

Le Framework SYMFONY

Table des matières

1.	Symfony, Quelques accroches.....	6
1.1.	Le succès de symfony	6
1.2.	Symfony, une licorne	7
1.3.	Symfony Philosophy	7
1.4.	symfony en 10 phrases.....	8
1.5.	Symfony dans le monde	9
2.	De la version 3.4 à la version 4.4.....	10
2.1.	Releases Calendar.....	10
2.2.	Quoi de plus.....	11
2.3.	Il sera désormais plus facile de commencer sur Symfony.....	12
2.4.	Changement radical au niveau de la structuration des dossiers :.....	13
2.5.	Symfony 4 bundle-less, pourquoi ?	14
2.5.1.	Mais au fait qu'est-ce qu'un Bundle.....	14
2.5.2.	Désormais les bundles ne sont plus <i>que</i> pour du code tiers.....	14
2.6.	MakerBundle, c'est quoi ?	15
3.	Vérifier votre environnement	16
3.1.	Ajuster la version de PHP	16
3.2.	Tester la commande en ligne "php"	17
3.3.	Ajuster la version de MySQL.....	19
4.	INSTALLATION DE Composer.....	22
4.1.	L'outil "Composer"	22
4.1.1.	Vérifier que "composer" est déjà installé	22
4.1.2.	Installer Composer	22
4.1.2.1.	Installation sous Windows :.....	22
4.1.2.2.	Installation en ligne de commande	22
5.	Installation de Symfony (en une ligne).....	24
6.	L'architecture des fichiers	28
6.1.	Model-View-Controller MVC	28
6.2.	Sept répertoires : bin, config, public, src, templates, var, vendor	29
7.	Notre fil rouge pour ce Quick Tour : Gestion d'un cabinet médical	30
8.	L'ORM Doctrine 2, Généralités et exemples	31

8.1.	Object-Relational Mapping	31
8.2.	Doctrine 2 en trois couches : DBAL, ORM, et Entité	31
8.3.	Webographie utilisée pour cette section	32
8.4.	Installation Doctrine 2	33
8.5.	Configurer la connexion à la base de données puis la créer	36
8.5.1.	Sous PHPMyAdmin, Vérifier la présence de votre BdD symfonyXX	38
8.6.	Mapping par les annotations : Liens Entité ↔ Schéma BdD.....	39
8.7.	Générer les entités par la commande <code>doctrine: make:entity</code>	40
8.8.	Création des tables de la Base de données : migrate	44
9.	Traitement d'une requête sous symfony	46
9.1.	Flux applicatif entre une requête et une réponse sous Symfony	46
9.2.	Votre premier contrôleur	47
9.3.	.htaccess	48
10.	Doctrine, le mapping par les annotations	50
10.1.	Les types de propriétés d'une entité.....	50
10.2.	Correspondance type PHP/SQL.....	51
10.3.	Le mapping des relations entre entités par les annotations.....	52
10.3.1.	La liste des annotations pour le mapping des relations.....	52
10.3.2.	Une différence avec Merise : Relation ManyToMany avec propriétés.....	53
10.3.3.	Notion de propriétaire et d'inverse.....	54
10.3.4.	De quel côté placer l'annotation ?	55
10.3.5.	Application, lien bidirectionnelle Patient ↔ Consultation	56
10.3.6.	Quelle différence entre <code>inversedBy</code> et <code>mappedBy</code> ?	59
10.3.7.	Reprendre ici le croisement <code>mappedBy</code> , <code>inversedBy</code> en mettant en parallèle les 2 entités CONCERNÉES.....	60
10.3.9.	Validation du schéma Doctrine	61
10.4.	Générer les nouveaux accesseurs	62
11.	TP Cabinet Médical, Partie 1 à rendre sur le serveur partagé	64
11.1.	Travail demandé :	64
11.2.	Rappel MLD	64
12.	L' EntityManager	67
12.1.	Le service Doctrine, persistance des objets	67
12.2.	Le service EntityManager	68
12.3.	Insertion d'un nouveau patient.....	69
12.3.1.	Reprendre le contrôleur <code>PatientController</code> déjà généré	69
12.3.2.	Ajouter à ce contrôleur et executer	70

12.4.	Persist() et flush()	71
13.	TP Cabinet Médical, Partie 2 à rendre sur le serveur partagé	72
14.	repositories, méthodes de récupération.....	73
14.1.	Un repository par entité.....	73
14.2.	Méthodes de récupération de base	74
14.2.1.	find(\$id)	74
14.2.2.	findAll()	74
14.2.3.	findBy()	74
14.3.	Les méthodes magiques	75
14.3.1.	findByX(\$valeur)	75
14.3.2.	findOneByX(\$valeur)	75
14.4.	Le QueryBuilder et le DQL	76
15.	TWIG : un tout premier exemple pour notre exemple fil rouge	77
15.1.	TWIG est un moteur de template créé par le même auteur : Fabien POTENCIER, le patron de Sensio 77	
15.2.	Premier exemple de syntaxe TWIG	78
15.2.1.	Affichage basique d'une variable \$ma_date	78
15.2.2.	Première vue list.html.twig	79
16.	TP Cabinet Médical, Partie 3 à rendre sur le serveur partagé	80
17.	Make :CRUD.....	81
17.1.	php bin/console make:crud.....	81
17.2.	CRUD entité Medecin	82
17.3.	CRUD sur ManyToOne, Consultation	84
18.	TP Cabinet Médical, Partie 4 à rendre sur le serveur partagé	86
19.	Routage, Contrôleurs et Actions	87
19.1.	Actions, Contrôleur	87
19.2.	Tester le routage de votre application	php bin/console
	debug :router	88
19.2.1.	Exemple de routes exprimées par les annotations dans les contrôleurs.....	88
19.3.	Config/routes.yaml.....	89
19.3.1.	Routes en annotation	89
19.3.2.	Routes équivalente en YAML dans config/routes.yaml	90
19.4.	Syntaxe YAML	91
19.4.1.	Une étiquette + une url + une sortie	91
19.4.2.	Paramètre.....	92
19.4.3.	Profiler	93

19.5.	Testez-vous	94
20.	TP Cabinet Médical, Partie 5 à rendre sur le serveur partagé	95
21.	Formulaire, Première prise en main simplifiée	96
21.1.	Installer le package form	96
21.2.	Construction d'un formulaire en 4 temps et à la main	97
21.3.	Twig associé au formulaire	99
22.	TP Cabinet Médical, Partie 6 à rendre sur le serveur partagé	100
23.	Gestion des utilisateurs et sécurité (sans FOSuser)	101
23.1.	Make :user	102
23.1.	Make :auth, Générer un formulaire d'authentification (connexion)	106
23.2.	Ajouter un formulaire d'enregistrement.....	107
23.2.1.	Avec son contrôleur et son action :	108
23.2.2.	Complétons le contrôleur appelant le formulaire d'enregistrement en fixant un rôle initial (ROLE_USER) :	109
23.2.3.	Créer plusieurs users	110
23.3.	Découvrir le fichier de configuration security.yaml	111
23.3.1.	Deux parties principales	111
23.4.	Encodage des mots de passe.....	112
23.5.	Providers.....	113
23.6.	Revue du fichier de configuration de la sécurité security.yaml	114
23.6.1.	Firewalls.....	114
23.6.2.	Pattern :	115
23.6.3.	Login	116
23.6.1.	Logout.....	117
23.7.	Contrôle d'accès et rôle.....	119
23.7.1.	access_control (security.yaml):	119
23.7.2.	Annotations dans le contrôleur	120
23.7.3.	Templates TWIG	121
23.8.	Role_hierarchy.....	122
23.9.	Vérifier votre configuration de la sécurité	122
23.10.	Accéder au user connecté :	123
23.10.1.	Depuis le contrôleur	123
23.10.2.	Depuis un TWIG	124
24.	TP Cabinet Médical, Partie 7 à rendre sur le serveur partagé	125
25.	Webographie	126
26.	TWIG, les fondamentaux	127

26.1.	Appel du service, et passage des variables	128
26.2.	Trois types de syntaxes.....	129
26.3.	Syntaxe de base d’affichage, les doubles accolades « {{ ... }} ».....	130
26.4.	Syntaxe Logique de base : {{ objet.attribut }}.....	131
26.5.	Filtres natifs	132
26.6.	Structures de contrôle et expressions.....	133
26.7.	Hériter et inclure des Templates	136
26.8.	Le modèle « triple héritage »	138
27.	Automatiser la construction d’un formulaire : Make:form.....	140
27.1.	Make:Form, génération d’un entity type	140
27.2.	EntityType.....	141
28.	Controle des formulaire avec Validator,	142
28.1.	Quelques exemples de contrôle de formulaire.....	142
28.2.	Stratégie de validation (théorie de la validation)	143
28.3.	Plusieurs modalités sont possibles	144
28.4.	Installation de Validator, configuration¶	145
28.5.	Annotation @Assert, exemple	146
28.6.	Syntaxe	147
28.7.	Contraintes de base.....	148
28.8.	Contraintes sur les dates	149
28.9.	documentation officielle et Webographie	150
28.10.	Déclencher le contrôle du formulaire : \$form->isValid()	151
29.	TP Cabinet Médical, Partie 8 à rendre sur le serveur partagé	152
30.	Résumé du TP :.....	153
31.	Commande citées dans ce cours :	155
32.	Upload de fichier	156
33.	ERREURS REPERTORIEES A L’INSTALLATION	157
33.1.	Certificate Problem :.....	157
33.2.	date.timezone	158

1. SYMFONY, QUELQUES ACCROCHES

1.1. LE SUCCÈS DE SYMFONY

<https://symfony.com/500million>

3,704,227,501
Total Symfony Downloads

This dashboard was part of the [500 million Symfony downloads](#) initiative that took place on September 2016.

1.2. SYMFONY, UNE LICORNE

Grey (WPP) rachète Extreme Sensio 20/07/2017 - par Thomas Pontiroli

<http://www.strategies.fr/actualites/agences/1069216W/grey-wpp-rachete-extreme-sensio.html>

Le géant de la publicité prend une part majoritaire dans le capital du français Extreme Sensio, sur le point de fêter ses 20 ans.

1.3. SYMFONY PHILOSOPHY

La meilleure des devises :

« Adopter les meilleures pratiques, la normalisation et l'interopérabilité des applications. Promouvoir le professionnalisme »

1.4. SYMFONY EN 10 PHRASES

- Un **Framework**
- Un modèle **MVC**
- L'exploitation d'un **ORM** comme Doctrine permettant un Mapping objet-relationnel
- Une **gestion des URL** parlante, permettant à une page d'avoir une URL distincte de sa position dans l'arborescence.
- Un système de configuration en cascade utilisant le langage YAML.
- Un système de **cache** afin d'assurer des temps de réponse optimaux. Des performances optimisées.
- Un **générateur de formulaire**, de contrôle et de validation de formulaire
- Un **générateur de back-office**, socle initiale de votre application complexe (lanceur de module ou *scaffolding*).
- Un support d'**AJAX**.
- Une **architecture extensible** permettant la création et l'utilisation de bundles (une forme de plugins). Symfony utilise des bundles pour ajouter, retirer, modifier des fonctionnalités sur vos projets. Il existe des bundles pour à peu près tout (CMS, blogs, forums, systèmes de paiement en ligne, etc.)
- **L'internationalisation** native.

1.5. SYMFONY DANS LE MONDE

<https://trends.google.com/trends/explore?q=laravel,Symfony,%2Fm%2F02qgdj,CakePHP,Zend>

Nb de requêtes :

● laravel ● Symfony ● CodeIgniter ● CakePHP ● Zend



Trier : Intérêt pour Symfony ▼

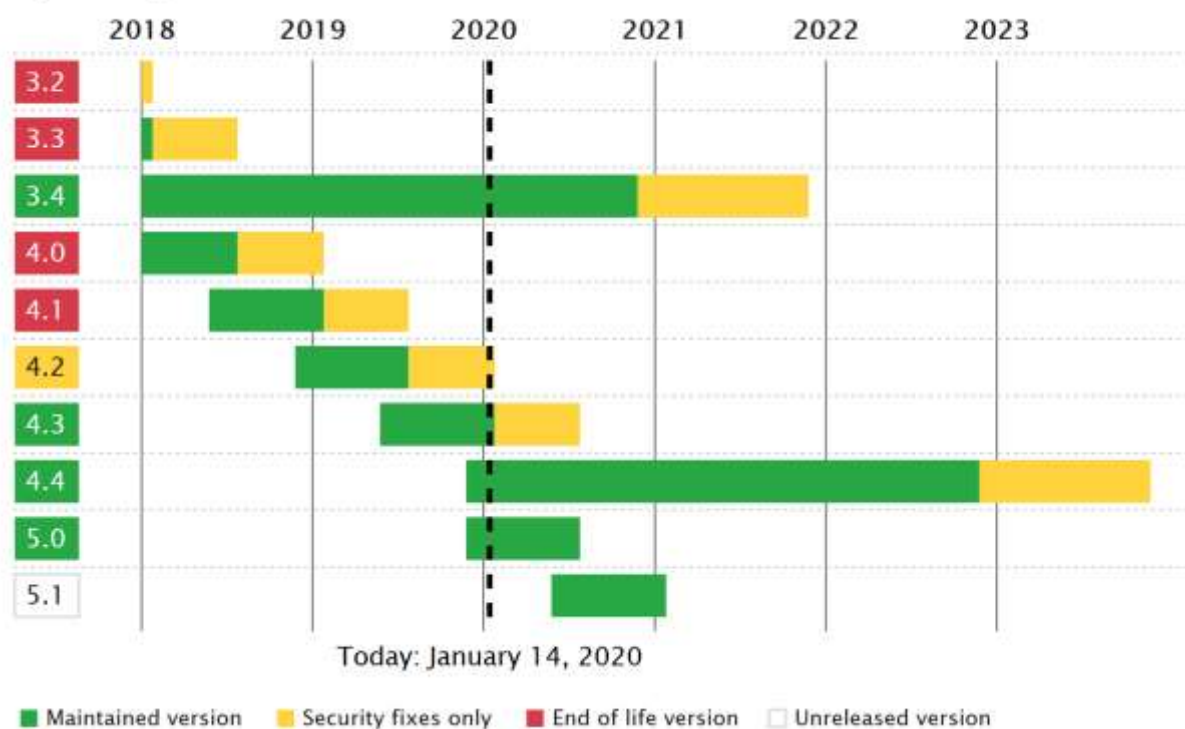
1	France	
2	Tunisie	
3	Madagascar	
4	Luxembourg	
5	Pologne	

2. DE LA VERSION 3.4 A LA VERSION 4.4

2.1. RELEASES CALENDAR

<https://symfony.com/releases>

Symfony Releases Calendar



Etude de « Latest Long-Term Support Release »

4.4.2

2.2. QUOI DE PLUS

Symfony 4.0 = Symfony 3.4

- Fonctionnalités obsolètes
- + Un nouveau moyen de développer des applications

D'après l'article de Fabien Potencier, le créateur de symfony : <http://fabien.potencier.org/symfony4-compose-applications.html>, <https://meritis.fr/langage/symfony-4-10-points/>

2.3. IL SERA DESORMAIS PLUS FACILE DE COMMENCER SUR SYMFONY

➔ Objectif 1 de symfony 4 :

- faciliter l'installation
- faciliter la prise en main du Framework.
- courbe d'apprentissage plus douce au démarrage
- réalisation plus rapide de prototype

➔ Objectif 2 :

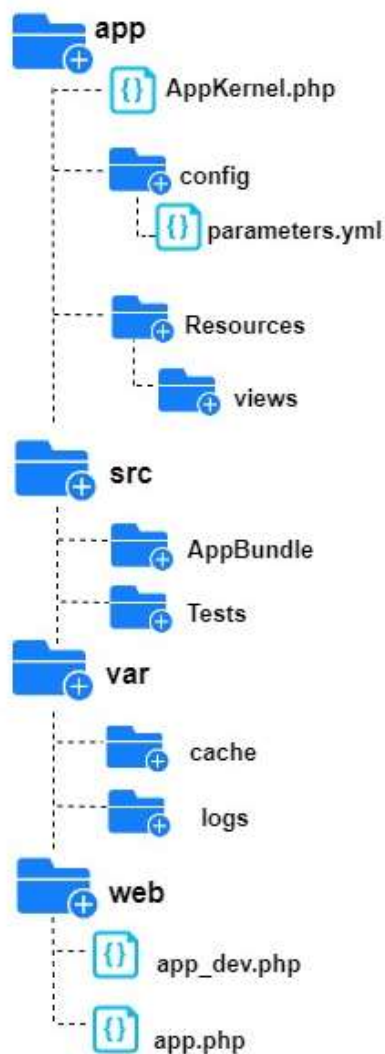
- Amélioration de l'installation des vendors et l'auto-configuration de bundles
- Pour les experts, plus de flexibilité pour perdre moins de temps sur les configurations, et pouvoir se concentrer sur le code métier.

(D'après <https://blog.sensiolabs.com/fr/2018/01/09/symfony-4-migration-projet-sensiolabs/#automatis%C3%A9e>)

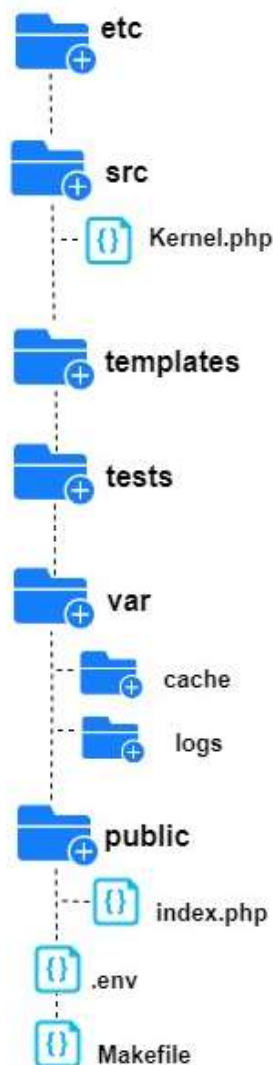
2.4. CHANGEMENT RADICAL AU NIVEAU DE LA STRUCTURATION DES DOSSIERS :

- dossier web → renommé en public
- app.php contenu dans ce dernier → renommé en index.php.
- dossier app → dossier config (équivalent au dossier app/config de la version 3 de Symfony mais il ne contient pas les fichiers parameters.yml et parameters.dist.yml.)
- La classe Kernel → déplacée dans src.
- Un nouveau fichier .env contient la configuration pour les variables d'environnements.

Symfony 3



Symfony 4



2.5. SYMFONY 4 BUNDLE-LESS, POURQUOI ?

<http://fabien.potencier.org/symfony4-monolith-vs-micro.html>

2.5.1. MAIS AU FAIT QU'EST-CE QU'UN BUNDLE

Jusqu'à sa version 3, symfony se déclinait en Bundle.

Un Bundle est un répertoire contenant un ensemble de fichiers (fichiers PHP, feuilles de style, JavaScripts, images,...) qui implémentent une seule fonctionnalité :

- un blog,
- un forum,
- une gestion d'utilisateurs
-
- ➔ Une architecture ouverte permettant d'assembler des bundles (paquet, plugin) d'origines diverses.
- ➔ Possibilité d'utiliser des fonctionnalités prédéfinies intégrées dans des offres tierces
- ➔ Possibilité de distribuer vos propres offres bundles

Symfony 4 n'a plus de bundles dans src non plus.

- ➔ Tous les fichiers se trouvent maintenant dans le répertoire **src /** (et non **App /** namespace par défaut).
- ➔ Suppression de la plupart des contraintes du développeur, notamment relatives au paramétrage des bundles.
- ➔ plus de facilité, d'efficacité

2.5.2. DESORMAIS LES BUNDLES NE SONT PLUS QUE POUR DU CODE TIERS

Bundle-less ne signifie pas que l'application ne consomme plus de bundles. La nuance est que désormais les bundles ne sont plus *que* pour du code tiers. Avant, dans Symfony 2 et 3, les applications étaient elles-mêmes des bundles.

2.6. MAKERBUNDLE, C'EST QUOI ?

Développé à partir de la version 3.4, MakerBundle accélère la production du développeur :

MakerBundle génère à la place du développeur un ensemble de code passe-partout (code snippets) opérationnel :

- ➔ Des entités
- ➔ Des contrôleurs
- ➔ Des formulaires
- ➔ Etc.

```
php bin/console list make
```

```
Available commands for the "make" namespace:
```

<code>make:auth</code>	Creates a Guard authenticator of different flavors
<code>make:command</code>	Creates a new console command class
<code>make:controller</code>	Creates a new controller class
<code>make:crud</code>	Creates CRUD for Doctrine entity class
<code>make:entity</code>	Creates or updates a Doctrine entity class, and optionally an API Platform resource
<code>make:fixtures</code>	Creates a new class to load Doctrine fixtures
<code>make:form</code>	Creates a new form class
<code>make:functional-test</code>	Creates a new functional test class
<code>make:migration</code>	Creates a new migration based on database changes
<code>make:registration-form</code>	Creates a new registration form system
<code>make:serializer:encoder</code>	Creates a new serializer encoder class
<code>make:serializer:normalizer</code>	Creates a new serializer normalizer class
<code>make:subscriber</code>	Creates a new event subscriber class
<code>make:twig-extension</code>	Creates a new Twig extension class
<code>make:unit-test</code>	Creates a new unit test class
<code>make:user</code>	Creates a new security user class
<code>make:validator</code>	Creates a new validator and constraint class
<code>make:voter</code>	Creates a new security voter class

3. VERIFIER VOTRE ENVIRONNEMENT

3.1. AJUSTER LA VERSION DE PHP

Symfony4 ne tourne que sur PHP7.1.3+

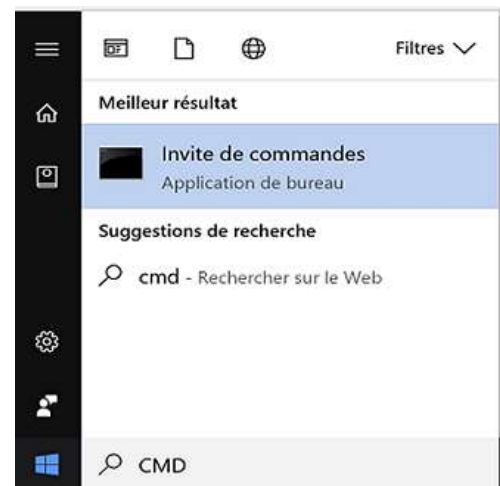
Si PHP7.1 est présent sur votre machine, ajuster sous Ampps la bonne version comme ci-dessous



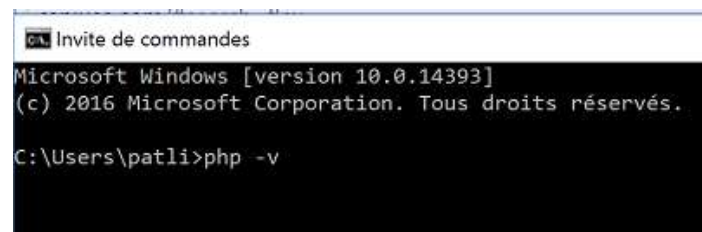
Si PHP7.1+ n'est pas présent, il vous reste à l'installer !

