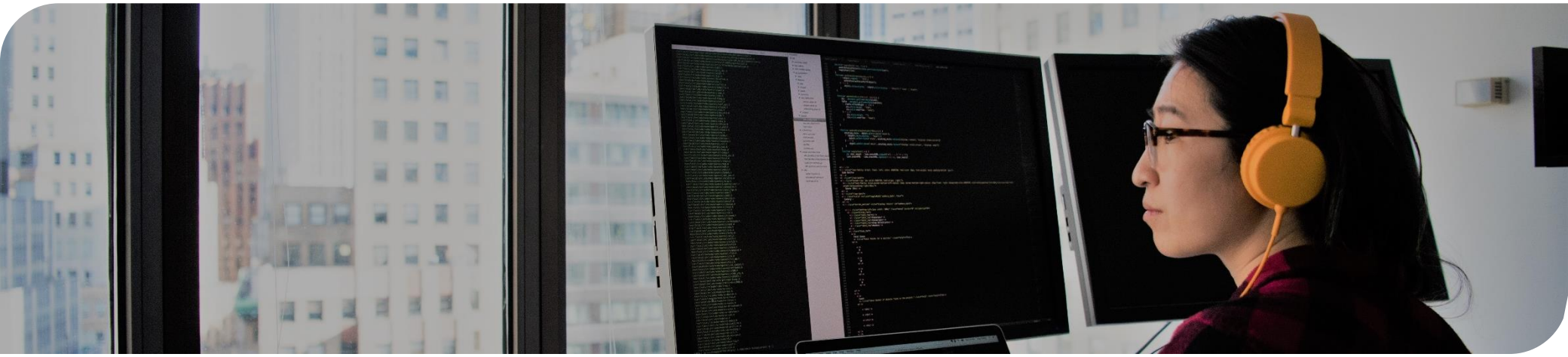
JUNIORISEP



Formation Symfony 5.0

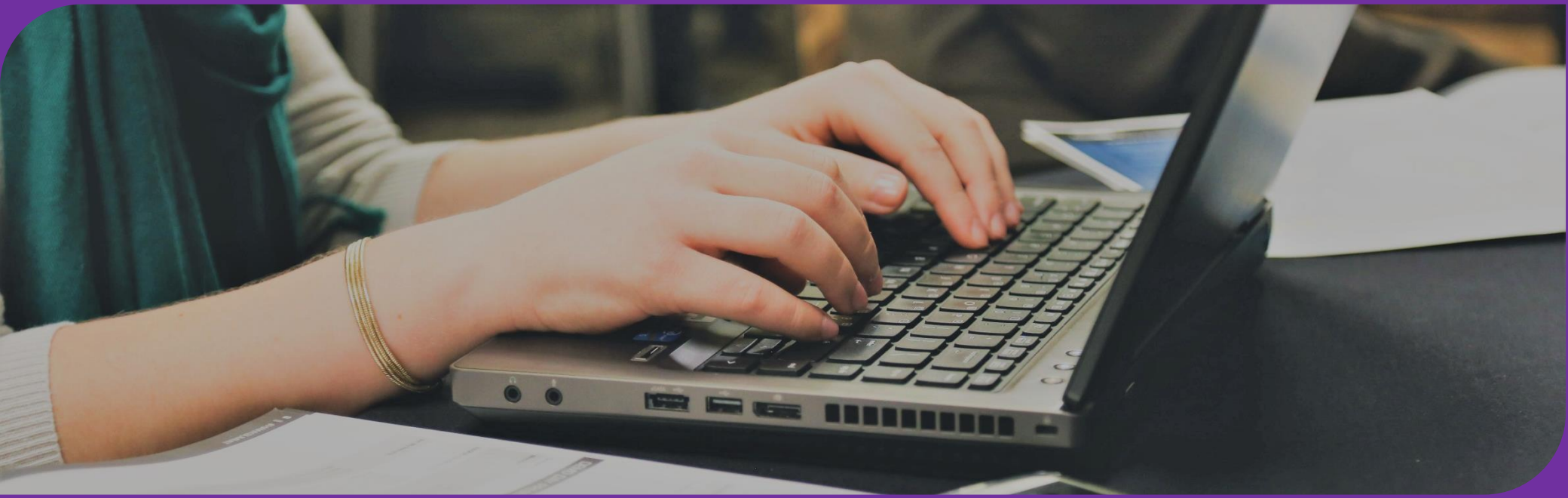
Cours 1

Lundi 25 novembre | Pierre MARQUET & Florent FAVOLE & Laurianne PROGENT

Symfony

Prérequis

- Niveau novice en **PHP** et connaissances en **HTML/CSS**
- Avoir un **IDE** (IntelliJ, PhpStorm, etc.)
- Savoir utiliser un **serveur local** du type WAMP / XAMP / LAMP
  - **Apache**
  - **PHP 7.2.9** minimum (avec les extensions Ctype, iconv, JSON, PCRE, Session, SimpleXML et Tokenizer qui sont normalement intallées automatiquement avec PHP)
  - **MySQL**
- Avoir **Git BASH**



1 Framework Symfony

Les frameworks

Ensemble d'outils et de composants logiciels à la base d'un logiciel ou d'une application. C'est le framework qui établit les fondations d'une application ou son squelette applicatif.

