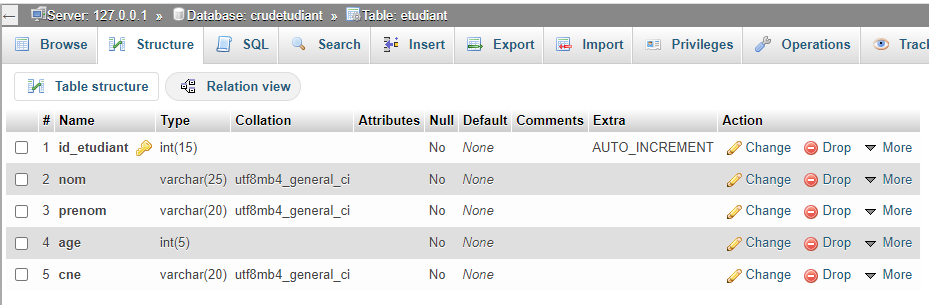
**Atelier 1**

**Réalisé par Najlae ABARGHACHE**

**Mise en place d'un CRUD Etudiant**

## **La BD :**

La base de donnée crée est nommée '**crudetudian**t' et se compose d'une table '**etudiant**', qui se compose de 5 colonnes; id\_etudiant, nom, prenom, age et cne.



## **La partie Modèle (dossier framework\_crud/Model) :**

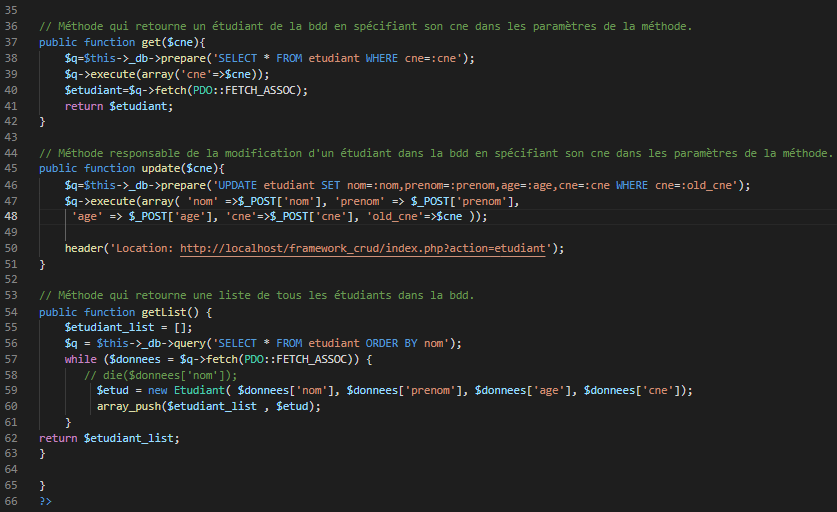
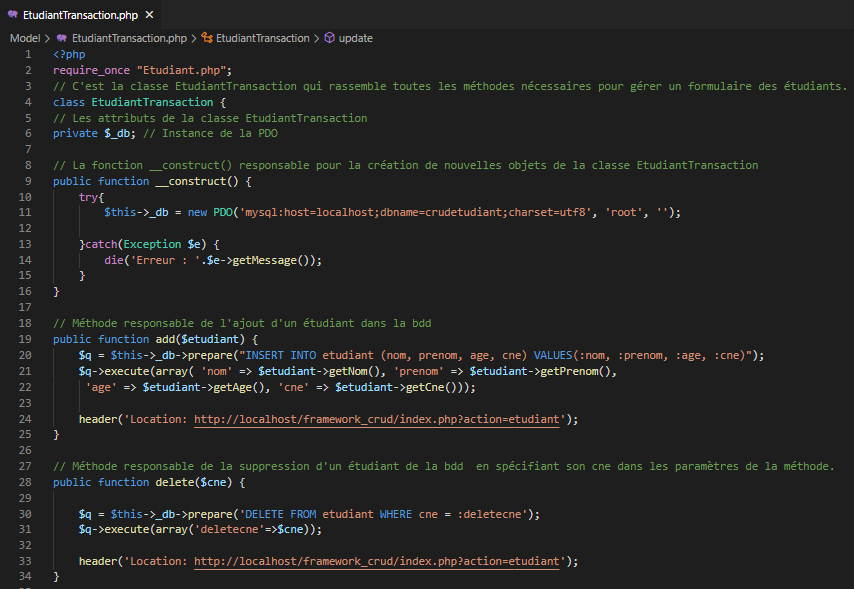
La partie **Modèle** a comme rôle d'encapsuler la logique métier et l'accès aux données. PHP est un langage qui supporte pleinement tous les concepts de la POO, donc afin d'améliorer notre architecture MVC, nous allons mettre en œuvre ces concepts dans la partie Modèle.

Nous devons alors créer une classe Métier '**Etudiant**' modélisant l'entité 'etudiant', et une 2ème classe Métier '**EtudiantTransaction**' rassemblant les méthodes nécessaires pour gérer un formulaire des étudiants; ajouter un étudiant, supprimer un étudiant, modifier dans les information d'un étudiant, une méthode qui retourne la liste des étudiants, et une méthode retourne un étudiant possédant un cne spécifié.

