Séance 06 :

→ DJANGO – Présentation des ORM – *Modèle*

INF37407 – Technologie de l'inforoute

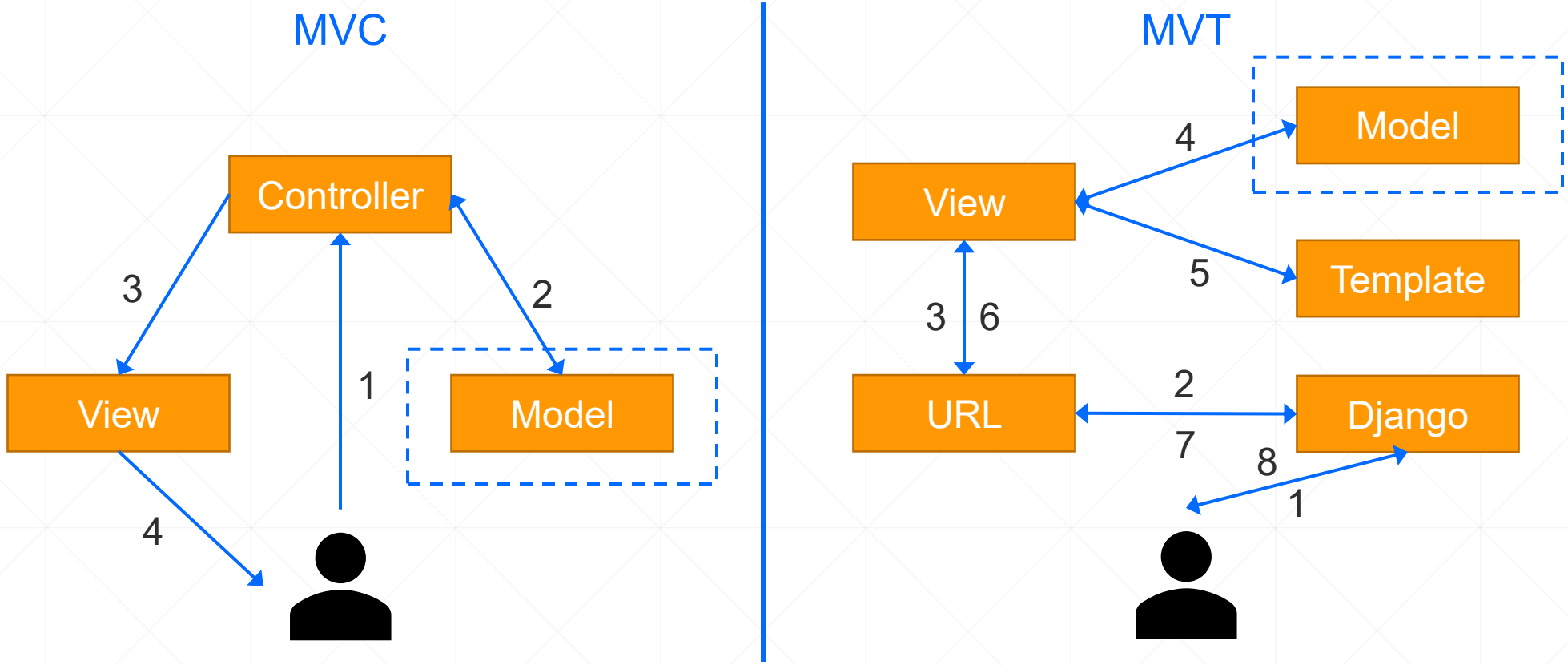
Prof. **Yacine YADDADEN**, Ph. D.

