Manual de Programador

Manual para el programador de la aplicación ExpiryDeals

Descripción General del Proyecto

Este proyecto se creó para la gestión de clientes de una aplicación móvil pensada para proporcionar información a dichos clientes de aquellos productos que su fecha de vencimiento era próxima y por lo tanto ofrecían un gran descuento. En este proyecto nos encargamos de que los clientes puedan suscribirse así como, una vez estén suscritos, puedan modificar sus datos. A su vez, los administradores se encargan de gestionar dichos clientes.

Creación del Proyecto

Para la creación del proyecto tenemos que seguir una serie de pasos.

Requisitos Necesarios

Para poder crear nuestro proyecto en Laravel necesitamos tener instalados los siguientes componentes:

- **Composer** → gestor de dependencias PHP
- **Artisan** →CLI que nos va a permitir gestionar nuestros proyectos desde la consola.
- **Xampp** → nuestro servidor local (también podríamos usar Laragon o LAMP para Linux)
- **[Node.js]** → dependiendo de la versión de Laravel que tengamos (para poder crear los elementos de autentificación de nuestro proyecto). Node.js es un entorno de ejecución de JavaScript , permitiendolo ejecutar fuera de un navegador web.

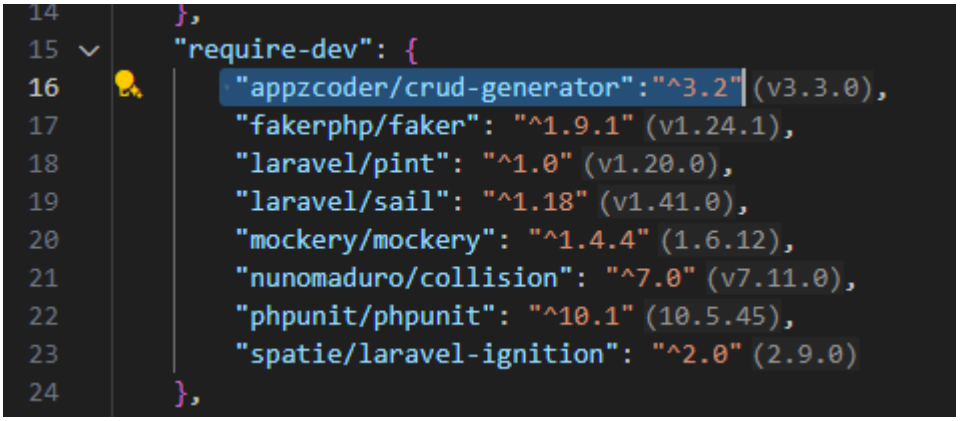
Una vez tenemos dichos componentes descargados podemos empezar. El primer paso es la creación del proyecto. Para ello nos metemos en la consola de Windows y nos introducimos de la carpeta htdocs e introducimos el siguiente comando:

composer create-project laravel/laravel proyecto_laravel“10.*”

Con este comando creamos un proyecto laravel en la versión 10. A partir de aquí hacemos una serie de modificaciones en el proyecto en visual studio code. Primero vamos al archivo .env y nos aseguramos de los siguientes datos:

```
11  DB_CONNECTION=mysql
12  DB_HOST=127.0.0.1
13  DB_PORT=3306
14  DB_DATABASE=proyecto_laravel
15  DB_USERNAME=root
16  DB_PASSWORD=
```

Es necesario crear una base de datos en xampp y darle el nombre de proyecto_laravel. Ese nombre será el mismo que introduzcamos en DB_DATABASE y que servirá de unión entre lo que hagamos en el visual y nuestra base de datos, por lo que es muy importante que coincidan los nombres.
Luego nos introducimos en “composer.json” y nos aseguramos de los siguientes datos:

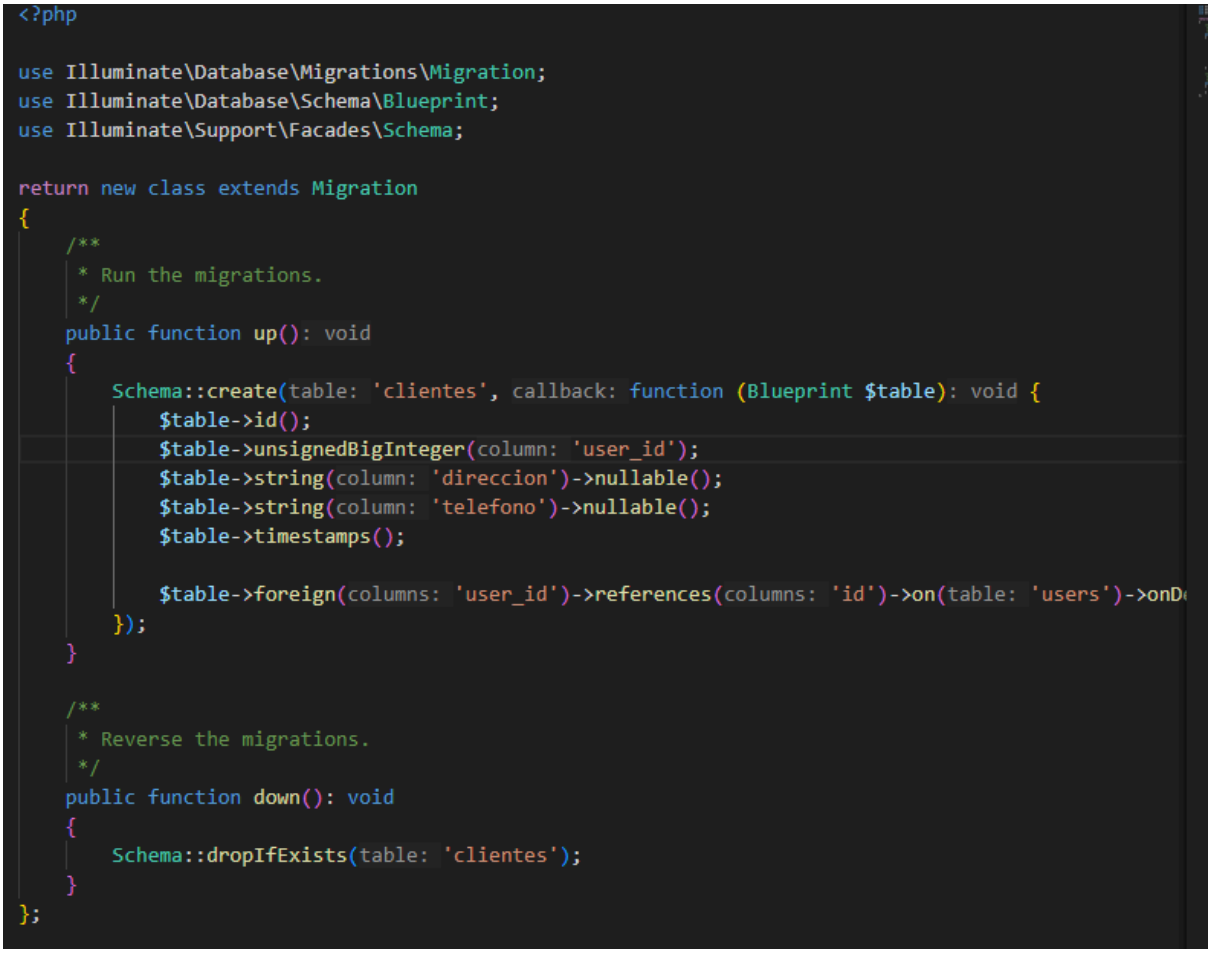


Una vez hecho esto podemos empezar con el proyecto.

