Etienne LANGLET.

9h00 - 12h30

14h00 - 17h00

30

---

Administratif :

→ Emargement : 1/2 journée

→ Questionnaires : 1° jour : Mesure des connaissances initiales

D. jour : Mesure des acquis

Satisfaction.

" IMPULSE Sign-In "

Code d'accès : WVVLYU

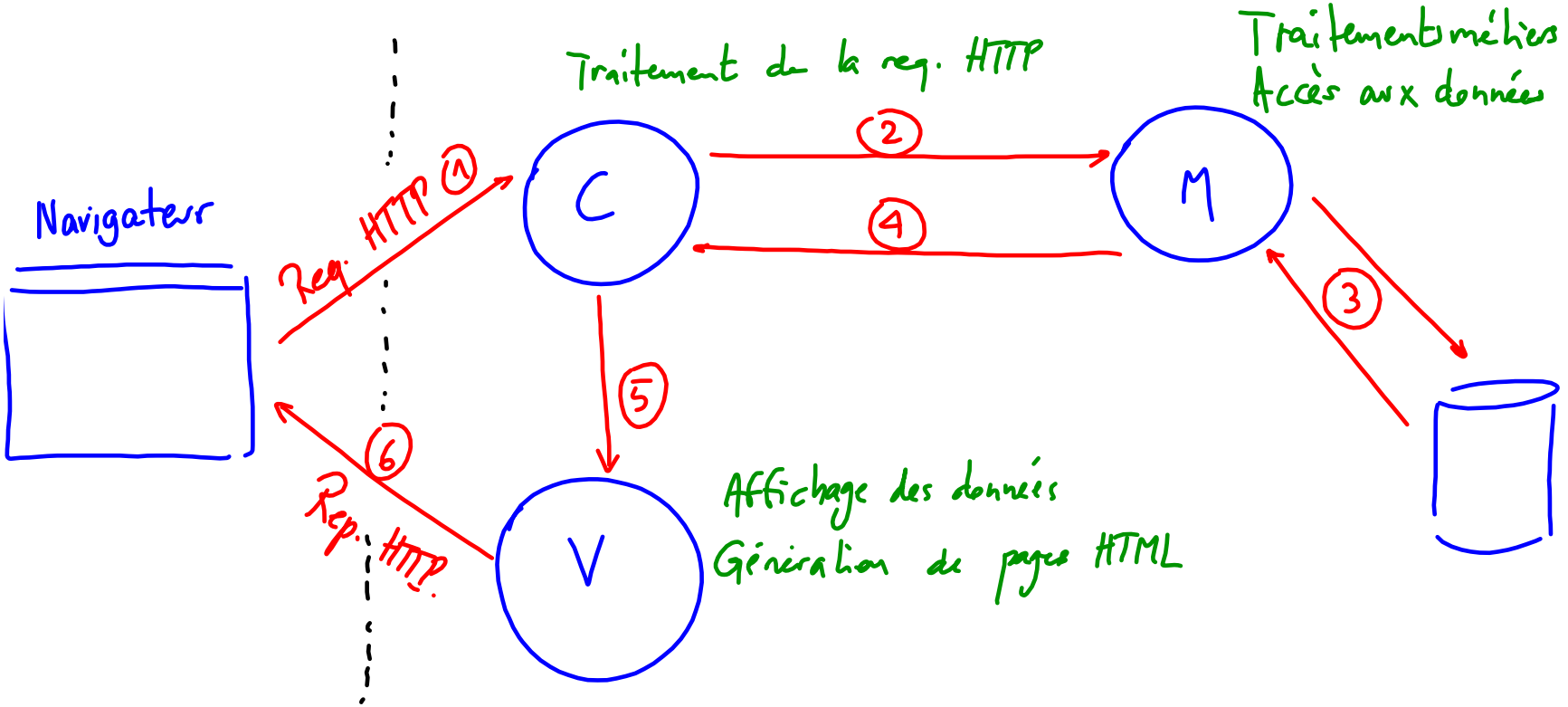
# Framework MVC

MVC → Design - Pattern

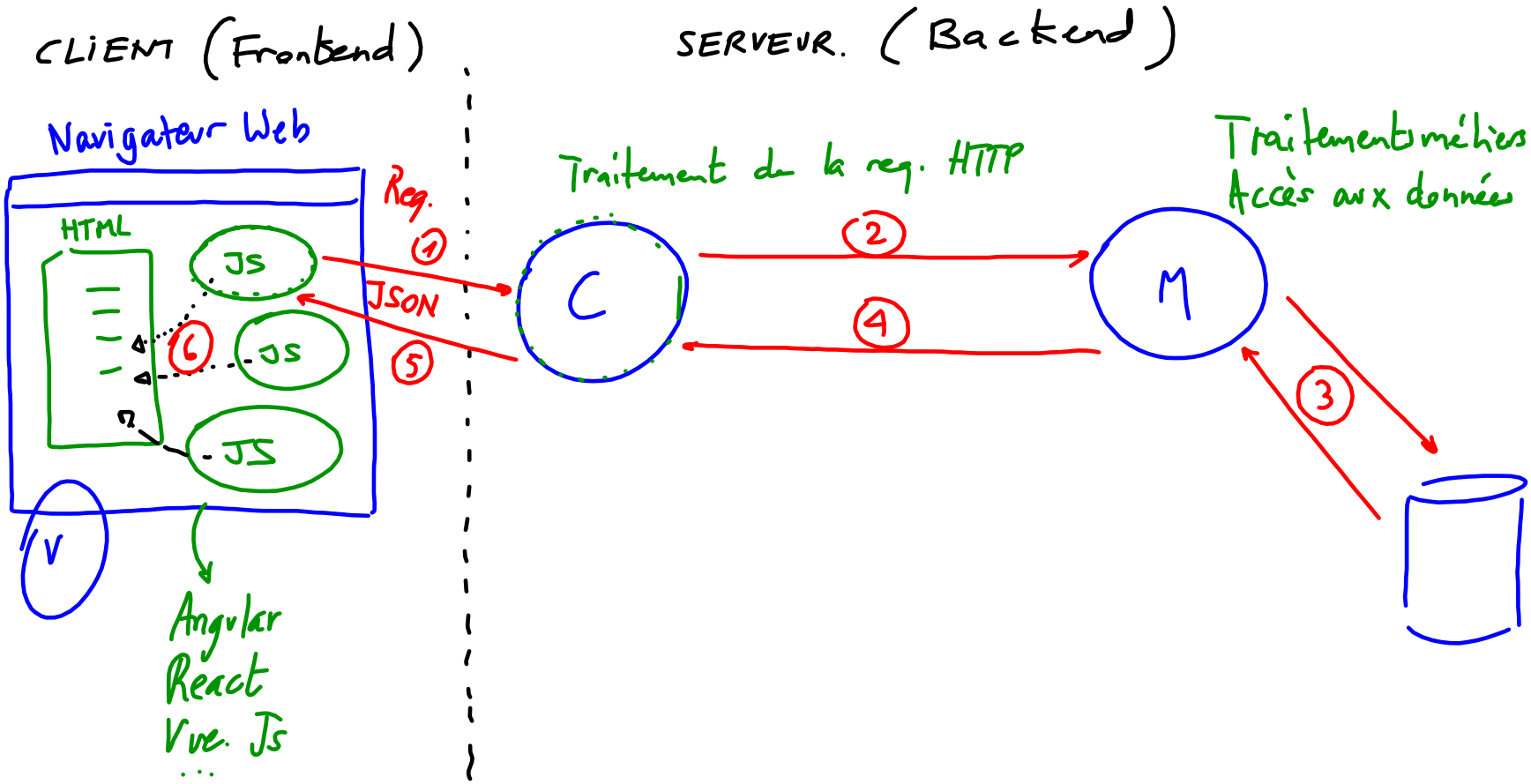
↳ Model / View / Controller.