3.2. TESTER LA COMMANDE EN LIGNE "PHP"

➔ Fenêtre « cmd »



➔ Testez "php -v"



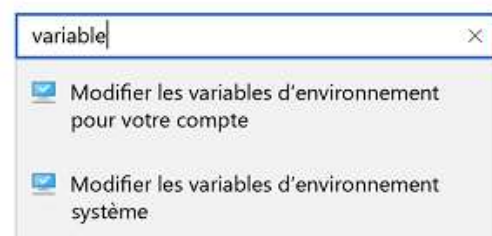
➔ Si reconnu (livre le numéro de version de PHP), quitter cette page

➔ Sinon, ajuster les variables d'environnement Path en procédant comme ci-dessous

➔ Rendez-vous dans
« Panneau de configuration > Système »,

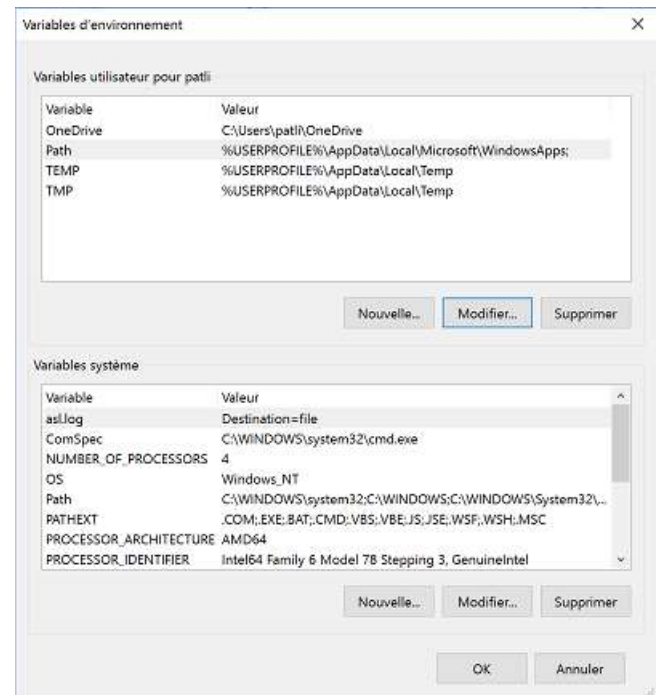
Dans le champ de recherche, rechercher
"variable"

Paramètres Windows

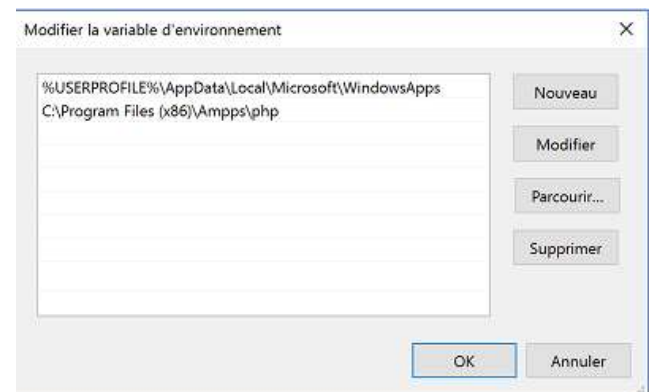


➔ Sélectionner « Variables d'environnement pour votre compte »

- ➔ Dans « variables utilisateur » sélectionnez la variable nommée « Path » et cliquez sur « Modifier »



- ➔ Ajouter dans la valeur un nouveau chemin : celui de l'exécutable PHP (si AMPPS sous Windows : C:\Program Files (x86)\Ampps\php-7.1)



- ➔ Fermez votre invite de commande Windows si ce n'est déjà fait
- ➔ Relancer le serveur Apache
- ➔ Rouvrez à nouveau votre invite de commande, et exécutez "php -v"

```
Invite de commandes
Microsoft Windows [version 10.0.17134.648]
(c) 2018 Microsoft Corporation. Tous droits réservés.

C:\Users\patli>php -v
PHP 7.1.8 (cli) (built: Aug  1 2017 21:10:46) ( ZTS MSVC14 (Visual C++ 2015) x86 )
Copyright (c) 1997-2017 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2017 Zend Technologies
    with the ionCube PHP Loader v10.0.0, Copyright (c) 2002-2017, by ionCube Ltd.
    with Xdebug v2.5.5, Copyright (c) 2002-2017, by Derick Rethans

C:\Users\patli>
```

3.3. AJUSTER LA VERSION DE MYSQL

La version >5.6 devient nécessaire afin que le format JSON des données soit supporté.

Si ce n'est pas le cas sur votre machine :

➔ Suivez le guide l'installation de la version 5.7.17

<https://the-allstars.com/blog/website-information/upgrade-ampps-mysql-to-5-7.html>

➔ Comment réinitialiser le pw de root :

<https://dev.mysql.com/doc/refman/5.7/en/resetting-permissions.html>

Lancer mysql après l'installation, sans mot de passe, sous C:\Program Files (x86)\Ampps\mysql\bin> :
mysql -uroot -p

➔ Connaître la version de mysql

Lieu d'exécution de mysqld : C:\Program Files (x86)\Ampps\mysql\bin>

```
mysql> SHOW VARIABLES LIKE "%version%";
```

Variable_name	Value
innodb_version	5.7.17
protocol_version	10
slave_type_conversions	
tls_version	TLSv1,TLSv1.1
version	5.7.17
version_comment	MySQL Community Server (GPL)
version_compile_machine	x86_64
version_compile_os	Win64

```
C:\Program Files (x86)\Ampps\mysql\bin>mysqld.exe --defaults-file=..\my.ini --initialize-insecure --console
```

```
2019-04-17T06:59:15.328004Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation for more details).
```

```
2019-04-17T06:59:15.343585Z 0 [Warning] You need to use --log-bin to make --binlog-format work.
```

```
2019-04-17T06:59:16.124803Z 0 [Warning] InnoDB: New log files created, LSN=45790
```

```
2019-04-17T06:59:16.280914Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
```

```
2019-04-17T06:59:16.390270Z 0 [Warning] No existing UUID has been found, so we assume that this is the first time that this server has been started. Generating a new UUID: 50aadae0-60de-11e9-a8f5-00e04c307e3f.
```

```
2019-04-17T06:59:16.390270Z 0 [Warning] Gtid table is not ready to be used. Table 'mysql.gtid_executed' cannot be opened.
```

```
2019-04-17T06:59:16.405835Z 1 [Warning] root@localhost is created with an empty password ! Please consider switching off the --initialize-insecure option.
```

4. INSTALLATION DE COMPOSER

L'installation de Symfony est devenue très simple à l'aide de l'outil « Composer »

<https://openclassrooms.com/fr/courses/3619856-developpez-votre-site-web-avec-le-framework-symfony/5066156-installer-symfony-grace-a-composer>

4.1. L'OUTIL "COMPOSER"

4.1.1. VERIFIER QUE "COMPOSER" EST DEJA INSTALLE

```
C:\Users\patli>composer --version  
Composer version 1.6.3 2018-01-31 16:28:17
```

Si composer n'est pas installé, suivre le guide ci-dessous

4.1.2. INSTALLER COMPOSER

4.1.2.1. INSTALLATION SOUS WINDOWS :

- Télécharger « Composer-Setup.exe » sur <https://getcomposer.org/download/>
- Exécuter le.
- Rajouter l'adresse de son dossier dans le path utilisateur comme déjà décrit



4.1.2.2. INSTALLATION EN LIGNE DE COMMANDE

L'installation est bien décrite sur son site <https://getcomposer.org/download/>

Voici les lignes à exécution dans la fenêtre CMD (les reprendre du site car le hash peut avoir changé)

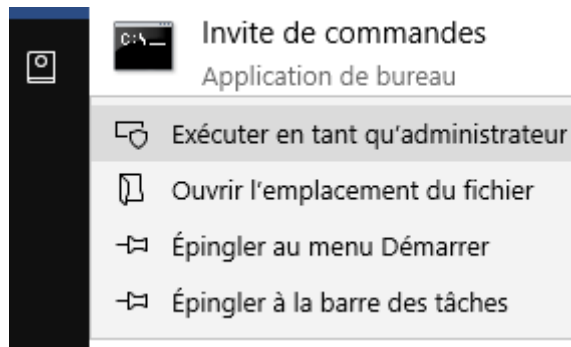
```
php -r "copy('https://getcomposer.org/installer', 'composer-  
setup.php');"

php -r "if (hash_file('sha384', 'composer-setup.php') ===  
'93b54496392c062774670ac18b134c3b3a95e5a5e5c8f1a9f115f203b75bf9a129d5  
daa8ba6a13e2cc8a1da0806388a8') { echo 'Installer verified'; } else {  
echo 'Installer corrupt'; unlink('composer-setup.php'); } echo  
PHP_EOL;"

php composer-setup.php
php -r "unlink('composer-setup.php');"
```

5. INSTALLATION DE SYMFONY (EN UNE LIGNE)

- 1- Placez-vous dans le répertoire www
- 2- Lancez « cmd »
 - a. dans la ligne d'adresse de l'explorer si vous travailler en local
 - b. A distance sur le serveur partagé, à partir de la recherche windows, lancer invite de commande



Se connecter au serveur du bureau 519 par
Ssh licencexx@134.157.46.186
Password : licencexx

- 3- Puis la ligne « `composer create-project symfony/website-skeleton symfony-test` »
(au 14/01/2020 installe par défaut la version 4.4)

```
C:\Program Files (x86)\Ampps\www>composer create-project symfony/website-skeleton symfony-test
```

Ou, à distance :

```
Last login: Tue Apr 16 17:01:44 2019 from 134.157.46.194  
licence23@Serveur24-14-519:/var/www/html/licence23$ composer create-project symfony/website-skeleton symfony-test
```


On obtient en lignes finales :

What's next?

- * Run your application:

1. Go to the project directory
2. Create your code repository with the git init command
3. Download the Symfony CLI at <https://symfony.com/download> to install a development web server,

or run `composer require symfony/server --dev` for a minimalist one

- * Read the documentation at <https://symfony.com/doc>

Database Configuration

- * Modify your DATABASE_URL config in .env

- * Configure the driver (mysql) and

server_version (5.7) in config/packages/doctrine.yaml

How to test?

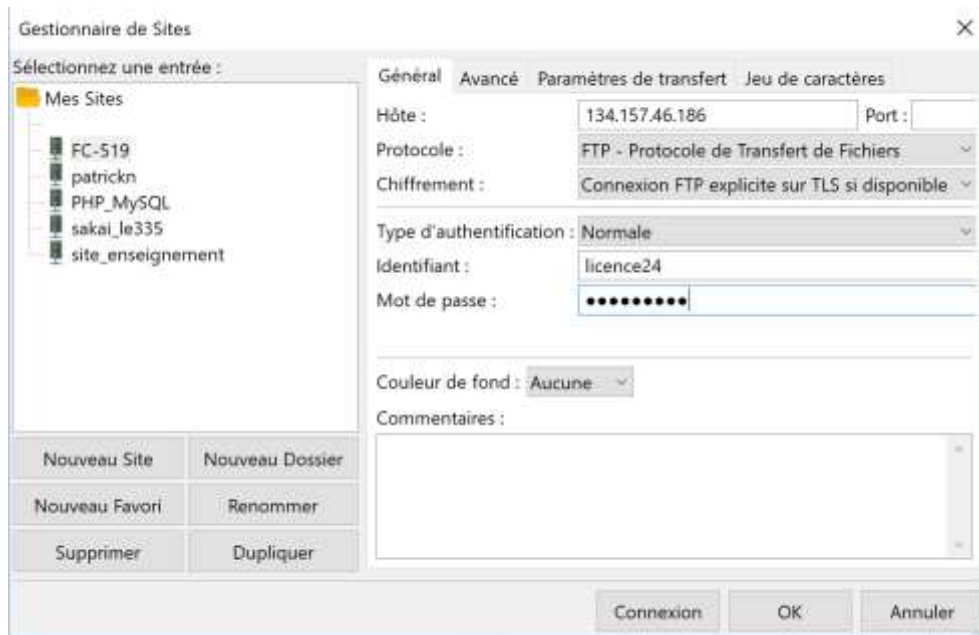
- * Write test cases in the tests/ folder

- * Run `php bin/phpunit`

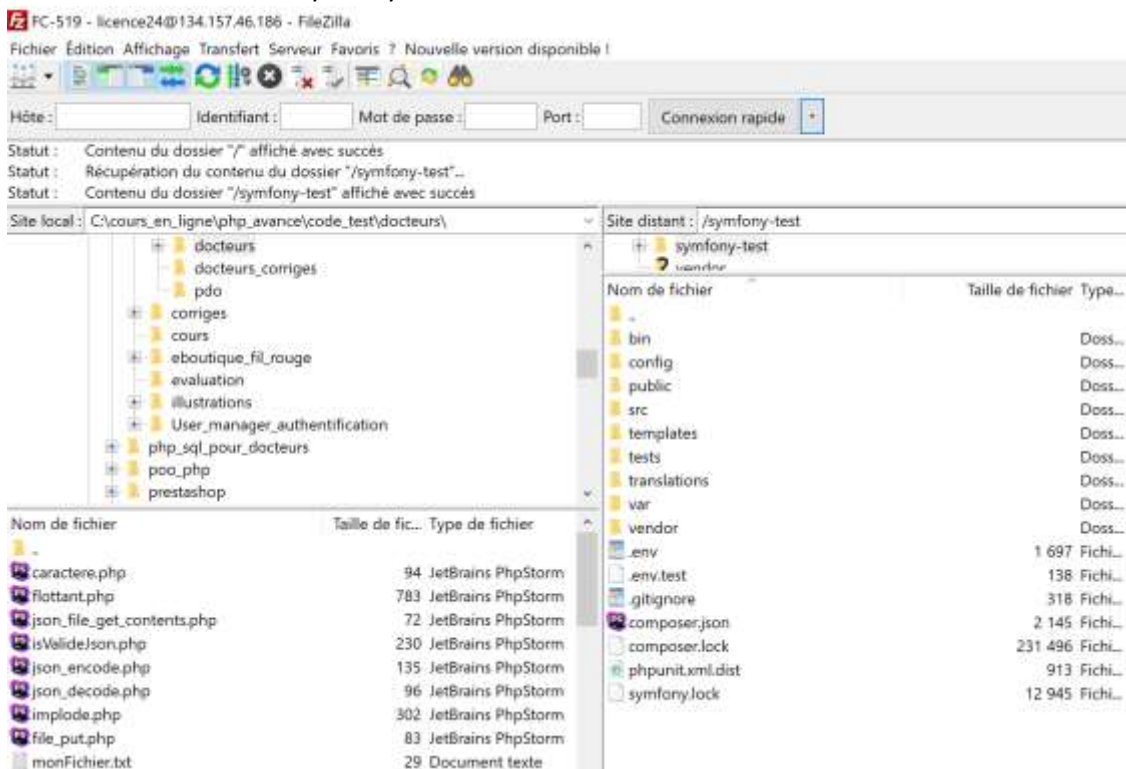
licence24@Serveur24-14-519:/var/www/html/licence24\$

Visualiser les dossiers créés :

- 1- Lancez Filezilla (Client FTP)
- 2- Connectez-vous comme ci-dessous



- 3- Visualiser votre dossier 'symfony-test'



- 4-

Il ne vous reste plus qu'à ouvrir votre navigateur : suivant le mode local ou distant ou serveur interne choisissez l'adresse :

Distant :

<http://134.157.46.186/licence00/symfony-test/public/>

Local :

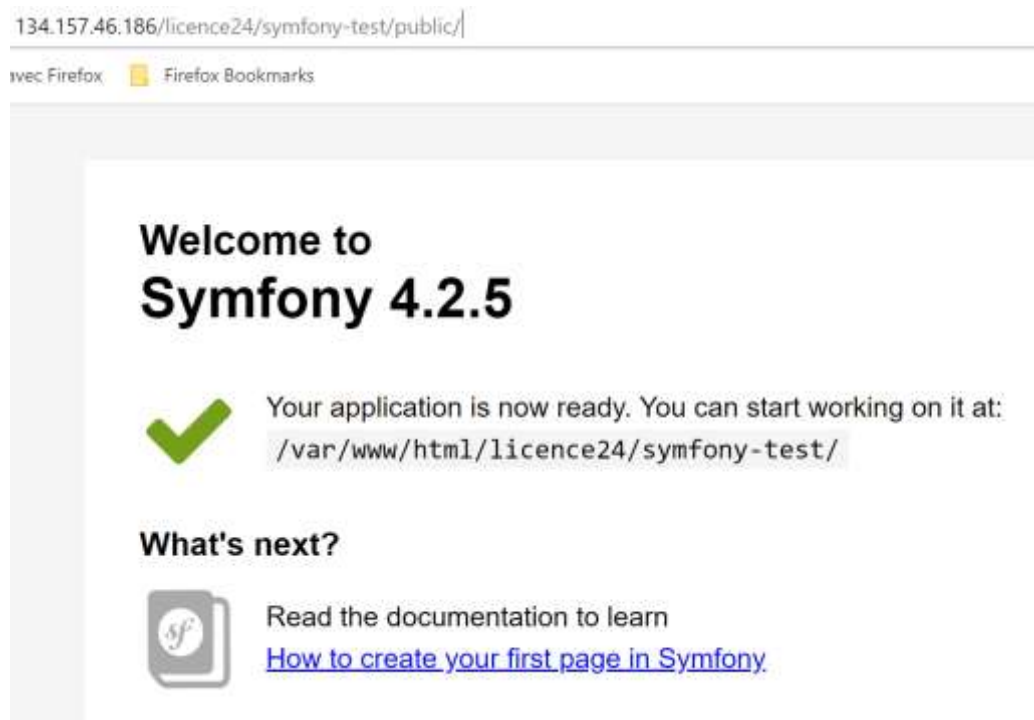
<http://localhost/symfony-test/public/>

En lançant le server php

```
C:\Program Files (x86)\Ampps\www\symfony-test>php bin/console server:run  
[OK] Server listening on http://127.0.0.1:8000  
// Quit the server with CONTROL-C.
```

