**1. Introduction à Django**

* Qu'est-ce que Django ?
* Installation de Django
* Introduction à l'architecture MVT (Model –View- Template)

**2. Mise en place d’un projet**

* Création d'un projet Django
* Structure d'un projet Django
* Exécution d’un projet Django (serveur de développement)

**3. Premiers pas avec Django**

* Création et gestion d'une application
* Concepts de base des vues et URL en Django
* Création de vues simples
* Manipulation des routes et URLs dans Django
* Navigation entre les applications

**4. Modèles et ORM de Django**

**https://chatgpt.com/c/66ead27a-6214-8009-8da3-a98edb87cc4f**

* Qu'est-ce que l'ORM de Django ?
* Création de modèles Django
* Champs de modèles courants (CharField, TextField, DateField, etc.)
* Relations entre modèles (OneToOne, ForeignKey, ManyToMany)
* Migrations avec makemigrations et migrate
* Utilisation de l'API de l'ORM pour interagir avec la base de données (CRUD)

**5. Les formulaires Django**

* Introduction aux formulaires Django
* Création et gestion de formulaires
* Validation des données dans les formulaires
* Utilisation des formulaires pour créer et modifier des objets de la base de données
* Modèles de formulaire (ModelForm)

**6. Templates Django**

* Moteur de template Django : Syntaxe de base
* Utilisation de balises et filtres dans les templates
* Héritage et inclusion de templates
* Gestion des fichiers statiques et des fichiers médias (CSS, JavaScript, images)

**7. Authentification et gestion des utilisateurs**

* Système d’authentification de Django
* Création et gestion des utilisateurs
* Gestion des sessions et des cookies
* Implémentation du système de connexion/déconnexion
* Gestion des permissions et groupes
* Réinitialisation du mot de passe et gestion des e-mails

**8. Django Admin**

* Introduction à l'interface d'administration de Django
* Personnalisation du panneau d'administration
* Gestion des utilisateurs et des groupes via l'admin
* Ajout de modèles dans l'admin
* Gestion avancée des actions d'administration

**9. Tests et Debugging**

* Création et exécution de tests unitaires avec Django
* Utilisation de la console Django pour tester des modèles et des requêtes
* Gestion des erreurs et debugging
* Utilisation de l'outil de débogage intégré de Django

**10. Django Rest Framework (DRF)**

* Introduction à Django Rest Framework
* Création d'une API REST avec Django
* Serializers dans DRF
* Vues basées sur les classes (CBV) et vues fonctionnelles dans DRF
* Authentification et permissions dans DRF
* Gestion des requêtes GET, POST, PUT, DELETE avec DRF

**11. Asynchronisme et Django Channels**

* Introduction à Django Channels
* Utilisation de WebSockets avec Django Channels
* Cas d'utilisation : Chat en temps réel, notifications, etc.
* Intégration de Django Channels avec une base de données

**12. Déploiement d'une application Django**

* Préparation de l'application pour le déploiement
* Utilisation de Gunicorn, Nginx
* Hébergement sur des services cloud comme Heroku, AWS ou DigitalOcean
* Configuration de la base de données pour la production
* Sécurisation d'une application Django (HTTPS, gestion des secrets)

**13. Fonctionnalités avancées**

* Tâches en arrière-plan avec Celery
* Gestion des fichiers statiques et des fichiers médias avec AWS S3
* Internationalisation et localisation
* Utilisation de Django avec des bases de données NoSQL
* Optimisation des performances d'une application Django

### 1. Introduction à Django

#### Qu'est-ce que Django ?

**Django** est un framework web open-source en Python qui permet de développer des applications web rapidement et efficacement. Il est conçu pour encourager les bonnes pratiques de développement, telles que la réutilisation de code, la sécurité et la rapidité.

##### Exemple :

Django est utilisé dans des projets variés comme :

* **Instagram** : pour gérer des millions de photos partagées.
* **Pinterest** : pour organiser des images en collections.
* **Mozilla** : pour des applications liées à Firefox.

#### • Installation de Django

Pour commencer à utiliser Django, il faut d'abord installer Python, puis utiliser **pip** pour installer Django.

##### Étapes d'installation :

1. Installez Python (si nécessaire) en le téléchargeant depuis [python.org](https://www.python.org/).
2. Vérifiez l'installation de Python : **python –version**
3. Créer un environnement virtuel : **python -m venv env**
4. Activer l’environnement virtuel : **env\Scripts\activate**
5. Installez Django avec pip : **pip install django**
6. Vérifiez l'installation de Django : **django-admin –version**

#### Introduction à l'architecture MVT (Model –View- Template)

Django suit l'architecture **MVT** :

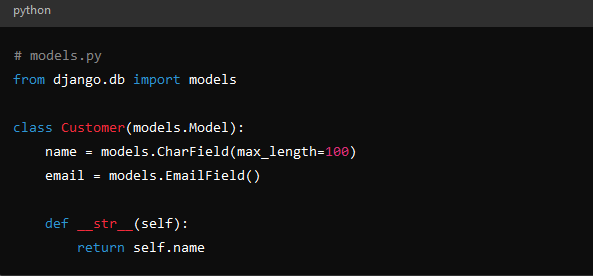
 **Modèle (Model)** : C'est la structure des données. Il définit la base de données via des classes Python.

