización de esta página web está dedicada a personas de todas las edades que les guste el mundo del cine. Esto implica diversos estilos de diseño:

- Debe ser una página sencilla.

- Debe ser confiable y seguro ya que los usuarios podrán acceder al perfil de otros usuarios para poder ver sus trabajos realizados con anterioridad.

- Debe ser fácil de usar, de forma que el usuario sepa en cada momento dónde está.

- **Accesibilidad**

La accesibilidad de esta web, no dispone, actualmente, de facilidades para personas ciegas. La página podrá verse a través de dispositivos móviles o tablets. Ya que es una página de información donde también podrás ver proyectos realizados de otros usuarios.

- **Tipografía y texto**

La tipografía utilizada en la página web estará predominada por dos fuentes:

- La fuente “Arial”: esta fuente asegura la legibilidad en cualquier navegador.

- La fuente “Lucida Calligraphy”: esta fuente da un aspecto elegante y confiable a la página web. Se utilizará principalmente en títulos y subtítulos.

Letra “Arial”: **Arial, Arial, Arial.**

Letra “Lucida Calligraphy”: ***Lucída, Lucída, Lucída.***

El tamaño de la letra dependerá del contexto utilizado.

- Títulos: 26 píxeles.
- Subtítulos: 18 píxeles.
- General y otros: 12 píxeles.
- El color del texto será predominantemente negro. Se incluirán en algunos detalles el rojo.
- La cabecera se tratará como títulos y subtítulos.
- Los elementos del menú y el pie se tratarán como subtítulos.
- Los formularios y la página en general utilizarán principalmente subtítulos y texto general.
- El formato de texto será principalmente normal, aunque se incluirá para remarcar temas importantes la negrita y para dar un aspecto más elegante la cursiva.
- Se usará un espaciado interlineal de 1,15 píxeles principalmente. El espaciado entre las letras será normal, salvo en algunos títulos y subtítulos que se podrá utilizar un espaciado de hasta 5 píxeles.

- **Paleta de colores**

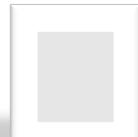
Los colores principales de la web deben definir el sentido de la web, por lo que los colores escogidos son:



#D69437



#000000



#CCCCCC

El negro será el color de la cabecera y de la letra del contenido de la página.

El rojo anaranjado será para el menú principal, y algunos títulos o subtítulos.

El gris para resaltar algo y hacerlo más llamativo.

El color blanco también es importante para los elementos del menú, el inicio de sesión y el registro.

- **Cabecera y pie**

La cabecera se situará el logo de la web a la izquierda, un buscador central, para hacer una búsqueda rápida de lo que se quiera encontrar, y el menú estará compuesto por cuatro enlaces inclinados hacia la izquierda y simétricos para dar un aspecto limpio y sencillo.

En la esquina de la derecha se situará el login y el registro del usuario, como es habitual en la mayoría de las páginas web.



La fuente utilizada es “arial” normal, el tamaño del menú son subtítulos (14px) y el de identificación tiene un tamaño algo más pequeño de 12px.

Y en cuanto al footer o pie de página tendrá otro estilo, será de color gris, con letra negra de un tamaño de 14px al igual que el del menú.

Preguntas más frecuentes | Política de privacidad / condiciones de uso  
© 2017 MovieExpert | All Rights Reserved - Todos los derechos reservados

- **Logo y fotos**

El logo se situará siempre en la cabecera. El color será los mismos que la página web, blanco, rojo anaranjado y gris, sobre fondo negro. El tamaño de este logo será siempre fijo y estará en la misma posición, es decir en la esquina superior izquierda, lo que permitirá seguir la lectura natural.



Las dimensiones del logo son: 387 x 67 píxeles.

En cuanto a las imágenes, se tienen 4 tipos de imágenes:

- **Imágenes de Inicio:**



Dimensiones: 145 x 170 píxeles.

Fondo: blanco.

Es la fotografía que sirve de enlace para conocer la información técnica de la película.

- **Imágenes de ficha técnica de la película:**



Dimensiones: 325 x 371 píxeles.

Fondo: blanco.

Es la imagen de la información detallada de la película.

- Foto de tráiler



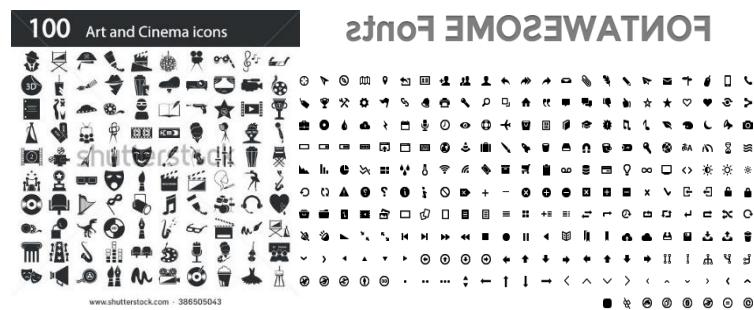
Dimensiones: 416 x 699 píxeles.

Fondo: blanco

Es donde se muestra el tráiler de la película.

- **Iconografía**

Los iconos a utilizar son los siguientes:



Estos iconos se usarán para los distintos motivos. Los colores serán siempre de la paleta de colores.

- **Elementos de interfaz**

Se entienden elementos de la interfaz como aquellos que permiten la interacción con el usuario, es decir, botones, aspectos gráficos y aquellas funciones que dan “movimiento” a la web.

Los botones deberán ser cuadrados con el fondo negro y las letras blancas. El texto será como los subtítulos. Cuando se pase el ratón por encima de una imagen de coche. Esto dará un aspecto moderno a la página.

Inscribirse al  
concurso

Algunos tendrán el borde redondeado.

- **Estructura de la web**

La estructura de la página web se estructura verticalmente, exponiendo la información de forma jerarquizada y explícita.

De esta manera, se consigue que el cliente pueda leer bien el contenido. La jerarquización de la información se basa en los contenidos más actualizados. La página estará continuamente actualizándose para mostrar los últimos estrenos cinematográficos. Esto implica que la visualización de la página debe ser limpia. En cuanto a las páginas de configuración y administración o formularios, deberán ser sencillos, de fácil usabilidad y que concuerden con la idea de la página web. La zona de trabajo será la central, dejando la parte superior y la inferior para una cabecera y un pie, respectivamente, estáticos.

- **Manual de administrador**

El administrador, cuando se loguee tendrá el menú igual que el resto de usuarios pudiendo ver lo que ven los usuarios, pero también podrá acceder a su propio menú pudiendo gestionar el contenido de la página web.



Usuarios



Estrenos



Concurso



Correo

Este será su propio menú. Los usuarios podrá ver todos los usuarios que están en la base de datos, y podrá eliminarlos si lo necesita. En estrenos, están las listas de las películas, podrá añadir nuevas películas, modificarlas y eliminarlas.

En concurso, el administrador puede añadir un nuevo concurso para que los usuarios puedan inscribirse, y por último el correo, donde podrá interactuar directamente con los usuarios, y resolverles las dudas que tengan.

## 6. Desarrollo de la Página Web.

Para desarrollar el proyecto vamos a utilizar un framework MVC (Modelo Vista Controlador) denominado Laravel, este framework permite el uso de una sintaxis elegante y expresiva para crear código de forma sencilla y permitiendo multitud de funcionalidades. Intenta aprovechar lo mejor de otros frameworks y aprovechar las características de las últimas versiones de PHP. Gran parte de Laravel está formado por dependencias, especialmente de Symfony.

El MVC es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador.

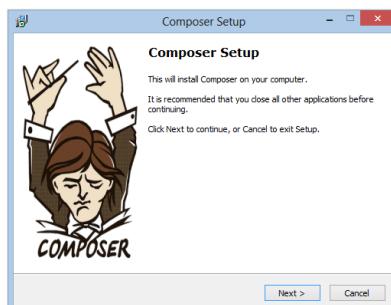
Previamente antes de instalar Laravel y empezar a programar nuestra página web necesitamos un servidor web y un servidor de base de datos, para el desarrollo del mismo vamos a trabajar con un servidor local llamado xampp.

Como el desarrollo del proyecto lo vamos a desarrollar entre dos personas para sincronizar el proyecto y poder programar a la vez vamos a utilizar una plataforma denominada github. Para poder utilizar esta herramienta en local y sincronizarlo a la plataforma en la red debemos tener instalado y configurado git en nuestro directorio del proyecto. Así podemos guardar los cambios realizados en el proyecto y poder sincronizarlo con nuestra compañera.

### 6.1. Instalación Laravel

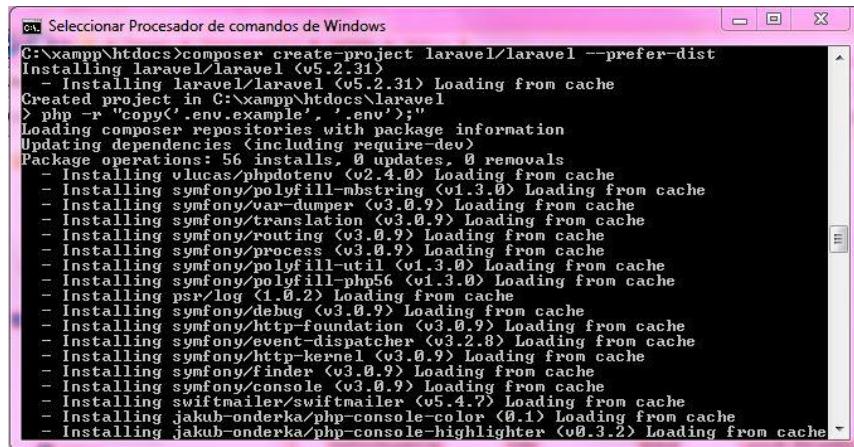
Para instalar Laravel necesitamos una herramienta conocida como Composer que se encarga de gestionar las dependencias en PHP. Composer te permite declarar las librerías de las cuales tu proyecto depende o necesita y las instala por ti.

Para descargar Composer vamos a la página oficial y descargamos el paquete de instalación. Ejecutamos el instalador descargado y seguimos la guía para configurar la instalación del paquete.



Una vez tenemos instalado Composer abrimos la consola de comando (CMD) nos situamos en la ruta donde queremos crear el proyecto, en nuestro caso dentro del servidor web en la ruta c:/xampp/htdocs y ejecutamos el siguiente comando para crear el proyecto:

### Composer create-project Laravel/Laravel --prefer-dist

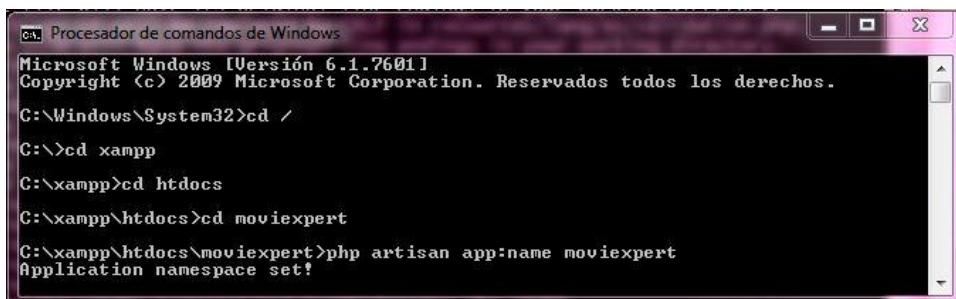


```
C:\ Seleccionar Procesador de comandos de Windows
C:\xampp\htdocs>composer create-project laravel/laravel --prefer-dist
Installing laravel/laravel (v5.2.31)
- Installing laravel/laravel (v5.2.31) Loading from cache
Created project in C:\xampp\htdocs\laravel
> php -r "copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 56 installs, 0 updates, 0 removals
- Installing vlucas/phpdotenv (v2.4.0) Loading from cache
- Installing symfony/polyfill-mbstring (v1.3.0) Loading from cache
- Installing symfony/var-dumper (v3.0.9) Loading from cache
- Installing symfony/translation (v3.0.9) Loading from cache
- Installing symfony/routing (v3.0.9) Loading from cache
- Installing symfony/process (v3.0.9) Loading from cache
- Installing symfony/polyfill-util (v1.3.0) Loading from cache
- Installing symfony/polyfill-php56 (v1.3.0) Loading from cache
- Installing psr/log (1.0.2) Loading from cache
- Installing symfony/debug (v3.0.9) Loading from cache
- Installing symfony/http-foundation (v3.0.9) Loading from cache
- Installing symfony/event-dispatcher (v3.2.8) Loading from cache
- Installing symfony/http-kernel (v3.0.9) Loading from cache
- Installing symfony/finder (v3.0.9) Loading from cache
- Installing symfony/console (v3.0.9) Loading from cache
- Installing swiftmailer/swiftmailer (v5.4.2) Loading from cache
- Installing jakub-onderka/php-console-color (0.1) Loading from cache
- Installing jakub-onderka/php-console-highlighter (v0.3.2) Loading from cache
```

El siguiente paso antes de empezar a programar será configurar correctamente nuestro proyecto:

- Nombre del proyecto: vamos a darle a nuestro proyecto el nombre de moviexpert para ello desde el CMD ejecutamos el comando :

**Php artisan app:name moviexpert**



```
C:\ Procesador de comandos de Windows
Microsoft Windows [Versión 6.1.7601]
Copyright © 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Windows\System32>cd /
C:\>cd xampp
C:\xampp>cd htdocs
C:\xampp\htdocs>cd moviexpert
C:\xampp\htdocs\moviexpert>php artisan app:name moviexpert
Application namespace set!
```

Comprobamos que se ha cambiado el nombre del proyecto abriendo cualquier fichero del proyecto y fijándonos en la primera línea donde hace referencia al namespace al que pertenece.



```
1 <?php
2
3 namespace moviexpert\Http\Controllers;
4
5 use Illuminate\Foundation\Bus\DispatchesJobs;
6 use Illuminate\Routing\Controller as BaseController;
7 use Illuminate\Foundation\Validation\ValidatesRequests;
8 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
9 use Illuminate\Foundation\Auth\Access\AuthorizesResources;
10
11 class Controller extends BaseController
12 {
13     use AuthorizesRequests, AuthorizesResources, DispatchesJobs, ValidatesRequests;
14 }
15
```

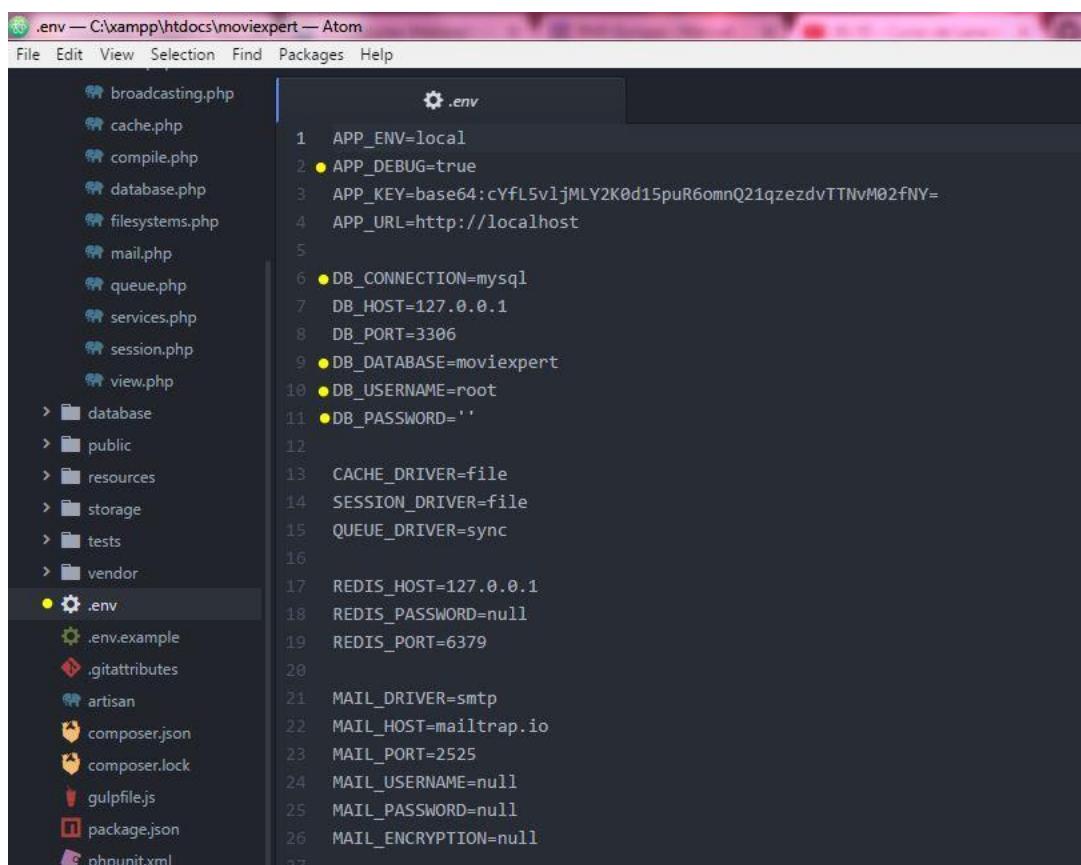
- Idioma del proyecto: por defecto viene configurado en inglés pero se puede cambiar para ello debemos editar el fichero **/moviexpert/config/app.php**, modificamos las siguientes líneas:

- Línea 55: '**timezon**' => '**Europe/Madrid**'
- Línea 67: '**locale**' => '**es**'

\*\*Previamente antes de modificar este fichero debemos hacer dos pasos:

- 1.- Tener la carpeta del idioma que queremos configurar en la ruta **/moviexpert/resources/lang**, sino es así debemos descargar el paquete con ese idioma y colocar el directorio del idioma en la ruta mencionada anteriormente.
- 2.- Buscar en google el phptimezone para ver los datos de tu región.

