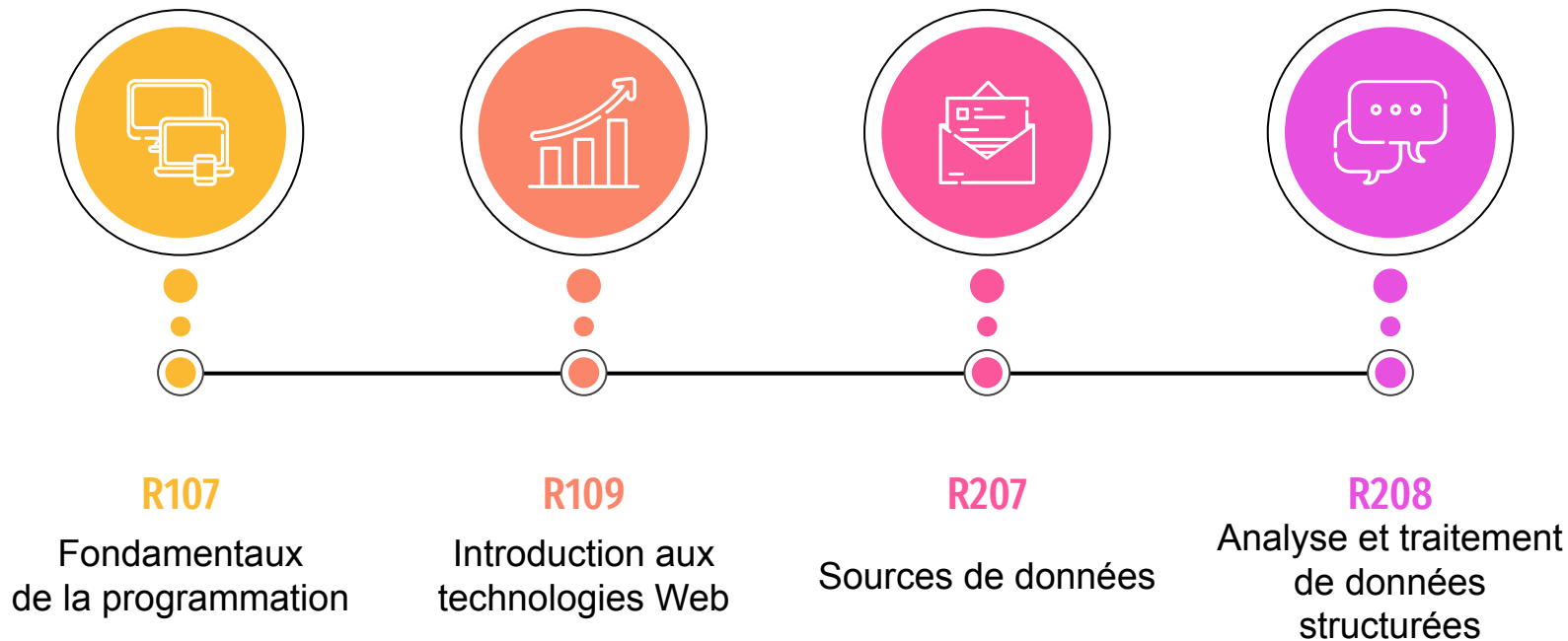


# **R209**

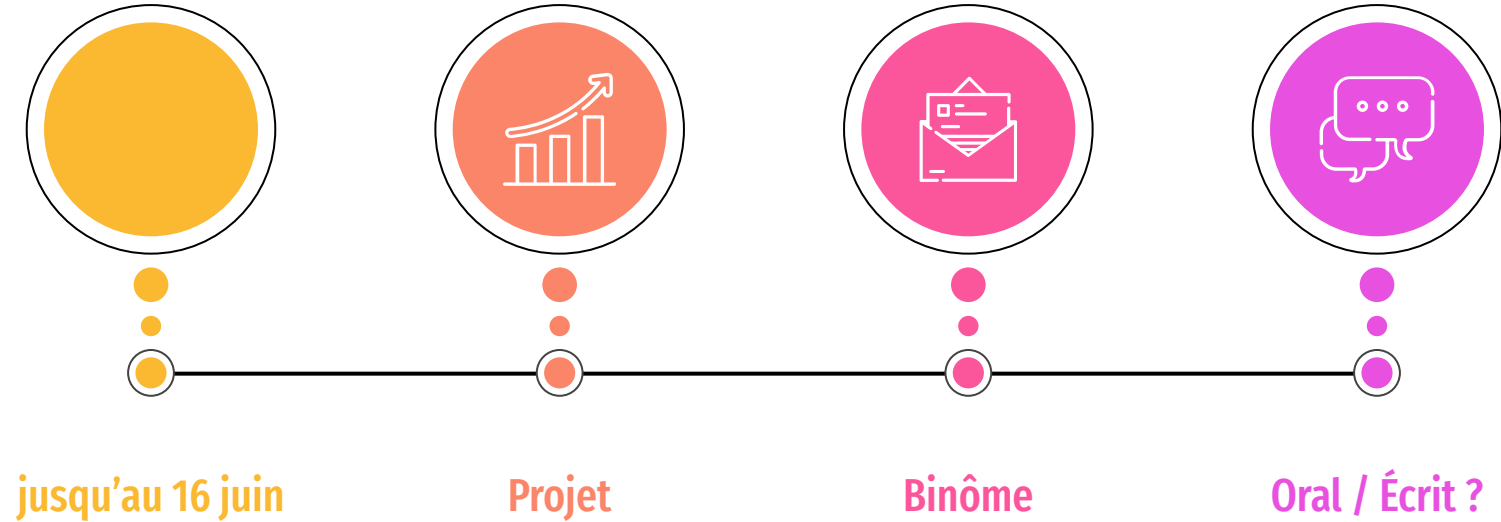
## **Initiation au développement web**

# Prérequis



if (lacunes>0) then do\_révisions([R107,R109,R207,R208,..]);

# Calendrier et méthode de travail



# Contenus



## Mise en forme de pages Web

- balises HTML avancées ;
- structure d'une page avec son DOM ;
- CSS avancé ou Framework ;
- initiation au dynamisme côté client



## Interaction client-serveur

Interrogation d'un SGBD ou d'une API

Des  
compétences  
transversales

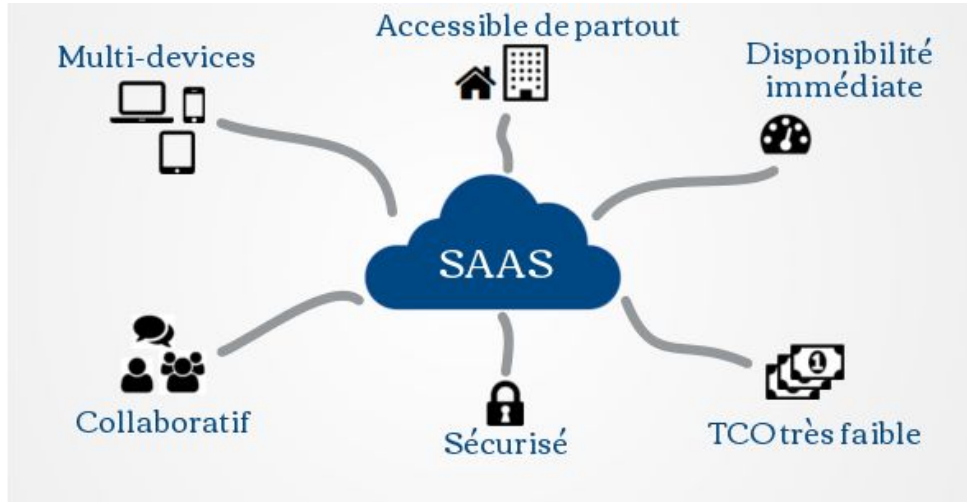


## Sensibilisation à la sécurisation de sites

failles XSS, XSS stockée, injections SQL



## Scripts côté serveur.



## Projet en mode Saas

Le **mode Saas** est la mise à disposition d'un logiciel accessible aux utilisateurs via internet.

Aucune installation sur les serveurs de l'entreprise cliente n'est requise. Chaque utilisateur dispose d'un compte, avec des niveaux de droit variables, lui permettant d'accéder au logiciel.