Creación de las tablas y sus migraciones

Procedemos a crear las tablas. Para ello tenemos que ejecutar el siguiente comando.
proyecto_laravel>php artisan make:migration create_clientes_table
proyecto_laravel>php artisan make:migration create_administradores_table
Estos comandos los usamos para crear las dos tablas principales, la de los clientes y la de los administradores

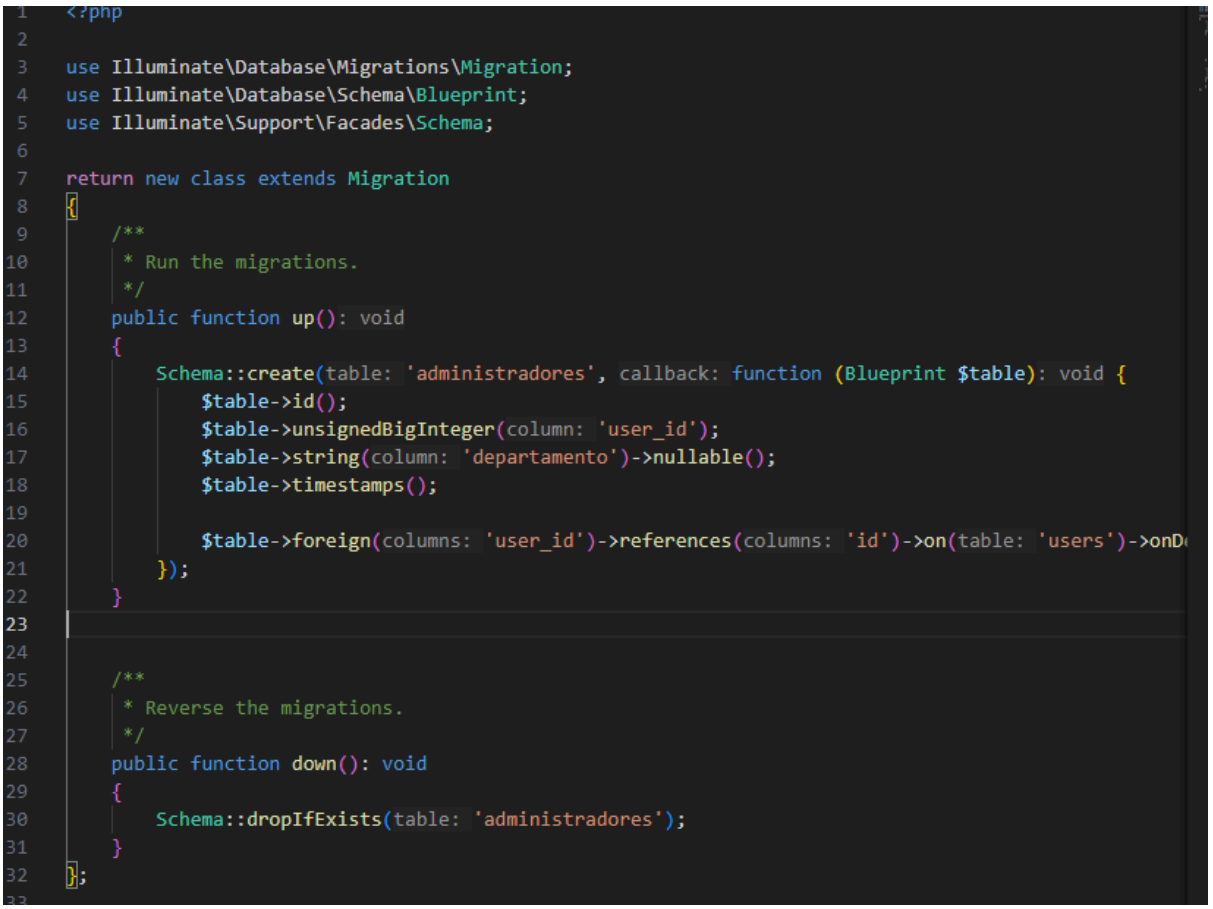
Tabla Clientes



```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: 'clientes', callback: function (Blueprint $table): void {
            $table->id();
            $table->unsignedBigInteger(column: 'user_id');
            $table->string(column: 'direccion')->nullable();
            $table->string(column: 'telefono')->nullable();
            $table->timestamps();

            $table->foreign(columns: 'user_id')->references(columns: 'id')->on(table: 'users')->onD
        });
    }

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

Esta es la estructura de la tabla clientes. Los datos de esta tabla se introducen mediante la propia aplicación. Más adelante explico como.

Tabla Administradores