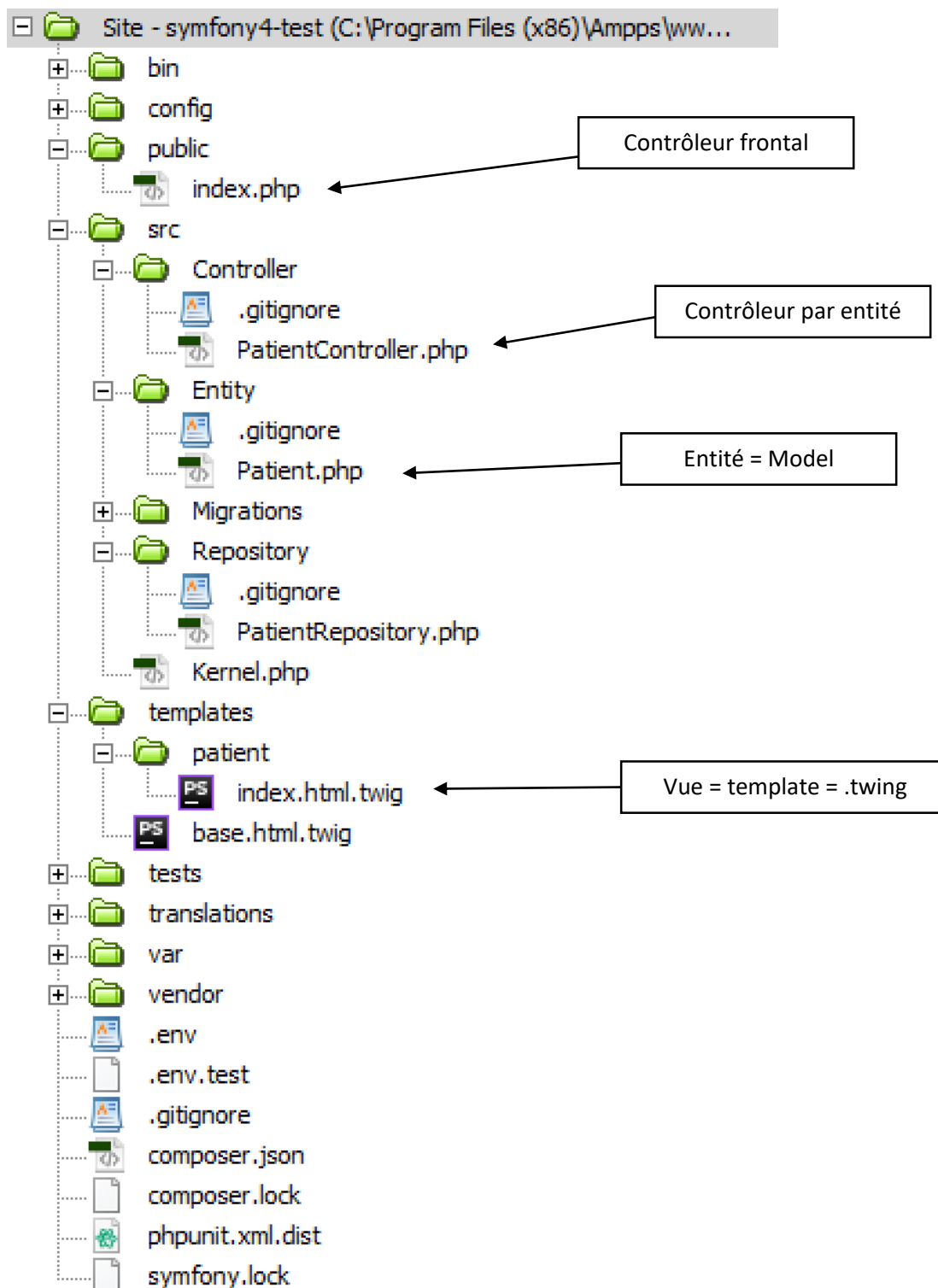
Puis

<http://localhost:8000/>

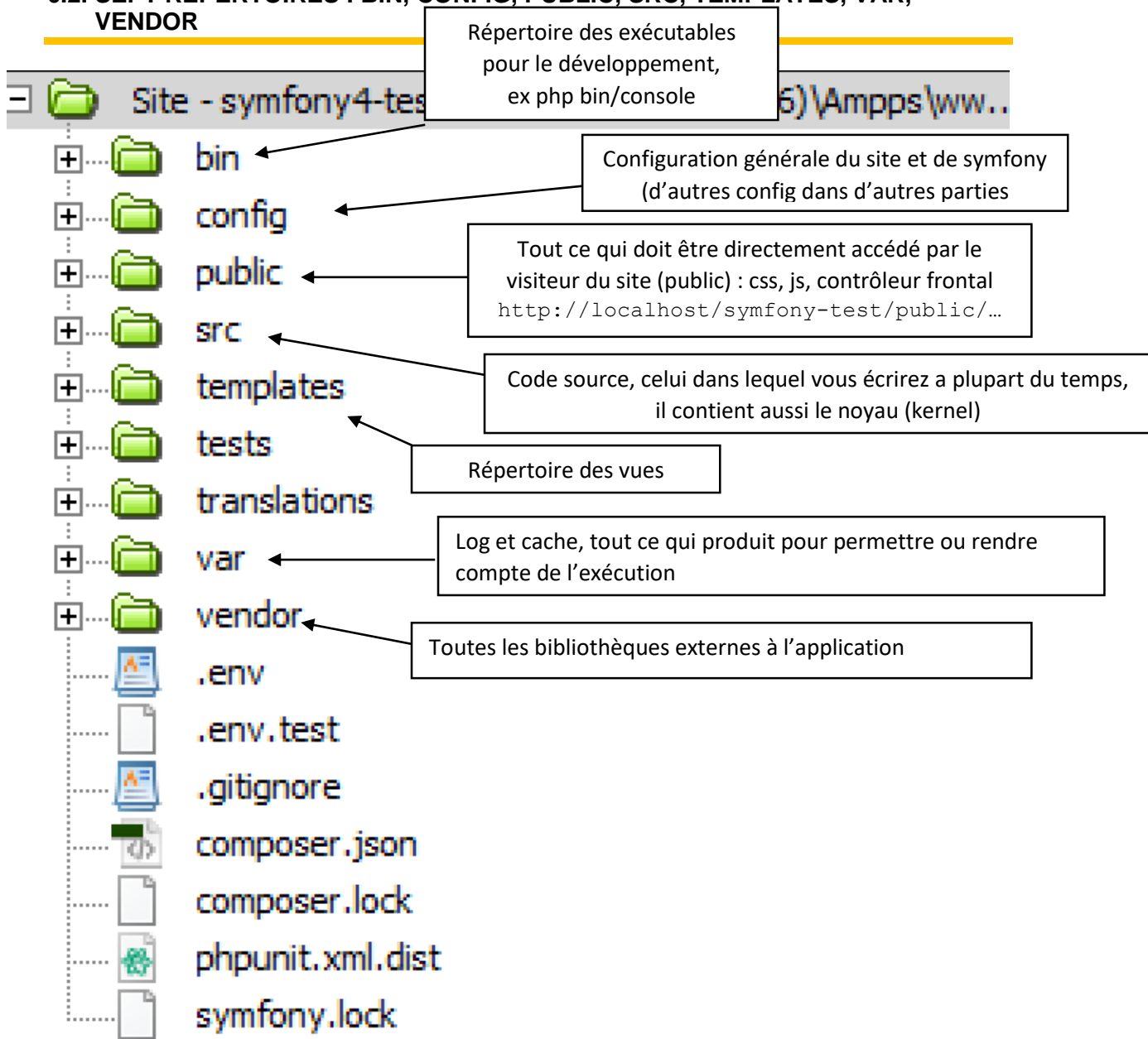


6. L'ARCHITECTURE DES FICHIERS

6.1. MODEL-VIEW-CONTROLLER MVC

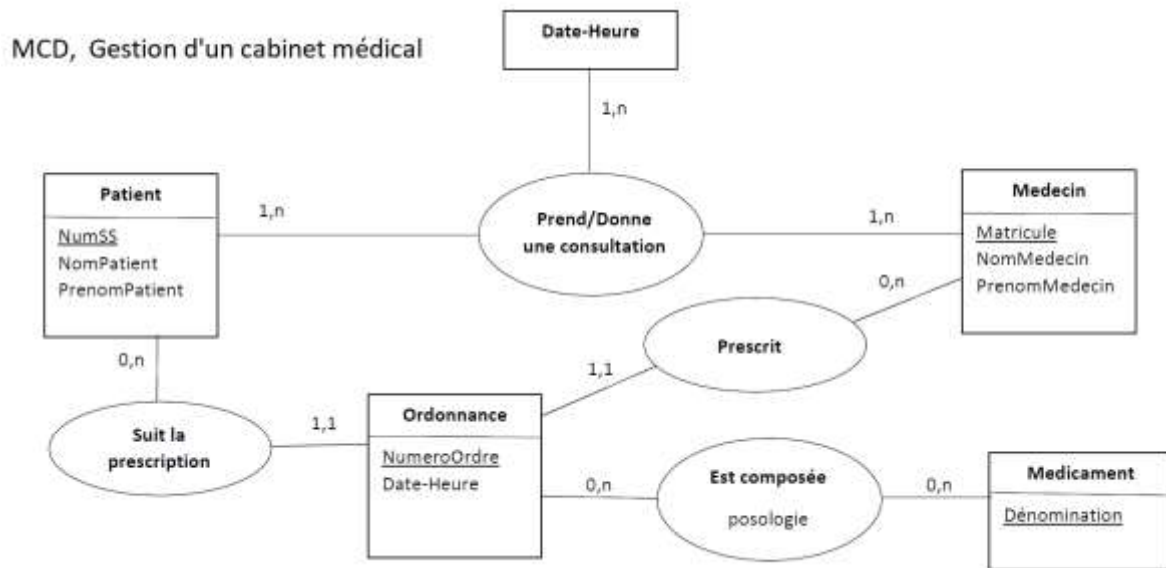


6.2. SEPT REPERTOIRES : BIN, CONFIG, PUBLIC, SRC, TEMPLATES, VAR, VENDOR

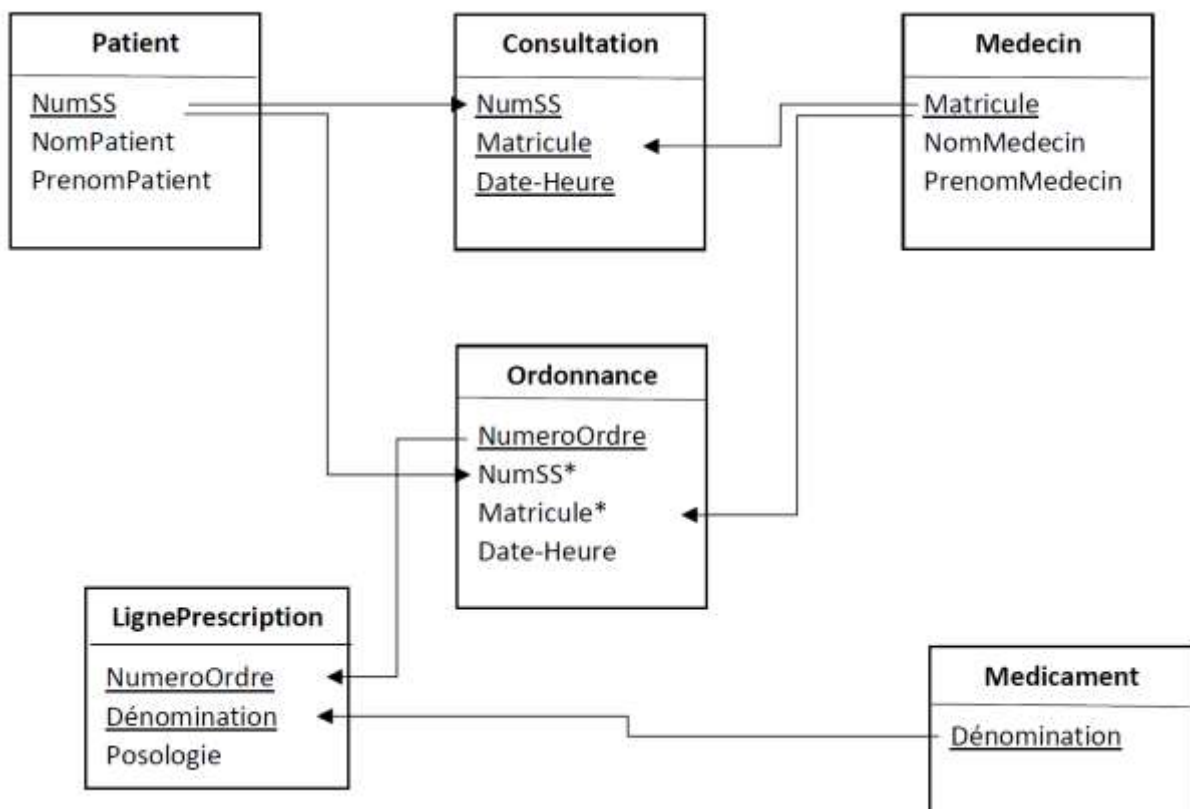


La plupart de votre activité de développeur concernera le dossier src, et de temps à autre le répertoire config. Parfois l'installation d'une nouvelle bibliothèque concernera le dossier vendor

7. NOTRE FIL ROUGE POUR CE QUICK TOUR : GESTION D'UN CABINET MEDICAL



MLD :



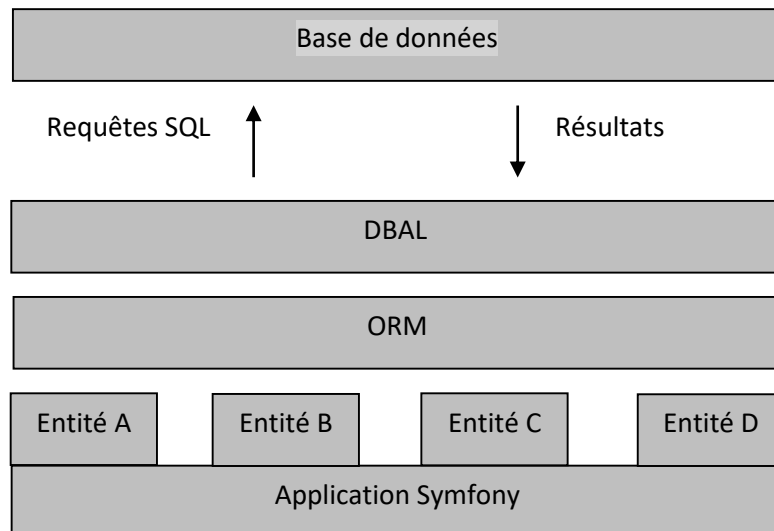
8. L'ORM DOCTRINE 2, GENERALITES ET EXEMPLES

8.1. OBJECT-RELATIONAL MAPPING

Ainsi l'ORM fait la correspondance entre les objets de l'application et la base de données relationnelle

Symfony a clairement fait de Doctrine 2 son option favorite (mais ce n'est pas la seule)

8.2. DOCTRINE 2 EN TROIS COUCHES : DBAL, ORM, ET ENTITE



- DBAL : (Database Abstraction Layer) couche de bas niveau sans logique applicative, comparable à une extension de PHP PDO (DBAL contient PHP PDO)
- Entité : Objet métier, reflet applicatif d'une table de la base de données.
- ORM : librairie intermédiaire entre DBAL et les entités, établit la correspondance entre la base de données relationnelle et la programmation orientée objet
 - Convertit les données tabulaires reçues depuis la DBAL en entité et ses attributs
 - Réciproquement, convertit les actions sur les entités ou entre entités en requête SQL transmises à DBAL
 - La création, modification, recherche, suppression des instances des classes/entités seront des méthodes des entités qui se propageront à la base de données sans écrire une ligne de SQL grâce à DOCTRINE2

8.3. WEBOGRAPHIE UTILISÉE POUR CETTE SECTION

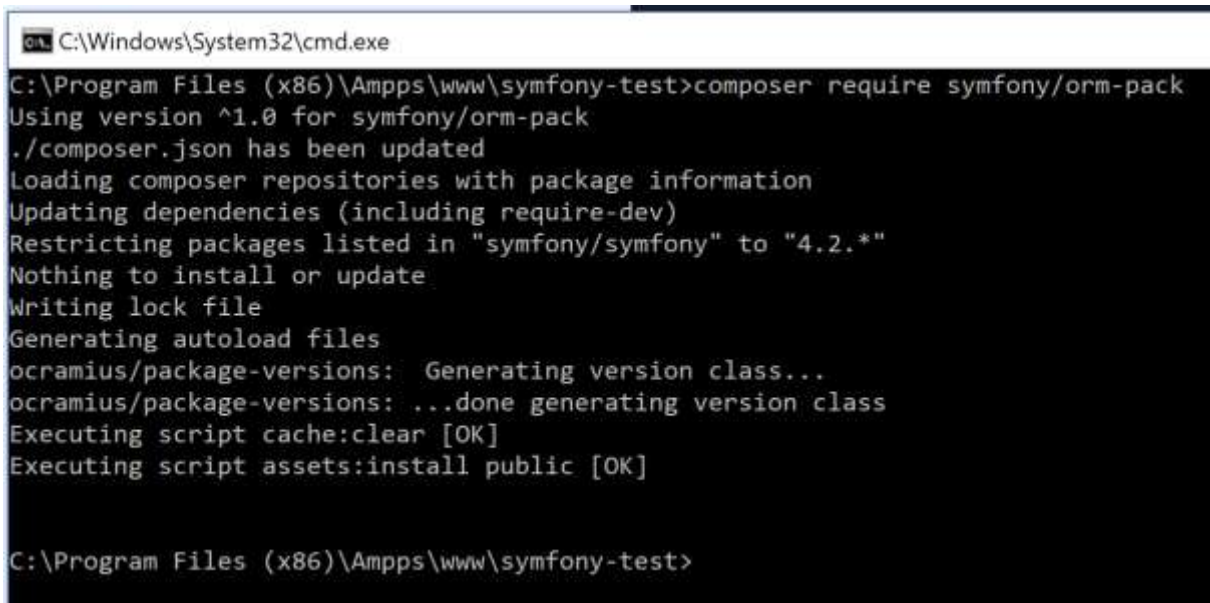
<https://symfony.com/doc/current/doctrine.html>

8.4. INSTALLATION DOCTRINE 2

Tout d'abord,

1. Vérifier dans le composer.json (à la racine du dossier symfony) que le package orm-pack a été installé,
2. sinon installer Doctrine via le pack ORM (lancer la fenêtre de commande sous C:\Program Files (x86)\Ampps\www\symfony-test), puis lancer

```
composer require symfony/orm-pack
```



```
C:\Windows\System32\cmd.exe
C:\Program Files (x86)\Ampps\www\symfony-test>composer require symfony/orm-pack
Using version ^1.0 for symfony/orm-pack
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "4.2.*"
Nothing to install or update
Writing lock file
Generating autoload files
ocramius/package-versions: Generating version class...
ocramius/package-versions: ...done generating version class
Executing script cache:clear [OK]
Executing script assets:install public [OK]

C:\Program Files (x86)\Ampps\www\symfony-test>
```

3. Idem pour le package MakerBundle, qui vous aidera à générer du code (Attention bien lire « tirt tirt dev »)

```
composer require symfony/maker-bundle --dev
```

```
C:\Windows\System32\cmd.exe

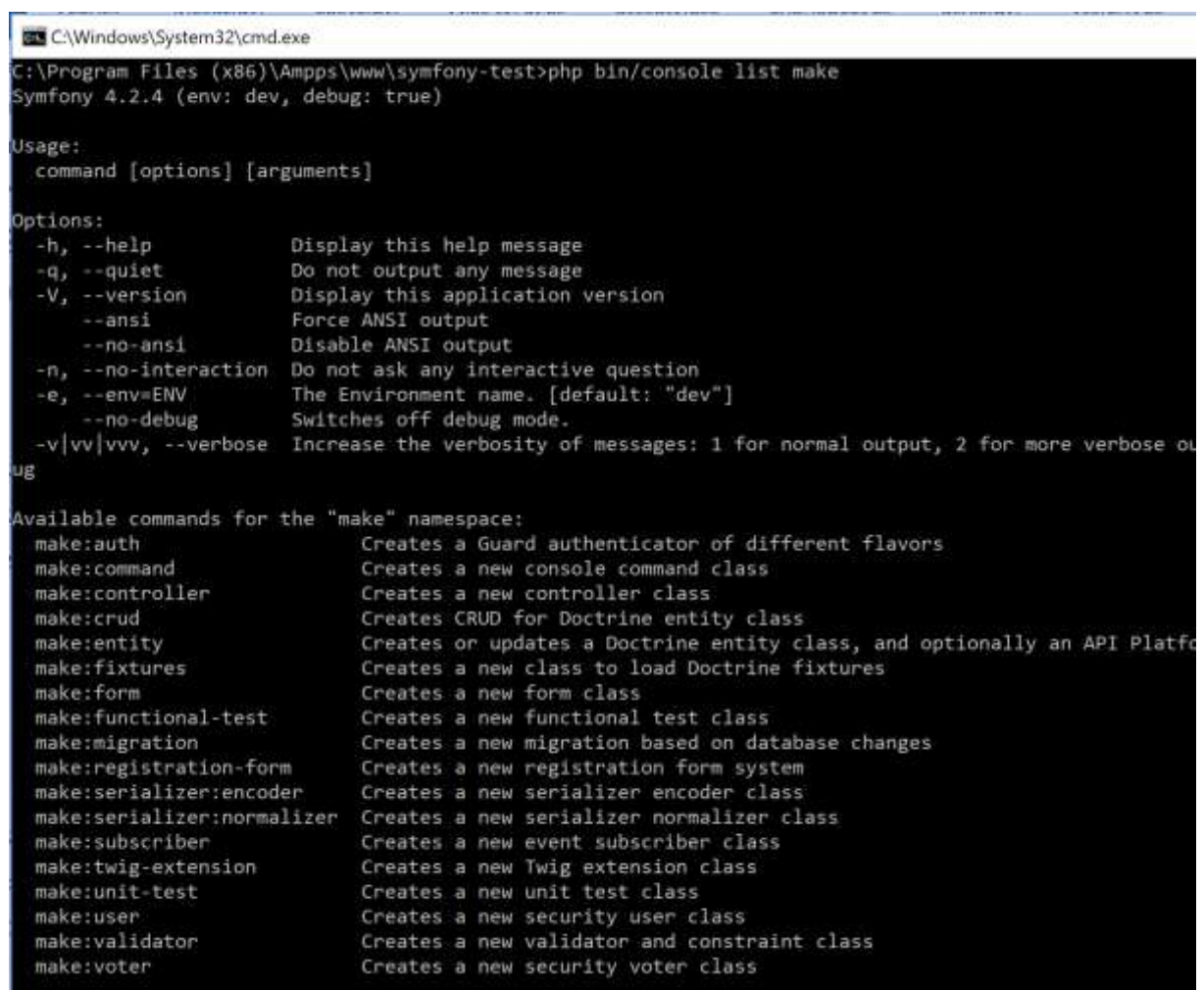
C:\Program Files (x86)\Ampss\www\symfony-test>composer require symfony/maker-bundle --dev
Using version ^1.11 for symfony/maker-bundle
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "4.2.*"
Nothing to install or update
Generating autoload files
ocramius/package-versions: Generating version class...
ocramius/package-versions: ...done generating version class
Executing script cache:clear [OK]
Executing script assets:install public [OK]
```

Pour voir la liste des options de la commande make , placez-vous dans votre répertoire symfony-test que vous venez de créer :

```
cd symfony-test
```

Puis exécutez la commande php suivante :

```
php bin/console list make
```



```
C:\Windows\System32\cmd.exe
C:\Program Files (x86)\Ampms\www\symfony-test>php bin/console list make
Symfony 4.2.4 (env: dev, debug: true)

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
  --ansi               Force ANSI output
  --no-ansi            Disable ANSI output
  -n, --no-interaction Do not ask any interactive question
  -e, --env=ENV         The Environment name. [default: "dev"]
  --no-debug           Switches off debug mode.
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose ou
ug

Available commands for the "make" namespace:
make:auth                Creates a Guard authenticator of different flavors
make:command             Creates a new console command class
make:controller          Creates a new controller class
make:crud                Creates CRUD for Doctrine entity class
make:entity              Creates or updates a Doctrine entity class, and optionally an API Platfo
make:fixtures            Creates a new class to load Doctrine fixtures
make:form                Creates a new form class
make:functional-test     Creates a new functional test class
make:migration            Creates a new migration based on database changes
make:registration-form   Creates a new registration form system
make:serializer:encoder  Creates a new serializer encoder class
make:serializer:normalizer Creates a new serializer normalizer class
make:subscriber          Creates a new event subscriber class
make:twig-extension      Creates a new Twig extension class
make:unit-test           Creates a new unit test class
make:user                Creates a new security user class
make:validator           Creates a new validator and constraint class
make:voter               Creates a new security voter class
```

Si message d'erreur : « There are no commands defined in the "make" namespace. »

Installer doctrine par : `composer require doctrine maker`

Si le serveur php interne ne fonctionne pas, l'installer par : `composer req webserver`

8.5. CONFIGURER LA CONNEXION A LA BASE DE DONNEES PUIS LA CREER.

A la racine de votre projet symfony-test, le fichier .env permet de configurer la connexion à la base de données (cf squelette du projet)

Modifier la variable d'environnement

```
DATABASE_URL=mysql://db_user:db_password@
```

En

```
DATABASE_URL=mysql://root:mysql@127.0.0.1:3306/symfonyXX
```

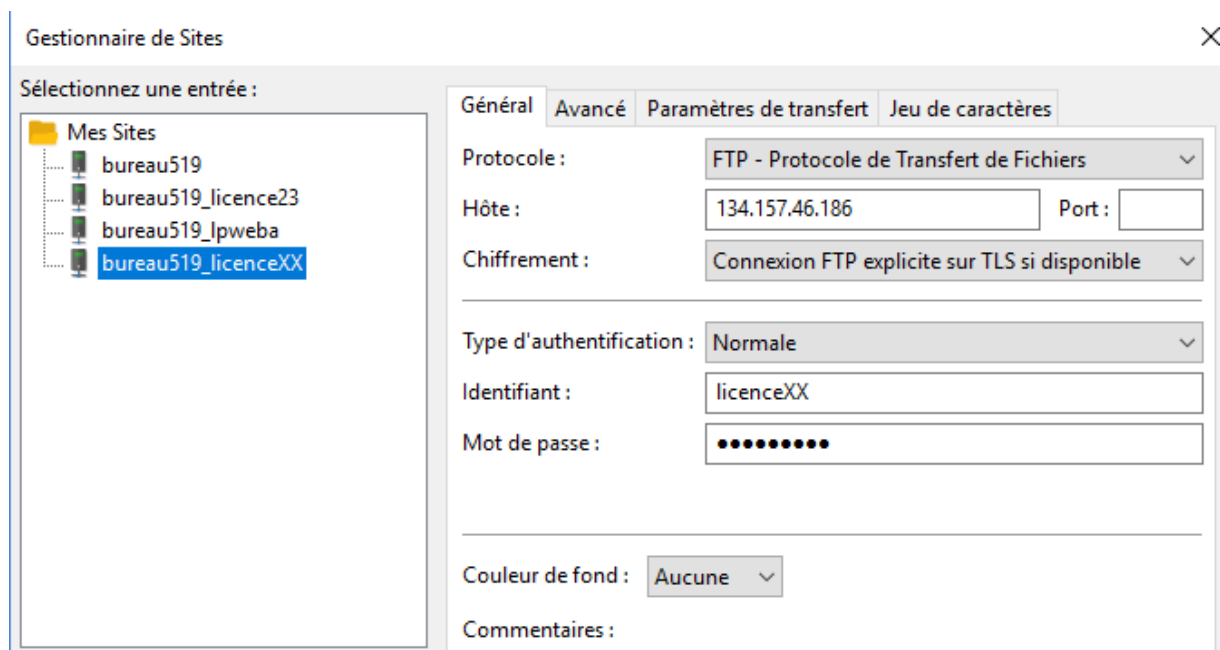
ou

```
DATABASE_URL=mysql://root:@127.0.0.1:3306/symfonyXX
```

Si vous travaillez sur le serveur du bureau 519 : lancez Filezilla et connectez-vous à votre compte FTP

Pour modifier .env en substituant :

```
DATABASE_URL=mysql://root:mysql@127.0.0.1:3306/symfonyXX
```



Et rechercher le fichier .env

Site distant : /symfony-test		
<ul style="list-style-type: none"> symfony-test <ul style="list-style-type: none"> bin config public src templates tests translations var vendor 		
Nom de fichier	Taille de fi...	Type de fic...
..		
bin		Dossier de ...
config		Dossier de ...
public		Dossier de ...
src		Dossier de ...
templates		Dossier de ...
tests		Dossier de ...
translations		Dossier de ...
var		Dossier de ...
vendor		Dossier de ...
.env	1 702	Fichier ENV
.env.test	138	Fichier TEST
.gitignore	318	Fichier GITI...