⊛ Dans une application Web traditionnelle

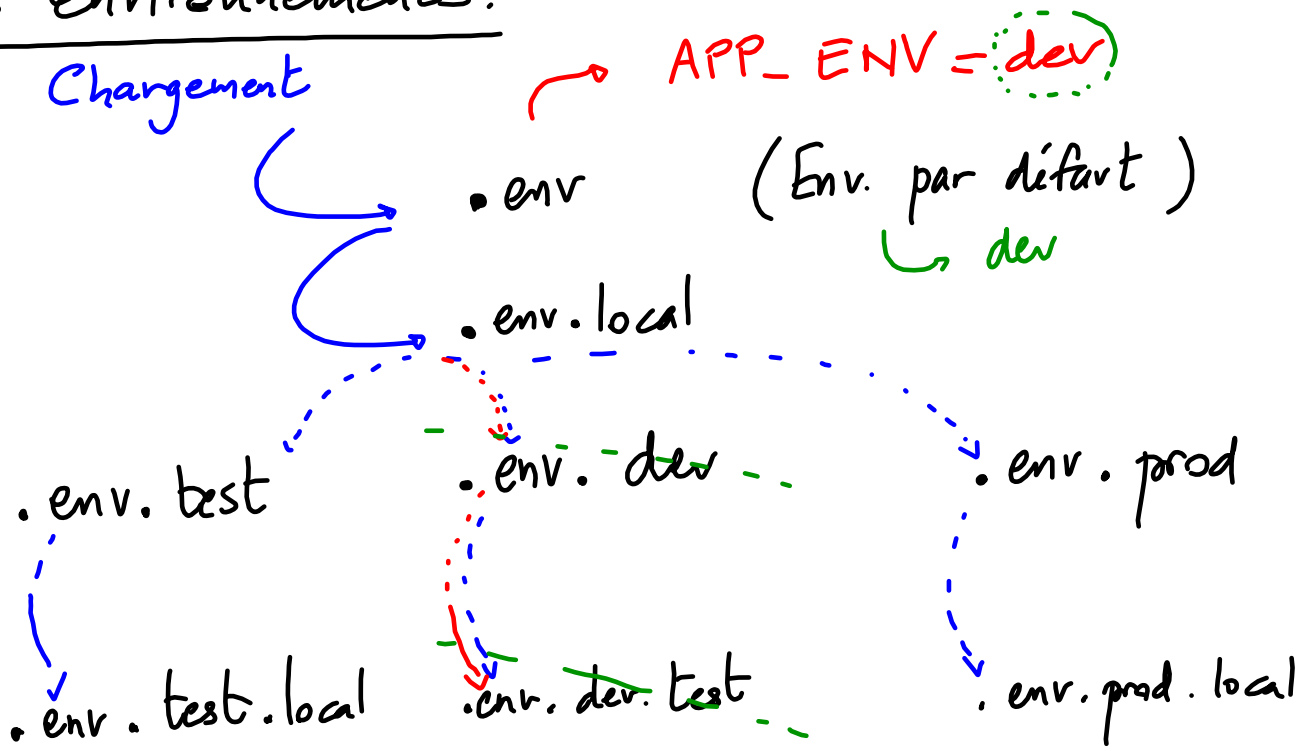
CLIENT SERVEUR.



⊛ Dans une application Web de 2° génération



# Les environnements.



# Chargement des classes & PSR-4.

```
"autoload": {
  "psr-4": {
    "App\\" : "src/"
  }
},
```

En cas de modification, il faut exécuter la commande 'composer update'

## Structure logique

namespace App\...

namespace App\Entity;

class Article {

}

## Structure Physique

src/

└ Entity/

└ Article.php

# Annotations Symfony & Attributs PHP 8

PHP 8 introduit les attributs
↳ Semblables aux annotations Symfony

**Annotation :**

```
/**
 * @Route("/article")
 */
```

**Attribut :**

```
#[Route('/article', name: 'app_article')]
```

Attributs PHP 8 utilisables à partir de Symfony 5.2.

# Obtenir des informations sur le routage

```
php bin/console debug:router -e prod
```

Pour éviter d'afficher les routes du profiler.

## Générer un contrôleur.

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

↳ Le nom sera demandé interactivement.

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

# Générer des URLs dans un contrôleur.

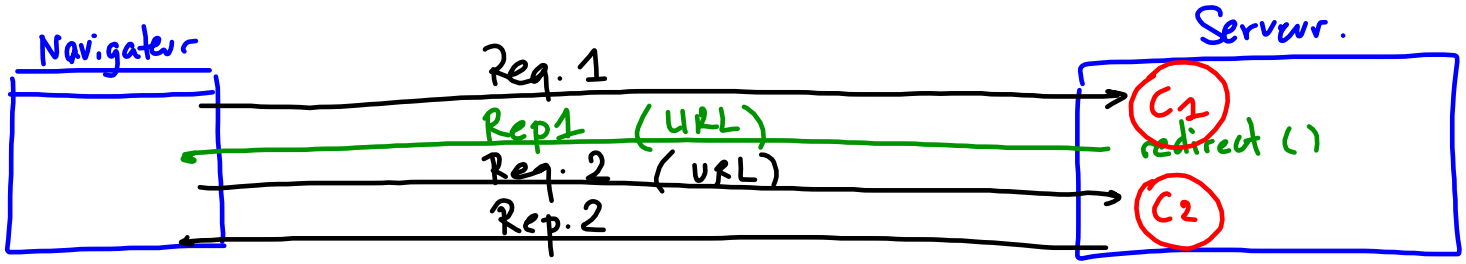
```
C:\Symfony\MonJournal>php bin/console debug:router -e prod
 -------------- -------- -------- ------ --------------
  article_all    ANY      ANY      ANY    /articles
  article_show   ANY      ANY      ANY    /article/{id}
 -------------- -------- -------- ------ --------------
```

```
$url = $this->generateUrl(
    'article_show',
    ['id' => 14]
);
```

