

Séance 05 :

→ **DJANGO** – Initiation aux moteurs de templates – *Vue*

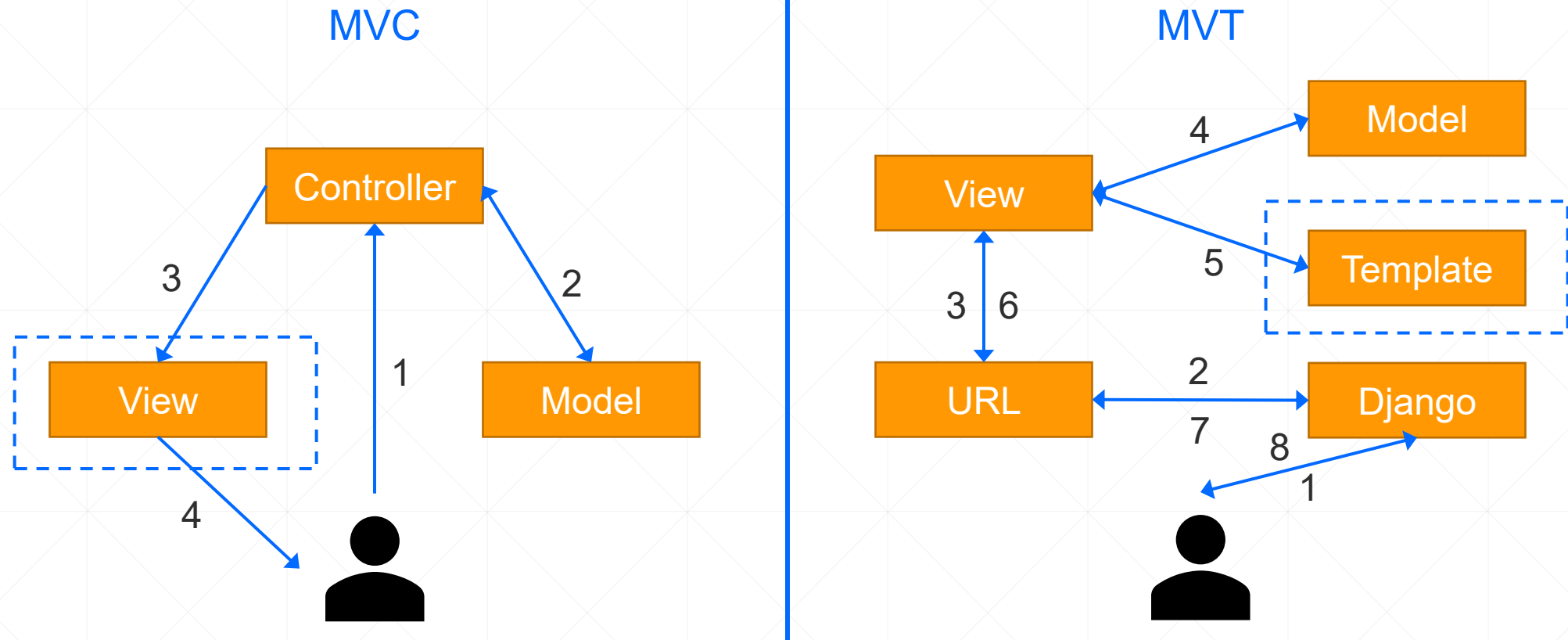
INF37407 – Technologie de l'inforoute

Prof. Yacine YADDADEN, Ph. D.