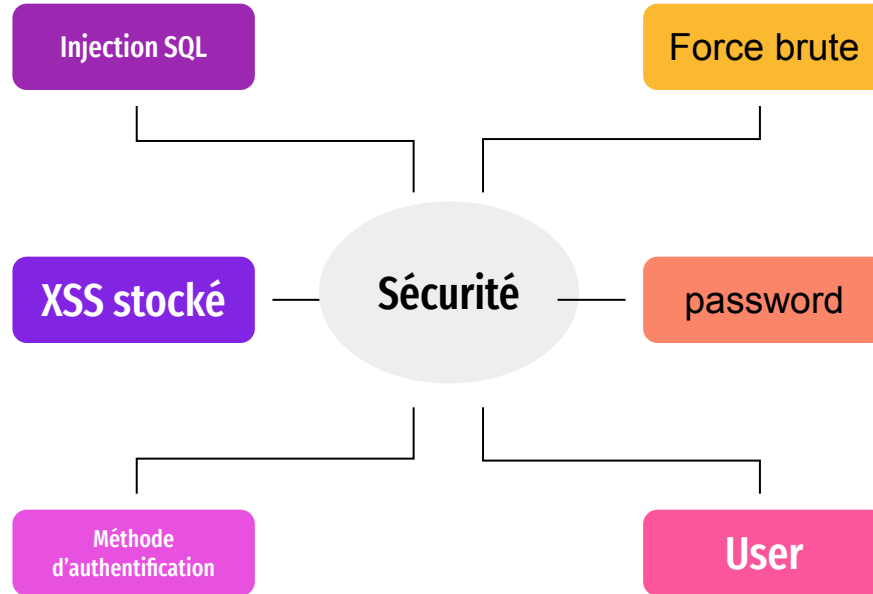
# Informations sur la sécurité

# Quelques failles de sécurité

Groupe de méthodes d'exploitation de faille de sécurité d'une application interagissant avec une base de données. Elle permet d'injecter dans la requête SQL en cours un morceau de requête non prévu par le système et pouvant compromettre la sécurité.

Le cross-site scripting est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page

Stratégie de sécurité



L'attaque par force brute est une méthode utilisée en cryptanalyse pour trouver un mot de passe ou une clé. Il s'agit de tester, une à une, toutes les combinaisons possibles. Cette méthode est en général considérée comme la plus simple concevable

Faiblesse des passwords

L'utilisateur de l'application



# ANSSI

Agence nationale de sécurité des systèmes  
d'information.



SecNum*académie*.gouv.fr

Formez-vous à la sécurité du numérique

# Bienvenue sur le MOOC de l'ANSSI.

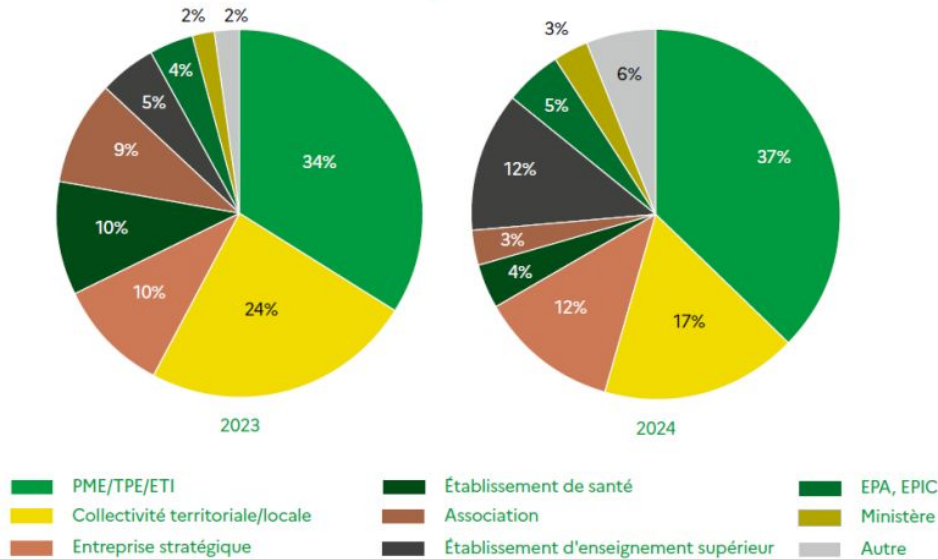
Vous y trouverez l'ensemble des informations pour vous **initier à la cybersécurité**, approfondir vos connaissances, et ainsi **agir efficacement sur la protection de vos outils numériques**. Ce dispositif est accessible gratuitement. Le suivi intégral de ce dispositif vous fera bénéficier d'une attestation de réussite.

Accéder au MOOC de l'ANSSI

Pour information

<https://secnumacademie.gouv.fr/>

Répartition des victimes d'attaques par le biais de rançongiciels



Chiffres 2024

## LES CHIFFRES CLÉS

## MATURITÉ CYBER DES TPE-PME\*

62%

pensent être **faiblement exposées** aux attaques ou l'ignorent

78%

se disent **non préparées** à une attaque ou l'ignorent

7/10



allouent moins de **2 000 €** à la cybersécurité

10%

seulement prévoient d'augmenter **ce budget**



15%

déclarent avoir déjà été **victime d'une cyberattaque** durant les 12 derniers mois

65%

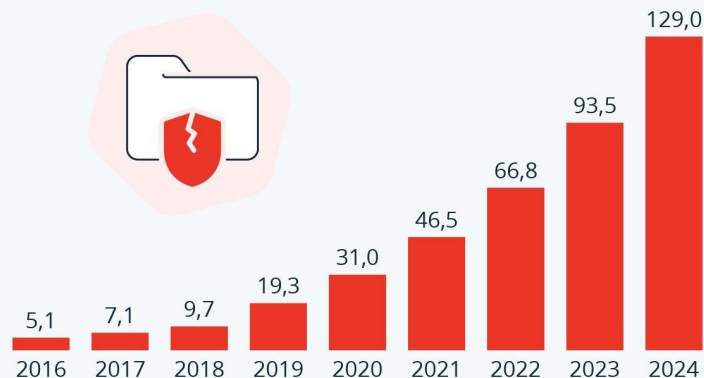
pensent qu'elles ne sauraient pas **évaluer les impacts**

\*Étude 2024 conduite par OpinionWay pour Cybermalveillance.gouv.fr du 10 juin au 16 juillet en ligne (CAWI) auprès d'un échantillon de 513 entreprises interrogées en France métropolitaine et Départements et Régions d'Outre-Mer.

## Chiffres 2024

### Le coût des cyberattaques explose en France

Estimation du coût annuel de la cybercriminalité en France, en milliards de dollars américains



Source : Statista Technology Market Insights



statista

Le sigle RGPD signifie « **Règlement Général sur la Protection des Données** » (en anglais « General Data Protection Regulation » ou GDPR).

Le RGPD encadre le traitement des données personnelles sur le territoire de l'Union européenne.



# Le RGPD

[david.lacan@aresformation.com] Detection d'une attaque sur l'IP 51.222.75.2



noreply@soyoustart.com

À moi ▼

SAS OVH - <https://www.soyoustart.com>

2 rue Kellermann

BP 80157

59100 Roubaix

Madame, Monsieur,

Nous venons de détecter une attaque sur l'adresse IP 51.222.75.212.

Afin de protéger votre infrastructure, nous avons aspiré votre trafic sur notre infrastructure de **mitigation**.

Toute l'attaque sera ainsi filtrée par notre infrastructure, et seul le trafic légitime arrivera jusqu'à vos serveurs.

A la fin de l'attaque, votre infrastructure sera immédiatement retirée de la **mitigation**.

Pour plus d'informations sur l'infrastructure de **mitigation** OVH : <http://www.soyoustart.com/fr/anti-ddos/>

Cordialement,

L'équipe So you Start

Vous avez une question ?

Contactez notre support technique et commercial par email ou au 09 72 100 111.

