- Cosas aprendidas:
  - Dia 1
    - El atajo "-mcr": Creando todo de un golpe
  - Dia 2
    - hasMany (En el modelo Cliente)
    - belongsTo (En el modelo Pedido)
  - Faker: Datos de prueba
    - Route::resource tiene 7 rutas
      - Las 7 rutas automáticas
- Diario de Trabajo: Dia 1
  - Cración del proyecto
  - Creación del modelo, Migracion y Controladores
  - Creación Factories y Seeders
  - Migraciones (Database)
    - create_clientes_table
    - create_pedidos_table
- Diario de Trabajo: Dia 2
  - Modelos y Relaciones
    - Modelo Cliente
    - Modelo Pedido
  - Factories y Seeders
    - ClienteFactory
    - DatabaseSeeder
  - Rutas
    - Web.php

Cosas aprendidas:

Dia 1

El atajo "-mcr": Creando todo de un golpe

¿Para qué sirve? : Con este añadido al final del comando, Laravel te construye la estructura básica de una sección de tu web de un solo golpe. Te crea tres archivos clave:

- **La M (Migración):** El plano para crear la tabla en la base de datos.
- **La C (Controlador):** El "cerebro" que recibe las peticiones de los usuarios.
- **La R (Recurso):** Hace que ese controlador ya venga con los métodos estándar para ver, crear, editar y borrar (el famoso CRUD) ya escritos, para que no tengas que crearlos tú uno a uno.

Uso real: Se utiliza para **ahorrar tiempo y evitar errores de nombres** .

Dia 2

`hasMany` (En el modelo Cliente)

¿Para qué sirve? : Para que cuando estemos con un cliente, podamos sacar de golpe todos sus pedidos.

Uso real : `$cliente->pedidos` (nos devuelve la lista de sus compras).

`belongsTo` (En el modelo Pedido)

¿Para qué sirve? : Para ponerle nombre y apellidos a cada pedido.

Uso real : `$pedido->cliente` (nos devuelve los datos del cliente que hizo esa compra).

Faker: Datos de prueba

¿Para qué sirve? : Genera datos falsos (nombres, emails) automáticamente para no tener que inventarlos y escribirlos tú a mano. **Uso real** : Rellena la base de datos en un segundo para probar que el diseño de la web funciona bien con mucha información.

Route::resource tiene 7 rutas

¿Para qué sirve? : Crea de golpe las 7 rutas estándar para gestionar algo (ver lista, crear, guardar, ver detalle, editar, actualizar y borrar) con una sola línea. **Uso real** :

Permite tener todas las funciones de "crear, leer, editar y borrar" (CRUD) listas y ordenadas sin tener que escribir cada ruta por separado.

Concretamente son estas 7 usando la entidad cliente:

Las 7 rutas automáticas

Dirección (URL)	Método	Qué hace (en sencillo)
`/clientes`	GET	Muestra la lista de todos los clientes.
`/clientes/create`	GET	Abre el formulario para escribir los datos del nuevo cliente.
`/clientes`	POST	Es la dirección interna donde se mandan los datos para guardarlos.
`/clientes/{id}`	GET	Muestra la ficha de un solo cliente (ej:`/clientes/5`).
`/clientes/{id}/edit`	GET	Abre el formulario con los datos ya rellenos para cambiarlos.
`/clientes/{id}`	PUT	Es la dirección que procesa los cambios que has editado.
`/clientes/{id}`	DELETE	La dirección que se encarga de borrar a ese cliente.

Diario de Trabajo: Dia 1

Cración del proyecto

Primero creamos el proyecto y entramos a el con los comandos:

```
composer create-project laravel/laravel manga-store
cd manga-store
```

Creación del modelo, Migracion y Controladores

Luego creamos el modelo, migraciones y controladores.

```
php artisan make:model Cliente -mcr
php artisan make:model Pedido -mcr
```

> Nota:
>
> `-mcr` crea el Modelo, La Migración y el Controlador con los métodos CRUD ya definidos

Creación Factories y Seeders

```
php artisan make:factory ClienteFactory
php artisan make:factory PedidoFactory
php artisan make:seeder ClienteSeeder
```

Migraciones (Database)

create_clientes_table

Esta tabla almacena la información básica de los usuarios que realizarán compras en la tienda.

- `id()` : Crea un campo autoincremental como clave primaria.
- `nombre` y `telefono` : Campos de texto estándar para identificación y contacto.
- `email` : Se marca como `unique()` para evitar que dos clientes se registren con el mismo correo.
- `direccion` : Se usa `text` en lugar de `string` para permitir descripciones largas de domicilio.

