Test Technique - Stage KYOLIS 2023

Affichage d'informations d'un modèle dans une template Django

Imane Makhloufi

Master 2 en informatique - Université Savoie Mont Blanc

Introduction :

L'objectif de cet exercice est de créer une application Django qui affiche des informations à partir d'un modèle dans une template. Nous allons créer un modèle simple représentant des livres et afficher les détails de ces livres dans une template.

Nous allons travailler dans l'environnement de développement PyCharm qui est un IDE puissant et qui nous fera bénéficier d'une interface intuitive, de fonctionnalités avancées telles que l'autocomplétion, la détection d'erreurs, le débogage, ainsi que d'une intégration étroite avec notre framework Django.

Exercice

Question 1 :

L'architecture d'une application Django suit le modèle MVC (Modèle-Vue-Contrôleur) ou, plus précisément, le modèle MTV (Modèle-Template-Vue). Voici une brève explication de chaque composant :

Modèle (Model) : Le modèle représente les données de l'application. Il définit la structure des données, les relations entre les différentes entités et les méthodes pour interagir avec ces données. En Django, les modèles sont généralement définis en utilisant la classe ***models.Model*** et ses différents champs (par exemple, CharField, IntegerField, etc.).

Vue (View) : La vue est responsable du traitement des requêtes HTTP et de la logique métier associée. Elle récupère les données du modèle, effectue les opérations nécessaires et renvoie les résultats à la template appropriée. En Django, les vues sont généralement des fonctions ou des classes basées sur la classe ***View***.

Template : La template est responsable de la présentation des données. Elle définit la structure HTML de la page et utilise des balises spéciales pour afficher les données dynamiques. En Django, les templates utilisent le moteur de template intégré pour rendre le contenu dynamique. On peut accéder aux données du modèle en utilisant la syntaxe de balises spécifiques dans les templates.

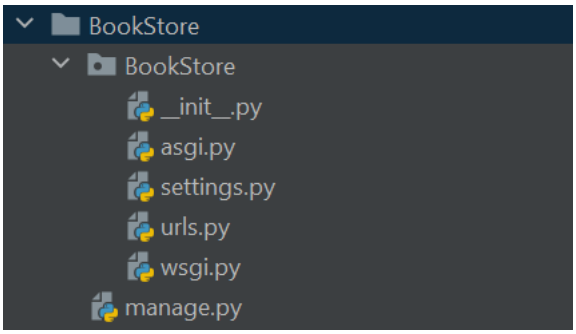
URLconf : La configuration des URL (URLconf) relie les URLs aux vues correspondantes. On spécifie les schémas d'URL dans un fichier appelé ***urls.py*** pour indiquer quelle vue doit être appelée lorsqu'une URL spécifique est requêtée.

Question 2 :

Une fois que Django est installé, nous pouvons créer un nouveau projet Django en utilisant la commande suivante :

```
PS C:\Users\HP\Desktop> django-admin startproject BookStore
```

Cette commande crée un répertoire avec le nom du projet et une structure de fichiers initiale pour le projet Django.



On accède au répertoire racine du projet en utilisant la commande ***cd BookStore.***

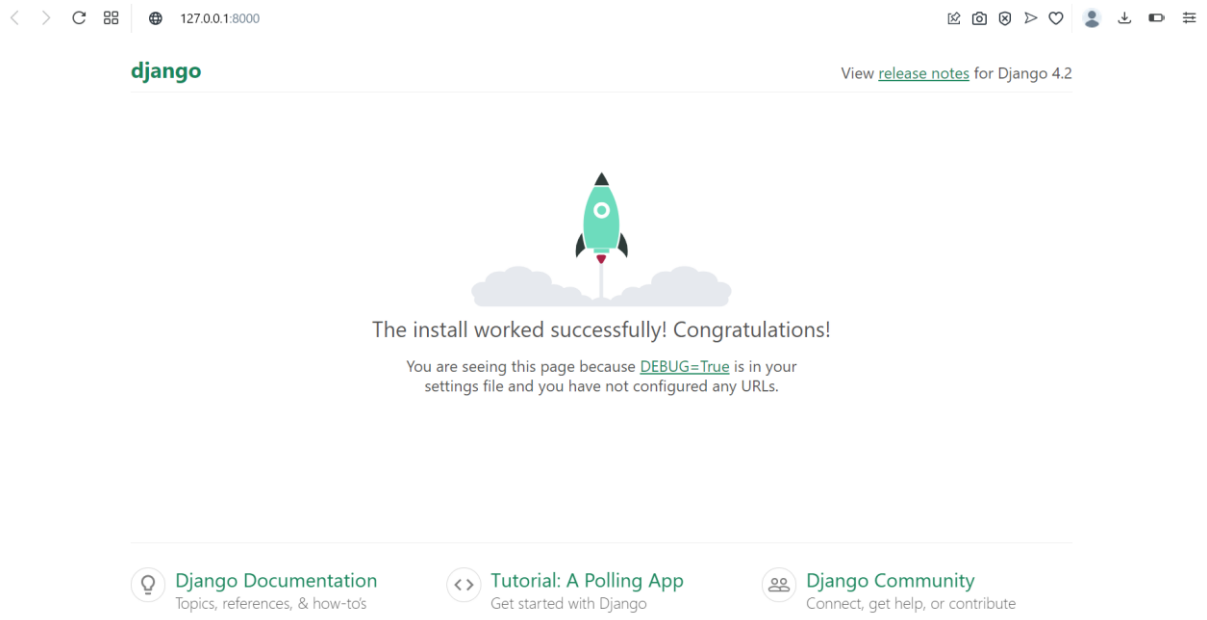
À l'intérieur du répertoire du projet, On trouvera plusieurs fichiers et répertoires. Les fichiers et répertoires importants pour l'instant sont :