Lancer maintenant :

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

```

C:\Windows\System32\cmd.exe

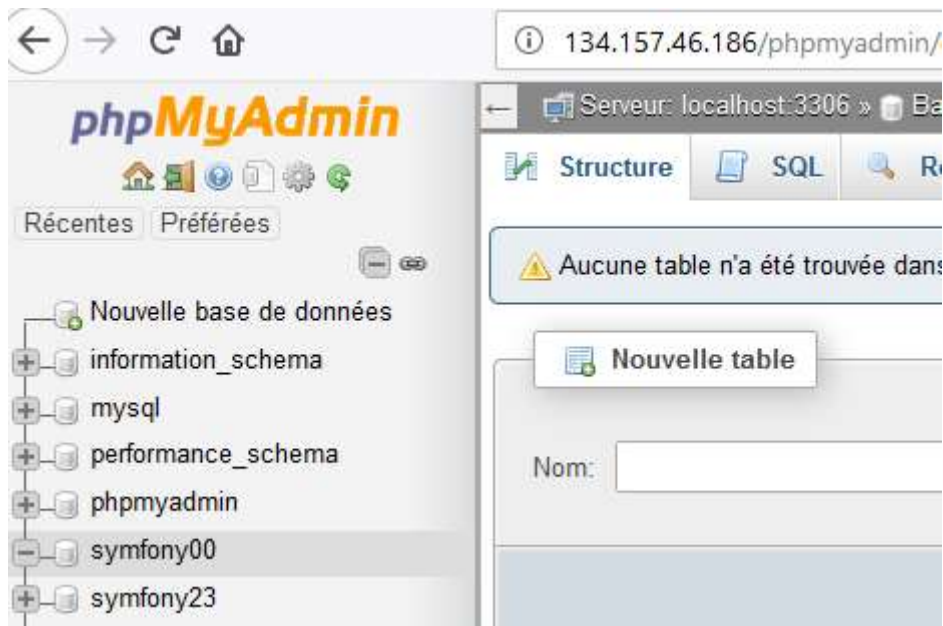
C:\Program Files (x86)\Ampps\www\symfony-test>php bin/console doctrine:database:create
Created database 'symfony' for connection named default

C:\Program Files (x86)\Ampps\www\symfony-test>

```

8.5.1. SOUS PHPMYADMIN, VERIFIER LA PRESENCE DE VOTRE BDD SYMFONYXX

Login : root, pw : mysql



8.6. MAPPING PAR LES ANNOTATIONS : LIENS ENTITE ↔ SCHEMA BDD

Plusieurs formats de configurations (comme pour le routage) :

- YAML,
- XML,
- PHP,
- Annotations → Le format « Annotation » est le plus rapide et le plus intuitif.

Un premier exemple :

```
<?php

namespace App\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\MedecinRepository")
 */
class Medecin
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=15)
     */
    private $matricule;

    /**
     * @ORM\Column(type="string", length=40)
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=40, nullable=true)
     */
    private $prenom;

    /**
     * @ORM\OneToMany(targetEntity="Ordonnance", mappedBy="medecin",
     cascade={"persist"})
     */
    private $ordonnance;
```

8.7. GENERER LES ENTITES PAR LA COMMANDE DOCTRINE: MAKE:ENTITY

Nous allons ensemble créer les entités de notre application fil rouge « cabinetMedical »

Pour créer les entités issues du MCD, plusieurs méthodes de création sont possibles, Se placer sous le répertoire symfony-test et lancer la commande :

```
php bin/console make:entity
```

Webographie : <https://openclassrooms.com/fr/courses/5489656-construisez-un-site-web-a-l-aide-du-framework-symfony-4/5517031-gerez-vos-donnees-avec-doctrine-orm>

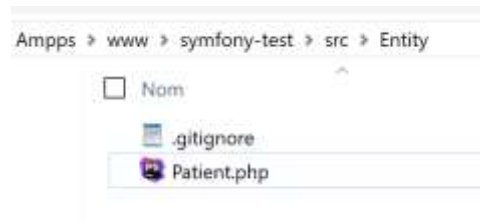
Prenons notre première entité Patient :

Patient
NumSS nom prenom dateNaissance sexe

La commande make :entity va demander le nom des propriétés ainsi que leurs types. Suite l'échange à la page suivante :


```
C:\Program Files (x86)\Amp\www\symfony-test>php bin/console make:entity
Class name of the entity to create or update (e.g. OrangeJellybean):
> Patient
created: src/Entity/Patient.php
created: src/Repository/PatientRepository.php
Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.
New property name (press <return> to stop adding fields):> NumSS
Field type (enter ? to see all types) [string]:
>
Field length [255]:
> 15
Can this field be null in the database (nullable) (yes/no) [no]:
>
updated: src/Entity/Patient.php
Add another property? Enter the property name (or press <return> to stop adding fields):
> nom
Field type (enter ? to see all types) [string]:
>
Field length [255]:
> 40
Can this field be null in the database (nullable) (yes/no) [no]:
>
updated: src/Entity/Patient.php
Add another property? Enter the property name (or press <return> to stop adding fields):
> Prenom
Field type (enter ? to see all types) [string]:
>
Field length [255]:
> 40
Can this field be null in the database (nullable) (yes/no) [no]:
>updated: src/Entity/Patient.php
Add another property? Enter the property name (or press <return> to stop adding fields):
> dateNaissance
Field type (enter ? to see all types) [string]:
> date
Can this field be null in the database (nullable) (yes/no) [no]:
> yes
updated: src/Entity/Patient.php
Add another property? Enter the property name (or press <return> to stop adding fields):
> sexe
Field type (enter ? to see all types) [string]:
>
Field length [255]:
> 1
Can this field be null in the database (nullable) (yes/no) [no]:
> yes
updated: src/Entity/Patient.php
Add another property? Enter the property name (or press <return> to stop adding fields):
>
Success!
Next: When you're ready, create a migration with make:migration
```

Une entité Patient.php vient d'être générée



Voir ci-dessous à la page suivante,

A noter qu'une propriété « id » clef primaire auto-incrémente a été automatique générée

Doctrine a aussi générée les getters et les setters

Doctrine génère aussi les annotations nécessaires lors de la création des tables SQL de la Bdd (commande make :migration)

```

<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\PatientRepository")
 */
class Patient
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=15)
     */
    private $NumSS;

    /**
     * @ORM\Column(type="string", length=40)
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=40)
     */
    private $prenom;

    /**
     * @ORM\Column(type="date", nullable=true)
     */
    private $dateNaissance;

    /**
     * @ORM\Column(type="string", length=1, nullable=true)
     */
    private $sexe;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getNumSS(): ?string
    {
        return $this->NumSS;
    }

    public function setNumSS(string $NumSS): self
    {
        $this->NumSS = $NumSS;

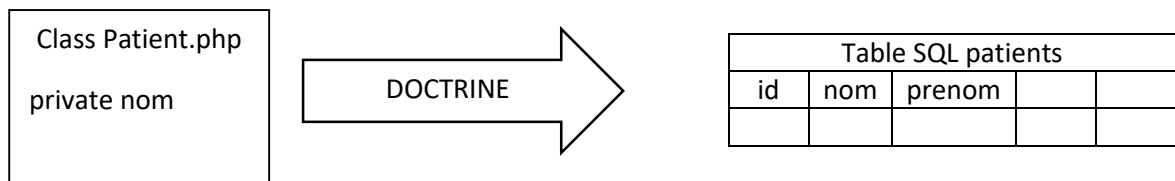
        return $this;
    }

    public function getNom(): ?string
    {
        return $this->nom;
    }

    public function setNom(string $nom): self
    {

```

8.8. CREATION DES TABLES DE LA BASE DE DONNEES : MIGRATE



```
php bin/console make:migration
```

```
C:\Program Files (x86)\Ampms\www\symfony-test>php bin/console make:migration
```

Success!

Next: Review the new migration "src/Migrations/Version20190403072826.php"

Then: Run the migration with `php bin/console doctrine:migrations:migrate`

See <https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html>

Si vous ouvrez le fichier **Version20190403072826.php**, il contient le code SQL nécessaire pour mettre à jour votre base de données! Pour exécuter ce SQL, exécutez :

```
php bin/console doctrine:migrations:migrate
```

```
C:\Program Files (x86)\Ampms\www\symfony-test>php bin/console  
doctrine:migrations:migrate
```

Application Migrations

WARNING! You are about to execute a database migration that could result in schema changes and data loss. Are you sure you wish to continue? (y/n)y

Migrating up to 20190403072826 from 0

++ migrating 20190403072826

-> CREATE TABLE patient (id INT AUTO_INCREMENT NOT NULL, num_ss VARCHAR(15) NOT NULL, nom VARCHAR(40) NOT NULL, prenom VARCHAR(40) NOT NULL, date_naissance DATE DEFAULT NULL, sexe VARCHAR(1) DEFAULT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci ENGINE = InnoDB

++ migrated (took 193.7ms, used 14M memory)

++ finished in 208.9ms

++ used 14M memory

++ 1 migrations executed

++ 1 sql queries

Résultat :

Serveur: localhost » Base de données: symfony » Table: patient

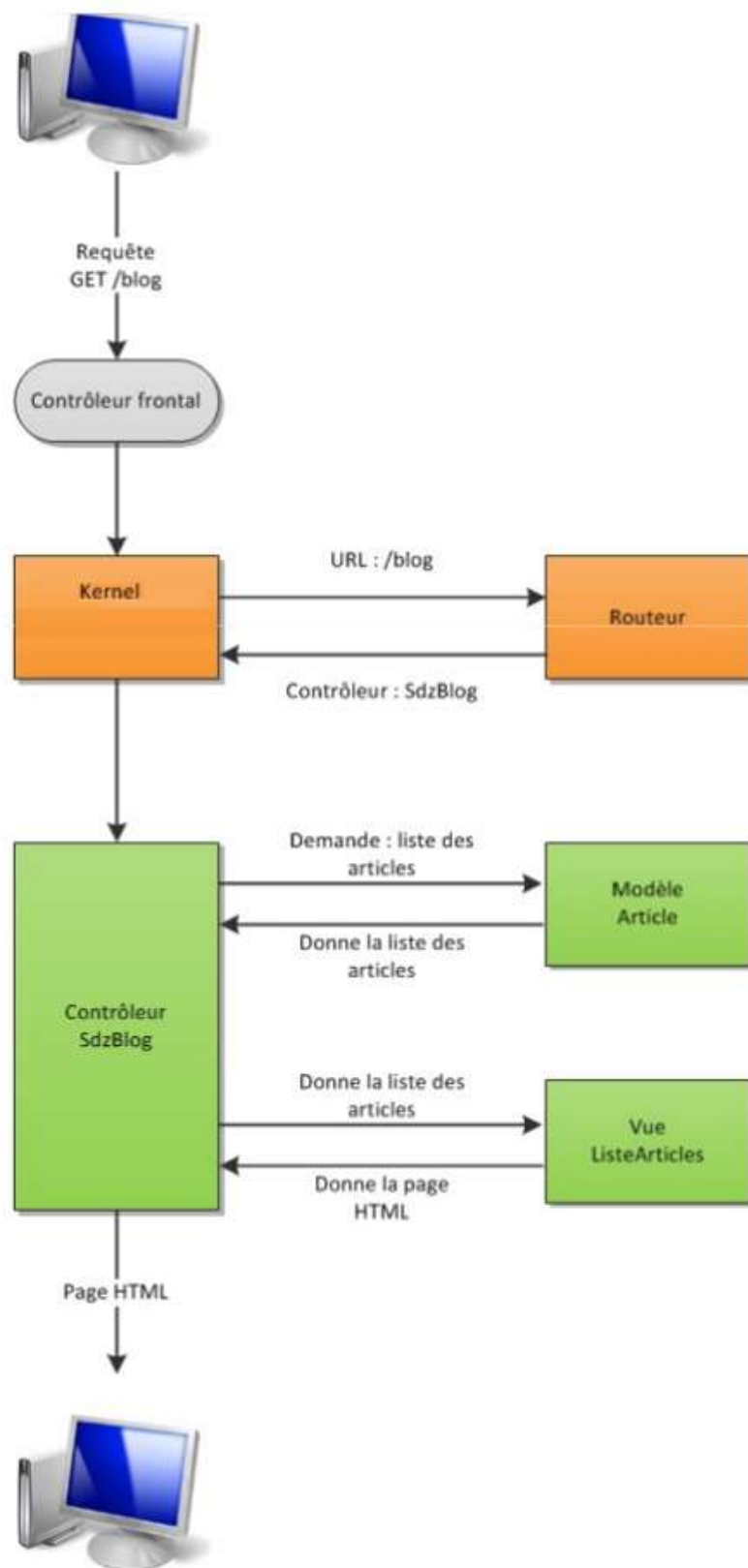
Afficher Structure SQL Rechercher Insérer

Structure de table Vue relationnelle

#	Nom	Type	Interclassement	Attributs	Null	Défaut	E
<input type="checkbox"/>	1 id 	int(11)			Non	Aucune	A
<input type="checkbox"/>	2 num_ss	varchar(15)	utf8mb4_unicode_ci		Non	Aucune	
<input type="checkbox"/>	3 nom	varchar(40)	utf8mb4_unicode_ci		Non	Aucune	
<input type="checkbox"/>	4 prenom	varchar(40)	utf8mb4_unicode_ci		Non	Aucune	
<input type="checkbox"/>	5 date_naissance	date			Oui	NULL	
<input type="checkbox"/>	6 sexe	varchar(1)	utf8mb4_unicode_ci		Oui	NULL	

9. TRAITEMENT D'UNE REQUETE SOUS SYMFONY

9.1. FLUX APPLICATIF ENTRE UNE REQUETE ET UNE REPONSE SOUS SYMFONY



9.2. VOTRE PREMIER CONTRÔLEUR

Construction automatisée d'un premier squelette de contrôleur

```
php bin/console make:controller PatientController
```

```
C:\Program Files (x86)\Ampps\www\symfony-test>php bin/console make:controller
PatientController
```

```
created: src/Controller/PatientController.php
```

```
created: templates/patient/index.html.twig
```

```
Success!
```

```
Next: Open your new controller class and add some pages!
```

Constater qu'au passage, Doctrine a créé non seulement un contrôleur mais aussi une vue !

Tester <http://localhost:8000/patient>

Ou sur le serveur distant, une fois que votre .htaccess installé (cf paragraphe suivant)

<http://134.157.46.186/licence23/symfony-test/public/patient/>

← → ↻ ⓘ localhost:8000/patient

Hello PatientController!

This friendly message is coming from:

- Your controller at `src/Controller/PatientController.php`
- Your template at `templates/patient/index.html.twig`

9.3. .HTACCESS

Pour exécuter notre requête sur le serveur distant, il faut placer un .htaccess dans le dossier public de symfony. Pour ce faire, exécutez la commande

```
composer require symfony/apache-pack
```

```
C:\Program Files (x86)\Ampps\www\symfony-test>composer require symfony/apache-pack
Using version ^1.0 for symfony/apache-pack
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Restricting packages listed in "symfony/symfony" to "4.2.*"
Package operations: 1 install, 0 updates, 0 removals
 - Installing symfony/apache-pack (v1.0.1): Downloading (100%)
Writing lock file
Generating autoload files
ocramius/package-versions: Generating version class...
ocramius/package-versions: ...done generating version class
Symfony operations: 1 recipe (13f1dede899e16165dfbc381ac6cce67)
 - WARNING symfony/apache-pack (>=1.0): From github.com/symfony/recipes-contrib:master
   The recipe for this package comes from the "contrib" repository, which is open to community
   contributions.
   Review the recipe at https://github.com/symfony/recipes-contrib/tree/master/symfony/apache-pack/1.0
   Do you want to execute this recipe?
   [y] Yes   [n] No   [a] Yes for all packages, only for the current installation session
   [p] Yes permanently, never ask again for this project
   (defaults to n): y
 - Configuring symfony/apache-pack (>=1.0): From github.com/symfony/recipes-contrib:master
Executing script cache:clear [OK]
Executing script assets:install public [OK]
Some files may have been created or updated to configure your new packages.
Please review, edit and commit them: these files are yours.
```

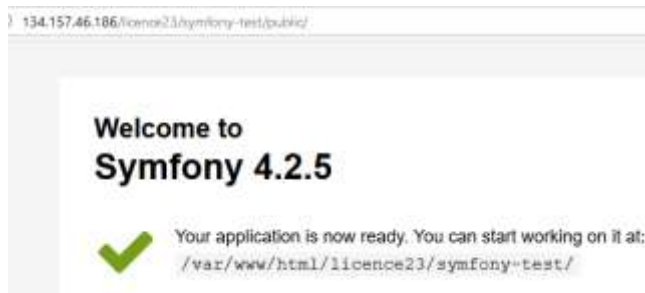
> www > symfony-test > public



Vérifier que votre .htaccess est bien placé dans le dossier public

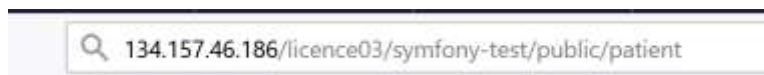
Tester votre installation :

<http://134.157.46.186/licenceXX/symfony-test/public/>



Puis

Relancer pour accéder par le serveur apache :



https://symfony.com/doc/current/setup/web_server_configuration.html

10. DOCTRINE, LE MAPPING PAR LES ANNOTATIONS

10.1. LES TYPES DE PROPRIETES D'UNE ENTITE

La commande "make:entity" permet de lister toutes les propriétés attendues :

```
Field type (enter ? to see all types) [string]:
> ?

Main types
* string
* text
* Boolean
* integer (or smallint, bigint)
* float

Relationships / Associations
* relation (a wizard will help you build the relation)
*ManyToOne
*OneToMany
*ManyToMany
*OneToOne

Array/Object Types
* array (or simple_array)
* json
* object
* binary
* blob

Date/Time Types
* datetime (or datetime_immutable)
* datetimetz (or datetimetz_immutable)
* date (or date_immutable)
* time (or time_immutable)
* dateinterval

Other Types
* json_array
* decimal
* guid
```

10.2. CORRESPONDANCE TYPE PHP/SQL

<https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/basic-mapping.html#doctrine-mapping-types>

Les types définis dans les annotations Doctrine ne correspondent ni aux types en PHP ni à ceux de la base de données, mais sont mappés aux deux.

Type	Mapping
string	Type that maps a SQL VARCHAR to a PHP string
integer	Type that maps a SQL INT to a PHP integer
smallint	Type that maps a database SMALLINT to a PHP integer
bigint	Type that maps a database BIGINT to a PHP string
boolean	Type that maps a SQL boolean or equivalent (TINYINT) to a PHP boolean
decimal	Type that maps a SQL DECIMAL to a PHP string
date	Type that maps a SQL DATETIME to a PHP DateTime object
time	Type that maps a SQL TIME to a PHP DateTime object
datetime	Type that maps a SQL DATETIME/TIMESTAMP to a PHP DateTime object
datetimetz	Type that maps a SQL DATETIME/TIMESTAMP to a PHP DateTime object with timezone
text	Type that maps a SQL CLOB to a PHP string
object	Type that maps a SQL CLOB to a PHP object using serialize() and unserialize()
array	Type that maps a SQL CLOB to a PHP array using serialize() and unserialize()
simple_array	Type that maps a SQL CLOB to a PHP array using implode() and explode(), with a comma as delimiter. IMPORTANT Only use this type if you are sure that your values cannot contain a ",".
json_array	Type that maps a SQL CLOB to a PHP array using json_encode() and json_decode()
float	Type that maps a SQL Float (Double Precision) to a PHP double. IMPORTANT: Works only with locale settings that use decimal points as separator.
guid	Type that maps a database GUID/UUID to a PHP string. Defaults to varchar but uses a specific type if the platform supports it
blob	Type that maps a SQL BLOB to a PHP resource stream

10.3. LE MAPPING DES RELATIONS ENTRE ENTITES PAR LES ANNOTATIONS

10.3.1. LA LISTE DES ANNOTATIONS POUR LE MAPPING DES RELATIONS

Suivant les cardinalités du MCD, Doctrine propose les annotations suivantes :

- @ManyToOne
- @OneToMany
- @ManyToMany
- @OneToOne
- @MappedSuperclass

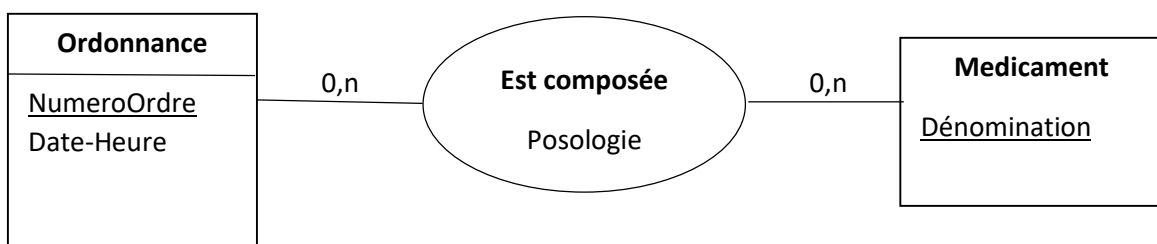
➔ **On retrouve les cardinalités MERISE, mais pas tout à fait : Attention, Voir slide suivant.**

10.3.2. UNE DIFFERENCE AVEC MERISE : RELATION MANYTOMANY AVEC PROPRIETES

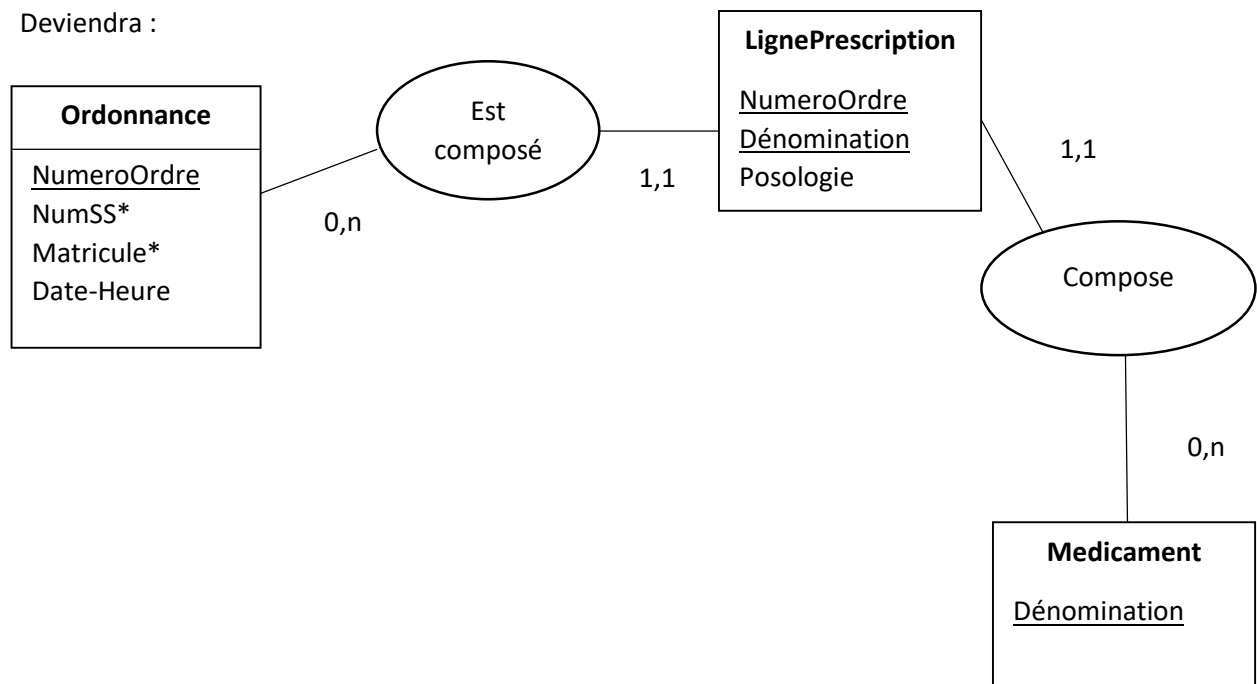
- Les relations traitées ne doivent pas être porteuses de propriétés
- Si une relation @ManyToMany porteuses de propriétés, elle devra être transformé en une entité + 2 relations OneToMany ou ManyToOne

<https://zestedesavoir.com/tutoriels/1713/doctrine-2-a-lassaut-de-lorm-phare-de-php/les-relations-avec-doctrine-2/relation-manytoone-et-onetomany-1-n/>

Exemple dans notre fil rouge :



Deviendra :



10.3.3. NOTION DE PROPRIÉTAIRE ET D'INVERSE

Dans une relation entre deux entités, il y a toujours :

- ➔ une entité dite propriétaire
- ➔ et une dite inverse (l'autre)

Pour comprendre cette notion, il faut revenir à la vieille époque, lorsque l'on faisait nos bases de données à la main.

