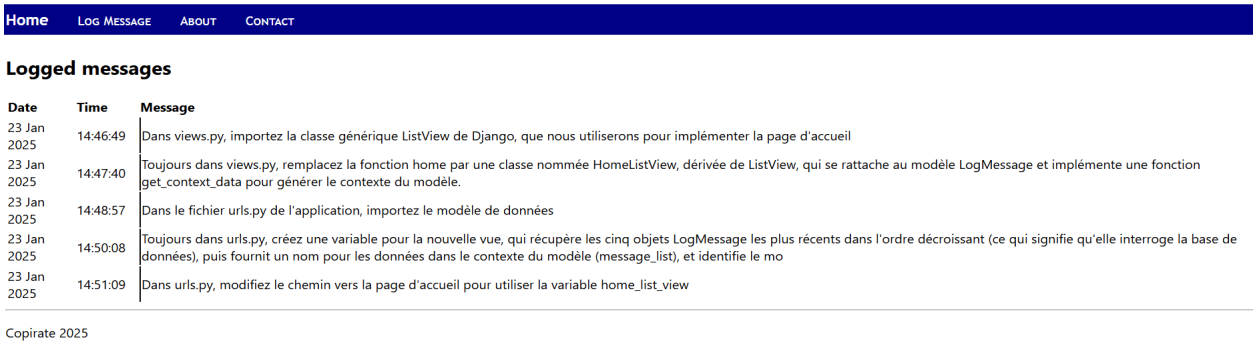
Projet 0 : Django Blog !

Objectifs

Vous allez créer un blog et afficher la liste des messages. Voici un aperçu du visuel.



L'objectif principal du projet est de vous faire découvrir par vous même la classe 🧙générique view proposée par django.

En effet, vous pouvez écrire la vue de la liste de messages comme une fonction de vue ordinaire interrogeant la base de données pour récupérer tous les messages. Ensuite, vous appelez la méthode render() pour passer les messages au template.

Avec les vues génériques, plus besoin de préciser tout cela, il suffit d'écrire

```
1. class MessageListView(ListView):
2.     model = Message
```

🪄Django s'occupe du reste, il va chercher un template dans message/message_list.html en lui passant comme context 'message_list'

Projet 0 : Django Blog !

Fiche technique

Voici un projet de création de tâches !

📄 Django : Task generic

Contraintes

Dans ce projet, vous devez utiliser une vue de liste générique basée sur une classe (ListView).

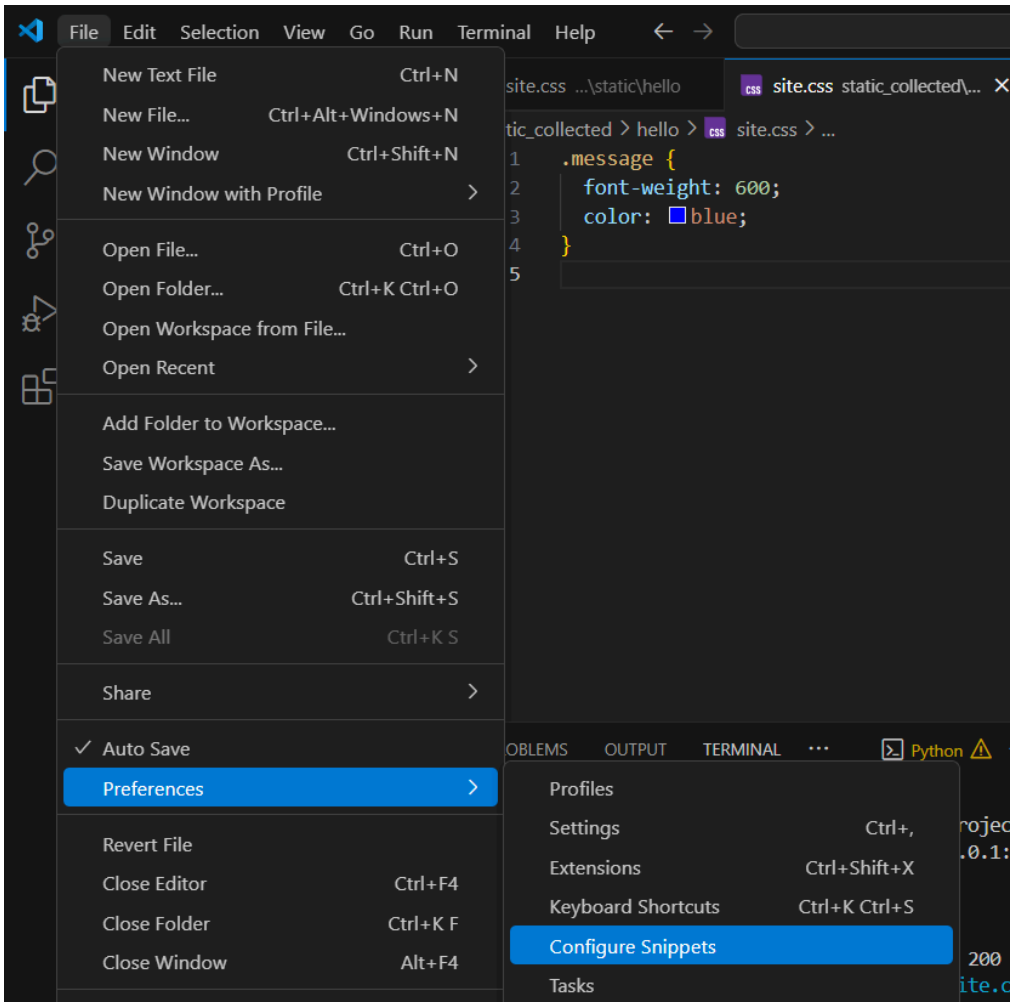
🚀Vous allez découvrir que la vue générique implémente la plupart des fonctionnalités dont vous aurez besoin.

