Au cours de l'année 2012, un site web, appelée Augustine, fut créé. Celle-ci est un « site vitrine » qui permet de présenté en ligne les produits ou les services d'une organisation dans le but d'attirer simplement l'attention et d'éveiller l'intérêt des internautes de passage,

- sans permettre d'interactions entre l'internaute et l'organisation.
- sans permettre à l'internaute d'acheter le produit ou le service proposé.

Augustine est en réalité une voiture à énergie solaire conçus par la section CPI du lycée Léonard de Vinci à Melun et en partenariat avec le lycée Diderot à Paris. Cette voiture participe au marathon Educ'Eco qui consiste à avoir la vitesse moyenne de la course le plus proche possible de la vitesse fixé dans le délai ou le nombre de tour imparti.

Notre mission fut de refaire entièrement le site web Augustine en binôme. Ce dernier était un camarade de classe. Par contre nous avions la contrainte d'utilisé un framework Symfony II.

C'est alors que, pendant la réalisation de ce projet (application développée en première année) nous avions dû s'auto-formé sur le framework Symfony II. Par la même occasion nous avons fait connaissance avec le format MVC (Modèle, Vue, Contrôleur).

I. Qu'est-ce que Symfony

Définitions & Conseils

C'est un **framework** (= cadre de travail). Il est organisé en 4 principaux répertoires:
/app → Dossier contenant la configuration, le cache, les fichiers logs, etc…
/src → Contient le code source du projet en cours
/vendor → Contient les bibliothèques externe à notre l'application.
Les **bundles** téléchargés iront dans ce répertoire.
/web → Dossier qui contient les images, les JS, les CSS, etc… ainsi que le contrôleur frontal. Ce dossier est le seul visible par l'internaute. Les autres dossiers sont pour le développeur. Le **.htaccess** interdit justement l'accès aux autres dossiers.

Symfony repose sur le modèle **MVC** :

- **M**odèle (Model) : Représente les entités (Les tables d'une base de données).
- **V**ue (View) : Construit la logique de la vue (Qui est envoyé au client).
- **C**ontrôleur (Controller) : Traite la requête http.

Bibliothèque : Une sorte de boite qui contient des fonctions précises.

Bundle : C'est un composant Symfony qui consiste à regrouper dans un même endroit tout ce qui concerne une même fonctionnalité (Est approfondi Page 3).

Contrôleur frontal : Est app.php (pour le visiteur, environnement « prod ») ou app_dev.php (pour le développeur, environnement « dev »), c'est LE fichier par lequel passe toutes vos pages. Le contrôleur frontal est le point d'entrer de notre application, il se limite donc à appeler le noyau (Kernel) de Symfony2 en disant « On vient de recevoir une requête, traite la et transforme-la en réponse s'il-te-plaît. »

Routeur : Détermine quel contrôleur exécuter en fonction de l'URL appelée (Est approfondi Page 5)
Kernel = Noyau

Conseil 1 :
Si une erreur n'est visible n'y pour le développeur, n'y pour le visiteur, ou incompréhensible, allez voir dans app/logs/prod.log
Vider le cache « dev » → php app/console cache:clear
Vider le cache « prod » → php app/console cache:clear --env=prod
Si ne fonctionne pas, supprimer manuellement → app/cache/dev ou app/cache/prod

