Application de e-learning avec Django

Introduction

Django est un framework Web Python de haut niveau qui encourage un développement rapide et une conception propre et pragmatique. Construit par des développeurs expérimentés, il prend en charge une grande partie des tracas du développement Web, vous pouvez donc vous concentrer sur l'écriture de votre application sans avoir à réinventer la roue. C'est gratuit et open source.

Prérequis

1. Installer python
2. Installer un éditeur de texte
3. Creation d’un environment virtuel

Un environnement virtuel Python est un environnement logiciel qui permet d'avoir plusieurs installations de Python avec des versions et des paquets différents sur un même ordinateur. Cela peut être utile lorsque vous travaillez sur plusieurs projets qui ont des exigences de versions ou de paquets différentes pour Python.

Pour utiliser un environnement virtuel Python, vous devez d'abord l'installer en utilisant pip, le gestionnaire de paquets pour Python. Ouvrez un terminal et exécutez la commande suivante:

pip install virtualenv

Une fois virtualenv installé, vous pouvez créer un nouvel environnement virtuel en utilisant la commande suivante:

virtualenv env

Cela créera un nouveau répertoire appelé "env" qui contient une nouvelle installation de Python et les outils nécessaires pour gérer les paquets.

Pour activer l'environnement virtuel, utilisez la commande suivante:

Sous Linux:

source env/bin/activate

sous Windows:

.\env\Scripts\activate

NB: pour les systemes windows, utilizer la commande suivante pour autoriser l’utilisateur courant

Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Unrestricted

Vous verrez que le nom de l'environnement virtuel s'affiche entre parenthèses à gauche de votre invite de commande, comme ceci:

(env) $

Vous pouvez maintenant utiliser Python et pip dans cet environnement virtuel. Pour installer un paquet, utilisez la commande suivante:

pip install nom\_du\_paquet

Pour quitter l'environnement virtuel, utilisez la commande de désactivation:

Deactivate

Vous pouvez créer autant d'environnements virtuels que vous le souhaitez et les utiliser séparément en activant l'environnement virtuel souhaité. Cela vous permet de travailler sur plusieurs projets qui ont des exigences de versions ou de paquets différents pour Python sans avoir à gérer manuellement les versions et les paquets.

1. Installation de Django

Voici comment démarrer avec Django en utilisant Python et en suivant les étapes ci-dessous:

1. Assurez-vous d'avoir Python installé sur votre ordinateur. Django nécessite Python 3.6 ou une version ultérieure.
2. Installez Django en utilisant pip, le gestionnaire de paquets pour Python. Ouvrez un terminal et exécutez la commande suivante:

pip install Django Pillow

python -m pip install --upgrade pip

1. Une fois Django installé, vous pouvez créer un nouveau projet en exécutant la commande suivante:

django-admin startproject eduplus

Cela créera un nouveau répertoire appelé "eduplus" avec les fichiers et répertoires de base nécessaires pour un projet Django.

1. Accédez au répertoire du projet:

cd eduplus

1. Exécutez le serveur de développement:

python manage.py runserver

Cela démarrera le serveur de développement à l'adresse <http://127.0.0.1:8000/>. Vous devriez voir une page de bienvenue.

