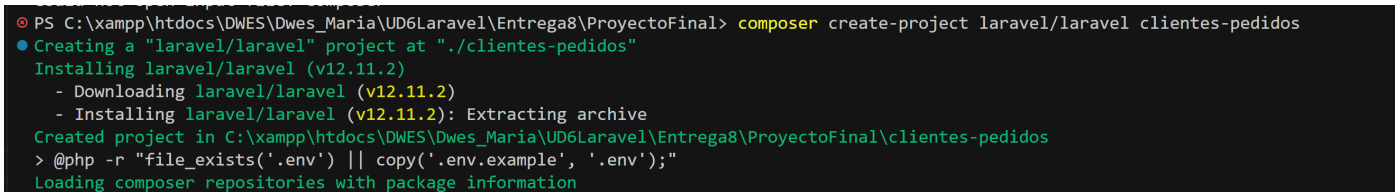
Proyecto cliente-pedido

Para esta parte de Laravel crearemos un proyecto donde gestionaremos clientes con sus respectivos pedidos como si fuesemos una tienda online de x producto. Para ello utilizaremos todo lo visto en clase que son: Blade, Migration, Model, Controller, CRUD, SeederFactory y las relaciones.

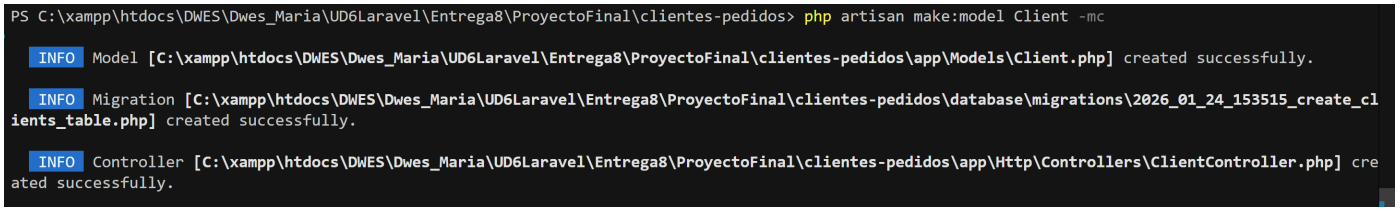
1º Creamos el proyecto

Primero vamos a crear nuestro proyecto en Laravel con el comando composer create-proyect laravel/laravel clientes-pedidos. Esto nos creará un proyecto básico con todo lo ensencial para que funcione.



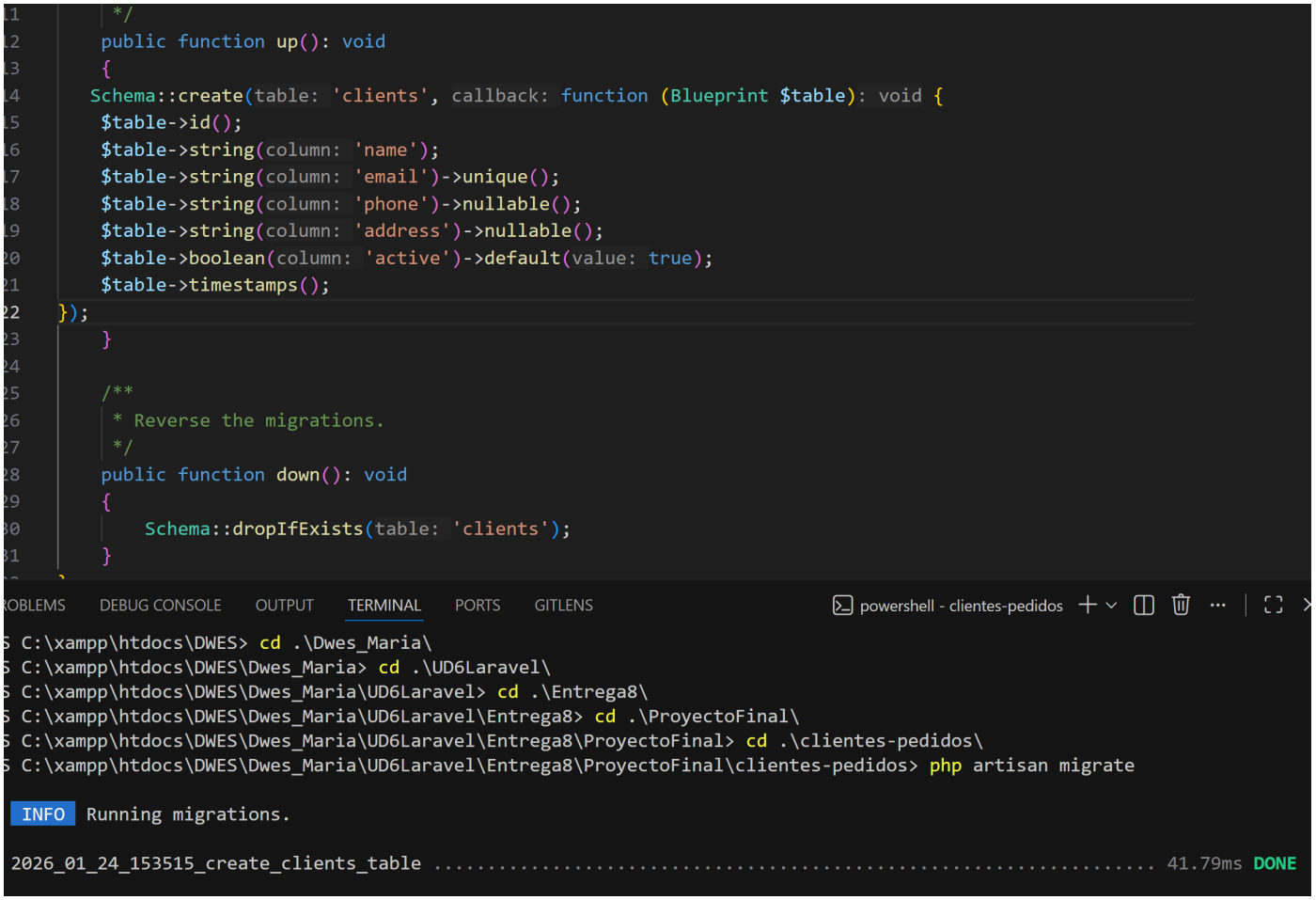
2º Creamos modelo, migración y controller de Cliente

