CRUD ALQUILERES – MANUAL DE PROGRAMADOR

La mesa cuadrada

DONDE CADA PARTIDA CUENTA

Adrián Estévez Cerqueira

LA MESA CUADRADA

Contenido

REQUISITOS DEL SISTEMA

Para mantener la aplicación, hace falta el siguiente software:

- xampp: Proporciona el servidor en el que se aloja la base de datos y los servicios php
- Composer: Gestiona las dependencias php del proyecto
- Node.js: Gestiona las dependencias javascript del proyecto

CREACIÓN DEL PROYECTO

La aplicación se intentará conectar a una base de datos MySQL llamada ***proyecto_laravel*** ubicada en el mismo equipo que el proyecto, mediante el usuario ***root***, sin contraseña. Por lo tanto, para que la aplicación funcione, se debe crear una base de datos con ese nombre en el gestor de bases de datos del equipo.

Para crear el proyecto, se ha utilizado la versión 10 del framework Laravel, así que el comando ejecutado para su creación, lanzado desde el directorio C:\xampp\htdocs, fue ***composer create-proyect laravel/laravel alquileres "10.*"***. Todos los comandos que se mencionen posteriormente serán ejecutados desde el directorio C:\xampp\htdocs\alquileres, es decir, la carpeta en la que se ha creado el proyecto.

Una vez creado, se añadió de manera manual al archivo **composer.json** , en el objeto ***"require-dev"***, la dependencia necesaria para generar los las vistas, modelos y controladores de manera automática, ***"appzcoder/crud-generator" : "^3.2"***.

```
},
"require-dev": {
    "appzcoder/crud-generator": "^3.2",
    "fakerphp/faker": "^1.9.1",
    "ibex/crud-generator": "^2.1",
    "laravel/pint": "^1.0",
    "laravel/sail": "^1.18",
    "mockery/mockery": "^1.4.4",
    "nunomaduro/collision": "^7.0",
    "phpunit/phpunit": "^10.1",
    "spatie/laravel-ignition": "^2.0"
},
"autoload": {
```

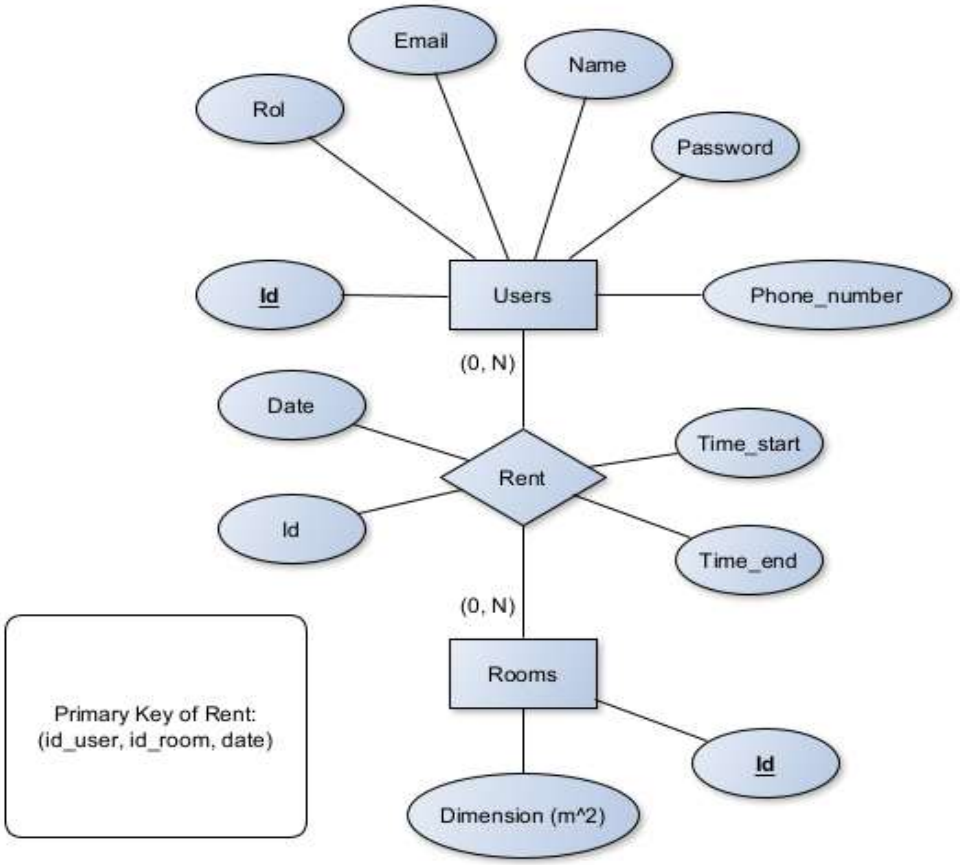
También se ha modificado el archivo **.env** para indicarle al proyecto el nombre de la base de datos a la que se debe conectar, tal y como se indica a continuación:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=proyecto_laravel
DB_USERNAME=root
DB_PASSWORD=
```

Si todo ha funcionado como debería, el comando ***php artisan serve*** lanzará el proyecto, y al introducir la dirección que nos proporciona en el navegador, debería ser visible la página de inicio de Laravel.

CREACIÓN DE LAS MIGRACIONES

Por defecto, Laravel nos crea ya una migración de usuarios. Esta migración ha sido modificada añadiendo dos campos: rol (pudiendo ser este admin o usuario) y número de teléfono. Además, se han creado dos migraciones, una siendo la tabla **salas**, donde se almacenará la información relativa a las mismas, siendo por el momento únicamente de interés su superficie; y la tabla **alquileres**, la cual almacenará la fecha del alquiler, la duración, la sala en concreto que ha sido alquilada y el usuario que la alquiló, todo esto siguiendo el siguiente esquema entidad-relación:



El id de los alquileres es necesario para que funcionen los modelos y vistas generados que se explicarán en un punto posterior, a pesar de no formar parte de la clave primaria, será una clave única.

Para crear las migraciones, se ejecutan los comandos en el siguiente orden:

php artisan make:migration create_rooms_table

php artisan make:migration create_rents_table

Una vez creadas, se modifican para que queden como se muestra a continuación:

```php
/**
 * Run the migrations.
 */
public function up(): void
{
    Schema::create('users', function (Blueprint $table) {
        $table->engine = "InnoDB";
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->integer("rol");
        $table->string("phone_number", 9)->nullable();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

```php
/**
 * Run the migrations.
 */
public function up(): void
{
    Schema::create('rooms', function (Blueprint $table) {
        $table->engine = "InnoDB";
        $table->id();
        $table->integer("dimension", false, true);
        $table->timestamps();
    });
}
```

```php
/**
 * Run the migrations.
 */
public function up(): void
{
    Schema::create('rentals', function (Blueprint $table) {
        $table->engine = "InnoDB";
        $table->integerIncrements("id")->unique("id");
        $table->date("date");
        $table->time("start_time");
        $table->time("end_time");
        $table->unsignedBigInteger("id_room");
        $table->foreign("id_room")->references("id_room")->on("rooms");
        $table->unsignedBigInteger("id_user");
        $table->foreign("id_user")->references("id")->on("users");
        $table->primary(["id_room", "id_user", "date"]);
        $table->timestamps();
    });
}
```

Una vez modificadas, se migran las tablas hacia la base de datos mediante el comando ***php artisan migrate***. En caso de que se modifiquen las migraciones o ocurra algún error que oblige a volver a hacer las migraciones, se ejecuta el comando ***php artisan migrate:refresh***. Se recomienda hacer una copia de seguridad de los datos antes de ejecutar el comando.

CREACIÓN DEL USUARIO ADMINISTRADOR

El usuario administrador de la aplicación es creado mediante un seeder. Para crear el seeder, se ejecutó el siguiente comando:

php artisan make:seeder UserSeeder

Una vez creado, se ponen los datos del usuario administrador, tal y como aparece reflejado a continuación:

```php
/**
 * Run the database seeds.
 */
public function run(): void
{
    DB::table("users")->insert([
        'name' => "admin",
        'email' => "lamesacuadradavigo@gmail.com",
        'phone_number' => null,
        'rol' => Rol::Admin,
        'password' => Hash::make("2626tonan55")
    ]);

}
```

El valor del rol proviene de un enumerado ubicado en **app\Enums\Rol.php**:

```php
namespace App\Enums;

enum Rol : int {
    case Admin = 0;
    case User = 1;
}
```

Una vez modificado, para poder ejecutarlo, se añade la clase a la función que se encuentra en **DatabaseSeeder.php** de la siguiente manera:

```php
/**
 * Seed the application's database.
 */
public function run(): void
{
    $this->call(UserSeeder::class);
}
```

Tras esto, se ejecuta mediante el comando ***php artisan db:seed***.

AUTENTICACIÓN

Para el sistema de autenticación se ha utilizado la dependencia de laravel/ui utilizando bootstrap para los estilos. Los comandos necesarios para ello han sido los siguientes:

composer require laravel/ui

php artisan ui bootstrap –auth

npm install

Una vez ejecutados, a partir de ahora, cada vez que se quiera lanzar el proyecto o hacer cambios en el crud, serán necesarias dos terminales ubicadas en el directorio raíz del proyecto, en una se ejecuta el comando ***npm run dev***, y en la otra se lanza el proyecto con ***php artisan serve***.

Al ejecutar la aplicación, ya debería aparecer una navbar con las opciones de log in y register.

