

Installation Django

Ouvrez la ligne de commande en mode administrateur pour exécuter les commandes qui suivent.

1. Assurez-vous que `pip` est installé sur votre système en exécutant la commande suivante dans votre terminal:

```
pip --version
```

pip est un gestionnaire de paquets pour Python, utilisé pour installer et gérer des bibliothèques et des modules Python supplémentaires qui ne font pas partie de la bibliothèque standard de Python. On trouve `npm` (pour Node.js) et `Composer` (pour PHP)

Mettre à jour les outils de travail avec python.

```
python -m pip install --upgrade pip
python -m pip install --upgrade setuptools
python -m pip install --upgrade pipenv
```

Si `pip` n'est pas installé, vous devez l'installer en suivant les instructions de la documentation officielle de Python.

```
python -m ensurepip --default-pip
```

Si vous n'arrivez pas à exécuter la commande `python -m ensurepip --default-pip`, vous pouvez télécharger manuellement le programme d'installation de PIP à partir du site officiel de Python (<https://pypi.org/project/pip/#files>) et l'installer manuellement.

Pour des raisons de compatibilité installer une version précise LTS **Django 4.2 LTS**:

```
pip install django~=4.2.0
```

Vous pouvez aussi lancer l'installation de Django en exécutant la commande suivante pour installer la version la plus adapté à votre environnement :

```
pip install django
```

La commande standard pour désinstaller Django, si vous l'avez installé avec `pip` (le gestionnaire de paquets Python le plus courant), est :

```
pip uninstall django
```

Pour vérifier la version de `django` installé on exécute :

```
django-admin version
```

Nous allons utiliser un outil d'administration de Django pour créer un nouveau projet Django nommé `projet1`. **Le point à la fin indique** qu'on va créer le projet dans le répertoire courant.

Ça veut dire que le système créera un projet Django nommé `projetG42` dans le même répertoire où vous exécutez la commande.

```
F:\projetG42>django-admin startproject projetG42 .
```

Dans cet exemple, il ne va pas créer le projet django dans un nouveau dossier `projetG42`, il va le créer à l'intérieur du dossier courant `projetG42`.

```
django-admin startproject projet1 .
```

Cette commande va créer un mini site qu'on va voir tout de suite.

Après on ouvre le projet avec vscode :

```
code .
```

Il y a le répertoire `PROJET1` et un sous répertoire `projet1`, le premier est le contenant de notre projet, par contre le deuxième est un package qui contient l'ensemble des fichiers qui initialise notre projet.

Voilà un résumé de toutes les étapes précédentes :

Si on veut l'installer de façon globale dans notre système :

```
F:\python2025\g0>pip install django
Requirement already satisfied: django in c:\python313\lib\site-packages (5.1.7)
Requirement already satisfied: asgiref<4,>=3.8.1 in c:\python313\lib\site-packages (from django) (3.8.1)
Requirement already satisfied: sqlparse>=0.3.1 in c:\python313\lib\site-packages (from django) (0.5.3)
Requirement already satisfied: tzdata in c:\python313\lib\site-packages (from django) (2025.2)

F:\python2025\g0>django-admin version
5.1.7

F:\python2025\g0>mkdir projet1

F:\python2025\g0>cd projet1

F:\python2025\g0\projet1>django-admin startproject projet1 .

F:\python2025\g0\projet1>code .
```

Démarrer votre serveur en exécutant la commande suivante :

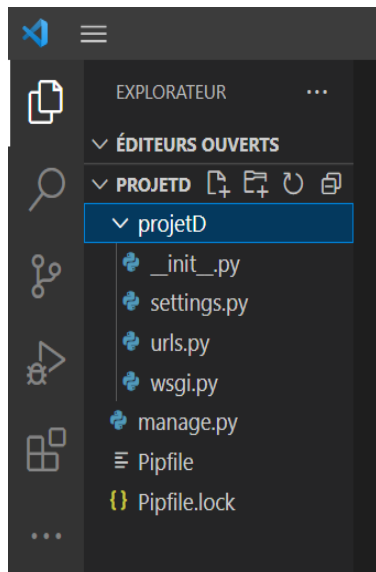
```
python manage.py runserver
```

Le serveur par défaut démarre `python manage.py runserver 8000`, si on a une autre application qui utilise ce port, on peut le démarrer avec un port spécifique.

```
python manage.py runserver 9090
```

```
You have 15 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 26, 2023 - 20:03:06
Django version 2.1, using settings 'projetD.settings'
Starting development server at http://127.0.0.1:8000/
```

Cliquez sur l'adresse pour accéder à l'application.



Le fichier `__init__.py` est un fichier spécial dans Python qui indique que le répertoire dans lequel il se trouve est un **package**.