Lo siguiente que haremos es crear el modelo, migración y controlador del cliente, elegimos el cliente ya que sería como el "padre" con respecto a los pedidos. Para ello usaremos el comando php artisan make:model Client -mc, este comando nos crea del tirón todo lo anterior



3º Actualizamos el contenido de la migracion y la ejecutamos

Dentro del archivo de la migración del cliente vamos a modificar los campos que queremos que tenga la tabla de los clientes, como curiosidad, el modelo se creará como Client mientras que la tabla irá en minúscula y en plural (clients). Nuestro cliente tendrá un id con el que se identificará, un email único, un teléfono que puede ser nulo, la dirección que puede ser nula también y un boleano activo que por defecto será true, es decir activo, además òr supuesto se guardará la hora. Por último ejecutamos la migración con el comando php artisan migrate para crear la tabla en la base de datos



4º Creamos el modelo con sus relaciones

Pasamos ahora a crear el modelo, esto nos servirá como bien dice su nombre de modelo para los objetos que vayan a guardarse en la base de datos, todos sus atributos serán protected y fillable lo que quiere decir que podrán ser modificados en la base de datos, el id no se podrá modificar por lo que sería guarded, pero laravel ya lo intuye ya que no está dentro del fillable. También tenemos aquí la relación, en este caso será hasMany con los pedidos. Como punto clave destacar el uso del use HasFactory, lo cual es esencial para que la factoria sepa como tiene que crear los objetos

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

17 references | 0 implementations
class Client extends Model
{
    use HasFactory;

    0 references
    protected $fillable = [
        'name',
        'email',
        'phone',
        'address',
        'active'
    ];

    0 references | 0 overrides
    public function orders(): HasMany
    {
        return $this->hasMany(related: Order::class);
    }
}

```

5º Hacemos los mismo con los pedidos

Usando el mismo comando que en el cliente vamos a hacer los pedidos para crear así su modelo su migración y su controlador

```
PS C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos> php artisan make:model Order -mc

  INFO  Model [C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos\app\Models\Order.php] cr
eated successfully.

  INFO  Migration [C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos\database\migrations\
2026_01_25_113116_create_orders_table.php] created successfully.

  INFO  Controller [C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos\app\Http\Controller
s\OrderController.php] created successfully.
```

6º Creamos la migración

Actualizamos el archivo de migración de los pedidos, como punto más importante aquí destacar el uso del enum, y el atributo client_id, que es el id que pasa de la tabla cliente, es decir de la 1 a la N. Tambien decir que tiene un on delete cascade, en caso de que el id del cliente se borre, todos los pedidos asociados a este id serán eliminados

```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: 'orders', callback: function (Blueprint $table): void {
            $table->id();
            $table->string(column: 'order_number')->unique();
            $table->date(column: 'date');
            $table->enum(column: 'status', allowed: ['pendiente', 'enviado', 'entregado', 'cancelado']);
            $table->decimal(column: 'total', total: 8, places: 2);
            $table->text(column: 'notes')->nullable();

            $table->foreignId(column: 'client_id')
                ->constrained()
                ->onDelete(action: 'cascade');