UQAR

Plan

1. Rappel – Architecture MVC & MVT
2. Les modèles Django et **ORM**
3. Notre premier modèle
4. Configuration de la base de données
5. Création du premier modèle – *Champs* et *Validation*
6. Introduction aux relations
7. Utilisation des modèles
8. Démonstration
9. Questions & Discussion

Rappel – Architecture MVC & MVT



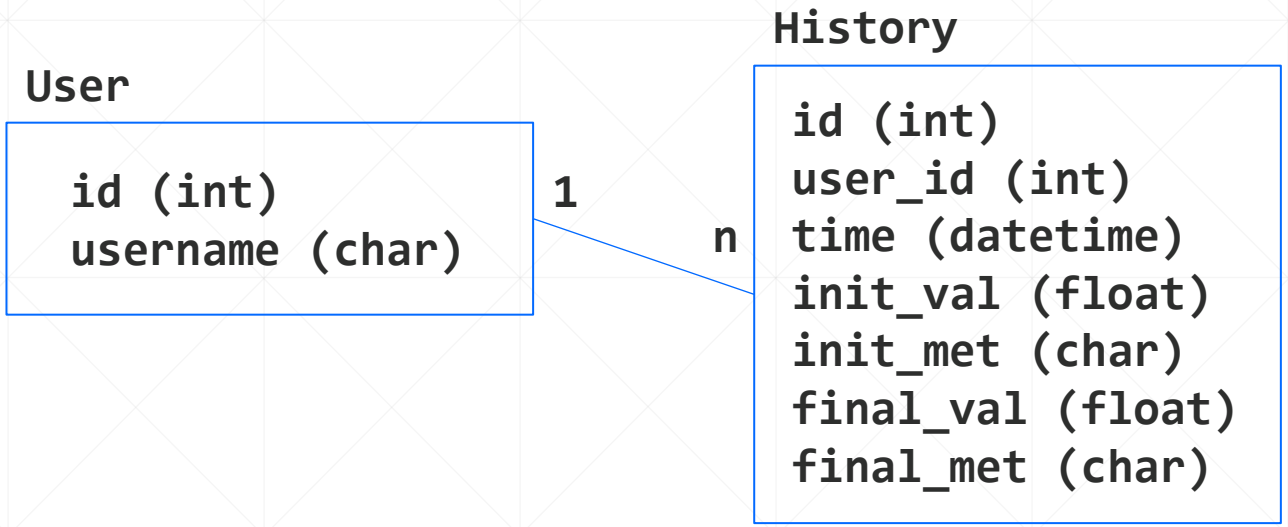
UQAR

Les modèles Django et ORM

- **Définition :** « *Le modèle est le composant du **MVC** ou **MVT** qui est en charge des données et plus précisément de leur représentation ainsi que de leur manipulation.* »,
- **ORM** ou ***O**bject-**R**elational **M**apping* (*en français : Mapping Objet-Relationnel*) est le programme qu'utilise le modèle pour la gestion des données,
- Il permet de créer une *couche d'abstraction* entre l'***aspect relationnel*** (*base de données SQL*) et l'***aspect orienté objet*** (*langage Python*),
- Ainsi les *tables* seront vues comme des *classe*, les *champs* comme des *attributs* et les *requêtes SQL* comme des *méthodes,*
- **Objectif :** *c'est de rendre la manipulation des données plus aisée et moins sujette à des erreurs.*

Notre premier modèle

- En se basant sur l'application de démonstration (*conversion de température*) :



- Ainsi pour chaque utilisateur, on gardera un historique des différentes conversions qu'il aura fait.

UQAR

Configuration de la base de données

- Par défaut, une base de données ***SQLite3*** est configurée,
- La configuration se trouve dans le fichier *project_name*/**settings.py** :

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

- C'est le moyen le plus simple d'implémenter la *persistance des données* pour une application Django.

UQAR

Changer la structure de la base de données

- Il y a *deux commandes principales* auxquelles on fait appel pour travailler avec le **modèle** et les bases de données :
  - **`makemigrations`** qui demande à Django d'*analyser le fichier* **`models.py`** pour voir s'il y eu des changements. Si c'est le cas, des fichiers de migrations seront créés.
  - **`migrate`** qui demande à Django d'aller chercher l'ensemble des *fichiers de migrations* créés par la commande précédente et de *les appliquer sur la base de données configurée*.

