Créer le squelette du site : django-admin.exe startproject mysite .

(ne pas oublier le .)

django-admin. est un script qui crée les dossiers et fichiers nécessaires pour vous. Vous devriez maintenant avoir une structure de dossier qui ressemble à celle-ci:

mysite

├───manage.py

├───mysite

│ settings.py

│ urls.py

│ wsgi.py

│ \_\_init\_\_.py

└───requirements.txt

**Note** : dans votre structure de dossier, vous pourrez voir également le répertoire venv que nous avons créé avant.

manage.py est un script qui aide à gérer ou maintenir le site. Entre autres, il permet notamment de lancer un serveur web sur notre ordinateur sans rien installer d'autre.

Le fichier settings.py contient la configuration de votre site web.

urls.py contient une liste de patterns d'urls utilisés par urlresolver

Ouvrir mysite/settings.py pour changer des paramètres du projet :

TIME\_ZONE = 'Europe/Paris'

LANGUAGE\_CODE = 'fr-fr'

Pour créer la base de données :  
python manage.py migrate

Lancez le serveur pour voir si cela fonctionne :

python manage.py runserver 0.0.0.0:8000

Pour créer une app dans Django :

python manage.py startapp <nom de l’app>

Pour déclarer l’app dans Django :

Modifier dans le fichier settings.py

INSTALLED\_APPS = [  
 **'django.contrib.admin'**,  
 **'django.contrib.auth'**,  
 **'django.contrib.contenttypes'**,  
 **'django.contrib.sessions'**,  
 **'django.contrib.messages'**,  
 **'django.contrib.staticfiles'**,  
 **'myblog',**]

Nous allons avoir besoin de créer les modèles dans notre app.

Il faut donc renseigner ces modèles dans le fichier myblog/models.py

Ici nous créons un modèle nommé Post  
Les variables sont les propriétés du modèle

\_\_str\_\_ est obligatoire  
publish() va nous servir à sauvegarder nos données

from django.conf import settings

from django.db import models

from django.utils import timezone

class Post(models.Model):

author = models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)

title = models.CharField(max\_length=200)

text = models.TextField()

created\_date = models.DateTimeField(default=timezone.now)

published\_date = models.DateTimeField(blank=True, null=True)

def publish(self):

self.published\_date = timezone.now()

self.save()

def \_\_str\_\_(self):

return self.title

Nous publierons notre modele en tapant :

python manage.py makemigrations myblog

Pour migrer notre blog :  
python manage.py migrate myblog

Pour pouvoir utiliser les Posts dans l’interface d’admin :  
Ajouter les informations suivantes dans myblog/admin.py:

**from** django.contrib **import** admin  
**from** .models **import** Post  
  
admin.site.register(Post)

Nous devons créer le superuser pour pouvoir nous connecter et administrer le site :

python manage.py createsuperuser

Relancer le serveur avec python manage.py runserver

Ouvrir le navigateur à la page : localhost:8000/admin

Allez dans la rubrique Post pour en créer quelques uns.

Nous allons maintenant créer des vues

Dans le fichier mysite/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('myblog.urls')),

]

Puis créer le fichier urls.py et ajouter ceci :

**from** django.urls **import** path  
**from** . **import** views  
  
urlpatterns = [  
 path(**''**, views.post\_list, name=**'post\_list'**),  
]

Dans le fichier myblog/views.py :

**from** django.shortcuts **import** render  
  
*# Create your views here.***def** post\_list(request):  
 **return** render(request, **'myblog/post\_list.html'**, {})

Dans le dossier myblog créer un dossier templates

puis à l’interieur, un autre dossier myblog

dans ce dossier créer la page post\_list.html

en relançant le serveur, vous devriez avoir une page blanche à l’adresse <http://localhost:8000>

Cela veut dire que vous avez bien été redirigé vers la page post\_list.html

Vous pouvez maintenant la modifier avec par exemple :

<**html**>  
<**body**>  
 <**p**>coucou</**p**>  
</**body**>  
</**html**>

Connectons maintenant notre page avec nos données :

Dans la console saisir :

$ python manage.py shell

Puis :

from myblog.models import Post

et

Post.objects.all()

Nous avons différents morceaux en place : le modèle Post qui est définit dans le fichier models.py, la vue post\_list dans views.py et nous venons de créer notre template. Mais comment allons-nous faire pour faire apparaître nos posts dans notre template HTML ?

