

Architecture MVC en PHP - Contrôleur

Partie 1 : généralités

Partie 2 : contrôleur

Partie 3 : vue

Partie 4 : modèle

Partie 5 : contrôleur principal et routage

Fonctionnement du contrôleur dans le patron de conception MVC

Rappel du contexte

R3st0.fr est un site web de critique de restaurant. À l'image des sites de ce type il a pour vocation le recensement des avis des consommateurs et la diffusion de ces avis aux visiteurs.

Ce site web est développé en PHP en suivant le patron de conception "modèle vue contrôleur" (pattern MVC). L'architecture retenue pour ce projet permet d'appréhender la programmation web d'une manière structurée.

L'objectif de ce document est d'analyser le fonctionnement du contrôleur, son lien avec les autres composants de l'architecture MVC puis de mettre à jour des contrôleurs pour implémenter ou compléter des fonctionnalités.

Ressources à utiliser

- Dossier "base de données" : fichier base.sql contenant les tables et les données de la base utilisée par l'application web étudiée.
- Dossier site : arborescence et fichiers de l'application web.

Ressources à utiliser téléchargeable

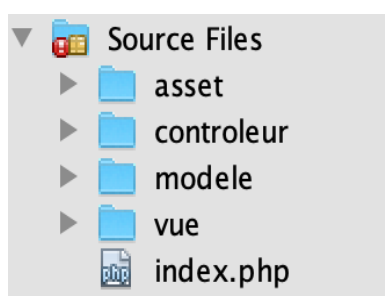
Dossier "base de données" : fichier base.sql contenant les tables et les données de la base utilisée par l'application web étudiée.

- Dossier site : arborescence et fichiers de l'application web.

Préparations *[ne pas recréer / importer une base de données si elle a été créée précédemment]*

Après avoir créé la base de données nommée « resto » (encodage utf8mb4) et importé le fichier base.sql sur PhpMyAdmin, créer un nouveau dossier nommé **MVCTP2** à la racine de votre serveur web. Copier le contenu du dossier « site » dans ce dossier. Importer le dossier **MVCTP2** dans Visual Studio Code.

L'arborescence devrait être semblable celle-ci :

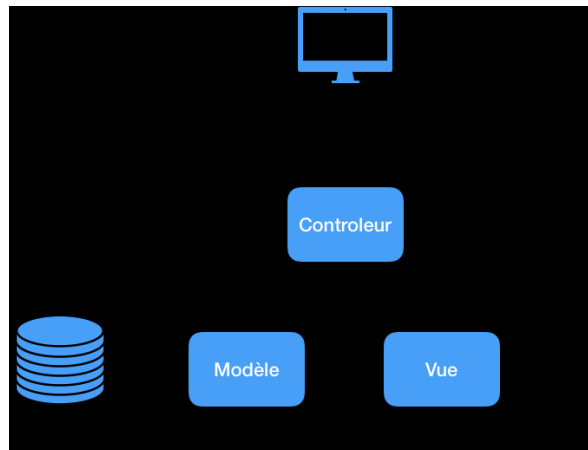


Avant de commencer le TP, le site doit être paramétré afin qu'il utilise votre base de données. Dans le script modele/**bd.inc.php**, modifier les lignes suivantes afin d'indiquer les bonnes informations :

```
$login = "votre login mariaDB";  
$mdp = "votre mot de passe mariaDB";  
$bd = "nom de votre base de données";  
$serveur = "localhost"; //le SGBRD est installé sur le serveur
```

Afin de mieux voir l'objectif du site, et les différentes fonctionnalités que vous devrez mettre en place tout au long de ces TP, le site final est consultable à [cette adresse](#).

Rappel sur les contrôleurs



Comme vu au TP précédent, un contrôleur est l'élément central d'une fonctionnalité sur un site web MVC. Chaque fonctionnalité est gérée par un contrôleur. C'est le contrôleur qui a pour rôle de :

- récupérer les données transmises par un formulaire,
- récupérer ou envoyer les données dans la base en faisant appel aux fonctions gérées par le modèle,
- traiter les données,
- appeler les vues permettant d'afficher les données récupérées, calculées, ou d'afficher les messages à destination de l'utilisateur.

Contexte

Le site r3st0.fr est un site collaboratif, les utilisateurs connectés peuvent émettre des critiques sur les restaurants et les noter. Ils peuvent aussi indiquer quel sont leurs types de cuisine préférés, ajouter des restaurants en favoris, etc...

