

# SYMPHONY : TRAVAUX PRATIQUES

INSTALLATION, CONTROLES, ROUTES ET VUES

Makrem Mhedhbi

# Création du premier projet

- Il est souvent conseillé de taper la commande **symfony check:requirements** pour vérifier que votre environnement est prêt pour créer un nouveau projet.
- Une fois cette étape est valide, nous pouvons commencer. Placer vous sous un répertoire, par exemple « Applications » et taper sur votre invite de commande, ce qui suit :
  - **composer create-project symfony/website-skeleton premier\_mvc 4.4.\***
- Une fois c'est bon, taper la commande « **cd premier\_mvc** » et lancer ensuite la commande « **symfony server:start** »
- Lancer votre navigateur et taper l'adresse **http://127.0.0.1:8000**

# Hiérarchie d'un projet symfony

- bin: c'est ici que se trouve les exécutables de Symfony, comme le fichier console. Les fichiers dans ce dossier seront donc exécutables en ligne de commande.
- config: comme son nom l'indique, ce dossier va contenir la configuration de notre application Symfony. C'est ici que nous allons configurer les routes, les services, ...
- public: Ceci est le dossier d'entrée de notre application, mais aussi le dossier public, nous allons y mettre tout les fichiers accessibles au public. Il contient notamment le contrôleur frontal de Symfony.
- src: C'est ici que la magie s'opère, c'est ici que vous écrirez vos fichiers PHP. Les contrôleurs, entités, migrations, ... sont dans ce dossier.

# Hiérarchie d'un projet symfony

- templates: Les vues de notre application sont ici, toutes les pages qui vont être affichées à l'écran vont être ici.
- tests: C'est ici que vous allez mettre les différents tests pour tester vos fonctionnalités.
- var: Dans ce dossier Symfony va mettre les caches et logs
- vendor: Ce dossier contient toutes les dépendances créées avec composer.
- .env: Ce fichier définit les variables d'environnement de notre application, il définit l'environnement dans lequel nous sommes, développement ou production, les informations de connexion à la BDD, ...

# Hiérarchie d'un projet symfony

## #Contrôleur Frontal

- Le contrôleur frontal est le fichier d'entrée de notre application, toute requête va passer par ce fichier avant de continuer. Le contrôleur frontal de Symfony 4 se situe dans `public/index.php`, il réceptionne donc toutes les requêtes qui arrivent à notre application et renvoie la réponse adéquate au client.

## #php bin/console

- Celui ci représente en quelque sorte la ligne de commande de Symfony, elle va par exemple nous permettre de créer nos contrôleurs, créer des entités, des migrations etc. Pour voir toutes les commandes possible, il suffit de faire:

# Hiérarchie d'un projet symfony

## \$ php bin/console

- Nous avons alors la liste de toutes les commandes disponibles, vous pouvez donc lire la documentation de chaque commande et essayer de comprendre ce qu'elle fait.
- **#Doctrine**
- Doctrine est ce qu'on appelle un ORM (Object Relational Mapper), pour faire simple, il va nous permettre de créer, modifier, supprimer ou récupérer des objets dans la base de données en utilisant des classes.

# Hiérarchie d'un projet symfony

## #Contrôleur

- L'un des composants de l'acronyme MVC, le contrôleur est une fonction PHP que nous allons créer et qui va permettre de lire une requête et renvoyer une réponse, c'est tout son fonctionnement, recevoir une requête et retourner une réponse.

## # Route

- La route représente le lien entre une URL et un contrôleur. Quand le serveur reçoit une requête HTTP, symfony regarde dans le fichier config/routes.yaml pour trouver la route, et cette route va définir le contrôleur à appeler dans ce cas.

# Hiérarchie d'un projet symfony

## #Requêtes et Réponses

- Symfony fournit une approche à travers deux classes pour interagir avec la requête et la réponse HTTP. La classe Request est une représentation de la requête HTTP, tandis que la classe Response est évidemment une représentation de la réponse HTTP.
- Un moyen de gérer ce qui se passe entre la requête et la réponse consiste à utiliser un contrôleur frontal (public/index.php). Ce fichier traitera chaque demande entrant dans notre application. Cela signifie qu'il sera toujours exécuté et qu'il gérera le routage de différentes URL vers différentes parties de notre application.

