**Contexte** : PPE 2ème année – Framework PHP Laravel

Découverte d’un des plus importants Framework PHP : Laravel

Pré requis – Installation – Notions d’architecture

Développement d’une **application de suivi d’ étudiant pour l’ ENC**. Cette application permettra de gérer par exemples les cartes d’étudiant, la cantine, les stages, la poursuite d’études …

# Introduction

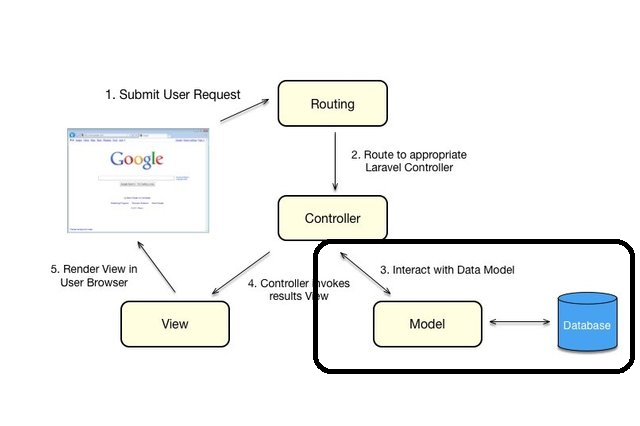
On doit avoir installé notre squelette d’application Laravel, accessible à l’URL carteenc.test

Dans cette partie nous allons aborder la partie Modele du framework Laravel.

# Modèles Laravel

## Introduction : qu’est ce qu’un modèle dans un Framework MVC

### Le M dans MVC



Le modèle est le composant central du Pattern MVC. Il contient les données dynamiques de l’application, de manière indépendante de l’interface utilisateur.

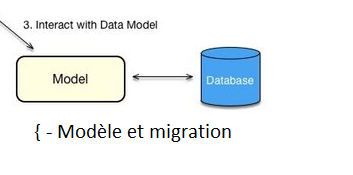
Typiquement, un Modèle est représenté par une **classe** (Java, C#, PHP …) dans l’application et les données sont persistées (sauvegardées) dans une table de la base de données (MySQL, SQL Server, PostGreSQL…).

Le modèle contient la logique applicative, indépendamment de la Vue et du Contrôleur.

### Le Model dans Laravel

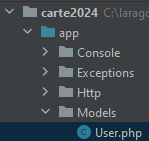
Laravel utilise l’ORM Eloquent avec le pattern Active Record (cf Annexe).

On va distinguer 2 choses, le Modèle lui-même et la migration, qui va permettre de créer la table correspondante dans la base de données.



A ce stade, nous avons juste les modèles et les migrations standard de Laravel automatiquement crées avec un nouveau projet.

* Les Modèles sont stockés dans **app\Models**

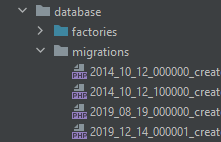


Les modèles dans le Framework Laravel sont des classes PHP qui vont hériter du Framework pour bénéficier de ses fonctionnalités.

Question 1 : Modèles standard

1. Quels modèles sont présents à ce stade dans l’application ? User.php
2. Quels sont les attributs ?name, email, password, remember\_token, email\_verified
3. Trouver et fournir le lien de la documentation correspondante

* Les Migrations sont stockées dans **database\migrations**



Les migrations vont permettre de créer des tables dans la base de données choisie et configurée, correspondant aux modèles.

A chaque Modèle va correspondre une table dans la base de données.

A chaque instance (objet) du Modele, si on souhaite la persister (sauvegarder), cela donnera une ligne dans la table correspondante.

Attention : la réciproque n’est pas vraie : on va trouver des migrations et des tables auxquelles ne correspondent pas des modèles dans notre application (cf Utilité du modèle standard, mise en place de l’authentification).

Dans Laravel, c’est l’ORM Eloquent qui est chargé d’assurer cette correspondance et qui fournit le support aux opérations CRUD (Create, Read, Update, Delete) associées.

Question 2 : Migrations standard

1. Quelles migrations sont présentes à ce stade dans l’application ?
2. Quels sont les champs définis dans la base de données ?
3. Examinez la table dans la base de données. Est-ce que tous les champs définis sont également présents dans le Modèle ? Pourquoi ?
4. Trouver et fournir le lien de la documentation correspondante

## Utilité du modèle standard

A ce stade nous disposons d’un modèle standard, mais que pouvons nous en faire ? Nous ne le voyons pas dans l’application puisque pour l’instant nous avons juste quelque chose qui ressemble à une page Web.

Nous allons anticiper sur la partie suivante, les VUES pour simplement voir l’utilité des modèles déjà à disposition.

Ils servent de support à l’authentification (on verra plus en détail dans le Bloc 3).

### Incorporation d’un mécanisme d’authentification standard de Laravel

Analyse : tout d’abord, examinez les routes avant de lancer la commande mettant en œuvre l’authentification dans Laravel.

Le fichier définissant les routes web (HTML) se nomme **routes\web.php**

Etape 1 : Dans la fenêtre Terminal de PHPStorm :



Etape 2 : Puis ensuite :



* Pour exécuter cette commande, via l’explorateur Windows, positionnez-vous dans le répertoire créé, par exemple : C:\laragon\www\carteENC puis ouvrir une ligne de commande (tapez cmd dans la fenêtre)
* Saisissez les instructions dans la fenêtre DOS

Pour examiner les routes :



Question n°3 :

* D’après vous, en examinant ces routes, quelles sont les URL relatifs à l’enregistrement d’un utilisateur puis à ses connexions ultérieures ?
* Quelles sont les nouvelles tables (migrations) qui ont été créées ?

