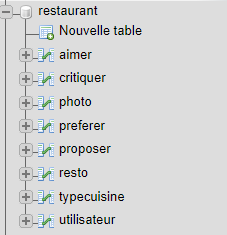
**BELABED Yacine**

**1TSSIOA**

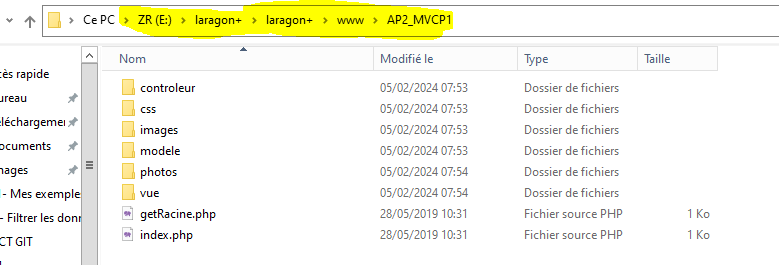
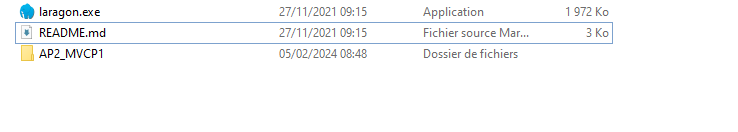
***05/02/2024***

**PARTIE 1 - Généralités :**

Importation de la base de données nommée “restaurant” directement sur PhpMyAdmin :



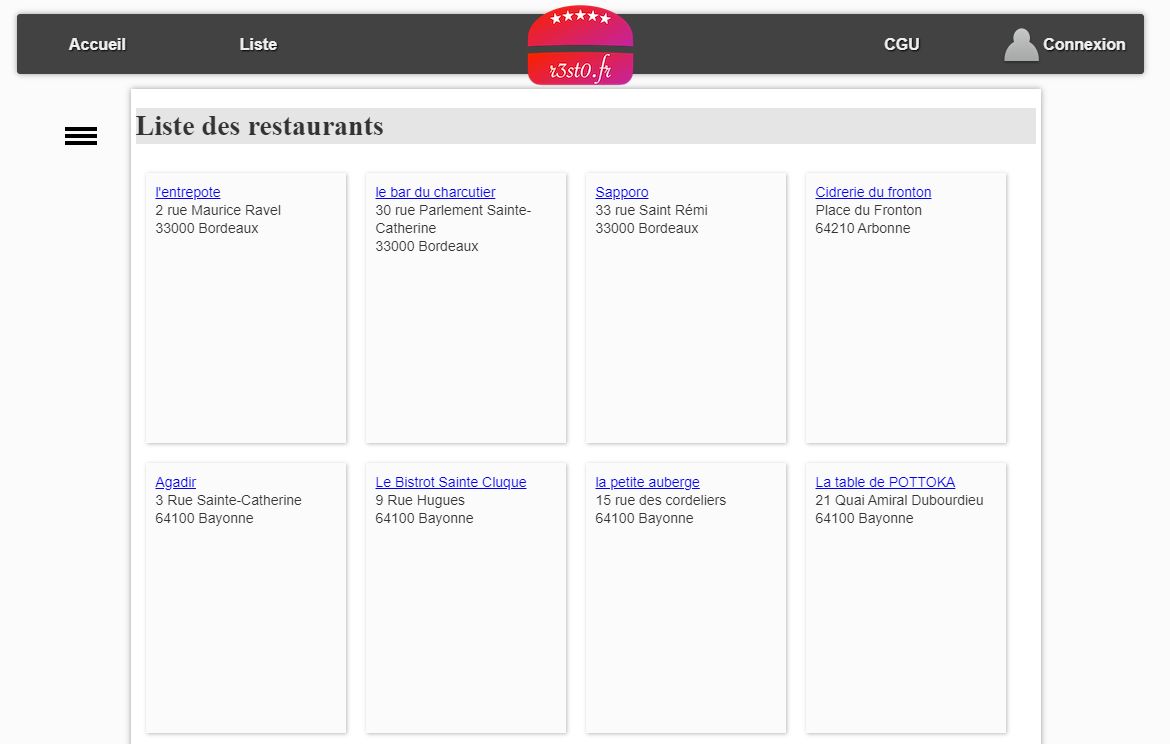
Création d’un nouveau projet PHP appelé AP2\_MVCP1 dans notre outil de développement, dans un dossier de votre serveur local (ici Laragon) :

Résultat de l’arborescence : 

Paramétrage du site pour qu'il utilise notre base de données (modifications apportés dans le script modele/bd.inc.php) :afin qu’il puisse donc la lire :



Résultat d’affichage :



**Questions 1 - Analyse de l'affichage de la liste des restaurants**

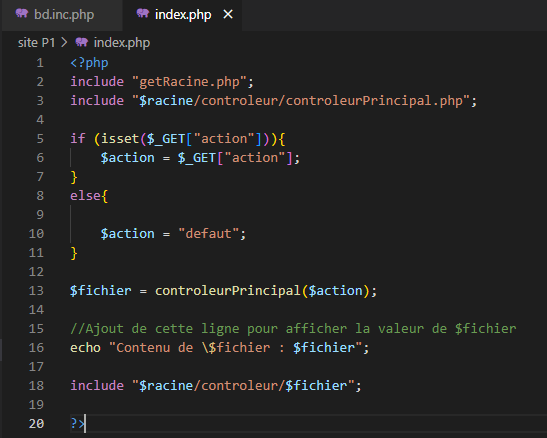
1.1°) *“include\_once”* est utilisé pour inclure et évaluer un fichier spécifié lors de l'exécution du script PHP. La principale différence entre *include\_once* et *include* réside dans le fait que *include\_once* vérifie si le fichier a déjà été inclus, et si c'est le cas, il ne le réinclut pas. Cela garantit que le même fichier n'est inclus qu'une seule fois, évitant ainsi les problèmes de redondance.

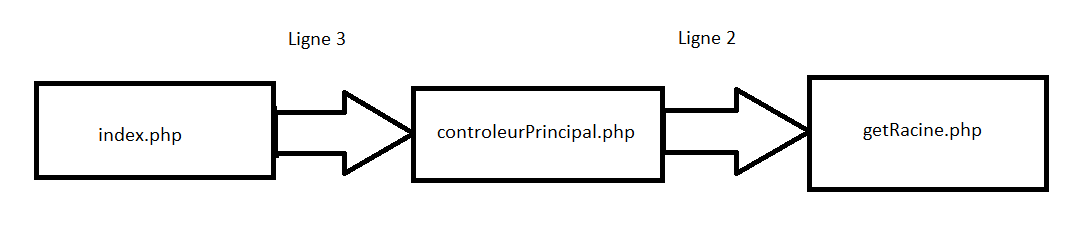
→ Syntaxe : ***include\_once 'fichier.php';***

*“include”* est utilisé pour inclure et évaluer un fichier spécifié lors de l'exécution du script PHP. Il peut être utilisé pour inclure du code, des fonctions, des classes, etc., provenant d'un fichier externe dans le script en cours.

→ Syntaxe : ***include 'fichier.php';***

1.2°) Dans le script PHP, la variable ***$fichier*** est définie en appelant la fonction ***controleurPrincipal($action)***. Cela signifie que la valeur de ***$fichier*** dépend de la valeur renvoyée par cette fonction.

Pour déterminer la valeur de la variable ***$fichier*** on utilise l’instruction d’affichage ***“echo”*** : 

1.3°) Schéma : 

→ ***“index.php”*** fait appel à ***“controleur.php”***. Dans le fichier index.php, cet appel se fait au sein de la vue nommé donc ***“getRacine.php”***.

**(Modèle → Contrôleur → Vue)**

1.4°)

| Fonction | Requête |
| --- | --- |
| function getRestoByIdR($idR) | select \* from resto where idR=:idR |
| function getRestos() | select \* from resto |
| function getRestosByNomR($nomR) | select \* from resto where nomR like :nomR |
| function getRestosByAdresse($voieAdrR, $cpR, $villeR) | select \* from resto where voieAdrR like :voieAdrR and cpR like :cpR and villeR like :villeR |

1.5°) Les requêtes SQL analysées sont conçues pour interroger la base de données en utilisant ***PDO*** pour une exécution sécurisée, visant à récupérer des données de la table resto. Elles se caractérisent par l'emploi de l'instruction **SELECT**, l'utilisation de conditions de filtrage via **WHERE** pour des recherches spécifiques, et l'application de paramètres nommés pour lier les valeurs et prévenir les injections SQL. Les résultats sont ensuite retournés sous forme de tableaux associatifs, assurant une manipulation facile des données récupérées.

