Formation Symfony DREAL Occitanie

Sommaire

PHP Objet

Un objet

Un composant dans un programme qui regroupe des variables (aspect statique) et des fonctions (aspect comportement).

A propos des variables on parle de (selon des personnes ou les livres) :

- attribut,
- propriété,
- ou champ

A propos des fonction on parle de méthodes.

Concrètement on cherche à appliquer des concepts de la vie de tous les jours à la programmation. Exemple, un ordinateur est un objet qui a des caractérisques (propriétés, ex: marque, modèle), et un comportement (méthodes, ex : démarrer, déplacer le curseur) et un type (classe).

Une classe

Le type d'un objet et ce qui permet de le créer (construire). On compare souvent les classes à un moule (à gâteau), permet de créer des objets qui ont une certaine forme.

Dans un programme on peut avoir une classe (ex : Ordinateur) et plusieurs objets créés en mémoire à partir de cette classe.

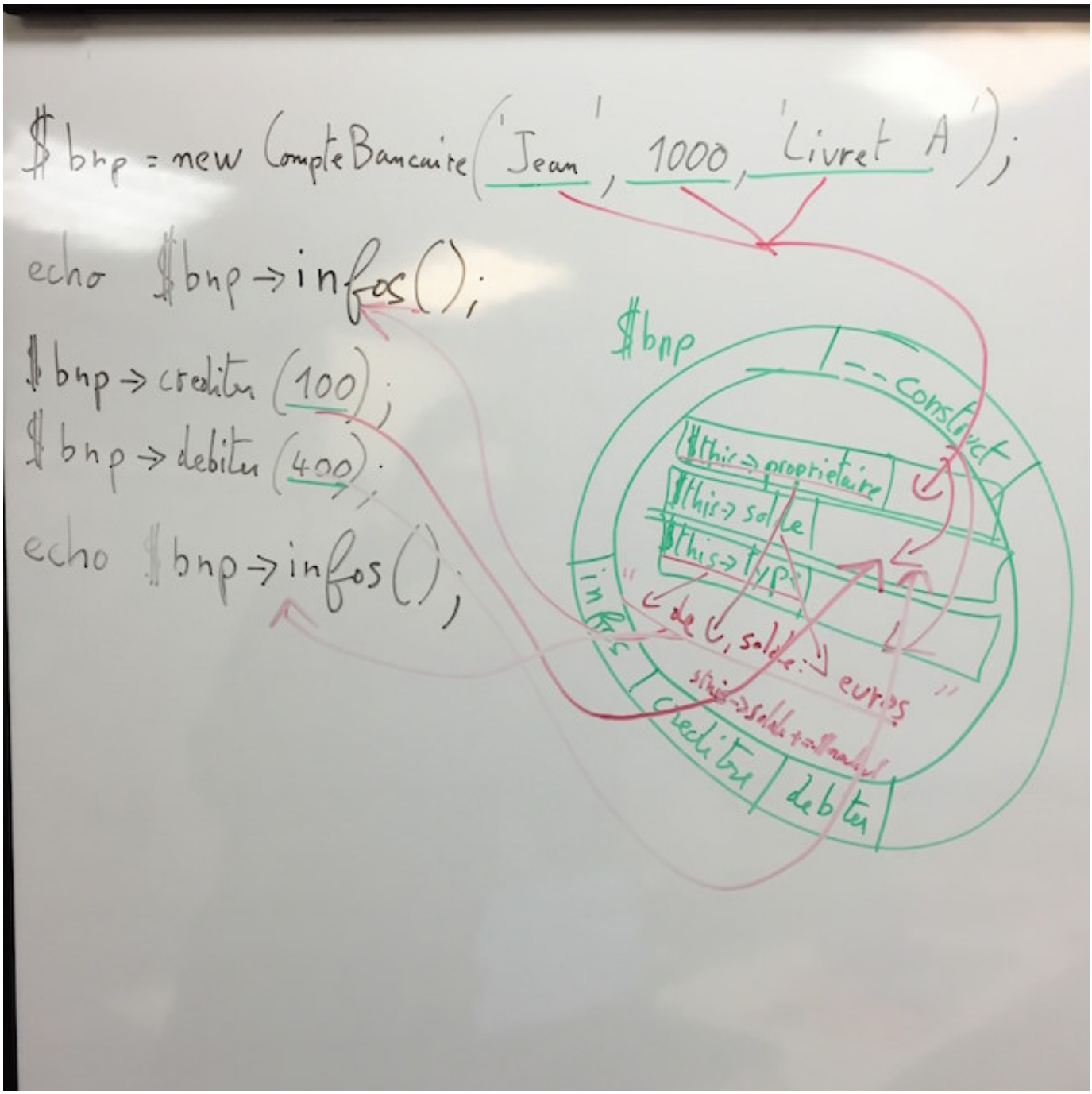
Un classe est la déclaration d'un concept (ex: Ordinateur) et l'objet son utilisation.

