Les fiches récap de l'école O'clock

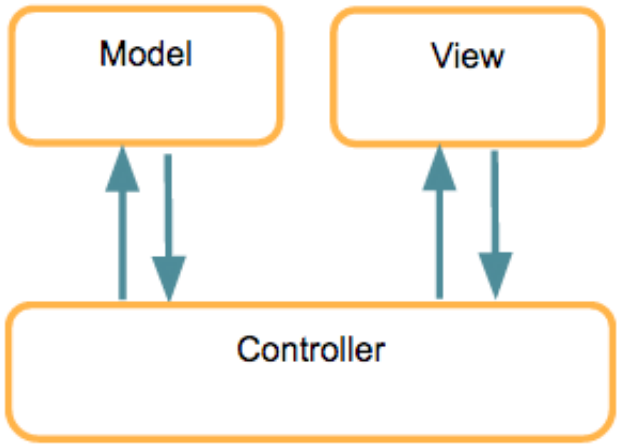
Gestion-projet

Architecture MVC : Model – View – Controller

Dernière modification: 6 juillet 2023

Architecture MVC : Model – View – Controller

- *MVC est un pattern* (modèle ou « patron de conception ») de programmation objet (mais qui peut être implémenté en procédural, notamment avec require). Il consiste en la séparation de l'application en trois *couches* (à titre de rappel une couche consiste de classes, ensemble de classes, ou composants), où chaque couche logicielle assure un rôle distinct : le Model, la View et le Controller.
- L'utilisation de l'architecture ou pattern MVC possède de nombreux avantages, et en particulier une meilleure lisibilité et organisation du code. De ces deux avantages en découlent beaucoup d'autres tel que la prévention des erreurs de part sa simplicité : MVC est un pattern relativement simple, mais puissant – il suit l'adage *KISS* (Keep It Simple, Stupid).



Model

> Gère l'accès aux entités manipulées par l'application:
>
> - protège l'intégrité des données en implémentant la logique métier
> - s'occupe du stockage

La définition des entités elles-mêmes (produit, utilisateur, employé) peut être séparée de l'accès aux données.

La couche Model est l'ensemble des classes qui définissent les objets manipulés par l'application, qui contiennent les *données* et réalisent les opérations de stockage.

Plusieurs stratégies existent pour implémenter cette couche de l'application.

Pour aller plus loin sur le sujet :

▶ Détails

View

> Gère la présentation des données:
> Les données sont « récupérées » par le Controller (via le Model) et « passées » à la View, qui est chargée de les présenter.

Systèmes de templates

- Plates : PHP natif
  http://platesphp.com/

- Smarty : langage dédié, templates compilés
  https://github.com/smarty-php/smarty/
- Twig (Symfony) : syntaxe dédiée, templates compilés
  https://twig.symfony.com/

