Documentation technique

GSB Frais

2014

Kévin Tirado

Kévin Tirado

01/01/2014

SOMMAIRE

1. Le MVC, c’est quoi ?
2. Plan du site.
3. Moteur de l’application, Framework, bibliothèques externe utilisées.
4. Descriptif des éléments, classes et bibliothèques externes utilisées.
5. Schéma de la base de données.
6. Propositions d’amélioration de l’application.

<http://lemalesaint.fr/wp-content/uploads/2012/03/pourquoi-devriez-vous-utiliser-un-framework-php-3.png>

# Le MVC, c’est quoi ?

Le MVC qui signifie Model View Controller (en anglais) est une architecture à suivre pour structurer le code source d’un projet. Ce paradigme consiste à séparer le code en trois parties.

1. La partie Vue : Ce que vois l’utilisateur quand il navigue sur un site.
2. La partie Contrôler : C’est ici qu’on retrouve la logique du code, le centre de contrôle en quelque sorte. Par exemple : Il autorise ou non les utilisateurs à voir certaines pages, le contrôleur analyse simplement les données, il transmet ensuite des informations à la vue et au model si besoin.
3. La partie Model : Cette partie gère les données de l’application. Toutes les demandes d’accès à des informations de la base de données ce fait ici, dans le model, nous allons donc retrouver des requête SQL dans celui-ci.

Dans l’arborescence d’un projet, chaque partie correspond à un dossier. Nous aurons donc, un dossier, Contrôleur, vue et model.

Pour être encore plus clair et disperser ce flou (s’il existe encore) sur le sujet, prenons un exemple pour comprendre l’utilisation et l’importance du concept MVC :

Une application qui gère des articles de journal en ligne.

Pour afficher les articles au visiteur, le contrôleur contact le model pour lui demander, les 5 derniers articles enregistrés dans la base de données. Le model va faire une requête SQL à la BDD (Base de données) et retourné les informations au contrôleur, qui va à son tour les transmettre à la vue. Mettons que l’utilisateur veuille éditer un article, c’est le contrôleur qui va décider de rendre possible ou non cette opération, d’où son nom, un contrôleur, contrôle les informations et affiche en conséquences les données destinés à tel ou tel personne.

# L’avantage du MVC.

Bien entendu que par le fait que l’application est une architecture propre, il est rendu possible la gestion rapide des erreurs ou autres bugs, problèmes en tout genre. Une erreur SQL ? Il suffit de vérifier la partie « model » pour corriger la syntaxe d’une requête (par exemple). Une erreur d’affichage sur la vue ? Oui, un petit coup d’œil dans la partie « vue » va surement régler rapidement le problème. Vous l’aurez compris, une application structurée par une architecture est un gain de temps énorme en rapport de débogage.

L’application va pouvoir grandir dans le temps, sans se heurter à des soucis d’évolution liée à une mauvaise organisation du code source. Une application structurée est un projet qui restera fiable et maniable dans le temps.

# Pour aller plus loin dans son fonctionnement.

Rien de tel qu’un croquis pour reconstituer correctement ces informations et les sauvegarder en mémoire.



Comme nous pouvons le voir quelques éléments en plus on était rajouté en plus du « Model, Vue, Contrôleur ». Le croquis ci-dessus représente la logique d’une application pendant son fonctionnement, donc quand l’utilisateur veut accéder à une page.

Explication en détail :

* En gris nous avons donc le navigateur du client. Le client demande l’affichage d’une page.
* En orange, nous avons donc le serveur, là où se trouve l’application. Maintenant une question se pose. Que veut voir l’utilisateur ? Pour gérer ça demande l’application à besoin de lire l’url du client. L’url correspond à une route, c’est-à-dire, des informations qui vont permettre à l’application d’afficher la page souhaité. L’url est donc transmise au « Dispatcher». Le dispatcher va rechercher le contrôleur associé à l’url s’il en existe un, dans le cas contraire, une page 404 pages non trouvé peut être retourné.
* En bleu, le contrôleur, qui vérifie par exemple si l’utilisateur peut voir la page désiré. Il peut contacter le model (en rouge) s’il a besoin d’informations stocké en base de données.
* En rouge donc, le model, contacté exclusivement par le contrôleur, pour récupérer des données.
* En vert, la vue, à qui le contrôleur a passé des informations.
* Retour au serveur, qui affiche dans la page web, dans le navigateur du client.

