L'ORM Eloquent

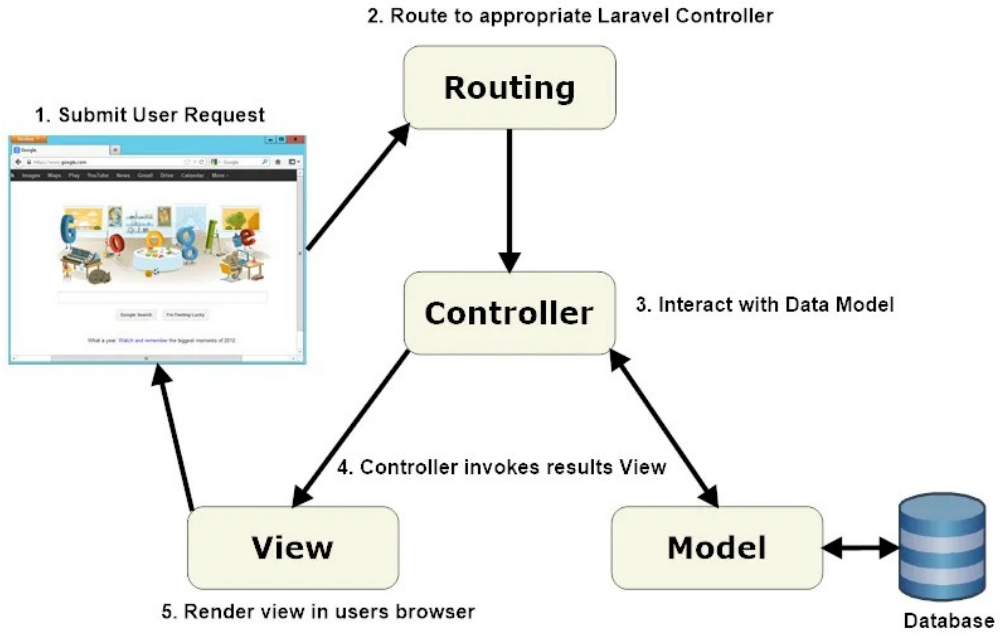
Introduction

L'application Web Laravel pourrait communiquer directement avec les tables de la base de données. Cependant, il sera plus intéressant d'utiliser des **modèles**, c'est-à-dire une représentation objet de chacune des tables.

Lorsqu'on utilise une telle couche, placée entre le code et la base de données, on dira qu'on fait du mapping objet-relationnel, souvent référé comme ***ORM (object-relational mapping).*** L'**ORM** livré avec Laravel s'appelle **Eloquent.**

Génération du modèle

Nous avons vu que Laravel utilise une architecture en **MVC** (***Model View Controller).*** Dans cette architecture c'est le Model qui est l'intermédiaire entre le Contrôleur et la base de données.



Une fois la base de données est modélisée, On doit créer des modèles pour nos tables : **A chaque table dans la BD doit correspondre un Model qui sera utilisé pour interagir avec la table.** Le fichier dans lequel le modèle est défini sera généré à l'aide de la commande : php artisan make:model NomModel

N.B :Pour suivre les standards de Laravel, le nom du modèle doit débuter par une majuscule et être au singulier. Exemple : php artisan make:model Stagiaire

La commande précédente créera un fichier nommé **Stagiaire.php** placé directement dans le dossier **app**, sous le dossier de votre projet. Voici le contenu initial du modèle :

```
namespace App;
use Illuminate\Database\Eloquent\Model;
class Stagiaire extends Model
{
    //
}
```

Le modèle **Stagiaire** est automatiquement lié à la table **stagiaires**. Laravel fait ce lien si le nom de la table est exactement le nom du modèle, tout en lettre minuscules avec en plus un **s** final.

Les conventions d'Eloquent pour les Models

- **Nom de la table**

Par convention le nom du Model est le singulier du nom de la table qui lui correspond dans la BD. Si le nom de la table est différent, il faut le spécifier dans le model : `protected $table = 'my_clients';`

- **Clé primaire**

Eloquent suppose également que chaque table possède une colonne de clé primaire appelée **id**. Vous pouvez définir une propriété $primaryKey pour remplacer cette convention : `protected $primaryKey = 'ncin' ;`

Eloquent suppose que la clé primaire est une valeur entière incrémentée, ce qui signifie que, par défaut, la clé primaire sera automatiquement convertie en int.

Si vous souhaitez utiliser une clé primaire non incrémentée ou non numérique, vous devez définir la propriété publique $incrementing sur votre modèle sur **false.**

```
public $incrementing = false ;

protected $keyType = 'string';
```

- **Timestamps**

Si vous ne souhaitez pas que les colonnes created_at et updated_at soient gérées automatiquement par Eloquent, définissez la propriété $timestamps sur votre modèle sur **false:** `public $timestamps = false;`

Opérations CRUD

Lorsqu'un modèle est créé et associé à une table de base de données, vous êtes prêt à commencer à récupérer les données de la base de données. Chaque modèle Eloquent peut être considéré comme un puissant générateur de requêtes qui vous permet d'interroger de manière fluide la table de base de données associée au modèle.