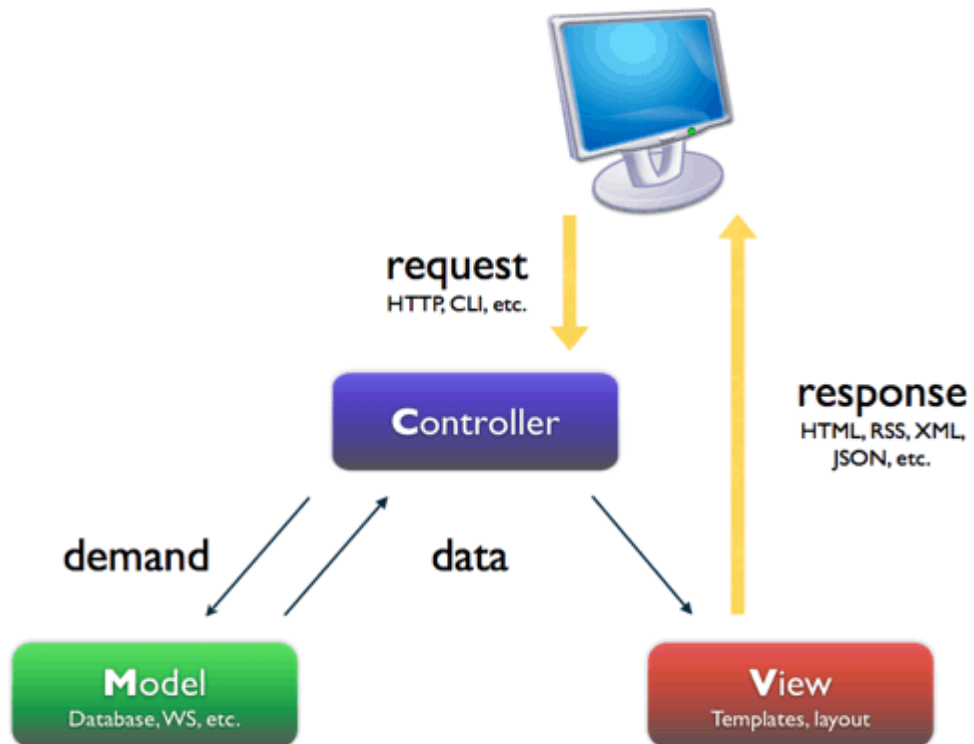


# Hiérarchie d'un projet symfony

- Dans Symfony, les demandes entrantes sont interprétées par le composant Routing et transmises aux fonctions PHP (contrôleurs) qui renvoient des réponse. Cela signifie que le contrôleur frontal transmettra la demande à Symfony. Ce dernier créera un objet de réponse et le transformera en en-têtes de texte et le contenu sera finalement renvoyé.

# Le Modèle MVC

- Symfony est un framework basé sur le modèle MVC (Model View Controller).
- Pour faire simple, le modèle MVC va nous aider à séparer les requêtes de la base de données (Model) de la logique relative au traitement des demandes (Controller) et au rendu de la présentation (View).



La demande de l'utilisateur (exemple : requête HTTP) est reçue et interprétée par le Contrôleur. Celui-ci utilise les services du Modèle afin de préparer les données à afficher. Ensuite, le Contrôleur fournit ces données à la Vue, qui les présente à l'utilisateur (par exemple sous la forme d'une page HTML).

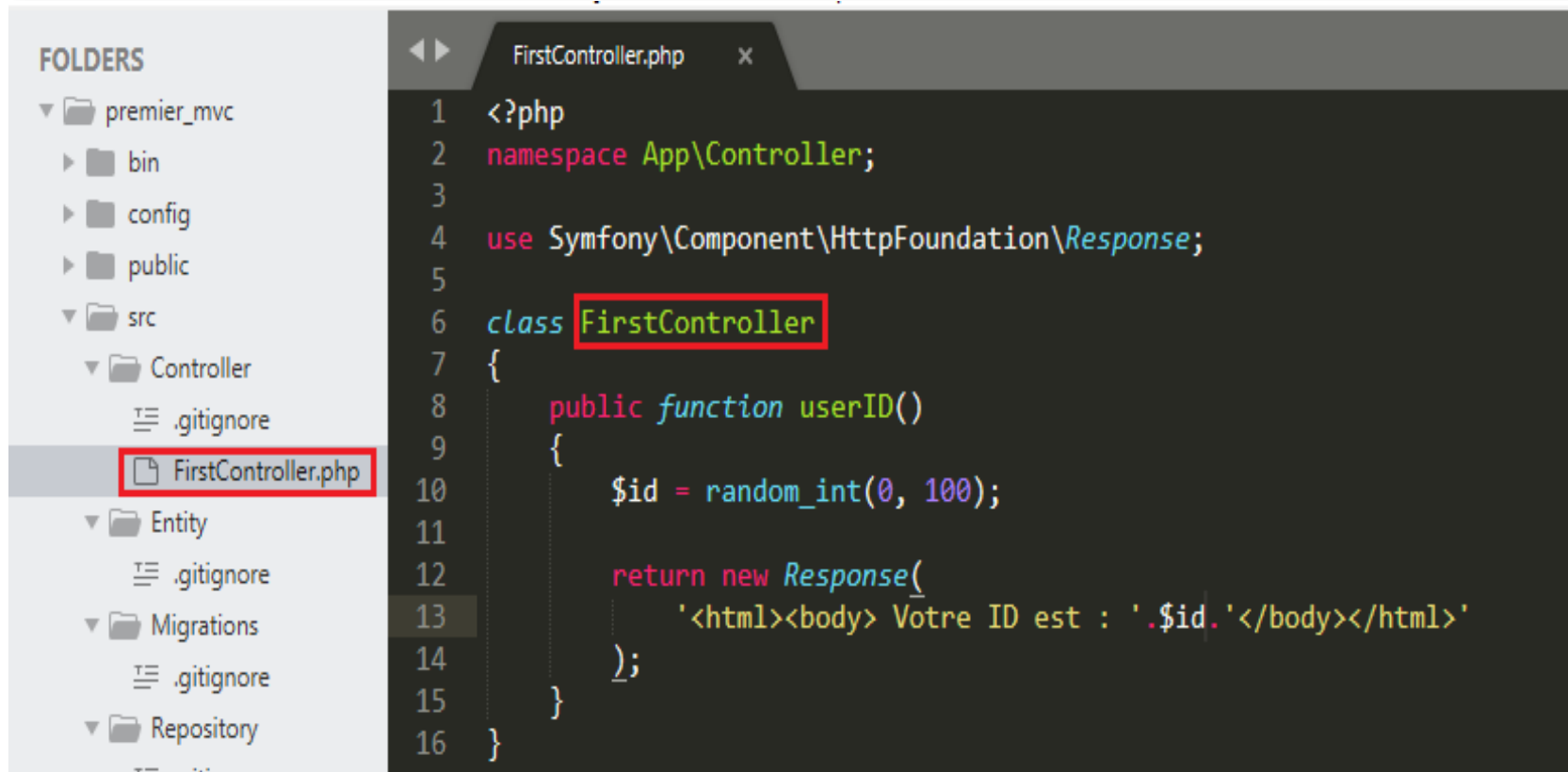
# Manipulation de la couche MVC

- Dans cette section, on va comprendre les concepts de base du framework Symfony à savoir :
  - Les contrôleurs
  - Les actions avec passage de paramètres
  - Le routage dans des fichiers yml et en mode annotation
  - L'affichage via l'objet response et le moteur twig.