N'hésitez pas à rejoindre notre communauté sur le forum : <https://community.ovh.com/>

So you Start

SAS OVH - <https://www.ovh.com/>

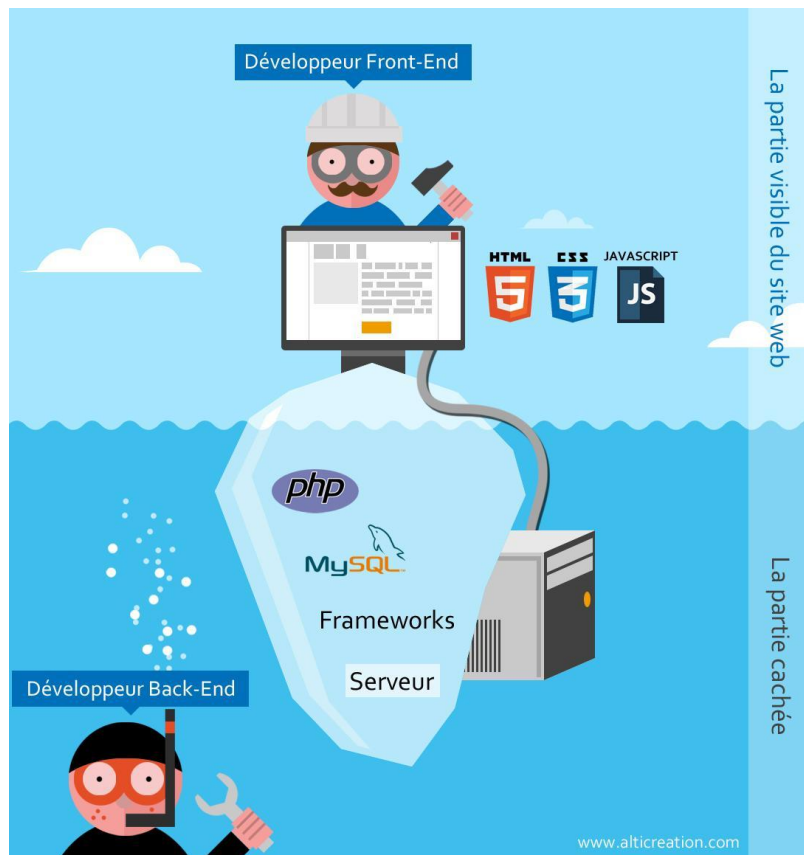
2 rue Kellermann

BP 80157

59100 Roubaix

# exemple attaque DDOS

# Le développement Web



# Le développement Web





# La révolution de l'IA



# Dans le code

- **Automatisation de la programmation**
- **Développement assisté par IA**
- **Amélioration des tests et de la détection des bugs**
- **Optimisation des performances**
- **Gestion et analyse des données**
- **Développement de nouvelles compétences**

# Dans le travail

- => Transformation des compétences et des emplois
- => fin de la cathédrale de l'esprit



# Difficultés

- Transversalité des savoirs
- Besoins des acquis
- Monter en compétences
- Mise en place de l'environnement de travail qui peut être chronophage

# Rappel notion Base de Données

## 1) clé primaire

- a) La clé primaire est un ou plusieurs attributs d'une table qui permet d'identifier de manière unique chaque enregistrement (ou ligne) de cette table. Elle assure l'unicité et l'intégrité des données au sein de la table.

## 2) clé étrangère

- a) Une clé étrangère (Foreign Key) est un ou plusieurs attributs d'une table qui établit et impose un lien entre les données de deux tables. Elle est utilisée pour maintenir l'intégrité référentielle entre les tables en s'assurant que les valeurs de la clé étrangère dans une table correspondent aux valeurs de la clé primaire dans une autre table.

# Rappel notion Base de Données : les relations

## **OneToOne (Un-à-Un)**

Une relation OneToOne signifie qu'une entité est associée à une seule instance d'une autre entité et vice versa. C'est souvent utilisé pour représenter des relations exclusives entre deux entités.

## **OneToMany (Un-à-Plusieurs)**

Une relation OneToMany signifie qu'une entité est associée à plusieurs instances d'une autre entité. C'est souvent utilisé pour représenter des collections ou des listes d'éléments.

Une catégorie peut contenir plusieurs produits/Chaque produit appartient à une seule catégorie.

## **ManyToOne (Plusieurs-à-Un)**

Une relation ManyToOne signifie que plusieurs instances d'une entité peuvent être associées à une seule instance d'une autre entité. Cette relation est souvent la contrepartie de la relation OneToMany.

Un client peut passer plusieurs commandes /Chaque commande est associée à un seul client.

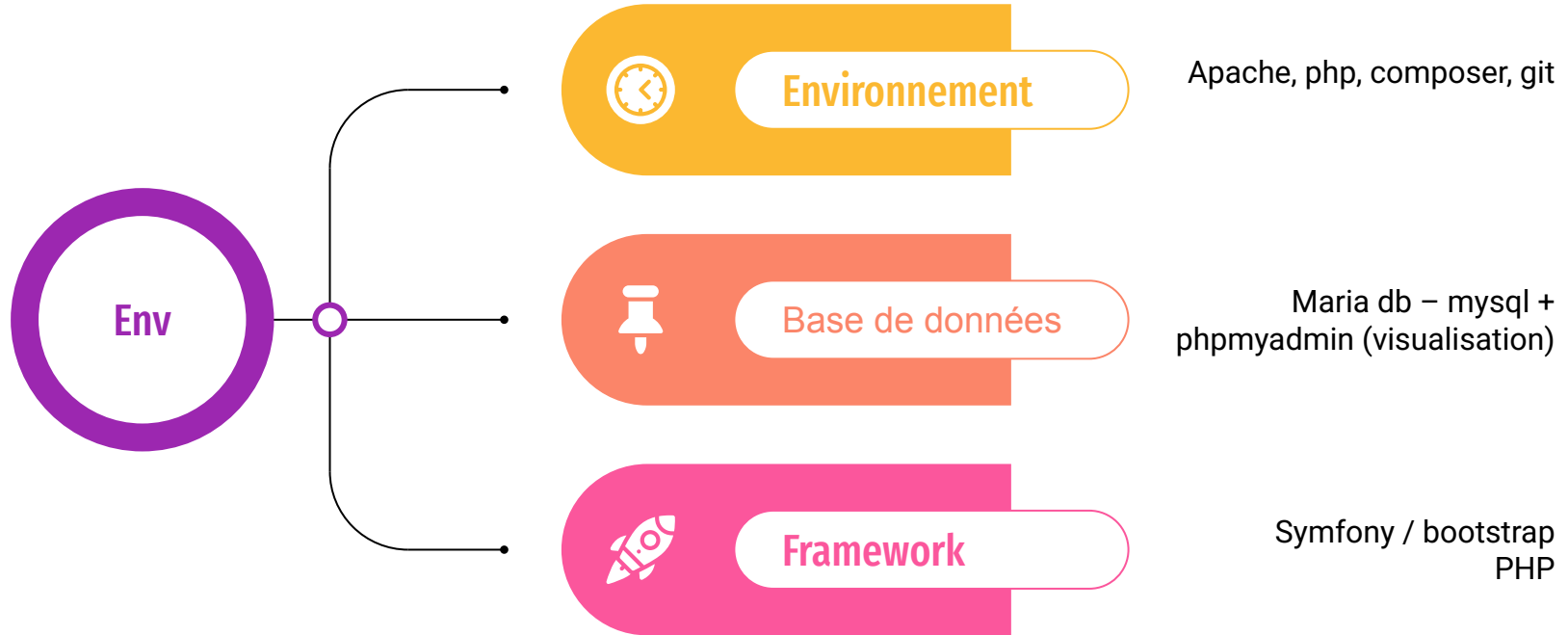
## **ManyToMany (Plusieurs-à-Plusieurs)**

Une relation ManyToMany signifie que plusieurs instances d'une entité peuvent être associées à plusieurs instances d'une autre entité. C'est souvent utilisé pour représenter des relations bidirectionnelles complexes où plusieurs entités peuvent être liées de manière non exclusive.

# Rappel notion Base de Données : les relations

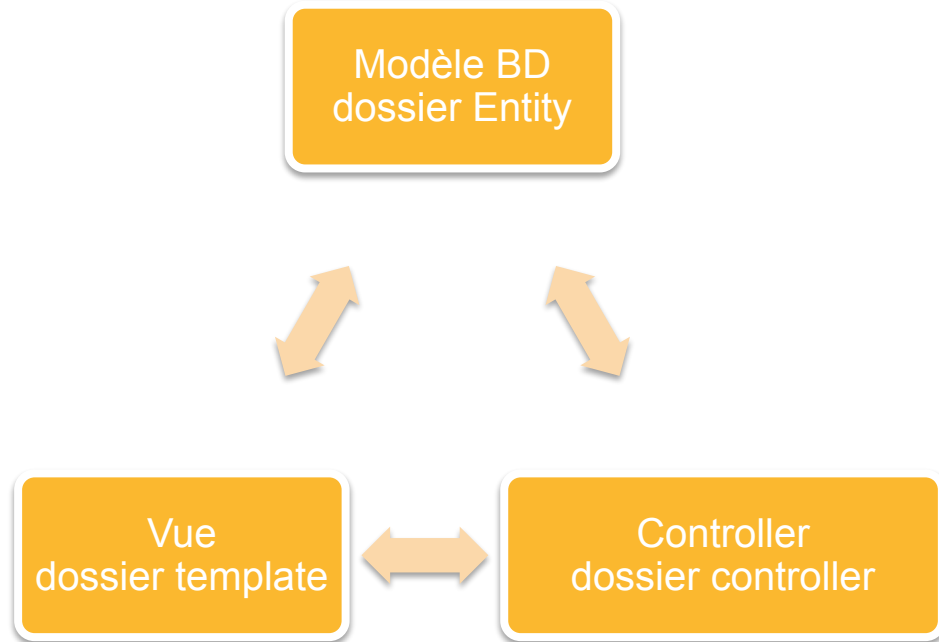
- Une formation a un type : ManyToOne
- Une formation a un niveau : ManyToOne
- Une experience a un type : ManyToOne
- une AF-Veille a un type : ManyToOne
- **Plusieurs formations peuvent avoir plusieurs compétences : ManyToMany**
- **Plusieurs expériences peuvent avoir plusieurs compétences : ManyToMany**
- **Plusieurs Af-Veille peuvent avoir plusieurs compétences : ManyToMany**

