

2025

# Manual del Programador

CATALOGO DE JUEGOS DE MESA DE LA MESA CUADRADA  
CRISTINA MÍGUEZ PIÑEIRO

## INDICE

Creación de proyecto .....	2
Creación de las migraciones.....	2
Migración de las tablas.....	3
Autenticación .....	3
Creación del los CRUDs. ....	4
Acceso a los CRUDs .....	4
Perfiles de usuarios: Admin.....	5
Acceso a los datos de tipos y editoriales desde juegos.....	6
Creación de select para las categorías de Tipo y editorial .....	7
Ejecución del proyecto .....	7
Estilos y diseño .....	7
Traducción de los mensajes de validación al español.....	9
Subir el proyecto a GitHub .....	9
Posibles ampliaciones/mejoras para el futuro .....	9

## CREACIÓN DE PROYECTO

Crear la base de datos en el gestor PHPMyAdmin llamada proyecto\_laravel.

Desde la carpeta de htdocs de Xampp ejecutar el siguiente comando para crear el proyecto:

- **htdocs>composer create-project laravel\laravel catalogo\_juegos\_mesa "10.\*"**

Modificar en .env el nombre de la base de datos (izquierda) y en composer.json añadimos la línea "appzocoder\crud-generator": "^3.2" (derecha):

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

"require-dev": {
    "appzocoder/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"
},
```

## CREACIÓN DE LAS MIGRACIONES

Crear las migraciones teniendo en cuenta que la tabla con las claves foráneas se crea en ultimo lugar. Estas se sitúan en database\migrations\ . A continuación, modificar la función up de cada una añadiendo \$table->engine="InnoDB" para permitir borrar los registros en cascada y las columnas que componen cada tabla:

- **catalogo\_juegos\_mesa >php artisan make:migration create\_editorials\_table**

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create(table: 'editorials', callback: function (Blueprint $table): void {
            $table->engine='InnoDB';
            $table->id();
            $table->string(column: 'nombre');
        });
    }

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

- **catalogo\_juegos\_mesa** >php artisan make:migration create\_tipos\_table

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create(table: 'tipos', callback: function (Blueprint $table): void {
            $table->engine='InnoDB';
            $table->id();
            $table->string(column: 'nombre');
        });
    }

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

- **catalogo\_juegos\_mesa** >php artisan make:migration create\_juegos\_table

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create(table: 'juegos', callback: function (Blueprint $table): void {
            $table->engine='InnoDB';
            $table->id();
            $table->string(column: 'nombre');
            $table->double(column: 'precio');
            $table->integer(column: 'unidades');
            $table->integer(column: 'duracion');
            $table->integer(column: 'edad_min');
            $table->integer(column: 'jugadores_max');
            $table->unsignedBigInteger(column: 'editorial_id');
            $table->unsignedBigInteger(column: 'tipo_id');
            $table->foreign(columns: 'editorial_id')->references(columns: 'id')->on(table: 'editorials')->onDelete(action: 'cascade');
            $table->foreign(columns: 'tipo_id')->references(columns: 'id')->on(table: 'tipos')->onDelete(action: 'cascade');
        });
    }

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

## MIGRACIÓN DE LAS TABLAS

Migrar las tablas a la base de datos ejecutando el siguiente comando:

- **catalogo\_juegos\_mesa** >php artisan migrate

## AUTENTICACIÓN

Crear los elementos de Login y Register en la página de bienvenida utilizando Bootstrap para el diseño de las vistas. Para ellos ejecutaremos los siguientes comandos:

- **catalogo\_juegos\_mesa** >composer require laravel/ui
- **catalogo\_juegos\_mesa** >php artisan ui bootstrap --auth
- **catalogo\_juegos\_mesa** >npm install

## CREACIÓN DE LOS CRUDS.

Realizar los modelos con bootstrap, para ello se indicará durante la ejecución de los make. En este punto se crean automáticamente los modelos, controladores y vistas básicas del proyecto. Para llevar a cabo la creación de CRUDs se necesitan dos cmd:

- En un terminal ejecutar el siguiente comando y dejarlo ejecutando:
  - **catalogo\_juegos\_mesa** >npm run dev
- Desde otro terminal ejecutar:
  - **catalogo\_juegos\_mesa** >composer require ibex\crud-generator --dev
  - **catalogo\_juegos\_mesa** >php artisan vendor:publish --tag=crud
  - **catalogo\_juegos\_mesa** >php artisan make:crud editorials
  - **catalogo\_juegos\_mesa** >php artisan make:crud tipos
  - **catalogo\_juegos\_mesa** >php artisan make:crud libros

## ACCESO A LOS CRUDS

Modificar el archivo app\resources\views\layouts\app.blade.php para añadir a la zona izquierda del nav botones para acceder a las tablas. Para poder acceder a estos botones debes estar logueado. Si estas con un perfil de user solo puedes ver la tabla juegos mientras que si estas como admin puedes acceder a todas las tablas:

```
<!-- -----INTRODUCCION DE LOS BOTONES DEL NAVBAR----- -->
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <!-- Left Side Of Navbar -->
    @if (Auth::check())<!-- AUTORIZADO SOLO SI ESTA LOGING -->
        <ul class="navbar-nav me-auto">
            <li class="nav-item">
                <a class="nav-link" href="{{ route(name: 'juegos.index') }}">{{ __(key: 'Juegos') }}</a>
            </li>
            @if (Auth::user()->rol == "admin")<!-- AUTORIZADO SOLO SI ESTA LOGING Y ES ADMIN -->
                <li class="nav-item">
                    <a class="nav-link" href="{{ route(name: 'tipos.index') }}">{{ __(key: 'Tipos de Juego') }}</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="{{ route(name: 'editorials.index') }}">{{ __(key: 'Editoriales') }}</a>
                </li>
            @endif
        </ul>
    @endif
</div>
```

Para acceder a los cruds hay que establecer una ruta a cada uno de ellos, para ellos añadimos las rutas en el archivo routes\web. Además, para poder bloquear los menús anteriores hay que añadir el middleware de autenticación a las rutas:

```
use Illuminate\Support\Facades\Route;
use Illuminate\Support\Facades\Auth;

Route::get(uri: '/', action: function () { return view('welcome'); });

Auth::routes();
Route::resource(name: 'juegos', controller: App\Http\Controllers\JuegoController::class)->middleware(middleware: 'auth');
Route::resource(name: 'editorials', controller: App\Http\Controllers\EditorialController::class)->middleware(middleware: 'auth');
Route::resource(name: 'tipos', controller: App\Http\Controllers\TipoController::class)->middleware(middleware: 'auth');

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

## PERFILES DE USUARIOS: ADMIN

En primer lugar, para hacer una distinción entre administrador y usuario añadir en la app\Enum la clase Rol.php (Izquierda) y modificar el app\Models\User.php (Derecha) añadiendo la columna de rol:

```
app > Enums > Rol.php > ...
1  <?php
2
3  namespace App\Enums;
4  enum Rol: string{
5      case admin = "admin";
6      case user = "user";
7  }
8

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

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

También hay que modificar en app\Http\Controllers\Auth\RegisterController.php las funciones validator() y create() añadiendo la nueva columna:

```
protected function validator(array $data): Validator
{
    return Validator::make($data, [
        'name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => ['required', 'string', 'min:8', 'confirmed'],
        'rol' => ['required', 'string']
    ]);
}

protected function create(array $data): Model|User
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
        'rol' => $data['rol']
    ]);
}
```

Además, hay que modificar en database\Migrations la migración de users añadiendo la nueva columna:

```
public function up(): void
{
    Schema::create('users', function (Blueprint $table): void {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('rol');
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

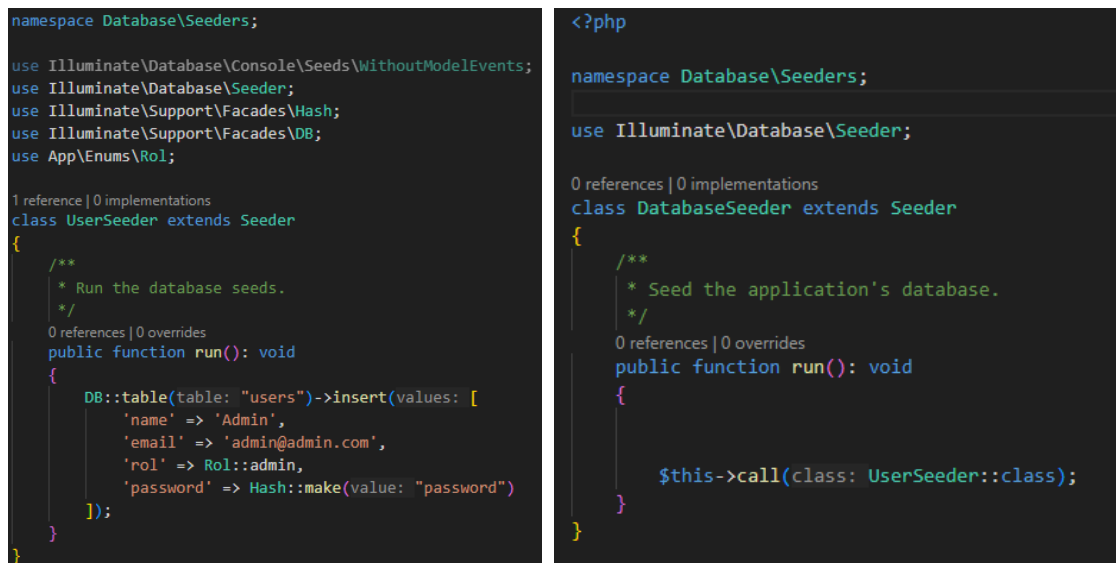
Finalmente, realizar la migración de los cambios con el siguiente comando:

- **catalogo\_juegos\_mesa >php artisan migrate:refresh**

Después, crear un seeder para introducir los datos del usuario administrador. El archivo creado se guarda en el directorio Database\Seeder\:

- **catalogo\_juegos\_mesa >php artisan make:seeder UserSeeder**

Dentro del método run() del archivo UserSeeder, definiremos los valores que para el administrador (Izquierda). Luego lo registramos dentro del archivo DatabaseSeeder.php llamándolo dentro del método run () (Derecha).



The image shows two code snippets side-by-side. The left snippet is the content of UserSeeder.php, and the right snippet is the content of DatabaseSeeder.php.

```
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\DB;
use App\Enums\Rol;

1 reference | 0 implementations
class UserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    0 references | 0 overrides
    public function run(): void
    {
        DB::table(table: "users")->insert(values: [
            'name' => 'Admin',
            'email' => 'admin@admin.com',
            'rol' => Rol::admin,
            'password' => Hash::make(value: "password")
        ]);
    }
}
```

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

0 references | 0 implementations
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    0 references | 0 overrides
    public function run(): void
    {
        $this->call(class: UserSeeder::class);
    }
}
```

Por último, crear Admin en la tabla users con los valores definidos:

- **catalogo\_juegos\_mesa > php artisan db:seed**

## ACCESO A LOS DATOS DE TIPOS Y EDITORIALES DESDE JUEGOS

Modificar el controlador JuegoController.php añadiendo el modelo tipo y el modelo editorial al principio del controlador y modificar las funciones create() y edit():



The image shows two code snippets. The first snippet shows the imports for Editorial and Tipo models. The second snippet shows the create method in the JuegoController.

```
use App\Models\Editorial;
use App\Models\Tipo;
```

```
public function create(): View
{
    $juego = new Juego();
    $editorial = Editorial::pluck(column: 'nombre',key: 'id');
    $tipo = Tipo::pluck(column: 'nombre',key: 'id');
    return view(view: 'juego.create', data: compact(var_name: 'juego',var_names: 'editorial','tipo'));
}
```

```

public function edit($id): View
{
    $juego = Juego::find(id: $id);
    $editorial = Editorial::pluck(column: 'nombre',key: 'id');
    $tipo = Tipo::pluck(column: 'nombre',key: 'id');
    return view(view: 'juego.edit', data: compact(var_name: 'juego',var_names: 'editorial','tipo'));
}

```

Además, para la validación de los datos de Juegos introducidos mediante formulario hay que modificar la función rules en el archivo app\Http\Request\JuegoRequest.php:

```

0 references | 0 overrides
public function rules(): array
{
    return [
        'nombre' => 'required|string|max:255',
        'precio' => 'required|numeric',
        'unidades' => 'required|integer',
        'duracion' => 'required|string|max:255',
        'edad_min' => 'required|integer',
        'jugadores_max' => 'required|integer',
        'editorial_id' => 'required|exists:editorials,id',
        'tipo_id' => 'required|exists:tipos,id',
    ];
}

```

## CREACIÓN DE SELECT PARA LAS CATEGORIAS DE TIPO Y EDITORIAL

Se modifica el Resource\view\Juego\form.blade.php para cambiar los inputs de Editorial y Tipo por selects referenciados con las dos tablas creadas anteriormente:

```

<div class="form-group mb-2 mb20">
    {{ Form::label('editorial') }}
    {{ Form::select('editorial_id', $editorial, $juego->editorial_id, ['class' => 'form-control' . ($errors->has('editorial_id') ? ' is-invalid' : '')]) }}
    {!! $errors->first('editorial_id', '<div class="invalid-feedback">:message</div>') !!}
</div>
<div class="form-group mb-2 mb20">
    {{ Form::label('tipo') }}
    {{ Form::select('tipo_id', $tipo, $juego->tipo_id, ['class' => 'form-control' . ($errors->has('tipo_id') ? ' is-invalid' : '')]) }}
    {!! $errors->first('tipo_id', '<div class="invalid-feedback">:message</div>') !!}
</div>

```

## EJECUCIÓN DEL PROYECTO

Para lanzar el proyecto se debe ejecutar los siguientes comandos:

- En un terminal ejecutar el siguiente comando y dejarlo ejecutando:
  - **catalogo\_juegos\_mesa >npm run dev**
- Desde otro terminal ejecutar:
  - **catalogo\_juegos\_mesa >php artisan serve**

Este último generará una dirección ip: puerto que tendremos que poner en nuestro navegador para ejecutar nuestro proyecto.

## ESTILOS Y DISEÑO

Para realizar los estilos se utilizó una combinación de CSS y Bootstrap. Se utilizó CSS para el posicionamiento y dimensionado de las imágenes en app\resources\views\welcome.blade.php y para aplicar la paleta de colores de la empresa realizando las clases logo-orange y logo-blue que se aplicaron a las diferentes view. Además, se realizó el redimensionamiento de las tablas



de todas las vistas y se cambiaron los botones de acción por iconos enlazados. También se aplicaron estos estilos a la barra de nav y se incluyó el logo de la empresa.

La vista `app\resources\views\layouts\app.blade.php` se reutiliza en todas las vistas y tiene enlazado un CSS donde se crean las clases que aplican la paleta de colores de la empresa y el enlace con FontAwesoma para poder obtener los iconos:

```
<!-- Scripts -->
@vite(entrypoints: ['resources/sass/app.scss', 'resources/js/app.js'])
@stack('styles')
<link rel="stylesheet" href="{{asset(path: 'css/app.css')}}">
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/font-awesome@6.4.2/css/all.min.css">
```

La vista `app\resources\views\welcome.blade.php` tiene enlazado un archivo `home.css` que tiene las clases que maneja el posicionamiento y dimensiones de las imágenes. Para poder acceder tanto a las imágenes como a los CSS se situaron en la carpeta `Public` del proyecto:

```
resources > views > welcome.blade.php > link
1 @extends(view: 'layouts.app')
2
3 @section(section: 'title', content: 'La mesa Cuadrada')
4
5 @section(section: 'content')
6
7 <link rel="stylesheet" href="{{ asset(path: 'css/home.css') }}">
8 <div class="image-container">
9   <div class="centered-content">
10     
11   </div>
12 </div>
13
14 @endsection
```

También se modificó el nombre del encabezado de Editorial y Tipo en la tabla Juegos:

```
<table class="table table-striped table-hover text-center">
  <thead class="thead-orange">
    <tr>
      <th>ID</th>
      <th>Nombre</th>
      <th>Precio</th>
      <th>Unidades</th>
      <th>Duración</th>
      <th>Edad Mín.</th>
      <th>Jugadores Máx.</th>
      <th>Editorial</th>
      <th>Tipo</th>
      <th>Acciones</th>
    </tr>
  </thead>
  <tbody>
    @foreach ($juegos as $juego)
      <tr>
        <td>{{ ++$i }}</td>
        <td>{{ $juego->nombre }}</td>
        <td>{{ $juego->precio }}</td>
        <td>{{ $juego->unidades }}</td>
        <td>{{ $juego->duracion }}</td>
        <td>{{ $juego->edad_min }}</td>
        <td>{{ $juego->jugadores_max }}</td>
        <td>{{ $juego->editorial->nombre }}</td>
        <td>{{ $juego->tipo->nombre }}</td>
      </tr>
    @endforeach
  </tbody>
</table>
```

## TRADUCCIÓN DE LOS MENSAJES DE VALIDACIÓN AL ESPAÑOL

Primero, ejecutar el siguiente comando para instalar el paquete que facilita las traducciones:

- `>composer require laravel-lang\common`

Luego, descargar la carpeta “es” desde el siguiente enlace:

- <https://github.com/Laraveles/spanish>

Crear la carpeta `resources\lang\es\` y colocar los siguientes archivos descargados:

- `auth.php`
- `pagination.php`
- `passwords.php`
- `validation.php`

En el archivo `config\app.php` configurar la línea locale para usar el español:

- `'locale' => 'es',`

Finalmente limpiamos cache:

- `>php artisan config:clear`
- `>php artisan cache:clear`
- `>php artisan optimize:clear`

## SUBIR EL PROYECTO A GITHUB

Se puede comprobar y descargar del siguiente enlace:

- [https://github.com/Cris-mp/LaMesaCuadrada\\_juegosMesa](https://github.com/Cris-mp/LaMesaCuadrada_juegosMesa)

## POSIBLES AMPLIACIONES/MEJORAS PARA EL FUTURO

En el futuro podría mejorarse implementando en las bases de datos la gestión de imágenes. Asociar a cada juego una imagen y mostrarla posteriormente mediante cards en la sección de users y en formato tabla en admin. También sería óptimo la creación de un carrito donde los usuarios puedan realizar las compras de juegos y no solo la consulta de los mismos.