Principe d'encapsulation

Les propriétés d'une classe ne doivent être accesibles qu'au travers de méthodes (jamais publiques en général protégées en PHP).

Intérets :

- moins il y de choses à connaitre à propos de la classe, plus elle est simple à utiliser (imaginer des propriétés complexes comme des sockets dans le cas d'une connexion à une base de données)
- permet au développeur qui a créé la classe que les autres développeurs qui vont l'utiliser n'intéragissent que avec vos propriétés de la manière dont vous l'avez prévu (ex : j'aimerai que la marque soit toujours en majuscule, avoir des propriétés en lecture seule...)



Getters/Setters (accesseurs en français)

Comme les propriétés ne sont pas publiques, si le développeur de la classe souhaite fournir un accès à ces propriétés il va créer des getters/setters. Ce sont des méthodes dont le nom est conventionné, pour la lecture : getLeNomDeLaPropriete, pour l'écriture setLeNomDeLaPropriete. Le getter retourne la propriété, le setter prend un paramètre d'entrée et modifie la propriété. Avec des IDEs comme NetBeans ou PHPStorm on peut les générer automatiquement.

Associations

Quand on travaille en objet, on va lier les objets entre eux par des associations, ex : un contact qui travaille dans une société sera lié à la société.

L'association peut être unidirectionnelle si elle se fait que dans un sens. Ex : le contact connait la société, mais la société ne connait pas ses contacts. Si le lien peut se faire dans les 2 sens, elle est dite bidirectionnelle.

En PHP web, on ne fera le lien que dans un sens, si vous n'avez pas besoin de l'association ne la faites pas.

Cardinalité

Il y a plusieurs types d'associations, la cardinalité permet de la déterminer.

En gros, pour un objet il faut savoir combien on retrouvera d'objets liés en face.

Ex : une famille

Question : Pour un mari combien de femmes minimum et maximum ? Réponse : ça dépend, en France ? à un instant précis ? etc.. En général les maquettes répondent à ce genre de question, mais il peut être nécessaire de préciser avec quelqu'un qui connait bien le métier de l'application modélisée.

Disons : Pour 1 mari on a entre 0 et 1 femme, pour 1 femme on a entre 0 et 1 mari. Le nom de la relation est déterminé à partir des cardinalités maximum, ici c'est une relation 1..1

Question : Pour une femme combien d'enfants ? Réponse : 0 minimum et n maximum (ou *), pour un enfant 1, relation 1..n

Question : pour un enfant combien de frères et soeurs ? Réponse : 1 enfant à entre 0 et n frères et soeurs, et en face un frère aura entre 0 et n frères et soeurs donc relation dite n..m (plusieurs - plusieurs).

En PHP :

Dans une classe si on a besoin de faire le lien vers une autre.

Soit il y a un seul objet possible dans le lien (ex : 1 contact lié à 1 société), c'est une simple variable

```php
<?php
class Contact
{
    protected $prenom;
    protected $nom;
    protected $email;
    protected $telephone;

    /**
     *
     * @var Societe
     */
    protected $societe;

    public function getSociete()
    {
        return $this->societe;
    }

    public function setSociete(Societe $societe)
    {
        $this->societe = $societe;
        return $this;
    }

    // getters/setters et autres ...
}
```

Soit il y a plusieurs objets possibles dans le lien (ex : plusieurs contact dans 1 société), c'est une variable de type tableau.

```php
<?php
class Societe
{
    protected $nom;
    protected $ville;

    /**
     *
     * @var Contact[]
     */
    protected $contacts = [];

    public function getContacts()
    {
        return $this->contacts;
    }

    public function addContact(Contact $contact)
    {
        // ajoute à la fin du tableau
        $this->contacts[] = $contact;
    }

    // getters/setters
}
```

Dans Symfony (Doctrine) les associations sont nommées OneToOne, ManyToOne (la plus répandue), OneToMany (l'inverse de ManyToOne), ManyToMany.

Composer

Composer est un programme PHP qui permet de télécharger des dépendances. Ex : le framework Slim et ses dépendances (ex : `Psr\Http\Message`)

Plutôt que de télécharger chaque dépendance manuellement puis de les installer manuellement et de créer un autoloader manuellement.

