DÉVELOPPEMENT WEB CÔTÉ SERVEUR
FRAMEWORK PYTHON-DJANGO

TRAVAUX PRATIQUE N°2
(PROJET MYSITE : SUITE TP N°1)
CRÉATION D'UNE APPLICATION DJANGO

Objectifs du TP :

- Présenter le modèle MTV de Django
- Etudier la structure d'un projet Django
- Créer une application Django
- Etudier le paramètre INSTALLED_APPS dans le fichier de configuration settings.py
- Etudier la structure d'une application Django
- Créer une fonction view qui répond à une requête http
- Configurer les urls dans les fichiers urls.py du site et de l'application

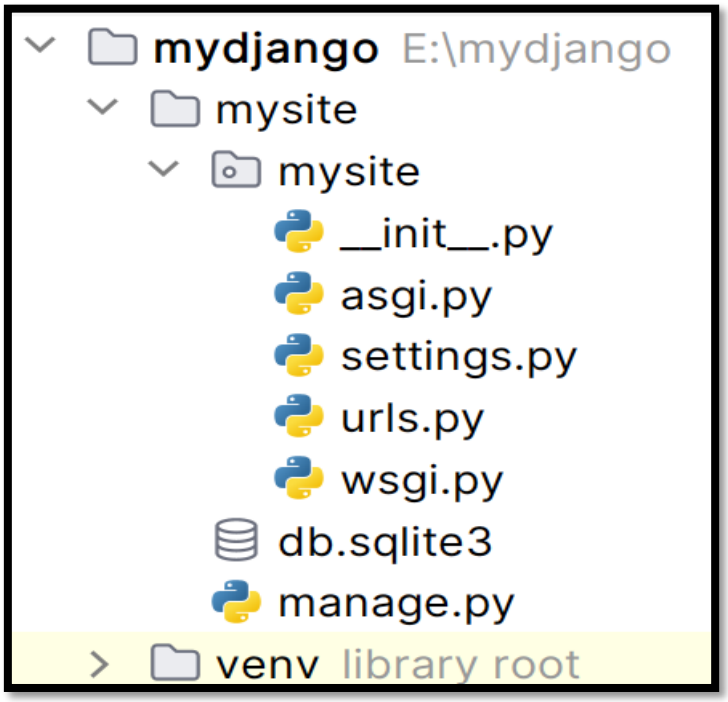
Modèle MTV de Django

Django utilise le modèle de conception **Model-Template-View (MTV).** Le flux de processus de MTV est comme suit :

1. le **Model :** retrouve les données à partir de la base de données.
2. **View** : applique la logique métier et le formatage aux données. Cherche les données à partir du Model puis envoie par la suite ces données empaquetées et formatées au Template.
3. **Template** : reçoit les données de la View puis les retourne au navigateur après les avoir mis en forme.

Structure d'un projet Django

Regardons de près la structure du projet (mysite) que nous avons créé lors du TPN°1. Ouvrez votre dossier de projet (mydjango). La structure du répertoire de dossiers est :



Examinons en détail ces dossiers et fichiers :

Le dossier **venv** stocke les fichiers de votre environnement virtuel.
Le dossier externe **mysite** contient votre projet django créé par la commande **startproject.** Ce dossier contient:

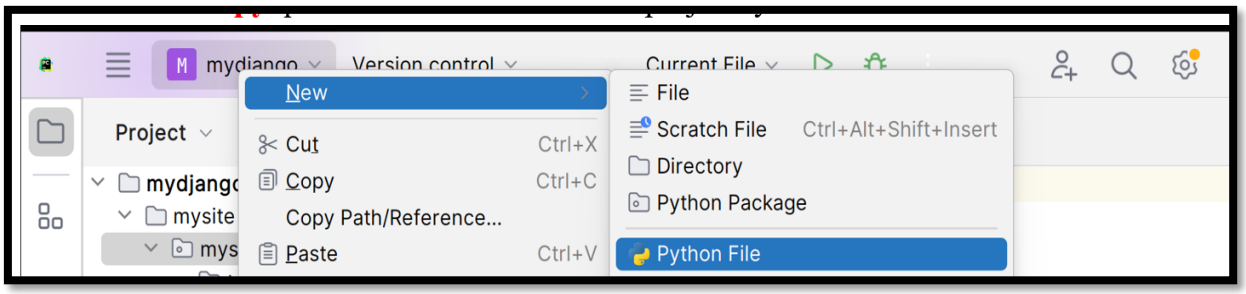
1. Deux fichiers:
   a. **db.sqlite3** : la base de données créée lors de l'exécution de la commande migrate
   b. **manage.py** ; un module utilitaire pour exécuter des commandes au sein de votre projet
2. Le dossier interne **mysite** contient votre vrai projet Django. Ce dossier contient les fichiers suivants
   a. __init__.py : indique à python que le dossier qui contient ce fichier est un package python.
   b. **asgi.py** : c'est le standard Python émergent pour les serveurs et applications asynchrones.
   c. **settings.py :** contient les différentes configurations de votre project.

