

COURS DJANGO

AHMANI OMAR
HAIRANE ZAKARIA
RAHMOUNE SIFAQES



Le Framework Django



Le Framework Django

Django est un cadre de développement web open source en Python. Il a pour but de rendre le développement d'applications web simple et basé sur la réutilisation de code. Développé en 2003 pour le journal local de Lawrence (État du Kansas, aux États-Unis), Django a été publié sous licence BSD à partir de juillet 2005.

- Créateur : Lawrence Journal-World
- Développé par : Django Software Foundation
- Github : <https://github.com/django/django>

Historique



Historique

- Développé à l'origine à partir de 2003 pour un journal local de la ville de Lawrence dans le Kansas par Adrian Holovaty et Simon Willison
- But : Réaliser une sorte de CMS (content management system), simple à utiliser pour les non informaticiens
- En Open Source sous Licence BSD depuis 2005
- Beaucoup d'évolutions depuis

django

Version





Version 0

Version 1

Version 2

Version 3

Version 4

django

vert : version actuelle ou supportée
bleu : version à venir

Version 3

3.2 LTS

Version 4

4.0 - 4.1 - 4.2 LTS

django

Exemples d'entreprises utilisant Django en 2022



Principe



Principe

Django est un cadre de développement qui s'inspire du principe MVC ou MTV (la vue est gérée par un gabarit) composé de trois parties distinctes :

- Un langage de gabarits flexible qui permet de générer du HTML, XML ou tout autre format texte ;
- Un contrôleur fourni sous la forme d'un « remapping » d'URL à base d'expressions rationnelles ;
- Une API d'accès aux données est automatiquement générée par le cadre compatible CRUD. Inutile d'écrire des requêtes SQL associées à des formulaires, elles sont générées automatiquement par l'ORM.

django

Autres caractéristiques

- MVT = Modèle Vue Template
- Système de templates
- ORM = Object Relational Mapper
- Serveur Web intégré
- Interface d'Admin complète, souple et extensible
- URL dispatcher