Le site doit ainsi proposer un système d'authentification. Dans les exercices suivants, vous devrez mettre en place les contrôleurs de connexion, de déconnexion et de recherche en utilisant les vues et les modèles déjà écrits.

Question 1 - Contrôleur de connexion : connexion.php

Documents à utiliser

- fichiers fournis en ressources
- annexes (**en bas de document**) 1, 2, 3, 4

Le contrôleur permettant de gérer la connexion d'un utilisateur est utilisé dans 2 cas.

1. Après avoir cliqué sur le lien "connexion" du menu principal, l'utilisateur doit saisir son identifiant et son mot de passe
2. L'utilisateur a saisi son login et son mot de passe dans le formulaire de connexion, l'a validé et ces deux informations sont transmises au contrôleur de connexion.

1er cas :

Le contrôleur est appelé sans paramètre, il ne peut alors pas établir la connexion et doit afficher la vue proposant le formulaire de connexion.

2eme cas :

Le contrôleur est appelé avec les paramètres login et mot de passe. Il peut alors tenter la connexion. Si la connexion se passe bien, un écran de confirmation est affiché. Dans le cas contraire, le formulaire de connexion est de nouveau proposé à l'utilisateur.

Pour mettre en place ce traitement, deux scripts de la couche Modèle seront utilisés :

- authentication.inc.php
- bd.utilisateur.inc.php

Les Vues sont aussi disponibles en ressources :

- vueAuthentification.php : formulaire de connexion,
- vueConfirmationAuth.php : confirmation de connexion lorsque l'utilisateur a pu être authentifié.

Remarque

Les vues `entete.html.php` et `pied.html.php` sont toujours incluses afin de garantir l'affichage de la structure fixe du site (menus, design etc...).

L'étude des scripts de modèle et de vues sera abordée plus tard dans ce cours.

Observation du modèle d'authentification

à l'aide des annexes 1 et 2

L'annexe 1 est la section de test du fichier `authentication.inc.php`. Cette section montre l'endroit où sont exécutées et testées les fonctions définies dans ce fichier.

L'annexe 2 correspond à l'affichage obtenu lors de l'exécution de l'annexe 1.

1.1. La fonction `isLoggedOn()` retourne un booléen, soit `true`, soit `false` selon que l'utilisateur est connecté ou pas sur le site.

Lors de l'appel à la fonction `isLoggedOn()` l'utilisateur est-il connecté par défaut ?

1.2. Quel est le mécanisme sur un navigateur qui rend à `TRUE` la fonction `isLoggedOn()` ?

1.3. Quelle fonction permet la connexion de l'utilisateur ? Quelle est sa définition (*prototype, signature*) ?

1.4. À l'aide des questions précédentes et en observant la section de test et le résultat d'exécution du script `authentication.inc.php`, indiquer le rôle des fonctions suivantes :

fonction	rôle
<code>login()</code>	
<code>isLoggedOn()</code>	
<code>getMailULogged()</code>	
<code>logout()</code>	

1.5. À l'aide des annexes 2 et 3 compléter le schéma ci-dessous en indiquant :

- la méthode utilisée pour transmettre les données au contrôleur depuis le formulaire de connexion,
- le nom des indices du tableau associatif qui seront utilisés pour récupérer les données saisies dans le formulaire,
- la valeur retournée par la fonction `isLoggedOn()` orientant l'utilisateur soit sur la vue d'authentification affichant le formulaire de connexion, soit sur la vue de confirmation indiquant que l'utilisateur est bien connecté.

