



**formation.tech**

# Formation Symfony

Romain Bohdanowicz

[formation.tech](http://formation.tech)



**formation.tech**

# PHP Objet

# PHP Objet - Introduction



- La programmation orientée objet est un paradigme de programmation c'est à dire un style de programmation que l'on donne à notre code
- Ce paradigme s'inspire des objets qui nous entourent, une table, une chaise, un ordinateur...
- Par rapport à d'autres paradigmes il met l'accent sur la réutilisation du code
- Autres paradigmes :
  - programmation impérative
  - programmation procédurale
  - programmation fonctionnelle
  - ...

# PHP Objet - Classe



- Une classe : un concept
- Un objet : la représentation en mémoire de ce concept

```
<?php

class Contact
{
    public $id = 1;
    public $firstName = '';
    public $lastName = '';
}
```

```
<?php
require_once 'classes/Entity/Contact.php';

$romain = new Contact();

$romain→id = 1;
$romain→firstName = 'Romain';
$romain→lastName = 'Bohdanowicz';

$eric = new Contact();

$eric→id = 2;
$eric→firstName = 'Eric';
$eric→lastName = 'Grondin';
```

# PHP Objet - Classe



- › Dans une classe on retrouve
  - des variables appelées : propriétés, attributs ou champs (selon la littérature)
  - des fonctions appelées : méthodes

```
<?php

class Contact
{
    public $firstName = 'Romain';

    public function hello() {
        return 'Hello';
    }
}
```

# PHP Objet - Objet



- Une classe permet la création de l'objet, via l'utilisation de l'opérateur new
- L'objet aura la forme définie par la classe, c'est à dire un ensemble de propriétés et de méthodes
- On dit souvent que la classe est un moule qui permet de créer l'objet
- La classe peut être utilisée comme un type

```
<?php
require_once 'classes/Entity/Contact.php';

$romain = new Contact();

function sayHello(Contact $contact) {
    var_dump($contact->hello()); // Hello
}

sayHello($romain);
```

# PHP Objet - Visibilité



- On peut définir une visibilité pour les propriétés et les méthodes (membres) :
  - public : le membre est accessible dès lors que l'on a accès à l'objet
  - protected : le membre est accessible uniquement dans un contexte d'héritage, via les classes parents et les classes enfants
  - private : le membre n'est accessible qu'à l'intérieur de la classe
- Par défaut les membres sont publics

# PHP Objet - Encapsulation



- Par convention une classe ne devrait jamais définir de propriétés publiques
- Comme beaucoup de concepts objets, c'est une convention héritée de Java
- En général les propriétés sont protégées sauf si l'on veut ne pas y avoir accès dans les classes liées par héritage
- C'est ce qu'on appelle le *principe d'encapsulation*
- La classe peut ainsi être vue comme une boite noire, nous n'avons pas besoin de connaître le détail de son implémentation interne, mais simplement comment on peut interagir avec elle
- C'est d'autant plus vrai lorsque la classe représente un concept informatique, par exemple une connexion à une base de données, a-t'on besoin de connaître les détails du protocole réseau ou simplement qu'il faut passer le nom d'utilisateur et le mot de passe pour se connecter ?
- Idem, dans la vie réelle je n'ai pas besoin connaître les détails des échanges entre mon téléphone portable et les antennes 4G/5G, juste qu'il faut que je compose un numéro et que j'appuie sur la touche appeler

# PHP Objet - Accesseurs



- Lorsqu'on applique le principe d'encapsulation, les propriétés n'étant pas publiques pourront tout de même être accessibles au travers de méthodes appelées accesseurs (getters/setters en anglais)
- Les getters permettent de lire la propriété, les setters d'y écrire une valeur
- En tant que créateur de la classe avons ainsi un contrôle plus grand, ne pas créer de setter implique une propriété en lecture seule
- On peut également y masquer du code assez simple comme des conversions / transformation (string vers integer, string to uppercase...)
- Un setter est une méthode qui commence par set suivi du nom de la propriété, elle affecte son paramètre d'entrée à la propriété
- Un getter est une méthode qui commence par get ou is (pour les propriété booléennes) suivi du nom de la propriété, elle retourne la propriété
- Ils peuvent être générés par les IDEs



# PHP Objet - Accesseurs

- Exemple

```
<?php

class Contact
{
    protected $firstName = '';
    protected $trainer = false;

    public function getFirstName() {
        return $this->firstName;
    }

    public function setFirstName(string $firstName) {
        $this->firstName = $firstName;
    }

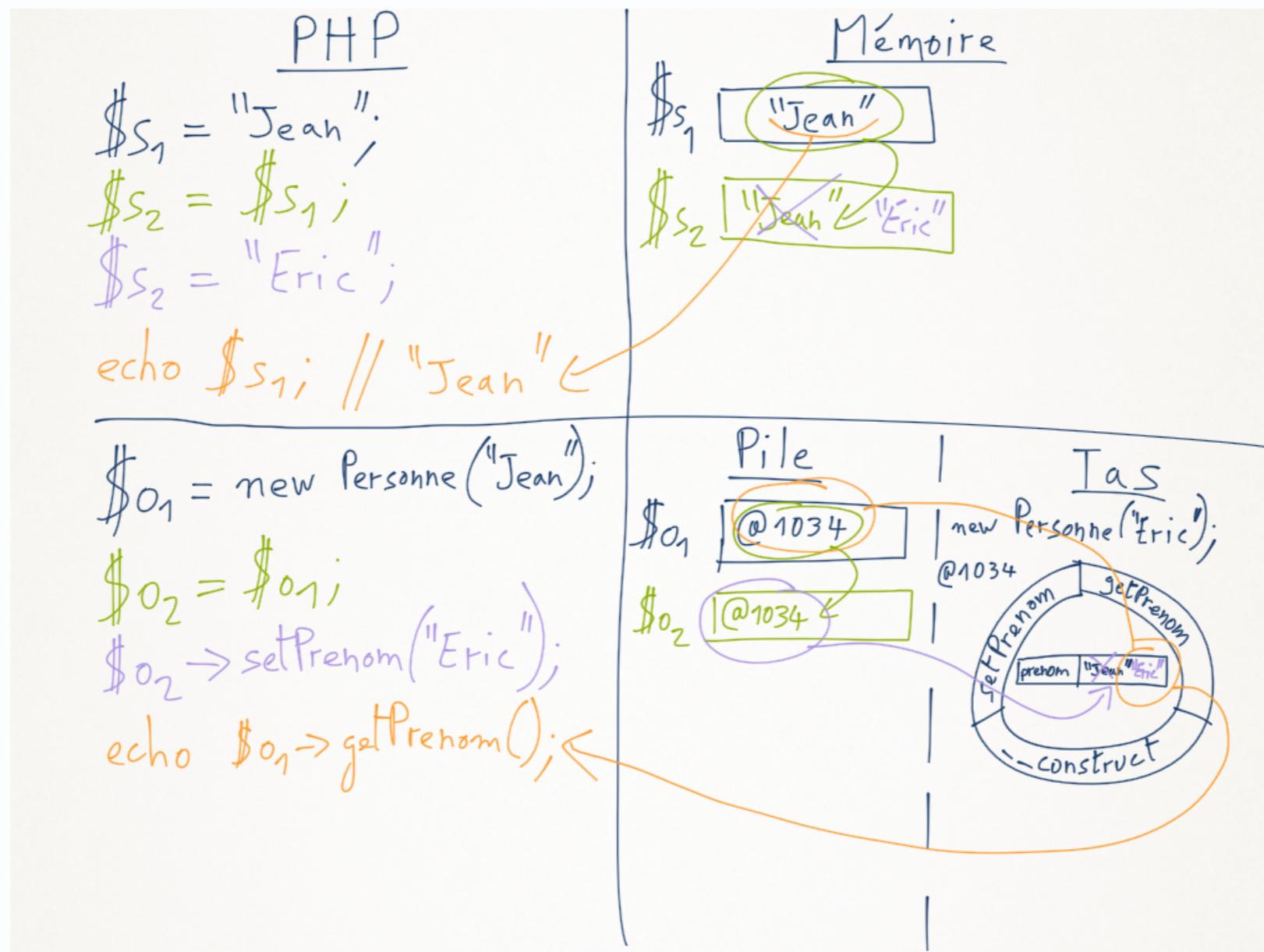
    public function isTrainer(): bool
    {
        return $this->trainer;
    }

    public function setTrainer(bool $trainer)
    {
        $this->trainer = $trainer;
    }
}
```

# PHP Objet - Référence



- A la différence des autres types (int, boolean, float, string, array...) les objets se manipulent au travers de références.
- Le contenu de la variable est en réalité un lien vers l'objet en mémoire





# PHP Objet - \$this

- \$this est une pseudo-variable (variable générée automatiquement) qui référence l'objet dans lequel on se trouve

```
<?php

class Contact
{
    protected $firstName = 'Romain';

    public function hello() {
        return 'Hello my name is ' . $this->firstName;
    }
}
```



# PHP Objet - Fluent setters

- Les setters n'ayant pas de valeur de retour, on en profite pour retourner \$this

```
<?php

class Contact
{
    protected $firstName = '';
    protected $lastName = '';

    public function getFirstName(): string
    {
        return $this->firstName;
    }

    public function setFirstName(string $firstName): Contact
    {
        $this->firstName = $firstName;
        return $this;
    }

    public function getLastName(): string
    {
        return $this->lastName;
    }

    public function setLastName(string $lastName): Contact
    {
        $this->lastName = $lastName;
        return $this;
    }
}
```



# PHP Objet - Fluent setters

- Cela permet de "chainer" les appels aux setters

```
<?php

require_once 'classes/Entity/Contact.php';

$romain = new Contact();
$romain→setFirstName('Romain');
$romain→setLastName('Bohdanowicz');

// Peut également s'écrire :
$romain = new Contact();
$romain→setFirstName('Romain')→setLastName('Bohdanowicz');
```

# PHP Objet - Associations



- Pour associer des objets entre eux, on va stocker la ou les références d'autres objets dans ses propriétés
- L'association peut être
  - unidirectionnelle : l'objet de la classe A a accès à l'objet de la classe B mais pas l'inverse
  - bidirectionnelle : les objets des classes A et B se référence dans les 2 sens
  - se référence elle même : l'objet de la classe A référence un autre objet de la classe A



# PHP Objet - Associations

- Exemple unidirectionnel

```
<?php

class Contact
{
    protected $firstName = '';

    /** @var Company */
    protected $company;
}
```

```
<?php

class Company
{
    protected $name;
}
```



# PHP Objet - Associations

- Exemple bidirectionnel

```
<?php

class Contact
{
    protected $firstName = '';

    /** @var Company */
    protected $company;
}
```

```
<?php

class Company
{
    protected $name;

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



# PHP Objet - Associations

- Exemple "self-referencing"

```
<?php

class Category
{
    protected $name;

    /** @var Category */
    protected $parent;

    /** @var Category[] */
    protected $children;
}
```

# PHP Objet - Namespaces



- En PHP il n'est pas possible de déclarer 2 classes ayant le même nom, qu'elles soient ou non dans le même fichier
- Pour éviter une erreur on peut préfixer les classes par des namespaces :

```
<?php

namespace FormationTech\Entity;

class Contact
{
    protected $firstName = '';
    protected $lastName = '';

    // ...
}
```

```
<?php

namespace FormationTech\Utils;

class Contact
{
    public function sendMail(\FormationTech\Entity>Contact $contact) {
        // ...
    }
}
```

```
<?php

require_once 'classes/Utils/Contact.php';
require_once 'classes/Entity/Contact.php';

$romain = new \FormationTech\Entity\Contact();
$romain->setFirstName('Romain');

$mailer = new \FormationTech\Utils\Contact();
$mailer->sendMail($romain, 'Hello');
```



# PHP Objet - Namespaces

- Le nom complet de la classe avec ses préfixes est appelées FQN ou FQCN pour Fully Qualified Class Name
- Il existe une convention sur comment devrait être nommée le FQN d'une classe appelée PSR-4 (cf le chapitre sur l'auto-chargement de classe)



# PHP Objet - Namespaces

- Pour raccourcir les lignes, on peut utiliser le mot clé use en début de fichier :

```
<?php

use FormationTech\Entity\Contact;

require_once 'classes/Entity/Contact.php';

$romain = new Contact();
$romain->setFirstName('Romain')->setLastName('Bohdanowicz');
```

- On peut aussi renommer (aliasser) une classe via le use :

```
<?php

use FormationTech\Entity\Contact;
use FormationTech\Utils>Contact as Mailer;

require_once 'classes/Utils/Contact.php';
require_once 'classes/Entity/Contact.php';

$romain = new Contact();
$romain->setFirstName('Romain');

$mailer = new Mailer();
$mailer->sendMail($romain, 'Hello');
```



# PHP Objet - Namespaces

- Les namespaces peuvent également être aliassés :

```
<?php

use \FormationTech\Entity as Entity;

require_once 'classes/Entity/Contact.php';
require_once 'classes/Entity/Company.php';

$company = new Entity\Company();
$company->setName('formation.tech')->setCity('Paris');

$contact = new Entity>Contact();
$contact->setFirstName('Romain')->setLastName('Bohdanowicz');
$contact->setCompany($company);
```



# PHP Objet - Namespaces

- Lorsque 2 classes se trouvent dans le même namespace, il n'y a pas besoin d'utiliser le mot clé use

```
<?php

namespace FormationTech\Entity;

class Contact
{
    /** @var Company */
    protected $company;

    public function getCompany(): Company
    {
        return $this->company;
    }

    public function setCompany(Company $company): Contact
    {
        $this->company = $company;
        return $this;
    }
}
```



# PHP Objet - Namespaces

- Lorsqu'on n'utilise pas use, on peut faire référence à des sous-namespaces un peu comme un chemin relatif :

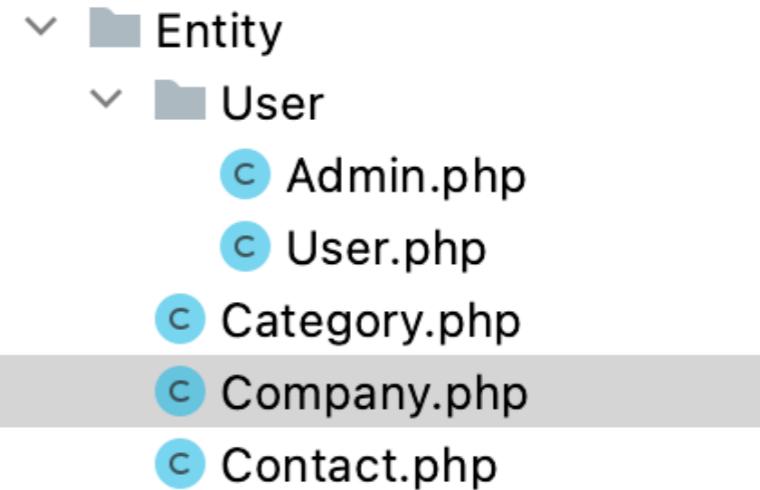
```
<?php

namespace FormationTech\Entity;

class Company
{
    /** @var User\Admin */
    protected $admin;

    public function getAdmin(): User\Admin
    {
        return $this->admin;
    }

    public function setAdmin(User\Admin $admin): Company
    {
        $this->admin = $admin;
        return $this;
    }
}
```



# PHP Objet - Namespaces



- Si une classe se trouve dans un namespace complètement différent, il faudra préfixer son nom par un antislash \ (pas de ../ possible)
- Ce sera également le cas pour les classes qui n'ont pas de namespace (PDO, Exception...)