# Les controleurs

- Un controleur n'est autre qu'une classe PHP contenant des méthodes qui s'exécutent une fois elles correspondent à une url (route).
- Dans une première étape, nous créons un controller de façons manuelle, suite à ça nous allons installer un bundle externe « **MakerBundle** » qui va nous simplifier la génération de code de façon automatique par simple ligne de commande.
- Créons notre premier controleur « **FirstController** » sous le répertoire Controller.
- La convention ici est que tout controleur doit être préfixé par le mot clé Controller.

# FirstController



The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer, titled 'FOLDERS', shows a project structure with 'premier\_mvc' as the root. It contains subfolders 'bin', 'config', 'public', and 'src'. The 'src' folder is expanded, showing 'Controller', 'Entity', 'Migrations', and 'Repository'. The 'Controller' folder is further expanded, showing a file named 'FirstController.php' which is highlighted with a red box. The code editor on the right shows the content of 'FirstController.php' with line numbers 1 through 16. The code is as follows:

```
1 <?php
2 namespace App\Controller;
3
4 use Symfony\Component\HttpFoundation\Response;
5
6 class FirstController
7 {
8     public function userID()
9     {
10         $id = random_int(0, 100);
11
12         return new Response(
13             '<html><body> Votre ID est : '.$id.'</body></html>'
14         );
15     }
16 }
```

# FirstController

- Nous avons également utiliser l'objet Response pour afficher du contenu sur navigateur.
- On peut également remarquer la présence de « namespace » : Espace de nommage, c'est une sorte de package pour la résolution de porté.

# Première Action

- `<?php`
- `namespace App\Controller;`
- `use Symfony\Component\HttpFoundation\Response;`
- `class FirstController {`
- `public function userID() {`
- `$id = random_int(0, 100);`
- `return new Response(`
- `'<html><body> Votre ID est : '.$id.'</body></html>');`
- `}`
- `}`

# Routing

- Dans une deuxième étape, il faut associer à cette action une route. Allez au fichier « config/routes.yaml » et ajouter les 3 lignes suivantes
- `first_controller_userID_route:`
- `path: /first/userid`
- `controller: App\Controller\FirstController::userID`
- Une route est identifiée par son nom, son path et enfin l'action associée. La tabulation c'est quatre espace. Lancer votre navigateur en tapant « <http://127.0.0.1:8000/first/userid> »
- Relancer avec un refresh et un nouvel ID apparaîtra.



# Nouveau Problème (Nouvelles versions)

- Si un problème de driver apparaît, il faut :
  - Aller au fichier .env dans l'arborescence (les dernières lignes)
  - Commenter avec # postgresql et décommenter mysql : login = root, MP vide , port 3306 ou 3308, nom de la base, puis sauvegarder
  - Tapez php bin/console doctrine:database:create (prochainement)
  - Sauvegardez et relancez
  - Si nécessaire, supprimez cache et log dans le répertoire var.

# Actions avec passage de paramètres

- `public function info($formation, $formateur)`
- `{`
- `return new Response(`
- `'<html><body> Formation : '.$formation.'`  
`Formateur : '.$formateur.'</body></html>'`
- `);`
- `}`

# Actions avec passage de paramètres

```
FirstController.php x routes.yaml x
1 #index:
2 #   path: /
3 #   controller: App\Controller\DefaultController::inde
4
5 first_controller_userID_route:
6     path: /first/userid
7     controller: App\Controller\FirstController::userID
8
9
10 first_controller_info_route:
11     path: /first/{formation}/{formateur}
12     controller: App\Controller\FirstController::info
```

# Routage dans des fichiers yaml et en mode annotation

- La définition des routes ne se limite pas au fichier **routes.yaml**. L'annotation qui consiste à définir les routes dans le fichier controller est désormais possible via le package « annotations » ou « sensio/framework-extra-bundle ».
- 1-Faites l'import suivant (ajouter dans les use du controller) :
- `use Symfony\Component\Routing\Annotation\Route;`
- 2-Ajouter avant l'action l'annotation suivante :

# Routage via Annotations

- `/**`
- `* @Route("/first/index")`
- `*/`

```
23
24  /**
25   * @Route("/first/index")
26   */
27  public function index()
28  {
29
30      return new Response(
31          '<html><body> <h1 align=center>Hello Symfony</h1></body></html>'
32      );
33  }
34 }
```

- Si l'action échoue, Taper la commande « **composer require annotations** » ou « **composer require sensio/framework-extra-bundle** »

# Affichage via twig

- Pour afficher un contenu issu d'une méthode, On peut également utiliser TWIG : c'est un moteur de templating comme on l'appelle permettant de gérer la partie vue du framework.
- Suite à cette modification, créer un fichier `index.html.twig` sous le dossier « `templates/first/index.html.twig` » et apporter les changements suivants à votre contrôleur.
- Dans des versions anciennes, il faut parfois installer le package twig : « `composer require twig` »

# Mise à jour controleur

```
6 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController; (1)
```

```
7  
8 class FirstController extends AbstractController (2)  
9 {
```

```
22     return $this->render('first/index.html.twig', [  
23         'formation' => $formation, 'formateur' => $formateur, (3)  
24     ]);  
25 }
```

# Le fichier twig

```
FirstController.php x index.html.twig x
1 <h1 align="center">Hello Twig</h1>
2
3
4 <h2>Formation : {{formation}}</h2>
5
6 <h2>Formateur : {{formateur}}</h2>
```



# Le fichier twig

- Ainsi tout contrôleur doit hériter de la classe `AbstractContrôleur` est ceci afin de bénéficier de plusieurs méthodes telque « **render()** ».
- Il est également conseillé de choisir un nom de dossier des vues comme celui du controller. Avec le moteur de template twig, on peut faire plein de choses qu'on peut pas faire dans un fichier html classique :
  - Les structures conditionnelles et itératives
  - Les filtres via les pipes
  - L'héritage de templates via les blocks
  - Portabilité via la fonction `asset()`
  - Créer des liens entre pages

# Le fichier twig

- La syntaxe Twig est basée sur ces trois constructions :
- `{{ ... }}`, utilisé pour afficher le contenu d'une variable ou le résultat de l'évaluation d'une expression.
- `{% ... %}`, utilisé pour exécuter une logique, telle qu'une condition ou une boucle.
- `{# ... #}`, utilisé pour ajouter des commentaires au modèle.
- (contrairement aux commentaires HTML, ces commentaires ne sont pas inclus dans la page rendue).