d. **urls.py :** contient les déclarations des URLs au niveau projet. Par défaut ce fichier contient un simple pattern d'URL pour admin.
e. **wsgi.py : Web Server Gateway Interface** est une spécification qui définit une interface entre des serveurs et des applications web pour le langage Python. Ce fichier est utilisé lors du déploiement du votre site.

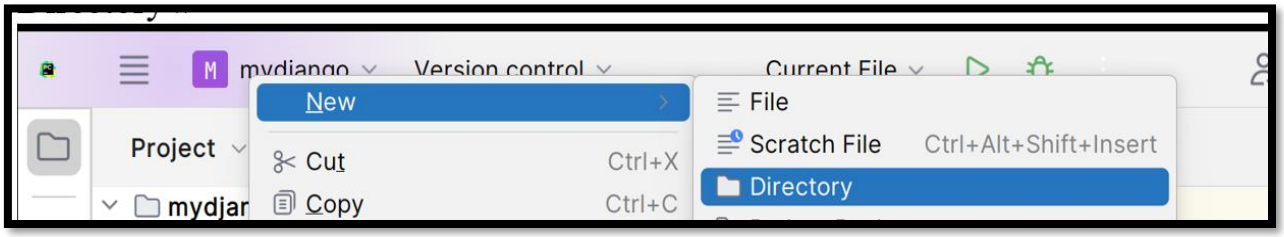
Fonctionnement des fichiers urls.py, views.py et les templates

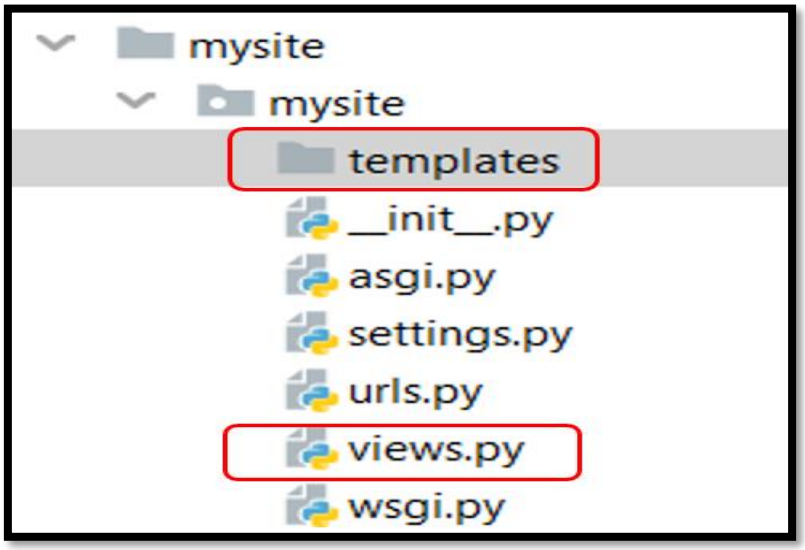
Commençons par créer dans le dossier **mysite** interne :

- le **fichier views.py** qui va contenir les views du projet mysite



- et le **dossier templates** qui va contenir les templates du projet mysite
  Cliquer avec le boiuton droit sur le dossier mysite interne, puis sélectionner « New-> Directory »





Créer une vue

Une vue Django est tout simplement une fonction Python qui prend en paramètre la requête envoyée par le navigateur et retourne un objet de type **HttpResponse :**