Toutes les commandes à partir de maintenant s'exécutent à partir de la racine du projet. La commande `cd` (Change Directory) permet de se placer à la racine, ex : `cd c:\xampp\htdocs\SlimHelloworld` (pour obtenir le chemin d'un fichier un d'un dossier, vous pouvez le glisser dans la fenêtre de commandes)

Ajouter une dépendance à un projet

`composer require NOM_DE_LA_DEPENDANCE`

Pour connaitre le nom de votre dépendance :

- lire la doc du projet (ex : Slim)
- chercher sur Packagist.org (annuaire de paquets installable via composer)

Création d'un squelette

`composer create-project NOM_DU_PAQUET CHEMIN_VERS_LE_PROJET`

composer.json

Enregistre les dépendances et jusqu'à quelle version on est prêt à mettre à jour

Plus d'infos sur http://composer.json.jolicode.com

Versionnage Sémantique

Un numéro de version X.Y.Z, (ex PHP 5.6.17)

X : Version Majeure Y : Version Mineure Z : Version Correctif

Version Correctif : correction de bugs ou de sécurité (sauf surprise pas de risque de bugs en montant de version)

Version Mineure : en général, nouveautés qui n'impactent pas le code existant, ex : nouvelles classes, nouvelles méthodes, nouveaux paramètres optionnels, parfois quelques changements (dans ce cas lire le guide de migration), la migration se passe rapidement.

Version Majeure : changements dans l'API, modification/suppression de classes méthodes, migration peut être très longue.

Exemple

```
{
    "require": {
        "php": ">=5.5.0",
        "slim/slim": "~3.0",
        "slim/php-view": "^2.0",
        "monolog/monolog": "^1.17"
    }
}
```

`"php": ">=5.5.0"` : Au moins PHP 5.5

`"slim/slim": "~3.0"` : Incrémente que les versions correctifs (3.0.3 mais pas 3.1)

`"slim/php-view": "^2.0"` : Incrémente les versions correctifs et mineures

composer.lock

Vérouille les versions installées pour éviter les mises à jours lors du passage en production par exemple.

Faire les mises à jours

`composer update` : va lire le fichier composer.json et faire les charges les versions les plus récentes

Réinstaller les versions déjà installées (ex : en production)

`composer install` : va lire le fichier composer.lock et réinstaller les mêmes version exactement (limite le risque de bugs)

Frameworks web

Un framework est un regroupement de bibliothèques qui chacune se destine à une tâche précise (envoyer un mail, gérer les formulaires, gérer les URL...).

Dans un framework web on retrouve un cadre de travail (on me dit où créer le fichier), ce qui oblige à avoir la bonne architecture. Le cadre de travail se matérialise par la doc et surtout par un squelette d'application (sources d'exemple avec déjà les bons répertoires).

Principaux Framework web

- Java : Struts (2000), Spring (2003), GWT (2006), Play (2007)…
- Ruby : Ruby on Rails (2005), Sinatra (2007)…
- Python : Django (2005)...
- JavaScript (serveur) : Express (2009), Meteor (2012), Sails.js (2012)…
- JavaScript (client) : AngularJS (2010), Ember.js (2011), React (2014), Angular (2016), VueJS (2016)...

Frameworks web PHP

On peut les ranger en 3 catégories :

- Micro-frameworks : Font peu de chose, si ce n'est mettre en place une architecture, souvent MVC (les plus connus : Slim, Silex, Expressive)
- Convention over configuration frameworks : Les conventions dominent, placer un fichier, nommé d'une certaine façon, dans le bon répertoire aura pour conséquence de créer une page, activer quelque chose (principaux framework : Laravel, CakePHP, CodeIgniter,
- Frameworks Entreprise : (le plus connu : Zend Framework 2+)

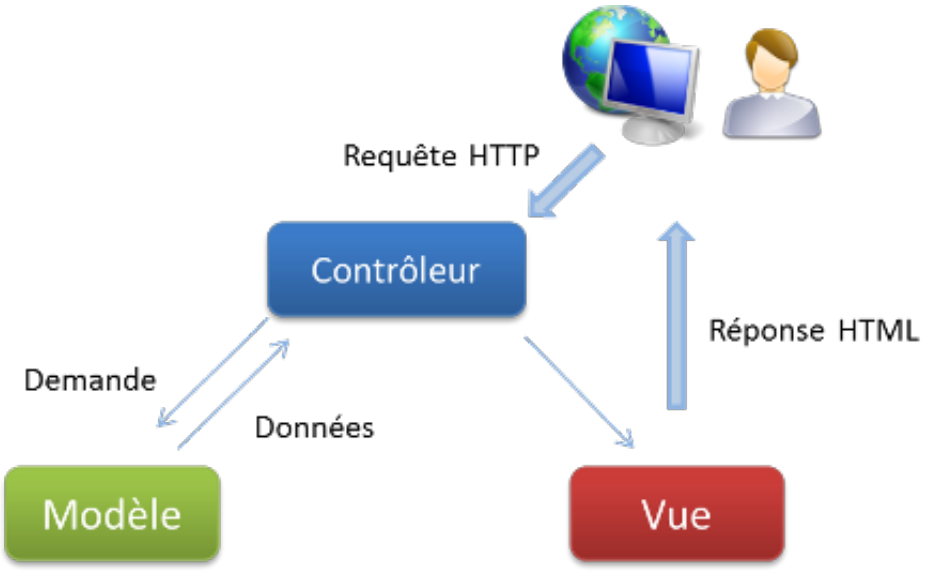
Symfony lui est à la limite entre un Framework Entreprise et un Convention over configuration framework, facile à prendre en main et reste pourtant extrêmement personnalisable !