            $table->timestamps();
        });
    }

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

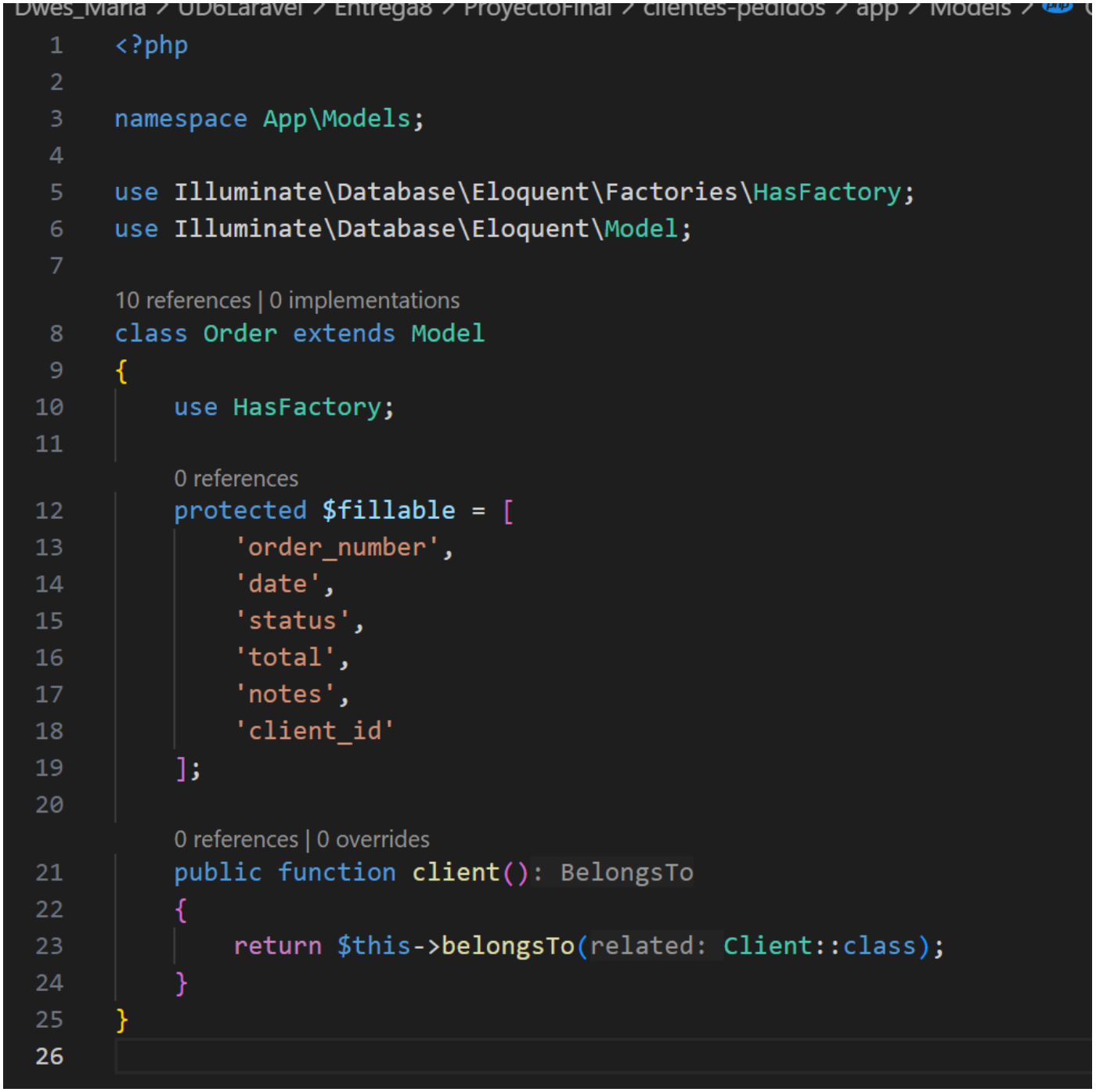
7º Ejecutamos la migración

Ejecutamos la migración para crear la tabla de pedidos

```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS  GITLENS        powershell - clientes-pedidos

  INFO  Model [C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos\app\Models\Order.php] c

  INFO  Controller [C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos\app\Http\Controlle
s\OrderController.php] created successfully.

PS C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos> php artisan migrate

  INFO  Running migrations.

  2026_01_25_113116_create_orders_table ..... 40.52ms DONE
```

8º Completamos el modelo y hacemos la relación

Vamos ahora a rellenar el modelo al igual que el cliente, el pedido hace uso del HasFactory, del fillable y en este caso la relación es belongsTo



9º Actualizamos las rutas

Una vez tenemos ya los modelos y las migraciones necesarias vamos a crear las rutas a través de las cuales navergaremos entre los diferentes CRUD, tenemos el welcome