Cela permet notamment de :

- Améliorer la **productivité** des développeurs
- Permettre le travail en équipe grâce à une **structure bien organisée**
- Garantir la **maintenance**, la **sécurité** et l'**évolutivité**

Symfony

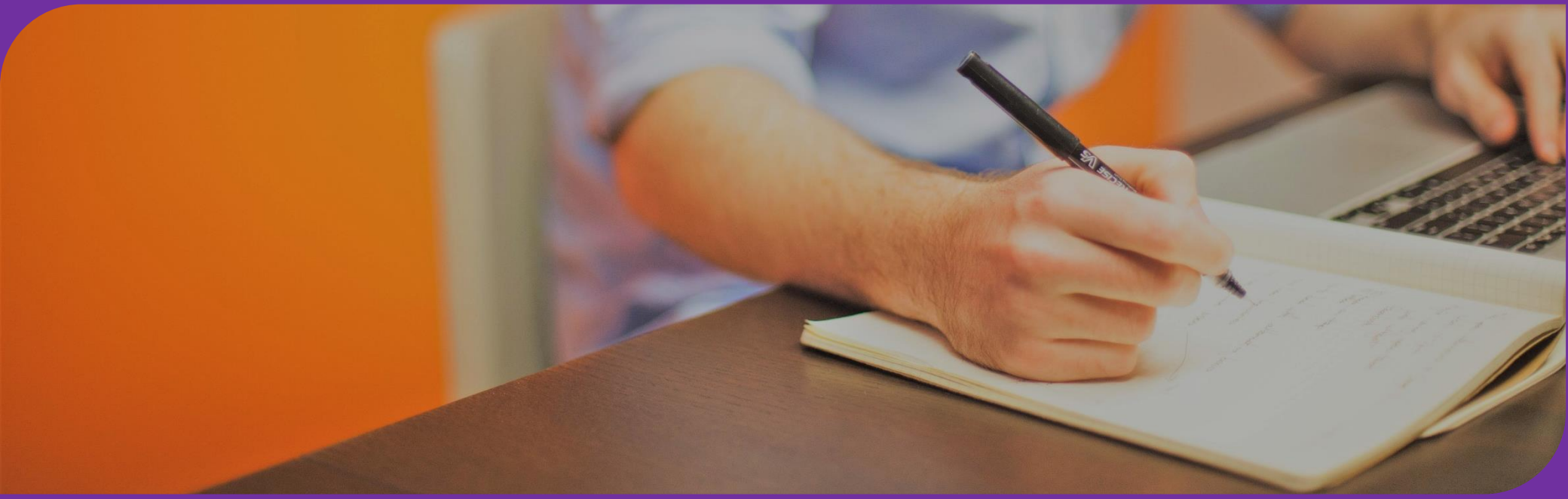
Framework MVC PHP **Open-Source** (gratuit et libre)

Développer des sites web de **qualité professionnelle**

Grande **communauté** de développeurs et beaucoup de **documentation**

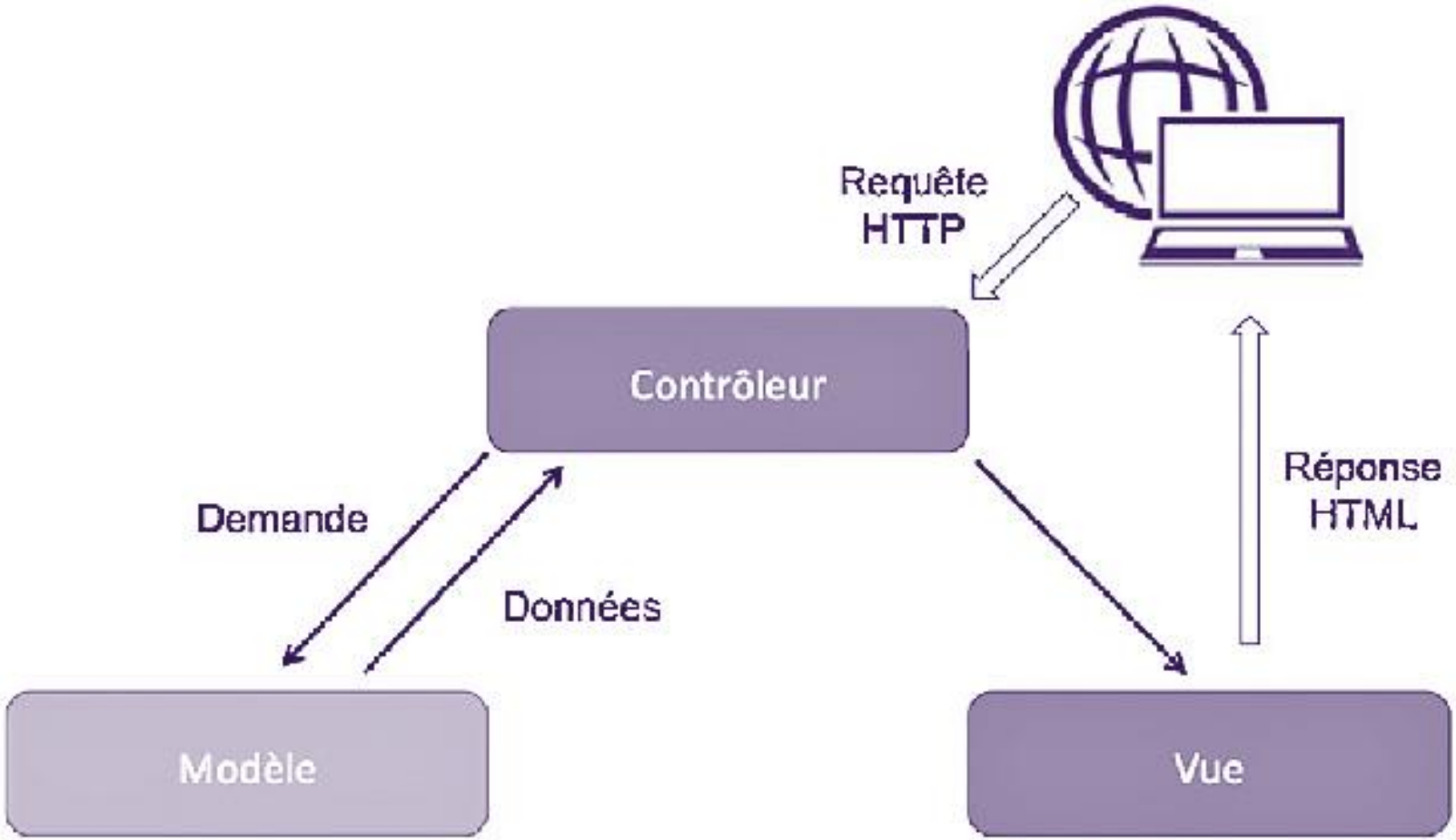
Système de **bundles** indépendants et flexibles