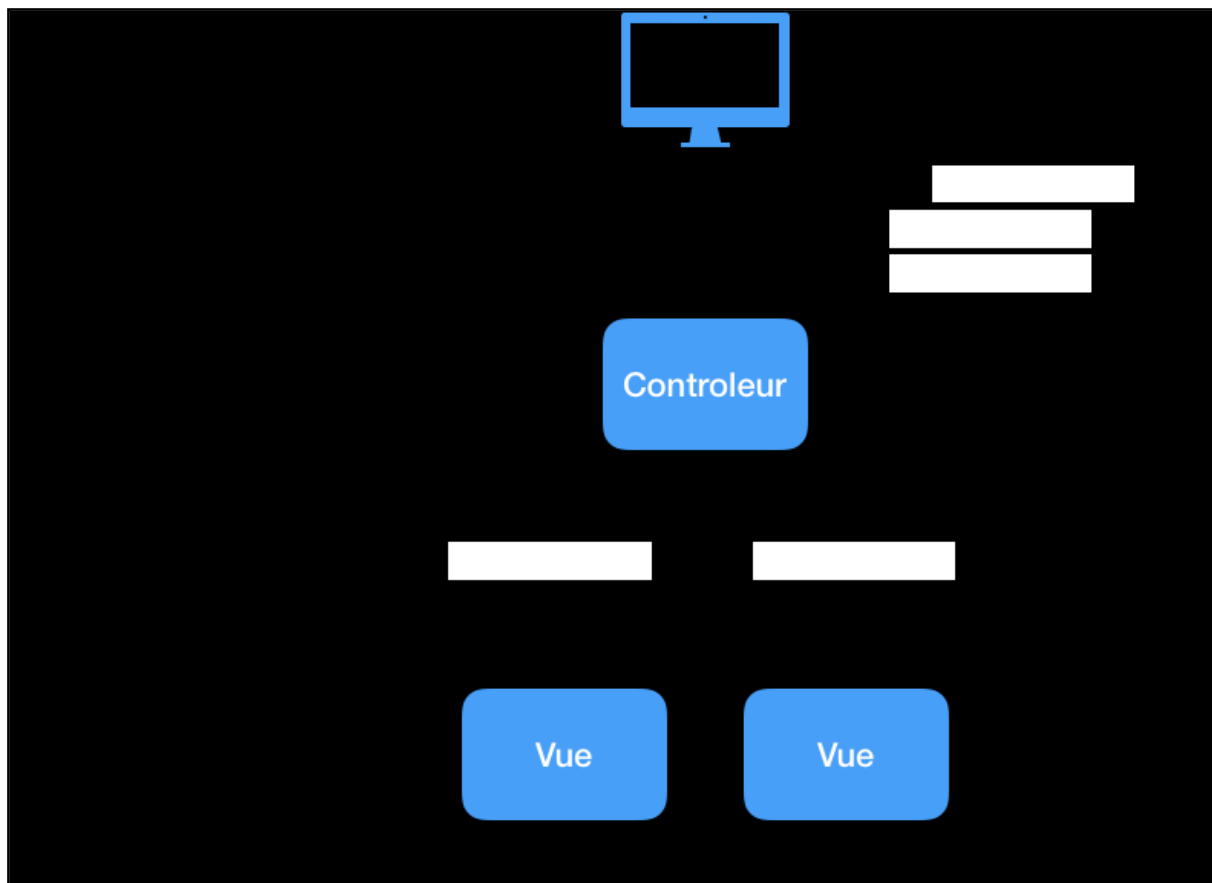


Schéma d'authentification

Remarque :

Ne pas tenir compte de la manière dont est utilisé le champ action dans la balise form du formulaire. Cette méthode spécifique au MVC sera étudiée dans l'un des cours suivants.

Répondre à la question suivante en s'inspirant de la section de test du fichier `authentification.inc.php` présent en annexe 1.

1.6. Compléter le code du contrôleur `connexion.php` en tenant compte des réponses données précédemment, et en respectant les indications suivantes :

Étapes du code du contrôleur de connexion :

- Si aucune donnée n'est transmise, la vue d'authentification doit être affichée
- Si les données sont transmises correctement :
 - récupérer les données transmises depuis le formulaire,
 - tenter la connexion,
 - la connexion est réussie si l'utilisateur est connecté,
 - afficher la confirmation de connexion,
 - sinon afficher à nouveau le formulaire d'authentification.

Remarque : la première étape est déjà faite.

aide contextuelle : sur le fichier `authentification.inc.php` repérer les 2 fonctions : `login()` et `isLoggedOn()`. Sur le fichier `connexion.php` ces 2 fonctions devront être implémentées sur ce modèle :

```
if( ! empty($_SESSION['valeur']) && ! empty($_SESSION['autre_valeur']) ) {
    session_destroy() ; //on vide toutes les sessions - à utiliser avec prudence !
    login( $_POST['valeur'] , $_POST['autre_valeur'] ); //sessions activées ?
    if(isLoggedOn()) {
        //ici les vues correspondantes si l'utilisateur est bien connecté
    }
}
```

Question 2 - Contrôleur de déconnexion : deconnexion.php

Documents à utiliser

- fichiers fournis en ressources
- annexes 1, 2 et 5

Le contrôleur `deconnexion.php` est appelé lorsque l'utilisateur clique sur le lien "déconnexion" dans le menu principal.