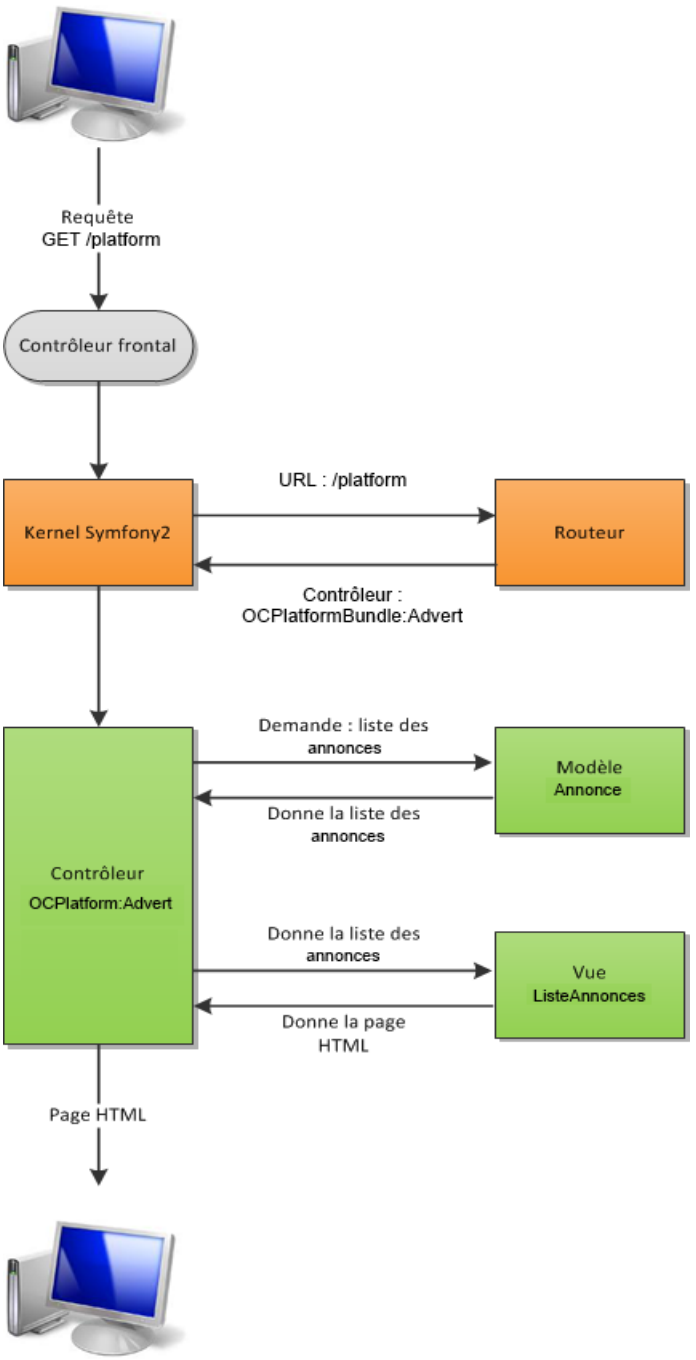
Conseil 2 :
Installation du CSS :
Copier le CSS dans src/ « Nom»/ « NomBundle»Bundle/ressources/public
Puis allez dans la console et taper (dans la racine de votre projet symfony) :
php app/console assets : install web

Schéma

Parcours complet d'une requête dans Symfony2 :

Des couleurs ont été mis en place pour distinguer les points où l'on intervient.

En gris, contrôleur frontal, on ne touchera pas !
En orange, le Kernel et le Routeur, c'est ce qu'on devra configurer.
En vert, les contrôleurs, modèle et vue, c'est ce qu'on devra développer nous-mêmes.



Le visiteur demande la page /platform.

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

Le Kernel demande au Routeur quel contrôleur exécuter pour l'URL /platform.
Ce Routeur est un composant Symfony2 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 .

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.

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

II. Qu'est-ce qu'un Bundle ?

http://knpbundles.com → Site de téléchargement Bundle Symfony II

Définitions & Structure

Un bundle Utilisateur, qui va gérer les utilisateurs ainsi que les groupes, intégrer des pages d'administration de ces utilisateurs, et des pages classiques comme le formulaire d'inscription, de récupération de mot de passe, etc.
Un bundle Blog, qui va fournir une interface pour gérer un blog sur le site. Ce bundle peut utiliser le bundle Utilisateur pour faire un lien vers les profils des auteurs des articles et des commentaires.
Un bundle Boutique, qui va fournir des outils pour gérer des produits et des commandes.

Structure d'un Bundle (Elle peut être étendue):

