

# Memoria de la práctica

## DESCRIPCION DEL PROYECTO:

El propósito de este proyecto es la creación de una especie de red social en la que cuando un usuario haga login tenga acceso a sus fotos, eliminarlas, subir nuevas o comentar y dar like en las publicaciones de otros usuarios.

Esto se realizará mediante un CRUD con Laravel.

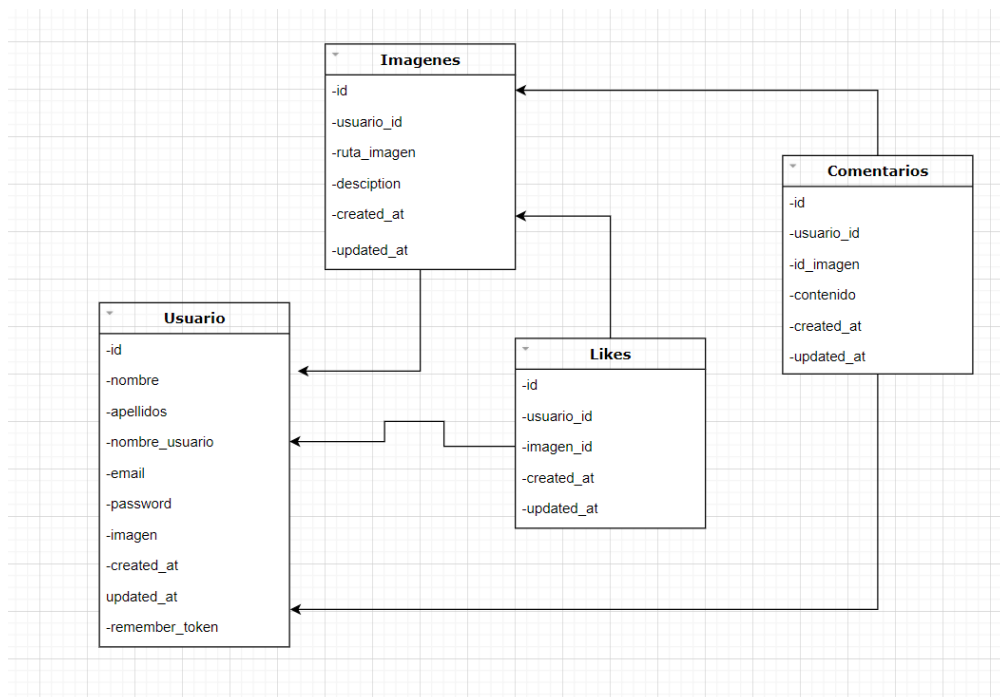
En el sistema habrá dos tipos de usuarios: administradores y usuarios simples. En la tabla posterior se esquematizarán dichos roles.

ROL	ACCIONES
USUARIO	Dar like y comentar en fotos. Subir imágenes
ADMINISTRADOR	Dar like, comentar, editar imágenes. Subir imágenes

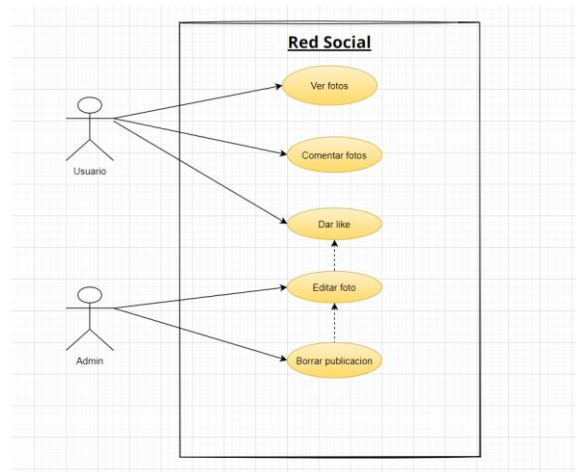
Como resumen se puede decir que lo que se intenta realizar es que un usuario haga login y que dependiendo de su rol pueda hacer unas acciones u otras como se ha indicado anteriormente. Esto se hará utilizando una vista general e incluyendo las vistas necesarias que contienen dichas acciones.

Dependiendo de la acción elegida se dirigirá mediante una ruta a un formulario o acción definida en los controladores de cada uno de los componentes del sistema.

## DIAGRAMA ENTIDAD RELACION:



## DIAGRAMA DE CASOS DE USO:



## CONFIGURACIONES INICIALES:

En un primer momento se creará el proyecto, el cual lo crearé en principio con xampp para usar la base de datos.

Una vez finalizada la creación del proyecto y de la base de datos, haremos que nuestro proyecto utilice la base de datos que hemos creado especificándolo en el archivo .env, situado en la carpeta raíz del proyecto.