Dans myblog/views.py, on doit maintenant trouver:

**from** django.shortcuts **import** render  
**from** django.utils **import** timezone  
**from** .models **import** Post  
  
*# Create your views here.***def** post\_list(request):  
 posts = Post.objects.filter(published\_date\_\_lte=timezone.now()).order\_by(**'published\_date'**)  
 **return** render(request, **'myblog/post\_list.html'**, {**'posts'**: posts})

maintenant que nous avons transmis les données à notre page, nous pouvons les afficher dans la page web **'myblog/post\_list.html' en la modifiant**:

<**html**>  
<**body**>  
 <**p**>coucou</**p**>  
 {% for post in posts %}  
 {{ post }}  
 {% endfor %}  
</**body**>  
</**html**>

Ou pour aller plus loin :

<div>

<h1><a href="/">Mon Blog</a></h1>

</div>

{% for post in posts %}

<div>

<p>published: {{ post.published\_date }}</p>

<h2><a href="">{{ post.title }}</a></h2>

<p>{{ post.text|linebreaksbr }}</p>

</div>

{% endfor %}

Diagramme des fichiers :



Et si on rendait la page plus jolie ?

Nous allons utiliser des feuilles de styles CSS.  
Pour cela nous allons nous aider de Bootstrap <https://getbootstrap.com/>  
Dans le fichier post\_list.html nous allons insérer ces deux lignes en haut de notre fichier :

<**link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css"**>  
<**link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css"**>

Normalement, le résultat est déjà mieux !

Mais allons plus loin dans la personnalisation et créons notre page de style et la ranger dans un dossier static/css

├── mysite

├── myblog

│ ├── migrations

│ ├── static

│ └── templates

myblog

└─── static

└─── css

└─── myblog.css

Nous pouvons ajouter ce code dans myblog.css :

h1 a, h2 a {

color: #C25100;

}

Afin de profiter de ce fichier, nous allons modifier notre post\_list.html :

{% load static %}

<html>

<head>

<title>My blog</title>

<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">

<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">

<link rel="stylesheet" href="{% static 'css/myblog.css' %}">

</head>

<body>

<div>

<h1><a href="/">My Blog</a></h1>

</div>

{% for post in posts %}

<div>

<p>published: {{ post.published\_date }}</p>

<h2><a href="">{{ post.title }}</a></h2>

<p>{{ post.text|linebreaksbr }}</p>

</div>

{% endfor %}

</body>

</html>

Nous avons toute la mécanique CSS en place, nous pouvons nous amuser :

myblog.css :

.page-header {

background-color: #C25100;

margin-top: 0;

padding: 20px 20px 20px 40px;

}

.page-header h1, .page-header h1 a, .page-header h1 a:visited, .page-header h1 a:active {

color: #ffffff;

font-size: 36pt;

text-decoration: none;

}

.content {

margin-left: 40px;

}

h1, h2, h3, h4 {

font-family: 'Lobster', cursive;

}

.date {

color: #828282;

}

.save {

float: right;

}

.post-form textarea, .post-form input {

width: 100%;

}

.top-menu, .top-menu:hover, .top-menu:visited {

color: #ffffff;

float: right;

font-size: 26pt;

margin-right: 20px;

}

.post {

margin-bottom: 70px;

}

.post h2 a, .post h2 a:visited {

color: #000000;

}

Context | Request Context

Et le fichier post\_list.html :

{% load static %}  
<**html**>  
 <**head**>  
 <**title**>My blog</**title**>  
 <**link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css"**>  
 <**link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css"**>  
 <**link href="//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext" rel="stylesheet" type="text/css"**>  
 <**link rel="stylesheet" href="{% static 'css/myblog.css' %}"**>  
 </**head**>  
 <**body**>  
 <**div class="page-header"**>  
 <**h1**><**a href="/"**>My Blog</**a**></**h1**>  
 </**div**>  
  
