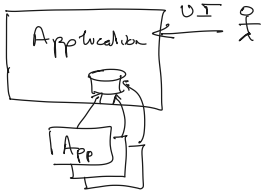
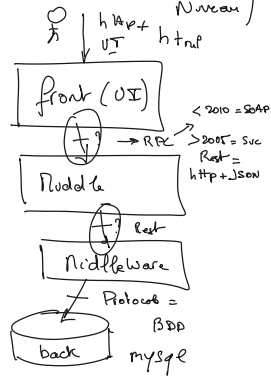
Architectures nTier

Monolithique :



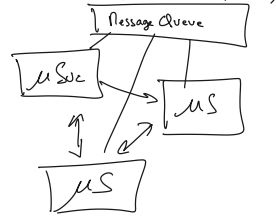
nTier (Couche / Niveau)



Si beaucoup d'applications =

architectures SOA (Service Oriented Archic.)

depuis > 2010 = micro Services -



/api/clients → 1 Projet

/commandes → 1 Projet

/produits → " Projet

/

URI comme identifiant :

/api/produits/1

Verbes HTTP : GET PUT
POST DELETE

Requête : ① Paramètre GET /api/produits/1 ? page=2
② header Requête : Cookie :
accept- :
Référence :
③ Corps/body { ... plus JSON }

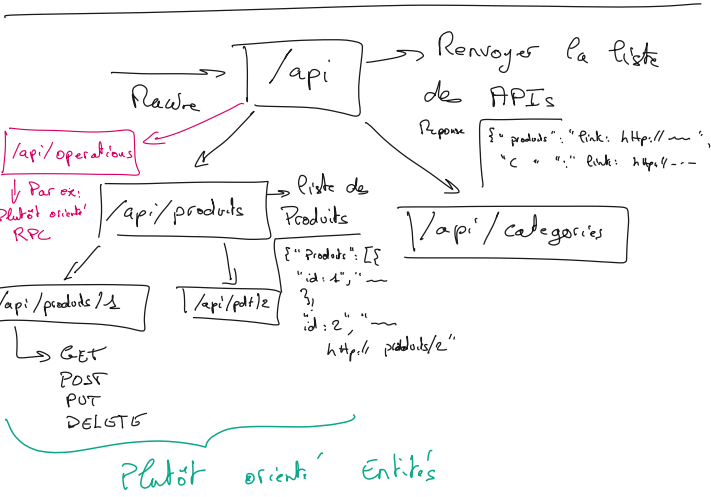
↓ Réponse

① headers Réponse clef : valeur
ex : Set-Cookie
Content-Type :
etc...

② Corps Réponse : { ..... JSON } (en général)

GET(safe)
• Lire une liste ou un élément (selon les champs de filtre transmis)
• HEAD (safe)
• Uniquement l'entête comme GET, mais sans le corps
• POST (non idempotent)
• Crée un nouveau contenu sans que son id ne soit transmis
• Cela peut aussi être enrichir l'état d'une ressource
• PUT (idempotent)
• Crée ou met à jour un contenu avec l'id dans l'URL
• PATCH
• Pour faire de la mise à jour partielle de contenu
• DELETE (idempotent)
• Suppression d'une ressource par son id, ou d'autres champs
• OPTIONS
• Découvrir les interfaces disponibles pour cette ressource

- Ne pas imbriquer les ressources
- GET: /authors/12/articles/
- Mais : GET: /articles/?author_id=12



Versionning Sémantique:

majeur . mineur . [revision [. build number]]

majeur: incrémenter si incompatibilité ascendante

mineur: incrémente si ajout, enrichissement de fct et respect de la compatibilité ascendante

revision: incrémente si optimisation, correction debugs, ... les interfaces ne changent pas

/api/produits

ou

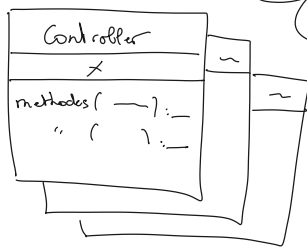
/api/1/produits (v1)

