Université Assane SECK de Ziguinchor

Unité de Formation et de Recherche des Sciences et Technologies

Département d'Informatique

Développement d'applications avec Django

Licence 2 en Ingénierie Informatique

Avril 2022

©Papa Alioune CISSE

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 « accueil.html » dans le dossier « templates/apptest » et y mettre un contenu pour voir le résultat en tapant l'url suivante : 127.0.0.1 :8000/apptest/.

Par exemple :

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <title>Candidature</title>
</head>
<body>
        {{ unObjet }}
</body>
</html>
```

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">
        <title>
                {% block title %} Titre par défaut {% endblock %}
        </title>
</head>
<body>
        {% block content %} Contenu par défaut {% endblock %}
</body>
</html>
```

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

```
{% extends "base.html" %}
{% block title %} Candidature {% endblock %}
{% block content %}
```

```
        <p>Mon contenu propre</p>
        {{ unObjet }}
{% 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_URL = '/static/'
STATICFILES_DIRS = (os.path.join(BASE_DIR, "static"),)
```

2 - Implémentation des fonctionnalités d'un cas d'étude : une application de recrutement

2.1 - Le Cas d'étude : une application de recrutement

Il s'agit d'une application comportant une page de CV pour présenter les CVs des développeurs issus de la L2I, une page de candidature avec un formulaire permettant à des potentiels recruteurs de contacter nos développeurs, en renseignant quelques informations en rapport avec nos critères de recrutement, des pages de gestion de candidatures pour que les recruteurs puissent gérer (visualiser, modifier, supprimer) leurs candidatures et des pages d'authentification et de gestion de comptes dans l'application.

Les CVs des développeurs doivent être ajoutés et administrés depuis l'espace d'administration de Django par l'administrateur du système.

L'application à développer et qui est destinée aux recruteurs doit donc offrir les fonctionnalités suivantes : ***visualiser de la liste exhaustive des développeurs*** ; ***visualiser le CV d'un développeur*** ; ***demander le recrutement d'un développeur*** en renseignant le formulaire de recrutement ; ***gestion des demandes de recrutement*** permettant aux recruteurs de modifier, supprimer et visualiser leurs demandes de recrutements. Pour cela, nous allons conserver notre application « apptest » et y ajouter ces fonctionnalités.

2.2 - Personnalisation du système d'authentification de Django

Django intègre un système d'authentification sous forme d'application qu'il est possible d'utiliser et personnaliser pour gérer 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.

Personnalisation du modèle utilisateur

Ajouter une 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 » (à définir) 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 CustomUser(AbstractUser):
    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

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)
```

A présent, vous pouvez faire les migrations et recréer le super user, puisque les tables sont recréées.

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 dans le dossier « codes sources/registration » du cours.

Il faut ensuite placer les liens de connexion (Déconnexion), de déconnexion (Se connecter) et de changement de mot de passe (Se connecter) dans la page « base.html » à la place appropriée.

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('apptest', include('apptest.urls')),
    path('', include('apptest.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 = '/'
```

2.4 - Administration des CVs des développeurs et des données de paramétrage depuis django admin

C'est à l'administrateur du système, depuis l'espace d'admin de django, d'ajouter les CVs des développeurs, de les modifier, de les supprimer, etc. C'est à lui aussi d'administrer les données utilisées par les futurs recruteurs pour faire des demandes de recrutement.

Ainsi, nous aurons principalement 2 entités : l'entité « CV » et l'entité « DemandeRecrutement ».

- Un « CV » comporte les éléments suivants : prénom, nom, téléphone, adresse, profession, mail et date de naissance du candidat ; ses expériences professionnelles, ses formations, ses éducations et ses langues et hobbies.
  - ✓ Une « expérience professionnelle », comme une « formation » et une « éducation » sont aussi des entités comportant toutes les informations suivantes : titre, détails, date début, date de fin et durée.
  - ✓ Une « langue et hobbie » est aussi une entité comportant les éléments titre et détails.
- Une « DemandeRecrutement » comporte les éléments suivants : l'utilisateur (recruteur) ayant fait la demande ; le candidat (CV) qui est demandé ; nom, adresse et ville de l'entreprise demandeuse ; poste (DG, DG adjoint, collaborateur, employé), temps de travail (mi-temps, plein-temps) et date démarrage proposés ; un commentaire sur la demande.
  - ✓ « Ville », « Poste », « TempsDeTravail » sont aussi des entités comportant chacun les éléments "code" et "nom" et dont les données sont paramétrées depuis l'espace d'administration.

Ces différentes entités sont implémentées dans le modèle « fichier models.py » de « apptest » que vous trouverez dans le dossier (codes soures/apptest) du cours.

Vous avez aussi dans ce dossier les fichiers « db_access.py » et « utilities.py » contenant respectivement un ensemble de fonctions relatives aux traitements sur la base de données (ajout, lecture, modification et suppression des données) et un ensemble de fonctions utiles comme le calcul d'un âge, …