Site de partage

De recettes de cuisine

Une image contenant intérieur, table

Description générée avec un niveau de confiance très élevé

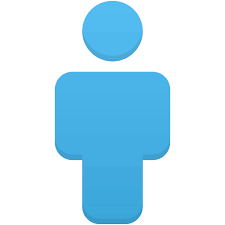
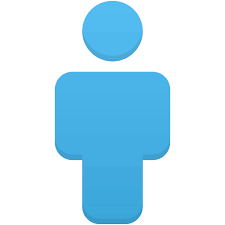


Table des matières

[Introduction 3](#_Toc503137852)

[Présentation du projet 3](#_Toc503137853)

[Documentations 3](#_Toc503137854)

[Analyse et choix techniques 4](#_Toc503137855)

[Résolution des problèmes rencontrés 4](#_Toc503137856)

[Arborescence du site 11](#_Toc503137857)

[Conception et mise en place de la BDD 12](#_Toc503137858)

[Diagramme UML 12](#_Toc503137859)

[Modèle physique des données 12](#_Toc503137860)

[Difficultés rencontrées et solutions 13](#_Toc503137861)

# Introduction

# 

## Présentation du projet

Ce site web permet à tout utilisateur de visualiser et de partager (s’il est authentifié) des recettes de cuisine organisées en 3 catégories : Entrées, Plats et Dessert.

Il est possible de voir les détails d’une recette tels que les ingrédients, le temps de préparation, le temps de cuisson, le nombre de personnes…

## Documentations

Les documentations utilisées relatent des techniques de codage en PHP, du framework CodeIgniter ainsi que du moteur de template Smarty.

**CodeIgniter :**

**https://www.codeigniter.com/user\_guide/**

**https://openclassrooms.com/courses/codeigniter-le-framework-au-service-des-zeros**

**PHP :**

[**http://php.net/docs.php**](http://php.net/docs.php)

**Smarty :**

**https://www.smarty.net/documentation**

# Analyse et choix techniques

## Résolution des problèmes rencontrés

* Le MVC :

Le MVC s’est avéré être une grande utilité au niveau de l’organisation du code et du problème d’inversion des dépendances.

PHP de base autorisant le codage volatil et très peu organisé, la logique en couches métiers permet une meilleure performance et lisibilité, ouvrant ainsi plus de souplesse sur la manière de coder et l’ajout de futures extensions.

Le framework CodeIgniter est un bon choix car il se veut simple et plutôt basique. Aucune fonctionnalité « en trop » n’est implémentée de base et la philosophie reste largement accessible.

En effet, il est possible de choisir et de configurer soi-même les fonctionnalités du framework dans un simple fichier PHP.

Mais la simplicité de CodeIgniter réside dans la division des couches métiers.

Il est aisé de circuler entre le Modèle, la Vue et le Contrôleur dû à la simplicité des procédés. N’importe quel contrôleur peut charger n’importe quel modèle de données, et il en va de même pour les vues.

Finalement, un des gros avantages de CodeIgniter est qu’il est possible de gérer les droits des utilisateurs avec la POO.

Pour cela, il suffit de créer un Contrôleur personnalisé qui implantera directement toutes les opérations de vérification (si l’user est connecté, s’il est admin…) et de faire hériter ce dernier à d’autres Contrôleurs spécialisés pour le type d’utilisateur (un pour l’user lambda, un pour les membres, un pour l’admin…).

Une image contenant capture d’écran

Description générée avec un niveau de confiance très élevé

Système de gestion des droits du site web.

Les contrôleurs permettent ou ne permettent pas de faire certaines choses en fonction des paramètres suivants :

* L’utilisateur est connecté ou non
* L’utilisateur est connecté et est admin
* Le Routing :

Le routing dans CodeIgniter masque au maximum l’arborescence du site pour les utilisateurs et se veut également très simple d’utilisation : on accède aux pages par les fonctions de chaque Contrôleur passées en URL.

Ex : pour accéder à la page de connexion, on saisit l’url suivante (en local) :



La fonction « connexion » du Contrôleur « membre » se charge de générer (ou charger) la vue du formulaire de connexion, et d’en effectuer les traitements nécessaires.

Le système de routing est totalement configurable. Pour ce projet, un fichier .htacces a été créé dans le but de masquer encore plus l’arborescence du site pour les utilisateurs, et cela en masquant la partie « index » susceptible d’apparaître dans les URL.

