

-php artisan tinker [para poder ejecutar y ver cosas que hayamos creado]
-composer dump-autoload

Crear nueva tabla en laravel en un proyecto existente

-[shell]

-php artisan make:model -cmrfs Empleado

[create_empleados_table.php](#) [en el Schema create empleados]:

```
*$table->decimal('numero', 4, 0)->unique  
*$table->string,decimal...('nombre_tabla')->unique.... (por defecto, not null, nullable\(\)=nulo)  
*$table->foreignId('departamento_id')->constrained();
```

-php artisan migrate

+Nos metemos en la base de datos y ingresamos los empleados que queramos.
[sudo -u postgres psql]

+Creamos una ruta.

[Web.php](#)

```
*Route::resource('empleados', EmpleadoController::class);
```

[EmpleadoController.php](#)

```
*public function index()  
{  
return view('empleados.index', ['empleados' => Empleado::all()  
]);  
}
```

El index.blade lo debemos crear en resources/views/nombre_tabla (por lo tanto si creamos Artículo, tendremos que crear antes obviamente una carpeta llamada articulos)

[Index.blade.php](#)

Aqui nos muestra en la pagina inicial todo lo que vayamos programando y haciendo

Esta parte es para saber a partir de un departamento, cuantos empleados hay o viceversa

[Departamento.php](#)

Nos creamos una funcion empleados. —> return \$this->hasMany(Empleado::class)

[Empleado.php](#)

Nos creamos una funcion departamento —> return \$this->belongsTo(Departamento::class)

Entrar al index.lade.php de empleados y cambiar para que te de en vez del departamento id empleado->departamento->denominacion

show.blade personalizar las tablas (en flowbyte puedo coger las tablas)

breeze and blade

Empezar proyecto ya empezado

- composer install
- npm install
- cambiar el .env [pgsql y puerto 5432]
- php artisan migrate
- php artisan key:generate

composer run dev *arrancarlo*

*Crear base de datos

sudo -u postgres createdb -O -P nombreDB nombreUsuario

*METERSE EN BASE DE DATOS

psql -h localhost -U nombreUser -d nombreDB

como crear un carrito

nueva carpeta en /app la podemos llamar generico

dentro creamos carrito.php

function meter(\$id) y sacar, creamos tambien Linea.php una clase con articulo y cantidad con los getters y demas.

crear proyecto desde 0.

ÍNDICE

1. Crear y conectar base de datos.
2. Crear migración.
3. Crear controlador y modelo.
4. Crear vista index.
5. CRUD crear.
6. CRUD leer.
7. CRUD update.
8. CRUD delete.
9. Cosas adicionales

-composer create-project laravel/laravel nombre_proyecto

1. Crear y conectar base de datos.

-sudo -i -u postgres *También sirve este método*
psql

-psql -h localhost -U nombreUser -d nombreDB *asi nos metemos en postgresql
[postgres es superusuario por si nos queremos meter con el directamente]

CREATE DATABASE nombre_base_datos;

CREATE USER nombre_usuario WITH PASSWORD 'contraseña_segura';

GRANT ALL PRIVILEGES ON DATABASE nombre_base_datos TO nombre_usuario;

*Si no sirve el GRANT ALL PRIVILEGE hacer: ALTER ROLE nombre_usuario WITH SUPERUSER; *

-composer install

-npm install

-cambiar el .env [pgsql y puerto 5432]

-php artisan migrate *php artisan migrate:rollback deshace las migraciones*

-php artisan key:generate

-composer run dev *arrancarlo*

-Ahora desde la documentación buscamos Breeze, y lo instalamos.

2. Crear migración

-php artisan make:model -cmrfs Empleado [primera en mayúscula y palabra en singular]

*Nos metemos en el archivo create_empleado_table y añadimos los datos a la tabla.

```
*$table->decimal('numero', 4, 0)->unique
```

```
*$table->string,decimal,dateTime,intenger...('nombre_tabla')->unique.... (por defecto, not null, nullable())=nulo)
```

```
*$table->foreignId('departamento_id')->constrained();
```

-php artisan migrate [para añadirlo la tabla con los nuevos elementos]

3. Crear controlador y modelo

*el comando php artisan make:model -cmrfs ya nos lo creo.

