# Práctica 2 Laravel

1. Requisitos del sistema

   En esta segunda práctica guiada, crearemos dos CRUDs **Libros** y **Categorías** y los relacionaremos. Además añadiremos un sistema de autenticación con UI y utilizaremos un paquete que no viene incluido por defecto en laravel, que es el **crud-generator**, para crear de forma rápida la estructura base de los cruds de nuestros proyectos.

   - Xampp → servidor local
   - Composer→gestor de dependencias PHP
   - Node.js→paquete de librerías en javascript

2. Creación del proyecto

   Antes de nada crearemos una base de datos en nuestro gestor PHPMyAdmin llamada ***libreria_laravel.***

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

   En la carpeta htdocs de xampp ejecutamos el siguiente comando para crear nuestro proyecto:

   htdocs>**composer create-project laravel/laravel libreria**

   Ejecutamos el proyecto para ver si se creó correctamente:

   libreria>**php artisan serve**

3. Creamos las migraciones

   Aquí **es muy importante el orden en el que creemos nuestras tablas**. Primero crearemos la tabla que no contiene el campo relacional. En nuestro caso es ***categorías***.

   libreria>**php artisan make:migration categorias**

   libreria>**php artisan make:migration libros**

```php
public function up()
{
    Schema::create('categorias', function (Blueprint $table) {

        $table->engine="InnoDB";
        $table->id();
        //$table->bigIncrements('id');
        $table->string('nombre');
        $table->timestamps();
    });
}

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

```php
public function up()
{
    Schema::create('libros', function (Blueprint $table) {

        $table->engine="InnoDB";
        $table->id();
        //$table->bigIncrements('id');
        $table->bigInteger('categoria_id')->unsigned();
        $table->string('nombre');
        $table->timestamps();
        $table->foreign('categoria_id')->references('id')->on('categorias')->onDelete('cascade');
    });
}

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

La instrucción $table->engine=”InnoDB” me va a permitir borrar los registros en cascada, es decir, cuando borre una categoría se eliminarán todos los libros con esa categoría.

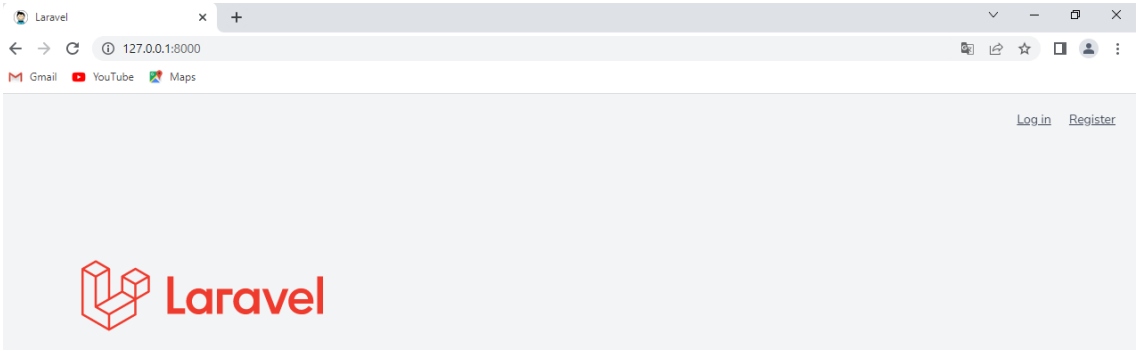
## 4. Autenticación

En este ejercicio utilizaremos **ui** con **bootstrap** (también podeis utilizar breeze con tailwind y flowbite, jetstream,…).Los comandos que tendremos que ejecutar son los siguientes y por este orden:

libreria>**composer require laravel/ui**
librería>**php artisan ui bootstrap --auth**
librería>**npm install**
librería>**npm run dev**

**Este último comando lo debemos ejecutar en otro terminal y dejarlo ejecutándose antes de que ejecutemos nuestro proyecto.**

Ahora, si ejecutamos nuestro proyecto desde la consola, nos debería aparecer en la parte superior derecha de la página de bienvenida dos elementos nuevos (Login y Register).




Accedemos a Register y nos registramos con un usuario nuevo (Laravel guardará estos datos en la tabla ***users*** que crea automáticamente al crear el proyecto).

5. Creación de los CRUDs. Uso de crud-generator

Ejecutamos los siguientes comandos desde la consola, situándonos como siempre en la carpeta de nuestro proyecto:

libreria>**composer require ibex/crud-generator --dev**
libreria>**php artisan vendor:publish --tag=crud**
libreria>**php artisan make:crud categorias** (debe coincidir con el nombre de la tabla)
libreria>**php artisan make:crud libros**

Y ya nos ha creado automáticamente los modelos, controladores y vistas básicas para nuestro proyecto.