```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: 'administradores', callback: function (Blueprint $table): void {
            $table->id();
            $table->unsignedBigInteger(column: 'user_id');
            $table->string(column: 'departamento')->nullable();
            $table->timestamps();

            $table->foreign(columns: 'user_id')->references(columns: 'id')->on(table: 'users')->onD
        });
    }


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

Estructura de la tabla administradores. Los datos de esta tabla los introducimos mediante un seeder.

Una vez creadas las dos tablas y creada también su estructura procedemos a hacer la migración, que consiste en establecer esta estructura en la base de datos previamente creada. Para ello realizamos el siguiente comando:
php artisan migrate

Seeder para los Administradores

Una vez hecho el punto anterior, procedemos a crear el seeder para los administradores. Hacemos el siguiente comando en la consola: php artisan make:seeder AdminSeeder

Esto creará el archivo AdminSeeder, una vez hecho completamos con los datos que vamos a necesitar y ejecutamos el siguiente comando: php artisan db:seed --class=AdminSeeder

Alteración de las tablas

Voy a necesitar modificar las tablas creadas, a mayores de crear una tabla nueva. Para modificar las tablas ya creadas hago el siguiente comando:php artisan make:migration add_column_to_clientes_table --table=clientes

Con esto puedo modificar la tabla de clientes. También lo haré con la tabla de administradores. A su vez necesito crear la tabla “aciones_administrativas” mediante el comando que explique anteriormente.

Creación de los Controladores Necesarios

Para la creación de los controladores podemos hacerlo de dos formas diferentes. La primera y a su vez la más cómoda es la realización de los comandos:

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

Sin embargo hace falta haber creado todas las tablas sin ningún problema, el cual no fue mi caso, por lo tanto crearé los controladores, los modelos y las vistas a mano.

Para ello uso el siguiente comando:

- php artisan make:controller ClienteController --resource
- php artisan make:controller AdminController --resource

AdminController

Index

- Recupera todos los clientes de la base de datos
- Usa with('user') para traer también la información de la tabla users.
- Retorna la vista admin.clientes con la lista de clientes.

DestroyCliente

- Busca el cliente por el id
- Guarda en acciones_administrativas que el cliente ha sido eliminado
- Borra el cliente de la tabla clientes
- Borra el usuario asociado en la tabla users

EditCliente

- Busca un cliente por id junto con su usuario (with('user')).

- Si no lo encuentra, devuelve un error 404.
- Retorna la vista admin.edit_cliente con los datos del cliente.

UpdateCliente

- Valida los datos del formulario.
- Busca al cliente en la base de datos.
- Guarda los datos anteriores y los compara con los nuevos.
- Actualiza el usuario en users.
- Actualiza la información en clientes.
- Guarda la acción en acciones_administrativas.

Profile

- Obtiene los datos del usuario autenticado.
- Busca los datos adicionales del administrador en administradores.
- Retorna la vista admin.profile.

UpdateProfile

- Valida los datos.
- Actualiza el usuario en users.
- Sincroniza el nombre y el departamento en administradores.

ClienteController

Profile

Este método es el que se ejecuta cuando un usuario accede a la página de su perfil, mostrando tanto su información personal como la de su cliente asociado (si existe).

UpdateProfile

Este método se encarga de actualizar tanto la información personal del usuario como la del cliente. Si el cliente no existe, solo actualiza al usuario.

UpdatePassword

Este método se utiliza para cambiar la contraseña de un usuario. Primero valida que la contraseña actual sea correcta y luego la actualiza por una nueva.