# Environnement Technique

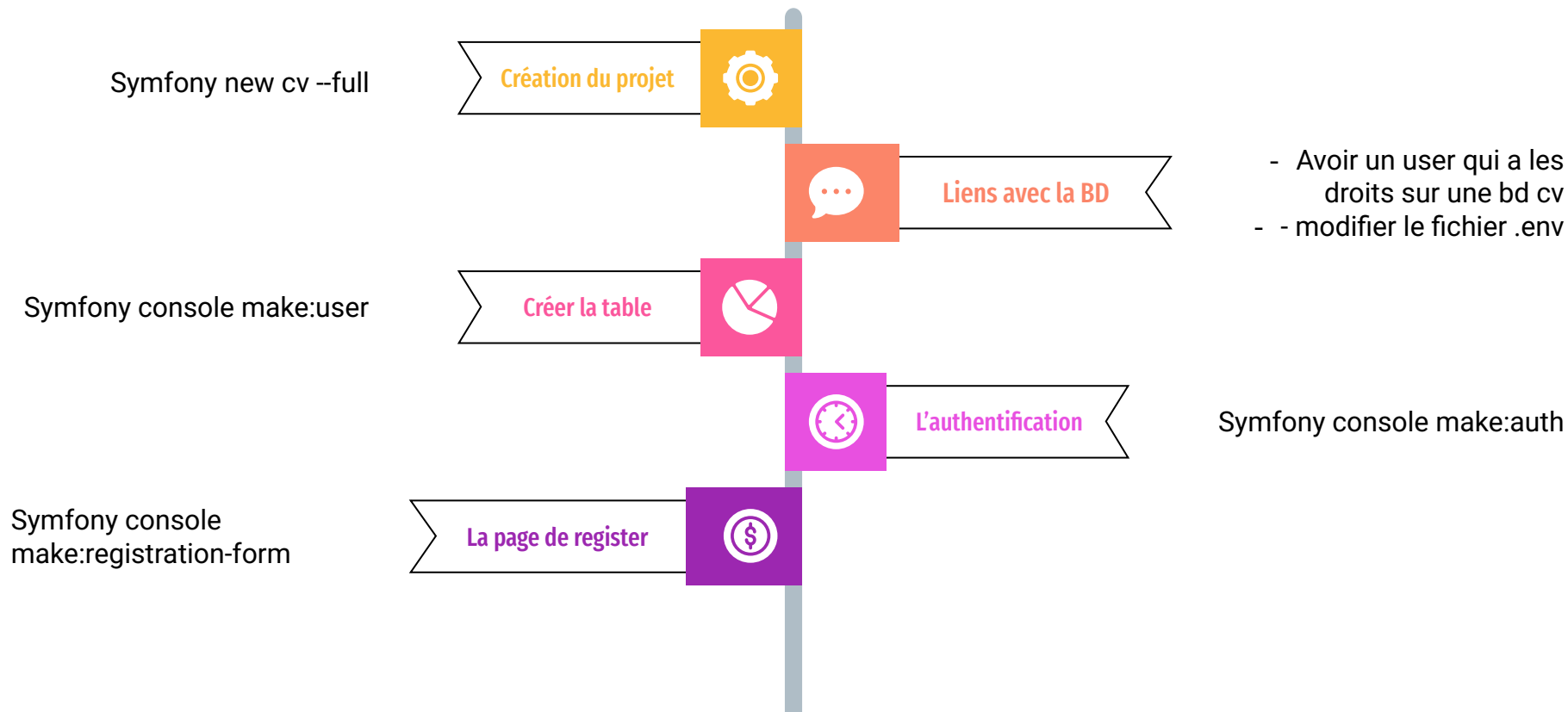




# Modèle MVC



# Démarrage projet symfony



# Arborescence

src / :

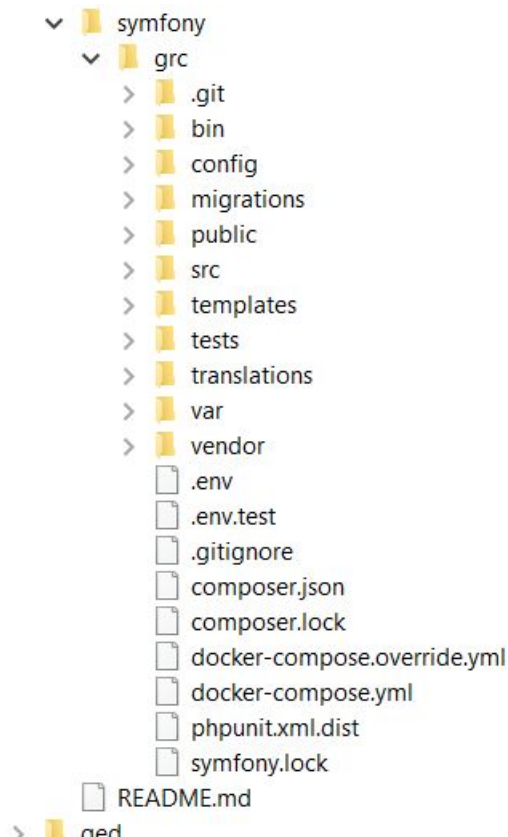
- Controller tous les contrôleurs
- Entity toutes les tables sous la forme Nom.php
- Repository un fichier par table pour créer des requêtes

templates/ :

- Liste des dossiers (un par controller) -> fichier twig (front)
- Fichier base.html.twig qui va contenir notre template et le menu

.env :

- Fichier qui permet d'identifier le user et la base de données



# Support pour ce module



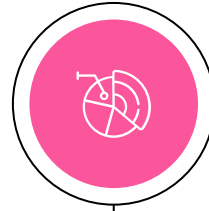
Documentation officielle

Symfony, bootstrap ...



Code sur git

Code sur github



IA

IA



google

...

# Pour la bonne réussite de ce module

## Les prérequis

Avoir les bases  
Ne pas prendre du retard



Comprendre le  
fonctionnement de  
symfony



Routage, interrogation  
bd



## Modèle – base de données

Créer le modèle  
rapidement,  
correctement et  
définitivement

utiliser les outils

bootstrap, bootswatch

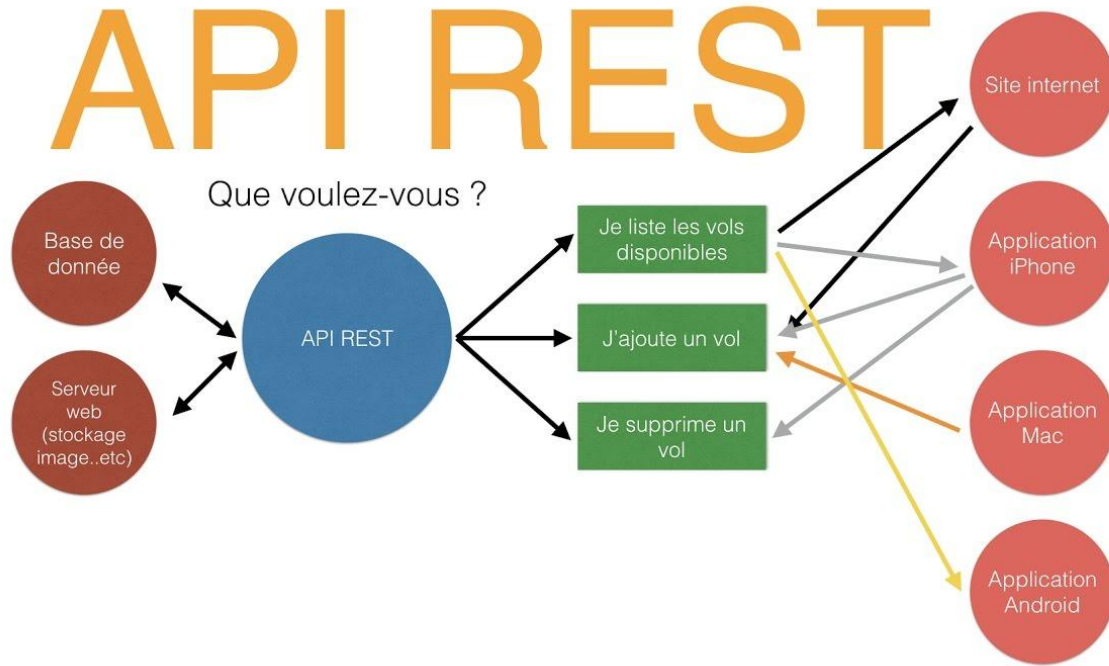
## Env de développement stable

Env portable  
Facilement  
reprogrammable  
sauvegarde

# Création de l'API

Une **API (Application Programming Interface)** est un ensemble de règles et de protocoles qui permet à différentes applications logicielles de communiquer entre elles. Elle définit les méthodes et les formats de données que les applications peuvent utiliser pour échanger des informations et accéder aux fonctionnalités d'un système logiciel.