- Gestor de la base de datos: podemos ver la configuración de la base de datos en el fichero **/moviexpert/config/databases.php**, pero el fichero que debemos modificar para especificar los valores a las variables de conexión de la base de datos, para ello debemos modificar el fichero **/moviexpert/.env** y pasarle los datos de conexión de nuestra base de datos.



The screenshot shows the Atom code editor with the file `.env` open. The left sidebar lists various PHP files and configuration files like `.env.example`, `.gitattributes`, and `artisan`. The right pane displays the contents of the `.env` file, which contains environment variables for a Laravel application. The variables include `APP_ENV=local`, `APP_DEBUG=true`, `APP_KEY=base64:cYfL5vljMLY2K0d15puR6omnQ21qzezdTTNvM02fNY=`, `APP_URL=http://localhost`, `DB_CONNECTION=mysql`, `DB_HOST=127.0.0.1`, `DB_PORT=3306`, `DB_DATABASE=moviexpert`, `DB_USERNAME=root`, `DB_PASSWORD=''`, `CACHE_DRIVER=file`, `SESSION_DRIVER=file`, `QUEUE_DRIVER=sync`, `REDIS_HOST=127.0.0.1`, `REDIS_PASSWORD=null`, `REDIS_PORT=6379`, `MAIL_DRIVER=smtp`, `MAIL_HOST=mailtrap.io`, `MAIL_PORT=2525`, `MAIL_USERNAME=null`, `MAIL_PASSWORD=null`, and `MAIL_ENCRYPTION=null`.

```

.env — C:\xampp\htdocs\moviexpert — Atom
File Edit View Selection Find Packages Help
broadcasting.php
cache.php
compile.php
database.php
filesystems.php
mail.php
queue.php
services.php
session.php
view.php
> database
> public
> resources
> storage
> tests
> vendor
● .env
.env.example
.gitattributes
artisan
composer.json
composer.lock
gulpfile.js
package.json
phpunit.xml
1 APP_ENV=local
2 ● APP_DEBUG=true
3 APP_KEY=base64:cYfL5vljMLY2K0d15puR6omnQ21qzezdTTNvM02fNY=
4 APP_URL=http://localhost
5
6 ● DB_CONNECTION=mysql
7 DB_HOST=127.0.0.1
8 DB_PORT=3306
9 ● DB_DATABASE=moviexpert
10 ● DB_USERNAME=root
11 ● DB_PASSWORD=''
12
13 CACHE_DRIVER=file
14 SESSION_DRIVER=file
15 QUEUE_DRIVER=sync
16
17 REDIS_HOST=127.0.0.1
18 REDIS_PASSWORD=null
19 REDIS_PORT=6379
20
21 MAIL_DRIVER=smtp
22 MAIL_HOST=mailtrap.io
23 MAIL_PORT=2525
24 MAIL_USERNAME=null
25 MAIL_PASSWORD=null
26 MAIL_ENCRYPTION=null
27

```

Comprobamos que todo está bien instalado y configuramos, abrimos el navegador y escribimos en la barra de navegación `localhost:8000/` y cómo podemos apreciar se carga una página por defecto que instala Laravel en la instalación. Para poder verlo tenemos que tener arrancado nuestro servidor para ello en el CMD ejecutamos el siguiente comando:

**Php artisan serve**



```
C:\xampp\htdocs\moviexpert>php artisan serve
Laravel development server started on http://localhost:8000/
[Wed May 10 21:41:29 2017] ::1:59717 [200]: /favicon.ico
^C
C:\xampp\htdocs\moviexpert>
```

Abrimos el navegador y vamos a la ruta localhost:8000 y cómo podemos apreciar se ha instalado Laravel correctamente



Laravel 5

## 6.2. Elaboración de la página web

Empezamos el desarrollo de nuestra página web por el backend que es la parte de administración de la página, donde los usuarios con permiso de administrador pueden gestionar los usuarios, películas, géneros de películas, concursos y chat. Para ello creamos los controladores, modelos y vistas necesarios.

- **Controlador:** Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información. Se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.
- **Modelo:** Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.
- **Vista:** Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

Para crear los controladores abrimos la consola de comandos y ejecutamos el siguiente comando:

**Php artisan make:controller Nombre\_Controlador**

```
C:\xampp\htdocs\moviexpert>php artisan make:controller AdminpeliculaController
Controller created successfully.

C:\xampp\htdocs\moviexpert>php artisan make:controller AdmingeneroController
Controller created successfully.

C:\xampp\htdocs\moviexpert>php artisan make:controller AdminconcursoController
Controller created successfully.

C:\xampp\htdocs\moviexpert>php artisan make:controller AdminchatController
Controller created successfully.
```

Para crear los modelos seguimos los mismos pasos para crear los controladores, pero son el siguiente comando:

**Php artisan make:models Nombre\_Modelo -m**

```
C:\xampp\htdocs\moviexpert>php artisan make:model Adminpelicula -m
Model created successfully.
Created Migration: 2017_05_11_214703_create_adminpeliculas_table

C:\xampp\htdocs\moviexpert>php artisan make:model Admingenero -m
Model created successfully.
Created Migration: 2017_05_11_214716_create_admingeneros_table

C:\xampp\htdocs\moviexpert>php artisan make:model Adminconcurso -m
Model created successfully.
Created Migration: 2017_05_11_214727_create_adminconcursos_table

C:\xampp\htdocs\moviexpert>php artisan make:model Adminchat -m
Model created successfully.
Created Migration: 2017_05_11_214734_create_adminchats_table

C:\xampp\htdocs\moviexpert>
```

El parámetro **-m** se utiliza para que se nos creen los ficheros de migraciones a la base de datos. Estos ficheros tenemos que editarlos para indicar todos los campos que tendrá la tabla en la base de datos. Estos archivos se encuentran en la ruta **/moviexpert/database/migrations**

```
Project — C:\xampp\htdocs\moviexpert — Atom
File Edit View Selection Find Packages Help
Project
  moviexpert
    BASE DE DATOS Sheila entra
    .git
    app
    bootstrap
    config
    database
      factories
      migrations
        .gitkeep
        2014_10_12_000001_create_users_table.php
        2014_10_12_100000_create_password_resets_table.php
        2017_05_11_214703_create_admingeneros_table.php
        2017_05_11_214703_create_adminpeliculas_table.php
        2017_05_11_214727_create_adminchats_table.php
        2017_05_11_214727_create_adminconcursos_table.php
        seeds
        .gitignore
    public
    resources
    storage
    tests
    vendor
    .env
    .env.example
```

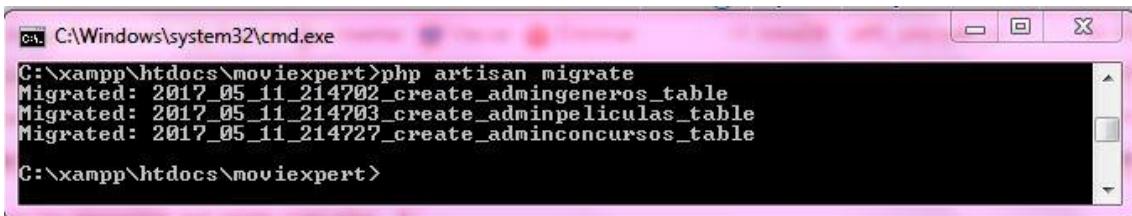
```
estilos.css UserController.php 2014_10_12_000001_create_users_table.php index.blade.php
```

```
use Illuminate\Database\Migrations\Migration;
class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('email')->unique();
            $table->string('password');
            $table->string('nombre');
            $table->string('apellidos');
            $table->string('direccion');
            $table->string('localidad');
            $table->string('genero');
            $table->date('fechanacimiento');
            $table->string('foto');
            $table->string('tipousuario');
            $table->rememberToken();
            $table->timestamps();
        });
    }
    /**

```

Una vez tenemos estos ficheros correctamente configurado pasamos a crear a las tablas en la base de datos. Abrimos el CMD y ejecutamos el siguiente comando:

**Php artisan migrate**



```
C:\Windows\system32\cmd.exe
C:\xampp\htdocs\moviexpert>php artisan migrate
Migrated: 2017_05_11_214702_create_admingeneros_table
Migrated: 2017_05_11_214703_create_adminpeliculas_table
Migrated: 2017_05_11_214727_create_adminconcursos_table
C:\xampp\htdocs\moviexpert>
```

Para comprobar que todo se ha creado correctamente vamos a nuestra base de datos.



Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
adminchats	Examinar Estructura Buscar Insertar Vaciar Eliminar	~0	InnoDB	utf8_unicode_ci	32 KB	
adminconcursos	Examinar Estructura Buscar Insertar Vaciar Eliminar	~0	InnoDB	utf8_unicode_ci	16 KB	
admingeneros	Examinar Estructura Buscar Insertar Vaciar Eliminar	~0	InnoDB	utf8_unicode_ci	16 KB	
adminpeliculas	Examinar Estructura Buscar Insertar Vaciar Eliminar	~0	InnoDB	utf8_unicode_ci	32 KB	
migrations	Examinar Estructura Buscar Insertar Vaciar Eliminar	~6	InnoDB	utf8_unicode_ci	16 KB	
password_resets	Examinar Estructura Buscar Insertar Vaciar Eliminar	~0	InnoDB	utf8_unicode_ci	16 KB	
users	Examinar Estructura Buscar Insertar Vaciar Eliminar	~0	InnoDB	utf8_unicode_ci	32 KB	
7 tablas	Número de filas	6	InnoDB	latin1_swedish_ci	160 KB	0

El siguiente paso será crear las vistas para ello en la ruta **/moviexpert/resources/view** creamos un directorio para cada controlador puesto que un mismo controlador va a estar asociado a varias vistas, este paso no es necesario es simplemente para tener más organizado el proyecto. Dentro de estos directorios empezamos a crear archivos para las vistas. Las vistas son ficheros que tiene el código de presentación. Por tanto, una vista será un archivo PHP que contendrá mayoritariamente código HTML, que se enviará al navegador para que este renderice la salida para el usuario.

Como todas las páginas tienen código en común como es la cabecera y el pie de página vamos a utilizar el motor de plantillas oficial de Laravel Blade. Con las plantillas blade de Laravel evitamos el uso de tags php, podemos crear una plantilla base e ir reutilizándola a medida que la necesitemos.

Para poder usar este motor de plantillas debemos crear nuestros archivos con la siguiente extensión “nombreVista.**blade.php**”, además si vamos a utilizar las funciones básicas definidas en Laravel con un enrutado resource los ficheros deberán ser llamados igual que la función (index.blade.php, create.blade.php, show.blade.php,...). Creamos una plantilla base que contendrá la cabecera y el pie de nuestra página web.

Creamos una plantilla para el backend la cual tendrá la cabecera y el pie. Esta plantilla va a ser invocada desde todas las vistas que queramos que herede esta cabecera y pie.

```

index.php          1 <!DOCTYPE html>
                  2 <html lang="en">
                  3 <head>
                  4   <meta charset="utf-8">
                  5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
                  6   <meta name="viewport" content="width=device-width, initial-scale=1">
                  7   <title>Backend</title>
                  8   {!!Html::style('css/bootstrap.min.css')!!}
                  9   {!!Html::style('css/metisMenu.min.css')!!}
                 10  {!!Html::style('css/estilos.css')!!}
                 11  {!!Html::style('css/font-awesome.min.css')!!}
                 12 </head>
                 13 <body>
                 14   <div id="wrapper">
                 15     <nav class="navbar navbar-default navbar-static-top" role="navigation">
                 16       <div class="navbar-header container col-xs-12 fondoCabecera height100">
                 17         <div class="col-xs-7 col-sm-5 col-sm-offset-0 col-md-5 col-md-offset-1 col-lg-5 col-lg-offset-1">
                 18           <a class="navabar-brand" href="/admin"></a>
                 19         </div>
                 20         <div class="marginTop25 hidden-xs col-sm-6 col-sm-offset-1 col-md-5 col-md-offset-1 col-lg-4 col-lg-offset-2">
                 21           <form class="navbar-form">
                 22             <div class="form-group">
                 23               <input type="text" class="form-control" placeholder="Texto a buscar">
                 24             </div>
                 25             <button type="submit" class="btn fondoMenu">Buscar</button>
                 26           </form>
                 27         </div>
                 28         <div class="hidden-sm hidden-md hidden-lg col-xs-1 col-xs-offset-1 marginTop25">
                 29           <a href="#" class="textoMenu glyphicon glyphicon-menu-hamburger"></a>
                 30         </div>
                 31       </div>
                 32     <div class="width100 fondoMenu">
                 33       <ul class="nav nav-pills container">
                 34         <li class=""><a class="textoMenu" href="/admin">Inicio</a></li>
                 35         <li class=""><a class="textoMenu" href="/adminpelicula">Peliculas</a></li>
                 36         <li class=""><a class="textoMenu" href="/adminconcurso">Concursos</a></li>
                 37         <li class=""><a class="textoMenu" href="/adminchat">Chats</a></li>
                 38         <li class=""><a class="textoMenu" href="/adminusuarios">Usuarios</a></li>
                 39         <li class=""><a class="textoMenu" href="/">Frontend</a></li>
                 40         <li class=""><a class="textoMenu" href="#">Cerrar Sesión</a></li>
                 41       </ul>
                 42     </div>
                 43   </nav>
                 44
                 45   <div id="page-wrapper">
                 46     @yield('content')
                 47   </div>
                 48 </div>
                 49 {!!Html::script('js/jquery.min.js')!!}
                 50 {!!Html::script('js/bootstrap.min.js')!!}
                 51 {!!Html::script('js/metisMenu.min.js')!!}
                 52 {!!Html::script('js/sb-admin-2.js')!!}
                 53
                 54 </body>
                 55 </html>