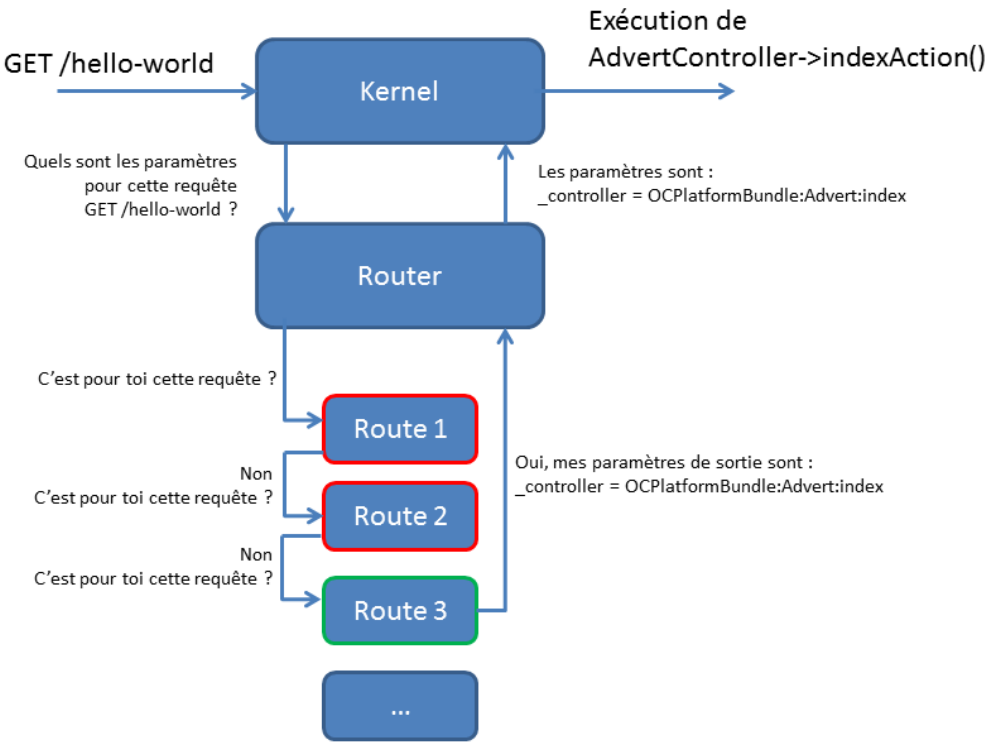
```
/Controller          | Contient vos contrôleurs
/DependencyInjection | Contient des informations sur votre bundle (ex. : chargement automatique de la configuration)
/Entity              | Contient vos modèles
/Form                | Contient vos éventuels formulaires
/Resources
--/config            | Contient les fichiers de configuration de votre bundle (nous placerons les routes ici)
-- -/public          | Contient les fichiers publics de votre bundle : fichiers CSS et JavaScript, images, etc.
--
/views               | Contient les vues de notre bundle, les templates Twig
```

Générer un Bundle

1. Lancer l'invite de commande.
2. Placez-vous dans le répertoire où vous avez mis le projet Symfony2 (Avec la commande cd)
3. Taper **php app/console generate :bundle**
4. Choisir le namespace
   Proposition de namespace. Nommez comme on veut **mais** par convention, on le composera en 3 parties.
   « Nom » le namespace racine : il vous représente vous ou votre entreprise. Vous pouvez mettre votre pseudo, le nom de votre site, celui de votre entreprise, etc. C'est un nom arbitraire. Exemple, DVIL est mon pseudonyme ;
   « Platform » le nom du bundle en lui-même : il définit ce que fait le bundle. Ici, nous créons une plateforme d'échange, nous l'avons donc simplement appelé « Platform » ;
   « Bundle » le suffixe obligatoire.
5. Choisir le nom : Symfony2 vous propose par défaut (la valeur entre les crochets). On valide.
6. Choisir la destination : On place nos bundles dans le répertoire /src
7. Choisir le format de configuration (YAML (YML), XML, PHP ou Annotations) : YAML (YML) → à choisir dans la plus part des cas.
8. Choisir quelle structure générer : Symfony propose de générer juste le minimum ou une structure plus complète. Choisir celle qu'on veut en validant **no** ou **yes** (Ici on choisit le minimum).
9. Confirmez, et c'est joué !
10. Appuyer sur Entrer pour le reste.

Que s'est-il passé ?
Symfony2 a généré la structure du *bundle puis* a enregistré notre bundle auprès du Kernel ensuite il a enregistré nos routes auprès du Routeur.

Routeur



Une route est composée au minimum de deux éléments : l'URL à faire correspondre (son path), et le contrôleur à exécuter (paramètre _controller).

Le routeur essaie de faire correspondre chaque route à l'URL appelée par l'internaute, et ce dans l'ordre d'apparition des routes : la première route qui correspond est sélectionnée.

