DJANGO🔫⛓️

```
django-admin startproject crepes_bretonnes
```

```
$ python manage.py runserver
Performing system checks...
```

```
python manage.py startapp blog
```

```
crepes_bretonnes/
├── blog
|   ├── admin.py
|   ├── apps.py
|   ├── __init__.py
|   ├── migrations
|   |   └── __init__.py
|   ├── models.py
|   ├── tests.py
|   └── views.py
├── crepes_bretonnes
|   ├── __init__.py
|   ├── settings.py
|   ├── urls.py
|   └── wsgi.py
└── manage.py
```

commentaire :

```
<p>Ma page HTML</p>
<!-- Ce commentaire HTML sera visible dans le code source. -->
    {# Ce commentaire Django ne sera pas visible dans le code source. #}
```

dans les template :

```
{% comment %}
    Ceci est une page d'exemple. Elle
    - tableau des ventes
    - locations
    - retours en garantie
{% endcomment %}
<table>...
```

— — — — — — — — URL ROOTING DANS URLS.PY : — — — — — — — —

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

urlpatterns = [
    re_path(r'^accueil', views.home),
    re_path(r'^article/(?P<id_article>.+)', views.view_article),
    re_path(r'^articles/(?P<tag>.+)', views.list_articles_by_tag),
    re_path(r'^articles/(?P<year>\d{4})/(?P<month>\d{2})', views.list_articles),
]
```

il faut ajouter le path de **l'url** et la **vue** correspondante ds **views**.py

```
urlpatterns = [
    path('accueil', views.home),
    path('article/<id_article>', views.view_article),
    path('articles/<str:tag>', views.list_articles_by_tag),
    path('articles/<int:year>/<int:month>', views.list_articles),
]
```

Ici, on peut imaginer que l'on a deux autres vues. La première qui prend en paramètre du texte, permettant de filtrer les articles à afficher selon leurs mots-clés, et une autre qui affiche les articles par mois. Il existe 5 types de données identifiables par ce système de routage :

- `str` : c'est le format par défaut (celui utilisé pour notre id_article, par exemple). Cela permet de récupérer une chaîne de caractères non vide, excepté le caractère "/" ;
- `int` : correspond à une suite de chiffres, et renverra donc un entier à notre vue ;
- `slug` : correspond à une chaîne de caractères sans accents ou caractères spéciaux. Un exemple de slug peut être mon-1er-article-de-blog ;
- `uuid` : format standardisé de données, souvent utilisé pour avoir des identifiants uniques ;
- `path` : similaire à str, mais accepte également le "/". Cela permet de récupérer n'importe quelle URL, quel que soit son nombre de segments.

— — — — — — — DANS SETTINGS.PY

— — — — — — — —

DEBUG = False/True

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

`

```
STATIC_URL = '/static/'   # Qui devrait déjà être la configuration par défaut

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "static"),
)
```

```
crepes_bretonnes/
    blog/
        static/
            blog/
                crepes.jpg
        templates/
            blog/
                addition.html
                date.html
    crepes_bretonnes/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    static/
        css/
        img/
            header.png
        js/
    templates/
        base.html
```

— — — — — — — — — DANS VIEWS.PY

— — — — — — — — —

il faut définir des fonction/vue, prenant minimum **request** comme 1er param.
les autres param sont récupérer depuis l'url
-> article/<int:argument> pour l'url article/4 la vue aura un param arg
ument = 4 pr générer l'article 4

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

def home(request):
    """ Exemple de page non valide au niveau HTML pour que l'exemple soit concis """
    return HttpResponse("""
        <h1>Bienvenue sur mon blog !</h1>
        <p>Les crêpes bretonnes ça tue des mouettes en plein vol !</p>
    """)
```

```python
# urls.py
urlpatterns = [
    path('articles/<int:year>/', views.list_articles),
    path('articles/<int:year>/<int:month>', views.list_articles),
]

# views.py
def list_articles(request, year, month=1):
    return HttpResponse('Articles de %s/%s' % (year, month))
```