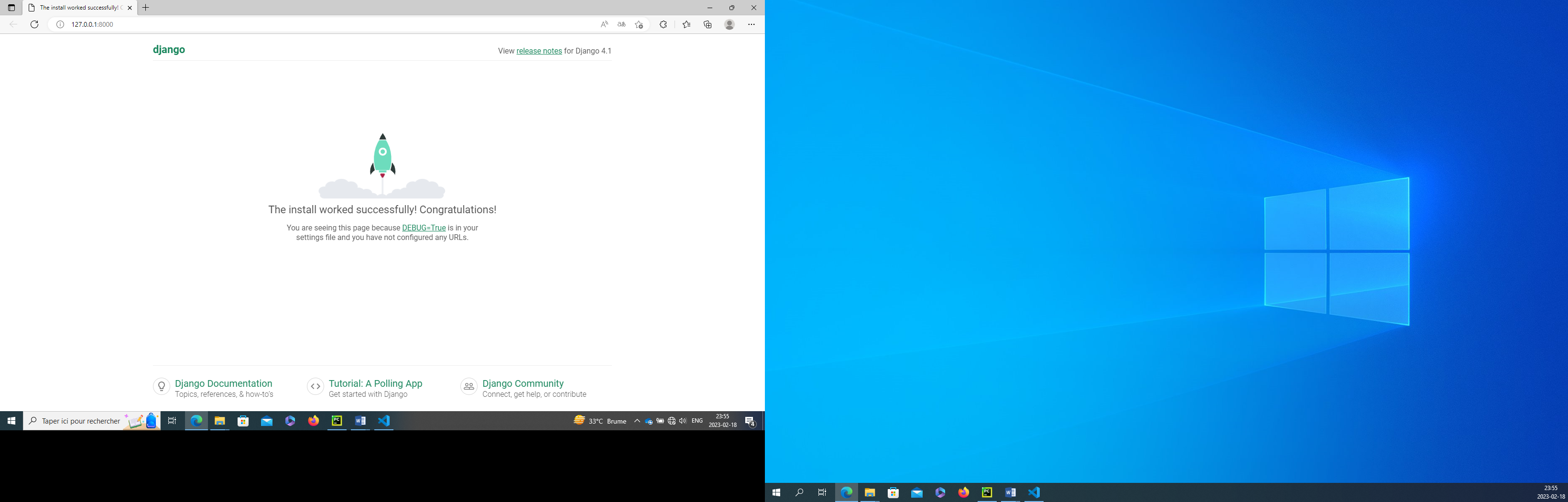


Figure 1: Capture de la page de bienvenue

1. Créer des applications

django-admin startapp core

django-admin startapp accounts

django-admin startapp blog

django-admin startapp shop

1. configurer des applications

Dans le fichier eduplus\settings.py

# Application definition

DEFAULT\_APPS=[

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

]

TIERS\_APPS=[

    'rosetta',

]

PROJECT\_APPS=[

    'core',

    'blog',

    'shop',

    'accounts',

]

 # Application definition

INSTALLED\_APPS = DEFAULT\_APPS + TIERS\_APPS + PROJECT\_APPS

1. Configuration des fichiers static

Commençons par crée le dossier static a la racine du projet et importons la bibliothèque os

import os

STATICFILES\_DIRS = (

os.path.join(BASE\_DIR, 'static'),

)

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

1. Configuration des fichiers media

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

Le template

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [str(BASE\_DIR.joinpath("templates"))],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.media',

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

1. Bases de données

Voici comment créer une base de données PostgreSQL et l'utiliser avec Django:

1. Assurez-vous d'avoir PostgreSQL installé sur votre ordinateur. Vous pouvez télécharger et installer PostgreSQL à partir du site web officiel: <https://www.postgresql.org/>
2. Créez un nouvel utilisateur et une nouvelle base de données dans PostgreSQL en utilisant la ligne de commande ou un outil de gestion tel que pgAdmin. Voici un exemple de commande pour créer un utilisateur appelé "eduplususer" et une base de données appelée "db\_eduplus":
3. Installer les paquets necessaires

pip install psycopg2-binary

python.exe -m pip install --upgrade pip

CREATE DATABASE db\_eduplus;

CREATE USER eduplususer WITH PASSWORD 'xxxxxxx';

ALTER ROLE eduplususer SET client\_encoding TO 'utf8';

ALTER ROLE eduplususer SET default\_transaction\_isolation TO 'read committed';

ALTER ROLE eduplususer SET timezone TO 'UTC';

GRANT ALL PRIVILEGES ON DATABASE db\_eduplus TO eduplususer;

\q

1. Modifiez les paramètres de la base de données dans le fichier "settings.py" de votre projet Django. Recherchez la section DATABASES et modifiez les paramètres comme suit:

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql',

        'NAME': 'db\_eduplus',

        'USER': 'eduplususer',

        'PASSWORD': 'xxxxxxxxxx',

        'HOST': 'localhost',

        'PORT': '',

    }

}

Remplacez "xxxxxxxxx" par le mot de passe de l'utilisateur "eduplususer". Si vous avez configuré un hôte et un port différents pour la base de données, modifiez les paramètres HOST et PORT en conséquence.

1. Exécutez la commande "migrate" pour créer les tables de la base de données:

python manage.py migrate

Cela créera les tables de la base de données en utilisant les modèles Django.

1. L’internalisation et localisation

L'internationalisation et la localisation représentent les deux faces d'une même médaille. Ensemble, ils vous permettent de diffuser le contenu de votre application Web dans différents paramètres régionaux.