```
<?php

namespace FormationTech\Entity;

class Company
{
    /** @var \FormationTech\Logger\Logger */
    protected $logger;

    public function getLogger(): \FormationTech\Logger\Logger
    {
        if (!$this->logger) {
            throw new \Exception('Missing logger');
        }
        return $this->logger;
    }
}
```

# PHP Objet - Namespaces



- › On pourra toujours utiliser use pour éviter les antislashes :

```
<?php

namespace FormationTech\Entity;

use Exception;
use FormationTech\Logger\Logger;

class Company
{
    /** @var Logger */
    protected $logger;

    public function getLogger(): Logger
    {
        if (!$this->logger) {
            throw new Exception('Missing logger');
        }
        return $this->logger;
    }

}
```

# PHP Objet - Auto-chargement



- Plutôt que d'inclure les classes une à une, on peut les auto-charger en écrivant une fonction d'auto-chargement
- Il faudra définir une convention qui corrèle le nom de la classe et le chemin du fichier la contenant
- Une convention largement répandue en PHP a été définie par le PHP Framework Interop Group, la convention PSR-4 (qui prend le relais sur la convention dépréciée PSR-0) :  
<https://www.php-fig.org/psr/psr-4/>
- Voir les autres conventions du PHP-FIG sur :  
<http://www.php-fig.org/>

# PHP Objet - Auto-chargement



- Dans la convention PSR-4 :
  - le terme classe peut faire référence à une classe ou des concepts proches comme des interfaces ou des traits
  - le FQN de la classe doit commencer par un Vendor : un namespace unique qui porte le nom du projet, le nom d'une société (Ex: FormationTech, Symfony, Zend...)
  - 0, 1 ou plusieurs namespace intermédiaire auxquels pourront correspondre des dossiers
  - le nom de la classe auquel correspondra un fichier (une classe par fichier et pas de side-effect : soit on déclare une classe, soit on l'utilise mais pas les 2).

# PHP Objet - Auto-chargement



- Exemples PSR-4 :
  - FormationTech\Entity\Contact → classes\Entity>Contact.php (le préfixe *FormationTech*\ est associé au dossier *classes*)
  - Symfony\Component\HttpFoundation\Response → vendor/symfony/http-foundation/Response.php (le préfixe *Symfony\Component\HttpFoundation*\ est associé au dossier *vendor/symfony/http-foundation*)



# PHP Objet - Auto-chargement

- Exemple d'autoloader :

```
<?php

spl_autoload_register(function ($class) {
    // project-specific namespace prefix
    $prefix = 'FormationTech\\';

    // base directory for the namespace prefix
    $base_dir = __DIR__ . '/classes/';

    // does the class use the namespace prefix?
    $len = strlen($prefix);
    if (strncmp($prefix, $class, $len) !== 0) {
        // no, move to the next registered autoloader
        return;
    }

    // get the relative class name
    $relative_class = substr($class, $len);

    // replace the namespace prefix with the base directory, replace namespace
    // separators with directory separators in the relative class name, append
    // with .php
    $file = $base_dir . str_replace('\\', '/', $relative_class) . '.php';

    // if the file exists, require it
    if (file_exists($file)) {
        require $file;
    }
});
```



# PHP Objet - Auto-chargement

- Plutôt que d'écrire son propre auto-loader, il est possible d'utiliser celui de composer, qui sera utilisé pour les classes installées via composer
- Dans le fichier composer.json, on peut ajouter une section autoload :

```
{  
    "autoload": {  
        "psr-4": {  
            "FormationTech\\": "classes"  
        }  
    }  
}
```

- Il faudra ensuite lancer la commande suivante pour générer les fichiers PHP  
`composer dump-autoload`

```
<?php  
  
// autoload_psr4.php @generated by Composer  
  
$vendorDir = dirname(dirname(__FILE__));  
$baseDir = dirname($vendorDir);  
  
return array(  
    'FormationTech\\' => array($baseDir . '/classes'),  
);
```

# PHP Objet - Méthodes magiques



- En PHP les méthodes magiques sont des méthodes appelées automatiquement lors de certaines opérations sur l'objet.
- `__construct()` : lors de la construction
- `__destruct()` : lors de la destruction
- `__clone()` : lors du clone
- `__toString()` : lors de la conversion vers le type string
- `__set()` : lors de l'écriture d'une propriété inexistante
- `__get` : lors de la lecture d'une propriété inexistante
- `__isset, __unset` : lors de l'appel des fonctions `isset/unset` sur une propriété inexistante
- `__call, __callStatic` : lors de l'appel d'une fonction inexistante
- ...

# PHP Objet - Méthodes magiques



- Exemple :

```
<?php

namespace FormationTech\Writer;

class FileWriter
{
    protected $handle;

    public function __construct($filename, $mode)
    {
        $this->handle = fopen($filename, $mode);
    }

    public function write($message) {

        fwrite($this->handle, "$message\n");
    }

    public function __destruct()
    {
        fclose($this->handle);
    }
}
```

# PHP Objet - Héritage



- Une autre façon d'associer 2 classes est d'utiliser l'héritage
- Une classe peut ainsi recevoir l'ensemble des propriétés et méthodes d'une autre classe sans duplication de code
- Pour des concepts métiers on doit pouvoir facilement placer les mots "est un" entre 2 classes, ex : un Admin est un User avec des droits plus élevés



# PHP Objet - Héritage

- Exemple :

```
<?php

namespace FormationTech\Entity;

class Admin extends User
{
    protected $canCreateUser = false;

    public function isCanCreateUser(): bool
    {
        return $this->canCreateUser;
    }

    public function setCanCreateUser(bool $canCreateUser): Admin
    {
        $this->canCreateUser = $canCreateUser;
        return $this;
    }
}
```

```
<?php
require_once 'vendor/autoload.php';

$user = new \FormationTech\Entity\User();
$user->setUsername('romain')->setPassword('123456');

$admin = new \FormationTech\Entity\Admin();
$admin->setUsername('romain')->setPassword('123456')->setCanCreateUser(true);
```

# PHP Objet - Héritage



- Lorsqu'on hérite d'une classe, il est parfois nécessaire de redéfinir des méthodes de la classe parent pour spécialiser leur comportement (créer une méthode ayant le même nom que la classe parent)
- Il peut être alors nécessaire dans ce cas d'appeler la méthode parent en utilisant le mot clé parent



# PHP Objet - Héritage

- Exemple

```
<?php

namespace FormationTech\Writer;

class CsvWriter extends FileWriter
{
    protected $separator = ',';

    public function __construct($filename, $separator = ',')
    {
        parent::__construct($filename, 'a');
        $this->separator = $separator;
    }

    public function write($message)
    {
        if (is_array($message)) {
            $message = implode(',', $message);
        }

        parent::write($message);
    }
}
```

# PHP Objet - Interface



- Une interface est un ensemble de méthodes dont on veut forcer la définition
- Elles sont souvent utilisées par les bibliothèques pour permettre d'étendre leur fonctionnalité
- Par exemple une bibliothèque d'envoi de mail qui propose de base l'envoi de mail via SMTP ou la fonction mail de PHP, pourrait proposer qu'on l'étendent pour pouvoir envoyer des emails via des services spécialisé : SendinBlue, Mailjet, SendGrid...
- Lorsque la bibliothèque devra envoyer son email elle dépendra de l'interface (une abstraction) plutôt qu'un des classes (une implémentation)

```
<?php  
  
namespace FormationTech\Writer;  
  
interface WriterInterface  
{  
    public function write($message);  
}
```



# PHP Objet - Interface

- Une classe qui implémente une interface devra redéfinir ses méthodes en respectant les signatures
- Une classe peut implémenter plusieurs interfaces

```
<?php

namespace FormationTech\Writer;

class FileWriter implements WriterInterface
{
    protected $handle;

    public function __construct($filename, $mode)
    {
        $this->handle = fopen($filename, $mode);
    }

    public function write($message) {

        fwrite($this->handle, "$message\n");
    }

    public function __destruct()
    {
        fclose($this->handle);
    }
}
```

# PHP Objet - Classe abstraite



- Parfois implémenter une interface peut se révéler contraignant car il faut redéfinir l'ensemble de ses méthodes

```
<?php

namespace FormationTech\Logger;

class Logger implements \Psr\Log\LoggerInterface
{
    public function log($level, $message, array $context = array())
    {
        $fd = fopen('app.log', 'a');
        fwrite($fd, "[{$level}: {$message}");
    }

    public function emergency($message, array $context = array())
    {
        $this→log(\Psr\Log\LogLevel::EMERGENCY, $message, $context);
    }

    public function alert($message, array $context = array())
    {
        $this→log(\Psr\Log\LogLevel::ALERT, $message, $context);
    }

    public function critical($message, array $context = array())
    {
        $this→log(\Psr\Log\LogLevel::CRITICAL, $message, $context);
    }

    public function error($message, array $context = array())
    {
```

# PHP Objet - Classe abstraite



- Dans ce cas il sera utile d'avoir recours à une classe abstraite, c'est à dire une classe dont certaines méthodes sont définies et d'autres restent à implémenter

```
<?php

namespace FormationTech\Logger;

class Logger extends \Psr\Log\AbstractLogger
{
    public function log($level, $message, array $context = array())
    {
        $fd = fopen('app.log', 'a');
        fwrite($fd, "[{$level}: {$message}");
    }
}
```

- Une classe abstraite peut également ne pas contenir de méthodes à implémenter, c'est dans ce cas le signe que cette classe est présente dans la bibliothèque pour permettre d'étendre ses fonctionnalités

# PHP Objet - Traits



- Les classes abstraites ont un inconvénients, elles nécessitent d'avoir recours à l'héritage qui est simple en PHP (on ne peut hériter que d'une classe à la fois)
- Pour passer outre cette limitation on peut avoir recours au combo interface + trait
- L'utilisation de traits est multiple et peut s'apparenter à un copier/coller de méthodes dans une classe

```
<?php

namespace FormationTech\Logger;

use Psr\Log\LoggerInterface;
use Psr\Log\LoggerTrait;

class Logger implements LoggerInterface
{
    use LoggerTrait;

    public function log($level, $message, array $context = array())
    {
        $fd = fopen('app.log', 'a');
        fwrite($fd, "[{$level}: {$message}]");
    }
}
```



formation.tech

# Injection de dépendance

# Injection de dépendance - Composition



- Il arrive souvent que 2 classes dépendent l'une de l'autre
- Exemple : pour créer une tasse de café, j'ai besoin d'une instance de la classe Tasse et une instance de la classe Café
- Lorsque le new est fait à l'intérieur d'une autre classe, on parle de composition

```
class Cafe
{
    protected $variете;

    public function __construct($variете)
    {
        $this->variете = $variете;
    }
}
```

```
class Tasse
{
    protected $contenu;

    public function __construct() {
        $this->contenu = new Cafe("Arabica");
    }
}
```

# Injection de dépendance - Composition



- Lorsque les classes représente des valeurs : DateTime, Currency, ArrayObject... ce n'est pas problématique
- Par contre lorsque les classes représente des concepts informatiques cela limite leur réutilisation
- Pour revenir à notre exemple, comment créer une tasse de café :

```
<?php  
  
require_once 'autoload.php';  
  
$tasseDeCafe = new \FormationTech\Tasse();
```

- Peut-on réutiliser ce code pour créer une tasse de thé ?



# Injection de dépendance - Exemple

- Pour permettre plus de flexibilité le mieux est d'avoir recours à l'injection de dépendance
- Le new se passe alors en dehors des classes :

```
class Tasse
{
    protected $contenu;

    public function __construct($contenu) {
        $this->contenu = $contenu;
    }
}
```

```
$cafe = new \FormationTech\Cafe();
$the = new \FormationTech\The();
$tasseDeCafe = new \FormationTech\Tasse($cafe);
$tasseDeThe = new \FormationTech\Tasse($the);
```

- On peut maintenant créer des tasses de café ou de thé simplement

# Injection de dépendance - Exemple



- Dans l'idéal on viendrait créer une catégorie de classe via une interface pour limiter l'injection à cette catégorie

```
interface Liquide {}
```

```
class Cafe implements Liquide
{
    protected $variете;

    public function __construct($variете)
    {
        $this->variете = $variете;
    }
}
```

```
class Tasse
{
    protected $contenu;

    public function __construct(Liquide $contenu) {
        $this->contenu = $contenu;
    }
}
```

# Injection de dépendance - Exemple



- Avec des classes représentant des concepts informatiques cela donnerait :

```
<?php

namespace FormationTech\Logger;

use FormationTech\Writer\WriterInterface;
use Psr\Log\LoggerInterface;
use Psr\Log\LoggerTrait;

class Logger implements LoggerInterface
{
    use LoggerTrait;
    protected $writer;

    public function __construct(WriterInterface $writer)
    {
        $this->writer = $writer;
    }

    public function log($level, $message, array $context = array())
    {
        // [14:50:04] Error : Erreur à la connection
        $heure = date('H:i:s');
        $this->writer->write("[$heure] $level: $message");
    }
}
```

# Injection de dépendance - Exemple



- Ici notre classe Logger peut recevoir n'importe quelle instance de WriterInterface et ainsi logguer dans un fichier, dans une base de données, par email, nul part...

```
<?php

use FormationTech\Logger\Logger;
use FormationTech\Writer\FileWriter;

require_once 'vendor/autoload.php';

$writer = new FileWriter('app.log', 'a');
$logger = new Logger($writer);
$logger->error('Ligne 1');
$logger->error('Ligne 2');
```

- L'injection de dépendance permettra de tester unitairement des classes qui ont des dépendances, on injectera des fausses dépendances définies le temps du tests (exemple : une classe writer qui enregistre les appels aux méthodes error, debug...)

# Injection de dépendance - Container



- A force d'utiliser l'injection de dépendance on peut se trouver dans des situations avec de nombreux objets à instancier.
- C'est d'autant plus difficile lorsqu'on utilise une bibliothèque que l'on a pas développé :

```
<?php

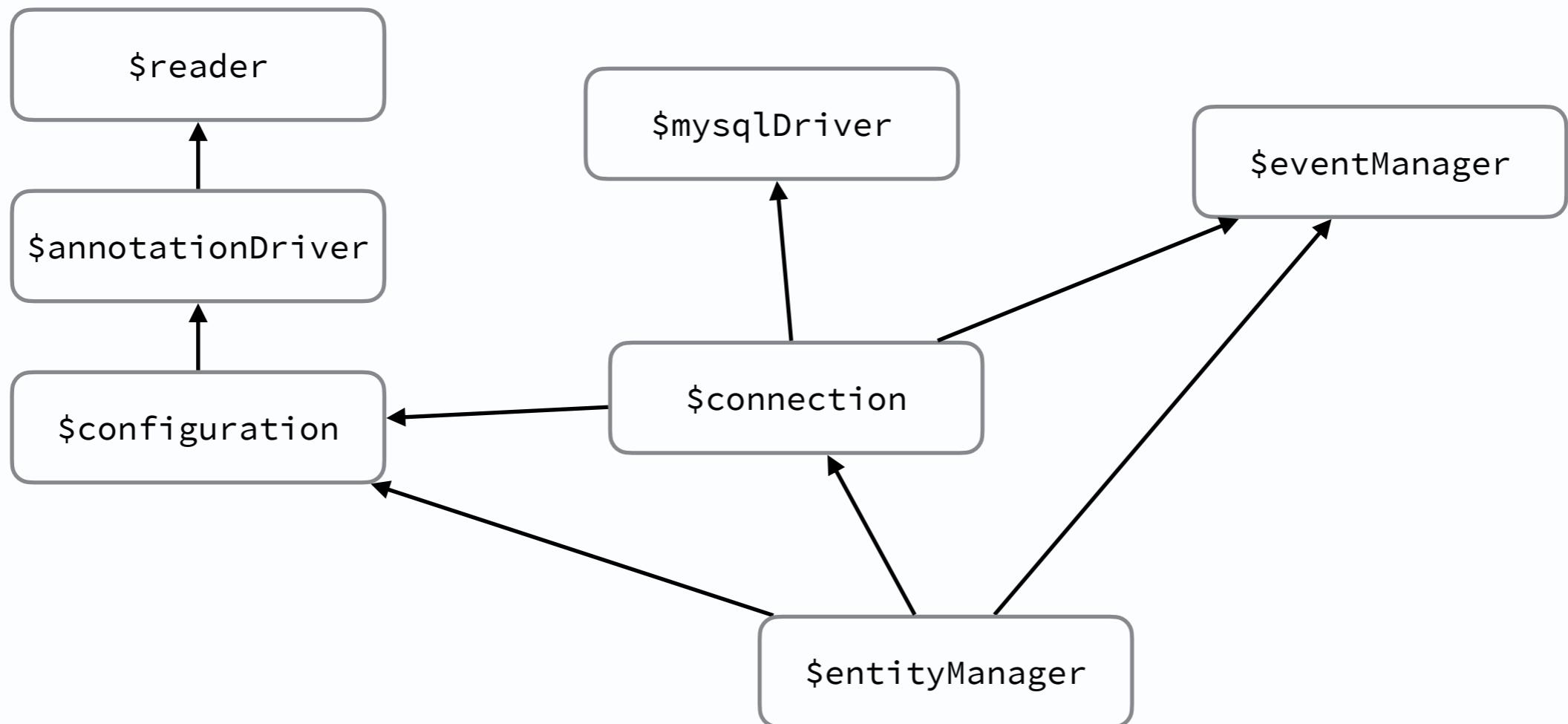
require 'vendor/autoload.php';