Plan

1. Rappel – Architecture MVC & MVT
2. Principe des gabarits ou *templates*
3. Le langage des gabarits ou *templates*
4. Application de l'*héritage* et des *inclusions*
5. Gestion des *fichiers statiques*
6. Utilisation des URLs
7. Introduction aux formulaires
8. Mise en pratique
9. Questions & Discussion

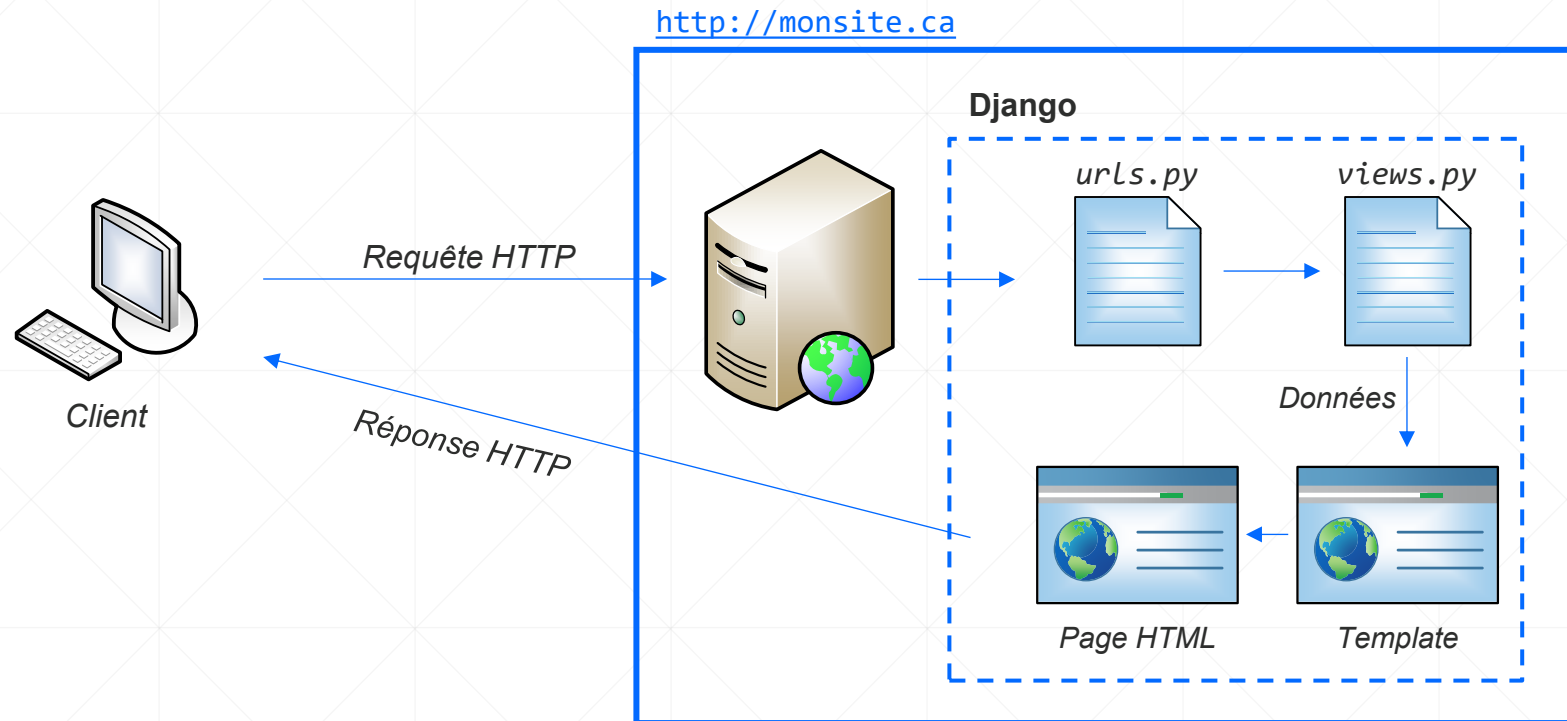
Rappel – Architecture MVC & MVT



Principe des gabarits ou *templates*

- **Définition :** « *Ce sont tout simplement des fichiers HTML. Par conséquent, ils contiennent des balises HTML. Ils peuvent également faire appel à d'autres fichiers CSS ou JavaScript.* »,
- **Rôle :**
 - Ils définissent l'interface ou le *rendu graphique* présenté à l'utilisateur,
 - Il est possible d'y insérer du code Python pour les *variables* et *structures de contrôles*.
- **Fonctionnement :**
 - Des gabarits ou *templates* (fichiers HTML) sont définis,
 - Il est possible de les appeler à partir des vues ou *views* pour un rendu graphique,
 - Ils peuvent être **dynamiser** par des *variables transmises* ou *structure de contrôles*,
 - Ils peuvent s'imbriquer, sont réutilisable et bénéficient du principe d'*héritage*.