CREACIÓN DE LAS VISTAS, MODELOS Y CONTROLADORES

Para que se creasen las vistas, modelos y controladores de manera automática para las migraciones **rooms** y **rentals**, se ejecutaron los siguientes comandos (con el comando ***npm run dev*** ejecutándose en otra terminal):

composer require ibex/crud-generator --dev

php artisan vendor:publish --tag=crud

php artisan make:crud rooms

php artisan make:crud rents

Al ejecutar estos dos últimos, fue solicitado un framework de estilos (tailwind, bootstrap, jetstream, ...), se seleccionó bootstrap.

ACCESO A LOS CRUDS

Una vez se ha creado todo, se crean las rutas para poder enlazar las vistas y los controladores en el archivo **web.php**, tal y como se muestra a continuación:

```
Route::resource("rooms", RoomController::class)->middleware("auth");
Route::resource('rentals', RentalController::class)->middleware("auth");
```

Una vez creadas, en **resources\views\layouts\app.blade.php**, se añaden al nav los enlaces para acceder a las vistas, como aparece reflejado en la siguiente imagen:

```
<!-- Left Side Of Navbar -->
@if (Auth::check())
    @if (Auth::user()->isAdmin())
        <ul class="navbar-nav me-auto">
            <li class="nav-item">
                <a class= "nav-link" href="{{ route('rentals.index') }}">{{ __('Alquileres') }}</a>
            </li>
            <li class="nav-item">
                <a class= "nav-link" href="{{ route('rooms.index') }}">{{ __('Salas') }}</a>
            </li>
        </ul>
    @else
        <ul class="navbar-nav me-auto">
            <li class="nav-item">
                <a class= "nav-link" href="{{ route('rentals.create') }}">{{ __('Alquilar') }}</a>
            </li>
        </ul>
    @endif
@endif
```

Como se puede observar, los usuarios administradores y los comunes no tienen los mismos elementos. Los usuarios administradores pueden acceder a todas las vistas, mientras que los usuarios comunes solo pueden acceder a la vista de creación de alquileres. Por defecto, en el controlador, tras añadir un nuevo elemento, el usuario es redireccionado a la vista de todos los alquileres, comportamiento que no queremos para el usuario común. El problema se resolvió en RentalController de la siguiente manera:

```php
/**
 * Store a newly created resource in storage.
 */
public function store(RentalRequest $request): RedirectResponse
{
    Rental::create($request->validated());

    //route for users
    $route = Auth::user()->rol === Rol::Admin->value ? "rentals.index" : "rentals.create";

    return Redirect::route($route)
        ->with('success', 'Rental created successfully.');
}
```

La función isAdmin() que se ve en el nav se encuentra en el modelo User:

```php
/**
 * Identifies if the user is an admin or not
 */
public function isAdmin() : bool {
    return $this->rol === Rol::Admin->value;
}
```

MODIFICACIONES DE LAS VISTAS

Rentals

A la hora de mostrar los alquileres, se muestra tanto el nombre como el email del usuario que realizó la reserva en lugar del id del mismo.

Al crear una nueva reserva, si la está realizando el usuario promedio, el input de id de usuario no aparece, ya que se obtiene automáticamente del usuario logeado. Si la realiza el usuario administrador, este la puede hacer para cualquier usuario existente en la base de datos.

Para que la reserva se realice correctamente, la hora de comienzo debe ser menor que la de final y la fecha debe ser como muy cercana a la actual, el día siguiente. Además, el usuario no puede tener alguna otra reserva para ese día, y debe realizarse para una sala que exista en la base de datos.

Estas restricciones se encuentran el archivo **app\Http\Requests\RentalRequest.php**, programadas de la siguiente manera:

```php
/**
 * Get the validation rules that apply to the request.
 *
 * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<
 */
public function rules(): array
{
    return [
        'date' => [
            'required',
            'after_or_equal:' . (new DateTime('tomorrow'))->format('Y-m-d')
        ],
        'start_time' => 'required|before:end_time',
        'end_time' => 'required|after:start_time',
        'id_room' => Rule::exists('rooms', 'id')->where(function ($query){
            return $query->whereNotNull("id");
        }),
        'id_user' => Rule::exists('users', 'id')->where(function ($query){
            return $query->whereNotNull("id");
        })
    ];
}
```

Rooms

La creación por defecto permitía introducir el id, se eliminó para que fuera autoincrementado.

Register

Al crearse un nuevo usuario, se le asigna el rol de usuario de manera automática.

También está presente la columna añadida a mayores del estándar, teléfono, como input.

```php
<input type="hidden" name="rol" value="{{App\Enums\Rol::User}}">
```

POSIBLES AMPLIACIONES

- Permitir al usuario promedio ver sus reservas.
- Generar un crud de usuarios para el administrador.
- Ampliar la migración Rooms con imágenes de las salas.
- Mejorar las restricciones de creación de nuevo alquiler.
- Permitir realizar un alquiler de una sala con un campo diferente al id de sala, o hacer que se asocie una sala libre automáticamente.
- Hacer que los nombres de usuario sean únicos, y permitir logearse mediante nombre de usuario además de correo.
- Unificar este crud con el crud de Cristina Míguez Piñeiro.