django

Architecture MVT



Shéma MVC

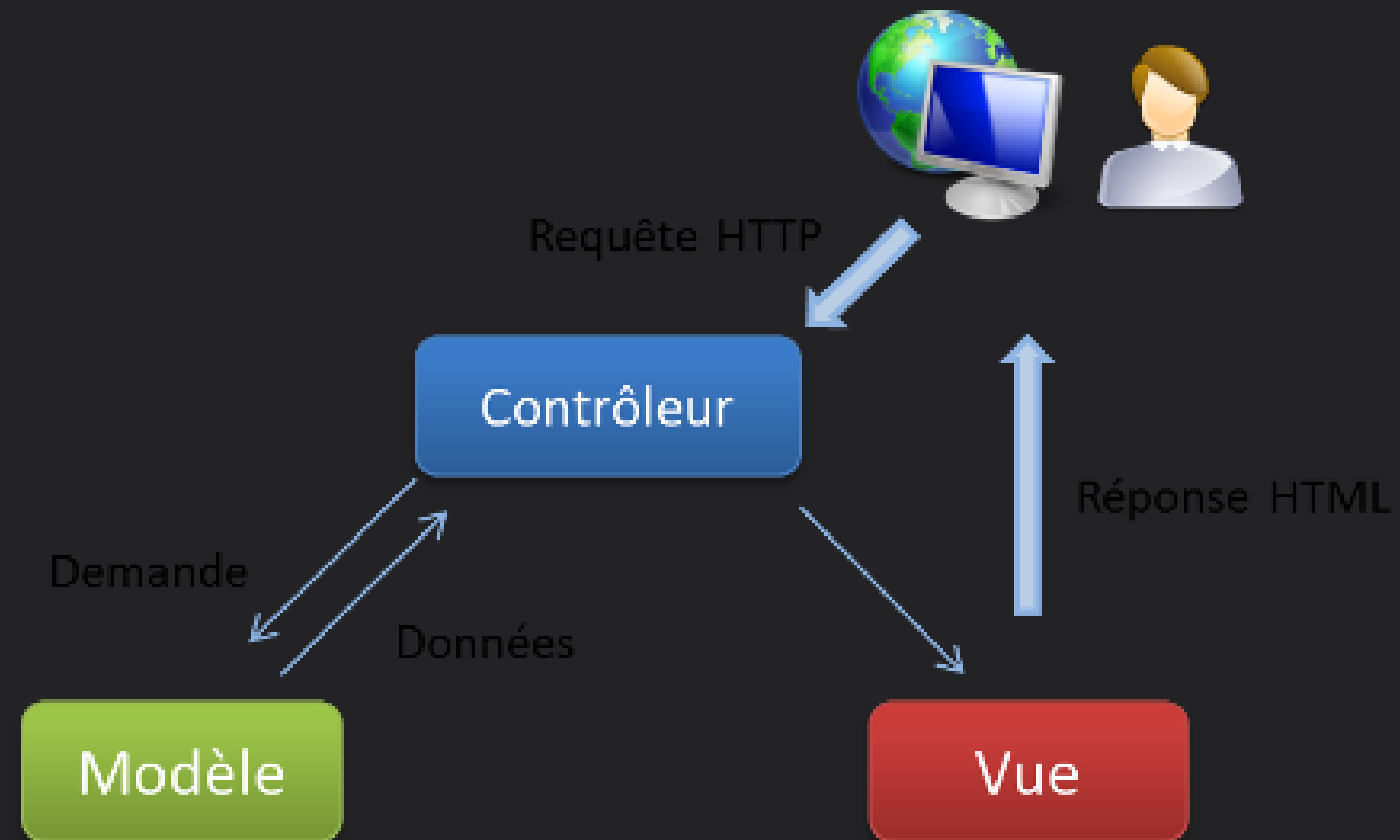


Figure : Schéma de l'architecture MVC

MVC est un modèle de conception de logiciel pour le développement d'applications web

MODEL

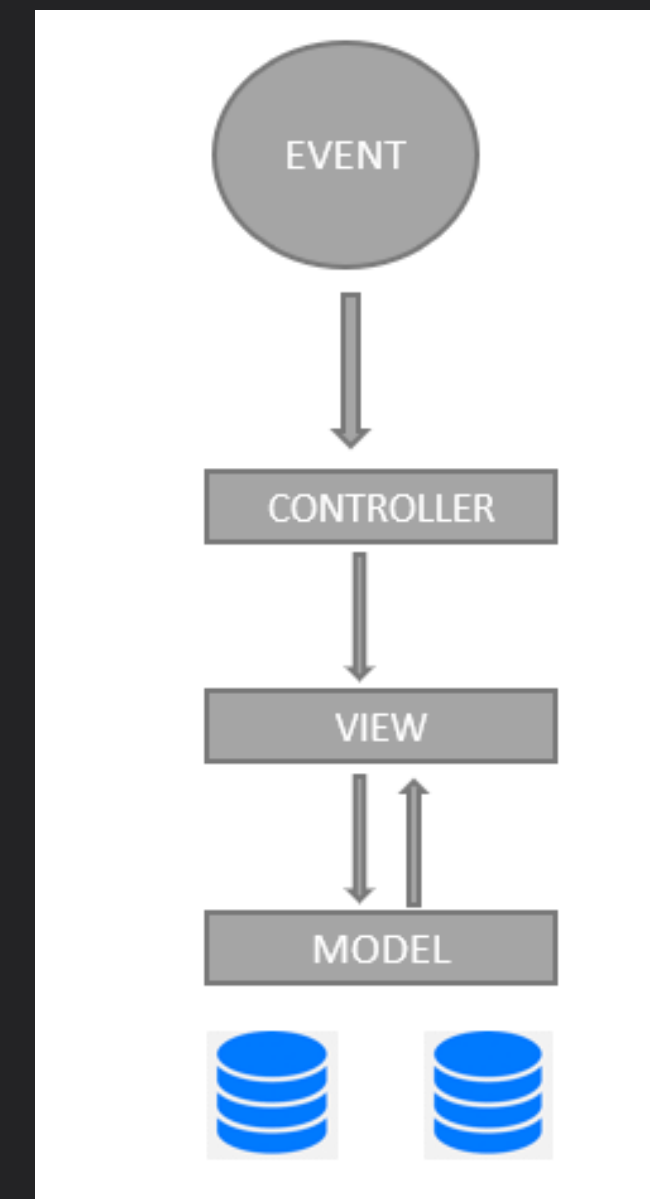
- Le modèle est responsable de la gestion et de la maintenance des données

