# **Mini-CRM con Laravel**

**Guia Practica para entender y crear un Mini-CRM con Laravel**

## 1. Usando Laravel Auth para poder hacer log in como administrador y como usuario.

***Entendiendo: composer require laravel/ui***

El primer paso para empezar a crear nuestro Mini-CRM es crear un proyeto, esto se logra facilmente ejecutando el siguiente codigo en la terminal:

>>> composer create-project laravel/laravel Mini-CRM

Luego de haber creado nuestro Proyecto con nombre Mini-CRM, vamos a usar:

**>>> composer require laravel/ui**

Pero que significa es comando, bueno veamos un poco mas para entenderlo

### ¿Qué es composer require laravel/ui?

**Composer require laravel/ui** es un comando de Composer que instala el paquete **laravel/ui** en un proyecto de Laravel. Este paquete proporciona un conjunto de herramientas para crear interfaces de usuario (UI) para aplicaciones Laravel.

Laravel/UI es un paquete que proporciona las vistas y los recursos necesarios para construir interfaces de usuario básicas. Incluye plantillas de autenticación, que son formularios de inicio de sesión y registro preconstruidos, estilos y scripts.

Al ejecutar:

>>> composer require laravel/ui

Se crean los siguientes archivos:

* composer.json
* composer.lock
* vendor/laravel/ui/composer.json
* vendor/laravel/ui/composer.lock
* vendor/laravel/ui/src/
* vendor/laravel/ui/auth-backend/

Estos archivos son necesarios para que el paquete laravel/ui funcione correctamente. Sin embargo, no contienen la interfaz de usuario básica de Laravel.

La interfaz de usuario básica de Laravel se genera al ejecutar el comando

>>> php artisan ui bootstrap

O el comando:

>>> php artisan ui vue --auth

Para el desarrollo de este proyecto usaremos el segundo.

El comando php artisan ui vue --auth genera los siguientes archivos y directorios:

* resources/views/auth
* resources/views/layouts
* resources/js/app.js
* resources/sass/app.scss

Estos archivos y directorios contienen la interfaz de usuario básica de Laravel, que incluye las páginas de autenticación, el diseño de la interfaz de usuario y el código JavaScript y CSS. La interfaz de usuario básica de Laravel generada por este comando utiliza el framework Vue.js.

Para ver si correctamente si el paquete se instalo y correra correctamente debemos ejecutar el siguiente comando:  
  
>>> npm run dev

Sin embargo si nos bota el siguiente mensaje

> dev

> vite

sh: 1: vite: not found

Es porque el paquete **vite** no esta instalado, para poder corregir esto debemos correr el siguiente comando:

>>> npm install vite

Y nuevamente corremos **npm run dev,** si todo esta bien podremos acceder al formulario de autenticacion a traves de la ruta :

**> localhost:8000/login**

Luego debemos modificar nuestros archivos de migracion de para la base de datos tal cual se puede ver en el archivo:

database/migrations/2014\_10\_12\_000000\_create\_users\_table.php

Luego de acuerdo a este archivo tambien debemos modificar el archivo:

App/Models/User.php

***Creamos un controlador para administracion de usuarios***

>>> php artisan make:controller AdminController

Al ejecutar este comando, estamos generando un controlador que será responsable de manejar las funciones relacionadas con la administración.

***Modificar AdminController para que use el middleware de autenticación:***

// app/Http/Controllers/AdminController.php

public function \_\_construct()

{

$this->middleware('auth');

}

En el constructor del controlador (AdminController), agregamos el middleware **$this->middleware('auth')**. Esto significa que cualquier método en este controlador requerirá que el usuario esté autenticado para acceder a él.

Entonces, cuando un usuario intenta acceder a una función en AdminController, Laravel verificará si el usuario está autenticado antes de permitir el acceso. Esto asegura que solo los administradores (o cualquier usuario autenticado) puedan interactuar con las funciones de administración.

## **2. Usando database seeds para poder crear los usuarios en la tabla users**

* Lo primero que tenemos que hacer es hacer una modificiacion a nuestro archivo **app/Models/User.php ,**  para agregar el campo **‘Role’** a nuestro modelo.
* Modificamos tambien la migracion para incluir el nuevo campo **‘Role’,**  esto en el archivo :

**database/migrations/2014\_10\_12\_000000\_create\_users\_table**

* Creamos un seeder llamado UserSeeder con el comando php artisan make:seeder UserSeeder.
* En UserSeeder, utilizamos Eloquent para insertar usuarios administrador y no administrador
* Utilizamos factory para crear 20 usuarios adicionales de forma aleatoria.
* En el seeder principal (DatabaseSeeder.php), llamamos al seeder de usuarios (UserSeeder) para que se ejecute cuando se ejecute php artisan db:seed.
* Finalmente, ejecutamos php artisan db:seed para poblar la base de datos con usuarios en la tabla users.

3. Implementando la funcionalidad CRUD para dos items de menu: Companies y Employees.

Esta parte de la implementacion lo haremos en 9 etapas, para asi tener una mejor estructura de actividades para su implementacion, como sigue a continuacion:

1. **Creación de Migraciones:**

Utilizaremos el comando php artisan make:model -m para crear modelos junto con migraciones para Company y Employee.

>>> php artisan make:model -m Company

>>> php artisan make:model -m Employee

Esto generará 4 archivos, 2 de modelo y 2 de migración para ambas entidades. Después de hacer esto, revisemos las migraciones para asegurarnos de que estén configuradas correctamente.

* + App/Models/Company.php
  + App/Models/Employee.php
  + database/Migrations/(timestamp)\_create\_companies\_table.php
  + database/Migrations/(timestamp)\_create\_employees\_table.php

1. Definición de Estructuras de Base de Datos:

En cada migración, definiremos los campos necesarios para las tablas de companies y employees.

Para la tabla companies:

Schema::create('companies', function (Blueprint $table) {

$table->id();

$table->string('name')->required();

$table->string('email')->nullable();

$table->string('logo', 255)->nullable();

$table->string('website');

$table->timestamps();

});

Para la tabla employees:

Schema::create('employees', function (Blueprint $table) {

$table->id();

$table->string('first\_name')->required();

$table->string('last\_name')->required();

$table->foreignId('company')->constrained('companies');

$table->string('email')->nullable();

$table->string('phone')->nullable();

$table->timestamps();

});

1. Ejecución de Migraciones:

Ejecutaremos php artisan migrate para aplicar los cambios a la base de datos y crear las tablas.

>>> php artisan migrate

1. Creación de Controladores:

Utilizaremos el comando php artisan make:controller para crear controladores llamados CompanyController y EmployeeController.

>>> php artisan make:controller CompanyController

>>> php artisan make:controller EmployeeController

1. Definición de Métodos del Controlador:

En los controladores, definiremos los métodos para realizar operaciones CRUD (index, create, store, show, edit, update, destroy).

1. Creación de Rutas:

Registraremos las rutas necesarias en el archivo web.php para que Laravel pueda dirigir las solicitudes a los métodos de los controladores.

1. Creación de Vistas (Opcional):

Podemos crear vistas para cada operación CRUD si deseamos una interfaz gráfica para interactuar con las entidades.

1. Configuración de Middleware de Autenticación:

Aseguraremos que las rutas de CRUD estén protegidas por el middleware de autenticación, para que solo los usuarios autenticados puedan acceder a ellas.

1. Prueba de Operaciones CRUD:

Utilizaremos herramientas como Postman o las rutas web para probar cada operación CRUD y verificar que funcione correctamente.