* Les helpers :

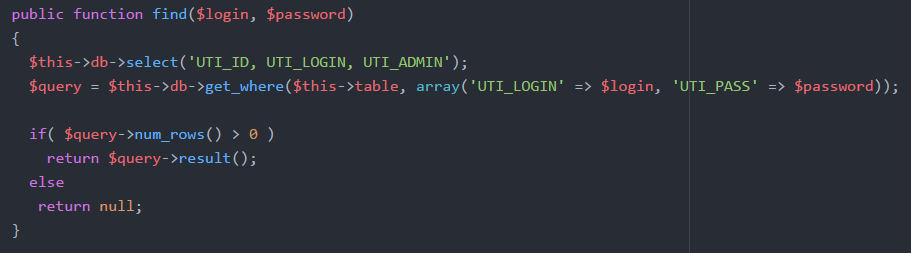
CodeIngiter propose des helpers (que l’on peut configurer dans le fichier autoload.php) afin d’alléger le code, d’en améliorer sa lisibilité, et surtout de rendre le site portable.

Le projet utilise les helpers ‘database’, ‘url’ et ‘upload’, qui permettent respectivement :

* De dialoguer avec la BDD avec le contrôleur directement et non pas avec des requêtes
* De rendre plus souple l’écriture des url et d’en masquer une partie à l’utilisateurs, avec par exemple la fonction ‘base\_url()’ renvoyant l’url configurée dans le fichier config.php
* D’upload un fichier plus facilement et de configurer des conditions (taille du fichier, dossier d’upload, largeur et longueur maximum d’une image…)

Les helpers constituent donc une souplesse de codage, mais aussi une couche de sécurité supplémentaire.

Voici un exemple d’utilisation de l’helper ‘database’ :



‘database’ propose un ensemble de méthodes portables et compatibles avec n’importe quelle version de base de données

* Les variables de session :

Le site utilise les variables de session afin de s’assurer qu’un utilisateur est connecté, de vérifier ses droits et ses informations personnelles.

Grâce à l’organisation des contrôleurs vue précédemment, certaines fonctionnalités réservées sont interdites d’accès, et ce en récupérant les informations stockées dans les variables de session.

Ainsi, le constructeur de chaque contrôleur du dossier ‘core’ vérifie les informations contenues dans les variables de session.

Par exemple, les fonctionnalités réservées aux membres sont inaccessibles pour les utilisateurs non connectés, puisqu’ils n’ont pas de session et donc aucune variable dans cette dernière.

La super globale $\_SESSION contient les informations suivantes pour un utilisateur authentifié :

* l’id de l’utilisateur
* son pseudo
* ses droits : 1 s’il est admin, 0 sinon
* un booléen indiquant l’état de la session : TRUE si l’utilisateur est connecté

Avec toutes ces informations, on est en mesure de restreindre ou d’autoriser aux fonctions des contrôleurs.

* Smarty :

Un des seuls inconvénients de CodeIgniter, c’est qu’il ne propose pas de moteur de template efficace directement implémenté.

C’est pourquoi le projet allie CodeIgniter et Smarty.

Grâce à la bibliothèque Smarty de Cryode, la Vue est entièrement dépourvue de code php (pas de balises < ?php ?>) et fonctionne uniquement selon les données que son contrôleur associé lui envoie au moment de la générer.

Un autre avantage est de ne pas réécrire plusieurs fois des éléments d’une page web tels que le header, la barre de navigation ou le footer.

Un template sous Smarty peut en hériter d’un autre.

Une image contenant capture d’écran

Description générée avec un niveau de confiance très élevé

Ci-contre le layout principal, layout.tpl, dont toutes les autres vues contenues dans le répertoire ‘apllication/views/modules’ héritent.

Une image contenant capture d’écran

Description générée avec un niveau de confiance très élevé

Ci-dessus la vue ‘accueil.tpl’, héritant de layout.tpl.

Les balises {block} {/block} correspondent au contenu variant entre les différentes pages.

# Arborescence du site

Une image contenant capture d’écran

Description générée avec un niveau de confiance très élevé

# Conception et mise en place de la BDD

## Diagramme UML

## Modèle physique des données

VOIR LE DOSSIER « DATABASE » DE L’ARCHIVE

# Difficultés rencontrées et solutions

* Découverte de l’organisation MVC et de l’utilisation d’un framework, plus particulièrement de CodeIgniter qui ne propose pas de fonctionnalités pouvant être générées automatiquement. Par exemple, le framework Laravel permet de générer les fonctionnalités d’authentification.
* La bibliothèque « session » de CodeIgniter qui ne marche pas à cause des droits d’accès au dossier session du serveur apashe. L’utilisation classique des sessions en php natif fonctionne.
* Les images des pages qui ne s’affichent plus sauf pour la page d’accueil et la page des recettes. Le problème viendrait du .htaccess qui réécrirait également les chemins menant au dossier ‘asset’ (contenant les images). Aucune solution trouvée.
* La répartition du travail au sein du groupe.

# Conclusion

## Points positifs

* Plus grande familiarisation avec le langage php natif, le MVC et les frameworks php
* Meilleure compréhension du traitement de données