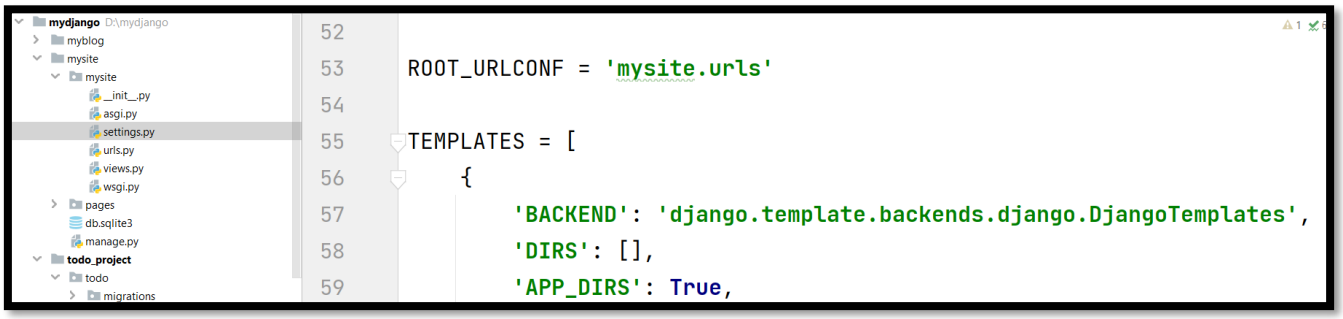
Vues qui retournent un objet HttpResponse
E:/mydjango/mysite/mysite/views.py

```
from django.shortcuts import render
from django.http import HttpResponse

def index_projet(request):
    return HttpResponse("<h1 style='text-align:center;'>Bienvenue au site de LP GLAASRI</h1>")
```

Créer un chemin d'URL à la vue

Django utilise la variable **ROOT_URLCONF** définie dans le fichier settings.py pour savoir quel fichier utiliser pour résoudre les chemins d'URLs.



Modifications à apporter au
Fichier : mysite/mysite/settings.py

```
from pathlib import Path

import os # à ajouter

BASE_DIR = Path(__file__).resolve().parent.parent

ROOT_URLCONF = 'mysite.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'mysite/templates')],#à modifier
        'APP_DIRS': True,
        .....
        },
    },
]
```

Par défaut, cette variable pointe vers le fichier **urls.py** qui se trouve dans le dossier principal de votre projet Django (Dossier mysite interne).
À l'intérieur de ce fichier, on peut utiliser la fonction **path** pour associer un chemin d'URL à une vue :

Association d'une url à une view

Fichier : mysite/mysite/ urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index_projet, name='index_projet'),
]
```

Tester votre projet

Dans le terminal de pycharm, lancer le serveur en tapant la commande :

```
(venv) PS E:\mydjango\mysite> python manage.py runserver
```

Le serveur est lancé comme indiqué ci-dessous

```
(venv) PS E:\mydjango\mysite> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
Django version 5.0.1, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Dans votre browser, aller à l'url : http://127.0.0.01:8000/.
Nous obtenons la réponse ci-dessous.

Applications Django

Pour qu'un site Django soit fonctionnel, il doit contenir des applications Django. Chaque application réalise une seule tâche : un blog, un répertoire d'articles, des transactions ecommerce, …etc.
Un projet Django est donc une collection d'applications et un ensemble de configuration qui composent le site Django.
Les applications sont l'une des fonctionnalités les plus originales de Django. Non seulement ils vous permettent d'ajouter des fonctionnalités à un projet Django sans interférer avec d'autres parties du site Web, mais les applications sont conçues pour être portables, de sorte que vous pouvez utiliser une application dans plusieurs projets.

