# Symfony 5

Christophe DUFOUR

# Introduction

- Framework Back PHP Open Source
- Développé par SensioLabs (Fabien Potencier)
- Première version: octobre 2005
- Orienté objet depuis Symfony 2
- Basé sur PHP version 7 depuis la depuis Symfony 4
- Symfony 5 sortie en novembre 2019 (suit la 4.4)
- Site officiel: https://symfony.com/
- Autres framework PHP: CodeIgniter, PHPCake, Yii, Zend, Laravel
- Plus bas niveau qu'un CMS mais plus haut niveau que PHP brut

# Caractéristiques principales

- Architecture MVC
- Modulaire. Composants utilisables en standalone (exemple: HttpFOundation utlisé par le CMS Drupal)
- Approche CaaC via des fichiers de configuration yaml
- Système de routage
- Moteur de template riche (TWIG par défaut)
- Approche par instanciation de services
- ORM doctrine pour l’accès aux données (bases relationnelles)
- Console intégrée (CLI) pour l’automatisation de tâches

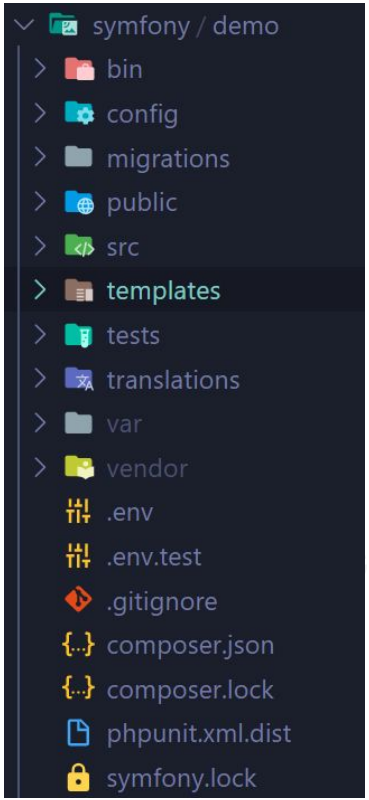
# Installation

- PHP >= 7.2
- Modules PHP suivants: Ctype, iconv, JSON, PCRE, Session, SimpleXML, Tokenizer
- Composer : https://getcomposer.org/download/
- Symfony CLI : https://symfony.com/download

# Environnement du prof

- Machine virtuelle (VirtualBox), 2 vCPU, 2 GO RAM
- Ubuntu Server 20.04 (focal)
- Editeur: vscode
- Connexion SSH (via extension Remote - SSH) à la VM

# Création d'un nouveau projet




```
symfony new projectName --version="4.4" --full
```

| | |
|---|---|
| **bin** | Console: **php bin/console [commande]** |
| **config** | Essentiellement, des fichiers de configurations .yaml |
| **migrations** | Fichiers .php contenant des instructions sql visant à synchroniser les base de données. Générés automatiquement par la commande migrate |
| **public** | Fichiers statiques (css, js, images, etc.). Contient **index.php**, le "front controller" |
| **src** | Fichiers .php sources de l'application (contrôleurs, entités, services, classes diverses) |
| **templates** | Fichiers de template (vue) au format .twig |
| **tests** | Fichiers .php de tests unitaires |
| **var** | Fichiers de cache et de log; non versionnés |
| **vendor** | Dépendances de l'application (cf composer.json), non versionnées |

# Démarrer le serveur de développement

- Au moins deux possiblités
- Depuis le dossier projet
  - php -S localhost:8000 -t ./public
  - symfony serve [-d]
- Quelques commandes utiles
  - symfony server:status
  - symfony server:log
  - symfony server:list
  - symfony server:stop|start

# Routage

Symfony fournit **deux** options principales non exclusives pour le routage

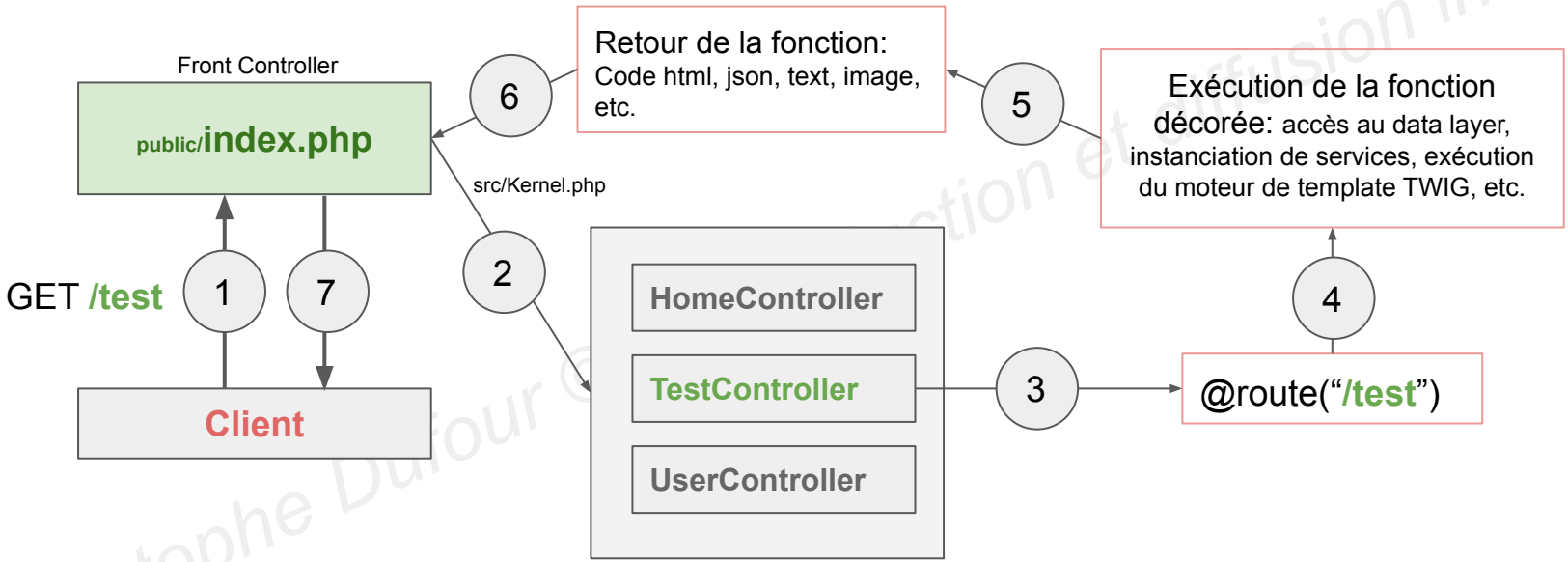
L'annotation **@Route()** pour décorer une méthode de contrôleur

```
/**
 * @Route("/index", name="index")
 */
public function index(): Response { }
```

Des fichiers de configuration dans le dossier config. Par exemple: **routes.yaml**

```
index:
   path: /index
   controller:
App\Controller\DefaultController::index
```

# Cycle requête-réponse




A partir du nom de la ressource requise, SF parcourt l'ensemble des contrôleurs à la recherche d'un décorateur de fonction correspondant à l'URL requise. S'il la trouve, la fonction décorée est exécutée, et le retour de cette fonction est ensuite renvoyé au client.