* **L'internationalisation** , représentée par i18n (18 est le nombre de lettres entre i et n), est le processus de développement de votre application afin qu'elle puisse être utilisée par différents paramètres régionaux. Ce processus est généralement géré par les développeurs.
* **La localisation** , représentée par l10n (10 est le nombre de lettres entre l et n), d'autre part, est le processus de traduction de votre application dans une langue et des paramètres régionaux particuliers. Ceci est généralement géré par des traducteurs.
* Rappelons que Django, via son [framework d'internationalisation](https://github.com/django/django/tree/main/django/utils/translation) , a été traduit dans plus de [100](https://docs.djangoproject.com/en/3.2/internals/contributing/) langues :
* Grâce au cadre d'internationalisation, nous pouvons facilement marquer les chaînes à traduire, à la fois dans le code Python et dans nos modèles. Il utilise la boîte à outils GNU [gettext](https://www.gnu.org/software/gettext/) pour générer et gérer un fichier de texte brut qui représente une langue connue sous le nom de [fichier de message](https://docs.djangoproject.com/en/3.2/topics/i18n/#term-message-file) . Le fichier de message se termine par .po comme extension. Un autre fichier est généré pour chaque langue une fois la traduction terminée, qui se termine par l' extension .mo . C'est ce qu'on appelle la traduction compilée.

## **Nomenclature**

**Django Internationalisation** = L'objectif de l'internationalisation et de la localisation est de permettre à une seule application web de proposer son contenu dans différentes langues et formats adaptés à l'audience

**USE\_I18N** = Si cette valeur est définie sur False, Django effectuera quelques optimisations afin de ne pas charger la machinerie de traduction

**USE\_L10N** = Un booléen qui spécifie si le formatage localisé des données sera activé par défaut ou non. S'il est défini sur True, par exemple, Django affichera les nombres et les dates en utilisant le format de la locale actuelle

Commençons par installer la boîte à outils [gettext .](https://www.gnu.org/software/gettext/)

## **Cadre d'internationalisation de Django**

Django est livré avec des paramètres d'internationalisation par défaut dans le fichier settings.py :

# Internationalization

# https://docs.djangoproject.com/en/4.0/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

Le premier réglage est le LANGUAGE\_CODE. Par défaut, il est défini sur l'anglais des États-Unis (en-us). Il s'agit d'un nom spécifique aux paramètres régionaux.

Ajoutons quelques paramètres supplémentaires pour compléter ceux existants :

from django.utils.translation import gettext\_lazy as \_

LANGUAGES = (

    ('en', \_('English')),

    ('fr', \_('French')),

    ('es', \_('Spanish')),

)

**Qu'est-ce qu'il se passe ici?**

1. Nous avons spécifié les langues dans lesquelles nous voulons que notre projet soit disponible. Si cela n'est pas spécifié, Django supposera que notre projet doit être disponible dans toutes ses langues prises en charge.
2. Ce paramètre [LANGUE](https://docs.djangoproject.com/en/3.2/ref/settings/#languages) comprend le code de langue et le nom de la langue. N'oubliez pas que les codes de langue peuvent être spécifiques aux paramètres régionaux, tels que « en-gb » ou génériques, tels que « en ».
3. En outre, gettext\_lazyest utilisé pour traduire les noms de langue au lieu d' gettextempêcher les importations circulaires. Vous devriez *presque* toujours utiliser [gettext\_lazy](https://docs.djangoproject.com/en/3.2/topics/i18n/translation/#lazy-translations) lorsque vous êtes dans la portée globale.

Ajouter django.middleware.locale.LocaleMiddlewareà la MIDDLEWAREliste des paramètres. Ce middleware devrait venir après le SessionMiddlewareparce LocaleMiddlewarequ'il a besoin d'utiliser les données de session. Il doit également être placé avant CommonMiddlewarecar le a CommonMiddlewarebesoin de la langue active pour résoudre les URL demandées. Par conséquent, l'ordre est très important.

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.locale.LocaleMiddleware', # new

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

Ce middleware est utilisé pour déterminer la langue actuelle en fonction des données de la requête.

Ajoutez un répertoire de chemin d'accès aux paramètres régionaux pour votre application où résideront les fichiers de messages :

LOCALE\_PATHS = [

BASE\_DIR / 'locale/',

]

Django examine le paramètre [LOCALE\_PATHS](https://docs.djangoproject.com/en/3.2/ref/settings/#std:setting-LOCALE_PATHS) pour les fichiers de traduction. Gardez à l'esprit que les chemins d'accès aux paramètres régionaux qui apparaissent en premier ont la priorité la plus élevée.

Vous devez créer le répertoire "locale" à l'intérieur de votre projet racine et ajouter un nouveau dossier pour chaque langue :

locale

├── en

├── es

└── fr

Ouvrez le shell et exécutez la commande suivante à partir de votre répertoire de projet pour créer un fichier de message .po pour chaque langue :

(env)$ django-admin makemessages --all --ignore=env

Vous devriez maintenant avoir :

locale

├── en

│ └── LC\_MESSAGES

│ └── django.po

├── es

│ └── LC\_MESSAGES

│ └── django.po

└── fr

└── LC\_MESSAGES

└── django.po

Prenez note de l'un des fichiers de message *.po :*

1. msgid: représente la chaîne de traduction telle qu'elle apparaît dans le code source.
2. msgstr: représente la traduction de la langue, qui est vide par défaut. Vous devrez fournir la traduction réelle pour toute chaîne donnée.