MVC : Modèle, Vue, Contrôleur

En anglais, Model View Controller ou MVC est un Design Pattern (réponse standard à un problème de programmation) architectural qui sépare le code en 3 grandes catégories :

- Model : Données (Entity), Accès aux Données (Mapper), Validation...
- View : Rendu
  - HTML (ou autre)
  - echo
  - if .. else
  - foreach
  - appels à des fonctions de filtrages ou de rendu (htmlspecialchars)

- Controller :
  - Chef d'orchestre (étant sur la page liste de contacts, je dois appeler la méthode findAll de VoitureMapper et transmettre le résultat à la vue index.phtml)
  - Gérer les problèmes HTTP (récupérer des paramètres dans l'URL, faire des redirections...)



https://fr.wikipedia.org/wiki/Modèle-vue-contrôleur

Symfony : Installation

Le squelette d'application de Symfony peut se créer avec composer et la commande :

```
cd c:\xampp\htdocs
composer create-project symfony/framework-standard-edition DOSSIER_A_CREER
```

A la fin de l'installation, un script interactif nous propose d'éditer les paramètres de l'application.

```
Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_host (127.0.0.1):
database_port (null):
database_name (symfony): contacts
database_user (root):
database_password (null):
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):
secret (ThisTokenIsNotSoSecretChangeIt): 04eb46846cba76ae7edbe407ae812120ea1927d8
```

La clé `secret` est utilisée par les composants de sécurité de Symfony. Il faut qu'elle soit différente pour chaque application. Vous pouvez en générer une sur : http://nux.net/secret

Attention : vérifier les droits de `var/session`, `var/logs`, `var/cache`. Cf le Symfony Book chapître *Setting up Permissions*

Vérifier la config du serveur

En ligne de commande : `php bin\symfony_requirements`

Et en PHP Web en affichant la page config.php dans le navigateur.

Routeur : Créer de nouvelles pages

Comme dans tous les frameworks Symfony inclus un Router. Une route (une page) est l'association entre une URL et une fonction.

Dans Symfony, les fonctions sont des méthodes d'une Classe Contrôleur et on appelle ces méthodes des actions.

La classe est un regroupement de pages par catégorie.

Ex : La catégorie utilisateur et ses pages

- inscription
- connexion
- déconnexion
- changement de mot de passe
- afficher son profil
- modifier ses informations

Pour créer un contrôleur il faut créer une classe dans un Bundle (ex : AppBundle) dans un dossier Controller et suffixée par Controller et en général hérite Symfony\Bundle\FrameworkBundle\Controller\Controller.

Idéalement en ligne de commande :

```
php bin\console generate:controller
```

Exemple :

```
MBP-de-Romain:SFContacts romain$ php bin/console generate:controller


  Welcome to the Symfony2 controller generator



Every page, and even sections of a page, are rendered by a controller.
This command helps you generate them easily.

First, you need to give the controller name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post

Controller name: AppBundle:Contact

Determine the format to use for the routing.

Routing format (php, xml, yml, annotation) [annotation]:

Determine the format to use for templating.

Template format (twig, php) [twig]:

Instead of starting with a blank controller, you can add some actions now. An acti
on
is a PHP function or method that executes, for example, when a given route is matc
hed.
Actions should be suffixed by Action.


New action name (press <return> to stop adding actions): listAction
Action route [/list]: /contacts
Template name (optional) [AppBundle:Contact:list.html.twig]:

New action name (press <return> to stop adding actions): showAction
Action route [/show]: /contacts/{id}
Template name (optional) [AppBundle:Contact:show.html.twig]:
```

```
New action name (press <return> to stop adding actions):


  Summary before generation


You are going to generate a "AppBundle:Contact" controller
using the "annotation" format for the routing and the "twig" format
for templating
Do you confirm generation [yes]?


  Controller generation


Generating the bundle code: OK


  Everything is OK! Now get to work :).
```

Pour afficher les pages dans le navigateur

Dans l'environnement de dev :

```
http://localhost/chemin_jusqua_votre_dossier_symfony/web/app_dev.php/contacts
```

Dans l'environnement de prod :

```
http://localhost/chemin_jusqua_votre_dossier_symfony/web/contacts
```

Ex :

```
http://localhost/SFContacts/web/app_dev.php/contacts
```

Pour obtenir des URLs plus proche de la prod (ex : `http://monsite.com/contacts`) il faut configurer Apache

Modifier les Routes

Les routes sont définies sous la formes d'annotations (commentaires au dessus des méthodes actions).

On peut modifier les liens par exemple.