Schéma global de fonctionnement



Un peu plus de détails

- **Le moteur de *templates* :**

- C'est le composant *responsable de la gestion et du rendu des templates*,
- Django dispose de son *propre moteur*, mais il est possible d'en utiliser d'autres : **Jinja2**.

- **Étapes suivies :**

1. Une *requête HTTP* est envoyée par l'utilisateur,
2. Un **view** est appelé après sélection en utilisant l'**url**,
3. Le **view** demande un rendu avec un *template* et transmet des données,
4. Sur la base du *template*, une page **HTML** est constituée à l'aide des données,
5. Une *réponse HTTP* avec la page **HTML** est renvoyée à l'utilisateur.

Le langage des gabarits ou *templates*

Ci-dessous quelques exemples, mais pour plus de détails, voir la documentation¹.

1. Les variables :

- Variable simple : `<h1>Salut {{ name }} !</h1>`
- Attribut d'un objet : `<h1>Salut {{ person.name }} !</h1>`
- Éléments d'une liste : `<h1>Salut {{ names.2 }} !</h1>` *Troisième élément*

2. Formatage et filtres :

- Mettre en minuscules : `<h1>Salut {{ name|lower }} !</h1>`
- Convertir le premier élément : `<h1>Salut {{ names|first|upper }} !</h1>`
- Afficher la longueur d'une variable : `<h1>Salut {{ name|length }} !</h1>`

¹ <https://docs.djangoproject.com/fr/4.1/ref/templates/>

Le langage des gabarits ou *templates*

Ci-dessous quelques exemples, mais pour plus de détails, voir la documentation¹.

3. Les conditions :

```
{% if person.gender == "F" %}  
  <h1>Bonjour madame {{ person.name }} !</h1>  
{% else %}  
  <h1>Bonjour monsieur {{ person.name }} !</h1>  
{% endif %}
```

4. Les boucles :

```
{% for person in persons %}  
  <h1>Bonjour monsieur {{ person.name }} !</h1>  
{% endfor %}
```

¹ <https://docs.djangoproject.com/en/3.1/ref/templates/>

Génération d'un rendu

- Il faut créer un dossier *templates* dans le dossier racine du projet Django,
 - Pour les fichiers HTML partagés par les applications, ils sont mis à la racine (*templates*),
 - Pour chaque application, un *sous-dossier* portant son nom doit être créé.
- Dans le fichier **settings.py** du projet Django, il faut modifier dans **TEMPLATES** :
 - De `'DIRS': [],` vers `'DIRS': [os.path.join(BASE_DIR, "templates")],`
 - Il faut également : `import os`
- Il faut ensuite créer un *template*, exemple : `base.html` (*voir diapo suivante*)
- Dans le fichier **views.py**, il faut mettre :

```
def home(request):  
    return render(request, "base.html", {"name": "Yacine"})
```

Génération d'un rendu

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Document</title>
  </head>
  <body>
    <h1>Salut {{ name }} !</h1>
  </body>
</html>
```

Application de l'héritage et des *inclusions*

L'héritage et les *inclusions* permettent la *réutilisation* des *templates* afin d'écrire moins de code HTML.

1. Héritage :

- On dispose d'un fichier **base.html** qui est considéré comme le *parent*,
- On y déclare des *blocs remplaçables* à l'aide de :