1.6°) Le script ***bd.resto.inc.php*** contient des fonctions pour interroger une base de données sur des restaurants, permettant de récupérer des informations basées sur l'identifiant d'un restaurant, le nom, ou des détails d'adresse. Les fonctions utilisent des requêtes SQL paramétrées pour assurer la sécurité et l'efficacité de la recherche. La section de test à la fin du script permet de valider individuellement chaque fonction en exécutant le fichier directement, facilitant le développement et le débogage en affichant les résultats de requêtes spécifiques.

1.7°) Le fichier **listeRestos.php** utilise effectivement la fonction **getRestos()** du script *bd.resto.inc.php* pour récupérer les informations sur tous les restaurants. Cette fonction est appelée pour obtenir un tableau contenant les détails de chaque restaurant présent dans la base de données, que le script stocke ensuite dans la variable ***$listeRestos***. Ces données sont destinées à être utilisées pour l'affichage dans la vue correspondante, comme indiqué par l'inclusion de **vueListeRestos.php**, qui gérerait probablement l'affichage de cette liste de restaurants à l'utilisateur.

**Questions 2 - Début d’analyse de la structure du site**

2.1°) Dans le contexte du modèle MVC (Modèle Vue Contrôleur), chaque composant a un rôle spécifique :

**Modèle** : Gère les données et la logique métier de l'application.

**Vue :** Présente les données à l'utilisateur, généralement à travers des éléments HTML.

**Contrôleur** : Fait le lien entre l'utilisateur, les modèles et les vues. Il traite les entrées de l'utilisateur, fait appel aux modèles pour les données et choisit la vue à présenter.

Après avoir observé le contenu des scripts dans les dossiers contrôleur et vue du projet fourni en ressources, on constate que :

→ ***Dans les fichiers du dossier contrôleur :*** On ne devrait normalement pas trouver d'éléments de CSS ou de HTML. Le contrôleur se concentre sur la logique de traitement des demandes de l'utilisateur, décide quelle action exécuter dans le modèle, et détermine quelle vue afficher. Tout contenu lié à la présentation (CSS, HTML) devrait être évité dans cette partie pour maintenir la séparation des préoccupations.

→ ***Dans les fichiers du dossier modèle :*** Il ne devrait également y avoir aucun élément de CSS ou de HTML. Le modèle est dédié à la gestion des données, à l'exécution des opérations de base de données (si applicable), et à la logique métier. Inclure du HTML ou du CSS dans cette partie irait à l'encontre des principes du MVC, car cela mélangerait la logique de présentation avec la logique métier et l'accès aux données.

2.2°) Les fichiers gérant les éléments de HTML et de CSS sont contenus dans les dossiers **css** et **vue**.

2.3°) Le code PHP contenu dans **vueListeRestos.php** n'est pas directement visible par le navigateur après réception. Lorsqu'un utilisateur visite la page dans son navigateur, le serveur exécute le code PHP côté serveur et génère une réponse HTML. Ce HTML résultant est ce qui est effectivement envoyé au navigateur de l'utilisateur.

Ainsi, le code PHP est remplacé par le HTML généré avant d'être envoyé au navigateur, et c'est le HTML qui est affiché côté client.

2.4°) Le point commun entre toutes les fonctions définies dans le script **bd.resto.inc.php** est qu'elles effectuent des opérations liées à la récupération d'informations depuis la base de données concernant les restaurants. Ces fonctions sont destinées à interagir avec une base de données pour récupérer des données spécifiques en utilisant des requêtes SQL.

2.5°) Pour déterminer si d'autres fichiers que ceux du dossier modèle contiennent des références à la base de données, il faut examiner le contenu de ces fichiers.

**Ici, nous ne trouvons pas d’autres fichiers que ceux du dossier modèle des références à la base de données.**

***Synthèse* :** Le dossier **"Modèle"** gère l'accès aux données et la logique de base de données, tandis que le dossier **"Vue"** est chargé de la présentation visuelle des données. Ces deux composants, en collaboration avec les contrôleurs, forment un modèle MVC qui sépare les préoccupations, améliorant ainsi la maintenabilité et la modularité de l'application.

**Questions 3 – Début d'analyse de la partie contrôleur**

3.1°) La fonction d'accès aux données utilisée dans ce contrôleur est **getRestos()**, et elle permet de récupérer la liste complète des restaurants à partir de la base de données.

3.2°) L'affichage des données récupérées est effectué dans le script de vue. Le contrôleur se charge principalement de récupérer les données nécessaires et d'inclure les scripts de vue appropriés, et non de l’affichage ici.

3.3°) Ces scripts inclus sont responsables de générer différentes parties de la page web. L'en-tête (**entete.html.php**), la vue principale (**vueListeRestos.php**), et le pied de page (**pied.html.php**) sont combinés pour former la page HTML complète qui sera affichée à l'utilisateur. Chacun de ces scripts a un rôle spécifique dans la présentation visuelle de la page.

3.4°) Le contrôleur utilise la méthode GET pour récupérer la valeur de ***"idR"*** depuis l'URL, puis utilise cette valeur pour obtenir les détails d'un restaurant spécifique à afficher sur la page.

3.5°) La fonction **getRestoByIdR()** permet de récupérer les détails d'un restaurant spécifique à partir de la base de données en utilisant son identifiant comme paramètre. Cela permet d'offrir une flexibilité car la fonction peut être réutilisée pour récupérer les informations de différents restaurants en fonction de l'identifiant fourni.

3.6°) Dans le fichier **vueDetailResto.php**, la variable **$unResto** est utilisée pour afficher les détails d'un restaurant spécifique. Voici les sections où cette variable est utilisée :

1. <h1><?= $unResto['nomR']; ?></h1>
2. <p id="principal"><?= $unResto['descR']; ?></p>
3. <h2 id="adresse">Adresse</h2>

<p>

<?= $unResto['numAdrR']; ?>

<?= $unResto['voieAdrR']; ?><br />

<?= $unResto['cpR']; ?>

<?= $unResto['villeR']; ?>

</p>

1. <h2 id="horaires">Horaires</h2>

<?= $unResto['horairesR']; ?>

3.7°) Dans le fichier **vueDetailResto.php** :

→ La variable **$menuBurger** est utilisée pour générer un menu contextuel sous forme d'icône burger, affichant des liens spécifiés dans un tableau associatif.

→ La variable **$listeRestos** est utilisée pour créer une liste de cartes affichant les informations de plusieurs restaurants, avec des liens vers les détails de chaque restaurant.

3.8°) Le contrôleur n'accède pas directement à la base de données. Son rôle principal est de coordonner l'interaction entre le modèle.

**Synthèse contrôleur**

1°) Les données créées ou récupérées sont affichées dans le contrôleur à plusieurs endroits. Voici où les données sont manipulées et affichées :

→ Récupération des données : Les données sont récupérées à l'aide de la fonction ***getRestos()*** dans le contrôleur. Cette fonction permet de récupérer les restaurants à afficher. Les données récupérées sont stockées dans la variable ***$listeRestos*** .

→ Affichage des données : Les données récupérées sont ensuite affichées dans la section suivante du code :

***for ($i = 0; $i < count($listeRestos); $i++) {***

*// Affichage des détails de chaque restaurant*

***}***

Dans cette boucle, les détails de chaque restaurant, **tels que le nom, l'adresse et d'autres informations**, sont affichés à l'aide de balises HTML et de variables PHP .

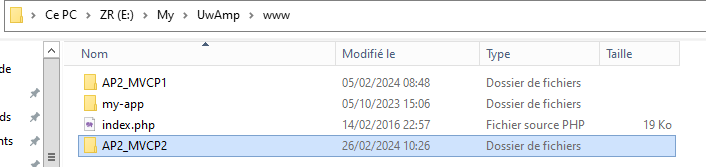
2°) Pour transmettre des ***données de l'utilisateur vers le contrôleur***, cela peut se faire par le biais de requêtes HTTP ou encore de formulaires web. Les données transmises sont ensuite traitées par le contrôleur pour effectuer les actions appropriées, telles que la récupération, la modification ou la suppression de données dans la base de données.

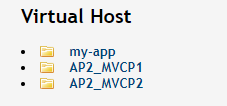
3°) Pour transmettre des ***données de la base de données vers le contrôleur***, le contrôleur utilise des requêtes pour récupérer les données souhaitées à partir de la base de données. Les données récupérées peuvent ensuite être traitées et utilisées par le contrôleur pour effectuer des actions spécifiques.

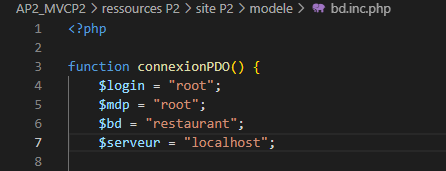
Ici, la récupération des données de la base de données se fait à l'aide de la fonction **getRestos()** qui est appelée dans le contrôleur. Les données récupérées sont stockées dans la variable **$listeRestos**. Ces données sont ensuite utilisées pour générer l'affichage des restaurants dans la vue.