Pour ajouter une contrainte sur un paramètre, on ajoute requirements (où [1-9][0-9]* est une expressions

régulières qui limite id à des entiers positifs)ex :

```
/**
 * @Route("/societes/{id}", requirements={"id": "[1-9][0-9]*"})
 */
public function showAction($id)
{
    // ...
}
```

Afficher toutes les routes

```
php bin/console debug:router
```

Cache de Symfony

Pour améliorer les performances, Symfony ne relit pas tous les fichiers à chaque requête, ex : fichiers de config, fichiers Twig, annotations @Route

En production le cache est dit très "agressif", c'est à dire que beaucoup de choses sont mise en cache dont les Routes.

Pour prendre en compte les nouvelles pages il faut vider le cache.

Vider le cache de dev :

```
php bin\console cache:clear
```

Vider le cache de prod :

```
php bin\console cache:clear --env=prod
```

Les vues avec Twig

Twig est un moteur de templates, permet de faire le rendu des vues sans utiliser PHP, mais une syntaxe simplifiée.

Faire un lien vers une route

Sans paramètre (app*contact*list étant le nom de la route):

```
<a href="{{ path('app_contact_list') }}">Retour à la liste</a>
```

Avec un paramètre ({id: 66}, remplace {id} dans l'URL par 66):

```
<a href="{{ path('app_contact_show', {id: 66}) }}">Afficher</a>
```

HTML commun à plusieurs pages

Sur un site on a souvent le même HTML qui s'affiche sur plusieurs pages (balise head, menus, bandeaux...)

Au lieu de faire un `include 'header.php'` puis `include 'footer.php'` comme en PHP classique on faire l'inverse, depuis une page base.html.twig on va inclure la vue.

Pour faire ça il faut hériter d'un fichier de base (Layout) (ici :: veut dire le dossier views dans app/Resources/views):

```
{% extends "::base.html.twig" %}
```

Dans ce fichier on a définit des blocks :

```
{% block body %}{% endblock %}
```

Ces blocks peuvent être écrasés par une vue :

```
{% block body %}
<h1>Welcome to the Societe:list page</h1>
{% endblock %}
```

Le modèle avec Doctrine

Doctrine est une bibliothèque qui simplifie l'accès aux données. L'implémentation du Design Pattern Data Mapper.

Créer la base de données

Si la base paramétrées dans parameters.yml n'existe pas encore on peut la créer avec la commande :

```
php bin/console doctrine:database:create
```

Créer une entité

Une entité est une classe qui contient une donnée de l'application, ex : un contact.

On ici on choisi les noms de propriétés de notre entité, et un type de Mapping, c'est à dire un type de

traduction entre PHP <-> Base de données, ex :

string traduit un string PHP en VARCHAR MySQL

Pour les autres types voir : http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/basic-mapping.html#doctrine-mapping-types

```
php bin/console doctrine:generate:entity

Welcome to the Doctrine2 entity generator



This command helps you generate Doctrine2 entities.

First, you need to give the entity name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post.

The Entity shortcut name: AppBundle:Contact

Determine the format to use for the mapping information.

Configuration format (yml, xml, php, or annotation) [annotation]:

Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically (named id).

Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, binary, blob, guid.

New field name (press <return> to stop adding fields): prenom
Field type [string]:
Field length [255]: 40
Is nullable [false]:
Unique [false]:

New field name (press <return> to stop adding fields): nom
Field type [string]:
Field length [255]: 40
Is nullable [false]:
Unique [false]:

New field name (press <return> to stop adding fields): email
Field type [string]:
Field length [255]: 80
Is nullable [false]: true
Unique [false]: true
```

```
New field name (press <return> to stop adding fields): telephone
Field type [string]:
Field length [255]: 20
Is nullable [false]: true
Unique [false]:

New field name (press <return> to stop adding fields):


  Entity generation


> Generating entity class src/AppBundle/Entity/Contact.php: OK!
> Generating repository class src/AppBundle/Repository/ContactRepository.php: OK!


  Everything is OK! Now get to work :).
```

Générer le code SQL

Afficher le code à exécuter :

```
php bin/console doctrine:schema:update --dump-sql
```

Exécuter les requêtes après vérification

```
php bin/console doctrine:schema:update --force
```

Modifier une entité

Faites vos modifications dans le code et les annotations, ex:

```
/**
 * @var string
 *
 * @ORM\Column(name="categorie", type="string", length=20)
 */
private $categorie;
```

Puis générer les Getters/Setters, peut se faire avec NetBeans mais le mieux est d'utiliser la commandes suivante (qui sait créer les add...) :

```
php bin/console doctrine:generate:entities AppBundle
```

Puis mettre à jour les tables avec les commandes :

```
php bin/console doctrine:schema:update --dump-sql

php bin/console doctrine:schema:update --force
```

ParamConverter

Si une URL contient la clé primaire, on peut faire traduire l'URL directement en objet pour le contrôleur :

```
/**
 * @Route("/actualites/{id}")
 */
public function showAction(\AppBundle\Entity\Actualite $actu)
{
    return $this->render('AppBundle:Actualite:show.html.twig', array(
        'actu' => $actu
    ));
}
```

Bundle : réutilisation de code

Un bundle est un dossier réutisable entre plusieurs projets Symfony.