$eventManager = new \Doctrine\Common\EventManager();
$mysqlDriver = new \Doctrine\DBAL\Driver\PDO\MySQL\Driver();
$configuration = new \Doctrine\ORM\Configuration();
$configuration->setProxyDir('./proxies');
$configuration->setProxyNamespace('FormationTechProxy');
$reader = new \Doctrine\Common\Annotations\AnnotationReader();
$annotationDriver = new \Doctrine\ORM\Mapping\Driver\AnnotationDriver($reader);
$connection = new \Doctrine\DBAL\Connection([], $mysqlDriver, $configuration,
$eventManager);
$configuration->setMetadataDriverImpl($annotationDriver);
$entityManager = \Doctrine\ORM\EntityManager::create($connection, $configuration,
$eventManager);
```

# Injection de dépendance - Container



- Schématiquement :



# Injection de dépendance - Container



- › Pour simplifier la création et la réutilisation des objets dans un graphe de dépendance, il devient indispensable d'utiliser un Conteneur d'injection de dépendance (DIC)
- › Dans ce cas il devient utile d'utiliser un conteneur d'injection de dépendance qu'on aura configuré au préalable. En PHP il existe quelques DIC connus :
  - Pimple par Fabien Potencier
  - Dice par Tom Butler
  - PHP-DI par Matthieu Napoli
  - Symfony\Dependency\Injection par Symfony
  - Zend\Di ou Zend\ServiceManager par Zend Framework

# Injection de dépendance - Container



- Avec un conteneur d'injection de dépendance, la création des objets se configure (sous forme de code PHP ou parfois en XML, YAML...)
- Les objets sont par défaut définis sous forme de Singleton (une instance de l'objet maximum)
- Les objets sont instanciés à la demande, si votre code n'en a pas besoin, ils ne sont pas créés inutilement



**formation.tech**

# Framework

# Framework - Définitions



- Qu'est-ce qu'un framework web ?
  - Un ensemble de bibliothèques, dont chacune se préoccupera d'une problématique précise (envoi de mail, gestion des formulaires, ...)
  - Un cadre de travail au travers d'une architecture recommandée (via des Design Patterns, des squelettes d'application, de la documentation sur les bonnes pratiques d'architecture...)



# Framework - Historique

- Java  
Struts (2000), Spring (2003), GWT (2006), Play (2007)...
- Ruby  
Ruby on Rails (2005), Sinatra (2007)...
- Python  
Django (2005)...
- JavaScript  
Express (2009), AngularJS (2010), Ember.js (2011), Meteor (2012), Sails.js (2012)...

# Framework - PHP



- Micro-framework
  - Avantages : Performance, facilité d'apprentissage
  - Inconvénients : Peu adaptés aux projets volumineux
  - Principaux frameworks : Slim (2010), Expressive (2016)
- « Convention over configuration » framework
  - Avantages : Développement rapide
  - Inconvénients : Conventions à connaître, difficile à personnaliser
  - Principaux frameworks : CakePHP (2005), CodeIgniter (2006), Laravel (2011), Phalcon (2012)
- Framework Entreprise
  - Avantages : Adaptables à toutes les situations
  - Inconvénients : Nécessite de configurer, développement ralenti
  - Principaux frameworks : Symfony (2005), Zend Framework / Laminas (2006)

# Framework - PHP



- 2 chapitres à lire avant de commencer
  - Symfony et les fondations HTTP  
[https://symfony.com/doc/current/introduction/http\\_fundamentals.html](https://symfony.com/doc/current/introduction/http_fundamentals.html)  
(en français, mais plus très à jour) : [https://github.com/symfony-fr/symfony-docs-fr/blob/master/book/http\\_fundamentals.rst](https://github.com/symfony-fr/symfony-docs-fr/blob/master/book/http_fundamentals.rst)
  - Passer de PHP "classique" à un framework :  
[https://symfony.com/doc/current/introduction/from\\_flat\\_php\\_to\\_symfony.html](https://symfony.com/doc/current/introduction/from_flat_php_to_symfony.html)  
(en français, mais plus très à jour) : [https://github.com/symfony-fr/symfony-docs-fr/blob/master/book/from\\_flat\\_php\\_to\\_symfony2.rst](https://github.com/symfony-fr/symfony-docs-fr/blob/master/book/from_flat_php_to_symfony2.rst)
- Ensuite lire Symfony : The Fast Track  
<https://symfony.com/doc/current/the-fast-track/fr/index.html>
-



**formation.tech**

# Symfony

# Symfony - Introduction



- Historique
  - Créé en 2005 par Fabien Potentier pour son agence SensioLabs sous le nom Sensio Framework, rendu Open Source sous le nom Symfony
  - Symfony 1 | PHP >= 5.0 | 2007
  - Symfony 2 | PHP >= 5.3 | 2011
  - Symfony 3 | PHP >= 5.4 | 2015
  - Symfony 4 | PHP >= 7.1 | 2017
  - Symfony 5 | PHP >= 7.2 | 2019
  - Symfony 6 | PHP >= 8.0 | 2021
- LTS
  - Les versions 1.4, 2.3, 2.7, 2.8, 3.4, 4.4, 5.4 sont dites LTS (Long Term Support)
  - Elles sont maintenues pendant 4 ans contre 8 mois en moyenne pour les autres versions

# Symfony - Installation



- Création du projet (depuis Symfony 4)
- Micro-framework :
  - `composer create-project symfony/skeleton NOM_PROJET`
- Site web :
  - `composer create-project symfony/website-skeleton NOM_PROJET`
- Avec `symfony/skeleton` il y aura probablement plusieurs composant à installer et savoir configurer (doctrine, log, mails...)
- On peut également simplifier l'installation en utilisation `symfony/cli`  
<https://symfony.com/download>

# Symfony - Flex



- A partir de Symfony 4, les projets créés suivent l'architecture Symfony Flex
- Pour migrer un projet plus ancien :  
<https://symfony.com/doc/current/setup/flex.html>
- Avec Symfony Flex, il est possible d'utiliser des plugins pour composer qui viendront réaliser des tâches automatiquement :
  - créer certains fichiers / dossiers
  - créer des fichiers de configuration
  - ...
- Ces plugins s'appellent des recettes et sont publiées sur <https://flex.symfony.com/>
- On peut utiliser le nom du paquet packagist ou des aliases :

A screenshot of a web browser showing the Packagist package page for "symfony/web-server-bundle". The page has a light gray background with a white header bar. The main title is "symfony/web-server-bundle" in bold black font. Below it is a green rectangular button with the word "official" in white. Underneath the button, there is a row of three small gray rectangular boxes containing the words "Aliases", "server", and "web-server". Below these boxes is another row with a single box containing the text "web-server-bundle". At the bottom of the page, there are two blue links: "Package details" and "Recipe".



# Symfony - Serveur local

- Pour démarrer un serveur local on peut :
  - Configurer Apache / nginx...
  - Utiliser la commande :  
`php -S 127.0.0.1:8000 -t public`
  - Utiliser symfony/cli  
`symfony server:start`
  - Utiliser symfony/web-server-bundle (jusqu'à Symfony 4.4)  
`bin/console server:start`

# Symfony - Architecture du Squelette



- .env | .env.test : fichier de configuration de l'environnement courant (dev/test/staging/prod...)
  - bin : programmes en ligne de commande
  - composer.{json|lock} : gestion des dépendances via composer
  - config : configuration de l'application
  - migrations : fichiers générés par doctrine/migrations
  - phpunit.xml.dist : config de PHPUnit
  - public : racine du serveur web (y placer css, js, img...)
  - src : classes de l'application
  - templates : templates / vues Twig
  - tests : tests automatisés
  - translations : fichiers de traduction
  - symfony.lock : lockfile pour Symfony Recipes
  - var : données volatiles
  - vendor : dépendances
- └ .env
  - └ .env.test
  - └ .gitignore
  - └ bin
  - └ composer.json
  - └ composer.lock
  - └ config
  - └ migrations
  - └ phpunit.xml.dist
  - └ public
  - └ src
  - └ symfony.lock
  - └ templates
  - └ tests
  - └ translations
  - └ var
  - └ vendor

# Symfony - Fonctionnement (Symfony 4)



- Le point d'entrée de l'application, quelque soit l'URL est public/index.php
- C'est le Design Pattern Front Controller  
<https://martinfowler.com/eaaCatalog/frontController.html>

```
<?php

use App\Kernel;
use Symfony\Component\Dotenv\Dotenv;
use Symfony\Component\ErrorHandler\Debug;
use Symfony\Component\HttpFoundation\Request;

require dirname(__DIR__).'/vendor/autoload.php';

(new Dotenv())->bootEnv(dirname(__DIR__).'/ .env');

if ($_SERVER['APP_DEBUG']) {
    umask(0000);

    Debug::enable();
}

$kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);
$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();
$kernel->terminate($request, $response);
```



# Symfony - Fonctionnement (Symfony 5)

- Le point d'entrée de l'application, quelque soit l'URL est public/index.php
- C'est le Design Pattern Front Controller  
<https://martinfowler.com/eaaCatalog/frontController.html>

```
<?php

use App\Kernel;

require_once dirname(__DIR__).'/vendor/autoload_runtime.php';

return function (array $context) {
    return new Kernel($context['APP_ENV'], (bool) $context['APP_DEBUG']);
};
```

# Symfony - Fonctionnement



- public/index.php va :
  - inclure l'autoloader de composer
  - charger la config présente dans les fichiers .env
  - activer le mode debug si nécessaire (en dev)
  - instancier la classe Kernel (le noyau de Symfony)
  - créer un objet Request
  - demander à symfony de créer la réponse associer à cette requête
  - retourner la réponse au client



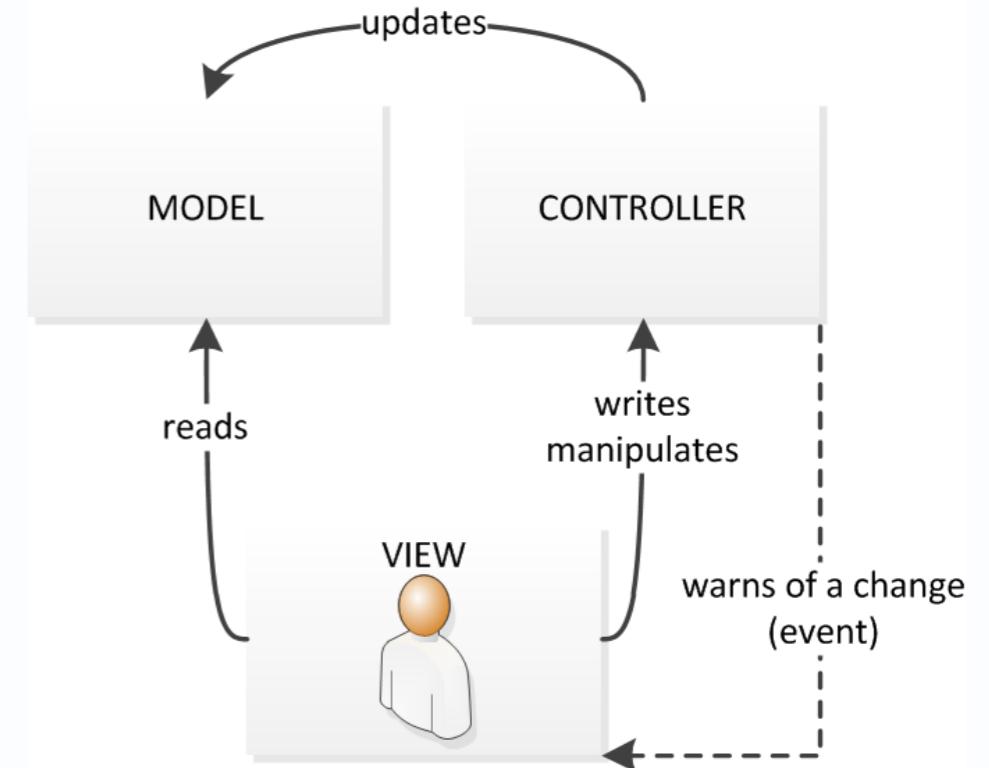
# Symfony - MVC

- Définition  
L'architecture MVC est un Design Pattern apparu en Smalltalk et très répandu dans les frameworks web
- Objectif  
L'objectif est de séparer les responsabilités de 3 types de composants : le Modèle (Model), la Vue (View), le Contrôleur (Controller)
- Documentation :  
<http://martinfowler.com/eaaCatalog/modelViewController.html>  
<http://martinfowler.com/eaaDev/uiArchs.html>  
<http://fr.wikipedia.org/wiki/Modèle-vue-contrôleur>



# Symfony - MVC

- Modèle  
Données, accès aux données, validation
- Vue  
Rendu. Se limiter à :
  - affichage de variable
  - bloc conditionnels if .. else if .. else (ex : afficher ou non message d'erreur, menu qui dépend d'une authentification)
  - boucles foreach (uniquement foreach, ce qui impose d'avoir trié/filtré au préalable)
  - appel à des fonctions de filtrage, de formatage, de rendu (parfois appelées aides de vues)
- Contrôleur  
Analyse de la requête, interrogation du Modèle, transmission des données à la vue, gestion des erreurs, des redirections (tout ce qui est lié à HTTP)...



# Symfony - Binaires



- Par défaut dans le dossier bin on retrouve 2 binaires :
  - console : un programme qui permet de générer du code, lancer un serveur de dev, vider les caches, débogguer certains aspects du framework...
  - phpunit : une version améliorée de phpunit qui liste notamment les messages de dépréciations
- Pour interagir avec ces programmes
  - se mettre à la racine du projet
  - sur Mac/Linux : bin/console
  - sur Windows : php bin\console
  - avec Symfony CLI : symfony console

```
bin
└── console
└── phpunit
```

# Symfony - Binaires



```
→ hello-symfony bin/console  
Symfony 5.2.3 (env: dev, debug: true)
```

Usage:

```
command [options] [arguments]
```

Options:

-h, --help	Display help for the given command. When no command is given display help for the list command
-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.
-vvvvv, --verbose	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Available commands:

about	Displays information about the current project
help	Displays help for a command
list	Lists commands
assets	
assets:install	Installs bundles web assets under a public directory
cache	
cache:clear	Clears the cache
cache:pool:clear	Clears cache pools
cache:pool:delete	Deletes an item from a cache pool
cache:pool:list	List available cache pools
cache:pool:prune	Prunes cache pools
cache:warmup	Warms up an empty cache

# Symfony - Binaires



config	
config:dump-reference	Dumps the default configuration for an extension
dbal	
dbal:run-sql	Executes arbitrary SQL directly from the command line.
debug	
debug:autowiring	Lists classes/interfaces you can use for autowiring
debug:config	Dumps the current configuration for an extension
debug:container	Displays current services for an application
debug:event-dispatcher	Displays configured listeners for an application
debug:form	Displays form type information
debug:router	Displays current routes for an application
debug:translation	Displays translation messages information
debug:twig	Shows a list of twig functions, filters, globals and tests
debug:validator	Displays validation constraints for classes

# Symfony - Binaires



doctrine	
doctrine:cache:clear-collection-region	Clear a second-level cache collection region
doctrine:cache:clear-entity-region	Clear a second-level cache entity region
doctrine:cache:clear-metadata	Clears all metadata cache for an entity manager
doctrine:cache:clear-query	Clears all query cache for an entity manager
doctrine:cache:clear-query-region	Clear a second-level cache query region
doctrine:cache:clear-result	Clears result cache for an entity manager
doctrine:database:create	Creates the configured database
doctrine:database:drop	Drops the configured database
doctrine:database:import	Import SQL file(s) directly to Database.
doctrine:ensure-production-settings	Verify that Doctrine is properly configured for a production environment
doctrine:mapping:convert	[orm:convert:mapping] Convert mapping information between supported formats
doctrine:mapping:import	Imports mapping information from an existing database
doctrine:mapping:info	
doctrine:migrations:current	[current] Outputs the current version
doctrine:migrations:diff	[diff] Generate a migration by comparing your current database to your mapping information.
doctrine:migrations:dump-schema	[dump-schema] Dump the schema for your database to a migration.
doctrine:migrations:execute	[execute] Execute one or more migration versions up or down manually.
doctrine:migrations:generate	[generate] Generate a blank migration class.
doctrine:migrations:latest	[latest] Outputs the latest version
doctrine:migrations:list	[list-migrations] Display a list of all available migrations and their status.
doctrine:migrations:migrate	[migrate] Execute a migration to a specified version or the latest available version.
doctrine:migrations:rollup	[rollup] Rollup migrations by deleting all tracked versions and insert the one version that exists.
doctrine:migrations:status	[status] View the status of a set of migrations.
doctrine:migrations:sync-metadata-storage	[sync-metadata-storage] Ensures that the metadata storage is at the latest version.
doctrine:migrations:up-to-date	[up-to-date] Tells you if your schema is up-to-date.
doctrine:migrations:version	[version] Manually add and delete migration versions from the version table.
doctrine:query:dql	Executes arbitrary DQL directly from the command line
doctrine:query:sql	Executes arbitrary SQL directly from the command line.
doctrine:schema:create	Executes (or dumps) the SQL needed to generate the database schema
doctrine:schema:drop	Executes (or dumps) the SQL needed to drop the current database schema
doctrine:schema:update	Executes (or dumps) the SQL needed to update the database schema to match the current mapping metadata
doctrine:schema:validate	Validate the mapping files

# Symfony - Binaires



lint	
lint:container declarations	Ensures that arguments injected into services match type
lint:twig	Lints a template and outputs encountered errors
lint:xliff	Lints a XLIFF file and outputs encountered errors
lint:yaml	Lints a file and outputs encountered errors
make	
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:docker:database	Adds a database container to your docker-compose.yaml file
make:entity	Creates or updates a Doctrine entity class, and optionally an API Platform resource
API Platform resource	
make:fixtures	Creates a new class to load Doctrine fixtures
make:form	Creates a new form class
make:message	Creates a new message and handler
make:messenger-middleware	Creates a new messenger middleware
make:migration	Creates a new migration based on database changes
make:registration-form	Creates a new registration form system
make:reset-password	Create controller, entity, and repositories for use with symfonycasts/reset-password-bundle
symfonycasts/reset-password-bundle	
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:test	[make:unit-test make:functional-test] Creates a new test class
make:twig-extension	Creates a new Twig extension 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

# Symfony - Binaires



router	
router:match	Helps debug routes by simulating a path info match
secrets	
secrets:decrypt-to-local	Decrypts all secrets and stores them in the local vault
secrets:encrypt-from-local	Encrypts all local secrets to the vault
secrets:generate-keys	Generates new encryption keys
secrets:list	Lists all secrets
secrets:remove	Removes a secret from the vault
secrets:set	Sets a secret in the vault
security	
security:encode-password	Encodes a password
server	
server:dump	Starts a dump server that collects and displays dumps in a single place
single place	
server:log	Starts a log server that displays logs in real time
translation	
translation:update	Updates the translation file



# Symfony - Binaires

- Certaines commandes sont interactives :

```
→ hello-symfony bin/console make:entity
```

Class name of the entity to create or update (e.g. OrangePizza):

```
> Contact
```

created: src/Entity/Contact.php

created: src/Repository/ContactRepository.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):

```
> firstName
```

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

```
> string
```

Field length [255]:

```
> 40
```



# Symfony - Binaires

- S'il n'y a pas ambiguïté on peut ne passer que les premières lettres de chaque mots :
- Exemple :
  - bin/console make:entity
  - bin/console m:e

```
→ hello-symfony bin/console m:e

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

Your entity already exists! So let's add some new fields!