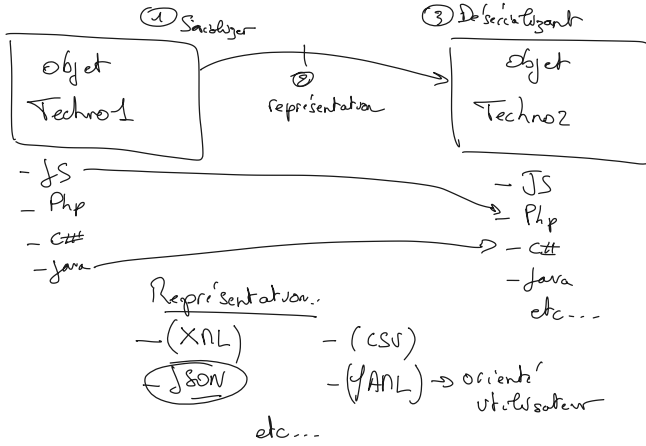
} v1 = numéro non spécifié

/api/2/produits (v2) → v2 en parallèle avec autres versions

---

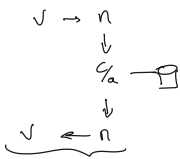
n Controller = 1 classe





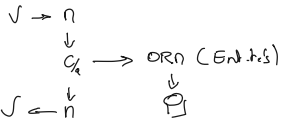
---

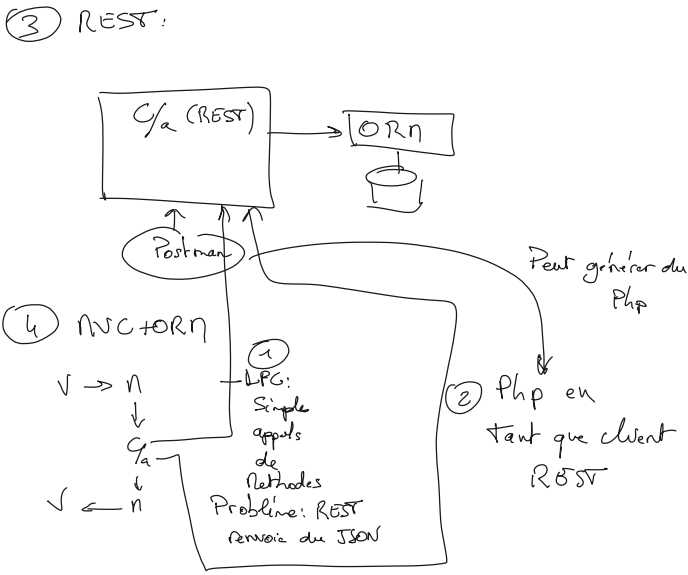
① → MVC 2 Tier



---

② MVC + ORM (doctrine)





Ajout de l'outil maker :
composer require --dev symfony/maker-bundle

Outils en ligne de commande :

composer
symfony
bin/console

Action RestFULL depuis un controleur MVC :

```
/**
    * @Route("/json", name="json")
    */
   public function j(Request $request): JsonResponse
   {
       $data = json_decode($request->getContent(), true);  // pas génial car je dois désérializer moi meme

       return new JsonResponse(['status' => 'Customer created!'], Response::HTTP_CREATED);
   }
```

Tester les APIs :

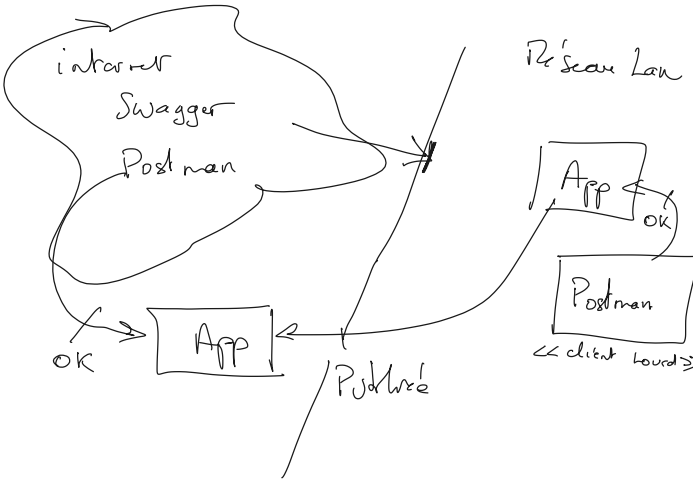
- Coder les Requêtes -
  - Javascript [/ jquery] } - Pas forcément efficace,
  - Php } - limité en Debug
    - Risques de bugs -
- Navigateur : On oublie
- Outils :
  - avec + simple : curl

