TEK-UP Ecole Supérieure Privée
Technologie & Ingénierie

Développement Web (PHP Symfony 6)

Jasser Jammeli
jasserjammeli98@gmail.com

A.U. 2025-2026

Plan du cours

I. Introduction Symfony

II. Routing

III. Les contrôleurs

IV. Les Twigs

V. ORM Doctrine

VI. Les Formulaires

Plan du cours

I. Introduction Symfony	TP1: Création d'un projet
II. Routing	TP2: Router + Controller
III. Les contrôleurs	
IV. Les Twigs	TP3: Twig
V. ORM Doctrine	TP4: Doctrine / TP5: DQL
VI. Les Formulaires	TP6: Forms

Ch. I

Introduction Symfony 6

Ch. I: Introduction Symfony 6

I.1. Introduction

I.2. Symfony 6

I.3. Architecture MVC

I.4. Architecture d’un projet

I.5. Contrôleur Frontal

I.6. Traitement d’une requête

I.1. Introduction

Symfony est un puissant framework qui permet de réaliser des sites complexes rapidement, mais de façon structurée et avec un code clair et maintenable

- le framework PHP de référence

➢ Pourquoi utiliser un framework ?
➢ Comment :
  - créer un nouveau projet de site web avec Symfony ?
  - mettre en place les environnements de test ?
  - concevoir les contrôleurs, les templates ?
  - gérer la traduction et communiquer avec une base de données via Doctrine ?

Mais tout d'abord, qu'est ce qu'un Framework ?!!!

Définition

- Un framework est une **boîte à outils** conçue par un ou plusieurs développeurs à destination d'autres développeurs

- C'est un ensemble de **composants** qui sert à créer les **fondations**, **l'architecture** et les grandes lignes d'un **logiciel**.

- Il existe des centaines de frameworks couvrant la plupart des langages de programmation. Ils sont destinés au développement de sites web ou bien à la conception de logiciels

Exemples :

❖ **PHP FrameWorks**

- CakePHP
- Laravel 4
- Symfony
- ZendFramework 2
- MKFramework…

❖ **Java FrameWorks**

- Apache Struts
- Hibernate
- JavaServer
- Faces…

❖ **Python Frameworks**

- Django

Objectif, Avantages :

- L'objectif d'un framework est généralement de **simplifier le travail des développeurs informatiques**, en leur offrant une **architecture "prête à l'emploi"** et qui leur permette de ne pas repartir de zéro à chaque nouveau projet

- Les principaux avantages sont :

  - Structurer votre projet
  - la réutilisation des codes, des bibliothèques et des composants
  - la standardisation de la programmation
  - la formalisation d'une architecture adaptée aux besoins de chaque entreprise
  - une documentation de qualité et régulièrement mise à jour

Objectif, Avantages :

- ***Abstraction de la base de donnée***
  - Un framework utilise **PDO** (PHP Data Objects) :
    Vous n'avez plus à vous soucier du type de base de données qui fonctionne derrière votre application
  - Un framework embarque généralement un **ORM** (Object Relational Mapper) :
    Vous pouvez ainsi faire des opérations courantes comme la récupération de données ou la sauvegarde d'un objet sans vous soucier du code SQL à écrire

- ***Couche d'abstraction du cache:*** Il permet de stocker les pages afin d'optimiser leur temps de chargement

- ***Gestion des formulaires:*** Générer en grande partie tous les widgets HTML, il se charge de la validation du formulaire

Objectif, Avantages :

- ***La Génération de code*** : Créer des fichiers et du contenu automatiquement par défaut afin de n'avoir pas besoin de le faire
- ***Internationalisation*** : Créer facilement le multilingue
- ***Moteur de template :*** Intégrer un moteur de templates. Celui-ci permet de simplifier grandement l'écriture de votre code HTML
- ***Gestion d'utilisateurs:*** gérer l'authentification d'utilisateurs. Ils gèrent la connexion, la déconnexion, la création, la gestion des sessions et des droits

Ch. I: Introduction Symfony 6

I.1. Introduction

I.2. Symfony 6

I.3. Architecture MVC

I.4. Architecture d’un projet

I.5. Contrôleur Frontal

I.6. Traitement d’une requête

I.2. Symfony 6

« Symfony is a set of PHP Components, a Web Application framework, a Philosophy, and a Community — all working together in harmony. »

❖ ***Symfony Framework***

Le principal framework PHP pour créer des sites Web et des applications Web

❖ ***Symfony Components***

Un ensemble de composants découplés et réutilisables sur lesquels sont construites les meilleures applications PHP.