[App/Models/Empleado.php y HTTP/Controllers/EmpleadoController.php]*

para mostrar elementos de 5 en 5, o el numero que sea nos metemos en AppServiceProvider.php y escribimos:

```
use Illuminate/Pagination/Paginator
```

y en el function boot() escribimos:

```
Paginator::useBootstrapFive()
```

y en nuestro controller debería verse tal que así:

```
$departamentos = Departamento::latest()->paginate(5);
```

```
return view('index', ['departamentos' => $departamentos]);
```

4. Crear vista index

creamos en /Views carpeta con nombre de la tabla (empleados) y dentro de esa carpeta el archivo index.blade.php

Creamos la ruta en web.php y importamos la ruta

```
(use App/Http/Controllers/EmpleadoController) -> para importar la ruta.
```

```
Route::resource('empleados', EmpleadoController::class); -> para crear la ruta.
```

Vamos al controlador y en la function index le ponemos lo siguiente:

```
return view('empleados.index',
```

```
['empleados' => Empleado::all());
```

-> nos conecta con el index.blade.php, lo que hayamos puesto ahí se mostrará por pantalla al llegar a dicha ruta.

-En navigation.blade.php metemos algo similar a esto:

```
<x-nav-link :href="route('libros.index')"
:active="request()->routeIs('dashboard') ">
    {{ __('Libros') }}
</x-nav-link>
```

5. CRUD crear

Si queremos insertar valores pero no hace falta crear CRUD podemos hacerlo en la base de datos directamente

-psql -h localhost -U libreria -d libreria

INSERT INTO TABLE nombre_tabla VALUES();

En EmpleadoController nos vamos al function create() y ponemos

```
return view('articulos.create')
```

- IMPORTANTE DECIR QUE LA SINTAXIS ES
view('nombre_carpeta.nombre_archivo(omitiendo el .blade.php)')

donde mismo creamos el index.blade.php, creamos el create.blade.php

Metemos formulario, de este modo:

```
<form action="{{route('empleados.store')}}" method="POST">
```

-php artisan route:list -> nos dice las rutas que tenemos

Ejemplo: (supongamos donde pone task, pone empleados: requiere método POST además dicha ruta conecta con el método store de nuestro controlador)

```
POST      tasks ..... tasks.store > TaskController@store
```

*dd() es equivalente a hacer var_dump() y die(), podemos hacer en el store
dd(\$request->all()) esto nos imprime toda la información que llega del formulario create*

Queremos que después de crear formulario nos vuelva donde antes, entonces en el function store escribimos:

```
Empleado::create($request->all());
return redirect()->route('empleados.index');
```

Pero para que funcione tenemos que arreglar el error de asignación masiva, se hace así:

Nos metemos en empleado.php y dentro de la clase escribimos:

```
protected $fillable = ['nombre','edad'...]
```

en definitiva valores que podrán ser asignados en el formulario

Vamos a validar los datos del formulario:

En el controller donde tenemos la function store, arriba de lo previamente creado hacemos:

```
$request->validate([
    'titulo' => 'required',
    'descripcion' => 'required|string|max:255'
]);
```

Así no se creará si no se cumple las condiciones (necesario que en create se gestione los errores, eso ya está puesto por el create de Ricardo)

6. CRUD leer

En el controlador de empleado nos tenemos en el function index y escribimos lo siguiente:

```
$empleados = Empleado::all(); *Nos muestra todos en general*
```

```
$empleados = Empleado::latest()->get(); *Nos trae todos del último hacia el primero*
```

```
return view('empleados.index', ['empleados' => $empleados]); *nos la pasa a la vista index*
```

y en el index.blade.php metemos un @foreach (\$empleados as \$empleado) y finalizamos con un @endforeach y entre medio metemos la tabla con las variables que queremos que se muestren.

7. CRUD update

En el controlador en la function edit escribimos dentro:

```
return view('empleados.edit', ['empleado' => $empleado]);
```

y en la carpeta /resources/views/empleados creamos archivo edit.blade.php.