Para comprobar que la conexión con la base de datos se ha realizado correctamente se lanzará el comando **php artisan migrate**

## CREACION DE LA PÁGINA DE LOGIN

Lo primero que se hará será instalar Laravel UI

```
D:\xampp\htdocs\proyectos\Laravel\Proyecto-Final>composer require laravel/ui
Using version ^3.1 for laravel/ui
./composer.json has been updated
Running composer update laravel/ui
```

A continuación se generará el código para realizar el login (scaffolding)

```
D:\xampp\htdocs\proyectos\Laravel\Proyecto-Final>php artisan ui vue --auth
Vue scaffolding installed successfully.
Please run "npm install && npm run dev" to compile your fresh scaffolding.
Authentication scaffolding generated successfully.
```

Para poder darle estilos a nuestra página de login vamos a lanzar el comando

**npm install && npm run dev.**

La configuración del Login se realizará en otros apartados del proyecto.

## UTILIZACION DE MIGRACIONES PARA CREAR LAS TABLAS:

Primeramente lanzaremos los comandos:

```
php artisan make:migration create_usuarios_table
php artisan make:migration create_comentarios_table
php artisan make:migration create_imagenes_table
php artisan make:migration create_likes_table
```

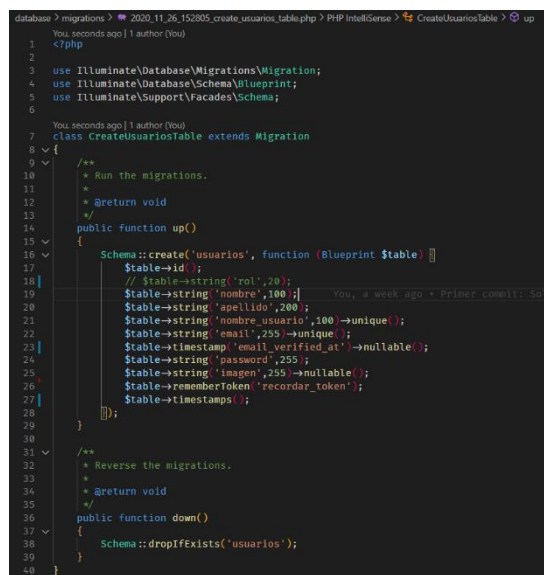
Decir que se hace para todas las tablas, incluida la de usuarios ya que queremos que se use esa para el proyecto y no la que el sistema crea por defecto. Tras ello se crearán las migraciones en el proyecto.

Lo siguiente que se hará es modificar estos archivos para darle los campos deseados a las tablas.

Empezaré por la tabla usuarios ya que esta no depende de ninguna otra tabla de nuestra base de datos, por lo tanto, es lo ideal para ir creando las tablas en la base de datos para que no produzca problemas.

Tras cada cambio en estos ficheros refrescaremos la migración con **php artisan migrate:refresh** en orden de que si añadimos una tabla que dependa de otra, no haya problemas tampoco.

Empezaré por usuarios y se establecerá que el nombre de usuario sea único, en orden de que no se puedan repetir nombres de usuarios en la red social



```
database > migrations > 2020_11_26_152805_create_usuarios_table.php > PHP IntelliSense > CreateUsuariosTable > up
1 You, seconds ago | 1 author (You)
2 <?php
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 You, seconds ago | 1 author (You)
8 class CreateUsuariosTable extends Migration
9 {
10     /**
11      * Run the migrations.
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('usuarios', function (Blueprint $table) {
17             $table->id();
18             $table->string('rol', 20);
19             $table->string('nombre', 100);
20             $table->string('apellido', 200);
21             $table->string('nombre_usuario', 100)->unique();
22             $table->string('email', 255)->unique();
23             $table->timestamp('email_verified_at')->nullable();
24             $table->string('password', 255);
25             $table->string('imagen', 255)->nullable();
26             $table->rememberToken('recordar_token');
27             $table->timestamps();
28         });
29     }
30
31     /**
32      * Reverse the migrations.
33      * @return void
34      */
35     public function down()
36     {
37         Schema::dropIfExists('usuarios');
38     }
39 }
40
```

En el resto de tablas se hará lo mismo dependiendo de los campos que se quieran en la base de datos. Como en estas tablas tendremos que indicar que hay campos foráneos usaremos :

```
$table->foreignId('')->index()->constrained()->onDelete('cascade');
```

```

2020_11_26_153058_create_comentarios_table.php
database > migrations > 2020_11_26_153058_create_comentarios_table.php > PHP IntelliSense > CreateComentariosTable
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class CreateComentariosTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('comentarios', function (Blueprint $table) {
17             $table->id();
18             $table->foreignId('usuario_id')->constrained('usuarios');
19             $table->foreignId('imagen_id')->constrained('imagenes');
20             $table->text('contenido_comentario');
21             $table->timestamps();
22         });
23     }
24
25     /**
26      * Reverse the migrations.
27      *
28      * @return void
29      */
30     public function down()
31     {
32         Schema::dropIfExists('comentarios');
33     }
34 }
35
36

```

Comentarios

```

2020_11_26_152932_create_imagenes_table.php
database > migrations > 2020_11_26_152932_create_imagenes_table.php > PHP IntelliSense > CreateImagenesTable > up > C
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class CreateImagenesTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('imagenes', function (Blueprint $table) {
17             $table->id();
18             $table->foreignId('usuario_id')->constrained('usuarios');
19             $table->string('ruta_imagen',255);
20             $table->text('descripcion');
21             $table->timestamps();
22         });
23     }
24
25     /**
26      * Reverse the migrations.
27      *
28      * @return void
29      */
30     public function down()
31     {
32         Schema::dropIfExists('imagenes');
33     }
34 }
35