« Ne pas réinventer la roue »



2 Quelques rappels

Le modèle MVC

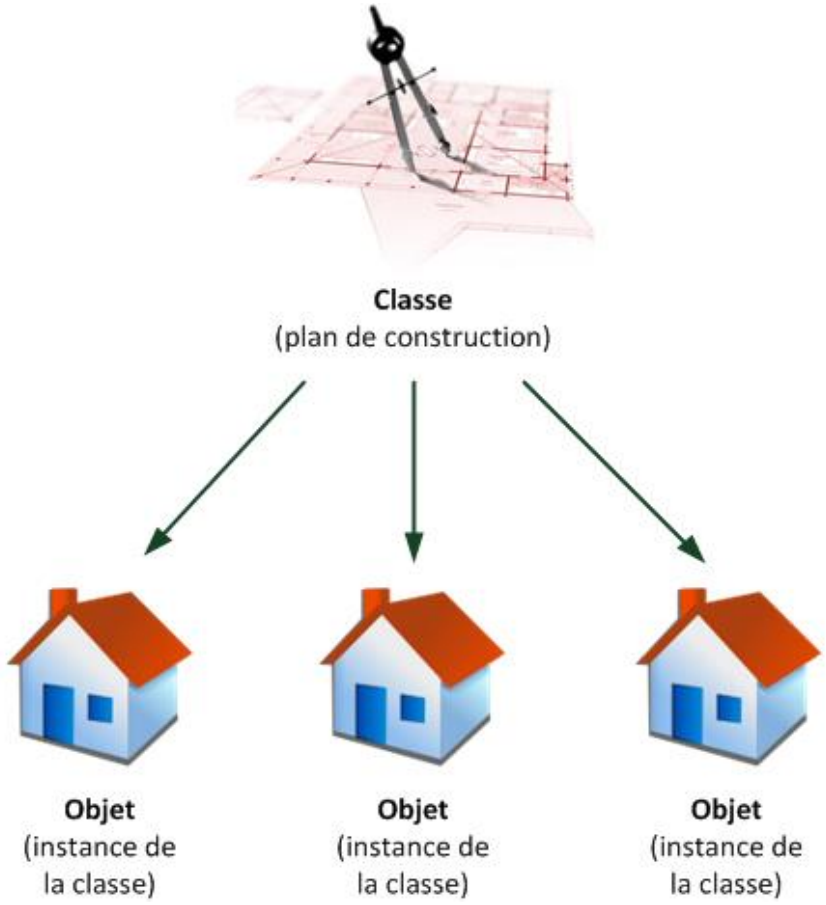


La POO

Modèle de langage de programmation

→ S'articule autour d'**objets** et de **données**, plutôt que d'actions et de logique

On décrit les systèmes en **classes d'objets** plutôt qu'en terme de fonctions



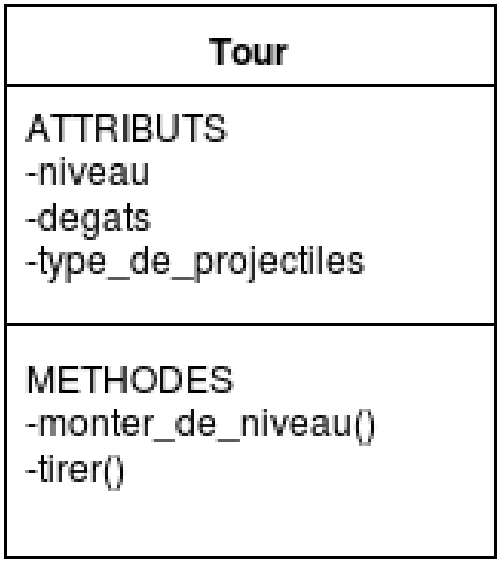
La POO

Un exemple ?

→ Vous voulez créer un Tower Defense ?

Les **tours**, les **ennemis**, ainsi que les **projectiles** peuvent être des **objets**.