Mise en bouche

Vous connaissez les snippets ?

Dans VS Code, sélectionnez le menu File

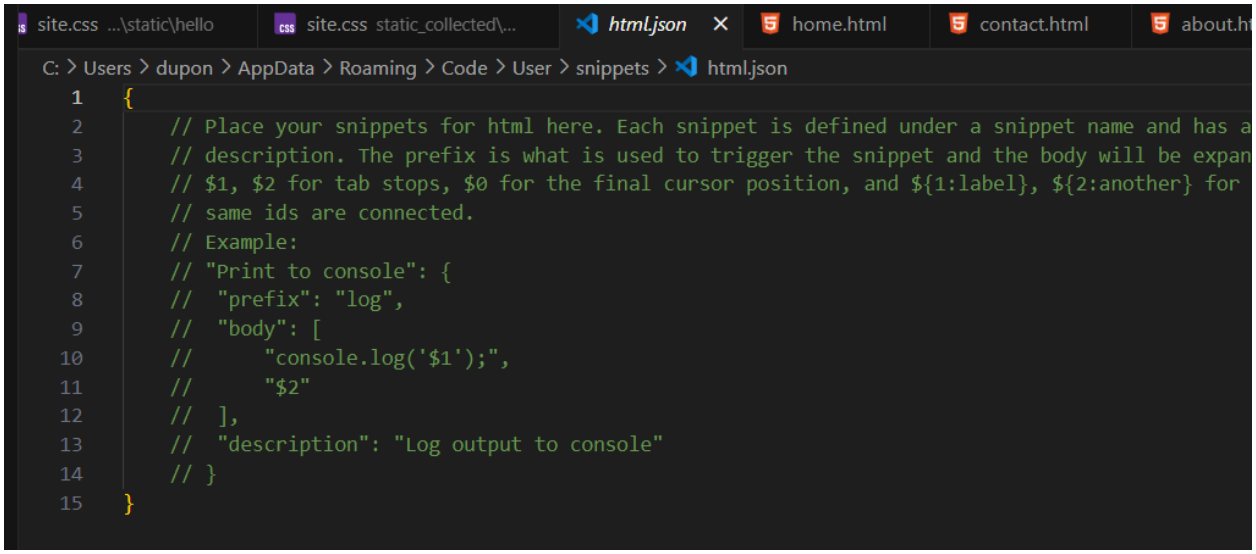
puis sélectionnez **Préférences >configure snippets.**



Dans la liste qui s'affiche, **sélectionnez html**. (L'option peut apparaître sous la forme "html.json" dans la section des extraits existants de la liste si vous avez créé des extraits précédemment).