# Le fichier twig

- Tout d'abord, on peut remarquer que notre barre de débogage est disparue, ceci est due au fait que notre vue est dépourvue de partie `<body></body>`. Pour ce faire on va apporter les modifications suivantes à nos vues « **index.html.twig** » et « **base.html.twig** » (qui représente le template père de toute l'application).

# Le fichier twig

```
FirstController.php x index.html.twig x base.html.twig x ro
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <meta charset="UTF-8">
5         <title>{% block title %}Welcome!{% endblock %}</title>
6         {% block stylesheets %}{% endblock %}
7     </head>
8     <body>
9         {% block body %}{% endblock %}
10        {% block javascripts %}{% endblock %}
11    </body>
12 </html>
13
```

# Héritage et Construction de blocks

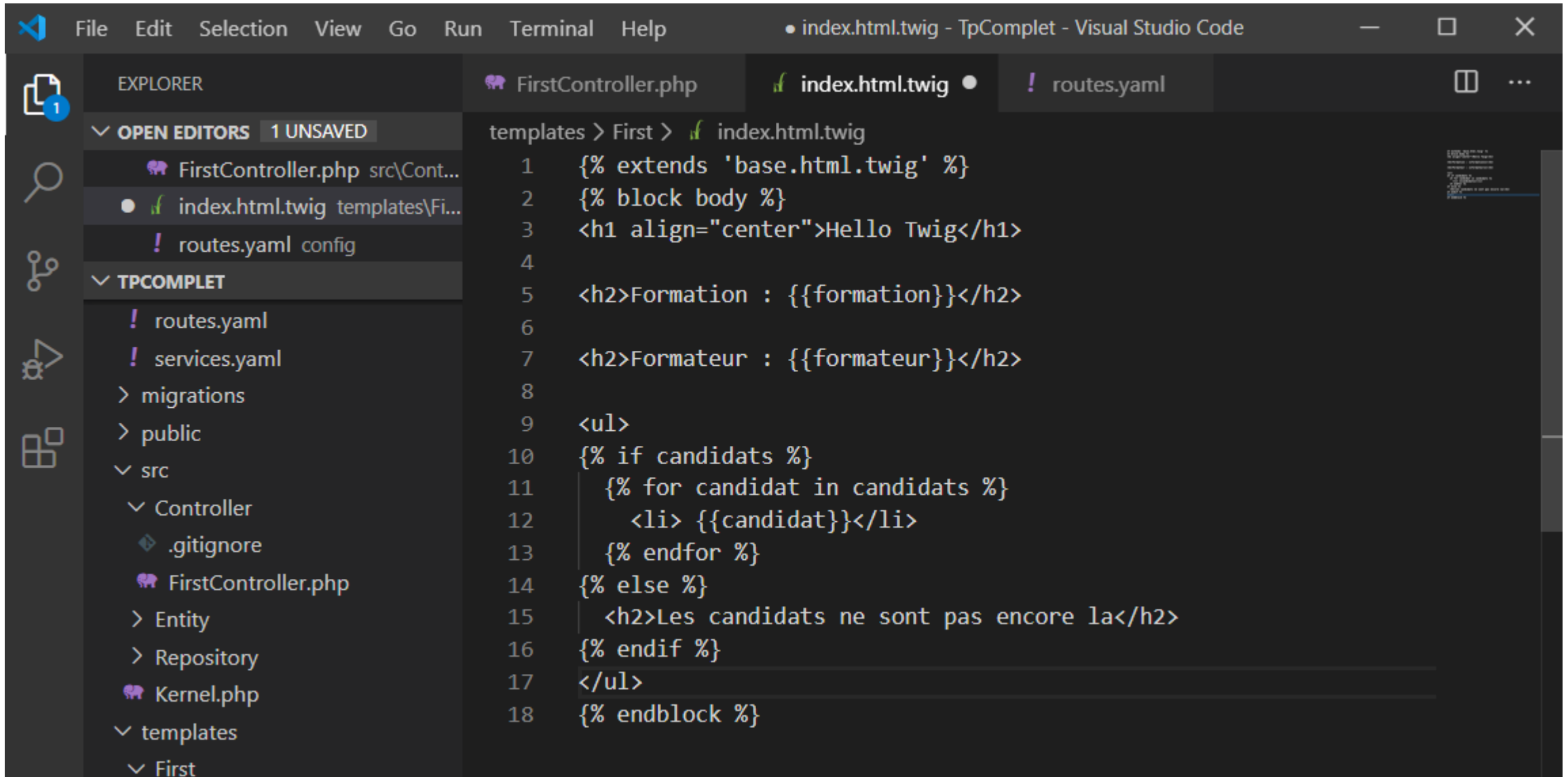
```
FirstController.php x index.html.twig
1
2 {% extends 'base.html.twig' %} (1)
3
4 {% block body %} (2)
5 <h1 align="center">Hello Twig</h1>
6
7
8 <h2>Formation : {{formation}}</h2>
9
10 <h2>Formateur : {{formateur}}</h2>
11 {% endblock %} (2)
12
```

