MANUAL DEL PROGRAMADOR

1. Creación del proyecto

Lo primero que se debe hacer para iniciar el proyecto en Laravel es crear la base de datos en PHPMyAdmin.

Posteriormente, es necesario modificar el archivo .env, estableciendo el nombre de la base de datos, que en este caso será proyecto_laravel como se ve en la imagen:

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

A continuación, se procede a crear el proyecto. Para ello, se abre la terminal y se accede a la carpeta donde se desea crearlo. Luego, se ejecuta el siguiente comando:

htdocs>composer create-project laravel/laravel nombre_proyecto “10.*”

Después de crear el proyecto, hay que abrirlo en el visual studio code y habrá que ir al archivo composer.json. Donde nos iremos a la sección de require-dev, una vez ahí se añade la siguiente línea:

“appzcoder/crud-generator”:”^3.2”

Por lo que quedaría así:

```
"require-dev": {
    "appzcoder/crud-generator":"^3.2",
    "fakerphp/faker": "^1.9.1" (v1.24.1),
    "laravel/pint": "^1.0" (v1.20.0),
    "laravel/sail": "^1.18" (v1.41.0),
    "mockery/mockery": "^1.4.4" (1.6.12),
    "nunomaduro/collision": "^7.0" (v7.11.0),
    "phpunit/phpunit": "^10.1" (10.5.44),
    "spatie/laravel-ignition": "^2.0" (2.9.0)
},
```

Para verificar que todo esté configurado correctamente, se inicia el servidor con el siguiente comando:

nombre_proyecto>php artisan serve

Si todo está bien tendría que aparecer la página de bienvenida de laravel.

2. Creación de las migraciones

Es importante tener en cuenta el orden de creación de las migraciones, ya que esto influye en la generación de los métodos up y down. Para crear las migraciones, se utilizan los siguientes comandos:

nombre_proyecto>php artisan make:migration create_nombreTabla_table (campo relacional)
nombre_proyecto>php artisan make:migration create_nombreOtraTabla_table
nombre_proyecto>php artisan migrate

Si el orden es correcto, Laravel generará automáticamente las funciones up y down. En caso contrario, deberán ser creadas manualmente.

```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: 'productos', callback: function (Blueprint $table): void {
            $table->id();
            $table->timestamps();
        });
    }

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

3. Autenticación

Para habilitar la autenticación en el proyecto, se utiliza laravel/ui con Bootstrap. Se ejecutan los siguientes comandos en la terminal:

nombre_proyecto>composer require laravel/ui
nombre_proyecto>php artisan ui bootstrap –auth
nombre_proyecto>npm install

Abre otro terminal, ve a la ruta donde está el proyecto y se pone este comando
nombre_proyecto>npm run dev

Ahora, si se ejecuta el proyecto debería aparecer arriba a la derecha login y register. Vamos a register y nos creamos un usuario nuevo (laravel guardara esos datos en la tabla users que crea automáticamente al crear el proyecto).

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

Con vite en ejecución, se ejecutan los siguientes comandos:

nombre_proyecto>composer require ibex/crud-generator –dev
nombre_proyecto>php artisan vendor:publish –tag=crud
nombre_proyecto>php artisan make:crud nombreTabla (campo relacional)
nombre_proyecto>php artisan make:crud nombreOtraTabla

Al ejecutar make, se solicita seleccionar el tipo de crud deseado. Las opciones son bootstrap, tailwind, liveweire y api. Se recomienda seleccionar bootstrap, ya que genera automáticamente los modelos, controladores y vistas básicas para el proyecto.

5. Acceso a los cruds

Para acceder a los CRUDs creados, es necesario definir las rutas correspondientes. Esto se realiza en el archivo web.php, ubicado en routes. Se deben agregar las siguientes rutas de tipo resource, además de importar Auth:

```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: 'ventas',controller: App\Http\Controllers\VentaController::class)->middleware(middleware: 'auth');
Route::resource(name: 'productos',controller: App\Http\Controllers\ProductoController::class)->middleware(middleware: 'auth');

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

6. Creación de los enlaces a Productos y Ventas

Al ejecutar el comando del crud-generator, se generan los archivos necesarios para un CRUD funcional, incluyendo las vistas. Para añadir un menú de navegación, se edita el archivo app\resources\views\layouts\app.blade.php. Dentro de la sección comentada como **"Left Side Of Navbar"**, se agregan los siguientes enlaces:

```
<div class="collapse navbar-collapse" id="navbarSupportedContent">
    <!-- Left Side Of Navbar -->
    <ul class="navbar-nav me-auto">
        @if(Auth::check())
    <ul class="navbar-nav me-auto">
        <li class="nav-item">
            <a class="nav-link" href="{{ route(name: 'productos.index') }}">{{ __(key: 'Productos') }}</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="{{ route(name: 'ventas.index') }}">{{ __(key: 'Ventas') }}</a>
        </li>
    </ul>
    @endif
    </ul>
```

7. Modificación del Nombre del Encabezado de Ventas en la Tabla

Se edita el archivo app\resources\views\producto\index.blade.php para personalizar los encabezados de la tabla. Además, se puede utilizar la localización en español con los siguientes comandos:

composer require laravel-latam/spanish
php artisan vendor:publish --provider="LaravelLatam\Spanish\SpanishServiceProvider" --tag="spanish"

Esto permitirá mostrar los nombres de los encabezados y mensajes del sistema en español.

8. Creador de administradores

Si queremos crear una serie de administradores tenemos que ir a DatabaseSeeder.php y tenemos que poner lo siguiente dentro de la función run:

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

    DB::table(table: 'users')->insert(values: [
        [
            'name' => 'Admin1',
            'email' => 'admin1@gmail.com',
            'email_verified_at' => now(),
            'password' => Hash::make(value: '1234'),
            'remember_token' => Str::random(length: 10),
        ],
        [
            'name' => 'Admin2',
            'email' => 'admin2@gmail.com',
            'email_verified_at' => now(),
            'password' => Hash::make(value: '1234'),
            'remember_token' => Str::random(length: 10),
        ]
    ]);
}
```