Idéalement quand un code (pages, fonctions, commandes...) peut être réutilisé dans différentes applications Symfony, on créé un nouveau Bundle, ex :

- pages liées aux utilisateurs (inscription, connexion, déconnexion...)
- pages et formulaires de newsletter
- fonctions twig pour simplifier l'écriture de composants Bootstrap
- commandes de génération de codes
- ...

Création d'un Bundle

```
php bin/console generate:bundle



Welcome to the Symfony bundle generator!


Are you planning on sharing this bundle across multiple applications? [no]: yes
```

```
Your application code must be written in bundles. This command helps
you generate them easily.

Each bundle is hosted under a namespace (like Acme/BlogBundle).
The namespace should begin with a "vendor" name like your company name, your
project name, or your client name, followed by one or more optional category
sub-namespaces, and it should end with the bundle name itself
(which must have Bundle as a suffix).

See http://symfony.com/doc/current/cookbook/bundles/best_practices.html#bundle-nam
e for more
details on bundle naming conventions.

Use / instead of \  for the namespace delimiter to avoid any problem.

Bundle namespace: Prepavenir\BootstrapBundle

In your code, a bundle is often referenced by its name. It can be the
concatenation of all namespace parts but it's really up to you to come
up with a unique name (a good practice is to start with the vendor name).
Based on the namespace, we suggest PrepavenirBootstrapBundle.

Bundle name [PrepavenirBootstrapBundle]:

Bundles are usually generated into the src/ directory. Unless you're
doing something custom, hit enter to keep this default!

Target Directory [src/]:

What format do you want to use for your generated configuration?

Configuration format (annotation, yml, xml, php) [xml]: annotation


  Bundle generation


> Generating a sample bundle skeleton into src/Prepavenir/BootstrapBundle OK!
> Checking that the bundle is autoloaded: OK
> Enabling the bundle inside app/AppKernel.php: OK
> Importing the bundle's routes from the app/config/routing.yml file: OK


  Everything is OK! Now get to work :).
```

Activer un bundle

Il faut ajouter une ligne du genre :

```
new Prepavenir\BootstrapBundle\PrepavenirBootstrapBundle(),
```

Dans le fichier app/AppKernel.

Activer les routes (pages) d'un bundle

Dans le fichiers app/config/routing.yml (dans tous les environnements dev, prod...), ou dans app/config/routing_dev.yml (que dans l'environnement de dev).

```
prepavenir_bootstrap:
  resource: "@PrepavenirBootstrapBundle/Controller/"
  type:     annotation
  prefix:   /
```

Copier le dossier public d'un bundle ou créer un lien symbolique

Dans un bundle on peut créer un dossier Resources/public qui contient des fichiers à placer dans web (ex: CSS, images, JavaScript), permet de limiter le nombre de manipulation de fichier à faire, ex: BootstrapBundle contient déjà les fichiers CSS.

Pour copier les fichiers :

```
php bin/console assets:install
```

Ou créer des liens symboliques (raccourcis mais que sur Mac/Linux) :

```
php bin/console assets:install --symlink
```

Formulaires

Générer un type de formulaire à partir d'une entité

`php bin/console doctrine:generate:form AppBundle:Societe` où `AppBundle:Societe` est le nom raccourci de l'entité.

Créer et valider le formulaire dans le contrôlleur

```
/**
 * @Route("/societes/ajouter")
 */
public function addAction(\Symfony\Component\HttpFoundation\Request $request)
{
    $form = $this->createForm('AppBundle\Form\SocieteType');
    // avec la complétion :
    // $form = $this->createForm(\AppBundle\Form\SocieteType::class)

    $form->handleRequest($request);

    if ($form->isValid()) {
        $data = $form->getData();

        $em = $this->getDoctrine()->getEntityManager();

        $em->persist($data);
        $em->flush();

        return $this->redirectToRoute('app_societe_list');
    }

    return $this->render('AppBundle:Societe:add.html.twig', array(
        'societeForm' => $form->createView()
    ));
}
```

Afficher le formulaire dans la vue

```
{% extends "::base.html.twig" %}

{% form_theme societeForm "bootstrap_3_layout.html.twig" %}

{% block title %}Ajouter une société{% endblock %}

{% block body %}
<h1>Ajouter une société</h1>

{{ form_start(societeForm) }}

    {{ form_row(societeForm.nom) }}
    {{ form_row(societeForm.siteWeb) }}
    {{ form_row(societeForm.telephone) }}

    <div>
        <button type="submit" class="btn btn-primary">Ajouter</button>
    </div>

{{ form_end(societeForm) }}

{% endblock %}
```