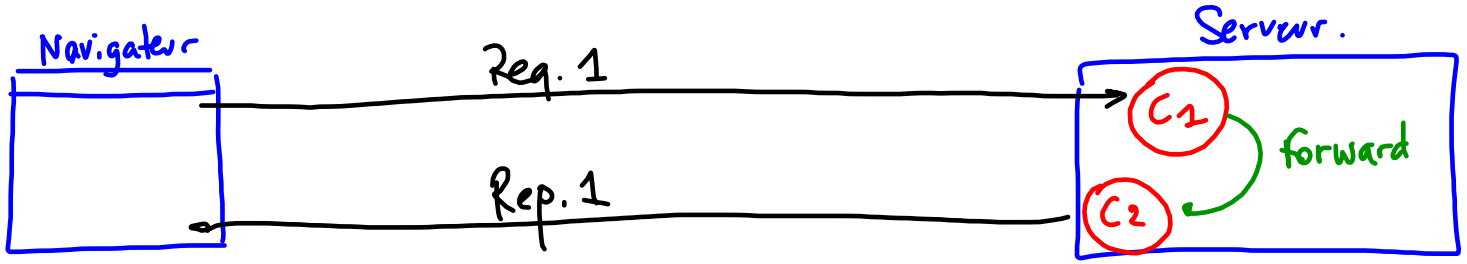
→ /article/14

# Redirection vs. Délégation

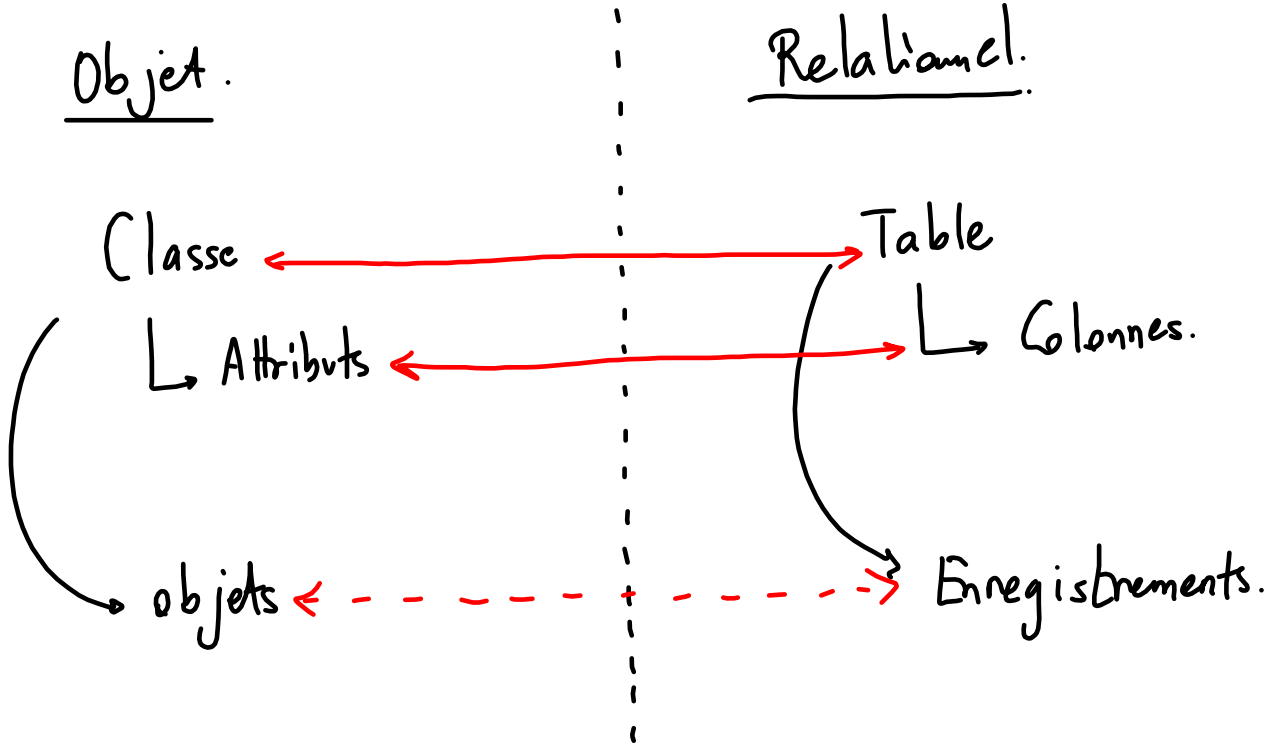
## Redirection redirect()



