PHP: modelli e database in Laravel

Nel pattern MVC i modelli rappresentano il punto di congiunzione tra il database e Laravel. In questo articolo vedremo come effettuare tale collegamento.

Il primo passo è quello di inserire nel file `.env` i dettagli di connessione al database MySQL.

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=nome_database
DB_USERNAME=username
DB_PASSWORD=password
```

Soffermiamoci ora sulle due possibili implementazioni dei model nel pattern MVC.

La prima, che potremo definire purista, prevede che un modello sia una classe che rappresenta con le sue proprietà una tabella del database senza però fornire alcun metodo per operare su di essa.

Quindi se la tabella ha un campo `title`, la classe avrà a sua volta una proprietà `Model::title` e potrà al limite fornire dei metodi getters e setters senza però dare la possibilità di effettuare operazioni CRUD (Create, Read, Update, Delete) sulla tabella sottostante.

All'implementazione purista si contrappone quella ibrida, seguita invece da Laravel. In questa implementazione la classe model fornisce anche metodi per effettuare operazioni CRUD sulla tabella sottostante.

Si tratta di un approccio che tiene conto dell'aspetto dinamico di un'applicazione web in contrapposizione al precedente approccio

sicuramente più adatto al caso di applicazioni desktop o mobile native.

In Laravel possiamo creare un modello digitando nella shell il seguente comando usando la directory principale dell'applicazione.

```
php artisan make:model Product
```

Questo comando creerà la classe `Product` che estende la classe base `Model` di Laravel. In questa fase la classe creata è vuota, ossia è presente solo la sua struttura base.

Adesso possiamo definire le proprietà di questa classe modello come segue:

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    protected $table = 'products';
    protected $primaryKey = 'id';
    public $incrementing = true;
    protected $fillable = ['title', 'description',
'manufacturer', 'price', 'image', 'slug'];
    public $timestamps = false;
}
```

`$table` collega il nostro modello alla tabella MySQL dei prodotti. `$primaryKey` rappresenta la chiave primaria della nostra tabella e viene usata da Laravel per ottimizzare le query al database. `$incrementing` è un flag booleano che indica se la chiave primaria usa la parola chiave MySQL `AUTO_INCREMENT` nella struttura della tabella.

`$fillable` indica quali campi della tabella devono essere presi in considerazione da Laravel durante le operazioni CRUD. Infine, `$timestamps` indica se la tabella presenta i campi predefiniti di Laravel `created_at` e `updated_at` che vengono usati come riferimento cronologico nelle operazioni CRUD.

A questo punto il nostro modello può essere usato all'interno dell'applicazione per le operazioni più comuni sui dati.

Ad esempio se volessimo reperire gli ultimi 10 prodotti ordinati per prezzo in modo ascendente e abilitando la paginazione, possiamo semplicemente scrivere:

```
$products = Product::orderBy('price', 'asc')-
>paginate(10);
```

Si può notare come Laravel riesca con una sola riga di codice a riassumere la logica di una query MySQL con paginazione che avrebbe invece richiesto svariate righe di codice usando un approccio tradizionale.

Se si legge attentamente la documentazione di Laravel, si noterà come le query al database vengono costruite usando la concatenazione dei metodi. Laravel adotta un approccio flessibile, nel senso che si possono usare sia i modelli sia la classe `DB` per effettuare operazioni sul database. Il design delle query, tuttavia, resta il medesimo in entrambi gli approcci.

Una query restituisce un oggetto o un array di oggetti. I campi della tabella diventano quindi proprietà pubbliche dell'oggetto restituito.