# Créer une API



# La sérialisation

## SÉRIALISATION DES DONNÉES

### VARIABLE

```
-----  
$posts = [...]  
$post = new Post()
```



### REPRESENTATION

```
-----  
N'importe quel format de  
représentation textuelle  
(JSON, XML, CSV,  
YML ...)
```



# Normalisation / Sérialisation

- La normalisation permet de transformer des objets en tableau associatif
- La sérialisation permet de transformer des tableaux associatifs en chaîne de caractère (XML, json, csv, ...)

# Que souhaitons nous

Nous souhaitons que lorsqu'un navigateur ou un code quelconque se rend à une adresse précise, la plateforme va chercher des informations à la base de données qui lui renvoie des données interprétables et compréhensibles facilement.

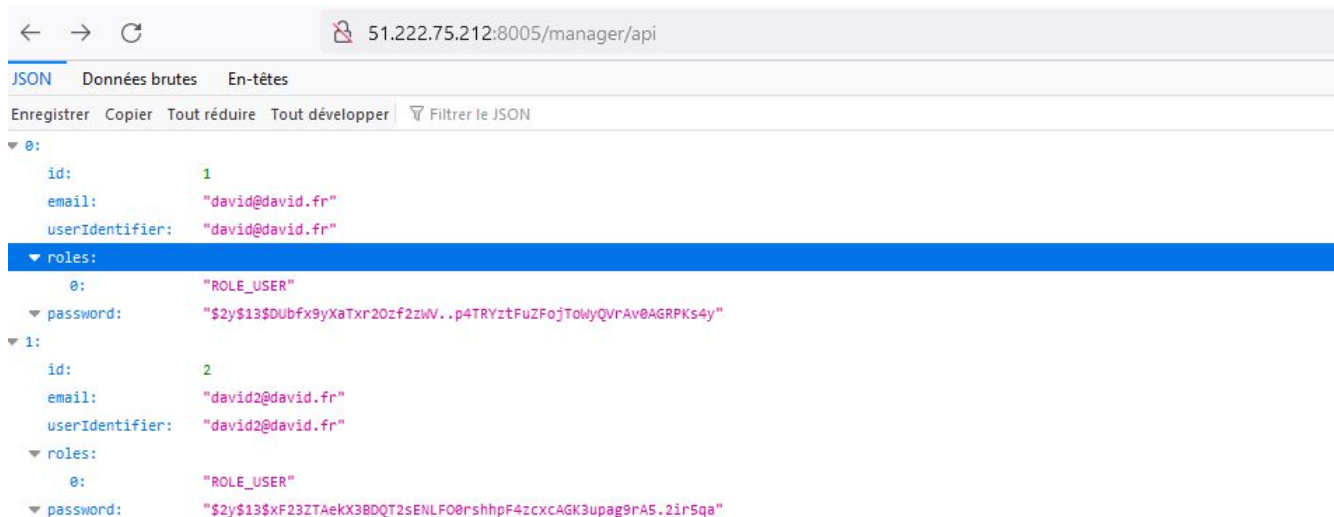
Obtenir un format json pour remplacer nos objets php qui arrivent sous forme de tableaux associatifs.

## **Idée générale :**

Lorsque l'on reçoit une requête HTTP , on répond par du texte. (json, csv, xml, ...)

# Exemple : liste des users de mon application

L'adresse : <http://51.222.75.212:8005/manager/api> renvoie la liste des user de mon application



```
{
  "0": {
    "id": 1,
    "email": "david@david.fr",
    "userIdentifier": "david@david.fr",
    "roles": {
      "0": "ROLE_USER",
      "password": "$2y$13$DUBfx9yXaTxr2Ozf2zW..p4TRYztFuZFoJtWMyQVrAv0AGRPKs4y"
    }
  },
  "1": {
    "id": 2,
    "email": "david2@david.fr",
    "userIdentifier": "david2@david.fr",
    "roles": {
      "0": "ROLE_USER",
      "password": "$2y$13$xF232TAekX3BDQT2sENLFO0rshhpF4zxcAGK3upag9rA5.2ir5qa"
    }
  }
}
```

# Exemple : liste des users de mon application

## Code de la route /manager/api

```
1 <?php
2
3 namespace App\Controller;
4
5 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
6 use Symfony\Component\HttpFoundation\Response;
7 use Symfony\Component\Routing\Annotation\Route;
8 use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
9 use App\Entity\User;
10 use App\Repository\UserRepository;
11 use Symfony\Component\Serializer\SerializerInterface;
12 use Symfony\Component\Serializer\Serializer;
13
14
15 class ManagerController extends AbstractController
16 {
17
```

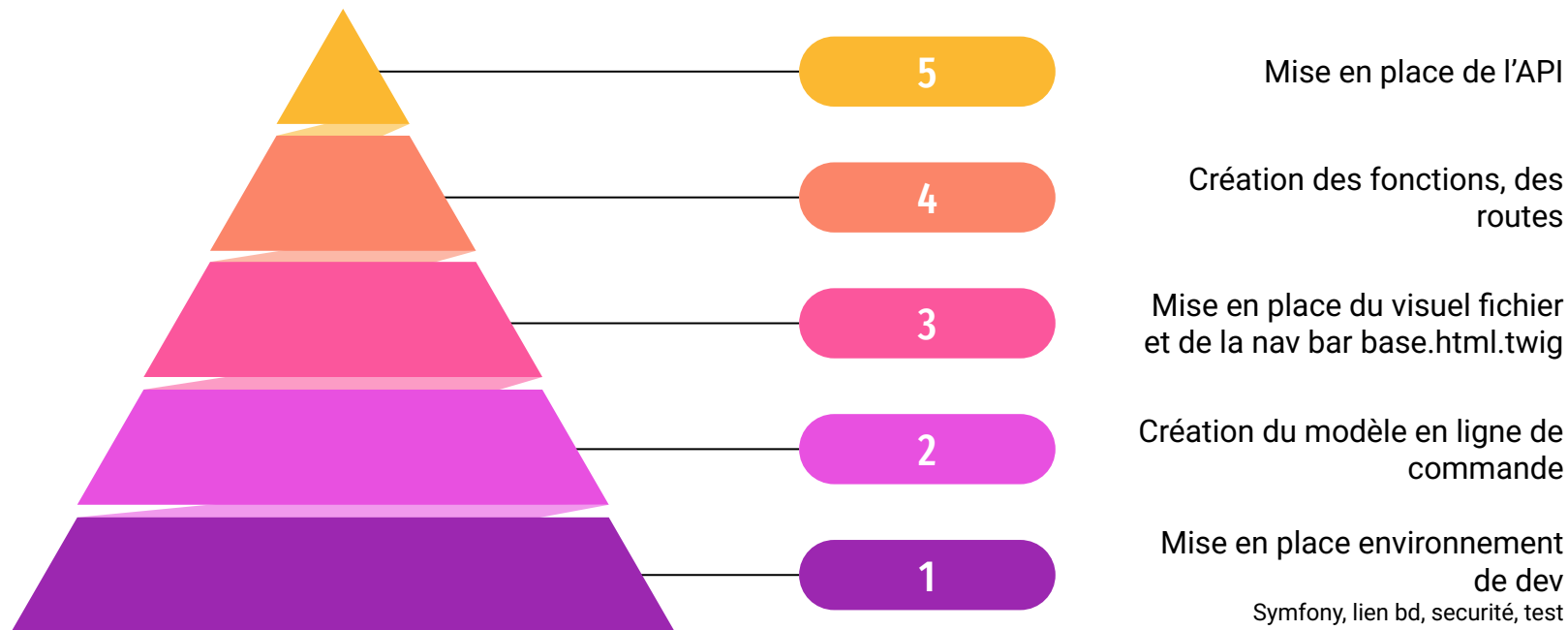
```
/**
 * @Route("/manager/api", name="app_manager_api")
 * @Method({"GET", "POST"})
 */
public function api(UserRepository $userRepo): Response
{
    $recupUserObject = $userRepo->findAll();
    $response = $this->json($recupUserObject, 200, []);
    return $response;
}
```