La tour par exemple peut être conçue de la manière ci-contre :

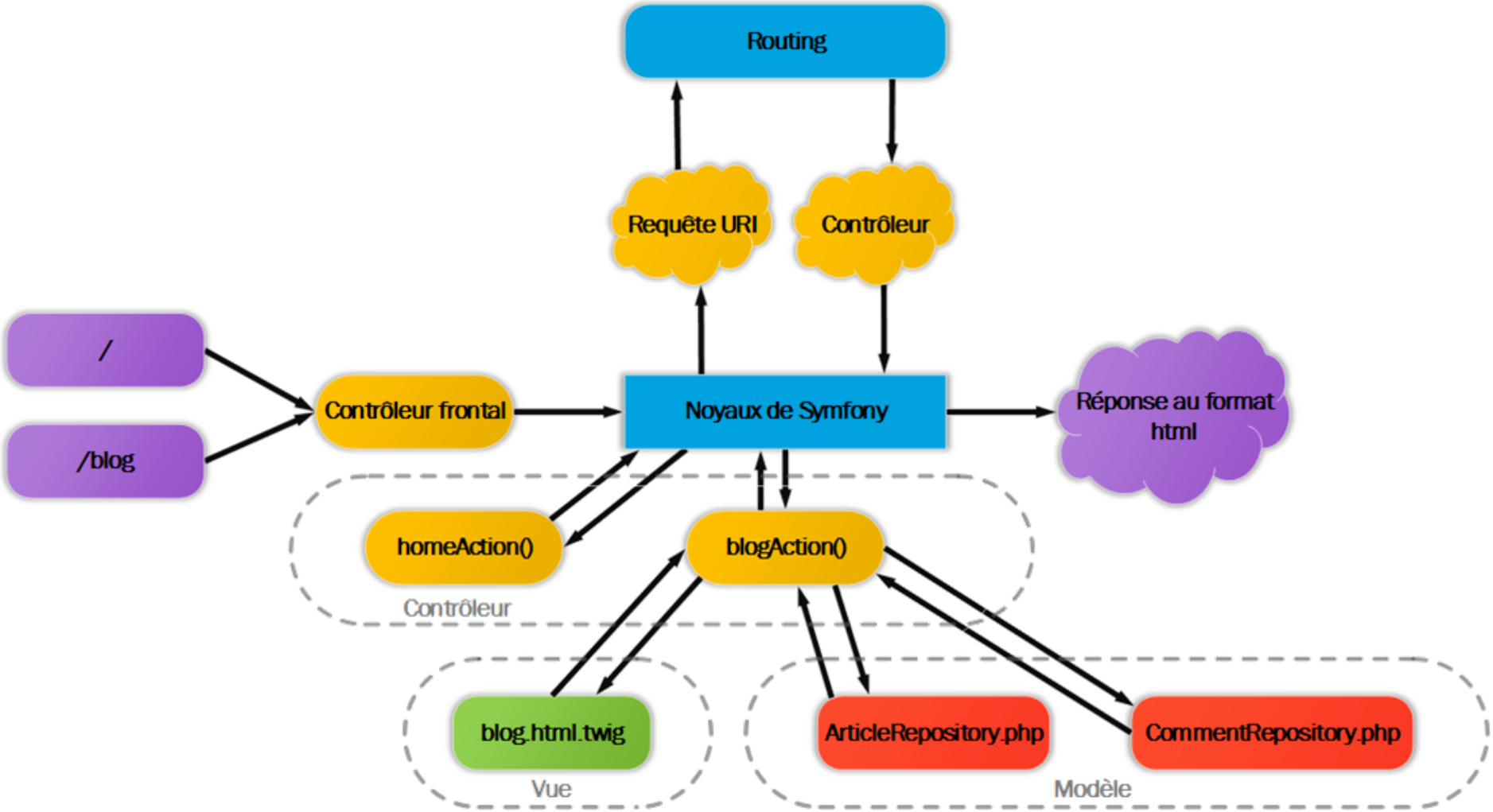


3 Les bases

Les fondamentaux de Symfony

Entité	C'est juste un **objet** ! Avec ses attributs, ses getters et ses setters.
Contrôleur	Contient toute la **logique** du site. Il utilise des services et les modèles et appelle les vues.
Routeur	Détermine à partir d'un **URL** quel **contrôleur** appeler et avec quels arguments. Cela permet de configurer son application pour avoir de très belles URL.
Service	Objet PHP qui remplit une **fonction** (envoyer des e-mails, gérer une base de données, etc.). Un service a pour vocation d'être accessible depuis n'importe où dans votre code.

Les fondamentaux de Symfony



Composer

Téléchargement : https://getcomposer.org/download/ (utiliser les 4 lignes de commandes dans GIT Bash)

Gestionnaire de dépendances utilisé en ligne de commande

→ Si vous souhaitez installer un **package**, composer se chargera d'installer les autres paquets dont il dépend

→ Création du fichier **composer.json** (fichier contenant la liste des bundles du projet, très utile car il suffit juste d'avoir le composer.json pour importer tout les bundles d'un projet)

Symfony

Téléchargement : https://symfony.com/download

Permet la **mise en place** et la **gestion** de son projet Symfony plus simplement via un **invite de commandes**

→ Cet outil est une **nouveauté** datant de la version **4.3** de Symfony, ne vous étonnez pas si certains tutoriels plus datés n'utilisent que composer

Cheat sheet : Composer

Mise à jour de toutes les dépendances :

$ composer update

Ajout de un ou plusieurs bundles :

$ composer require nom_bundle

$ composer req bundle1 bundle2 bundle3

Supprimer bundle :

$ composer remove nom_bundle

Installer les dépendances :

$ composer install

⇒ Lit le fichier composer.json, résout les dépendances et les installe

Les bundles

On peut voir un bundle comme un **plug-in** dans Symfony.

