

# Manual de programador

---

VIRTUAL MEDITATE

Alejandro Miranda Comesaña

SGE | 2º DAM




## ÍNDICE

Requisitos .....	2
Creación del proyecto.....	2
Creación base de datos .....	2
Creación de la carpeta de proyecto .....	2
Configuración .....	2
Creación de migraciones .....	3
Creación autenticación.....	3
Creación CRUDs.....	4
Configuración rutas .....	4
Creación del menú .....	4
Configuración de controladores .....	5
Creación de selects.....	5
Modificación de cabeceras y datos .....	6
Restringir acceso .....	7
Creación seeders .....	8
Configuración locales .....	8
Estilos.....	9
Posibles implementaciones .....	10

## REQUISITOS

Antes de empezar a crear la página necesitamos las siguientes herramientas:

- Composer : gestor de dependencias PHP
- Artisan : CLI que nos va a permitir gestionar nuestros proyectos desde la consola.
- Xampp : nuestro servidor local (o cualquier otro servidor local)

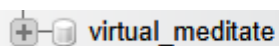
[Node.js]  Nos permitirá crear los elementos de autenticación del proyecto. Node.js es un entorno de ejecución de JavaScript , permitiéndolo ejecutar fuera de un navegador web.

## CREACIÓN DEL PROYECTO

### CREACIÓN BASE DE DATOS

Creamos la base de datos desde PHPMyAdmin:

En nuestro caso se llama virtual\_meditate igual que el nombre de la empresa



### CREACIÓN DE LA CARPETA DE PROYECTO

En la ruta que prefiramos, aunque preferiblemente desde htcdocs lanzaremos el siguiente comando de consola para crear el proyecto:

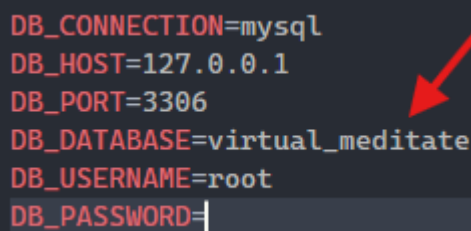
```
C:\Users\Alejandro\Desktop>composer create-project laravel/laravel virtualmeditate "10.*"
```

Elegimos la versión 10, pues es la que nos permitirá hacer uso del crud-generator.

## CONFIGURACIÓN

Una vez creado el proyecto nos dirigimos a nuestro editor de código de confianza y retocamos los siguientes archivos:

En el archivo .env nos aseguramos de configurar la base de datos para que tenga el mismo nombre que la base de datos creada en el paso 1.

A screenshot of the .env file configuration. The text is as follows:  
DB\_CONNECTION=mysql  
DB\_HOST=127.0.0.1  
DB\_PORT=3306  
DB\_DATABASE=virtual\_meditate  
DB\_USERNAME=root  
DB\_PASSWORD=|  
A red arrow points to the DB\_DATABASE=virtual\_meditate line.

A continuación, nos dirigimos al archivo composer.json y añadimos la dependencia para poder usar el crud generator:

```

"require-dev": {
  "appzcoder/crud-generator": "^3.2" (v3.3.0), v3.3.0
  "fakerphp/faker": "^1.9.1" (v1.24.1), v1.24.1
  "ibex/crud-generator": "^2.1" (v2.1.2), v2.1.2
  "laravel/pint": "^1.0" (v1.20.0), v1.20.0
  "laravel/sail": "^1.18" (v1.41.0), v1.41.0
  "mockery/mockery": "^1.4.4" (1.6.12), v1.6.12
  "nunomaduro/collision": "^7.0" (v7.11.0), v7.11.0
  "phpunit/phpunit": "^10.1" (10.5.45), v10.5.45
  "spatie/laravel-ignition": "^2.0" (2.9.0) v2.9.0
},

```

## CREACIÓN DE MIGRACIONES

Ahora es tiempo de crear nuestras tablas, el orden en este caso es importante ya que algunas poseen claves foráneas de otras tablas. Para ello crearemos primero la tabla categorías, luego entornos y por último objetos.

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan make:migration create_categorias_table_
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan make:migration create_entornos_table
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan make:migration create_objetos_table
```

De esta forma nos ahorramos tener que crear las funciones up() y down() ya que ya vendrán creadas con la estructura correcta. Si quisiéramos podríamos poner la instrucción `$table->engine="InnoDB"` para habilitar el borrado en cascada, cosa que no queremos en nuestro caso y por eso no ponemos.

## CREACIÓN AUTENTIFICACIÓN

Para proceder a configurar el proceso de autenticación necesitamos ejecutar los siguientes comandos.