Voici le fichier **Model\Etudiant.php** où on a définit le classe Etudiant :

## 

Voici le fichier **Model\EtudiantTransaction.php** où on a définit le classe EtudiantTransaction:

****

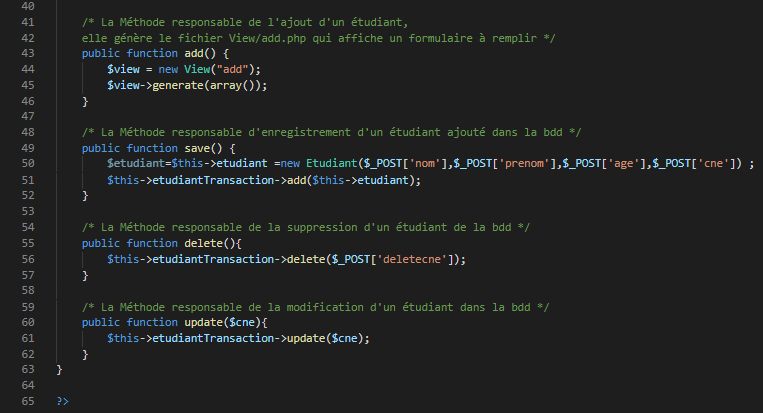
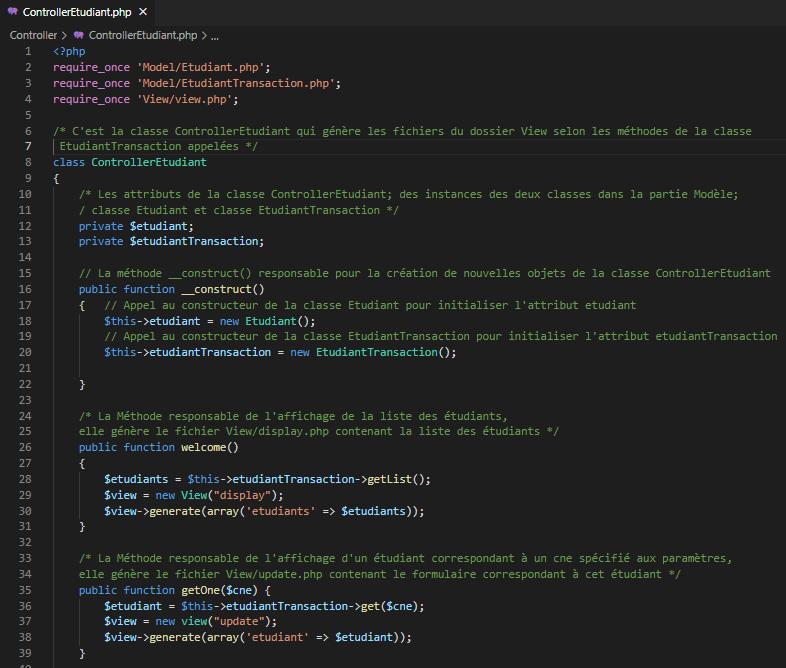
## **La partie Contrôleur (dossier framework\_crud/Controller) :**

La partie **Contrôleur** a comme rôle d'encapsuler de gérer la dynamique de l'application. Elle fait le lien entre l'utilisateur et le reste de l'application.

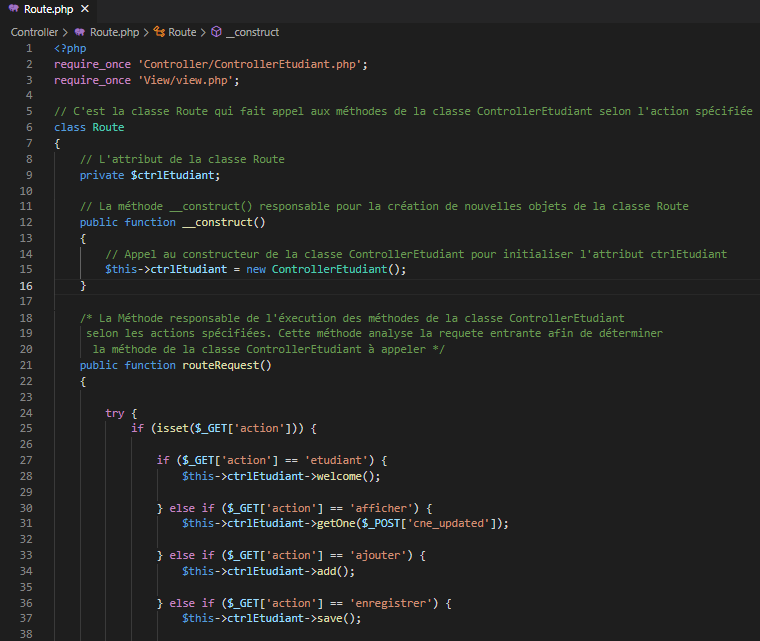
Nous devons alors créer une classe **ControllerEtudiant** qui aura comme rôle la récupération des données nécessaires aux vues, càd une liaison entre la génération des fichiers de la partie Vue et la classe EtudiantTransaction, décrite avant, qui rassemble les méthodes nécessaires pour gérer un formulaire des étudiants. Cette classe génère les fichiers du dossier framework\_crud/View selon les méthodes appelées de la classe EtudiantTransaction définie dans le fichier framework\_crud/Model/ EtudiantTransaction.php

Nous devons alors créer une classe **Route** qui sera une liaison entre les méthodes de la classe ControllerEtudiant et les actions spécifiées. C'est un contrôleur frontal, son rôle principal est d'analyser la requête entrante pour déterminer l'action à entreprendre à l'aide de sa méthode routeRequest. Cette classe a donc comme rôle le routage de la requête.