Une fois que VS code a ouvert le fichier html.json,

Projet 0 : Django Blog !



Ajoutez le code ci-dessous à l'intérieur des accolades existantes.

```
{
    "Django: template extending layout.html": {
        "prefix": "djextlayout",
        "body": [
            "{% extends \"hello/layout.html\" %}",
            "{% block title %}",
            "${1:${TM_FILENAME_BASE}}",
            "{% endblock %}",
            "{% block content %}",
            "$0",
            "{% endblock %}"
        ],
        "description": "Boilerplate template that extends layout.html"
    }
}
```

Projet 0 : Django Blog !

🥷C'est quoi tout ces **$**. Lorsque vous appuyez sur la touche **Tab** vous naviguer dans les espaces réservés par les $i.

lig.7 : ${1 :...} est un espace réservé avec un index de 1, ce qui signifie qu'il s'agit de la première position à laquelle le curseur se rendra lors de l'insertion de l'extrait. Le 1 indique l'ordre dans lequel le curseur se déplacera dans les espaces réservés lorsque vous appuierez sur la touche Tab.

Lig. 7 : ${1:${TM_FILENAME_BASE}} fixe la valeur par défaut de l'espace réservé au nom de base du fichier. Génial !

Lig.10 : $0 est un espace réservé spécial qui représente la position finale du curseur après que tous les autres espaces réservés ont été remplis.

Lorsque vous appuyez sur la touche Tab pour naviguer dans les espaces réservés, $0 est la dernière position à laquelle le curseur se rendra.

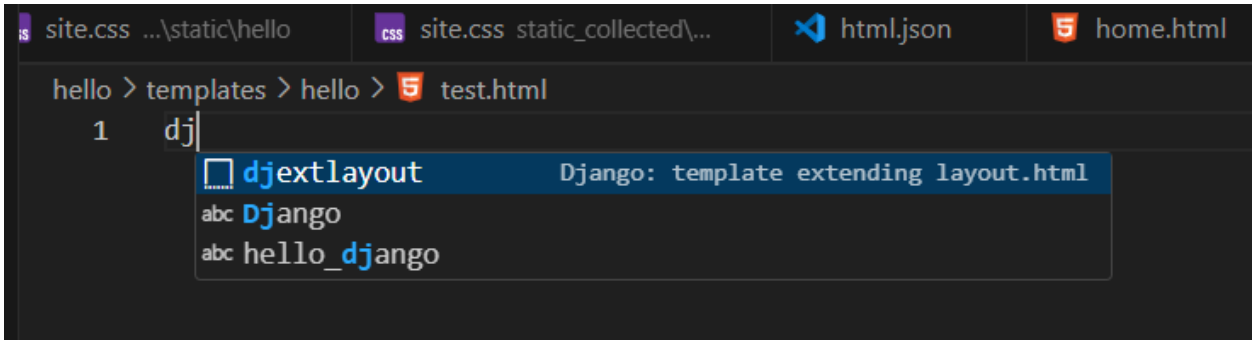
🪛Il vous suffit d'enregistrer le fichier html.json (Ctrl+S).

Test :

Créez un fichier  test.html.

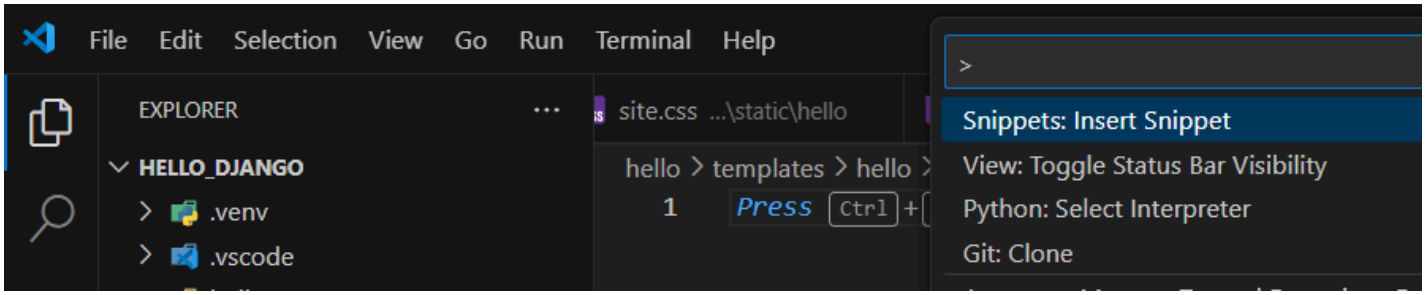
🥷Désormais, lorsque vous commencez à taper le préfixe du snippet, tel que dj, VS Code propose le snippet en tant qu'option d'autocomplétion.