# Plan du site

Tableau représentant les pages disponibles pour les deux modules.

|  |  |
| --- | --- |
| Module espace visiteur | Module espace comptable |
| Page de connexion | |
| Page saisir une fiche de frais | Page Validation d’une fiche de frais |
| Page consulter le suivis d’une fiche de frais |  |

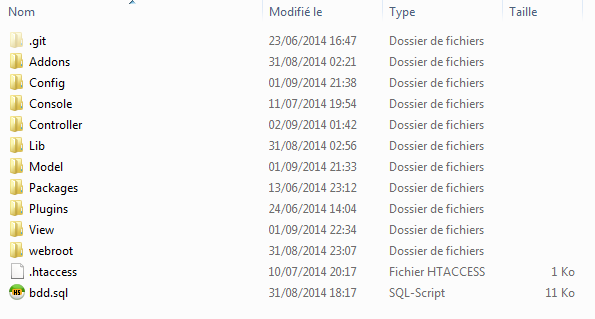
# Moteur de l’application, Framework, bibliothèques externe utilisées

L’application « GSB-AppliFrais-MVC » a été adapté comme l’autorisé le cahier des charges. Une adaptation de celle-ci a donc été faite sur un Framework.

## Présentation rapide du Framework

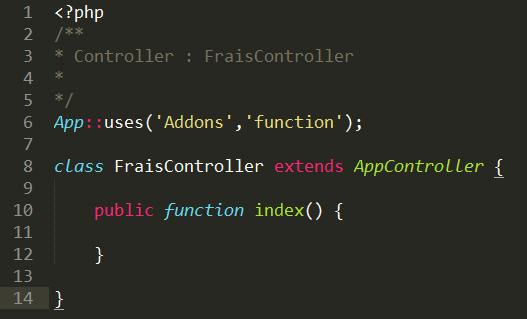
Le Framework utilisé a été conçu de A à Z en suivant l’architecture MVC et est entièrement développé en POO (Programmation Orienté Objet). Il comprend aussi un ORM (couche d'abstraction à la base de données) en cours de développement, qui lui sera propre.

Arborescence du Framework :



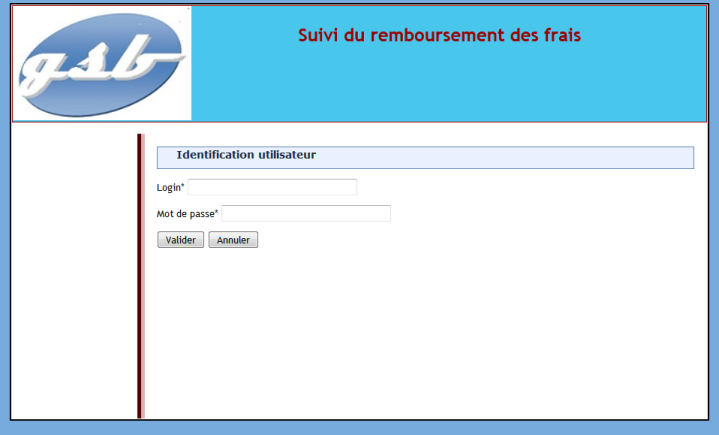
**MVC**

La structure d’un contrôleur est représenté par une class comme l’illustre l’image ci-dessous :



La méthode « index » représente la partie « logique » du code, c’est le contenu de cette méthode qui va être exécuté lors de l’appel de la page « index » du contrôleur « FraisController ». Exemple d’url : « https://localhost/frais/index ».

# Descriptif des éléments, classes et bibliothèques externes utilisées

En termes de bibliothèques externes utilisées, l’application tourne avec JQuery, une bibliothèque complète, écrite en JavaScript, disponible directement avec le Framework.

1. Page de connexion

La page de connexion, sert à identifier l’utilisateur. Un jeu de login et mot de passe est demandé pour accéder à l’application.

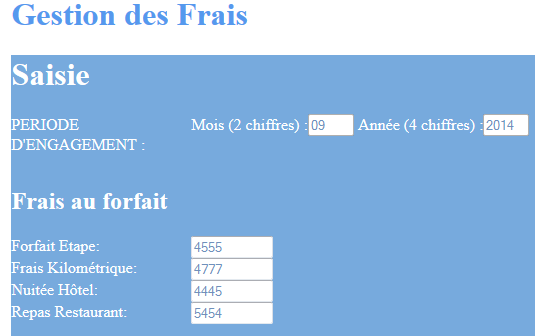
La page représente une méthode, nommé « login » accessible via le contrôleur  « UserController »



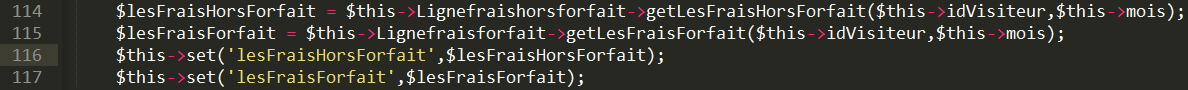
Class UserController



Méthode login

1. Saisir fiche de frais

La page  « saisir fiche de frais », permet aux visiteurs de renseigner leurs frais du mois en cours. Un exemple en illustration ci-dessous de la logique du code utilisé pour charger une fiche de frais.

