Django : la dernière tendance des framework pour le développement backend en Python

Laura JEANNETON

Master Informatique - Parcours Ingénierie du développement logiciel

Aix-Marseille Université

Année 2021-2022

Lien Github : https://github.com/LauraJeanneton/AnnuaireLaura

Résumé :

De nos jours, les frameworks sont utilisés par de plus en plus de développeur, débutants comme expérimentés. En effet, ils permettent de fournir un squelette de projet prêt à l'emploi, et font donc gagner du temps aux développeurs. Ceux-ci n'ont plus à réfléchir à l'architecture de leur projet, ni à créer son squelette, puisqu'il peut être généré automatiquement.

Le framework Django fait partie des plus utilisés ces 5 dernières années. Pour les projets Python, il permet donc de grandement aider les développeurs à réaliser leur projet web. Toutes les fonctionnalités qu'ils possèdent permettent d'optimiser l'organisation et la conception de l'application web. L'objectif de départ étant de le rendre le plus simple possible, tous les processus sont en effet extrêmement simplifiés.

Au-delà du peu de prérequis nécessaire, et de son installation d'une grande simplicité, il possède de grands avantages. La création du projet et de l'application, ainsi que la gestion du serveur se font très rapidement et simplement.

Il permet de mettre en place rapidement une base de données associées au projet, et dont la gestion est simplifiée par un site d'administration internet à l'application. Ce site permet de gérer en quelques clics toutes les données, les utilisateurs et les permissions nécessaires au bon fonctionnement du projet. Les modèles de données associés au projet sont très simples à rédiger et à concevoir, grâce à la très grandes variétés de types et d'options possibles.

Le développement à proprement parlé de l'application est faciliter par les fichiers Python générés automatiquement lors de la création du projet, qui permettent de réaliser la création de vues et le routage très simplement. Tout cela est renforcé par de nombreux modules Python existants qui facilite encore l'écriture du code. L'authentification des utilisateurs est notamment prise en charge par un module Python, qui permet le routage et l'authentification des utilisateurs ; tout ce système peut donc être mis en place en quelques minutes par les développeurs.

Tous ces atouts placent Django parmi les frameworks les plus appréciés dans le développement d'application web Python, notamment par les développeurs juniors. En effet, le framework a déjà une vingtaine d'année, et la forte tendance d'utilisation permettent aux développeurs d'avoir l'accès à de nombreux forums, tutoriels et documentations.

Table des matières

1. Introduction

Depuis plusieurs années, il est devenu courant pour les développeurs d'utiliser des framework pour les aider à développer leurs applications. Un framework est un outils qui permet d'établir l'architecture de base d'une application, dans le but de simplifier le développement d'un logiciel. En effet, il permet au développeur d'avoir un cadre de travail préétabli, afin de travailler plus rapidement et efficacement. Un framework est associé à un langage (Python, Java…), et permet d'avoir une organisation optimale du code et des fichiers de l'application.

Il existe différents types de frameworks, permettant de développer différentes sortes de projets : des applications web (Django) , des applications de bureau (Qt) , etc. Le choix du framework dépend donc du langage, de l'application souhaitée et des préférences du développeur. Certains frameworks sont plus populaires que d'autres, et disposent donc de plus grandes documentations et de plus de forums de discussions, il sera donc plus simple pour un développeur débutant de se tourner vers ce type de framework.

Choisir un framework pour le développement d'un projet a pour but de simplifier et de réduire la période de démarrage du projet, d'avoir une meilleure organisation du code et des fichiers, et d'être plus efficaces.

En France, Python est en deuxième position dans le classement des langages informatiques les plus utilisés par les développeurs. Lancé en 1991, c'est un langage orienté objet et multiplateformes (Linux, Windows, IOS, Android …), notamment utilisé dans le développement web. Pour faciliter le développement d'application web en Python, plusieurs framework ont été développé au fils des années. C'est le cas de Django, le framework qui nous intéresse aujourd'hui.

2. Présentation de Django

Ces 5 dernières années, le framework Django s'est placé parmi les framework les plus utilisés en France. Il est qualifié de simple à utiliser, sécurisé et sa rapidité de prise en main est appréciée, ce qui le rend réputé auprès des développeurs web. Lancé en 2003, c'est un framework Python initialement développé pour un journal au Kensas, États-Unis. Il devait aider au développement d'application web complètes rapidement. Le framework a été développer dans le but d'être simple d'utilisation, car les journalistes sans formations en programmation devaient s'en servir aisément. Il ensuite été publié sous licence libre en 2005.

Django est basé sur le modèle de conception logiciel MVC (Model-View-Controller), afin d'avoir une organisation optimale du code de l'application :

- Model : Le modèle gère les données de l'application.
- View : La vue gère l'affichage des données.
- Controller : Le controller permet de faire le lien entre les modèles et les vues, c'est le routeur d'une application.

Ce framework permet de générer automatiquement les répertoires et les paquets Python, afin que le développeur se concentre sur le code, et non sur la mise en place de l'architecture du projet. En effet, toute l'architecture du projet est auto-généré et prête à l'emploi. Il dispose de nombres fonctionnalités mises en place pour simplifier le travail du développeur : des modules de connexions/déconnexions préexistants, des routages simplifiés, un système d'authentification et de sessions complets et compréhensible. Ce document se concentre sur la phase de développement d'une application web, bien que Django permette également de faciliter le processus de tests et de déploiements des projets.

3. Prérequis et installation de Django

La mise en place du framework ci-dessous est détaillée pour un poste de travail sous Linux.