Créer un projet entièrement en ligne de commande

```
Last login: Thu Feb  4 10:30:30 on console
MBP-de-Romain:~ romain$ composer create-project symfony/framework-standard-edition
 ~/Desktop/SFContactsFromCLI
You are running composer with xdebug enabled. This has a major impact on runtime p
erformance. See https://getcomposer.org/xdebug
Installing symfony/framework-standard-edition (v3.0.2)
  - Installing symfony/framework-standard-edition (v3.0.2)
    Downloading: 100%

Created project in /Users/romain/Desktop/SFContactsFromCLI
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
  - Installing doctrine/lexer (v1.0.1)
    Loading from cache

  - Installing doctrine/annotations (v1.2.7)
    Loading from cache

  - Installing twig/twig (v1.24.0)
    Loading from cache

  - Installing symfony/polyfill-util (v1.1.0)
```

```
  Loading from cache

- Installing paragonie/random_compat (1.1.6)
  Loading from cache

- Installing symfony/polyfill-php70 (v1.1.0)
  Loading from cache

- Installing symfony/polyfill-php56 (v1.1.0)
  Loading from cache

- Installing symfony/polyfill-mbstring (v1.1.0)
  Loading from cache

- Installing symfony/symfony (v3.0.2)
  Downloading: 100%

- Installing symfony/polyfill-intl-icu (v1.1.0)
  Loading from cache

- Installing psr/log (1.0.0)
  Loading from cache

- Installing doctrine/inflector (v1.1.0)
  Loading from cache

- Installing doctrine/collections (v1.3.0)
  Loading from cache

- Installing doctrine/cache (v1.6.0)
  Loading from cache

- Installing doctrine/common (v2.6.1)
  Loading from cache

- Installing jdorn/sql-formatter (v1.2.17)
  Loading from cache

- Installing doctrine/doctrine-cache-bundle (1.3.0)
  Loading from cache

- Installing doctrine/dbal (v2.5.4)
  Loading from cache

- Installing doctrine/doctrine-bundle (1.6.1)
  Loading from cache

- Installing doctrine/instantiator (1.0.5)
```

```
    Loading from cache

  - Installing doctrine/orm (v2.5.4)
    Loading from cache

  - Installing incenteev/composer-parameter-handler (v2.1.2)
    Loading from cache

  - Installing sensiolabs/security-checker (v3.0.2)
    Loading from cache

  - Installing sensio/distribution-bundle (v5.0.3)
    Loading from cache

  - Installing sensio/framework-extra-bundle (v3.0.12)
    Loading from cache

  - Installing monolog/monolog (1.17.2)
    Loading from cache

  - Installing symfony/monolog-bundle (v2.8.2)
    Loading from cache

  - Installing swiftmailer/swiftmailer (v5.4.1)
    Loading from cache

  - Installing symfony/swiftmailer-bundle (v2.3.11)
    Loading from cache

  - Installing sensio/generator-bundle (v3.0.5)
    Loading from cache

  - Installing symfony/phpunit-bridge (v2.8.2)
    Loading from cache

paragonie/random_compat suggests installing ext-libsodium (Provides a modern crypt
o API that can be used to generate random bytes.)
doctrine/doctrine-cache-bundle suggests installing symfony/security-acl (For using
 this bundle to cache ACLs)
sensio/framework-extra-bundle suggests installing symfony/psr-http-message-bridge
(To use the PSR-7 converters)
monolog/monolog suggests installing aws/aws-sdk-php (Allow sending log messages to
 AWS services like DynamoDB)
monolog/monolog suggests installing doctrine/couchdb (Allow sending log messages t
o a CouchDB server)
monolog/monolog suggests installing ext-amqp (Allow sending log messages to an AMQ
P server (1.0+ required))
monolog/monolog suggests installing ext-mongo (Allow sending log messages to a Mon
```

```
goDB server)
monolog/monolog suggests installing graylog2/gelf-php (Allow sending log messages
to a GrayLog2 server)
monolog/monolog suggests installing php-console/php-console (Allow sending log mes
sages to Google Chrome)
monolog/monolog suggests installing raven/raven (Allow sending log messages to a S
entry server)
monolog/monolog suggests installing rollbar/rollbar (Allow sending log messages to
 Rollbar)
monolog/monolog suggests installing ruflin/elastica (Allow sending log messages to
 an Elastic Search server)
monolog/monolog suggests installing videlalvaro/php-amqplib (Allow sending log mes
sages to an AMQP server using php-amqplib)
Generating autoload files
> Incenteev\ParameterHandler\ScriptHandler::buildParameters
Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_host (127.0.0.1):
database_port (null):
prepavenir_addressbook_cli
database_user (root):
database_password (null):
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):
secret (ThisTokenIsNotSoSecretChangeIt): C4KCN35CHKX23H5K23CH52KV5HK234HV52KHVK2H4
TVKH23
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::buildBootstrap
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::clearCache

 // Clearing the cache for the dev environment with debug true



 [OK] Cache for the "dev" environment (debug=true) was successfully cleared.




> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::installAssets

 Trying to install assets as relative symbolic links.

 --- ------------------ ------------------
      Bundle             Method / Error