Modèle de curl pour tester :
curl -i -H "Content-Type: application/json" -X POST -d '{"NOM":"Dalton", "PRENOM":"joe", "NAISSANCE":"2000-08-15", "VILLE":"Orleans"}' http://127.0.0.1:33939/home/json
0.1:33
-i : afficher les headers de la réponse
-H passer des headers en requete
-d : le body
-X : la méthode

Swagger → orienté conception / documentation

Postman → orienté tests / implémentation

↳ outil Local (client lourd)



---

1) Créer un compte Postman

2) Télécharger et exécuter en local

3) inscrire votre compte | en ligne
et | en local

Dev d'API Rest

Approche Symfony standard

API (1)

- Exposition d'entités ORM sous forme d'API
- → bien pour exposer des Tables
- → Peu (pas) de code
- → Pas du RPC (ne sont pas des API de code)

Controller (1)

- Renvoie - JSON
- Vue JSON
- → MVC JSON

Approche Communautaire

FOSRest Bundle

FOSRest

- → API par le code
- → a l'essentiel

Doc : https://github.com/FriendsOfSymfony/FOSRestBundle/tree/3.x/Resources/doc

---

(1) Créer un Projet de base (non FULL)

(2) Installer FOSRest

(3) Créer une première API / Tester via Postman

```
symfony new --dir=api api

composer require jms/serializer-bundle
composer require friendsofsymfony/rest-bundle
composer require symfony/validator
composer require sensio/framework-extra-bundle
option :
composer require symfony/maker-bundle --dev
composer require --dev symfony/profiler-pack
```

1. activer la config dans config/packages/fos_rest.yaml :

```
# Read the documentation:
https://symfony.com/doc/master/bundles/FOSRestBundle/index.html
fos_rest:
    param_fetcher_listener:  true
    allowed_methods_listener:  true
    routing_loader: false
    view:
        view_response_listener:  true
#    exception:
#        codes:
#            App\Exception\MyException: 403
#        messages:
#            App\Exception\MyException: Forbidden area.
    format_listener:
        rules:
            a.              - { path: ^/api, prefer_extension: true, fallback_format: json, priorities:
                [ json, html ] }
```

2) controller de base :

```
<?php

namespace App\Controller;

use FOS\RestBundle\Controller\AbstractFOSRestController;
use FOS\RestBundle\Controller\Annotations as Rest;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route("/api/demo", name="api_demo_")
 */
class DemoController extends AbstractFOSRestController
{

    /**
     * @Rest\Get("/", name="list")
     */
    public function list()
    {
        $data = [];
        $view = $this->view($data, 200);

        return $this->handleView($view);
    }
}
```

Travailler avec l'injection d'objets :
1) composer require sensio/framework-extra-bundle

2) Spécifier un type en entrée :

```
* @ParamConverter("article", converter="fos_rest.request_body")
     */
    // nécessite composer require symfony/monolog-bundle
    public function list(LoggerInterface $logger, Request $request, Article $article )
```

3) il faut créer cette classe (on peut passer par l'orm)
composer require orm
bin/console make:entity -> répondez aux questions...

ce qui génère :

```
<?php

namespace App\Entity;

use App\Repository\ArticleRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ArticleRepository::class)
 */
class Article
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $title;

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

    public function getTitle(): ?string
    {
        return $this->title;
    }

    public function setTitle(string $title): self
    {
        $this->title = $title;

        return $this;
    }
}
```