**PARTIE 2 - Contrôleur :**

Vérification de l’intégrité de l'arborescence des fichiers :





Modification des lignes de la fonction ***“connexionPDO()”*** dans le script *modele/bd.inc.php* :

**Question 1 - Contrôleur de connexion : connexion.php**

1.1°) Lors du premier appel à la fonction ***isLoggedOn()***, l'utilisateur n'est pas connecté. Cela est confirmé par le résultat d'exécution du script *authentification.inc.php*, où la première sortie est ***"not logged"*** Par conséquent, lors du premier appel à la fonction, l'utilisateur n'est pas connecté.

1.2°) Lors du deuxième appel à la fonction ***isLoggedOn()***, l'utilisateur est connecté. Cela est confirmé par le résultat d'exécution du script authentification.inc.php, où la deuxième sortie est ***"logged"***. Par conséquent, lors du deuxième appel à la fonction, l'utilisateur est connecté.

1.3°) La fonction qui permet la connexion de l'utilisateur est la fonction ***"login"***. Sa définition (prototype, signature) n'est pas explicitement fournie dans les résultats de recherche actuels. Cependant, dans le contexte donné, la fonction "login" semble être utilisée pour connecter un utilisateur en utilisant un identifiant (adresse e-mail) et un mot de passe.

La fonction "login" est appelée avec les paramètres suivants :

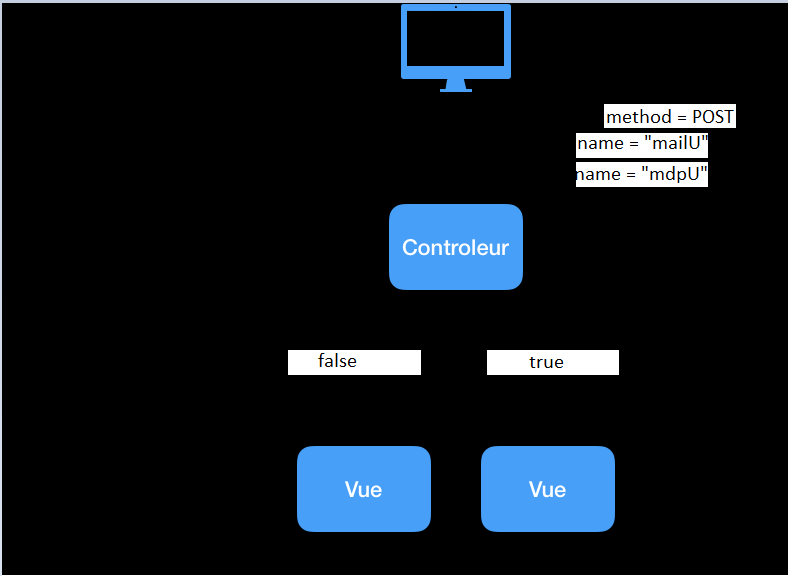
→ L'adresse e-mail de l'utilisateur ;

→ Le mot de passe de l'utilisateur.

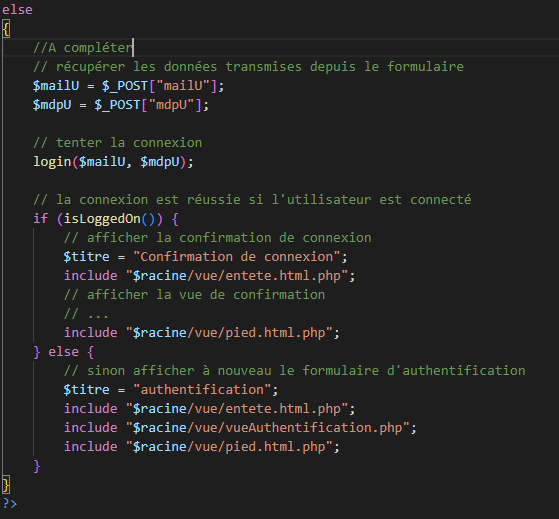
1.4°)

| fonction | rôle |
| --- | --- |
| login() | connecter un utilisateur en utilisant son adresse e-mail et son mot de passe. |
| isLoggedOn() | vérifier si un utilisateur est actuellement connecté au site. Elle retourne un **booléen**, soit true si l'utilisateur est connecté, soit false s'il ne l'est pas. |
| getMailULogged() | récupérer l'adresse e-mail de l'utilisateur connecté. |
| logout() | déconnecter l'utilisateur du site. |

1.5°)



1.6°) Code du contrôleur **connexion.php** compléter en tenant compte des réponses données précédemment :



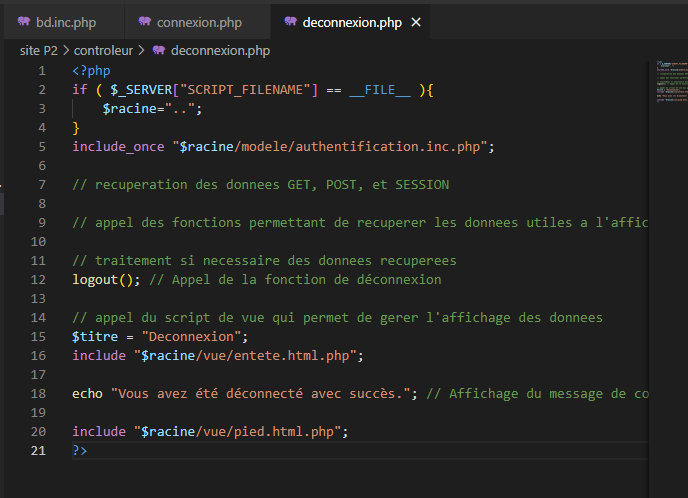
**Question 2 - Contrôleur de déconnexion : deconnexion.php**

2.1°) La fonction du modèle qui permet de déconnecter l'utilisateur actuellement connecté sur le site est la fonction ***logout()***. Cette fonction est utilisée pour détruire la session de l'utilisateur et le déconnecter du site. Lorsque cette fonction est appelée, elle effectue la déconnexion de l'utilisateur et le redirige vers la page de connexion.

2.2°) La vue qui devra être appelée par le contrôleur *deconnexion.php* pour confirmer la déconnexion est la **vue de confirmation de déconnexion**.

2.3°) Lors de la déconnexion, il n'est pas utile de transmettre une donnée au contrôleur.

2.4°) Code du contrôleur ***deconnexion.php*** compléter :



**Question 3 - Analyse et adaptation du contrôleur de recherche : rechercheResto.php**

3.1°) En consultant le script ***bd.resto.inc.php***, les fonctions admettent comme paramètres :

**→ getRestoByIdR** :

Signature : function getRestoByIdR(int $idR)

Paramètre : $idR - un entier représentant l'identifiant du restaurant.

**→ getRestosByNom** :

Signature : function getRestosByNomR(string $nomR)

Paramètre : $nomR - une chaîne de caractères représentant le nom du restaurant.

**→ getRestosByAdresse** :

Signature : function getRestosByAdresse(string $voieAdrR, string $cpR, string $villeR)

Paramètres :

$voieAdrR - une chaîne de caractères représentant la rue de l'adresse du restaurant.

$cpR - une chaîne de caractères représentant le code postal de l'adresse du restaurant.

$villeR - une chaîne de caractères représentant la ville de l'adresse du restaurant.

3.2°) → ***Recherche d'un nom de restaurant*** :

La variable **$\_POST** contiendra les données envoyées par le formulaire de recherche par nom. Si l'utilisateur a saisi un nom de restaurant pour effectuer la recherche, $\_POST ressemblera à ceci :

**$\_POST = array(**

**"nomR" => "le nom du restaurant recherché"**

**);**

Par exemple, si l'utilisateur recherche un restaurant appelé "Chez MBappe", $\_POST['nomR'] contiendra la chaîne de caractères "Chez MBappe".

→ ***Recherche d'un restaurant selon l'adresse suivante***-***rue saint remi 33000 bordeaux:***

Ici, la variable **$\_POST** contiendra les données envoyées par le formulaire de recherche par adresse. Si l'utilisateur a rempli les champs correspondant à l'adresse "rue saint remi 33000 bordeaux", $\_POST ressemblera à ceci :

**$\_POST = array(**

**"voieAdrR" => "rue saint remi",**

**"cpR" => "33000",**

**"villeR" => "bordeaux"**

**);**

**Conclusion :** Ces valeurs correspondent donc respectivement à la rue, au code postal et à la ville entrés dans le formulaire de recherche par adresse.