 <**div class="content container"**>  
 <**div class="row"**>  
 <**div class="col-md-8"**>  
 {% for post in posts %}  
 <**div class="post"**>  
 <**div class="date"**>  
 <**p**>published: {{ post.published\_date }}</**p**>  
 </**div**>  
 <**h2**><**a href=""**>{{ post.title }}</**a**></**h2**>  
 <**p**>{{ post.text|linebreaksbr }}</**p**>  
 </**div**>  
 {% endfor %}  
 </**div**>  
 </**div**>  
 </**div**>  
 </**body**>  
</**html**>

Nous allons maintenant créer un lien pour afficher sur une seule page le contenu d’un article :

En substituant :

<h2><a href="">{{ post.title }}</a></h2>

Par :

<h2><a href="{% url 'post\_detail' pk=post.pk %}">{{ post.title }}</a></h2>

Mais nous devons créer la page post\_detail.html !

Déclarons la dans myblog/urls.py en ajoutant :

path('post/<int:pk>/', views.post\_detail, name='post\_detail'),

Créons dans notre vue la capacité à renvoyer un article unique :

myblog/views.py :

**from** django.shortcuts **import** render, get\_object\_or\_404  
**from** django.utils **import** timezone  
**from** .models **import** Post  
  
*# Create your views here.***def** post\_list(request):  
 posts = Post.objects.filter(published\_date\_\_lte=timezone.now()).order\_by(**'published\_date'**)  
 **return** render(request, **'myblog/post\_list.html'**, {**'posts'**: posts})  
  
**def** post\_detail(request, pk):  
 post = get\_object\_or\_404(Post, pk=pk)  
 **return** render(request, **'myblog/post\_detail.html'**, {**'post'**: post})

et enfin la page html dans templates/myblog:

post\_detail.html :

{% block content %}  
 <**div class="post"**>  
 {% if post.published\_date %}  
 <**div class="date"**>  
 {{ post.published\_date }}  
 </**div**>  
 {% endif %}  
 <**h2**>{{ post.title }}</**h2**>  
 <**p**>{{ post.text|linebreaksbr }}</**p**>  
 </**div**>  
{% endblock %}

Création d’une page statique :

On peut aussi avoir besoin d’une page statique.

Créer une page courbe.html dans myblog/templates/myblog

Ajouter dans myblog/urls.py

path(**'courbe'**, views.courbe,name=**'courbe'** )

Et dans myblog/views.py

**def** courbe(request):  
 print(randint(10))  
 fig = Figure()  
 ax = fig.add\_subplot(111)  
 x = []  
 y = []  
 now = datetime.datetime.now()  
 delta = datetime.timedelta(days=1)  
 **for** i **in** range(10):  
 x.append(now)  
 now += delta  
 y.append(randint(0, 1000))  
 ax.plot\_date(x, y, **'-'**)  
 ax.xaxis.set\_major\_formatter(DateFormatter(**'%Y-%m-%d'**))  
 fig.autofmt\_xdate()  
   
 buf = BytesIO()  
 fig.savefig(buf, format=**'png'**, dpi=300)  
 image\_base64 = base64.b64encode(buf.getvalue()).decode(**'utf-8'**).replace(**'\n'**, **''**)  
 buf.close()  
  
 **return** render(request, **'myblog/courbe.html'**, {**'image\_base64'**:image\_base64})

et dans le fichier courbe.html

<!DOCTYPE **html**>  
<**html lang="fr"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Courbe</**title**>  
</**head**>  
<**body**>  
<**img src="data:image/png;base64,{{image\_base64}}" alt="some text to display to your users when the image does not show correctly" width=500 height=auto** />  
</**body**>  
</**html**>

Regardez le résultat sur localhost:8000/courbe

Comment pourrait-on mettre un graphe des températures de toulouse dans la page ?

Trouver dans la documentation :

-Comment créer un formulaire de contact et envoyer un mail  
-Comment générer un pdf