- ➔ L'entité propriétaire est celle qui contient la référence à l'autre entité (la clef étrangère de l'autre entité).

(Alexandre BACCO, <https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/les-relations-entre-entites-avec-doctrine2>)

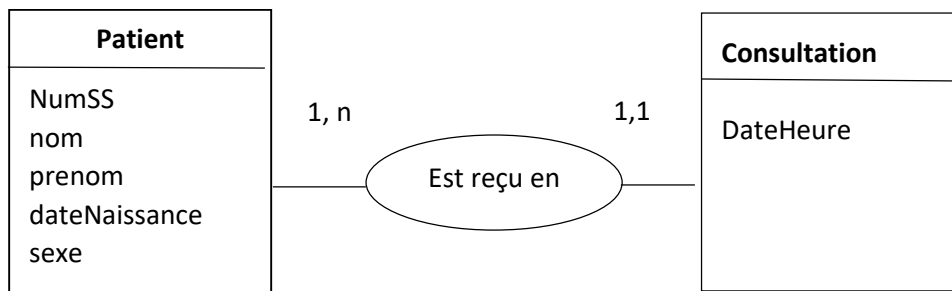
L'entité propriétaire doit obligatoirement porter l'annotation, l'entité inverse porte une annotation seulement si besoin de la relation inverse

Voir exemple ci-dessous

10.3.4. DE QUEL COTE PLACER L'ANNOTATION ?

Reprenons la relation Patient \leftrightarrow Consultation

- ➔ ManyToOne ?
du côté de l'entité propriétaire : Consultation,
- ➔ OneToMany ?
du côté de l'entité inverse : Patient



L'entité propriétaire porte l'annotation, ici Consultation, le côté inverse porte une annotation si la relation est bidirectionnelle.

L'annotation sur le côté propriétaire génère

- la **clef étrangère** dans la base de données,
 - et **une méthode** dans l'objet pour accéder à la classe inverse :
- ➔ Consultation->getPatient()

L'annotation du côté inverse permettra la méthode

- ➔ Patient->getConsultation()

Dans ce cas, cette méthode livre une collection

10.3.5. APPLICATION, LIEN BIDIRECTIONNELLE PATIENT ↔ CONSULTATION

On peut travailler directement sur les annotations des entités générées

- 1) Sur l'entité Patient, ajouter (choix d'un travail direct sur l'entité)

```
/**
 * @ORM\OneToMany(targetEntity="App\Entity\Consultation", mappedBy="NumSS")
 */
private $consultations;
```

L'attribut targetEntity désigne la classe cible (cela s'entend sans le dire), dans le cas,

L'attribut mappedBy est obligatoire sur l'entité inverse.

- 2) Générer l'entité Consultation selon la méthode utilisée pour Patient.
Ajouter,

```
/**
 * @ORM\ManyToOne(targetEntity="App\Entity\Patient", inversedBy="consultations")
 */
private $NumSS;
```


La commande « make :entity » permet aussi de générer ces annotations relationnelles

```
C:\Program Files (x86)\Ampps\www\symfony-test3>php bin/console make:entity
Class name of the entity to create or update (e.g. DeliciousElephant):

> Consultation
created: src/Entity/Consultation.php
created: src/Repository/ConsultationRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> dateHeure
Field type (enter ? to see all types) [string]:
> datetime
Can this field be null in the database (nullable) (yes/no) [no]:
>
updated: src/Entity/Consultation.php
Add another property? Enter the property name (or press <return> to stop adding fields):
> matricule
Field type (enter ? to see all types) [string]:
> ManyToOne
What class should this entity be related to?:
> Medecin
Is the Consultation.matricule property allowed to be null (nullable)? (yes/no) [yes]:
> no
Do you want to add a new property to Medecin so that you can access/update Consultation
objects from it - e.g. $medecin->getConsultations()? (yes/no) [yes]:
>
A new property will also be added to the Medecin class so that you can access the related
Consultation objects from it.
New field name inside Medecin [consultations]:
>
Do you want to activate orphanRemoval on your relationship?
A Consultation is "orphaned" when it is removed from its related Medecin.
e.g. $medecin->removeConsultation($consultation)
NOTE: If a Consultation may *change* from one Medecin to another, answer "no".
Do you want to automatically delete orphaned App\Entity\Consultation objects
(orphanRemoval)? (yes/no) [no]:
>
updated: src/Entity/Consultation.php
updated: src/Entity/Medecin.php
Add another property? Enter the property name (or press <return> to stop adding fields):
> NumSS
Field type (enter ? to see all types) [string]:
> ManyToOne
```

- 3) Et régénérer la base de données
On observe la création de la clef étrangère et de l'index associé

```
php bin/console make:migration  
php bin/console doctrine:migrations:migrate
```

10.3.6. QUELLE DIFFERENCE ENTRE INVERSEDBY ET MAPPEDBY?

```
php bin/console doctrine:schema:update --dump-sql --force
ALTER TABLE ordonnance ADD patient_id INT DEFAULT NULL;
ALTER TABLE ordonnance ADD CONSTRAINT FK_924B326C6B899279 FOREIGN
KEY (patient_id) REFERENCES patient (id);
CREATE INDEX IDX_924B326C6B899279 ON ordonnance (patient_id);

Updating database schema...
Database schema updated successfully! "3" queries were executed

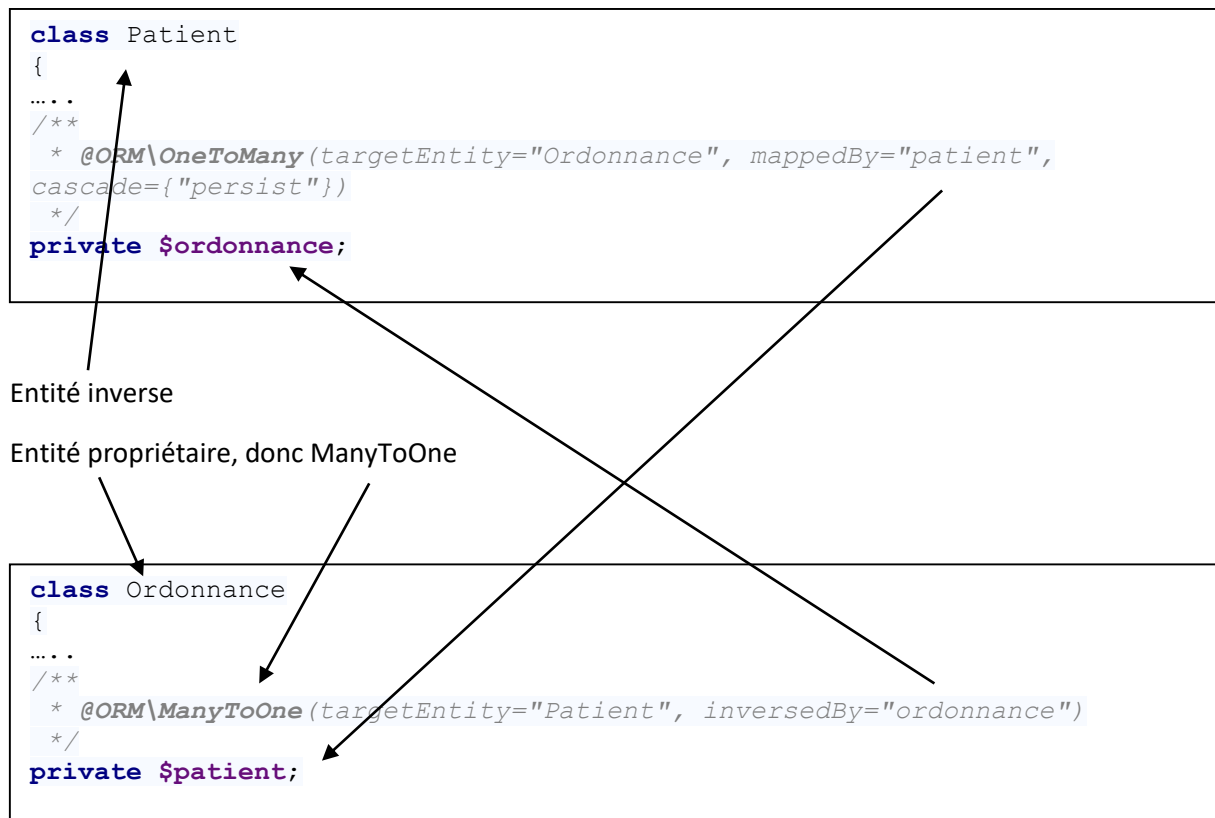
C:\Program Files (x86)\Ampss\www\my_project_name>
```

- mappedBy doit être spécifié sur le côté inversé de l'association bidirectionnelle
- inverseBy doit être spécifié sur le côté propriétaire de l'association bidirectionnelle

Remarque:

- « ManyToOne » est toujours sur le côté propriétaire de l'association bidirectionnelle
- « OneToMany » est toujours sur le côté inverse de l'association bidirectionnelle.

10.3.7. REPREDRE ICI LE CROISEMENT MAPPEDBY, INERSEBY EN METTANT EN PARRALELE LES 2 ENTITES CONCERNÉES.



10.3.9. VALIDATION DU SCHÉMA DOCTRINE

```
php bin/console doctrine:schema:validate
```

Exemple d'une faute... à corriger

```
C:\Program Files (x86)\Ampps\www\my_project_name>php bin/console  
doctrine:schema:validate
```

```
[Mapping] FAIL - The entity-class 'PN\CabinetMedicalBundle\Entity\Patient' mapping is  
invalid:
```

```
* The field PN\CabinetMedicalBundle\Entity\Patient#ordonnance is on the inverse side of a bi-  
directional relationship, but the specified mappedBy association on the target-entity  
PN\CabinetMedicalBundle\Entity\Ordonnance#patient does not contain the required  
'inversedBy="ordonnance"' attribute.
```

```
[Database] OK - The database schema is in sync with the mapping files.
```

```
doctrine:schema:validate [--skip-mapping] [--skip-sync] [--em [EM]] [-h|--help] [-q|--quiet] [-  
v|vv|vvv|--verbose] [-V|--version] [--ansi] [--no-ansi] [-n|--no-interaction] [-e|--env ENV] [--no-  
debug] [--] <command>
```

10.4. GÉNÉRER LES NOUVEAUX ACCESSEURS

<https://symfony.com/doc/current/doctrine.html#doctrine-generating-getters-and-setters>

Nouvelles annotations ManyTo.... Etc. → de nouveaux accesseurs (Getters et Setters) sont nécessaires

Par la commande :

```
php bin/console make:entity --regenerate
```

Pour écraser les anciens accesseurs, on dispose de l'option : `--overwrite`

```
C:\Program Files (x86)\Ampps\www\symfony-test>php bin/console make:entity --regenerate

This command will generate any missing methods (e.g. getters & setters) for a class or all classes
in a namespace.

To overwrite any existing methods, re-run this command with the --overwrite flag

Enter a class or namespace to regenerate [App\Entity]:
>
updated: src/Entity/Consultation.php
updated: src/Entity/LignePrescription.php
updated: src/Entity/Medecin.php
updated: src/Entity/Medicament.php
updated: src/Entity/Ordonnance.php
updated: src/Entity/Patient.php

Success!
```

Cette commande génère ainsi les accesseurs des propriétés liant les entités :

```
/**
 * @ORM\ManyToOne(targetEntity="Medecin", inversedBy="consultation")
 */
private $medecin;
.....
.....
    public function getMedecin(): ?Medecin
    {
        return $this->medecin;
    }

    public function setMedecin(?Medecin $medecin): self
    {
        $this->medecin = $medecin;

        return $this;
    }
}
```

Dans le cas d'une relation OneToMany, les setters ajoutent les éléments d'une collection

```
/**
 * @return Collection|LignePrescription[]
 */
public function getLignePrescription(): Collection
{
    return $this->lignePrescription;
}

public function addLignePrescription(LignePrescription
$lignePrescription): self
{
    if (!$this->lignePrescription->contains($lignePrescription)) {
        $this->lignePrescription[] = $lignePrescription;
        $lignePrescription->setOrdonnance($this);
    }

    return $this;
}
```

11. TP CABINET MEDICAL, PARTIE 1 A RENDRE SUR LE SERVEUR PARTAGE

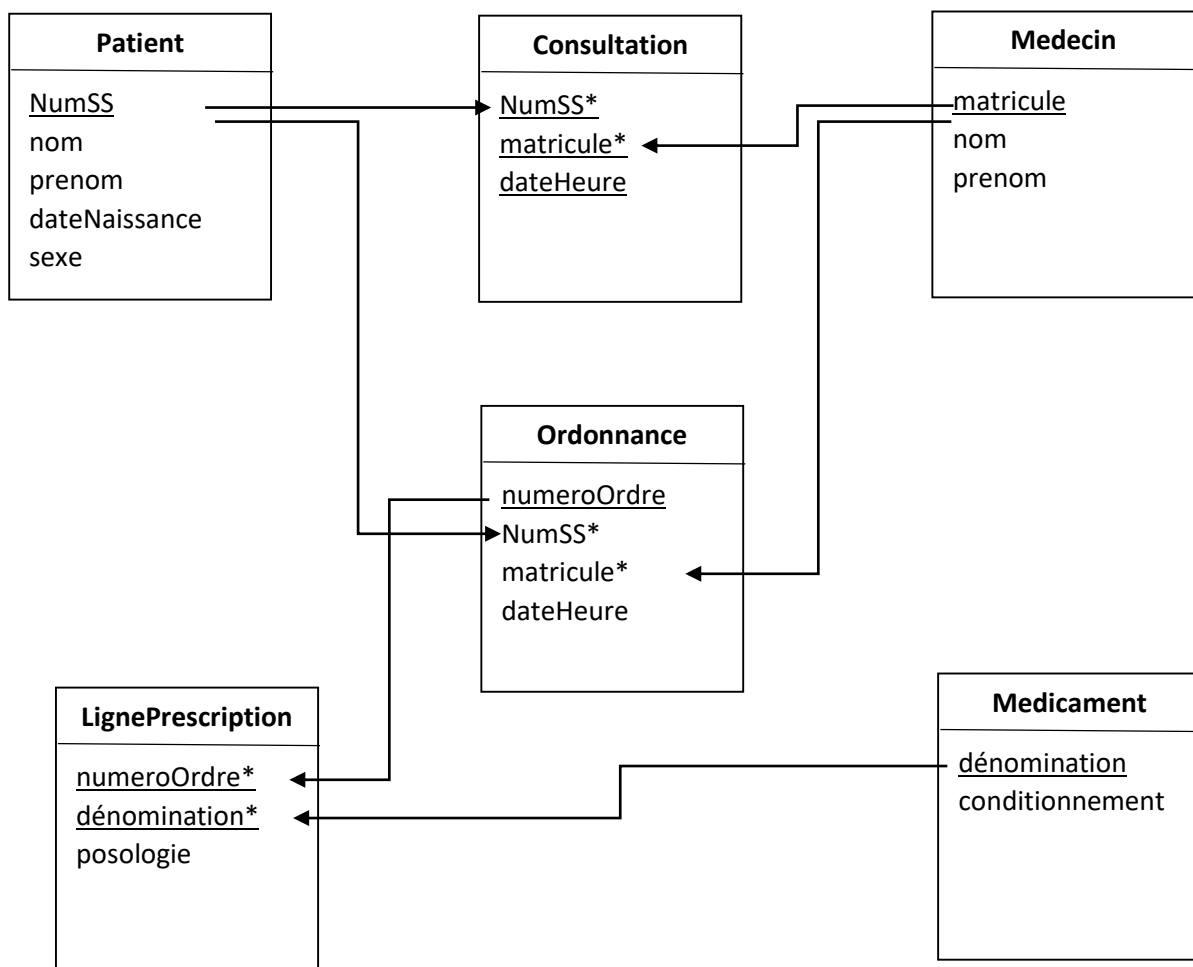
11.1. TRAVAIL DEMANDÉ :

1. Générer toutes les entités de l'application Cabinet Médical, avec leurs propriétés, leurs getters et setters.
2. Ajouter toutes les annotations nécessaires à leurs relations
3. Valider les annotations par doctrine:schema:validate
4. Générer la base de données correspondante make:migration

11.2. RAPPEL MLD

Ce MLD est à transcrire en symfony.

- ➔ Attention, pour transcrire ce MLD en entité symfony, les reports de clef primaire en clef étrangère deviennent des annotations du type * @ORM\ManyToOne(targetEntity= "... "). Ainsi toutes les clefs étrangères devront faire l'objet d'un traitement ManyToOne et non d'un simple ajout à l'entité. On devra alors traiter aussi l'entité inverse avec un OneToMany



Exemple : Pour l'entité ordonnance, on aura :

```
<?php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
/**
 * @ORM\Entity(repositoryClass="App\Repository\OrdonnanceRepository")
 */
class Ordonnance
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;
    /**
     * @ORM\Column(type="string", length=15)
     */
    private $numeroOrdre;
    /**
     * @ORM\ManyToOne(targetEntity="Patient")
     */
    private $patient;
    /**
     * @ORM\ManyToOne(targetEntity="Medecin")
     */
    private $medecin;
    /**
     * @ORM\Column(type="date")
     */
    private $dateHeure;
    public function getId(): ?int
    {
        return $this->id;
    }
    public function getNumeroOrdre(): ?string
    {
        return $this->numeroOrdre;
    }
    public function setNumeroOrdre(string $numeroOrdre): self
    {
        $this->numeroOrdre = $numeroOrdre;
        return $this;
    }
    public function getDateHeure(): ?\DateTimeInterface
    {
        return $this->dateHeure;
    }
    public function setDateHeure(\DateTimeInterface $dateHeure): self
    {
        $this->dateHeure = $dateHeure;
        return $this;
    }
}
```

En retour, si l'on souhaite lister les ordonnances d'un patient, l'entité « patient » doit contenir les lignes :

```
/**
 * @ORM\OneToMany(targetEntity="Ordonnance", mappedBy="patient",
 * cascade={"persist"})
 */
private $ordonnance;
```

12. L' ENTITYMANAGER

- ➔ La synchronisation entre les entités et la base de données s'effectue à l'aide de l'entityManager.
- ➔ Le service entityManager est accessible par le service Doctrine.

<https://symfony.com/doc/current/doctrine.html#persisting-objects-to-the-database>

12.1. LE SERVICE DOCTRINE, PERSISTANCE DES OBJETS

Le service Doctrine est celui qui va nous permettre de gérer la persistance de nos objets.

Ce service est accessible depuis le contrôleur comme n'importe quel service par :

```
<?php  
$doctrine = $this->get('doctrine');
```

Ou

```
<?php  
$doctrine = $this->getDoctrine();;
```

12.2. LE SERVICE ENTITYMANAGER

L'EntityManager de Doctrine est récupéré depuis le contrôleur via :

```
<?php  
$em = $this->getDoctrine()->getManager();
```

Ou,

```
<?php  
$em = $this->get('doctrine.orm.entity_manager');
```

12.3. INSERTION D'UN NOUVEAU PATIENT

12.3.1. REPREDRE LE CONTROLEUR PATIENTCONTROLLER DEJA GENERE

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;

class PatientController extends AbstractController
{
    /**
     * @Route("/patient", name="patient")
     */
    public function index()
    {
        return $this->render('patient/index.html.twig', [
            'controller_name' => 'PatientController',
        ]);
    }
}
```

12.3.2. AJOUTER A CE CONTROLEUR ET EXECUTER

1. La création d'une instance de la classe Patient :
 - a. `$patient = new Patient;`
 - b. Son chemin dans l'espace de noms : `use App\Entity\Patient;`
2. L'hydratation de cette instance
3. La récupération de l'entityManager
4. Les instructions de persistance dans la base de données
5. Modifier le commentaire dans l'appel à la vue
6. Exécuter `http://localhost:8000/patient`

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Annotation\Route;
use App\Entity\Patient;

class PatientController extends AbstractController
{
    /**
     * @Route("/patient", name="patient")
     */
    public function index()
    {
        $patient = new Patient;
        // Hydratation de l'instance de l'entité Patient
        $patient->setNumSS('16807751115257');
        $patient->setNom('Durand');
        $patient->setPrenom('Alex');
        $maDate = new \DateTime("1968-07-18");
        $patient-> setDateNaissance($maDate);
        $patient->setSexe('M');
        // *****
        // Récupération du service Doctrine
        $doctrine = $this->getDoctrine();
        // Récupération du service gestionnaire d'entités,
        $entityManager = $doctrine->getManager();
        // *****
        $entityManager->persist($patient);
        $entityManager->flush();

        return $this->render('patient/index.html.twig', [
            'controller_name' => 'Nouveau patient créé',
        ]);
    }
}
```

12.4. PERSIST() ET FLUSH()

Deux méthodes sont nécessaires pour l'insertion :

➔ **Persist()** :

Cela veut dire qu'à partir de maintenant cette entité (qui n'est qu'un simple objet !) est gérée par Doctrine.

Cela n'exécute pas encore de requête SQL, ni rien d'autre.

➔ **Flush()**

Doctrine va exécuter de manière effective les requêtes nécessaires pour sauvegarder les entités qu'on lui a dit de persister précédemment ;

Ces deux temps pour sauvegarder notre entité peuvent apparaître déroutants.

Ce découpage permet l'introduction de transactions.

Ainsi on peut tout à fait faire plusieurs persists sur différentes entités avant d'exécuter un seul flush. Le flush permet d'exécuter les requêtes les plus optimisées pour enregistrer tous nos persists.

Concrètement, avec notre EntityManager,

➔ `$entityManager->persist()` est équivalent à dire :

« Garde cette entité en mémoire, tu l'enregistreras au prochain flush(). »

➔ `$entityManager->flush()` est équivalent à :

« Ouvre une transaction et enregistre toutes les entités qui t'ont été données depuis le dernier flush(). »

13. TP CABINET MEDICAL, PARTIE 2 A RENDRE SUR LE SERVEUR PARTAGE

Une seule consigne :

Ajouter des consultations à ce nouveau patient

14. REPOSITORIES, METHODES DE RECUPERATION

entityManager → persistance des données

repository → récupération des données

14.1. UN REPOSITORY PAR ENTITE

```
<?php
$repository = $this
    ->getDoctrine()
    ->getManager()
    ->getRepository(Patient::class)
;
```

14.2. MÉTHODES DE RÉCUPÉRATION DE BASE

14.2.1. FIND(\$ID)

```
$patient = $repository->find(2);
```

14.2.2. FINDALL()

Toutes, format de retour Array()

14.2.3. FINDBY()

Avec filtre

```
$listPatients = $repository->findBy(  
    array('nom' => 'NOLLET'), // Critere  
    array('dateNaissance' => 'desc'), // Tri  
    5, // Limite  
    0 // Offset  
);
```

14.3. LES MÉTHODES MAGIQUES

14.3.1. FINDBYX(\$VALEUR)

En remplaçant « X » par le nom d'une propriété de votre entité

```
$listPatients = $repository->findByNom('NOLLET');
```

14.3.2. FINDONEBYX(\$VALEUR)

```
$medicament = $repository->findOneByNom('Doliprane');
```

14.4. LE QUERYBUILDER ET LE DQL

Ces méthodes de base ne permettent pas de faire des jointures !!!?

➔ Nous devons construire nos propres méthodes de récupération.

Ce sera l'objet d'une étude plus approfondie, avec le QueryBuilder et le DQL

15. TWIG : UN TOUT PREMIER EXEMPLE POUR NOTRE EXEMPLE FIL ROUGE

Nous reviendrons de manière approfondie sur la construction des templates TWIG, mais pour l'heure, nous avons besoin d'illustrer les premières notions abordées par un exemple fonctionnelle : gestion d'un cabinet médical.