# Cycle requête-réponse

```php
namespace App\Controller;

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

class TestController extends AbstractController
{
    /**
     * @Route("/test", name="test")
     */
    public function index()
    {
        $res = new Response("Réponse de TestController.index()");
        return $res;
    }
}
```

src/Controller/TestController.php

GET /test

# Request

https://symfony.com/doc/current/introduction/http_fundamentals.html

```php
$request = Request::createFromGlobals();
$request
    ->headers->get("content-type")
    ->query->get("param1")
    ->getMethod()
    ->getContent()
```

# Response

https://symfony.com/doc/current/introduction/http_fundamentals.html

```php
$response = new Response();
$response
    ->header->set("X-Token", "abc123")
    ->request->get("param1")
    ->setStatusCode(Response::HTTP_NOT_FOUND)
    ->setContent("<p>Coucou</p>");
```

# Contrôleur

- Classe PHP héritant de la classe Symfony\Bundle\FrameworkBundle\Controller**AbstractController**
- Namespace **App\Controller**
- Placé dans le dossier **src/Controller**
- Définit des méthodes publiques de routage, via l'annotation **Route()** ou via le fichier **config/routes.yaml**
  - doivent retourner un objet **Response**
  - peuvent prendre en entrée un object **Request**
- Dispose d'un accès:
  - À la couche Vue (service Twig): $this->render()
  - A la couche Model: $this->getDoctrine()
- php bin/console make:controller [ControllerName] [--no-template]

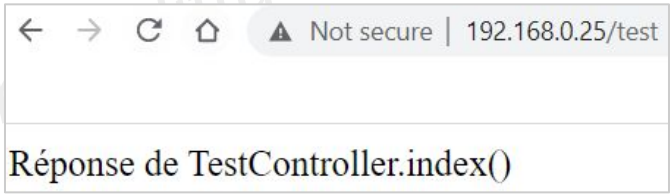
# Les annotations (décorateurs)

- Les annotations SF se basent sur le modèle de conception **Decorator**
- Permettent de fournir des métadonnées à une méthode ou une propriété afin d'en modifier le comportement ou de la configurer
- Syntaxe: **/** @AnnotationName(param1, param2=, ...) */**
- Une annotation est une fonction prenant d'éventuels paramètres en entrée
- Doivent être importées dans la classe souhaitant s'en servir
- Exemples:
  - **@Route("/test", name="test')** public function index() {...} // décorateur de méthode
  - **@ORM\Column(type="string", length=255)** private $name; // décorateur de propriété

# Les annotations (décorateurs)

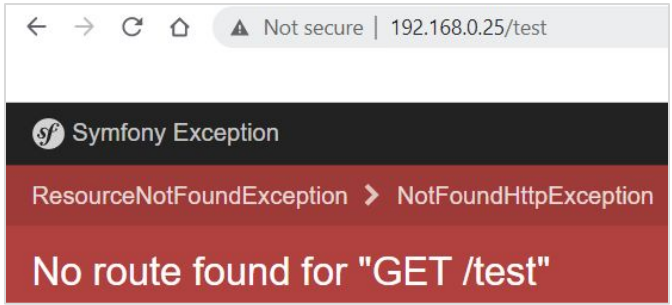
```php
class TestController extends AbstractController
{
    /** @Route("/test", name="test") */
    public function index()
    {
        return new Response("Réponse de TestController.index()");
    }
}
```

Avec annotation




```php
class TestController extends AbstractController
{
    public function index()
    {
        return new Response("Réponse de TestController.index()");
    }
}
```

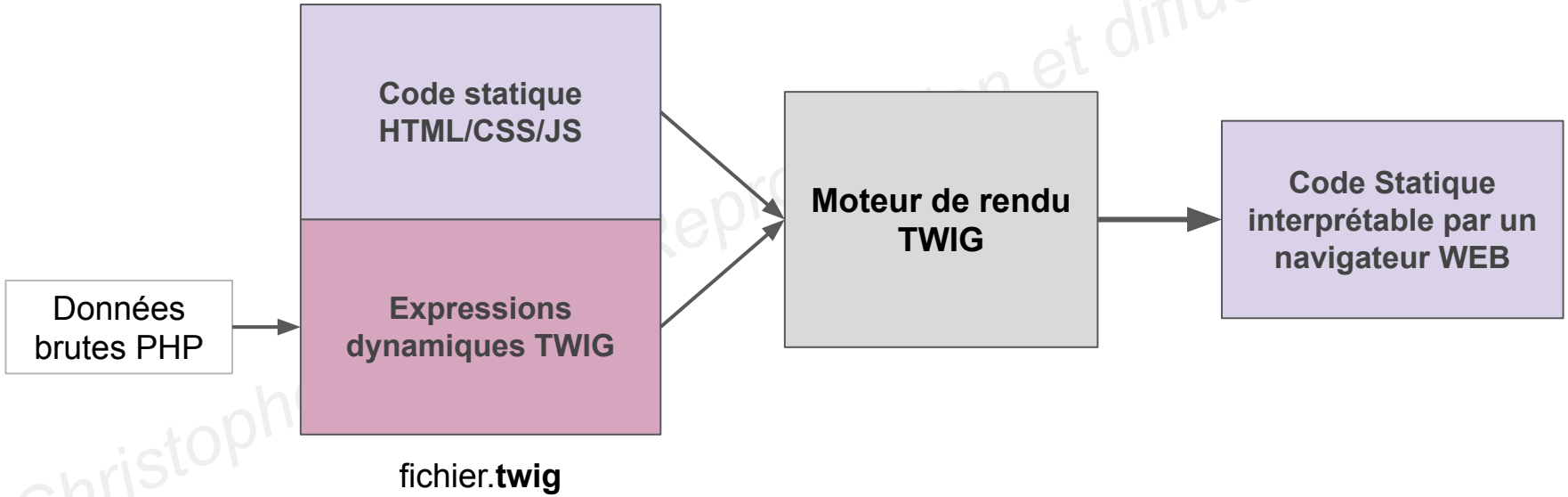
Sans annotation



# Twig: introduction

- Moteur de rendu (Template Engine) pour PHP
- Expressions plus lisibles qu'en PHP pur
- Développé par SensioLabs (Fabien Potencier)
- Utilisé nativement par Symfony
- Site officiel: https://twig.symfony.com/
- Fonctionnalités majeures:
  - Création de variables de template
  - Structures conditionnelles et itératives
  - Héritage multiple et extension
  - Filtres
  - Concaténation

# Moteur Twig: schéma



# Twig: itération

```php
$items = array(1,2,3);
```

**Syntaxe Twig**

```twig
{% for item in items %}
    <p>Paragraphe</p>
{% endfor %}
```

**Syntaxes équivalentes PHP**

```php
<?php
    foreach($items as $item) {
        echo "<p>Paragraphe</p>";
    }
?>

<?php foreach ($items as $item): ?>
    <p>Paragraphe</p>
<?php endforeach;?>
```

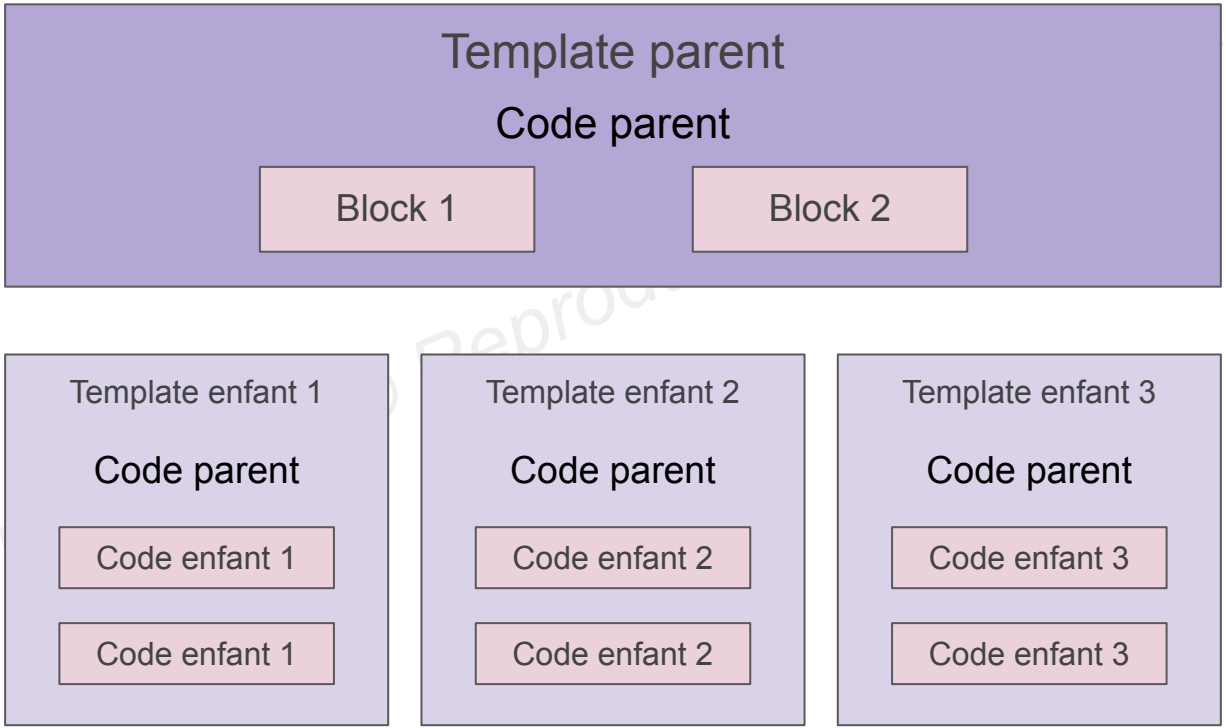
**Rendu HTML**

```html
<p>Paragraphe</p>
<p>Paragraphe</p>
<p>Paragraphe</p>
```