 **Vue (View)** : C'est la logique qui relie le modèle et le template. Elle traite les requêtes et retourne les réponses.

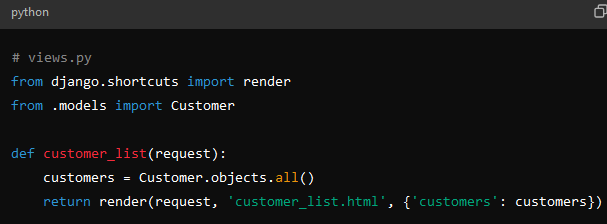
 **Template** : C'est l'interface utilisateur. Les templates sont des fichiers HTML qui affichent les données.

#### Exemple simple :

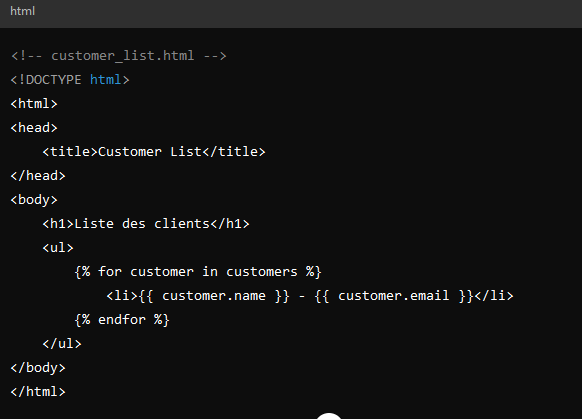
1. **Modèle :** Définition d'un modèle Customer dans models.py.



**Vue :** Création d'une vue qui affiche tous les clients dans views.py.



**Template :** Affichage des données dans un template customer\_list.html.



### Mise en place d’un projet Django

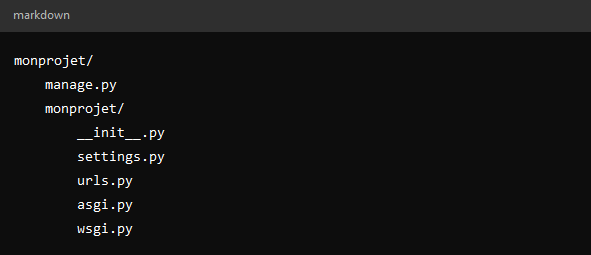
#### Créer un projet Django :

* Exécutez la commande suivante pour créer un nouveau projet :

**django-admin startproject monprojet :** Cela créera un répertoire monprojet contenant la configuration de base du projet.

#### • Structure d'un projet Django

La commande précédente génère une structure comme celle-ci :

****

 **manage.py** : Un script pour gérer le projet (lancer le serveur, faire des migrations, etc.).

 **settings.py** : Le fichier de configuration contenant les paramètres du projet.

 **urls.py** : Gère le routage des URLs vers les vues correspondantes.

###  **wsgi.py/asgi.py** : Fichiers d'entrée pour déployer le projet sur des serveurs web (WSGI/ASGI).

#### • Exécution d’un projet Django (serveur de développement)

1. **Lancer le serveur de développement :**
   * Pour démarrer le serveur de développement, accédez au répertoire du projet (là où se trouve manage.py) et tapez : **python manage.py runserver**
2. **Accéder au projet :**

* Par défaut, le serveur est accessible à l'adresse http://127.0.0.1:8000/. Ouvrez un navigateur et entrez cette URL pour voir la page d'accueil de Django.

### 3. Premiers pas avec Django

#### • Création et gestion d'une application

1. **Créer une application Django :**
   * Dans le répertoire principal du projet (où se trouve manage.py), exécutez :

**python manage.py startapp monapplication :** Cela crée un répertoire monapplication avec une structure de base pour une application Django.

1. **Ajouter l'application au projet :**

* Ouvrez settings.py dans le répertoire du projet (monprojet/monprojet/).
* Ajoutez monapplication à la liste INSTALLED\_APPS :



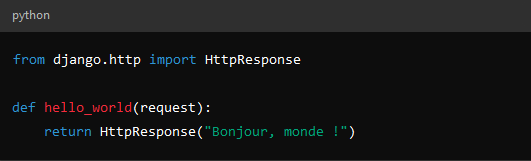
#### • Concepts de base des vues et URL en Django

* **Vues (Views) :** Les vues traitent les requêtes et renvoient des réponses. Elles sont définies dans le fichier views.py de l'application.
* **URLs :** Les URL définissent comment les requêtes HTTP sont acheminées vers les vues. Elles sont configurées dans urls.py.