15.1. TWIG EST UN MOTEUR DE TEMPLATE CREE PAR LE MEME AUTEUR : FABIEN POTENCIER, LE PATRON DE SENSIO

- TWIG est un moteur de Template très concis et claire, rapide, extensible, sécurisé.
- Le formatage est directement spécifié dans la page .html.twig
- TWIG fourni des messages d'erreur explicites
- Il n'est pas obligatoire, la vue peut aussi être rédigée en PHP.

15.2. PREMIER EXEMPLE DE SYNTAXE TWIG

15.2.1. AFFICHAGE BASIQUE D'UNE VARIABLE \$MA_DATE

```
<p class="date">{{ ma_date }}p>
```

Equivaut en PHP

```
<p class="date"><?php echo $ma_date; ?></p>
```

Twig permet de préciser le format attendu

```
<p class="date">{{ ma_date|date("d/m/Y") }}p>
```

15.2.2. PREMIERE VUE LIST.HTML.TWIG

```
{% extends 'base.html.twig' %}

{% block title %}Liste des patients{% endblock %}

{% block body %}
<style>
  .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
  .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
  <h1>Liste des patients</h1>
  <table>
    {% for a in patients %}
      <tr>
        <td>{{ a.nom }}</td>
        <td>{{ a.prenom }}</td>
        <td>{{ a.dateNaissance | date('d/m/Y') }}</td>
        <td>{{ a.NumSS }}</td>
      </tr>
    {% else %}
      <tr><td>Aucun patient n'a été trouvé.</td></tr>
    {% endfor %}
  </table>
</div>

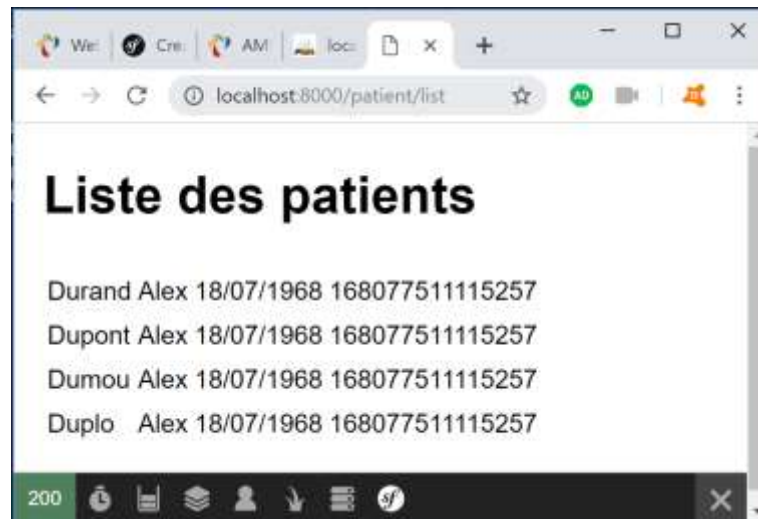
{% endblock %}
```

Ce Template permettra d'afficher la liste des patients avec ses attributs.

16. TP CABINET MEDICAL, PARTIE 3 A RENDRE SUR LE SERVEUR PARTAGE

Afficher la liste des patients comme ci-dessous :

- Ajouter l'action (la méthode) list au contrôleur PatientController
 - Ajuster la route comme suit :
 - Ajuster l'envoi à la vue
- Reprendre la vue list.html.twig du paragraphe précédent



17. MAKE :CRUD

17.1. PHP BIN/CONSOLE MAKE:CRUD

Cette commande demande pour quelle entité déjà créée par make:entity (ou par édition d'une classe correspondante dans le dossier src et son repository)

Elle génère alors :

- ➔ un nouveau contrôleur avec les actions du CRUD :
 - lister,
 - créer,
 - modifier,
 - supprimer
 - et visualiser des entités
- ➔ les templates correspondants et surtout les formulaires de création et mise à jour :

C'est au développeur d'y ajouter toute autre fonctionnalité tel que des filtres ou la pagination des listes ainsi que la mise en page.

17.2. CRUD ENTITÉ MEDECIN

```
C:\Program Files (x86)\Ampms\www\symfony-test2>php bin/console make:crud
```

```
The class name of the entity to create CRUD (e.g. GrumpyPizza):
```

```
> Medecin
```

```
created: src/Controller/MedecinController.php
```

```
created: src/Form/MedecinType.php
```

```
created: templates/medecin/_delete_form.html.twig
```

```
created: templates/medecin/_form.html.twig
```

```
created: templates/medecin/edit.html.twig
```

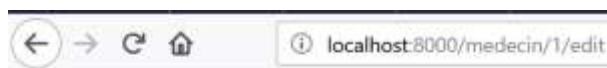
```
created: templates/medecin/index.html.twig
```

```
created: templates/medecin/new.html.twig
```

```
created: templates/medecin/show.html.twig
```

```
Success!
```

```
Next: Check your new CRUD by going to /medecin/
```



Edit Medecin

Matricule: R2345_QA_123
Nom: Nolle
Prenom: Patrick

[back to list](#)



Medecin index

Id	Matricule	Nom	Prenom	actions
1	R2345_QA_123	Nolle	Patrick	show edit
2	9876543ER45	Vaillant	Matthieu	show edit

[Create new](#)



Create new Medecin

Matricule:
Nom:
Prenom:

[back to list](#)

17.3. CRUD SUR MANYTOONE, CONSULTATION

Sur une entité en relation ManyToOne comme l'entité Consultation

```
C:\Program Files (x86)\Ampmps\www\symfony-test>php bin/console make:crud

The class name of the entity to create CRUD (e.g. FiercePuppy):

> Consultation

created: src/Controller/ConsultationController.php
created: src/Form/ConsultationType.php
created: templates/consultation/_delete_form.html.twig
created: templates/consultation/_form.html.twig
created: templates/consultation/edit.html.twig
created: templates/consultation/index.html.twig
created: templates/consultation/new.html.twig
created: templates/consultation/show.html.twig

Success!

Next: Check your new CRUD by going to /consultation/
```

Avant de tester la création d'une consultation, ajouter la méthode `__toString()` , aux entités Patient et Medecin

```
public function __toString()
{
    return $this->nom;
}
```

Elle apparait comme nécessaire à la création des listes déroulantes du formulaire.

← → ↻ ⓘ localhost:8000/consultation/new

Create new Consultation

Date heure
2014 ▼ Jan ▼ 1 ▼
Patient Dumou ▼
Medecin Nollet ▼
Save
[back to list](#)
Nollet
Vaillant

← → ↻ ⓘ localhost:8000/consultation/

Consultation index

Id DateHeure actions

1 2019-05-06 [show](#) [edit](#)

2 2014-01-03 [show](#) [edit](#)

[Create new](#)

← → ↻ ⓘ localhost:8000/consultation/1/edit

Edit Consultation

Date heure
2019 ▼ May ▼ 6 ▼
Patient Durand ▼
Medecin Vaillant ▼
Update
[back to list](#)
Delete

18. TP CABINET MEDICAL, PARTIE 4 A RENDRE SUR LE SERVEUR PARTAGE

Générer l'ensemble des CRUD nécessaires à l'application Cabinet Médical.

L'application fonctionnelle doit tourner sur le serveur partagé.

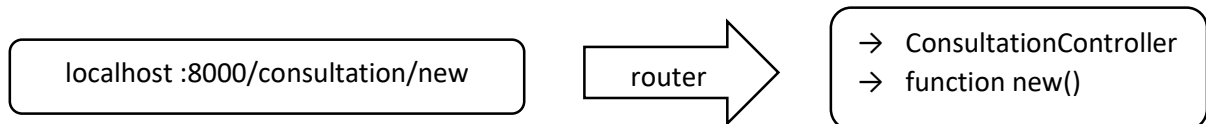
19. ROUTAGE, CONTROLEURS ET ACTIONS

19.1. ACTIONS, CONTÔLEUR

« Editer un patient » est une action.

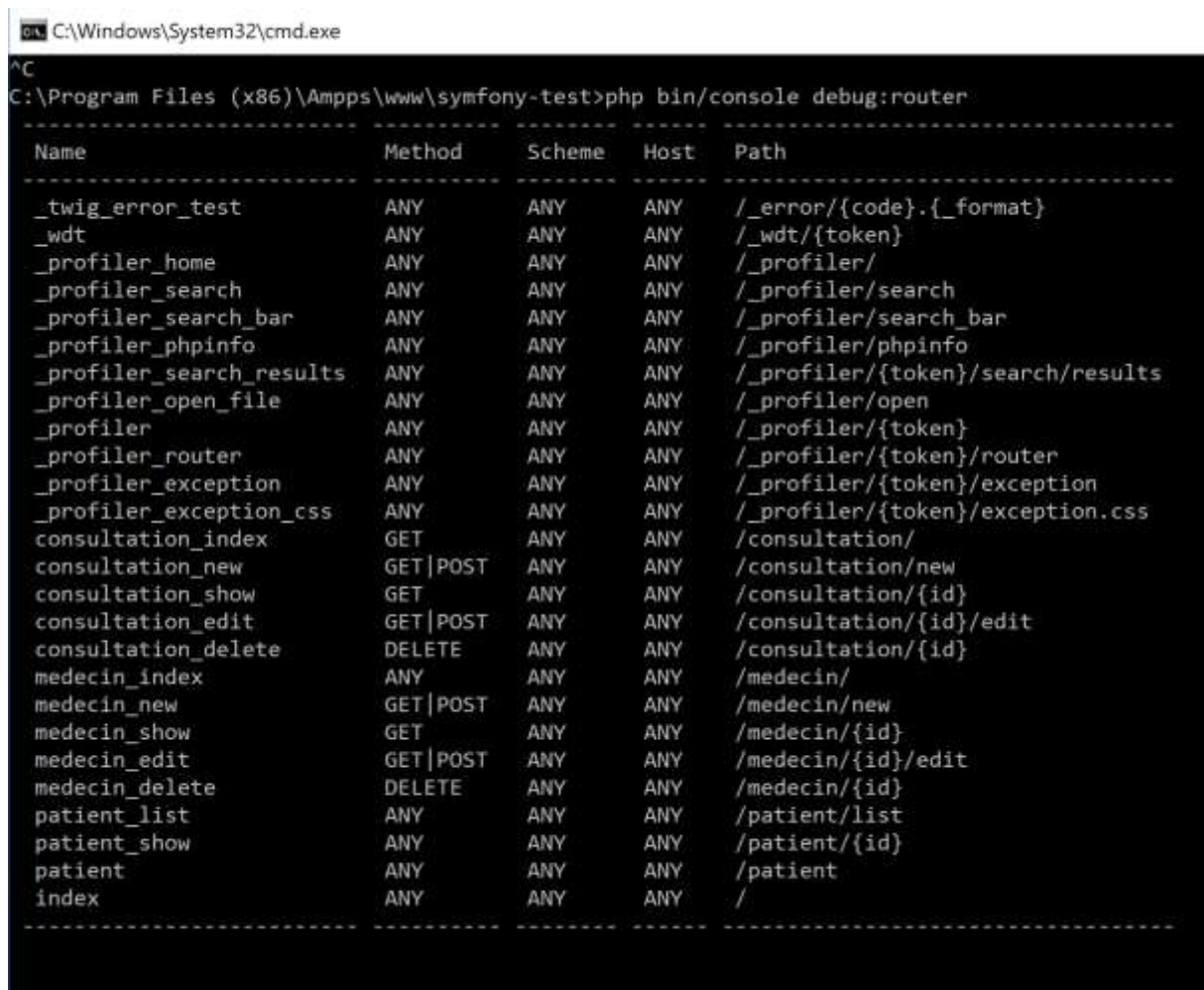
MERISE : une action est une tâche, le composant le plus élémentaire d'une procédure

Comment lier une requête http à un contrôleur secondaire et une action de ce contrôleur ?



19.2. TESTER LE ROUTAGE DE VOTRE APPLICATION

PHP BIN/CONSOLE DEBUG :ROUTER



Name	Method	Scheme	Host	Path
_twig_error_test	ANY	ANY	ANY	/_error/{code}._{format}
_wdt	ANY	ANY	ANY	/_wdt/{token}
_profiler_home	ANY	ANY	ANY	/_profiler/
_profiler_search	ANY	ANY	ANY	/_profiler/search
_profiler_search_bar	ANY	ANY	ANY	/_profiler/search_bar
_profiler_phpinfo	ANY	ANY	ANY	/_profiler/phpinfo
_profiler_search_results	ANY	ANY	ANY	/_profiler/{token}/search/results
_profiler_open_file	ANY	ANY	ANY	/_profiler/open
_profiler	ANY	ANY	ANY	/_profiler/{token}
_profiler_router	ANY	ANY	ANY	/_profiler/{token}/router
_profiler_exception	ANY	ANY	ANY	/_profiler/{token}/exception
_profiler_exception_css	ANY	ANY	ANY	/_profiler/{token}/exception.css
consultation_index	GET	ANY	ANY	/consultation/
consultation_new	GET POST	ANY	ANY	/consultation/new
consultation_show	GET	ANY	ANY	/consultation/{id}
consultation_edit	GET POST	ANY	ANY	/consultation/{id}/edit
consultation_delete	DELETE	ANY	ANY	/consultation/{id}
medecin_index	ANY	ANY	ANY	/medecin/
medecin_new	GET POST	ANY	ANY	/medecin/new
medecin_show	GET	ANY	ANY	/medecin/{id}
medecin_edit	GET POST	ANY	ANY	/medecin/{id}/edit
medecin_delete	DELETE	ANY	ANY	/medecin/{id}
patient_list	ANY	ANY	ANY	/patient/list
patient_show	ANY	ANY	ANY	/patient/{id}
patient	ANY	ANY	ANY	/patient
index	ANY	ANY	ANY	/

On y retrouve l'ensemble des routes décrites dans les annotations dans les contrôleurs

19.2.1. EXEMPLE DE ROUTES EXPRIMEES PAR LES ANNOTATIONS DANS LES CONTROLEURS

```
/**
 * @Route("/new", name="consultation_new", methods={"GET","POST"})
 */
/**
 * @Route("/{id}", name="consultation_show", methods={"GET"})
 */
/**
 * @Route("/{id}", name="consultation_delete", methods={"DELETE"})
 */
```


19.3. CONFIG/ROUTES.YAML

Les routes peuvent indifféremment se rédiger à l'aide

- Des annotations (cf ci-dessus)
- En YAML, dans le fichier config/routes.yaml

19.3.1. ROUTES EN ANNOTATION

```
/**
 * @Route("/new", name="consultation_new", methods={"GET", "POST"})
 */
/**
 * @Route("/{id}", name="consultation_show", methods={"GET"})
 */
/**
 * @Route("/{id}", name="consultation_delete", methods={"DELETE"})
 */
```

19.3.2. ROUTES EQUIVALENTE EN YAML DANS CONFIG/ROUTES.YAML

```
# config/routes.yaml
index:
  path: /
  controller: App\Controller\DefaultController::index
consultation_edit:
  path: /consultation/{id}/edit
  controller: App\Controller\ConsultationController::edit
  requirements:
    id: '\d+'
consultation_show:
  path: /consultation/{id}
  controller: App\Controller\ConsultationController::show
  requirements:
    id: '\d+'
    method: POST
consultation_list:
  path: /consultation/
  controller: App\Controller\ConsultationController::index
```

Résultat dans le debug :router

consultation_index	GET	ANY	ANY	/consultation/
consultation_new	GET POST	ANY	ANY	/consultation/new
consultation_delete	DELETE	ANY	ANY	/consultation/{id}
medecin_index	ANY	ANY	ANY	/medecin/
medecin_new	GET POST	ANY	ANY	/medecin/new
medecin_show	GET	ANY	ANY	/medecin/{id}
medecin_edit	GET POST	ANY	ANY	/medecin/{id}/edit
medecin_delete	DELETE	ANY	ANY	/medecin/{id}
patient_list	ANY	ANY	ANY	/patient/list
patient_show	ANY	ANY	ANY	/patient/{id}
patient	ANY	ANY	ANY	/patient
index	ANY	ANY	ANY	/
consultation_edit	ANY	ANY	ANY	/consultation/{id}/edit
consultation_show	ANY	ANY	ANY	/consultation/{id}
consultation_list	ANY	ANY	ANY	/consultation/

19.4. SYNTAXE YAML

19.4.1. UNE ETIQUETTE + UNE URL + UNE SORTIE

1. Chacun des blocs est identifiée par une étiquette, cette étiquette pourra être reprise dans les vues pour construire les liens correspondants.
2. Puis une ligne path, la description du path est un filtre permettant de capturer (ou non) l'url proposée
3. Elle est suivie d'une ligne décrivant le contrôleur et l'action appelés.
4. Le routeur essaie les paths les uns derrière les autres jusqu'au premier qui fait la correspondance correcte. L'ordre des routes est donc très importante : On placera les plus spécifiques en premier, les plus générales en dernier...

Si nécessaire des conditions sont ajoutées dans une quatrième ligne "requirements" : une expression régulière. C'est un filtre qui détaille les conditions que doivent remplir les paramètres.

ATTENTION : L'INDENTATION N'EST PAS UNE TABULATION MAIS 4 BLANCS

19.4.2. PARAMÈTRE

La présence de {id} précise qu'un paramètre est attendu et obligatoire : il sera transmis au contrôleur.

La condition id: \d+ impose que ce paramètre doit être numérique, d'un ou plusieurs chiffres.

Exemples à plusieurs paramètres

L'url : /salleTp/salle/A/23 sera capturée par la route suivante

```
qcm_salle_tp_voir_salle:  
  path: /salleTp/salle/{batiment}/{numero}  
  controller: App\Controller\SalleController::voirSalle  
  requirements:  
    batiment: A|B|C|D  
    numero: \d{1,3}
```

Paramètre format

Ce paramètre spécifique détermine le Content-type de la réponse

```
qcm_salle_tp_voir_salle:  
  path: /salleTp/salle/{document}._format  
  controller: App\Controller\SalleController::voirSalle  
  requirements:  
    _format: xml|rss
```

19.4.3. PROFILER

La recherche du bon pattern est visualisable par le profiler (onglet Routing), comme ci-dessous

19	medecin_edit	/medecin/{id}/edit	Path does not match
20	medecin_delete	/medecin/{id}	Path does not match
21	patient_list	/patient/list	Path does not match
22	patient_show	/patient/{id}	Path does not match
23	patient	/patient	Path does not match
24	index	/	Path does not match
25	consultation_edit	/consultation/{id}/edit	Route matches!

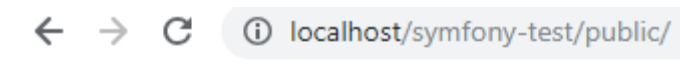
19.5. TESTEZ-VOUS

Les URLs suivantes peuvent-elles être capturées par la route ?

- `/articles/en/2010/my-post`
- `/articles/fr/2010/my-post.rss`
- `/articles/de/2013/my-first-post.html`
- `/articles/en/2013/my-latest-post.html`

20. TP CABINET MEDICAL, PARTIE 5 A RENDRE SUR LE SERVEUR PARTAGE

Créer une page d'accueil accessible à partir de localhost/symfony-test/public/ , par exemple :



Accueil

- [Consultations](#)
- [Médecins](#)
- [Patients](#)
- [Médicaments](#)

- 1- Créer un nouveau templates/home/index.html.twig
- 2- Dans ce fichier, ajuster les liens à l'aide la commande {{ path('etiquette_router') }}, par

```
<a href="{{ path('consultation_list') }}">Consultations</a>
```

exemple

- 3- Toutes les étiquettes doivent être définies dans le config/router.yaml, par exemple

```
consultation_list:  
  path: /consultation/  
  controller: App\Controller\ConsultationController::index
```

21. FORMULAIRE, PREMIERE PRISE EN MAIN SIMPLIFIEE

<https://symfony.com/doc/current/forms.html>

21.1. INSTALLER LE PACKAGE FORM

Afin de construire des formulaires, il faut charger les packages correspondant

```
composer require symfony/form
```


21.2. CONSTRUCTION D'UN FORMULAIRE EN 4 TEMPS ET A LA MAIN

La construction d'un formulaire peut être largement automatisée :

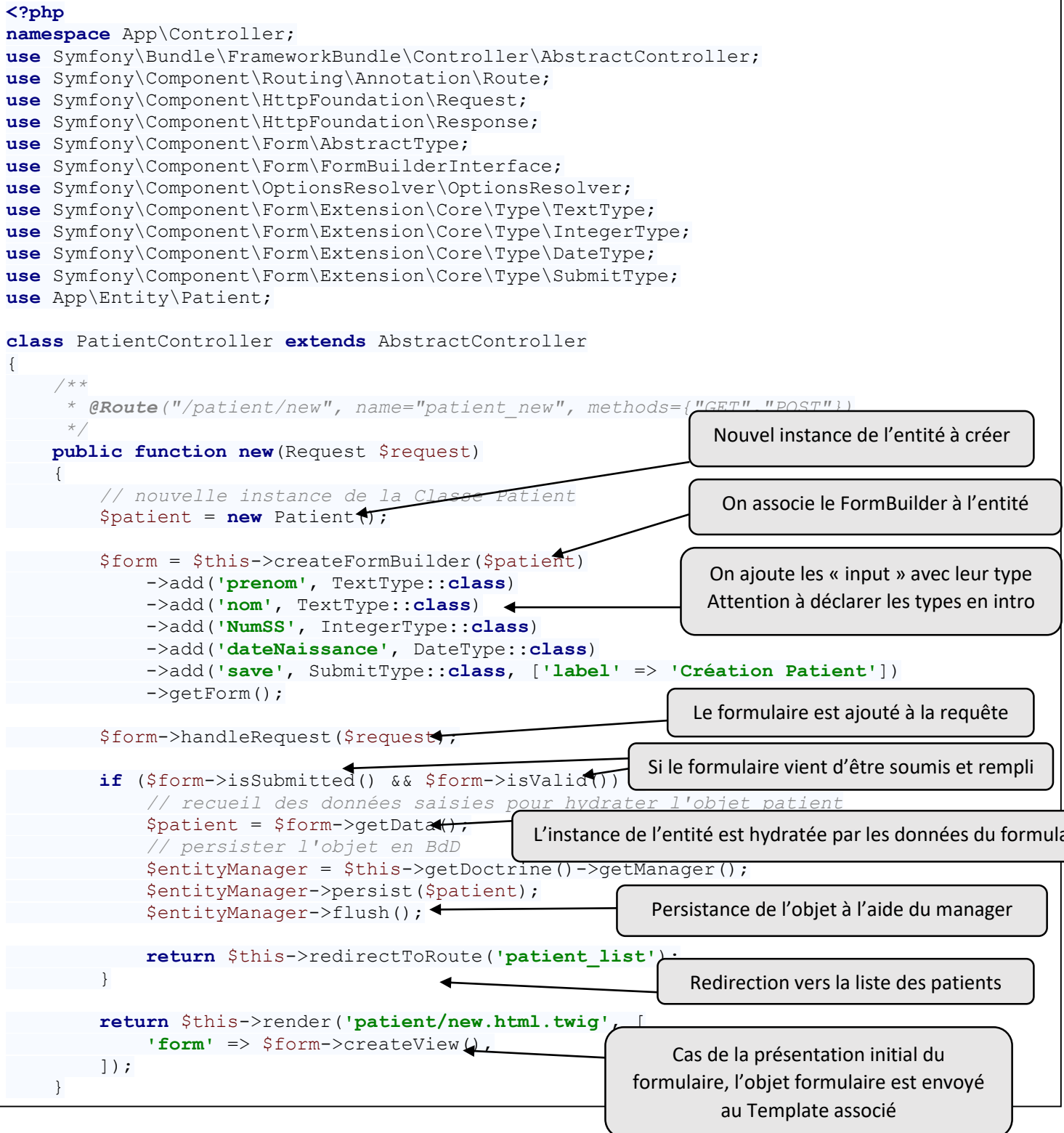
- ➔ cf la section make:form,
- ➔ ou la section make:crud (plus loin dans ce cours)

MAIS apprendre symfony, c'est aussi plonger dans le code

Dans un premier temps, nous allons apprendre à construire un formulaire à la mano. Voici la marche à suivre :

1. On attache l'entité à hydrater (pour nous \$patient) à FormBuilder
(L'objet FormBuilder n'est pas un formulaire mais un constructeur de formulaire)
2. On ajoute au FormBuilder les champs nécessaires (add)
3. On génère l'objet \$form, notre objet formulaire
4. Voir slide suivant

5. On le passe à la vue qui l'affichera



21.3. TWIG ASSOCIÉ AU FORMULAIRE

```
{% extends 'base.html.twig' %}

{% block title %}Liste des patients{% endblock %}

{% block body %}
    <h1>Patient index</h1>

    <table class="table">
        <thead>
            <tr>
                <th>Id</th>
                <th>NumSS</th>
                <th>Nom</th>
                <th>Prenom</th>
                <th>Date de Naissance</th>
                <th>actions</th>
            </tr>
        </thead>
        <tbody>
            {% for patient in patients %}
                <tr>
                    <td>{{ patient.id }}</td>
                    <td>{{ patient.NumSS }}</td>
                    <td>{{ patient.nom }}</td>
                    <td>{{ patient.prenom }}</td>
                    <td>{{ patient.dateNaissance|date("d/m/Y") }}</td>
                    <td>
                        <a href="{{ path('patient_show', {'id': patient.id})
}}">show</a>
                    </td>
                </tr>
            {% else %}
                <tr>
                    <td colspan="5">no records found</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>

    <a href="{{ path('patient_new') }}">Create new</a>
{% endblock %}
```

22. TP CABINET MEDICAL, PARTIE 6 A RENDRE SUR LE SERVEUR PARTAGE

Intégrer le formulaire spécifique de création d'un patient à votre application.