3.1. Prérequis

Avant de commencer l'installation des outils prérequis, il faut tout d'abord mettre à jour le poste de travail : il faut rechercher et installer d'éventuelles mises à jour des paquets.

```
laura@LAPTOP-I5CHVOT0:~$ sudo apt-get update && sudo apt-get -y upgrade
```

Pour pouvoir utiliser Django, il faut que Python 3 soit installé sur le poste de travail. Pour cela il suffit de vérifier, depuis un terminal, la version installée :

```
laura@LAPTOP-I5CHVOT0:~$ python3 -V
Python 3.8.10
```

Si la sortie indique une version (voir ci-dessus) alors Python3 est installé. Si Python 3 n'est pas installé, il faut procéder comme suit :

```
laura@LAPTOP-I5CHVOT0:~/djangoInstall$ sudo apt-get install python3
```

Il est également nécessaire que *pip* soit installé sur le poste de travail : ce gestionnaire de paquets permet d'installer et de gérer des paquets Python.
Il faut ensuite vérifier que l'installation s'est déroulée correctement, en vérifiant la version.

```
laura@LAPTOP-I5CHVOT0:~$ sudo apt-get install -y python3-pip
laura@LAPTOP-I5CHVOT0:~$ pip3 -V
pip 22.1.2 from /usr/local/lib/python3.8/dist-packages/pip (python 3.8)
```

Il faut également installer *virtualenv*, un outils qui permet de créer des environnement virtuel Python. Pour vérifier que l'installation s'est déroulée correctement, il faut vérifier la version installée.

```
laura@LAPTOP-I5CHVOT0:~$ pip3 install virtualenv
laura@LAPTOP-I5CHVOT0:~$ virtualenv --version
virtualenv 20.0.17 from /usr/lib/python3/dist-packages/virtualenv/__init__.py
```

3.2. Installation du framework

Pour installer Django, il faut créer un répertoire d'installation à la racine du poste de travail, puis y accéder.

```
laura@LAPTOP-I5CHVOT0:~$ mkdir djangoInstall
laura@LAPTOP-I5CHVOT0:~$ cd djangoInstall/
laura@LAPTOP-I5CHVOT0:~/djangoInstall$
```

Dans ce répertoire d'installation, il est nécessaire de créer l'environnement virtuel de travail, et de l'activer.

```
laura@LAPTOP-I5CHVOT0:~/djangoInstall$ python3 -m venv ./env
laura@LAPTOP-I5CHVOT0:~/djangoInstall$ virtualenv env
laura@LAPTOP-I5CHVOT0:~/djangoInstall$ . env/bin/activate
```

Une fois l'environnement virtuel activé, un préfixe (*env*) apparait (voir l'image ci-dessous)

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall$
```

Il faut maintenant installer Django de la façon suivante, puis contrôler l'installation en vérifiant la version.

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall$ pip install django
Collecting django
  Using cached Django-4.0.6-py3-none-any.whl (8.0 MB)
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting backports.zoneinfo; python_version < "3.9"
  Using cached backports.zoneinfo-0.2.1-cp38-cp38-manylinux1_x86_64.whl (74 kB)
Collecting asgiref<4,>=3.4.1
  Using cached asgiref-3.5.2-py3-none-any.whl (22 kB)
Installing collected packages: sqlparse, backports.zoneinfo, asgiref, django
Successfully installed asgiref-3.5.2 backports.zoneinfo-0.2.1 django-4.0.6 sqlparse-0.4.2
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall$ django-admin --version
2.2.12
```

Le framework Django est maintenant correctement installé et prêt à être utilisé.

4. Mise en place

Le projet présenté est une application web d'Annuaire, ayant pour but d'avoir accès à un répertoire de contact.

4.1. Création du projet

Pour démarrer un projet web, il faut tout d'abord créer le squelette de base du projet : Django permet de le générer automatiquement.

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall$ django-admin startproject annuaireLaura
```

Cela a permis de créer différents fichiers dans le répertoire de notre projet.

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall$ cd annuaireLaura/
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura$ ls
annuaireLaura  manage.py
```

- Le fichier *manage.py :* Ce fichier permet de pointer sur le fichier *settings.py* grâce à l'initialisation automatique de la variable d'environnement *DJANGO_SETTINGS_MODULE.*
- Le répertoire *annuaireLaura* : Il contient lui-même différents fichiers.

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura/annuaireLaura$ ls
__init__.py  asgi.py  settings.py  urls.py  wsgi.py
```

  - *_init_.py :* Ce fichier est le point d'entrée du projet. C'est un fichier vide, qui permet simplement de dire au projet que le répertoire est un paquet Python.
  - *settings.py :* Ce fichier décrit la configuration du projet. Il permet de définir quels paramètres sont disponibles, comme les adresses IP utilisables par le serveurs.
  - *urls.py :* Ce fichier décrit la table de routage du projet, qui permet de faire le lien entre les URL et leur vue.
  - *wsgi.py* et *asgi.py :* Ces deux fichiers permettent de faire le lien entre l'application et le serveur web. Ils fournissent des normes pour les applications synchrones (*wsgi.py* et *asgi.py*) et asynchrone (*asgi.py*) : *wsgi* est le prédécesseur de *asgi.*
  - Ce fichier contient la configuration de l'interface du serveur Web. Cette interface est la norme Python pour le développement d'application Web. Ce fichier est modifiable par le développeur.

4.2. Lancement du serveur

L'application web sera disponible sur un navigateur web, Django permet de définir les adresses IP autorisées à accéder au projet. Si une adresse IP non autorisée essaie d'accéder à l'application web, cela déclenchera une erreur.

Les adresses IP autorisées doivent être ajoutées dans le paramètres *ALLOWED_HOSTS* du fichier *settings.py*.

