

Chapitre I : Introduction au Framework Django

Enseignantes :

Naima Halouani & Hounaida Moalla



Plan

- Introduction
 - Définition
 - Historique
- Points Forts de Django
- Environnement Virtuel
- Installation de Django
- Création d'un projet
- Serveur de développement
- Application DjangoVs Projet
- Modèle MVT
- Création d'une application

Introduction

Définition

Django est un **framework web** écrit en Python, qui offre une multitude d'outils complets pour implémenter des applications web complètes.

Il a été développé initialement pour un journal local dans le Kansas : World Online.

Introduction

Historique

- En 2003, deux développeurs (Adrian Holovaty et Simon Willison) abandonnent le langage PHP au profit de Python pour développer une application web.
- Ils mettent en place un framework, afin de réduire le temps de développement d'un site.
- Django est disponible depuis 2005 sous la licence BSD et nommé au nom du guitariste « Django Reinhardt ».
- En 2008, une fondation nommée Django Software Foundation (DSF) a été créée pour maintenir Django.

Points Forts de Django

- Un moteur de template très puissant implémentant un concept d'héritage de templates,
- Un ORM (Object Relational Mapper) simple, syntaxe facile, et des performances très honorables,
- Une gestion du routage d'URL (contrôleur frontal des applications) distinguée, fondée sur les patrons d'expressions régulières nommés de Python,
- Un backend puissant permettant la validation des données, une bonne gestion de la base de données et la gestion des utilisateurs, etc.
- Une interface d'administration générée depuis le modèle de données
- Un langage de base, Python, réputé par sa simplicité et sa popularité.

Exemples de sites utilisant de Django

- Parmi les sites web qui utilisent Django, on retrouve (source : Page d'accueil de Django):



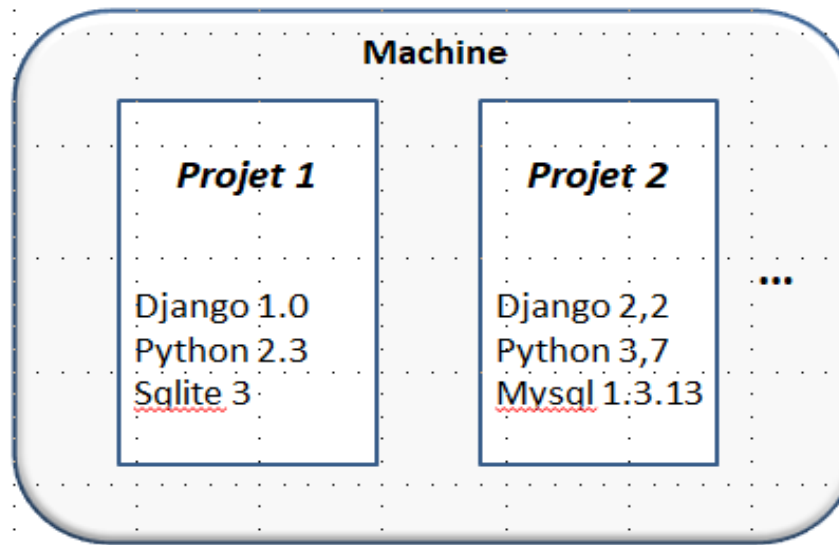
MacArthur Foundation



Environnement virtuel

Description

- L'objectif des environnements virtuels est de créer un environnement comprenant une certaine version de Python et les librairies nécessaires pour le développement.



Environnement virtuel

Mise en œuvre

- **virtualenv** et **virtualenvwrapper** : deux commandes qui fournissent un environnement dédié pour chaque projet Django créé.
- La virtualisation est une bonne pratique pour gagner du temps au moment du déploiement du projet.
- Pour pouvoir exploiter la virtualisation, il faut exécuter l'instruction suivante :

```
... \> py -m pip install virtualenvwrapper-win
```

- Puis il faut créer un environnement virtuel pour le projet :

```
... \> mkvirtualenv env
```

- Pour activer à nouveau l'environnement:

```
... \> workon env
```


Installation de Django

- Vérifier que l'environnement virtuel est actif et ensuite exécuter la commande suivante :

```
...\> py -m pip install Django
```

- Pour vérifier le bon déroulement de l'installation, taper

```
django-admin --version
```

- On peut vérifier que django est installé en lançant la commande suivante :

```
python -m django --version
```

Création d'un projet(1)

- Depuis un terminal en ligne de commande, il faut se déplacer à l'aide de la commande **cd** dans un répertoire dans lequel on souhaite conserver le projet, puis lancer la commande suivante :

py -m django startproject mysite

Ou django-admin startproject mysite

- En résultat un répertoire **mysite** sera créé dans le répertoire courant.