Dans le contexte de Django, le fichier `__init__.py` est utilisé pour marquer les répertoires contenant des modules Python comme des packages Django, ce qui permet à Django de les importer correctement.

En outre, le fichier `__init__.py` peut également contenir du code Python qui est exécuté lors de l'importation du package. Cela peut être utile pour effectuer des tâches d'initialisation spécifiques à l'application, telles que la configuration de variables d'environnement ou l'enregistrement de modèles Django dans l'application.

Le fichier `manage.py` est un fichier important dans un projet Django. Il s'agit d'un script Python qui permet de gérer diverses opérations de votre projet Django, telles que la création de tables de base de données, la gestion de migrations, le lancement du serveur de développement et bien plus encore.

Le fichier `settings.py` est un fichier qui contient des paramètres qui contrôlent le comportement de votre application Django, tels que les paramètres de base de données, les réglages de sécurité, les réglages de cache et bien plus encore.

Le fichier `urls.py` est utilisé pour router les demandes d'URL de votre application vers les vues qui doivent les traiter. Il définit les correspondances entre les URLs et les vues de votre application.

Le fichier `wsgi.py` ("Web Server Gateway Interface") est un fichier d'interface entre le serveur web et votre application Django. Il est utilisé pour déployer votre application Django sur un serveur web compatible WSGI en chargeant l'application Django dans la mémoire.

Nous allons créer notre première application (app).

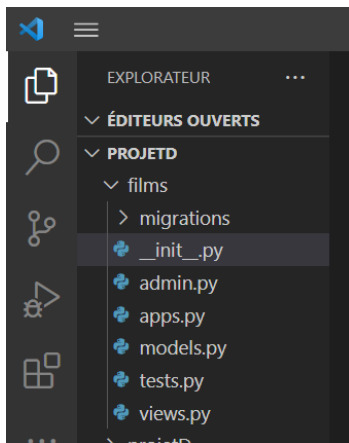
Dans le cadre de Django, une application (app) est une partie autonome d'un projet Django qui fournit une fonctionnalité spécifique.

Une application Django peut inclure des modèles de base de données, des vues, des URL, des templates, des fichiers statiques, etc. Les applications sont conçues pour être réutilisables et peuvent être intégrées à plusieurs projets Django. Par exemple un Blog pourra être développer et réutiliser dans d'autre projets.

Taper CTRL + C pour arrêter le projet.

Pour créer une app, taper la commande suivante :

```
python manage.py startapp films
```



Le fichier `admin.py` permet de personnaliser l'apparence et le comportement de l'interface d'administration de Django pour les modèles de données spécifiques à l'application, dans ce cas « films ». On peut y spécifier des options pour la liste des modèles dans l'interface d'administration, des champs à afficher, des filtres, des actions, etc.

Le fichier `apps.py` définit une classe d'application qui hérite de la classe `django.apps.AppConfig`. Cette classe permet de configurer divers aspects de l'application, tels que son nom, sa description, son label, son chemin de module, etc. Ils auraient dû la nommer `config.py`

Le fichier `models.py` contient des classes qui représentent les tables de la base de données de l'application. Chaque classe de modèle représente une table dans la base de données, et chaque attribut de la classe représente une colonne de la table.

Dans Django, le fichier `tests.py` est un module Python qui fait partie de l'application Django et qui permet de définir les tests unitaires pour l'application.

Plus précisément, le fichier `tests.py` contient des classes de tests qui héritent de la classe `django.test.TestCase`. Chaque classe de test contient des méthodes de test qui vérifient le comportement attendu de l'application pour une fonctionnalité donnée.

Le fichier `views.py` contient des fonctions ou des classes de vue qui gèrent les requêtes HTTP entrantes et renvoient une réponse HTTP appropriée. Une vue est une fonction ou une classe qui reçoit une requête HTTP en tant que paramètre et renvoie une réponse HTTP. Les vues peuvent effectuer des opérations telles que la récupération ou la mise à jour des données de la base de données, la validation des entrées utilisateur et la génération de réponses HTTP dynamiques.

Il faut savoir que notre projet principal n'est pas au courant de l'application que nous avons créé.

La première étape est d'enregistrer notre app films dans django.

Il faut aller dans `settings.py` de notre projet et l'ajouter.