nommer la vue : param **name**

```python
path('article/<int:id_article>$', views.view_article, name='afficher_article'),
```

par la méthode GET sont bien évidemment récupérables, via le dictionnaire `request.GET` dans la vue.
Ici, `request.GET['ref']` retournerait `'twitter'`.

```python
from django.http import HttpResponse, Http404
from django.shortcuts import redirect

def view_article(request, id_article):
    if id_article > 100:
        raise Http404

    return redirect(view_redirection)

def view_redirection(request):
    return HttpResponse("Vous avez été redirigé.")
```

```python
path('redirection', views.view_redirection),
```

```python
return redirect(view_article, id_article=42)
```

```python
from django.shortcuts import redirect, get_object_or_404, render
from mini_url.models import MiniURL
from mini_url.forms import MiniURLForm


def liste(request):
    """ Affichage des redirections """
    minis = MiniURL.objects.order_by('-nb_acces')

    return render(request, 'mini_url/liste.html', locals())


def nouveau(request):
    """ Ajout d'une redirection """
    if request.method == "POST":
        form = MiniURLForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect(liste)
    else:
        form = MiniURLForm()

    return render(request, 'mini_url/nouveau.html', {'form': form})


def redirection(request, code):
    """ Redirection vers l'URL enregistrée """
    mini = get_object_or_404(MiniURL, code=code)
    mini.nb_acces += 1
    mini.save()

    return redirect(mini.url, permanent=True)
```

dans models.py

```python
from django.shortcuts import redirect, get_object_or_404, render
from mini_url.models import MiniURL
from mini_url.forms import MiniURLForm


def liste(request):
    """ Affichage des redirections """
    minis = MiniURL.objects.order_by('-nb_acces')

    return render(request, 'mini_url/liste.html', locals())


def nouveau(request):
    """ Ajout d'une redirection """
    if request.method == "POST":
        form = MiniURLForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect(liste)
    else:
        form = MiniURLForm()

    return render(request, 'mini_url/nouveau.html', {'form': form})


def redirection(request, code):
    """ Redirection vers l'URL enregistrée """
    mini = get_object_or_404(MiniURL, code=code)
    mini.nb_acces += 1
    mini.save()

    return redirect(mini.url, permanent=True)
```

– – – – – – – TEMPLATE – – – – – – – – –

```python
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            # Cette ligne ajoute le dossier templates/ à la racine du projet
            os.path.join(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',
            ],
        },
    },
]
```

Nous vous conseillons de créer un dossier `templates` à la racine du projet.

python

```python
from datetime import datetime
from django.shortcuts import render

def date_actuelle(request):
    return render(request, 'blog/date.html', {'date': datetime.now()})


def addition(request, nombre1, nombre2):
    total = nombre1 + nombre2

    # Retourne nombre1, nombre2 et la somme des deux au tpl
    return render(request, 'blog/addition.html', locals())
```

python

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

urlpatterns = [
    path('date', views.date_actuelle),
    path('addition/<int:nombre1>/<int:nombre2>/', views.addition)
]
```

Comme vous pouvez le voir, la fonction `render` prend en argument trois paramètres :

1. la requête HTTP initiale, que l'on appelle `request` par convention, pour rappel ;
2. le chemin vers le template adéquat dans *un des dossiers* de templates donnés dans `settings.py` ;
3. un dictionnaire reprenant les variables qui seront accessibles dans le template.

Comme ça par exemple dans date.html :

```html
<h1>Bienvenue sur mon blog</h1>
<p>La date actuelle est : {{ date }}</p>
```