```
ALLOWED_HOSTS = ['localhost']
```

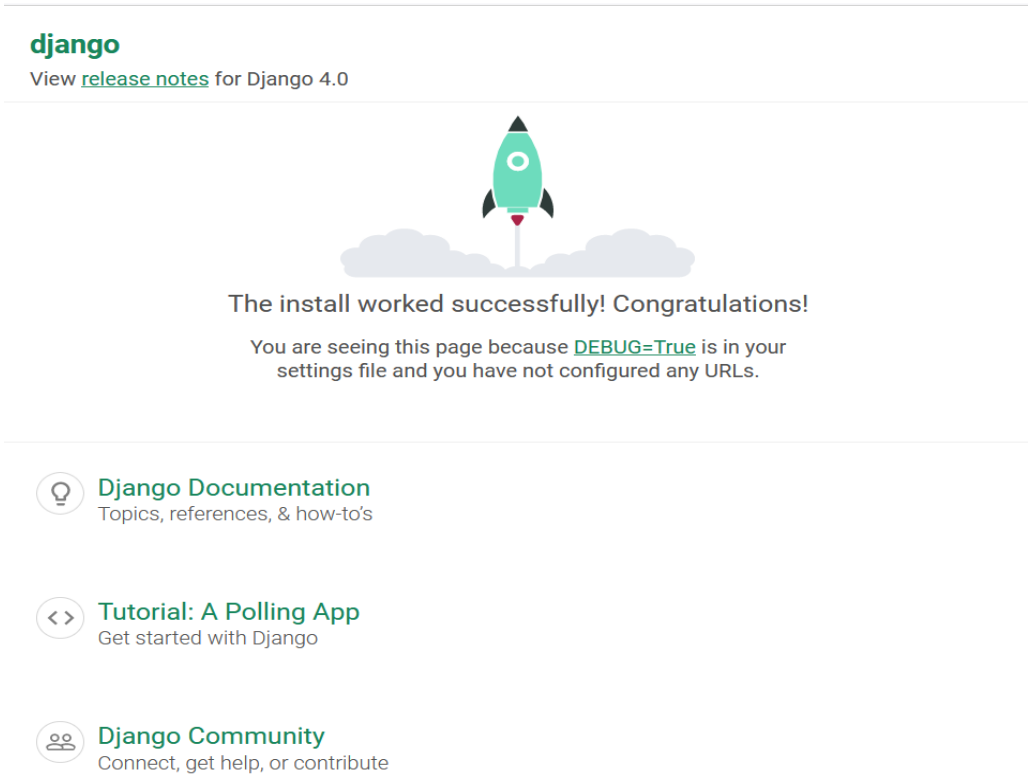
Une fois les adresses IP ajoutées, le serveur peut être lancé sur celles-ci.

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura/annuaireLaura$ cd ..
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura$ python manage.py runserver localhost:808
```

Lorsque le serveur est lancé, le terminal doit afficher une sortie similaire à l'image ci-dessous.

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
July 18, 2022 - 13:05:05
Django version 4.0.6, using settings 'annuaireLaura.settings'
Starting development server at http://localhost:8080/
Quit the server with CONTROL-C.
```

L'application web est accessible à l'adresse IP et au port indiqués lors du démarrage du serveur (*localhost :8080/*). L'affichage par défaut est une page d'accueil Django :



Le projet web est maintenant créer et opérationnel.

4.3. Arrêt du serveur et de l'environnement

Pour arrêter le serveur, il suffit de taper *CTRL+C* dans le terminal. Afin de quitter l'environnement Python, il faut désactiver celui-ci.

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall$ deactivate
```

Le préfixe *env* disparait quand l'environnement a été correctement stoppé.

4.4. Relancer le serveur

Pour réactiver l'environnement de travail, il faut se placer dans le répertoire où se trouve le répertoire *env*, puis activer l'environnement.

```
laura@LAPTOP-I5CHVOT0:~$ cd djangoInstall/
laura@LAPTOP-I5CHVOT0:~/djangoInstall$ ls
annuaireLaura  env
laura@LAPTOP-I5CHVOT0:~/djangoInstall$ . env/bin/activate
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall$
```

Pour relancer le serveur, il faut se placer dans le répertoire où se trouve le fichier *manage.py* et lancer le serveur.

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall$ cd annuaireLaura/
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura$ ls
annuaireLaura  db.sqlite3  manage.py  repertoire
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura$ python manage.py runserver localhost:8080
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
July 18, 2022 - 13:12:16
Django version 4.0.6, using settings 'annuaireLaura.settings'
Starting development server at http://localhost:8080/
Quit the server with CONTROL-C.
```

4.5. Création de l'application.

Django permet de créer une nouvelle application en une seule ligne de commande *startapp*. Celle-ci va générer automatiquement plusieurs fichiers essentiels au développement de l'application web. D'autres fichiers pourront ensuite être ajoutés à l'application. Tout les fichiers générés seront détaillés par la suite.

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura$ python3 manage.py startapp repertoire

(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura$ cd repertoire/
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura/repertoire$ ls
__init__.py  admin.py  apps.py  migrations  models.py  tests.py  views.py
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura/repertoire$
```

- *admin.py* : Ce fichier permet de gérer l'administration de l'application, en sauvegardant les modèles de données.
- *apps.py* : Ce fichier contient le registre des applications du projet et leurs configurations.
- *models.py* : Ce fichier permet de stocker les différents modèles de données.
- *views.py :* Ce fichier contient toutes les vues de l'application.
- *migrations* : Ce dossier permet de gérer les migrations et les modifications de la base de données faites dans les fichiers *models.py*.

Afin de pouvoir être automatiquement prise en charge par le projet, l'application doit être enregistrée dans la variable des apps installées, dans le fichier *settings.py* du projet (à la ligne *INSTALLED_APPS)*.

annuaireLaura/annuaireLaura/settings.py

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

Cette entrée fait référence à l'objet créé automatiquement dans *annuaireLaura/repertoire/apps.py.*

annuaireLaura/repertoire/apps.py

```
from django.apps import AppConfig

class RepertoireConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'repertoire'
```

5. Développement de l'application

Django permet de réaliser facilement le développement d'une application web, avec des processus simplifié et rapide.