# Exemple : liste des users de mon application

Code de la route /manager/api

```
/**
 * @Route("/manager/api", name="app_manager_api")
 * @Method({"GET", "POST"})
 */
public function api(UserRepository $userRepo): Response
{
    $recupUserObject = $userRepo->findAll();
    $response = $this->json($recupUserObject, 200, []);
    return $response;
}
```

# Feuille de route



# La technique : environnement

# Schéma des outils

## Votre machine

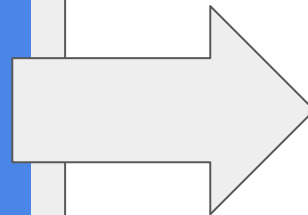
- un répertoire de travail
- un ide

### côté client

- un navigateur

### côté serveur

- un serveur web
- env php
- composer
- git
- symfony
- serveur maria db
- phpmyadmin



VM  
serveur distant



git  
documentation  
IA



# Liste des commandes

## Liste des commandes de départ

Symfony new grc –full  
Modification du fichier.env  
Symfony console doctrine:database:create  
Symfony console make:user  
Symfony console make:auth  
Symfony console make:registration-form  
Symfony console serve –d  
Test d'accès via un navigateur

## Liste des commandes

Symfony console make:controller  
Symfony console make:entity  
Symfony console make:migration  
Symfony console doctrine:migrations:migrate

# Le fichier .env

## Liste des commandes de départ

Le fichier .env se situe à la racine

Il permet d'appeler les modules : mail, api et notamment base de données.

- 1) vous devez inscrire la ligne :

```
DATABASE_URL="mysql://identifiant:mdp@127.0.0.1:3306/grc"
```

- 2) puis en ligne de commande, à la racine de votre projet vous devez taper :

```
Symfony console doctrine:database:create
```

**La technique : code php**

# Objet

- 1) En PHP, un objet est une instance d'une classe.
- 2) Les objets sont au cœur de la programmation orientée objet (POO) et permettent de regrouper des données et des comportements associés en un seul concept cohérent.
- 3) Cela facilite la gestion et l'organisation du code, le rend plus modulaire, réutilisable et maintenable.

# Objet

- 1) **Classe** : Une classe est un modèle ou un plan à partir duquel des objets sont créés. Elle définit les propriétés (attributs) et les méthodes (comportements) que les objets de cette classe auront.
- 2) **Objet** : Un objet est une instance concrète d'une classe. Lorsque vous créez un objet, vous allouez de la mémoire pour une instance de la classe et vous pouvez accéder aux propriétés et aux méthodes définies dans cette classe.
- 3) **Propriétés** : Les propriétés sont des variables qui appartiennent à une classe. Elles définissent les caractéristiques de la classe.
- 4) **Méthodes** : Les méthodes sont des fonctions qui appartiennent à une classe. Elles définissent les comportements que les objets de la classe peuvent réaliser.

# get et set

Les getters et les setters sont des méthodes utilisées en programmation orientée objet pour accéder et modifier les valeurs des propriétés privées ou protégées d'une classe.

L'utilisation de getters et setters permet de contrôler et encapsuler l'accès aux données, ajoutant ainsi des niveaux de sécurité et de validation.

# get et set

```
<?php
class Person
{
    private $name;
    private $age;

    // Getter pour la propriété $name
    public function getName()
    {
        return $this->name;
    }

    // Setter pour la propriété $name
    public function setName($name)
    {
        // Vous pouvez ajouter des validations ici
        if (is_string($name) && !empty($name)) {
            $this->name = $name;
        } else {
            throw new Exception("Invalid name");
        }
    }

    // Getter pour la propriété $age
    public function getAge()
    {
        return $this->age;
    }
}
```

```
// Setter pour la propriété $age
public function setAge($age)
{
    // Vous pouvez ajouter des validations ici
    if (is_int($age) && $age > 0) {
        $this->age = $age;
    } else {
        throw new Exception("Invalid age");
    }
}

// Utilisation de la classe Person
$person = new Person();
$person->setName("John Doe");
$person->setAge(30);

echo "Name: " . $person->getName() . "<br>";
echo "Age: " . $person->getAge();
?>
```

# Le fichier base.html.twig - Exemple

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>{% block title %}Welcome!{% endblock %}</title>
    {% block stylesheets %}{% endblock %}
  </head>
  <body>
    <nav><!-- Contenu de ma navbar --></nav>

    {% block body %}{% endblock %}

    <footer><!-- Contenu de mon footer --></footer>

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

```
{# Template général de la page #}
{% extends 'base.html.twig' %}

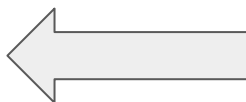
{# Titre de la page #}
{% block title %}Mon titre{% endblock %}

{# Ici vous mettez vos codes HTML et balises
Twig au besoin #}
{% block body %}
  <h1>Mon super titre H1</h1>
  <p>Un paragraphe contenant pleins de mots à
lire</p>
{% endblock %}
```



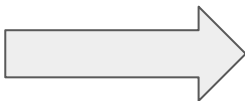
# Les fichiers twig

```
/**
 * @Route("/home", name="home")
 */
public function index()
{
    return $this->render('home/index.html.twig', [
        'firstname' => 'John',
    ]);
}
```



controller

Vue



```
{# Template générale de la page #}
{% extends 'base.html.twig' %}

{# Titre de la page #}
{% block title %}Mon titre{% endblock %}

{# Ici vous mettez vos codes HTML et balises Twig au besoin #}
{% block body %}
    <p>Bonjour {{ firstname }}</p>
{% endblock %}
```

# Le fichier base.html.twig

## apport du fichier base.html.twig

Le fichier de vue va hériter du fichier de base et remplir les parties du squelette grâce aux blocs.

Le fichier "*base.html.twig*" va récupérer le contenu du bloc `title` et le placer en lieu et place du sien et faire la même chose pour le bloc `body`. Cela vous donnera un fichier HTML complet et propre. Pour ce qui est des blocs `stylesheets` et `javascripts`, il ne remplira rien, car ces blocs n'existent pas dans le fichier "*index.html.twig*" et il fera de même avec `title` et `body` s'ils sont manquants.

## Exemple échange controller avec vue insertion en base de données

```

namespace App\Entity;
use App\Repository\VilleRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
#[ORM\Entity(repositoryClass: VilleRepository::class)]
class Ville
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $nom;

    #[ORM\OneToMany(mappedBy: 'ville', targetEntity: Diplome::class)]
    private $diplomes;

    public function __construct()
    {
        $this->diplomes = new ArrayCollection();
    }

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

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

    public function setNom(string $nom): self
    {
        $this->nom = $nom;

        return $this;
    }
}