3.3°) Lors d'une recherche d'un nom de restaurant, la variable **$\_POST** contiendra les données suivantes :

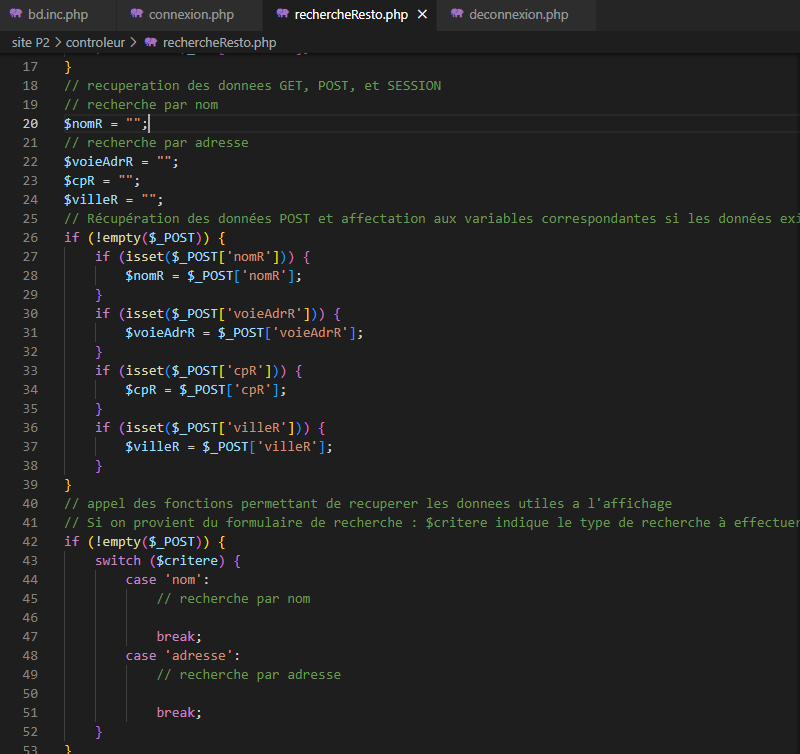
→**nomR** : le nom du restaurant recherché, de type chaîne de caractères.

Pour une recherche d'un restaurant selon l'adresse "rue saint remi 33000 bordeaux", la variable $\_POST contiendra :

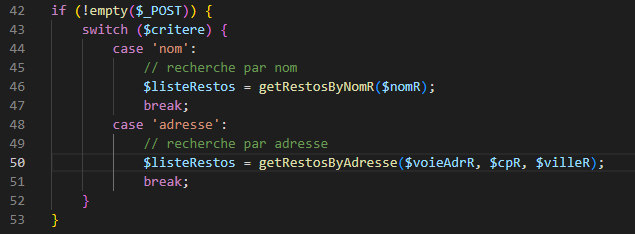
→ **voieAdrR** : la voie de l'adresse, ici "rue saint remi", de type chaîne de caractères.

→ **cpR** : le code postal, ici "33000", de type chaîne de caractères.

→ **villeR** : la ville, ici "bordeaux", de type chaîne de caractères(pas en entier dans notre cas).

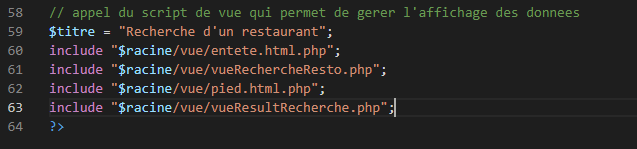
3.4°) Ajout de la section de récupération des données POST dans le contrôleur afin de faire en sorte que les variables $nomR, $voieAdrR, $cpR et $villeR soient valorisées correctement en fonction de la recherche effectuée : 

3.5°) Ajout de code de chaque cas du switch dans le contrôleur en faisant appel à la fonction appropriée du modèle :



3.6°) Les fonctions du modèle sont appelées lorsque la condition “***!empty($\_POST)***” est vérifiée afin de récupérer les données transmises via la méthode POST. Cela garantit que les fonctions du modèle sont appelées uniquement lorsqu'il y a des données à traiter, ce qui optimise les performances et assure que seules les requêtes pertinentes sont exécutées en fonction des actions de l'utilisateur.

3.7°) Ajout dans le contrôleur l'appel à la vue “***vueResultRecherche.php”*** (ligne 63 ici) :



**Question 4 - Analyse de la partie existante du contrôleur rechercheResto.php**

4.1°) Dans le code source de la vue ***vueRechercheResto.php***, la variable qui permet de choisir l'affichage du formulaire de recherche par nom ou par adresse est **$critere**. Cette variable est utilisée pour déterminer quel formulaire de recherche doit être affiché en fonction de sa valeur.

4.2°) La variable transmise en méthode GET au contrôleur qui permet de connaître le type de recherche - par nom ou par adresse - que l'on souhaite effectuer est **critere**. Dans le code du contrôleur, la valeur de cette variable est récupérée à l'aide de ***$\_GET["critere"]*** pour déterminer le type de recherche à effectuer.

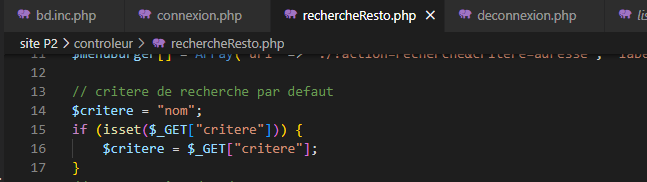
4.3°) La variable transmise en méthode GET au contrôleur qui permet de connaître le type de recherche - par nom ou par adresse - que l'on souhaite effectuer est **critere**. Cette variable est transmise dans l'URL en tant que paramètre GET. Dans le code du contrôleur **rechercheResto.php**, la valeur de cette variable est récupérée à l'aide de ***$\_GET["critere"]*** à la ligne 15 :

**if (isset($\_GET["critere"])) {**

**$critere = $\_GET["critere"];**

**}**

**→** Affichage à l’écran :



4.4°) Sans cette transmission d'information, le contrôleur ne pourrait pas savoir quelle recherche effectuer.

4.5°) Sans cette transmission d'information, le script de vue ne pourrait pas savoir quel formulaire afficher.

4.6°) Lorsqu'une recherche est effectuée, la vue affiche à nouveau le formulaire de recherche avec des valeurs prédéfinies. Les variables utilisées dans ce contexte sont les suivantes :

→ **$nomR** : Utilisée pour afficher la valeur prédéfinie du champ de recherche par nom.

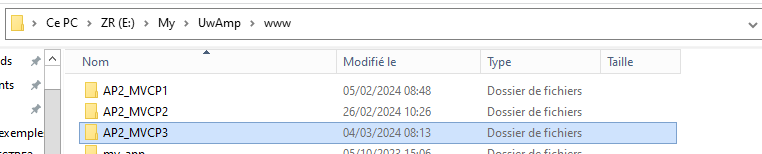
→ **$voieAdrR** : Utilisée pour afficher la valeur prédéfinie du champ de recherche par adresse (rue).

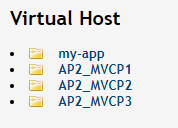
→ **$cpR** : Utilisée pour afficher la valeur prédéfinie du champ de recherche par adresse (code postal).

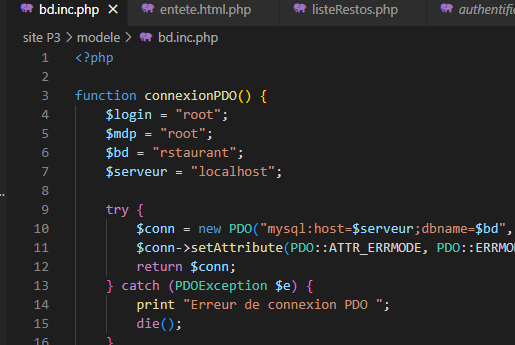
→ **$villeR** : Utilisée pour afficher la valeur prédéfinie du champ de recherche par adresse (ville).

**PARTIE 3 - Vue :**

Vérification de l’intégrité de l'arborescence des fichiers :





Modification des lignes de la fonction ***“connexionPDO()”*** dans le script *modele/bd.inc.php* :

***Partie A : Analyse d'une vue existante***

**Question 1 - Analyse du script de vue vueDetailResto.php : les photos**

1.1°) Les fonctions définies dans le modèle qui sont utilisées dans le contrôleur ***detailResto.php*** sont :

→ **getRestoByIdR($idR)**: Cette fonction permet de récupérer les détails d'un restaurant en fonction de son identifiant. Elle est utilisée pour obtenir les informations sur le restaurant à afficher.

→ **getTypesCuisineByIdR($idR)**: Utilisée pour récupérer les types de cuisine associés à un restaurant en fonction de son identifiant. Ces informations sont utilisées pour afficher les types de cuisine proposés par le restaurant.

→ **getPhotosByIdR($idR)**: Cette fonction est utilisée pour récupérer les photos associées à un restaurant en fonction de son identifiant. Les photos sont ensuite affichées dans la section dédiée.

1.2°) L'affichage du résultat d'exécution des fonctions du modèle trouvées à la question précédente est présenté dans les annexes 5 et 6.