- Lors de la création d'un nouveau projet Django, *une structure de base de données* est créée également avec les informations du *super utilisateur*.

UQAR

Base de données MySQL

Afin de configurer une base de données MySQL, il faut :

1. Procéder à l'installation du *connecteur* avec :
   - Commande : **`pip install`** *`mysqlclient`*
   - Ensuite éditer la configuration dans *`project_name`*`/`**`settings.py`** :

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'OPTIONS': {
            'read_default_file': os.path.join(BASE_DIR, "my.cnf"),
        },
    }
}
```

UQAR

Base de données MySQL

3. Créer un fichier de configuration : **`my.cnf`**
4. Il doit contenir les informations suivantes :

```
# my.cnf
[client]
Host = HOST
port = PORT
database = NAME
user = USER
password = PASSWORD
default-character-set = utf8
```

Remplacer les valeurs

5. Il faut remplacer les valeurs **`HOST`**, **`PORT`**, **`NAME`**, **`USER`** et **`PASSWORD`**.

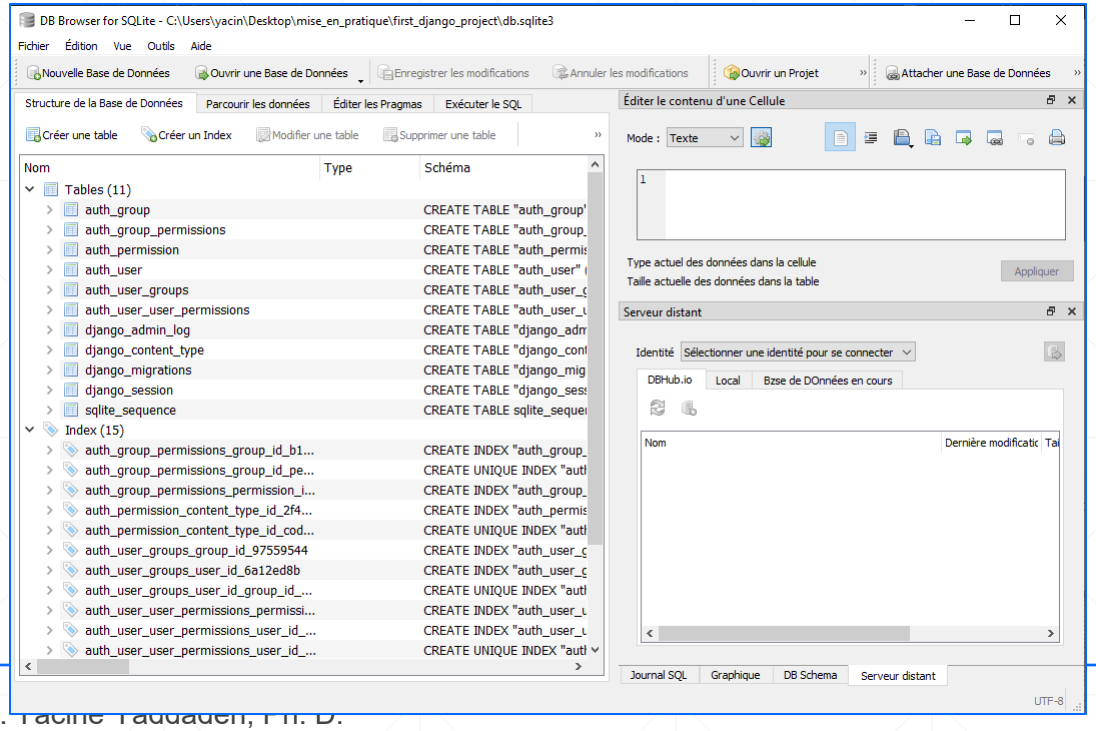
Visualiser la base de données

Suivant le type de base de données adopté, on utilisera :

- **SQLite3** :
  - Outil : **DB Browser for SQLite**
  - Lien : https://sqlitebrowser.org/
- **MySQL** :
  - Outil : **MySQL Workbench**
  - Lien : https://www.mysql.com/fr/products/workbench/

Démonstration

1. Une fois installé, ouvrir l'utilitaire **DB Browser for SQLite**,
2. Ouvrir le fichier **`db.sqlite3`** et avoir le résultat suivant :



UQAR

Création du premier modèle – *Les champs*

Pour créer notre modèle, il faut éditer le fichier *app_name*/**`models.py`** :

1. Création de la *table* **`User`** :