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

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

Field length [255]:
> 40
```



# Symfony - Cache

- Pour optimiser les performances, Symfony va mettre toute sorte de choses en caches :
  - templates Twig compilés
  - annotations Doctrine
  - ...
- Vider le cache de dev :  
`bin/console cache:clear`
- Vider le cache de prod :  
`bin/console cache:clear --env=prod`



**formation.tech**

# Symfony Routing

# Symfony Routing - Contrôleur



- Un contrôleur est une fonction qui lit les données présentes dans un objet Request et retourne un objet Response
- La réponse peut être une page HTML, du JSON, du XML, un fichier à télécharger, une redirection, une erreur 404 ou tout autre chose
- Le contrôleur appelle le code du Modèle nécessaire pour faire le rendu d'une page web
- En résumé, sa responsabilité :
  - traiter les problématiques HTTP
  - appeler les méthodes du Modèle
  - retourner une réponse (éventuellement via une Vue)

# Symfony Routing - Contrôleur



- Les contrôleurs sont regroupées dans des classes, appelées `ActionController`
- Le but est d'y regrouper les contrôleurs et donc les pages par thème (souvent par entités), de sorte à ce que le contrôleur dépende des mêmes classes Modèle
- Le contrôleur doit retourner une instance de la classe `Response` (de `HttpFoundation`)

```
<?php

namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class HelloController
{
    /**
     * @Route("/hello")
     */
    public function hello()
    {
        return new Response('Hello');
    }
}
```

# Symfony Routing - Route



- Il faut ensuite configurer une route, c'est à dire la configuration permettant d'accéder au contrôleur, à minima quelle URL
- Les routes peuvent se configurer aux formats : Annotations, YAML, XML, code PHP
- Le squelette et la plupart des applications utilisent les annotations
- Dans le squelette c'est la méthode configureRoutes qui fait ce choix :

```
protected function configureRoutes(RoutingConfigurator $routes): void
{
    $routes->import('../config/{routes}/'.$this->environment.'/*.yaml');
    $routes->import('../config/{routes}/*.yaml');

    if (is_file(__DIR__).'./config/routes.yaml')) {
        $routes->import('../config/routes.yaml');
    } elseif (is_file($path = __DIR__).'./config/routes.php')) {
        (require $path)($routes->withPath($path), $this);
    }
}
```



# Symfony Routing - Route

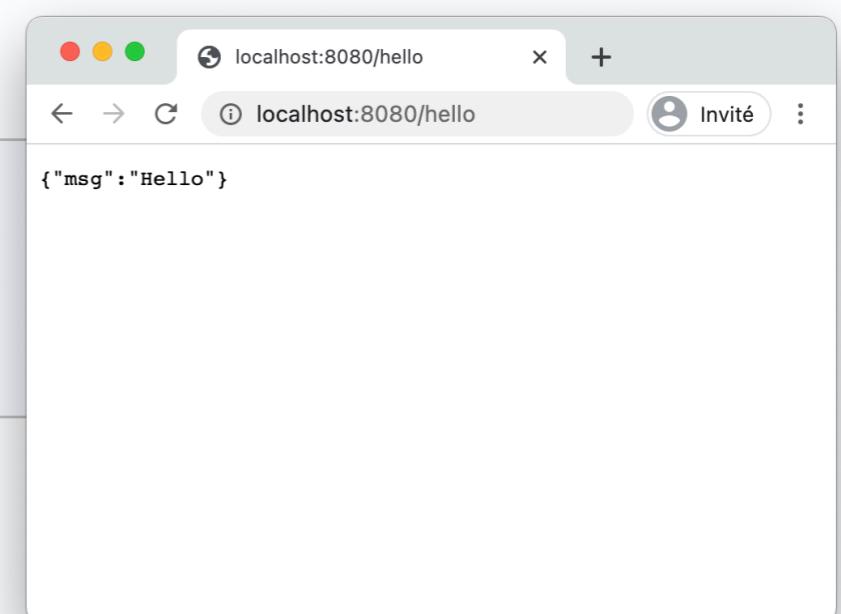
- Les réponses peuvent être configurées

[https://symfony.com/doc/current/components/http\\_foundation.html#response](https://symfony.com/doc/current/components/http_foundation.html#response)

```
public function hello()
{
    $response = new Response();
    $response->statusCode(200)
        ->setContent("Hello")
        ->headers->set('Content-type', 'text/plain');
    return $response;
}
```

- Pour retourner une réponse en JSON, on utilise la classe JsonResponse, le content-type sera automatiquement mis à application/json

```
/** @Route("/hello") */
public function hello()
{
    return new JsonResponse(['msg' => 'Hello']);
}
```

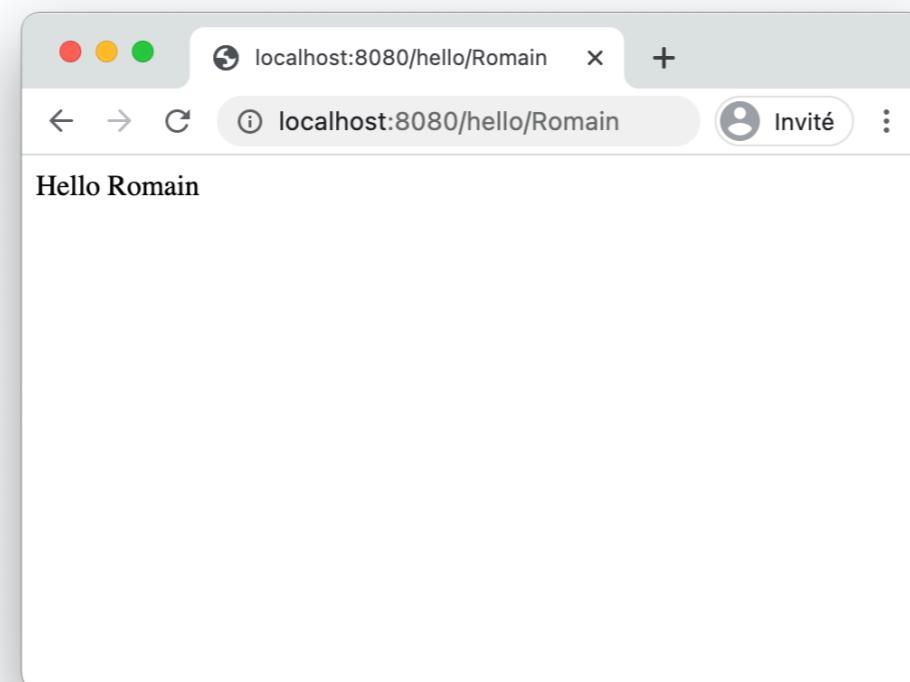




# Symfony Routing - Paramètres

- Une route permet de définir des paramètres
- Le paramètre est défini avec des accolades
- Le contrôleur reçoit la valeur dans un paramètre d'entrée du même nom

```
/** @Route("/hello/{name}") */
public function hello($name)
{
    return new Response("Hello $name");
}
```





# Symfony Routing - Paramètres spéciaux

- Certains paramètres ont des noms spéciaux :
  - `_controller` : permet de déterminer le contrôleur à utiliser (lorsqu'on n'utilise pas les annotations)
  - `_format` : permet de définir le type de réponse (JSON, XML, HTML...)
  - `_fragment` : permet de définir le fragment (ce qui suit le # dans une URL, ex : /settings#password)
  - `_locale` : permet de définir la locale (fr, en, es...)



# Symfony Routing - Paramètres

- Les routes peuvent définir :
  - un nom (utilisable pour générer un lien)
  - une liste de méthodes HTTP
  - un pattern pour les paramètres
  - un nom d'hôte
  - un schéma (HTTP/HTTPS...)
  - des conditions
  - des valeurs par défaut
  - des chemins internationnalisés

```
/**  
 * @Route(  
 *     "/hello/{name}",  
 *     name="hello",  
 *     methods={"GET"},  
 *     requirements={"name": "[a-zA-Z]+"}  
 */
```



# Symfony Routing - Paramètres

- En définissant des routes on peut avoir un conflit
  - De nom, 2 routes avec le même nom, la deuxième écrase la première
  - D'URL, 2 routes avec la même URL, Symfony s'arrête à la première
- Par défaut les routes ont un nom automatique : app\_[controller]\_[method]
- Pour débugger on peut utiliser la commande :  
`bin/console debug:router`
- Pour les routes de prod seules :  
`bin/console debug:router --env prod`

```
→ hello-symfony bin/console debug:router --env prod
-----
Name          Method  Scheme  Host   Path
-----
hello         GET     ANY     ANY    /hello/{name}
app_user_register ANY    ANY     ANY    /user/register
app_user_login  ANY    ANY     ANY    /user/login
-----
```



# Symfony Routing - Préfixes

- › Il est possible de préfixer l'ensemble des routes d'une classe
- › L'annotation avec le préfixe se définit au dessus de la classe
- › Ici : /user/register et /user/login

```
<?php

namespace App\Controller;

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

/** @Route("/user") */
class UserController extends AbstractController
{
    /** @Route("/register") */
    public function register()
    {
        return $this->json(['msg' => "Register"]);
    }

    /** @Route("/login") */
    public function login()
    {
        return $this->json(['msg' => "Login"]);
    }
}
```

# Symfony Routing - AbstractController



- La classe `AbstractController` contient un certain nombre d'utilitaires pour simplifier l'écriture du contrôleur

```
<?php

namespace App\Controller;

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

class HelloController extends AbstractController
{
    /**
     * @Route("/hello")
     */
    public function hello()
    {
        return $this->json(['msg' => 'Hello']);
    }
}
```

- Pour créer un contrôleur plus rapidement on peut utiliser la commande  
`bin/console make:controller`



**formation.tech**

# Twig



# Twig - Introduction

- Twig est un moteur de template développé par les auteurs de Symfony
- Il est largement inspiré des moteurs de templates Python Django Template Engine et Jinja
- Le code du template va être compilé en PHP
- La sécurité est amélioré car le moteur va échapper automatiquement les entrées
- <https://twig.symfony.com/>



# Twig - Rendu

- Rendu d'un template Twig

```
/**
 * @Route("/product", name="product")
 */
public function index(): Response
{
    return $this->render('product/index.html.twig', [
        'title' => 'Products list',
    ]);
}
```

- Les templates se trouvent dans le répertoire templates
- Pour passer des paramètres à un templates on passe un tableau associatif en 2e paramètre de render



# Twig - Rendu

- › Pour remplacer le echo, on utilise les {{ }}

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

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

{% block body %}
    <h1>{{ title }}</h1>
{% endblock %}
```

- › Pour factoriser le contenu commun d'une page on utilise extends et les blocks  
Pattern Two Step View : <https://martinfowler.com/eaaCatalog/twoStepView.html>

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Welcome!{% endblock %}</title>

        {% block stylesheets %}{% endblock %}

        {% block javascripts %}{% endblock %}
    </head>
    <body>
        {% block body %}{% endblock %}
    </body>
</html>
```



# Twig - Rendu

- › Un if

```
{% if is_granted('IS_AUTHENTICATED_FULLY') %}  
    <p>Vous êtes connecté</p>  
{% endif %}
```

- › Une boucle

```
<ul>  
    {% for p in products %}  
        <li>{{ p.name }}</li>  
    {% endfor %}  
</ul>
```

- › Lorsqu'on utilise . (ici dans p.name), Twig va chercher automatiquement la clés du tableau associatif ou le getter si p est un objet
- › Un commentaire

```
{# Commentaire #}
```



# Twig - Rendu

- Filtre

```
{ { title | upper } }
```

- Fonction

```
<a href=" { { path('app_user_login') } } >Login</a>
```

- Référence

<https://twig.symfony.com/doc/3.x/functions/index.html>

<https://twig.symfony.com/doc/3.x/filters/index.html>

[https://symfony.com/doc/current/reference/twig\\_reference.html](https://symfony.com/doc/current/reference/twig_reference.html)



**formation.tech**

# Symfony DependencyInjection

# Symfony DI - Services



- Dans une application il arrive fréquemment que l'on repose sur des classes utilitaires, par exemple :
  - Envoi d'un email
  - Srialisation d'un objet
  - Connexion à la base de données
  - ...
- On appelle ces objets des services (ils nous rendent des services)
- Par opposition, les autres objets qui contiennent nos données sont des Value Objects (Entity, DateTime, ArrayCollection...)

# Symfony DI - Services



- Les services peuvent être configurées différemment
  - Génération d'un mot de passe plus ou moins sécurisé
  - Paramètres de connexion à une base de données
  - ...
- Selon l'environnement
  - En dev, on logue plus d'informations qu'en prod
  - Plus de cache sur la base de données en prod
  - ...
- Ils peuvent évoluer
  - Migration d'une base MySQL vers une base MongoDB
  - Envoi d'un mail en utilisant un API REST plutôt qu'un serveur SMTP
  - ...

# Symfony DI - Services



- Pour permettre de faciliter leur changement dans l'application il faut :
  - qu'ils soient hautement configurables via un fichier de configuration
  - qu'ils soient eux même créés à partir de la configuration
- Autrement dit : il faut faire disparaître les 'new' dans votre code
- Le conteneur de Symfony est là pour ça



# Symfony DI - Conteneur

- Qu'est ce qu'un conteneur ?
  - On peut voir un conteneur comme un annuaire d'objets
  - C'est en quelque sorte un tableau associatif dans lequel seraient placés nos objets

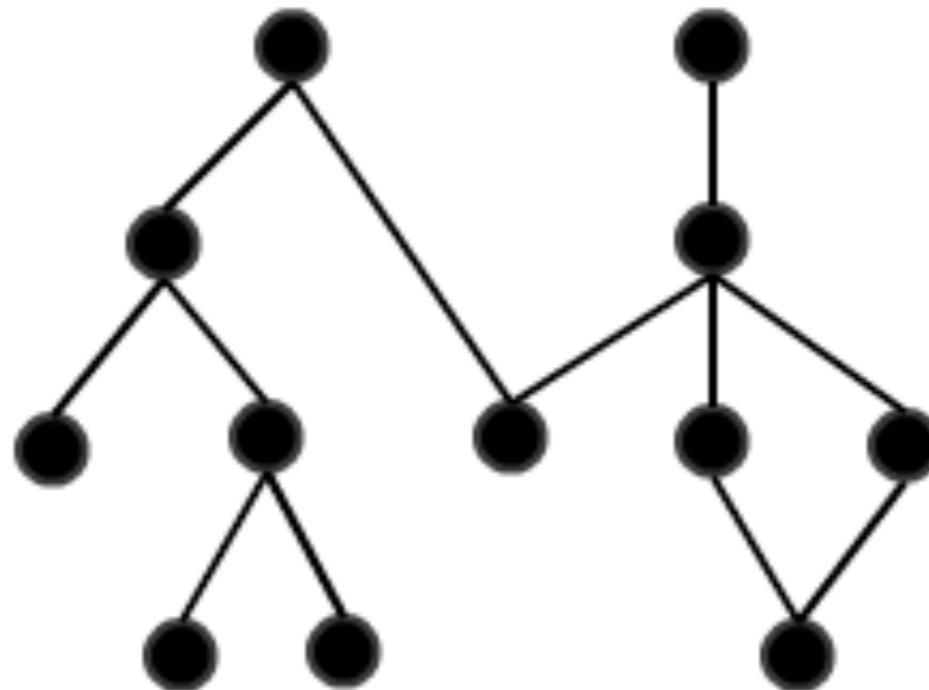
doctrine.orm.default_entity_manager	Doctrine\ORM\EntityManager
form.type.text	Symfony\Component\Form\Extension\Core\Type\TextType
logger	Symfony\Bridge\Monolog\Logger
mailer	Swift_Mailer

- En réalité les objets sont instanciés à la demande (sinon on parlerait de Registre)
- Pour y placer et récupérer nos objets on a des méthodes basiques : get(), set(), has()



# Symfony DI - Conteneur

- Dépendances
  - Les objets peuvent dépendre les uns des autres et leur dépendance devrait être injectée (voir injection de dépendance)
  - Symfony permet de configurer comment ses dépendances s'emboitent les unes dans les autres, jusqu'à former un graphe





# Symfony DI - Conteneur

- Exemple : ContactManager

```
<?php

namespace App\Manager;

class ContactManager
{
    public function getAll(): array
    {
        return [
            ['firstName' => 'A', 'lastName' => 'B'],
            ['firstName' => 'C', 'lastName' => 'D'],
        ];
    }
}
```



# Symfony DI - Conteneur

- Exemple : ContactController

```
<?php

namespace App\Controller;

use App\Manager>ContactManager;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\Routing\Annotation\Route;

/** @Route("/contacts") */
class ContactController
{
    /** @var ContactManager */
    protected $contactManager;

    public function __construct(ContactManager $contactManager)
    {
        $this->contactManager = $contactManager;
    }

    /** @Route("/") */
    public function list()
    {
        return new JsonResponse($this->contactManager->getAll());
    }
}
```



# Symfony DI - Conteneur

- Le container se configure dans le fichier config/services.yaml

```
services:  
    App\Manager>ContactManager:  
        class: App\Manager>ContactManager  
  
    App\Controller>ContactController:  
        class: App\Controller>ContactController  
        arguments: [ '@App\Manager>ContactManager' ]  
        tags: [ 'controller.service_arguments' ]
```

- Le code est ensuite compilé et mis en cache