Projet 0 : Django Blog !



🪛Sélectionnez **djextlayout** et remarquer que la place du curseur dans le code ($0)

Vous pouvez également utiliser la commande **Snippets: insert**



🚀En savoir plus sur les $i : VStudio code snippet

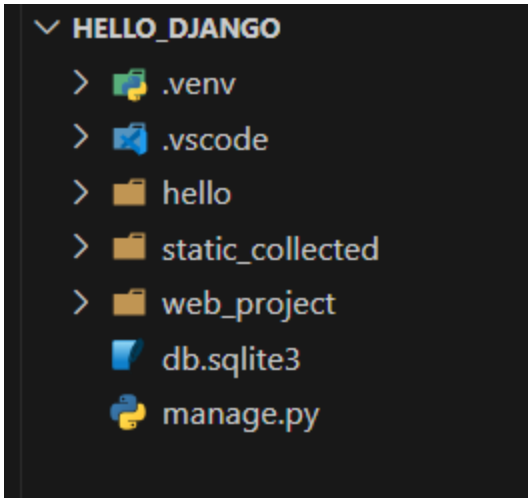
Projet

Mise en place

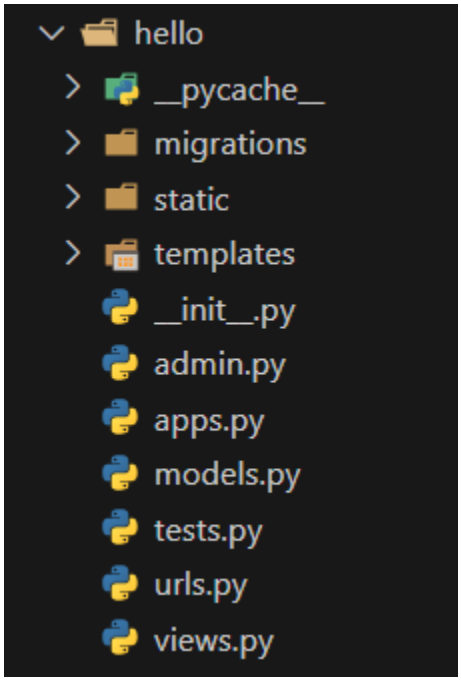
Mettez en place un projet **web_projet**[1] avec une application **hello.**

[1] Ignorez pour l'instant le répertoir static_collected

Projet 0 : Django Blog !



Let's start



🪛Mettez en place les templates

Pour guide, voici le fichier hello\templates\layout.html

layout.html

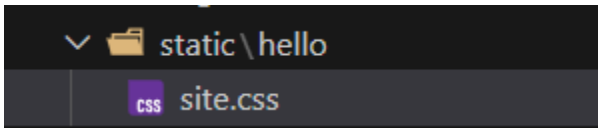
Projet 0 : Django Blog !

```
1.  <!DOCTYPE html>
2.  <html>
3.    <head>
4.      <meta charset="utf-8" />
5.      <title>{% block title %}My site{% endblock %}</title>
6.      {% load static %}
7.      <link
8.        rel="stylesheet"
9.        type="text/css"
10.       href="{% static 'hello/site.css' %}"
11.     />
12.   </head>
13.
14.   <body>
15.     <div class="navbar">
16.       <a href="{% url 'home' %}" class="navbar-brand">Home</a>
17.       {% comment %} if you want to comment something in Django
          template, you can use this syntax
18.
19.       {% endcomment %}
20.       <a href="{% url 'about' %}" class="navbar-item">About</a>
21.       <a href="{% url 'contact' %}" class="navbar-item">Contact</a>
22.     </div>
23.
24.     <div class="body-content">
25.       {% block content %} {% endblock %}
26.       <hr />
27.       <footer>
```

```
28.         <p>Copirate 2025</p>
29.       </footer>
30.     </div>
31.   </body>
32. </html>
```