Pour que le convetisseur fonctionne, il faut l'activer :
dans fos_rest.yaml :

```
# Read the documentation: https://symfony.com/doc/master/bundles/FOSRestBundle/index.html
fos_rest:
    body_converter:
        enabled: true
```

Ce qui donne :

```
use FOS\RestBundle\Controller\AbstractFOSRestController;
use FOS\RestBundle\Controller\Annotations as Rest;
use Symfony\Component\Routing\Annotation\Route;
use Psr\Log\LoggerInterface;
use Symfony\Component\HttpFoundation\Request;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\ParamConverter;
use App\Entity\Article;


/**
 * @Route("/api/demo", name="api_demo_")
 */
class DemoController extends AbstractFOSRestController
{

    /**
     * @Rest\Get("/", name="list")
     * @ParamConverter("article", converter="fos_rest.request_body")
     */
    // nécessitye composer require symfony/monolog-bundle
    public function list(LoggerInterface $logger, Request $request, Article $article )
    {
        $data = [1,2];
        $formatted[] = [
            'id' => 'id',
            'name' => 'thename',
            'address' => 'address',
        ];
        $got = [ $request->get('place_id') ];   // Postman : Query params
        $hpost = [ $request->headers->get('myheader') ];   // Postman : Headers params
        $body = [ "Le titre " => $article->getTitle() ];

        $view = $this->view($body, 200);

        return $this->handleView($view);
    }
```

ORM

ORM sous Symfony

implémenter une BDD postgres sous Docker

y injecter un schema d'exemple

Créer des Entités Doctrine

Rendre ces entités accessible en API via Symfony API

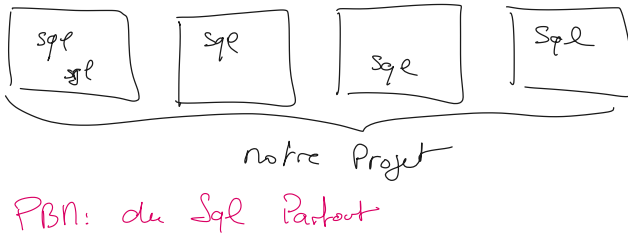
les

nom du FW d'API Symfony pour entités

Le Design Pattern ORM

- Bonne pratique -

Pbm Apps monolithiques



PBM: du Sql Partout

→ Prog Orientée Objet

⇒ implémenter le Pattern ORM

DTO : Data Transfer Objects :

Simple classe POPO = Plain Old Php Object

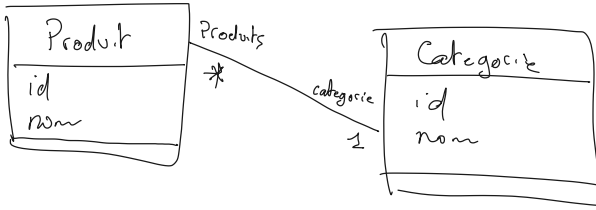
POJO = Java

POCO = C#

Table Produits = classe Produit

Champs Nom Varchar Propr: nom (string)

UML



2] DAO → Data Access Object

→ N'est pas créé par Doctrine

→ A vous de les créer => Recommandation

→ Souvent dans les exemples, tutos etc...
on ne les retrouves pas.

→ Directement du Doctrine dans le front

→ Le Framework Rest API de Symfony

→ Implémente cela pour nous
derrière la scène

→ Les méthode GET, POST, PUT, etc...
seront l'équivalent DAO via REST

Docker

Virtualisation / Isolation de Process, Réseau, File System etc...