```
{% block block_name %}{% endblock %}
```

- D'autres fichiers **page1.html** et **page2.html** peuvent hériter de **base.html** avec :

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

- Ensuite, on y remplace le contenu des blocs (*voir diapo suivante*).

Application de l'héritage et des *inclusions*

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>{% block title %}{% endblock %}</title>
  </head>
  <body>

  </body>
</html>
```

base.html

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

{% block title %}Page 1{% endblock %}
```

page1.html

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

{% block title %}Page 2{% endblock %}
```

page2.html

Application de l'héritage et des *inclusions*

2. Inclusion :

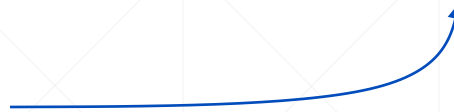
- On dispose de partie de la page HTML `part.html` qu'on souhaite réutiliser :

```
<h1>Salut {{ name }} !</h1>
```

- Dans la page HTML principale `main.html`, on fera appel à `part.html` via :

```
{% include "part.html" with name="Yacine" %}
```

- On y passe également des variables.



Gestion des *fichiers statiques*

- Lors du développement d'une application, il sera éventuellement nécessaire de faire appel à des *fichiers statiques*, exemple : CSS, JavaScript, images, ...
- Il faut créer un dossier *static* dans le dossier racine du projet Django,
 - Pour les fichiers statiques partagés par les applications sont mis à la racine (*static*),
 - Pour chaque application, un *sous-dossier* portant son nom doit être créé.
- Dans le fichier **settings.py** du projet Django, rajouter :

```
STATICFILES_DIRS = [os.path.join(BASE_DIR, "static")]
```

- Il faut ensuite créer un *fichier statique*, exemple : `style.css`
- Dans le fichier **base.html**,
 - il faut mettre :

```
{% load static %}
```

```
<link rel="stylesheet" href="{% static 'style.css' %}" />
```

Utilisation des URLs

- Il est possible d'utiliser des URLs dans les *templates*,
- L'écriture se fait comme suit :

```
<a href="{% url 'view_name' 'variable_value' %}">Lien</a>
```

- Dans le fichier `urls.py` :

```
path('view/<str:name>', view, name="view_name")
```

- Dans le fichier `views.py` :

```
def view(request, name):  
    return render(request, "base.html", {"name": name})
```

Introduction aux formulaires

Afin d'utiliser les formulaires, il faut s'assurer de :

- Pour éviter les erreurs **CSRF** : `{% csrf_token %}`
- Mettre une valeur à `name` pour les différents champs, `request.method`
- Il y a différentes méthodes à utiliser dans l'objet `request` : **POST** et **GET**,
- Afin de récupérer une valeur d'un certain champs :

`request.POST["field_name"]` Ou `request.POST.get("field_name")`

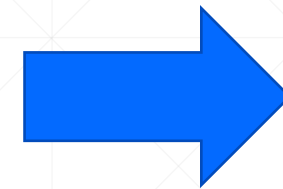
- Pour utiliser des redirections :

```
from django.shortcuts import redirect
```

