PROYECTO RED SOCIAL

DISEÑO DE LA BASE DE DATOS

Empezaremos el proyecto diseñando la base de datos. Se propone trabajar con las siguientes tablas:

Users

- Id
- Role
- Name
- Surname
- User_name
- Email
- Password
- Image
- Created_at
- Updated_at

Images

- Id
- User_id
- Image_path
- Description
- Created_at
- Updated_at

Comments

- Id
- User_id
- Image_id
- Content

- Created_at
- Updated_at

Likes

- Id
- User_id
- Image_id
- Created_at
- Updated_at

Esta es una propuesta, la base de datos que creéis en cada uno de vuestro proyecto no tiene por qué ser similar. Si consideráis que podéis complementarla o añadirle cualquier mejora no dudéis en hacerlo.

CREACIÓN DEL PROYECTO EN LARAVEL

Crea el proyecto a través de la terminal, el host virtual y el repositorio en Git Hub.

CREACIÓN DE LA BASE DE DATOS

Crea la base de datos a partir de PhpMyAdmin con el nombre que le des a tu red social. Después, conecta la base de datos con el proyecto creado.

Ahora tienes diferentes formas de generar las tablas, pero sería conveniente realizarla a través de una migración para poder duplicarla fácilmente, ya bien sea en otro servidor local o en otra base de datos. Además, trabajar con las migraciones también permite mantener un historial de las modificaciones y facilita el trabajo en equipo.

Creamos las migraciones de la siguiente forma:

```
PS C:\proyecto_rrss> php artisan make:migration create_users_table
```

Definimos los campos en el método up().

```php
public function up()
{
    Schema::create( table: 'users', function (Blueprint $table) {
        $table->increments( column: 'id');
        $table->string( column: 'role');
        $table->string( column: 'name');
        $table->string( column: 'surname');
        $table->string( column: 'user_name');
        $table->string( column: 'email');
        $table->string( column: 'password');
        $table->string( column: 'image');
        $table->timestamps();
    });
}
```

Para definir la clave primaria tenemos que utilizar el método primary(), sin embargo, el tipo de campo increments la asigna automáticamente.

Para asignar claves ajenas y poder relacionar las tablas:

```php
public function up()
{
    Schema::create('images', function (Blueprint $table) {
        $table->integer('id',1);
        $table->integer('user_id');
        $table->foreign('user_id')->references('id')->on('users');
        $table->string('image_path');
        $table->string('description');
        $table->timestamps();
    });
}
```

Completa la creación de tablas y los tipos de campo mediante las migraciones. En la siguiente dirección tienes más información:

https://laravel.com/docs/9.x/migrations

CREACIÓN DE MODELOS y RELACIONES

Para trabajar con modelos y entidades vamos a emplear el ORM de Laravel, que es Eloquent, que nos permitirá interactuar de una manera muy simple con nuestra base de datos.

Si bien en el anterior proyecto contábamos con una clase QueryBuilder y un repositorio para cada entidad o modelo, ahora únicamente tendremos que definir las relaciones entre las diferentes tablas y no preocuparnos por las consultas SQL. Así pues, con una serie de métodos que incorpora Eloquent podemos hacer un CRUD de nuestras tablas de una forma mucho más intuitiva y fácil.

Si bien recuerdas, las entidades representan a los registros que tenemos en la base de datos. Por lo tanto, si tenemos una entidad que se denomina “usuarios”, esta entidad representa a un registro de esta tabla. Se tiene entonces, un objeto por cada registro en la base de datos.

Por defecto viene en el proyecto de Laravel un modelo llamado User:

User.php

```php
<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;
use ...

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
```

Para crear el modelo, abrimos la consola y:

php artisan make:model User

El nombre del modelo empieza por mayúsculas y está en singular, ya que hace referencia a una entidad de la tabla 'Users'.

Crea un modelo para cada tabla que has creado.

Ahora vamos a pasar a la configuración de los modelos, es decir, a establecer las relaciones que hay entre las diferentes tablas de la base de datos. De esta forma podremos realizar consultas mediante Eloquent que tengan en cuenta diferentes tablas. Por ejemplo, al realizar una consulta puedo obtener el objeto imagen, y a su vez, sacar el usuario que ha subido la imagen o aquellos que han comentado, puesto que ya hemos indicado las relaciones en el modelo.