```

En la plantilla hay que indicarle en que parte de la estructura vamos a añadir el contenido específico de cada vista, para ello tenemos que definir una sección, para definirla usamos el siguiente código:

```

<div id="page-wrapper">
  @yield('content') En esta línea definimos la sección.
</div>

```

```

78   <div class="hidden-sm hidden-md hidden-lg col-xs-1 col-xs-offset-1 marginTop25 ">
79     <a href="#" class="textoMenu glyphicon glyphicon-menu-hamburger"></a>
80   </div>
81 </div>
82 <div class="width100 fondoMenu">
83   <ul class="nav nav-pills container">
84     <li class=""><a class="textoMenu" href="/admin">Inicio</a></li>
85     <li class=""><a class="textoMenu" href="/adminpelicula">Peliculas</a></li>
86     <li class=""><a class="textoMenu" href="/adminconcurso">Concursos</a></li>
87     <li class=""><a class="textoMenu" href="/adminchat">Chats</a></li>
88     <li class=""><a class="textoMenu" href="/adminusuarios">Usuarios</a></li>
89     <li class=""><a class="textoMenu" href="/">Frontend</a></li>
90     <li class=""><a class="textoMenu" href="#">Cerrar Sesión</a></li>
91   </ul>
92 </div>
93
94   <div id="page-wrapper">
95     @yield('content')
96   </div>
97 </div>
98 {!!Html::script('js/jquery.min.js')!!}
99 {!!Html::script('js/bootstrap.min.js')!!}
100 {!!Html::script('js/metisMenu.min.js')!!}
101 {!!Html::script('js/sb-admin-2.js')!!}
102
103 </body>
104 </html>

```

Después de crear las plantillas creamos las vistas que vamos a utilizar. Para que la vista herede la plantilla que contiene la cabecera y el pie utilizamos la siguiente sintaxis:

```

@extends('layouts.admin') --> La plantilla de la que va a heredar la estructura
@section('content')--> La sección de la plantilla donde incluimos el código de la vista
<div>
  <p>Contenido en HTML específico para la vista</p>
</div>
@endsection

```

```

index.php
robots.txt
web.config
resources
  assets
  lang
  views
    admin

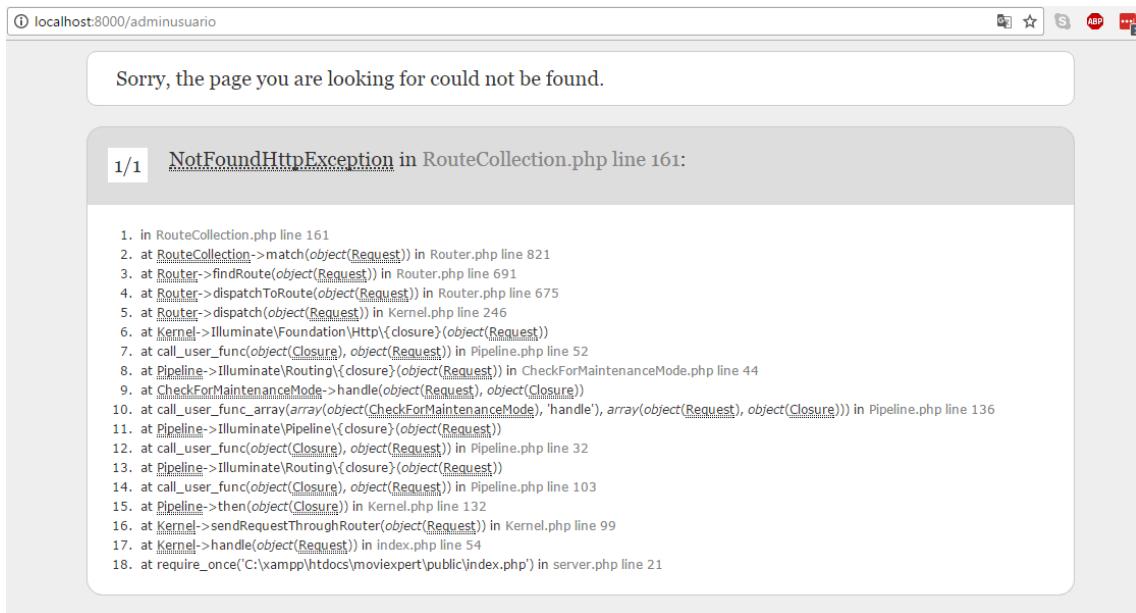
```

```

1 v @extends('layouts.admin')
2 v   @section('content')
3     <div class="col-lg-2 col-lg-offset-1 col-md-2 col-md-offset-2 col-sm-3 col-sm-offset-2 col-xs-6 col-xs-offset-3 margin-top25"><img class="wid
4       <div class="col-lg-2 col-lg-offset-2 col-md-2 col-md-offset-3 col-sm-3 col-sm-offset-2 col-xs-6 col-xs-offset-3 margin-top25"><img class="wid
5         <div class="col-lg-2 col-lg-offset-2 col-md-2 col-md-offset-2 col-sm-3 col-sm-offset-2 col-xs-6 col-xs-offset-3 margin-top25"><img class="wid
6           <div class="col-lg-2 col-lg-offset-5 col-md-2 col-md-offset-3 col-sm-3 col-sm-offset-2 col-xs-6 col-xs-offset-3 margin-top25"><img class="wid
7             @endsection
8

```

Una vez tenemos los tres componentes del MVC solo nos faltaría enlazarlos entre ellos para poder visualizarlo en el navegador. También hay que definir las rutas de navegación puesto que si no tenemos definidas las rutas aunque tengamos los tres componentes enlazados nos va a mostrar un error que nos indica que no está definida la ruta a la que queremos acceder.



Las rutas se definen en el fichero **/moviexpert/app/http/routes.php**. Existen dos formas de crear las rutas:

1. **Route::get('ruta','nombreControlador@funcion o función');** -> Lo usamos para una ruta concreta que va a mostrar una vista determinada. Podemos ir directamente a una vista estática que no vaya estar enlazada con ningún controlador ni modelo o podemos utilizar una función concreta del controlador definido y este retorne la vista.

Ejemplo:

```
Route::get('admin', 'HomeAdminController@index');
```

2. **Route::resource('ruta','nombre\_controlador');** -> Este método agiliza mucho las cosas a la hora de utilizar un controlador como intermediario puesto si un controlador va a tener varias vistas asociadas no hace falta crear un Route::get para cada una de ellas, sino que con Route::resource le indicamos el controlador y lo único que tenemos que hacer es crear la vista con el nombre de la función a la que se va a enlazar al controlador y el solo interpreta las rutas de las funciones básicas definidas en Laravel (index, create, store, edit, update, show y destroy).

Ejemplo:

```
Route::resource('adminusuarios','UserController');
```

```

1 <?php
2
3
4 Route::get('/', function () {
5     return view('welcome');
6 });
7 Route::get('admin', 'HomeAdminController@index');
8 Route::resource('adminpelicula', 'AdminpeliculaController');
9 Route::resource('admingenero', 'AdmingeneroController');
10 Route::resource('adminconcurso', 'AdminconcursoController');
11 Route::resource('adminchat', 'AdminchatController');
12 Route::resource('adminusuarios', 'UserController');
13

```

El siguiente paso sería enlazar los componentes para ello vamos realizamos las siguientes acciones:

1. Editamos el modelo para indicarle cuales son los campos de la tabla con los que vamos a trabajar, es decir, que campos va a llenar cuando hagamos una inserción de datos a esa tabla o los campos de los cuales nos va a devolver los datos cuando hagamos una consulta.

```

Project
  Admingenero.php
  Adminpelicula.php
  Controller.php
  Homeadmin.php
  UserController.php
  Middleware.php
  Requests.php
  Kernel.php
  routes.php
  Jobs.php
  Listeners.php
  Policies.php
  Providers.php
    Adminchat.php
    Adminconcurso.php
    Admingenero.php
    Adminpelicula.php
  User.php
    bootstrap.php
    config.php
    database.php
    public.php
    resources.php
    storage.php
    tests.php

estilos.cssUserController.phpadmin.blade.phpUser.php

1 <?php
2
3 namespace moviexpert;
4
5 use Illuminate\Foundation\Auth\User as Authenticatable;
6
7 class User extends Authenticatable
8 {
9     /**
10      * The attributes that are mass assignable.
11      *
12      * @var array
13      */
14     protected $fillable = [
15         'email', 'password', 'nombre', 'apellidos', 'direccion', 'localidad', 'genero', 'fechanacimiento', 'foto', 'tipousuario',
16     ];
17
18     /**
19      * The attributes that should be hidden for arrays.
20      *
21      * @var array
22      */
23     protected $hidden = [
24         'password', 'remember_token',
25     ];
26 }
27

```

2. Editamos el controlador y vamos a la función asociada a la vista que estamos creando y vamos hacer dos cosas, la primera guardar los datos que le solicitamos al modelo en una variable y en segundo lugar retornamos a la vista pasándola la variable con todo los datos.

```

Project
└── Admingen
    ├── AdminController.php
    ├── Controller.php
    └── HomeController.php
        ├── middleware
        ├── Requests
        ├── Kernel.php
        └── routes.php
    ├── Jobs
    ├── Listeners
    ├── Policies
    ├── Providers
    ├── Adminchat.php
    ├── Adminconcurso.php
    ├── Adminingenero.php
    ├── Adminpelicula.php
    └── User.php
        ├── bootstrap
        ├── config
        ├── database
        ├── public
        ├── resources
        ├── storage
        └── tests
estilos.css
UserController.php
└── admin.blade.php
└── index.blade.php

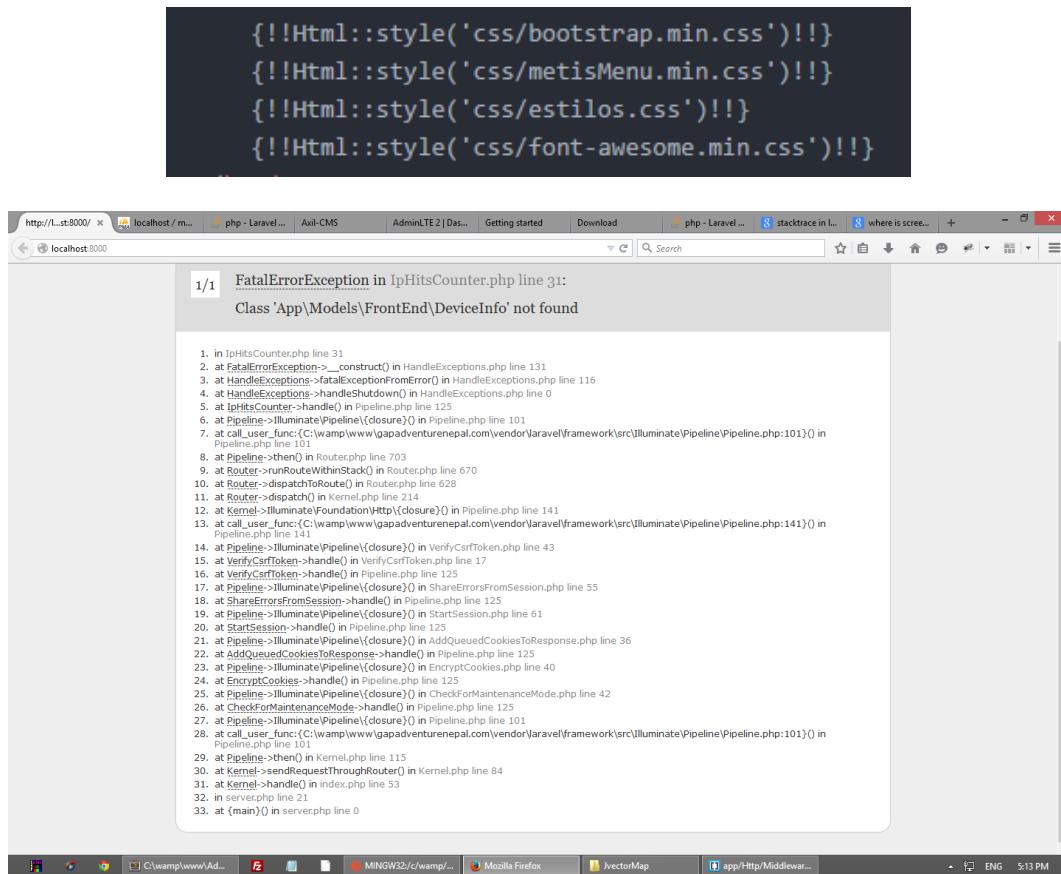
```

```

1 <?php
2
3 namespace moviexpert\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use moviexpert\Http\Requests;
7 use moviexpert\Http\Controllers\Controller;
8 use Session;
9 use Redirect;
10 use Illuminate\Routing\Route;
11 use moviexpert\User;
12 class UserController extends Controller
13 {
14     //
15     public function index(){
16         /*Creamos una variable para almacenar todos los datos de la base de datos*/
17         $users=moviexpert\User::all();
18         /*Retornamos a la vista user carpeta index vista y le pasamos la variable con los datos*/
19         return view('user.index',compact('users'));
20     }
21     public function create(){
22         /*Retornamos a la vista create*/
23         return view('user.create');
24     }
25     public function store(Request $request){
26         /*Esta función se ejecuta al pulsar el botón registrar del create y lo que haces dar de alta a un nuevo usuario en la base de datos*/
27         \moviexpert\User::create([
28             /*Nombre campo base datos -> nombre del campo del formulario*/
29

```

Vamos al navegador e introducimos la ruta localhost:8000/admin y nos debería mostrar la vista creada pero nos da un error que nos indica que no reconoce las etiquetas HTML y FORM, estas etiquetas las utilizamos porque Laravel tiene un etiquetado propio a la hora de crear formularios y enlaces.



Para solucionar este problema tenemos que instalar el LaravelCollective vamos a la página oficial de Laravel para seguir el manual y poder configurarlo.

## # Installation

Begin by installing this package through Composer. Run the following from the Terminal:

```
composer require "laravelcollective/html":'^5.2.0'
```

Next, add your new provider to the `providers` array of `config/app.php`:

```
'providers' => [
    // ...
    Collective\Html\HtmlServiceProvider::class,
    // ...
],
```

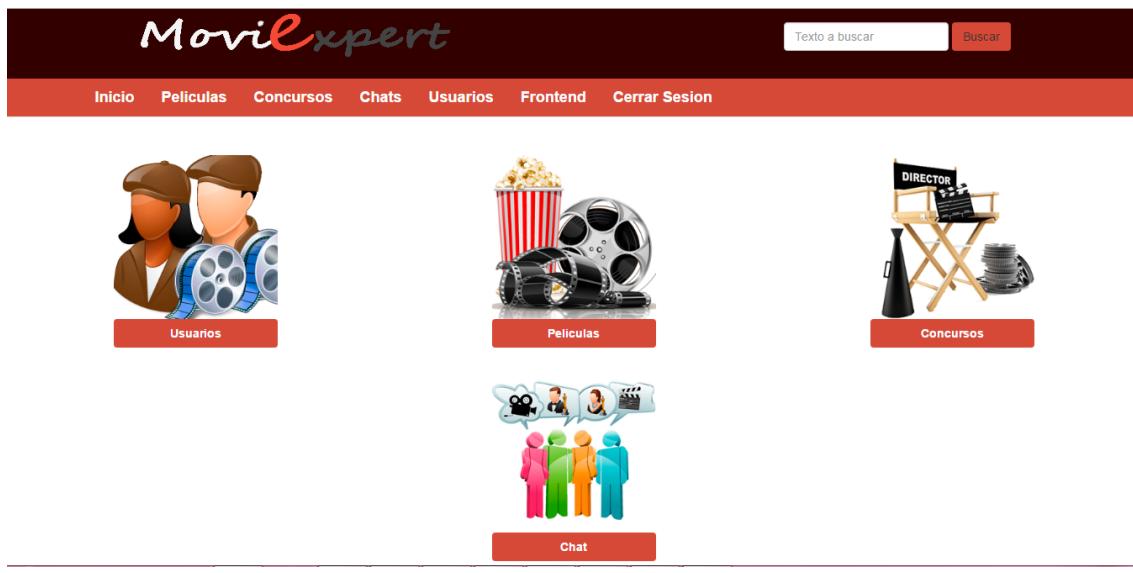
Finally, add two class aliases to the `aliases` array of `config/app.php`:

```
'aliases' => [
    // ...
    'Form' => Collective\Html\FormFacade::class,
    'Html' => Collective\Html\HtmlFacade::class,
    // ...
],
```

Tenemos las siguientes etiquetas para elaboración de formularios en LaravelCollective:

Nombre	Sintaxis
Etiquetas	Form::label
Campo texto	Form::text
Campo Email	Form::email
Campo Password	Form::password
Campo numérico	Form::number
Checkbox	Form::checkbox
Radiobutton	Form::radio
Select	Form::select
Select Rango Numerico	Form::selectrange
Select Nombre del mes	Form::selectmonth
Campo File (Archivos de entrada)	Form::file
Botones	Form::submit
Macros	Form::macro

Cuando tengamos configurado y funcionando el LaravelCollective volvemos al navegar y volvemos hacer la prueba y cómo podemos apreciar ya podemos visualizar la vista. Configuramos todas las vistas, modelos y controladores para todas las opciones que aparecen en el menú.