Actuellement, seuls les fichiers LANGUAGESde notre fichier settings.py ont été marqués pour traduction. Par conséquent, pour chacun msgstr des répertoires "fr" et "es", saisissez manuellement l'équivalent français ou espagnol du mot, respectivement. Vous pouvez modifier les fichiers .po à partir de votre éditeur de code habituel ; cependant, il est recommandé d'utiliser un éditeur conçu spécifiquement pour .po comme [Poedit](https://poedit.net/) .

Apportez les modifications suivantes :

locale\en\LC\_MESSAGES\django.po

#: .\eduplus\settings.py:120

msgid "English"

msgstr "Anglais"

#: .\eduplus\settings.py:121

msgid "French"

msgstr "Français"

#: .\eduplus\settings.py:122

msgid "Spanish"

msgstr "Espagnol"

Ensuite, compilons les messages en exécutant la commande suivante :

(env)$ django-admin compilemessages --ignore=env

Un fichier de message compilé .mo a été généré pour chaque langue :

locale

├── en

│ └── LC\_MESSAGES

│ ├── django.mo

│ └── django.po

├── es

│ └── LC\_MESSAGES

│ ├── django.mo

│ └── django.po

└── fr

└── LC\_MESSAGES

├── django.mo

└── django.po

## **Ajouter un préfixe de langue aux URL**

Avec le cadre d'internationalisation de Django, vous pouvez servir chaque version linguistique sous une extension d'URL différente. Par exemple, la version anglaise de votre site peut être diffusée sous /en/, la version française sous /fr/, etc. Cette approche rend le site optimisé pour les moteurs de recherche car chaque URL sera indexée pour chaque langue, ce qui à son tour sera mieux classé pour chaque langue. Pour ce faire, le framework d'internationalisation de Django doit identifier la langue actuelle à partir de l'URL demandée ; par conséquent, les LocalMiddlewarebesoins doivent être ajoutés dans le MIDDLEWAREcadre de votre projet, ce que nous avons déjà fait.

Ensuite, ajoutez la i18n\_patternsfonction à django\_lang/urls.py :

from django.conf.urls.i18n import i18n\_patterns

from django.contrib import admin

from django.urls import path

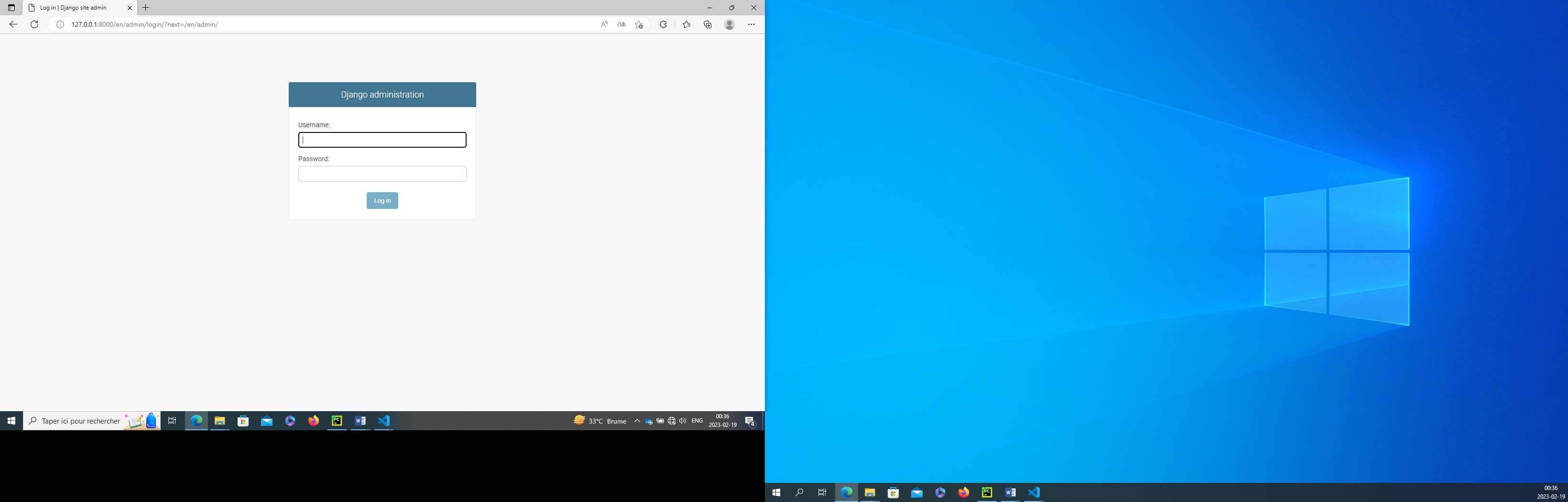
urlpatterns = i18n\_patterns(

    path('admin/', admin.site.urls),

)