VIEW

- La vue s'occupe de la présentation. Elle affiche tout ou partie des données à l'utilisateur

CONTROLLER

- Il contrôle les interactions entre le modèle et la vue



Django MCV-MVT Pattern



- MVC est légèrement différent de MVT car Django lui-même s'occupe de la partie contrôleur.
- Il laisse le modèle qui est un fichier HTML mélangé avec le langage de modèle de Django (DTL).

MODEL

VIEW

TEMPLATE

Schéma MVT

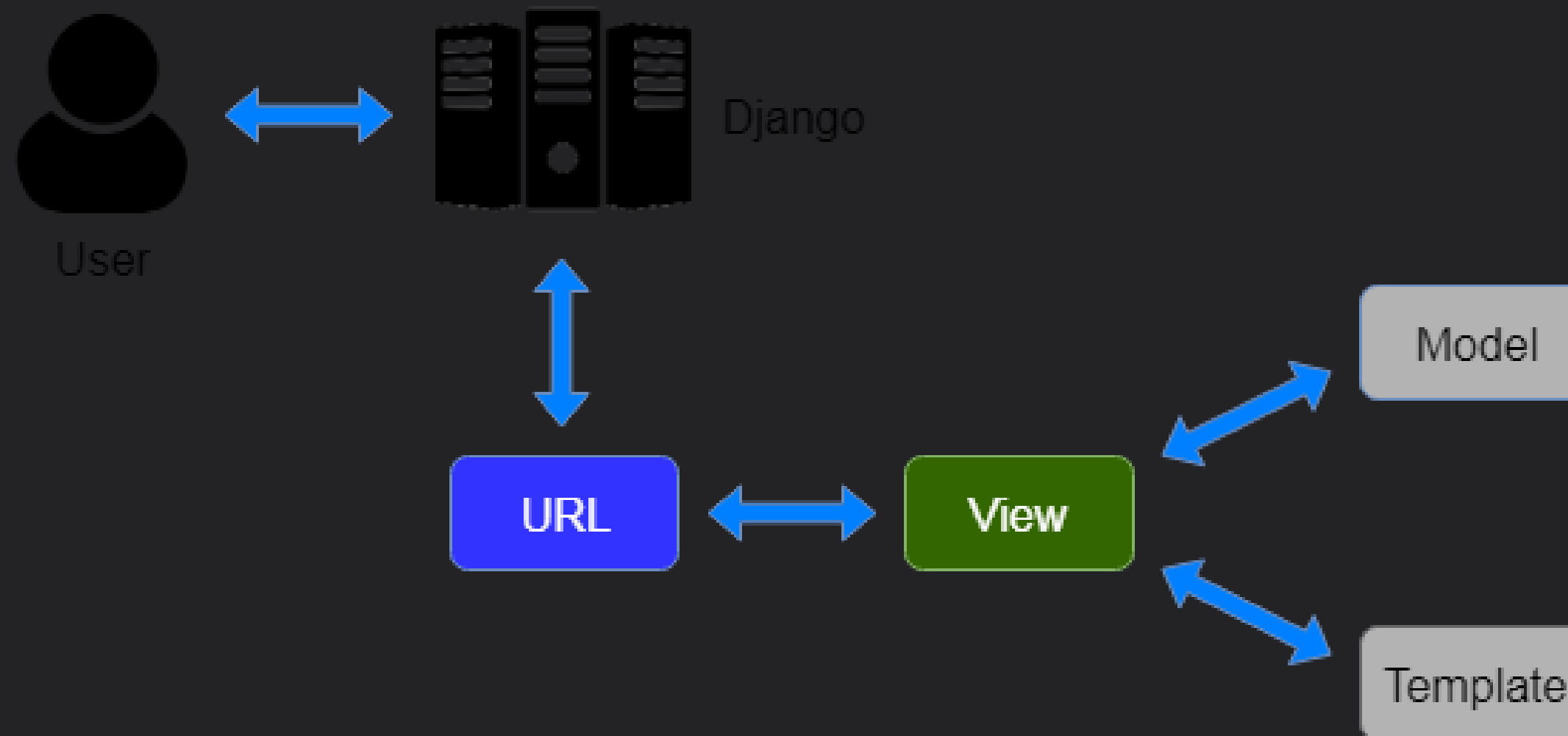


Figure : Schéma de l'architecture MVT

MODEL

```
class Person(models.Model) :  
    first_name = models.CharField(max_length=30)  
    last_name = models.CharField(max_length=30)
```

- Exemples types champs : BooleanField, CharField, DateField...

VIEW

```
class PersonView(django.views.generic.View) :
```

```
    def get(request, id):
```

```
        person = Person.objects.get(id=id)
```

```
        context = { 'myperson': person, }
```

```
        return HttpResponse(template.render(context,request))
```

TEMPLATE

```
<html>  
  <body>  
    <h1>{{ person.first_name }}</h1>  
    <p>{{ person.last_name }}</p>  
  </body>  
</html>
```

TEMPLATE VARIABLE

- Les variables sont entourées par {{et }}comme ceci :

My first name is {{ first_name }}. My last name is {{ last_name }}

Avec un objet :{'first_name': 'John', 'last_name': 'Doe'}

My first name is John. My last name is Doe.

- La recherche de dictionnaire, la recherche d'attributs et les recherches d'index de liste sont implémentées avec une notation par points :

{{ my_dict.key }}

{{ my_object.attribute }}

{{ my_list.0 }}

TEMPLATE TAGS

