Université Assane SECK de Ziguinchor

Unité de Formation et de Recherche des Sciences et Technologies

Département d'Informatique

Développement Back-end avec Django - bis

Licence 2 en Ingénierie Informatique

Avril 2022

Papa-alioune.cisse@univ-zig.sn

Résumé : À la différence du Développement Front-End, qui concerne l'aspect visuel et ergonomique d'un site web, le Développement Back-End se penche sur les aspects techniques et fonctionnels du développement d'un site web dynamique. Il s'agit du développement de l'ensemble des fonctionnalités "invisibles" d'un site web (le serveur, le code applicatif, la base de données, etc.) qui sont indispensables au bon fonctionnement d'un site.
Ainsi, on peut dire que les Développeurs Back End travaillent sur la partie immergée de l'iceberg, alors que les Développeurs Front End se chargent essentiellement de la partie visible.
Ce chapitre est une continuité du précédent. Il aborde la partie sur les « Templates Django » et la personnalisation du système d'authentification de Django.

1 - Les Templates Django

1.1 - Ajout d'un template (une page HTML)

Il faut commencer par ajouter un dossier nommé « templates » à la racine du projet et à la racine de chaque application. Dans le dossier « templates » de chaque application, il faut ensuite ajouter un dossier qui porte le même nom que l'application et qui doit contenir désormais toutes les pages HTML de l'application.

Il faut ensuite en informer Django dans le fichier de configuration : paramètre « DIRS » de la variable « TEMPLATES » du fichier « setting.py ».

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [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',
            ],
        },
    },
]
```

Il faut importer « os » en ajoutant « import os » en haut du fichier « settings.py ».

Vous pouvez maintenant ajouter la page « home.html » dans le dossier « templates/mon_application1 » et y mettre un contenu pour voir le résultat en tapant l'url suivante : 127.0.0.1 :8000/mon_application1/.

1.2 - Mise en place d'un gabarit de page

Généralement, dans un site web, la plupart des pages ont la même configuration : des parties communes à toutes les pages (header, footer, menu, etc.) et des parties propres à chaque page (title, contenu propre de chaque page, …).

Django offre la possibilité de définir une page gabarit qui contient tout le code commun à toutes les pages et qui contient également des trous (block) laissés vides et nommés, que chaque page HTML qui étend la page gabarit remplisse pour ajouter son code propre.

Cette page gabarit (que nous appelons ici « base.html ») doit être placée dans le dossier « templates » du projet.

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>
                {% block title %} Titre par défaut {% endblock %}
        </title>
</head>
<body>
        {% block content %} Contenu par défaut {% endblock %}
</body>
</html>
```

La page « home.html » de l'application « mon_application1 » qui étend la page gabarit peut ressembler à :

```
{% extends "base.html" %}
{% block title %} Accueil Mon application {% endblock %}
{% block content %}
        <p>Mon contenu propre</p>
{% endblock %}
```

1.3 - Ajout des fichiers statiques

Pour ajouter des fichiers statiques (css, js, images, …), il faut d'abord ajouter un dossier nommé « static » à la racine du projet pour contenir tous les fichiers statiques du projet, ensuite en informer Django dans le fichier de configuration en y ajoutant ces deux variables :

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.0/howto/static-files/
```

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

2 - Personnalisation du système d'authentification de Django

2.1 - Introduction

Nous créons une application à part entière pour la gestion de l'authentification des utilisateurs. Pour cela, nous allons d'abord personnaliser le modèle utilisateur de Django, ensuite personnaliser les différents écrans relatifs à l'authentification : connexion, changement de mot de passe, inscription, etc.

2.2 - Personnalisation du modèle utilisateur

Ajouter l'application nommée « accounts » pour gérer le système d'authentification.

Il faut ensuite informer Django qu'on change de modèle utilisateur en ajoutant dans « settings.py » la variable « AUTH_USER_MODEL » comment suit :

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

    'gedt',
    'accounts'
]

AUTH_USER_MODEL = 'accounts.CustomUser'
```

Cela veut dire que le modèle utilisateur est désormais la classe « CustomUser » de l'application « accounts ».

Il faut maintenant ajouter la classe « CustomUser » dans le fichier « models.py » de « accounts ».