```

<?php

```
namespace App\Controller;
```

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
```

```
use Symfony\Component\HttpFoundation\Response;
```

```
use Symfony\Component\Routing\Annotation\Route;
```

```
use Symfony\Component\HttpFoundation\Request;
```

```
use App\Entity\Ville;
```

```
use Doctrine\ORM\EntityManagerInterface;
```

```
class FormulaireController extends AbstractController
```

```
{  
    #[Route('/insertVille', name: 'app_insert_ville')]  
    public function index(): Response  
    {  
        return $this->render('formulaire/insertVille.html.twig', [  
            'controller_name' => 'nouvelle ville',  
        ]);  
    }  
    #[Route('/ville', name: 'app_ville')]  
    public function ville(Request $request, EntityManagerInterface $manager): Response  
    {  
        $ville = new Ville();  
        $ville->setNom($request->request->get("ville"));  
        $manager->persist($ville);  
        $manager->flush();  
        return $this->render('formulaire/insertVille.html.twig', [  
            'controller_name' => 'nouvelle ville',  
        ]);  
    }  
}
```

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

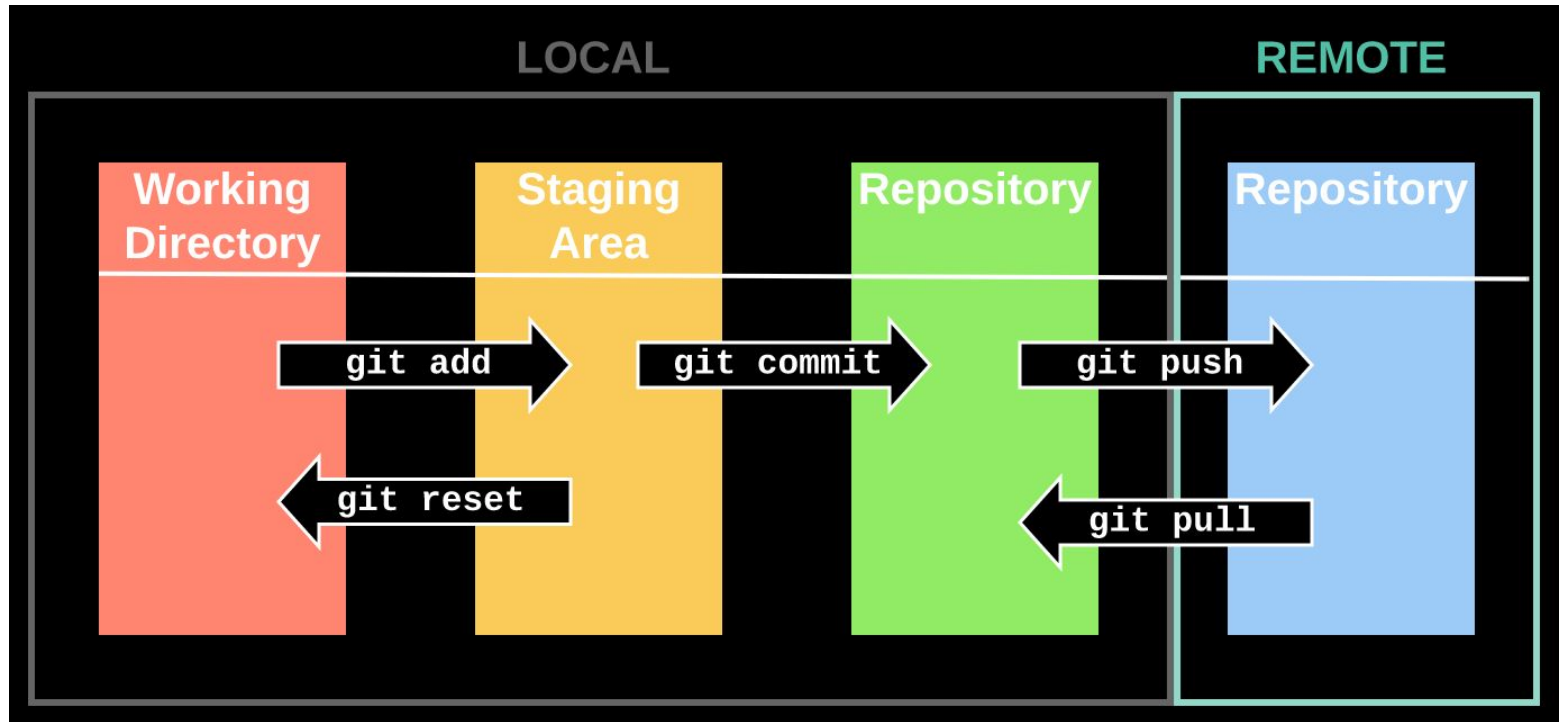
{% block title %}Hello FormulaireController!{% endblock %}

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

<div class="example-wrapper">
    <h1>{{ controller_name }}! 
```

# Mémo Git

# Mémo GIT





# Mémo GIT

une fois votre repository créé et lié. Pour envoyer votre travail, vous devez effectuer **à la racine de votre répertoire** les commandes suivantes dans l'ordre :

- 1) `git add .`
- 2) `git commit -m "message"`
- 3) `git push`

une demande d'authentification vous sera demandée.

# PHP

- 1) fichier .php
- 2) bloc d'instruction encadré par {...}
- 3) une instruction termine par ;
- 4) attention à l'indentation
- 5) besoin de commentaires

# Gestion des erreurs Symfony

# No route found for "GET /zeroiueuirhgiuoerh"



Exceptions 2

Logs 1

Stack Traces 2

Symfony\Component\HttpFoundation\Exception\

## NotFoundHttpException

in D:\Documents\demos\symfony-upload-multiple\vendor\symfony\http-kernel\EventListener\RouterListener.php (line 136)

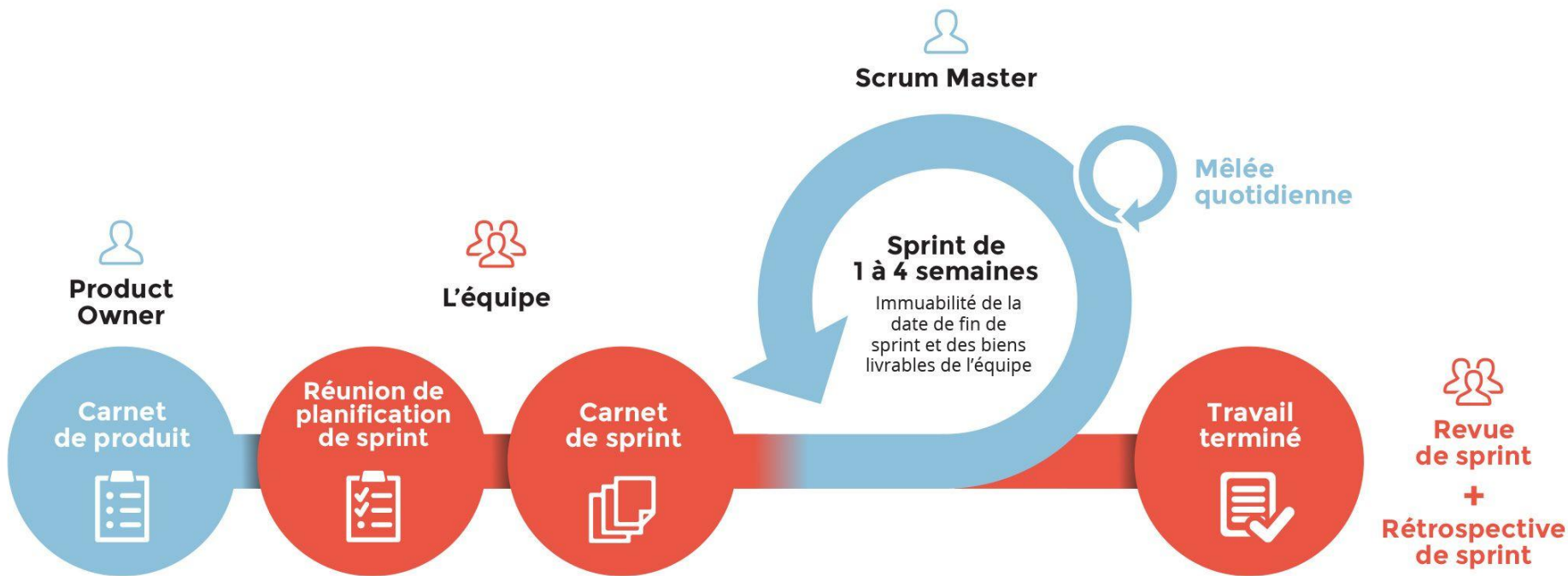
```

131.
132.         if ($referer = $request->headers->get('referer')) {
133.             $message .= sprintf(' (from "%s")', $referer);
134.         }
135.
136.         throw new NotFoundHttpException($message, $e);
137.     } catch (MethodNotAllowedException $e) {
138.         $message = sprintf('No route found for "%s %s": Method Not Allowed (Allow: %s)', $request->getMethod(), $request->getPat
139.
140.         throw new MethodNotAllowedHttpException($e->getAllowedMethods(), $message, $e);
141.     }
    
```

+ RouterListener->onKernelRequest(object(RequestEvent), 'kernel.request', object(TraceableEventDispatcher))  
in D:\Documents\demos\symfony-upload-multiple\vendor\symfony\event-dispatcher\Debug\WrappedListener.php (line 117)

+ WrappedListener->invoke(object(RequestEvent), 'kernel.request', object(TraceableEventDispatcher))

# Exemple de management de projet



# Préparatifs

- 1) Environnement technique avec php >8
- 2) bootswatch (choix du thème) ou bootstrap classique
- 3) BD avec user et password
- 4) IDE paramétré (twig, php, ...)
- 5) config > package > twig.yaml

```
form_themes: ['bootstrap_5_layout.html.twig']
```

# Notre projet



# Résumé

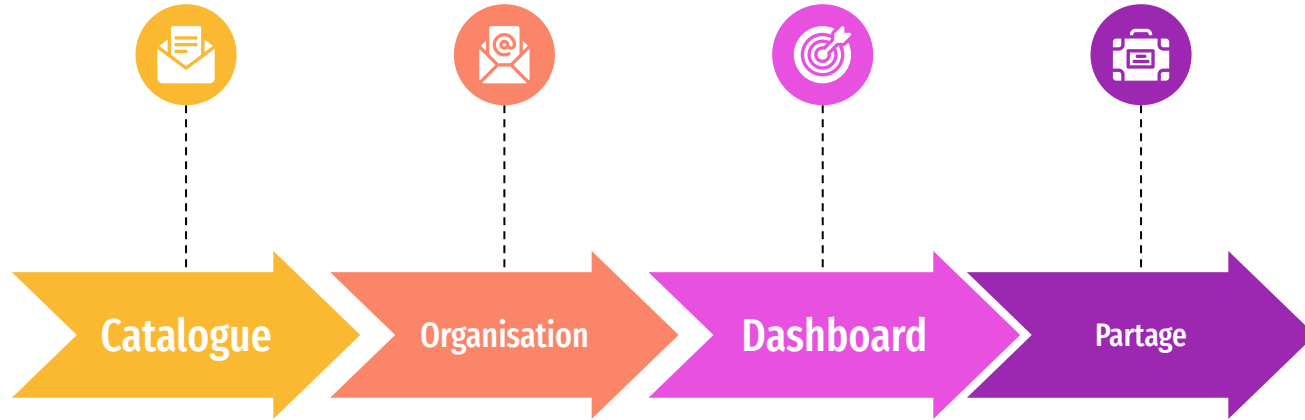
L'objectif est de créer une **application de gestion de notes personnelles ou professionnelles**, avec un système de **tags**, d'**états**, d'**utilisateurs**, et un suivi des **modifications**.

