TRINAN

MANUAL DEL PROGRAMADOR

1.Requisitos del sistema:

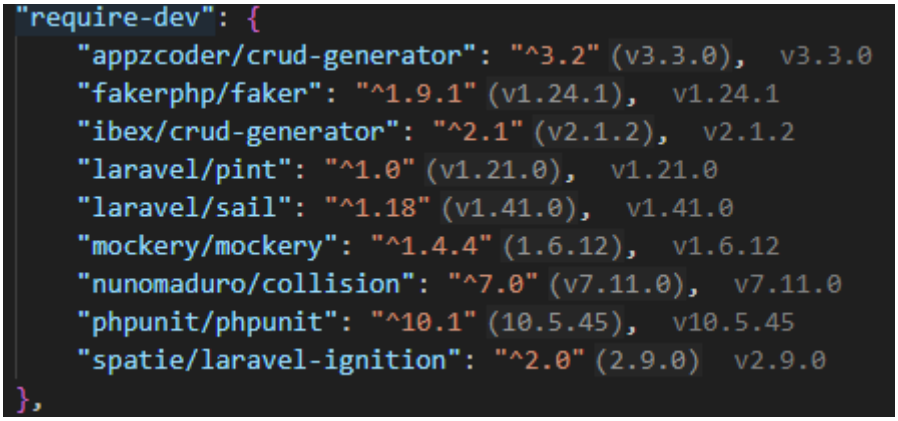
-Xampp: servidor local
-Composer: gestor de dependencias PHP
-Node.js: paquete de librerías en javascript

2.Creación del proyecto:

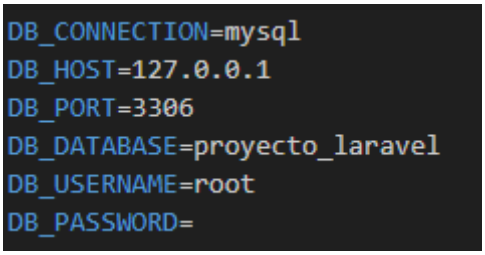
Antes de nada crearemos una base de datos en nuestro gestor PHPMyAdmin llamada **proyecto_laravel**.

Desde C:\xampp\htdocs ejecutamos el comando: **composer create-project laravel/laravel personas "10.*"**

Una vez creado, se añadió debe añadir al archivo composer.json , en "require-dev", "appzcoder/crud-generator" : "^3.2".



A continuación en el archivo .env se debe añadir la base de datos en DB_DATABASE



3. Creamos las migraciones

Desde nuestra carpeta de proyecto ejecutaremos los siguientes comandos en el siguiente oden para crear nuestras tablas:

personas>php artisan make:migration create_condiciones_table
personas>php artisan make:migration create_personas_table

Una vez creadas introduciremos sus respectivos campos en cada una de las tablas;

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

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

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

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create(table: 'personal_access_tokens', callback: function (Blueprint $table): void {
            $table->id();
            $table->morphs(name: 'tokenable');
            $table->string(column: 'name');
            $table->string(column: 'token', length: 64)->unique();
            $table->text(column: 'abilities')->nullable();
            $table->timestamp(column: 'last_used_at')->nullable();
            $table->timestamp(column: 'expires_at')->nullable();
            $table->timestamps();
        });
    }

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

Despues de esto migraremos nuestras tablas a la base de datos ejecutando **personas>php artisan migrate** .

4. Autenticación

Los comandos que tendremos que ejecutar son los siguientes y por este orden:
personas>composer require laravel/ui
personas>php artisan ui bootstrap --auth
personas>npm install
personas>npm run dev
Este último comando lo debemos ejecutar en otro terminal y dejarlo ejecutándose antes de que ejecutemos nuestro 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:

Desde otrp terminal se debe estar ejecutando:
libreria>npm run dev

personas>composer require ibex/crud-generator --dev
personas>php artisan vendor:publish --tag=crud
personas>php artisan make:crud condiciones
personas>php artisan make:crud personas

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
<?php

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


/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "web" middleware group. Make something great!
|
*/

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

Auth::routes();
Route::resource(name: 'personas',controller: App\Http\Controllers\PersonaController::class)->middleware(middleware: 'auth');
Route::resource(name: 'condiciones',controller: App\Http\Controllers\CondicioneController::class)->middleware(middleware: 'auth');
Route::get(uri: '/home', action: [App\Http\Controllers\HomeController::class, 'index'])->name(name: 'home');
```

7. Creación de los enlaces a Personas y Condiciones

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 Personas y a las Condiciones. Para ello vamos a app\resources\views\layouts\app.blade.php y creamos dos enlaces en la parte izquierda del menú de navegación:

```
<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(name: 'personas.index') }}">{{ __(key: 'Personas') }}</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route(name: 'condiciones.index') }}">{{ __(key: 'Condiciones') }}</a>
        </li>
    </ul>
    @endif
```

8. Acceso a los datos de Condiciones desde Personas

Para poder acceder a los datos de las categorías desde el crud Personas, debo poder llamar a la clase Condiciones para poder acceder a sus datos. Para ello vamos a modificar un par de métodos de PersonaController.php. Lo primero que debemos hacer es añadir el modelo Condicione al principio del controlador:
use App\Models\Condicione