1- Pour commencer, on génére une migration et un contrôleur de type ressource avec le modèle : php artisan make:model Stagiaire **-mcr**

2- Création de la route : on utilise une route resource

```
Route::resource('stagiaires',StagiaireController::class);
```

Cette déclaration de route unique crée plusieurs routes pour gérer diverses actions sur la ressource :

Méthode	URI	Action	Nom de la route
get	/stagiaires	index	stagiaires.index
get	/stagiaires/create	create	stagiaires.create
post	/stagiaires	store	stagiaires.store
Get	/stagiaires/{s}	show	stagiaires.show
Get	/stagiaires/{s}/edit	edit	stagiaires.edit
put/patch	/stagiaires/{s}	update	stagiaires.update
delete	/stagiaires/{s}	destroy	stagiaires.destroy

I- Affichage des données :

La méthode Eloquent **all** renvoie tous les résultats dans la table du modèle.

```
$data= Stagiaire::all();
return view('stg.showStg',['stgs'=>$data]);
```

Étant donné que chaque modèle sert de générateur de requêtes, des contraintes peuvent également être ajoutées aux requêtes, puis utiliser la méthode get pour récupérer les résultats :

```
$data=Stagiaire::where('age','>=', 10)
->take(10)
->get();
```

Remarque : La méthode **find**

pour chercher un enregistrement par son clé primaire , on peut utiliser la méthode find.

II- Insertion d'un nouvel enregistrement :

Pour insérer un nouvel enregistrement dans la base de données, vous devez instancier une nouvelle instance de modèle et définir des attributs sur le modèle. Ensuite, appelez la méthode **save()** sur l'instance de

```php
public function store(Request $request)
 {
    $Stagiaire = new Stagiaire();
    $Stagiaire->nom = $request->nom;
    $Stagiaire->prenom = $request->prenom;
    $Stagiaire->age = $request->age;
    $Stagiaire->save();
  return response(« le stagiaire est ajouter !");
    }
```

Remarque: méthode create()

Avant d'utiliser cette méthode, vous devrez spécifier une propriété fillable ou guarded sur votre classe de modèle. Ces propriétés sont requises car tous les modèles Eloquent sont protégés par défaut contre les vulnérabilités d'affectation de masse.

```php
class Stagiaire extends Model
{
    use HasFactory;
    protected $fillable=['nom','prenom','age'];
   // protected $guarded=[];
   // protected $guarded=['age'];
}
```

Vous pouvez également utiliser la méthode **create** pour "enregistrer" un nouveau modèle à l'aide d'une seule instruction PHP.

```php
public function store(Request $request)
{
 Stagiaire::create([
  'nom'=>$request->nom,
  'prenom'=>$request->prenom,
  'age'=>$request->age
        ]);
 return response(« le stagiaire est ajoute !");
    }
```

ou

```php
Stagiaire::create($request->all());
```

III-Modification :

La méthode save peut également être utilisée pour mettre à jour des modèles qui existent déjà dans la base de données. Pour mettre à jour un modèle, vous devez le récupérer et définir les attributs que vous souhaitez mettre à jour. Ensuite, vous devez appeler la méthode save du modèle.

Exemple :

```
use App\Models\Stagiaire;
$s = Stagiaire::find(1);
$s->nom = 'AbouAli';
$s->save();
```

Exemple 2 :

```
Stagiaire::where('filiere',' dev'  )
->where('ville', 'safi')
->update(['etablissement' => ' ntic' ]);
```

IV-Suppression des enregistrements :

Pour supprimer un modèle, vous pouvez appeler la méthode delete sur l'instance de modèle :

```
use App\Models\Stagiaire;
$s = Stagiaire::find(1);
$s->delete();
```

Dans l'exemple ci-dessus, nous récupérons le modèle de la base de données avant d'appeler la méthode de suppression. Cependant, si vous connaissez la clé primaire du modèle, vous pouvez supprimer le modèle sans le récupérer explicitement en appelant la méthode destroy. En plus d'accepter la clé primaire unique, la méthode destroy acceptera plusieurs clés primaires, un tableau de clés primaires ou une collection de clés primaires :

```
Stagiaire::destroy(1);
Stagiaire::destroy(1, 2, 3);
Stagiaire::destroy([1, 2, 3]);
Stagiaire::destroy(collect([1, 2, 3]));
```

Bien sûr, vous pouvez créer une requête Eloquent pour supprimer tous les modèles correspondant aux critères de votre requête.
Exemple

```
use App\Models\Stagiaire;
$s = Stagiaire::where('age','>=', 10)->delete();
```

TP

Réaliser le projet qui permet de manipuler la table module comme suit : (utiliser ELOQUENT)