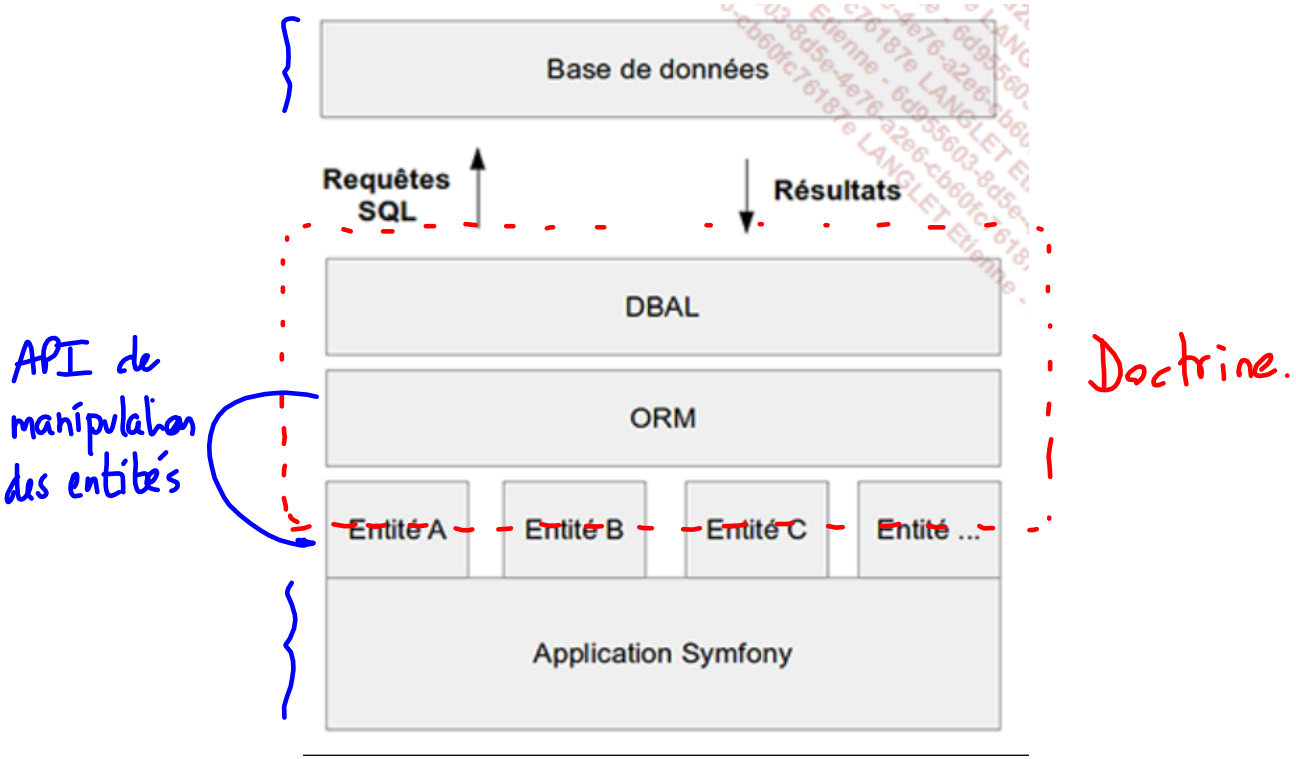

## Délégation forward()



# Principes de l'ORM



# Architecture de Doctrine



# Générer les entités à partir des tables.

## 1. / Importer le mapping

```
php bin/console doctrine:mapping:import App\Entity annotation --path=src/Entity
```

## 2. / Générer les entités

```
php bin/console make:entity App\Entity --regenerate
```

⚠ La génération des entités considère toujours que la clé primaire est générée par la base → @GeneratedValue.

# L'API ORM de Doctrine

↳ Manipuler les entités

## 2 objets:

- EntityManager → Modification de données (INSERT, UPDATE, DELETE)
- Repository → Récupération de données (SELECT)

# Opération de l'EntityManager

→ INSERT : persist ($entite) + flush ()

→ UPDATE : 1/ l'objet à modifier n'est pas présent dans l'appli.

a. Aller chercher l'objet avec le Reposity (find ())
b. Modifier l'objet
c. persist ($objet) + flush ()

2 / l'objet est présent dans l'appli.

merge ($objetModifie) + flush ()

→ DELETE : remove ($entite) + flush ()

# Mise en oeuvre : Création d'un service d'accès aux données

```
src/
  └ Model/
      └ ArticleService.php
```

```
namespace App\Model;

class ArticleService {


}
```

# Récupération d'un Repository

└ Objet de sélection d'entité.

FQCN

```
$repo = $this->em->getRepository('App\Entity\Article');
```

```
$repo = $this->em->getRepository('App:Article');
```

Shortcut Notation

# Methodes du Repository

findAll()  SELECT * FROM table

find(15)  SELECT * FROM table WHERE pk = 15

findOneBy(['auteur' => 'DUPONT'])  SELECT * FROM table WHERE auteur = 'DUPONT'

L'attribut $auteur ⚠

RACCOURCI !

findOneByAuteur('DUPONT')

findBy(['datePublication' => '...'])  SELECT * FROM table WHERE date_publication = '...'

```
findBy(
  ['auteur' => 'DUPON'],
  ['datePublication' => 'DESC']
)
```

SELECT * FROM table WHERE auteur = 'DUPONT' ORDER BY date_publication DESC

# DQL

SQL : SELECT * FROM article

DQL : SELECT a FROM App:Article a

Alias sur les objets retournés

↳ l'objet complet.