```

Imágenes

```

2020_11_26_153142_create_likes_table.php
database > migrations > 2020_11_26_153142_create_likes_table.php > PHP IntelliSense > CreateLikesTable
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class CreateLikesTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         Schema::create('likes', function (Blueprint $table) {
17             $table->id();
18             $table->foreignId('usuario_id')->constrained('usuarios');
19             $table->foreignId('imagen_id')->constrained('imagenes');
20             $table->timestamps();
21         });
22     }
23
24     /**
25      * Reverse the migrations.
26      *
27      * @return void
28      */
29     public function down()
30     {
31         Schema::dropIfExists('likes');
32     }
33 }
34

```

Likes

En caso de que todo sea correcto, la base de datos se actualizará y ya se tendrán las tablas creadas en ella.

## CREACION Y CONFIGURACION DE MODELOS:

En este paso me centraré en crear los modelos necesarios para el proyecto.

Aunque el modelo User (modelo para los usuarios de mi aplicación) ya viene creado por defecto al crear el proyecto, se necesitan el de Imagen, Comentario y Like. Para ello se ejecutará el comando: **php artisan make:model nombre\_modelo**.

Tras ello se van a configurar las entidades y relaciones de cada uno de estos modelos.

Empezaré por el modelo User ya que es el más sencillo. En él se especifica que un usuario tiene más de una imagen. Este modelo por defecto coje la tabla users que se crea en la primera migración, sin embargo quiero que utilice la tabla usuarios creada por mí, por lo tanto también habrá que indicarlo.

```

1 namespace App\Models;
2
3 use Illuminate\Contracts\Auth\Authenticatable;
4 use Illuminate\Database\Eloquent\Factories\HasFactory;
5 use Illuminate\Foundation\Auth\User as Authenticatable;
6 use Illuminate\Notifications\Notifiable;
7
8 /**
9  * @property string $name
10  */
11 class User extends Authenticatable
12 {
13     use HasFactory, Notifiable;
14
15     /**
16      * The attributes that are mass assignable.
17      *
18      * @var array
19      */
20     protected $fillable = [
21         'name',
22         'email',
23         'password',
24     ];
25
26     /**
27      * The attributes that should be hidden for arrays.
28      *
29      * @var array
30      */
31     protected $hidden = [
32         'password',
33         'remember_token',
34     ];
35
36     /**
37      * The attributes that should be cast to native types.
38      *
39      * @var array
40      */
41     protected $casts = [
42         'email_verified_at' => 'datetime',
43     ];
44
45     /**
46      * Get the images for the user.
47      *
48      * @return \Illuminate\Database\Eloquent\Relations\HasMany
49      */
50     public function images()
51     {
52         return $this->hasMany('App\Models\Imagen');
53     }
54 }

```

Tras ello procederé a modificar el modelo Comentario, en el que se establecerá que un determinado comentario pertenece a una sola imagen y a un solo usuario, para ello usaré **belongsTo()**. También se establecerá que usará la tabla de la BD comentarios.

La estructura de este modelo será la siguiente:

```

app > Models > Comentario.php > ...
You, a minute ago | 1 author (You)
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 /**
9  * @property string $name
10  */
11 class Comentario extends Model
12 {
13     use HasFactory;
14     //Especifico la tabla de la BD que va a utilizar este modelo → Comentarios
15     protected $table = 'comentarios';
16
17     //Que un comentario pertenezca a un solo usuario
18     public function usuario(){
19         return $this->belongsTo('App\Models\User','id_usuario');
20     }
21
22     //Que un comentario pertenezca a una sola imagen
23     public function imagen(){
24         return $this->belongsTo('App\Models\Imagen','id_imagen');
25     }
26 }

```

En el modelo Imagen se establecerá, al igual que anteriormente, la tabla que se va a usar de la BD y las relaciones que está tiene con el resto de tablas y modelos. En este caso, la cantidad de la relación cambia en algunos casos, por lo tanto se utilizará **hasMany()**

```
app > Models > Imagen.php > ...
You, seconds ago | 1 author (You)
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 You, seconds ago | 1 author (You)
9 class Imagen extends Model
10 {
11     use HasFactory;
12     protected $table = 'imagenes';
13
14     //Una imagen → muchos comentarios
15     public function comentarios(){
16         return $this->hasMany('App\Models\Comentario');
17     }
18
19     //Una imagen → muchos likes
20     public function likes(){
21         return $this->hasMany('App\Models\Like');
22     }
23
24     //Una imagen → un solo usuario
25     public function usuario(){
26         return $this->belongsTo('App\Models\User', 'id_usuario');
27     }
28 }
```