# Expressions Twig

| | |
|---|---|
| {{ }} | Accès en **lecture** à un élément de type chaîne de caractères ou convertible en chaîne de caractère. Exemple: **{{** name **}}** ou $name = “Chris” |
| {% %} | Bloc **d’instructions**. Exemple: **{%** if condition **%} {%** endif **%}** |
| {# #} | Bloc de **commentaire**. Exemple: **{#** commentaire Twig **#}** |

# HéritageTwig



# Entité

- Classe PHP de modélisation
- Située dans le dossier **src/Entity**
- Namespace **App\Entity**
- Classe et attributs décorés par l'annotation **@ORM**xxx
- Méthodes **getters** et **setters** d'accès aux propriétés privées
- Peut être créée et modifiée manuellement ou via la console:
  - php bin/console make:entity
- Doctrine fait la correspondance (mapping) entre la classe Entité à la structure équivalente en base de donnée

# Création de la base de données (mysql)

- Vérifier le que driver **php-mysql** est installé
  - Installation sur debian/ubuntu: sudo apt install php-mysql
- Configurer l'accès au serveur de base de données
  - **DATABASE_URL** dans le fichier **.env** ou **.env.local**
- Exécuter la commande de création
  - php bin/console doctrine:database:create

# Doctrine (Base de données)

- ORM (Object Relational Mapping) Open Source pour PHP
- Première version: avril 2006
- Doctrine 2.0 sorti en 2010
- Utilisée par Symfony depuis sa version 1.3
- Optionnel mais recommandé
- Dispose de son propre langage de requête: **DQL** (Doctrine Query Language)
- Possibilité d'exécuter des requêtes en SQL natif
- composer require symfony/orm-pack

# Créer/modifier une entité

```
chris@aramis:~/symfony/test$ php bin/console make:entity

 Class name of the entity to create or update (e.g. OrangeGnome):
 > Student

 created: src/Entity/Student.php
 created: src/Repository/StudentRepository.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):
 > name

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

 Field length [255]:
 >

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Student.php

 Add another property? Enter the property name (or press <return> to stop adding fields):
 >
```

**make:entity** Student

src/Entity/
**Student.php**

src/Repository/
**StudentRepository.php**

*Student.php* contient les propriétés définies par les saisies Console.
Les Getters et Setters pour ces propriétés sont générés automatiquement.

*StudentRepository.php* est la classe exposant les différentes méthodes de recherche en base (findAll, findBy, etc.)

# Migrations

La création/modification d'une entité est généralement suivie d'une "migration", via la commande php bin/console **make:migration**.

Un fichier de migration est alors généré et placé dans le dossier **/migrations**.

Ce fichier php contient les méthodes up et down permettant de refléter en base de données les changements apportées à l'entité.

L'exécution des ces méthodes se fait par la commande php bin/console **doctrine:migrations:migrate**

# Migrations

```
 Next: When you're ready, create a migration with php bin/console make:migration

chris@aramis:~/symfony/test$ php bin/console make:migration


  Success!


 Next: Review the new migration "migrations/Version20201121163133.php"
 Then: Run the migration with php bin/console doctrine:migrations:migrate
 See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
chris@aramis:~/symfony/test$ php bin/console doctrine:migrations:migrate

 WARNING! You are about to execute a database migration that could result in schema chan
ges and data loss. Are you sure you wish to continue? (yes/no) [yes]:
 > yes

[notice] Migrating up to DoctrineMigrations\Version20201121163133
[notice] finished in 107.5ms, used 18M memory, 1 migrations executed, 1 sql queries
```


doctrine:migrations:migrate

Exécute le code SQL

Version20201121163133.php

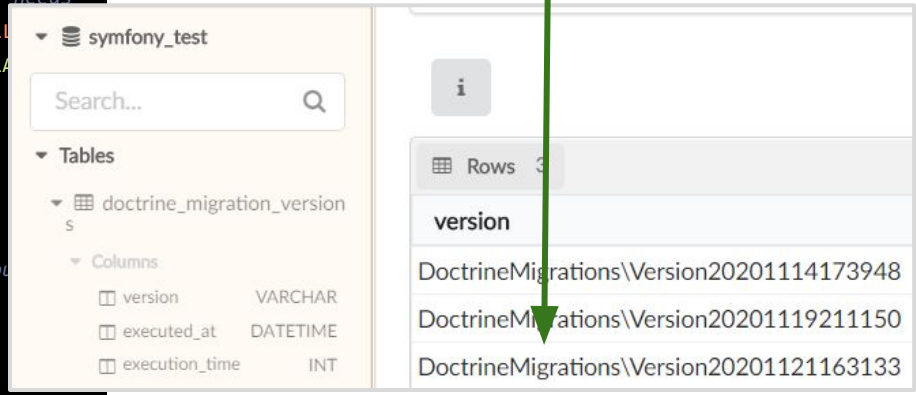
# Migrations

```php
final class Version20201121163133 extends AbstractMigration
{
    public function getDescri
    {
        return '';
    }

    public function up(Schema $schema) : void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE student (id INT AUTO_INCREMENT NOT NULL
VARCHAR(255) NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLA
`utf8mb4_unicode_ci` ENGINE = InnoDB');
    }

    public function down(Schema $schema) : void
    {
        // this down() migration is auto-generated, please modify it to you
needs
        $this->addSql('DROP TABLE student');
    }
}
```

/migrations/Version20201121163133.php

# Mapping

Classe d'entité

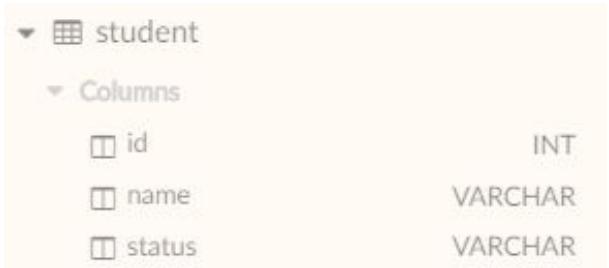
Attributs

```php
class Student
{
    /**
     * @ORM\Id @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;
    /** @ORM\Column(type="string", length=255) */
    private $name;
    /** @ORM\Column(type="string", length=255) */
    private $status;
}
```