Para administrar los datos en el backend vamos a utilizar las funciones básicas que tiene definidas Laravel, explicamos cada una de ellas:

- **Index()** -> Esta función vamos a mostrar un listado con todo los datos de la tabla. Para ello hacemos una consulta a la base de datos para recuperar toda la información de esa tabla y la almacenamos en una variable, la cual se la enviamos a la vista.

```
public function index(){
    /*Creamos una variable para almacenar todos Los datos de la base de datos*/
    $participanchat=DB::table('miembrochats')->paginate(6);
    /*Retornamos a la vista user carpeta index vista y le pasamos la variable con los datos*/
    /*Retornamos a la vista user carpeta index vista y le pasamos la variable con los datos*/
    return view('participanchat.index',compact('participanchat'));
}
```

- **Create()**-> Esta función únicamente se utiliza para retornar a la vista que contiene el formulario para insertar datos en una tabla de la base de datos. También se pueden hacer consultas a la base de datos para recuperar otros datos que se quieran mostrar en la vista al igual que en la función del index. Para poder dar de alta en la base de datos esta función va encadenada a la siguiente función “**store**” ya que sin ella solo se mostraría el formulario no se haría la inserción.

```
public function create(){
    /*Creamos una variable donde almacenamos todos los miembro del chat
    Esta variable la vamos a utilizar en la vista para cargar Los datos del select de miembros*/
    $users=\movieexpert\miembrochat::lists('id','id');
    /*Retornanmos a La vista create y le pasamos todos Los miembros*/
    return view('adminmensajechat.create',compact('users'));
}
```

- **Store(request \$request)->** Esta función la utilizamos para hacer la inserción en la base de datos, en ella recogemos los datos enviados en el formulario **create** y ejecutamos la consulta para insertarlos en la base de datos. Esta función recibe dos parámetros el primero es el nombre del fichero **request** el cual contiene las reglas de formulario para validar los datos y el segundo parámetro es un array que contiene todos los datos enviados por el formulario. Una vez se ha hecho la inserción redireccionamos a la ruta deseada.

```
public function store(VotosConcursoRequest $request){
    \moviexpert\votosconcurso::create([
        /*Nombre campo base datos => nombre del campo del formulario*/
        'idcortoconcurso'=> $request['idcortoconcurso'],
        'idusuario'=>$request['idusuario'],
        'voto'=> $request['voto'],
        'fechavoto'=> $request['fechavoto'],
    ]);
    /* Redireccionamos a la ruta del index y indicamos que muestre un mensaje*/
    return redirect('adminvotosconcurso');
}
```

- **Edit(\$id)->** Esta función la utilizamos para retornar al formulario para editar un fila concreta de la base de datos. Al igual que **create** no modifica los campos de la base de datos sino que necesita a la función **update** para realizar la modificación en la base de datos. Esta función recibe un único parámetro que es el identificador de la fila que queremos modificar. Con este parámetro realizamos una consulta para buscar esa fila completa, almacenar sus datos en una variable y enviárselos a la vista.

```
public function edit($id){
    /*Buscamos la pelicula que queremos modificar para pasarsela a la vista*/
    $peliculas = Adminpelicula::find($id);
    /*Recuperamos el nombre del genero para mostrar su nombre y no su identificador*/
    $generos=\moviexpert\Admingenero::lists('genero','id');
    /*Retornamos al vista que contiene el formulario para editar la pelicula y le pasamos los datos*/
    return view('peliculas.edit',compact("generos"))->with('pelicula', $peliculas);
}
```

- **Update(request \$request \$id)->** Esta función es la que ejecuta la funcionalidad para poder realizar la actualización de los datos modificados en el formulario **edit** en la base de datos. Esta función recibe tres parámetros, el primero es el fichero **request** que contiene las reglas de validación de formulario, el segundo es la variable **\$request** que contiene todos los datos del formulario y el tercero es el identificador de la fila de la base de datos que queremos modificar.

```
public function update(ConcursoUpdateRequest $request,$id){
    $concursos = Adminconcurso::find($id);
    $concursos->fill($request->all());
    $concursos->save();
    Session::flash('message','concursos Actualizado Correctamente');
    return Redirect::to('/adminconcurso');
}
```

- **Show(\$id)->** Esta función la utilizamos para ver los datos de una fila concreta de la base de datos. Recibe un parámetro que es el identificador de la fila la cual queremos mostrar detalladamente o con una vista más personalizada a la del index. En esta función hacemos lo mismo que la función **edit**, buscamos la fila que queremos mostrar a través de su identificador y almacenamos todos sus datos en un array que le enviamos a la vista.

```
public function show($id){
    $concursos=$concursos = \moviexpert\AdminConcurso::find($id);
    $inscripcion = DB::table('participanconcursos')
        ->join('adminconcursos', 'adminconcursos.id', '=', 'participanconcursos.idconcurso')
        ->select('participanconcursos.*' )
        ->where('participanconcursos.idconcurso', $id)
        ->get();
    return view('inscripcionconcurso.show',compact('concursos','inscripcion'));
}
```

- **Destroy(\$id)->** Esta función es la encargada de eliminar una fila de la base de datos a través de un identificador que recibe por parámetro. Esta función no retorna a ninguna vista, pero si redirecciona a una ruta la cual tiene asignada una vista, una vez que se ha eliminado la fila deseada de la base de datos.

```
public function destroy($id){
    \moviexpert\mensajechat::destroy($id);
    Session::flash('message','Mensaje Borrado');
    return redirect::to('/adminmensajechat');

}
```

Una vez tenemos creadas las funciones en nuestro controladores, vamos a pasar a desarrollar las vistas para cada función.

- **Vista index ->** En esta vista mostramos una tabla con todos los datos que recibimos del controlador. Estos datos los recibimos en un array por los que para poder mostrarlos tenemos que recorrer dicho array para ello utilizamos un **foreach**. Para los enlaces debemos utilizar la siguiente sintaxis, si no utilizamos esta sintaxis no funciona los enlaces o muestran un error al pulsarlos. Debemos pasarle los siguientes argumentos
  - Ruta del controlador que queremos utilizar, este a su vez da paso a una vista.
  - Nombre del botón o texto del enlace.
  - Parámetros que queremos enviarle al controlador
  - Atributos son las clases que queremos añadirle a este enlace, para darle estilos personalizados.

```
{!!link_to_route('miembrochat.show', $title = "Chatear", $parameters = $idmiembro,
$attributes = ['class'=>'btn boton2 margin5 col-sm-4'])!!}
```

```

Project
> chat
> chats
> concursos
> errors
> frontend
> generos
> inscripcionconcurso
> layout
> mensajechat
> miembrochat
  <--> create.blade.php
  <--> index.blade.php
  <--> show.blade.php
> participanchat
  <--> create.blade.php
  <--> edit.blade.php
  <--> index.blade.php
  <--> show.blade.php
> participanconcursos
> peliculas
> perfumeria
> user
> vendor
> votosconcursos
  <--> create.blade.php
  <--> edit.blade.php

```

```

index.blade.php — user
1 @extends('layouts.frontend')
2 @section('content')
3     @include('chat.menuchat')
4     @foreach($chats as $chat)
5         <?php
6             $iduser=(integer)($chat->creadorchat);
7             $nombre=UserController::nombreUser($iduser);
8         ?>
9         <?php
10            @foreach($nombre as $nom)
11                <div class="col-xs-12 col-sm-4 col-md-4 col-md-offset-1 cuadrado">
12                    <h1 class="col-xs-12 text-center">{$chat->nombre}</h1>
13                    <p class="col-xs-12 text-center">Administrador del Grupo: {$chat->creadorchat} </p> echo " - ".$nom->nombre;></p>
14                </div>
15            </?php
16        <?php
17            $idchat=$chat->id;
18            $id=$chat->id;
19            $usuariAuth=User::find($id);
20            $numcam=MiembrochatController::contarcamaras($idchat);
21            $numact=MiembrochatController::contaractores($idchat);
22            $numdir=MiembrochatController::contardirectores($idchat);
23            $nunsg=MiembrochatController::contargolucionistas($idchat);
24            $idmiembro=$chat->participacion;
25        ?>
26        <h3 class="col-xs-6 col-md-6 text-center"></p> echo $numact; </?> / {($chat->nunactores)} <i class="glyphicon glyphicon-user"></i></h3><h3 class="col-xs-6 col-md-6 text-center"></p> echo $numdir; </?> / {($chat->nundirectores)} <i class="glyphicon glyphicon-film"></i></h3>
27        <?link_to_route('miembrochat.show', $title = 'Chatear', $parameters = $idmiembro, $attributes = ['class'=>'btn boton2 margin5 col-sm-4']);!!>
28        {!!Form::open(['route'=>['miembrochat.destroy',$idmiembro], 'method'=>'DELETE'])!!}
29        {{ csrf_field() }}
30        <div class="input-group">
31            <input type="text" name="id" value="{{$id}">
32            <input type="submit" value="Salir del Grupo",['class'=>'btn btn-danger margin5 col-sm-4']);!!>
33        </div>
34        <?Form::close()!!>
35    <?endif>
36    <?foreach

```

- **Vista Create ->** En esta vista mostramos un formulario con todos los campos necesarios para dar de alta una fila en la tabla de la base de datos deseada. Para poder enviar los datos a la función **store** para que realice la inserción debemos definir un formulario donde le indicamos el nombre del controlador, la función **store** donde vamos a realizar la inserción y el método por el que vamos a enviar los datos. Los datos de los formularios de alta los vamos a enviar por el método **POST**.

```

1 @extends('layouts.admin')
2     @section('content')
3     {!!Form::open(['route'=>'adminmensajechat.store','method'=>'POST'])!!}
4
5     {!!Form::close()!!}
6
7     @endsection
8

```

Para crear los campos de formulario utilizamos la estructura de LaravelCollective ya que funciona mejor son su propia sintaxis.

```

1 @extends('layouts.admin')
2     @section('content')
3     {!!Form::open(['route'=>'adminmensajechat.store','method'=>'POST'])!!}
4     {{ csrf_field() }}
5     @include('alerts.errorformulario')
6
7     <div class="col-xs-12 col-xs-offset-0 col-sm-10 col-sm-offset-1 cuadrado">
8         <h1 class="textMarron text-center">Nuevo Mensaje</h1>
9         <div class="input-group input-group-lg margin10">
10             <span class="input-group-addon glyphicon glyphicon-envelope"></span>
11             {!!Form::select('idmiembro',$users,null,['class'=>'form-control','placeholder'=>'Miembro'])!!}
12         </div>
13         <div class="input-group input-group-lg margin10">
14             <span class="input-group-addon glyphicon glyphicon-user"></span>
15             {!!Form::text('mensaje',null,['class'=>'form-control','placeholder'=>'Mensaje'])!!}
16         </div>
17         {!!Form::submit('Enviar Mensaje',['class'=>'btn boton2 margin10'])!!}
18     <div class="input-group input-group-lg margin10">
19         <span class="input-group-addon glyphicon glyphicon-envelope"></span>
20         <input type="text" name="id" value="{{$id}">
21     <div class="input-group input-group-lg margin10">
22         <span class="input-group-addon glyphicon glyphicon-user"></span>
23         <input type="submit" value="Salir del Grupo",['class'=>'btn btn-danger margin10']!!>
24     </div>
25     <?Form::close()!!>
26     @endsection
27

```

- **Vista Edit ->** En la vista **edit** mostramos un formulario con todos los campos que contiene la fila de la tabla y además rellenamos cada campo con su valor, para ello utilizamos la variable que recibimos del controlador. Para llenar los campos del formulario utilizamos los contenidos de un modelo por lo que debemos utilizar el método **form::model** para la apertura del formulario.

La etiqueta **form::model** tiene los siguientes argumentos:

- El primer argumento es la variable que recibimos del controlador con todos los datos de la fila. Automáticamente rellena los datos en los campos siempre que el nombre del campo del formulario sea igual al nombre del campo de la base de datos.
- El segundo argumento le indicamos la función del controlador que queremos que se ejecute al pulsar el botón submit del formulario, también enviamos al controlador el identificador de la fila que queremos actualizar.
- El tercer argumento es el método por el que vamos a enviar los datos del formulario, para las actualizaciones se utiliza PUT.

```
{!!Form::model($mensajechat,['route'=>'adminmensajechat.update',$mensajechat->id], 'method'=>'PUT')!!}
```