# Boucle sur Array

```
16  /**
17   * @Route("/first/index")
18   */
19   public function index(){
20   //      return new Response('<html><body><h1 align=center>Hello Symfony</h1></body></html>');
21       $formation = "Symfony 4.4";
22       $formateur = "Makrem Mhedhbi";
23       $candidats = array("Anissa","Walid","Olfa","Moez");
24       return $this->render('first/index.html.twig',[
25           'formation' => $formation, 'formateur' => $formateur, 'candidats' => $candidats
26       ]);
27   }
```

```
FirstController.php  index.html.twig ×  routes.yaml
templates > First > index.html.twig
1  {% extends 'base.html.twig' %}
2  {% block body %}
3      <h1 align="center">Hello Twig</h1>
4
5      <h2>Formation : {{formation}}</h2>
6
7      <h2>Formateur : {{formateur}}</h2>
8
9      <ul>
10         {% for candidat in candidats %}
11             <li> {{candidat}}</li>
12         {% endfor %}
13     </ul>
14     {% endblock %}
```

# Tests et boucles



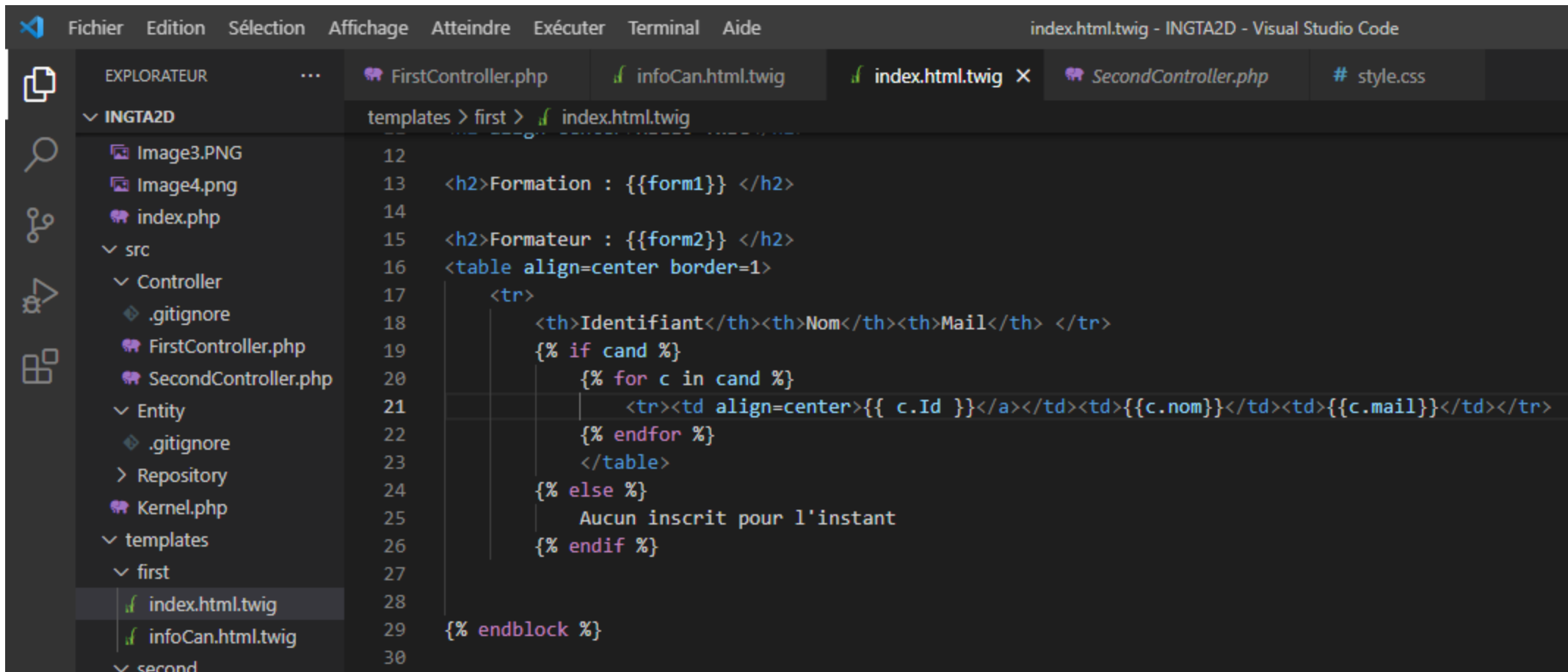
The screenshot shows the Visual Studio Code interface with the following components:

- Explorer Panel:** Displays the project structure. The 'templates' directory is expanded, showing a 'First' subdirectory. The file 'index.html.twig' is selected.
- Editor Panel:** Shows the content of 'index.html.twig'. The file is a Twig template with the following code:

```
1 {% extends 'base.html.twig' %}
2 {% block body %}
3 <h1 align="center">Hello Twig</h1>
4
5 <h2>Formation : {{formation}}</h2>
6
7 <h2>Formateur : {{formateur}}</h2>
8
9 <ul>
10 {% if candidats %}
11     {% for candidat in candidats %}
12         <li> {{candidat}}</li>
13     {% endfor %}
14 {% else %}
15     <h2>Les candidats ne sont pas encore la</h2>
16 {% endif %}
17 </ul>
18 {% endblock %}
```
- Terminal Panel:** Empty.