**ORM Doctrine**

Table SQL

Colonnes

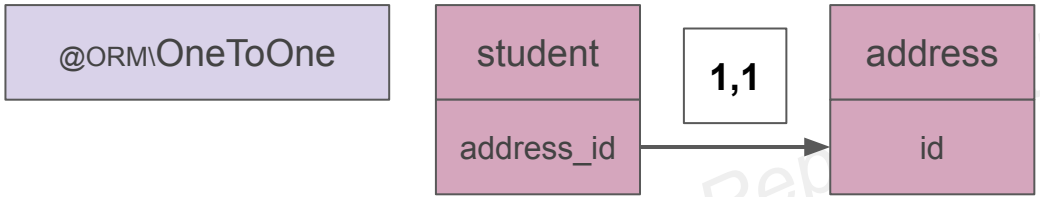


# Relation entre entités

| | | | | |
|---|---|---|---|---|
| @ORM\OneToOne | Student | 1,1 | Address | Un étudiant a une seule adresse. Une même adresse n'appartient qu'à un seul étudiant. *One student To One address* |
| @ORM\ManyToOne | Student | n,1 | Country | Un pays peut être celui de plusieurs étudiants. Un étudiant ne réside que dans un pays. *Many students To One country* |
| @ORM\OneToMany | Student | 1,n | Photo | Un étudiant peut avoir plusieurs photos. Une même photo n'appartient qu'à un seul étudiant. *One student To Many photos* |
| @ORM\ManyToMany | Student | n,n | Training | Un étudiant peut suivre plusieurs formations. Une formation peut être suivie par plusieurs étudiants. *Many students To Many trainings* |

# OneToOne

```php
/** @ORM\OneToOne(targetEntity=Address::class, cascade={"persist", "remove"}) */
private $address;
```




Un étudiant a une seule adresse. Une même adresse n'appartient qu'à un seul étudiant.
*One student To One address*

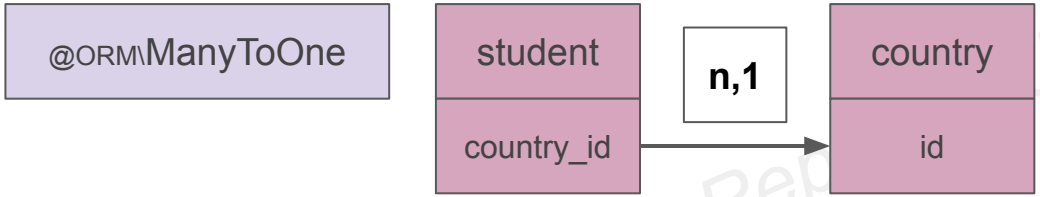
La migration ajoute une clé étrangère **address_id** dans la table **student** permettant d'établir une relation avec la clé primaire **id** de la table **address.**

Le paramètre cascade permet de persister un étudiant et son adresse en une déclaration. La suppression d'un étudiant entraîne la suppression "en cascade" de l'adresse associée.

```php
$student = new Student();
$student->setName("Chris");
$student->setStatus("prof");
$student->setAddress(
  new Address("rue du travail", "10 bis"));
$em = $this->getDoctrine()->getManager();
$em->persist($student);
$em->flush();
```

# ManyToOne

```php
/** @ORM\ManyToOne(targetEntity=Country::class) */
private $country;
```




La migration ajoute une clé étrangère **country_id** dans la table **student** permettant d'établir une relation avec la clé primaire **id** de la table **country.**

La suppression d'un étudiant **ne doit pas** entraîner la suppression de pays associé

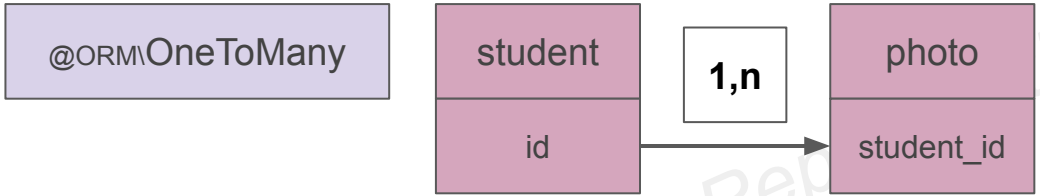
Un pays peut être celui de plusieurs étudiants.
Un étudiant ne réside que dans un pays.
*Many students To One country*

```php
public function getCountry(): ?Country
{
  return $this->country;
}

public function setCountry(?Country $country): self
{
  $this->country = $country;
  return $this;
}
```

# OneToMany

```php
/** @ORM\OneToMany(targetEntity=Photo::class, mappedBy="student") */
private $photos;
```




Un étudiant peut avoir plusieurs photos. Une même photo n'appartient qu'à un seul étudiant.
*One student To Many photos*

La migration ajoute une clé étrangère **student_id** dans la table **photo** permettant d'établir une relation avec la clé primaire **id** de la table **student.**

La propriété photos de la classe Student est une instance de la classe **ArrayCollection()**

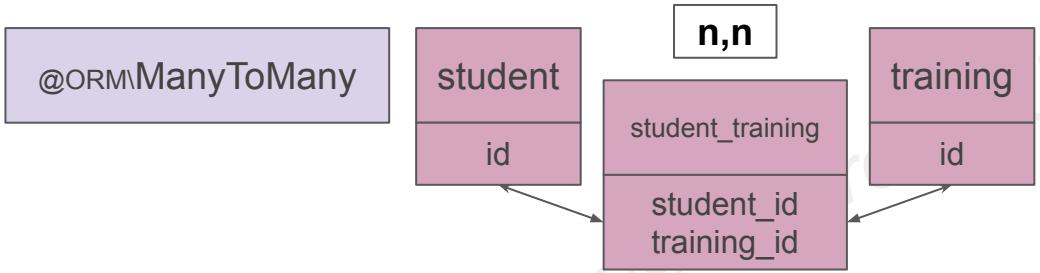
```php
public function __construct()
{
  $this->photos = new ArrayCollection();
}

/** @return Collection|Photo[]*/
public function getPhotos(): Collection {}

public function addPhoto(Photo $photo): self {}
public function removePhoto(Photo $photo): self {}
```

# ManyToMany (bidirectionnelle)

```php
/** @ORM\ManyToMany(targetEntity=Training::class, inversedBy="students") */
private $trainings;
```




Un étudiant peut suivre plusieurs formations.
Une formation peut être suivie par plusieurs étudiants. *Many students To Many trainings*

```php
/** @return Collection|Training[]*/
public function getTrainings(): Collection {}
public function addTraining(Training $training): self {}
public function removeTraining(Training $training): self {}
```

La migration crée une table de jointure **student_training** permettant de mettre en relation ses clés étrangères **student_id** et **training_id** avec les clés primaires **id** des tables **student** et **training**.

En ayant opté par une relation bidirectionnelle, la relation inverse est créée et la classe Training est mise à jour. Une propriété de type ArrayCollection **students** lui est ajoutée et permet d'obtenir les étudiants d'une formation.

```php
/** @ORM\ManyToMany(targetEntity=Student::class,
mappedBy="trainings") */
private $students;

/** @return Collection|Student[]*/
public function getStudents(): Collection {}
public function addStudent(Student $student): self {}
public function removeStudent(Student $student): self {}
```

# Repository et méthodes de recherche (finders)

- Une classe de **Repository** accompagne toujours une classe d'entité
- Exemple: StudentRepository.php ⇔ Student.php
- Le repository expose en standard les méthodes de recherche:
  - find(), findBy(), findOneBy(), findAll()
- Le repository est extensible: on peut y ajouter ses propres méthodes personnalisées de recherche
- Accès depuis un contrôleur:
  - $this->getDoctrine()->getRepository(EntityClassName::class)
- Accès depuis tout autre classe (Service, Commande, etc):
  - Instantiation de la classe (par Inj. de dép.) App\Repository\RepositoryClassName

# Exemple d'utilisation

**/src/Controller/DemoController.php**

```php
/**
* @Route("/demo22", name="student_list")
*/
public function demo22(Request $request)
{
    // Récupération des étudiants enregistrés en DB
    // Instantiation du Repository
    $repo = $this->getDoctrine()
         ->getRepository(Student::class);
    $students = $repo->findAll();

    // Transmission des données au template
    $res = $this->render("demo22.html.twig", [
        "students" => $students
    ]);
    return $res;
}
```