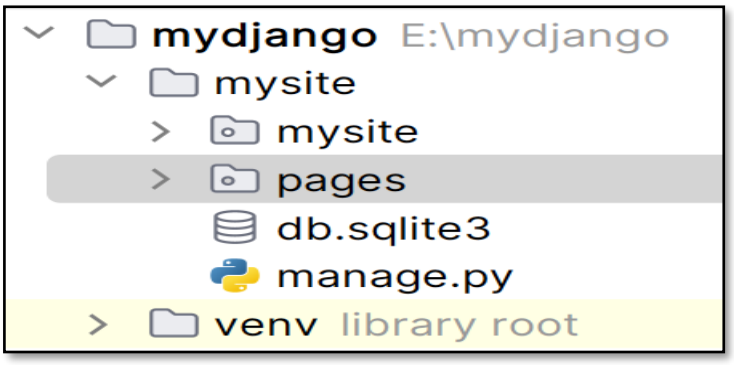
Création d'une première application *pages*

On se propose de créer une application qui va afficher des pages d'information sur notre société fictive GLAASRI Web Design.

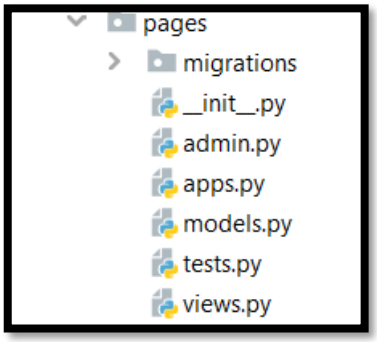
Pour créer une application Django, utiliser la commande :

```
(venv) PS E:\mydjango\mysite> python manage.py startapp pages
```

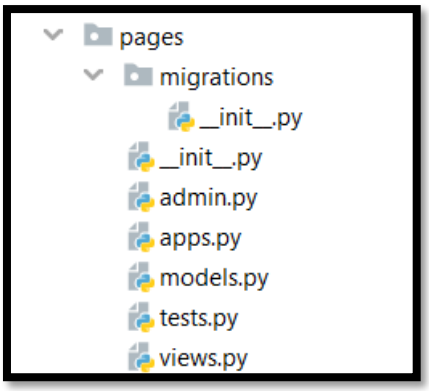
Le système crée un sous répertoire pages dans le répertoire mysite externe de votre projet:



Le sous-répertoire pages contient les fichiers et dossiers ci-dessous :



Le sous-dossier pages contient un sous-dossier migrations qui contient un seul fichier qui indique qu'il s'agit d'un package python.



Une fois que vous avez créé votre application, vous devez indiquer à Django de l'installer dans votre projet. Pour ce faire, il faut l'ajouter à la liste des applications installées dans votre projet. Django est livré avec quelques applications préinstallées.

La liste des applications installées dans votre projet est indiquée par le paramètre **INSTALLED_APPS** du fichier **settings.py**.

La figure ci-dessous montre la valeur par défaut du paramètre **INSTALLED_APPS** du fichier **settings.py**.

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

À l'intérieur de chaque application, Django crée un fichier, **apps.py**, qui contient une classe de configuration portant le nom de votre application. Dans ce cas, la classe s'appelle **PagesConfig**. Cette classe contient par défaut une option de configuration, qui est le nom de l'application ("pages").

fichier : E:/mydjango/mysite/pages/apps.py

```
from django.apps import AppConfig


class PagesConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'pages'
```