❖ ***Symfony Community***

Un groupe passionné de plus de 600 000 développeurs de plus de 120 pays, tous déterminés à aider PHP à surmonter l'impossible.

❖ ***Symfony Philosophy***

Adopter et promouvoir les meilleures pratiques, la normalisation et l'interopérabilité des applications.

Version 1.x

- La version 1.x ayant connu un succès notable, une refonte complète a donné naissance à la version 2.x qui tire partie des évolutions de PHP 5.3

Version 2.x

- La version 2.x offre donc un framework incluant un ORM (Doctrine 2), le moteur de templating Twig, la gestion des emails avec SwiftMailer , encore un composant de sécurité pour la gestion de l'authentification utilisateur et des permissions…

Version 3.x

- La version 3.x utilise la version 5.5.9+ de PHP. Il est plus découplé et plus réutilisable que jamais

Version 4.x | Version 5.x | Version 6.x

- La version 4.x ,,,,

- La version 5.x ,,,

- La version 6.x,,,

Ch. I: Introduction Symfony 6

I.1. Introduction

I.2. Symfony 6

I.3. Architecture MVC

I.4. Architecture d’un projet

I.5. Contrôleur Frontal

I.6. Traitement d’une requête

I.3. Architecture MVC

L'architecture applicative de gestion des interactions utilisateur est généralement mise en œuvre autour du motif de conception MVC (Modèle-Vue-Contrôleur) :

- le *Modèle* représente l'ensemble des composants qui sont chargés de réaliser des appels à des services et de mettre les résultats de l'appel à la disposition de la Vue

- la *Vue* représente l'interface utilisateur

- le *Contrôleur* gère la synchronisation entre la Vue et le Modèle. Le contrôleur réagit aux actions de l'utilisateur en effectuant les actions nécessaires sur le modèle. Le contrôleur surveille les modifications du modèle et informe la Vue des mises à jour nécessaires

I.3. Architecture MVC

L'organisation de l'exécution du code au sein de Symfony respecte le modèle MVC

- Modèle : rassemble des données du domaine, des connaissances du système. Contient les classes dont les instances doivent être vues et manipulées
- Vue : utilisé pour présenter/afficher les données du modèle dans l'interface utilisateur
- Contrôleur : contient les fonctionnalités nécessaires pour gérer et contrôler les interactions de l'utilisateur avec la vue et le modèle

I.3. Architecture MVC

Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

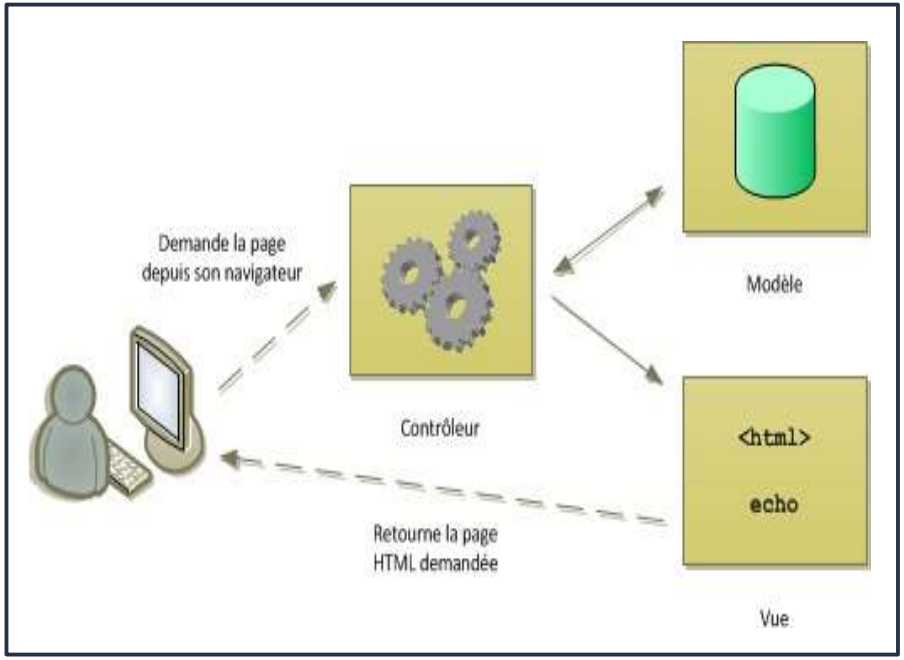
• **Modèle** : cette partie gère les *données* de votre site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autres les requêtes SQL.