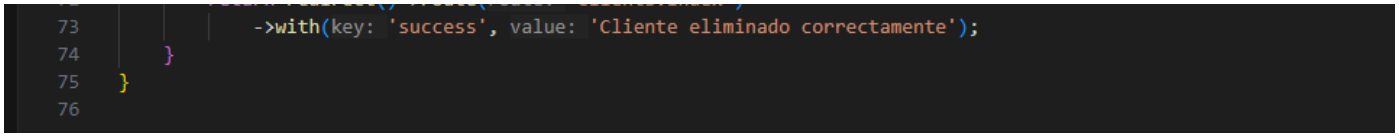
tipico de laravel, el clients y el orders

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ClientController;
use App\Http\Controllers\OrderController;

Route::get(uri: '/', action: function (): View {
    return view(view: 'welcome');
});
Route::resource(name: 'clients', controller: ClientController::class);
Route::resource(name: 'orders', controller: OrderController::class);
```

10 º Creamos clientController

Vamos a pasar ahora con la creación del controller del cliente que contará con con las funciones de index, create, store, show,edit, update y destroy

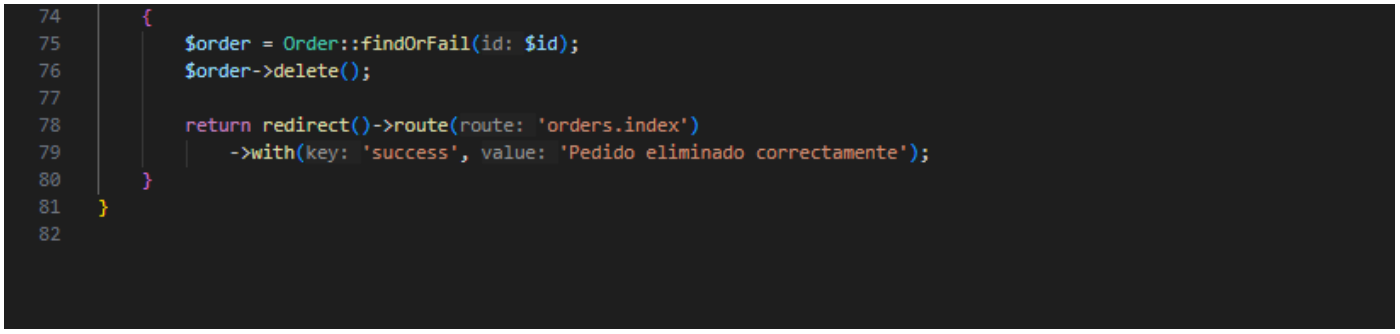
```php
<?php

namespace App\Http\Controllers;

use App\Models\Client;
use Illuminate\Http\Request;

2 references | 0 implementations
class ClientController extends Controller
{
    0 references | 0 overrides
    public function index(): View
    {
        $clients = Client::all();
        return view(view: 'clients.index', data: compact(var_name: 'clients'));
    }

    0 references | 0 overrides
    public function create(): View
    {
        return view(view: 'clients.create');
    }

    0 references | 0 overrides
    public function store(Request $request): RedirectResponse
    {
        $request->validate(rules: [
            'name' => 'required|string|max:255',
            'email' => 'required|email|unique:clients,email',
            'phone' => 'nullable|string|max:20',
            'address' => 'nullable|string|max:255',
            'active' => 'boolean'
        ]);

        Client::create(attributes: $request->all());

        return redirect()->route(route: 'clients.index')
            ->with(key: 'success', value: 'Cliente creado correctamente');
    }

    0 references | 0 overrides
    public function show($id): View
    {
        $client = Client::findOrFail(id: $id);
        return view(view: 'clients.show', data: compact(var_name: 'client'));
    }

    0 references | 0 overrides
    public function edit($id): View
    {
        $client = Client::findOrFail(id: $id);
        return view(view: 'clients.edit', data: compact(var_name: 'client'));
    }

    0 references | 0 overrides
    public function update(Request $request, $id): RedirectResponse
    {
        $client = Client::findOrFail(id: $id);

        $request->validate(rules: [
            'name' => 'required|string|max:255',
            'email' => 'required|email|unique:clients,email,' . $client->id,
            'phone' => 'nullable|string|max:20',
            'address' => 'nullable|string|max:255',
            'active' => 'boolean'
        ]);

        $client->update(attributes: $request->all());

        return redirect()->route(route: 'clients.index')
            ->with(key: 'success', value: 'Cliente actualizado correctamente');
    }