# Tests et boucles

Nous pouvons utiliser la balise table pour améliorer l'affichage



```
Fichier  Edition  Sélection  Affichage  Atteindre  Exécuter  Terminal  Aide  index.html.twig - INGTA2D - Visual Studio Code

EXPLORATEUR  ...  FirstController.php  infoCan.html.twig  index.html.twig x  SecondController.php  # style.css

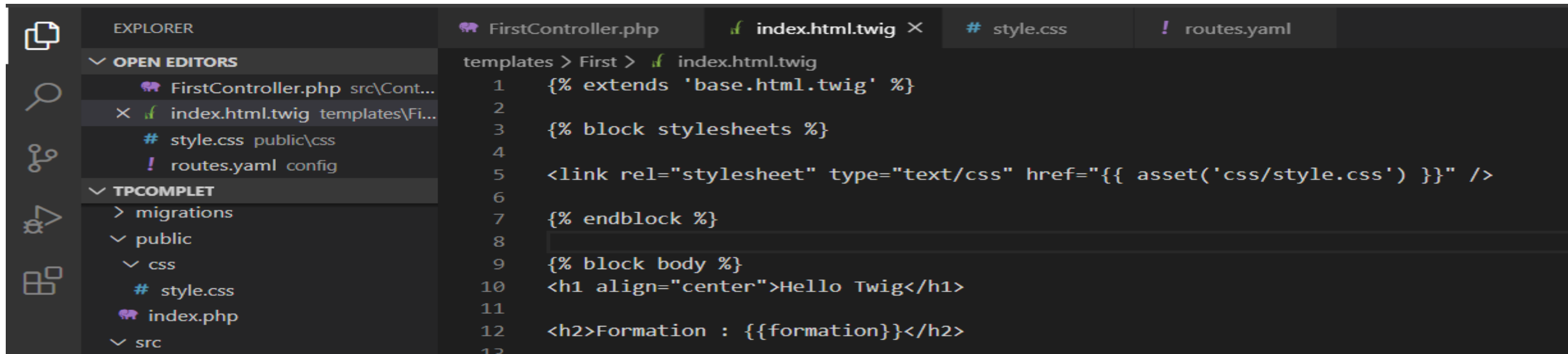
INGTA2D
├── Image3.PNG
├── Image4.png
├── index.php
├── src
│   ├── Controller
│   │   ├── .gitignore
│   │   ├── FirstController.php
│   │   └── SecondController.php
│   ├── Entity
│   │   ├── .gitignore
│   │   └── Repository
│   ├── Kernel.php
│   └── templates
│       ├── first
│       │   ├── index.html.twig
│       │   ├── infoCan.html.twig
│       └── second
```

```
12
13 <h2>Formation : {{form1}} </h2>
14
15 <h2>Formateur : {{form2}} </h2>
16 <table align=center border=1>
17     <tr>
18         <th>Identifiant</th><th>Nom</th><th>Mail</th> </tr>
19         {% if cand %}
20             {% for c in cand %}
21                 <tr><td align=center>{{ c.Id }}</a></td><td>{{c.nom}}</td><td>{{c.mail}}</td></tr>
22             {% endfor %}
23         </table>
24         {% else %}
25             Aucun inscrit pour l'instant
26         {% endif %}
27
28
29 {% endblock %}
30
```



# Twig et Effet CSS

- Pour ajouter un effet css à votre page, on pourra également utiliser la méthode **asset** pour lier des ressources statiques dans votre page.
- Ajouter asset via composer en tapant : `composer require symfony/asset`
- Dans `index.html.twig`, ajouter

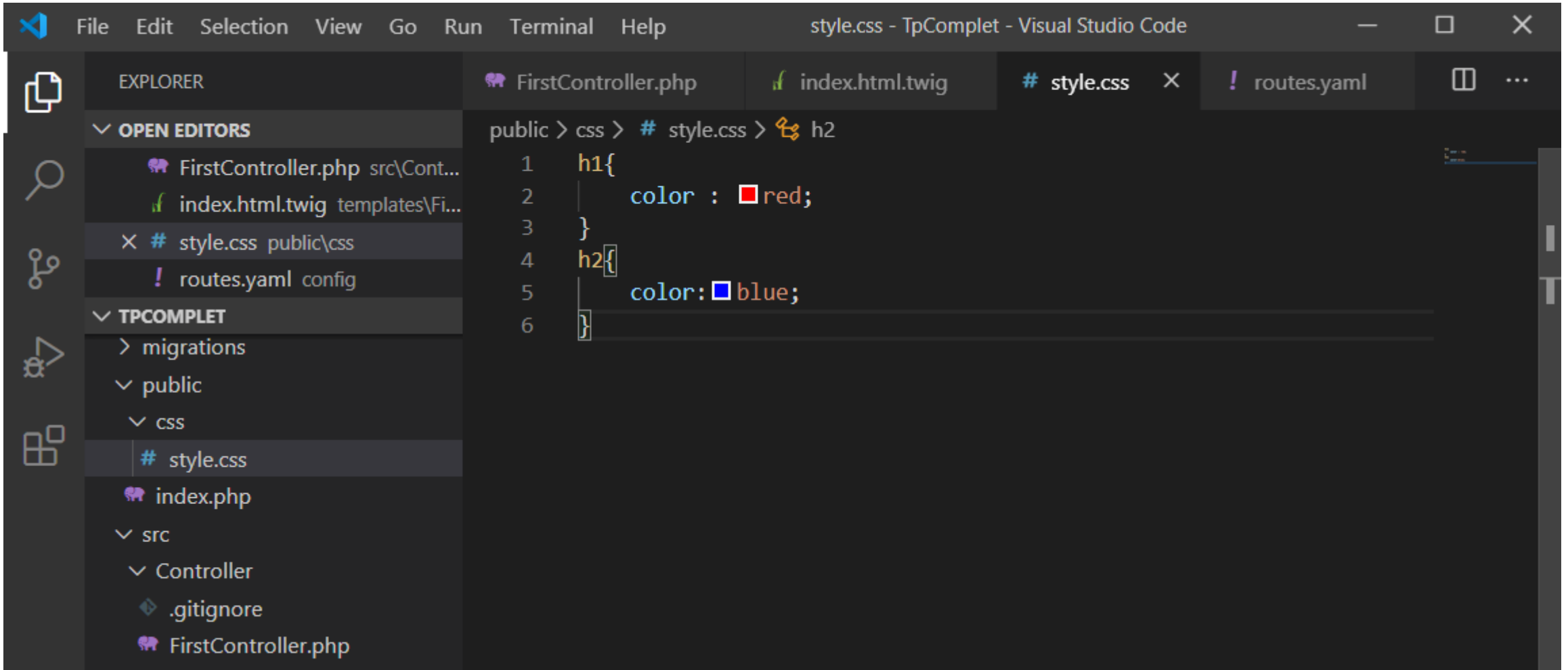


```
EXPLORER
├── OPEN EDITORS
│   ├── FirstController.php src\Cont...
│   └── index.html.twig templates\Fi...
├── # style.css public\css
└── ! routes.yaml config

TPCOMPLET
├── > migrations
├── > public
│   ├── > css
│   │   ├── # style.css
│   │   └── index.php
└── > src

templates > First > index.html.twig
1  {% extends 'base.html.twig' %}
2
3  {% block stylesheets %}
4
5  <link rel="stylesheet" type="text/css" href="{{ asset('css/style.css') }}" />
6
7  {% endblock %}
8
9  {% block body %}
10 <h1 align="center">Hello Twig</h1>
11
12 <h2>Formation : {{formation}}</h2>
13
```