Por último, en el modelo Like, se establecerá también la tabla a la que hace referencia así como la relaciones que tiene con los demás modelos.

```
app > Models > Like.php > PHP Intelephense > Like
You, seconds ago | 1 author (You)
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 You, seconds ago | 1 author (You)
9 class Like extends Model
10 {
11     use HasFactory;
12     //Especifico la tabla de la BD que va a utilizar este modelo → Likes
13     protected $table = 'likes';
14
15     //Muchos likes → un usuario
16     public function usuario(){
17         return $this->belongsTo('App\Models\User', 'id_usuario');
18     }
19
20     //Muchos likes → una imagen
21     public function imagen(){
22         return $this->belongsTo('App\Models\Imagen', 'id_imagen');
23     }
24
25 }
```

## IMPLEMENTACION DEL LOGIN

Lo primero que se hará será una nueva migración para darles a los usuarios un rol con el comando `php artisan make:migration columna_rol_usuarios_table`

```
migrated: 2020_11_26_155142_create_likes_table (3,624.36ms)
D:\xampp\htdocs\proyectos\Laravel\Proyecto-Trimestre>php artisan make:migration columna_rol_usuarios_table
Created Migration: 2020_12_03_082456_columna_rol_usuarios_table
```

En el archivo que se ha generado debemos tener finalmente la siguiente estructura:

```

1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class ColumnaRolUsuariosTable extends Migration
8 {
9     /**
10      * Run the migrations.
11      *
12      * @return void
13      */
14     public function up()
15     {
16         //Del esquema coge la tabla usuarios y en el campo rol le doy un dato entre
17         //user, manager o admin. Por defecto será user
18         Schema::table('usuarios',function (Blueprint $table){
19             $table->enum('rol',['user','manager','admin'])->default('user');
20         });
21     }
22
23     /**
24      * Reverse the migrations.
25      *
26      * @return void
27      */
28     public function down()
29     {
30         //
31     }
32 }
33

```

En la parte marcada lo que se hace es coger del esquema la tabla usuarios y en especificar que para el campo rol habrá tres posibilidades: user y admin. Por defecto el valor será user.

Lo siguiente que se hará será Modificar la factoría UserFactory, la cual viene definida por defecto en el proyecto. Con ella lo que se hará será indicar los elementos con los que se va a rellenar la tabla usuarios del proyecto.

```

1 You, seconds ago | 1 author (You)
2 <?php
3
4 namespace Database\Factories;
5
6 use App\Models\User;
7 use Illuminate\Database\Eloquent\Factories\Factory;
8 use Illuminate\Support\Str;
9 use Illuminate\Support\Facades\Hash;
10
11 You, seconds ago | 1 author (You)
12 class UserFactory extends Factory
13 {
14     /**
15      * The name of the factory's corresponding model.
16      *
17      * @var string
18      */
19     protected $model = User::class;
20
21     /**
22      * Define the model's default state.
23      *
24      * @return array
25      */
26     public function definition()
27     {
28         return [
29             'rol' => $this->faker->randomElement(['admin','user','manager']),
30             'nombre' => $this->faker->name,
31             'apellido' => $this->faker->lastName,
32             'nombre_usuario' => $this->faker->userName,
33             'email' => $this->faker->unique()->safeEmail,
34             'password' => Hash::make('1234'), // password
35             'imagen' => $this->faker->word(10).'.jpg',
36             'email_verified_at' => now(),
37             'fecha_creacion' => $this->faker->dateTime(),
38             'fecha_actualizacion' => $this->faker->dateTime(),
39             'remember_token' => Str::random(10),
40         ];
41     }
42 }

```

En este punto, con la modificación del archivo DatabaseSeeder, lo que se hará es indicar a la factoría de un usuario (el archivo anterior), que nos cree 13 usuarios con sus datos.

```

1  UserFactory.php
2  DatabaseSeeder.php X
3  User.php
4
5  database > seeders > DatabaseSeeder.php > ...
6  You, seconds ago | 1 author (You)
7  <?php
8
9  namespace Database\Seeders;
10
11 use Illuminate\Database\Seeder;
12 use \App\Models\User;
13
14 class DatabaseSeeder extends Seeder
15 {
16     /**
17      * Seed the application's database.
18      *
19      * @return void
20      */
21     public function run()
22     {
23         // \App\Models\User::factory(10)→create();
24         User::factory(13)→create();
25     }
26 }

```

Para rellenar la base de datos, lanzaremos el comando `php artisan migrate:refresh --seed`

```
D:\xampp\htdocs\proyectos\Laravel\Proyecto-Trimestre>php artisan migrate:refresh --seed
Nothing to rollback.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (598.14ms)
```

Tras ello, la tabla usuarios de la base de datos tendrá datos generados aleatoriamente gracias al uso de factory