#### • Création de vues simples

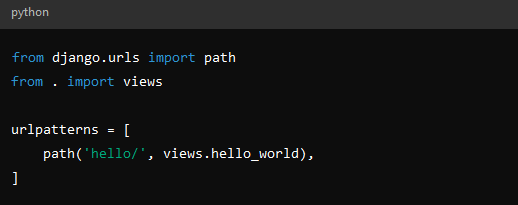
**Définir une vue :**

* Ouvrez views.py dans votre application (monapplication/views.py).
* Créez une vue simple qui retourne une réponse HTTP :



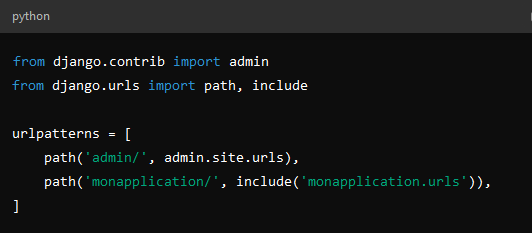
**Configurer l'URL pour la vue :**

* Créez ou ouvrez le fichier urls.py dans le répertoire de l'application (monapplication/urls.py). Si le fichier n'existe pas, créez-le.
* Configurez les routes pour vos vues :



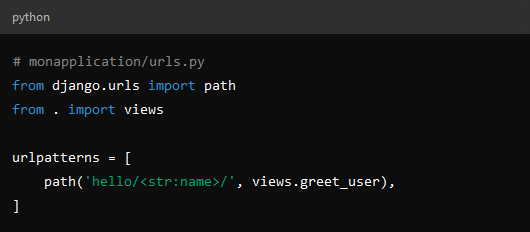
**Inclure les URLs de l'application dans le projet :**

* Ouvrez urls.py dans le répertoire du projet (monprojet/monprojet/urls.py).
* Ajoutez une ligne pour inclure les URLs de l'application :

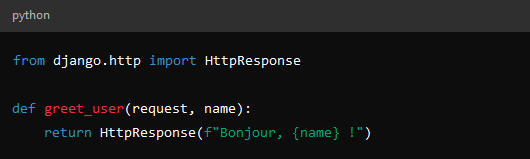


#### • Manipulation des routes et URLs dans Django

1. **Utiliser des paramètres dans les URLs :**
   * Vous pouvez ajouter des paramètres dans les URLs pour les passer aux vues :

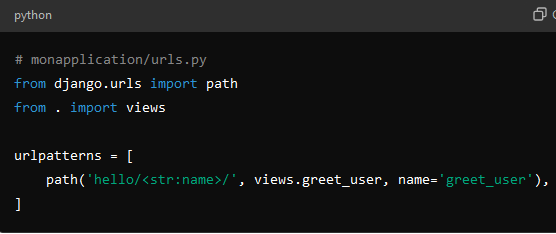


* Dans views.py, créez une vue pour gérer les paramètres :

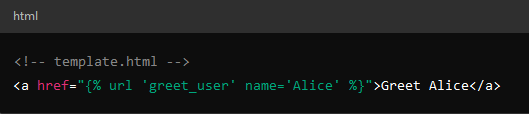


**Utiliser des URL nommées :**

* Vous pouvez donner un nom aux URLs pour une utilisation facile dans les templates :



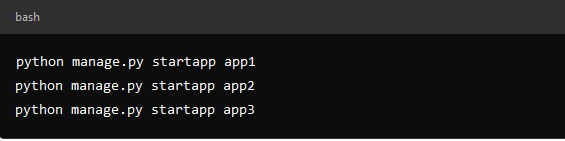
* Dans les templates, vous pouvez utiliser le nom de l'URL pour générer des liens :



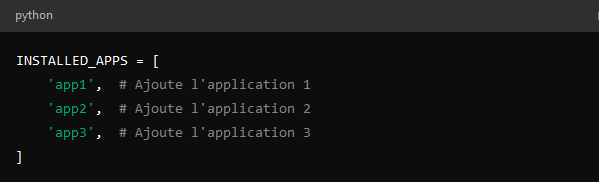
#### Navigation entre les applications

**Ajoutons 3 app1, app2 et app3 dans le projet**

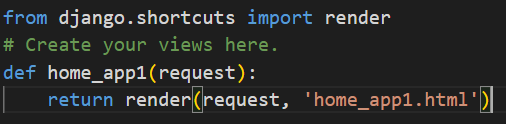
* 1. **Création de trois applications :**



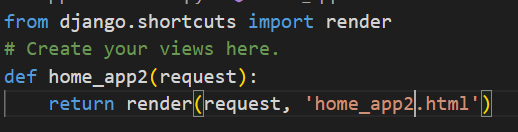
* 1. **Ajout des applications dans settings.py :** Dans le fichier monprojet/settings.py, ajoute chaque application dans la section INSTALLED\_APPS :



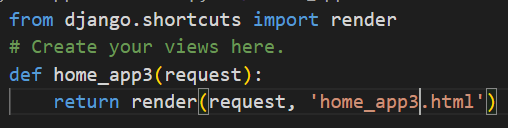
* 1. **Création des vues simples pour chaque application :** Dans chaque application, nous allons créer des vues qui renverront des pages HTML statiques.
* **app1/views.py**