```
class getContactControllerService extends App_KernelDevDebugContainer  
{  
    public static function do($container, $lazyLoad = true)  
    {  
        include_once \dirname(__DIR__, 4).'./src/Controller/ContactController.php';  
        include_once \dirname(__DIR__, 4).'./src/Manager/ContactManager.php';  
  
        return $container->services['App\\Controller\\ContactController'] = new  
        \App\Controller\ContactController(new \App\Manager>ContactManager());  
    }  
}
```

# Symfony DI - Conteneur



- Autowiring
  - Depuis Symfony 3.3, la plupart des classes du dossier src sont disponibles sous la forme de services, y compris les contrôleurs
  - Le conteneur se base sur le FQCN des classes pour savoir quoi injecter

```
services:  
    _defaults:  
        autowire: true  
        autoconfigure: true  
  
    App\  
        resource: '../src/'  
        exclude:  
            - '../src/DependencyInjection/'  
            - '../src/Entity/'  
            - '../src/Kernel.php'  
            - '../src/Tests/'  
  
    App\Controller\  
        resource: '../src/Controller/'  
        tags: ['controller.service_arguments']
```



# Symfony DI - Conteneur

- Autoconfigure
  - Certaines classes ont besoins d'être taguées, ce qui leur permet de recevoir du code automatiquement (console.command, form.type, twig.extension...)
  - Depuis Symfony 3.3 on peut auto configurer ces classes, le conteneur se base alors sur l'héritage ou les implémentations pour déterminer les tags à ajouter
- Debug
  - Pour connaitre la liste des services disponibles dans l'application :  
`bin/console debug:container`



**formation.tech**

# Doctrine\DBAL

# Doctrine\DBAL



- Doctrine DBAL (database abstraction & access layer) est un API similaire à PDO qui lui ajoute des fonctionnalités comme l'introspection et la manipulation de bases et de tables.
- Installation  
`composer require doctrine/dbal`
- SGBD supportés
  - MySQL
  - Oracle
  - Microsoft SQL Server
  - PostgreSQL
  - SAP Sybase SQL Anywhere SQLite
  - Drizzle



# Doctrine\DBAL

- 2 classes principales
  - Doctrine\DBAL\Connection  
Equivalent à PDO
  - Doctrine\DBAL\Statement  
Equivalent à PDOStatement

# Doctrine\DBAL



- Manipuler des bases de données

```
<?php

require_once 'vendor/autoload.php';

$connectionParams = [
    'user' => 'root',
    'password' => '',
    'host' => 'localhost',
    'driver' => 'pdo_mysql',
];

$conn = \Doctrine\DBAL\DriverManager::getConnection($connectionParams);

$sm = $conn->getSchemaManager();
$sm->createDatabase('tests_doctrine');

var_dump($sm->listDatabases());
```



- Créer des tables

```
$table = new \Doctrine\DBAL\Schema\Table('contact');
$table->addColumn('id', 'integer', ['autoincrement' => true]);
$table->addColumn('prenom', 'string', ['length' => 40]);
$table->addColumn('nom', 'string', ['length' => 40]);
$table->addColumn('email', 'string', ['length' => 80, 'notNull' => false]);
$table->addColumn('telephone', 'string', ['length' => 20, 'notNull' => false]);
$table->addColumn('date_naissance', 'date', ['notNull' => false]);
$table->setPrimaryKey(['id']);

// Pour afficher la requête
var_dump($sm->getDatabasePlatform()->getCreateTableSQL($table));

// Pour créer la table
$sm->createTable($table);
```



- Recherches
  - Avec un API style PDO

```
$stmt = $conn->executeQuery('SELECT * FROM contact');

var_dump($stmt->fetchAll(PDO::FETCH_ASSOC));
```

- Avec un Query Builder

```
$qb = $conn->createQueryBuilder();

$qb->select('prenom', 'nom')
    ->from('contact')
    ->orderBy('prenom', 'ASC');

$stmt = $qb->execute();

var_dump($stmt->fetchAll(PDO::FETCH_ASSOC));
```



- Différences entre 2 bases de données (scripts de migration)

```
$oldTable = new \Doctrine\DBAL\Schema\Table('contact');
$oldTable->addColumn('prenom', 'string');
$oldTable->addColumn('nom', 'string');
$oldSchema = new \Doctrine\DBAL\Schema\Schema([$oldTable]);

$newTable = new \Doctrine\DBAL\Schema\Table('contact');
$newTable->addColumn('prenom', 'string');
$newTable->addColumn('nom', 'string');
$newTable->addColumn('email', 'string', ['notNull' => false]);
$newTable->addColumn('telephone', 'string', ['notNull' => false]);
$newSchema = new \Doctrine\DBAL\Schema\Schema([$newTable]);

$dbPlatform = $conn->getDatabasePlatform();

$comparator = new \Doctrine\DBAL\Schema\Comparator();
$diff = $comparator->compare($oldSchema, $newSchema);
var_dump($diff->toSql($dbPlatform));

$comparator = new \Doctrine\DBAL\Schema\Comparator();
$diff = $comparator->compare($newSchema, $oldSchema);
var_dump($diff->toSql($dbPlatform));
```



**formation.tech**

# Doctrine\ORM

# Doctrine\ORM - Introduction



- Doctrine\ORM

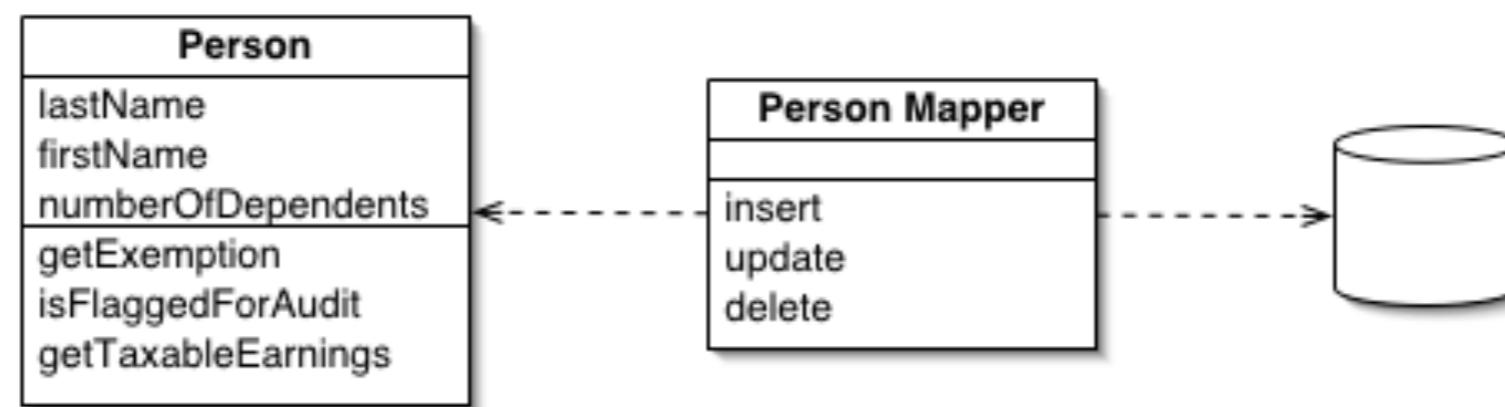
Doctrine ORM (object relational mapping) est un API permettant de faire traduire des manipulations d'objets en requêtes SQL.

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/>

- Data Mapper

Doctrine ORM est une implémentation du Design Pattern Data Mapper, c'est un composant qui permet de communiquer avec une base de données de manière objet. Il est inspiré de la bibliothèque Hibernate en Java.

<http://martinfowler.com/eaaCatalog/dataMapper.html>



- Installation

composer require doctrine/orm



# Doctrine\ORM - Principales classes

- `Doctrine\ORM\EntityManager`  
L'objet responsable de la persistence des entités.
- `Doctrine\ORM\EntityRepository`  
L'objet responsable de la récupération des entités.
- Portabilité vers Doctrine\ODM  
Pour être portable vers Doctrine\ODM, utiliser les interfaces  
`Doctrine\Common\Persistence\ObjectManager` et  
`Doctrine\Common\Persistence\ObjectRepository` dans les DocBlocks.

# Doctrine\ORM - Mapping



- › Configuration du Mapping

Le mapping est la traduction entre le monde objet et le monde de la base de données.

- › Formats possibles

- Annotations
- XML
- YAML

# Doctrine\ORM - Mapping



```
<?php

namespace AddressBook\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Contact
 *
 * @ORM\Table(name="contact", uniqueConstraints={@ORM\UniqueConstraint(name="email_UNIQUE", columns={"email"})},
 * indexes={@ORM\Index(name="fk_contact_societe_idx", columns={"societe_id"}), @ORM\Index(name="fk_contact_membre1_idx",
 * columns={"membre_id"})})
 * @ORM\Entity
 */
class Contact
{



}
```

- **@ORM\Entity**  
Pour déclarer la classe persistente
- **@ORM\Table(name="contact")**  
Pour déclarer à quelle table est associée cette entité  
Permet également de définir des index et contrainte unique.

# Doctrine\ORM - Mapping



- Pour lier une propriété à une colonne, attributs name id le nom de colonne est différent, length taille, nullable (true/false), type parmi :
  - **string** (string <> VARCHAR)
  - **integer** (int <> INT)
  - **smallint** (int <> SMALLINT)
  - **bigint** (string <> BIGINT)
  - **boolean** (boolean)
  - **decimal** (float <> DECIMAL avec des options precision et scale)
  - **date** (\DateTime <> DATETIME)
  - **time** (\DateTime <> TIME)
  - **datetime** (\DateTime <> DATETIME/TIMESTAMP)
  - **text** (string <> TEXT), object (serialize(object) <> unserialize(TEXT))
  - **array** (serialize(object) <> unserialize(TEXT))
  - **float** (float <> FLOAT ( séparateur décimale .))

# Doctrine\ORM - Mapping



- Il est également possible de définir ses propres types (peut-être utiliser pour les ENUM) :

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/cookbook/mysql Enums.html>



# Doctrine\ORM - Mapping

- `@ORM\Id`  
Si la colonne est la primaire.
- `@ORM\GeneratedValue(strategy="IDENTITY")`  
Si la clé primaire est générée par le SGBDR (strategy="IDENTITY" pour MySQL/SQLite/MSSQL), (strategy="SEQUENCE" pour PostgreSQL/Oracle), (strategy="AUTO" pour être portable)



# Doctrine\ORM - Mapping

- Exemple :

```
/**  
 * @var integer  
 *  
 * @ORM\Column(name="id", type="integer", nullable=false)  
 * @ORM\Id  
 * @ORM\GeneratedValue(strategy="IDENTITY")  
 */  
private $id;  
  
/**  
 * @var string  
 *  
 * @ORM\Column(name="prenom", type="string", length=45, nullable=false)  
 */  
private $prenom;  
  
/**  
 * @var \DateTime  
 *  
 * @ORM\Column(name="date_naissance", type="date", nullable=true)  
 */  
private $dateNaissance;  
  
/**  
 * @var int  
 *  
 * @ORM\Column(name="taille", type="integer", nullable=true)  
 */  
private $taille;
```

# Doctrine\ORM - Associations



- ▶ Associations

Doctrine ORM permet de définir directement les associations entre les entités et s'occupera de faire la plupart des jointures et insertions multiples automatiquement.

- OneToOne

Relation 1..1

- OneToMany

Relation 1..n du côté 1

- ManyToOne

Relation 1..n du côté n (du côté de la clé étrangère)

- ManyToMany

Relation n..m

- ▶ Unidirectionnelle / Bidirectionnelle

La relation peut-être unidirectionnelle (une entité en connaît une autre mais l'inverse n'est pas vrai) ou bidirectionnelle (chaque entité connaît l'autre).

Les relations bidirectionnelles doivent déclarer la relation opposées (`mappedBy`/`inversedBy`).

# Doctrine\ORM



```
class Contact
{
    /**
     * @var \Application\Entity\Membre
     *
     * @ORM\OneToOne(targetEntity="AddressBook\Entity\Membre")
     * @ORM\JoinColumns({
     *     @ORM\JoinColumn(name="membre_id", referencedColumnName="id")
     * })
     */
    private $membre;

    /**
     * @var \Application\Entity\Societe
     *
     * @ORM\ManyToOne(targetEntity="Application\Entity\Societe", inversedBy="contacts")
     * @ORM\JoinColumns({
     *     @ORM\JoinColumn(name="societe_id", referencedColumnName="id")
     * })
     */
    private $societe;

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\ManyToMany(targetEntity="Application\Entity\Association",
     * @ORM\JoinTable(name="adhésion",
     *     joinColumns={
     *         @ORM\JoinColumn(name="contact_id", referencedColumnName="id")
     *     },
     *     inverseJoinColumns={
     *         @ORM\JoinColumn(name="association_id", referencedColumnName="id")
     *     }
     * )
     */
    private $associations;
}
```

# Doctrine\ORM



- Exemple AddressBook\Entity\Societe :

```
class Societe
{
    // ...

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\OneToMany(targetEntity="Application\Entity>Contact", mappedBy="societe")
     */
    private $contacts;

    /**
     * Constructor
     */
    public function __construct()
    {
        $this->contacts = new \Doctrine\Common\Collections\ArrayCollection();
    }
}
```

# Doctrine\ORM



- bootstrap.php  
Création de l'EntityManager (\$isDevMode pour désactiver le cache)

```
<?php
// bootstrap.php
require_once "vendor/autoload.php";

// Create a simple "default" Doctrine ORM configuration for XML Mapping
$isDevMode = true;
$config =
\Doctrine\ORM\Tools\Setup::createAnnotationMetadataConfiguration(array(__DIR__."/src"),
$isDevMode);
// or if you prefer yaml or annotations
// $config = Setup::createXMLMetadataConfiguration(array(__DIR__."/config/xml"),
// $isDevMode);
// $config = Setup::createYAMLMetadataConfiguration(array(__DIR__."/config/yaml"),
// $isDevMode);

// database configuration parameters
$conn = array(
    'driver' => 'pdo_sqlite',
    'path' => __DIR__ . '/db.sqlite',
);

// obtaining the entity manager
$entityManager = \Doctrine\ORM\EntityManager::create($conn, $config);
```



- **cli-config.php**

Pour pouvoir utiliser les binaires de Doctrine\ORM il faut créer un fichier cli-config à la racine du projet ou dans un répertoire config.

```
<?php
// cli-config.php
require_once "bootstrap.php";

$helperSet = new \Symfony\Component\Console\Helper\HelperSet(array(
    'em' => new \Doctrine\ORM\Tools\Console\Helper\EntityManagerHelper($entityManager),
    'db' => new
\Doctrine\DBAL\Tools\Console\Helper\ConnectionHelper(\Doctrine\DBAL\DriverManager::getConnection($conn))
));

return $helperSet;
```

# Doctrine\ORM



- ▶ Insertions

```
<?php
// create_user.php
require_once "bootstrap.php";

$user = new User();
$user->setName('Romain');

$entityManager->persist($user);
$entityManager->flush();

echo "Created User with ID " . $user->getId() . "\n";
```

# Doctrine\ORM



- Recherches

```
<?php
// list_bugs.php
require_once "bootstrap.php";

$dql = "SELECT b, e, r FROM Bug b JOIN b.engineer e JOIN b.reporter r ORDER BY b.created DESC";

$query = $entityManager->createQuery($dql);
$query->setMaxResults(30);
$bugs = $query->getResult();

foreach($bugs AS $bug) {
    echo $bug->getDescription()." - ".$bug->getCreated()->format('d.m.Y')."\n";
    echo "    Reported by: ".$bug->getReporter()->getName()."\n";
    echo "    Assigned to: ".$bug->getEngineer()->getName()."\n";
    foreach($bug->getProducts() AS $product) {
        echo "        Platform: ".$product->getName()."\n";
    }
    echo "\n";
}
```



# Doctrine - Requêtes « complexes »

- extends Doctrine\ORM\EntityRepository

Parfois les requêtes ne peuvent pas s'écrire où ne sont pas bien optimisés, il faut alors créer une classe Repository qui héritera de Doctrine\ORM\EntityRepository

Les requêtes peuvent alors s'écrire :

- En SQL

Il faudra alors expliquer à Doctrine comment cette requête se traduit en entité

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/native-sql.html>

- Avec le QueryBuilder

Equivalent de Zend\Db\Sql

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html>

- En DQL

Language propre à Doctrine proche de SQL mais manipulant des entités.

- Crédit du Repository

Modifier l'annotation @ORM\Entity d'une entité pour :

@ORM\Entity(repositoryClass="AddressBook\Entity\Repository>ContactRepository")

Exécuter la commande :

vendor/bin/doctrine-module orm:generate:repositories module/AddressBook/src



# Doctrine - Requêtes « complexes »

- Exemple
- Dans showAction du contrôleur Contact, nous requérons une entité par sa clé primaire. Dans la vue nous demandons d'accéder à sa Société liée.  
Par défaut Doctrine fait 2 requêtes SQL, nous aimerais en faire 1 seule requête avec une jointure.

```
// AddressBook\Controller>ContactController

public function showAction()
{
    $id = $this->params("id");

    $contact = $this->getRepository()->find($id);

    return new ViewModel(
        array(
            "contact" => $contact
        )
    );
}
```

```
<!-- view/address-book/contact/show.phtml -->

<?=$this->escapeHtml($contact->getSociete()->getNom())?>
```

```
» DoctrineORMModule
» Queries for doctrine.sql_logger_collector.orm_default

SQL:
SELECT t0.id AS id1, t0.prenom AS prenom2, t0.nom AS nom3, t0.telephone AS telephone4, t0.email AS email5, t0.membre_id AS membre_id6, t0.societe_id AS societe_id7 FROM contact t0 WHERE t0.id = ?