Très légé = Isolation logique

Basé sur les "namespaces"
fonctionnalité du Noyau (linux, windows)

Docker est l'outil pour exploiter les namespaces

→ il faut l'installer

"Conteneur" = instance d'une "image"

image = "comme" une distrib linux

Un Conteneur Docker est NanoProcess

→ Pas de Service

→ Pas recommandé d'avoir de Process Background (avec &)

image depuis hub.docker.com

Pour Lancer un SRV PG Conteneurisé:

1. installer Docker — apt install docker.io
2. Lancer un conteneur avec les bons Paramètres.

docker container run --name postgres -d -p 5432:5432 --restart unless-stopped -e POSTGRES_PASSWORD=password postgres

-p <port du host>:<port dans le conteneur>

Lister les images :
docker image ls
Lister les conteneurs :
docker container ls

Lancer un client postgres dans le conteneur postgres :
docker container exec -it postgres psql -U postgres
docker container exec **-it : veut dire "interactif"** <nom du conteneur:postgres>
<commande à y exécuter:psql -U postgres>

Dans postgres, créer une BDD comptoirs :
create database comptoirs;

sudo docker container cp create_comptoir.sql postgres:/create_comptoirs.sql
sudo docker container cp drop_comptoirs.sql postgres:/drop_comptoirs.sql
sudo docker container cp pg_comptoir.sql postgres:/pg_comptoirs.sql

Lancer les scripts :
sudo docker container exec -it postgres psql -U postgres -d comptoirs -f /pg_comptoirs.sql

---

Créer le projet
injecter les dépendances :
composer require symfony/maker-bundle --dev
composer require symfony/orm-pack
composer require symfony/annotations-pack -W

paramétrer le fichier .env :
DATABASE_URL="postgresql://postgres:password@127.0.0.1:5432/comptoirs"

Importer les entités :
bin/console doctrine:mapping:import Client --path=src
php bin/console doctrine:mapping:import "App\Entity" annotation --path=src/Entity

TESTS UNITAIRES :

composer require --dev phpunit/phpunit symfony/test-pack
Pour lancer les tests :
php ./vendor/bin/phpunit

Créer une classe de Tests :
bin/console make:test TestCase DatabaseTest

Attention a la config/packages/test/doctrine.yaml : il y a un préfixe au nom de la base de données.

Le reverse engineering de Doctrine ne génère pas les getter/setters
Vous pouvez demander au générateur d'entités de le faire :
php bin/console make:entity --regenerate
Puis appuyez simplement sur entrée pour tout autoriser.

```
<?php

namespace App\Entity;

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

/**
 * Produits
 *
 * @ORM\Table(name="produits", indexes={@ORM\Index(name="IDX_BE2DDF8CE538AE0C",
columns={"code_categorie"}), @ORM\Index(name="IDX_BE2DDF8CAC8812A6",
columns={"no_fournisseur"})})
 * @ORM\Entity
 */
class Produits
{
    /**
     * @var int
     *
     * @ORM\Column(name="ref_produit", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="SEQUENCE")
     * @ORM\SequenceGenerator(sequenceName="produits_ref_produit_seq", allocationSize=1,
initialValue=1)
     */
    private $refProduit;
```

```
/**
     * @var string
     *
     * @ORM\Column(name="nom_produit", type="string", length=40, nullable=false)
     */
    private $nomProduit;

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\ManyToMany(targetEntity="Commandes", mappedBy="refProduit")
     */
    private $noCommande;

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

    public function getRefProduit(): ?int
    {
        return $this->refProduit;
    }

    public function getNomProduit(): ?string
    {
        return $this->nomProduit;
    }

    public function setNomProduit(string $nomProduit): self
    {
        $this->nomProduit = $nomProduit;

        return $this;
    }
```

DatabaseTest.php :

```
<?php

namespace App\Tests;

use PHPUnit\Framework\TestCase;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

use App\Entity\Produits;

class DatabaseTest extends KernelTestCase
{
   private $em;

   protected function setUp(): void
   {
      $kernel = self::bootKernel();

      $this->em = $kernel->getContainer()->get('doctrine')->getManager();
   }

   public function testEntities(): void
   {
      $produit = $this->em->getRepository(Produits::class)->findOneBy(['nomProduit'
=> 'Chai']);
      $this->assertSame('Chai', $produit->getNomProduit());
   }

public function testGetProducts(): void
   {
      $q = $this->em->createQuery("select p from App\Entity\Produits p where
p.prixUnitaire>1000");
      $produits = $q->getResult();

      $this->assertCount(0, $produits, "Aucun produits de la requete");
   }
}
```

st1@StudentVM:~/orm-api/sql$ ./vendor/bin/phpunit
PHPUnit 9.5.10 by Sebastian Bergmann and contributors.