* **app2/views.py**

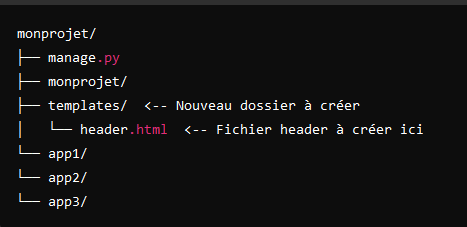


* **app3/views.py**



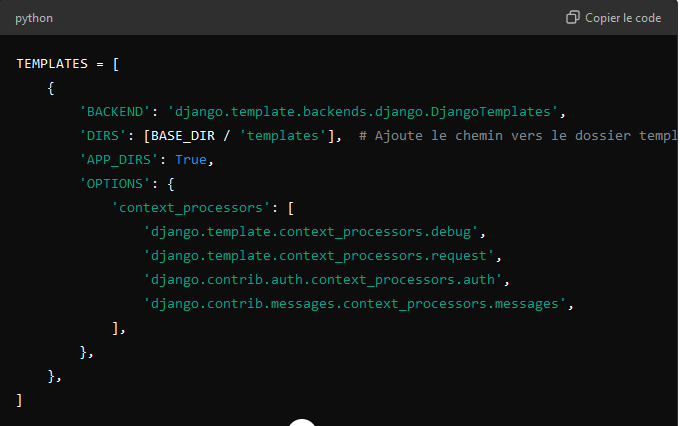
**Création des templates HTML :** Pour chaque vue, on va créer un fichier HTML simple.

On va créer un fichier header.html dans le dossier templates a la racine du projet qui sera partagé par toutes les pages via le mécanisme d'inclusion de Django.



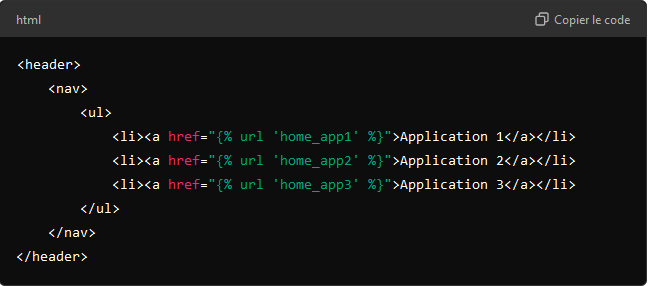
**Configurer Django pour reconnaître le dossier templates :**

Dans le fichier monprojet/settings.py, assurez-vous que Django sait où chercher les templates globaux. Cela se fait en ajoutant le chemin vers le dossier templates dans la configuration TEMPLATES :



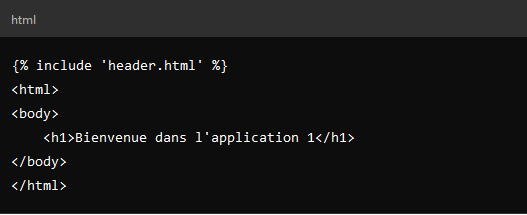
* **Création du template header.html :**

Crée un nouveau fichier templates/header.html qui contiendra les liens de navigation :

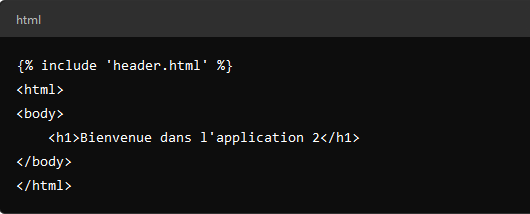


**Création des templates HTML avec inclusion du header :** Chaque page HTML inclura désormais ce header.html pour afficher les liens de navigation dynamiques.

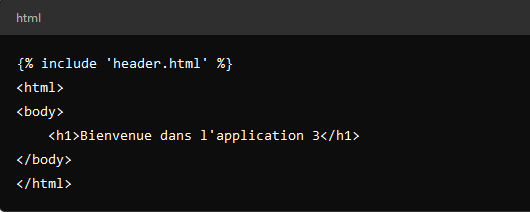
* **app1/templates/home\_app1.html**



* **app2/templates/home\_app2.html**

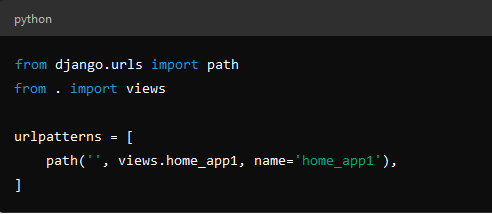


* **app3/templates/home\_app3.html**

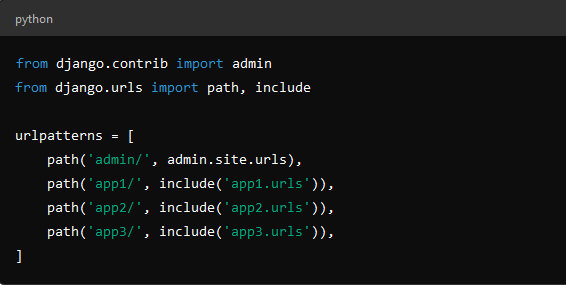


**Définition des URLs pour chaque application :** Ajoutons les routes pour chaque application.

* **app1/urls.py**



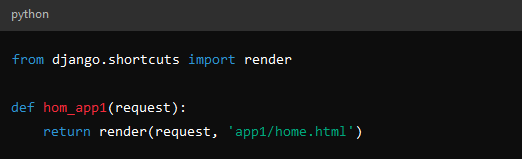
**Ajouter les URLs principales dans monprojet/urls.py :** Enfin, on relie chaque application à l'URL principale.