Params:      0 => string '1' (length=1)
Types:       0 => string 'integer' (length=7)
Time:        0.00042605400085449

SQL:
SELECT t0.id AS id1, t0.nom AS nom2, t0.ville AS ville3 FROM societe t0
WHERE t0.id = ?

Params:      0 => int 3

2 queries in 876.19 µs   4 mappings
```



# Doctrine - Requêtes « complexes »

- Exemple

- Avec le Repository, on obtient une seule requête (temps d'exécution divisé par 2).

```
// AddressBook\Entity\Repository>ContactRepository

class ContactRepository extends EntityRepository
{
    public function findWithSociete($id)
    {
        $dql = "SELECT c, s
                FROM AddressBook\Entity\Contact c
                LEFT JOIN c.societe s
                WHERE c.id = :id";

        return $this->getEntityManager()->createQuery($dql)
            ->setParameter("id", $id)
            ->getSingleResult();
    }
}
```

```
// AddressBook\Controller>ContactController

public function showAction()
{
    $id = $this->params("id");

    $contact = $this->getRepository()->findWithSociete($id);

    return new ViewModel(
        array(
            "contact" => $contact
        )
    );
}
```

```
» DoctrineORMModule
» Queries for doctrine.sql_logger_collector.orm_default

SQL:
SELECT c0_.id AS id0, c0_.prenom AS prenom1, c0_.nom AS nom2,
c0_.telephone AS telephone3, c0_.email AS email4, s1_.id AS id5, s1_.nom AS
nom6, s1_.ville AS ville7, c0_.membre_id AS membre_id8, c0_.societe_id AS
societe_id9 FROM contact c0_ LEFT JOIN societe s1_ ON c0_.societe_id =
s1_.id WHERE c0_.id = ?

Params:          0 => string '1' (length=1)
Types:           0 => int 2
Time:            0.0004878044128418
1 queries in 487.80 µs   4 mappings
```



**formation.tech**

# Symfony Validation

# Symfony Validation - Introduction



- Symfony propose un composant pour valider des objets appelé `symfony/validator`
- Comme avec les routes et Doctrine, le composant accepte des configurations au format Annotation, YAML, XML, code PHP
- Les annotations se trouvent dans le namespace `Symfony\Component\Validator\Constraints`
- Un service implémentant `Symfony\Component\Validator\Validator\ValidatorInterface` permet de valider les entités

# Symfony Validation - Exemple



- En utilisant les annotations

```
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ORM\Entity(repositoryClass=ContactRepository::class)
 */
class Contact
{
    /**
     * @ORM\Column(length=40, nullable=false)
     * @Assert\NotBlank()
     * @Assert\Length(max=40)
     */
    protected $firstName;
```

- La liste des contraintes de validation disponibles nativement :  
<https://symfony.com/doc/current/validation.html#constraints>

# Symfony Validation - Exemple



- En utilisant les annotations

```
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ORM\Entity(repositoryClass=ContactRepository::class)
 */
class Contact
{
    /**
     * @ORM\Column(length=40, nullable=false)
     * @Assert\NotBlank()
     * @Assert\Length(max=40)
     */
    protected $firstName;
```

- La liste des contraintes de validation disponibles nativement :  
<https://symfony.com/doc/current/validation.html#constraints>



# Symfony Validation - Auto Mapping

- A partir de Symfony 4.3 les annotations Doctrine sont automatiquement validées selon les règles suivantes

Doctrine mapping	Automatic validation constraint
nullable=false	@Assert\NotNull
type=...	@Assert\Type(...)
unique=true	@UniqueEntity
length=...	@Assert\Length(...)

- On peut l'activer au niveau d'une entité avec l'annotation  
`@Assert\EnableAutoMapping()`
- Ou bien globalement via le fichier config/packages/validator.yaml



**formation.tech**

# Tests automatisés

# Tests automatisés



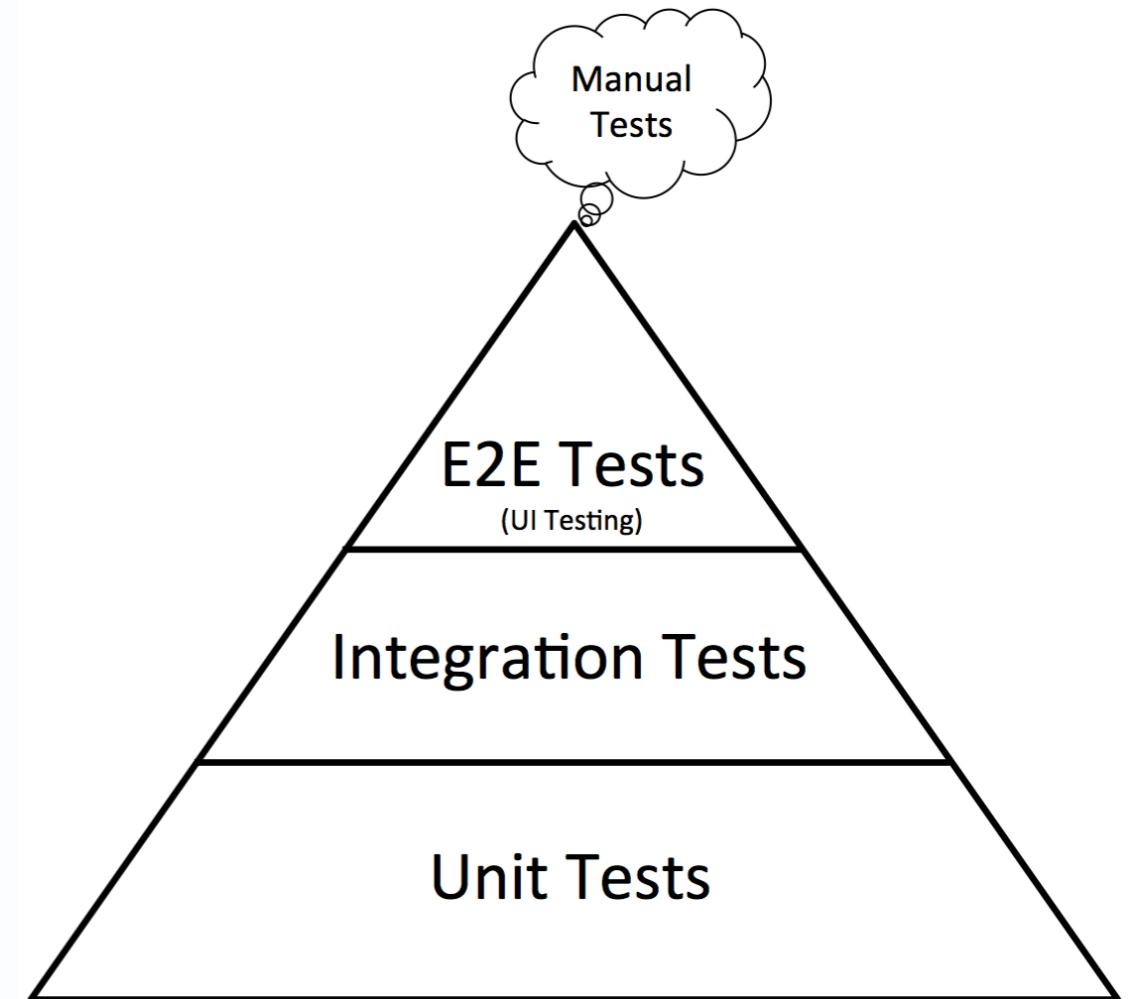
- Vérification manuelle
  - Ecrire une recette de tests et demander à une personne de la rejouer à des étapes clés (nouvelle version)
  - Ecrire le test sous la forme de code, et vérifier visuellement que les résultats attendus soit les bons
- Tests automatisés
  - Le test est codé, la vérification se fait dans un rapport
- Historique
  - sUnit en 1994 (SmallTalk), JUnit en 1997 (Java)
  - Les frameworks s'inspirant de jUnit sont catégorisés xUnit (PHPUnit, CUnit...)



# Pyramide des Tests

## ► Types de tests

- **Unitaire** : tests des méthodes d'une classe
- **Intégration** : teste l'intégration entre plusieurs classes
- **Fonctionnels** : teste l'application du point de vue du client (HTTP dans le cas du web)
- **End-to-End (E2E)** : teste l'application dans le client (y compris JavaScript, CSS...)





**formation.tech**

# PHPUnit

# PHPUnit - Introduction



- Crée en 2001 par Sebastian Bergmann
- Framework de tests de référence en PHP  
Utilisé, même étendu par Symfony et Zend Framework

- Documentation  
<https://phpunit.de/documentation.html>
- Open Source  
Licence BSD Modifiée
- Concurrents :  
atoum (FR), Behat (BDD), SimpleTest

The screenshot shows the homepage of Packagist.org, which is described as "The PHP Package Repository". The page features a search bar at the top with the placeholder "Search packages...". Below the search bar, there is a section titled "Popular Packages" displaying a list of seven packages:

Package	Description	Type	Downloads	Stars
<a href="#">psr/log</a>	Common interface for logging libraries	PHP	21 655 201	302
<a href="#">symfony/event-dispatcher</a>	Symfony EventDispatcher Component	PHP	19 207 164	125
<a href="#">symfony/console</a>	Symfony Console Component	PHP	17 596 105	310
<a href="#">symfony/yaml</a>	Symfony Yaml Component	PHP	15 412 245	221
<a href="#">monolog/monolog</a>	Sends your logs to files, sockets, inboxes, databases and various web services	PHP	21 646 693	4 312
<a href="#">phpunit/phpunit</a>	The PHP Unit Testing framework.	PHP	15 009 357	5 020
<a href="#">doctrine/inflector</a>	Common String Manipulations with regard to casing and singular/plural rules.	PHP	15 296 907	89

# PHPUnit - Installation globale



- ▶ **PHAR**
  - Dernière Version  
<https://phar.phpunit.de/phpunit.phar>
  - Version spécifique  
<https://phar.phpunit.de/phpunit-X.Y.Z.phar>
- ▶ **Composer**
  - Dernière Version  
composer global require phpunit/phpunit
  - Version spécifique  
composer global require phpunit/phpunit:5.0.\*
  - Penser à ajouter le répertoire bin global au PATH, sur UNIX :  
~/.composer/vendor/bin

# PHPUnit - Installation locale



## ▶ Composer

- Dernière Version  
    composer require phpunit/phpunit --dev
- Version spécifique  
    composer require phpunit/phpunit:5.0.\* --dev
- Ou en éditant directement le fichier composer.json puis composer update

```
{  
    "require-dev": {  
        "phpunit/phpunit": "5.1.*"  
    }  
}
```

- Exécution depuis la racine du projet:  
    ./vendor/bin/phpunit

# PHPUnit - Structure d'un test



- **Conventions**
  - Un test PHPUnit est une méthode dont le nom commence par test :  
`testMaFonction()`
  - Cette méthode se trouve dans une classe dont le nom se termine par Test et qui hérite de `\PHPUnit_Framework_TestCase` ou `\PHPUnit\Framework\TestCase`
- **Bonnes pratiques**
  - Ne pas hésiter à être le plus verbeux possible dans le nom des méthodes
  - L'arborescence du répertoire test correspond au répertoire src (ex : `src/MonNamespace/MaClasse.php` → `tests/MonNamespaceTest/MaClasseTest.php`)



# PHPUnit - Exemple

```
<?php

namespace FormationTechTest\Entity;

use FormationTech\Entity\CompteBancaire;

class CompteBancaireTest extends \PHPUnit_Framework_TestCase
{
    public function testCrediter()
    {
        $compte = new CompteBancaire(0);
        $compte->crediter(1000);
        $this->assertEquals(1000, $compte->getSolde());

        $compte->crediter(500);
        $this->assertEquals(1500, $compte->getSolde());
    }
}
```

```
|MBP-de-Romain:PrepaFormationPHPUnit roman$ ./vendor/bin/phpunit tests/Entity/CompteBancaireTest.php --colors
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

```
.
```

```
1 / 1 (100%)
```

```
Time: 39 ms, Memory: 1.50Mb
```

```
OK (1 test, 2 assertions)
```

# PHPUnit - Appels automatiques



- PHPUnit peut appeler des méthodes avant et après chaque test
  - `setUp`
  - `tearDown`
- Avant ou après chaque classe (méthodes statiques)
  - `setUpBeforeClass`
  - `tearDownAfterClass`



# PHPUnit - Ligne de commande

```
MBP-de-Romain:PrepaFormationPHPUnit roman$ ./vendor/bin/phpunit -h  
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

Usage: `phpunit [options] UnitTest [UnitTest.php]`  
`phpunit [options] <directory>`

## Code Coverage Options:

<code>--coverage-clover &lt;file&gt;</code>	Generate code coverage report in Clover XML format.
<code>--coverage-crap4j &lt;file&gt;</code>	Generate code coverage report in Crap4J XML format.
<code>--coverage-html &lt;dir&gt;</code>	Generate code coverage report in HTML format.
<code>--coverage-php &lt;file&gt;</code>	Export PHP_CodeCoverage object to file.
<code>--coverage-text=&lt;file&gt;</code>	Generate code coverage report in text format. Default: Standard output.
<code>--coverage-xml &lt;dir&gt;</code>	Generate code coverage report in PHPUnit XML format.
<code>--whitelist &lt;dir&gt;</code>	Whitelist <dir> for code coverage analysis.

## Logging Options:

<code>--log-junit &lt;file&gt;</code>	Log test execution in JUnit XML format to file.
<code>--log-tap &lt;file&gt;</code>	Log test execution in TAP format to file.
<code>--log-teamcity &lt;file&gt;</code>	Log test execution in TeamCity format to file.
<code>--log-json &lt;file&gt;</code>	Log test execution in JSON format.
<code>--testdox-html &lt;file&gt;</code>	Write agile documentation in HTML format to file.
<code>--testdox-text &lt;file&gt;</code>	Write agile documentation in Text format to file.
<code>--reverse-list</code>	Print defects in reverse order

# PHPUnit - Ligne de commande



## Test Selection Options:

--filter <pattern>	Filter which tests to run.
--testsuite <pattern>	Filter which testsuite to run.
--group ...	Only runs tests from the specified group(s).
--exclude-group ...	Exclude tests from the specified group(s).
--list-groups	List available test groups.
--test-suffix ...	Only search for test in files with specified suffix(es). Default: Test.php,.phpt

## Configuration Options:

--bootstrap <file>	A "bootstrap" PHP file that is run before the tests.
-c --configuration <file>	Read configuration from XML file.
--no-configuration	Ignore default configuration file (phpunit.xml).
--no-coverage	Ignore code coverage configuration.
--include-path <path(s)>	Prepend PHP's include_path with given path(s).
-d key[=value]	Sets a php.ini value.

## Miscellaneous Options:

-h --help	Prints this usage information.
--version	Prints the version and exits.
--atleast-version <min>	Checks that version is greater than min and exits.

# PHPUnit - Ligne de commande



## Test Execution Options:

--report-useless-tests	Be strict about tests that do not test anything.
--strict-coverage	Be strict about unintentionally covered code.
--strict-global-state	Be strict about changes to global state
--disallow-test-output	Be strict about output during tests.
--disallow-resource-usage	Be strict about resource usage during small tests.
--enforce-time-limit	Enforce time limit based on test size.
--disallow-todo-tests	Disallow @todo-annotated tests.
--process-isolation	Run each test in a separate PHP process.
--no-globals-backup	Do not backup and restore \$GLOBALS for each test.
--static-backup	Backup and restore static attributes for each test.
--colors=<flag>	Use colors in output ("never", "auto" or "always").
--columns <n>	Number of columns to use for progress output.
--columns max	Use maximum number of columns for progress output.
--stderr	Write to STDERR instead of STDOUT.
--stop-on-error	Stop execution upon first error.
--stop-on-failure	Stop execution upon first error or failure.
--stop-on-warning	Stop execution upon first warning.
--stop-on-risky	Stop execution upon first risky test.
--stop-on-skipped	Stop execution upon first skipped test.
--stop-on-incomplete	Stop execution upon first incomplete test.
-v -verbose	Output more verbose information.
--debug	Display debugging information during test execution.
--loader <loader>	TestSuiteLoader implementation to use.
--repeat <times>	Runs the test(s) repeatedly.
--tap	Report test execution progress in TAP format.
--teamcity	Report test execution progress in TeamCity format.
--testdox	Report test execution progress in TestDox format.
--printer <printer>	TestListener implementation to use.



# PHPUnit - phpunit.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit colors="true">

    <testsuites>
        <testsuite name="AllTests">
            <directory>tests/Mapper</directory>
        </testsuite>
    </testsuites>

    <filter>
        <blacklist>
            <directory suffix=".php"></directory>
            <file></file>
            <exclude>
                <directory suffix=".php"></directory>
                <file></file>
            </exclude>
        </blacklist>
        <whitelist processUncoveredFilesFromWhitelist="true">
            <directory suffix=".php">classes</directory>
            <file></file>
            <exclude>
                <directory suffix=".php"></directory>
                <file></file>
            </exclude>
        </whitelist>
    </filter>

    <logging>
        <log type="coverage-clover" target="logs/phpunit-coverage.xml"/>
        <log type="junit" target="logs/phpunit-log.xml" logIncompleteSkipped="false"/>
    </logging>

</phpunit>
```



- Un fichier de bootstrap peut être exécuté au démarrage de PHPUnit
- Intérêts :
  - Autochargement de classe (sauf si phpunit a été installé avec Composer et que l'autoloader est celui de composer)
  - Modification du include\_path
  - Chargement de fichiers de configuration

# PHPUnit - IDE



## ▶ PHPStorm

The screenshot shows the PHPStorm IDE interface with the following components:

- Project Structure:** Shows the project structure under "Tests > PrepaFormationPHPUnit > classes > Entity > Database.php".
- Code Editor:** Displays the `Database.php` file content:<?php  
namespace FormationTech\Entity;  
  
class Database  
{  
 protected \$name;  
  
 public function \_\_construct(\$name)  
 {  
 \$this->name = \$name;  
 }  
  
 public function getName()  
 {  
 return \$this->name;  
 }  
}
- Coverage PHPUnit:** A panel showing coverage statistics:

Element	Statistics, %
CompteBancaire.php	50% files, 18% lines in 'Entity'
Database.php	60% lines
- Run:** A "PHPUnit" run configuration is selected in the dropdown.
- Test Results:** A tree view showing test results for "AllTests" and "FormationTechTest\Mapper\DatabaseMapp". All tests passed in 400ms.
- Terminal:** Output of the PHPUnit run:

```
All 9 tests passed - 400ms  
/usr/local/bin/php -dxdebug.coverage_enable=1 /Users/romain/www/Learning/PHP/Tests/PrepaFormationPHPUnit/vendor/phpunit/phpunit/phpunit  
Testing started at 23:40 ...  
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.  
  
Time: 943 ms, Memory: 4.00Mb  
OK (9 tests, 18 assertions)  
Generating code coverage report in Clover XML format ... done  
Process finished with exit code 0
```
- Bottom Status Bar:** Shows "Tests Passed: 9 passed (2 minutes ago)" and the current time "19:2 LF UTF-8".