Testing
..   2 / 2 (100%)

Time: 00:00.478, Memory: 18.00 MB

OK (2 tests, 2 assertions)
st1@StudentVM:~/orm-api/sql$

La reference de Doctrine :
https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/working-with-objects.html#querying

Pour mettre nos entités en tant que API Rest :
You can also mark this class as an API Platform resource. A hypermedia CRUD API will automatically be available for this entity class:

composer require api

Dans chaque Entité, vous devez :

Entetes :
use Symfony\Component\Serializer\Annotation\Groups;
use ApiPlatform\Core\Annotation\ApiResource;

* @ApiResource()
 */
class Client

La Reference de API Platform :
https://api-platform.com/docs/core

```
/**
 * Clients
 *
 * @ORM\Table(name="clients")
 * @ORM\Entity
 * ApiResource(
 *     collectionOperations={"get"={"normalization_context"={"groups"="conference:list"}}},
 *     itemOperations={"get"={"normalization_context"={"groups"="conference:item"}}},
 *     order={"year"="DESC", "city"="ASC"},
 *     paginationEnabled=false
 * )
 * Filtrage pour n'afficher en Q ( CRUDQ : tous les elments ), uniquement l'id et le name :
 * @ApiResource(
 *     collectionOperations={"get"={"normalization_context"={"groups"="client_listing:read"}}},
 * )
 */
class Client
{
    /**
     * @var string
     *
     * @ORM\Column(name="code_client", type="string", length=5, nullable=false, options={"fixed"=true})
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="SEQUENCE")
     * @ORM\SequenceGenerator(sequenceName="clients_code_client_seq", allocationSize=1, initialValue=1)
     * @Groups({"client_listing:read"})
     */
    private $codeClient;
```

Pour Développer en HTTP Client d'appli REST :
1) Soit du code généré depuis Postman :
a) <?php
$client = new http\Client;
$request = new http\Client\Request;
$request->setRequestUrl('http://127.0.0.1:8001/api/clients/ALFKI');
$request->setRequestMethod('GET');
$request->setOptions(array());

$client->enqueue($request)->send();
$response = $client->getResponse();
echo $response->getBody();

b)<?php
require_once 'HTTP/Request2.php';
$request = new HTTP_Request2();
$request->setUrl('http://127.0.0.1:8001/api/clients/ALFKI');
$request->setMethod(HTTP_Request2::METHOD_GET);
$request->setConfig(array(
  'follow_redirects' => TRUE
));
try {
  $response = $request->send();
  if ($response->getStatus() == 200) {
    echo $response->getBody();
  }
  else {
    echo 'Unexpected HTTP status: ' . $response->getStatus() . ' ' .
    $response->getReasonPhrase();
  }
}
catch(HTTP_Request2_Exception $e) {
  echo 'Error: ' . $e->getMessage();
}

c)<?php

$curl = curl_init();

curl_setopt_array($curl, array(
  CURLOPT_URL => 'http://127.0.0.1:8001/api/clients/ALFKI',
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_ENCODING => '',
  CURLOPT_MAXREDIRS => 10,
  CURLOPT_TIMEOUT => 0,
  CURLOPT_FOLLOWLOCATION => true,
  CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
  CURLOPT_CUSTOMREQUEST => 'GET',
));

$response = curl_exec($curl);

curl_close($curl);
echo $response;

2) Soit via Symfony :
https://symfony.com/doc/current/http_client.html

Http Client Symfony :

composer require symfony/http-client

Code pour pouvoir tester vos APIs depuis les tests Unitaires :
https://symfony.com/doc/current/http_client.html#full-example