Chapitre 3
Architecture du framework

1. Le modèle de conception MVC

1.1 Définitions et responsabilités

L'acronyme MVC (en anglais : *Model View Controller*) est un terme très répandu dans l'univers du développement logiciel. Il qualifie un modèle de conception (ou *Design-Pattern* en anglais), dont l'objectif est d'identifier précisément les responsabilités des différents composants d'une application afin d'augmenter sa maintenabilité et son évolutivité.

Le modèle MVC est un modèle de conception d'architecture logicielle dont les premières implémentations remontent au début des années 1980 dans le langage SmallTalk. L'objectif étant de maîtriser une certaine complexité applicative, il est décidé d'affecter des tâches précises à trois catégories de composants :

– Le modèle : des composants responsables de la gestion des données et des traitements métier de l'application.
– La vue : pour prendre en charge l'affichage des informations à destination de l'utilisateur final.
– Le contrôleur : servant d'aiguilleur entre l'utilisateur, les données et la vue.

Repris à la fin des années 1990 par la plateforme Java Enterprise Edition dédiée au développement web, il a ensuite été appliqué à bien d'autres technologies de développement d'applications web et de sites web, et donc en PHP dans Symfony.

1.1.1 La vue

La vue correspond à la partie « présentation ». C'est par celle-ci que l'utilisateur interagit avec l'application.

Elle désigne souvent des templates ; un template est un « gabarit », une mise en page permettant de présenter des informations aux utilisateurs au travers d'une interface.

Dans le contexte d'une application web, un template contient essentiellement du HTML pour la mise en page, ainsi que du code spécifique, PHP ou autre, pour y intégrer les données dynamiques venant du modèle. L'utilisateur interagit avec l'application en cliquant par exemple sur un lien ou en remplissant un formulaire.

1.1.2 Le modèle

Le modèle, quant à lui, désigne les fonctionnalités de l'application. Il couvre un spectre assez large, allant des « classes métier » (Utilisateur, Produit, Commande, etc.) aux classes chargées de manipuler celles-ci et de gérer leur lien avec la base de données.

Il sera donc nécessaire de délimiter chacune des responsabilités des classes du modèle.

1.1.3 Le contrôleur

Le contrôleur est le chef d'orchestre de l'application ; il est l'intermédiaire entre l'utilisateur et les couches Modèle et Vue.

Le contrôleur est composé d'une multitude d'actions, et une action est associée à chaque requête. L'action contient de la logique mais aucun traitement, ce dernier étant pris en charge par la couche Modèle.

1.2 En pratique

En parcourant les documentations de plusieurs frameworks pour applications web, on remarque différentes politiques quant à leur compatibilité avec ce modèle de conception. Tandis que certains s'en targuent fièrement en page d'accueil, d'autres en parlent très peu, voire ne l'évoquent pas du tout. En réalité, la quasi-totalité le sont.

En effet, ce modèle de conception désigne trois couches assez abstraites. Il n'est ni un choix architectural fort, ni un qualificatif suffisant pour désigner l'organisation d'un projet. C'est le minimum qu'un framework se doit de proposer (même si ce modèle de conception MVC n'est pas obligatoire ou configuré par défaut).

Examinons maintenant le fonctionnement d'un framework MVC au travers d'un exemple, une requête HTTP donnée :

GET /articles/introduction.html

Par défaut, en recevant cette requête, un serveur web essayerait de mettre en relation l'URL avec le système de fichiers du serveur. Apache, par exemple, irait chercher le fichier **DocumentRoot/articles/introduction.html**.

Remarque

DocumentRoot est le chemin vers le dossier public de l'application, c'est-à-dire celui dont les fichiers sont accessibles par les utilisateurs.

Ce comportement implique la création d'un fichier pour chaque URL du site, qui devra contenir le même code nécessaire au démarrage de l'application.

Ceci est incompatible avec le fonctionnement des frameworks, car ils embrassent le principe du **D**on't **R**epeat **Y**ourself (en français, « ne pas se répéter »).

Répéter le même code sur tous les fichiers complique la maintenance du site web si une partie de ce code doit être modifiée, vous devrez mettre à jour beaucoup de fichiers.

La solution à ce problème est d'utiliser un seul fichier pour gérer les échanges entre le serveur web et le framework, qui agira en tant qu'intermédiaire entre ces deux entités : le contrôleur frontal.

1.2.1 Le contrôleur frontal

C'est le point d'entrée de l'application, il contient le code nécessaire à l'amorçage (*bootstrap*) de l'application.