```
class User(models.Model):
    id = models.AutoField(primary_key=True)
    username = models.CharField(max_length=10)
```

Définition d'un champ

2. Création de la *table* **`History`** :

```
class History(models.Model):
    id = models.AutoField(primary_key=True)
    user = models.ForeignKey("User", on_delete=models.CASCADE)
    time = models.DateTimeField()
    initial_val = models.FloatField()
    final_val = models.FloatField()
    MET_CHOICES = (("C", "Celsieus"), ("F", "Fahrenheit"))
    initial_met = models.CharField(max_length=1, choices=MET_CHOICES, validators=[validate_choice])
    final_met = models.CharField(max_length=1, choices=MET_CHOICES, validators=[validate_choice])
```

La suppression d'un utilisateur, supprime son historique de conversion

Application des validateurs

N'est pas créé

UQAR

Création du premier modèle – *Validation* [1]

- Il est possible d'ajouter des validations, par exemple :
  - Sur les champs **`init_met`** et **`final_met`**,
  - Ils ne peuvent avoir que les valeurs : "F" (*Fahrenheit*) ou "C" (*Celsius*).
- Au niveau du *modèle*, on ajoute :

```
from django.core.exceptions import ValidationError

def validate_choice(value):
    if (value != "C" and value != "F"):
        raise ValidationError("The metric must be either 'C' or 'F'.")
```

[1] https://docs.djangoproject.com/fr/4.1/ref/validators/

Création du premier modèle – *Validation* (suite)

- Habituellement, la *validation* est appliquée lors de la réception des données du formulaires,
- Cependant, il est possible de forcer la validation au niveau du modèle en redéfinition la méthode **`save()`** comme suit :

```
def save(self, *args, **kwargs):
    self.full_clean()
    super(History, self).save(*args, **kwargs)
```

- La méthode **`full_clean()`** permet de déclencher la validation,
- Voici un exemple de message d'erreur :

```
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "C:\Users\yaddya01\Desktop\Démonstration_Django\second_app\models.py", line 23, in sa
    super(History, self).save(*args, **kwargs)
  File "C:\Users\yaddya01\Desktop\Démonstration_Django\django_venv\lib\site-packages\django\
full_clean
    raise ValidationError(errors)
django.core.exceptions.ValidationError: {'final_met': ["Value 'D' is not a valid choice."]}
```

UQAR

Création du premier modèle – *Les champs*

3. Afin d'appliquer les changements au niveau de la base de données :
   - Première commande : `python manage.py` **`makemigrations`** *`app_name`*
   - Seconde commande : `python manage.py` **`migrate`** *`app_name`*

Important : *S'il y a des erreurs au niveau du* **`makemigrations`***, il faut les corriger avant d'appliquer le* **`migrate`***.*

Création du premier modèle – *Les champs*

- Il y a *différents champs* qui peuvent être utilisés (voir la documentation[1]) :
  - **Booléen** : `models. [BooleanField(), NullBooleanField ()]`
  - **Date** *et/ou* **Temps** : `models. [DateField (), TimeField (), DateTimeField (), …]`
  - **Nombre** : `models. [IntegerField (), FloatField (), DecimalField (), …]`
  - **Texte** : `models. [CharField (), TextField (), FilePathField (), EmailField (), …]`
- Chacun d'eux peut avoir des arguments en entrée tel que :