CSS

Voici le code du fichier hello\static\hello\site.css



site.css

```
1.  /* Variables */
2.  :root {
3.    --navbar-bg-color: darkblue;
4.    --navbar-font-family: "Trebuchet MS", "Lucida Sans Unicode", "Lucida
      Grande", "Lucida Sans", Arial, sans-serif;
5.    --body-font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
6.    --navbar-text-color: white;
7.    --navbar-padding: 8px 5px;
8.    --navbar-brand-font-size: 1.2em;
9.    --navbar-brand-font-weight: 600;
10.   --navbar-item-margin-left: 30px;
11.   --body-content-padding: 5px;
12. }
13.
14. /* Navbar Styles */
```

```
15..navbar {
16.  background-color: var(--navbar-bg-color);
17.  font-size: 1em;
18.  font-family: var(--navbar-font-family);
19.  color: var(--navbar-text-color);
20.  padding: var(--navbar-padding);
21.}
22.
23..navbar a {
24.  text-decoration: none;
25.  color: inherit;
26.}
27.
28..navbar-brand {
29.  font-size: var(--navbar-brand-font-size);
30.  font-weight: var(--navbar-brand-font-weight);
31.}
32.
33..navbar-item {
34.  font-variant: small-caps;
35.  margin-left: var(--navbar-item-margin-left);
36.}
37.
38./* Body Content Styles */
39..body-content {
40.  padding: var(--body-content-padding);
41.  font-family: var(--body-font-family);
42.}
```

Lig. 1 : La définition de variables faciliterait la modification des valeurs sur l'ensemble du code.

Test

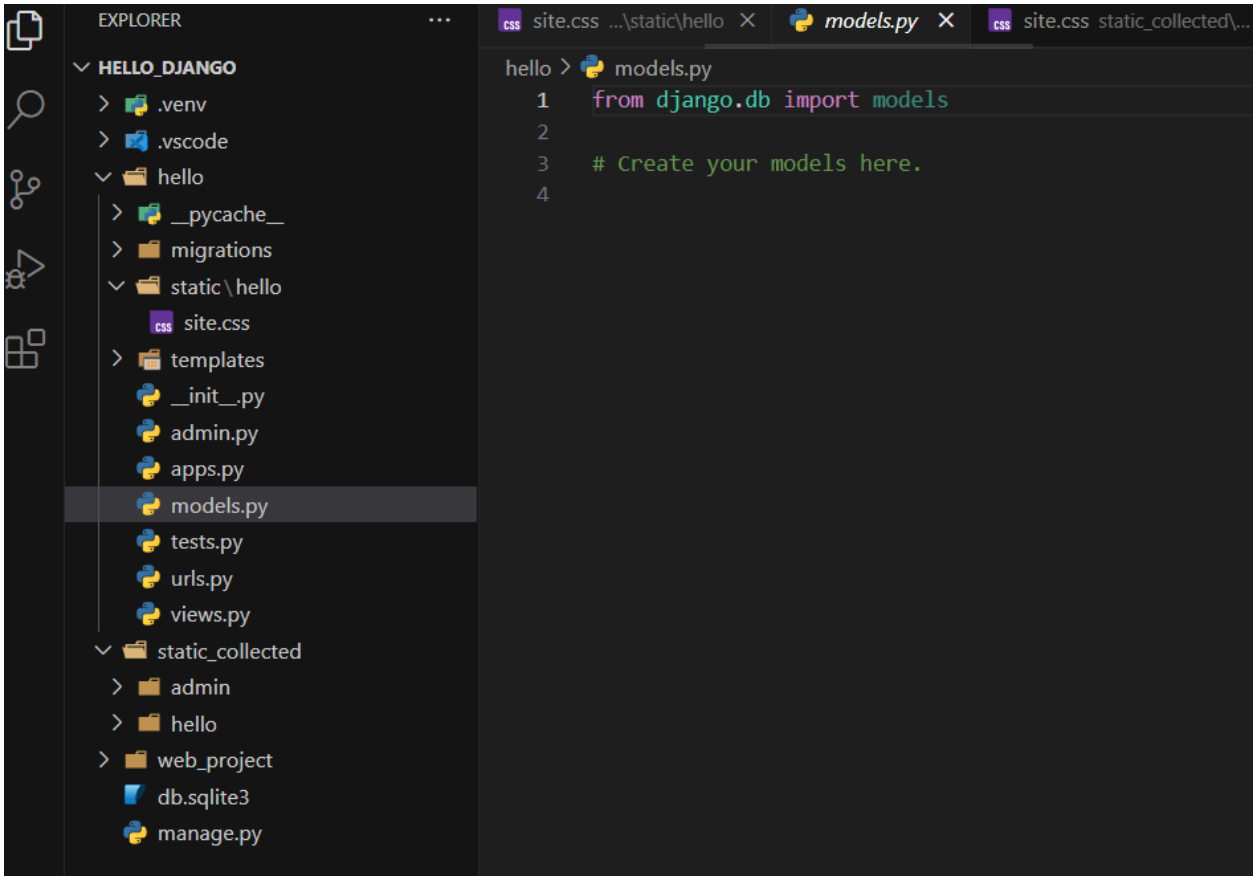
Voici le résultat attendu à l'adresse : http://127.0.0.1:8000/home/