Notre requête a donné lieu à l'interprétation de ce fichier et non de **DocumentRoot/articles/introduction.html**.

Ceci est rendu possible par les modules de réécriture d'URL proposés par les serveurs web. Ils sont capables d'analyser la requête HTTP entrante et de la réécrire à la volée pour qu'elle soit orientée vers un autre fichier.

1.2.2 Le routage

Une fois l'application démarrée par le contrôleur frontal, le routage doit décider de l'action à effectuer pour satisfaire la requête du client, à savoir une requête GET sur la ressource */articles/introduction.html*.

Ici, le routage décide qu'il faut exécuter l'action **voir** du contrôleur **Article**. Cette décision a été prise car des règles de routage ont préalablement été définies par le développeur. Nous reviendrons sur ces dernières au cours du chapitre Routage et contrôleur.

1.2.3 Le contrôleur et le modèle

Les actions sont des tâches. Elles sont regroupées par thèmes au sein de contrôleurs.

Concrètement, un contrôleur est une classe et chacune de ses méthodes publiques est une action. Ici, l'utilisateur souhaite voir un article, c'est-à-dire exécuter l'action **voir** du contrôleur **Article**.

« Exécuter l'action voir du contrôleur Article » se traduit en PHP par « invoquer la méthode **voir** de la classe **ArticleControleur** ».

Voici ce à quoi pourrait ressembler ce contrôleur :

```
<?php

class ArticleControleur
{
    public $modele;
    public $vue;

    // Le titre de l'article ('introduction' dans notre cas)
    // est passé en argument lors de l'invocation de l'action.
    public function voir($titre)
    {
        $article = $this
            ->modele
            ->chercherArticle($titre);

        $this->vue->afficher(array('article' => $article));
    }
}
```

La classe ci-dessus est une parfaite illustration de la description de la couche Contrôleur, contenant uniquement de la logique, que nous avons faite (cf. section Le modèle de conception MVC - Définitions et responsabilités). Ici, le contrôleur demande à la couche Modèle de rechercher l'article, puis le transmet, via une méthode **afficher**, à la couche Vue.

1.2.4 La vue

En dernier lieu, la vue s'occupe de présenter la page HTML. Ici, elle affiche le titre de l'article ainsi que son contenu :

```
<!DOCTYPE html>
<html>
    <body>
        <h1><?php echo $article->getTitre() ?></h1>
        <p><?php echo $article->getContenu()?></p>
    </body>
</html>
```

1.2.5 En synthèse

Bien qu'il ne s'agisse ici que de « pseudo code » et en aucun cas du code Symfony, il n'en illustre pas moins le principe du modèle MVC tel qu'il est mis en œuvre dans les frameworks. Les frameworks PHP tels que Symfony, Zend Framework ou encore Laravel, utilisent ce principe et seule la syntaxe d'appel diffère, le principe reste le même.

Une bonne compréhension du principe de fonctionnement du modèle MVC est aujourd'hui une compétence essentielle pour appréhender les frameworks de développement, et pas seulement Symfony ou bien les autres frameworks PHP.

2. Architecture de Symfony

2.1 Présentation

Nous allons maintenant découvrir les principales entités du framework participant au traitement d'une requête, et en découvrir deux qui sont spécifiques à Symfony : le *Kernel* et le *Service Container*.

Avant de schématiser cette architecture, voici un bref descriptif de ces deux entités :

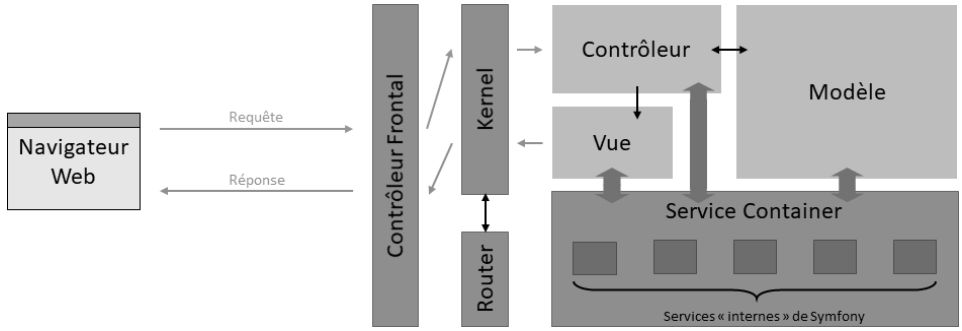
- Le **Kernel** (noyau, en français) est le cœur du framework. C'est un composant interne et il n'est pas obligatoire de connaître son existence pour développer sous Symfony. Cependant, comprendre son fonctionnement et savoir écouter les signaux qu'il émet est essentiel pour bénéficier pleinement des possibilités offertes par le framework.
- Le **Service Container** est un composant incontournable. C'est une sorte de boîte à outils, dans laquelle vous trouverez ce qu'on appelle des « services ». Ces services sont divers et variés. Certains vous permettent de requêter des bases de données, d'autres de sérialiser des objets. Vous aurez même le droit de créer vos propres services.