    0 references | 0 overrides
    public function destroy($id): RedirectResponse
    {
        $client = Client::findOrFail(id: $id);
        $client->delete();

        return redirect()->route(route: 'clients.index')
```



11º Creamos orderController

Al igual que acabamos de hacer en ClienteController hacemos en order controller con todas las funciones típicas del CRUD

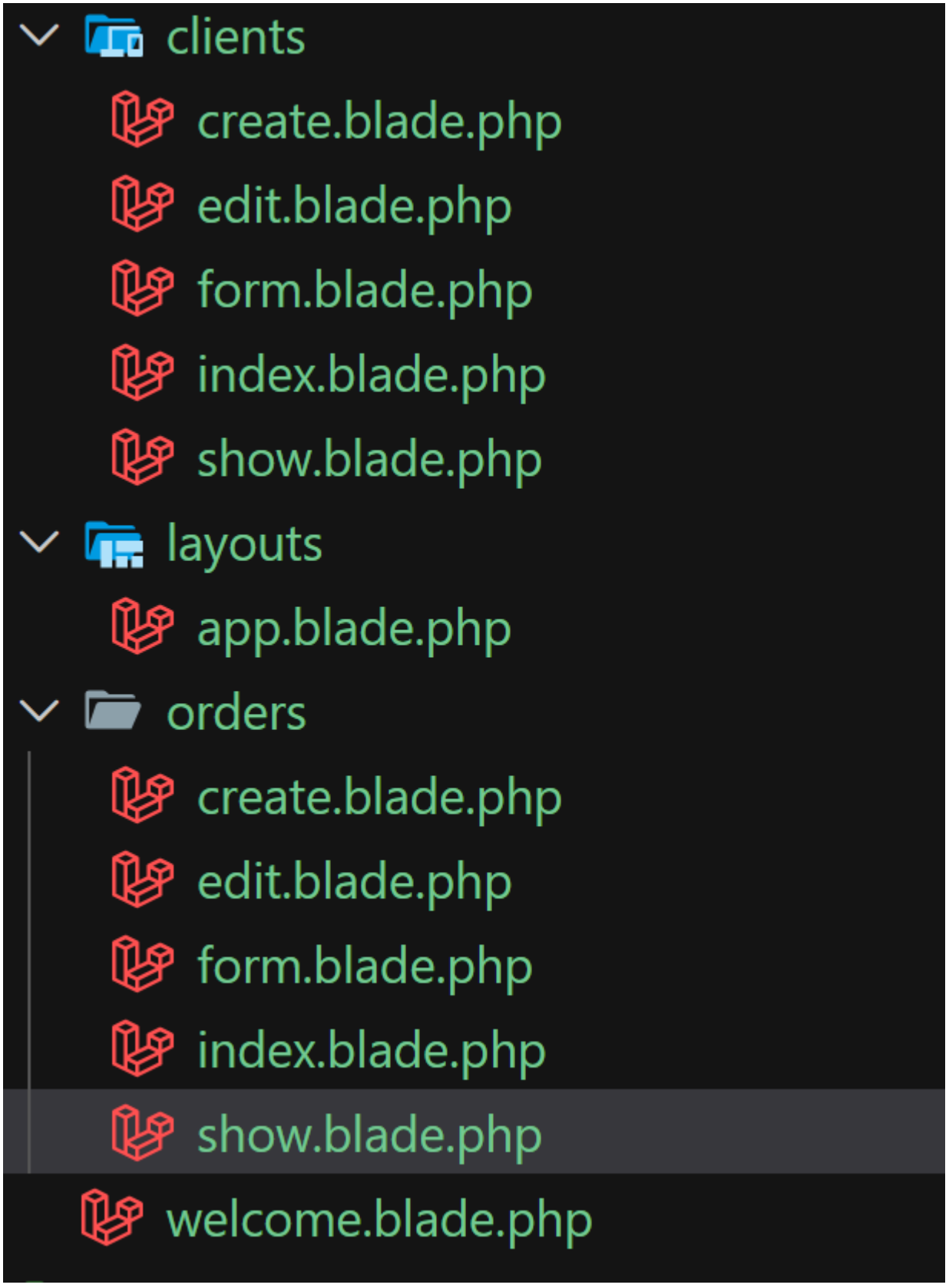
```php
<?php

namespace App\Http\Controllers;

use App\Models\Order;
use App\Models\Client;
use Illuminate\Http\Request;

2 references | 0 implementations
class OrderController extends Controller
{
    0 references | 0 overrides
    public function index(): View
    {
        $orders = Order::with(relations: 'client')->get();
        return view(view: 'orders.index', data: compact(var_name: 'orders'));
    }

    0 references | 0 overrides
    public function create(): View
    {
        $clients = Client::all();
        return view(view: 'orders.create', data: compact(var_name: 'clients'));
    }