Ajouter à l'entité Patient une propriété « email »

Suite à cet ajout, ajuster les pages de visualisations et de création.

23. GESTION DES UTILISATEURS ET SECURITE (SANS FOSUSER)

Avec les versions antérieures à Symfony 4.0, la gestion des utilisateurs et de la sécurité pouvaient être confiés au célèbre bundle très performant : FOSUSERBundle .

La version 4 permet maintenant de façon simple et directe d'installer les premières fonctionnalités de gestion des utilisateurs et de sécurité, à l'aide de 2 commandes :

- `make :user`
- `make :auth`

Cette installation simple est bien décrite dans la documentation officielle :

<https://symfony.com/doc/current/security.html>

Nous allons suivre ce plan :

1. Installer le security-bundle
2. Créez la classe Users : `php bin/console make :user`
3. Création d'un formulaire d'authentification `make :auth`
4. Créer le formulaire d'enregistrement d'un user, `make:registration-form`
5. Configurer les pare-feux avec leurs modes d'authentification, de login et de logout
6. Ajuster le contrôle d'accès suivant les rôles
7. Récupération de l'objet utilisateur actuel.

23.1. MAKE :USER

En premier vérifier que le security-bundle est bien à jour :

```
composer require symfony/security-bundle
```

Puis créer votre Classe User

```
php bin/console make:user
```

```
C:\Program Files (x86)\Ampps\www\symfony-test>php bin/console make:user
```

```
The name of the security user class (e.g. User) [User]:
```

```
>
```

```
Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
```

```
>
```

```
Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
```

```
> email
```

Email est choisi comme identifiant

```
Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).
```

```
Does this app need to hash/check user passwords? (yes/no) [yes]:
```

```
>
```

```
created: src/Entity/User.php
```

```
created: src/Repository/UserRepository.php
```

```
updated: src/Entity/User.php
```

```
updated: config/packages/security.yaml
```

make:user produit

- Une Classe User dans le dossier entity
- Une Classe UserRepository.php dans le dossier UserRepository
- Un fichier de configuration security.yaml

```
Success!
```

```
Next Steps:
```

- Review your new App\Entity\User class.
- Use make:entity to add more fields to your User entity and then run make:migration.
- Create a way to authenticate! See <https://symfony.com/doc/current/security.html>

Les étapes suivantes :

- Visualiser User class
- Ajouter des champs nécessaires au User par make:entity
- Mettre à jour la Bdd
- Créer une authentification

```

namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;
/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 */
class User implements UserInterface
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;
    /**
     * @ORM\Column(type="string", length=180, unique=true)
     */
    private $email;
    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];
    /**
     * @var string The hashed password
     * @ORM\Column(type="string")
     */
    private $password;
    public function getId(): ?int
    {
        return $this->id;
    }
    public function getEmail(): ?string
    {
        return $this->email;
    }
    public function setEmail(string $email): self
    {
        $this->email = $email;
        return $this;
    }
    /**
     * A visual identifier that represents this user.
     *
     * @see UserInterface
     */
    public function getUsername(): string
    {
        return (string) $this->email;
    }
    /**
     * @see UserInterface
     */
    public function getRoles(): array
    {
        $roles = $this->roles;
        // guarantee every user at least has ROLE_USER
        $roles[] = 'ROLE_USER';
        return array_unique($roles);
    }
    public function setRoles(array $roles): self
    {
        $this->roles = $roles;
        return $this;
    }

```

```

/**
 * @see UserInterface
 */
public function getPassword(): string
{
    return (string) $this->password;
}
public function setPassword(string $password): self
{
    $this->password = $password;
    return $this;
}

/**
 * @see UserInterface
 */
public function getSalt()
{
    // not needed when using the "bcrypt" algorithm in security.yaml
}

/**
 * @see UserInterface
 */
public function eraseCredentials()
{
    // If you store any temporary, sensitive data on the user, clear
    it here
    // $this->plainPassword = null;
}
}

```

La commande `make :user` a aussi généré un fichier de configuration de la sécurité :

`config/packages/security.yaml`

Nous le découvrirons dans la suite.

Mettre à jour la base de données :

```
php bin/console make:migration
```

```
php bin/console doctrine:migrations:migrate
```

La table user est créée.

Attention si la version de MySQL <5.7 → problème avec le format Json de la rubrique rôle, → Upgrader

23.1. MAKE :AUTH, GENERER UN FORMULAIRE D'AUTHENTIFICATION (CONNEXION)

La commande `make:auth` amorce la création d'un formulaire d'authentification. Ce formulaire à compléter manuellement permettra en outre de disposer à tout moment de l'exécution de l'application de l'utilisateur connecté.

(cf détails : https://symfony.com/doc/current/security/form_login_setup.html)

```
php bin/console make:auth
```

```
C:\Program Files (x86)\Ampps\www\symfony-test>php bin/console make:auth
```

```
What style of authentication do you want? [Empty authenticator]:
```

```
[0] Empty authenticator
```

```
[1] Login form authenticator
```

```
> 1
```

```
The class name of the authenticator to create (e.g. AppCustomAuthenticator):
```

```
> LoginForm
```

```
Choose a name for the controller class (e.g. SecurityController) [SecurityController]:
```

```
>
```

```
created: src/Security/LoginFormAuthenticatorAuthenticator.php
```

```
updated: config/packages/security.yaml
```

```
created: src/Controller/SecurityController.php
```

```
created: templates/security/login.html.twig
```

```
Success!
```

```
Next:
```

```
- Customize your new authenticator.
```

```
- Finish the redirect "TODO" in the
```

```
App\Security\LoginFormAuthenticatorAuthenticator::onAuthenticationSuccess() method.
```

```
- Review & adapt the login template: templates/security/login.html.twig.
```

- Création d'un Authenticator
- Mise à jour de la configuration de sécurité
- Création d'un contrôleur SecurityController
- Création d'un Twig

Tester avec `localhost/symfony-test/public/login`



← → ↻ localhost/symfony-test/public/login

🔍 Débuter avec Firefox 📖 Débuter avec Firefox 📌 Firefox Bookmarks

Please sign in

Email Password

Nous pouvons utilement commenter le contrôleur correspondant (à éditer dans votre IDE courant)

```
src/Controller/SecurityController.php
```

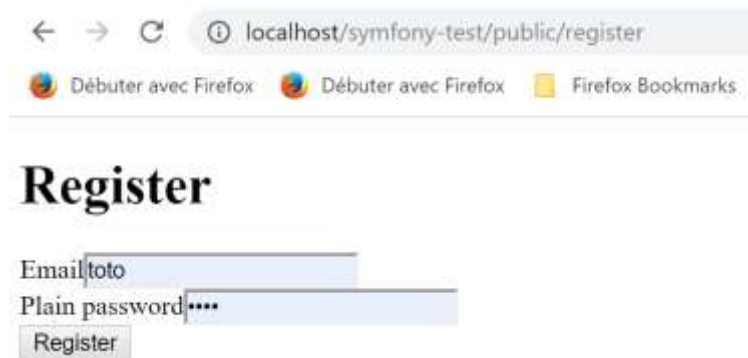
23.2. AJOUTER UN FORMULAIRE D'ENREGISTREMENT

```
php bin/console make:registration-form
```

```
C:\Program Files (x86)\Ampps\www\symfony-test>php bin/console make:registration-form

Creating a registration form for App\Entity\User
Do you want to add a @UniqueEntity validation annotation on your User class to make sure
duplicate accounts aren't created? (yes/no) [yes]:
> yes
Do you want to automatically authenticate the user after registration? (yes/no) [yes]:
>
updated: src/Entity/User.php
created: src/Form/RegistrationFormType.php
created: src/Controller/RegistrationController.php
created: templates/registration/register.html.twig
Success!
Next: Go to /register to check out your new form!
Make any changes you need to the form, controller & template.
```

Un formulaire d'enregistrement est donc maintenant créé :



← → ↻ ⓘ localhost/symfony-test/public/register

🦊 Débuter avec Firefox 🦊 Débuter avec Firefox 📌 Firefox Bookmarks

Register

Email

Plain password

23.2.1. AVEC SON CONTROLEUR ET SON ACTION :

```
<?php
namespace App\Controller;
use App\Entity\User;
use App\Form\RegistrationFormType;
use App\Security\LoginFormAuthenticatorAuthenticator;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use
Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;
use Symfony\Component\Security\Guard\GuardAuthenticatorHandler;

class RegistrationController extends AbstractController
{
    /**
     * @Route("/register", name="app_register")
     */
    public function register(Request $request,
UserPasswordEncoderInterface $passwordEncoder, GuardAuthenticatorHandler
$guardHandler, LoginFormAuthenticatorAuthenticator $authenticator):
Response
    {
        $user = new User();
        $form = $this->createForm(RegistrationFormType::class, $user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // encode the plain password
            $user->setPassword(
                $passwordEncoder->encodePassword(
                    $user,
                    $form->get('plainPassword')->getData()
                )
            );
            $entityManager = $this->getDoctrine()->getManager();
            $entityManager->persist($user);
            $entityManager->flush();
            // do anything else you need here, like send an email
            return $guardHandler->authenticateUserAndHandleSuccess(
                $user,
                $request,
                $authenticator,
                'main' // firewall name in security.yaml
            );
        }

        return $this->render('registration/register.html.twig', [
            'registrationForm' => $form->createView(),
        ]);
    }
}
```

23.2.2. COMPLETONS LE CONTROLEUR APPELANT LE FORMULAIRE D'ENREGISTREMENT EN FIXANT UN ROLE INITIAL (ROLE_USER) :

C:\Program Files (x86)\Ampps\www\symfony-test\src\Form\RegistrationFormType.php

Complétons l'action d'enregistrement en imposant un rôle :

C:\Program Files (x86)\Ampps\www\symfony-test\src\Controller\RegistrationController.php

```
if ($form->isSubmitted() && $form->isValid()) {  
    // encode the plain password  
    $user->setPassword(  
        $passwordEncoder->encodePassword(  
            $user,  
            $form->get('plainPassword')->getData()  
        )  
    );  
    $user->setRoles(["ROLE_USER"]);  
    $entityManager = $this->getDoctrine()->getManager();  
    $entityManager->persist($user);  
    $entityManager->flush();  
    // do anything else you need here, like send an email  
    return $guardHandler->authenticateUserAndHandleSuccess(  
        $user,  
        $request,  
        $authenticator,  
        'main' // firewall name in security.yaml  
    );  
}
```

23.2.3. CRÉER PLUSIEURS USERS



The screenshot shows a web browser window with the address bar displaying `localhost/symfony-test/public/register`. The page title is "Register". The form contains the following fields and values:

- Email: `patrick.nollet@sorbonne-un`
- Plain password: `*****`
- Register button: A button labeled "Register".
- Username: `Patrick`

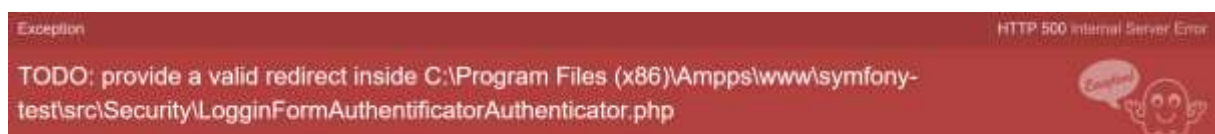
Aller, sous PHPMyAdmin, modifier le rôle d'un user en `ROLE_ADMIN`



The screenshot shows the PHPMyAdmin user editing interface. The fields and their values are as follows:

Field	Type	Value
id	int(11)	3
email	varchar(180)	patrick.nollet@sorbonne-universite.fr
roles	json	["ROLE_ADMIN"]
password	varchar(255)	\$2y\$13\$c./w0wGUY0mGo.6p1bpyG0AcCVtDbKcs4e2T52c8xV2qjxHpduGK
username	varchar(255)	Patrick

ATTENTION : La création d'un user renvoie vers l'authentification, si elle n'a pas été encore définie (make :auth) vous obtiendrez l'alerte suivante :



23.3. DECOUVRIR LE FICHIER DE CONFIGURATION SECURITY.YAML

Editer le fichier de configuration de la sécurité security.yaml : \config\packages\security.yaml

23.3.1. DEUX PARTIES PRINCIPALES

- L'authentification
 - L'autorisation.
- ➔ L'authentification consiste à découvrir qui vous êtes et à vous le prouver. C'est le processus de connexion.
- ➔ L'autorisation survient après l'authentification: il s'agit de déterminer si vous avez accès à quelque chose.

```
security:
  encoders:
    App\Entity\User:
      algorithm: bcrypt

# https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
  providers:
    # used to reload user from session & other features (e.g. switch user)
    app_user_provider:
      entity:
        class: App\Entity\User
        property: email
  firewalls:
    dev:
      pattern: ^/(_(profiler|wdt)|css|images|js)/
      security: false
    main:
      anonymous: true
      guard:
        authenticators:
          - App\Security\AppUserAuthenticator

      # activate different ways to authenticate

      # http_basic: true
      # https://symfony.com/doc/current/security.html#a-configuring-how-your-
      # users-will-authenticate

      # form_login: true
      # https://symfony.com/doc/current/security/form_login_setup.html

      # Easy way to control access for large sections of your site
      # Note: Only the *first* access control that matches will be used
  access_control:
    # - { path: ^/admin, roles: ROLE_ADMIN }
    # - { path: ^/profile, roles: ROLE_USER }
```

23.4. ENCODAGE DES MOTS DE PASSE

```
encoders:
    # use your user class name here
    App\Entity\User:
        # bcrypt or argon2i are recommended
        # argon2i is more secure, but requires PHP 7.2 or the Sodium
        extension
        algorithm: bcrypt
        cost: 12
```

Choix de l'algorithme

Coût consenti (facultatif)

23.5. PROVIDERS

Un provider est un fournisseur d'utilisateurs. Les firewalls s'adressent aux providers pour récupérer les utilisateurs et les identifier.

```
# https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
```

Nom arbitraire étiquetant le provider

Entity : les users sont en bases de données, Le fournisseur Entity a besoin :
d'une classe, ici User
d'une propriété pour y accéder, ici email

Dans l'exemple suivant, le fournisseur d'utilisateurs est memory, les utilisateurs sont réduits à deux : « bob » et « sarah »

```
providers:
    mes_utilisateurs :
        memory:
            users:
                bob:
                    password: pa$$
                    roles: ['ROLE_USER']
                sarah:
                    password: 4Dm1nP4$$
                    roles: ['ROLE_USER', 'ROLE_ADMIN']
```

Le fournisseur d'utilisateur peut être :

- Une base de données non gérées par Doctrine, dans ce cas il faudra définir une classe UserProvider implémentant l'interface UserProviderInterface [Bilal ARMANI, Symfony 3 ENI Edition, p 360]
- un annuaire LDAP : utiliser le bundle IMAGLdapBundle
- une authentification OAuth (par Twitter ou Facebook), à l'aide du bundle HWIOAuthBundle

23.6. REVUE DU FICHIER DE CONFIGURATION DE LA SECURITE SECURITY.YAML

La configuration de la gestion de la sécurité de l'application est consignée dans le fichier security.yaml (\config\packages\ security.yaml)

23.6.1. FIREWALLS

<https://symfonycasts.com/screencast/symfony-security/firewalls-authenticator>

Dans symfony, le travail du pare-feu est de vous authentifier :

- 1 - Un pattern d'URL définissant une partie de l'application
- 2 – Une modalité d'authentification
- 3 – les routes du login et du logout

```
firewalls:
  dev:
    pattern: ^/(_(profiler|wdt)|css|images|js)/
    security: false
  main:
    anonymous: true
    guard:
      authenticators:
        - App\Security\LoginFormAuthenticatorAuthenticator

    # activate different ways to authenticate

    # http_basic: true
    # https://symfony.com/doc/current/security/how-your-users-will-authenticate.html

    # form_login: true
    # https://symfony.com/doc/current/security/form-login.html
```

Un pare-feu (firewall), c'est :

- Un nom ou étiquette
- Un domaine de l'application définit par un pattern (expression régulière)
- Une modalité d'authentification
 - Directement par le serveur
 - Ou par un formulaire d'authentification
- Indique la route du formulaire d'authentification

Le premier firewall définit est un faux : il permet simplement au développeur d'avoir toujours accès aux outils de débogage _profiler et _dwt

23.6.2. PATTERN :

C'est le premier firewall dont le pattern matche avec l'url qui prend en main la requête et la garde (les firewalls suivants sont ignorés). Exemple : un pattern captant toutes les routes commençant par /admin

```
security:
  firewalls:
    secured_area:
      pattern: ^/admin
      # ...
```

Rappel : L'accent circonflexe ^ précise que seules les URLs commençant avec le pattern seront captures. Par exemple, le pattern /admin (sans le ^) matche /admin/foo mais aussi avec /foo/admin.

23.6.3. LOGIN

```
firewalls:
    main:
# ...
    form_login:
        check_path: app_login
```

Le check_path reprend la route définie dans le contrôleur

```
class SecurityController extends AbstractController
{
    /**
     * @Route("/login", name="app_login")
     */
    public function login(AuthenticationUtils $authenticationUtils):
Response
```

Web : https://symfony.com/doc/current/security/form_login_setup.html

23.6.1. LOGOUT

Le logout doit être activé au niveau du pare-feu.

Il n'y a ni contrôleur ni action explicite pour le logout, elle s'effectue de manière automatique.

```
firewalls:
  main:
    # ...
    logout:
      path: app_logout
      target: accueil

    # where to redirect after logout
    # target: app_any_route
```

La route du logout doit figurer dans le routes.yaml, mais elle n'est associée à aucun contrôleur

```
index:
  path: /
  controller: App\Controller\DefaultController::index
app_logout:
  path: /logout
  methods: GET
```

La redirection après logout est une route définie dans un contrôleur ou dans routes.yaml

```
class DefaultController extends AbstractController
{
    /**
     * @Route("/index", name="accueil")
     */
    public function index(): Response
```


23.7. CONTROLE D'ACCÈS ET RÔLE

https://symfony.com/doc/current/security/access_control.html

Il y a 3 (voire 4) manière de contrôler l'accès :

- Contrôle d'accès à un sous-domaine (dans le fichier security.yaml)
- Contrôle d'accès au contrôleur
- Contrôle d'accès à une action d'un contrôleur
- Contrôle d'accès à une partie d'un TWIG

23.7.1. ACCESS_CONTROL (SECURITY.YAML):

```
access_control:
  # matches /admin/users/*
  - { path: ^/admin/users, roles: ROLE_SUPER_ADMIN }

  # matches /admin/* except for anything matching the above rule
  - { path: ^/admin, roles: ROLE_ADMIN }
```

23.7.2. ANNOTATIONS DANS LE CONTRÔLEUR

<https://symfony.com/doc/current/bundles/SensioFrameworkExtraBundle/annotations/security.html>

```
<?php
// src/Controller/AdminController.php
// ...

+ use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;

+ /**
+  * Require ROLE_ADMIN for every controller method in this class.
+  *
+  * @IsGranted("ROLE_ADMIN")
+  */
class AdminController extends AbstractController
{
+     /**
+      * Require ROLE_ADMIN for only this controller method.
+      *
+      * @IsGranted("ROLE ADMIN")
+      */
    public function adminDashboard()
    {
        // ...
    }
}
```


23.7.3. TEMPLATES TWIG

```
{% if is_granted('ROLE_ADMIN') %}  
<a href="...">Delete</a>  
{% endif %}
```

23.8. ROLE_HIERARCHY

La section `role_hierarchy` permet de définir une hiérarchie des rôles (`security.yaml`).

```
security:
  role_hierarchy:
    # Un admin hérite des droits d'auteur et de modérateur
    ROLE_ADMIN: [ROLE_AUTEUR, ROLE_MODERATEUR]
    # On garde ce rôle superadmin, il nous ressortira par la suite
    ROLE_SUPER_ADMIN: [ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]
```

<https://openclassrooms.com/fr/courses/3619856-developpez-votre-site-web-avec-le-framework-symfony/3624755-securite-et-gestion-des-utilisateurs>

23.9. VERIFIER VOTRE CONFIGURATION DE LA SECURITE

```
php bin/console debug:config security
```

23.10. ACCEDER AU USER CONNECTE :

23.10.1. DEPUIS LE CONTRÔLEUR

:\\Program Files (x86)\\Ampmps\\www\\symfony-test\\src\\Security\\
LoginFormAuthenticatorAuthenticator.php

```
<?php

// On récupère le service
$security = $this->container->get('security.token_storage');

// On récupère le token
$token = $security->getToken();

// Si la requête courante n'est pas derrière un pare-feu, $token est
null

// Sinon, on récupère l'utilisateur
$user = $token->getUser();

// Si l'utilisateur courant est anonyme, $user vaut « anon. »

// Sinon, c'est une instance de notre entité User, on peut l'utiliser
normalement
$user->getUsername();
```

23.10.2. DEPUIS UN TWIG

- ➔ A l'aide de la variable globale {{ app }};
- ➔ on retrouve l'utilisateur courant {{ app.user }}:

```
Bonjour {{ app.user.username }} - {{ app.user.email }}
```

24. TP CABINET MEDICAL, PARTIE 7 A RENDRE SUR LE SERVEUR PARTAGE

1 - Créer la gestion des users (suppression, modification, modification du rôle, ajout de rôle, la création existe déjà par « register ») dans un sous-domaine /admin

2 - Limiter l'accès à cette partie à un unique utilisateur de ROLE_ADMIN

<https://www.youtube.com/watch?v=H-PdDqgfFwg> à 21.24

3 - Ajouter un écran qui liste les consultations pour un médecin

4 - Puis, dans un second temps, limiter la liste à un mois et faire apparaître les consultations sur un calendrier mensuel.