**/templates/demo22.html.twig**

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

{% block pageTitle %}demo 22{% endblock %}

{% block body %}
    <h3>Liste des étudiants</h3>
    <ul id="list">
        {% for student in students %}
            <li data-id="{{student.id}}">
                <span>{{ student.name }}</span>
            </li>
        {% endfor %}
    </ul>
{% endblock %}
```

# Exemples de recherche via les “finder” standard

```php
$repo = $this->getDoctrine()->getRepository(Student::class);
```

```php
// Recherche des tous les étudiants
$students = $repo->findAll();
```

```php
// Recherche l’étudiant d’id 5
$student = $repo->find(5);
```

```php
// Recherche du premier étudiant nommé Chris
$student = $repo->findOne(["name" => "Chris"]);
```

```php
/* Recherche des étudiants au statut élève,
ordonnés par nom ascendant (A vers Z) avec une
limite de 5 et un offset de 1 */
$students = $repo->findBy(
  ["status" => "élève"], // criteria
  ["name" => "ASC"], // orderby
  5, // limit
  1 // offset
);
```

# Le Query Builder de Doctrine

- Le QB est un objet permettant de “construire” des requêtes
- Il contient un ensemble de méthodes, chaînables, aux noms proches de la terminologie SQL:
  - select, from, delete, having, join, etc
- Instantiation: $qb = xxx**->createQueryBuilder();**
- Le QB renvoie des objets du type de l’entité recherchée
- S’utilise fréquemment dans les classes Repository
- https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/reference/query-builder.html

# Le Query Builder: exemple

/src/Repository/StudentRepository.php

```php
/**
* @return Student[] Returns an array of Student objects
*/
public function findByCountry($countryName)
{
    return $this->createQueryBuilder('s')
        ->join('s.country', 'c', 'country_id')
        ->where('c.name = :countryName')
        ->setParameter('countryName', $countryName)
        ->orderBy('s.name', 'ASC')
        ->getQuery()
        ->getResult()
    ;
}
```

La méthode renvoie les étudiants dont le pays correspond à celui passé en entrée.

Une jointure sur l'entité Country est réalisée grâce à la clé étrangère country_id

```
FOREIGN KEY (country_id) REFERENCES country (id)'
```

/src/Entity/Student.php

```php
/**
* @ORM\ManyToOne(targetEntity=Country::class)
*/
private $country;
```

# Doctrine Query Language (DQL)

- Langage de requête fourni par Doctrine
- Proche de la syntaxe SQL native
- Réalise le passage entre la création d'objets par instanciation des classes concernées et la structure SQL sous-jacente
- Ils s'agit d'interroger des classes plutôt que des tables
- Accessible à partir de l'EntityManager via la méthode **createQuery()**
- https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/reference/dql-doctrine-query-language.html

# DQL: exemple 1

```
$query = $em->createQuery(
  'SELECT s FROM App\Entity\Student s
   WHERE s.status = :status
   ORDER BY s.name ASC'
);
```

Sélectionne les étudiants dont le statut est fourni en paramètre triés dans l'ordre alphabétique par rapport à leur nom

```
$query->setParameter("status", "prof");
```

"Binding" du paramètre prévenant un risque d'injection sql

```
$students = $query->getResults();
```

Retourne un tableau d'objets Student

# DQL: autres exemples

```php
$query = $em->createQuery(
   'SELECT s FROM App\Entity\Student s
    LEFT JOIN s.training t
    WHERE t.title = :trainingTitle
    ORDER BY s.name ASC'
);
```

Sélectionne les étudiants ayant suivi la formation dont le nom est fourni en paramètre

/src/Repository/PhotoRepository.php

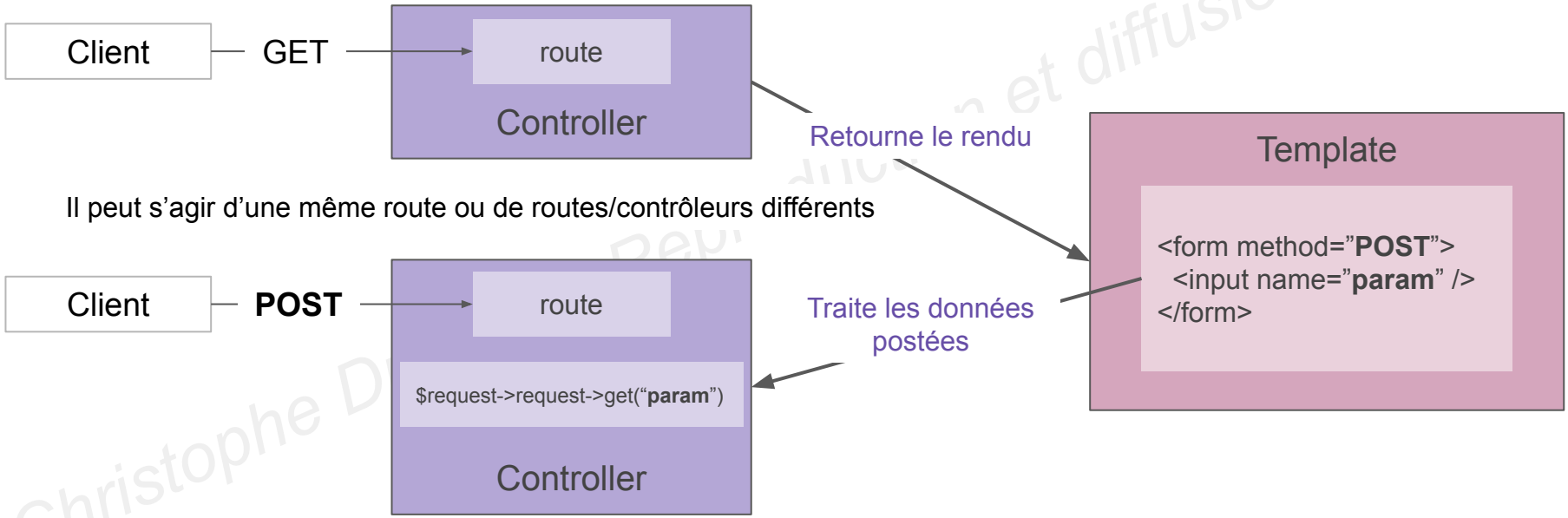
```php
public function findPNG()
{
  $em = $this->getEntityManager();
  $query = $em->createQuery(
    'SELECT p FROM App\Entity\Photo p
     WHERE p.filename LIKE :extension');
  $query->setParameter("extension", "%.png");
  return $query->getResult();
}
```

Retourne un tableau d'objets Photo dont la propriété filename se termine par la chaîne .png

# Les formulaires

- SF propose plusieurs manières de créer des formulaires
  - Orientée html/twig
  - Orientée php : createFormBuilder() ou createForm()
- Peuvent être gérés dans des classes spécifiques (src/Form/xxxType.php)
  - php bin/console create:form FormName
- Les formulaires orientés code offrent une meilleure réutilisabilité et maintenabilité
- Un contrôleur peut fournir à la vue le retour de **createView**
- La template dispose de helpers pour rendre le formulaire:
  - form, form_start, form_end, form_row, etc.

# Les formulaires



# createFormBuilder

```php
$training = new Training();
$form = $this->createFormBuilder($training)
  ->add("title", TextType::class, ["label" => "Intitulé"])
  ->add("level", TextType::class, ["label" => "Niveau"])
  ->add("duration", TextType::class, ["label" => "Nombre de jours"])
  ->add("save", SubmitType::class, ["label" => "Enregistrer"])
  ->getForm();

// Connexion entre le formulaire et la requête
$form->handleRequest($request);

// Détection de la soumission du formulaire
if ($form->isSubmitted()) {
  $training = $form->getData();
}
return $this->render("demo25.html.twig", [
  "form" => $form->createView() ]);
```

Formulaire construit dans le contrôleur et transmis au template