# PHPUnit - Intégration continue

- JUnit Plugin

The screenshot shows a Jenkins test results page for a build named "BuildApp #26". The URL is `localhost:8080/job/BuildApp/26/testReport/FormationTechTest/Mapper/DatabaseMapperTest`. The page title is "Test Result : FormationTechTest\Mapper\DatabaseMapperTest". It displays 9 test cases, all of which have passed. The test names and their details are listed in a table.

Test name	Duration	Status
<a href="#">testFindAllWithDummyProphecy</a>	0.1 sec	Passed
<a href="#">testFindAllWithFake</a>	12 ms	Passed
<a href="#">testFindAllWithMock</a>	15 ms	Passed
<a href="#">testFindAllWithMockProphecy</a>	52 ms	Passed
<a href="#">testFindAllWithMySQL</a>	33 ms	Passed
<a href="#">testFindAllWithSpyProphecy</a>	34 ms	Passed
<a href="#">testFindAllWithStubFromClass</a>	17 ms	Passed
<a href="#">testFindAllWithStubFromInterface</a>	53 ms	Passed
<a href="#">testFindAllWithStubProphecy</a>	43 ms	Passed

# PHPUnit - Intégration continue



- ▶ Clover PHP plugin

The screenshot shows a Jenkins build page for 'BuildApp' with build number '#26'. The main content is the 'Clover PHP Coverage Report [Jenkins]'. The report includes:

- Clover PHP Coverage Report**: A chart showing coverage metrics (method, statement, total) across two builds (#23 and #26). The total coverage is approximately 40%.
- Overall Coverage Summary**: A table showing project-level coverage details.

name	Total Coverage %	LOC	NCLOC	method, %	statements, %
Project	38.8%	177	177	41.2% (7/17)	38% (19/50)
- Coverage Breakdown by File**: A table showing coverage for specific files.

name	Total Coverage %	LOC	NCLOC	method, %	statements, %
/Users/romain/www/Learning/PHP/Tests/PrepaFormationPHPUnit/classes/Gateway/DatabaseGatewayInterface.php	- (0/0)	7	7	- (0/0)	- (0/0)
/Users/romain/www/Learning/PHP/Tests/PrepaFormationPHPUnit/classes/Writer/WriterInterface.php	- (0/0)	8	8	- (0/0)	- (0/0)
- Coverage Breakdown by Namespace**: A table showing coverage for specific namespaces.

name	Total Coverage %	LOC	NCLOC	method, %	statements, %
FormationTech\Entity	14.3%	45	45	16.7% (1/6)	13.3% (2/15)
FormationTech\Gateway	100%	36	36	100% (4/4)	100% (7/7)
FormationTech\Logger	0%	28	28	0% (0/2)	0% (0/9)
FormationTech\Mapper	100%	31	31	100% (2/2)	100% (10/10)
FormationTech\Writer	0%	22	22	0% (0/3)	0% (0/9)



**formation.tech**

# Assertions



# Assertions - Introduction

- Dans un framework xUnit, les assertions sont les méthodes qui vérifient qu'un résultat espéré corresponde au résultat attendu
- Le test échoue et s'arrête à la première assertion qui n'est pas vérifiée
- Bonnes pratiques :
  - Plusieurs assertions par test
  - Utiliser la méthode d'assertion la plus précise possible pour avoir un message d'erreur clair :  
Ex : `assertEmpty($tableau)`  
plutôt que `assertEquals(0, count($tableau))`
  - Si possible ajouter un message personnalisé



# Assertions - Basiques

- assertContains
- assertEquals
- assertFalse
- assertGreaterThan
- assertGreaterThanOrEqual
- assertInfinite
- assertInternalType
- assertLessThan
- assertLessThanOrEqual
- assertNaN
- assertRegExp
- assertSame
- assertStringEndsWith
- assertStringMatchesFormat
- assertStringStartsWith
- assertThat
- assertTrue



# Assertions - Tableaux

- `assertArrayHasKey`
- `assertArraySubset`
- `assertCount`
- `assertContains`
- `assertContainsOnly`
- `assertContainsOnlyInstancesOf`
- `assertEmpty`



# Assertions - Fichiers et Formats

- ▶ Fichiers
  - assertFileEquals
  - assertFileExists
  - assertStringEqualsFile
  - assertStringMatchesFormatFile
- ▶ XML
  - assertEqualsXMLStructure
  - assertXmlFileEqualsXmlFile
  - assertXmlStringEqualsXmlFile
  - assertXmlStringEqualsXmlString
- ▶ JSON
  - assertJsonFileEqualsJsonFile
  - assertJsonStringEqualsJsonFile
  - assertJsonStringEqualsJsonString



# Assertions - Classes et Objets

- assertClassHasAttribute
- assertClassHasStaticAttribute
- assertInstanceOf
- assertObjectHasAttribute
- assertNull



**formation.tech**

# Types de tests



# Types de tests - Test Unitaire

```
<?php

namespace FormationTech\Entity;

class CompteBancaire
{
    protected $solde;

    public function __construct($solde = 0)
    {
        $this->solde = (double) $solde;
    }

    public function getSolde()
    {
        return $this->solde;
    }

    public function debiter($montant)
    {
        $this->solde -= (double) $montant;
    }

    public function crediter($montant)
    {
        $this->solde += (double) $montant;
    }
}
```

```
<?php

namespace FormationTechTest\Entity;

use FormationTech\Entity\CompteBancaire;

class CompteBancaireTest extends \PHPUnit_Framework_TestCase
{
    public function testCrediter()
    {
        $compte = new CompteBancaire(0);
        $compte->crediter(1000);
        $this->assertEquals(1000, $compte->getSolde());

        $compte->crediter(500);
        $this->assertEquals(1500, $compte->getSolde());
    }
}
```

```
▼ └─ PrepaFormationPHPUnit
    └─ classes
        └─ Entity
            └─ CompteBancaire.php
    └─ tests
        └─ Entity
            └─ CompteBancaireTest.php
    └─ vendor
        └─ composer.json
        └─ composer.lock
```

```
[MBP-de-Romain:PrepaFormationPHPUnit roman$ ./vendor/bin/phpunit tests/Entity/CompteBancaireTest.php --colors
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

1 / 1 (100%)

Time: 39 ms, Memory: 1.50Mb

OK (1 test, 2 assertions)



# Types de tests - Test d'intégration

```
<?php

namespace FormationTech\Logger;

use FormationTech\Writer\WriterInterface;
use Psr\Log\LoggerInterface;
use Psr\Log\LoggerTrait;

class Logger implements LoggerInterface
{
    use LoggerTrait;

    protected $writer;

    public function __construct(WriterInterface $writer)
    {
        $this->writer = $writer;
    }

    public function log($level, $message, array $context = array())
    {
        $datetime = date('Y-m-d H:i:s');
        $logMessage = "[{$level}] - {$datetime} - {$message}";

        $this->writer->write($logMessage);
    }
}
```

```
<?php

namespace FormationTech\Writer;

class FileWriter implements WriterInterface
{
    protected $fic;

    public function __construct($filePath)
    {
        $this->fic = fopen($filePath, 'a');
    }

    public function write($message)
    {
        fwrite($this->fic, "$message\n");
    }

    public function __destruct()
    {
        fclose($this->fic);
    }
}
```

- Exemple de communication entre 2 classes :
  - Logger dépend de Writer (WriterInterface) et est compatible PSR-4
  - FileWriter implémente WriterInterface et sa méthode write



# Types de tests - Test d'intégration

```
<?php

namespace FormationTechTest\Logger;

use FormationTech\Logger\Logger;
use FormationTech\Writer\FileWriter;
use Psr\Log\LogLevel;

class LoggerTest extends \PHPUnit_Framework_TestCase
{
    public function testLogWithFileWriter()
    {
        $testFile = __DIR__ . '/../../tests.log';
        $fw = new FileWriter($testFile);
        $logger = new Logger($fw);

        $logger->log(LogLevel::NOTICE, 'Un message');
        $content = file_get_contents($testFile);

        $this->assertRegExp('/\[notice\] - \d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2} - Un message\n/' ,
$content);
    }
}
```

```
MBP-de-Romain:PrepaFormationPHPUnit roman$ ./vendor/bin/phpunit tests/Logger/LoggerTest.php --colors
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

.

1 / 1 (100%)

Time: 38 ms, Memory: 1.50Mb

OK (1 test, 1 assertion)



# Types de tests - Test fonctionnel

```
<?php
require_once __DIR__ . '/vendor/autoload.php';

$pdo = new \PDO('mysql:host=localhost', 'root', '');
$gateway = new \FormationTech\Gateway\DatabaseGateway($pdo);
$dbList = $gateway->listDbs();
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Database list</title>
    </head>
    <body>
        <h2>Database list</h2>
        <ul>
            <?php foreach ($dbList as $db) : ?>
            <li><?=htmlspecialchars($db)?></li>
            <?php endforeach; ?>
        </ul>
    </body>
</html>
```

- Démarrage du PHP Built-in Server  
php -S localhost:8080



# Types de tests - Test fonctionnel

```
<?php

namespace FormationTechTest\Functionnal;

use Goutte\Client;

class DatabaseListTest extends \PHPUnit_Framework_TestCase
{
    public function testListDbs()
    {
        $client = new Client();
        $crawler = $client->request('GET', 'http://localhost:8080/database-list.php');

        $this->assertEquals(200, $client->getResponse()->getStatus());
        $this->assertEquals('Database list', $crawler->filter('h2')->text());
        $this->assertCount(13, $crawler->filter('ul > li'));
    }
}
```

```
MBP-de-Romain:PrepaFormationPHPUnit romain$ ./vendor/bin/phpunit tests/Logger/LoggerTest.php --colors
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

```
.
```

```
1 / 1 (100%)
```

```
Time: 38 ms, Memory: 1.50Mb
```

```
OK (1 test, 1 assertion)
```



**formation.tech**

# Doubles

# Double - Introduction



- Le code PHP fait souvent appel à des composants externes :
  - Accès aux entrées/sorties
  - Accès à une base de données
  - Accès à un Service Web
- Certaines classes ne peuvent être testées de manières unitaires car elles dépendent d'autres classes.
- **Solutions : les Doubles**  
Objets ou fonctions qui ressemblent et se comportent comme le composant qu'ils imitent, mais qui sont en réalité des versions simplifiée qui permettent de faciliter l'écriture du test.

# Double - Introduction



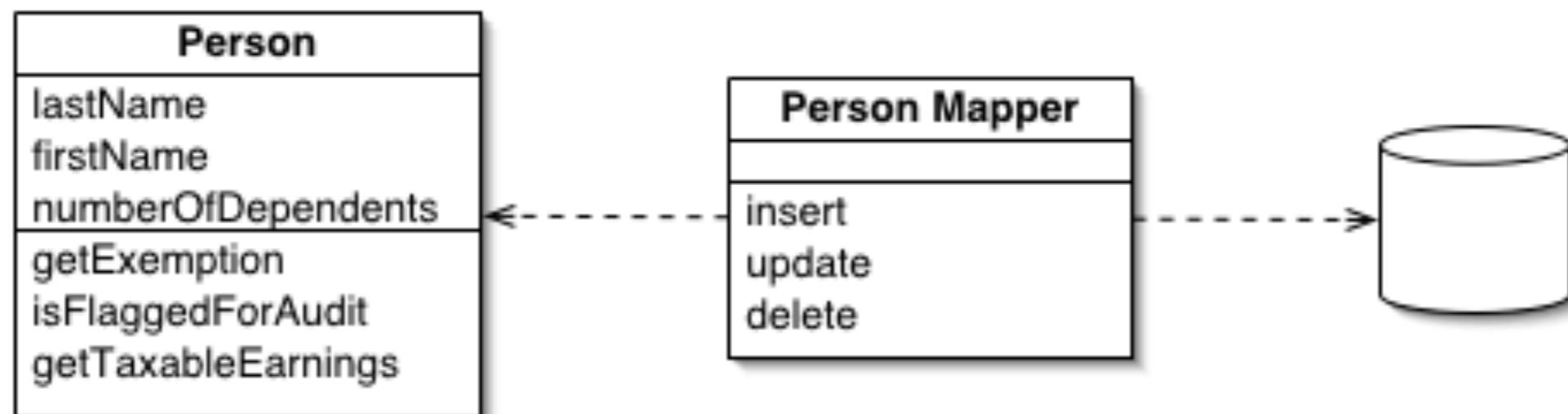
- 5 types de Doubles :
  - Fake (une classe créée par l'utilisateur qui fera les opérations en mémoire)
  - Dummy (une classe générée dont les méthodes ne font rien)
  - Stub (une classe générée dont les méthodes ont le même comportement)
  - Mock (Stub + vérification que les méthodes soient bien appelée)
  - Spy (Dummy + vérification que les méthodes soient bien appelée à postériori)
- Bonnes pratiques :
  - Injection de Dépendance (pas de composition)
  - Registre ou Container d'injection de Dépendance



# Double - Classes d'exemple

- Un DataMapper

<http://martinfowler.com/eaaCatalog/dataMapper.html>



# Double - Classes d'exemple



- ▶ Entité

```
<?php

namespace FormationTech\Entity;

class Database
{
    protected $name;

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

    public function getName()
    {
        return $this->name;
    }
}
```

# Double - Classes d'exemple



## ► Gateway

```
<?php

namespace FormationTech\Gateway;

class DatabaseGateway implements DatabaseGatewayInterface
{
    protected $pdo;

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

    public function listDbs()
    {
        $stmt = $this->pdo->query('SHOW DATABASES');

        return $stmt->fetchAll(\PDO::FETCH_COLUMN);
    }
}
```

```
<?php

namespace FormationTech\Gateway;

interface DatabaseGatewayInterface
{
    public function listDbs();
}
```

# Double - Classes d'exemple



- Mapper (classe à tester unitairement)

```
<?php

namespace FormationTech\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGatewayInterface;

class DatabaseMapper
{
    protected $gateway;

    public function __construct(DatabaseGatewayInterface $gateway)
    {
        $this->gateway = $gateway;
    }

    public function findAll()
    {
        $dbsArray = $this->gateway->listDbs();
        $dbsObj = [];

        if (! $dbsArray) {
            return $dbsObj;
        }

        foreach ($dbsArray as $dbName) {
            $dbsObj[] = new Database($dbName);
        }
    }
}
```

# Double - Sans Double



- › Test sans double

```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    public function testfindAllWithMySQL()
    {
        $pdo = new \PDO('mysql:host=localhost', 'root', '');
        $gateway = new DatabaseGateway($pdo);
        $mapper = new DatabaseMapper($gateway);

        $dbs = $mapper->findAll();

        $this->assertCount(13, $dbs);
        $this->assertContainsOnlyInstancesOf(Database::class, $dbs);
    }
}
```

- › Problème : changement dans la base de données ?
- › Solution : fixture dans un setUp ? double ?

# Double - Fake



## ▶ Fake

```
<?php

namespace FormationTech\Gateway;

class DatabaseGatewayFake implements DatabaseGatewayInterface
{
    protected $dbs;

    public function __construct(Array $dbs)
    {
        $this->dbs = $dbs;
    }

    public function listDbs()
    {
        return $this->dbs;
    }
}
```

```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGatewayFake;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    public function testFindAllWithFake()
    {
        $gateway = new DatabaseGatewayFake(['db1', 'db2', 'db3']);
        $mapper = new DatabaseMapper($gateway);

        $dbs = $mapper->findAll();