Voici le fichier **Controller\ ControllerEtudiant.php** où on a définit le classe ControllerEtudiant:



Voici le fichier **Controller\ Route.php** où on a définit le classe Route:



## **La partie Vue (dossier framework\_crud/View) :**

La partie **Vue** s'occupe des interactions avec l'utilisateur: présentation, saisie et validation des données**.**

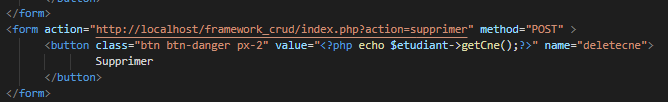
Afin de gérer un formulaire pour les étudiants, il sera impossible de réduire notre site web à une seule page.

* On doit avoir une page affichant la liste des étudiants enregistrés, qui sera définie dans le fichier **display.php**
* Une autre affichant le formulaire à remplir pour ajouter un étudiant, qui sera définie dans le fichier **add.php**
* Une affichant le formulaire à remplir pour modifier les informations d'un étudiant choisi, qui sera définie dans le fichier **update.php**

Ces trois premières pages vont avoir des éléments communs affichées à l'utilisateur. Il serait donc souhaitable de les définir à un seul endroit, càd dans un seul fichier. On parle alors de l'utilisation d'un fichier template, un modèle contenant tous les éléments communs et permettant d'ajouter les éléments spécifiques à chaque vue. On va définir ce modèle dans le fichier **gabarit.php**

On aura également besoin d'une page pour l'affichage des erreurs qu'on va définir en fichier **Error.php**

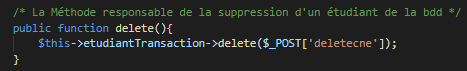
Le sous dossier View va contenir ainsi un fichier **delete.php** qui aura en code php la fonction var\_dump() qui affiche les informations structurées d'une variable passée en paramètres, ici c'est la variable $deletecne, qui est envoyée par la méthode POST du fichier display.php, et reçue par la méthode GET dans les deux fichier delete.php, et le fichier Controller/Route.php où la classe Route est définie, la méthode routeRequest de cette classe va analyser la requete recue par $\_GET['action'] et puisque lors de l'envoi de la variable $deletecne, on a spécifié l'action action=supprimer dans le fichier display.php:



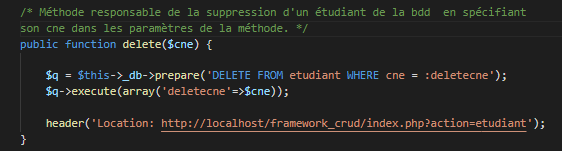
Alors la méthode routeRequest va executer le cas où $\_GET['action'] = supprimer:



alors elle va faire appel à la méthode delete() de la classe ControlollerEtudiant définie dans le fichier Controller/ ControlollerEtudiant.php,



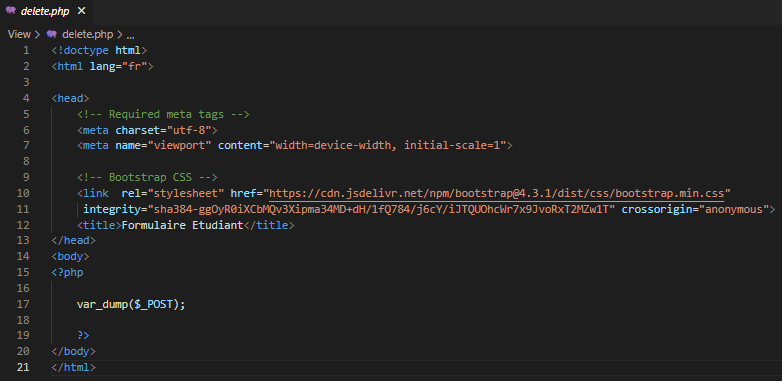
qui va aussi faire appel à la méthode delete de la classe EtudiantTransaction en lui passant aux paramètres la variable $deletecne par la méthode POST, comme on voit dessus.



Cette méthode exécute alors la requête sql delete où cne = deletecne, et dès qu'on clique sur le bouton Supprimer définie sur le fichier display.php, la requête s'exécute et on se dirige directement vers la page principale où la liste des étudiants enregistrés est affichée.

Et comme ça l'étudiant choisi sera supprimé de la bdd, càd supprimé du formulaire des étudiants.

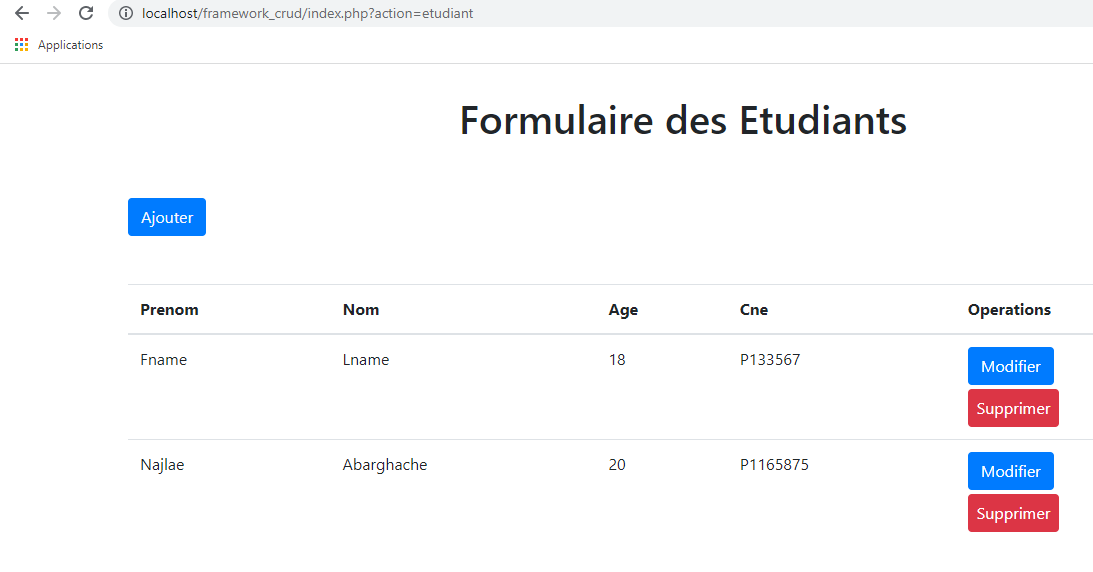
Voici le fichier **View\ delete.php** :



Presque le même cas pour l'option modifier (update), sauf que là, l'utilisateur aura besoin d'entrer les nouvelles informations de l'étudiant choisi, donc les choses vont être un peu compliqués. Essayons d'aller étape par étape !