Phase 1 du projet

Ajout de la base de données

Avec Django, vous travaillez avec votre base de données presque exclusivement par le biais des modèles que vous définissez dans le code.

Projet 0 : Django Blog !



Vous allez définir le modèle du blog. Je vous rappelle que vous devrez ensuite exécuter **python manage.py makemigrations** pour générer des scripts dans le dossier migrations qui migrent la base de données de son état actuel vers le nouvel état, puis **python manage.py migrate** pour appliquer les scripts à la base de données actuelle.

Modèle

Compléter dans le modèle de hello\models.py les deux !!.

models.py

```
1. from django.db import models
2. from django.utils import timezone
3.
4. class LogMessage(models.Model):
5.     message = !!
6.     log_date = !!
7.
8.     def __str__(self):
9.         """Returns a string representation of a message."""
10.        date = timezone.localtime(self.log_date)
11.        return f"'{self.message}' logged on {date.strftime('%A, %d %B, %Y at %X')}"
12.
```

Utiliser la base de données à travers les modèles

Dans le dossier hello, créez un nouveau fichier  hello\forms.py, qui définit un formulaire Django contenant un champ tiré du modèle de données, LogMessage.

Complétez le code !!.

 forms.py

```
1. from django import forms
2. from hello.models import LogMessage
3.
```

```
4. class LogMessageForm(forms.ModelForm):
5.     class Meta:
6.         model = !!
7.         fields = !!    # NOTE: the trailing comma is required
```

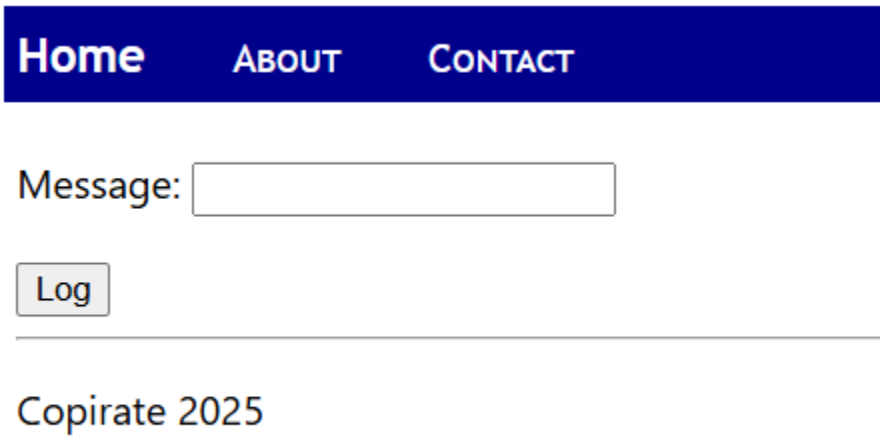
Lig. 5 : Django définit une classe Meta qui permet d'indiquer différentes méta-données comme ici le modèle, les champs à afficher …

Dans le dossier templates/hello, créez un nouveau fichier nommé hello\templates\hello\log_message.html

log_message.html

```
1. {% extends "hello/layout.html" %}
2. {% block title %}
3. log_message
4. {% endblock %}
5. {% block content %}
6. <form method="POST" class="log-form">
7. !!
8. !!
9. !!
10. </form>
11. {% endblock %}
```

Lig.7-9 : Il doit comporter le formulaire et un bouton "log"



Modifiez le template pour avoir dans la barre de navigation 

Ajoutez dans le fichier hello\views.py le code pour

def log_message(request):

Projet 0 : Django Blog !