→ Il n'est pas nécessaire de les configurer, depuis la version 4.0 de Symfony c'est automatique !

Il est possible de créer ses propres bundles, nous ne couvrirons pas ce sujet dans cette formation.

Cheat sheet : Quelques bundles utiles

Nom	Utilité
maker-bundle	Créer facilement des contrôleurs, entités, formulaires, etc.
annotations	Permet de mettre les routes directement dans le contrôleur
profiler	Outil pour le débuggage
twig	Moteur de template
orm	Base de données
form	Formulaires

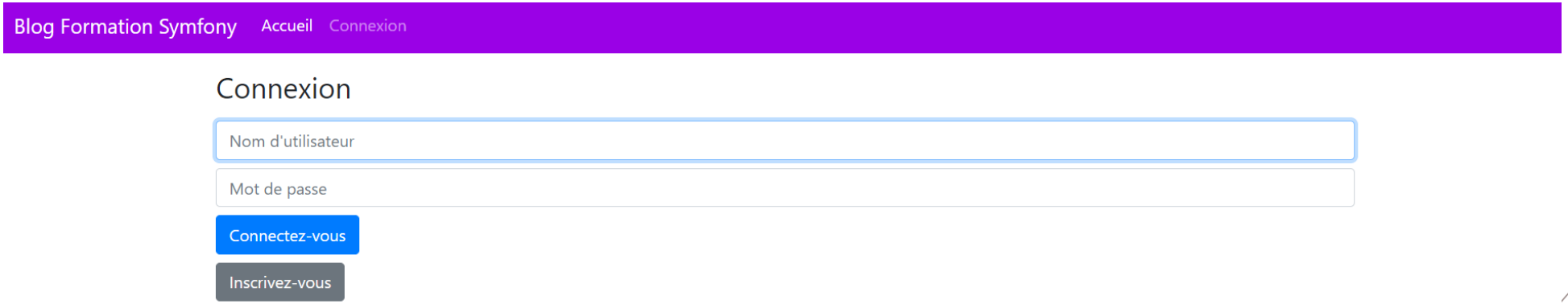
Notre projet de TP

Vue de la page d'accueil :

On rappelle que nous nous concentrons sur le back et non sur le front !

Notre projet de TP

Vue des modules de connexion et d'inscription :



Notre projet de TP

Vue du back office:

Blog Formation Symfony Accueil Administration des articles Déconnexion

Article index

Titre	Auteur	Date de création	Actions
quas voluptates numquam	Augustin Blanchet	2019-05-13 10:07:00	Voir Éditer Supprimer
quasi dolor ut	André Joly	2019-05-13 10:07:00	Voir Éditer Supprimer
harum et officia	Benjamin Royer	2019-05-13 10:07:00	Voir Éditer Supprimer
tenetur ut perferendis	Christophe-Nicolas Gilles	2019-05-13 10:07:00	Voir Éditer Supprimer
adipisci dolore iure	Sébastien Caron	2019-05-13 10:07:00	Voir Éditer Supprimer
nesciunt explicabo et	Thibault Valentin	2019-05-13 10:07:00	Voir Éditer Supprimer
praesentium labore voluptas	Dominique-Guillaume Besnard	2019-05-13 10:07:00	Voir Éditer Supprimer
alias eum voluptatum	Laurent de la Bruneau	2019-05-13 10:07:00	Voir Éditer Supprimer

Notre projet de TP

Vue d'un article :

Blog Formation Symfony Accueil Administration des articles Déconnexion

quasi dolor ut

$$X_e(f) = \frac{1}{T_e} \sum_{k=-\infty}^{+\infty} X_c\left(f - \frac{k}{T_e}\right) = F_e \sum_{k=-\infty}^{+\infty} X_c(f - kF_e)$$

Auteur : André Joly Date de publication : 2019-05-13

Lorem ipsum dolor sit amet consectetur adipisicing elit, sed do eiusmod tempor lectus eget felis incididunt ut labore et **dolore magna aliqua**: Duis aute irure dolor in sunt in culpa reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Pellentesque tincidunt excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia occaecat cupidatat deserunt mollit anim id est laborum.

* Ut enim ad minim veniam
* Quis nostrud exercitation *ullamco laboris*
* Nisi ut aliquip ex ea commodo consequat

Praesent id fermentum lorem. Ut est lorem, fringilla at accumsan nec, euismod at mi ut semper pulvinar nunc. Aenean mattis sollicitudin mattis. Nullam pulvinar vestibulum bibendum. Pellentesque tortor magna,

4 Commencement du projet

Débuter le projet

Ouvrir Git Bash dans le dossier désiré (clic droit dans l'explorateur → « Git Bash here »)

Commencer par vérifier que le setup est en ordre :

```
$ symfony check:requirements
```

Créer le projet :

```
$ symfony new NomDuProjet          (microservices & API)
$ symfony new NomDuProjet --full   (projet web complet)
```

Passer dans le dossier du projet :

```
$ cd NomDuSite
```

Le serveur

Démarrer le serveur de développement de Symfony :

$ symfony server:start

Si tout va bien, aller dans le navigateur et taper l'url :

localhost:8000

Notre premier Hello World!