Certaines fonctionnalités de Django seront détaillées dans une section « *En pratique* ».

5.1. Création de la base de données

Django permet d'associer simplement une base de données à l'application. Bien que plusieurs bases de données soient utilisables (MySQL, Oracle ou PostgreSQL), celles-ci nécessiteraient des paramètres de connexions supplémentaires (login, mot de passe, port…). Le plus simple est donc d'utiliser SQLite.

La connexion à la base de données SQLite est définie automatiquement lors de la génération de l'application.

annuaireLaura/annuaireLaura/settings.py,

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Les objets du modèle de données sont définis dans le fichier *annuaireLaura/repertoire/models.py*, sous forme de classes. Chaque objet peut être déterminés par différents éléments :

- Des attributs (ou champs) : Le nombre d'attributs est illimité, et correspond à un champs d'une base de données. Ils ont des types prédéfinis (Voir Annexe En savoir plus : les modèles En savoir plus : les modèles) et peuvent avoir plusieurs arguments, dont :
  - *default :* Définit une valeur par défaut.
  - *null :* Définit si le champ est facultatif (True) ou obligatoire (False)
  - *primary_key :* Définit si le champ est une clé primaire.
- Des métadonnées : Il est possible de les déclarer grâce à la classe Meta, afin de maitriser le classement des données. Pour cela, il faut préciser les champs dans l'attribut *ordering* entre simple guillemet, ajouter un '-' devant un champs l'ordonnera de manière décroissante (Voir section « En pratique »).
- La méthode *__str__()* , qui est obligatoire dans chaque modèle de données, renvoie une chaine de caractère correspondant à l'objet créé.
- La méthode *get_absolute_url()*, qui est utilisée par le site d'administration de l'application (Voir Administration Administration). Elle permet de rediriger le super-utilisateur vers la page de détails du contact.

EN PRATIQUE

Pour l'application Répertoire, on utilise le fichier suivant :

annuaireLaura/repertoire/models.py

```python
from django.db import models
from django.urls import reverse

class Contact(models.Model):
    # Champs
    id = models.AutoField(primary_key=True)
    nom = models.CharField(max_length=20, null=False)
    prenom = models.CharField(max_length=30, null=False)
    email=models.CharField(max_length=50, null=True)


    # Metadatas
    class Meta:
        ordering = ['nom','-prenom']

    # Méthodes
    def get_absolute_url(self):
        return reverse('details', args=[str(self.id)])

    def __str__(self):
        return self.nom + " "+self.prenom
```

Nous utilisons un modèle de données très simple : chaque contact possède un identifiant unique auto-incrémenté, un nom (obligatoire), un prénom(obligatoire) et une email (facultatif). Les données seront ordonnées par ordre croissant de Nom et par ordre décroissant de prénom. Ce tri sera visible sur le site d'administration de l'application (voir 5.2 Administration)

5.2. Administration

5.2.1. L'administrateur

Django possède un système d'administration utilisé pour gérer les données de l'application (sauvegarder, création d'instance…). Pour pouvoir être utilisé, les modèles de données doivent être enregistrées sur le site d'administration via le fichier *annuaireLaura/repertoire/admin.py*.

Le site d'administration a besoin d'un super-utilisateur (avec authentification) afin de pouvoir gérés les enregistrements de la base de données. Cet utilisateur aura un login, un email et un mot de passe. Le terme 'enregistrement' désigne une ligne, une instance, une entrée dans la base de données.

```
(env) laura@LAPTOP-I5CHVOT0:~/djangoInstall/annuaireLaura$ python3 manage.py createsuperus
```

Le serveur doit ensuite être relancer (Voir Relancer le serveur Relancer le serveur), afin de prendre en compte ce nouvel utilisateur.

Le site d'administration permet au super-utilisateur d'avoir accès aux données de l'application. Pour chaque modèle de données, l'administrateur pourra :

- Ajouter un nouvel enregistrement (via le bouton Add) dans la base de données. Les champs disponibles sont ceux définit dans le modèle de données, et respectent les contraintes qui leurs ont été affectés.
- Accéder au modèle de données (via le bouton Change) : Tout les enregistrements sont alors présents, visibles et éditables (modifications et suppression). Les actions suivantes sont alors possibles :
  - Modifier les données de l'enregistrement
  - Supprimer l'enregistrement
  - Accéder à l'historique de l'enregistrement
  - Accéder à la page de l'enregistrement

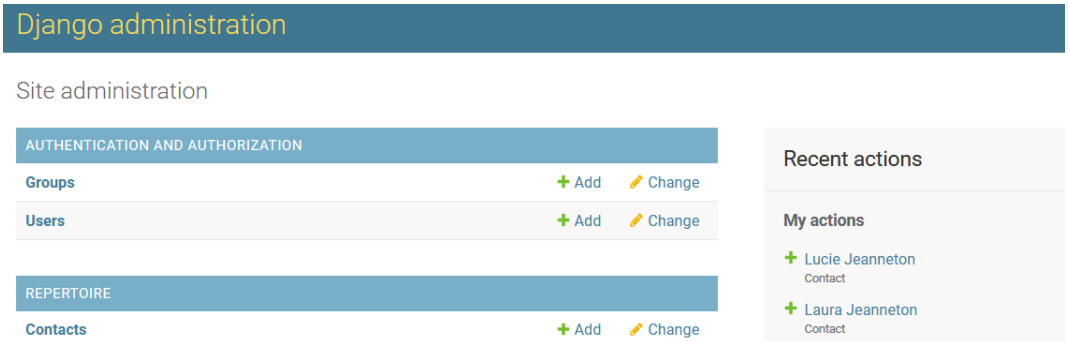
EN PRATIQUE

Il faut tout d'abord enregistrer le modèle de donnée dans l'application.