```twig
{% extends "demo.base.html.twig" %}
{% block pageTitle %}demo 25{% endblock %}
{% block body %}
    <h3>Ajout d'une formation</h3>
    {{ form(form) }}
{% endblock %}
```

**Ajout d'une formation**

Intitulé
Niveau
Nombre de jours
Enregistrer

# Classes de formulaire

Importe des types

Définit une méthode **buildForm**

Construit le formulaire

Ajoute des champs au formulaire par la méthode **add**

Add relie les champs de l'entité cible avec la champs du formulaire

EntityType.php

Template

```
<form method="POST">
  <input name="param" />
</form>
```

# Classes de formulaire

/src/Form/StudentType.php

```
class StudentType extends AbstractType
{
  public function buildForm(FormBuilderInterface $builder, array
$options)
  {
    $builder->add('name', TextType::class)
      ->add('status', TextType::class)
      ->add('email', EmailType::class)
      ->add('country', EntityType::class, [
        "class" => Country::class,
        "choice_label" => "name"])
      ->add('training', EntityType::class, [
        "class" => Training::class,
        "choice_label" => "title",
        "multiple" => true])
      ->add('save', SubmitType::class, ["label" => "Enregistrer"]);
  }
}
```

controller

```
$student = new Student();
$form = $this->createForm(StudentType::class,
$student);

$form->handleRequest($req);

if ($form->isSubmitted()) {
  $student = $form->getData();}

return $this->render("demo27.html.twig", [
 "form" => $form->createView() ]);
```

L'objet construit par la méthode **buildForm** de la classe **StudentType** est passé en entrée de la méthode **createForm** accessible depuis tout contrôleur

# Validation de formulaire

- SF offre un bundle de validation
  - composer require symfony/validator doctrine/annotations
- Utilisation sous forme de service et d'annotations (attributs d'entité)
- use Symfony\Component\Validator\Constraints as Assert;
- Le bundle validator expose un ensemble de contraintes
  - @Assert\[NomDeLaContrainte]
- Certaines contraintes ont des options de configuration

# Tests unitaires et fonctionnels

- SF prévoit le dossier /tests pour y placer les fichiers php de test
- Le framework phpunit est utilisé pour les tests
- L'exécutable **phpunit** est accessible comme la console dans le dossier **/bin**
- Une bonne pratique consiste à organiser les fichiers de tests en respectant une structure dossier similaire aux classes testées et de leur ajouter le suffixe **Test**
  - Exemple: /src/**Utils**/Calculator.php ⇔ /tests/**Utils**/Calculator**Test**.php
- https://symfony.com/doc/4.4/testing.html

# Test unitaire: exemple

```php
namespace App\Util;

class Calculator
{
    public function add($a, $b)
    {
        return $a + $b;
    }
}
```

```php
namespace App\Tests\Util;
use App\Util\Calculator;
use PHPUnit\Framework\TestCase;

class CalculatorTest extends TestCase
{
    public function testAdd()
    {
        $calculator = new Calculator();
        $result = $calculator->add(30, 12);
        $this->assertEquals(42, $result);
    }
}
```

La classe **CalculatorTest**, étendant **TestCase** (fourni par PHPUnit) contient un TU portant sur la function **add**. Le test réalise une assertion via la méthode **assertEquals**.

# Test unitaire: bin/phpunit

```
chris@aramis:~/symfony/test$ php bin/phpunit tests/Util/
PHPUnit 7.5.20 by Sebastian Bergmann and contributors.

Testing tests/Util/
.                                                                   1 / 1 (100%)

Time: 33 ms, Memory: 6.00 MB

OK (1 test, 1 assertion)
```

La commande php bin/phpunit [dossier] indique le dossier à parcourir. Phpunit examine les classes de Test placées dans */tests/Util* et exécute les méthodes qu'il y trouve. Ici, un test (contenant une assertion) est exécuté. Ce test passe.

# Injection de dépendance

- L’injection de dépendances (DI - Dependency Injection) est un modèle de conception (Design Pattern) très commun en POO
- Consiste à instancier une classe A dans le constructeur de classe B
- Liée à la notion de “service”
- Abondamment utilisée par SF, exemples: Doctrine, FormBuilder, Render, etc.
- Permet de centraliser des traitements afin de leurs rendre “injectables” pour toute classe (Controller, Service, etc.) en ayant l’utilité
- Un service se place dans le dossier **/src/Service** et porte le suffixe Service
- Les services sont paramétrables dans le fichier **services.yaml**

# Service: exemple

```php
namespace App\Service;