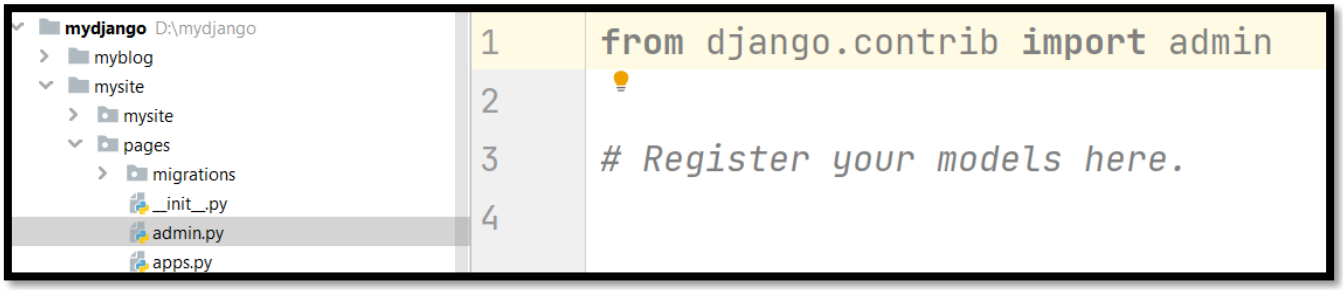
Pour enregistrer notre application avec Django, nous devons pointer vers la classe **PagesConfig** en ajoutant la ligne (**'pages.apps.PagesConfig',)** à la liste stockée dans le Paramètre **INSTALLED_APPS** comme indiqué dans la figure ci-dessous.
Attention : Ne pas oublier la virgule en fin de ligne car vous vous trouvez dans une liste Python dont les éléments doivent être séparés par une virgule

```
INSTALLED_APPS = [
    'pages.apps.PagesConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Structure d'une application Django

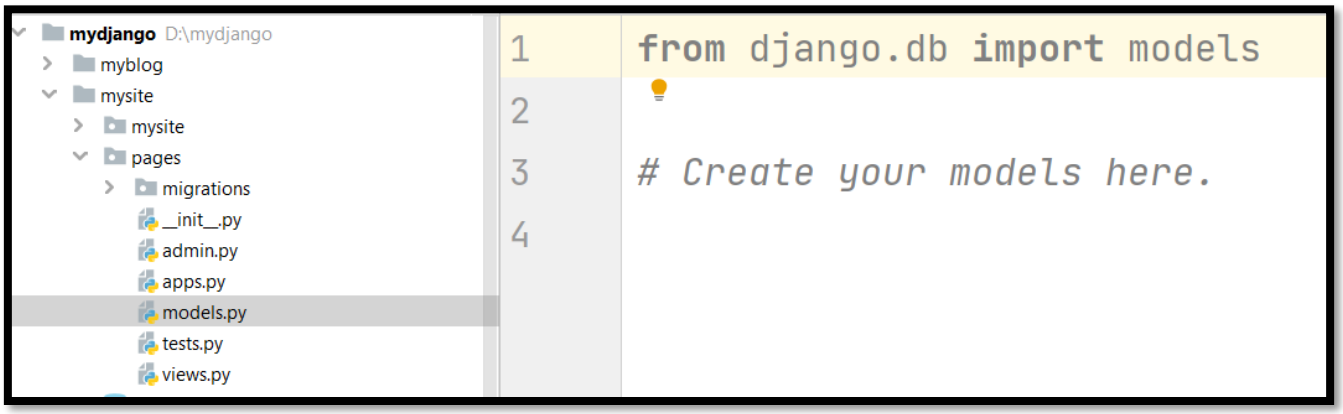
Comme vu ci-dessus, lorsqu'on crée une application, Django crée, dans le dossier du projet, un sous-dossier pour l'application portant son nom. Comme indiqué dans la figure présentée précédemment, le sous dossier contient les fichiers et dossiers ci-dessous :

- **migrations :** sous-dossier dans lequel Django stocke les migrations ou changements apportés à votre base de données. Par défaut ce sous-dossier contient uniquement le fichier __init__ pour indiquer que migrations est un package python.
- **__init__.py :** fichier qui indique que le sous dossier de l'application **pages** est un package.
- **admin.py :** fichier python dans lequel on enregistre les modèles Django avec l'application admin.



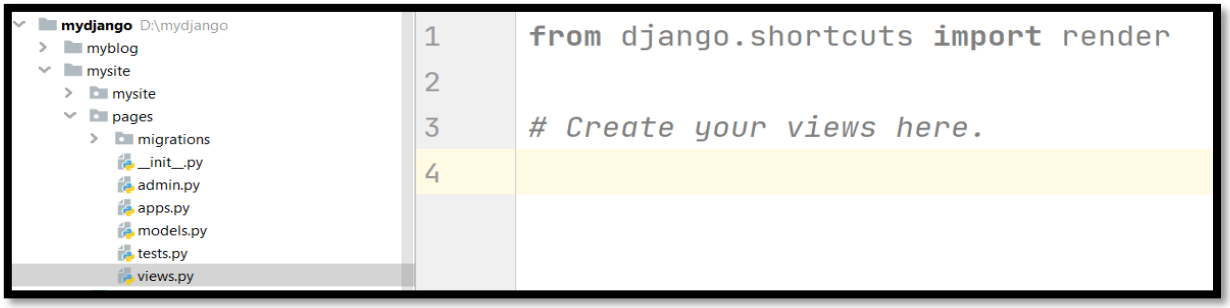
- **apps.py :** Fichier de configuration commun à toutes les applications Django. (Voir son contenu ci-dessus)

- **models.py :** fichier python où sont placés les modèles pour votre application.



- **tests.py :** Contient des procédures de test qui sont exécutées lors du test de votre application.

- **views.py :** Contient les vues de l'application.



Vous remarquez que la plupart de ces fichiers sont vides ou ne contiennent que quelques lignes de code. En effet, les utilitaires **startproject** et **startapp** de Django ne créent que le cadre minimum à partir duquel on construit un site web Django.

Il convient également de noter que vous pouvez créer manuellement tous les fichiers et dossiers d'un site Web Django si vous le souhaitez. Bien que la structure créée par **startproject** et **startapp** soit très courante, ce n'est pas la seule façon de structurer un site web Django.

Création d'une fonction View dans l'application pages

Pour créer notre première fonction View de l'application pages, nous devons modifier le fichier **views.py** dans notre application pages (modifications en gras):