**NB:** Pour créer des vues distinctes pour plusieurs applications (app1, app2, app3) dans un projet Django, en s'assurant que chaque vue référence le bon template, même si les noms des templates sont identiques il faudra respecter cette structure :

****

Et faire ceci pour toutes les vue :



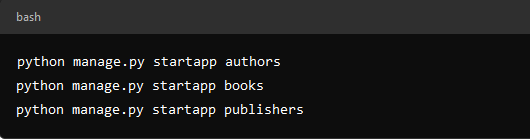
**4. Modèles et ORM de Django**

**1- Qu'est-ce que l'ORM de Django ?**

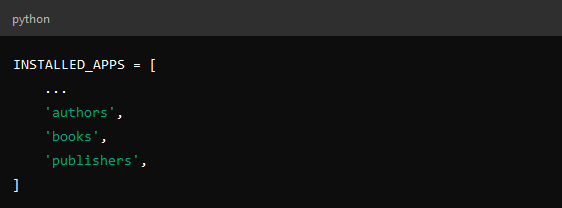
L'ORM (Object-Relational Mapping) de Django est un système qui permet de manipuler les bases de données en utilisant des objets Python. Au lieu d'écrire des requêtes SQL directement, tu définis des modèles Python qui correspondent aux tables de la base de données, et Django s'occupe de la traduction entre ces objets et les tables SQL.

Creeon un nouveau projet simple dans lequel nous allons appliquer les relations **OneToOneField**, **ForeignKey**, et **ManyToManyField**, ainsi que réaliser des opérations CRUD (sans utiliser de formulaires pour le moment):

* + 1. django-admin startproject library\_project
    2. **Créer trois applications Author, Book, et Publisher** :

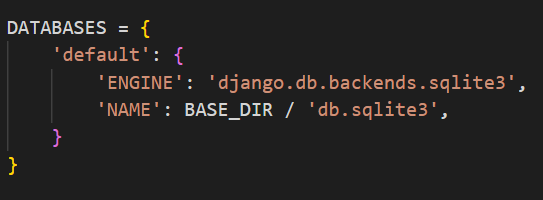


* + 1. **Ajouter les applications à INSTALLED\_APPS** dans library\_project/settings.py :

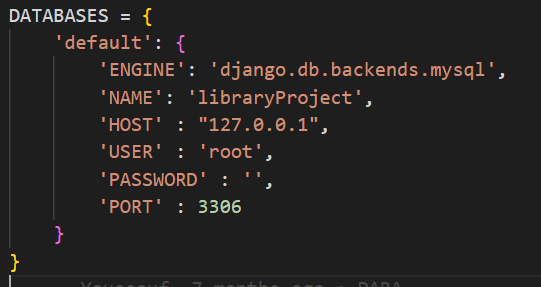


* + 1. Configuration de la base de donnée dans le settings

Par defaut c’est sqlit qui est utiliser et moi j’utilise pas ce truc dans mes TP



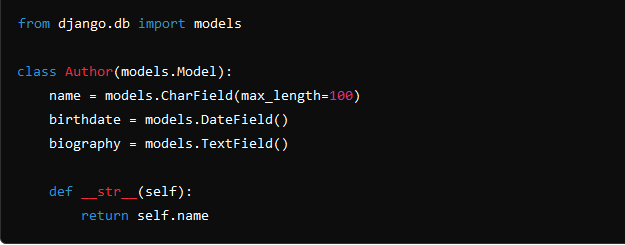
Nous allons utiliser mysql a la palce en installant ‘**pip install mysqlclient’**



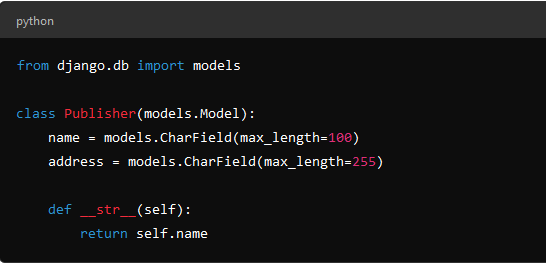
Avant de quitter le settings nous allons profiter pour configuer le chemin des finchier globaux(voir la section precedente)

* + 1. Définir les Modèles

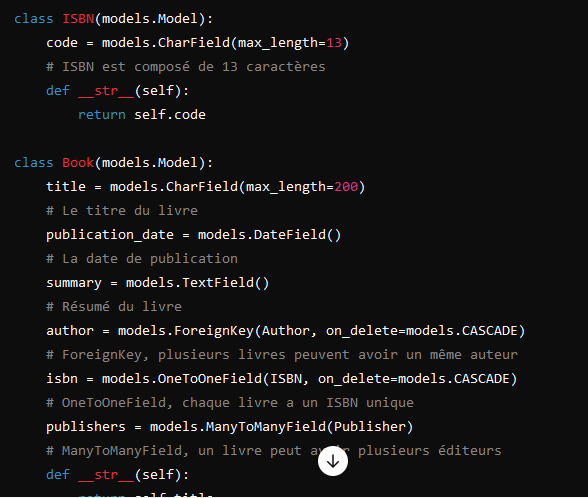
**Modèle Author** (authors/models.py) :



