UD 3.1

Introducción a Eloquent

¿Qué es Eloquent?

Eloquent es el **ORM (Object-Relational Mapping)** incluido en Laravel. Permite interactuar con la base de datos utilizando modelos PHP en lugar de escribir consultas SQL directamente.

- **Definición:** Un ORM es una herramienta que mapea tablas de una base de datos relacional a objetos en el código, simplificando la manipulación de datos.
  - Tablas → Clases.
  - Filas → Instancias de clases.
  - Columnas → Atributos de las clases.
- **Propósito de Eloquent:** Facilitar el acceso y la manipulación de datos en bases de datos relacionales de manera intuitiva y eficiente, integrándose perfectamente con la arquitectura de Laravel.

Ejemplo: Consultar todos los usuarios activos

- **SQL Tradicional:**

```
SELECT * FROM users WHERE active = 1;
```

- **Eloquent:**

```
$usuarios = User::where('active', 1)->get();
```

Ventajas de Eloquent

1. **Productividad:**
   - Reduce la cantidad de código necesario para realizar operaciones CRUD.
2. **Legibilidad:**
   - El código es más fácil de entender gracias a su estructura clara.
3. **Mantenibilidad:**
   - Es fácil extender o modificar funcionalidades.
4. **Relaciones predefinidas:**
   - Maneja relaciones complejas como One-to-Many o Many-to-Many de forma nativa.
5. **Protección contra inyección SQL:**
   - Eloquent usa consultas preparadas, lo que mejora la seguridad.

Configuración inicial

Verificar conexión con la base de datos

1. **Configurar** `.env` **:** Asegúrate de que las credenciales de la base de datos estén correctamente configuradas:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=nombre_base
DB_USERNAME=usuario
DB_PASSWORD=contraseña
```

2. **Probar la conexión:** Usa el siguiente comando para verificar que Laravel puede conectarse a la base de datos:

```
php artisan migrate
```

   Si hay un error, revisa el archivo `.env` y la configuración del servidor.

Migraciones en Laravel

Las **migraciones** en Laravel son una herramienta que permite crear y gestionar la estructura de las tablas en la base de datos de forma programática y controlada. Facilitan trabajar en equipo y mantener un historial de cambios en la base de datos.

¿Qué es una migración?

- Una migración es un archivo de clase PHP que define la estructura de una tabla.
- Se utiliza para:
  - Crear nuevas tablas.
  - Modificar tablas existentes.
  - Eliminar tablas.
- Su objetivo principal es mantener la consistencia de la base de datos entre diferentes entornos.

Comandos básicos

1. **Crear una nueva migración:**

   `php artisan make:migration nombre_migracion`

   Esto generará un archivo en `database/migrations` con una estructura similar a:

   `public function up() { // Crear tabla }`

   `public function down() { // Revertir cambios }`

2. **Ejecutar las migraciones:**

   `php artisan migrate`

   Esto aplica todos los cambios pendientes a la base de datos.

3. **Revertir migraciones:**

   `php artisan migrate:rollback`

4. **Volver a ejecutar todas las migraciones:**

   `php artisan migrate:refresh`

Estructura de una migración

Cuando creas una migración, tienes dos métodos principales:

1. Método `up`

Define los cambios que quieres aplicar a la base de datos (crear o modificar tablas).

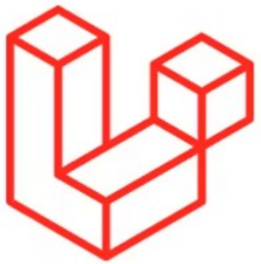
2. Método `down`

Define cómo revertir los cambios realizados en el método `up`.

Tipos de columnas en migraciones

Laravel ofrece una variedad de métodos para definir las columnas de una tabla. Algunos de los más comunes son:

- `$table->id()` Llave primaria autoincremental.
- `$table->string()` Cadena de texto (máx. 255 caracteres).
- `$table->text()` Texto largo.
- `$table->integer()` Número entero.
- `$table->boolean()` Valor verdadero o falso.
- `$table->date()` Fecha.
- `$table->timestamps()` Crea los campos created_at y updated_at



Laravel

Eloquent

Modificaciones a tablas existentes

1. **Crear una migración para modificar una tabla:**

   `php artisan make:migration add_column_to_posts_table`

2. **Definir los cambios en el método** `up` **:**

```
public function up() {
    Schema::table('posts', function (Blueprint $table) {
        $table->string('slug')->after('titulo'); }); }
```

Llaves primarias, foráneas e índices

1. **Llave primaria:**

   ```
   $table->id();
   ```

2. **Llave foránea:**

   ```
   $table->unsignedBigInteger('user_id');

   $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
   ```

3. **Índices:**

   ```
   $table->unique('email');
   $table->index('titulo');
   ```

Seeders en Laravel

Los **seeders** en Laravel se utilizan para poblar las tablas de la base de datos con datos iniciales o de prueba. Son especialmente útiles durante el desarrollo para probar funcionalidades sin necesidad de ingresar datos manualmente.

¿Qué es un seeder?

Un seeder es una clase PHP que define un conjunto de datos que se insertarán en las tablas de la base de datos. Se crean y gestionan mediante comandos Artisan.

Crear un seeder

Usa el comando Artisan para generar un seeder:

```
php artisan make:seeder UserSeeder
```

Esto generará un archivo en `database/seeders/UserSeeder.php`.

Definir el seeder

Define los datos que deseas insertar en la tabla:

```
class UserSeeder extends Seeder {
 public function run() {
   DB::table('users')->insert([
      'name' => 'Juan Pérez',
      'email' => 'juan.perez@example.com',
      'password' => bcrypt('password123'),
   ]);
}
```

3. Ejecutar un seeder

Para ejecutar un seeder específico:

```
php artisan db:seed --class=UserSeeder
```

Para ejecutar todos los seeders registrados en el archivo `DatabaseSeeder.php` :

```
php artisan db:seed
```

Seeder Principal: `DatabaseSeeder`

El archivo `DatabaseSeeder.php` es el punto central donde se pueden registrar y ejecutar múltiples seeders. Está ubicado en `database/seeders/DatabaseSeeder.php`.

Ejemplo de registro de seeders:

```
class DatabaseSeeder extends Seeder {
  public function run() {
    $this->call([
      UserSeeder::class,
      PostSeeder::class,
  ]);
}
```

Uso de Faker para datos dinámicos

Laravel incluye la librería **Faker** para generar datos aleatorios.

```
namespace Database\Seeders;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Str;

class UserSeeder extends Seeder {
  public function run() {
      DB::table('users')->insert([
        'name' => fake()->name(),
        'email' => fake()->unique()->safeEmail(),
       'password' => bcrypt('password123'),
  ]);
 }
}
```

`fake()` es un método integrado que proporciona diferentes tipos de datos, como nombres, correos electrónicos, direcciones, etc.