```
models.CharField(max_length=10)
```

Longueur maximale de la chaîne de caractères

[1] https://docs.djangoproject.com/fr/4.1/topics/db/models/

Introduction aux relations

- **Définition :** « *C'est ce qui permet de créer des liaisons entre les différentes entités ou tables de la base de données.* »,
- Il y a principalement trois types de relation :
  1. **Liens *1-n*** : *une personne peut avoir plusieurs numéros, mais un numéro n'appartient qu'à une seule personne.*
     - Utilise le champ : **`models.`***`ForeignKey('entity_name')`*

     Spécifier la table de liaison
  2. **Liens *n-n*** : *un étudiant peut suivre plusieurs cours et un même cours peut être suivi par plusieurs étudiants. (interdit en **mode relationnel**, utilise une **table intermédiaire**).*
     - Utilise le champ : **`models.`***`ManyToManyField ('entity_name')`*
  3. **Liens *1-1*** : *un étudiant possède un dossier étudiant et un dossier étudiant n'appartient qu'à un seul et unique étudiant.*
     - Utilise le champ : **`models.`***`OneToOneField('entity_name')`*

UQAR

L’héritage

Il est possible qu’*une entité hérite d’une autre* étant donné qu’elles sont définies sous *forme de classes*, par exemple :

- Entité *utilisateur* :

```
class User(models.Model):
    username = models.CharField(max_length=10)
```

- Entité *administrateur* qui a en plus un champs mot de passe :

```
class Admin(User):
    password = models.CharField(max_length=10)
```

Important : *L’entité utilisateur doit être définie avant celle d’administrateur.*

UQAR

Utilisation des modèles

Plusieurs opérations peuvent effectuée au niveau du modèle, mais il faut *importer les modèles* au niveau de la *vue (`views.py`)* :

```
from .models import User, History
```

1. Ajout des données :

```
user = User(username='yacine')
user.save()
```

2. Récupération de plusieurs enregistrements :

```
users = User.objects.all()
```

   - Il y a possibilité d'*appliquer des filtres* avec **`.filter()`**

3. Tri des données :

```
users = User.objects.all().order_by('username')
```

UQAR

Utilisation des modèles

4. Récupération d'un enregistrement unique : `user = User.objects.get(id=1)`
5. Suppression d'enregistrements :

```
user = User.objects.get(id=1)
user.delete()
```

```
users = User.objects.all()
users.delete()
```

6. Accès aux objets *liés* : `user = History.objects.get(id=1).user`

```
history = User.objects.get(id=1).history_set.all()
```

7. Mise à jour :