6. Acceso a los cruds

Ahora debemos acceder a los cruds que hemos generado anteriormente. Para ello debemos establecer una ruta a cada uno de ellos. Nos dirigimos al archivo de rutas web.php y escribimos las siguientes rutas de tipo **resource**.

```php
Route::get('/', function () {
    return view('welcome');
});

Auth::routes();

Route::resource('libros',App\Http\Controllers\LibroController::class)->middleware('auth');
Route::resource('categorias',App\Http\Controllers\CategoriaController::class)->middleware('auth');

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

7. Creación de los enlaces a Libros y Categorias

   Al ejecutar el crud-generator se nos han creado los achivos básicos para tener un crud funcional desde el principio, incluidas las vistas. Nosotros simplemente podemos modificar ahora el código creado. Vamos añadir un menú en nuestro proyecto para acceder a los Libros y a las Categorias.
   Para ello vamos a app\resources\views\layouts\app.blade.php y creamos dos enlaces en la parte izquierda del menú de navegación:

```php
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <!-- Left Side Of Navbar -->
    @if(Auth::check())
    <ul class="navbar-nav me-auto">
        <li class="nav-item">
            <a class="nav-link" href="{{ route('libros.index') }}">{{ __('Libros') }}</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route('categorias.index') }}">{{ __('Categorias') }}</a>
        </li>
    </ul>
    @endif
    <!-- Right Side Of Navbar -->
```

8. Acceso a los datos de Categorias desde Libros

   Para poder acceder a los datos de las categorías desde el crud Libros, debo poder llamar a la clase Categoria para poder acceder a sus datos. Para ello vamos a modificar un par de métodos de **LibroController.php**. Si nos fijamos, crud-generator ya ha creado los controladores y los métodos por defecto propios de un crud (index,create,store,edit,destroy,..).Lo primero que debemos hacer es añadir el modelo **Categoria** al principio del controlador:

   use App\Models\Categoria

```php
public function create()
{
    $libro = new Libro();
    $categorias=Categoria::pluck('nombre','id');
    return view('libro.create', compact('libro','categorias'));
}
```

```php
public function edit($id)
{
    $libro = Libro::find($id);
    $categorias=Categoria::pluck('nombre','id');

    return view('libro.edit', compact('libro','categorias'));
}
```

El método pluck es como el get o el all, lo único que hace es devolver los campos en el orden que nosotros le indiquemos como parámetros.

9. Creación de un select para las categorías

libreria > resources > views > libro > form.blade.php

```blade
<div class="box box-info padding-1">
    <div class="box-body">
        <div class="form-group">
            {{ Form::label('categoria_id') }}
            {{ Form::select('categoria_id',$categorias, $libro->categoria_id, ['class' => 'form-co
            {!! $errors->first('categoria_id', '<div class="invalid-feedback">:message</div>') !!}
        </div>
        <div class="form-group">
            {{ Form::label('nombre') }}
            {{ Form::text('nombre', $libro->nombre, ['class' => 'form-control' . ($errors->has('no
            {!! $errors->first('nombre', '<div class="invalid-feedback">:message</div>') !!}
        </div>

    </div>
    <div class="box-footer mt20">
        <button type="submit" class="btn btn-primary">Submit</button>
    </div>
</div>
```

10. Modificación del nombre del encabezado Categoria de mi tabla

Para ello nos dirigimos al archivo app\resources\views\libro\index.blade.php

```blade
<thead class="thead">
    <tr>
        <th>No</th>

        <th>Categoria</th>
        <th>Nombre</th>

        <th></th>
    </tr>
</thead>
<tbody>
    @foreach ($libros as $libro)
        <tr>
            <td>{{ ++$i }}</td>