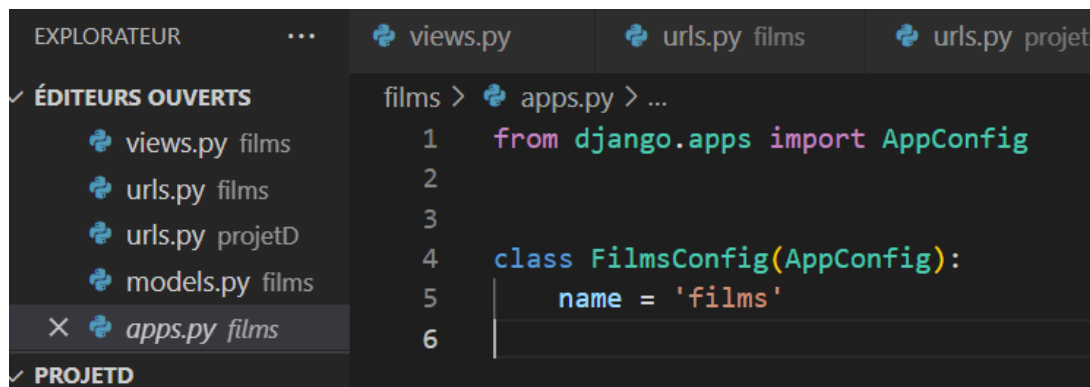
Par défaut on trouve plusieurs app installées par défaut :

```
INSTALLED_APPS = [
'django.contrib.admin', # administration panel
'django.contrib.auth', # authentification des users
'django.contrib.contenttypes', # permet de créer une relation entre les modèles
'django.contrib.sessions', # stocker temporairement des données des users
'django.contrib.messages', # affiche des messages tel que un film a bien été ajouté
'django.contrib.staticfiles', # pour gérer les fichiers statique tel que les CSS
]
```

Dans la plupart des projets ont besoin de ces app, mais si on n'en a pas besoin on peut les supprimer.

Pour notre cas, on doit enregistrer notre app ici pour que django assure une veille sur nos models et détecte s'il a des changements.

Si on vérifié notre répertoire films, on trouvera un fichier `apps.py` qui contient la classe `FilmsConfig`.



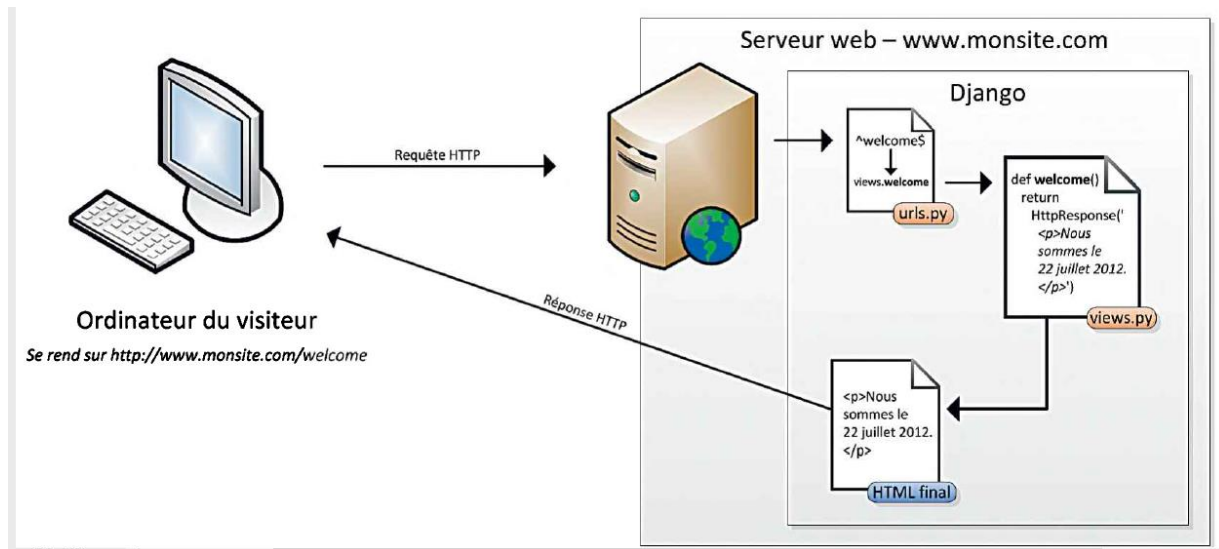
```
films > apps.py > ...
1  from django.apps import AppConfig
2
3
4  class FilmsConfig(AppConfig):
5      name = 'films'
6
```

Pour enregistrer notre app films, on doit donner le chemin complet de cette classe (`FilmsConfig`)

```
INSTALLED_APPS = [  
    "django.contrib.admin",  
    "django.contrib.auth",  
    "django.contrib.contenttypes",  
    "django.contrib.sessions",  
    "django.contrib.messages",  
    "django.contrib.staticfiles",  
    "films.apps.FilmsConfig",  
]
```

films est le package, apps est le nom du module , FilmsConfig et le nom de la classe.

Création d'une view



Nous allons créer la fonction `index` dans notre view. On la nomme ainsi car elle représente la page principale de notre app.