- **Vista show ->** En esta vista mostramos los datos de una fila concreta de la tabla de la base de datos. Estos datos los recibimos en un array que envía el controlador donde contiene todos los campos de la fila y para poder mostrarlos tenemos que recorrer el array con un foreach, al igual que en la vista del index con la diferencia que en el index se muestran todas las filas y el show solo se muestra una.

```
4   <?php $message=Session::get('message')?>
5   @if($message=='store')
6   <div class="alert alert-success alert-dismissible" role="alert">
7     <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button>
8     <strong>Mensaje enviado</strong>
9   </div>
10  @endif
11  <div class="container-fluid cuadrado">
12    <h1 class="text-center">Mensajes Chat</h1><br>
13    <table class="table table-hover text-center table-bordered">
14      <thead class="fondoMenu">
15        <th class="text-center textoBlanco">ID</th>
16        <th class="text-center textoBlanco">ID Miembro</th>
17        <th class="text-center textoBlanco">Mensaje</th>
18        <th class="text-center textoBlanco">Acciones</th>
19      </thead>
20      @foreach($mensajes as $chat)
21        <?php
22          $id=$chat->idmiembro;
23          $nombreusuario=MiembrochatController::nombreuser($id);
24        >
25        <tbody>
26          <td>{{$chat->id}}</td>
27          <td>{{$chat->idmiembro}} @foreach($nombreusuario as $user)
28            <?php echo " - ".$user->nombre;?>
29          @endforeach</td>
30          <td>{{$chat->mensaje}}</td>
31          <td class="fila">
32            {!!link_to_route('adminmensajechat.edit', $title = "Editar", $parameters = $chat->id, $attributes = ['class'=>'btn boton2 margin5'])!!}
33            {!!Form::open(['route'=>['adminmensajechat.destroy',$chat->id], 'method'=>'DELETE'])!!}
34            {{ csrf_field() }}
35            {!!Form::submit('Eliminar',['class'=>'btn btn-danger margin5'])!!}
36            {!!Form::close()!!}
37          </td>
38        ... .
39    </tbody>
40  </table>
41  <div class="text-center">
42    <button type="button" class="btn btn-primary" data-toggle="modal" data-target="#exampleModal">Nuevo Mensaje</button>
43  </div>
44  <div id="exampleModal" class="modal fade" tabindex="-1" role="dialog">
45    <div class="modal-dialog" role="document">
46      <div class="modal-content">
47        <div class="modal-header">
48          <h4 class="modal-title">Nuevo Mensaje</h4>
49          <button type="button" class="close" data-dismiss="modal" aria-label="Close">&times;</button>
50        </div>
51        <div class="modal-body">
52          <form method="post" action="adminmensajechat.store">
53            {{csrf_field()}}
54            <div class="form-group">
55              <label for="idmiembro">ID Miembro:</label>
56              <input type="text" name="idmiembro" value="1" class="form-control" />
57            </div>
58            <div class="form-group">
59              <label for="mensaje">Mensaje:</label>
60              <input type="text" name="mensaje" value="Primer mensaje" class="form-control" />
61            </div>
62            <div class="form-group">
63              <label for="status">Estado:</label>
64              <input checked="" type="checkbox" name="status" value="1" />
65            </div>
66            <div class="form-group">
67              <label for="fecha">Fecha:</label>
68              <input type="text" name="fecha" value="2018-01-01" class="form-control" />
69            </div>
70            <div class="form-group">
71              <label for="hora">Hora:</label>
72              <input type="text" name="hora" value="10:00" class="form-control" />
73            </div>
74            <div class="form-group">
75              <label for="idchat">ID Chat:</label>
76              <input type="text" name="idchat" value="1" class="form-control" />
77            </div>
78            <div class="form-group">
79              <label for="iduser">ID Usuario:</label>
80              <input type="text" name="iduser" value="1" class="form-control" />
81            </div>
82            <div class="form-group">
83              <label for="idmiembro">ID Miembro:</label>
84              <input type="text" name="idmiembro" value="1" class="form-control" />
85            </div>
86            <div class="form-group">
87              <label for="status">Estado:</label>
88              <input checked="" type="checkbox" name="status" value="1" />
89            </div>
90            <div class="form-group">
91              <label for="fecha">Fecha:</label>
92              <input type="text" name="fecha" value="2018-01-01" class="form-control" />
93            </div>
94            <div class="form-group">
95              <label for="hora">Hora:</label>
96              <input type="text" name="hora" value="10:00" class="form-control" />
97            </div>
98            <div class="form-group">
99              <label for="idchat">ID Chat:</label>
100             <input type="text" name="idchat" value="1" class="form-control" />
101           </div>
102         </div>
103         <div class="modal-footer">
104           <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancelar</button>
105           <button type="submit" class="btn btn-primary">Guardar</button>
106         </div>
107       </div>
108     </div>
109   </div>
110   <script>
111     $(document).ready(function() {
112       $('#exampleModal').modal('show');
113     });
114   </script>
115 
```

## Validaciones de formulario

Para validar los campos de los formularios Laravel proporciona varios enfoques diferentes para validar los datos de entrada de la aplicación. Por defecto, la clase controlador base de Laravel utiliza una **ValidatesRequests** característica que proporciona un método conveniente para validar la solicitud HTTP entrante con una variedad de reglas de validación de gran alcance.

Para validar nuestro formulario vamos a utilizar este sistema que utiliza Laravel por defecto, para ello debemos crear un **request** para cada formulario que queremos validar puesto que cada formulario tiene sus propios campos y cada uno necesita una regla concreta.

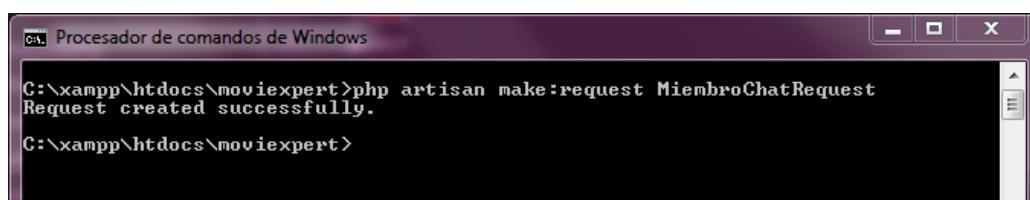
## # Reglas de validación disponibles

A continuación se muestra una lista de todas las reglas de validación disponibles y su función:

<u>Aceptado</u>	<u>Distinto</u>	<u>actual</u>
<u>URL activo</u>	<u>E-Mail</u>	<u>de expresiones regulares</u>
<u>Después de (Fecha)</u>	<u>existe (base de datos)</u>	<u>Requerido</u>
<u>Alfa</u>	<u>del archivo</u>	<u>Requerido Si</u>
<u>Alfa Dash</u>	<u>Lleno</u>	<u>necesario, a menos</u>
<u>alfanumérico</u>	<u>imagen (Imagen)</u>	<u>Requerido Con</u>
<u>de matriz</u>	<u>En</u>	<u>necesario con todas</u>
<u>antes del (fecha)</u>	<u>En matriz</u>	<u>requerida sin</u>
<u>Entre</u>	<u>de enteros</u>	<u>requerida sin todos</u>
<u>booleana</u>	<u>Dirección IP</u>	<u>el mismo</u>
<u>confirmada</u>	<u>JSON</u>	<u>tamaño</u>
<u>Fecha</u>	<u>Max</u>	<u>de cuerda</u>
<u>Formato de fecha</u>	<u>tipos MIME</u>	<u>Zona horaria</u>
<u>Diferentes</u>	<u>Tipo MIME Por extensión de</u>	<u>único (base de datos)</u>
<u>Dígitos</u>	<u>archivo</u>	<u>URL</u>
<u>Dígitos Entre</u>	<u>Min</u>	
<u>Dimensiones (archivos de</u>	<u>No En</u>	
<u>imagen)</u>	<u>numérico</u>	

Para crear los **request** ejecutamos el siguiente comando en el CMD, siempre dentro de la ruta del proyecto.

**Php artisan make:request NombreRequest**



```
C:\xampp\htdocs\moviexpert>php artisan make:request MiembroChatRequest
Request created successfully.
C:\xampp\htdocs\moviexpert>
```

Una vez creado el fichero vamos a la ruta **moviexpert/app/http/request** que es donde se encuentra todos los **request** y modificamos el **request** para indicarle los campos que queremos validar y las reglas que le queremos añadir.

```
<?php

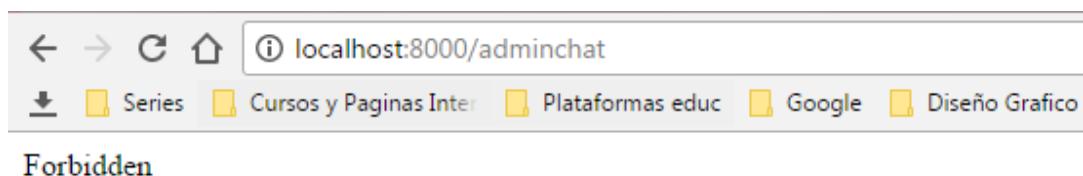
namespace moviexpert\Http\Requests;

use moviexpert\Http\Requests\Request;

class MiembroChatRequest extends Request
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return false;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            //
        ];
    }
}
```

Lo primero que tenemos que hacer es en la **function authorize()** cambiar el retorno de false a true para dar autorización para utilizar este **request**, sino cambiamos este valor en la página web nos da un error que nos indica que no tenemos permisos.



Lo segundo que tenemos que hacer es definir las reglas dentro de la función **rules()**, utilizamos la siguiente sintaxis:

```
return [
    'nombre_campo' => 'reglas',
];
```

```

<?php

namespace moviexpert\Http\Requests;

use moviexpert\Http\Requests\Request;

class ChatCreateRequest extends Request
{
    /**
     * Determine if the user is authorized to make this request.
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
    /*Get the validation rules that apply to the request.
     * @return array
     */
    public function rules()
    {
        return [
            'nombre' => 'required',
            'descripcion' => 'required',
            'numguionistas' => 'required|numeric',
            'numactores' => 'required|numeric',
            'numdirectores' => 'required|numeric',
            'numcamaras' => 'required|numeric',
        ];
    }
}

```

Una vez tenemos el fichero de validación con todas las reglas lo que tenemos que hacer es irnos al controlador y cambiar el **request** por defecto por el **request** concreto que tiene las reglas para ese formulario específico que queremos validar.

```

public function store(ChatCreateRequest $request){
    \moviexpert\Adminchat::create([
        /*Nombre campo base datos => nombre del campo del formulario*/
        'nombre'=> $request['nombre'],
        'descripcion'=>$request['descripcion'],
        'numguionistas'=> $request['numguionistas'],
        'numactores'=> $request['numactores'],
        'numdirectores'=> $request['numdirectores'],
        'numcamaras'=> $request['numcamaras'],
        'creadorchat'=> $request['creadorchat'],
    ]);
}

```

Para terminar con las validaciones creamos una alerta de error que vamos a incluir en las vistas de los formularios para que si no se cumple alguna regla nos muestre una alerta que nos indique que los datos no son válidos.

```
@if(count($errors)>0)
<div class="alert alert-danger alert-dismissible container" role="alert">
    <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button>
    <ul>
        @foreach($errors->all() as $error)
            <li>{!!$error!!}</li>
        @endforeach
    </ul>
</div>
@endif
```

Una vez tenemos la plantilla con la estructura de la alerta lo único que tenemos que hacer es incluirla en las vistas donde queremos que se muestre, para ello utilizamos el siguiente código:

```
@include('alerts.errorformulario')
```

```
    {{ csrf_field() }}
    @include('alerts.errorformulario')
<div class="col-xs-10 col-xs-offset-1 cuadrado container-fluid margintopbottom25">
    <h1 class="col-xs-12 text-center">{!!$concurso->nombre}!</h1>
    <p class="textoMarron"><strong class="col-sm-6">Plazo Inscripción: {!!$concurso->fechafinal}!</strong>
    <strong class="col-sm-6 col-xs-12 text-right"> El concurso finaliza: {!!$concurso->fechafinconcurso}!</strong>
    <div class="col-xs-10 col-xs-offset-1 margintop25">
        <p class="col-xs-12">{!!$concurso->descripcion}</p>
    </div>
    <div class="col-xs-10 col-xs-offset-1 borde fondoItem margintopbottom25">
        <h1 class="text-center">Participar en el Concurso</h1>
        <hr style="border-top: 1px solid #ccc; margin-top: 10px;">
```

## Configuración del sistema de autenticación.

Vamos a utilizar el sistema de autenticación que incorpora Laravel por defecto puesto que hace que la implementación de la autenticación sea muy simple. Para instalar este sistema utilizamos la siguiente sintaxis:

```
Php artisan make:auth
```

En este sistema Laravel lo tiene casi todo configurado por lo que resulta más fácil su implementación. El archivo de configuración de autenticación se encuentra en la ruta **movieexpert/config/auth.php**.

The screenshot shows a code editor with two panes. On the left is the project navigation pane, displaying the directory structure of a Laravel application named 'moviexpert'. The 'config' folder is expanded, showing files like app.php, auth.php, broadcasting.php, cache.php, compile.php, database.php, filesystems.php, mail.php, queue.php, services.php, session.php, and view.php. The 'public' folder also contains css, fonts, imagenes, js, .htaccess, and favicon.ico. On the right is the content of the 'auth.php' configuration file, which defines guards for web and api, and a provider for passwords.

```

Project
└─ moviexpert
    ├─ .Datos del Proyecto y BBDD
    ├─ .git
    ├─ app
    ├─ bootstrap
    └─ config
        ├─ app.php
        └─ auth.php
            └─ broadcasting.php
            └─ cache.php
            └─ compile.php
            └─ database.php
            └─ filesystems.php
            └─ mail.php
            └─ queue.php
            └─ services.php
            └─ session.php
            └─ view.php
        └─ database
        └─ public
            ├─ css
            ├─ fonts
            ├─ imagenes
            ├─ js
            └─ .htaccess
            └─ favicon.ico

```

```

auth.php
1 <?php
2
3 return [
4
5     'defaults' => [
6         'guard' => 'web',
7         'passwords' => 'users',
8     ],
9
10    'guards' => [
11        'web' => [
12            'driver' => 'session',
13            'provider' => 'users',
14        ],
15
16        'api' => [
17            'driver' => 'token',
18            'provider' => 'users',
19        ],
20    ],
21
22
23    'passwords' => [
24        'users' => [
25            'provider' => 'users',
26            'email' => 'auth.emails.password',
27            'table' => 'password_resets',
28            'expire' => 60,
29        ],
30    ],
31
32];
33

```

Para enrutar todo el sistema de autenticación y que no de errores al acceder al login o al formulario de registro bastara con añadir la siguiente linea al fichero **routes.php**

**Route::auth();**

```

61 Route::get('perfilusuario/destroy', 'PerfilusuarioController@destroy');
62 Route::get('miembrochat/participanchat/{id}', "MiembrochatController@participan
63 Route::get('concursos/participanconcurso/{id}', 'FrontendController@participancc
64 Route::post('concursos/participanconcurso/{id}/altaparticipacion', 'FrontendCont
65
66
67
68 Route::auth();
69
70
71 Route::get('/home', 'HomeController@index');
72

```

Una vez tenemos configurado el fichero de configuración de la autenticación y está definida la ruta solo nos falta configurar el controlador y definir la vista para realizar el login si estas registrado o el formulario de registro en caso de no estar registrado.

```

Project
└ moviexpert
  └ app
    └ Controllers
      └ Auth
        └ AuthController.php

AuthController.php

11 class AuthController extends Controller
12 {
13     use AuthenticatesAndRegistersUsers, ThrottlesLogins;
14     protected $redirectTo = '/';
15     public function __construct()
16     {
17         $this->middleware($this->guestMiddleware(), ['except' => 'logout']);
18     }
19     protected function validator(array $data)
20     {
21         return Validator::make($data, [
22             'email' => 'required|email|max:255|unique:users',
23             'password' => 'required|min:6|confirmed',
24             'nombre' => 'required|max:255',
25             'apellidos' => 'required|max:255',
26             'direccion' => 'required|max:255',
27             'localidad' => 'required|max:255',
28             'genero' => 'required|max:255',
29             'fechanacimiento' => 'required|date',
30         ]);
31     }
32     protected function create(array $data)
33     {
34         return User::create([
35             'email'=> $data['email'],
36             'password'=> bcrypt($data['password']),
37             'nombre'=> $data['nombre'],
38             'apellidos'=> $data['apellidos'],
39             'direccion'=> $data['direccion'],
40             'localidad'=> $data['localidad'],
41             'genero'=> $data['genero'],
42             'fechanacimiento'=> $data['fechanacimiento'],
43             'foto'=>'perfildefecto.png',
44             'tipousuario'=> 'normal',

```

En este controlador tenemos dos funciones la primera **validator** la utilizamos para validar los campos del formulario hasta ahora lo hemos validado a través de ficheros request pero como indicamos hay varias formas para validar los campos y otra de ella será esta, crear una función que valide los datos dentro del controlador. La segunda función es la encargada de dar de alta a los usuarios en la base de datos a esta función la llamamos desde la vista del formulario de registro de usuario.