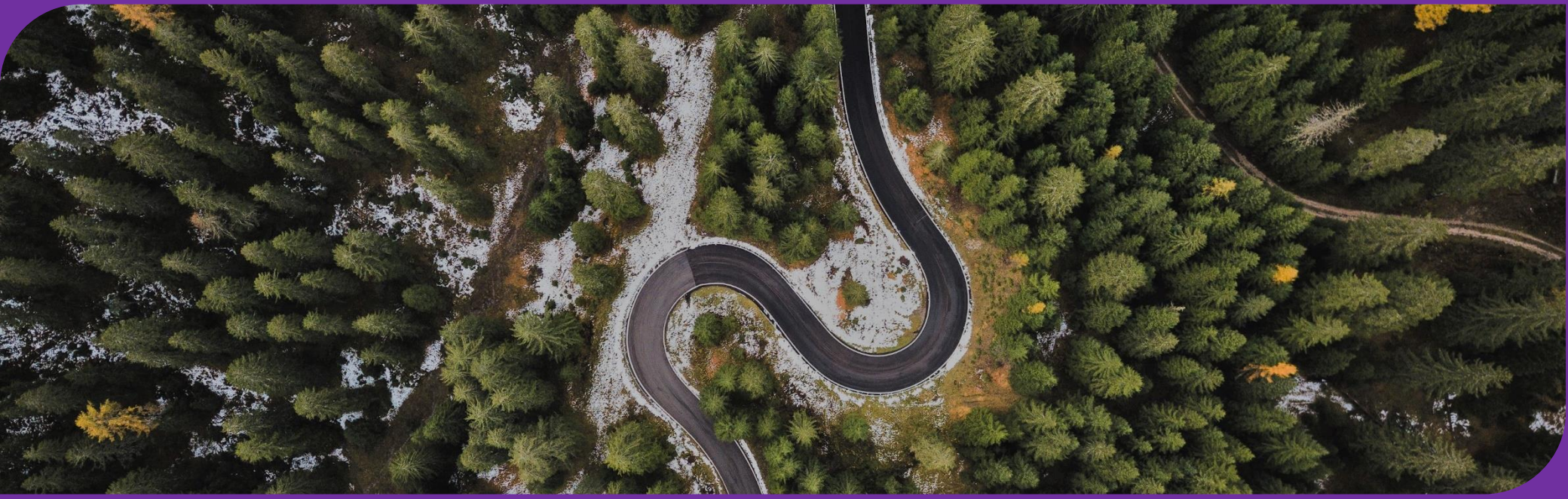
Créer un nouveau contrôleur :

$ php bin/console make:controller Nom

→ le nom final du fichier sera automatiquement NomController.php

Après avoir retiré la méthode d'exemple, créer une méthode **index()** dans ce contrôleur qui retourne un objet **Response** (Symfony\Component\HttpFoundation\Response) avec en paramètre « Hello World » :

```php
/**
 * @Route("/hello")
 */
public function index()
{
    return new Response( content: "Hello World!");
}
```



5 Configuration des routes et des services

Les Routes

Route : associe une URL à une action du contrôleur

Routeur : détermine quelle route utiliser pour la requête

Dans notre exemple du Hello World :

URL	Contrôleur	Action
/hello	NomController.php	index()

➔ Dans notre formation, les routes sont définies dans les **annotations** du contrôleur, ce qui nous permet de nous passer de routeur (il existe d'autres méthodes pour définir les routes).

Les Routes

Il est possible de faire passer des **paramètres** dans une route

Par exemple :

```
/**
 * @Route({"fr": "/bonjour", "en": "/hello"}, name="nom.route.hello")
 */
```

Pour voir toutes les routes :

$ php bin/console debug:router

Les Services

Service : classe PHP globale qui remplit une **fonction** bien spécifique (envoyer des mails, effectuer une tâche, etc.)

→ Permet de **séparer** la logique métier dans l'application (on allège le contrôleur de ce qu'il n'est pas censé effectuer, il fera seulement appel aux services)

Tout est service ! (Twig, templating, kernel, profiler, session, ...)

→ **Réutilisabilité** partout dans le code !

Pour voir les services par défaut :

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

Pour voir tous les services :

```
$ php bin/console debug:autowiring
```

6 Un super service : TWIG

Introduction à TWIG

C'est un **moteur de templates** !

Permet de **séparer** le code **PHP** (au niveau du contrôleur) du code **HTML** (au niveau des vues)

→ **Simplifie** l'affichage des données et le rend plus lisible

→ Système d'**héritage** très utile pour l'organisation des vues

Comment ça fonctionne ?

Les templates TWIG utilisent l'**héritage**

Un **fichier père** appelé « **base.html.twig** » sera à la racine du dossier templates → les éléments dans ce fichier seront communs à tous les autres templates TWIG en ajoutant la ligne

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

→ Cela fait gagner un temps fou de bien savoir exploiter ces « **couches** » proposées par TWIG (il est possible de faire d'autres bases de templates en fonction des parties du site par exemple)

Comment ça fonctionne ?

Dans **base.html.twig** :

```
<!DOCTYPE html>

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

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

</html>
```

Le contenu de ces bornes sera remplacé par les **titres** spécifiés dans chaque page

Entre ces bornes on met les liens pour des **stylesheets**

Entre ces bornes c'est le contenu du **body** (dans la base on peut ajouter quelque chose avant ce block pour que ça apparaisse sur TOUTES les pages de notre site)

Entre ces bornes on met les liens pour le **javascript**

Comment ça fonctionne ?