Filas 0 - 12 (total de 13, La consulta tardó 0.0177 segundos)											
usuarios*											
<input type="checkbox"/> Perfilando <a href="#">[Editar en línea]</a> <a href="#">[Editar]</a> <a href="#">[Explicar SQL]</a> <a href="#">[Crear código PHP]</a> <a href="#">[Actualizar]</a>											
Orden	Número de filas:	25	Filtrar filas:	Buscar en esta tabla	Sort by key:	Ninguna					
	ID	nombre	apellido	nombre_usuario	email	password	imagen	remember_token	created_at	updated_at	roles
		Alyana Larson	Schuster	edare	esther64@example.com	\$2y\$10\$MKnqKv4c5Vj3ZdeYEPOR5n9cd1z3vZwCEXKm3.	et.jpg	DzG2GL1c7dOluvt5Cs4JaeS4CDCA6gT8Nv3pZvRu6b..	2010-11-03 19:18	1995-03-07 16:22	ad
		Mastie Overmunnor DDS	VonRueden	Wlame okunwa	enola.bailey@example.net	\$2y\$10\$Xoa5dFTXps9FWUeJL SZ6zvZh6dKRDK4dcOGP.	sit.jpg	Oq64smfSS	2005-10-23 19:51	1986-01-16 12:27	31 usu
		Julien McDermott	Garcas	kondicka	sbruens@example.com	\$2y\$10\$URjKAACEwReC10nfuzVB0tVe5vrGrZ1NNxSK1OfqBS.	aut.jpg	DjG8fyBu	2001-03-27 08:18	1979-01-11 23:19	24 ma
		Matthew Ebert	Nicolas	kertzmann.lucas	thansen@example.net	\$2y\$10\$bt6pE1I1NVSldQp3Sz ay7ghLHV9JFPegHtJkL	ellegend.jpg	z5ngheDcj	2005-05-19 04:58	1984-04-03 04:18	25 ad
		Linda Armstrong	Rice	jackeline.haldemairich	kish.tanner@example.com	\$2y\$10\$4WnTi IFTH5wQY8BRi QbaDuA1H F6vDFrhcwHu5uM	eat.jpg	eAf7FucFEYT	1987.5.13 17:22	2012.05.10 08:27	10

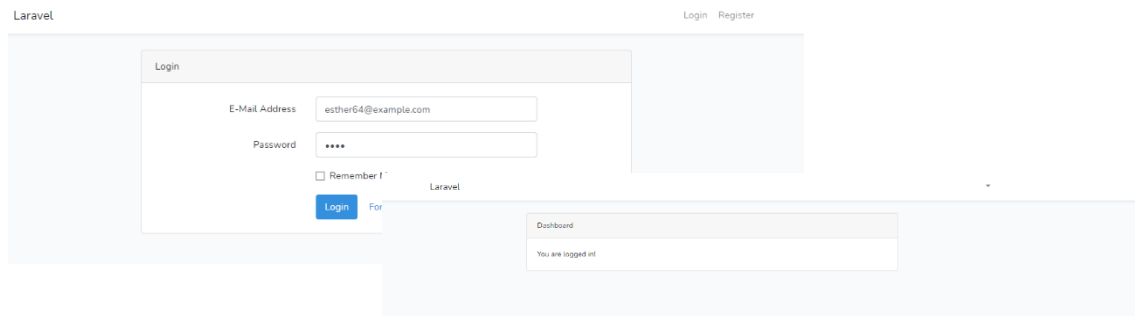
Ahora se va a probar que un usuario definido en la base de datos puede que entrar y que por el contrario uno que no está no tiene acceso a la aplicación.

Primeramente probaré con el usuario con id 1 de la tabla de usuarios:

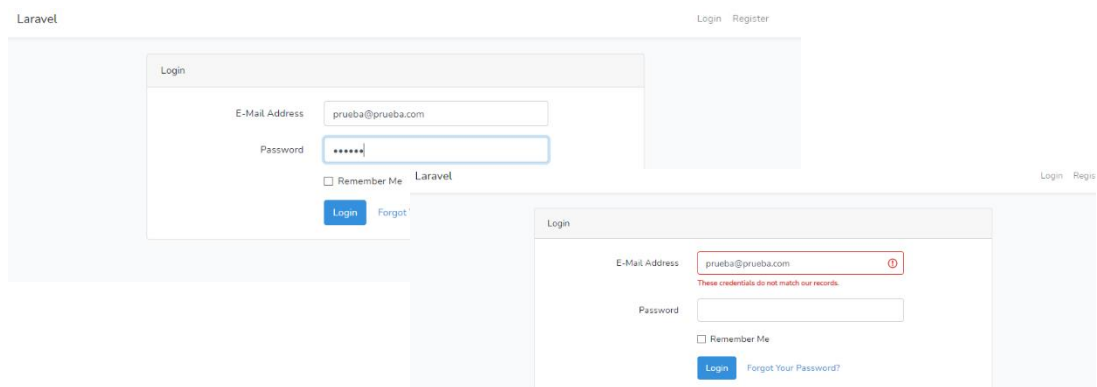
id	nombre	apellido	nombre_usuario	email	password	imagen	remember_token	created_at	updated_at	rol
1	Alyana	Schuster	edare	eshter4@example.com	\$2y\$10\$MvKmfV4c5vY3Zdy9\$EPOr5xh8cd18z2vX6Ym3x	et.jpg	Dc2b76Lg1c7d0havetScx4Ja5e94CD6A9p78m3pZvNru8h	2010-11-11 03:19:18	1995-03-13 07:16:22	admin



Como este usuario está definido en nuestra tabla, al introducir sus datos en el login, aparece que ha iniciado sesión



Finalmente se hará la prueba con el usuario [prueba@prueba.com](mailto:prueba@prueba.com) y contraseña prueba. Como se intenta acceder mediante un usuario no definido el login lanzará un error.