Dans les modèles Django, vous pouvez exécuter une logique de programmation telle que l'exécution d'if instructions et de for boucles.

Ces mots-clés, **if** et **for**, sont appelés "balises de modèle" dans Django.

Pour exécuter les balises de modèle, nous les encadrons entre **{% %}** parenthèses.

```
{% if greeting == 1 %}  
  <h1>Hello</h1>  
  {% else %}  
  <h1>Bye</h1>  
  {% endif %}
```

```
<ul>  
  {% for x in mymembers %}  
    <li>{{ x.firstname }}</li>  
  {% endfor %}  
</ul>
```

TEMPLATE FILTERS

- Django Template Engine fournit des filtres qui sont utilisés pour transformer les valeurs des variables et les arguments des balises.

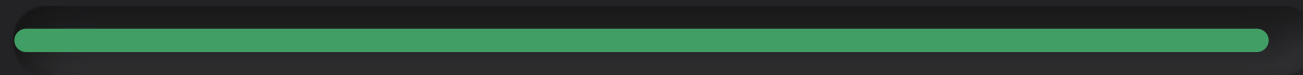
Syntaxe : `{{ nom_variable | nom_filtre }}`

Exemple : `{{ valeur | length }}` , Si la valeur est ['a', 'b', 'c', 'd'] , la sortie sera 4 .

- Il y a plusieurs Principaux filtres de modèles dans Django :

Lien : <https://www.geeksforgeeks.org/django-template-filters/>

Django REST Framework



Django REST Framework

- Le framework Django REST est une boîte à outils puissante et flexible pour la création d'API Web.

Installation: `pip install djangorestframework`

Ajoutez l'application 'rest_framework' à votre INSTALLED_APPS

- Principe :

Création Serializer

Création VIEW

APIVIEW | VIEWSET

Création Chemin URL

Django REST Framework

Création VIEW

APIVIEW

- Utiliser la classe `APIView` est la technique la plus longue pour créer un API, mais elle permet une très grande personnalisation.
- `APIView` fournit un gestionnaire de méthodes pour les verbes HTTP : `get`, `post`, `put` et `patch`. Mais vous devrez écrire une méthode pour chaque verbe HTTP.