Une route peut contenir des paramètres, facultatifs ou non, représentés par les accolades {paramètre}, et dont la valeur peut être soumise à des contraintes via la section requirements.

Le routeur est également capable de générer des URL à partir du nom d'une route, et de ses paramètres éventuels.

Contrôleur

Type de paramètres	Méthode Symfony2	Méthode traditionnelle	Exemple
Variables d'URL	$request->query	$_GET	$request->query->get('tag')
Variables de formulaire	$request->request	$_POST	$request->request->get('tag')
Variables de cookie	$request->cookies	$_COOKIE	$request->cookies->get('tag')
Variables de serveur	$request->server	$_SERVER	$request->server->get('REQUEST_URI')
Variables d'entête	$request->headers	$_SERVER['HTTP_*']	$request->headers->get('USER_AGENT')
Paramètres de route	$request->attributes	n/a	On utilise $id dans les arguments de la méthode

Twig

Première chose à savoir sur Twig : *vous pouvez afficher des variables et pouvez exécuter des expressions*. Ce n'est pas la même chose :
{{ … }} *Affiche* quelque chose ;
{% … %} *Fait* quelque chose ;
{# … *#} n'affiche rien* et *ne fait rien* : c'est la syntaxe pour les commentaires, qui peuvent être sur plusieurs lignes.

Description	Exemple Twig	Équivalent PHP
Afficher une variable	Pseudo : {{ pseudo }}	Pseudo : <?php echo $pseudo; ?>
Afficher l'index d'un tableau	Identifiant : {{ user['id'] }}	Identifiant : <?php echo $user['id']; ?>
Afficher l'attribut d'un objet, dont le getter respecte la convention $objet->getAttribut()	Identifiant : {{ user.id }}	Identifiant : <?php echo $user->getId(); ?>
Afficher une variable en lui appliquant un filtre. Ici, « upper » met tout en majuscules :	Pseudo en majuscules : {{ pseudo|upper }}	Pseudo en lettre majuscules : <?php echo strtoupper($pseudo); ?>
Afficher une variable en combinant les filtres. « striptags » supprime les balises HTML. « title » met la première lettre de chaque mot en majuscule. Notez l'ordre d'application des filtres, ici striptags est appliqué, puis title.	Message : {{ news.texte|striptags|title }}	Message : <?php echo ucwords(strip_tags($news->getTexte())); ?>
Utiliser un filtre avec des arguments. Attention, il faut que datesoit un objet de typeDatetime ici.	Date : {{ date|date('d/m/Y') }}	Date : <?php echo $date->format('d/m/Y'); ?>
Concaténer	Identité : {{ nom ~ " " ~ prenom }}	Identité : <?php echo $nom.' '.$prenom; ?>

Filtre (Twig)

Filtre	Description	Exemple Twig
Upper	Met toutes les lettres en majuscules.	{{ var|upper }}
Striptags	Supprime toutes les balises XML.	{{ var|striptags }}
Date	Formate la date selon le format donné en argument. La variable en entrée doit être une instance de Datetime.	{{ date|date('d/m/Y') }} Date d'aujourd'hui : {{ "now"|date('d/m/Y') }}
Format	Insère des variables dans un texte, équivalent à printf.	{{ "Il y a %s pommes et %s poires"|format(153, nb_poires) }}
Length	Retourne le nombre d'éléments du tableau, ou le nombre de caractères d'une chaîne.	Longueur de la variable : {{ texte|length }} Nombre d'éléments du tableau : {{ tableau|length }}

Un moteur de templates tel que Twig permet de bien séparer le code PHP du code HTML, dans le cadre de l'architecture MVC.

La syntaxe {{ var }}*affiche* la variable var.

La syntaxe {% if %} *fait* quelque chose, ici une condition.

Twig offre un système d'héritage (via {% extends %}) et d'inclusion (via {{ include() }} et {{ render() }}) très intéressant pour bien organiser les templates.

Le modèle triple héritage est très utilisé pour des projets avec Symfony2.