- manage.py : Un script de gestion du projet Django. On l'utilisera pour effectuer diverses tâches de gestion de projet, telles que le lancement du serveur de développement Django.
- Le répertoire avec le nom du projet : Ce répertoire contient les fichiers de configuration principaux du projet Django.

Pour vérifier que notre projet est correctement configuré, On peut lancer le serveur de développement Django en utilisant la commande suivante :

```
PS C:\Users\HP\Desktop\BookStore> python manage.py runserver
```

Qui utilise par défaut le port 8000.



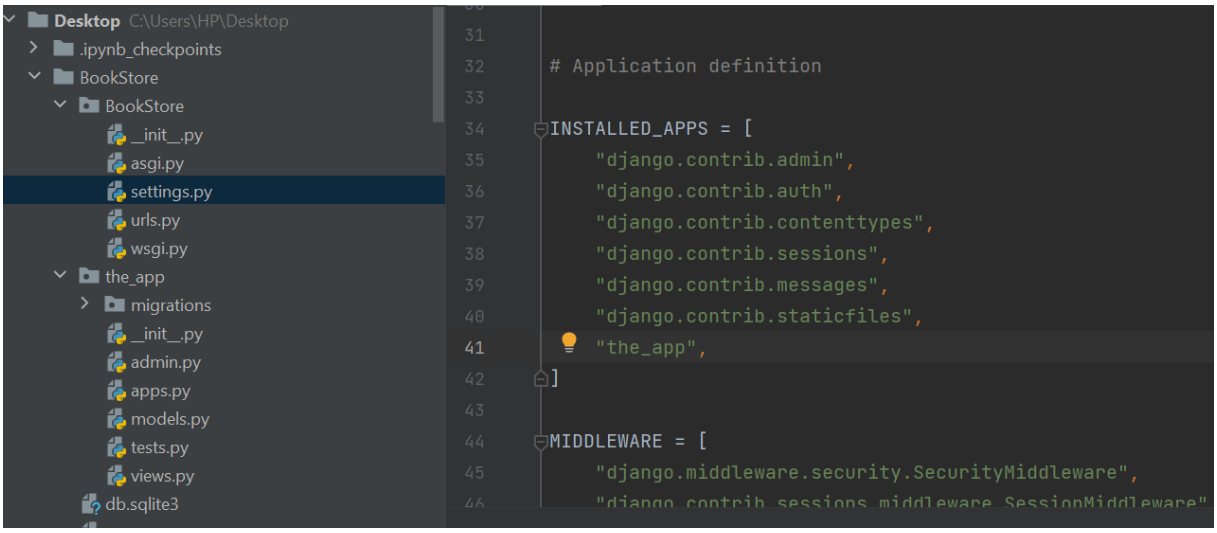
Question 3 :

Les applications Django sont des composants réutilisables qui effectuent des fonctionnalités spécifiques au sein d'un projet Django. Elles sont conçues pour être indépendantes et modulaires, ce qui facilite leur intégration dans différents projets.

Pour créer une application, on utilise la commande suivante :

```
PS C:\Users\HP\Desktop\BookStore> python manage.py startapp the_app
```

On procède par la suite à l'ajout de l'application dans la liste des applications déjà installées :



Question 4 :

On ouvre le fichier models.py dans le répertoire de notre application créée précédemment.

Par la suite on importe la classe *models* depuis le module django.db.

On Définit la classe *Book* en tant que sous-classe de *models.Model*. Cette classe représente le modèle de données pour les livres.

```python
from django.db import models


class Book(models.Model):
    title = models.CharField(max_length=50)
    author = models.CharField(max_length=50)
    publication_date = models.DateField()
    editor = models.CharField(max_length=50)
    language = models.CharField(max_length=20)
    price = models.DecimalField(max_digits=8, decimal_places=2)

```

Après avoir défini le modèle Book, on procède à la migration des changements de modèle dans la base de données.

```
PS C:\Users\HP\Desktop\BookStore> python manage.py makemigrations
Migrations for 'the_app':
  the_app\migrations\0001_initial.py
    - Create model Book
PS C:\Users\HP\Desktop\BookStore> python manage.py migrate
Operations to perform:
```