```
C:\Users\Alejandro\Desktop\virtualmeditate>composer require laravel/u
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan ui bootstrap --auth
```

Desde otra terminal:

```
C:\Users\Alejandro\Desktop\virtualmeditate>npm install_
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>npm run dev_
```

Ahora el resultado es el siguiente:

```
C:\Users\Alejandro\Desktop\virtualmeditate>npm run dev

> dev
> vite

VITE v5.4.14 ready in 609 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help

LARAVEL v10.48.28 plugin v1.2.0
```

Y ya podríamos registrarnos e iniciar sesión en la página. Los datos serán guardados en users

## CREACIÓN CRUDS

Vamos a proceder a crear los CRUDs para las tablas que ya tenemos, para ello usaremos los siguientes comandos:

```
C:\Users\Alejandro\Desktop\virtualmeditate>composer require ibex/crud-generator --dev_
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan make:crud categorias_
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan vendor:publish --tag=crud_
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan make:crud entornos_
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan make:crud objetos
```

Nos han preguntado con qué queríamos crear los CRUDs y hemos elegido la opción de bootstrap para todos ellos.

## CONFIGURACIÓN RUTAS

Ahora que ya están creados los CRUDs, vamos a configurarles una ruta, para ello nos dirigimos al archivo web.php dentro de la carpeta routes y creamos las siguientes rutas:

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

Auth::routes();
Route::resource(name: 'categorias', controller: App\Http\Controllers\CategoriaController::class)→middleware(middleware: 'auth');
Route::resource(name: 'entornos', controller: App\Http\Controllers\EntornoController::class)→middleware(middleware: 'auth');
Route::resource(name: 'objetos', controller: App\Http\Controllers\ObjetoController::class)→middleware(middleware: 'auth');

You, 2 weeks ago • Configuración del proyecto de laravel
Route::get(uri: '/home', action: [App\Http\Controllers\HomeController::class, 'index'])→name(name: 'home');
```

Como se puede apreciar hay una ruta por cada tabla.

## CREACIÓN DEL MENÚ

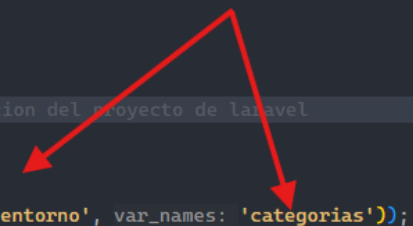
Ahora crearemos un menú para poder acceder a nuestras tablas, para ello seguimos la siguiente ruta dentro de nuestro proyecto app/resources/views/layouts/app.blade.php y ahí añadimos lo siguiente:

```
<ul class="navbar-nav me-auto">
  <li class="nav-item">
    <a class="nav-link text-light" href="{{ route(name: 'categorias.index') }}">{{ __(key: 'Categorias') }}</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-light" href="{{ route(name: 'entornos.index') }}">{{ __(key: 'Entornos') }}</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-light" href="{{ route(name: 'objetos.index') }}">{{ __(key: 'Objetos') }}</a>
  </li>
</ul>
```

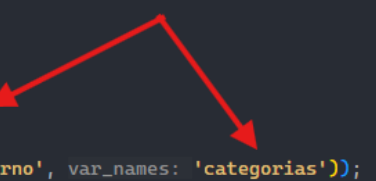
## CONFIGURACIÓN DE CONTROLADORES

Ahora, para poder acceder a datos de otra tabla como por ejemplo acceder al nombre de una categoría en base a su id necesitamos hacer lo siguiente, primero en EntornoController.php, en los métodos create() y edit(), pondremos lo siguiente:

```
/**
 * Show the form for creating a new resource.
 */
0 references | 0 overrides
public function create(): View
{
    $entorno = new Entorno();
    $categorias = Categoria::pluck(column: 'nombre', key: 'id');
    return view(view: 'entorno.create', data: compact(var_name: 'entorno', var_names: 'categorias'));
}
```

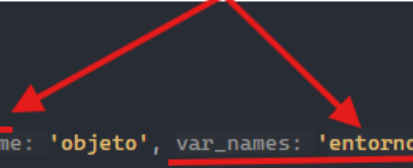


```
0 references | 0 overrides
public function edit($id): View
{
    $entorno = Entorno::find(id: $id);
    $categorias = Categoria::pluck(column: 'nombre', key: 'id');
    return view(view: 'entorno.edit', data: compact(var_name: 'entorno', var_names: 'categorias'));
}
```