    0 references | 0 overrides
    public function store(Request $request): RedirectResponse
    {
        $request->validate(rules: [
            'order_number' => 'required|unique:orders,order_number',
            'date' => 'required|date',
            'status' => 'required|in:pendiente,enviado,entregado,cancelado',
            'total' => 'required|numeric|min:0',
            'notes' => 'nullable|string',
            'client_id' => 'required|exists:clients,id'
        ]);

        Order::create(attributes: $request->all());

        return redirect()->route(route: 'orders.index')
            ->with(key: 'success', value: 'Pedido creado correctamente');
    }

    0 references | 0 overrides
    public function show($id): View
    {
        $order = Order::with(relations: 'client')->findOrFail(id: $id);
        return view(view: 'orders.show', data: compact(var_name: 'order'));
    }

    0 references | 0 overrides
    public function edit($id): View
    {
        $order = Order::findOrFail(id: $id);
        $clients = Client::all();

        return view(view: 'orders.edit', data: compact(var_name: 'order', var_names: 'clients'));
    }

    0 references | 0 overrides
    public function update(Request $request, $id): RedirectResponse
    {
        $order = Order::findOrFail(id: $id);

        $request->validate(rules: [
            'order_number' => 'required|unique:orders,order_number,' . $order->id,
            'date' => 'required|date',
            'status' => 'required|in:pendiente,enviado,entregado,cancelado',
            'total' => 'required|numeric|min:0',
            'notes' => 'nullable|string',
            'client_id' => 'required|exists:clients,id'
        ]);

        $order->update(attributes: $request->all());

        return redirect()->route(route: 'orders.index')
            ->with(key: 'success', value: 'Pedido actualizado correctamente');
    }

    0 references | 0 overrides
    public function destroy($id): RedirectResponse
```



```
    {
        $order = Order::findOrFail(id: $id);
        $order->delete();

        return redirect()->route(route: 'orders.index')
            ->with(key: 'success', value: 'Pedido eliminado correctamente');
    }
}
```

12º Creamos todas las vistas

A continuación una vez creados los controllers deberemos de crear las vistas por las que el usuario irá navegando, habrá una vista principal app.blade y luego tanto clients como orders tendrán las vistas de create, edit, form, index, y show



13º Creamos las factorias de cliente

Pasamos ahora a crear las factorias. Las factorias se usan para poder probar los programas con datos falsos pero en grandes cantidades por ejemplo, en este caso, vamos a crear a 10 clientes donde cada uno de ellos tendrá por defecto 3 pedidos

```
PS C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos> php artisan make:factory ClientFact
ory

  INFO  Factory [C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos\database\factories\Cl
ientFactory.php] created successfully.
```

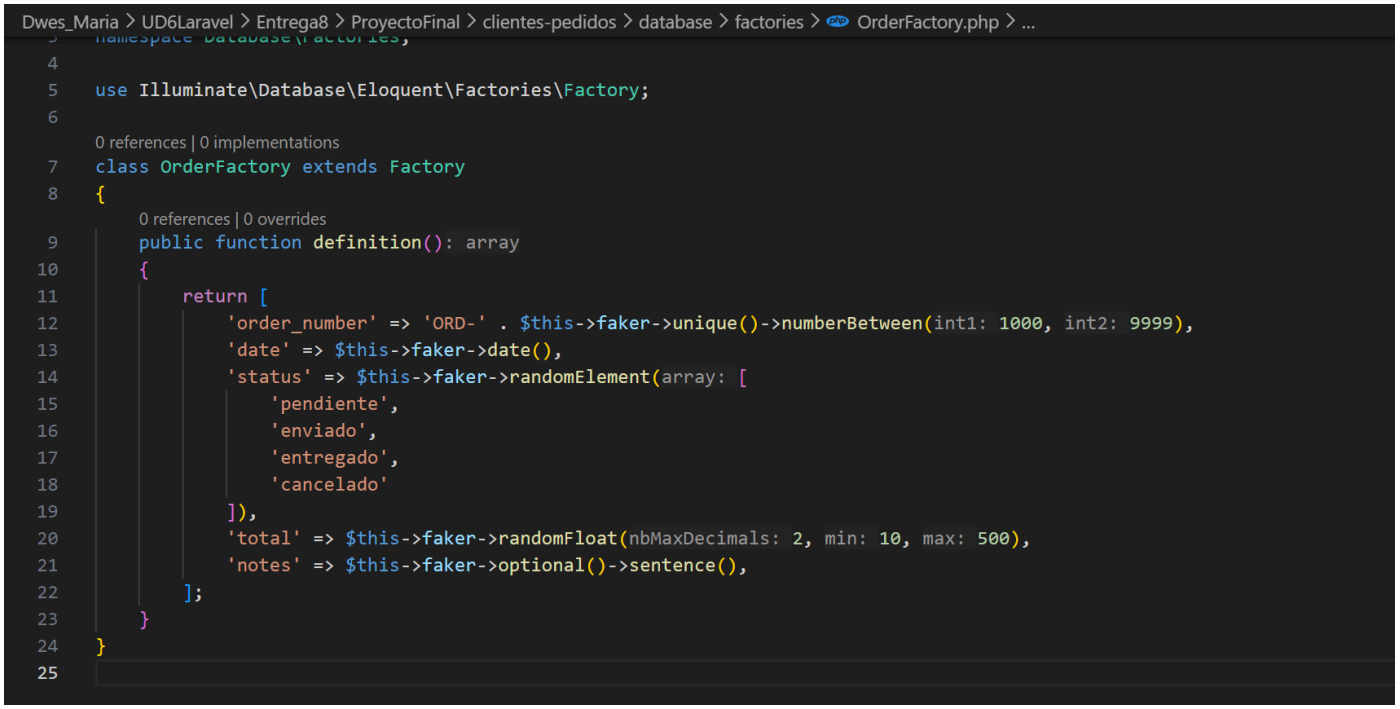
Dwes_Maria > UD6Laravel > Entrega8 > ProyectoFinal > clientes-pedidos > database > factories > ClientFactory.php > ...

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

0 references | 0 implementations
class ClientFactory extends Factory
{
    0 references | 0 overrides
    public function definition(): array
    {
        return [
            'name' => $this->faker->name(),
            'email' => $this->faker->unique()->safeEmail(),
            'phone' => $this->faker->phoneNumber(),
            'address' => $this->faker->address(),
            'active' => $this->faker->boolean(chanceOfGettingTrue: 90),
        ];
    }
}