Lo siguiente que necesitamos es crear las vistas para el login y el registro, en el login simplemente creamos un formulario de dos campos y un botón para enviar los datos al controlador y este pueda realizar la consulta a la base de datos para comprobar que son correctos si es así accedes a la página y sino muestra una alerta indicando porque no has podido acceder.

```

Project
└ resources
  └ views
    └ auth
      └ emails
        └ login.blade.php

login.blade.php

12 <div class="form-group({ $errors->has('email') ? ' has-error' : '' })>
13   <label for="email" class="col-md-4 control-label">E-Mail</label>
14
15   <div class="col-md-6">
16     <input id="email" type="email" class="form-control" name="email" value="{{ old('email') }}>
17     @if ($errors->has('email'))
18       <span class="help-block">
19         <strong>{{ $errors->first('email') }}</strong>
20       </span>
21     @endif
22   </div>
23 </div>
24 <div class="form-group({ $errors->has('password') ? ' has-error' : '' })>
25   <label for="password" class="col-md-4 control-label">Password</label>
26
27   <div class="col-md-6">
28     <input id="password" type="password" class="form-control" name="password">
29     @if ($errors->has('password'))
30       <span class="help-block">
31         <strong>{{ $errors->first('password') }}</strong>
32       </span>
33     @endif
34   </div>
35 </div>
36 <div class="form-group">
37   <div class="col-md-6 col-md-offset-4">
38     <div class="checkbox">
39       <label>
40         <input type="checkbox" name="remember"> Remember Me
41       </label>
42     </div>
43   </div>
44 </div>
45

```

Para el registro hacemos lo mismo creamos una vista con un formulario donde pedimos al usuario todos los datos necesarios y llamamos a la función del controlador para que inserte los datos en la base de datos, estos datos se envían por parámetro en un array asociativo.

The screenshot shows a code editor with two tabs open. The left tab is 'AuthController.php' and the right tab is 'register.blade.php'. The file structure on the left includes 'css', 'fonts', 'imagenes', 'js', '.htaccess', 'favicon.ico', 'index.php', 'robots.txt', 'web.config', 'resources', 'assets', 'lang', and 'views' (containing 'admin', 'adminmensajeschat', 'alerts', 'auth' (which contains 'emails' with 'password.blade.php', 'register.blade.php', and 'login.blade.php'), 'chat', 'chats', 'concursos', and 'errors'). The 'register.blade.php' file content is as follows:

```


<div class="panel panel-default cuadrado">
        <div class="panel-heading"><h3>Formulario de Registro</h3></div>
        <div class="panel-body">
            <form class="form-horizontal" role="form" method="POST" action="{{ url('/register') }}>
                {{ csrf_field() }}


```

The code continues with form fields for email and password, validation logic using @if and @endif, and ends with a submit button.

También hemos incorporado la opción para recuperar la contraseña a través del email y la opción de recordar contraseña en el inicio de sesión.

The screenshot shows the 'password.blade.php' file content. It contains a single line of code:

```
1 Click here to reset your password: <a href="{{ $link = url('password/reset', $token). '?email=' . urlencode($user->getEmailForPasswordReset()) }}>{{ $link }}</a>
```

Si queremos saber si algún usuario a iniciado sesión y si es así mostrar algún dato de este usuario utilizamos el siguiente código:

```

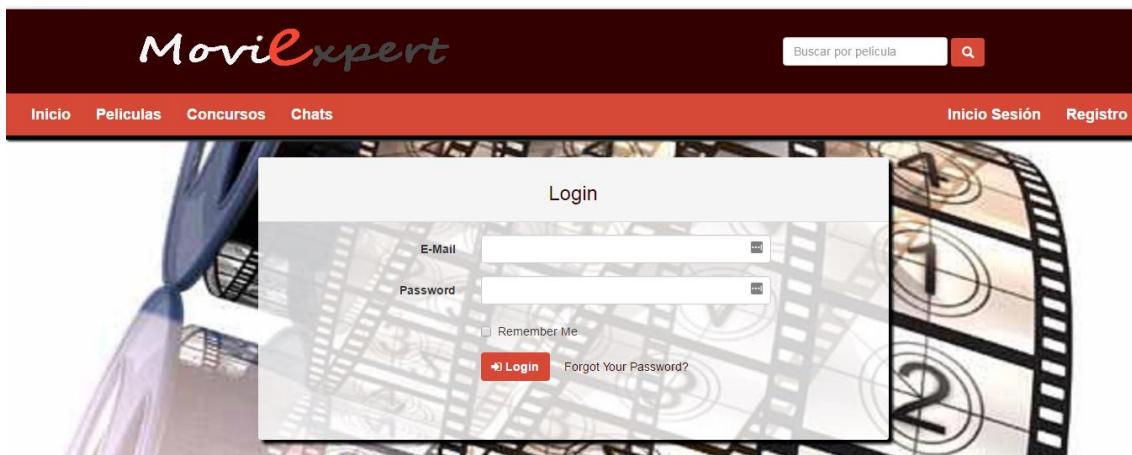
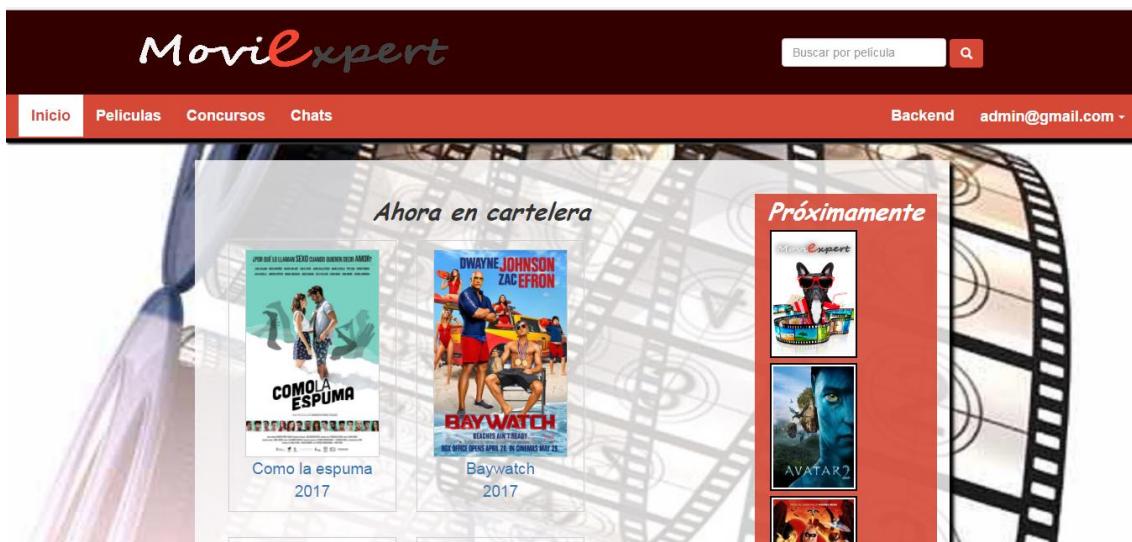
@if (Auth::guest())
    -> Si ningún usuario a iniciado sesión
    Contenido NO inicio sesión
@else
    -> Si hay algún usuario que ha iniciado sesión
    @if(Auth::user()->tipousuario=='admin')
        -> Si el usuario es administrador
        Contenido que solo ver el administrador
    @endif
    Contenido que ven todos los usuarios normales y administradores
    {{ Auth::user()->email }}
        -> Para mostrar un dato del usuario que se ha logueado
@endif
```

```

<div id="menuadmin" class="fondoMenu borderBottom container col-xs-12">
    <ul class="nav navbar-nav navbar-left">
        <li class=""><a class="textoMenu text-center" href="/">Inicio</a></li>
        <li class=""><a class="textoMenu text-center" href="/peliculas">Películas</a></li>
        <li class=""><a class="textoMenu text-center" href="/concursos">Concursos</a></li>
        <li class=""><a class="textoMenu text-center" href="/chat">Chats</a></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
        <!-- Authentication Links -->
        @if (Auth::guest())
            <li><a class="textoMenu text-center" href="{{ url('/login') }}>Inicio Sesión</a></li>
            <li><a class="textoMenu text-center" href="{{ url('/register') }}>Registro</a></li>
        @else
            @if(Auth::user()->tipousuario=='admin')
                <li class=""><a class="textoMenu text-center" href="/admin">Backend</a></li>
            @endif
            <li class="dropdown">
                <a href="#" class="dropdown-toggle btn textoMenu" data-toggle="dropdown" role="button" aria-expanded="false">
                    {{ Auth::user()->email }} <span class="caret"></span>
                </a>
                <ul class="dropdown-menu" role="menu">
                    <li><a href="{{ url('/perfilusuario') }}><i class="fa fa-pencil" aria-hidden="true"></i>Editar Perfil</a></li>
                    <li><a href="{{ url('/misCriticas') }}><i class="fa fa-film" aria-hidden="true"></i> Mis críticas </a></li>
                    <li><a href="{{ url('/logout') }}><i class="fa fa-btn fa-sign-out"></i>Cerrar Sesión</a></li>
                </ul>
            </li>
        @endif
    </ul>
</div>
</nav>

```

Vemos como se vería la vista con un usuario logueado y otra sin.



# FRONTEND

## Inicio

En el Frontend o la vista al usuario se puede observar que hay 4 enlaces que nos llevan a Inicio, Películas, Concursos y Chats.



En la página de inicio nos encontramos con una serie de imágenes de películas con sus títulos y sus años. Son las películas en cartelera, las películas actuales que el usuario puede ir al cine a verlas, al cliquear sobre ellas nos lleva directamente a su ficha con su descripción más detallada. Estas películas están ordenadas por año en orden descendente. A la derecha de estas imágenes se encuentran imágenes de películas con un tamaño inferior, estas películas son las que todavía no están en el cine, pero estarán próximamente, y en algunas podemos obtener información acerca de ellas y aún así los usuarios pueden comentar ofreciendo así una página interactiva.

Estos estrenos también están ordenados de forma descendente y se mostrarán de 8 en 8. Cuando la película salga en cartelera automáticamente pasará a ser una película de estreno y se mostrará a la izquierda, donde están las películas de estreno.



Para visualizar las películas actuales, en la función de **Index** se ha hecho con una consulta y con php sacando el año actual para ver que películas son las de este año, que son las actuales. Luego se ordena descendente el año y el id, visualizando solo las películas actuales. El código es así:

```
public function index(){
    $peliculas=DB::select('select * from adminpeliculas where anio="'.date('Y').'" order by anio DESC,id DESC');
    return view("frontend.index",compact('peliculas'));
}
```

En la variable **películas** se hace la consulta directamente. Y se retorna la vista de películas.

En la vista de películas es donde se encuentra el html, que se puede modificar el estilo con el CSS y usando Boostrap y para mostrar los datos que tenemos guardados

en la base de datos, se realiza con un **foreach** recorriendo la variable **\$peliculas** que hemos definido en la función de **index** del controlador, y para mostrar los datos con la variable llamando al campo de la base de datos que se quiere mostrar de uno en uno. Así quedaría el código:

```
@foreach($peliculas as $pelicula)
    <a class="col-xs-12 col-md-6 fichaRes2" href="{{ url('/ficha') }}/{{ $pelicula->id}}>
        
        <div class="marginBottomFicha">
            <div class=" colorNegro negrita"> <span class="titulosPeliculas negrita">Título: </span> {{ $pelicula->titulo}}</div>
            <div class="colorNegro negrita"> <span class="titulosPeliculas negrita">Año: </span> {{ $pelicula->anio}}</div>
            <div class="colorNegro negrita"> <span class="titulosPeliculas negrita">Duración: </span> {{ $pelicula->duracion}}</div>
            <div class="colorNegro negrita"> <span class="titulosPeliculas negrita">Género: </span> {{ $pelicula->genero}}</div>
        </div>
    </a>

```

Para mostrar los estrenos que es otra consulta independiente de películas, en el controlador se define otra función que sea para mostrar estos estrenos. Como lo que queremos mostrar es diferente a las otras películas, porque éstas películas son diferentes, éstas son las películas que todavía no se han estrenado y que ya se está dando pequeña información sobre ellas para que los usuarios estén al tanto de las películas que saldrán próximamente. La consulta sería parecida a la de películas pero con la modificación de que si antes eran las películas de este año, pues con un signo de > mayor a este año, ya saldrían las de próximamente .

```
public static function estrenos(){
    $estreno=DB::select('select * from adminpeliculas where anio>"'.date('Y').'" order by anio DESC,id DESC limit 8');
    return $estreno;
}
```

## Ficha

Como se ha descrito con anterioridad, las películas llevan a su ficha, donde se detalla sus características principales. La novedad en esta página es que el usuario puede realizar un voto.

Los datos de las películas vienen de la misma tabla que películas, y ya se ha explicado cómo sacar los datos de la base de datos. Lo único nuevo que aparece es lo de votar. Para poder realizar un voto se crea un modelo que sea para votar para poder crear ahí los métodos que se necesitan. El método para que se visualice la puntuación de cada película necesita que se pase por parámetro el id de la película para identificar qué película es la que tiene esa puntuación. En esta función se hace una consulta comparando el id del película con la variable que pasamos por parámetro y haciendo la media del campo voto.

```
static public function avgVotos($idpeli) {
    return Votospeliculas::where('idpelicula', '=', $idpeli)->avg('voto');
}
static public function countVotos($idpeli){
```

Y en el controlador se guarda en una variable el resultado de este método, llamándole en la función.

Para contar los votos se crea otra función estática en el mismo controlador de votos películas, y se hace de manera similar a la media, pero en vez de avg, se hace un count y así se cuenta los votos, pasando también por parámetro el id de la película, para identificar en qué película se está haciendo.

```
static public function countVotos($idpeli){  
    return Votospeliculas::where('idpelicula', '=', $idpeli)->count();  
}
```

En el controlador, en la misma función donde se ha hecho la media, se crea otra variable para guardar el resultado de la cuenta de votos. Para poder retornar la vista con diferentes variables de distintas tablas, se usa el **with** con dos parámetros, el campo de la base de datos y el nombre de la variable. También se ha usado el **numberFormat** para que solo se pongan dos decimales por cuestión de estética. Para crear los votos como está en la ficha de la película, se ha usado su propio método para la visualización, quedando así:

```
public function ficha($id){  
    $pelicula=\moviexpert\Adminpelicula::find($id);  
    $mediaVotos=\moviexpert\Votospeliculas::avgVotos($id);  
    $mediaVotos=number_format($mediaVotos,1);  
    $cuentaVotos=\moviexpert\Votospeliculas::countVotos($id);  
    $generos=\moviexpert\Admingenero::lists('genero','id');  
    return view('frontend.ficha',compact('pelicula'))->with("generos",$generos)->with("mediaVotos",$mediaVotos);  
}
```

Añadiendo los otros **with** con las otras variables.

Para poder realizar un voto se necesita crear otra función, que es la que tiene las peticiones del formulario.

Primero en el modelo se crean los métodos necesarios. Métodos de comprobar los votos, donde se hace una consulta para hacer un **count** que cuenta y se comprueba que el **idusuario** y el **idpelicula** coinciden con los argumentos.

Y otro método que se necesita es encontrar a qué película se hace referencia y qué usuario.

```
static public function findByPeliAndUser($idpeli,$idusuario) {  
    return Votospeliculas::where('idpelicula', '=', $idpeli)->where('idusuario', '=', $idusuario)->select('id')->take(1)->get();  
}
```

Quedando así la función de enviar votos que es la que se hace a través del formulario.

Y así sería la vista de votos:



Indicando así con las estrellas cuál es su valoración. Cada estrella representa 2 puntos, llegando así al 10 si se eligen las 5 estrellas. Como se ve en la imagen también se ve la nota que tiene ya, que automáticamente al realizar el voto cambiará porque hará la media con el resto de votaciones que hayan realizado otros usuarios distintos. Otra cosa a tener en cuenta es el número de votos. Se puede ver el número de votos que se ha realizado por cada película.

## Trailer

Al ir a la vista de ficha, nos aparece un submenú con la ficha ya explicada y también con otros enlaces como son tráiler y críticas.

El tráiler nos lleva a una página donde se encuentran los trailers de cada película y una foto de esta misma, y las críticas son las críticas que realizan los usuarios por cada película.