Dans une page **index.html.twig** :

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

On **hérite** du contenu de base.html.twig

```
{% block title %}Titre de la page{% endblock %}
```

On donne un **titre** à la page

```
{% block body %}
    <h1>Voici la vue</h1>
    <p>Ce contenu ne concerne que la page index.html.twig, qui hérite de base.html.twig</p>
{% endblock %}
```

Contenu du **body** pour cette page

Cheat sheet : Syntaxe TWIG usuelle

Afficher une variable :

{{ maVariable }}

Afficher l'attribut d'un objet :

{{ user.email }}

Déclarer un bloc :

{% block block_name %}
...
{% endblock %}

Conditions :

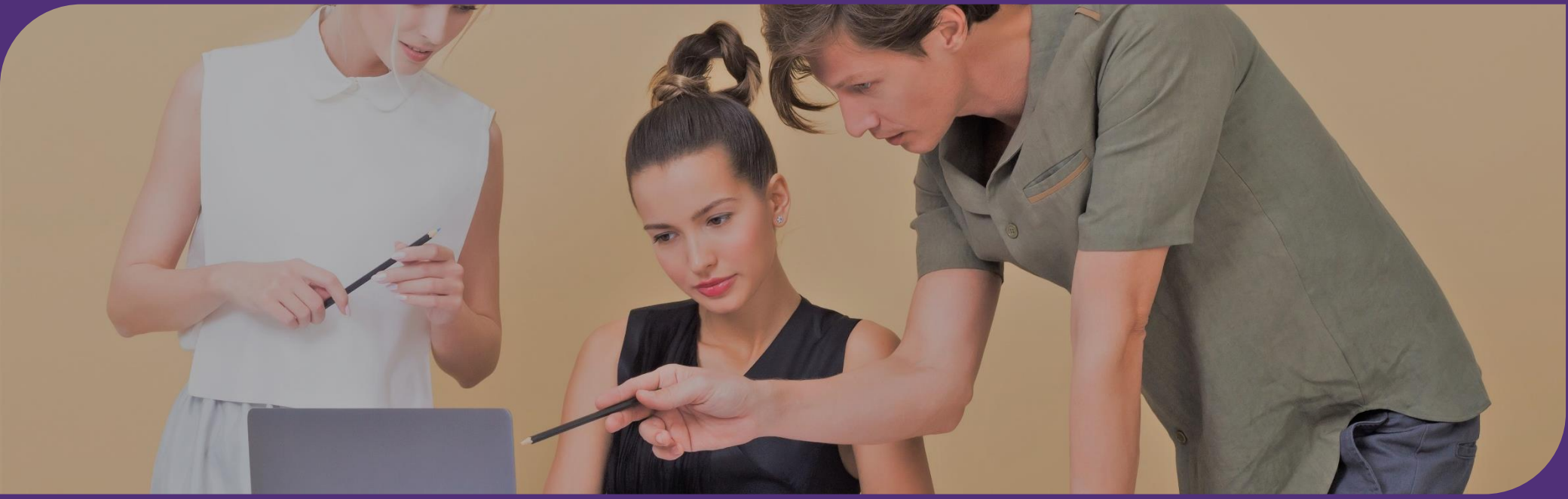
{% if ... %} ...
{% elseif %} ...
{% else %} ...
{% endif %}

Boucle :

{% for user in listUsers %}
...
{% endfor %}

Appeler une route (par exemple dans une barre de navigation) :

```
<a href="{{ path("show.movie", {'paramètre': 'valeur'}) }}">Voir les films</a>
```



7

TP

TP

But du TP : Réalisation d'une barre de navigation

Cette barre de navigation aura pour but de **lier au minimum 2 pages entre elles**.

On utilisera pour cela au minimum:

- 2 contrôleurs
- 2 vues

TP

Exemple de rendu final sans l'utilisation de bootstrap :

Code du HomeController :

```php
<?php

namespace App\Controller;

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

class HomeController extends AbstractController
{
    /**
     * @Route("/", name="home")
     * @return Response
     */
    public function  index() : Response
    {
        return $this->render( view: 'pages/home.html.twig');
    }
}
```

TP

Exemple de rendu final sans l'utilisation de bootstrap :

Deuxième page

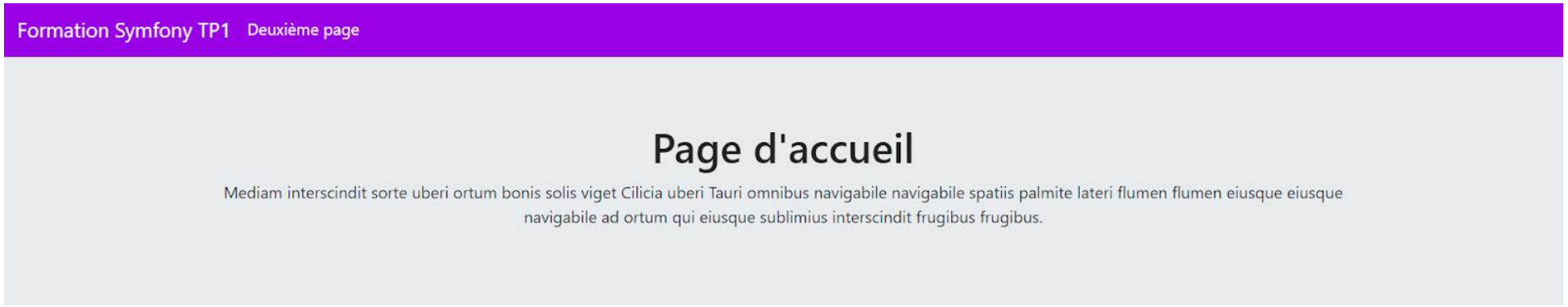
Code du SecondController :

```php
<?php

namespace App\Controller;

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

class SecondController extends AbstractController {

    /**
     * @Route("/ExemplePage2", name="page2")
     * @return Response
     */
    public function index() : Response
    {
        return $this->render( view: 'pages/second.html.twig');
    }
}
```