1.3°) Chaque cellule du tableau ***$lesPhotos*** est composée de trois éléments :

**idP**: Identifiant de la photo.

**cheminP**: Chemin de la photo.

**idR**: Identifiant du restaurant auquel la photo est associée.

→ Ces éléments permettent de stocker les informations relatives à chaque photo associée au restaurant, telles que l'identifiant de la photo, son chemin d'accès et l'identifiant du restaurant auquel elle est liée.

1.4°) Dans l'annexe 1 l'instruction PHP permettant d'afficher le nom du fichier de la première photo est la suivante :

***echo $lesPhotos[0]['cheminP'];***

1.5°) L'instruction ***if(count($lesPhotos) > 0)*** vérifie si le nombre d'éléments dans le tableau ***$lesPhotos*** est supérieur à zéro. Son rôle dans le code de la vue est de conditionner l'affichage ou l'exécution de certaines instructions en fonction de la présence ou de l'absence de photos dans le tableau.

Ainsi, si le tableau contient au moins une photo, les instructions à l'intérieur du bloc if seront exécutées, sinon, elles seront ignorées.

1.6°) La syntaxe générale permettant d'accéder à un champ **(idP, CheminP ou idR)** contenu dans la variable ***$lesPhotos*** en PHP est la suivante :

**$lesPhotos[0]['idP'];** → Pour accéder à l'idP de la première photo

**$lesPhotos[0]['CheminP'];** → Pour accéder au CheminP de la première photo

**$lesPhotos[0]['idR'];** → Pour accéder à l'idR de la première photo

**Question 2 - vueDetailResto.php : les types de cuisine**

2.1°) La variable **$lesTypesCuisine** est produite par la fonction **getTypesCuisineByIdR** du modèle **bd.typecuisine.inc.php**.

2.2°) La variable **$lesTypesCuisine** est produite par la fonction ***getTypesCuisineByIdR*** du modèle ***bd.typecuisine.inc.php***. Cette fonction effectue une requête pour récupérer les types de cuisine associés à un restaurant spécifique en utilisant l'identifiant du restaurant.

Le résultat d'exécution de ***bd.typecuisine.inc.php*** montre un exemple de sortie similaire à celle attendue pour la variable **$lesTypesCuisine**. En utilisant la fonction ***getTypesCuisineByIdR*** avec l'identifiant du restaurant, on obtient un tableau associatif contenant les types de cuisine disponibles pour ce restaurant, avec chaque élément du tableau représentant un type de cuisine et ses attributs associés.

2.3°) La variable **$lesTypesCuisine** contient trois types de cuisine, comme indiqué dans l'extrait du résultat d'exécution de *bd.typecuisine.inc.php* :

→ Type de cuisine avec l'identifiant 1 et libellé sud ouest.

→ Type de cuisine avec l'identifiant 8 et libellé sandwich.

→ Type de cuisine avec l'identifiant 11 et libellé grillade.

En résumé, la variable **$lesTypesCuisine** contient trois types de cuisine différents.

2.4°) Pour accéder au libellé d'un type de cuisine contenu dans la variable **$lesTypesCuisine**, on peut utiliser la syntaxe suivante en PHP :

→ **$lesTypesCuisine[$indice]["libelleTC"];**

**Question 3 : vueDetailResto.php - le restaurant**

3.1°) La variable **$unResto** est produite par le contrôleur ***detailResto.php*** et la fonction associée est ***getRestoByIdR*** du modèle *bd.resto.inc.php*.

3.2°) La variable **$unResto** contient les détails d'un restaurant spécifique. Voici un schéma du contenu de la variable **$unResto** :

**idR** : 1

**nomR** : l'entrepote

**numAdrR** : 2

**voieAdrR** : rue Maurice Ravel

**cpR** : 33000

**villeR** : Bordeaux

**latitudeDegR** : 44.7948

**longitudeDegR** : -0.58754

**descR** : description

**horairesR** : <table>...</table>

3.3°) Les sections de code dans la vue qui utilisent la variable ***$unResto*** sont les suivantes :

La section qui affiche le nom du restaurant :

**<h1><?= $unResto['nomR']; ?></h1>**

La section qui affiche la description du restaurant :

**<?= $unResto['descR']; ?>**

La section qui affiche l'adresse du restaurant :

**<p>**

**<?= $unResto['numAdrR']; ?>**

**<?= $unResto['voieAdrR']; ?><br />**

**<?= $unResto['cpR']; ?>**

**<?= $unResto['villeR']; ?>**

**</p>**

La section qui affiche les horaires du restaurant :

**<?= $unResto['horairesR']; ?>**

3.4°) Cette variable contient 1 restaurant.

3.5°) Pour accéder au nom d'un restaurant contenu dans la variable **$unResto**, on peut utiliser la syntaxe suivante en PHP :

**$unResto['nomR']**

***Partie B : Adaptation de vues***

**Question 4 : Adaptation du menu général**

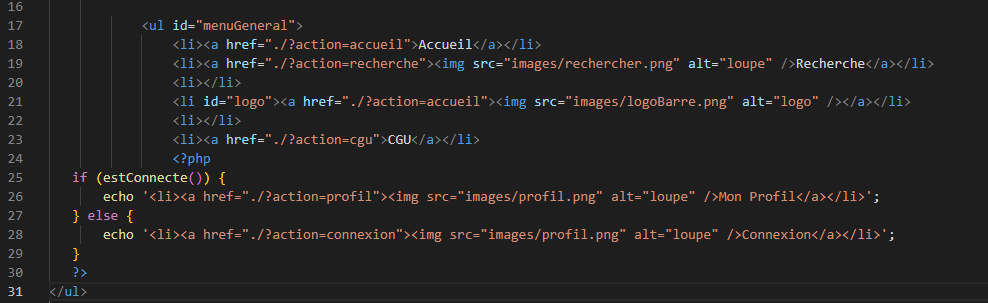
4.1°) Pour connaître l'état de connexion du visiteur du site, la fonction du modèle *authentification.inc.php* qui permet d'exécuter cela est la fonction **estConnecte()**. Cette fonction vérifie si l'utilisateur est connecté en renvoyant vrai ***(true)*** si l'utilisateur est connecté, et faux ***(false)*** s'il ne l'est pas.

4.2°) Comme vu précédemment, la fonction **estConnecte()** renvoie un type de donnée booléen, c'est-à-dire qu'elle renvoie soit true (vrai) si l'utilisateur est connecté, soit false (faux) s'il ne l'est pas.

Par exemple, en Python, la fonction **return** est utilisée pour renvoyer une valeur à partir d'une fonction, elle peut être ***True ou False*** en fonction des cas d’utilisations.

4.3°) L'élément de la liste du menu général qui est affiché lorsque l'utilisateur n'est pas connecté (comportement par défaut), il s'agit de ***l'option "Connexion"*** qui est affichée par défaut.

4.4°) Pour adapter le code de la vue entete.html.php en utilisant la fonction ***estConnecte()*** afin de faire en sorte que l'option "Mon Profil" soit affichée lorsque l'utilisateur est connecté et que l'option "Connexion" soit affichée lorsque aucun utilisateur n’est connecté, on procède de la manière suivante :



→ **Résultat de connexion :**



**Question 5 : Profil utilisateur - affichage des types de cuisine préférés par l'utilisateur**

5.1°) Pour afficher la variable **$mesTypesCuisinesAimes** dans le contrôleur à l’aide de la fonction *print\_r()*, on utilise le code suivant :

**<?php**

**// Affichage de la variable $mesTypeCuisineAimes à l'aide de la fonction print\_r()**

**print\_r($mesTypeCuisineAimes);**

**?>**

→ La structure de la variable **$mesTypeCuisineAimes** dépendra de son contenu spécifique. Si **$mesTypeCuisineAimes** est un tableau, la sortie de la fonction *print\_r()* affichera les clés et les valeurs du tableau.

5.2°) Pour atteindre le libellé d'un type de cuisine contenu dans la variable ***$mesTypeCuisineAimes***, on peut utiliser la syntaxe suivante en PHP :

→ **$libelleTypeCuisine = $mesTypeCuisineAimes['libelle'];**

5.3°) Dans le code HTML fourni, la partie qui affiche les types de cuisine se trouve à l'intérieur de la balise ***<ul>*** avec l'identifiant "tagFood". Voici le code correspondant :

**<ul id="tagFood">**

**<li class="tag"><span class="tag">#</span>sud ouest</li>**

**<li class="tag"><span class="tag">#</span>viande</li>**

**<li class="tag"><span class="tag">#</span>grillade</li>**

**</ul>**

→ Dans ce code, à l'intérieur de chaque élément de liste, il y a un span avec la classe "tag" qui contient le symbole "#" suivi du libellé du type de cuisine.

5.4°) Voici la partie répétée du code :

**<li class="tag"><span class="tag">#</span>sud ouest</li>**

**<li class="tag"><span class="tag">#</span>viande</li>**

**<li class="tag"><span class="tag">#</span>grillade</li>**

→ Chaque ligne représente un type de cuisine avec une classe ***"tag"*** et un libellé spécifique.