```
user = User.objects.get(id=1)
user.username = 'nadia'
user.save()
```

UQAR

Faire des tests de façon interactive

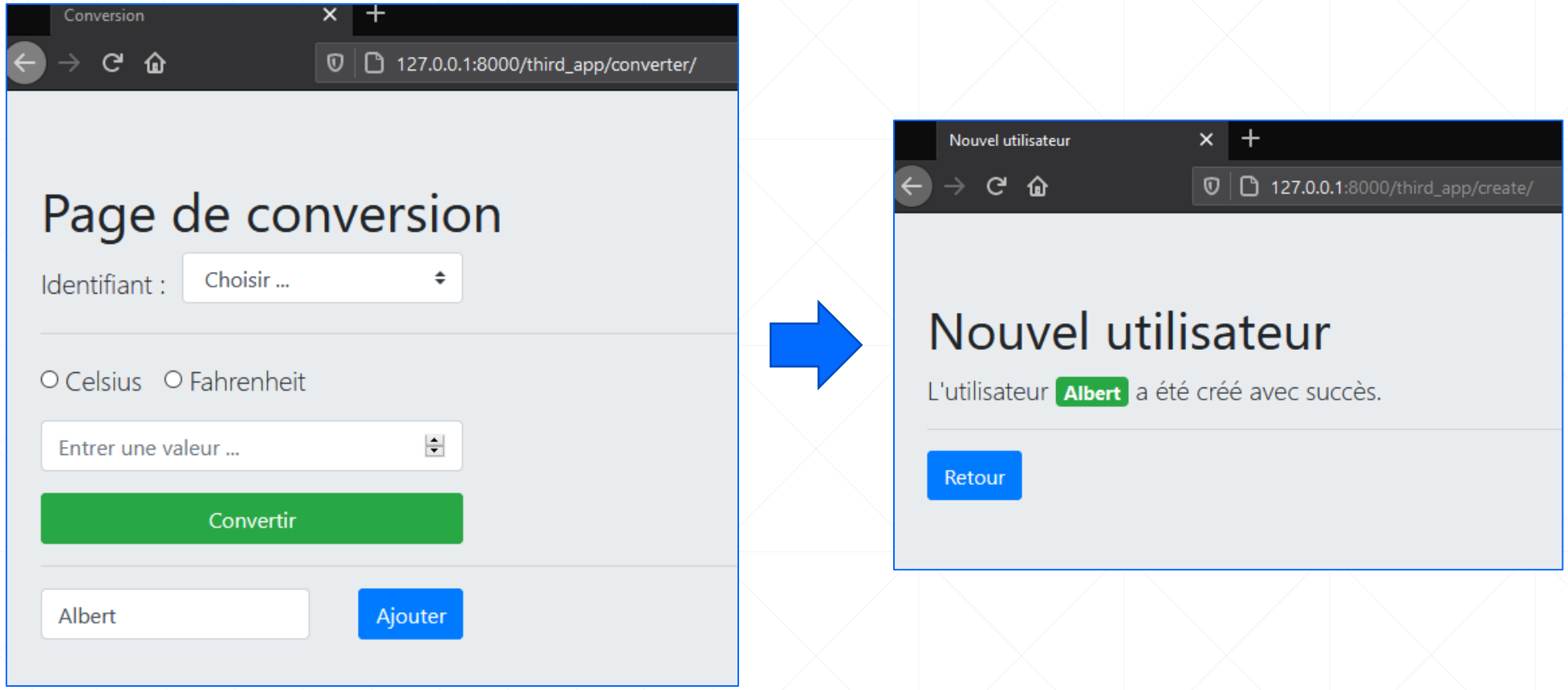
1. Lancer la commande suivante :
   - Commande : `python manage.py shell`
2. Importer le modèle à utiliser :
   - Commande : `from app_name.models import User`
3. Effectuer une opération :
   - Commande (*récupérer*) : `users = User.objects.all()`
   - Commande (*afficher*) : `print(users.first().username)`

UQAR

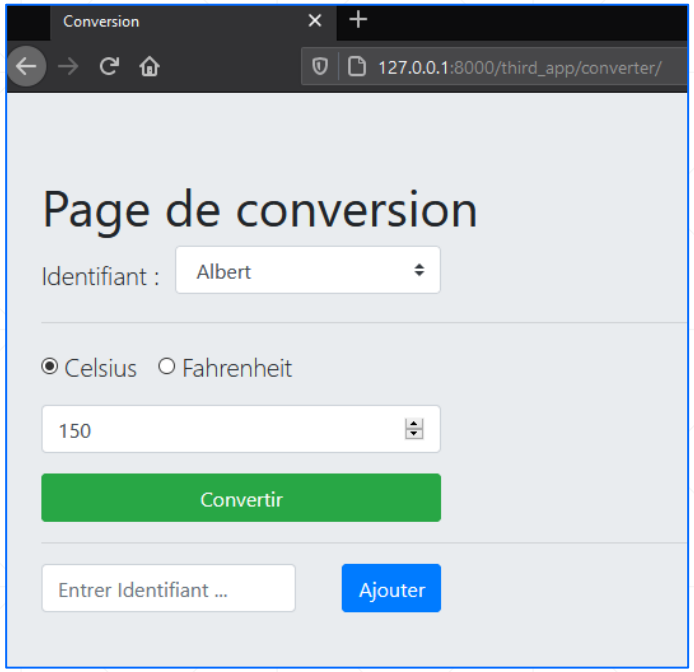
Démonstration

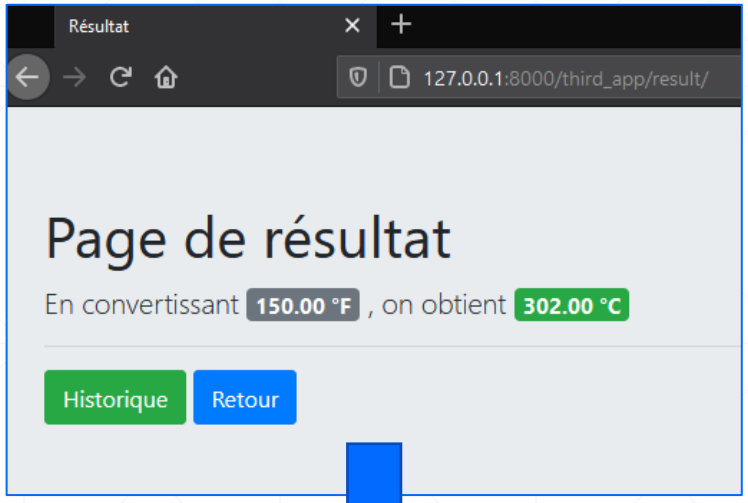
- Implémenter la *persistance des données* pour l'application pour la conversion entre :
  - **Degré Celsius °C** et **Degré Fahrenheit °F**