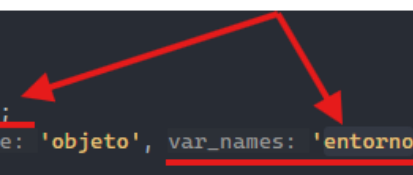


Y hacemos lo mismo en objetos con referencia a entornos Para ello en ObjetosController.php:

```
0 references | 0 overrides
public function create(): View
{
    $objeto = new Objeto();
    $entorno = Entorno::pluck(column: 'nombre', key: 'id');
    return view(view: 'objeto.create', data: compact(var_name: 'objeto', var_names: 'entorno'));
}
```



```
public function edit($id): View
{
    $objeto = Objeto::find(id: $id);
    $entorno = Entorno::pluck(column: 'nombre', key: 'id');
    return view(view: 'objeto.edit', data: compact(var_name: 'objeto', var_names: 'entorno'));
}
```



## CREACIÓN DE SELECTS

Vamos a crear un select para aquellos campos relacionados para ellos haremos lo siguiente dentro de form.blade.php dentro de entornos:

```

<div class="form-group mb-2 mb20">
  <label for="entorno_id" class="form-label">{{ __key: 'Entorno Id' }}</label>
  {{ Form::select('entorno_id', $entornos, $objeto->entorno_id, ['class' => 'form-control' . ($errors->has('entorno_id') ? ' is-invalid' : ''), 'placeholder' => 'Entorno Id']) }}
  {!! $errors->first('entorno_id', '<div class="invalid-feedback">:message</div>') !!}
</div>

```

Y dentro de objetos hacemos algo similar:

```

<div class="form-group mb-2 mb20">
  <label for="categoria_id" class="form-label">{{ __key: 'Categoria Id' }}</label>
  {{ Form::select('categoria_id', $categorias, $entorno->categoria_id, ['class' => 'form-control' . ($errors->has('categoria_id') ? ' is-invalid' : ''), 'placeholder' => 'Categoria Id']) }}
  {!! $errors->first('categoria_id', '<div class="invalid-feedback">:message</div>') !!}
</div>

```

## MODIFICACIÓN DE CABECERAS Y DATOS

Ahora modificaremos tanto el nombre del encabezado de las tablas como el dato que tiene dentro para que haga referencia al nombre asociado a la propia id del campo, para ello vamos a index.blade dentro de entornos y objetos y retocamos lo siguiente:

```

<div class="card-body bg-white">
  <div class="table-responsive">
    <table class="table table-hover">
      <thead class="thead">
        <tr class="table-primary">
          <th>No</th>

          <th>Nombre</th>
          <th>Entorno</th>
          <th>Dimensiones</th>
          <th>Posicion</th>
          <th>Color</th>

        <th></th>
        </tr>
      </thead>
      <tbody>
        @foreach ($objetos as $objeto)
          <tr class="table-info">
            <td>{{ ++$i }}</td>

            <td>{{ $objeto->nombre }}</td>
            <td>{{ $objeto->entorno->nombre }}</td>
            <td>{{ $objeto->dimensiones }}</td>
            <td>{{ $objeto->posicion }}</td>
            <td>{{ $objeto->color }}</td>
          </tr>
        @endforeach
      </tbody>
    </table>
  </div>
</div>

```

```

<div class="card-body bg-white">
  <div class="table-responsive">
    <table class="table table-striped table-hover">
      <thead class="thead">
        <tr class="table-danger border-3 border-dark">
          <th>No</th>

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

          <th></th>
        </tr>
      </thead>
      <tbody>
        @foreach ($entornos as $entorno)
          <tr class="table-danger border-3 border-dark">
            <td>{{ ++$i }}</td>

            <td>{{ $entorno->categoria->nombre }}</td>
            <td>{{ $entorno->nombre }}</td>
            <td>{{ $entorno->musica }}</td>
          </tr>
        @endforeach
      </tbody>
    </table>
  </div>
</div>