5.5°) Pour parcourir la variable **$mesTypeCuisineAimes** et afficher pour chaque type le libellé associé, on utilise la **boucle foreach**. Voici le code correspondant :

**<?php**

**foreach ($mesTypeCuisineAimes as $typeCuisine) {**

**echo $typeCuisine['libelle'] . "<br>";**

**}**

**?>**

5.6°) Modification du code produit à la question précédente afin de faire en sorte que chaque type de cuisine soit affiché en respectant en respectant les balises HTML trouvées à la question 5.3 :



**Question 6 : Profil utilisateur - affichage des restaurants aimés par l'utilisateur**

6.1°) Pour accéder dans la variable **$mesRestosAimes** à l’identifiant et au nom de chaque restaurant, on utilise une boucle foreach pour parcourir chaque élément de la variable **$mesRestosAimes**, contenant donc les balises **<a>** représentant les restaurants aimés par l'utilisateur.

6.2°) Le code permettant de parcourir la variable $mesRestosAimes et d’afficher pour chaque restaurant le nom associé est le suivant :

**<?php**

**foreach ($mesRestosAimes as $restaurant)**

**$nomRestaurant =substr($restaurant,strpos($restaurant,"\">") + 2, strpos($restaurant, "</a>") - (strpos($restaurant, "\">") + 2));**

**echo $nomRestaurant . "<br>";**

**}**

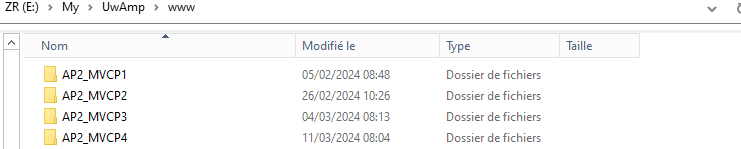
**?>**

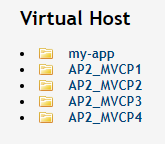
6.3°) Même code mais en spécifiant à chaque fois le type de cuisine (question similaire 5.6).

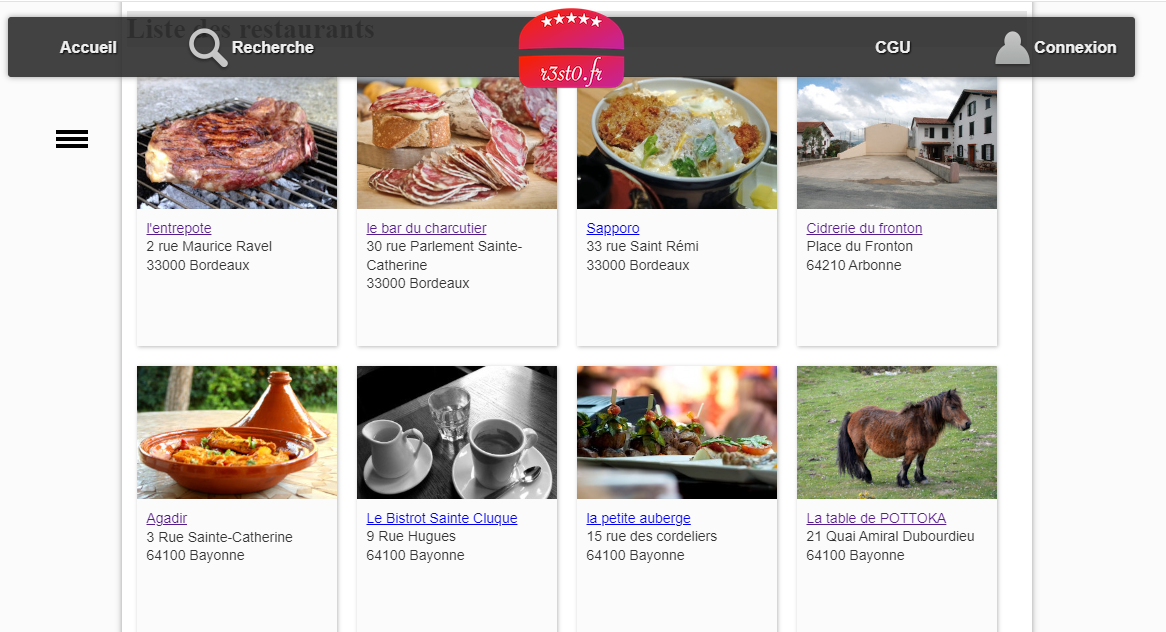
**PARTIE 4 - Modèle :**

**Fonctionnement du modèle dans le patron de conception MVC**

Vérification de l’intégrité de l'arborescence des fichiers :







Modification des lignes de la fonction ***“connexionPDO()”*** dans le script *modele/bd.inc.php* :



***A - Analyse du fonctionnement du modèle***

**Question 1 - Analyse de la fonction getRestoByIdR() du modèle bd.resto.inc.php**

1.1°) Dans l'exemple d'utilisation de l'annexe 4, la fonction **getRestoByIdR()** est appelée avec le paramètre 1. Cela signifie que l'identifiant du restaurant passé en paramètre est 1.

1.2°) La requête SQL correspondante est ***"select \* from resto where idR=1"*** et le résultat obtenu est un tableau associatif contenant les informations du restaurant avec l'identifiant 1 .

1.3°) La requête SQL ne retourne qu’une seule ligne car la condition **“WHERE”** filtre les résultats des colonnes de la table "resto" où la colonne "idR" est égale à 1, en l’occurrence ici il n’y a qu’un seul résultat, par conséquent 1 seule ligne.

1.4°) La syntaxe utilisée pour accéder à un champ d’un tableau associatif est la suivante :

**$NomTableau = [‘NomChamp’];**

1.5°) Pour accéder au nom du restaurant dans la variable retournée par la fonction **getRestoByIdR()** :

**$resultat['nomR'];**

**Question 2 - Analyse de la fonction getRestosByNomR()**

2.1°) La requête SQL envoyée à la fonction **prepare()** dans **getRestosByNomR()** est la suivante :

**"select \* from resto where nomR like :nomR"**

2.2°) La requête SQL exécutée après l’appel à **bindValue()** est :

**select \* from resto where nomR like '%charcut%'**

2.3°) Cette requête est susceptible de retourner plusieurs lignes car elle utilise la condition **“LIKE”** avec comme caractères spécifiques **“%”**. Cela retournera donc toutes les lignes respectant cette condition en fonction du mot **“charcut”**.

2.4°) La proposition correcte est la proposition B : **la variable *$resultat* retournée par la fonction *getRestosByNomR()* est un tableau dont chaque case est un tableau associatif. Cette variable peut contenir les informations sur plusieurs restaurants.**

→ En effet, l’usage de case vide comme le montre la ligne **“$resultat[] = $ligne;”** et l’usage la boucle while qui la précède  **“while ($ligne)”** font que cette variable peut retourner plusieurs tableaux associatifs en fonction du nombre de case défini faisant qu’ainsi, plusieurs informations peuvent être retournés sur plusieurs restaurants.

**Question 3 - Analyse de la fonction getNoteMoyenneByIdR() du modèle bd.critiquer.inc.php**

3.1°) La requête SQL présentée dans la fonction **getNoteMoyenneByIdR()** a pour rôle de calculer la note moyenne attribuée à un restaurant spécifique.

3.2°) 1 seul résultat est attendu de la requête SQL la moyenne calculée en fonction des critiques. L’utilisation de la fonction **AVG()** calcule ici la moyenne des notes dans la table "critiquer" pour le restaurant identifié par son ID.

3.3°)

Script : 

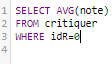
→ Résultats (ici surligné) :



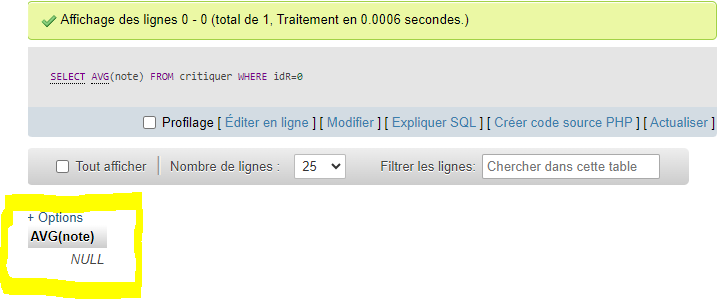
La colonne affichée dans le résultat se nomme donc ***AVG(note)***.

3.4°)

Script (Test avec 0) :



→ Résultats (ici surligné) :



La valeur obtenue est ***NULL***, ce qui signifie donc une absence de valeur. Que ce soit 0 et -1, la valeur renvoyée sera nulle.

3.5°) Lorsque l'identifiant d'un restaurant passé en paramètre existe ou n'existe pas dans la base de données, la fonction retournera soit la note moyenne calculée à partir des critiques, soit 0 en l'absence de critiques ou d'un restaurant.