• **Vue** : cette partie se concentre sur l'*affichage*. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples, pour afficher par exemple une liste de messages.

• **Contrôleur** : cette partie gère la logique du code qui prend des *décisions*. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non

I.3. Architecture MVC

Concrètement, le visiteur demandera la page au contrôleur et c'est la vue qui lui sera retournée, comme schématisé sur la figure. Bien entendu, tout cela est transparent pour lui, il ne voit pas tout ce qui se passe sur le serveur.



La requête du client arrive au contrôleur et celui-ci lui retourne la vue

Ch. I: Introduction Symfony 6

I.1. Introduction

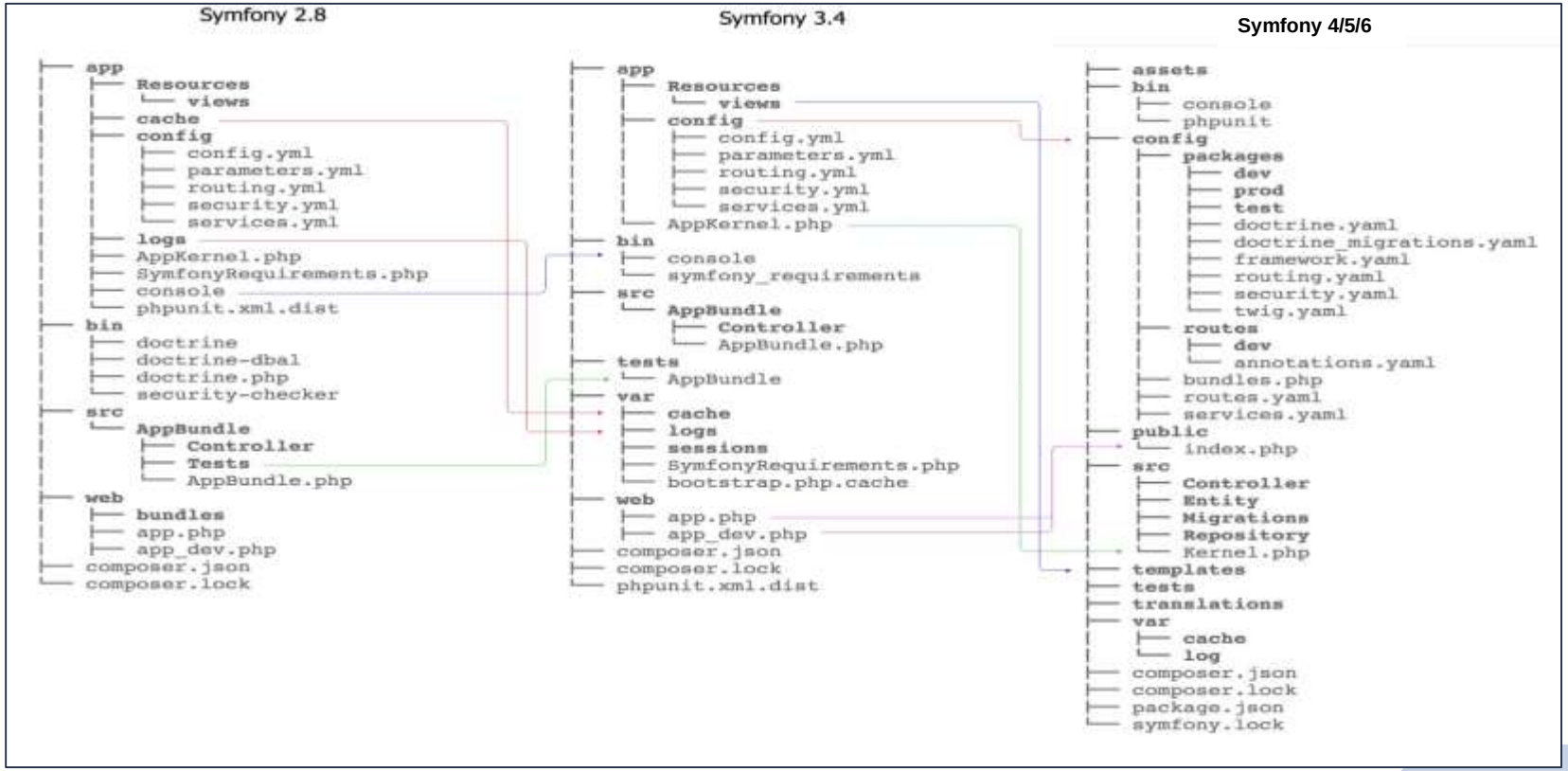
I.2. Symfony 6

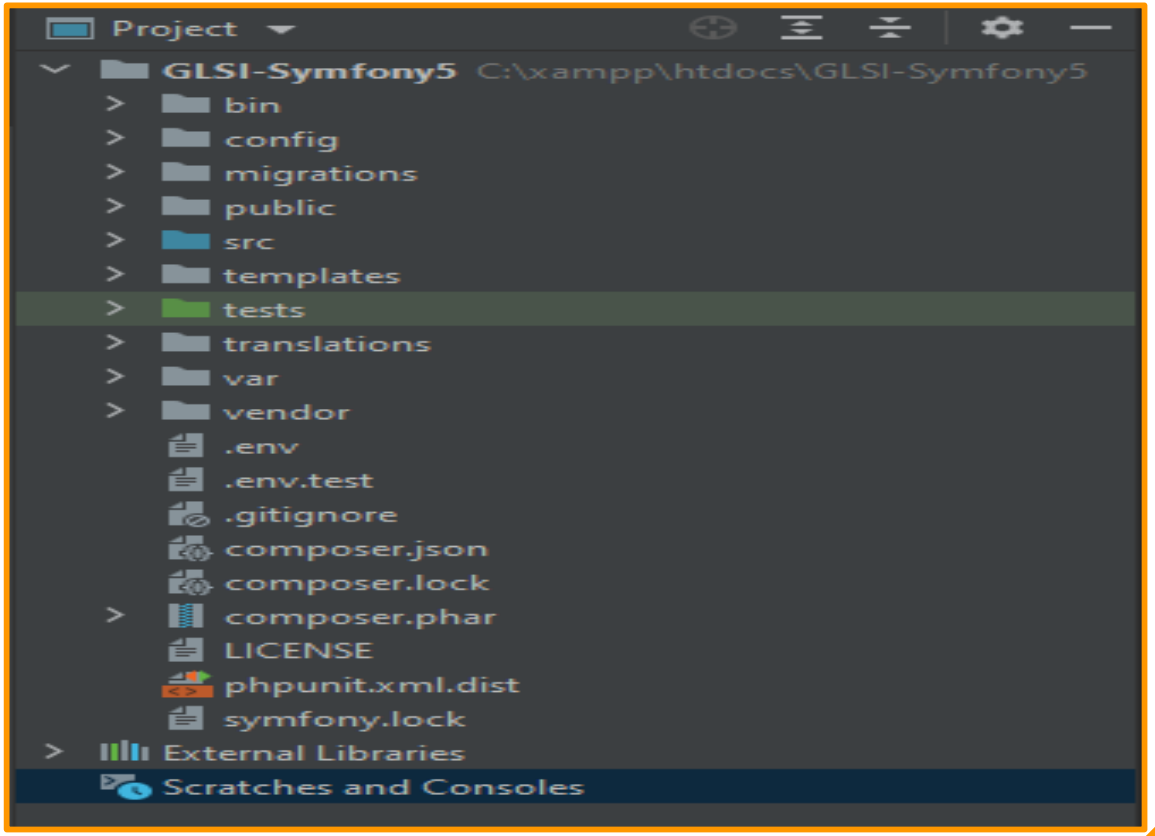
I.3. Architecture MVC

I.4. Architecture d’un projet Symfony

I.5. Contrôleur Frontal

I.6. Traitement d’une requête


Symfony 2.8
app
Resources
views
cache
config
config.yml
parameters.yml
routing.yml
security.yml
services.yml
logs
AppKernel.php
SymfonyRequirements.php
console
phpunit.xml.dist
bin
doctrine
doctrine-dbal
doctrine.php
security-checker
src
AppBundle
Controller
Tests
AppBundle.php
web
bundles
app.php
app_dev.php
composer.json
composer.lock
Symfony 3.4
app
Resources
views
config
config.yml
parameters.yml
routing.yml
security.yml
services.yml
AppKernel.php
bin
console
symfony_requirements
src
AppBundle
Controller
AppBundle.php
tests
AppBundle
var
cache
logs
sessions
SymfonyRequirements.php
bootstrap.php.cache
web