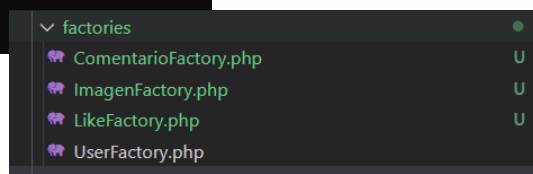
Las vistas asociadas a cada tipo de usuario se detallarán más adelante.

## LLENADO DE TABLAS CON SEEDERS

Creación de factories de cada una de las tablas de la base de datos. En este caso, el factory de usuario ya está creado del punto anterior, por lo que se crearán el de Likes, Comentarios e Imágenes.

```
Php artisan make:factory LikeFactory
Php artisan make:factory ImagenFactory
Php artisan make:factory ComentarioFactory
```

```
D:\xampp\htdocs\proyectos\Laravel\Proyecto-Trimestre>php artisan make:factory LikeFactory
Factory created successfully.
D:\xampp\htdocs\proyectos\Laravel\Proyecto-Trimestre>php artisan make:factory ComentarioFactory
Factory created successfully.
D:\xampp\htdocs\proyectos\Laravel\Proyecto-Trimestre>php artisan make:factory ImagenFactory
Factory created successfully.
D:\xampp\htdocs\proyectos\Laravel\Proyecto-Trimestre>
```



Como he indicado, el archivo factory de usuarios ya estaba creado desde el punto anterior, por lo que empezaré por el de Imágenes.

Las tablas de la base de datos tienen la peculiaridad de que están unidas entre ellas mediante claves foráneas, por lo tanto hay que indicar que los datos pertenecientes a estos campos se equiparen con los de la otra tabla a la que hacen referencia. Para ello lo haremos de la siguiente manera:

```
'nombre_campo' => \App\Models\Modelo_otra_tabla::all()->random()->campo_referencia
```

- **Nombre\_campo:** Indica el campo de la tabla que tiene una clave foránea.
- **Modelo\_otra\_tabla:** Hace referencia al modelo de la tabla a la que se está haciendo referencia en dicho campo
- **Campo\_referencia:** Campo de la otra tabla a la que el campo hace referencia

```
database > factories > ImagenFactory.php > PHP Intelephense > ImagenFactory > definition
1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Imagen;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7
8 class ImagenFactory extends Factory
9 {
10     /**
11      * The name of the factory's corresponding model.
12      *
13      * @var string
14      */
15     protected $model = Imagen::class;
16
17     /**
18      * Define the model's default state.
19      *
20      * @return array
21      */
22     public function definition()
23     {
24         return [
25             //
26             'usuario_id' => \App\Models\User::all()->random()->id,
27             'ruta_imagen' => $this->faker->imageUrl(),
28             'description' => $this->faker->text(),
29         ];
30     }
31 }
32
33 }
```

En segundo lugar completaremos el fichero referente al factory de los comentarios

```
database > factories > ComentarioFactory.php > PHP Intelephense > ComentarioFactory > definition
1 <?php
2
3 namespace Database\Factories;
4
5 use App\Models\Comentario;
6 use Illuminate\Database\Eloquent\Factories\Factory;
7
8 class ComentarioFactory extends Factory
9 {
10     /**
11      * The name of the factory's corresponding model.
12      *
13      * @var string
14      */
15     protected $model = Comentario::class;
16
17     /**
18      * Define the model's default state.
19      *
20      * @return array
21      */
22     public function definition()
23     {
24         return [
25             //
26             'usuario_id' => \App\Models\User::all()->random()->id,
27             'imagen_id' => \App\Models\Imagen::all()->random()->id,
28             'contenido_comentario' => $this->faker->realText(50),
29         ];
30     }
31 }
32
33 }
```

Y por último el referente a los likes:

```
database > factories > LikeFactory.php > ...
1  <?php
2
3  namespace Database\Factories;
4
5  use App\Models\Like;
6  use Illuminate\Database\Eloquent\Factories\Factory;
7
8  class LikeFactory extends Factory
9  {
10     /**
11      * The name of the factory's corresponding model.
12      *
13      * @var string
14      */
15     protected $model = Like::class;
16
17     /**
18      * Define the model's default state.
19      *
20      * @return array
21      */
22     public function definition()
23     {
24         return [
25             //
26             'usuario_id' => \App\Models\User::all()->random()->id,
27             'imagen_id' => \App\Models\Imagen::all()->random()->id,
28         ];
29     }
30 }
31
```

En estos ficheros hemos utilizado faker

Lo siguiente que se hará para que estos datos lleguen a la base de datos será modificar el fichero DatabaseSeeder, en él se indicará que datos se quieren crear y que cantidad de los mismos.