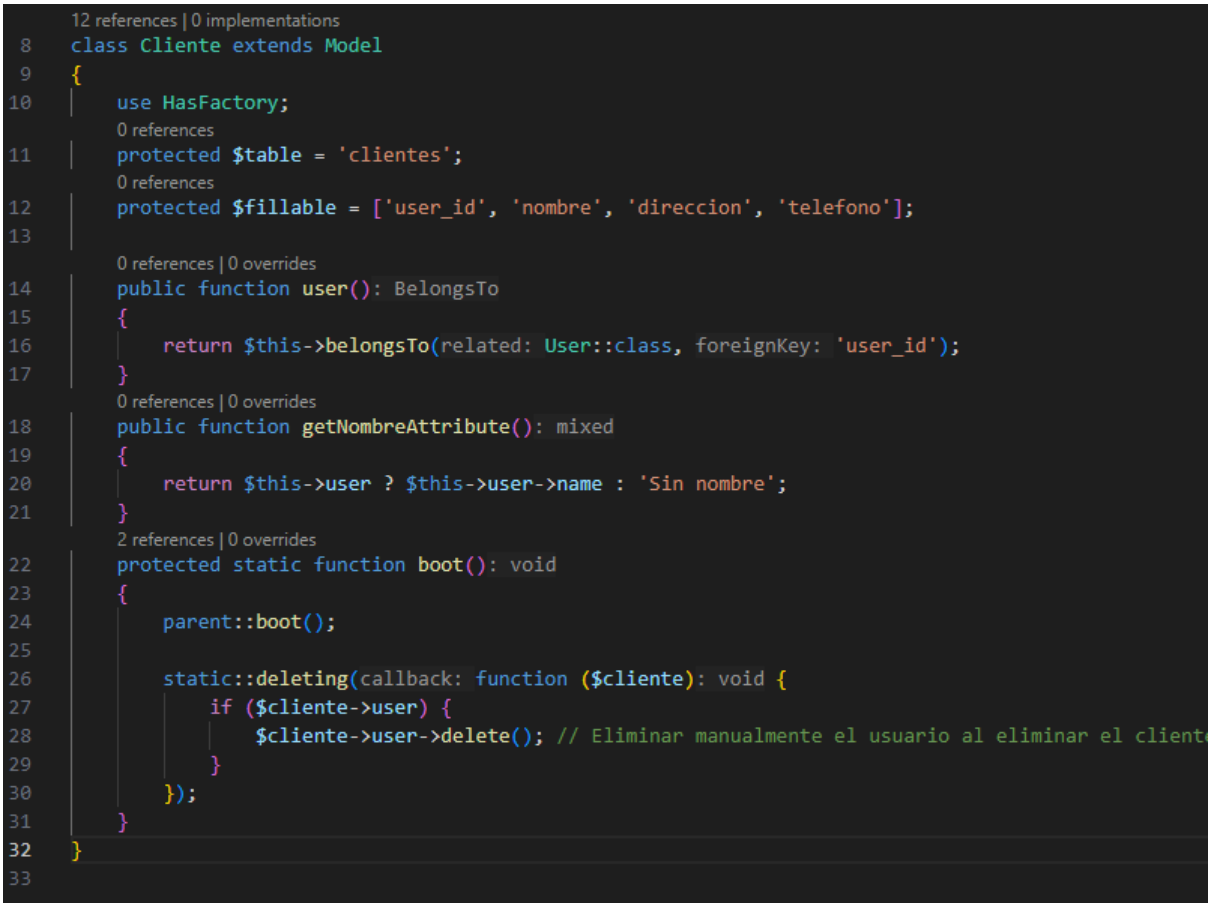
Destroy

Este método elimina al usuario y a su cliente de la base de datos. Se usa cuando un usuario decide eliminar su cuenta.