app.php
app_dev.php
composer.json
composer.lock
phpunit.xml.dist
Symfony 4/5/6
assets
bin
console
phpunit
config
packages
dev
prod
test
doctrine.yaml
doctrine_migrations.yaml
framework.yaml
routing.yaml
security.yaml
twig.yaml
routes
dev
annotations.yaml
bundles.php
routes.yaml
services.yaml
public
index.php
src
Controller
Entity
Migrations
Repository
Kernel.php
templates
tests
translations
var
cache
log
composer.json
composer.lock
package.json
symfony.lock


Project
GLSI-Symfony5 C:\xampp\htdocs\GLSI-Symfony5
bin
config
migrations
public
src
templates
tests
translations
var
vendor
.env
.env.test
.gitignore
composer.json
composer.lock
composer.phar
LICENSE
phpunit.xml.dist
symfony.lock
External Libraries
Scratches and Consoles

[bin]

Ce dossier contient les fichiers de commandes permettant, par exemple, de vider le cache Symfony, mettre à jour la base de données ou encore lancer nos tests unitaires. On utilise généralement la commande php bin/console qui affiche toutes les commande Symfony disponibles.

[config]

Toute la configuration des packages, services et routes se fera dans ce dossier. Cela permettra, entre autre, de configurer notre connexion à la base de données, mettre en place tout un système de sécurité, ou encore personnaliser les services que nous développerons. Les fichiers de configuration sont par défaut en YAML, même s'il est tout à fait possible d'utiliser PHP ou XML.

[public]