Controller

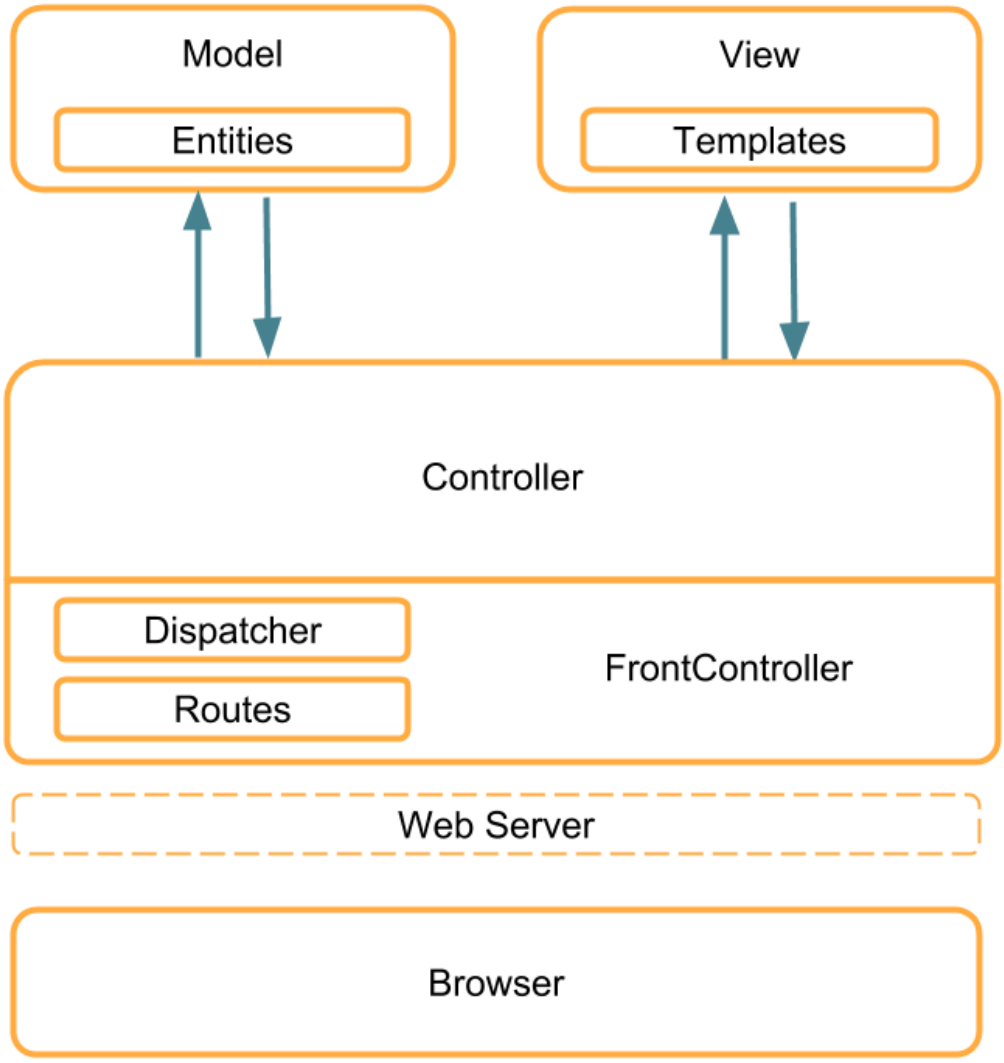
> Objectif : Gérer l'aspect dynamique de l'application :
> A partir de l'action demandée (requête utilisateur), il récupère les données (avec le Model) les injecte dans la vue adéquate, et envoie la réponse produite.
> (selon l'implémentation, parfois c'est la vue qui renvoie elle-même la réponse)

Le Front Controller

> Objectif: centraliser les requêtes à l'application :
> Grâce à un « Single Point of Entry », qui va analyser la requête et appeler l'action correspondante, du Controller correspondant.

Le FrontController implémente 2 étapes de traitement:

- le routage de la requête (recevoir la requête, identifier l'action à exécuter et les paramètres)
- le dispatch (instancier le Controller et exécuter l'action)



Implémentation

FrontController

Routage

Système minimaliste de routage:

- le fichier `.htaccess`

```
RewriteEngine On
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php?params=$1 [L,QSA]
```

- le fichier `index.php`

```php
if(empty($_GET))
{
  $params[0] = "default";
}
else
{
  $params = explode('/', $_GET["params"]);
}
```

Stratégie / Structure de projet

Les stratégies d'implémentation du modèle MVC varient: il n'y a pas *une seule* façon de structurer son application, ses classes, son système de fichiers.

Quelques questions à se poser lors de la planification du développement:

- Quelles actions pour mes utilisateurs?
- Quelles routes correspondantes? Est ce que j'ai besoin d'un système de routage?
- Quels controllers pour implémenter ces actions?
  - un controller par model? convient pour les applications CRUD (gestion...)
  - un controller par vue? convient pour les applications/vues complexes
  - un controller par domaine fonctionnel?
- Quelle stratégie pour mes modèles? Est ce que j'ai besoin d'un ORM?
- Quel système de templates?