Ajoutez dans le fichier hello\urls.py le code pour le chemin sur **views.log_message**

Modifiez la page d'accueil pour afficher les messages enregistrés.

🆘On pourra utiliser :

```
all_messages = LogMessage.objects.all()
has_messages = all_messages.exists()
```

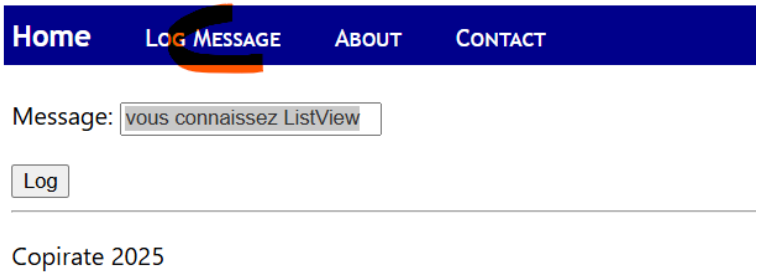
que l'on passe au template.

On signale d'abord l'absence de message.

Test

Ajoutez un message !

Projet 0 : Django Blog !



Vous devez afficher la liste des messages !

Bon, nous avions déjà vu la plupart du code à réaliser. Nous allons découvrir que django à partir du modèle nous permet en fait d'afficher l'ensemble des données (ou une donnée en particulier) avec un minimum de code.

Encore une fois, tout est dans votre définition du modèle !

💥Le reste, Django peut le gérer pour vous. En effet, si vous aviez défini le modèle Book(ou Airport …), il est évident que vous voudriez afficher les livres (ou les aéroports …).

Projet 0 : Django Blog !

🚀Etude de listView

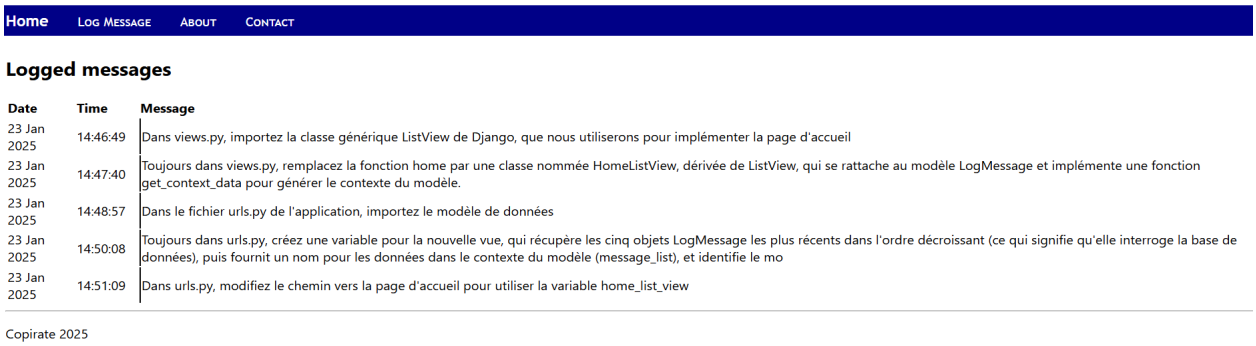
Vous allez modifier entièrement votre code. Pour cela, vous devez étudier et utiliser une vue de liste générique basée sur une classe (ListView).

La vue générique interroge la base de données pour obtenir tous les enregistrements du modèle spécifié (ici Logmessage), puis rendra un template.

Comme la vue générique implémente déjà la plupart des fonctionnalités dont vous avez besoin et qu'elle suit les meilleures pratiques de Django, vous pouvez créer une vue de liste plus robuste avec moins de code, moins de répétitions et, au final, moins de maintenance.

🪛Commencez par étudier `Generic display views | Django documentation`

Voici sans plus de détails, les étapes à réaliser, elles sont affichées sous la forme d'un message du blog.



La figure précédente vous montre le résultat attendu pour ce projet.