- 2.1 Quelle fonction du modèle permet de déconnecter l'utilisateur actuellement connecté sur le site ?
- 2.2. Quelle vue permettant de confirmer la déconnexion devra être appelée par ce contrôleur ?
- 2.3. Lors de la déconnexion, est-il utile de transmettre une donnée au contrôleur ?
- 2.4. Compléter le code du contrôleur `deconnexion.php`. Celui-ci doit appeler la fonction appropriée du modèle, puis afficher à l'utilisateur un message de confirmation conformément aux questions précédentes.

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

Documents à utiliser

- fichiers fournis en ressources
- annexes 6,7, 8 et 9

En version finale, le contrôleur `rechercheResto.php` permet différents modes de recherche. Pour le moment il s'agit de se focaliser sur les deux recherches suivantes : par nom et par adresse.

Le contrôleur `rechercheResto.php` peut être appelé dans 2 situations :

- soit pour demander l'affichage du formulaire de recherche par l'intermédiaire de la vue `vueRechercheResto.php`,
- soit pour effectuer la recherche selon les valeurs saisies dans le formulaire et afficher les résultats à l'aide de la vue `vueResultRecherche.php`.

Lorsque l'utilisateur a rempli puis validé le formulaire de recherche, le contrôleur doit effectuer la bonne recherche en utilisant la fonction du modèle appropriée.

Le script de vue `vueResultRecherche.php` est capable d'afficher la variable appelée `$listeRestos`. Le type de données de cette variable est compatible avec ce que retournent les fonctions du modèle : `getRestos()`, `getRestosByNomR()` et `getRestosByAdresse()`.

- 3.1. En consultant le script `bd.resto.inc.php`, indiquer pour chacune de ces trois fonctions leurs signatures (prototype ou définition). Préciser le nom des paramètres attendus en plus de leurs types.

À l'aide de la fonction `print_r()`, afficher le contenu de la variable `$_POST` dans le contrôleur `rechercheResto.php`.

- 3.2. Quel est le contenu de la variable `$_POST` dans les 2 situations suivantes :

- recherche d'un nom de restaurant
- recherche d'un restaurant selon l'adresse suivante : rue saint remi 33000 bordeaux

- 3.3. Quels sont les noms de variables transmises au contrôleur en méthode POST lors de la recherche ?

- 3.4. Compléter 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. Par défaut ces variables sont initialisées à chaîne vide.

exemple : `$nomR` ne peut recevoir `$_POST['nomR']` que si `$_POST['nomR']` existe. Dans le cas contraire, `$nomR` doit se voir affecté chaîne vide.

Les paramètres de recherche sont donc contenus dans les variables mentionnées au dessus.

3.5. Compléter le code de chaque cas du switch dans le contrôleur en faisant appel à la fonction appropriée du modèle. Utiliser les paramètres tels qu'ils ont été décrits en question 3.1.

Rappel : Les données récupérées doivent être placées dans la variable `$listeRestos` pour que la vue puisse l'afficher.

3.6. Pourquoi l'appel aux fonctions du modèle est fait lorsque la condition `!empty($_POST)` est vérifiée ?

La vue affichant le résultat de la recherche (`vueResultRecherche.php`) doit être appelée dans le bloc de fin du contrôleur.

La syntaxe d'appel de vue est similaire aux autres vues incluses. Par exemple :

```
include RACINE . "/vue/entete.html.php";
```

L'affichage du résultat de la recherche n'est pas systématique, il faut que des données aient été trouvées.

Dans tous les cas, le formulaire de recherche est affiché afin de permettre à l'utilisateur de modifier sa recherche.

Consulter la version définitive du site pour avoir un aperçu du comportement attendu lorsque l'utilisateur recherche un restaurant.

3.7. Ajouter dans le contrôleur l'appel à la vue `vueResultRecherche.php`.

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

Documents à utiliser

- fichiers fournis en ressources
- annexes 6, 7 et 8

Le même contrôleur permet de rechercher selon le nom du restaurant ou son adresse. Pourtant il n'y a qu'une seule vue qui gère l'affichage des formulaires de recherche : `vueRechercheResto.php`.