class CalculatorService
{
  private $tva_taux;
  public function __construct($tva_taux = 20
  {
      $this->tva_taux = $tva_taux;
  }
  public function tva($prix_ht)
  {
    $ttc = $prix_ht + ($prix_ht * ( $this->tva_taux/100 ));
    return $ttc;
  }
}
```

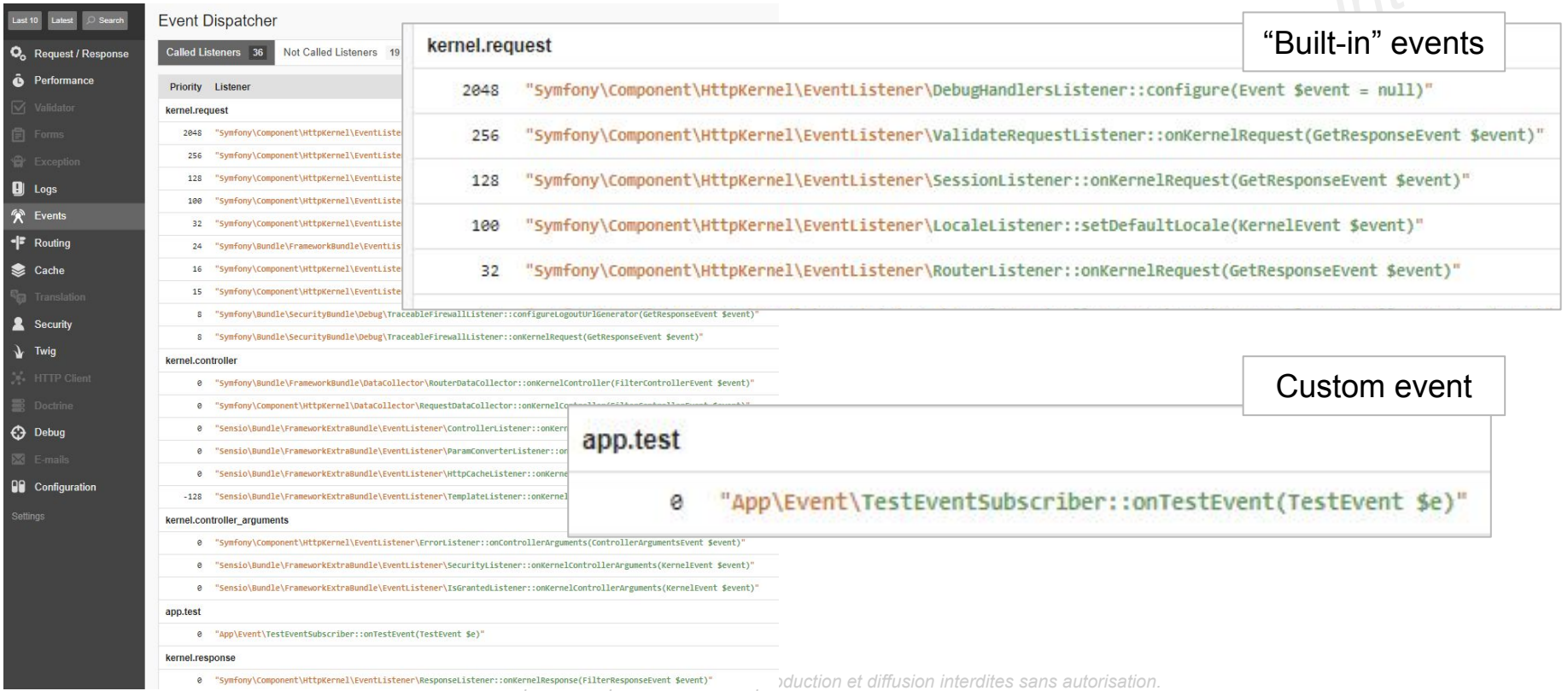
```php
class TestController extends AbstractController
{
  private $calculator;
  public function __construct(
    CalculatorService $calculator)
  {
    $this->calculator = $calculator;
  }
```

Une Injection de dépendance est réalisée au niveau de constructeur de la classe *TestController*. La propriété objet **$this->calculator** est accessible pour toute méthode de la classe *TestController*

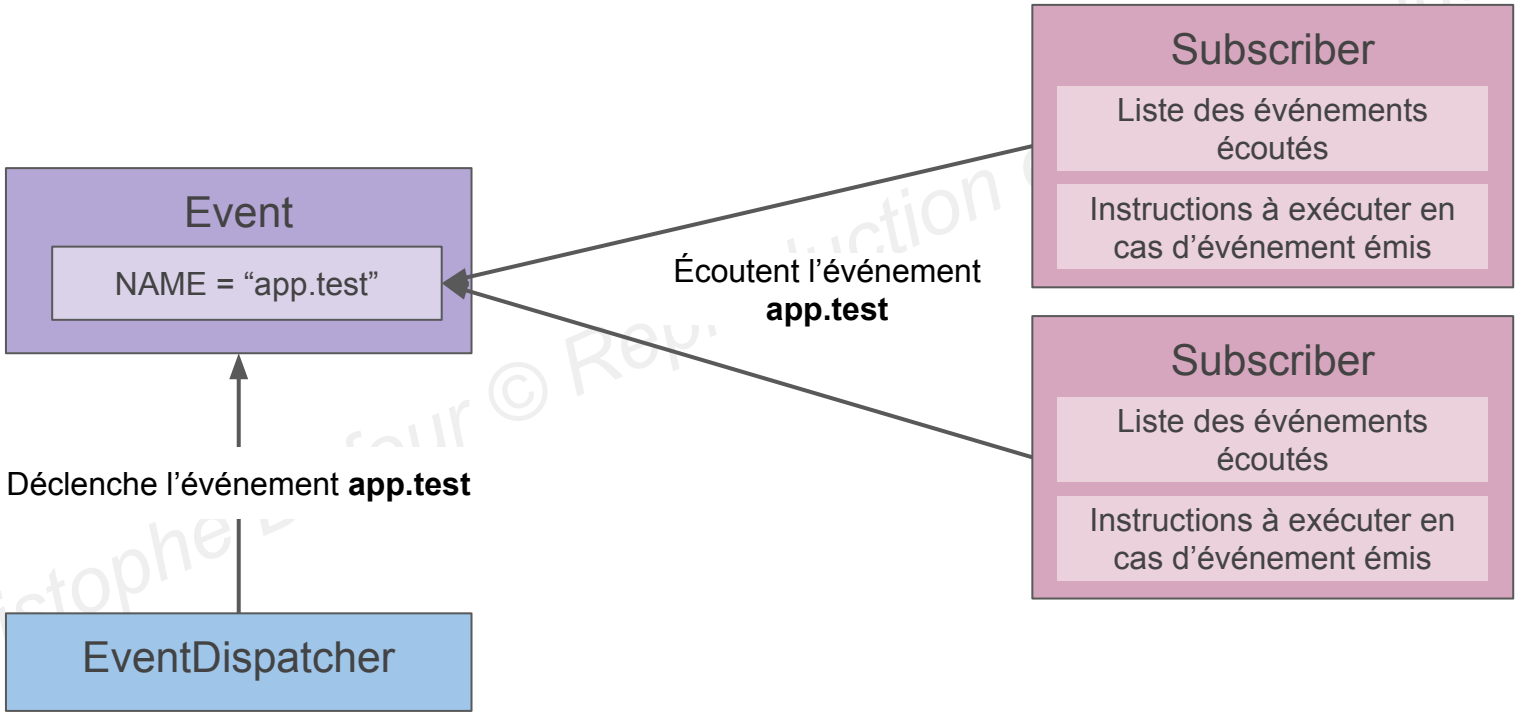
# Les événements Symfony

- SF inclut un mécanisme de publication/souscription permettant à des composants de l'application d'émettre des événements (signaux) à destination d'autres composants
- Implémentation du modèle de conception **Subscriber** (PUB/SUB)
- Il existe des évéments built-in, fournis en standard. Exemple: kernel.request
- Il est possible de créer ses propres événements
- Les classes d'événement sont à placer dans le dossier **/src/Event**

# Les événements Symfony



# Les événements: schéma



# Les événements: exemple

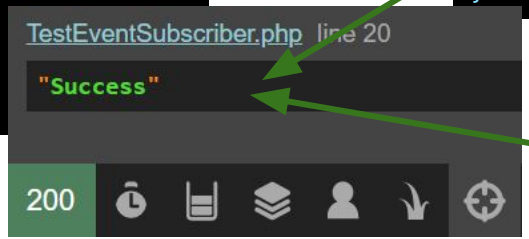
```php
namespace App\Event;
use Symfony\Contracts\EventDispatcher\Event;
class TestEvent extends Event
{
    public const NAME = "app.test";
    private $message;

    public function __construct(string $message)
    {
        $this->message = $message;
    }

    public function getMessage(): string
    {
        return $this->message;
    }
}
```

```php
namespace App\Event;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
class TestEventSubscriber implements EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        return [TestEvent::NAME => "onTestEvent"];
    }

    public function onTestEvent(TestEvent $e)
    {
        dump($e->getMessage());
    }
}
```




controller

```php
$dispatcher = new EventDispatcherInterface()
$e = new TestEvent("Success");
$dispatcher->dispatch($e, TestEvent::NAME);
```

# Créer une extension Twig

- Twig embarque en standard de nombreuses balises, filtres et fonctions:
  - Exemples: for, if; date, length, min, max, etc.
- Il est possible d'étendre Twig en étendant la classe **Twig\Extension\AbstractExtension**
- La classe peut définir:
  - De nouveaux **filtres** par le retour de la méthode **getFilters()**
    - Retourne un tableau de Twig**TwigFilter**
  - De nouvelles **fonctions** par le retour de la méthode **getFunctions()**
    - Retourne un tableau de Twig**TwigFunction**
- Les classes d'extension Twig sont à placer dans un dossier **/src/Twig**

# Filtre Twig

```php
namespace App\Twig;
use Twig\Extension\AbstractExtension;
use Twig\TwigFilter;

class AppExtension extends AbstractExtension
{
    public function getFilters()
    {
        $twigFilter = new TwigFilter("price", [$this, "formatPrice"]);
        return [$twigFilter];
    }
    public function formatPrice($number, $currency)
    {
        if ($currency == "USD") {
            return trim($currency) . " " . $number;
        } else {
            return $number . " " . trim($currency);
        }
    }
}
```

Template twig

```twig
<p>{{ "15" | price("EUR") }}</p>
```

Le constructeur de la classe **TwigFilter** associe le nom donné au filtre *price* à un nom de méthode *formatPrice*.
Cette méthode associée est définie dans la même classe *AppExtension*.

# Fonction Twig

```php
use Twig\TwigFunction;

class AppExtension extends AbstractExtension
{
    public function getFunctions()
    {
        $twigFn1 = new TwigFunction("tva", [$this, "tva"]);
        $twigFn2 = new TwigFunction("square", [$this, "square"]);
        return [$twigFn1, $twigFn2];
    }
    public function tva($prix_ht, $taux = 0.2)
    {
        return $prix_ht + ($prix_ht * $taux);
    }
    public function square($num)
    {
        return $num * $num;
    }
}
```

Template twig