👉 on devrait éviter de nommer les projets en utilisant des noms réservés de Python ou des noms de composants de Django. Cela signifie en particulier qu'on devrait éviter d'utiliser des noms comme **django** (qui entrerait en conflit avec Django lui-même) ou **test** (qui entrerait en conflit avec un composant intégré de Python).

Création d'un projet(2)

- La commande startproject a créé l'arborescence suivante :

mysite/

manage.py

mysite /

__init__.py

settings.py

urls.py

asgi.py

wsgi.py

Création d'un projet(3)

| Fichier | fonction |
|---------------------|--|
| manage.py | Un utilitaire en ligne de commande qui vous permet d'interagir avec ce projet Django de différentes façons. |
| mysite / | Le paquet Python effectif au projet |
| mysite /__init__.py | Un fichier vide qui indique à Python que ce répertoire doit être considéré comme un module python. |
| mysite /settings.py | Un fichier contenant les réglages et configuration de ce projet. |
| mysite /urls.py | Un fichier contenant les déclarations des URL de ce projet Django, une sorte de « table des matières » de votre site Django. |
| mysite /asgi.py | un point d'entrée pour les serveurs Web compatibles aSGI pour déployer votre projet. |
| mysite /wsgi.py | un point d'entrée pour les serveurs Web compatibles WSGI pour déployer votre projet. |

Le serveur de développement (1)

Lancement du serveur

- Pour vérifier que le projet Django fonctionne, il faut se déplacer dans le répertoire **mysite** et lancer la commande suivante:

python manage.py runserver

- Le serveur de développement de Django est démarré, c'est un serveur Web léger entièrement écrit en Python.
- Maintenant que le serveur tourne, il faut accéder à l'adresse <http://127.0.0.1:8000> avec un navigateur Web. En résultat une page apparaît avec le message « Félicitations » ainsi qu'une fusée qui décolle.

Le serveur de développement (2)

Modification du port/IP

☞ Modification du port/IP

- Par défaut, la commande `runserver` démarre le serveur de développement sur l'IP interne sur le port 8000.
- Pour changer cette valeur, il faut la passer comme paramètre sur la ligne de commande. Par exemple, cette commande démarre le serveur sur le port 8080 :

`python manage.py runserver 8080`

- De même pour changer l'adresse IP du serveur, il faut aussi la passer comme paramètre avec le port comme le montre la commande suivante :

`python manage.py runserver 0:8000`

➡ 0 est un raccourci pour 0.0.0.0. qui est ip qui peut atteindre toutes les machines du réseau

Le serveur de développement (2)

Modification du port/IP

☞ Modification du port/IP

- Pour exécuter un serveur sur une machine **A**

`python manage.py runserver 0:8000`

- Puis sur le navigateur de la machine **B** tapez

`http://A:8000`

Le serveur de développement (3)


Rechargement automatique de runserver

- Le serveur de développement recharge automatiquement le code Python lors de chaque requête si nécessaire.
- Pas besoin de redémarrer le serveur pour que les changements de code soient pris en compte.
- Certaines actions comme l'ajout de fichiers ne provoque pas de redémarrage, il est donc nécessaire de redémarrer manuellement le serveur dans ces cas.

Application Django (1)

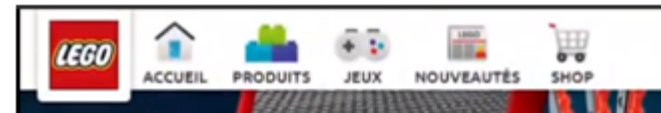
Projets vs. applications

Une application Django est une application Web qui accomplit une tâche précise

Magasin en ligne → 
PRODUITS

Jeux Electronique → 
JEUX

- Un projet est un ensemble de réglages et d'applications pour un site Web particulier.
- Un projet peut contenir plusieurs applications



Une application peut apparaître dans plusieurs projets.

Application Django (2)

Modèle MVT

- Les applications Django sont structurées suivant le patron de conception MVT.
- Le MVT représente une **architecture** orientée autour de trois composants: le **modèle**, la **vue** et le **template**.
- MVT s'inspire de l'architecture MVC, très répandue dans les Framework web

| MVT | MVC |
|------------------------|------------|
| Modèle models.py | Modèle |
| Vue views.py | Contrôleur |
| Template Index.html | Vue |

Application Django (3)

Modèle MVT (`models.py`)

Tous les modèles sont réunis dans un fichier python **models.py**

Le modèle interagit avec une base de données via un **ORM** (*Object Relational Mapping*).

ORM 

l'O.R.M offre une couche d'abstraction qui réalise la traduction des données extraites de la base de données vers un objet propre au langage de programmation.