        $this->assertCount(3, $dbs);
        $this->assertContainsOnlyInstancesOf(Database::class, $dbs);
    }
}
```

# Double - Prophecy



- › Sebastian Bergman à propos de l'API de Mock de PHPUnit :  
<https://thephp.cc/news/2015/02/phpunit-4-5-and-prophecy>
- › L'ancien API continue d'exister pour rester compatible avec les anciens tests
- › PHPUnit depuis la version 4.5 intègre un framework de test moderne : Prophecy
- › Documentation  
<https://github.com/phpspec/prophecy>

# Double - Prophecy Dummy



```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    // ...

    public function testfindAllWithDummyProphecy()
    {
        $dummy = $this->prophesize(DatabaseGateway::class);

        $mapper = new DatabaseMapper($dummy->reveal());

        $dbs = $mapper->findAll();

        $this->assertEmpty($dbs);
    }

    // ...
}
```

# Double - Prophecy Stub



```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    // ...

    public function testfindAllWithStubProphecy()
    {
        $stub = $this->prophesize(DatabaseGateway::class);

        $stub->listDbs()->willReturn(['db1', 'db2', 'db3', 'db4']);

        $mapper = new DatabaseMapper($stub->reveal());

        $dbs = $mapper->findAll();

        $this->assertCount(4, $dbs);
        $this->assertContainsOnlyInstancesOf(Database::class, $dbs);
    }

    // ...
}
```

# Double - Prophecy Mock



```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    // ...

    public function testFindAllWithMockProphecy()
    {
        $mock = $this->prophesize(DatabaseGateway::class);

        $mock->listDbs()->willReturn(['db1', 'db2'])->shouldBeCalledTimes(1);

        $mapper = new DatabaseMapper($mock->reveal());

        $dbs = $mapper->findAll();

        $this->assertCount(2, $dbs);
        $this->assertContainsOnlyInstancesOf(Database::class, $dbs);
    }

    // ...
}
```

# Double - Prophecy Spy



```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{

    // ...

    public function testfindAllWithSpyProphecy()
    {
        $spy = $this->prophesize(DatabaseGateway::class);
        $mapper = new DatabaseMapper($spy->reveal());
        $dbs = $mapper->findAll();
        $this->assertEmpty($dbs);
        $spy->listDbs()->shouldHaveBeenCalledTimes(1);
    }

    // ...
}
```

# Double - Autres frameworks



- ▶ **Mockery**

<https://github.com/padraic/mockery>

<http://docs.mockery.io/en/latest/>

- ▶ **Phake**

<https://github.com/mlively/Phake>

<http://phake.readthedocs.org/en/2.1/>



**formation.tech**

# Tests Symfony



# Tests Symfony - Tests Fonctionnels

## ▶ Test fonctionnels

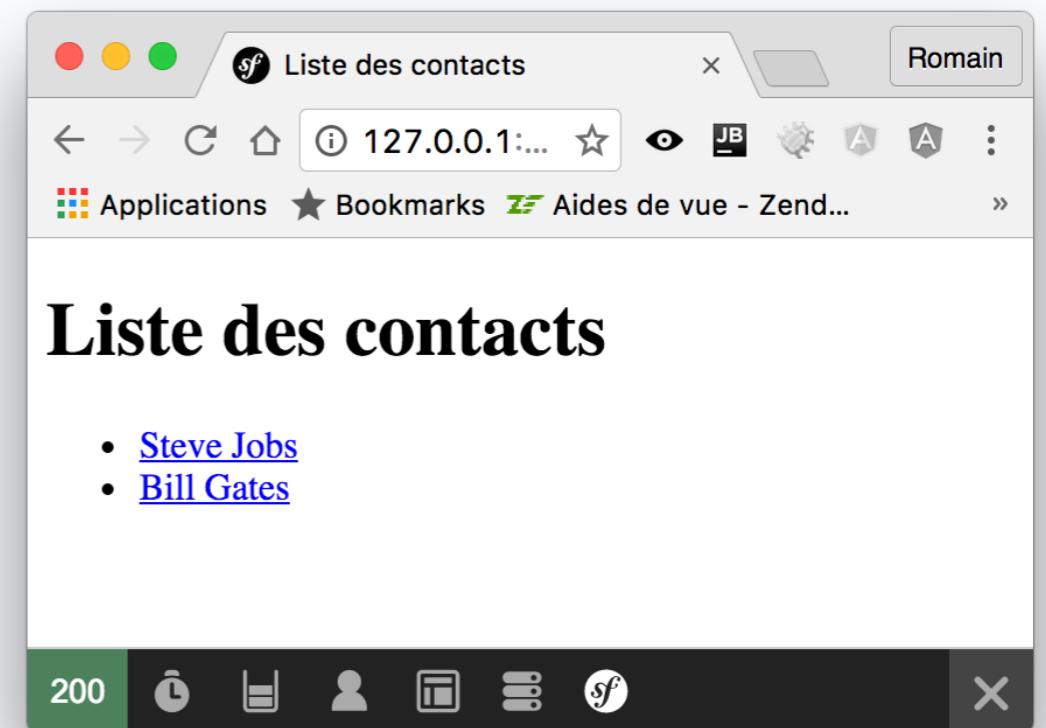
Symfony\Bundle\FrameworkBundle contient 2 classes pour faciliter les tests

- ▶ Symfony\Bundle\FrameworkBundle\Test\WebTestCase (et sa méthode createClient pour les tests fonctionnels)
- ▶ Symfony\Bundle\FrameworkBundle\Test\KernelTestCase (et sa méthode bootKernel pour les tests qui nécessitent un kernel)

```
public function listAction()
{
    $repo = $this->getDoctrine()
        ->getRepository('AppBundle:Contact');

    $contacts = $repo->findAll();

    return $this->render('list.html.twig', array(
        'contacts' => $contacts
    ));
}
```





# Tests Symfony - Tests Fonctionnels

## ▶ Test fonctionnels

```
<?php

namespace AppBundle\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class ContactControllerTest extends WebTestCase
{
    public function testListAvecMysql()
    {
        $client = static::createClient();

        $crawler = $client->request('GET', '/contacts/');
        $this->assertEquals(200, $client->getResponse()->getStatusCode());

        $this->assertContains('Liste des contacts', $crawler->filter('h1')->text());
        $this->assertCount(2, $crawler->filter('h1 + ul > li'));
    }
}
```

- ▶ Problème : le tests dépend d'un composant extérieur (base de données)
- ▶ Solution 1 : Réinitialiser la base de données entre chaque test (dans une méthode setup)
- ▶ Solution 2 : Utiliser les mocks



# Tests Symfony - Tests Fonctionnels

- ▶ Test fonctionnels (avec mock)

```
public function testListAvecMock()
{
    $client = static::createClient();

    $contacts = [
        (new Contact())->setId(1)->setPrenom('A')->setNom('B'),
        (new Contact())->setId(2)->setPrenom('C')->setNom('D'),
        (new Contact())->setId(3)->setPrenom('E')->setNom('F'),
    ];

    $mockRepo = $this->prophesize(ContactRepository::class);
    $mockRepo->findAll()->willReturn($contacts)->shouldBeCalledTimes(1);

    $mockRegistry = $this->prophesize(Registry::class);
    $mockRegistry->getConnectionNames()->shouldBeCalledTimes(1);
    $mockRegistry->getManagerNames()->shouldBeCalledTimes(1);
    $mockRegistry->getRepository('AppBundle:Contact')->willReturn($mockRepo->reveal())->shouldBeCalledTimes(1);

    $client->getContainer()->set('doctrine', $mockRegistry->reveal());

    $crawler = $client->request('GET', '/contacts/');

    $this->assertCount(3, $crawler->filter('h1 + ul > li'));
}
```

- ▶ Problème : le contrôleur dépend de la classe Registry pour obtenir le Repository, 2 mocks à créer
- ▶ Solution : avoir une dépendance directe dans le Conteneur (couche Service par exemple)



# Tests Symfony - Tests Fonctionnels

- ▶ Test fonctionnels (avec couche Service)

```
public function testListAvecMockEtServiceLayer()
{
    $client = static::createClient();

    $contacts = [
        (new Contact())->setId(1)->setPrenom('A')->setNom('B'),
        (new Contact())->setId(2)->setPrenom('C')->setNom('D'),
        (new Contact())->setId(3)->setPrenom('E')->setNom('F'),
    ];

    $mockRepo = $this->prophesize(ContactManager::class);
    $mockRepo->findAll()->willReturn($contacts)->shouldBeCalledTimes(1);

    $client->getContainer()->set('app.manager.contact', $mockRepo->reveal());

    $crawler = $client->request('GET', '/contacts/list-avec-manager');

    $this->assertCount(3, $crawler->filter('h1 + ul > li'));
}
```

```
# services.yml
services:
    app.manager.contact:
        class: AppBundle\Manager>ContactManager
        arguments: ["@doctrine.orm.entity_manager"]
```



# Tests Symfony - Tests de Commande

## ▶ Test de Commande

```
<?php

namespace AppBundle\Command;

use Symfony\Bundle\FrameworkBundle\Command\ContainerAwareCommand;
use Symfony\Component\Console\Input\InputArgument;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Output\OutputInterface;

class HelloWorldCommand extends ContainerAwareCommand
{
    protected function configure()
    {
        $this->setName('hello:world')
            ->setDescription('A Hello command')
            ->addArgument('name', InputArgument::OPTIONAL, 'Your name')
            ->addOption('upper', 'u', InputOption::VALUE_NONE, 'Capitalize answer');
    }

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        $name = $input->getArgument('name');
        $message = ($name) ? "Hello $name :" : "Hello !";

        if ($input->getOption('upper')) {
            $message = strtoupper($message);
        }

        $output->writeln($message);
    }
}
```



# Tests Symfony - Tests de Commande

## ▶ Test de Commande

```
<?php

namespace Tests\AppBundle\Command;

use AppBundle\Command\HelloWorldCommand;
use Symfony\Bundle\FrameworkBundle\Console\Application;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;
use Symfony\Component\Console\Tester\CommandTester;

class HelloWorldCommandTest extends KernelTestCase
{
    public function testExecute()
    {
        $kernel = $this->createKernel();
        $kernel->boot();

        $application = new Application($kernel);
        $application->add(new HelloWorldCommand());

        $command = $application->find('hello:world');
        $commandTester = new CommandTester($command);
        $exitCode = $commandTester->execute(array(
            'command' => $command->getName(),
            'name' => 'Romain',
            '-u' => true
        ));

        $output = $commandTester->getDisplay();
        $this->assertEquals(0, $exitCode, 'Returns 0 in case of success');
        $this->assertContains('HELLO ROMAIN :)', $output);
    }
}
```



# Tests Symfony - Tests de Formulaire

## ▶ Test de Formulaire

```
<?php

namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class ContactType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('prenom')
            ->add('nom')
        ;
    }

    /**
     * @param OptionsResolver $resolver
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'AppBundle\Entity\Contact'
        ));
    }
}
```



# Tests Symfony - Tests de Formulaire

```
<?php

namespace AppBundle\Tests\Form;

use AppBundle\Entity>Contact;
use AppBundle\Form\ContactType;
use Symfony\Component\Form\Test\TypeTestCase;

class ContactTypeTest extends TypeTestCase
{
    public function testSubmitValidData()
    {
        $formData = array(
            'prenom' => 'Romain',
            'nom' => 'Bohdanowicz',
        );

        $form = $this->factory->create(ContactType::class);

        $contact = (new Contact())->setPrenom('Romain')->setNom('Bohdanowicz');

        // submit the data to the form directly
        $form->submit($formData);

        $this->assertTrue($form->isSynchronized());
        $this->assertEquals($contact, $form->getData());

        $view = $form->createView();
        $children = $view->children;

        foreach (array_keys($formData) as $key) {
            $this->assertArrayHasKey($key, $children);
        }
    }
}
```



**formation.tech**

# API Platform



# API Platform - Introduction

- Bibliothèque pour accélérer le développement d'API REST
- Principales fonctionnalités
  - CRUD Automatique (GET, POST, PUT, DELETE)
  - Requêtes Hypermedia
  - Documentation Swagger/OpenAPI
  - Pagination
  - Filtres
  - Tri
  - Validation
  - Authentification
  - ...



# API Platform - Introduction

- Installation :  
`composer require api`
- Documentation :  
<https://api-platform.com/docs/v2.5/core/>



# API Platform - Configuration

- Pour créer une resource on ajoute l'annotation `ApiResource` à nos entités

```
use ApiPlatform\Core\Annotation\ApiResource;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity()
 * @ApiResource()
 */
class Post
{
```

- Notre entité est désormais exposé sous
  - GET /api/posts → liste
  - POST /api/posts → création
  - GET /api/posts/{id} → détails
  - PUT /api/posts/{id} → remplacement
  - DELETE /api/posts/{id} → suppression
  - PATCH /api/posts/{id} → mise à jour

## API Platform - Accès



- Dans le navigateur, l'accès à /api ou /api/resources afficher la documentation Swagger/OpenAPI
  - Dans Postman on reçoit les données par défaut au format JSON-LD
  - On peut spécifier qu'on souhaite du JSON via l'entête  
Accept: application/json

The screenshot illustrates the integration of API documentation and testing. On the left, a browser window displays the API Platform documentation for the 'Post' entity, listing various HTTP methods and their descriptions. On the right, the Postman application is used to send a GET request to the endpoint `http://localhost:4000/api/posts`. The Headers tab shows 'Connection: keep-alive' and 'Accept: application/json'. The response body is displayed in JSON format, showing a single post object with fields like id, title, content, created, user, and comments.

API Platform

localhost:4000/api

Invité

Post

GET /api/posts Retrieves the collection of Post resources.

POST /api/posts Creates a Post resource.

GET /api/posts/{id} Retrieves a Post resource.

PUT /api/posts/{id} Replaces the Post resource.

DELETE /api/posts/{id} Removes the Post resource.

PATCH /api/posts/{id} Updates the Post resource.

Postman

Home Workspaces Reports Explore

GET http://localhost:4000/api/posts

http://localhost:4000/api/posts

GET http://localhost:4000/api/posts

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

Connection: keep-alive

Accept: application/json

Key Value Description

Status: 200 OK Time: 1400 ms Size: 7.91 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 [ { 2 "id": 110, 3 "title": "atque quo nihil ut", 4 "content": "Praesentium rerum iusto ut consequuntur et cupiditate quo doloremque. Id quae dolor id vero dignissimos nostrum. Dolores sunt voluptates quo.", 5 "created": "2021-01-25T16:23:50+00:00", 6 "user": "/api/users/33", 7 "comments": [ 8 " /api/comments/151" ] } ]
```

Find and Replace Console

# API Platform - Operations



- Il est possible de limiter les actions disponibles

```
/**  
 * @ORM\Entity()  
 * @ApiResource(  
 *     collectionOperations={"get"},  
 *     itemOperations={"get"}  
 * )  
 */
```

- collectionOperations → les opérations sur /api/resources (get ou post)
- itemOperations → les opérations sur /api/resources/{id} (get, put, delete, patch)
- Créer des routes custom :  
<https://api-platform.com/docs/core/controllers/#creating-custom-operations-and-controllers>

# API Platform - DataProvider



- Les DataProvider permet de redéfinir les méthodes d'accès aux collections ou aux items
- 3 interfaces
  - CollectionDataProviderInterface / ContextAwareCollectionDataProviderInterface  
Permet de redéfinir la méthode d'accès à la collection
  - ItemDataProviderInterface  
Permet de redéfinir la méthode d'accès à l'item
  - RestrictedDataProviderInterface  
Permet de restreindre l'utilisation du provider à une Entity, une opération...

# API Platform - DataProvider



```
<?php

namespace App\DataProvider;

use ApiPlatform\Core\DataProvider\ContextAwareCollectionDataProviderInterface;
use ApiPlatform\Core\DataProvider\RestrictedDataProviderInterface;
use App\Entity\Post;
use App\Manager\PostManager;

class PostCollectionDataProvider implements ContextAwareCollectionDataProviderInterface,
RestrictedDataProviderInterface
{
    /** @var PostManager */
    protected $postManager;

    public function __construct(PostManager $postManager)
    {
        $this->postManager = $postManager;
    }

    public function supports(string $resourceClass, string $operationName = null, array
    $context = []): bool
    {
        return Post::class === $resourceClass;
    }

    public function getCollection(string $resourceClass, string $operationName = null,
array $context = []): iterable
    {
        return $this->postManager->getAll();
    }
}
```

# API Platform - DataPersister



- A la manière des DataProviders, les DataPersister permettent de redéfinir le comportement des méthodes d'écriture
- Interface à implémenter : ContextAwareDataPersisterInterface
- 3 méthodes
  - persist: pour insérer/mettre à jour
  - remove: pour supprimer
  - supports: pour restreindre l'utilisation du provider à une Entity, une opération...



# API Platform - Filtres

- Les filtres sont déclarés avec l'annotation `ApiFilter`
- Type de filtres
  - `search`
  - `date`
  - `boolean`
  - `numeric`
  - `range`
  - `exists`
  - `order`
  - `custom`



# API Platform - SearchFilter

```
/**  
 * @ORM\Entity()  
 * @ApiResource()  
 * @ApiFilter(SearchFilter::class, properties={"title": "ipartial", "content":  
 "iword_start"})  
 */  
class Post  
{
```

- Si on ne spécifie pas de propriétés, le filtre est disponible sur tous les champs
- 5 valeurs possibles (si on préfixe par i, devient case-insensitive)
  - exact -> la colonne est
  - partial -> la colonne contient
  - start -> commence par
  - end -> se termine par
  - word\_start -> un mot commence par
- On passe les valeurs via la query string  
`/api/posts?title=test`