Empezamos con el modelo Image:

```php
class Image extends Model
{
    use HasFactory;

    protected $table = 'images';

    //Relación uno a muchos. Porque una foto va a poder tener más de un comentario y más de un like
    //Servirá para sacar los comentarios que tenga una imagen
    public function comments(){
        return $this->hasMany( related: Comment::class);
    }

    //Relación uno a muchos
    public function likes(){
        return $this->hasMany( related: Like::class);
    }

    //Relación muchos a uno, porque el mismo usuario puede crear muchas imagenes.
    public function user(){
        return $this->belongsTo( related: User::class, foreignKey: 'user_id');
    }
}
```

La variable $table se utiliza en el caso de que el nombre de la tabla no se corresponda con el plural del nombre del modelo.

Vamos a tener dos tipos de relaciones, uno a muchos y muchos a uno. Completa los demás modelos a partir de las relaciones que definimos al principio del documento.

RELLENO DE TABLAS

Vamos a utilizar los seeder, factories y la librería faker (dentro de los models factories) para rellenar las tablas con datos de prueba.

En primer lugar, creamos un seeder para cada modelo desde la consola.

En segundo lugar, creamos un Factory para cada uno de los modelos. A través de la consola:

php artisan make:factory ImageFactory --model=Image

Aparece una clase con la siguiente estructura:

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Comment>
 */
class CommentFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition()
    {
        return [
            //
        ];
    }
}
```

Ahora dentro del array return vamos a especificar cada uno de los campos de nuestras tablas y nos ayudaremos de faker para generar datos falsos. Por ejemplo, en ImageFactory:

```php
    protected $model = Image::class;
    public function definition()

    {
        $faker = Faker::create();
        $user_id = User::all( columns: 'id')->random()->id;

        return [
                'user_id'=>$user_id,
                'image_path'=>'ejemplo.png',
                'description'=>$faker->sentence( nbWords: 12),
                'created_at'=>$faker->dateTimeBetween( startDate: '-1 year', endDate: 'now'),
                'updated_at'=>$faker->dateTimeBetween( startDate: '-1 year', endDate: 'now'),
        ];
    }
}
```

Visita la documentación oficial de la librería Faker para ver todos los tipos de datos aleatorios que se pueden generar.

Fíjate que para el campo user_id se llama al modelo User para seleccionar de manera aleatoria el id de alguno de los usuarios que se ha creado. Para los campos created_at y updated_at tienes que rescatar también del modelo User la fecha de creación, puesto que la foto de un usuario no puede ser creada antes de que se haya creado el usuario.

Completa los factories para los demás modelos para poder continuar con los seeders.

Ahora que ya tenemos los factories creados vamos a pasar a los seeders, donde únicamente, en la función run(), llamaremos al factory y le especificaremos el número de registros que queremos crear:

ImagesSeeder.php

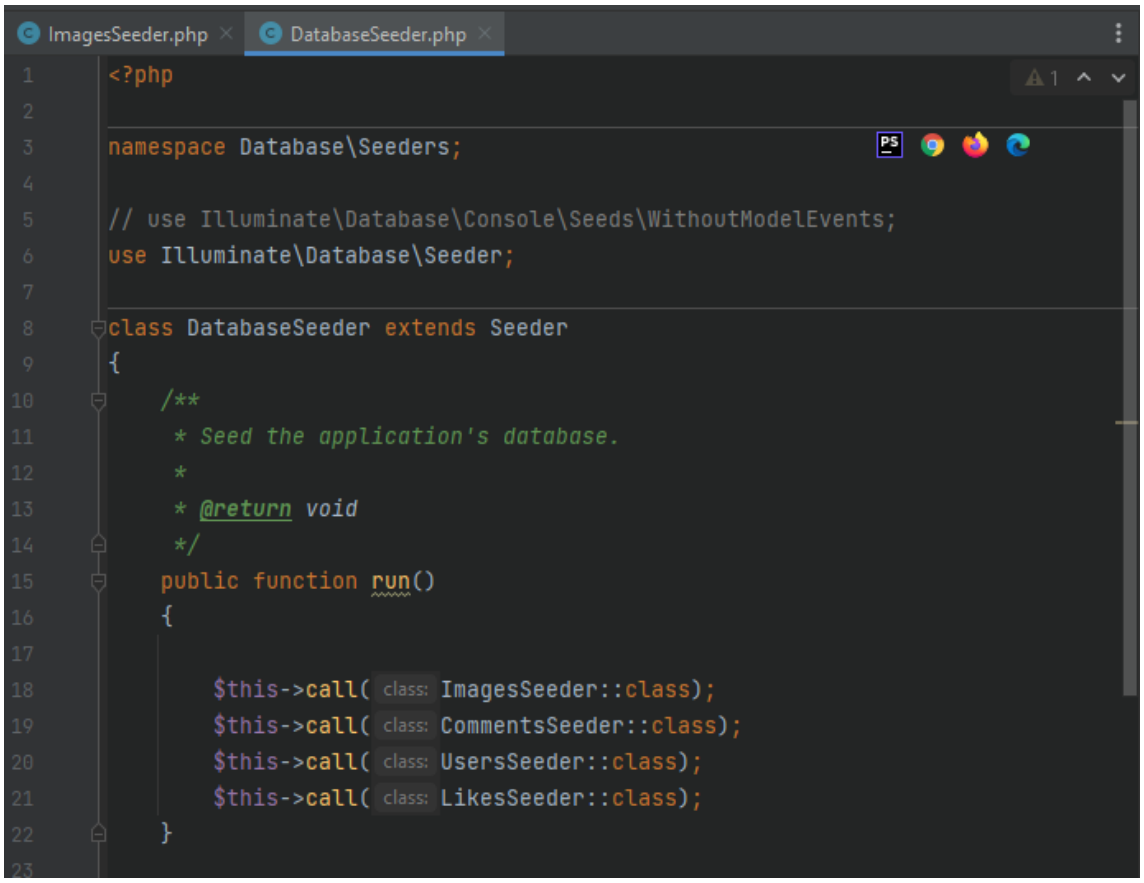
```php
namespace Database\Seeders;