Voici un exemple de la vue avec une `get()` méthode pour renvoyer tous les produits de la base de données :

```
class ProductView(APIView):
```

```
    serializer_class = ProductSerializer
```

```
    def get(self, request, *args, **kwargs):
```

```
        products = Product.objects.all()
```

```
        serializer = self.serializer_class(products, many=True)
```

```
        return Response(serializer.data)
```

```
    def post(self, request, *args, **kwargs):
```

```
        serializer = self.serializer_class(data=request.data)
```

```
        serializer.is_valid(raise_exception=True)
```

```
        return Response(serializer.data)
```

Django REST Framework

Création VIEW
VIEWSET

Utiliser la classe ViewSet est la technique la plus rapide afin de créer un API basé sur un modèle Django lié à une base de donnée.

La classe ViewSet va créer pour vous les 7 actions les plus souvent utilisées lorsque l'on crée un API: list, create, retrieve, update, partial_update et destroy. C'est donc beaucoup plus rapide que créer chaque action manuellement comme avec les APIView.

```
from rest_framework import viewsets

from .models import Post
from .serializers import PostSerializer

class PostViewSet(viewsets.ModelViewSet):
    serializer_class = PostSerializer
    queryset = Post.objects.all()
```



Avec ce simple code vous aurez accès aux 6 actions toutes héritées du ModelViewSet et de ses 6 méthodes équivalentes : list, create, retrieve, update, partial_update et destroy

Django REST Framework

Serializer

VIEW

URL

```
from django.urls import path, include
from django.contrib.auth.models import User
from rest_framework import routers, serializers, viewsets

# Serializers define the API representation.
class UserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = User
        fields = ['url', 'username', 'email', 'is_staff']

# ViewSets define the view behavior.
class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer

# Routers provide an easy way of automatically determining the URL conf.
router = routers.DefaultRouter()
router.register(r'users', UserViewSet)

# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
    path('', include(router.urls)),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

INSTALLATION



Django peut être installé sur le système en utilisant le gestionnaire de paquets de Python

- Vous devez avoir installé python et pip, le gestionnaire de paquets de python, au préalable

- Ouvrez le terminal et tapez la commande suivante pour l'installation de Django

```
pip install django
```

1

<https://www.djangoproject.com/download/>

django

The web framework for
perfectionists with deadlines.

OVERVIEW

DOV

Download

How to get Django

Django is available open-source under the [BSD license](#). We recommend using the latest version of Django. The last version to support Python 2.7 is Django 1.11 LTS. See [the FAQ](#) for the Python versions supported by each version of Django. Here's how to get it:

Option 1: Get the latest official version

The latest official version is 4.1.3. Read the [4.1.3 release notes](#), then install it with [pip](#):

```
pip install Django==4.1.3
```

2

Installer la dernière version de
Django

Hands on



Comment créer un projet Django ?

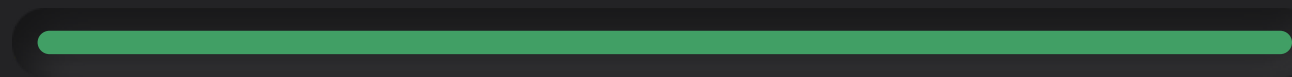
Ouvrez le terminal et naviguez vers le dossier où le projet doit être créé.

```
$ django-admin startproject myproject
```

Cela créera un dossier "myproject" avec la structure suivante