- Ajouter les fonctionnalités :
  - Ajout de nouveaux utilisateurs,
  - Gérer l'historique des conversions par utilisateur.
- Pour cela, il faut utiliser :
  - Configurer l'accès à la base de données,
  - Créer les modèles nécessaires,
  - Adapter les *vues* pour les traitements.
- Le résultat attendu est comme suit :

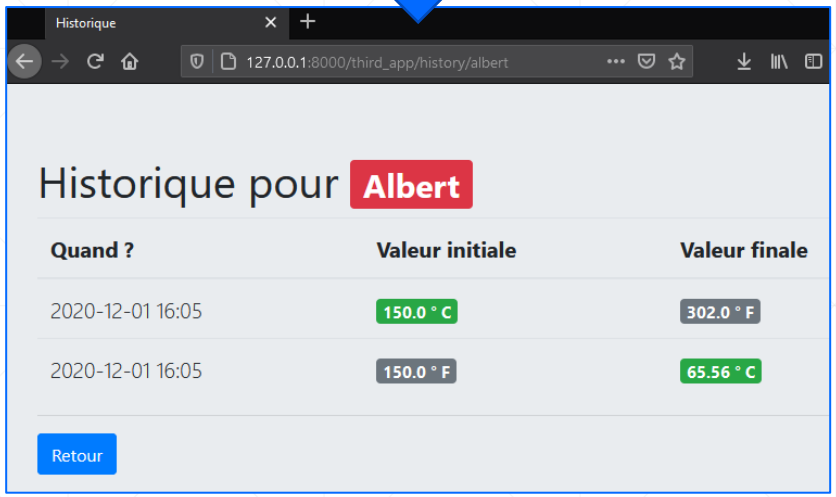
Démonstration



Démonstration







Questions & Discussion

UQAR

Bibliographie

1. Bersini, H, Alexis, P. & Degols, G. (2018). Apprendre la programmation web avec Python et Django. Eyrolles.
2. Samson, P. (2020). DJANGO Developpez 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.ackt Publishing Ltd.