4.1. Dans le code source de la vue, quelle variable permet de choisir l'affichage du formulaire de recherche par nom ou par adresse ?

4.2. Quelle variable transmise en méthode GET au contrôleur permet de connaître le type de recherche - par nom ou par adresse - que l'on souhaite effectuer ?

Le rôle de la variable action sera étudié plus tard.

4.3. Rechercher où est faite cette transmission : quel script ? quelle ligne ?

4.4. Sans cette transmission d'information, le contrôleur pourrait-il savoir quelle recherche effectuer ?

4.5. Sans cette transmission d'information, le script de vue pourrait-il savoir quel formulaire afficher ?

4.6. Lorsqu'une recherche est effectuée, la vue affiche à nouveau le formulaire de recherche avec des valeurs prédéfinies. Quelles variables sont alors utilisées ?

Synthèse sur le rôle et le fonctionnement du contrôleur

Le contrôleur est l'élément central dans les fonctionnalités proposées par une application MVC. Dans notre site web il permet de :

- récupérer les actions de l'utilisateur en terme d'action sur l'interface (saisies, choix, sélections, etc.) : informations transmises en méthode GET ou POST.
- Récupérer, depuis le modèle, les données qui seront utiles pour la fonctionnalité. Une liste de restaurants par exemple.
- coder la logique applicative : traiter les données, créer les variables qui seront utiles pour le fonctionnement de l'application et l'affichage. Vérifier que l'utilisateur a bien saisi son login et son mot de passe par exemple.
- commander l'affichage des vues pour afficher à l'écran les données récupérées ou calculées. La liste des restaurants recherchés par exemple.

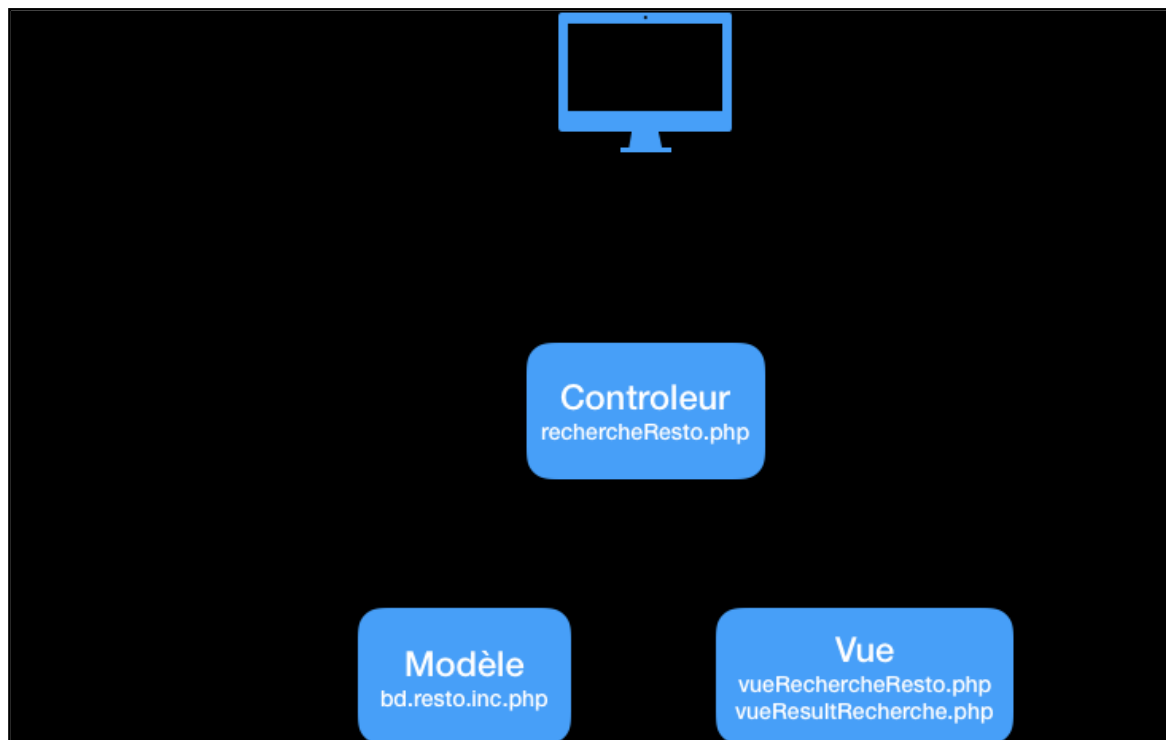


Schéma du fonctionnement du contrôleur rechercheResto.php lors d'une recherche par nom

Les fonctions du modèle sont accessibles grâce à l'inclusion des scripts nécessaires depuis le contrôleur.