N'importe quel view doit avoir comme paramètre `request` ("`request`" est un paramètre qui est souvent utilisé dans les fonctions de vue pour représenter l'objet `HttpRequest` qui contient les informations sur la requête HTTP qui a été effectuée par le client.) et doit retourner une réponse http. Alors on doit importer `HttpResponse`.

```

1 from django.http import HttpResponse
2 from django.shortcuts import render
3
4 # Create your views here.
5 def index(request):
6     return HttpResponse("bonjour tout le monde")
7

```

The screenshot shows the Visual Studio Code interface with the file explorer on the left showing the project structure (films, migrations, init.py). The main editor shows the `views.py` file with the following code:

Au lieu de `return HttpResponse`, on va utiliser plus tard la fonction `render` pour générer une réponse http.

Après avoir créé notre view on doit la mapper avec url. On doit créer un fichier, par convention on doit le nommer `urls.py` dans le répertoire de notre application `films`.

Dans ce fichier on doit créer une variable qui s'appelle `urlpatterns` de type liste qui contient les chemins de nos urls.

On doit utiliser la fonction `path` qui doit être importé au préalable.

`path` ('' = les deux apostrophe font référence au root de notre application, par exemple :

films/

films/1/details

Dans notre cas le root est films, car le nom de notre app est films

On fait comme ça pour permettre la réutilisation de notre application, au cas où quelqu'un veut télécharger notre application et changer le root et mettre au lieu de films collection_films

C'est ce qu'on appelle l'utilisation des urls relatives.

Dans la fonction path on doit mapper ces urls avec notre fonction view.

```
urlpatterns = [path("", views.index, name="index_films"),
```

Cette ligne indique que si l'utilisateur tape <http://127.0.0.1:8000/films/> dans l'url, on va exécuter la fonction index qui se trouve dans views.py

Pour exécuter la fonction index qui se trouve dans views.py, On doit importer views, où nous avons créé notre fonction index tout à l'heure. On utilise l'instruction

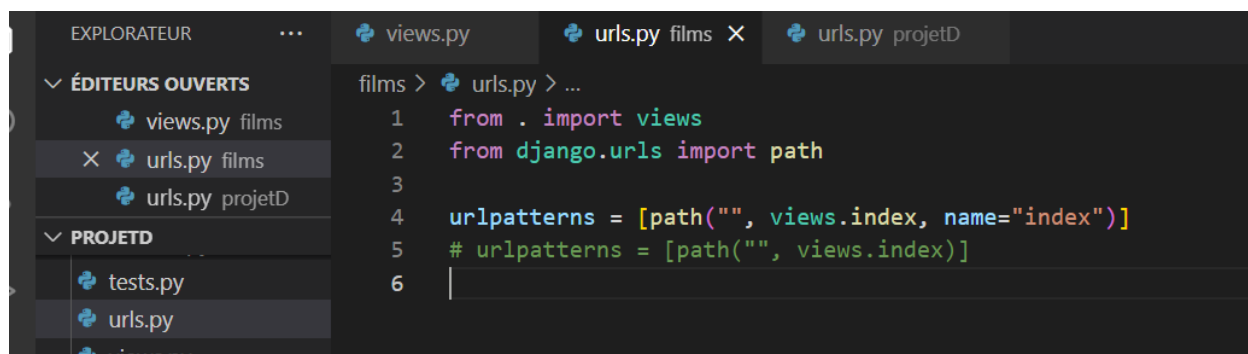
```
from . import views
```

pourquoi on a pas fait import views car :

En Python, l'instruction `from . import views` est utilisée pour importer le module "views" à partir du package ou répertoire courant. Au fond on importe `views.py` qui se trouve au même emplacement que notre fichier sur lequel on travaille qu'est `urls.py`

Tandis que l'instruction `import views` est utilisée pour importer un module indépendant appelé views.

`name="index"` est facultative, mais ça fait partie des bonnes pratiques



Mais jusqu'à maintenant notre projet principal 'projet1' n'est pas en courant de l'existence de notre application films.

On doit aller dans notre projet principal 'projet1' et ouvrir urls.py et on doit ajouter un nouvel objet path qui indique que pour n'importe quel chemin qui commence par films, doit nous diriger vers notre application films.

On doit aussi importer une autre fonction 'include' qui nous permettra d'importer les urls.


```

projectD > urls.py > ...
14 | 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15 | """
16 | from django.contrib import admin
17 | from django.urls import path, include
18 |
19 | urlpatterns = [
20 |     path('admin/', admin.site.urls),
21 |     path('films/', include('films.urls'))
22 | ]
23 |

```

```

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path("admin/", admin.site.urls),
    path("films/", include("films.urls"))
]

```