# Résultat attendu

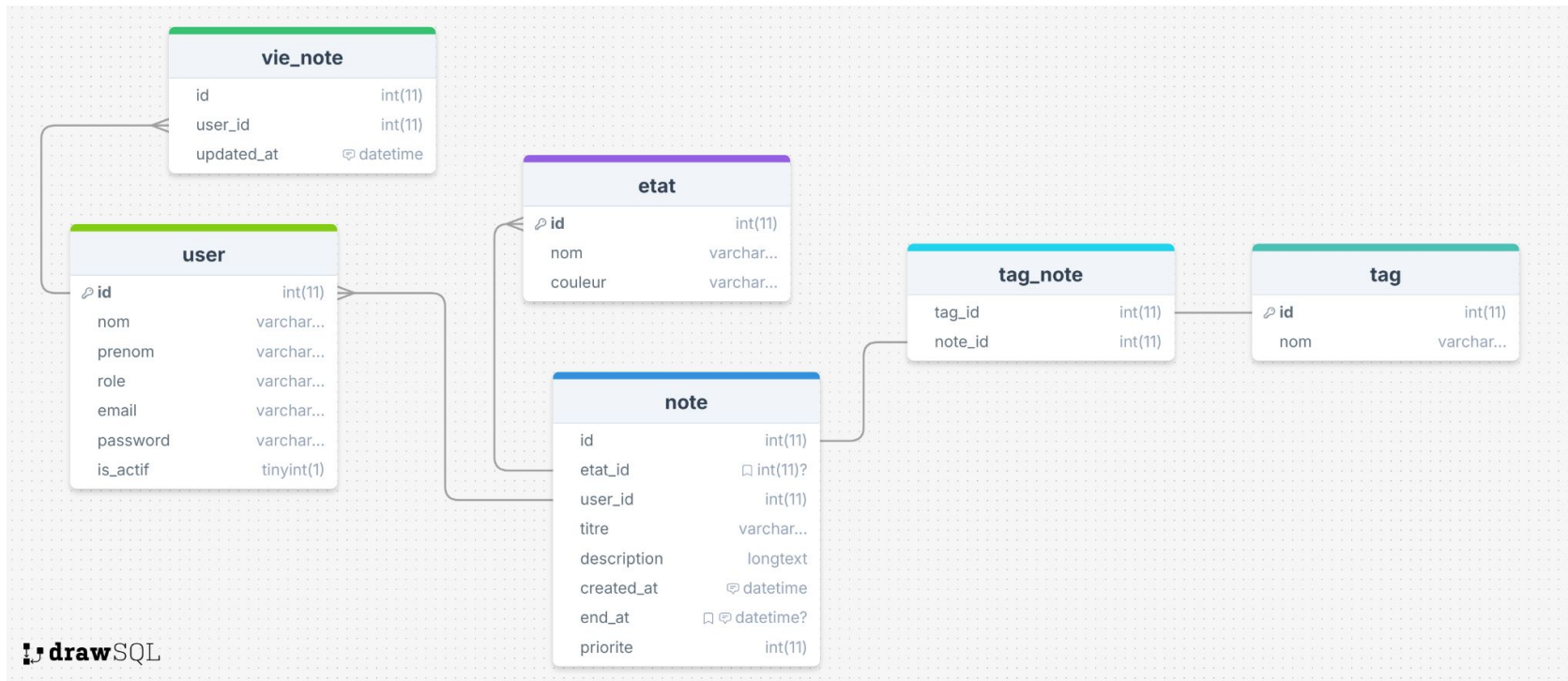
À la fin du projet, chaque étudiant ou binôme aura développé une **application web fonctionnelle** permettant de :

- ✓ Gérer ses notes
- ✓ Les classer par tags, états, priorités -> dashboard dynamique
- ✓ Voir l'historique des modifications
- ✓ Naviguer facilement dans une interface claire et responsive

# User story



# Modèle du projet



## Modèle, notion d'entity

Une entité est une **classe PHP** mappée à une **table de base de données** grâce à Doctrine (ORM).

# Entity : USER

Représente un utilisateur du système.

Attributs :

- `nom, prenom, email, password, role, isActive`.

Relations :

- **OneToMany** avec `Note` : un utilisateur peut créer plusieurs notes.
- **OneToMany** avec `VieNote` : un utilisateur peut avoir plusieurs enregistrements d'historique de note.

# Entity : NOTE

## Note

- Représente une **note ou tâche** créée par un utilisateur.
- Attributs :
  - `titre, description, createdAt, endDate, priority`.
- Relations :
  - **ManyToOne** avec `User` : chaque note appartient à un utilisateur.
  - **ManyToOne** avec `Etat` : chaque note a un état (ex : "En cours", "Terminée").
  - **ManyToMany** avec `Tag` : une note peut avoir plusieurs tags (et inversement).

# Entity : TAG

## Tag

- Représente une **étiquette** ou **catégorie** qu'on peut associer à une ou plusieurs notes.
- Attributs :
  - **nom** (nom du tag).
- Relations :
  - **ManyToMany** avec **Note**.



# Entity : ETAT

Représente un **état de progression** d'une note (ex: "À faire", "En cours", "Fait").

Attributs :

- **nom, couleur** (pour UI/UX).

Relations :

- **OneToMany** avec **Note** : un état peut être assigné à plusieurs notes.

# Entity : VieNote

Sert à **tracer l'historique de modifications** des notes d'un utilisateur (ou une autre forme de versionnage).

Attributs :

- `updatedAt` : la date de mise à jour.

Relations :

- **ManyToOne** avec `User`.

# Création d'une entity

- Ouvrir le terminal dans le dossier du projet Symfony (à la racine)
- Taper la commande `php bin/console make:entity`
  - a. symfony va poser une série de question
- Puis générer une migration avec les deux commandes suivantes :
  - a. `php bin/console make:migration`
  - b. `php bin/console doctrine:migrations:migrate`

# Les différents types

| Type Symfony                    | Type SQL généré | Utilisation   |
|---------------------------------|-----------------|---|
| <code>string</code>             | VARCHAR(255)    | Texte court (nom, titre...)                         |
| <code>text</code>               | TEXT            | Texte long (description)                            |
| <code>integer</code>            | INT             | Nombre entier                                       |
| <code>boolean</code>            | BOOLEAN         | Vrai/faux   |
| <code>datetime_immutable</code> | DATETIME        | Date/heure (ne change pas)                          |
| <code>float</code>              | DOUBLE / FLOAT  | Nombre à virgule                                    |
| <code>relation</code>           | clé étrangère   | Lier à une autre entité ( <code>User</code> , etc.) |

# La notion de CRUD

La notion de **CRUD** est fondamentale en développement web.

**CRUD** est un acronyme qui signifie :

| Lettre | Action | Traduction      |
|--------|--------|-----------------|
| C      | Create | Créer           |
| R      | Read   | Lire / Afficher |
| U      | Update | Mettre à jour   |
| D      | Delete | Supprimer       |

Ces 4 opérations correspondent aux **actions de base** qu'on effectue sur des données dans une base de données.

# La notion de CRUD

Symfony propose une commande pour générer tout cela automatiquement

```
php console make:crud Note
```

Cela va créer :

- un **contrôleur** `NoteController.php` avec les méthodes :
  - `index()` → affiche la liste des notes (Read)
  - `new()` → formulaire d'ajout (Create)
  - `edit()` → formulaire de modification (Update)
  - `delete()` → suppression d'une note (Delete)
  - `show()` → vue des détails (Read)
- un **formulaire** : `NoteType.php`
- des **vues Twig** pour chacune de ces actions

**Merci de votre attention**