**Modèle Publisher** (publishers/models.py) :



**Modèle Book** (books/models.py) :

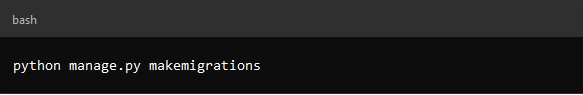


* + 1. Créer et Appliquer les Migrations dans Django

Django utilise un système de migrations pour synchroniser les modèles (définis dans le code) avec la base de données. Lorsque tu modifies les modèles (comme ceux pour Book, Author, Publisher, ou ISBN), tu dois générer et appliquer des **migrations** pour que ces modifications soient reflétées dans la base de données.

Après avoir défini ou modifié les modèles dans ton projet Django, la première étape est de générer des migrations. Les migrations sont des fichiers qui enregistrent les changements apportés à la structure de la base de données en fonction des modifications dans les modèles.

#### Commande :



* + 1. **Appliquer les Migrations** avec migrate

Une fois les migrations créées, la commande suivante permet d'appliquer ces changements à la base de données.

#### Commande :



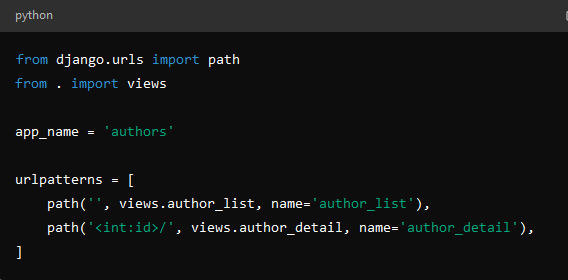
 Cette commande exécute les fichiers de migration générés et applique les changements à la base de données.

 Elle va créer les tables SQL correspondantes dans la base de données (si elles n'existent pas encore) ou les modifier selon les changements dans tes modèles.

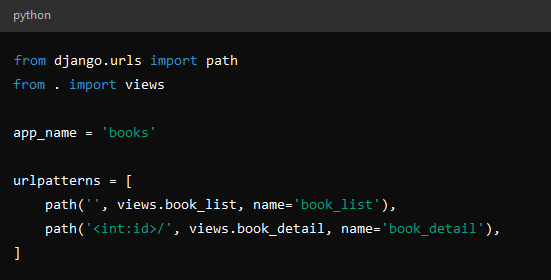
g- **Configurer les URLs**

Pour chaque application, nous allons définir des vues pour lister, afficher les détails, et gérer le CRUD (Create, Read, Update, Delete).

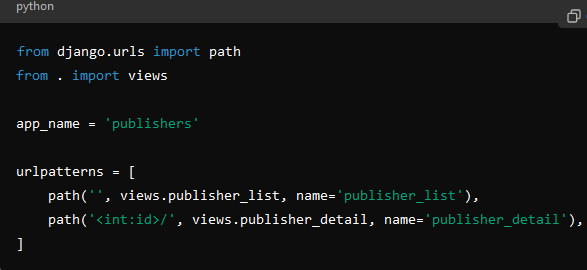
**URLs pour l'application authors** (authors/urls.py) :



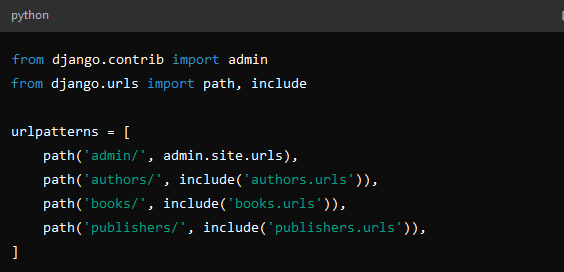
**URLs pour l'application books** (books/urls.py) :