```
Myproject/  
    manage.py  
    myproject/  
        __init__.py  
        settings.py  
        urls.py  
        wsgi.py
```

Structure du projet



Structure du projet

Myproject/

myproject

__init__.py

settings.py

urls.py

Wsgi.py

manage.py

db.sqlite3

Le dossier "myproject" est le répertoire de projet. Vous pouvez le renommer comme vous le souhaitez.

Structure du projet

Myproject/

myproject

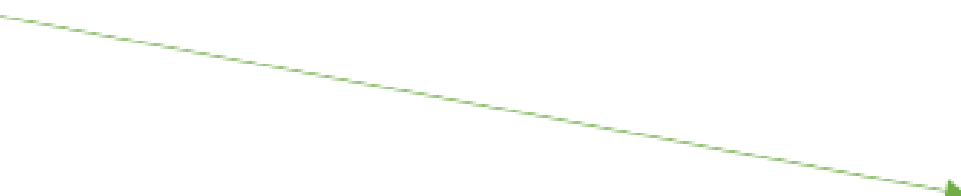
__init__.py

settings.py

urls.py

Wsgi.py

manage.py



Il s'agit d'un utilitaire en ligne de commande qui vous permet d'interagir avec le projet Django.

Structure du projet

Myproject/

myproject

__init__.py

settings.py

urls.py

wsgi.py

Ce dossier est le python package actuel de votre projet qui contient quelques fichiers par défaut. Il est conseillé que son nom soit celui du package Python créé à l'avance, Ces ici que vous devrez importer toutes les connexions avec vos applications.

Structure du projet

Myproject/

myproject

`__init__.py`

settings.py

urls.py

wsgi.py

C'est un fichier vide qui indique à python que ce dossier doit être traité comme un package.

Structure du projet

Myproject/

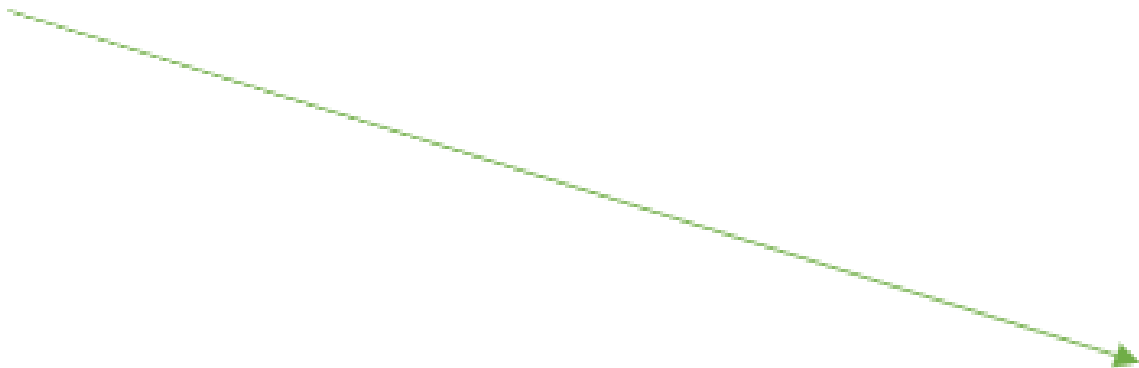
myproject

__init__.py

settings.py

urls.py

wsgi.py



Il contient les paramètres ou les configurations du projet.

Informations sur le fichier settings.py

Un fichier de paramètres Django contient toute la configuration de votre projet Django.

Quelques paramétrages :

- INSTALLED_APPS

```
INSTALLED_APPS = [  
    'polls', // n'oubliez pas de le citer ainsi que des virgules après chaque application  
    'restframework', //exemple  
]
```


Informations sur le fichier settings.py

- BASES DE DONNÉES

Django supporte officiellement les bases de données suivantes : PostgreSQLName, MariaDB, MySQL, Oracle, **SQLite** <– Par défaut