3.6°) Cette condition permet à la fonction de retourner la moyenne des notes si des critiques existent pour le restaurant :

→ **if ($resultat["avg(note)"] != NULL) {**

**return $resultat["avg(note)"];**

Sinon elle retourne 0 en l'absence de critiques :

→ **} else {**

**return 0;**

**}**

**Question 4 - insertion de données, fonction addAimer() du modèle bd.aimer.inc.php**

4.1°) La fonction **addAimer()** ajoute une nouvelle entrée dans la table *"aimer"* pour indiquer qu'un utilisateur spécifique aime un restaurant spécifique. Cette fonction retourne TRUE si l'opération est réussie et FALSE dans le cas contraire.

4.2°) La méthode **bindValue()** de PDO permet de lier une valeur à un paramètre dans une requête SQL préparée. Le 3ème paramètre de cette méthode est utilisé pour spécifier le type de données du paramètre.

4.3°) A l’aide de l’annexe 11, la valeur retournée par la fonction **execute()** en cas de réussite est *TRUE*, en cas d’échec *FALSE*.

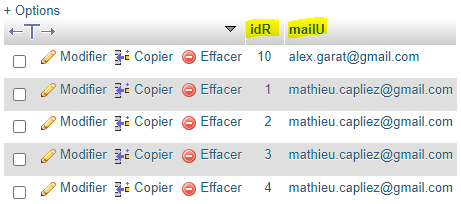
4.4°) La fonction **addAimer()** retourne donc un booléen.

4.5°) Si un utilisateur tente d'aimer un restaurant qu'il aime déjà, la valeur retournée par la fonction **addAimer()** est **FALSE**.

***B - Ajout de fonctionnalités au modèle***

**Question 5 - mise en place de la fonction delAimer()**

5.1°) La clé primaire de la table aimer est composé des colonnes **idR** et **mailU** :



5.2°) La syntaxe de la requête de suppression en SQL est la suivante :

→ **DELETE**

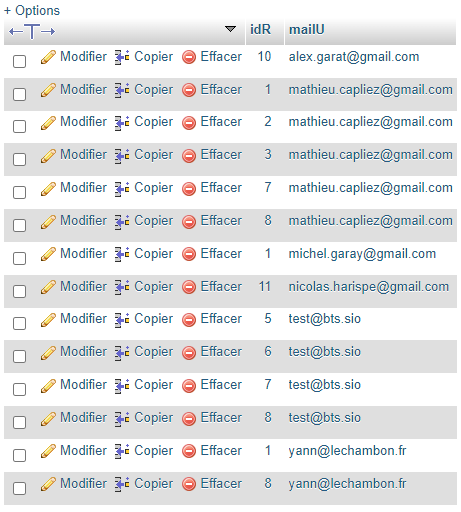
Dans notre cas, elle peut-être utilisé comme dans cet exemple :

→ **DELETE FROM NomTable WHERE condition;**

5.3° et 5.4°) Requête SQL permettant de supprimer une occurrence précise de la table aimer :

→ **DELETE FROM aimer where idR = 4;**

**Résultats :**



→ Dans cet exemple, on supprime l'occurrence de la table "aimer" où la valeur de la colonne "idR" est égale à 4. Par conséquent, toutes les colonnes avec comme idR ont été supprimées.

5.5°) La fonction **delAimer()** devrait être placée dans un script où la suppression d'une occurrence de la table "aimer" est nécessaire.

5.6°) Pour supprimer précisément une occurrence de la base de données à l'aide de la fonction **delAimer()**, les paramètres à transmettre à cette fonction sontl'identifiant unique de l'occurrence à supprimer.

→ Voici un prototype de la fonction :

**function delAimer(idOccurrence) {**

**}**

5.7°) Voici le code :

**function delAimer(idOccurrence) {**

**……**

**DELETE FROM aimer WHERE idOccurrence = “ “;**

**}**

**Question 6 - consultation des critiques - notes et commentaires - d'un restaurant.**

6.1°)Voici le prototype de la fonction **getCritiquerByIdR()** :

**function getCritiquerByIdR(idR) {**

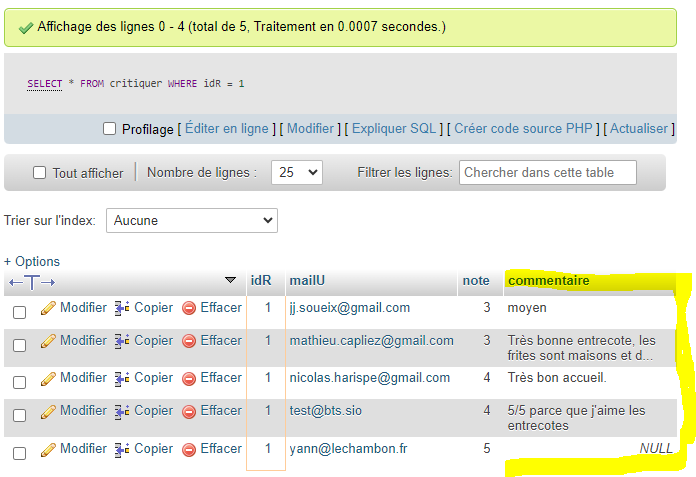
**……**

**SELECT \* FROM critiquer WHERE idR = “ “;**

**}**

6.2°) Requête SQL permettant de récupérer les critiques associées à un restaurant *(ici on prend comme exemple les restaurants avec comme identifiant 1)* :



**→ Résultats :** 

6.3°) La structure de données retournée par cette fonction serait une liste contenant les critiques associées à l'identifiant spécifique du restaurant.

La structure de données est composé des éléments suivants, correspondants donc à la structure de la table "critiquer" :

**idR** : L'identifiant du restaurant auquel la critique est associée.

**mailU** : L'adresse e-mail de l'utilisateur ayant émis la critique.

**note** : La note attribuée à ce restaurant dans la critique.

**commentaire** : Le commentaire/critique.

6.4°) Voici le code :

**function getCritiquerByIdR($idRestaurant) {**

**……**

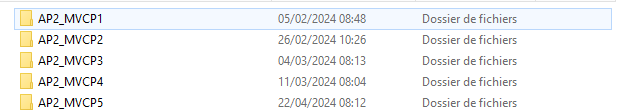
**SELECT \* FROM critiquer WHERE idR = $idRestaurant";**

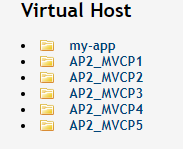
**}**

**PARTIE 5 - Contrôleur principal :**

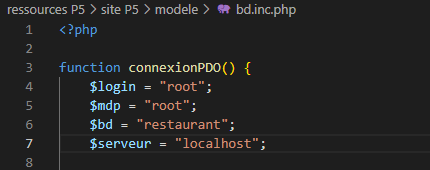
**Fonctionnement du contrôleur principal dans le patron de conception MVC**

Vérification de l’intégrité de l'arborescence des fichiers :





Modification des lignes de la fonction ***“connexionPDO()”*** dans le script *modele/bd.inc.php* :



***A - Analyse du fonctionnement du contrôleur principal***

**Question 1 - Analyse de l'existant**

1.1°)

| Ecran | URL | Paramètre | Valeur |
| --- | --- | --- | --- |
| connexion | http://localhost/AP2\_MVCP5/ressources%20P5/site%20P5/?action=connexion | Action | connexion |
| recherche | http://localhost/AP2\_MVCP5/ressources%20P5/site%20P5/?action=recherche | Action | recherche |
| accueil | http://localhost/AP2\_MVCP5/ressources%20P5/site%20P5/?action=accueil | Action | accueil |

1.2°) D'après l'annexe 1 et la réponse à la question précédente, la variable $action peut prendre les valeurs suivantes :

**→ Pour l'écran de connexion : *connexion***

**→ Pour l'écran de recherche : *recherche***

**→ Pour l'écran d'accueil : *accueil***

1.3°) Schéma de la structure de la variable **$lesActions** créée dans la fonction ***controleurPrincipal()***.

connexion → [action=connexion] → fichier\_connexion.php

recherche → [action=recherche] → fichier\_recherche.php

accueil → [action=accueil] → fichier\_accueil.php

1.4°) ***La correspondance*** entre les valeurs des actions dans les URL et les clés de la variable **$lesActions** est la partie similaire entre les deux structures.

1.5°) Les valeurs qui peuvent être transmises à la fonction **controleurPrincipal()** sont :

→ defaut

→ liste

→ detail

→ recherche

→ connexion

→ deconnexion

→ profil

Ces valeurs proviennent des clés du tableau ***$lesActions*** définis dans l’annexe 2 - **controleurPrincipal().**

1.6) Pour defaut, la valeur retournée sera **listeRestos.php**.

→ Pour liste, la valeur retournée sera **listeRestos.php**.