App\Entity\Article

SQL : SELECT * FROM article WHERE date_publication LIKE '... %'

DQL : SELECT a FROM App:Article a WHERE a.datePublication LIKE '... %'

Colonne

↳ Attribut.

SQL : SELECT texte FROM article

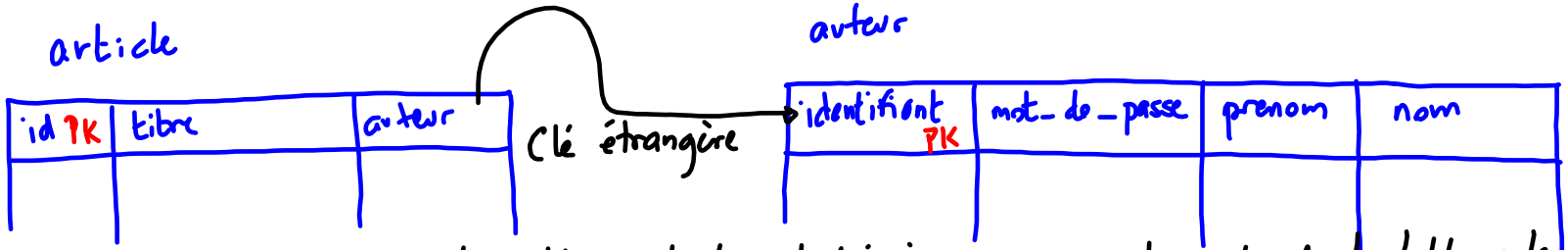
DQL : SELECT a.texte FROM App:Article a

## Référence DQL:

https://www.doctrine-project.org/projects/doctrine-orm/en/2.9/reference/dql-doctrine-query-language.html

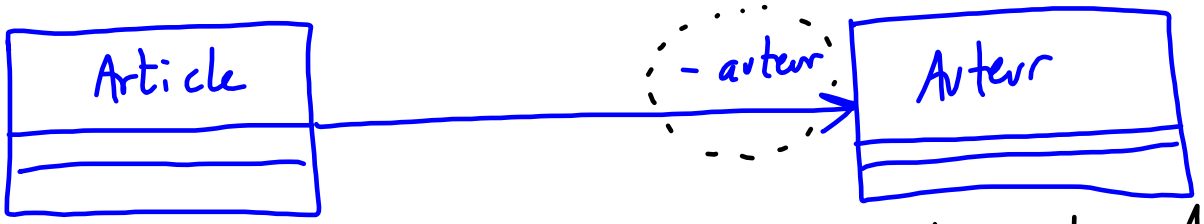
# Relations & Associations.

**Relationnel** : Relation de clé étrangère.




1 enregistrement de la table article est lié à un enregistrement de la table auteur.

**Objet**: Association entre classe




1 objet de type Article est associé à un objet de type Auteur.
La classe Article possède un attribut de type Auteur.

# Notion de "fetch"

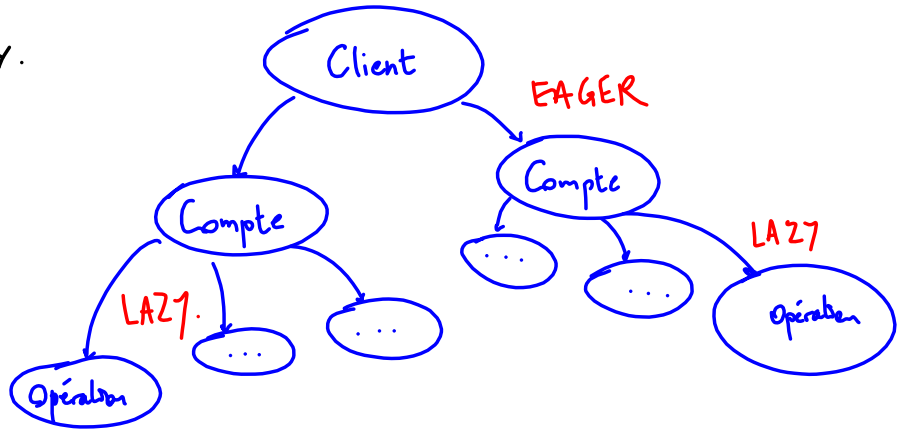
Stratégie de chargement des entités.

* LAZY (par défaut)
  - ↳ Chargement à la demande.
    - . 1 requête par type d'entité.
    - . On ramène les entités liées uniquement si on les manipule.

* EAGER
  - ↳ Chargement immédiat.
    - . 1 seul requête avec jointure pour ramener l'entité et son entité liée.

—→ On décide de la stratégie pour chaque relation / association !

Ex.