Exécutez à nouveau le serveur de développement et accédez à <http://127.0.0.1:8000/> dans votre navigateur. Vous serez redirigé vers l'URL demandée, avec le préfixe de langue approprié. Jetez un oeil à l'URL dans votre navigateur ; il devrait maintenant ressembler à <http://127.0.0.1:8000/en/> .

Remplacez l'URL demandée par enou frpar es. Le titre devrait changer.



[Poedit](https://poedit.net/) vs **Rosetta**

[Poedit](https://poedit.net/) est un éditeur conçu spécifiquement pour les fichiers .po .

Rosetta est une application Django qui facilite le processus de traduction de projets Django.

Installer Rosetta

pip install django-rosetta

Ajoutez 'rosetta'aux INSTALLED\_APPS dans votre projetsettings.py

Ajoutez une entrée d'URL à votre projet urls.py, par exemple :

from django.conf import settings

from django.conf.urls import include, re\_path

if 'rosetta' in settings.INSTALLED\_APPS:

    urlpatterns += [

        re\_path(r'^rosetta/', include('rosetta.urls'))

    ]

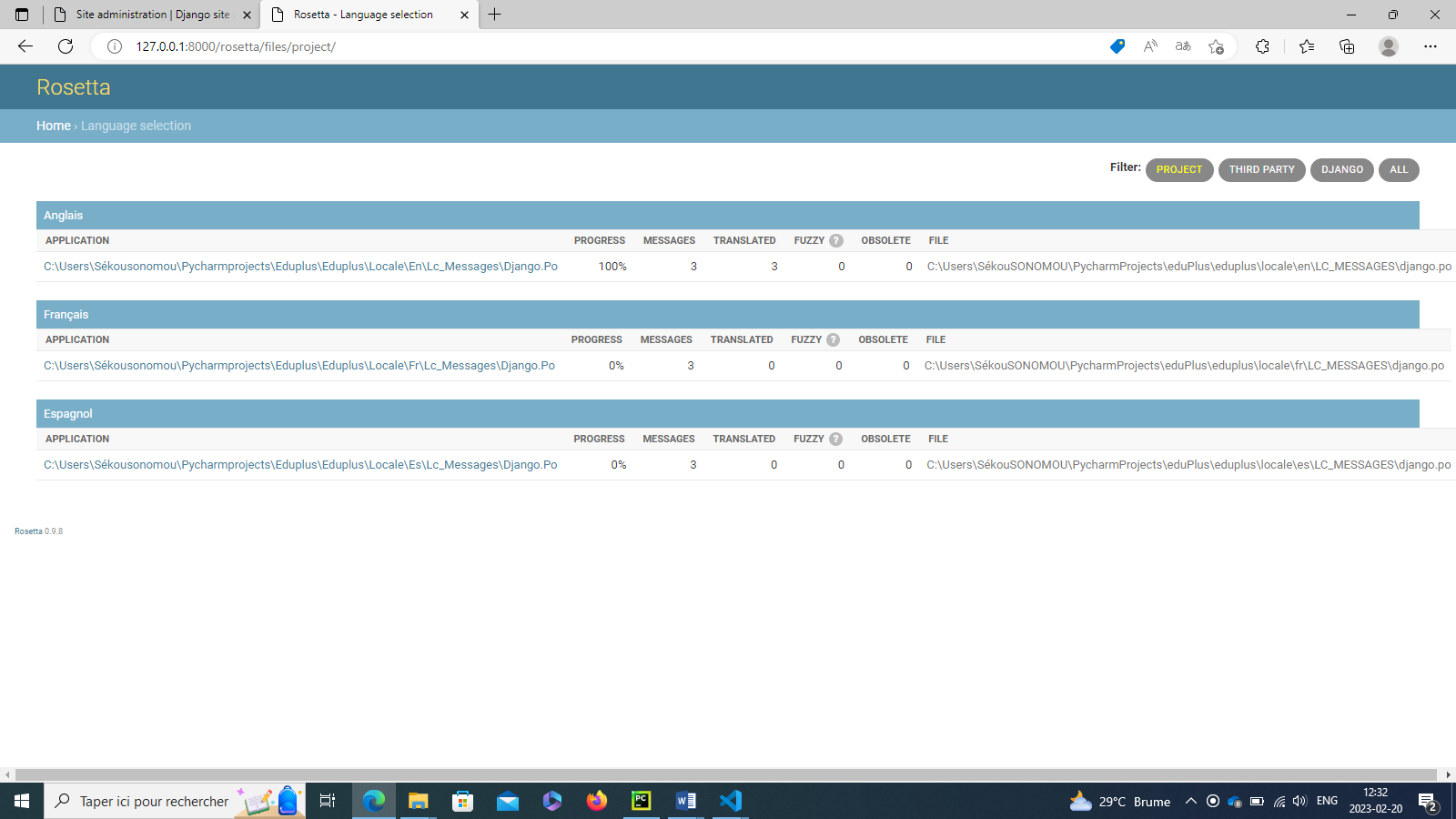
Créez et appliquez les migrations, puis exécutez le serveur :