```twig
<p>{{ tva(100) }}</p>
<p>{{ square(6) | price("USD") }}</p>
```

```
<p>120</p>
<p>USD 36</p>
```

Le constructeur de la classe **TwigFunction** associe nom de fonction et nom de méthode associée.
Les méthodes associées sont définies dans la même classe *AppExtension*.
Dans l'exemple, la sortie de la function square est "pipée" vers le filtre price

# Créer une commande Console

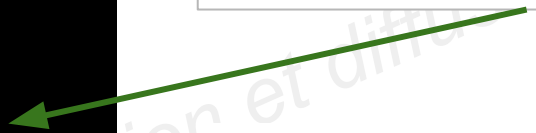
- SF fourni en standard en grand nombre de commandes Console
  - Exemples: make:controller, make:entity, doctrine:migrations:migrate, cache:clear, etc.
- SF permet de créer ses propres commandes
- Permet de réaliser des opérations d'administration, sans passer par l'UI front
- Classe étendant Symfony\Component\Console\Command**Command**
- A placer le dossier **/src/Comand** et portant le suffixe **Command**
- Le nom de la commande est une propriété statique (ex: "app:stuff")
- Deux méthodes à implémenter
  - **configure()**: détermine le menu d'aide et les arguments de commande attendus
  - **execute()**: détermine les actions que la commande doit réaliser

# Créer une commande: exemple

/src/Command/CreateUserCommand.php

```php
namespace App\Command;
class CreateUserCommand extends Command
{
    protected static $defaultName = "app:user:create";
    private $em;
    private $passwordEncoder;

    public function __construct(
        EntityManagerInterface $em,
        UserPasswordEncoderInterface $passwordEncoder)
    {
        $this->em = $em;
        $this->passwordEncoder = $passwordEncoder;
        parent::__construct();
    }
```




Commande permettant d'enregistrer un utilisateur en base de données.

Cette commande réalise deux injections de dépendance. L'une pour accéder à doctrine, l'autre pour l'encodage du mot de passe utilisateur

# Créer une commande: exemple

/src/Command/CreateUserCommand.php

```
protected function configure()
{
  $this
    ->setHelp("Cette commande permet de créer un utilisateur...")
    ->addArgument("email", InputArgument::REQUIRED, "Email")
    ->addArgument("password", InputArgument::REQUIRED, "Password")
     ;
}
```

```
chris@aramis:~/symfony/formation-symfony-4/demo$ php bin/console app:user:create

  Not enough arguments (missing: "email, password").
```

La méthode **configure()** permet de définir:
- le texte d'aide lorsqu'on utilise la commande avec l'option --help
- les arguments. Ici *email* et *password* sont 2 arguments obligatoires à fournir

# Créer une commande: exemple

```php
protected function execute(InputInterface $input, OutputInterface $output)
{
  $output->writeln("*********");
  $output->writeln(["Création d'un utilisateur", "**********"]);
  $email = $input->getArgument("email");
  $password = $input->getArgument("password");
  $output->writeln(["Email: ".$email, "Password: ".$password]);
  $user = new User();
  $user->setEmail($email);
  $user->setPassword(
   $this->passwordEncoder->encodePassword($user,$password));
  $user->setRoles(["ROLE_USER", "ROLE_ADMIN"]);
  $this->em->persist($user);
  $this->em->flush();
  return 0; // en SF5 Command::SUCCESS
}
```

La méthode **execute()** permet de récupérer les arguments fournis en entrée grâce à l'objet **$input**, et d'effectuer des sorties dans la console grâce à l'objet **$output**

Ici, email et password en clair (encodé lors de l'exécution) sont récupérés afin d'enregistrer un utilisateur en base de données

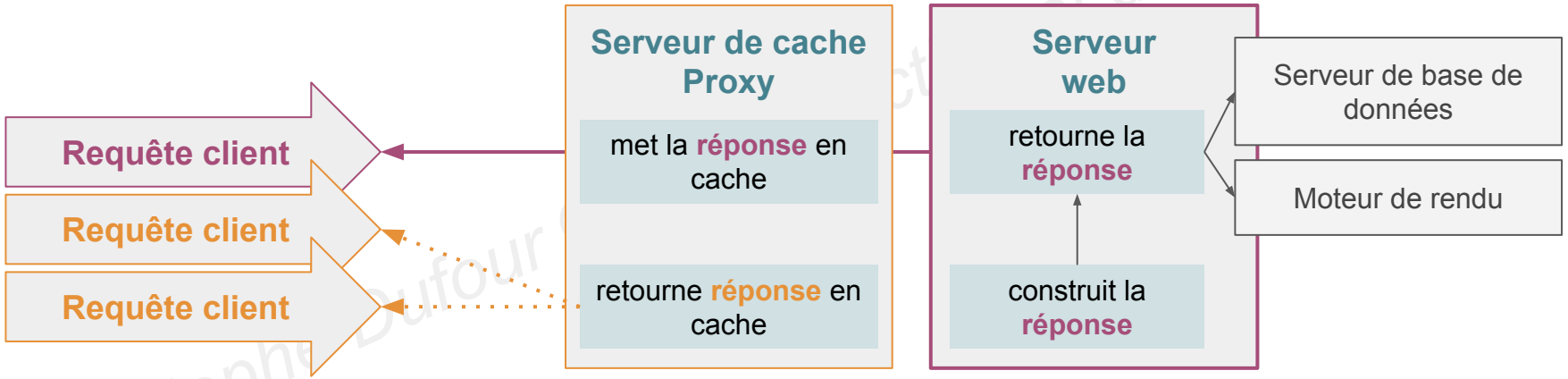
# Le cache HTTP

Le cache HTTP offre des gains en performance en allégeant la charge serveur. Les réponses mises en cache (cached) peuvent être directement retournées aux clients sans redemander au serveur de les “reconstruire” intégralement(accès DB, template engine, etc).

Il existe des serveurs de cache spécialisés et optimisés tels que *Varnish*

SF fournit son propre mécanisme de cache, le **Symfony Reverse Proxy,** très simple à mettre en place et à utiliser.

# Le cache HTTP: schéma



# Le reverse proxy de SF

/src/CacheKernel.php

```php
namespace App;
use Symfony\Bundle\FrameworkBundle\HttpCache\HttpCache;
class CacheKernel extends HttpCache {}
```

/public/index.php

```php
use App\CacheKernel;
$kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);

// Mise en place du Cache Proxy Symfony
// se fait normalement dans l'env de production
if ($kernel->getEnvironment() == "dev") {
    $kernel = new CacheKernel($kernel); // wrapping du kernel
}
$request = Request::createFromGlobals();
```

Pour mettre le Cache Proxy de SF en place, il suffit:

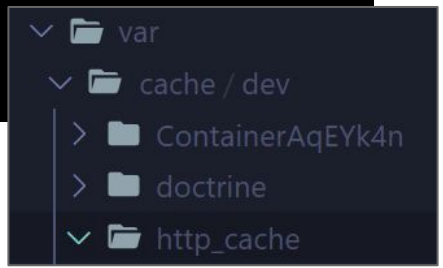
- de créer une classe vide **CacherKernel.php**

- d'instancier cette classe dans le front controller (index.php) en passant l'objet **$kernel** à son constructeur.

Se fait normalement dans l'environnement de production.

# Reverse proxy: exemple d'utilisation

```php
/** @Route("/demo33") */
public function demo33()
{
  $repo = $this->getDoctrine()->getRepository(Student::class);
  $students = $repo->findAll();
  $res = $this->render("demo33.html.twig", [
   "students" => $students]);
  $res->setPublic();
  $res->setMaxAge(60 * 10); // 10 minutes en cache
  return $res;
}
```




La réponse à cette requête est mise en cache sous forme d'un fichier statique placé dans le dossier **/var/cache/dev/http_cache**.
Pendant 10 minutes, le reverse proxy servira directement au client ce fichier statique

# Configuration pour serveur Apache

- https://symfony.com/doc/current/setup/web_server_configuration.html
- composer require symfony/apache-pack: génère .htaccess dans le dossier public
- sudo a2enmod rewrite # active le module rewrite
- Créer un hôte virtuel