admin.site.register

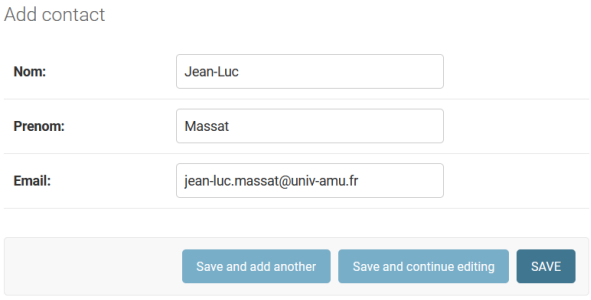
```
from django.contrib import admin
from repertoire.models import Contact

admin.site.register(Contact)
```

Après avoir relancer le serveur, le site d'administration est accessible à l'URL *adresse_ip :8000/admin/* et fourni l'affichage suivant :

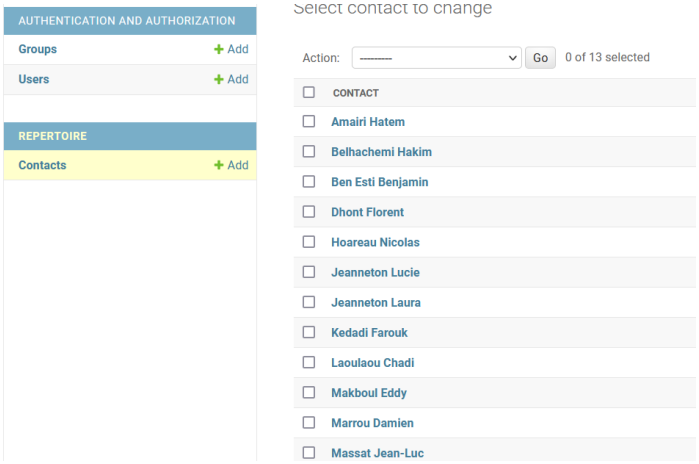


Cela affiche tout les modèles de données dans la section « *REPERTOIRE* », qui peuvent être gérés très facilement. Pour chaque modèle, deux actions sont possibles :

- *Add* : Le bouton permet d'ajouter un nouveau contact au répertoire. Le nom et le prénom sont obligatoires.

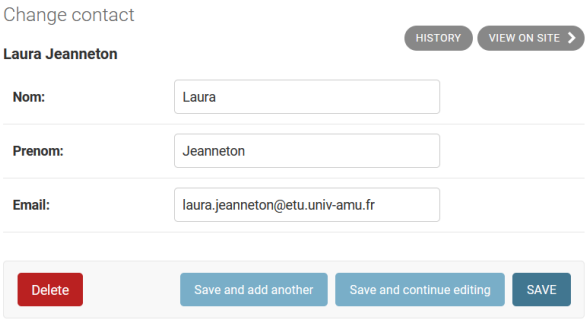


- *Change* : Les contacts ajoutés sont alors présents, visibles et éditables (modifications et suppression.



La page de visualisation des enregistrements permet de :

- Modifier les données du contact : nom, prénom, email
- Supprimer le contact de la base de données
- Accéder à l'historique du contact : cela permet de voir les actions récentes liées à cet utilisateur (ajout dans la base de données, modifications des données.
- Accéder à la page du contact : cela permet d'accéder à la page de détail du contact (voir Création d'une vue)



5.2.2. Les utilisateurs

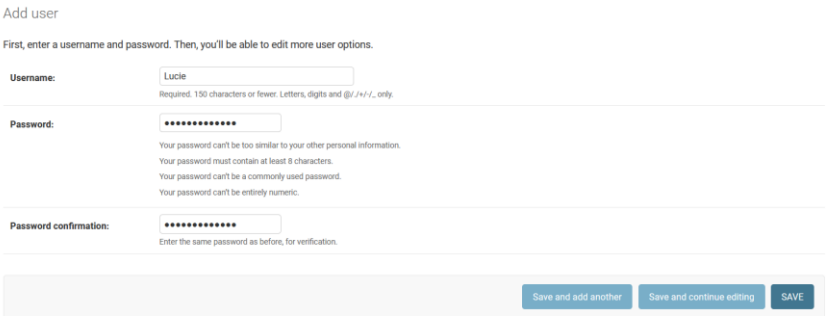
Django permet également de gérer l'authentification d'utilisateur, ainsi que leurs permissions : un utilisateur pourra se connecter/déconnecter et accéder à certaines informations, selon ses autorisations. Le super-utilisateur créé précédemment dispose de toutes les autorisations.

L'ajout d'un simple utilisateur se fait en cliquant sur le bouton 'ADD' de la ligne *User* dans la section *Authentification et Administration* du site d'administration. Le super-utilisateur peut alors lui accorder des droits, l'assigner à un groupe ou ajouter des informations personnelles sur l'utilisateur.

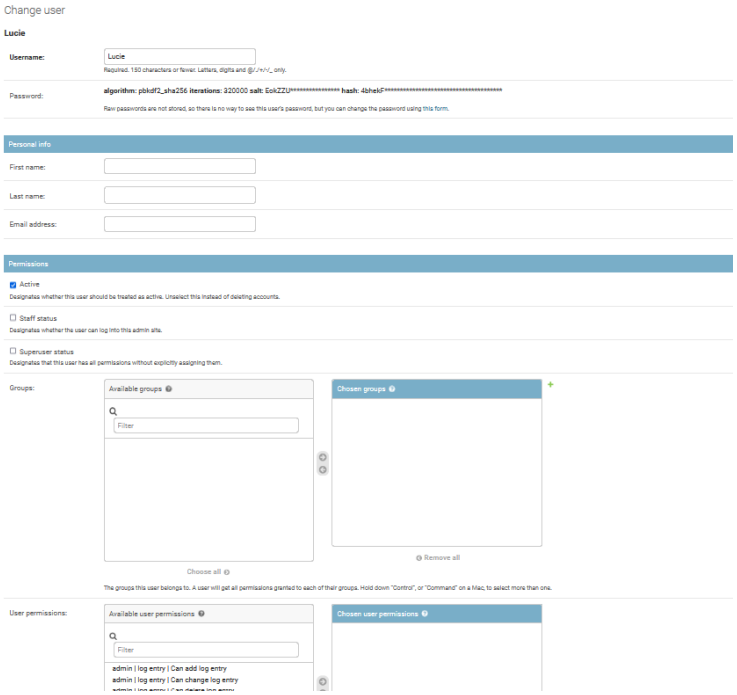
Si plusieurs utilisateurs doivent posséder les mêmes permissions, nous pouvons alors créer un groupe (qui définit les autorisations), et assigner ce groupe à tout les utilisateurs).