(env)$ python manage.py makemigrations

(env)$ python manage.py migrate

(env)$ python manage.py runserver

Assurez-vous d'être connecté en tant qu'administrateur, puis accédez à <http://127.0.0.1:8000/rosetta/> dans votre navigateur :



Lorsque vous avez terminé de modifier les traductions, cliquez sur le bouton "Enregistrer et traduire le bloc suivant" pour enregistrer les traductions dans leur fichier .po respectif . Rosetta compilera ensuite le fichier de messages, il n'est donc pas nécessaire d'exécuter manuellement la django-admin compilemessages --ignore=envcommande.

Notez qu'après avoir ajouté de nouvelles traductions dans un environnement de production, vous devrez recharger votre serveur après avoir exécuté la django-admin compilemessages --ignore=envcommande, ou après avoir enregistré les traductions avec Rosetta, pour que les modifications prennent effet.

1. Protéger les secrets avec python-dotenv

Tout d'abord, installons ce module.

pip install python-dotenv

Créez ensuite un fichier *.env* dans le répertoire racine de notre projet. C'est là que nous placerons les paires clé-valeur de toutes les variables d'environnement requises par notre application.

.env

SECRET\_KEY = 'django-insecure-d$\*95uwuc)&yc3jn2-o%g$609\*!w$f(s4\*g@-y\*#^j)lz)\*m@2'

* Extrayez ces configurations de .env et chargez-les dans les paramètres.

settings.py

# To keep secret keys in environment variables

import os

from dotenv import load\_dotenv

dotenv\_path = os.path.join(os.path.dirname(\_\_file\_\_), '.env')

load\_dotenv(dotenv\_path)

Maintenant, au lieu d'exposer nos clés secrètes et nos clés OAuth dans les paramètres, récupérons-les via leurs noms de clé comme suit.

settings.py

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = str(os.getenv('SECRET\_KEY'))

1. Les modèles en Django

Les modèles Django sont des classes Python qui définissent les champs et les comportements d'un objet de base de données. Ils sont utilisés pour stocker et gérer les données de votre application dans une base de données.

Commençons par la partie blog

Dans blog\models.py

from django.db import models

from django.contrib.auth.models import User

from django.urls import reverse

import uuid

# Create your models here.

STATUS = (

    (0, "Draft"),

    (1, "Published")

)

class Category(models.Model): #Category for the Article

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False, unique=True) #Unique identifier for the Category

    title = models.CharField(max\_length=200) #Title of the Category

    created\_on = models.DateTimeField(auto\_now\_add=True) #Date of creation

    class Meta:

        verbose\_name = "Category"

        verbose\_name\_plural = "Categories"

        ordering = ['title']

    def \_\_str\_\_(self):

        return self.title

class BlogPost(models.Model):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False, unique=True) #Unique identifier for the article

    title = models.CharField(max\_length=200, unique=True) #Title of the Article

    author = models.ForeignKey(User, on\_delete=models.SET\_NULL,null=True,blank=True, related\_name='blog\_posts') #Author of the Article

    description = models.TextField(blank=True,null=True) #Short Description of the article

    category = models.ForeignKey('Category', related\_name='category', on\_delete=models.CASCADE) #Category of the article

    keywords = models.CharField(max\_length=500,blank=True,null=True) #Keywords to be used in SEO

    views = models.IntegerField(default=0)

    is\_popular = models.BooleanField(default=False)

    image = models.ImageField(upload\_to='blog/images') #Cover Image of the article

    bg = models.ImageField(upload\_to='blog/bg\_images', blank=True, null=True)

    created\_on = models.DateTimeField(auto\_now\_add=True) #Date of creation

    updated\_on = models.DateTimeField(auto\_now=True) #Date of updation

    status = models.IntegerField(choices=STATUS, default=0) #Status of the Article either Draft or Published

    def \_\_str\_\_(self):

        return self.title

    def get\_absolute\_url(self):

        return reverse("blog\_detail", kwargs={'pk': self.pk})

1. **django-taggit**

django-taggit est une application Django réutilisable pour le balisage simple [1].