- Le modèle contient une implémentation des classes qui décrivent les objets manipulés par l'application ainsi que les associations entre ses classes.
- Ces différentes classes seront transformées par Django, en tables dans la base de données.

Application Django (4)

Modèle MVT (Template)

- Un template est un fichier HTML qui peut manipuler des objets Python et qui est lié à une vue.
- Il est placé dans le dossier **templates**.
- Un template peut interpréter des variables python et les afficher.

Application Django (5)

Modèle MVT (`views.py`)

- Une vue reçoit une requête HTTP et y répond de manière intelligible par le navigateur:
 - si une interaction avec la base de données est requise, la vue appelle un modèle et récupère les objets renvoyés par ce dernier.
 - si un gabarit est nécessaire, la vue l'appelle.
- Chaque vue est associée à une url.
- Les urls d'un projet sont regroupés dans le fichier **`urls.py`**

Une **vue** est une **méthode** qui génère le contenu à renvoyer à une requête



une réponse à une requête HTTP.

Application Django (6)

Modèle MVT (urls.py)

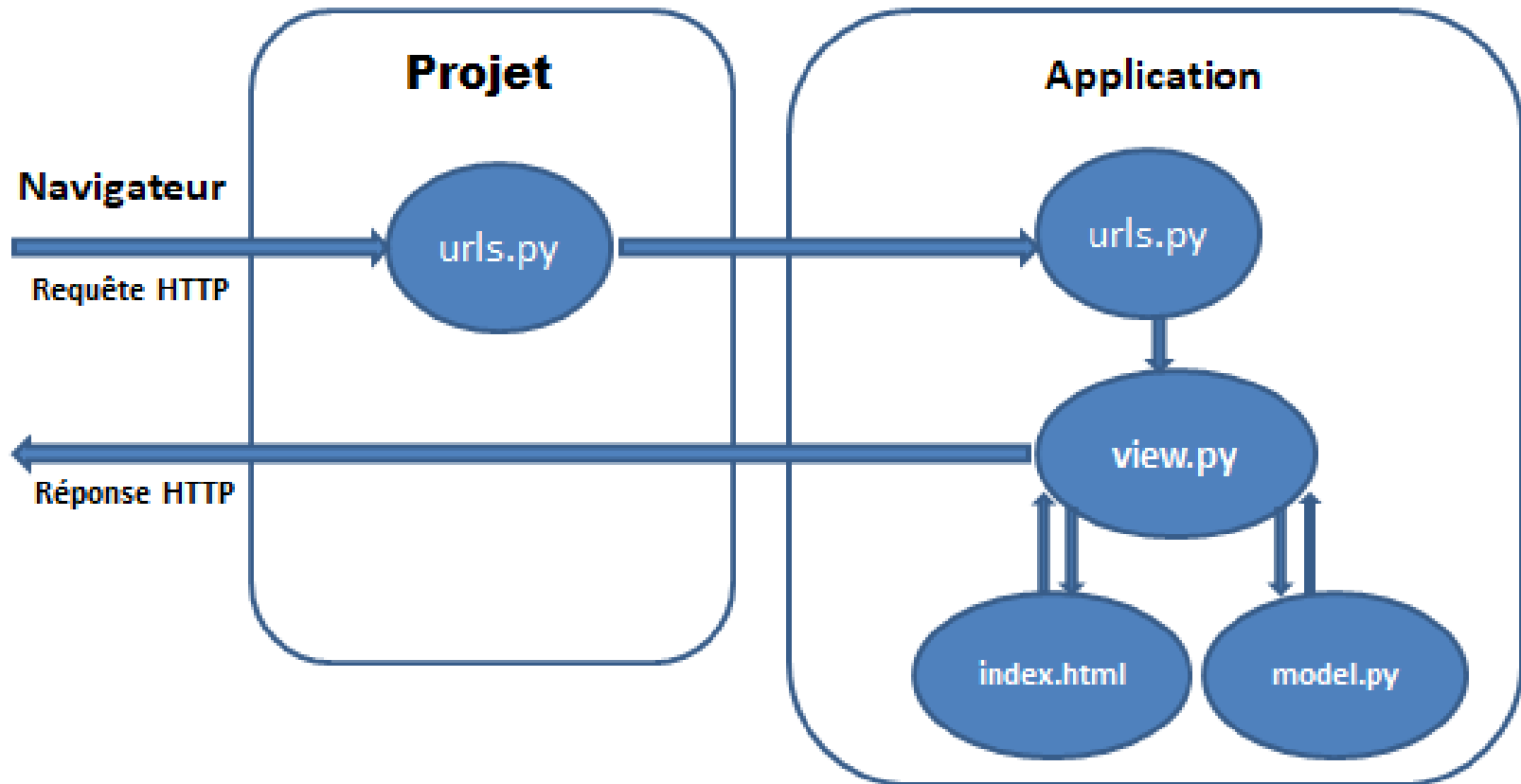
Le routage des URLs est géré à l'aide de la variable `urlpatterns`. Elle constitue une liste Python de fonctions `path()`.