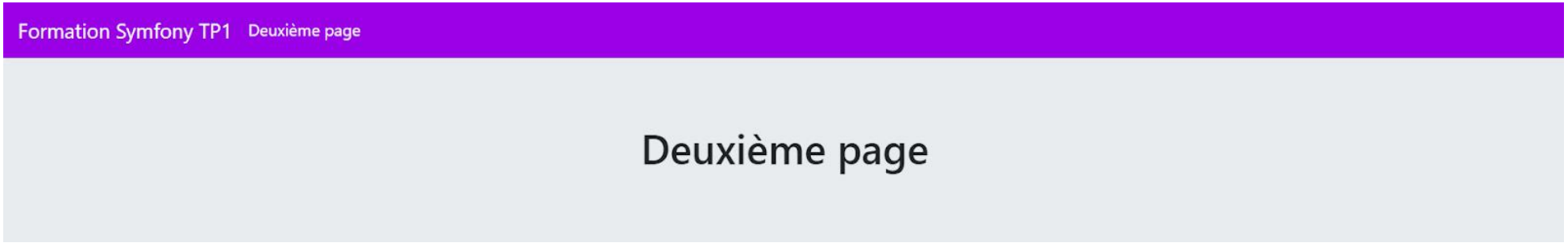
TP

Exemple de rendu final avec l'utilisation de bootstrap :

La page d'accueil :



Exemple de deuxième page :



Merci d’avoir participé à cette formation Junior ISEP !

N’hésitez pas à nous contacter :

- lprogent@juniorisep.com
- ffavole@juniorisep.com
- pmarquet@juniorisep.com

8 Bonus

Bootstrap

Framework CSS (avec également des composants HTML et JavaScript)

→ Permet d'incorporer un minimum de **visuel** très **rapidement**

→ Ici nous allons utiliser un petit peu de Bootstrap pour rendre notre blog un peu plus agréable pour les yeux

Ajouter Bootstrap au projet

Aller sur le site :

https://getbootstrap.com/docs/4.3/getting-started/introduction/

→ Copier-coller le **<link>** pour le CSS dans le bloc correspondant de **base.html.twig**

```
{% block stylesheets %}{% endblock %}
```

→ Pareil pour les **<script>** pour le JavaScript

```
{% block javascripts %}{% endblock %}
```

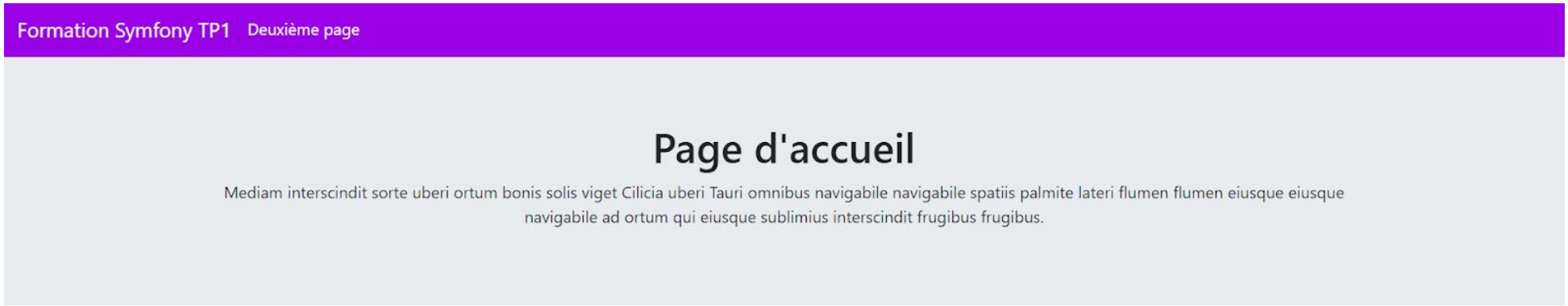
Ajouter une barre de navigation stylée

Copier-coller le code Bootstrap et l'adapter au TP :

https://getbootstrap.com/docs/4.3/components/navbar/

→ Ajouter dans **base.html.twig** avant le bloc body

Résultat attendu :



Ajouter un peu de style

Pour la **page 1**, ajouter une **div** autour du contenu de la page avec la classe **« jumbotron text-center »**

Pour la **page 2**, ajouter une **div** autour du contenu de la page avec la classe **« container mt-4 »** (mt signifie margin-top)

Pour personnaliser encore plus le visuel du projet :

https://getbootstrap.com/docs/4.3/utilities/borders/ (liste des classes pré-faites de Bootstrap pour faire un petit peu de front sans se fouler)