# Mise en oeuvre : La gestion des auteurs.

→ Classe de gestion des entités.

AuteurService

```
public function ajouterAuteur(Auteur $auteur): void
public function rechercherTousLesAuteurs(): array
public function rechercherAuteurParIdentifiant(string $identifiant): Auteur
public function rechercherAuteursParNomEtPrenom(string $nom, string $prenom)
    : array
```

⚠ A l'ordre de déclaration des routes.

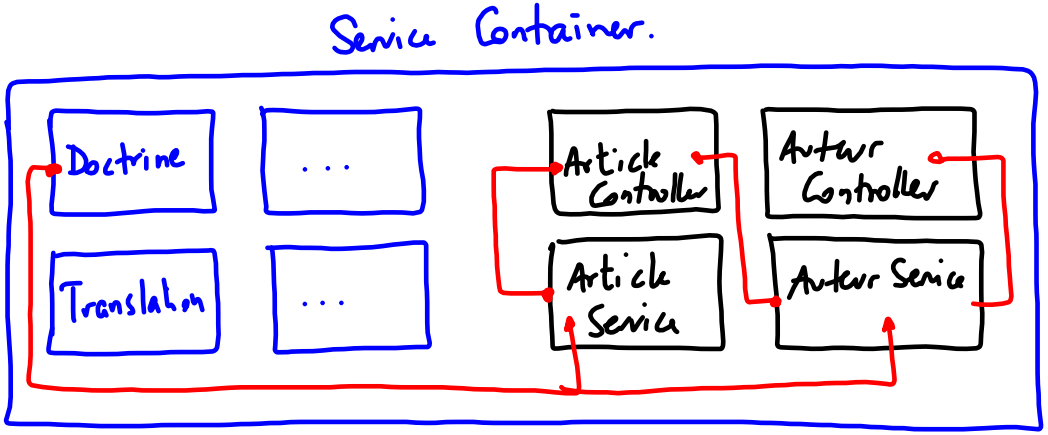
→ Contrôleur

AuteurController

| | |
|---|---|
| add() | /auteur/add |
| all() | /auteurs |
| show(string $identifiant) | /auteur/{identifiant} |
| showByName(string $nom, string $prenom) | /auteur/{nom}/{prenom} |

# Le Service Container.

- Crée les objets : Des "Singletons" ( 1 seul et unique objet par Classe ).
- Il les mets en relation



# Création de service & Injection

## ⊛ Injection par le constructeur.

```yaml
# config/services.yaml

services:
    my_pdo:
        class: PDO
        arguments: ['mysql:host=localhost', 'bilal', 'pass']
    x:
        class: X
        arguments: @my_pdo
```

Référence au service 'my_pdo'

Arguments du constructeur

## ⊛ Injection par méthode

```yaml
# config/services.yaml

services:
    my_pdo:
        class: PDO
        arguments: ['mysql:host=localhost', 'bilal', 'pass']
    x:
        class: X
        calls:
            - [setDb, [@my_pdo]]
```

On injecte le service 'my_pdo' en paramètre de la méthode 'setDb'

## ⊛ Injection par propriété

```yaml
# config/services.yaml

services:
    my_pdo:
        class: PDO
        arguments: ['mysql:host=localhost', 'bilal', 'pass']
    x:
        class: X
        properties:
            db: @my_pdo
```

On injecte 'my_pdo' dans l'attribut (public) $db

Attribut de la classe X

# Déclaration et injection automatique de services

config/service.yaml :

```yaml
services
    # default configuration for services in *this* file
    _defaults:
        autowire: true
        autoconfigure: true

    # makes classes in src/ available to be used as services
    # this creates a service per class whose id is the fully-qualified class name
    App\:
        resource: '../src/'
        exclude:
            - '../src/DependencyInjection/'
            - '../src/Entity/'
            - '../src/Kernel.php'
```

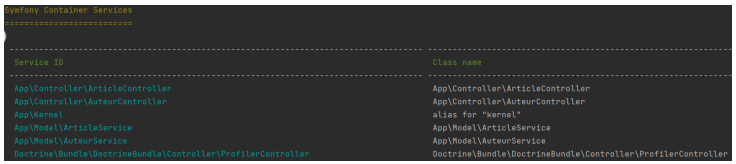
Tous les services sont injectés automatiquement PAR LEUR TYPE
Toutes les classes sont déclarées comme des services

Les répertoires sont exclus de la création automatique de service.
Important pour les entités.