```

También se cambiaron las vistas show() de la siguiente forma:

```

<div class="card-body bg-white">
  <div class="form-group mb-2 mb20">
    <strong>Nombre:</strong>
    {{ $objeto->nombre }}
  </div>
  <div class="form-group mb-2 mb20">
    <strong>Entorno:</strong>
    {{ $objeto->entorno->nombre }}
  </div>
  <div class="form-group mb-2 mb20">
    <strong>Dimensiones:</strong>
    {{ $objeto->dimensiones }}
  </div>
  <div class="form-group mb-2 mb20">
    <strong>Posicion:</strong>
    {{ $objeto->posicion }}
  </div>
  <div class="form-group mb-2 mb20">
    <strong>Color:</strong>
    {{ $objeto->color }}
  </div>
</div>

```

```

<div class="card-body bg-white">
  <div class="form-group mb-2 mb20">
    <strong>Categoria:</strong>
    {{ $entorno->categoria->nombre }}
  </div>
  <div class="form-group mb-2 mb20">
    <strong>Nombre:</strong>
    {{ $entorno->nombre }}
  </div>
  <div class="form-group mb-2 mb20">
    <strong>Musica:</strong>
    {{ $entorno->musica }}
  </div>
</div>

```

## RESTRINGIR ACCESO

Ahora vamos a hacer que no se pueda acceder a las vistas si el usuario no está loggeado para ello añadiremos lo siguiente tanto en web.php como en app.blade.php:

```

Auth::routes();
Route::resource(name: 'categorias', controller: App\Http\Controllers\CategoriaController::class)->middleware(middleware: 'auth');
Route::resource(name: 'entornos', controller: App\Http\Controllers\EntornoController::class)->middleware(middleware: 'auth');
Route::resource(name: 'objetos', controller: App\Http\Controllers\ObjetoController::class)->middleware(middleware: 'auth');

@if (Auth::check())
<ul class="navbar-nav me-auto">
  <li class="nav-item">
    <a class="nav-link text-light" href="{{ route(name: 'categorias.index') }}">{{ __(key: 'Categorias') }}</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-light" href="{{ route(name: 'entornos.index') }}">{{ __(key: 'Entornos') }}</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-light" href="{{ route(name: 'objetos.index') }}">{{ __(key: 'Objetos') }}</a>
  </li>
</ul>
@endif

```



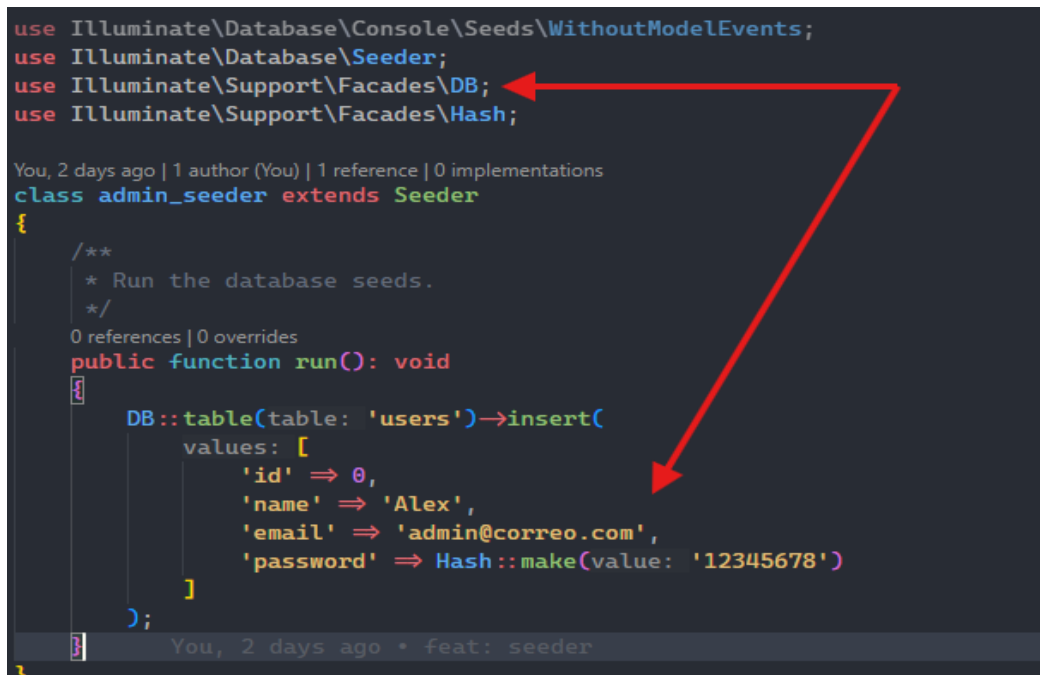
De esta forma solo podrá ver los accesos a las vistas si está registrado en la aplicación.