EN PRATIQUE.

Nous souhaitons ajouter l'utilisateur Lucie à notre application web. Il suffit donc de cliquer sur le bouton *Add,* et d'entrer le login et le mot de passe de cet utilisateur.



Une page récapitulative de l'utilisateur sera maintenant disponible, afin de gérer son profil : entrer des données personnelles (nom, prénom, email), modifier ses permissions ou encore son groupe. Les données personnelles de l'utilisateur n'ont pas de liens avec le modèle de données définis précédemment.



5.3. Création d'une vue

Les vues servent à renvoyer des pages HTML à la suite d'un traitement d'une requête. Ce sont des templates qui permettent de retourner des réponse HTTP. Les vues sont définies dans le fichier *views.py*. Elles définissent des fonctions, qui permettent de réaliser le traitement d'une requête, c'est-à-dire d'une informations envoyées par le serveur web. La syntaxe de base d'une vue est :

```
from django.shortcuts import render

def nom_fonction(request):
    #Définition des variables nécessaires
                ...
    #Création de l'objet retourné par la fonction
    context = {
                ...
    }
    return render(request,'page.html',context=context)
```

La fonction de retour *render* permet d'afficher la page HTML *'page.html'* et de lui fournir les éléments de *context* afin qu'elle puisse les traiter et les afficher. Dans la page HTML, les éléments du *context* peuvent être récupéré avec la syntaxe *{{nom_variable}}* .Afin que la page HTML puisse être chargée par la vue, il faut l'indiquer à l'application : le chemin du répertoire *templates* doit être ajouté à la liste des variables d'environnements de l'application (dans le fichier *settings.py*, dans *Templates* à la ligne *DIRS*).

EN PRATIQUE

Nous allons créer la fonction *home()* qui va permettre d'afficher la liste des contacts, avec leurs coordonnées (toutes ces informations seront au préalable sauvegardées dans la base de données de l'application) ; ainsi que le nombre total de contacts dans le répertoire.

annuaireLaura/repertoire/views.py

```python
from django.shortcuts import render
from repertoire.models import Contact

def home(request):
    nb_users = Contact.objects.all().count()
    all_users = Contact.objects.all()
    context = {
        'nb_users' : nb_users,
        'all_users' : all_users,
    }
    return render(request,'home.html',context=context)
```

Il suffit maintenant de créer la page HTML *home.html* dans un nouveau dossier *annuaireLaura/repertoire/templates*.

annuaireLaura/repertoire/templates/home.html

```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Annuaire</title>
    {% load static %}

<link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}">
  </head>
  <body>
    <h1> <center>Mon annuaire</center> </h1>

<table>
   <tr>
       <th>Nom Prenom</th>
       <th>Coordonnée</th>
   </tr>

{% for book in all_users %}
    <tr>

          <td>{{ book.nom }} {{book.prenom}}</td>
          <td>{{ book.email }}</td>
      </tr>
   {% endfor %}
   </table>
<br>
   Nombre total d'utilisateurs :  {{nb_users}}

  </body>
```

Cette page HTML va permettre d'afficher la liste des contacts sous la forme d'un tableau (Voir Création d'une vue).

Il faut maintenant permettre à l'application de charger les vues en ajoutant le chemin du répertoire */templates* au paramètre *DIRS* de la variable d'environnement *TEMPLATES*.

annuaireLaura/annuaireLaura/settings.py

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates/'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

5.4. Le routage

Le framework permet de faciliter la prise en charge des URL grâce à une table de routage, stockée dans le fichier *urls.py*

annuaireLaura/annuaireLaura/urls.py.

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

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

Tout le système de routage de l'application est administrée grâce à la variable *urlpatterns* : c'est une liste de *path()*,qui associe une URL à une vue. A la création de l'application, la table de routage ne contient qu'un seul lien : l'URL */admin* qui renvoit vers le module *admin.site.urls*. Comme vu précédemment, c'est cette URL qui permet d'accéder au site administrateur.

Il existe plusieurs façons de créer une nouvelle entrée dans la table de routage, afin de satisfaire à tous les éventuels besoin des développeurs :

- Route simple : Création d'une route avec une URL et une vue. La syntaxe est la suivante : *path('URL',vue,name='Nom').* Pour que cette route fonctionne, il faut également importer la vue au début du fichier (*from … import …*)
- Route déléguée : Création d'une route avec une URL et un second fichier de routage. La syntaxe est la suivante :
  path('URL',include(secondFichier.urls)).
  Cela permet de déléguer le traitement de l'URL à un second fichier de routage afin d'optimiser l'organisation du code. Il faut au préalable importer les fonctions *include* et *path* de *django.urls* en début de fichier.
- Route par une vue : Création d'une route avec une URL et le nom d'une vue générique. Il faut au préalable l'importer du module *django.views.generic.* La syntaxe est la suivante:
  path('URL', Vue.as_view(url='/URL_de_redirection/')).
  Ce type route permet par exemple de rediriger la racine du projet.