# Fichier CSS



style.css - TpCompleet - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

OPEN EDITORS

- FirstController.php src\Cont...
- index.html.twig templates\Fi...
- # style.css public\css
- ! routes.yaml config

TPCOMPLET

- migrations
- public
  - css
    - # style.css
- src
  - Controller
  - .gitignore
  - FirstController.php

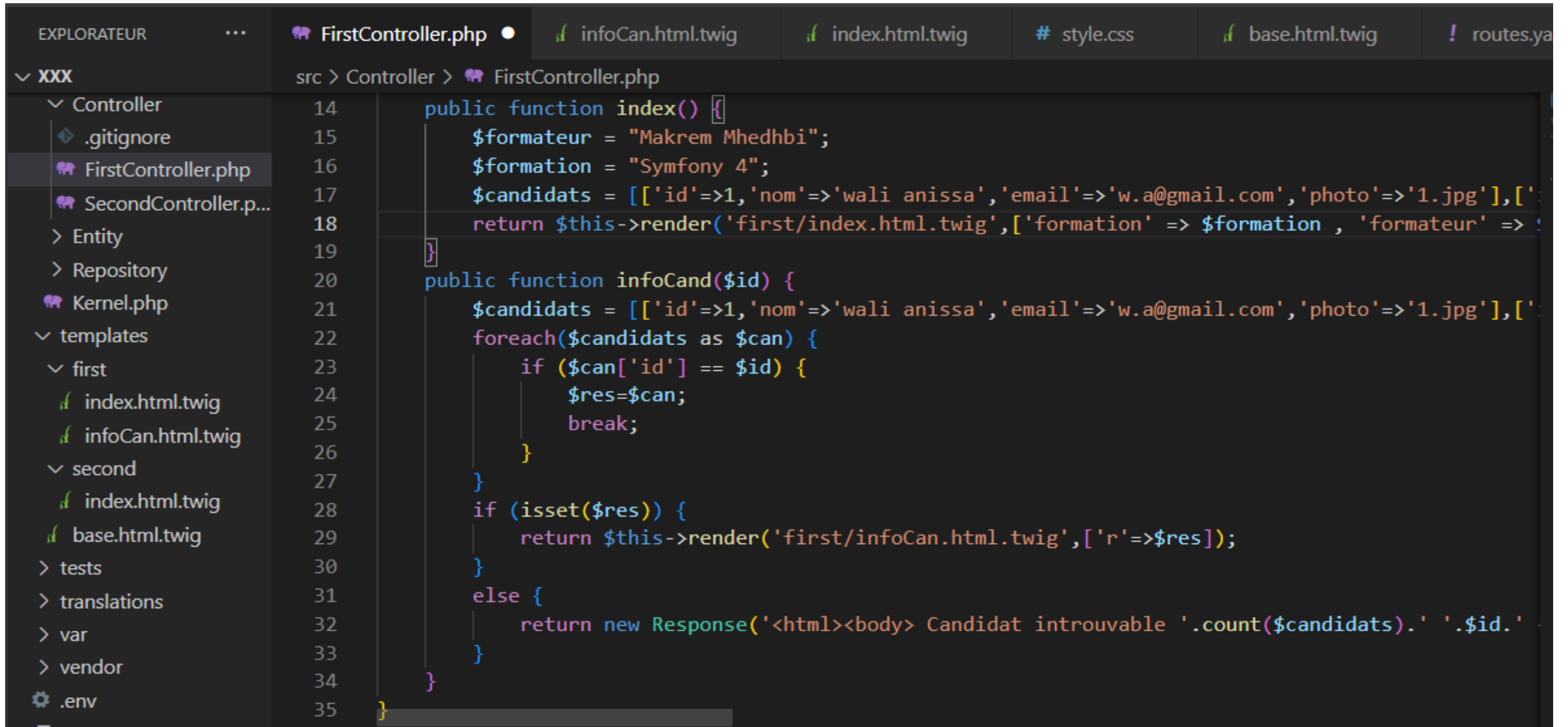
public > css > # style.css > h2

```
1 h1{
2   color : red;
3 }
4 h2{
5   color: blue;
6 }
```

## Liens entre vues

- Nous avons l’affichage du résultat de l’action index (id, nom, mail).
- Objectif : l’identifiant de chaque candidat sera sous forme d’un lien hypertexte. Si on clique dessus, nous serons dirigés vers une autre vue, qui contient toutes les informations du candidat choisi.
- 1<sup>ère</sup> étape : préparer la nouvelle action infoCan et sa vue (first/infoCan.html.twig)