Podemos copiar el contenido del create y adaptarlo

En el formulario escribimos lo siguiente:

```
<form action="{{route('empleados.update', $empleado)}}" method="POST">
```

@method('PUT') *Esto es para falsear el método, va justo después de abrir el form*

Ahora en el controller en el function update:

```
$empleado->update($request->all());  
return redirect()->route('empleados.index');
```

En caso de queramos mostrar algún mensaje se podría poner así

```
return redirect()->route('empleados.index')->with('success', 'empleado exitosamente');
```

Y validamos (podemos coger el mismo que en crear):

```
$request->validate([  
    'titulo' => 'required',  
    'descripcion' => 'required|string|max:255'  
]);
```

-Y creamos en el index.blade.php el botón de editar:

```
<a href="{{route('libros.edit', $libro)}}" class="font-medium  
text-blue-600 dark:text-blue-500 hover:underline">Editar</a>
```

8. CRUD delete

En el index.blade.php, ya deberíamos tener el botón de eliminar.

```
<form action="{{route('tasks.destroy', $task)}}" method="POST" class="d-inline">  
    @csrf  
    @method('DELETE')  
    <button type="submit" class="btn btn-danger">Eliminar</button>  
</form>
```

Debería verse tal que así (donde pone tasks, supón que pone empleados)

Ahora en el controlador, en el function destroy:

```
$empleado->delete();  
return redirect()->route('empleados.index');
```

Así se borraría y nos redirigirá de nuevo al index

9. Cosas adicionales

-Crear tabla pivote

php artisan make:migration create_articulo_departamento_table

belongsToMany() esto va en el controlador de cada tabla, solamente cuando hay tabla pivote

-return \$this->hasMany(Factura::class, 'user_id'); [esto va en el modelo (nombre_tabla.php)]

*El que posee la clave foranea, es belongsTo(), y el otro hasMany()

-Fechas

use Carbon\Carbon // importar

\$date = Carbon::now(); // fecha actual

\$date->toDateString(); // mostrar sólo fecha

\$date->toTimeString(); // mostrar sólo hora

\$date = Carbon::createFromDate(2002,28,05)->age; //Obtener edad a partir de fecha

\$fechaFutura = Carbon::create(2024, 12, 31); // Fecha futura (año, mes, día)

\$diasFaltantes = Carbon::now()->diffInDays(\$fechaFutura); //num de días que faltan

TODOS LOS COMANDOS JUNTOS

```
`echo Introduce el nombre del proyecto:`
```

```
`read proyecto`
```

```
`echo Introduce el nombre de la BBDD:`
```

```
`read bbdd`
```

```
`composer create-project laravel/laravel $proyecto &&`
```

```
`cd $proyecto &&`
```

```
`composer install &&`
```

```
`npm install &&`
```

```
`composer require laravel/breeze --dev &&`
```

```
`php artisan breeze:install &&`
```

```
`npm install -D tailwindcss postcss autoprefixer flowbite &&`
```

```
`npx tailwindcss init -p &&`
```

```
`sudo -u postgres createuser -P $bbdd &&`
```

```
`sudo -u postgres createdb -O $bbdd $bbdd &&`
```

```
`code .`
```


Para cambiar nombre a tabla cambiar lo siguiente:

```
new Go Run Terminal Help
2024_12_06_114653_create_autores_table.php X
database > migrations > 2024_12_06_114653_create_autores_table.php > ...
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14         Schema::create('autores', function (Blueprint $table) {
15             $table->id();
16             $table->timestamps();
17             $table->integer('codigo')->unique();
18             $table->string('nombre');
19         });
20     }
21
22     /**
23      * Reverse the migrations.
24      */
25     public function down(): void
26     {
27         Schema::dropIfExists('autores');
28     }
29 };
30
```

```
new Go Run Terminal Help
2024_12_06_114653_create_autores_table.php Autor.php X
app > Models > Autor.php > Autor
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Autor extends Model
9 {
10     /** @use HasFactory<Database\Factories\AutorFactory> */
11     use HasFactory;
12
13     protected $fillable = ['codigo', 'nombre'];
14
15     protected $table = 'autores';
16
17     public function libros()
18     {
19         return $this->hasMany(Libro::class);
20     }
21 }
22
```

```
new Go Run Terminal Help
2024_12_06_114653_create_autores_table.php Autor.php 2024_12_06_115348_create_libros_table.php X
database > migrations > 2024_12_06_115348_create_libros_table.php > ...
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14         Schema::create('libros', function (Blueprint $table) {
15             $table->id();
16             $table->timestamps();
17             $table->integer('codigo')->unique();
18             $table->foreignId('autor_id')->constrained('autores');
19             $table->string('titulo');
20         });
21     }
22
23     /**
24      * Reverse the migrations.
25      */
26     public function down(): void
27     {
28         Schema::dropIfExists('libros');
29     }
30 };
31
```