```
<h1>Ma super calculatrice</h1>
<p>{{ nombre1 }} + {{ nombre2 }}, ça fait <strong>{{ total }}</strong> !<br />
Nous pouvons également calculer la somme dans le template : {{ nombre1|add:nombre2 }}.</p>
```

{{ variable_transmise_par_la_vue }} = valeur de la variable
{{ var|filtre:valeur }} -> les filtres avec |filtre:

```
Bienvenue {{ pseudo|default:"visiteur" }}
```

```
{{ texte|truncatewords:80 }}
```

```
{% if age > 25 %}
    Bienvenue Monsieur, passez un excellent moment dans nos locaux.
{% elif age > 16 %}
    Vas-y, tu peux passer.
{% else %}
    Tu ne peux pas rentrer petit, tu es trop jeune !
{% endif %}
```

parcours de liste

```
Les couleurs de l'arc-en-ciel sont :

<ul>
{% for couleur in couleurs %}
    <li>{{ couleur }}</li>
{% endfor %}
</ul>
```

```
Les couleurs de l'arc-en-ciel sont :
<ul>
    <li>rouge</li>
    <li>orange</li>
    <li>jaune</li>
    <li>vert</li>
    <li>bleu</li>
    <li>indigo</li>
    <li>violet</li>
</ul>
```

parcours de dictionnaire

```
couleurs = {
    'FF0000':'rouge',
    'ED7F10':'orange',
    'FFFF00':'jaune',
    '00FF00':'vert',
    '0000FF':'bleu',
    '4B0082':'indigo',
    '660099':'violet',
}
```

```
Les couleurs de l'arc-en-ciel sont :

<ul>
{% for code, nom in couleurs.items %}
    <li style="color: #{{ code }}">{{ nom }}</li>
{% endfor %}
</ul>
```

empty pour les cases vides :

```
1 <h3>Commentaires de l'article</h3>
2 {% for commentaire in commentaires %}
3     <p>{{ commentaire }}</p>
4 {% empty %}
5     <p class="empty">Pas de commentaires pour le moment.</p>
6 {% endfor %}
```

heritage :

seront définis dans un autre template, et réutilisables dans le template actuel. Dès lors, nous pouvons créer un fichier, appelé usuellement `base.html`, qui va définir la structure globale de la page, autrement dit son *squelette*. Par exemple :

django

```
<!DOCTYPE html>
<html lang="fr">
<head>
   <link rel="stylesheet" href="/media/css/style.css" />
   <title>{% block title %}Mon blog sur les crêpes bretonnes{% endblock %}</title>
</head>
<body>
<header>Crêpes bretonnes</header>
    <nav>
       {% block nav %}
       <ul>
            <li><a href="/">Accueil</a></li>
            <li><a href="/blog/">Blog</a></li>
            <li><a href="/contact/">Contact</a></li>
       </ul>
       {% endblock %}
   </nav>
   <section id="content">
       {% block content %}{% endblock %}
   </section>
<footer>&copy; Crêpes bretonnes</footer>
```

```
{% extends "base.html" %}
{% block title %}Ma page d'accueil{% endblock %}
{% block content %}
    <h2>Bienvenue !</h2>
    <p>
       Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec rhoncus
       massa non tortor. Vestibulum diam diam, posuere in viverra in,
       ullamcorper et libero. Donec eget libero quis risus congue imperdiet ac
       id lectus. Nam euismod cursus arcu, et consequat libero ullamcorper sit
       amet.
    </p>
{% endblock %}
```

```
<a href="{% url "afficher_article" ID_article %}">
    Lien vers mon super article n° {{ ID_article }}
</a>
```

génère si ID_article = 42:

```
<a href="/blog/article/42">
    Lien vers mon super article n° 42
</a>
```

load fichier :

```
{% load static %}
<img src="{% static 'blog/crepes.jpg' %}" alt="Mon image" />
```

— — — — — — — MODELS — — — — — — — — —