25. WEBOGRAPHIE

<https://blog.dev-web.io/2018/10/30/symfony-4-gestion-utilisateurs-sans-fosuserbundle-v2018-chapitre-1/>

<https://numa-bord.com/miniblog/symfony-4-les-base-dune-gestion-des-utilisateurs-inscription-connexion-droits-dacces/>

tuto trop general :

<https://www.supinfo.com/articles/single/7190-gestion-utilisateurs-sans-fosuserbundle-symfony-4>

et le source :

<https://gitlab.com/Checkspear/ManageUsersWithoutFOSUserBundleSF4>

Doc officielle :

<https://symfony.com/doc/current/security.html>

Symfony 4 - make:user & make:auth - Système d'authentification automatique

https://www.youtube.com/watch?v=3oS0I_hmnmA

Gérer la sécurité et les users avec Symfony 4 (Merci la Doc officielle !)

<https://www.youtube.com/watch?v=H-PdDqgfFwg>

https://symfony.com/doc/current/doctrine/registration_form.html

générer l'authentification

https://symfony.com/doc/current/security/form_login_setup.html

graficart : le composant Security

<https://www.youtube.com/watch?v=5LfSTeyvyuM>

Symfony 4 : pour comprendre l'authentification

<https://www.youtube.com/watch?v=GjHWa9hQic>

[Lior CHAMLA](#)

certification symfony :

https://www.youtube.com/watch?v=DT_hJSgDrwE

26. TWIG, LES FONDAMENTAUX

- TWIG moteur de Template, permet de générer les vues
- indépendant mais intégré Symfony.
- Sa syntaxe provient des templates de Jinja and Django
- TWIG acronyme de ????

26.1. APPEL DU SERVICE, ET PASSAGE DES VARIABLES

```
public function show(Consultation $consultation): Response
{
    return $this->render('consultation/show.html.twig', [
        'consultation' => $consultation,
    ]);
}
```

Passage des variables

Appel du template Twig

```
public function index(ConsultationRepository $consultationRepository):
Response
{
    return $this->render('consultation/index.html.twig', [
        'consultations' => $consultationRepository->findAll(),
    ]);
}
```

```
{% extends 'base.html.twig' %}

{% block title %}Consultation{% endblock %}

{% block body %}
    <h1>Consultation</h1>

    <table class="table">
        <tbody>
            <tr>
                <th>Id</th>
                <td>{{ consultation.id }}</td>
            </tr>
            <tr>
                <th>DateHeure</th>
                <td>{{ consultation.dateHeure ?
consultation.dateHeure|date('Y-m-d H:i:s') : '' }}</td>
            </tr>
        </tbody>
    </table>

    <a href="{{ path('consultation_index') }}">back to list</a>

    <a href="{{ path('consultation_edit', {'id': consultation.id})
}}">edit</a>

    {{ include('consultation/_delete_form.html.twig') }}
{% endblock %}
```


26.2. TROIS TYPES DE SYNTAXES

`{{ ... }}`

Equivalent à un “echo”: affiche une variable ou le résultat d’une expression

`{% ... %}`

“Faire quelque chose”: ce tag annonce une expression logique à exécuter comme une boucle for par exemple.

`{# ... #}`

Met en commentaire, c’est l’équivalent PHP de `/* comment */`

Twig permet d’ajouter des filtres qui modifieront le contenu permettant son affichage.

26.3. SYNTAXE DE BASE D’AFFICHAGE, LES DOUBLES ACCOLADES « {{ ... }} ».

Voici quelques exemples dont la syntaxe sera détaillée par la suite :

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 date soit un objet de type Datetime ici.	Date : {{ date date('d/m/Y') }}	Date : <?php echo \$date->format('d/m/Y'); ?>
Concaténer	Identité : {{ nom ~ " " ~ prenom }}	Identité : <?php echo \$nom.' '.\$prenom; ?>

26.4. SYNTAXE LOGIQUE DE BASE : { { OBJET.ATTRIBUT } }

TWIG interroge la nature de "objet" (tableau ou objet ?), suivant ce qu'elle trouve :

- Elle vérifie si objet est un tableau, et si attribut est un index valide. Si c'est le cas, elle affiche **objet['attribut']**.
- Sinon, et si objet est un objet, elle vérifie si **attribut** est un attribut valide (public donc). Si c'est le cas, elle affiche **objet->attribut**.
- Sinon, et si objet est un objet, elle vérifie si **attribut()** est une méthode valide (publique donc). Si c'est le cas, elle affiche **objet->attribut()**.
- Sinon, et si objet est un objet, elle vérifie si **getAttribut()** est une méthode valide. Si c'est le cas, elle affiche **objet->getAttribut()**.
- Sinon, et si objet est un objet, elle vérifie si **isAttribut()** est une méthode valide. Si c'est le cas, elle affiche **objet->isAttribut()**.
- Sinon, elle n'affiche rien et retourne **null**.

26.5. FILTRES NATIFS

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

26.6. STRUCTURES DE CONTRÔLE ET EXPRESSIONS

Condition : {% if %}

```
{% if membre.age < 12 %}  
    Il faut avoir 12 ans pour ce film.  
{% elseif membre.age < 18 %}  
    OK bon film.  
{% else %}  
    Un peu vieux pour voir ce film non ?  
{% endif %}
```

Boucle : {% for %}

```
<ul>
  {% for membre in liste_membres %}
    <li>{{ membre.pseudo }}</li>
  {% else %}
    <li>Pas d'utilisateur trouvé.</li>
  {% endfor %}
</ul>
```

Boucle : {% for %} en accédant à la clef

```
<select>
  {% for valeur, option in liste_options %}
    <option value="{{ valeur }}">{{ option }}</option>
  {% endfor %}
</select>
```

Une structure {% for %} définit une variable {{ loop }} au sein de la boucle, dont voici les attributs :

Variable	Description
{{ loop.index }}	Le numéro de l'itération courante (en commençant par 1).
{{ loop.index0 }}	Le numéro de l'itération courante (en commençant par 0).
{{ loop.revindex }}	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 1).
{{ loop.revindex0 }}	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 0).
{{ loop.first }}	true si c'est la première itération, false sinon.
{{ loop.last }}	true si c'est la dernière itération, false sinon.
{{ loop.length }}	Le nombre total d'itérations dans la boucle.

Définition : {% set %}

```
{% set foo = 'bar' %}
```

26.7. HÉRITER ET INCLURE DES TEMPLATES

- un Template père appelle des Templates enfants
- un Template fils peut modifier son Template père

Le layout (le Template père)

- Il définit la structure globale (pattern)
- Il est stocké dans Resources/views/layout.html.twig
- Il est composé de « block » : c'est le template fils qui remplira ces « blocks »

```
{# src/OC/PlatformBundle/Resources/views/layout.html.twig #}

<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>{% block title %}OC Plateforme{% endblock %}</title>
  </head>
  <body>
    {% block body %}
    {% endblock %}
  </body>
</html>
```

Voir le Template fils ci-dessous

Exemple de Template fils :

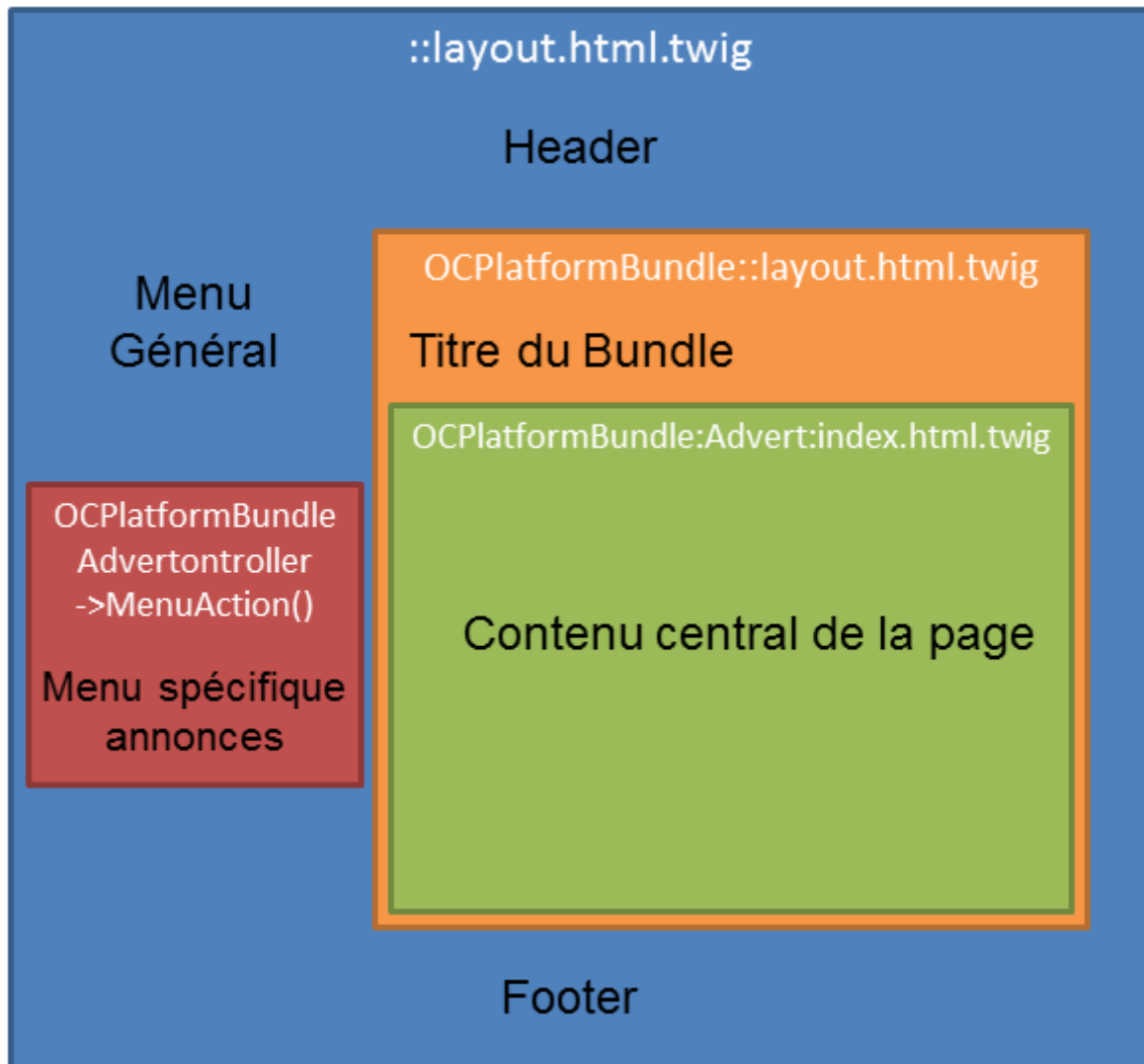
src/OC/PlatformBundle/Resources/views/Advert/index.html.twig:

```
{# src/OC/PlatformBundle/Resources/views/Advert/index.html.twig #}

{% extends "OCPlatformBundle::layout.html.twig" %}
{% block title %}{{ parent() }} - Index{% endblock %}
{% block body %}

    Notre plateforme est un peu vide pour le moment, mais cela
    viendra !

{% endblock %}
```



- ➔ Layout général :
 - c'est le design de votre site, indépendamment de vos bundles.
 - contient le header, le footer, etc. La structure de votre site donc.
- ➔ Layout du bundle : il hérite du layout général et contient les parties communes à toutes les pages d'un même bundle.
- ➔ Template de page : il hérite du layout du bundle et contient le contenu central de votre page.
- ➔ Le menu gauche (pavé en rouge) n'est pas un Template TWING mais il est généré par une action de contrôleur ! Il ne fait pas partie du modèle triple héritage. Nous l'étudierons par la suite car il permet de construire par exemple des renouvellements partiels de la page en AJAX.
- ➔ Le Layout général n'est pas l'arborescence du Bundle, mais dans : **`app/Resources/views/layout.html.twig`**.
Pour l'appeler depuis vos templates, la syntaxe est la suivante : « `::layout.html.twig` » .

27. AUTOMATISER LA CONSTRUCTION D'UN FORMULAIRE : MAKE:FORM

27.1. MAKE:FORM, GÉNÉRATION D'UN ENTITY TYPE

```
php bin/console make:form
```

The name of the form class (e.g. TinyKangarooType):

> PatientType

The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):

> Patient

created: src/Form/PatientType.php

Success!

Next: Add fields to your form and start using it.

Find the documentation at <https://symfony.com/doc/current/forms.html>

27.2. ENTITYTYPE

La commande `make :form` produit un `EntityType`, constructeur de formulaire qu'il faut compléter des champs

```
<?php

namespace App\Form;

use App\Entity\Patient;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class PatientType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('NumSS')
            ->add('nom')
            ->add('prenom')
            ->add('dateNaissance')
            ->add('sexe')
            ->add('email')
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Patient::class,
        ]);
    }
}
```

Si cette nouvelle génération a eu lieu après l'adjonction d'un nouvel email, il apparait désormais sur le formulaire de création :

Create new Patient

Num s s

Nom

Prenom

Date naissance

Sexe

Email

Save

[back to list](#)

Nouveau champ

28. CONTROLE DES FORMULAIRE AVEC VALIDATOR,

28.1. QUELQUES EXEMPLES DE CONTROLE DE FORMULAIRE

La date doit être une date valide ;

Le sujet du message doit faire au moins 3 caractères de long ;

Le contenu ne doit pas être vide ;

Le destinataire doit exister dans la base ;

28.2. STRATEGIE DE VALIDATION (THEORIE DE LA VALIDATION)

1. Les règles de validation seront rattachées à une classe, décrites et encodées dans la classe.
2. Un service extérieur (validator) viendra lire sur l'objet ses règles de validation et en déduira si elles sont respectées par les valeurs saisies ou manquantes.

28.3. PLUSIEURS MODALITES SONT POSSIBLES

Le composant Validator permet de générer des contrôles. Plusieurs modalités sont possibles :

- A partir des annotations
- A l'aide du fichier de configuration
`yaml : config/packages/framework.yaml`

Le plus souvent, ces deux moyens sont équivalents.

Pour les contrôles spécifiques, on emploie un "callback" annoncé par les annotations

28.4. INSTALLATION DE VALIDATOR, CONFIGURATION¶

Compléter votre installation par le composant Validator et ses annotations :

```
composer require symfony/validator doctrine/annotations
```

Assurez-vous que le composant Validator est activé dans le fichier de configuration principal:
`config/packages/framework.yaml`

```
framework:
    validation: { enabled: true }
```

Comme nous allons utiliser nos contraintes à l'aide des annotations, modifier cette configuration comme suit :

```
framework:
    validation: { enable_annotations: true }
```

28.5. ANNOTATION @ASSERT, EXEMPLE

Exemple, complétons notre entité patient :

- ➔ Les contraintes sont annoncées par le mot `@Assert`. L'alias doit être annoncé en préambule par un « use »

```
<?php
use Symfony\Component\Validator\Constraints as Assert;
```

- ➔ `@Assert` introduit une contrainte sur la propriété

```
/**
 * @var string
 *
 * @Assert\NotBlank(message="Le nom doit être renseigné")
 * @Assert\Length(
 *     min = 3,
 *     max =30,
 *     minMessage ="Le nom du patient doit contenir au moins {{ limit }} caracteres.",
 *     maxMessage ="Le nom du patient doit contenir au plus {{ limit }} caracteres."
 *)
 * @ORM\Column(name="nomPatient", type="string", length=255)
 */
private $nomPatient;
```

28.6. SYNTAXE

→ Forme

simple

```
@Assert\Contrainte(valeur de l'option par défaut)
```

→ Forme étendue

```
@Assert\Contrainte(option1="valeur1", option2="valeur2", ...)
```

Exemples

```
@Assert\NotBlank(message="Le nom doit être renseigné")  
  
@Assert\MinLength(limit=3,message ="Le sujet doit contenir  
au moins {{limit}} characters.")
```

28.7. CONTRAINTES DE BASE

Contrainte	Rôle	Options
NotBlank Blank	La contrainte NotBlank vérifie que la valeur soumise n'est ni une chaîne de caractères vide, ni NULL. La contrainte Blank fait l'inverse.	
Type	La contrainte Type vérifie que la valeur est bien du type donné en argument.	type (option par défaut) : le type duquel doit être la valeur, parmi array, bool, int, object, etc .

28.8. CONTRAINTES SUR LES DATES

Contrainte	Rôle
Date	La contrainte Date vérifie que la valeur est un objet de type Datetime, ou une chaîne de caractères du type YYYY-MM-DD.
Time	La contrainte Time vérifie que la valeur est un objet de type Datetime, ou une chaîne de caractères du type HH:MM:SS.
DateTime	La contrainte Datetime vérifie que la valeur est un objet de type Datetime, ou une chaîne de caractères du type YYYY-MM-DD HH:MM:SS.

28.9. DOCUMENTATION OFFICIELLE ET WEBOGRAPHIE

Visualiser la documentation officielle :

<http://symfony.com/doc/current/reference/constraints.html>

Un tuto agréable :

<https://symfonycasts.com/screencast/symfony-forms>

28.10. DECLENCHER LE CONTROLE DU FORMULAIRE : \$FORM->ISVALID()

L'ordre "form->isValid()" déclenche la validation du formulaire. Il est placé dans le contrôleur lors de la réception des données (" \$request->isMethod('POST') ")

```
<?php
if ($request->isMethod('POST')) {
    // L'ordre ci-dessous place les données du formulaire dans l'entité
    $message
    $form->handleRequest($request);

    // Si le formulaire est valide (les valeurs entrées sont correctes)
    if ($form->isValid()) {
        // On persiste $message
        $em = $this->getDoctrine()->getManager();
        $em->persist($message);
        $em->flush();
    }
}
```

29. TP CABINET MEDICAL, PARTIE 8 A RENDRE SUR LE SERVEUR PARTAGE

- 1 – Ajouter une propriété email sur l'entité patient (si cela n'a pas été déjà fait)
- 2 – Imposer sur son formulaire la contrainte « email »

30. RESUME DU TP :

1. `composer create-project symfony/website-skeleton symfony-test`
2. `composer require symfony/orm-pack` (sous `symfonyTraining`)
3. `composer require symfony/maker-bundle --dev`
4. `php bin/console list make`
5. si message d'erreur : `composer require doctrine maker`
6. Configurer la connexion à la base de données (la ligne prévoit une base du nom de « symfony » à modifier si nécessaire.
`DATABASE_URL=mysql://root:mysql@127.0.0.1:3306/symfony`
ou
`DATABASE_URL=mysql://root:@127.0.0.1:3306/symfony`
7. `php bin/console doctrine:database:create`
8. `php bin/console make:entity`
 - a. Patient (NumSS, nom, prenom, dateNaissance, sexe)
9. `php bin/console make:migration`
10. `php bin/console doctrine:migrations:migrate`
11. `php bin/console make:controller PatientController`
12. `composer require symfony/apache-pack` (pour le .htaccess)
`make :entity` sur les autres entités
 - a. Medecin (matricule, nom, prenom)
 - b. Consultation (dateHeure)
 - c. Ordonnance (numeroOrdre, dateHeure)
 - d. Medicament (denomination, conditionnement)
 - e. LignePrescription (posologie)
13. Ajouter à toutes ces entités les relations ManyToOne, etc...
14. `php bin/console doctrine:schema:validate`
15. `php bin/console make:entity --regenerate` (regeneration des accesseurs)
16. Régénérer la base de données (pour obtenir toutes les tables sur la BdD MySQL)
 - a. `php bin/console make:migration`
 - b. `php bin/console doctrine:migrations:migrate`
17. Compléter le contrôleur du patient pour faire un new patient
18. Construire un TWIG pour lister les patients
19. `php bin/console make:crud`
 - a. Medecin
 - b. Consultation
 - c. Medicament
 - d. LignePrescription
 - e. Ordonnance
20. `php bin/console debug :router` (teste des routes)
21. Créer un TWIG `index.html.twig` pour la page d'accueil avec un lien vers chacune des entités :
 - a. Consultations
 - b. Médecins
 - c. Patients
 - d. Médicaments
22. Créer la route vers chacune des listes d'entités dans le `config/router.yaml`
23. Créer un formulaire spécifique à la création d'un patient, ajouter une propriété email au patient, suite à cet ajout, ajuster les pages de visualisations et de créations du patient
24. Installer le security-bundle : `composer require symfony/security-bundle`

25. Créez la classe Users : `php bin/console make :user`, ajouter un champ username (avec ou sans `make :entity` ?)
26. Maj de la BdD : `php bin/console make:migration`, `php bin/console doctrine:migrations:migrate`
27. composer require orm-fixtures --dev (non a priori pas la peine)
28. `php bin/console make:fixtures` (non a priori pas la peine)
29. Si le formulaire n'a pas de champs 'username', le rajouter
30. Créer plusieurs users
31. Générer le formulaire de connexion : `Make :auth`
32. Finaliser la redirection "TODO" in the `src\Security\AppUserAuthenticator::onAuthenticationSuccess()` method.
par :

`return new RedirectResponse($this->urlGenerator->generate('index'));`

décommenter aussi dans le fichier `routes.yaml` la route 'index'

Créer le `DefaultController.php` avec l'action `index` s'il n'existe pas
33. Ajout d'un formulaire d'enregistrement : `php bin/console make:registration-form`
(A priori crée le retour vers l'authentification après l'enregistrement s'il a bien été généré avant par `make :auth`)
34. Maj de la BdD : `php bin/console make:migration`,
`php bin/console doctrine:migrations:migrate`
35. Vérifier l'ensemble des routes générées `php bin/console debug:router` (cf c-dessous)
36. Activer la déconnexion (logout) dans le fichier de configuration de la sécurité
`logout` : `path`, `target`
et dans le fichier `routes.yaml`
<https://symfony.com/doc/current/security.html#logging-out>
37. Vérifier la configuration de sécurité :
`php bin/console config:dump-reference security`
Voir sa propre configuration de sécurité :
`php bin/console debug:config security`
38. Améliorer la barre de navigation en ajoutant le username, et un bouton de connexion/déconnexion
39. Créer un domaine réservé à l'administrateur pour la gestion des médecins.
40. Ajouter une propriété email au Patient (`make :entity`)
41. Régénérer un formbuilder et un PatientType (`make :form`)
42. Ajouter un contrôle d'email sur cette nouvelle propriété (`@Assert email`)

31. COMMANDE CITEES DANS CE COURS :

php bin/console debug:router

32. UPLOAD DE FICHIER

https://symfony.com/doc/current/controller/upload_file.html

https://www.youtube.com/watch?v=jBeeV2_lmbw

33. ERREURS REPERTORIEES A L'INSTALLATION

33.1. CERTIFICATE PROBLEM :

A l'installation, il peut apparaître l'erreur suivante :

```
1 [GuzzleHttp\Exception\RequestException]
2   cURL error 60: SSL certificate problem: unable to get local issuer certificate
3
4
5 [GuzzleHttp\Ring\Exception\RingException]
6   cURL error 60: SSL certificate problem: unable to get local issuer certificate
7
8
9 new <directory> [<version>]
```

1) Copiez sur votre disque, le certificat à l'adresse :

<https://curl.haxx.se/ca/cacert.pem>

2) Indiquez son adresse locale dans le php.ini (dans celui actif)

```
1 [curl]
2 ; A default value for the CURLOPT_CAINFO option. This is required to be an
3 ; absolute path.
4 curl.cainfo = "C:/wamp64/bin/php/cacert.pem"
```

3) Redémarrez les services AMPPS ou WAMP ou MAMP

33.2. DATE.TIMEZONE

Si besoin

- 1) Editer le PHP.ini
- 2) Ajuster le date.timezone comme suit :
[Date]

; Defines the default timezone used by the date functions

; http://php.net/date.timezone

date.timezone = Europe/Paris

- 3) Redémarrer le serveur.