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('magasin/', include('magasin.urls')),]
```

| | |
|--|---|
| <code>path(route,view,...route, view, kwargs=None, name=None)</code> ou bien <code>re_path(...)</code> #route et view sont requises | Renvoie un élément à inclure dans <code>urlpatterns</code> |
| <code>include('NomApplication.urls')</code> | Une fonction acceptant un chemin d'importation Python complet vers un autre module de configuration d'URL devant être « inclus » à cet endroit. |

Application Django (7)

Modèle MVT (interaction entre les composants)



Application Django (8)

Le fichier de configuration (setting.py)

| Variable | Description |
|---|--|
| <code>DEBUG = True</code> | Le mode de débogage affiche des informations pour déboguer vos applications en cas d'erreur. |
| <code>BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))</code> | <p>Cette variable est utilisée pour référencer des chemins vers des fichiers au sein du projet (ressources CSS, JavaScript, fichiers de test, base de données <u>SQLite</u>...).</p> <p>La fonction <u>dirname</u> est appelée deux fois, afin de remonter d'un dossier et ainsi d'avoir le chemin vers la racine du projet.</p> |
| <code>DATABASES</code> | C'est un dictionnaire qui contient la configuration de la base de données. |
| <code>INSTALLED_APPS = ['...', ...]</code> | C'est une liste qui contient la liste des applications gérées par le projet. |
| <code>LANGUAGE_CODE = 'fr-FR'</code> | Elle fixe le langage utilisé au sein de Django, pour afficher les messages d'information et d'erreurs notamment. |
| <code>TIME_ZONE = 'UTC'</code> | Cette variable définit le fuseau horaire, pour l'enregistrement et l'affichage des dates. Choisir UTC dans le cas de l'Europe de l'Ouest. |

Application Django (9)

Création d'une application

Créer une application nommée « magasin »,
`python manage.py startapp magasin`

Cela va créer un répertoire magasin, qui est structuré de la façon suivante :

magasin

`__init__.py`

`admin.py`

`apps.py`

`migrations/`

`__init__.py`

`models.py`

`tests.py`

`views.py`

Application Django (10)

Description des fichiers

| Fichier | Description |
|-------------|--|
| admin.py | <p>C'est le fichier dans lequel nous spécifions les modèles qui seront gérés par l'interface d'administration de Django.</p> <p>Exemple:</p> <pre>from django.contrib import admin from .models import Produit admin.site.register(Produit)</pre> |
| apps.py | Ce fichier est créé pour aider l'utilisateur à configurer certains des attributs de l'application. |
| migrations/ | <p>C'est un répertoire conçu pour loger les fichiers de migration.</p> <p>Les migrations sont considérées un système de contrôle de versions pour un schéma de base de données.</p> |
| models.py | C'est le fichiers contenant les classes qui décrivent les entités manipulées par l'application. |
| test.py | C'est un fichier qui contient les méthodes des tests unitaires de l'application. |
| views.py | C'est un fichier contenant des méthodes qui contrôlent l'interaction entre les templates et les modèles. Une vue est une méthode qui renvoie une réponse à une requête HTTP. |

Application Django (11)

Configurations

1. Inclure l'application dans le projet, on a besoin d'ajouter une référence à sa classe de configuration dans le réglage INSTALLED_APPS. La classe MagasinConfig se trouve dans le fichier magasin/apps.py, ce qui signifie que son chemin pointé est **'magasin.apps.MagasinConfig'**.

Modifiez le fichier mysite/settings.py et ajoutez ce chemin pointé au réglage INSTALLED_APPS.

```
INSTALLED_APPS = [  
    .....  
    'magasin.apps.MagasinConfig',  
]
```

Application Django (12)

Configurations

2. Il faut créer le lien entre le projet django et l'application magasin à travers le fichier `urls.py` du projet.

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('magasin/', include('magasin.urls'))
]
```

Application Django (13)

Configurations

3. Il faut modifier le fichier `magasin/views.py` en lui ajoutant la méthode `index` qui sera appelée en recevant l'url relatif à l'application `magasin` (<http://127.0.0.1:8000/magasin>).

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Première application django")
```

4. Créer le fichier `magasin/urls.py` pour associer l'URL à la méthode correspondante de la vue.

```
from django.urls import path, include
from . import views
urlpatterns = [
    path('', views.index),
]
```