

¿Qué es un Modelo en Laravel?

En Laravel, un modelo es una **clase PHP que representa una tabla de tu base de datos** y actúa como un puente entre tu aplicación y los datos almacenados. Gracias a **Eloquent ORM** (Object-Relational Mapping), cada tabla tiene un modelo correspondiente que te permite no solo consultar registros, sino también crearlos, actualizarlos y eliminarlos usando objetos PHP en lugar de escribir SQL manualmente.

Convenciones Clave de Eloquent

Eloquent sigue convenciones de nombres muy intuitivas que te ahorran configuración:

- **Ubicación:** Los modelos se guardan por defecto en el directorio `app/Models`.
- **Nombre del Modelo:** Se usa el **singular** y en **PascalCase** (ej. `User`, `Flight`).
- **Nombre de la Tabla:** Eloquent asume que la tabla es el **plural en snake_case** del nombre del modelo. Por ejemplo, el modelo `User` se conecta automáticamente a la tabla `users`; el modelo `Flight` a `flights`, y así sucesivamente.

Creación de un Modelo

Para crear un modelo, utilizas el comando de Artisan `make:model`:

```
bash
```

```
php artisan make:model Flight
```

Este comando generará un archivo como `app/Models/Flight.php` con una estructura básica que extiende la clase `Model` de Eloquent.

Creación con archivos adicionales (Migración, Controlador, etc.)

El comando `make:model` es muy potente y permite generar, de una sola vez, otros archivos relacionados usando banderas opcionales. Algunas de las más útiles son:

- `-m` o `--migration`: Crea también el archivo de migración para la tabla.
- `-c` o `--controller`: Crea un controlador.
- `-f` o `--factory`: Crea una factory para generar datos de prueba.
- `-s` o `--seed`: Crea un seeder.
- `-a` o `--all`: Crea el modelo junto con su migración, factory, seeder, controlador y políticas (policy).

Por ejemplo, para crear el modelo, la migración y el controlador de una vez:

```
bash
```

```
php artisan make:model Flight -m -c
```

Configuración Básica del Modelo

Al abrir tu modelo recién creado, verás una clase muy simple. A partir de ahí, puedes configurar aspectos clave:

La propiedad \$fillable (Asignación Masiva)

Esta propiedad es una **lista blanca** de los atributos que se pueden asignar de forma masiva (por ejemplo, al crear un registro con `Flight::create([...])`). Es una medida de seguridad crucial para prevenir vulnerabilidades de asignación masiva.

php

```
namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Model;
```

```
class Flight extends Model
```

```
{
```

```
    protected $fillable = ['name', 'airline', 'destination'];
```

```
}
```

Otras Propiedades Útiles

- `$hidden`: Una lista de atributos que deseas **ocultar** cuando el modelo se convierta a un array o JSON (por ejemplo, contraseñas).
- `$casts`: Permite **convertir** automáticamente el tipo de un atributo (ej. de integer a boolean, o de string a datetime) al acceder a él.



Operaciones Básicas (CRUD)

Con tu modelo listo, puedes empezar a interactuar con la base de datos de forma expresiva.

1. Crear (Create)

php

```
// Usando asignación masiva (requiere $fillable)
```

```
$flight = Flight::create([
```

```
    'name' => 'Vuelo 123',
```

```
    'airline' => 'Aerolíneas Ejemplo',
```

```
]);
```

```
// O instanciando y guardando
```

```
$flight = new Flight;  
$flight->name = 'Vuelo 456';  
$flight->save();
```

2. Leer (Read)

php

// Obtener todos los registros

```
$flights = Flight::all();
```

// Buscar por clave primaria

```
$flight = Flight::find(1);
```

// Usar condiciones (query builder)

```
$activeFlights = Flight::where('active', 1)  
    ->orderBy('name')  
    ->take(10)  
    ->get();
```

3. Actualizar (Update)

php

```
$flight = Flight::find(1);  
$flight->name = 'Nuevo Nombre de Vuelo';  
$flight->save();
```

// O usando actualización masiva

```
Flight::where('active', 1)  
    ->update(['delayed' => 1]);
```

4. Eliminar (Delete)

php

```
$flight = Flight::find(1);  
$flight->delete();
```

```
// O eliminar por clave primaria
```

```
Flight::destroy(1);
```

Relaciones entre Modelos

Una de las mayores fortalezas de Eloquent son las relaciones. Puedes definir métodos en tus modelos para representar relaciones como **uno a muchos**, **pertenece a**, **muchos a muchos**, etc.

Por ejemplo, si un User (Usuario) puede tener muchos Post (Publicaciones), definirías la relación en el modelo User:

```
php
```

```
// En el modelo User.php
```

```
public function posts()
{
    return $this->hasMany(Post::class);
}
```

Y en el modelo Post, la relación inversa:

```
php
```

```
// En el modelo Post.php
```

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

Luego podrías acceder a los datos relacionados fácilmente:

```
php
```

```
$user = User::find(1);
foreach ($user->posts as $post) {
    echo $post->title;
}
```