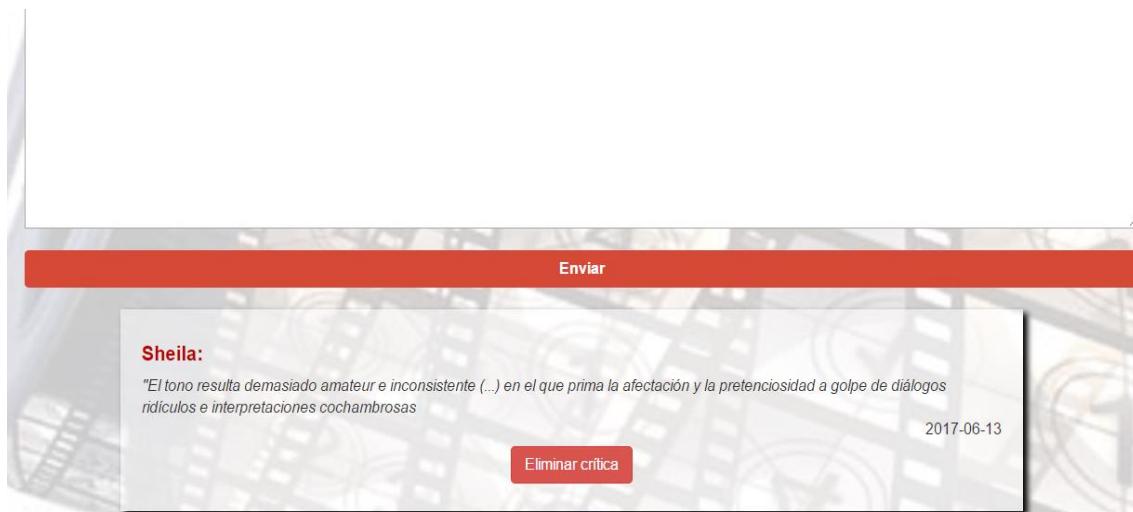
A screenshot of a movie page for 'Como la espuma'. The top navigation bar includes 'Ficha', 'Trailer', and 'Críticas'. Below the navigation is a movie poster for 'COMOLA ESPUMA' showing a man and a woman in swimwear. To the right of the poster is the title 'Como la espuma' in large red letters. Below the title is a video player showing a scene from the trailer where a woman in a pink bikini is surrounded by men.

## Críticas

En el controlador se crea la función de críticas que sólo ordena las críticas de orden descendente. Hay otra función que es la que trabaja el formulario que es la de procesar críticas pasando por parámetros los request. La fecha se hace con un método propio de php para que salga la fecha exacta de hacer la crítica.

```
public function procesarCriticas(criticasPeliculasCreateRequest $request){  
    \moviexpert\CriticaPelículas::create([  
        'idpelícula'=> $request['idpelícula'],  
        'idusuario'=> $request['idusuario'],  
        'crítica'=> $request['crítica'],  
        'fechavoto'=> date('Y-m-d H:i:s'))];  
    return redirect("/críticas/".$request['idpelícula']);  
}
```

Al irnos a la ventana de críticas tendremos un textarea grande para poder realizar los comentarios, y podremos ver las críticas del resto de usuarios con la fecha en la que fue realizada y el nombre del usuario. El administrador puede eliminar las críticas que quiera si ve palabras malsonantes o algún comentario fuera de lugar. El usuario puede eliminar sus propias críticas en el momento que desee. Las críticas están ordenadas de forma descendente por lo que siempre se ve como primera crítica la última realizada.



## Películas

En el enlace de películas vemos un listado de películas con diferente vista a la de inicio, éstas están ordenadas por orden alfabético y aquí aparecen todas las películas de la página con una breve descripción y su puntuación en el momento.

Vista general    [Top 10](#)

**2001: A Space Odyssey**  
An epic drama of adventure and exploration  
Título: 2001: A Space Odyssey  
Año: 1968  
Director: Stanley Kubrick  
**10.0**

**Abuelos al poder**  
Título: Abuelos al poder  
Año: 2012  
Director: Andy Fickman  
**0.0**

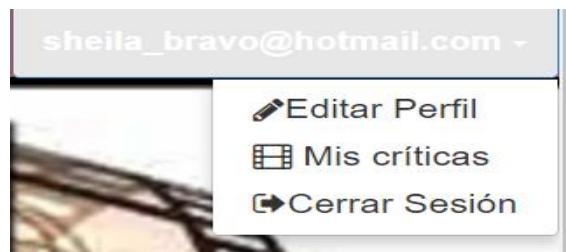
También nos encontramos con otra pestaña que es top10, como su propio nombre indica son las 10 películas con mayor puntuación de nuestra página.

**2001: A Space Odyssey**  
An epic drama of adventure and exploration  
Título: 2001: A Space Odyssey  
Año: 1968  
**Sinopsis:** La película de ciencia-ficción por excelencia de la historia del cine narra los diversos períodos de la historia de la humanidad, no sólo del pasado, sino también del futuro. Hace millones de años, antes de la aparición del "homo sapiens", unos primates descubren un monolito que los conduce a un estadio de inteligencia superior. Millones de años después, otro monolito, enterrado en una luna, despierta el interés de los científicos. Por último, durante una misión de la NASA, HAL 9000, una máquina dotada de inteligencia artificial, se encarga de controlar todos los sistemas de una nave espacial tripulada.  
**10.0**

**La mécanique de l'ombre**  
Título: La mécanique de l'ombre  
Año: 2017  
**Sinopsis:** Un enigmático hombre de negocios en nombre de una misteriosa organización se pone en contacto con Duval (François Cluzet) para ofrecerle un trabajo sencillo y bien remunerado: transcribir escuchas telefónicas interceptadas. Duval, económicamente desesperado, acepta sin preguntar sobre la finalidad de la empresa que lo contrata. De pronto, envuelto en un complot político, debe afrontar la brutal mecánica del mundo oculto de los servicios secretos.  
**10.0**

Esto quiere decir que están ordenadas por puntuación.

## Perfil de Usuario



Como se puede observar en la fotografía, al cliquear sobre el nombre, en este caso email del usuario, se abre una nueva navegación. Editar perfil, donde el usuario podrá ver sus datos, modificarlos y también cerrar su cuenta, no sin antes mandar un mensaje de alerta para que se lo piense antes de cerrarla.

Mis críticas son las críticas realizadas por el usuario logueado, solo aparecen las suyas propias y le da la opción de eliminarlas, así puede tener acceso a ellas en cualquier momento para poder borrarlas. Aparece también el título de la película a la que ha hecho la crítica y la fecha en la que la realizó.

Para hacer esto, en el controlador se crea la función donde se usan dos variables, una para comprobar qué usuario está logueado y que se ordene, esto se hace a través de una consulta,

Y otra variable que guarde el título de la película pasando por parámetro el id también para identificar la película.

```
public function misCriticas(){
    $criticas=DB::table('criticaPeliculas')->where('idusuario', '=', Auth::user()->id)->orderBy('id', 'DESC')->paginate(10);
    $peliculas=\moviexpert\Adminpelicula::lists('titulo','id');
    return view("frontend.misCriticas",compact('criticas'))->with('peliculas',$peliculas);
}
```

En este menú tenemos otra opción que nos permite editar el perfil de usuario el sistema es el mismo que utilizamos para editar usuarios en el backend, con la diferencia que en el backend puedes editar cualquier usuario y aquí solo puedes editar tus datos de usuario.

A screenshot of a user profile edit form titled "Mis datos". The form includes fields for "Email" (admin@gmail.com), "Contraseña" (Ingresar la contraseña), "Nombre" (admin), "Apellido" (admin), "Fecha de nacimiento" (02/06/2017), and gender selection buttons for "Hombre" and "Mujer". There is also a "Actualizar Datos" button at the bottom. The background features a film strip graphic.

Y cerrar sesión que es salir de su cuenta.

## Concursos

En la pestaña de concursos mostramos una lista con todos los concursos que existe en la base de datos. En esta pestaña podemos hacer varias cosas:

1. Si está abierto el plazo de inscripción podemos inscribirnos en el concurso una única vez por usuario.
2. Si el plazo de inscripción ha finalizado pero el del concurso sigue abierto podemos entrar a ver todos los participantes y votar los cortos que más o menos nos gusten. Solo se puede votar una vez por usuario un corto
3. Si ambos plazos han finalizado podemos ver el nombre del usuario ganador.



Para comprobar que el mismo usuario se inscriba dos veces al mismo concurso hemos creado una función que nos cuenta las veces que aparece un usuario con el mismo código de concurso si aparece una vez cambiamos el botón por una etiqueta ya estas inscrito en el concurso.

```
public static function existeusuario($idusuario,$idconcurso){  
    $numvotos= DB::table('participanconcursos')  
    ->where([  
        ['idusuario', '=', $idusuario],  
        ['idconcurso', '=', $idconcurso],  
    ])  
    ->count();  
    return $numvotos;  
}
```

Si no estamos inscritos pulsamos sobre inscribirnos y rellenamos el formulario.

The screenshot shows a competition registration page. At the top, it says 'Tecnologías'. Below that, the inscription period is listed as 'Plazo Inscripción: 2017-06-13 al 2017-07-23' and the finalization date is 'El concurso finaliza: 2017-07-30'. A subtitle reads 'Como los avances tecnológicos están cambiando la vida cotidiana de las personas'. The main form area is titled 'Participar en el Concurso' and contains the following fields:

Código del Concurso	2
Participante Principal	11
<input type="checkbox"/> Ingrese otros participantes	
Título del Corto	<input type="button" value=""/>
<input type="checkbox"/> Ingrese la descripción	
<input type="checkbox"/> Ruta del Corto	

At the bottom is a red button labeled 'Participar Concurso'.

Si ya ha finalizado el plazo de inscripción podemos ver los participantes y votar.

The screenshot shows a competition voting interface. At the top, it says 'Naturaleza'. Below that, the inscription period is listed as 'Plazo Inscripción: 2017-06-13 al 2017-06-14' and the finalization date is 'El concurso finaliza: 2017-06-18'. A subtitle reads 'Cortometraje sobre la naturaleza, cómo viven los animales'. The main content area is titled 'Los animales nuestros amigos' and contains the following information:

Usuario: 12 - admin  
Otros Participantes: 13  
Un corto rodado al natural , con la naturaleza tal cual es  
Ruta Corto: <https://www.youtube.com/watch?v=5uBSQOn1Sx8>

To the right, there is a voting panel with the following text and options:

Total Votos: 0  
Vota este corto

<input type="radio"/>	★★★★★
<input type="radio"/>	★★★
<input type="radio"/>	★★★
<input type="radio"/>	★★
<input type="radio"/>	★

Al igual que no dejamos que un usuario se inscriba dos veces en un concurso tampoco podemos permitir que un usuario vote todas las veces que quiera, por lo que hemos utilizado una función igual a la anterior pero con distinta consulta a la base de datos.



## Chat

Si vamos a la pestaña chat podemos ver una lista con todos los chat, esta vista además incorpora un menú propio con todas las opciones que tenemos disponibles en el chat.

The screenshot shows the MovielExpert website's chat section. At the top, there's a navigation bar with "MovielExpert", a search bar, and links for "Backend" and "admin@gmail.com". Below the navigation, there are buttons for "Actor", "Guionista", "Director", and "Camara". A menu bar includes "Lista Chat", "Mis Chat", "Administrar Grupo Chat", and "Nuevo Grupo Chat". The main content area displays two movie entries in a grid:

- Los amantes inaguantables**  
Administrator del Grupo: 12 - Sheila  
Un corto sobre personas que se quieren pero no pueden estar juntas.  
1 / 12 0 / 2   
0 / 1 1 / 4
- Corto sobre el reciclaje**  
Administrator del Grupo: 13 - Alicia  
Un cortometraje que sirva para concientizar sobre el reciclaje.  
0 / 4 0 / 3   
0 / 2 0 / 2

Al igual que en los concurso hemos validado que un usuario no se pueda registrar varias veces en un concurso así como que solo se puedan registrar si quedan plazas disponibles y que sean en estas plazas disponibles donde hagan su incorporación.



Para validar que los usuarios no se inscriban si está completo hemos creado las siguientes funciones en las cuales recuperamos la suma los miembros de cada categoría de cada grupo de chat y preguntamos si esta suma es igual al total de miembros de esa categoría si es así indicamos que está completo.

```

Project
├── PasswordController.php
├── AdminchatController.php
├── AdminconcursoController.php
├── AdmingeneroController.php
├── AdmimnajeschatController
├── AdminparticipanconcursoCon
├── AdminparticipanteschatContr
├── AdminpeliculaController.php
├── AdmivotosconcursoController
└── Controller.php

InscripcionconcursoController.php
    public static function usuarioInscrito($idusuario,$idchat){
        $num= DB::table('miembrochats')
            ->where([
                ['idchat', '=', $idchat],
                ['idusuario', '=', $idusuario],
            ])
            ->count();
        return $num;
    }

    public static function contarcamaras($idchat){
        $num= DB::table('miembrochats')
            ->where([
                ['idchat', '=', $idchat],
                ['tipomembro', '=', 'camara'],
            ])
            ->count();
        return $num;
    }

    public static function contaractores($idchat){
        $num= DB::table('miembrochats')
            ->where([
                ['idchat', '=', $idchat],
                ['tipomembro', '=', 'actor'],
            ])
            ->count();
        return $num;
    }

    public static function contardirectores($idchat){
        $num= DB::table('miembrochats')
            ->where([
                ['idchat', '=', $idchat],
                ['tipomembro', '=', 'director'],
            ])
            ->count();
        return $num;
    }

MensajechatController.php
MiembrochatController.php
PerfilusuarioController.php
UserController.php
VotarcortoController.php
VotospeliculaController.php

Middleware
Requests
Kernel.php
routes.php

```

Para crear un nuevo grupo de chat bastara con pulsar sobre el botón de nuevo grupo de chat y automáticamente nos aparece un formulario para crear el nuevo grupo. En el controlador tenemos una función denominada **usuarioInscrito** que es la utilizaremos para validar que dos usuarios no se inscriban en el mismo grupo de chat con categorías distintas.