**URLs pour l'application publishers** (publishers/urls.py) :

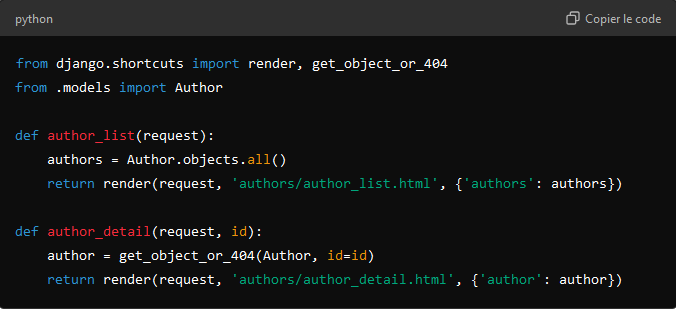


**Configurer les URLs du projet** (library\_project/urls.py) :

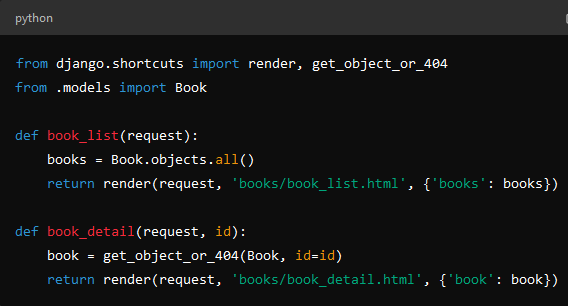


**h- Les vues pour les applications**

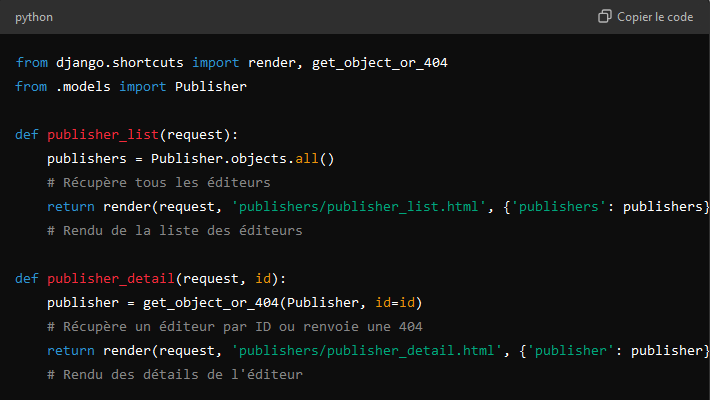
**Vues pour l'application authors** (authors/views.py) :

****

**Vues pour l'application books** (books/views.py) :



**Vues pour l'application publishers** (publishers/views.py) :

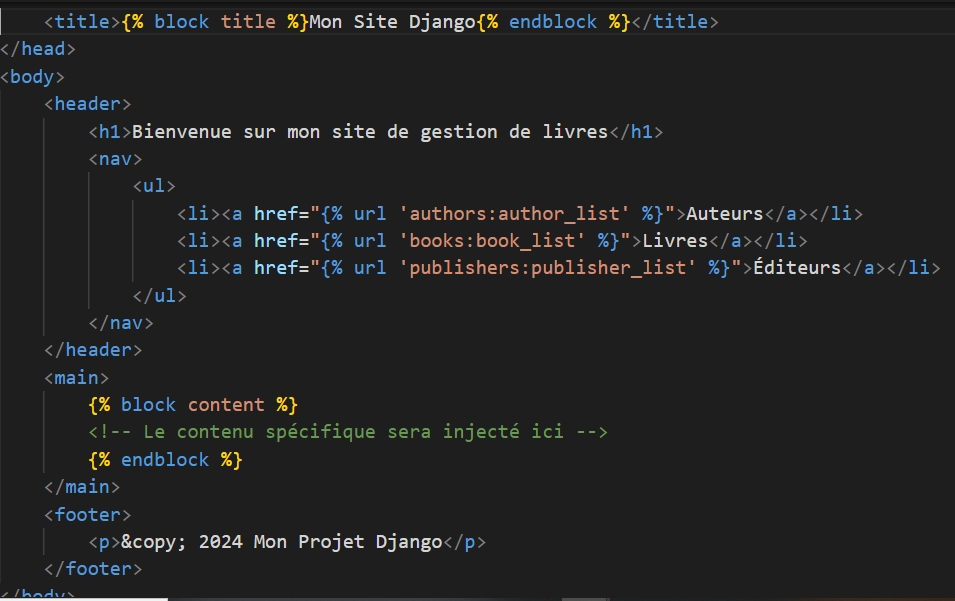


Cette fois ci nous allons changer de technique histoire de grandir un peu +

Au lieu de faire comme avant avec les inclusions , nous allons faire heriter les pages, genre créer une page de base que toutes les autres pages vont heriter pour afficher leurs contenue on parle souvent de block.

Il faut créer un dossier qui s’appelle exactement ‘**templates’**  à la racine du projet comme précédemment avec les app (1, 2 et 3) qu’on a eu à faire (voir en haut), dans ce dossier nous allons créer toutes les pages globales.

Exemple de page de base :

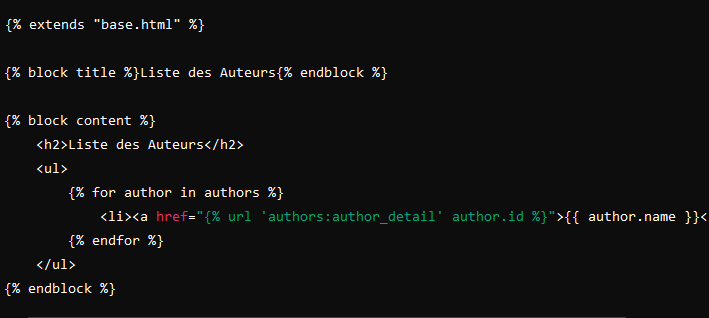
 Vous remarquez qu’il y a une balise étranges ‘**block’** et ‘**endblock’**  avec un attribut ‘**content’** cette balise n’appartient pas au HTML mais à Django, son importance est d’afficher tout le contenu d’une quelconque page qui l’héritera dans le future, et tout le contenu de la page héritant est injecter dans le ‘**content’** puis Django vas se débrouiller pour nous montrer le contenu.