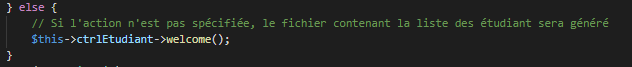
Tout d'abord, on doit parler un peu de la page principal et du fichier responsable de son affichage.

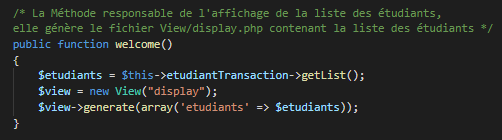
On est dans la page d'accueil, la page où la liste des étudiants est affiché, la page qui s'affiche si action=etudiant où bien si on spécifie pas l'action comme on a décrit dans le fichier Route.php , voici un apperçu de cette page:

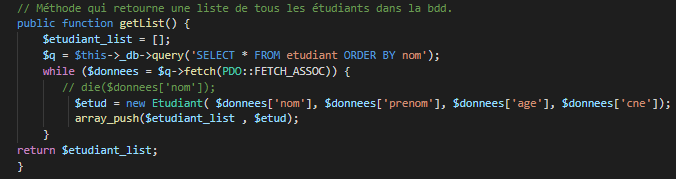


Cette page est définie dans le fichier display.php qui se génère lors de l'appel de la méthode welcome() de la classe ControllerEtudiant qui de même fait appel à la méthode getList() de la classe EtudiantTransaction. La méthode welcome() est appelé si $\_GET['action'] = supprimer ou bien si l'action n'est pas spécifiée comme on a dit avant :

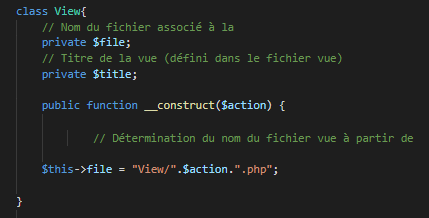




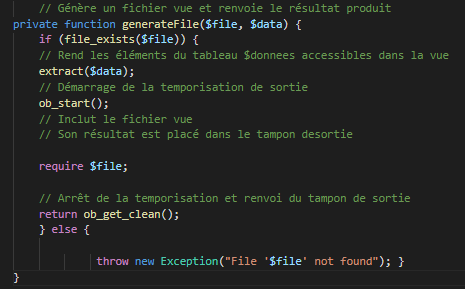




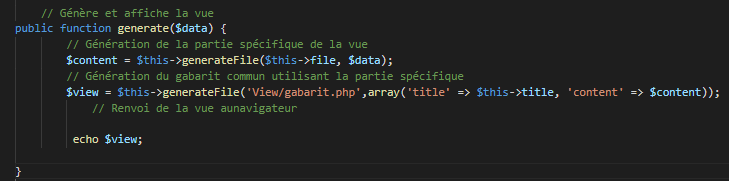
On voit bien dans la méthode welcome() l'instanciation d'un objet $view de la classe View, l'instanciation de cet objet permet d'indiquer le fichier View/display.php :



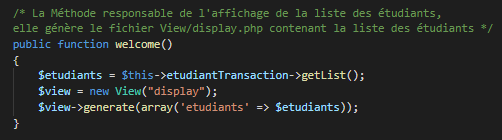
Après l'instanciation de l'objet $view, on a fait appel à la méthode generate de la classe View, en lui passant aux paramètres la liste des étudiants stockée dans le variable $etudiants à travers l'appel de la fonction getList() qui retourne un tableau de toutes les données contenues dans la table etudiant. Avant d'étudier le rôle de la méthode generate de la classe View, on doit tout d'abord passer par la fonction generateFile qui prend en paramètres le nom du fichier et les données à rendre accessibles dans ce fichier, elle inclut ce fichier s'il existe et place le résultat, qui est les données devenant accessibles dans ce fichier, dans le tampon de sortie et le retourne après avoir arrêter la temporisation:



Et voici la méthode generate qui prend en paramètres les données à rendre accessibles dans le fichier $data, fait appel à la fonction generateFile en lui passant aux paramètres le nom du fichier de l'objet instanciée (et donc ici View/display.php) et $data, elle stocke le résultat dans la variable $content, et puis fait appel encore une autre fois à la fonction generateFile en lui passant aux paramètres le nom du fichier View/gabarit.php et un tableau des données représentant le titre dans le fichier de l'objet instanciée et le contenu de la variable $content, elle stocke le résultat dans la variable $view et l'affiche :