 --- ------------------ ------------------
  ✔   FrameworkBundle   relative symlink
```

```
 --- ----------------- ------------------


 [OK] All assets were successfully installed.



> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::installRequirementsFile
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::prepareDeploymentTarget
MBP-de-Romain:~ romain$ cd /Users/romain/Desktop/SFContactsFromCLI
MBP-de-Romain:SFContactsFromCLI romain$ php bin/console server:run


 [OK] Server running on http://127.0.0.1:8000


 // Quit the server with CONTROL-C.
^C
MBP-de-Romain:SFContactsFromCLI romain$
MBP-de-Romain:SFContactsFromCLI romain$ php bin/console server:start


 [OK] Web server listening on http://127.0.0.1:8000


MBP-de-Romain:SFContactsFromCLI romain$ php bin/console doctrine:database:create
Created database `prepavenir_addressbook_cli` for connection named default
MBP-de-Romain:SFContactsFromCLI romain$ php bin/console doctrine:generate:entity


  Welcome to the Doctrine2 entity generator



This command helps you generate Doctrine2 entities.

First, you need to give the entity name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post.

The Entity shortcut name: AppBundle:Contact

Determine the format to use for the mapping information.

Configuration format (yml, xml, php, or annotation) [annotation]:

Instead of starting with a blank entity, you can add some fields now.
```

```
Note that the primary key will be added automatically (named id).

Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, binary, blob, guid.

New field name (press <return> to stop adding fields): prenom
Field type [string]:
Field length [255]: 40
Is nullable [false]:
Unique [false]:

New field name (press <return> to stop adding fields): nom
Field type [string]:
Field length [255]: 40
Is nullable [false]:
Unique [false]:

New field name (press <return> to stop adding fields): email
Field type [string]:
Field length [255]: 80
Is nullable [false]: true
Unique [false]:

New field name (press <return> to stop adding fields): telephone
Field type [string]: 20
 Invalid type "20".
Field type [string]:
Field length [255]: 20
Is nullable [false]: true
Unique [false]:

New field name (press <return> to stop adding fields):


  Entity generation


> Generating entity class src/AppBundle/Entity/Contact.php: OK!
> Generating repository class src/AppBundle/Repository/ContactRepository.php: OK!


  Everything is OK! Now get to work :).


MBP-de-Romain:SFContactsFromCLI romain$ php bin/console doctrine:generate:crud
```

```
  Welcome to the Doctrine2 CRUD generator


This command helps you generate CRUD controllers and templates.

First, give the name of the existing entity for which you want to generate a CRUD
(use the shortcut notation like AcmeBlogBundle:Post)

The Entity shortcut name: AppBundle:Contact

By default, the generator creates two actions: list and show.
You can also ask it to generate "write" actions: new, update, and delete.

Do you want to generate the "write" actions [no]? yes

Determine the format to use for the generated CRUD.

Configuration format (yml, xml, php, or annotation) [annotation]:

Determine the routes prefix (all the routes will be "mounted" under this
prefix: /prefix/, /prefix/new, ...).

Routes prefix [/contact]: /contacts


  Summary before generation


You are going to generate a CRUD controller for "AppBundle:Contact"
using the "annotation" format.

Do you confirm generation [yes]?


  CRUD generation


Generating the CRUD code: OK
Generating the Form code: OK
Updating the routing: OK


  Everything is OK! Now get to work :).


MBP-de-Romain:SFContactsFromCLI romain$ php bin/console doctrine:schema:update --d
ump-sql
```

```
CREATE TABLE contact (id INT AUTO_INCREMENT NOT NULL, prenom VARCHAR(40) NOT NULL,
 nom VARCHAR(40) NOT NULL, email VARCHAR(80) DEFAULT NULL, telephone VARCHAR(20) D
EFAULT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci E
NGINE = InnoDB;
MBP-de-Romain:SFContactsFromCLI romain$ php bin/console doctrine:schema:update --f
orce
Updating database schema...
Database schema updated successfully! "1" query was executed
MBP-de-Romain:SFContactsFromCLI romain$
```

Récupérer un projet Symfony existant

- Rappatrier les sources
- S'il n'y pas de dossier vendor, faire un `composer install`, s'il n'y a pas non plus de parameters.yml, `composer install` vous propose de le recréer
- Recréer si besoin le projet sous NetBeans
  - S'il y a un dossier nbproject (File -> Open Project)
  - Sinon (File -> New Project -> PHP with Existing Sources)

- Recréer la base de données si besoin `php bin/console doctrine:database:create`
- Recréer ou modifier les tables si besoins
  `php bin/console doctrine:schema:update --dump-sql` puis
  `php bin/console doctrine:schema:update --force`
- Pour lancer le projet `php bin/console server:run` puis aller à l'adresse http://127.0.0.1:8000/ (pas besoin dans ce cas de app_dev.php car on est déjà sur un serveur de développement)