## CREACIÓN SEEDERS

Ahora creamos el seeder para tener insertados los datos de algunos usuarios, para ellos hicimos lo siguiente:

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan make:seeder admin_seeder
```

Ahora añadimos la siguiente línea y a continuación los usuarios, en nuestro caso solo 1.



```
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;

You, 2 days ago | 1 author (You) | 1 reference | 0 implementations
class admin_seeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    0 references | 0 overrides
    public function run(): void
    {
        DB::table('users')->insert(
            values: [
                'id' => 0,
                'name' => 'Alex',
                'email' => 'admin@correo.com',
                'password' => Hash::make(value: '12345678')
            ]
        );
    }
}

You, 2 days ago * feat: seeder
```

A continuación, nos dirigimos a DatabaseSeeder.php y llamamos a nuestro seeder.

```
0 references | 0 overrides
public function run(): void
{
    // \App\Models\User::factory(10)->create();

    // \App\Models\User::factory()->create([
    //     'name' => 'Test User',
    //     'email' => 'test@example.com',
    // ]);

    $this->call(class: admin_seeder::class);
}
```

Por último debemos insertar el siguiente comando en consola, para generar todos los seeders que tengamos:

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan db:seed
```

## CONFIGURACIÓN LOCALES

Para tener la validación de errores en español primero debemos descargar las traducciones de:

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

Luego, en config:

```

/*
|-----|
| Application Locale Configuration      You, 2 weeks ago • Configuración del
|-----|
|
| The application locale determines the default locale that will be used
| by the translation service provider. You are free to set this value
| to any of the locales which will be supported by the application.
|
*/

'locale' => 'es',

```

Luego en resources creamos una carpeta y la llamamos lang e introducimos lo que hemos descargado y por último ejecutamos los últimos comandos:

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan config:clear
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan cache:clear
```

```
C:\Users\Alejandro\Desktop\virtualmeditate>php artisan optimize:clear
```

## ESTILOS

Por último los estilos actuales, en la página principal:

```

@section(section: 'content')
<div class="d-flex justify-content-center align-items-center vh-100">
  
  <div
    class="justify-content-center flex-column absolute-center text-center position-absolute top-10 start-50 translate-middle">
    <p class="fw-bolder font sized-font">Virtual meditate</p>
    <h2 class="fw-bolder font text-black-50">A moment of calm in a world of hurry</h2>
  </div>
</div>
<style>
  @import url('https://fonts.googleapis.com/css2?family=Eagle+Lake&family=Syne+Mono&display=swap');
  You, 19 hours ago | 1 author (You)
  .font{
    font-family: 'Eagle Lake', cursive;
  }

  Alex, 14 hours ago | 2 authors (You and one other)
  .sized-font{
    font-size: 6dvw;
  }

  You, 19 hours ago | 1 author (You)
  main{
    position: fixed;
    width: 100%;
  }
  You, 19 hours ago • feat: update category references and improve la...

```

Se colocó una imagen de fondo, así como el nombre de y slogan de la empresa.

```

<nav class="navbar navbar-expand-md navbar-light bg-black shadow-sm">
  <div class="container">
    <a class="navbar-brand" href="{{ url(path: '/') }}">
      
    </a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
      data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
      aria-expanded="false" aria-label="{{ __(key: 'Toggle navigation') }}">
      <span class="navbar-toggler-icon"></span>
    </button>

    <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 text-light" href="{{ route(name: 'categorias.index') }}">{{ __(key: 'Categorias') }}</a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-light" href="{{ route(name: 'entornos.index') }}">{{ __(key: 'Entornos') }}</a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-light" href="{{ route(name: 'objetos.index') }}">{{ __(key: 'Objetos') }}</a>
        </li>
      </ul>
    </div>
  </div>
</nav>

```

El nav ha sido cambiado para tener un fondo negro con letras blancas, así como la inclusión del nombre de la empresa.

Luego, en el resto de vistas de las tablas se han cambiado los estilos para separar el contenido del nav así como resaltar cada tabla de un color diferente que representa el logo de la empresa inspirado en el equilibrio cada vista tiene pequeños detalles que la diferencian de las demás.

## POSIBLES IMPLEMENTACIONES

Pese a los cambios realizados, los estilos podrían mejorarse añadiendo estilos acordes a las vistas de añadir, editar y visualizar de las diferentes tablas.

Con el tiempo y con el crecimiento de la empresa podría ser necesaria una ampliación en la base de datos, así como en la cantidad de tablas y campos de estas.