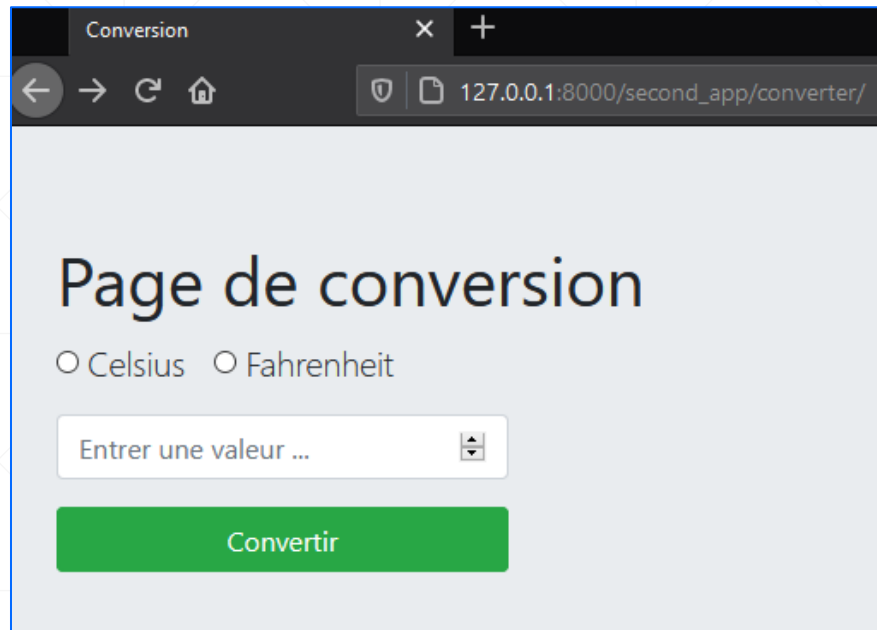
```
return redirect('view_name')
```

Démonstration

- Améliorer le rendu de l'application pour la conversion entre :
 - **Degré Celsius °C** et **Degré Fahrenheit °F**
- Pour cela, il faut utiliser :
 - Création de la page HTML de base,
 - Création des différentes pages HTML (*héritage*),
 - Utilisation du **Bootstrap** et *fichier CSS* pour améliorer le rendu graphique.
- Le résultat attendu est comme suit :



Démonstration



Conversion

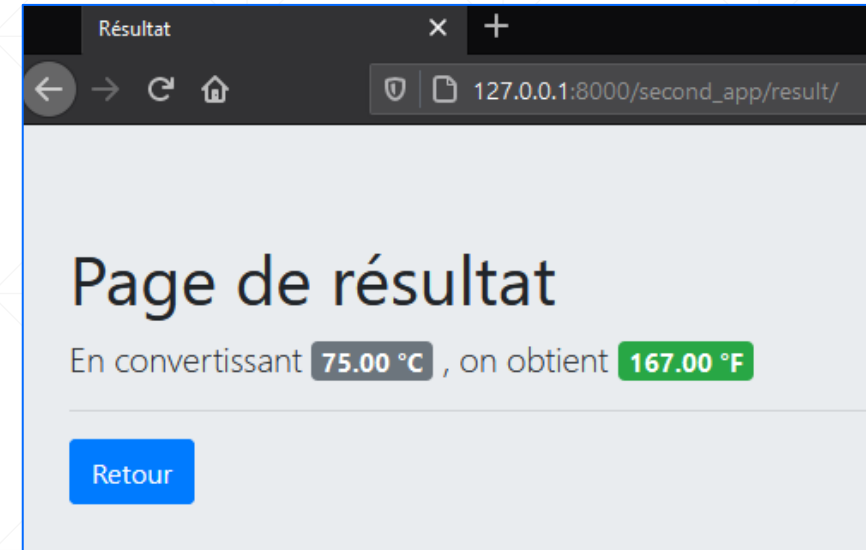
127.0.0.1:8000/second_app/convert/

Page de conversion

☐ Celsius ☐ Fahrenheit

Entrer une valeur ...

Convertir



Résultat

127.0.0.1:8000/second_app/result/

Page de résultat

En convertissant 75.00 °C , on obtient 167.00 °F

Retour

Questions & Discussion

Bibliographie

1. Bersini, H, Alexis, P. & Degols, G. (2018). Apprendre la programmation web avec Python et Django. Eyrolles.
2. Samson, P. (2020). DJANGO Développez vos applications web en Python. Édition ENI.
3. Haverbeke, M. (2014). Eloquent javascript: A modern introduction to programming. No Starch Press.
4. Melé, A. (2020). Django 3 By Example: Build powerful and reliable Python web applications from scratch. P
5. Hillar, G. C. (2018). Django RESTful Web Services: The Easiest Way to Build Python RESTful APIs and Web Services with Django. Packt Publishing Ltd.ack Publishing Ltd.