# Obtenir des informations sur les services.

## Afficher tous les services

```
php bin/console debug:container
```



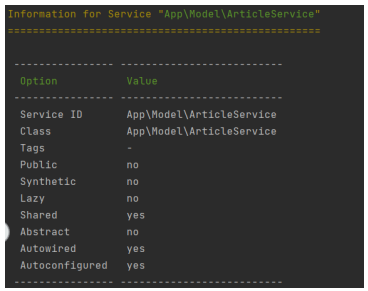
```
Symfony Container Services
==========================

 ------------------------------------------------------------------------ ------------------------------------------------------------------------
  Service ID                                                               Class name
 ------------------------------------------------------------------------ ------------------------------------------------------------------------
  App\Controller\ArticleController                                         App\Controller\ArticleController
  App\Controller\AuteurController                                          App\Controller\AuteurController
  App\Kernel                                                               alias for "kernel"
  App\Model\ArticleService                                                 App\Model\ArticleService
  App\Model\AuteurService                                                  App\Model\AuteurService
  Doctrine\Bundle\DoctrineBundle\Controller\ProfilerController             Doctrine\Bundle\DoctrineBundle\Controller\ProfilerController
```

## Afficher les infos d'un service

```
php bin/console debug:container App\Model\ArticleService
```

Id du service



```
Information for Service "App\Model\ArticleService"
==================================================

 --------------- -------------------------
  Option          Value
 --------------- -------------------------
  Service ID      App\Model\ArticleService
  Class           App\Model\ArticleService
  Tags            -
  Public          no
  Synthetic       no
  Lazy            no
  Shared          yes
  Abstract        no
  Autowired       yes
  Autoconfigured  yes
 --------------- -------------------------
```

## Afficher les infos du service avec les arguments du constructeur.

```
php bin/console debug:container App\Controller\ArticleController --show-arguments
```

```
Information for Service "App\Controller\ArticleController"
==========================================================

 --------------- ----------------------------------
  Option          Value
 --------------- ----------------------------------
  Service ID      App\Controller\ArticleController
  Class           App\Controller\ArticleController
  Tags            controller.service_arguments
                  container.service_subscriber
  Calls           setContainer
  Public          yes
  Synthetic       no
  Lazy            no
  Shared          yes
  Abstract        no
  Autowired       yes
  Autoconfigured  yes
  Arguments       Service(App\Model\ArticleService)
                  Service(App\Model\AuteurService)
 --------------- ----------------------------------
```

# Les Services Web.

- Début 2000.
  - Format XML : Echanges inter-applications.
  - Protocole SOAP : Echanger des messages XML sur HTTP.
- Fin 2000.
  - Remise en cause du format XML
    - Trop Verbeux
    - Pas aux "sites" Web.
  - Architecture REST
    - Pas un standard, utilise des standards existants.

# Architecture REST

Internet est constitué de ressources identifiées par des URLs uniques

Ex. : http://www.monjournal.fr/article/12

A éviter : http://www.monjournal.fr/article?id=12

URL — Query String

Verbes HTTP : Pour déterminer dans quel état on veut placer la ressource

- GET → Lecture
- DELETE → Suppression

Pas de corps de requête
Données transmises dans l'URL

- POST → Création
- PUT → Mise à jour

Possèdent un corps de requête
dans lequel on peut transmettre des données.

Convention !

# Gestion du format JSON dans Symfony

Service "serializer" fournit par le framework.


- serialize()
- deserialize()

# Mise en oeuvre : Conception de l'API REST.

Pour les articles :

| | | | |
|---|---|---|---|
| ✓ - Créer un article | POST | /article | Auteurs |
| ✓ - Afficher tous les articles | GET | /articles | Visiteurs. |
| - Afficher les articles du jour | GET | /article/today | |
| - Afficher un article par son id | GET | /article/{id} | |
| - Modifier un article par son id | PUT | /article | Auteurs. |
| - Supprimer un article par son id | DELETE | /article/{id} | |

Pour les auteurs :

| | | | |
|---|---|---|---|
| - Créer un auteur | POST | /auteur | Admins. |
| - Afficher tous les auteurs | GET | /auteurs | Visiteurs. |
| - Afficher un auteur par son identifiant | GET | /auteur/{identifiant} | |
| - Afficher les auteurs par nom et prénom | GET | /auteur/{nom}/{prenom} | |

# Le fichier security.yaml

security:

encoders: Définir les algo. de chiffrement pour les mots de passe.

role_hierarchy: Définir une hiérachie de permissions.

providers: Référentiels d'authentification. Où sont stockées les infos. de comptes utilisateurs.

firewalls: Contraintes d'authentification.

access_control: Contraintes d'autorisation.

# Authentification

1. Authentification HTTP

   http_basic: ~

   ↳ Pour les accès distants. (ex. FRONTEND ⟶ BACKEND)

2. Par formulaire.

   form_login: ~

   ↳ Pour les applications Web traditionnelles

# Mise en oeuvre : Sécurisation de l'API.

- Visiteurs (Sans rôle, permissions)
- Auteurs (ROLE_AUTEUR)
- Administrateurs (ROLE_ADMIN)