Question 5 :

1- Pour ouvrir la console interactive de Django, on exécute la commande :

```
PS C:\Users\HP\Desktop\BookStore> python manage.py shell
```

2- À l'intérieur de la console, on importe le modèle *Book* en utilisant la commande de la ligne [1] ci-dessous, et par la suite on instancie le modèle Book (ligne [2]) et on le sauve dans la base de données (ligne [3])

```
In [1]: from the_app.models import Book

In [2]: book1 = Book(title="La Cité des Livres", author="Lucie M. Montgomery", publication_date="2010-09-15", language="French", price=12.99)

In [3]: book1.save()
```

Ainsi, on crée de nouvelles instances Book :

```
In [4]: book2 = Book(title="Hijo de la luna", author="Carlos Ruiz Zafón", publication_date="2001-04-01",language="Spanish", price=14.50)

In [5]: book2.save()
```

```
In [6]: book3 = Book(title="Harry Potter", author="J.K. Rowling", publication_date="1997-06-26",language="English", price=14.99)

In [7]: book3.save()
```

Pour vérifier que les instances ont été créées avec succès, on peut exécuter la commande suivante pour récupérer tous les livres de la base de données :

```
In [8]: books = Book.objects.all()

In [9]: print(books)
<QuerySet [<Book: Book object (1)>, <Book: Book object (2)>, <Book: Book object (3)>]>
```

Question 6 :

On doit créer une vue dans Django qui récupère les livres depuis la base de données et les passe à une template pour affichage.

- On ouvre le fichier *views.py* dans le répertoire de l'application the_app.
- On importe le modèle Book.
- On crée une fonction pour la vue et qui sera responsable de récupérer les livres depuis la base de données et de les passer à la template.

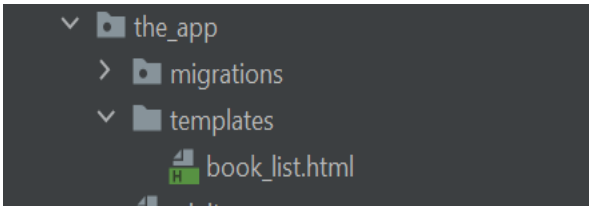
```python
from .models import Book
from django.shortcuts import render


def book_list(request):
    books = Book.objects.all()
    return render(request, 'book_list.html', {'books': books})
```

Dans cette fonction, nous utilisons Book.objects.all() pour récupérer tous les livres de la base de données et les stockons dans la variable books. Ensuite, nous utilisons la fonction render() pour rendre la template book_list.html en passant le contexte qui contient les livres.

Question 7 :

Tout d'abord, on crée le dossier 'templates' à la racine de notre application, ce dossier est utilisé pour stocker toutes les templates de cette application.

On crée un fichier HTML nommé *book_list.html* dans le répertoire 'templates' qui sera utilisé pour afficher la liste des livres, et on s'attaque à la définition de notre document html.



Dans notre fichier html on a utilisé une boucle ‘ {% for %}’ pour itérer sur la liste des livres passée depuis la vue. À l'intérieur de la boucle, on affiche les informations du livre, telles que le titre, l'auteur et la date de publication, en utilisant la syntaxe ‘ {{ book.field_name }}’.

```
<!DOCTYPE html>
<html>
<head>
    <title>Bibliothèque</title>
</head>
<body>
    <h1>Liste de nos livres</h1>
     <table>
        <thead>
            <tr>
                <th>Titre</th>
                <th>Auteur</th>
                <th>Date de publication</th>
                <th>Prix</th>
                <th>Langue</th>
            </tr>
        </thead>
        <tbody>
            {% for book in books %}
                <tr>
                    <td>{{ book.title }}</td>
                    <td>{{ book.author }}</td>
                    <td>{{ book.publication_date }}</td>
                    <td>{{ book.price }}</td>
                    <td>{{ book.language }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>
```

Question 8 :

On crée le fichier ‘urls.py’ dans la racine de notre application ‘the_app’ et c’est ici qu’on va mapper notre url à notre fonction dans vue.

On importe la fonction path depuis le module django.url et on s’assure aussi qu’on a importé notre vue correspondante.

Par la suite on définit une variable, cette entrée spécifiera le chemin d'URL que nous souhaitons associer à notre vue.

```
from django.urls import path
from . import views


urlpatterns = [path('books/', views.book_list, name='book_list')]
```

Et pour importer cette nouvelle configuration dans notre projet principal, On n'a qu'à ouvrir le fichier urls.py de notre projet BookStore et importer la fonction *include* également de django.urls et ajouter notre nouvel URL dans les urlpatterns.

Question 9 :

```
June 01, 2023 - 10:49:44
Django version 4.2.1, using settings 'BookStore.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[01/Jun/2023 10:50:53] "GET /the_app/books HTTP/1.1" 301 0
[01/Jun/2023 10:50:53] "GET /the_app/books/ HTTP/1.1" 200 1224
```

Question 10 :