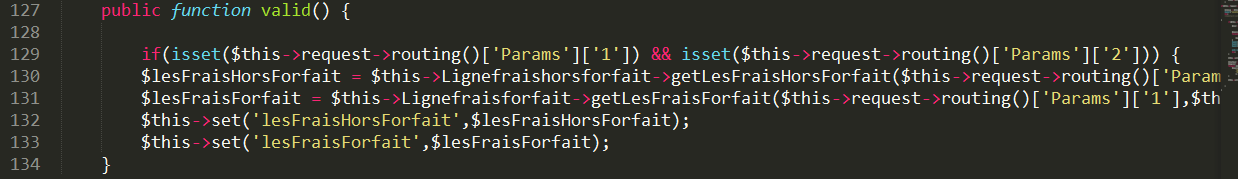
On récupère les frais hors forfait s’il existe, ainsi que les frais forfait, puis on transmet les données à la vue.

1. Consulter une fiche de frais

Le visiteur peut consulter ses fiches de frais qui sont en cours de traitements. En effet après chaque 20 du mois, le visiteur ne peut plus modifier la fiche de frais de celui-ci. Il peut en revanche consulter les avancements de la fiche de frais dans la partie « Consultation », plus précisément « Mes fiches de frais ». Pour cette partie, le contrôleur principal se nomme « FraisControlleur » et la vue « valid ». Au niveau de l’url cela donnerai :

« https://localhost/frais/valid »

Exemple de code dans le Framework représentant une page de l’application.



1. Valider une fiche de frais

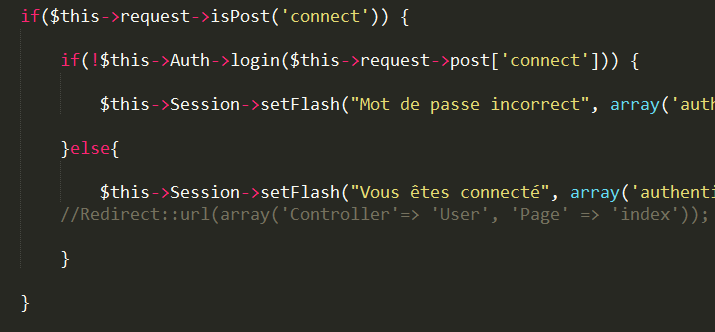
La validation d’une fiche de frais ce fait exclusivement par un comptable, après le 20 de chaques mois. Pour que l’application reconnaisse l’utilisateur, il a besoin d’information relative à celui-ci, comme son identifiant.

Le Framework dispose de plusieurs composants, parmi la liste, se nomme « Auth », c’est une class qui aide le développeur dans sa tâche de gestion de session d’utilisateur.

Accessible dans tous les contrôleurs via $this->Auth, il permet rapidement de mettre en place un espace protégé. En rapport avec la validation des frais, cette class « AuthComponent » est appelé pour vérifier divers points, comme, la vérification de la session utilisateur et s’il a les droits d’accès à la modification d’une fiche.

## Exemple d’utilisation rapide de la class « AuthComponent »

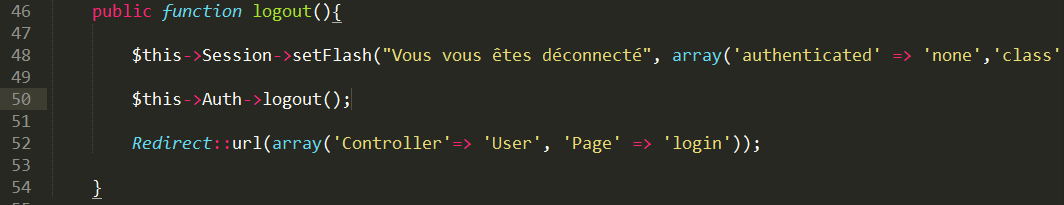
Le premier bloc conditionnel, vérifie s’il existe un POST nommé ‘connect’ sur la page. Si oui, viens le deuxième bloc conditionnel, qui appel une méthode « login », en arguments passé, un tableau contenant les identifiants de connexion envoyé par l’utilisateur. Dans le cas où l’utilisateur ne serez pas reconnu un message s’affichage pour prévenir l’échec de l’authentification.



Une fois l’utilisateur connecté, toutes les informations le concernant, sont disponible via la méthode « user » du « AuthComponent ». Par exemple, l’illustration ci-dessous, affichera l’identifiant de l’utilisateur connecté :

Avec ce procédé, il est donc possible d’accéder rapidement et simplement aux données relatives à l’utilisateur qui est authentifié, comme l’identifiant de son groupe, pour pouvoir afficher en conséquence les bonnes permissions de droits.

La déconnexion de l’utilisateur ce fait via l’appel de la méthode « logout » du « AuthComponent ». Ainsi dans le contrôleur « UserController », une méthode représente la page de déconnexion :



La méthode « logout » est appelé ligne 50. ( $this->Auth->logout(); )

# C:\Users\Skillendo PC1\Desktop\mcd.pngSchéma de la base de données MCD

# Propositions d’amélioration de l’application

Faire les requêtes en Ajax. Du formulaire de connexion jusqu’à la déconnexion. L’utilisateur navigue sur l’application, sans recharger à aucun moment, la page où il se trouve.