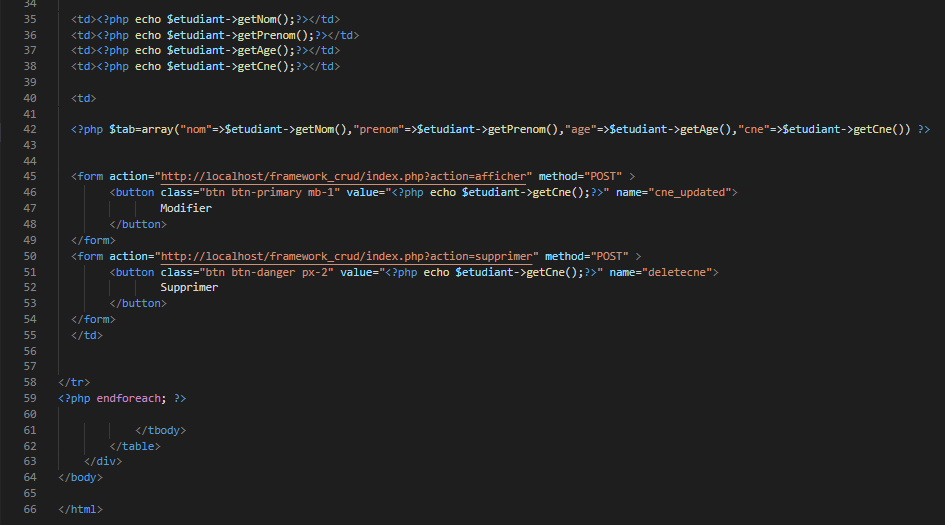
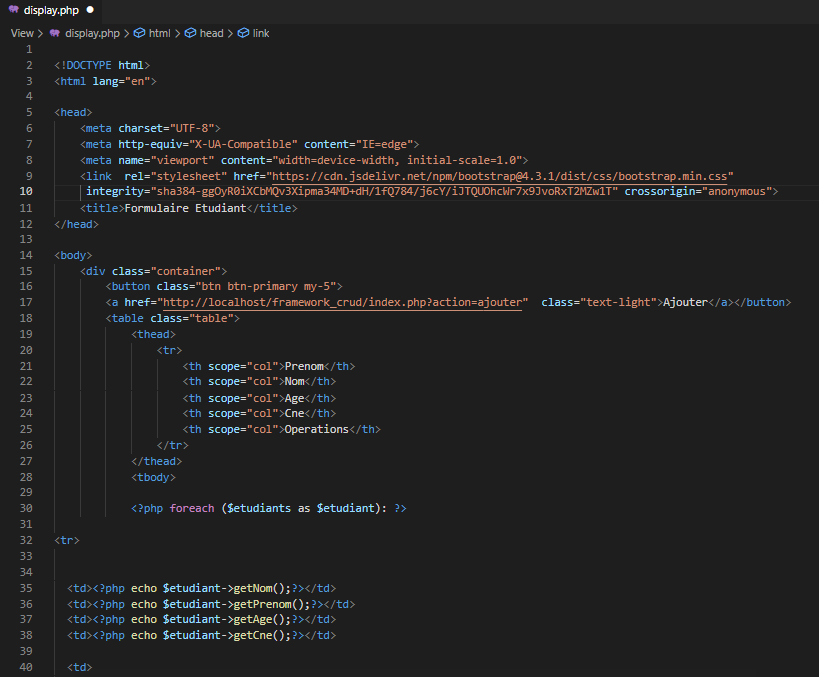


En générant un fichier, les données passées en paramètres $data vont devenir accessibles dans ce fichier. Dans le cas du fchier View/display.php , un tableau contenant la variable $etudiants qui stocke une liste des étudiants enregistrés va devenir accessible dans le fichier display.php



On peut alors référer à ce tableau qui se compose d'une case par son étiquette $etudiants dans le fichier display.php

Voici le fichier **View\ display.php** :

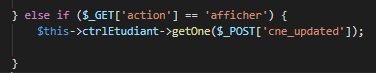


On a construit un tableau en HTML qui se compose des colonnes Prenom, Nom, Age, Cne, et Opérations et puis à travers un code php on fait appel appel aux getters de la classe Etudiant pour chaque etudiant du tableau $etudiants dont on a parlé avant, et on affiche ces données.

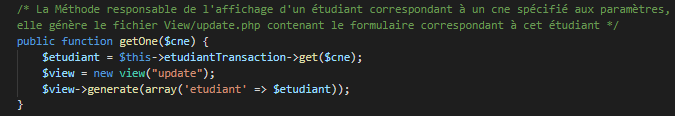
Encore une fois, on fait appel aux getters pour chaque etudiant et on stocke les informations dans un tableau $tab aux étiquettes 'nom', 'prenom', 'age', 'cne' , afin de pouvoir les accéder dans les autres fichiers.

On est arrivée à la partie des boutons ! On a déjà parlé de l'action supprimer, on parlera alors de l'action modifier.

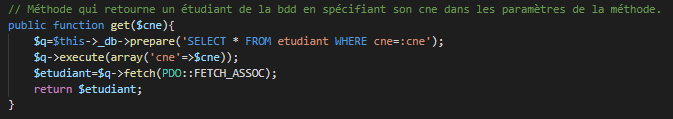
Comme il est indiqué dans le code si on clique sur Modifier dans la ligne correspondant à un étudiant choisi de la liste, la méthode POST va envoyer la valeur du cne de cet étudiant en la stockant dans la valeur du $cne\_updated, après l'avoir eu à l'aide du getter, ça sera alors action=afficher, on sera dirigé vers le fichier généré à cette action, on doit donc aller vérifier le cas dans le fichier Route.php :



On fait appel à la méthode getOne de la classe ControllerEtudiant en lui passant aux paramètres la valeur du cne\_updated reçue par la méthode POST:

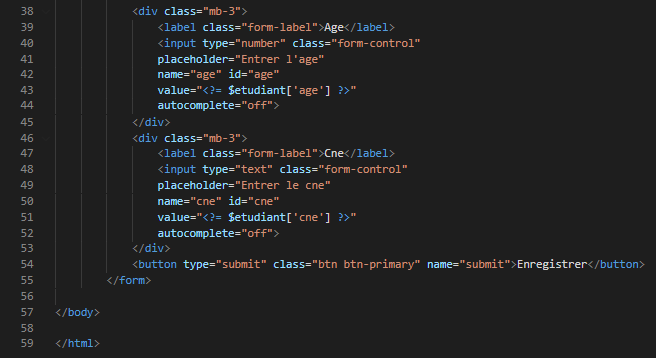
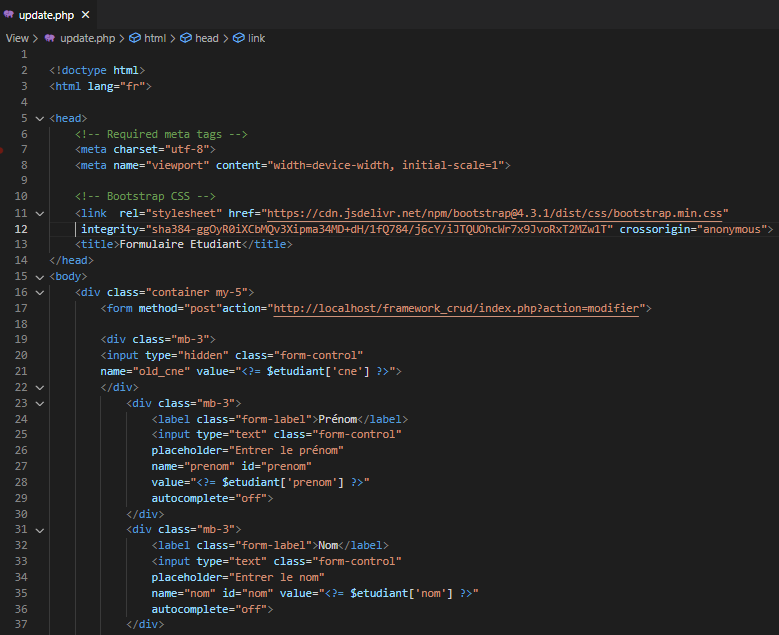


Cette méthode fait appel à la méthode get de la classe EtudiantTransaction afin de retourner à partir de la table etudiant les données de l'étudiant possédant le cne équivalent à la valeur cne\_updated et le stocke dans la variable $etudiant:

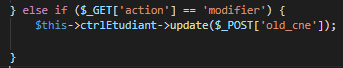


Et puis elle génére le fichier **View/update.php** et fait appel à la méthode generate de la classe View afin de rendre le tableau qui stocke les données de la variable $etudiant accessible dans le fichier update.php sous l'étiquette 'etudiant'.

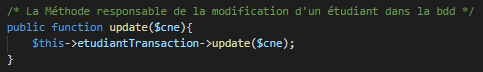
On passe alors au fichier update.php:



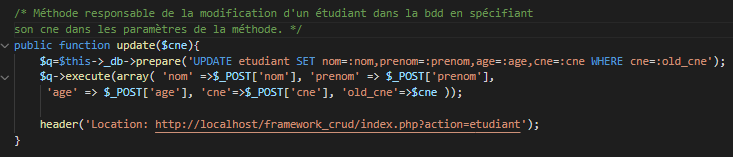
On voit bien qu'à partir de la méthode POST on a envoyé les nouvelles valeurs saisies et les stocker dans les variables nom, prenom, age, cne afin de les exploiter dans la méthode update de la classe EtudiantTransaction pour effectuer les changements sur les données de l'etudiant possédant le cne équivalent à la valeur de old\_cne, et cela lorsque action\_modifier, on retourne au fichier Route.php:



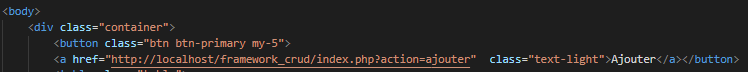
On fait appel à la méthode update de la classe ControllerEtudiant en lui passant aux paramètres la valeur du old\_cne par la méthode POST:



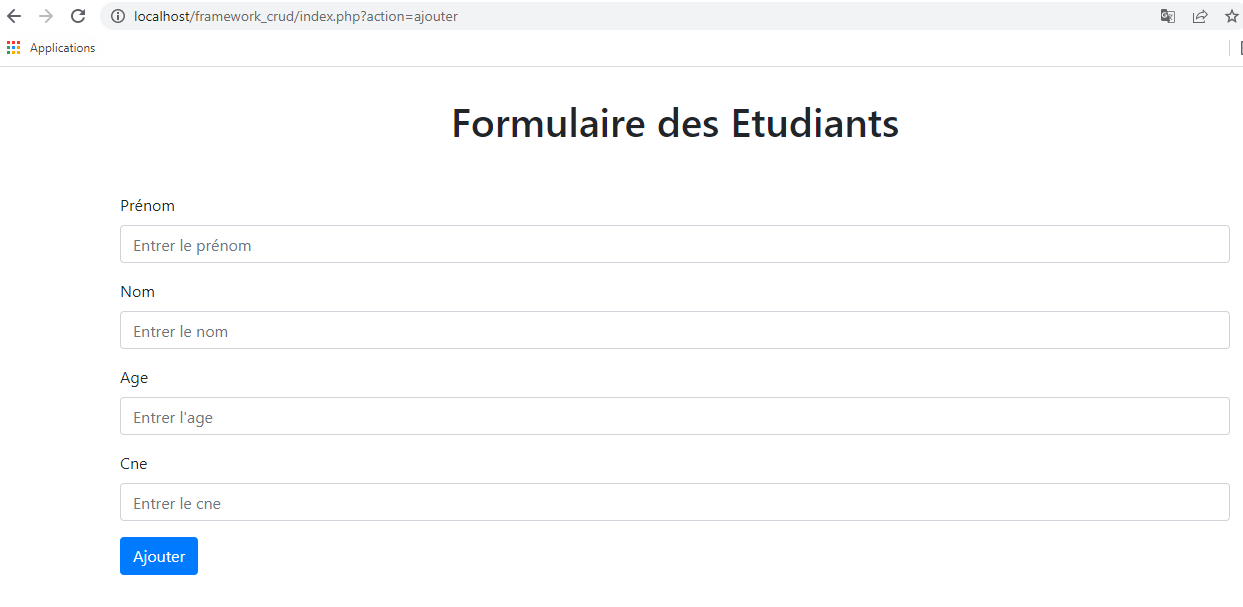
Cette méthode fait appel à la méthode update de la classe EtudiantTransaction en lui passant aux paramètres le cne équivalent à la valeur old\_cne, elle exécute alors la requête sql update sur cet ligne où cne = old\_cne, et change ses données par les valeurs reçues par la méthode POST du fichier update.php et dès qu'on clique sur le bouton Enregistrer définie sur le fichier update.php, la requête s'exécute et on se dirige directement vers la page principale où la liste des étudiants enregistrés est affichée:



Afin d'ajouter un étudiant à la liste des étudiants, on doit cliquer sur le bouton étudiant, et c'est ce qui est défini dans le fichier display.php ; le bouton Ajouter:



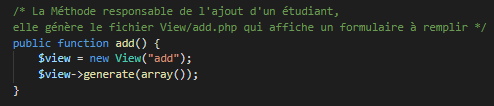
Comme il est défini dans le code, si on clique sur ce bouton, on se dirige vers la page où action=ajouter :



on retourne au fichier Route.php pour comprendre le processus:

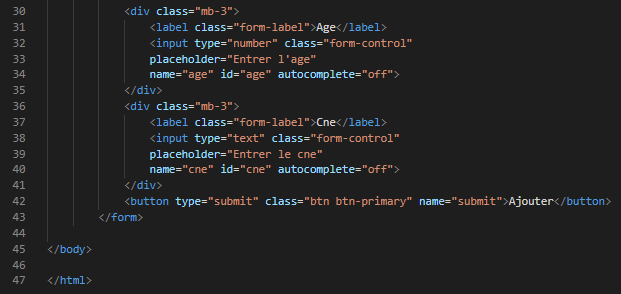
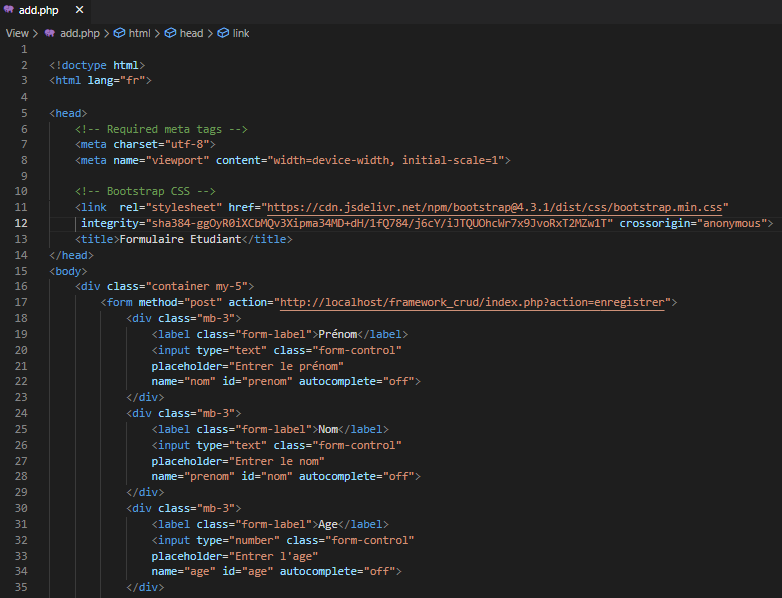


On fait appel à la méthode add de la classe ControllerEtudiant:



Cette méthode génère le fichier **View/add.php** et fait appel à la méthode generate de la classe View en lui passant aux paramètres un tableau vide qui sera rempli par les saisies de l'utilisateur dans le fichier add.php .

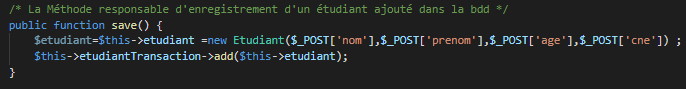
Voici le fichier **View\ add.php** :



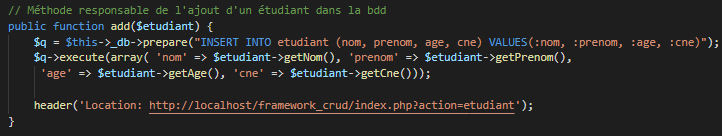
A travers la méthode POST on a envoyé les nouvelles valeurs saisies l'action égale à enregistrer et les stocker dans les variables nom, prenom, age, cne afin de les exploiter dans la méthode save de la classe EtudiantTransaction pour effectuer l'enregistrement de ces données saisies dans la table etdiant, et cela lorsque action\_enregistrer, on retourne au fichier Route.php:



On fait appel à la méthode save de la classe ControllerEtudiant:

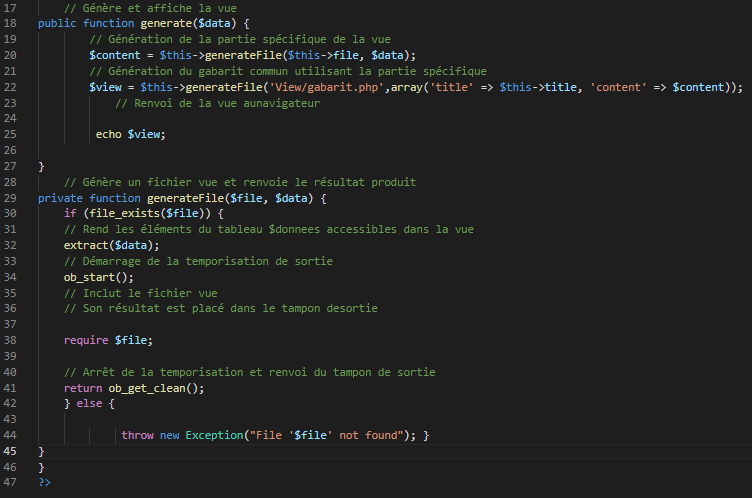
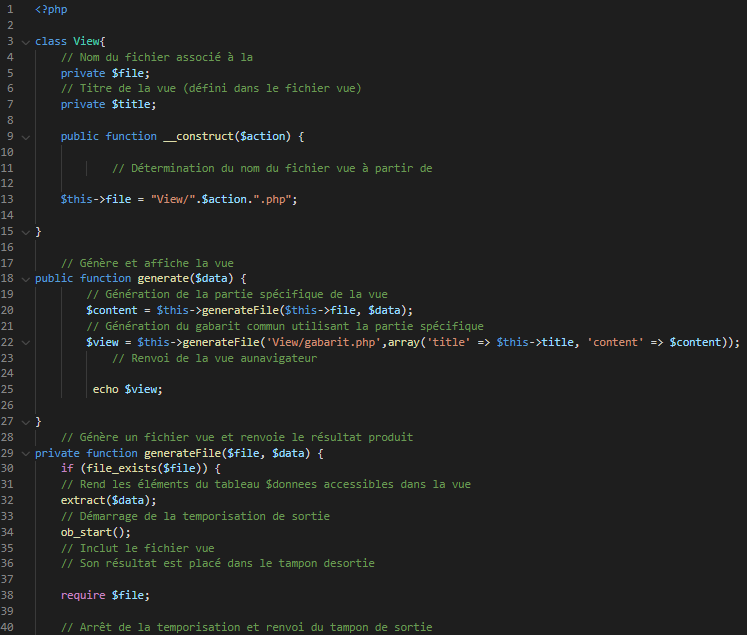


Cette méthode crée une instance $etudiant de la classe Etudiant à travers les valeurs reçues par la méthode POST du fichier add.php, et puis elle fait appel à la méthode add de la classe EtudiantTransaction en lui passant aux paramètres l'étudiant crée (à ajouter) $etudiant:

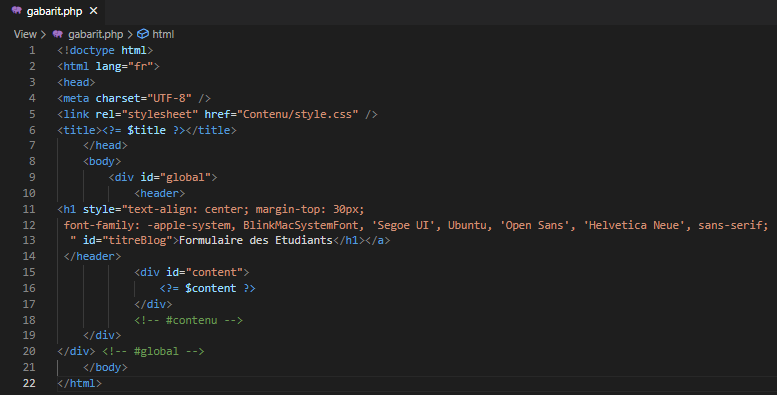


Cette méthode exécute la requête sql 'insert into' dans la table etudiant et passe les données de la variable $etudiant à l'aide des getters à la requête, et dès qu'on clique sur le bouton Ajouter définie sur le fichier add.php, la requête s'exécute et on se dirige directement vers la page principale où la liste des étudiants enregistrés est affichée.

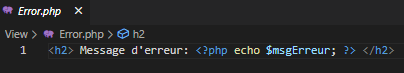
Voici le fichier **View\ view.php** où on a définit la classe View responsable de la génération des fichiers de la partie Vue avec acces aux données spécifiées de la bdd:



Voici le fichier **View\ gabarit.php** qui tous les éléments communs entre les fichiers vues:

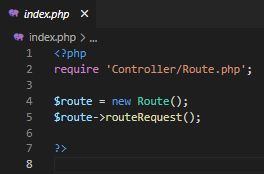


Voici le fichier **View\ Error.php** responsable de l'affichage des erreurs:



## **La racine du dossier framework\_crud :**

On définit dans la racine du dossier framework\_crud le fichier **index.php** où on instancie un objet de la classe Route après avoir inclut le fichier Controller/Route.php et on fait appel à la fonction responsable de l'analyse des requêtes (des actions):



et comme ça, lorsqu'on colle le lien :

http://localhost/framework\_crud/index.php

* Sans action spécifiée, la page principales sera affichée.
* Avec action spécifiée, une page correspondante à cette action sera affichée (comme on a expliqué avant).