C'est le point d'entrée de l'application : chaque requête / demande passe forcément par ce dossier et le fichier index.php. Étant accessible par tous, il est généralement utilisé pour mettre à disposition des fichiers de ressources, principalement des images.

[src]
C'est le cœur du projet ! L'endroit où vous passerez le plus de temps à coder. Il regroupe tout le code PHP de votre application, c'est ici que vous mettrez en place toute la logique de votre application. Les dossiers qui seront obligatoires à utiliser pour le fonctionnement de l'application sont :

[Controller] : Définition des points d'entrée de votre application. Il se charge de rediriger vers les Manager / Service / Repository. Aucun traitement de données, accès à la BDD (base de données) ne doit se faire depuis un Controller (très important). Possibilité de choisir les méthodes d'entrée (GET, POST, PUT, DELETE, …) ainsi que le type de réponse retournée (JSON, XML, …).

[Entity] : Définition de la structure de votre BDD (base de données) au travers de classes. Chaque Entity représente généralement une table en BDD. La commande php bin/console doctrine:migrations nous permettra de mettre à jour notre BDD à chaque modification de l'Entity.

[Repository] : Un Repository est toujours rattaché à une Entity, il nous permet de créer nos fonctions qui iront requêter la table de notre Entity (ainsi que les tables liées). Symfony utilise l'ORM Doctrine qui permet de créer nos requêtes SQL à travers les queryBuilder (très utile si l'on déteste faire du SQL).

[templates]

Symfony utilise depuis ses débuts le moteur de templates Twig.