```

12º Creamos factorias de order

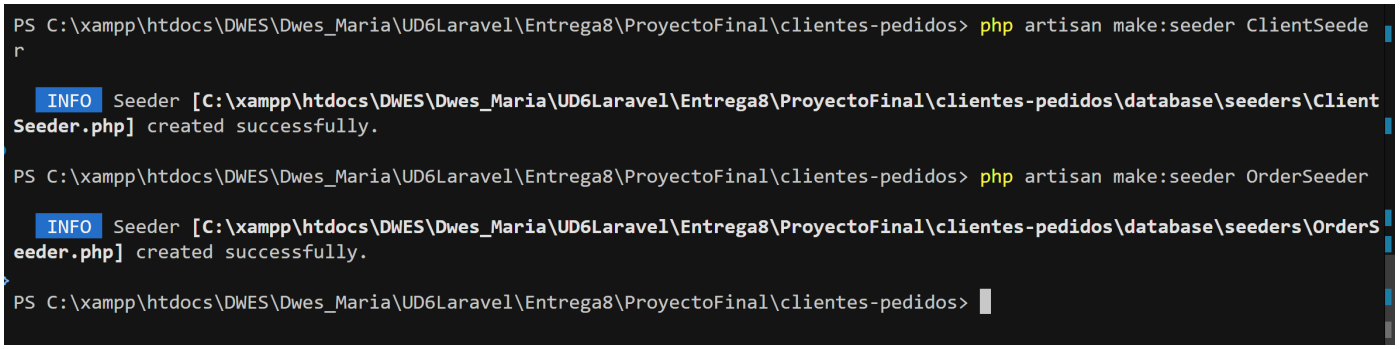
Usamos el random para que la factoria elija aleatoriamente el valor que ella decida entre las opciones que les damos

```
PS C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos> php artisan make:factory OrderFacto
ry

  INFO  Factory [C:\xampp\htdocs\DWES\Dwes_Maria\UD6Laravel\Entrega8\ProyectoFinal\clientes-pedidos\database\factories\Or
derFactory.php] created successfully.
```