```
#E:\...\mydjango\mysite\pages\views.py
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  def index_pages(request):

5      return HttpResponse("<h1 style='text-align :center ;'>Site de GLAASRI : Application pages</h1>")
```

Examinons de près ce code:

Ligne 1. Importe la méthode **render**. Cette ligne a été ajoutée automatiquement par l'utilitaire **startapp**. **render()** est utilisée pour retourner les templates Django au browser. (Voir TP N°4)

Ligne 2. Importe la classe HttpResponse. Le protocole http utilisé par les browsers pour communiquer avec un serveur web utilise les objets request (instance de HttpRequest) et response (instance de HttpResponse) pour éhanger les données entre le browser et le serveur web.

Lignes 4 et 5. On définit notre fonction View ayant le nom index_pages. Elle reçoit l'objet **request** qui encapsule les informations contenues dans la requête http, et retourne un objet response qui encapsule la réponse envoyée par le serveur au browser.

E:/mydjango/mysite/pages/views.py

```
from django.shortcuts import render
from django.http import HttpResponse


def index_pages(request):
    return HttpResponse("<h1 style='text-align:center;'>Site de GLAASRI : Application pages</h1>")
```

Configuration des URLs

Pour répondre aux urls du fichier urls.py de notre application pages nous devons ajouter au fichier urls du projet un path qui permet d'inclure le fichier urls.py de l'application pages pour qu'il soit consulté chaque fois qu'une url commence par **pages/** .

Lorsque **startproject** a créé notre site Web, il a créé un fichier **urls.py** dans notre site racine (D:\...\ mysite \ urls.py). C'est un bon endroit pour la navigation à l'échelle du site, mais c'est rarement un bon endroit pour mettre des URLs relatives à des applications individuelles. Non seulement le fait d'avoir toutes nos URL dans un fichier est plus complexe et moins portable, mais peut conduire à un comportement étrange si deux applications utilisent une vue avec le même nom.

Pour résoudre ce problème, nous créons un nouveau fichier **urls.py** pour chaque application. L'utilitaire **startapp** ne le fait pas automatiquement car pas toutes les applications ont des vues publiques accessibles via l'URL. Un programme utilitaire qui effectue des tâches d'arrière-plan, par exemple, n'aurait pas besoin d'un fichier urls.py. Rappelez-vous que Django n'assume rien, ce qui vous permet de décider si votre application a besoin de son propre fichier urls.py.

❖ Tout d'abord, nous devons créer un nouveau fichier urls.py dans notre application pages avec le contenu ci-dessous:

E:/mydjango/mysite/pages/urls.py