Les fichiers de template Twig ont comme format monfichier.html.twig et viennent rajouter quelques fonctionnalités au HTML classique.

[var]

Ici seront stockés le cache et les fichiers de log.
Il est possible dans les fichiers de config de paramétrer la mise en cache et ce que l'on écrit dans les logs.

[composer.json]

Tous les packages sont enregistrés dans ce fichier.
Ils sont installés automatiquement dans le dossier vendors lors de l'initialisation du projet mais on peut utiliser la commande composer install pour les installer manuellement si besoin.
Pour mettre à jour les packages, on utilise la commande composer update et pour ajouter un package on utilise composer require monpackage

Ch. I: Introduction Symfony

I.1. Introduction

I.2. Symfony 6

I.3. Architecture MVC

I.4. Architecture d’un projet

I.5. Contrôleur Frontal

I.6. Traitement d’une requête

I.5. Contrôleur Frontal

- Le contrôleur frontal (*front controller*) est le point d'entrée de votre application. C'est *le* fichier par lequel passent toutes vos pages.
- Dans Symfony 3, le contrôleur frontal se situe dans le répertoire /public, il s'appelle index.php.
- Il joue le rôle de dispatcheur:

« On vient de recevoir une requête, transforme-la en réponse s'il-te-plaît. »

Ch. I: Introduction Symfony 6

I.1. Introduction

I.2. Symfony 6

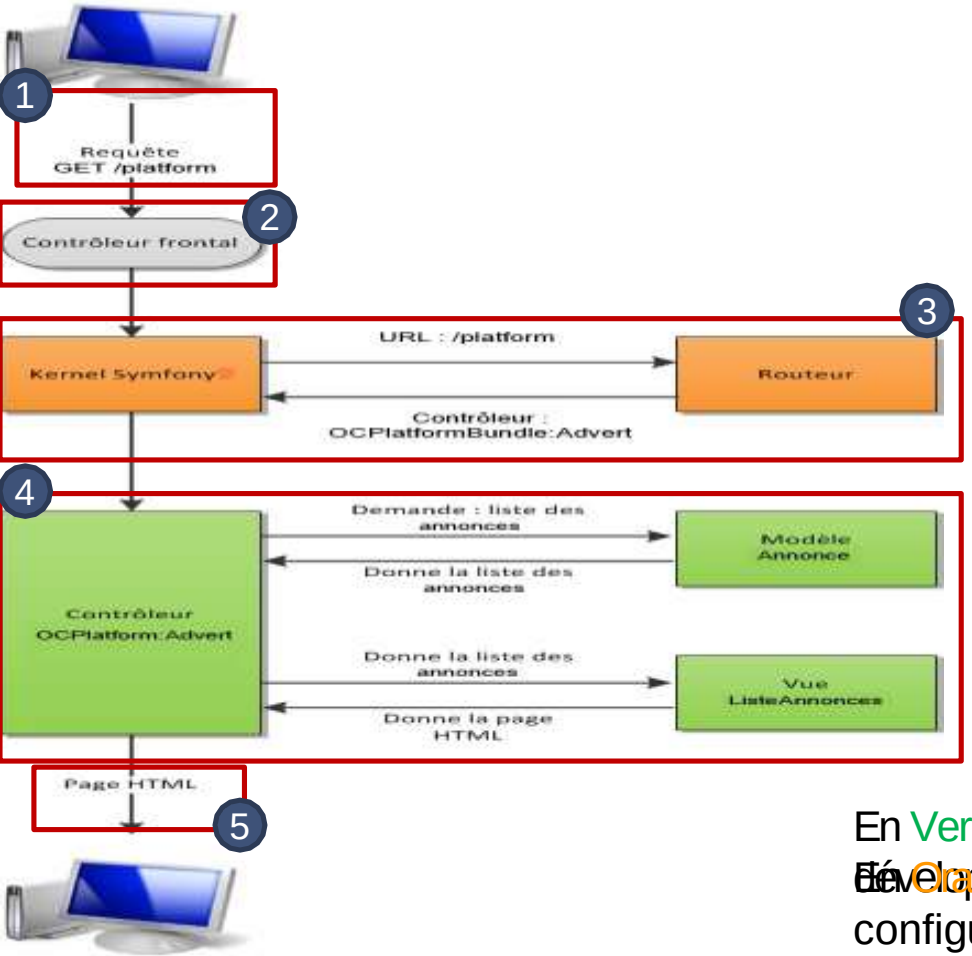
I.3. Architecture MVC

I.4. Architecture d’un projet

I.5. Contrôleur Frontal

I.6. Traitement d’une requête

I.6. Traitement d'une requête



1. Le visiteur demande la page /platform ;

2. Le contrôleur frontal reçoit la requête, charge le Kernel et la lui transmet ;

3. Le Kernel demande au Routeur quel contrôleur exécuter pour l'URL /platform. Ce Routeur est un composant Symfony qui fait la correspondance entre URL et contrôleurs. Le Routeur fait donc son travail, et dit au Kernel qu'il faut exécuter le contrôleur OCPlatform:Advert ;

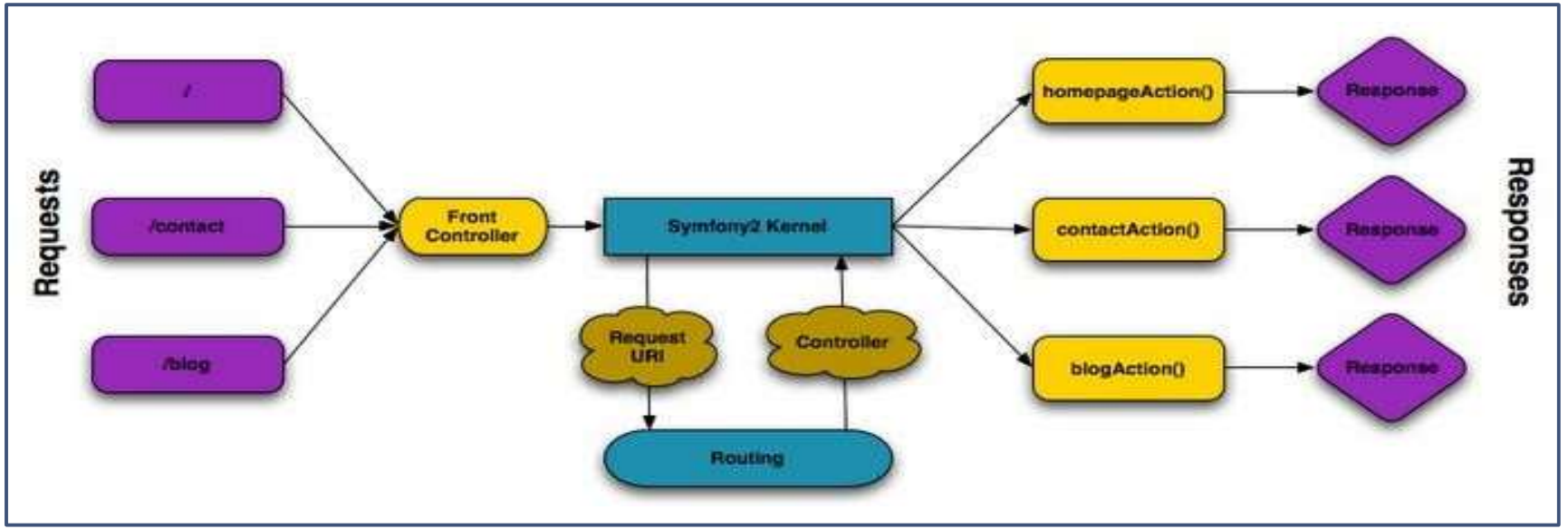
4. Le Kernel exécute donc ce contrôleur. Le contrôleur demande au modèle Annonce la liste des annonces, puis la donne à la vue ListeAnnonces pour qu'elle construise la page HTML et la lui retourne.

5. Une fois cela fini, le contrôleur envoie au visiteur la page HTML complète.

En Vert: A
Et Orange: A
configurer

I.6. Traitement d'une requête

Symfony suit un schéma simple et identique pour toutes les requêtes



Les requêtes entrantes sont interprétées par le Routing et passées aux fonctions des contrôleurs qui retournent des objets Response