Par exemple dans le script contrôleur rechercheResto.php :

```
include_once RACINE . "/modele/bd.resto.inc.php";
```

De même, les vues sont incluses en fin de contrôleur, une fois toute la logique applicative traitée. Les données créées ou récupérées dans le contrôleur y sont affichées.

Par exemple dans le script contrôleur listeResto.php :

```
include RACINE . "/vue/entete.html.php";  
include RACINE . "/vue/vueListeRestos.php";  
include RACINE . "/vue/pied.html.php";
```

Annexe 1 - section de test du fichier modèle authentication.inc.php

```
if ($_SERVER["SCRIPT_FILENAME"] == __FILE__) {
    // prog principal de test
    header('Content-Type:text/plain');

    // test de connexion
    if (isLoggedIn()) {
        echo "logged\n";
    } else {
        echo "not logged\n";
    }

    // login et mot de passe de test
    login("test@kercode.dev", "kercode");

    if (isLoggedIn()) {
        echo "logged\n";
    } else {
        echo "not logged\n";
    }

    $mail=getMailULoggedIn();
    echo "utilisateur connecté avec cette adresse : $mail \n";

    // deconnexion
    logout();
}
```

Annexe 2 - résultat d'exécution du script authentication.inc.php

```
not logged
logged
utilisateur connecté avec cette adresse : test@kercode.dev
```

Annexe 3 - vueAuthentication.php

```
<?php

if ($_SERVER["SCRIPT_FILENAME"] == __FILE__) {
    // Un MVC utilise uniquement ses requêtes depuis le contrôleur principal :
    index.php
    die('Erreur : '.basename(__FILE__));
}

?>
<h1>Connexion</h1>
<form action="./?action=connexion" method="POST">

    <input type="text" name="mailU" placeholder="Email de connexion" /><br />
    <input type="password" name="mdpU" placeholder="Mot de passe" /><br />
    <input type="submit" />

</form>
<br />
<a href="./?action=inscription">Inscription</a>
```


Annexe 4 - contrôleur connexion.php à compléter

```
<?php
/**
 *      Controleur secondaire : connexion
 */

if ($_SERVER["SCRIPT_FILENAME"] == __FILE__) {
    // Un MVC utilise uniquement ses requêtes depuis le contrôleur principal :
    index.php
    die('Erreur : '.basename(__FILE__));
}

require_once RACINE . "/modele/authentification.inc.php";

// creation du menu burger
$menuBurger = array();
$menuBurger[] = ["url"=>"./?action=connexion","label"=>"Connexion"];
$menuBurger[] = ["url"=>"./?action=inscription","label"=>"Inscription"];

// recuperation des donnees GET, POST, et SESSION
if (!isset($_POST["mailU"]) || !isset($_POST["mdpU"])){
    // on affiche le formulaire de connexion
    $titre = "authentification";
    include RACINE . "/vue/entete.html.php";
    include RACINE . "/vue/vueAuthentification.php";
    include RACINE . "/vue/pied.html.php";
} else {
    // à completer
}
?>
```

Annexe 5 - contrôleur deconnexion.php à compléter

```
<?php
/**
 *      Controleur secondaire : deconnexion
 */

if ($_SERVER["SCRIPT_FILENAME"] == __FILE__) {
    // Un MVC utilise uniquement ses requêtes depuis le contrôleur principal :
    index.php
    die('Erreur : '.basename(__FILE__));
}

require_once RACINE . "/modele/authentification.inc.php";

// recuperation des donnees GET, POST, et SESSION
// appel des fonctions permettant de recuperer les donnees utiles a l'affichage
// traitement si necessaire des donnees recuperees

// appel du script de vue qui permet de gerer l'affichage des donnees
$titre = "Deconnexion";

include RACINE . "/vue/entete.html.php";

include RACINE . "/vue/pied.html.php";
?>
```