Installation

pip install django-taggit

django-taggit une approche plus simple du balisage avec Django. Ajoutez "taggit" à vos INSTALLED\_APPS puis ajoutez simplement un TaggableManager à votre modèle et c'est parti :

dans le fichier eduplus\settings.py

INSTALLED\_APPS = [

    'taggit',

]

Dans le fichier eduplus\settings.py

from taggit.managers import TaggableManager

class BlogPost(models.Model):

    tags = TaggableManager() #Tags for a Particular Article, You need to install Taggit

1. **Pillow**

Pour la gestion des images, installon ce paquet

python -m pip install Pillow

1. **django-ckeditor [2]**
2. Installation

pip install django-ckeditor-5

1. Ajoutez "django\_ckeditor\_5" à vos INSTALLED\_APPS dans votre eduplus\settings.py comme ceci :

INSTALLED\_APPS = [

    ...

    'django\_ckeditor\_5',

]

# ckeditor

customColorPalette = [

        {

            'color': 'hsl(4, 90%, 58%)',

            'label': 'Red'

        },

        {

            'color': 'hsl(340, 82%, 52%)',

            'label': 'Pink'

        },

        {

            'color': 'hsl(291, 64%, 42%)',

            'label': 'Purple'

        },

        {

            'color': 'hsl(262, 52%, 47%)',

            'label': 'Deep Purple'

        },

        {

            'color': 'hsl(231, 48%, 48%)',

            'label': 'Indigo'

        },

        {

            'color': 'hsl(207, 90%, 54%)',

            'label': 'Blue'

        },

    ]

# CKEDITOR\_5\_CUSTOM\_CSS = 'path\_to.css' # optional

# CKEDITOR\_5\_FILE\_STORAGE = "path\_to\_storage.CustomStorage" # optional

CKEDITOR\_5\_CONFIGS = {

    'default': {

        'toolbar': ['heading', '|', 'bold', 'italic', 'link',

                    'bulletedList', 'numberedList', 'blockQuote', 'imageUpload', ],

    },

    'extends': {

        'blockToolbar': [

            'paragraph', 'heading1', 'heading2', 'heading3',

            '|',

            'bulletedList', 'numberedList',

            '|',

            'blockQuote',

        ],

        'toolbar': ['heading', '|', 'outdent', 'indent', '|', 'bold', 'italic', 'link', 'underline', 'strikethrough',

        'code','subscript', 'superscript', 'highlight', '|', 'codeBlock', 'sourceEditing', 'insertImage',

                    'bulletedList', 'numberedList', 'todoList', '|',  'blockQuote', 'imageUpload', '|',

                    'fontSize', 'fontFamily', 'fontColor', 'fontBackgroundColor', 'mediaEmbed', 'removeFormat',

                    'insertTable',],

        'image': {

            'toolbar': ['imageTextAlternative', '|', 'imageStyle:alignLeft',

                        'imageStyle:alignRight', 'imageStyle:alignCenter', 'imageStyle:side',  '|'],

            'styles': [

                'full',

                'side',

                'alignLeft',

                'alignRight',

                'alignCenter',

            ]

        },

        'table': {

            'contentToolbar': [ 'tableColumn', 'tableRow', 'mergeTableCells',

            'tableProperties', 'tableCellProperties' ],

            'tableProperties': {

                'borderColors': customColorPalette,

                'backgroundColors': customColorPalette

            },

            'tableCellProperties': {

                'borderColors': customColorPalette,

                'backgroundColors': customColorPalette

            }

        },

        'heading' : {

            'options': [

                { 'model': 'paragraph', 'title': 'Paragraph', 'class': 'ck-heading\_paragraph' },

                { 'model': 'heading1', 'view': 'h1', 'title': 'Heading 1', 'class': 'ck-heading\_heading1' },

                { 'model': 'heading2', 'view': 'h2', 'title': 'Heading 2', 'class': 'ck-heading\_heading2' },

                { 'model': 'heading3', 'view': 'h3', 'title': 'Heading 3', 'class': 'ck-heading\_heading3' }