Il est également possible de faire passer des informations à travers l'URL, par exemple pour afficher le détails d'une information(dans notre exemple, le détail d'un contact). Pour cela, l'identifiant du détail souhaité doit être indiqué entre chevrons (<>) dans l'URL. Cette données sera ensuite transmise à la fonction via un paramètre, portant le même nom. Pour plus d'informations, voir la section En pratique.

EN PRATIQUE

Nous allons d'abord rediriger la racine de l'application vers l'URL */repertoire/* grâce à une nouvelle entrée dans la table de routage :

annuaireLaura/annuaireLaura/urls.py

```
from django.contrib import admin
from django.urls import path
from django.urls import include
from django.views.generic import RedirectView
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/',admin.site.urls),
    path('', RedirectView.as_view(url='/repertoire/')),

]
```

À cette étape, si on lance le serveur de l'application, on obtient l'erreur suivante :

En effet, nous n'avons pas encore définie l'entrée de la table de routage de l'URL */repertoire*. Pour ce faire, nous allons utiliser la seconde méthode de routage : l'ajout d'une entrée de redirection.

Dans le fichier *annuaireLaura/annuaireLaura/urls.py* , il faut donc créer une entrée pour définir la route de l'URL */repertoire/*

annuaireLaura/annuaireLaura/urls.py

```
from django.contrib import admin
from django.urls import path
from django.urls import include
from django.views.generic import RedirectView
from django.conf import settings

urlpatterns = [
    path('admin/',admin.site.urls),
    path('', RedirectView.as_view(url='/repertoire/')),
    path('repertoire/',include('repertoire.urls')),
]
```

Il faut ensuite créer le fichier de routage *annuaireLaura/repertoire/urls.py* dans le module *repertoire* et ajouter la variable *urlpatterns* une route vers la vue.

annuaireLaura/repertoire/urls.py

```
from django.urls import path
from . import views

urlpatterns=[
    path('',views.home,name='home'),

]
```

L'attribut *name* permet d'avoir un identifiant unique à la vue associé à cette URL.

La routage de la racine de l'application est maintenant opérationnel. Pour récapituler, nous avons indiqué que l'URL */repertoire/* renvoyait vers une seconde table de routage (le fichier *annuaireLaura/repertoire/urls.py)*, et la racine de l'application est elle-même redirigée vers cette URL. Cette URL renvoie alors vers la vue views.home que nous avions précédemment défini.

Si nous lançons le serveur, la page d'accueil de l'application doit s'afficher correctement.

Pour créer la page de détails d'un contact, il suffit de définir la route dans le fichier *annuaireLaura/repertoire/urls.py*, en passant en paramètre de l'URL l'identifiant d'un contact:

annuaireLaura/repertoire/urls.py

```
from django.urls import path
from . import views

urlpatterns=[
    path('',views.home,name='home'),
    path('<idUser>',views.details, name='details')
]
```

Nous pouvons maintenant modifier la view afin d'y ajouter la fonction *details*.

annuaireLaura/repertoire/views.py

```
def details(request, idUser):
    user = Contact.objects.get(id=idUser)
    context = {
        'user' : user,

    }
    return render(request,'details.html',context=context)
```

Ici, *IdUser* correspond à l'identifiant du contact passé en paramètre de l'URL. On peut voir qu'il est bien en paramètre de la fonction *details*, et qu'il permet de récupérer le bon contact: *Contact.objects.get* permet d'accéder au contact ayant l'*id* égal à l'*idUser*.

5.5. Les sessions

Django permet de simplifier le système de sessions d'une application. Une session permet de garder en mémoire des données et de les mettre à disposition de l'application web à chaque connexion. Pour cela, Django utilise une variable *session*, dont les données sont stockées dans la base de données. Le système de session est automatiquement activé lors de la génération de l'application, et est définie dans le fichier *settings.py (ligne 37 et 45)*.

annuaireLaura/annuaireLaura/settings.py

```
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'repertoire.apps.RepertoireConfig'
41 ]
42
43 MIDDLEWARE = [
44     'django.middleware.security.SecurityMiddleware',
45     'django.contrib.sessions.middleware.SessionMiddleware',
46     'django.middleware.common.CommonMiddleware',
47     'django.middleware.csrf.CsrfViewMiddleware',
48     'django.contrib.auth.middleware.AuthenticationMiddleware',
49     'django.contrib.messages.middleware.MessageMiddleware',
50     'django.middleware.clickjacking.XFrameOptionsMiddleware',
51 ]
```

Une session est accessible à partir du paramètre *request* d'une vue, et représentera alors la connexion de l'utilisateur actuel. Des opérations telles que la suppression de données, le parcours des données ou encore la recherche de clés sont possibles sur la session.

EN PRATIQUE

annuaireLaura/repertoire/views.py

```
def home(request):
    nb_users = Contact.objects.all().count()
    all_users = Contact.objects.all()
    expiration_date = request.session.get_expiry_date()
    context = {
        'nb_users' : nb_users,
        'all_users' : all_users,
        'expiration_date':expiration_date
    }
    return render(request,'home.html',context=context)
```

Nous pouvons par exemple récupérer la date d'expiration de la session actuelle. Pour cela, dans la vue *home*, il suffit de créer une variable, d'y stocker cette date et d'ajouter cette nouvelle variable dans le *context* (afin de la transmettre à la page HTML).

AnnuaireLaura/repertoire/templates/home.html

```
<footer>
  Expiration de la session : {{expiration_date}}
```

Nous affichons ensuite cette donnée dans le pied de page de la page d'accueil.

5.6. Authentification

5.6.1.