L’intérêt est qu’on ne va plus se balader avec le include par tout dans l’application mais on va juste faire hériter les pages qui ont besoin de la base et 80% des projets fullStack Django professionnel son conçu presque de la sorte.

Voyons comment faire ce fameux héritage avec les différentes pages qu’on a besoin

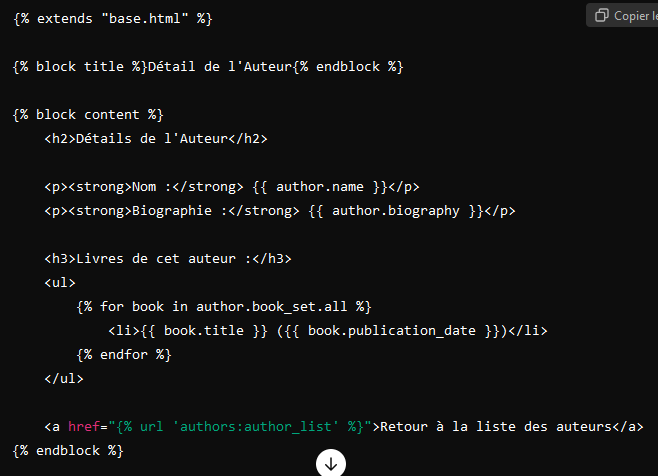
**Template de Liste (author\_list.html) :**

Le fichier author\_list.html étend le fichier base.html pour utiliser la structure de base du site :



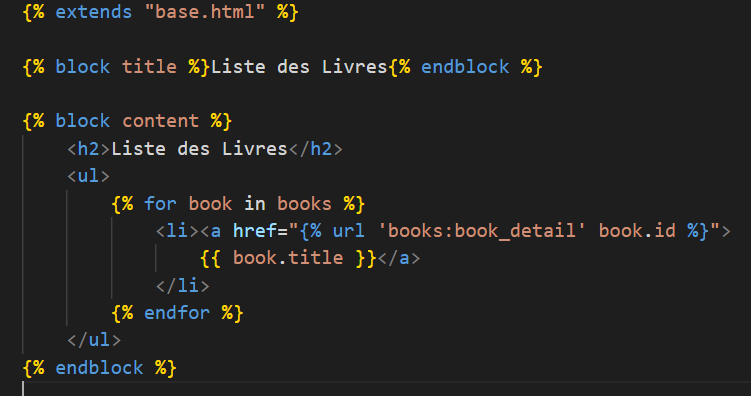
Cette page affiche tous les auteurs mais qui seront pousser (binder) dans la page de base autrement dit tout le contenu sera visible dans base.html.

**Template de detail (author\_detail.html) :**

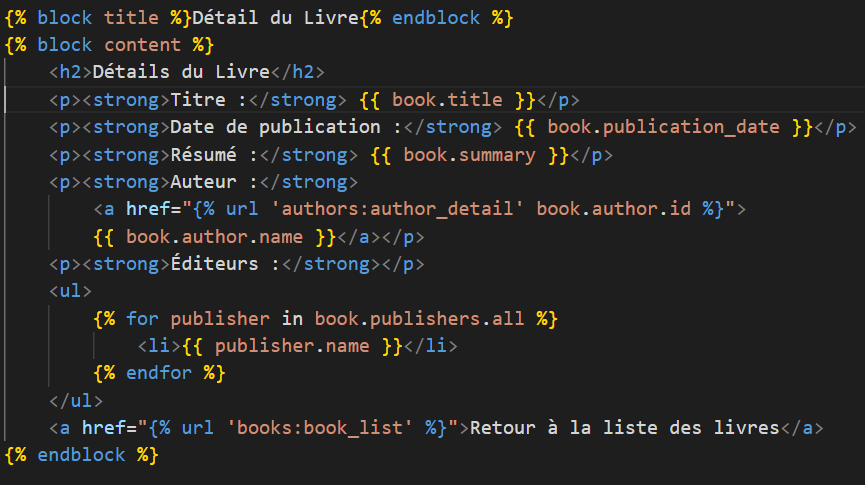
****

La méthode author.book\_set.all est obtenue automatiquement par Django grâce à la relation **ForeignKey** définie dans le modèle Book.

**Template de liste (**book\_list**.html) :**

****

Template book\_detail.html :



 Pour accéder aux objets liés par un ManyToManyField, utilise simplement le nom du champ (comme book.publishers.all()).

 Django gère automatiquement cela sans nécessiter le suffixe set, car chaque côté de la relation peut contenir plusieurs objets.

Template publisher\_list.html :



Template publisher\_detail.html :



Utiliser un module Django ‘faker’ pour generer des fausses donnée dans la base de donnée

Installer faker : **pip install faker**

Puis ecrire un srcipt avec fake pour remplire les modeles

Lien github pour telecharger le fichier :

Executer le fichier : python populate.py , si le fichier s’appel populate.py