            <td>{{ $libro->categoria->nombre }}</td>
            <td>{{ $libro->nombre }}</td>
```

Puedo acceder a $libro->categoría->nombre directamente porque en el modelo de Libro (Libro.php) tengo implementado un método categoría ():

public function categoria(){

return $this>hasOne(`App\Models\Categoria`,`id`,`categoria_id`);

}

## 11. Ajustes finales

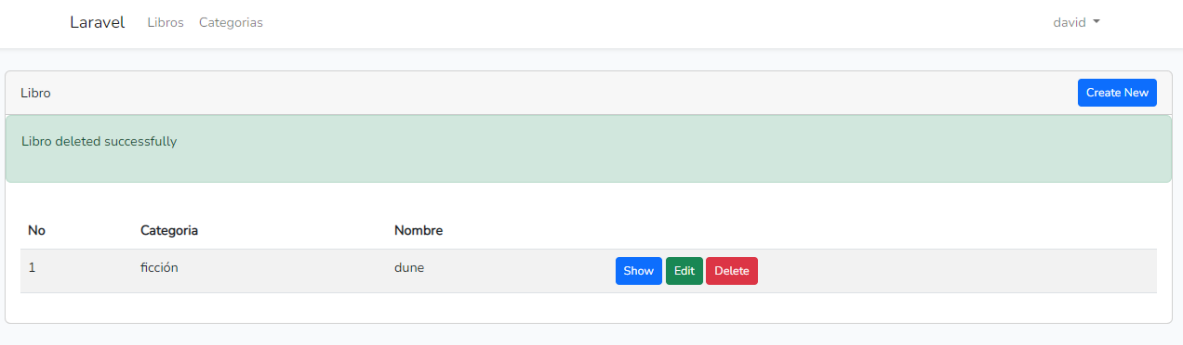
Por último vamos a bloquear los menús de Libros y Categorias para que no aparezcan cuando no estoy logueado. Lo que vamos a hacer es añadir el middleware de autenticación a las rutas del archivo web.php.

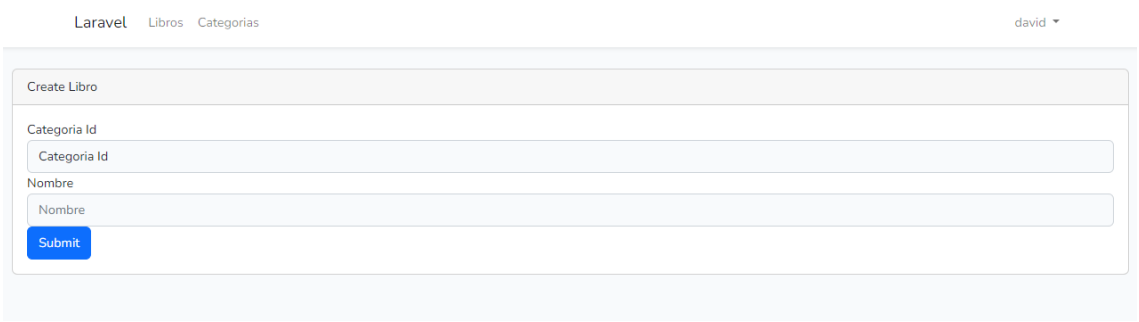
```php
Route::get('/', function () {
    return view('welcome');
});

Auth::routes();

Route::resource('libros',App\Http\Controllers\LibroController::class)->middleware('auth');
Route::resource('categorias',App\Http\Controllers\CategoriaController::class)->middleware('auth');

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

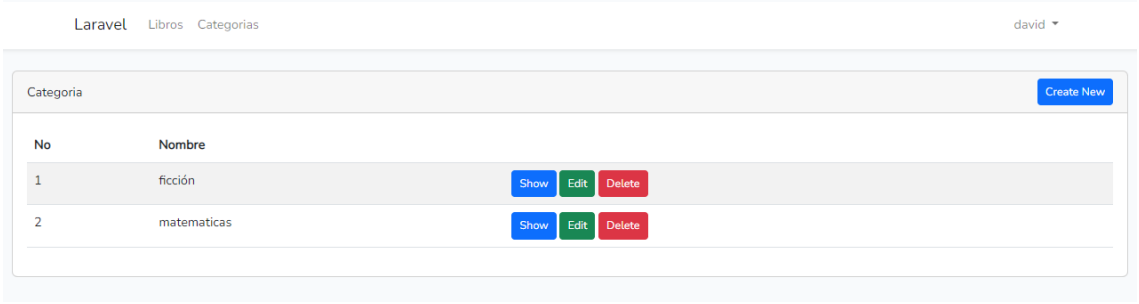
Por último debo indicar con un @if que mis menús sólo se muestren si alguien autorizado está logueado en mi aplicación.

```php
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <!-- Left Side Of Navbar -->
    @if(Auth::check())
    <ul class="navbar-nav me-auto">
        <li class="nav-item">
            <a class="nav-link" href="{{ route('libros.index') }}">{{ __('Libros') }}</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route('categorias.index') }}">{{ __('Categorias') }}</a>
        </li>
    </ul>
    @endif
    <!-- Right Side Of Navbar -->
```

Al final nuestro proyecto debe tener un aspecto como este:




Laravel
Libros
Categorias
david
Create Libro
Categoria Id
Categoria Id
Nombre
Nombre
Submit


Laravel
Libros
Categorias
david
Categoria
Create New
No
Nombre
1
ficción
Show
Edit
Delete
2
matematicas
Show
Edit
Delete