Comme vu précédemment, Django permet la création simplifiée d'utilisateur de l'application web. De même, le processus d'authentification des utilisateurs est grandement simplifié : Django gère la connexion, la déconnexion, et les réinitialisation des données. Le système de gestion d'authentification est géré par un module Django préexistant : *django.contrib.auth.urls*. Ce module permet d'accéder à plusieurs URL (Voir Sources Sources - Documentation de l'authentification)

La page de connexion de l'application sera alors accessible par l'URL */accounts/login/.* La page HTML correspondant n'est pas automatiquement créée, et doit donc être codée par le développeur. Attention, ces pages HTML doivent être placées dans le répertoire *templates/registration*. La page de connexion doit être nommée *login.html*.

La page de déconnexion de l'application sera accessible par l'URL */accounts/logout/.* Si aucun fichier *logged_out.html* n'est crée manuellement, alors cette URL renverra l'utilisateur vers le message de déconnexion du site d'administration. Pour permettre à des utilisateurs lambdas d'avoir une page de déconnexion personnalisée, il faut donc créer la page *logged_out.html* manuellement dans le répertoire *templates/registration.*

EN PRATIQUE

Il faut tout d'abord créer la page de connexion *login.html.* Elle est très basique et ne contient qu'un formulaire d'authentification.

annuaireLaura /templates/registration/ login.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Annuaire</title>
    {% load static %}

<link rel="stylesheet" type="text/css" href="{% static 'css/registration.css' %}">
  </head>
  <body>
    <h1> <center>Connexion</center> </h1>


{% if form.errors %}
<p>Le login ou le mot de passe est incorrect. Réessayer.</p>
{% endif %}

<form method="post" action="{% url 'login' %}">
{% csrf_token %}
<table>
<tr>
  <td>{{ form.username.label_tag }}</td>
  <td>{{ form.username }}</td>
</tr>
<tr>
  <td>{{ form.password.label_tag }}</td>
  <td>{{ form.password }}</td>
</tr>
</table>

<input type="submit" value="login">
<input type="hidden" name="next" value="{{ next }}">
</form>
```

Il faut ensuite définir le routage de l'URL *accounts/.* La redirection de l'URL *accounts/login* vers cette page HTML est automatique et ne doit pas être ajoutée à la table de routage.

annuaireLaura/annuaireLaura/urls.py

```
from django.contrib import admin
from django.urls import path
from django.urls import include
from django.views.generic import RedirectView
from django.conf import settings

urlpatterns = [
    path('admin/',admin.site.urls),
    path('', RedirectView.as_view(url='/repertoire/')),
    path('repertoire/',include('repertoire.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
]
```

À ce stade, la connexion fonctionne mais renvoie une erreur. En effet, une authentification réussie redirige vers la route */accounts/profile*, que nous n'avons pas définie. Pour corriger cette erreur, il suffit d'ajouter la route de redirection d'authentification dans le fichier *settings.py*.

```
LOGIN_REDIRECT_URL = '/'
```

Pour personnaliser la page de déconnexion des utilisateurs, la page *logged_out.html* est la suivante :

annuaireLaura /templates/registration/ logged_out.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Annuaire</title>
    {% load static %}

<link rel="stylesheet" type="text/css" href="{% static 'css/registration.css' %}">
  </head>
  <body>
    <h1> <center>Déconnexion</center> </h1>

    <p>Déconnexion réussie!</p>
      <a href="{% url 'login'%}">Se reconnecter</a>
  </body>
```

La connexion et la déconnexion d'un utilisateur est maintenant fonctionnelle.

6. Conclusion

Django offre donc une grande variété de fonctionnalités, permettant de faciliter et d'optimiser le travail des développeurs. Bien que nous n'ayons abordés ces possibilités qu'en surface, on peut imaginer à quels points la réalisation d'une application web complète serait grandement simplifiée par ce framework. Les développeurs n'auraient pas à passer du temps à concevoir l'architecture du projet, à penser au routage des données et à tout le système d'authentification, qui peut être très complexes à développer.

En plus de toutes les fonctionnalités présentées dans ce document, Django permet également de gérer, par exemple :

- Les permissions des utilisateurs via le site d'administration
- La réinitialisation des données d'un utilisateur : il possède par exemple un système de réinitialisation de mot de passe très recherché, via une adresse mail de récupération.
- La gestion de formulaire simplifiées

Toutes ces fonctionnalités et leurs facilités d'utilisation et d'adaptation à un projet font donc de Django un framework fortement utilisé et appréciés de tout les développeurs, jeunes comme séniors.

7. Annexe

7.1. Lexique

Métadonnées : Informations permettant de décrire et de traiter du contenu numérique.

Modèle de données : Modèle décrivant la représentation des données dans une base de données.

Paquet python : Ensemble de modules, de fichiers Python

Protocole HTTP : Protocole de communication client-serveur.

Requete HTTP : Requête qui renvoie un protocole HTTP utilisé par le navigateur web.

Table de routage : Structure de données qui permet le transfert de paquets de données

7.2. Acronymes

ASGI : Asynchronous Server Gateway Interface

HTML : HyperText Markup Language

HTTP : HyperText Tranfer Protocol

IP : Internet Protocol

URL : Uniform Resource Locator

WSGI : Web Server Gateway Interface

7.3. En savoir plus : les modèles

Les champs : https://docs.djangoproject.com/fr/2.2/ref/models/fields/#field-types

Les métadonnées : https://docs.djangoproject.com/fr/2.2/ref/models/options/

7.4. En savoir plus : les sessions

https://docs.djangoproject.com/fr/4.0/topics/http/sessions/ (liste des sessions)

7.5. Sources

Tutoriel : https://developer.mozilla.org/fr/docs/Learn/Server-side/Django

Page wikipedia : https://fr.wikipedia.org/wiki/Django_(framework)

Site web : https://www.djangoproject.com/

Documentation Django : https://docs.djangoproject.com/en/4.0/

Documentation de l'authentification : https://docs.djangoproject.com/fr/4.0/topics/auth/default/