```python
from django.db import models
from django.utils import timezone

class Article(models.Model):
    titre = models.CharField(max_length=100)
    auteur = models.CharField(max_length=42)
    contenu = models.TextField(null=True)
    date = models.DateTimeField(default=timezone.now,
                                verbose_name="Date de parution")

    class Meta:
        verbose_name = "article"
        ordering = ['date']

    def __str__(self):
        """
        Cette méthode que nous définirons dans tous les modèles
        nous permettra de reconnaître facilement les différents objets que
        nous traiterons plus tard dans l'administration
        """
        return self.titre
```

```
>>> for article in Article.objects.filter(auteur="Maxime"):
... print(article.titre, "par", article.auteur)

La Bretagne par Maxime
```

```
>>> Article.objects.all()
<QuerySet [<Article: Les crêpes>, <Article: La Bretagne>]>
```

```
>>> article.delete()
```

```
>>> article.save()
```

```
article = Article(titre="Bonjour", auteur="Maxime")
article.contenu = "Les crêpes bretonnes sont trop bonnes !"
```

```
>>> from blog.models import Article
```

```
>>> for article in Article.objects.exclude(auteur="Maxime"):
```

```
>>> Article.objects.order_by('date')
```

```
>>> Article.objects.filter(date__lt=timezone.now()).order_by('date','titre').reverse()
```

```
>>> Article.objects.get(auteur__startswith="M")
```

```
Article.objects.get_or_create(auteur="Mathieu")
```

— — — — — — — FORM — — — — — — — — —

Un formulaire hérite de la classe mère `Form` du module `django.forms`. Tous les champs sont également dans ce module et reprennent la plupart du temps les mêmes noms que ceux des modèles. Voici un bref exemple de formulaire de contact :

pythor

```python
from django import forms

class ContactForm(forms.Form):
    sujet = forms.CharField(max_length=100)
    message = forms.CharField(widget=forms.Textarea)
    envoyeur = forms.EmailField(label="Votre adresse e-mail")
    renvoi = forms.BooleanField(help_text="Cochez si vous souhaitez obtenir une copie du mail envoyé.", required=False)
```

```python
from .forms import ContactForm

def contact(request):
    # Construire le formulaire, soit avec les données postées,
    # soit vide si l'utilisateur accède pour la première fois
    # à la page.
    form = ContactForm(request.POST or None)
    # Nous vérifions que les données envoyées sont valides
    # Cette méthode renvoie False s'il n'y a pas de données
    # dans le formulaire ou qu'il contient des erreurs.
    if form.is_valid():
        # Ici nous pouvons traiter les données du formulaire
        sujet = form.cleaned_data['sujet']
        message = form.cleaned_data['message']
        envoyeur = form.cleaned_data['envoyeur']
        renvoi = form.cleaned_data['renvoi']

        # Nous pourrions ici envoyer l'e-mail grâce aux données
        # que nous venons de récupérer
        envoi = True

    # Quoiqu'il arrive, on affiche la page du formulaire.
    return render(request, 'blog/contact.html', locals())
```

```python
path('contact/', views.contact, name='contact'),
```

```python
class Article(models.Model):
    titre = models.CharField(max_length=100)
    auteur = models.CharField(max_length=42)
    slug = models.SlugField(max_length=100)
    contenu = models.TextField(null=True)
    date = models.DateTimeField(default=timezone.now, verbose_name="Date de parution")
    categorie = models.ForeignKey(Categorie)

    def __str__(self):
        return self.titre
```

Pour faire un formulaire à partir de ce modèle, c'est très simple :

pytho

```python
from django import forms
from .models import Article

class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = '__all__'
```

Et c'est tout ! Notons que nous héritons maintenant de `forms.ModelForm` et non plus de `forms.Form`. Il y a également une sous-classe `Meta` (comme pour les modèles), qui permet de spécifier des informations supplémentaires. Dans l'exemple, nous avons juste indiqué sur quelle classe le `ModelForm` devait se baser (à savoir le modèle `Article`,