2.2 Le contrôleur frontal

Il existe un troisième composant important : le **contrôleur frontal**.

Le contrôleur frontal n'est pas véritablement une spécificité de Symfony ; il correspond à une approche standard du modèle MVC qui part du principe que tout le traitement bas niveau des requêtes se fait par un seul et unique point d'entrée. Ce contrôleur frontal est notamment chargé de transformer les requêtes et réponses HTTP en objets PHP.

Avant d'approfondir, découvrons au travers d'un schéma le traitement d'une requête dans une application Symfony.



Flux applicatifs entre une requête et une réponse dans une application Symfony

Le Kernel reçoit la requête du client, il interroge le « router » dans le but de savoir quel contrôleur il doit invoquer.

Une fois ce contrôleur invoqué, le Kernel attend de lui qu'il retourne une réponse HTTP, peu importe la manière dont elle est générée.

De ce fait, le contrôleur est libre d'invoquer différents services, comme le modèle ou la vue, mis à sa disposition au sein du Service Container. De ces différentes interactions naît une réponse HTTP, qui est renvoyée au Kernel.

Finalement, le Kernel transmet cette réponse au client.

2.3 Le Service Container

Le Service Container contient des « services » sur le schéma précédent, on y aperçoit par exemple les services « Modèle » ou « Vue ».

Remarque

Sur le schéma, nous décrivons uniquement « Modèle » et « Vue », mais il existe bien d'autres services ; certains sont présents par défaut, d'autres sont créés par le développeur.

Les services ne sont pas des concepts abstraits : chaque service est un objet PHP.

Le Service Container est la seule entité mise à disposition du contrôleur pour l'aider à générer sa requête HTTP, c'est donc un élément central de l'application.

En installant Symfony, nous nous rendons compte qu'il est préconfiguré avec un certain nombre de services, chacun répondant à un besoin particulier (gérer les templates, communiquer avec des bases de données, etc.).

2.4 Le modèle MVC dans Symfony

Il serait réducteur de qualifier Symfony de framework MVC. En effet, bien que celui-ci soit configuré par défaut, nous voyons par exemple que le service Modèle n'est pas obligatoire. De plus, les services Modèle et Vue étant contenus dans le Service Container au milieu d'autres services, ils sont considérés comme des services quelconques par le framework, qui n'a en aucun cas conscience de cette architecture.

Le fait est que Symfony n'a pas été spécialement conçu pour accueillir ce modèle de conception. Hormis la couche Contrôleur qui est obligatoire, le développeur est libre d'implémenter l'architecture de son choix pour son application.

Cependant, nous travaillons essentiellement selon l'approche MVC tout au long de cet ouvrage, car ce modèle de conception répond naturellement à la plupart des besoins d'une application web moderne et est gage d'évolutivité et de maintenabilité pour les applications.

2.5 L'approche par composant

Symfony est connu pour être un puissant framework **PHP** mais il est tout de même important de savoir que c'est aussi un ensemble de composants.

Chaque composant est une sorte de librairie autonome, utilisable dans n'importe quel projet PHP. En voici quelques-uns :

– Form : gestion des formulaires.
– Console : création de programmes en ligne de commande.
– Translation : internationalisation d'applications.

Remarque

Pour un listing complet : https://symfony.com/components

Les composants Symfony peuvent être utilisés dans n'importe quel projet PHP. Le framework Symfony est le projet qui les exploite pleinement et de manière approfondie. Mais il en existe d'autres : Silex, par exemple, est un framework PHP ultraléger, développé entièrement autour des composants Symfony ; autre exemple : Drupal, un système de gestion de contenu pour le Web (CMS : *Content Management System*) lui aussi construit majoritairement autour des composants de Symfony.

Dans leurs premières versions, les composants étaient très difficilement utilisables de manière autonome, notamment à cause d'un système d'autoload (autochargement des classes) peu standard et d'une omniprésence du modèle de conception Singleton rendant très monolithique la structure du framework.