```php
public function create(): View
{
    $persona = new Persona();
    $condiciones=Condicione::pluck(column: 'nombre',key: 'id');
    return view(view: 'persona.create', data: compact(var_name: 'persona',var_names: 'condiciones'));
}
```

```php
public function edit($id): View
{
    $persona = Persona::find(id: $id);
    $condiciones=Condicione::pluck(column: 'nombre',key: 'id');
    return view(view: 'persona.edit', data: compact(var_name: 'persona',var_names: 'condiciones'));
}
```

9. Creación de un select para las condiciones

En la vista form.blade.php de Persona:

```blade
<div class="form-group mb-2 mb20">
    {{-- <label for="condiciones_id" class="form-label">{{ __('Condiciones Id') }}</label>
    <input type="text" name="condiciones_id" class="form-control @error('condiciones_id') is-invalid @enderror" value="{{ old('condiciones_id', $persona?->condiciones_id) }}" id="condiciones_id" placeh
    {!! $errors->first('condiciones_id', '<div class="invalid-feedback" role="alert"><strong>:message</strong></div>') !!} --}}
    {{ Form::label('condiciones') }}
    {{ Form::select('condiciones_id',$condiciones, $persona->condiciones_id, ['class' => 'form-control' . ($errors->has('condiciones_id') ? ' is-invalid' : ''), 'placeholder' => 'Condicion']) }}
    {!! $errors->first('condiciones_id', '<div class="invalid-feedback">:message</div>') !!}
</div>
```

10. Modificación del nombre del encabezado Condicion de mi tabla

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

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

    <th >Condiciones</th>
    <th >Nombre</th>
    <th >Apellidos</th>
    <th >Dni</th>
    <th >Telefono</th>
    <th >Direccion</th>

        <th>Acciones</th>
    </tr>
</thead>
<tbody>
    @foreach ($personas as $persona)
        <tr>
            <td>{{ ++$i }}</td>

        <td >{{ $persona->condicione->nombre }}</td>
        <td >{{ $persona->nombre }}</td>
        <td >{{ $persona->apellidos }}</td>
        <td >{{ $persona->dni }}</td>
        <td >{{ $persona->telefono }}</td>
        <td >{{ $persona->direccion }}</td>
```

Puedo acceder a $persona->condicione->nombre directamente porque en el modelo de Persona (Persona .php) tengo implementado un método condicione () con dicha asociación.

En la vista show.blade.php de persona:

```
<div class="card-body bg-white">

            <div class="form-group mb-2 mb20">
                <strong>Condiciones:</strong>
                {{ $persona->condicione->nombre }}
            </div>
            <div class="form-group mb-2 mb20">
                <strong>Nombre:</strong>
                {{ $persona->nombre }}
            </div>
```

11. Creación del usuario por defecto

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

personas>php artisan make:seeder users_Seeder

Una vez creado, se ponen los datos del usuario en users_Seeder.php, tal y como aparece a continuación:

```php
class users_seeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    0 references | 0 overrides
    public function run(): void
    {
        DB::table(table: 'users') -> insert(values: [
            'id' => 1,
            'name' => "admin",
            'email' => "admin@gmail.com",
            'password' =>Hash::make(value: '12345678')
        ]);
    }
}
```

Después de esto se ejecuta este comando para introducir el usuario puesto en la base de datos
personas>php artisan db:seed –class=users_seeder

12. Ajustes finales

Por último vamos a bloquear los menús de Personas y Condiciones 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(uri: '/', action: function (): Factory|View {
    return view(view: 'welcome');
});

Auth::routes();
Route::resource(name: 'personas',controller: App\Http\Controllers\PersonaController::class)->middleware(middleware: 'auth');
Route::resource(name: 'condiciones',controller: App\Http\Controllers\CondicioneController::class)->middleware(middleware: 'auth');
Route::get(uri: '/home', action: [App\Http\Controllers\HomeController::class, 'index'])->name(name: 'home');
```

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

```
<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(name: 'personas.index') }}">{{ __(key: 'Personas') }}</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route(name: 'condiciones.index') }}">{{ __(key: 'Condiciones') }}</a>
        </li>
    </ul>
    @endif

    <!-- Right Side Of Navbar -->
```

13. TRADUCCIONES AL ESPAÑOL

Para traducir los mensajes por defecto del crud al español, se siguieron los siguientes pasos:
1. Ejecutar el comando composer require laravel/lang common.
2. Descargar la carpeta es de https://github.com/Laraveles/spanish.
3. Ubicar la carpeta en resources/lang.
4. Modificar el archivo config/app.php en la linea ‘locale’ para que quede ‘locale’ => ‘es’.
5. Limpiar la caché mediante los comandos
personas>php artisan config:clear
personas>php artisan cache:clear
personas>php artisan optimize:clear.