Annexe 6 - contrôleur rechercheResto.php à compléter

```
<?php

/**
 *    Controleur secondaire : rechercheResto
 */

if ($_SERVER["SCRIPT_FILENAME"] == __FILE__) {
    // Un MVC utilise uniquement ses requêtes depuis le contrôleur principal :
    index.php
    die('Erreur : '.basename(__FILE__));
}

require_once RACINE . "/modele/bd.resto.inc.php";

// creation du menu burger
$menuBurger = array();
$menuBurger[] = ["url"=>"./?action=recherche&critere=nom","label"=>"Recherche
par nom"];
$menuBurger[] = ["url"=>"./?action=recherche&critere=adresse","label"=>"Recherche par adresse"];

// critere de recherche par default
$critere = "nom";
if (isset($_GET["critere"])) {
    $critere = $_GET["critere"];
}
// recuperation des donnees GET, POST, et SESSION
// recherche par nom
$nomR = null;
// recherche par adresse
$voieAdrR = null;
$cpR = null;
$villeR = null;

// appel des fonctions permettant de recuperer les donnees utiles a l'affichage
// Si on provient du formulaire de recherche : $critere indique le type de
recherche à effectuer
if (!empty($_POST)) {
    switch ($critere) {
        case 'nom':
            // recherche par nom
            break;
        case 'adresse':
            // recherche par adresse
            break;
    }
}

// traitement si necessaire des donnees recuperees

// appel du script de vue qui permet de gerer l'affichage des donnees
$titre = "Recherche d'un restaurant";
include RACINE . "/vue/entete.html.php";
include RACINE . "/vue/vueRechercheResto.php";
include RACINE . "/vue/pied.html.php";
?>
```

Annexe 7 - vue vueRechercheResto.php

```
<?php
if ($_SERVER["SCRIPT_FILENAME"] == __FILE__) {
    die('Erreur : '.basename(__FILE__));
}
?>

<h1>Recherche d'un restaurant</h1>
<form action="./?action=recherche&critere=<?= $critere ?>" method="POST">

    <?php
    switch ($critere) {
        case "nom":
            ?>
            Recherche par nom : <br />
            <input type="text" name="nomR" placeholder="nom" value="<?= $nomR
?>" /><br />
            <?php
            break;
        case "adresse":
            ?>
            Recherche par adresse : <br />
            <input type="text" name="villeR" placeholder="ville" value="<?=
$villeR ?>" /><br />
            <input type="text" name="cpR" placeholder="code postal" value="<?=
$cpR ?>" /><br />
            <input type="text" name="voieAdrR" placeholder="rue" value="<?=
$voieAdrR ?>" /><br />
            <?php
            break;
    }
    ?>
    <br /><br />
    <input type="submit" value="Rechercher" />

</form>
```

Annexe 8 - vue vueResultRecherche.php

```
<?php
if ($_SERVER["SCRIPT_FILENAME"] == __FILE__) {
    die('Erreur : '.basename(__FILE__));
}
?>

<h1>Liste des restaurants</h1>
<?php
for ($i = 0; $i < count($listeRestos); $i++) {
    ?>
    <div class="card">
        <div class="descrCard"><?php echo "<a href='./?action=detail&idR=" .
$listeRestos[$i]['idR'] . "'>" . $listeRestos[$i]['nomR'] . "</a>"; ?>
        <br />
        <?= $listeRestos[$i]["numAdrR"] ?>
        <?= $listeRestos[$i]["voieAdrR"] ?>
        <br />
        <?= $listeRestos[$i]["cpR"] ?>
    </div>
}
```

```

        <?= $listeRestos[$i]["villeR"] ?>
    </div>
    <div class="tagCard">
        <ul id="tagFood">
            </ul>
        </div>
    </div>
</div>
<?php
}
?>

```

Annexe 9 - extrait du modèle bd.resto.inc.php

```

<?php
include_once "bd.inc.php";

function getRestoByIdR($idR) {...}

function getRestosByNomR($nomR) {...}

function getRestosByAdresse($voieAdrR, $cpR, $villeR) {...}

if ($_SERVER["SCRIPT_FILENAME"] == __FILE__) {
    // prog principal de test
    header('Content-Type:text/plain');

    echo "getRestos() : \n";
    print_r(getRestos());

    echo "getRestoByIdR(idR) : \n";
    print_r(getRestoByIdR(1));

    echo "getRestosByNomR(nomR) : \n";
    print_r(getRestosByNomR("charcut"));

    echo "getRestosByAdresse(voieAdrR, cpR, villeR) : \n";
    print_r(getRestosByAdresse("Ravel", "33000", "Bordeaux"));
}
?>

```