Pour être un modèle utilisateur, la classe « CustomUser » peut hériter de la classe Django de base appelée « AbstractUser ».

```
from django.db import models
from django.contrib.auth.models import AbstractUser

class UserProfil(models.Model):
        code = models.CharField(max_length=50, verbose_name="Code profil")
        libelle = models.CharField(max_length=150, verbose_name="Libéellé profil")

class CustomUser(AbstractUser):
    #profil = models.ForeignKey(UserProfil, on_delete=models.CASCADE, verbose_name
="Profil")
    cni = models.CharField(max_length=25, verbose_name="CNI")
    telephone = models.CharField(max_length=50, verbose_name="Téléphone", null=Tru
e, blank=True)
    adresse = models.CharField(max_length=150, verbose_name="Adresse", null=True,
blank=True)

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

Nous avons aussi besoin de personnaliser les formulaires d'ajout et de modification d'un utilisateur. Il faut, pour cela, ajouter le fichier « forms.py » dans la racine de l'application « accounts » avec le code suivant :

```
from django import forms
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from .models import CustomUser

class CustomUserCreationForm(UserCreationForm):

    class Meta:
        model = CustomUser
        fields = ('username', 'email')

class CustomUserChangeForm(UserChangeForm):

    class Meta:
        model = CustomUser
```

```
        fields = ('username', 'email')
```

Nous avons finalement besoin de mettre à jour « admin.py » pour prendre en compte le nouveau modèle utilisateur :

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin

from .forms import CustomUserCreationForm, CustomUserChangeForm
from .models import CustomUser, UserProfil

class CustomUserAdmin(UserAdmin):
    add_form = CustomUserCreationForm
    form = CustomUserChangeForm
    model = CustomUser
    list_display = ['email', 'username',]
    fieldsets = UserAdmin.fieldsets + (
    (None, {'fields': ('cni','telephone','adresse')}),
    )

admin.site.register(CustomUser, CustomUserAdmin)

@admin.register(UserProfil)
class UserProfilAdmin(admin.ModelAdmin):
    list_display = ('code', 'libelle')
```

Ajoutons maintenant le champ « profil » dans le modèle utilisateur comme suit :

```
from django.db import models
from django.contrib.auth.models import AbstractUser

class UserProfil(models.Model):
        code = models.CharField(max_length=50, verbose_name="Code profil")
        libelle = models.CharField(max_length=150, verbose_name="Libéellé profil")
        def __str__(self):
                return self.libelle

class CustomUser(AbstractUser):
    profil = models.ForeignKey(UserProfil, on_delete=models.CASCADE, verbose_name=
"Profil", default=1)
    cni = models.CharField(max_length=25, verbose_name="CNI")
```

```
    telephone = models.CharField(max_length=50, verbose_name="Téléphone", null=Tru
e, blank=True)
    adresse = models.CharField(max_length=150, verbose_name="Adresse", null=True,
blank=True)

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

A présent, vous pouvez faire les migrations et recréer le super user.

2.3 - Mise en place du système d'authentification

Nous commençons par ajouter un dossier nommé « registration » dans le dossier « templates » du projet qui va contenir toutes les pages HTML relatives à l'authentification selon les fonctionnalités à implémenter (login.html, logged_out.html, signup.html, password_change_done.html, password_change_form.html, password_reset_complete.html, password_reset_confirm.html, password_reset_done.html, password_reset_email.html, password_reset_form.html).

Ajouter aussi une 2ème page gabarit différent de la page « base.html » pour les pages de l'authentification. Elle sera ajoutée directement dans le dossier « templates » du projet au même niveau que la page « base.html » et appelée « base_registration.html ».

Le code de certaines de ces pages est donné ci-dessous :

```
<!-- templates/base_registration.html -->
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>{% block title %}Django Auth{% endblock %}</title>
</head>
<body>
  <main>
    {% block content %}
    {% endblock %}
  </main>
</body>
</html>
```

```
<!-- templates/registration/login.html -->
{% extends 'base.html' %}

{% block title %}Log In{% endblock %}

{% block content %}
<h2>Log In</h2>
<form method="post">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Log In</button>
</form>
{% endblock %}
```

Il faut ensuite placer les liens de connexion (Déconnexion) et de déconnexion (Se connecter) dans les pages appropriés.

Nous allons à présent ajouter les « routes » vers le système d'authentification dans le fichier « urls.py » du projet :

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path(mon_application1/', include(mon_application1.urls')),
    path('accounts/', include('accounts.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
]
```

Maintenant, il faut ajouter ces 2 paramétrages en bas dans le fichier « settings.py » pour dire à Django où se rediriger à la connexion et la déconnexion d'un utilisateur.

```
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/'
```