13º Creamos las seeders de cliente y de order

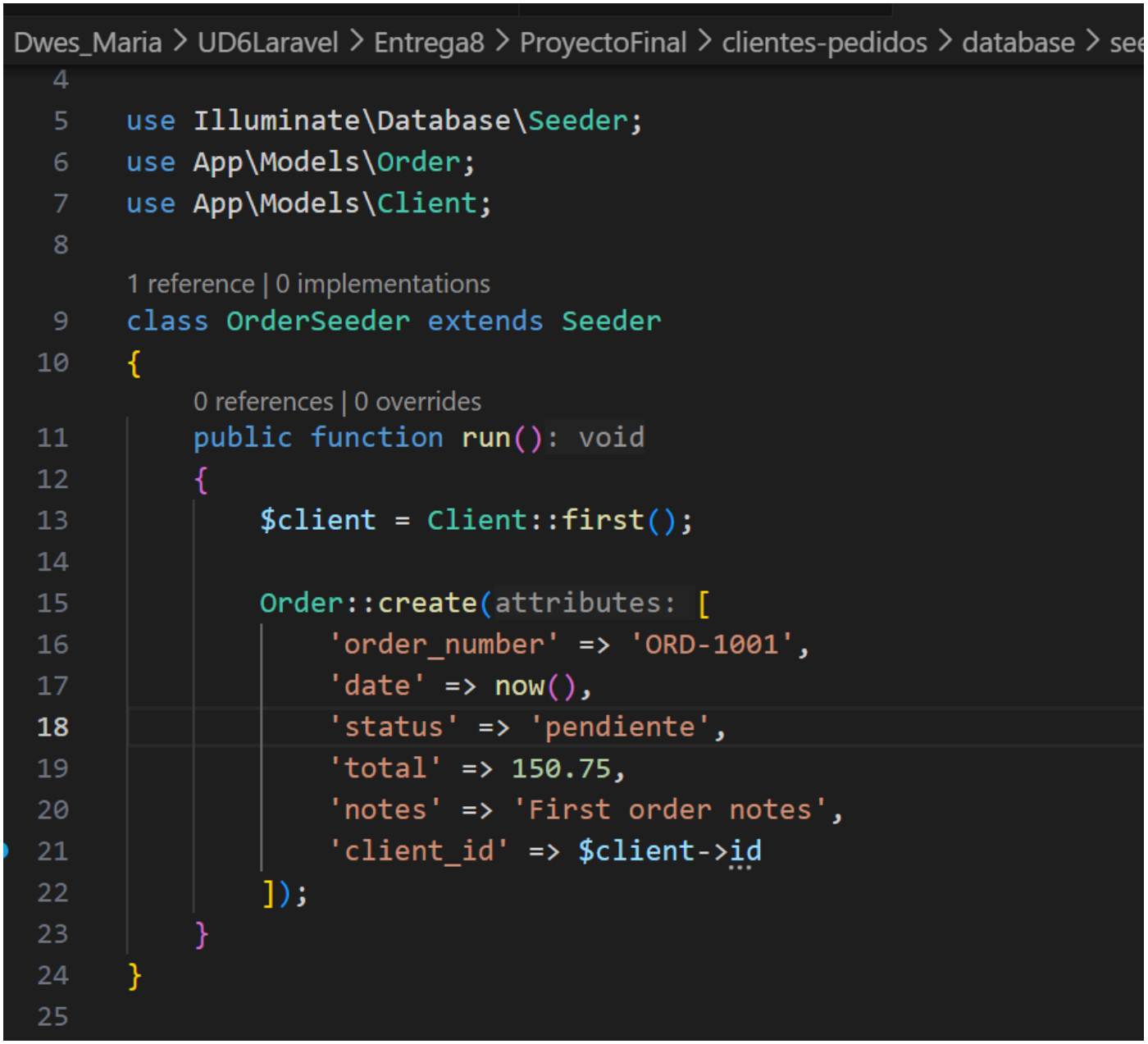
Pasamos ahora a crear las seeders que son como los datos a fuego que va a llevar nuestra tabla en la base de datos tanto clientes como pedidos en este caso uno de cada



README.md ...\clientes-pedidos U | ClientSeeder.php U | OrderSeeder.php U | database.sq

Dwes_Maria > UD6Laravel > Entrega8 > ProyectoFinal > clientes-pedidos > database > seeders > ClientSeeder.ph

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\Client;

1 reference | 0 implementations
class ClientSeeder extends Seeder
{
    0 references | 0 overrides
    public function run(): void
    {
        Client::create(attributes: [
            'name' => 'Juan Pérez',
            'email' => 'juan.perez@example.com',
            'phone' => '123456789',
            'address' => 'Calle Falsa 123, Ciudad',
            'active' => true
        ]);
    }
}

```


Dwes_Maria > UD6Laravel > Entrega8 > ProyectoFinal > clientes-pedidos > database > see
use Illuminate\Database\Seeder;
use App\Models\Order;
use App\Models\Client;
1 reference | 0 implementations
class OrderSeeder extends Seeder
{
0 references | 0 overrides
public function run(): void
{
$client = Client::first();
Order::create(attributes: [
'order_number' => 'ORD-1001',
'date' => now(),
'status' => 'pendiente',
'total' => 150.75,
'notes' => 'First order notes',
'client_id' => $client->id
]);
}
}

```php
<?php

namespace Database\Seeders;

use App\Models\Client;
use Illuminate\Database\Seeder;

0 references | 0 implementations
class DatabaseSeeder extends Seeder
{
    0 references | 0 overrides
    public function run(): void
    {
        $this->call(class: [
            ClientSeeder::class,
            OrderSeeder::class,
        ]);

        Client::factory()
        ->count(10)
        ->hasOrders(3)
        ->create();

    }
}

```