```
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('', views.index_pages, name='index_pages'),
6  ]
```

Examinons ce code de plus près:

Ligne 1: Importe la fonction **path**. Cette importation est nécessaire pour que le dispatcher d'URL fonctionne et est commune à tous les fichiers **urls.py.**

Ligne 2 : Importe le fichier **views.py** local. L'opérateur de point (".") dans ce cas est un raccourci pour le paquet en cours, donc cela dit "importer le module **views.py** du package actuel (pages).

Ligne 4: Répertorie les patterns d'URLs enregistrés pour cette application. Pour des raisons de lisibilité, la liste est divisée en plusieurs lignes, avec un pattern d'URL par ligne.

Ligne 5: Le dispatcher d'URLs réel:

- ''. Correspond à une chaîne vide. Il correspondra également au «/» car Django supprime automatiquement la barre oblique. En d'autres termes, cela correspond à la fois à http://127.0.0.1:8000/pages/.
- **views.index_pages**: Pointe vers notre view **index_pages**. L'opérateur "point" pointe vers la fonction **index_pages** à l'intérieur du fichier **views.py** que nous avons importé à la ligne 2.

❖ Par la suite, nous devons apporter au fichier urls.py de notre site les changements ci-dessous:

D:/mydjango/mysite/mysite/urls.py

```
1    from django.contrib import admin
2    from django.urls import path, include
3    from . import views
4
5    urlpatterns = [
6        path('admin/', admin.site.urls),
7        path('', views.index_projet, name='index_projet'),
8        path('pages/', include('pages.urls'))
9    ]
```

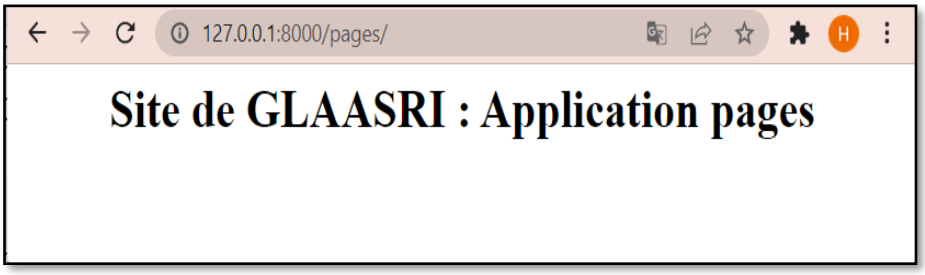
```
1 from django.contrib import admin
2 from django.urls import path, include
3
4 from . import views
5
6 urlpatterns = [
7    path('admin/', admin.site.urls),
8    path('pages/', include('pages.urls'), name="pages_application"),
9    path('', views.index_projet, name='index_projet')
]
```

Nous avons apporté deux modifications importantes au fichier :

Ligne 2: Nous avons ajouté la fonction **include ()** à nos importations.

Ligne 8: Nous avons ajouté un nouveau dispatcher d'URL qui consiste à inclure le fichier **urls.py** de l'application **pages**. Cette ligne indique que chaque fois qu'une url commence par **« pages/ »** (après le nom du domaine), le système doit aller dans le fichier **pages.urls.py** pour chercher la view qui va s'occuper de l'url.

Testez votre travail en lançant le serveur de développement de Django, puis lancez votre navigateur et saisissez l'url http://127.0.0.1:8000/pages/. Vous obtiendrez la page ci-dessous.



Comment cela fonctionne?

1. Le browser a envoyé un message au serveur de développement Django pour lui demander de renvoyer le contenu situé à l'URL racine (http://127.0.0.1:8000/pages/).

2. Django recherche ensuite un pattern d'URL qui correspond à la requête, en commençant la recherche d'abord dans le fichier **urls.py du site**, puis dans le fichier **urls.py de l'application pages**.
3. Django vérifie le premier pattern (admin /) au niveau dans le fichier urls.py de notre site qui ne correspond pas, puis passe à la deuxième ligne dans laquelle se trouve la chaîne vide (URL racine) qui elle aussi ne correspond pas. Puis passe à la troisième ligne dont le paterne correspond cette fois-ci.
4. Le pattern correspondant inclut **l'urls.py** de l'application **pages**. Fondamentalement, cette inclusion dit "aller chercher dans le fichier **urls.py** de l'application des pages un motif qui correspond".
5. Dans le fichier urls.py de l'application pages, la chaîne vide correspond à nouveau au pattern, mais cette fois la demande est envoyée à la fonction View nommée **"index_pages"**.
6. La fonction View **"index_pages"** crée un objet response instance de HttpResponse qui envoie notre message HTML au browser.
7. Le browser affiche ensuite le message sur une page web.