Composer

Installer (se placer dans le /www de Wamp)
```
php -r "eval('?>'.file_get_contents('http://getcomposer.org/installer'));"
```

Vérifier la version
```
php composer.phar --version
```

Mise à jour composer.phar
```
php composer.phar self-update
```

Mise à jour composer.json (se placer dans le projet)
```
composer update
```

Valeur	Exemple	Description
Un numéro de version exact	"2.0.17"	Ainsi, Composer téléchargera cette version exacte.
Une plage de versions	">=2.0,<2.6"	Ainsi, Composer téléchargera la version la plus à jour, à partir de la version 2.0 et en s'arrêtant avant la version 2.6. Par exemple, si les dernières versions sont 2.4, 2.5 et 2.6, Composer téléchargera la version 2.5.
Une plage de versions sémantique	"~2.1"	Ainsi, Composer téléchargera la version la plus à jour, à partir de la version 2.1 et en s'arrêtant avant la version 3.0. C'est une façon plus simple d'écrire ">=2.1,<3.0" avec la syntaxe précédente. C'est la façon la plus utilisée pour définir la version des dépéndances.
Un numéro de version avec joker « * »	"2.0.*"	Ainsi, Composer téléchargera la version la plus à jour qui commence par 2.0. Par exemple, il téléchargerait la version 2.0.17, mais pas la version 2.1.1.
Un nom de branche "dev-XXX"		C'est un cas un peu particulier, où Composer ira chercher la dernière modification d'une branche Git en particulier. N'utilisez cette syntaxe que pour les bibliothèques dont il n'existe pas de vraie version. Vous verrez assez souvent "dev-master", où "master" correspond à la branche principale d'un dépôt Git.

Doctrine (Les entitées)

Type Doctrine	Type SQL	Type PHP	Utilisation
string	VARCHAR	string	Toutes les chaînes de caractères jusqu'à 255 caractères.
integer	INT	integer	Tous les nombres jusqu'à 2 147 483 647.
smallint	SMALLINT	integer	Tous les nombres jusqu'à 32 767.
bigint	BIGINT	string	Tous les nombres jusqu'à 9 223 372 036 854 775 807. Attention, PHP reçoit une chaîne de caractères, car il ne supporte pas un si grand nombre (suivant que vous êtes en 32 ou en 64 bits).
boolean	BOOLEAN	boolean	Les valeurs booléennes true et false.
decimal	DECIMAL	double	Les nombres à virgule.
date ou datetime	DATETIME	objet DateTime	Toutes les dates et heures.
time	TIME	objet DateTime-	Toutes les heures.
text	CLOB	string	Les chaînes de caractères de plus de 255 caractères.
object	CLOB	Type de l'objet stocké	Stocke un objet PHP en utilisant serialize/unserialize.
array	CLOB	array	Stocke un tableau PHP en utilisant serialize/unserialize.
float	FLOAT	double	Tous les nombres à virgule. Attention, fonctionne uniquement sur les serveurs dont la locale utilise un point comme séparateur.

Paramètre	Valeur par défaut	Utilisation
type	string	Définit le type de colonne comme nous venons de le voir.
name	Nom de l'attribut	Définit le nom de la colonne dans la table. Par défaut, le nom de la colonne est le nom de l'attribut de l'objet, ce qui convient parfaitement. Mais vous pouvez changer le nom de la colonne, par exemple si vous préférez « isExpired » en attribut, mais « is_expired » dans la table.
length	255	Définit la longueur de la colonne. Applicable uniquement sur un type de colonne string.

Paramètre	Valeur par défaut	Utilisation
unique	false	Définit la colonne comme unique. Par exemple sur une colonne e-mail pour vos membres.
nullable	false	Permet à la colonne de contenir des NULL.
precision	0	Définit la précision d'un nombre à virgule, c'est-à-dire le nombre de chiffres en tout. Applicable uniquement sur un type de colonne decimal.
scale	0	Définit le *scale* d'un nombre à virgule, c'est-à-dire le nombre de chiffres après la virgule. Applicable uniquement sur un type de colonne decimal.

Créer base de donné : `php app/console doctrine:database:create`

Créer table : `php app/console doctrine:schema:update --dump-sql`

Exécution de la requête : `php app/console doctrine:schema:update –force`

Génère ce qui manque (ex : get/set) : `php app/console doctrine:generate:entities OCPlatformBundle:Advert`

Enregistrer un changement dans la base de données : `php app/console doctrine:schema:update --dump-sql`

À retenir
À chaque modification du *mapping* des entités, ou lors de l'ajout/suppression d'une entité, il faudra répéter cette commande pour mettre à jour la base de données **:**
doctrine:schema:update --force