use App\Models\Image;
use App\Models\User;
use Faker\Factory as Faker;
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class ImagesSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        Image::factory()->count( count: 1)->create();
    }

```

Una vez completados todos los seeders vamos al seeder general, DatabaseSeeder, y llamamos a todos los seeders a la vez. De esta manera, con un único comando en la consola de artisan podemos llamar a todos a la vez, aunque también se podría ir llamando cada seeder de manera individual.



Por último, desde la consola llamamos al seeder:

php artisan db:seed

Comprueba que las tablas se han rellenado de manera correcta.

PROBANDO EL ORM

Para finalizar, vamos a realizar una serie de consultas de prueba para comprobar cómo funciona Eloquent y ver que hemos realizado las relaciones de manera correcta.

Los modelos que hemos ido creando contienen una serie de métodos que nos van a permitir interactuar con la base de datos de una manera sencilla.

Vamos a hacer las pruebas directamente desde las rutas, en primer lugar, tenemos que importar las clases que vayamos a emplear. Empezaremos con el modelo Image:

```php
Route::get( uri: '/', function () {
    $images = Image::all();
    foreach($images as $image){
        echo $image->image_path;
    }
    die();
    return view( view: 'welcome');
});
```

El método all() nos permite obtener todos los registros de la tabla. En este caso se muestra el path de cada una de las imágenes.

Ahora vamos a añadir que se muestre el usuario que ha creado cada una de las imágenes. Para ello, debemos haber creado previamente las relaciones.

```php
Route::get( uri: '/', function () {
    $images = Image::all();
    foreach($images as $image){
        echo $image->image_path . "<br>";
        echo $image->description . "<br>";
        echo $image->user->name . ' '. $image->user->surname;
        echo "<hr>";
    }
```

User será el objeto con todas las propiedades, y de ella elegimos name y surname.

Ahora prueba a extraer los comentarios y el número de likes, pero ten en cuenta que debes emplear un bucle porque al contrario de los usuarios, una imagen puede tener más de un comentario y más de un like.

Por último, indica que usuario ha hecho cada uno de los comentarios.