→ Pour detail, la valeur retournée sera **detailResto.php**.

→ Pour recherche, la valeur retournée sera **rechercheResto.php**.

→ Pour connexion, la valeur retournée sera **connexion.php**.

→ Pour deconnexion, la valeur retournée sera **deconnexion.php**.

→ Et enfin pour profil, la valeur retournée sera **monProfil.php**.

1.7°) Si l'action transmise à la fonction **controleurPrincipal()** existe dans la variable **$lesActions**, la fonction retournera le fichier associé à cette action.

En revanche, si l'action n'existe pas dans **$lesActions**, la fonction retournera la valeur par défaut "*listeRestos.php*". Il n'y aura pas d'erreur car la fonction array\_key\_exists() retournera simplement FALSE si la clé n'est pas trouvée dans le tableau, conformément à la documentation PHP.

1.8°) La condition avec **array\_key\_exists()** vérifie si l'action transmise est une clé valide dans le tableau ***$lesActions***.

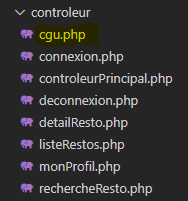
1.9°) Le script contrôleur appelé par **index.php** lorsque la variable GET action n'est pas renseignée rappelle la clé *defaut*, et donc ***“listeRestos.php***”..

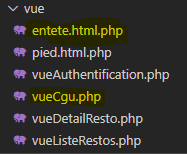
1.10°) Lorsque la variable GET action contient le mot clé "*liste*", le même script contrôleur "***listeRestos.php***" est appelé par **index.php**.

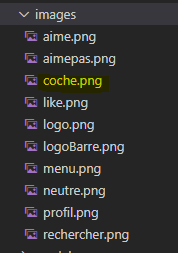
***B - Intégration de contrôleurs pré-existants***

***Question 2 - CGU***

2.1°) Dans le menu général, l'action correspondant au contrôleur chargé de l'affichage des CGU est celle associée au lien **<a href="./?action=cgu">CGU</a>**.

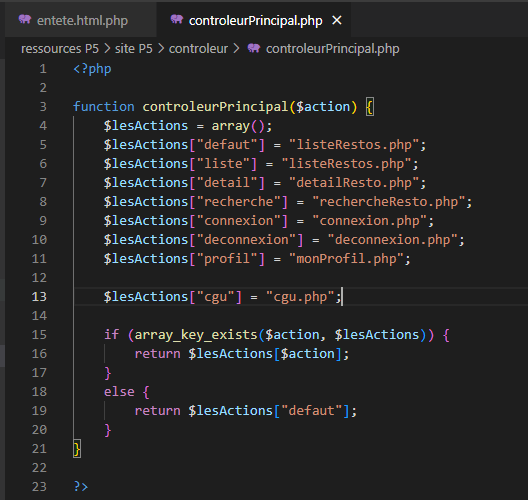
2.2°) cgu.php → **controleur** :

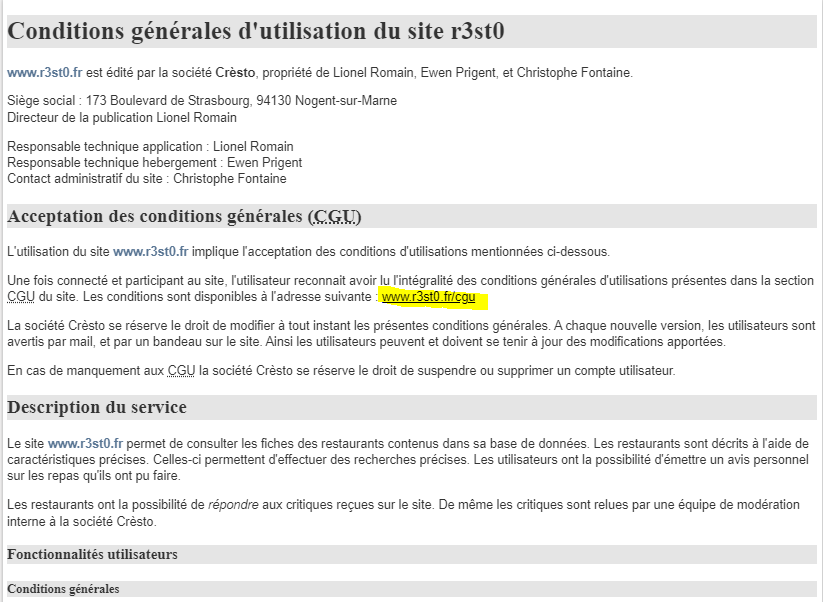
vueCgu.php & entete.html.php → **vue** :

coche.png → **images** :

2.3°) L’instruction à ajouter à la fonction **controleurPrincipal()** pour ajouter la nouvelle action dans la variable ***$lesActions*** est la suivante :

**$lesActions["nom\_de\_l\_action"] = "nom\_du\_fichier.php";**

2.4°) Voici la nouvelle action à la fonction alors situé dans le **controleurPrincipal()** : 



**Question 3 - aimer un restaurant**

3.1°) La vue qui permet d’afficher la fiche descriptive d’un restaurant est la vue **“vueDetailResto.php”** .

3.2°) Le code de la vue correspondant au lien reliant l’étoile se situe dans la vue **vueDetailResto.php**, voici son code :

**<?php if ($aimer != false) { ?>**

**<a href="./?action=aimer&idR=<?= $unResto['idR']; ?>" ><img class="aimer" src="images/aime.png" alt="j'aime ce restaurant"></a>**

**<?php } else { ?>**

**<a href="./?action=aimer&idR=<?= $unResto['idR']; ?>" ><img class="aimer" src="images/aimepas.png" alt="je n'aime pas encore ce restaurant"></a>**

**<?php } ?>**

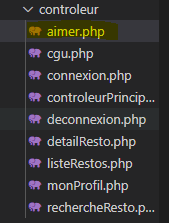
3.3°) Les paramètres envoyés en méthode GET lorsque l'on clique sur le lien sont :

nom : **action** || valeur : ***aimer***

nom : **idR** || valeur : ***<?= $unResto['idR']; ?>***

3.4°) Le script contrôleur ***"aimer.php"*** gère l'action d'aimer ou de ne pas aimer un restaurant. Lorsqu'un utilisateur clique sur le lien pour aimer ou ne pas aimer un restaurant, deux paramètres sont transmis en méthode GET : ***"action" et "idR"***. La variable "action" est utilisée pour déterminer l'action à effectuer, tandis que la variable "idR" représente l'identifiant unique du restaurant concerné.

→ Ces paramètres sont utilisés dans le script pour ajouter ou supprimer le restaurant des favoris de l'utilisateur dans la base de données.

3.5°) Le fichier **"aimer.php"** doit être placé dans le répertoire contrôleur, comme suit : 

3.6°) Pour ajouter la nouvelle action dans la variable **$lesActions**, il suffit de faire le code suivant :

**function controleurPrincipal($action) {**

**$lesActions = array(**

**"accueil" => "afficherAccueil",**

**"recherche" => "afficherRecherche",**

**"cgu" => "afficherCGU",**

**);**

**if (array\_key\_exists($action, $lesActions)) {**

**$fonction = $lesActions[$action];**

**$fonction();**

**} else {**

**echo "Action non valide";**

**}**

**}**

3.7°) *Code à effectuer* :

**case 'aimer':**

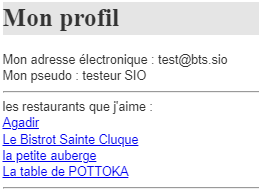
**include\_once 'controllers/aimer.php';**

**controleurAimer();**

**break;**

***→ Résultat :***

**AVANT :**



**APRÈS :**





3.8°) Non, le contrôleur ***`aimer.php`*** gère les actions liées aux appréciations des utilisateurs pour les restaurants sans faire appel à une vue directement.

3.9°) → Le terme ***"referer"*** (référent en français) désigne l'adresse URL de la page précédemment visitée par l'utilisateur, qui a conduit à la page actuelle via un lien hypertexte.

3.10°) La variable ***$\_SERVER['HTTP\_REFERER']*** contient l'URL de la page précédemment visitée par l'utilisateur, qui a conduit à la page actuelle via un lien hypertexte.

3.11°) L'instruction **`header('Location:' $\_SERVER['HTTP\_REFERER']);`** redirige l'utilisateur vers la page précédente qu'il a visitée. Cela permet généralement de renvoyer l'utilisateur à la page d'où il vient après une action effectuée sur la page actuelle.

**Question 4 - inscription**

4.1°) *Voici comment procéder* :

→ **Modifier le lien dans la section "connexion" pour rediriger vers la page d'inscription** :

**<li><a href="./?action=inscription">Inscription</a></li>**

**→ Ajouter une nouvelle action dans le contrôleur pour gérer l'inscription :**

**$lesActions["inscription"] = "inscription.php";**