**Introducir chat**

Nombre del Chat  
Descripción del Chat

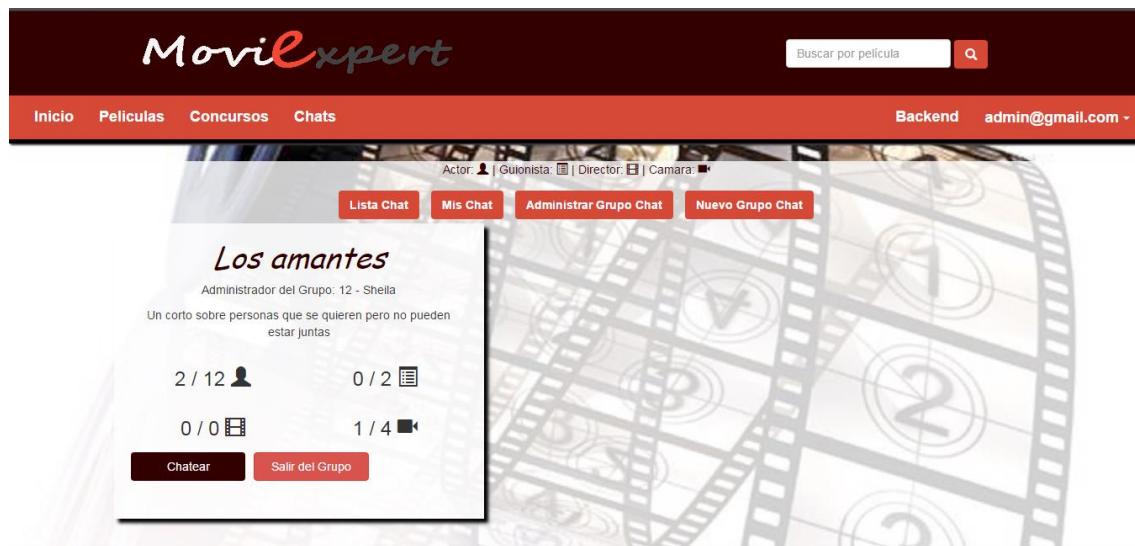
Nº Guionistas  
Nº Actores  
Nº Directores  
Nº Cámaras

Crear Grupo Chat

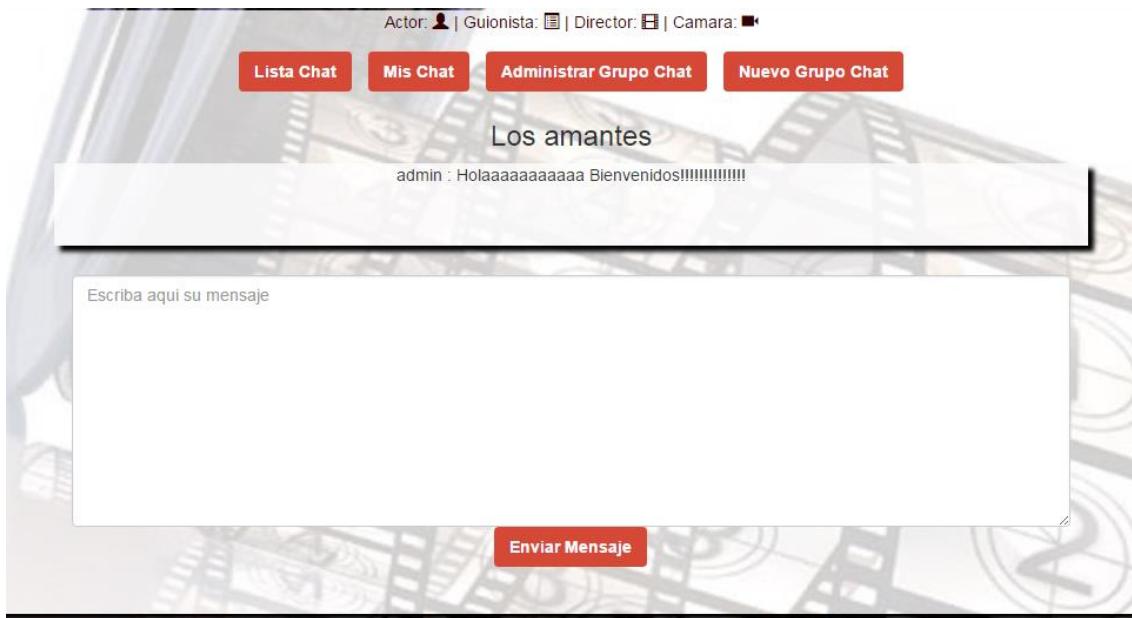
Si pulsamos sobre la opción de administrar grupo de chat podemos ver una lista con todos los grupos que hemos creado y borrarlo si lo deseáramos.



Si pulsamos en la opción de mis chat podemos ver una lista de todos los chat en los estamos participando y desde aquí podemos salirnos del grupo o chatear con los otros miembros del grupo.



Si pulsamos sobre el botón de chatear nos redirige a la vista que hemos desarrollado para esta funcionalidad. En el panel superior se visualizan los mensajes ya enviados en este grupo y en segundo panel nos encontramos un campo de texto donde escribir nuestro mensaje y un botón para enviar nuestro mensaje y que el resto de participantes puedan visualizarlo.



Para mostrar los mensajes ordenados según se enviaron por los usuarios para no alterarla la conversación utilizamos la siguiente consulta. Almacénanos los datos que devuelve la consulta en un variable que se la pasamos a la vista y esta la cargara en el panel de visualización de los mensajes.

```
public function show($id){
    $chat = DB::select('select * from mensajechats where idmiembro IN (select id from miembrochats where idchat=(select idchat from miembrochats where id=:id)) ORDER BY id ASC',['id' => $id]);
    $nombre =DB::select('select * from adminchats where id=(select idchat from miembrochats where id=:id)', ['id' => $id]);
    return view('miembrochat.show',['chat'=>$chat,'nombre'=>$nombre,'miembro'=>$id]);
}
```

Almacénanos los datos que devuelve la consulta en un variable que se la pasamos a la vista y esta información la cargara en el panel de visualización de los mensajes. Esta consulta no devuelve el código del usuario no su nombre y para visualizarlo quedaría mejor el nombre ya que sino aparecería un identificador numero para ello utilizamos la siguiente función.

```
public static function nombreuser($id){
    $nombre = DB::select('select nombre from users where id=(select idusuario from miembrochats where id=:id)', ['id' => $id]);
    return $nombre;
}
```

Como podemos ver en la vista cargamos los mensajes de la variable enviada a la vista y para el nombre del usuario llamamos a la función **nombreuser** del controlador.

```

1 <?php use moviexpert\Http\Controllers\MiembrochatController;?>
2 @extends('layouts.frontend')
3 @section('content')
4 @include('chat.menuchat')
5 <h3>
6 @foreach($nombre as $chatnom)
7 <?php echo $chatnom->nombre;?>
8 @endforeach
9 </h3>
10 <div class="col-xs-12 col-sm-12 col-md-8 col-md-offset-2 cuadrado">
11 @foreach($chat as $chat)
12 <?php
13     $id=$chat->idmiembro;
14     $nombreusuario=MiembrochatController::nombreuser($id);
15   ?>
16 <p class="col-xs-12">
17     @foreach($nombreusuario as $user)
18       <?php echo $user->nombre." : ".$chat->mensaje;?>
19     @endforeach
20   </p>
21 @endforeach
22 </div>
23 <div class="col-xs-12 col-sm-12 col-md-8 col-md-offset-2 margin-top25">
24   {!!Form::open(['route'=>'mensajechat.store','method'=>'POST'])!!}
25   {{ csrf_field() }}
26   {!!Form::textarea('mensaje',null,['class'=>'form-control','placeholder'=>'Escriba aqui su mensaje'])!!}
27   {!!Form::text('idmiembro',$miembro,['class'=>'form-control novisible','placeholder'=>''])!!}
28   {!!Form::submit('Enviar Mensaje',['class'=>'btn boton'])!!}
29   {!!Form::close()!!}
30 </div>
31 @endsection

```

## Buscadores

El buscador principal se sitúa en la cabecera de la página, y éste hace una búsqueda de películas exhaustiva, ya que busca por título, director reparto y fecha, si los caracteres introducidos están en título, director, reparto o fecha, entonces aparecerá en la vista.

*Películas cuyo título coincide con: "br":*



**Título:** Brave  
**Año:** 2012  
**Director:** Mark Andrews, Brenda Chapman, Steve Purcell  
**Reparto:** Animation

*Películas cuyo director coincide con: "br":*



**Título:** Los increíbles 2  
**Año:** 2018  
**Director:** Brad Bird  
**Reparto:** Animation

Es muy importante en una página como ésta, hacer una búsqueda por año, por si al usuario le interesa saber que películas son de otro año distinto a este, y así las puede encontrar rápidamente.

*Películas cuyo año coincide con: "2016":*



**Título:** Marie Curie  
**Año:** 2016  
**Director:** Marie Noëlle  
**Reparto:** Karolina Gruszka, Arieh Worthalter, Charles Berling, Izabela Kuna, Malik Zidi, André Wilms, Daniel Olbrychski, Marie Denarnaud, Samuel Finzi, Piotr Glowacki, Jan Frycz, Sabin Tambrea, Sasha Crapanzano, Rose Montron, Adele Schmitt, Emma Pokromska, Edgar Sehr, Nikolaus Frei, Artur Dziurman, Piotr Bartuszek, Aldona Bonarowska, Klara Bielawka, Mariola Brycht, Paweł Kleszcz, Wenanty Nosul, Jakub Kotynski, Ksawery Szlenkier, Michał Meyer, Konrad Bugaj, Krzysztof Bochenek



**Título:** Al final del túnel  
**Año:** 2016  
**Director:** Rodrigo Grande  
**Reparto:** Leonardo Sbaraglia, Clara Lago, Pablo Echarri, Federico Luppi, Javier Godino, Walter Donado, Uma Salduende, Daniel Morales Comini, Laura Faienza, Sergio Ferreiro, Facundo Nahuel

En el Backend también podemos encontrar buscadores propios, por ejemplo en usuarios busca por nombre.



Aparecerán los resultados que empiecen por la letra que se haya introducido en el buscador, y es igual para cada tabla de administrador. Cada buscador tiene su propio **placeholder** para indicar por lo que se tiene que buscar.

Hay muchas formas de hacer buscadores, Laravel utiliza **scope** para hacer la consulta en la base de datos y buscar por el campo que se desee. Pero se puede hacer sin **scope** como se muestra a continuación.

```
static public function titulo($titulo){
    return DB::select("SELECT * FROM adminpeliculas WHERE titulo like '".$titulo."%'");
}
```

Se puede observar cómo se crea una función estática pasando un parámetro. En este caso la variable se llama título puesto que vamos a buscar por título. Se hace una consulta a la base de datos, se busca todos los campos en la tabla de **adminpeliculas** que es donde están todas las películas con sus campos correspondientes y se le dice que donde título sea como el parámetro que le pasamos, con el porcentaje indicamos que busque por los primeros caracteres de título. En el controlador es donde se crea la función para poder buscar películas pasándole los argumentos request. Se almacena en las variables el método que se ha construido en el modelo, donde hace la consulta SQL, quedando de esta forma:

```
public function buscarPeliculas(Request $request){  
    $titulo=$request['titulo'];  
    $pelicula=\moviexpert\Adminpelicula::titulo($titulo);
```

Se observa como la variable `pelicula` llama al método `título` que hemos creado en el modelo, pasando por parámetro la variable creada `$titulo`.

También hay que crear un formulario donde hay que indicar que el role sea `search` y utilizar el método Post. El atributo `action` indica el tipo de acción que va a realizar el formulario, en este caso la acción será `buscarpeliculas`.

```
<form class="navbar-form navbar-left col-xs-12" role="search" method="POST" action="/buscelpeliculas">  
    {{ csrf_field() }}  
    <div class="form-group col-xs-6 col-md-2">  
        <input type="text" name="titulo" class="form-control" placeholder="Buscar por título">  
    </div>  
    <button type="submit" class="btn btn-danger col-xs-3 col-md-1">Buscar</button>
```

## Pagination

El **pagination** es muy útil a la hora de mostrar datos. Si tenemos una tabla con muchos registros, es imposible leerlos todos y además quedaría una página pesada, por eso gracias al método `paginate` de Laravel podemos realizar de forma sencilla un **pagination**, donde se puede escoger la cantidad de registros que se quiera mostrar por página. El **pagination** no aparece si no tiene los registros mínimos, esto quiere decir, que si ponemos un **pagination** de 5 y yo tengo 4 registros, no aparecería el **pagination**, pero al visualizar el sexto dato, entonces si aparecerá. Como por ejemplo aquí, en la tabla de películas, aparece porque hay un **pagination** de 5 y hay más registros.



Gracias al método `render` se crea una vista con **Bootstrap**, fácilmente personalizable, y por la clase activa nos marca en qué página estamos, así hace una navegación fácil, porque no te pierdes. En este caso el **pagination** nos indica que estamos en la página 1.

Para poder añadir un **pagination** a nuestra página, gracias a Laravel hacer un **pagination** es fácil, basta con añadir a la función del controlador donde queremos que aparezca, el método `paginate` como se puede ver en la imagen de a continuación.

```
$pelicula=DB::table('adminpeliculas')->orderBy('titulo')->paginate(5);
```

Con esto a partir de que tuviéramos 6 registros, sólo aparecerían los 5 primeros, pero sólo con esto no basta, se necesita construir visualmente lo que es el **pagination** para poder navegar por las distintas páginas. Para ello se llama al método

**render** que viene con **Bootstrap** en la vista donde queremos insertar el **pagination**. En la variable **película** tenemos todos los campos de la base de datos, ordenados gracias al **orderBy** por título, y visualizados de 5 en 5. Viene un estilo definido por **Bootstrap**, pero se puede personalizar el diseño en el CSS, con la clase **pagination**.

```
[!! $pelicula->render() !!]
```

## Protección CSRF

Laravel hace que sea fácil de proteger la solicitud de petición de sitios cruzados y ataques (**CSRF**). **Cross-site** son un tipo malicioso mediante el cual los comandos no autorizados se realizan en nombre de un usuario autenticado.

Laravel genera automáticamente un "**token**" **CSRF** para cada sesión de usuario activo administrado por la aplicación. Este símbolo se utiliza para verificar que el usuario autenticado es el que realmente hacer las peticiones a la aplicación.

Cada vez que se define un formulario HTML en la aplicación, se debe incluir un campo **token CSRF** oculto en el formulario para que el middleware de **protección CSRF** puede validar la solicitud, si no se hace así nos puede dar un error como este.

```
1/1 TokenMismatchException in VerifyCsrfToken.php line 67:
```

Esto se soluciona poniendo en los formularios esto:

```
<h1 class="text-center">Listado usuarios</h1><br>
<form class="navbar-form navbar-left col-xs-12" role="search" method="POST" action="/buscarusuarios">
    {{ csrf_field() }}
```

Introduciendo eso en cada formulario se puede resolver el problema. Si no se soluciona debemos borrar la cache de Laravel mediante el comando **php artisan cache:clear**, este comando se ejecuta en la consola de comandos dentro de la ruta de nuestro proyecto, borrar la cache y las cookies del navegador o abrir una página en modo incognito en Google Chrome.

## 7. Conclusión

Con este proyecto hemos conseguido obtener una plataforma para la gestión y visualización del campo cinematográfico donde los usuarios pueden ver las películas de cartelera, ranking de películas, participar en concursos, chatear en grupos privados limitados, etc. Esta plataforma se puede ampliar o servir de punto de partida de otros proyectos similares como podría ser una web de videojuegos, música,... Para este proyecto hemos aprendido a utilizar herramientas como sería el framework de Laravel o la herramienta de sincronización de proyectos git, herramientas de diseño gráfico para crear nuestras propias imágenes.

## 8. Propuestas de ampliación o mejoras.

Una vez finalizada esta primera versión se proponen las siguientes mejoras de la aplicación:

- Ampliación de la funcionalidad del backend
- Ampliación de la funcionalidad del frontend
- Desarrollar aplicación para dispositivos móviles
- Implementar funcionalidad en dispositivos móviles para poder realizar tareas de administración.

## 9. Webgrafía

- <https://laravel.com>
- <https://laravelcollective.com/>
- <https://github.com/AliciaPeris/moviexpert>
- <http://getbootstrap.com/>
- <http://fontawesome.io/icons/>
- <https://www.uno-de-piera.com/plantillas-blade-en-laravel/>
- <https://laracasts.com/>
- <https://laraveles.com/foro/viewtopic.php?id=3087>
- [https://www.youtube.com/watch?v=togljDT95wo&list=PLIdmSRJEJ0u-5Nv2k6W8Vhe0wUP\\_7H5W](https://www.youtube.com/watch?v=togljDT95wo&list=PLIdmSRJEJ0u-5Nv2k6W8Vhe0wUP_7H5W)
- <https://www.youtube.com/watch?v=pOdWEoUTwD4&list=PLEtcGQaT56chQN6n4byqRIMYnTHpgkG6m>
- <https://www.youtube.com/watch?v=Zj0pshSSIEo>
- <https://www.youtube.com/watch?v=hd3wKNo-htY>
- <https://laravel.com/docs/5.2/authentication>
- <https://stackoverflow.com/questions/34866404/tokenmismatchexception-in-verifycsrf-token-php-line-67>