Rappel : Le répertoire source des données migrées est dans : **database\migrations**

Etape 3 : Migration vers la Base de données :

Dans le répertoire du projet :



Cette commande migre les données dans la base de données.

Remarque : artisan est le nom du langage de commande des Laravel

# Test de la configuration avec authentification

Vous pouvez revenir sur l’application.

Questions 4 : Validation de l’authentification

* Qu’est ce qui a changé ?
* Testez les fonctionnalités qui ont été mises en place et documentez vos tests.

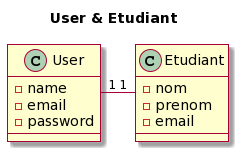
## Création d’un modèle Laravel

### Objectif :

On souhaite introduire la notion d’étudiant. Pour le moment un étudiant sera simplement caractérisé par son nom, son prénom et son email.

On notera qu’on fait la distinction entre utilisateur (User) et Etudiant car les utilisateurs de l’application pourraient être des membres de l’administration, des enseignants ou pourquoi pas des tiers, comme des parents d’élèves, tuteurs de stage …

Il semble logique d’associer un Etudiant à un User, d’où le modèle simple ci-dessous.



### Création du modèle Etudiant :

Etudiant - Etape 1 :



La commande artisan make : model permet de créer un modèle, qui va être une fabrique d’objets de la classe mentionnée, ici Etudiant, qui vont correspondre à une table dans la base de données.

Cette correspondance est assurée via l’option –m , qui est l’abréviation de migration.

En français : effectuez une création d’un modèle de Etudiant et préparez la migration correspondante.

Cette commande crée donc 2 fichiers :

1. Un fichier modèle, **app\Etudiant.php**
2. Un fichier de migration : **database\migrations**\XXXX\_ create\_ etudiants\_table.php

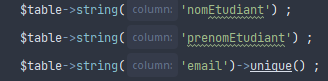
Etudiant - Etape 2 : exécution de la migration

L’exécution de la migration va permettre de créer la table avec la structure dans la base de données.

### modification de la structure de la table pour la carte d’étudiant

Etudiant - Etape 3 : modification de la structure

Editez le fichier **database\migrations**\XXXX\_create\_carte\_etudiants\_table.php, et modifiez la méthode (fonction) up() de la manière suivante : on va ajouter 3 champs



Etudiant - Etape 4 : Mise à jour de la structure des données

Effectuez la commande suivante :

Dans une fenêtre terminal PHPStorm ou autre :



Cette commande équivaut à un ***rollback***, c’est-à-dire l’annulation de toutes les transactions en base de données, et à un ***migrate***, donc elle va réinitialiser la base de données en tenant compte des modifications apportées à la table.

### Validation

Vérifiez en base de données que la table a bien été modifiée

# Echantillonnage de la base de données

## Fonctionnement

Lorsqu’on développe on souhaite très rapidement pouvoir disposer de données pour vérifier que les fonctionnalités que l’on implémente sont correctes et lorsqu’on fait des ajouts qu’on ne casse pas l’existant.

En général on passe pas mal de temps à tester à travers l’application, formulaires et autres.

C’est très chronophage.

Les frameworks modernes permettent de peupler la base de données facilement puis en suite de faire des tests pour valider les données et le principe d’alimentation.

Nous allons dans cette partie aborder le peuplement et ensuite les tests, déjà introduits l’an dernier.

## Peuplement de la base de données

Pour peupler une base de données avec des données fictives on utilise des Factories (usines) et des échantillonneurs (Seeders).

Par défaut, on a une Factory pour la table User, située dans database🡪factories et nommée UserFactory

Par défaut on a un échantillonneur nommé DatabaseSeeder dans database 🡪 seeders

Exercice 1 : Utilisation de l’échantillonneur par défaut

* Dans le fichier DatabaseSeeder.php, décommentez le code de la méthode ***run***
* Exécutez l’échantillonnage de la base de données

Commande à taper : php artisan db :seed

* Vérifier dans la table que les données ont bien été peuplées

Exercice 2 : Création d’un échantillonneur pour les Etudiants

* Créer une Factory pour l’Etudiant
* Ajouter 10 étudiants dans le DatabaseSeeder en utilisant la Factory créée

# Annexe : pattern Active Record

L’ORM Eloquent de Laravel utilise le pattern Active Record. L’ORM le plus connu de l’éco système PHP, Doctrine, utilisé par Symfony, suit le pattern Data Mapper, un peu plus complexe.

Cf ces articles pour une présentation et une comparaison : <https://www.infinitypp.com/software-patterns/activerecord-vs-datamapper-pattern-php-laravel/> ou <https://tsh.io/blog/active-record-vs-data-mapper-patterns-in-php/> ou encore <https://www.thoughtfulcode.com/orm-active-record-vs-data-mapper/>

Pour simplifier, Eloquent que nous utilisons avec Laravel est plus simple mais il y a un couplage fort entre la base de données et le modèle.

Il est très simple à utiliser avec des opérations CRUD standard (Create, Read, Update, Delete). Chaque instance de la classe (le Model) correspond à une ligne de la table.