```
BASES DE DONNEES = {  
    'défaut': {  
        'MOTEUR' : 'django.db.backends.postgresql',  
        'NAME' : VOTRE_DB_NAME,  
        'UTILISATEUR' : NOM D'UTILISATEUR,  
        'MOT DE PASSE' : PASSWORD_FOR_DB,  
        'HOST' : 'localhost' // dans le développement.  
    }  
}
```

Informations sur le fichier settings.py



- DÉBOGUER

DEBUG = True // Il s'agit de la valeur par défaut et n'est préférée que dans la phase de développement.

- BASE_DIR

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

Structure du projet

Myproject/


myproject

__init__.py

settings.py

urls.py

wsgi.py



Ce fichier contient tous les liens de votre projet et les fonctions à appeler.

Structure du projet

Myproject/

myproject

__init__.py

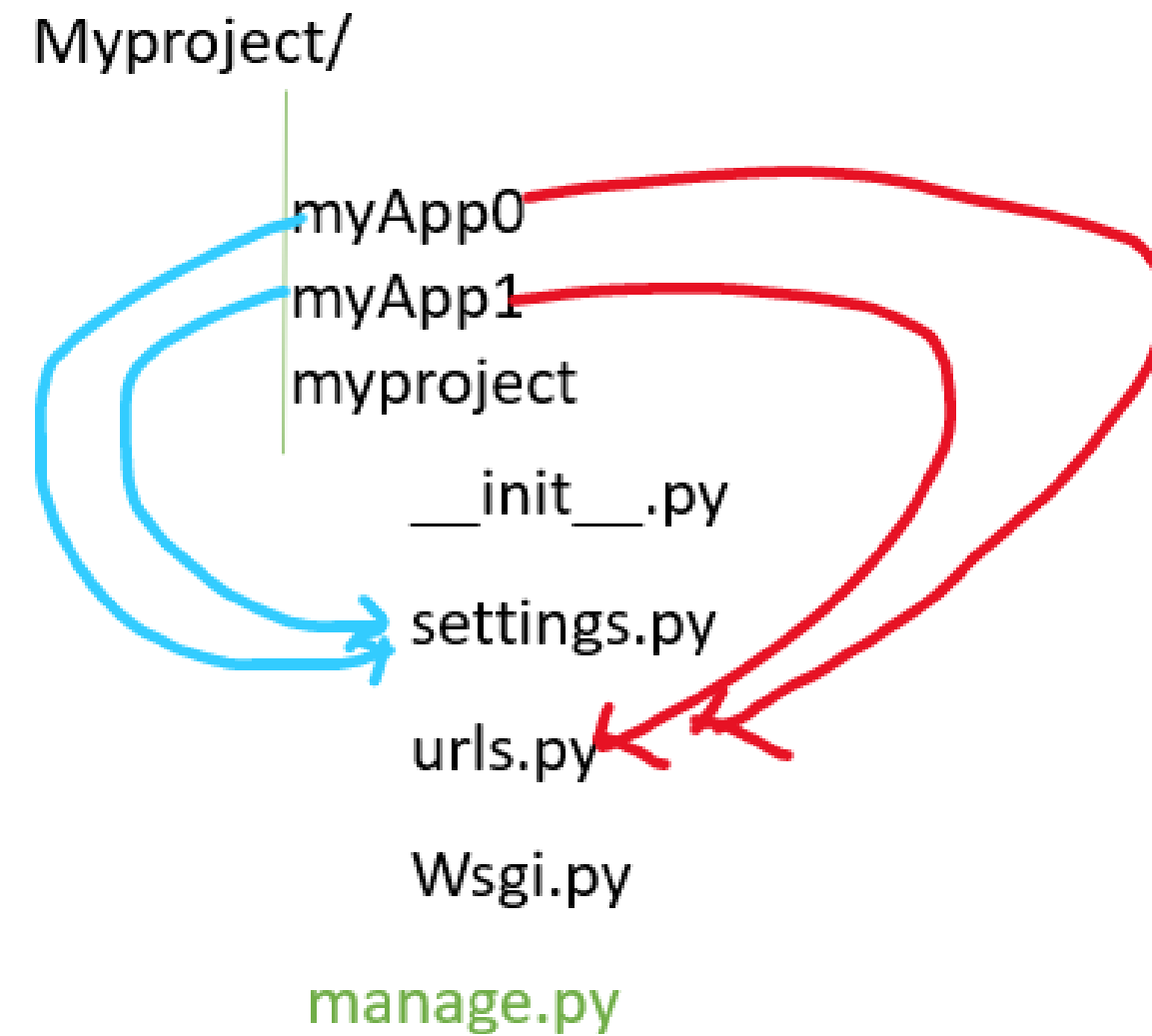
settings.py

urls.py

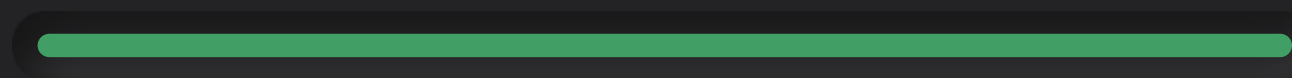
wsgi.py

C'est un point d'entrée pour les services web compatibles WSGI pour servir votre projet.

Structure du projet



Merci à vous



Formulaire de satisfaction