- `activo` : Un booleano que permite "desactivar" clientes sin borrar sus datos (borrado lógico).
- `timestamps()` : Crea automáticamente las columnas `created_at` y `updated_at`.

```
public function up(): void {
    Schema::create('clientes', function (Blueprint $table) {
        $table->id();
        $table->string('nombre');
        $table->string('email')->unique();
        $table->string('telefono');
        $table->text('direccion');
        $table->boolean('activo')->default(true);
        $table->timestamps();
    });
}
```

create_pedidos_table

Esta tabla gestiona las transacciones de la tienda y vincula cada compra con un cliente específico.

- `numero_pedido` : Un identificador único para seguimiento comercial (diferente al ID interno).
- `fecha` : Registra el momento exacto de la venta.
- `estado` : Utiliza un `enum`, lo que restringe los valores posibles a solo cuatro opciones específicas, garantizando la integridad de los datos.
- `total` : Definido como `decimal(8, 2)` para manejar dinero con precisión (evitando los errores de redondeo de los tipos *float*).
- `foreignId('cliente_id')` : Es la pieza clave de la relación. Conecta el pedido con un ID de la tabla `clientes`.
- `constrained()` : Asegura que el cliente realmente exista.
- `onDelete('cascade')` : Si un cliente es eliminado de la base de datos, todos sus pedidos se borrarán automáticamente para no dejar datos huérfanos.

```
public function up(): void {
    Schema::create('pedidos', function (Blueprint $table) {
        $table->id();
        $table->string('numero_pedido')->unique(); // Ej: MANGA-1001
        $table->date('fecha');
        $table->enum('estado', ['pendiente', 'enviado', 'entregado', 'cancelado'])-
>default('pendiente');
        $table->decimal('total', 8, 2);
```

```
            $table->text('notas')->nullable();
            // Relación: Un pedido pertenece a un cliente
            $table->foreignId('cliente_id')->constrained()->onDelete('cascade');
            $table->timestamps();
        });
    }
```

Diario de Trabajo: Dia 2

Modelos y Relaciones

En esta parte definimos cómo se comportan nuestros datos y cómo se conectan entre ellos.

Modelo Cliente

```
class Cliente extends Model {
    protected $fillable = ['nombre', 'email', 'telefono', 'direccion', 'activo'];

    // Un cliente tiene muchos pedidos
    public function pedidos() {
        return $this->hasMany(Pedido::class);
    }
}
```

- `$fillable` : Es una medida de seguridad. Aquí ponemos los campos que permitimos que se rellenen.
- **Relación** `pedidos()` : Aquí le decimos a Laravel que un cliente puede tener muchos pedidos. Gracias a esto, luego podremos sacar los pedidos de alguien con un simple `$cliente->pedidos`.

Modelo Pedido

```
class Pedido extends Model {
    protected $fillable = ['numero_pedido', 'fecha', 'estado', 'total', 'notas',
'cliente_id'];

    // Un pedido pertenece a un cliente
```

```
    public function cliente() {
        return $this->belongsTo(Cliente::class);
    }
}
```

Relación `cliente()` : Es la inversa de la anterior. Cada pedido pertenece a un único cliente. Esto nos permite saber quién hizo la compra fácilmente.

Factories y Seeders

Para no tener que estar metiendo datos a mano cada vez que probamos la web, usamos estas herramientas para generar "relleno" automático.

ClienteFactory

```
public function definition(): array {
    return [
        'nombre' => $this->faker->name(),
        'email' => $this->faker->unique()->safeEmail(),
        'telefono' => $this->faker->phoneNumber(),
        'direccion' => $this->faker->address(),
        'activo' => true,
    ];
}
```

Usamos la librería **Faker** , que es como un generador de identidades falsas. Nos crea nombres, emails, teléfonos y direcciones que parecen reales pero son aleatorios. El campo `activo` lo dejamos en `true` por defecto.

DatabaseSeeder

```
public function run(): void {
    Cliente::factory(10)->create()->each(function ($cliente) {
        Pedido::factory(3)->create([
            'cliente_id' => $cliente->id,
        ]);
    });
}
```

Este es el "botón de encendido" de nuestros datos de prueba. Lo que hace el código es:

1. Crear 10 clientes ficticios.
2. Para cada uno de esos clientes, crear 3 pedidos automáticamente. Así, nada más empezar, tenemos una base de datos con 30 pedidos listos para mostrar.

Rutas

Web.php

```
use App\Http\Controllers\ClienteController;
use App\Http\Controllers\PedidoController;

Route::resource('clientes', ClienteController::class);
Route::resource('pedidos', PedidoController::class);
```

`Route::resource`: Esta línea es un "atajo" que crea automáticamente las 7 rutas típicas de un CRUD (Index, Create, Store, Show, Edit, Update, Destroy). Es mucho más limpio y sigue el estándar de Laravel. "# Proyecto-Laravel"