# Liens entre vues



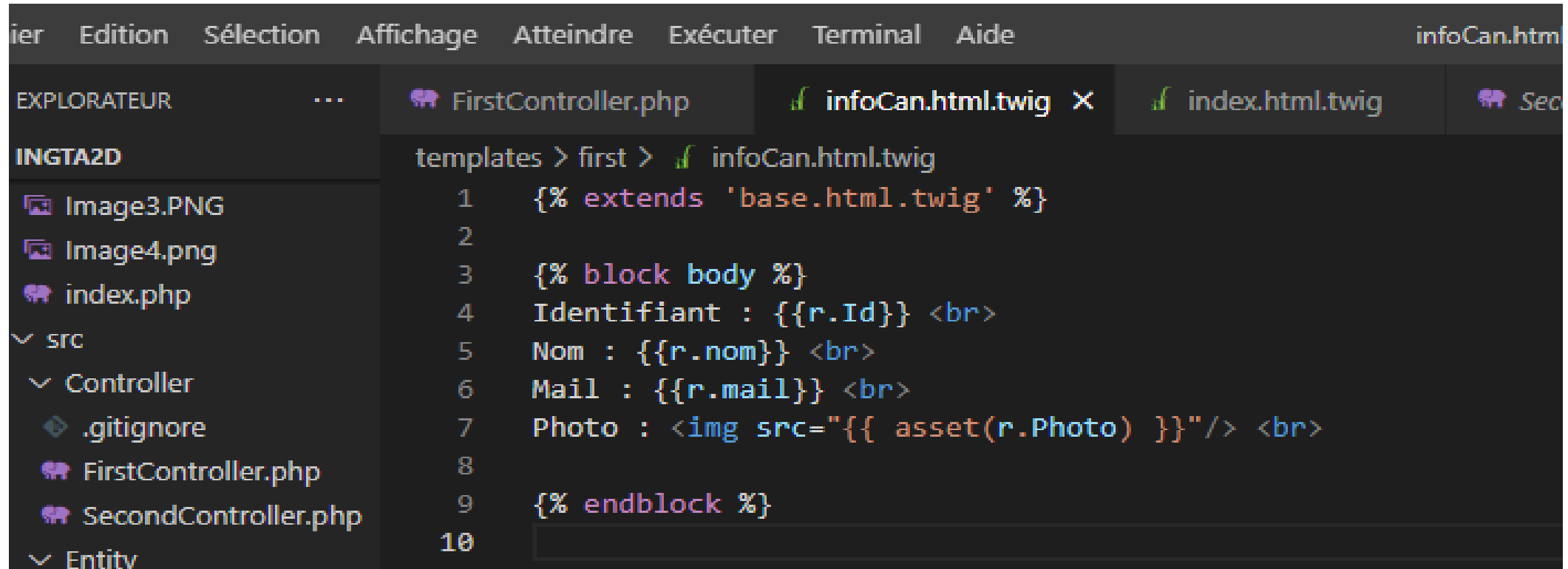
The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'Controller' directory containing 'FirstController.php' and 'SecondController.php', and a 'templates' directory with sub-directories 'first' and 'second'. The code editor displays the 'index()' and 'infoCand()' methods of 'FirstController.php'. The 'index()' method renders 'first/index.html.twig' with context variables. The 'infoCand()' method finds a candidate by ID and renders 'first/infoCan.html.twig' with the candidate's data.

```
EXPLORATEUR  ...  FirstController.php  infoCan.html.twig  index.html.twig  # style.css  base.html.twig  ! routes.ya

src > Controller > FirstController.php

14 public function index() {
15     $formateur = "Makrem Mhedhbi";
16     $formation = "Symfony 4";
17     $candidats = [['id'=>1,'nom'=>'wali anissa','email'=>'w.a@gmail.com','photo'=>'1.jpg'],[
18     return $this->render('first/index.html.twig',['formation' => $formation , 'formateur' =>
19 }
20 public function infoCand($id) {
21     $candidats = [['id'=>1,'nom'=>'wali anissa','email'=>'w.a@gmail.com','photo'=>'1.jpg'],[
22     foreach($candidats as $can) {
23         if ($can['id'] == $id) {
24             $res=$can;
25             break;
26         }
27     }
28     if (isset($res)) {
29         return $this->render('first/infoCan.html.twig',['r'=>$res]);
30     }
31     else {
32         return new Response('<html><body> Candidat introuvable '.count($candidats).' '.$id.'
33     }
34 }
35 }
```

# Liens entre vues

A screenshot of an IDE window showing a Twig template file named 'infoCan.html.twig'. The left sidebar shows a project structure with folders like 'src', 'Controller', and 'Entity'. The main editor area displays the content of 'infoCan.html.twig' with line numbers 1 through 10. The code extends 'base.html.twig' and contains a block 'body' with several lines of HTML output using Twig variables and functions like 'asset' and 'path'.

```
1 {% extends 'base.html.twig' %}
2
3 {% block body %}
4   Identifiant : {{r.Id}} <br>
5   Nom : {{r.nom}} <br>
6   Mail : {{r.mail}} <br>
7   Photo :  <br>
8
9 {% endblock %}
10
```

- Dans index.html.twig, remplacer la ligne 21 (cf diapo 31) par :
- `<a href="{{path('first_controller_infoCand_route',{id:c.Id})}}">{{ c.Id }}</a>`

# Liens entre vues (Autre exemple)

- Dans le controleur, ajouter :
- `public function description(){  
    $description = "Ce workshop aura comme objectifs la présentation des: controleurs, routes et vues";  
    return $this->render('first/description.html.twig',['description' => $description]); }`
- Dans routes.yaml, ajouter :
  - `first_controller_description_route:`
    - `path: /first/description`
    - `controller: App\Controller\FirstController::description`
- Puis, créez le fichier `description.html.twig`

# Liens entre vues (Autre exemple)

- `{% extends 'base.html.twig' %}`
- `{% block body %}`
- `<h2 align="center">Description de la formation</h2>`
- `{{description}}`
- `{% endblock %}`
- Dans `index.html.twig`, ajouter un lien vers `description.html.twig`
  - pour plus de détails, cliquer `<a href=`  
`"{{path('first_controller_description_route')}}" >ici</a>`
- Tester le résultat

# Génération automatique de contrôleurs

- Pour cela, il faut installer le composant maker-bundle :
- `composer require symfony/maker-bundle --dev`
- Ensuite, pour créer un nouveau contrôleur par exemple :
- `php bin/console make:controller SecondController`
- Le résultat est la création de deux fichiers :
  - `SecondController.php`
  - `Index.html.twig` dans le répertoire `second` du répertoire `templates`
- Dans le navigateur, tapez `127.0.0.1:8000/second`