En mi caso en un primer momento crearé 10 usuarios ,20 imágenes,30 comentarios y 60 comentarios.

```
database > seeders > DatabaseSeeder.php > ...
1  ...
2  <?php
3
4  namespace Database\Seeders;
5
6  use Illuminate\Database\Seeder;
7  use \App\Models\User;
8
9  ...
10 class DatabaseSeeder extends Seeder
11 {
12     /**
13      * Seed the application's database.
14      *
15      * @return void
16      */
17     public function run()
18     {
19         //En estas líneas especifico que factory quiero utilizar y cuantos datos
20         //de cada tipo quiero introducir en la BD
21         \App\Models\User::factory(10)->create();
22         \App\Models\Imagen::factory(20)->create();
23         \App\Models\Comentario::factory(30)->create();
24         \App\Models\Like::factory(60)->create();
25     }
26 }
27
28
29
```























Una vez realizado esto se lanzará el comando `php artisan migrate:refresh --seed`

```
D:\xampp\htdocs\proyectos\Laravel\Proyecto-Trimestre>php artisan migrate:refresh --seed
Rolling back: 2020_12_03_082456_columna_rol_usuarios_table
Rolled back: 2020_12_03_082456_columna_rol_usuarios_table (119.51ms)
```

En caso de que el resultado sea satisfactorio veremos que todas las tablas en la base de datos estarían pobladas.

Opciones															
			id	nombre	apellido	nombre_usuario	email	email_verified_at	password	imagen	remember_token	created_at	update		
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	1	Jaunita Purdy Crooks	abergstrom	kwolf@example.net	2020-12-06 20:34:40	\$2y\$10\$aBaU8ggyqKqV4B3PDZYB6WucrtS7p6YLTSoYvu...	totam.jpg	WzAKKRNhG	1998-03-31 18:32:50	1981-0
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	2	Anahi Lueslitz Harris	smitham nils	bcole@example.com	2020-12-06 20:34:40	\$2y\$10\$JFwtz3pTrH8DJKnpDwLkOghr3aHnADY6jglo7nq...	nobis.jpg	wJa580v48l	2017-02-12 08:41:41	2008-0
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	3	Mr. Silas Hinz III Beatty	mekhl07	sawayn.lilian@example.com	2020-12-06 20:34:40	\$2y\$10\$uEva34d8AdxKGFNPQL01l.8ml5bNcYZqloL7c56kLm...	eligendi.jpg	NnQ0fm4jf	2017-12-19 06:21:57	2006-0
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	4	Kathryn Murazik Jaskolski	shakira53	dtorphy@example.org	2020-12-06 20:34:41	\$2y\$10\$1fPZhKz40Ec0e0k3lAZ5u oD9JnXHGvPPhyAWV1TGS...	commodi.jpg	78FmhgtnGh	1983-08-29 08:05:50	1995-0

## Usuarios

+ Opciones			id	usuario_id	imagen_id	created_at	updated_at	
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1	10	1	2020-12-06 20:34:46	2020-12-06 20:34:46
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	2	7	20	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	3	5	8	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	4	4	15	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	5	2	15	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	6	1	20	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	7	9	16	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	8	4	6	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	9	5	11	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	10	2	9	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	11	3	12	2020-12-06 20:34:47	2020-12-06 20:34:47
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	12	1	20	2020-12-06 20:34:48	2020-12-06 20:34:48
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	13	6	20	2020-12-06 20:34:48	2020-12-06 20:34:48
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	14	3	15	2020-12-06 20:34:48	2020-12-06 20:34:48

## Likes

+ Opciones			id	usuario_id	ruta_imagen	description	created_at	updated_at			
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	1	6	https://via.placeholder.com/640x480.png/00ccce?tex...	Rerum est dolor corrupti ex et. Expedita fuga quia...	2020-12-06 20:34:42	2020-12-06 20:34:42
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	2	5	https://via.placeholder.com/640x480.png/002222?tex...	Omnis placeat labore quod consequatur facilis non...	2020-12-06 20:34:42	2020-12-06 20:34:42
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	3	1	https://via.placeholder.com/640x480.png/00dd33?tex...	Possimus est non odio esse nam quibusdam. Ratione...	2020-12-06 20:34:42	2020-12-06 20:34:42
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	4	8	https://via.placeholder.com/640x480.png/007799?tex...	Est dolorem commodi veniam eius cupiditate necessi...	2020-12-06 20:34:42	2020-12-06 20:34:42
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	5	10	https://via.placeholder.com/640x480.png/0033bb?tex...	Inventore similique aliquam excepturi quasi rem. P...	2020-12-06 20:34:43	2020-12-06 20:34:43
<input type="checkbox"/>	Editar	<input type="checkbox"/>	Copiar	<input type="checkbox"/>	Borrar	6	1	https://via.placeholder.com/640x480.png/00dd77?tex...	Ad et neque placeat odit voluptas eligendi praesen...	2020-12-06 20:34:43	2020-12-06 20:34:43