Creación de los modelos necesarios

Para la creación de los modelos realizamos los siguientes comandos:
php artisan make:model Cliente -m
php artisan make:model Administrador -m

```
6 references | 0 implementations
class Administrador extends Model
{
    use HasFactory;
    0 references
    protected $table = 'administradores';
    0 references
    protected $fillable = ['user_id', 'departamento'];

    0 references | 0 overrides
    public function user(): BelongsTo
    {
        return $this->belongsTo(related: User::class);
    }
    0 references | 0 overrides
    public function getNombreAttribute(): mixed
    {
        return $this->user ? $this->user->name : 'Sin nombre';
    }
    2 references | 0 overrides
    protected static function boot(): void
    {
        parent::boot();

        static::deleting(callback: function ($admin): void {
            if ($admin->user) {
                $admin->user->delete(); // Eliminar manualmente el usuario al eliminar el administ
            }
        });
    }
}
```



```
12 references | 0 implementations
class Cliente extends Model
{
    use HasFactory;
    0 references
    protected $table = 'clientes';
    0 references
    protected $fillable = ['user_id', 'nombre', 'direccion', 'telefono'];

    0 references | 0 overrides
    public function user(): BelongsTo
    {
        return $this->belongsTo(related: User::class, foreignKey: 'user_id');
    }
    0 references | 0 overrides
    public function getNombreAttribute(): mixed
    {
        return $this->user ? $this->user->name : 'Sin nombre';
    }
    2 references | 0 overrides
    protected static function boot(): void
    {
        parent::boot();

        static::deleting(callback: function ($cliente): void {
            if ($cliente->user) {
                $cliente->user->delete(); // Eliminar manualmente el usuario al eliminar el client
            }
        });
    }
}
```

$fillable define los campos que pueden ser modificados.
belongsTo(User::class) indica que clientes y administradores pertenecen a users.

Creación de las Rutas

Las rutas en este archivo están divididas en varias secciones, cada una destinada a diferentes partes de la aplicación y protegidas por el sistema de autenticación de Laravel.

Bienvenida

- Propósito: Esta es la ruta por defecto que se carga al acceder al dominio principal de la aplicación (/). Muestra la vista welcome.
- Uso: Se utiliza para la bienvenida a los usuarios cuando visitan la página inicial.

Autenticación

- Propósito: Genera todas las rutas necesarias para el sistema de autenticación de usuarios (registro, inicio de sesión, recuperación de contraseñas, etc.).
- Uso: Permite a los usuarios registrarse, iniciar sesión, y gestionar sus credenciales.

Clientes

- Propósito: Estas rutas están protegidas por el middleware auth (para garantizar que solo los usuarios autenticados puedan acceder a ellas) y role:cliente (para restringir el acceso a los usuarios con el rol de cliente).

    - /profile: Muestra el perfil del cliente.
    - /profile/update: Permite actualizar el perfil del cliente.
    - /profile/change-password: Permite cambiar la contraseña del cliente.
    - /profile/delete: Permite eliminar la cuenta del cliente.

- Uso: Estas rutas permiten gestionar el perfil de los clientes, incluida la actualización de sus datos y la modificación de la contraseña.

Administradores

- Propósito: Estas rutas están protegidas por el middleware auth (para usuarios autenticados) y role:admin (para usuarios con el rol de administrador). Permiten la gestión de los clientes y el acceso a datos administrativos.
    - /admin/clientes: Muestra todos los clientes registrados.
    - /admin/perfil: Muestra el perfil del administrador.
    - /admin/perfil/update: Permite actualizar el perfil del administrador.
    - /admin/perfil/change-password: Permite cambiar la contraseña del administrador.
    - /admin/clientes/{id}/edit: Muestra un formulario para editar los datos de un cliente específico.
    - /admin/clientes/{id}: Actualiza la información de un cliente específico.
    - /admin/clientes/{id}/destroy: Elimina un cliente específico.
    - /admin/historial: Muestra el historial administrativo de la aplicación.
- Uso: Estas rutas permiten a los administradores gestionar los clientes, editar sus datos, eliminar cuentas y ver el historial de actividades.

Home

- Propósito: Esta ruta es la página de inicio después de que un usuario se autentique.
- Uso: Después de iniciar sesión, el usuario es redirigido a esta página (/home). Generalmente, esta página puede contener un dashboard o página principal de la aplicación.

Futuras Mejoras

- Implementar paginación en la vista de clientes.
- Mejorar el diseño del dashboard para administradores.
- Agregar gráficos de actividad en el panel de administración.