## Imágenes

		id	usuario_id	imagen_id	contenido_comentario	created_at	updated_at
<input type="checkbox"/>	<div>Editar</div>	1	7	16	King. The next witness was the BEST butter,' the	2020-12-06 20:34:44	2020-12-06 20:34:44
<input type="checkbox"/>	<div>Editar</div>	2	15	15	NOT marked 'poison,' so Alice ventured to.	2020-12-06 20:34:44	2020-12-06 20:34:44
<input type="checkbox"/>	<div>Editar</div>	3	9	15	Alice, and sighing. 'It IS a long and a long.	2020-12-06 20:34:44	2020-12-06 20:34:44
<input type="checkbox"/>	<div>Editar</div>	4	10	2	King repeated angrily, 'or I'll have you.	2020-12-06 20:34:44	2020-12-06 20:34:44
<input type="checkbox"/>	<div>Editar</div>	5	7	10	I shan't grow any more--As it is, I suppose?'	2020-12-06 20:34:44	2020-12-06 20:34:44
<input type="checkbox"/>	<div>Editar</div>	6	9	12	And the Gryphon went on at last, they must be a.	2020-12-06 20:34:44	2020-12-06 20:34:44
<input type="checkbox"/>	<div>Editar</div>	7	4	19	Gryphon repeated impatiently: 'it begins 'I.	2020-12-06 20:34:45	2020-12-06 20:34:44
<input type="checkbox"/>	<div>Editar</div>	8	8	6	After a while she remembered that she was quite.	2020-12-06 20:34:45	2020-12-06 20:34:44
<input type="checkbox"/>	<div>Editar</div>	9	7	17	Alice cautiously replied: 'but I know all the.	2020-12-06 20:34:45	2020-12-06 20:34:44

## Comentarios

### CREACIÓN DE CONTROLADORES:

En este paso se van a crear los controladores de cada uno de los modelos, para ello utilizaremos el comando `php artisan make:controller nombre_controlador`

Comenzaré a comentar el controlador de comentarios, en este se ha implementado simplemente la función store para almacenar datos en la base de datos los cuales recibe de un formulario. Antes de ello se validarán los datos usando Validator.

```

public function store(Request $request)
{
    $validator = Validator::make($request->all(), [
        'usuario' => 'required',
        'imagen' => 'required',
        'contenido' => 'required'
    ]);

    if ($validator->fails()) {
        return redirect('/')->withErrors($validator)->with('error',
    }

    $comentario = new Comentario();

    $comentario->usuario_id = $request->usuario;
    $comentario->imagen_id = $request->imagen;
    $comentario->contenido_comentario= $request->contenido;

    $comentario->save();

    return redirect()->route('home')->with('success', 'Se ha actua
}

```

Otro de los controladores es el de las imágenes. En este se han definido las siguientes funciones:

- Index → Coge las rutas de todas las imágenes y las muestra.
- Create → Llama a una vista que hace de formulario para crear los mismos.
- Store → Recibe los datos de formulario anterior y los introduce en la base de datos tras validarlos.
- Edit → Devuelve una vista que hace de formulario para editar la descripción de una imagen.
- Update → Recoge los datos del formulario anterior y tras validar los datos, modifica ese campo en la base de datos.
- Destroy → Elimina la imagen seleccionada.

#### PROVIDERS:

Con esto lo que se hace es definir el tipo de usuario que es el que se ha logueado, para así establecer las características de la aplicación deseables para cada rol. En mi caso simplemente hay dos roles, el de administrador y el de un usuario común.

Al iniciarse la aplicación se comprueba el rol en **Providers>AuthServicePolicies** utilizando una función denominada boot la cual devuelve el mismo.

```

public function boot()
{
    $this->registerPolicies();

    /* define a admin user role */
    Gate::define('isAdmin', function($user) {
        return $user->rol == 'admin';});

    /* define a user role */
    Gate::define('isClient', function($user) {
        return $user->rol == 'user';});

    //
}

```

## RUTAS:

En este archivo se definirán las diferentes rutas que se van a usar en el proyecto. Mediante estas se realizarán acciones como las llamadas a formularios desde los botones de la aplicación o el paso de parámetros a funciones para realizar cambios en el sistema.

Está compuesto por una serie de declaraciones con distintos métodos (put,post,get...) en los cuales se define la ruta para llegar a un determinado recurso de la aplicación. En mi caso he indicado la ruta así como el controlador al que se dirige y el método en concreto que utiliza. También se le ha asignado un nombre para que su acceso sea más sencillo.

```
Auth::routes();

Route::get('/', [App\Http\Controllers\HomeController::class, 'index'])->name('home');

Route::resource('imagen',App\Http\Controllers\ImagenController::class);
Route::resource('like',App\Http\Controllers\LikeController::class);

// Route::get('comentarios/create/{imagen}/{usuario}',[App\Http\Controlle
rs\ComentarioController::class,'create'])->name('comentarios.create');
Route::post('comentarios',[App\Http\Controllers\ComentarioController::cla
ss,'store'])->name('comentarios.store');

Route::post('imagen/{imagen}/edit', [App\Http\Controllers\ImagenControlle
r::class, 'edit'])->name('imagenes.editar');
Route::put('imagen/{imagen}', [App\Http\Controllers\ImagenController::cla
ss, 'update',])->name('imagenes.update');
Route::post('imagenes/subir/{usuario}',[App\Http\Controllers\ImagenContro
ller::class,'create'])->name('imagenes.subir');
```