            ]

        }

    },

    'list': {

        'properties': {

            'styles': 'true',

            'startIndex': 'true',

            'reversed': 'true',

        }

    }

}

1. Ajout d’url dans eduplus\urls.py :

from django.conf import settings

from django.conf.urls.static import static

# [ ... ]

urlpatterns += [

    path("ckeditor5/", include('django\_ckeditor\_5.urls')),

]

Ajoutez à votre monapp\models.py :

from django\_ckeditor\_5.fields import CKEditor5Field

class BlogPost(models.Model):

    content = CKEditor5Field(config\_name='extends') #Content of the article, you need to install CKEditor

les migrations

python manage.py makemigrations

python manage.py migrate

eduPLus\contact\models.py

from django.db import models

import uuid

# Create your models here.

# Contact

class Contact(models.Model):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False)

    nom=models.CharField(blank=True,null=True,max\_length=100,name='nom',verbose\_name='Nom')

    email=models.EmailField(blank=True,null=True,max\_length=100,name='email',verbose\_name='Email')

    message=models.TextField(blank=True,null=True,verbose\_name='Message',name='message')

    created=models.DateField(auto\_now\_add=True,blank=True,null=True,verbose\_name='Create date')

    updated=models.DateTimeField(auto\_now=True,verbose\_name='Update date')

    class Meta:

        ordering=['-updated']

        verbose\_name="Contact"

    def \_\_str\_\_(self) -> str:

        return self.nom

class NewsLetter(models.Model):

    email=models.EmailField(blank=True,null=True,max\_length=100,verbose\_name='Email',name='email')

    created=models.DateTimeField(auto\_now\_add=True,verbose\_name='Date')

    class Meta:

        verbose\_name="NewsLetter"

        ordering=['-created']

    def \_\_str\_\_(self) -> str:

        return self.email

1. Espace d’administration

Une des parties les plus puissantes de Django est l’interface d’administration automatique. Elle lit les métadonnées de vos modèles pour fournir une interface rapide et centrée sur les modèles où les utilisateurs de confiance peuvent gérer le contenu du site. L’utilisation recommandée du site d’administration est limitée à un outil de gestion interne d’un organisme. Il n’est pas conçu pour servir de base à la construction d’une interface publique complète.

Le site d’administration présente de nombreux points d’entrée visant la personnalisation, mais ne vous bornez pas à utiliser exclusivement ces points d’entrée. Si vous avez besoin de fournir une interface plus centrée sur les processus et qui fait abstraction des détails d’implémentation autour des tables et champs de bases de données, c’est qu’il est vraisemblablement le moment d’écrire vos propres vues[3].

Création d’un super user

Il nous faut créer un compte super-utilisateur. Il sera au début le seul compte à pouvoir accéder à l'administration.

Pour ce faire, vous devez taper la commande suivante :

python manage.py createsuperuser

eduPLus\blog\admin.py

from django.contrib import admin

from .models import Category,BlogPost

# Register your models here.

admin.site.register(BlogPost)

admin.site.register(Category)

eduPLus\contact\admin.py

from django.contrib import admin

from .models import \*

admin.site.register(Contact)

admin.site.register(NewsLetter)

ajoute d’une barre d'outils de débogage dans le projet

La boîte à outils Django est un outil de débogage utilisé pour déboguer les requêtes de base de données, la vitesse de chargement du site Web Django et bien d'autres choses. La barre d'outils de débogage est très populaire parmi les développeurs et tout le monde l'utilise.

Installation

pip install django-debug-toolbar

Maintenant, ajoutez 'debug\_toolbar' à votre INSTALLED\_APPS dans settings.py -

TIERS\_APPS=[

    'debug\_toolbar',

]

Cela ajoutera la barre d'outils de débogage en tant qu'application dans notre projet.

Ensuite, dans votre **middleware** , ajoutez ce qui suit -

MIDDLEWARE = [

    'debug\_toolbar.middleware.DebugToolbarMiddleware', #new

]

Ceci est utilisé pour donner accès à la base de données.

Maintenant, dans urls.py du répertoire principal de votre projet, ajoutez l'URL de la barre d'outils de débogage -

import debug\_toolbar

from django.conf import settings

from django.urls import include, path

urlpatterns = [

   ...

   path('\_\_debug\_\_/', include(debug\_toolbar.urls)),

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

]

L'URL définira où tous les rapports de débogage doivent s'afficher et où la barre d'outils de débogage doit être hébergée.

Maintenant, dans **settings.py,** ajoutez une autre variable **INTERNAL\_IPS** et mentionnez localhost dedans -

INTERNAL\_IPS = [

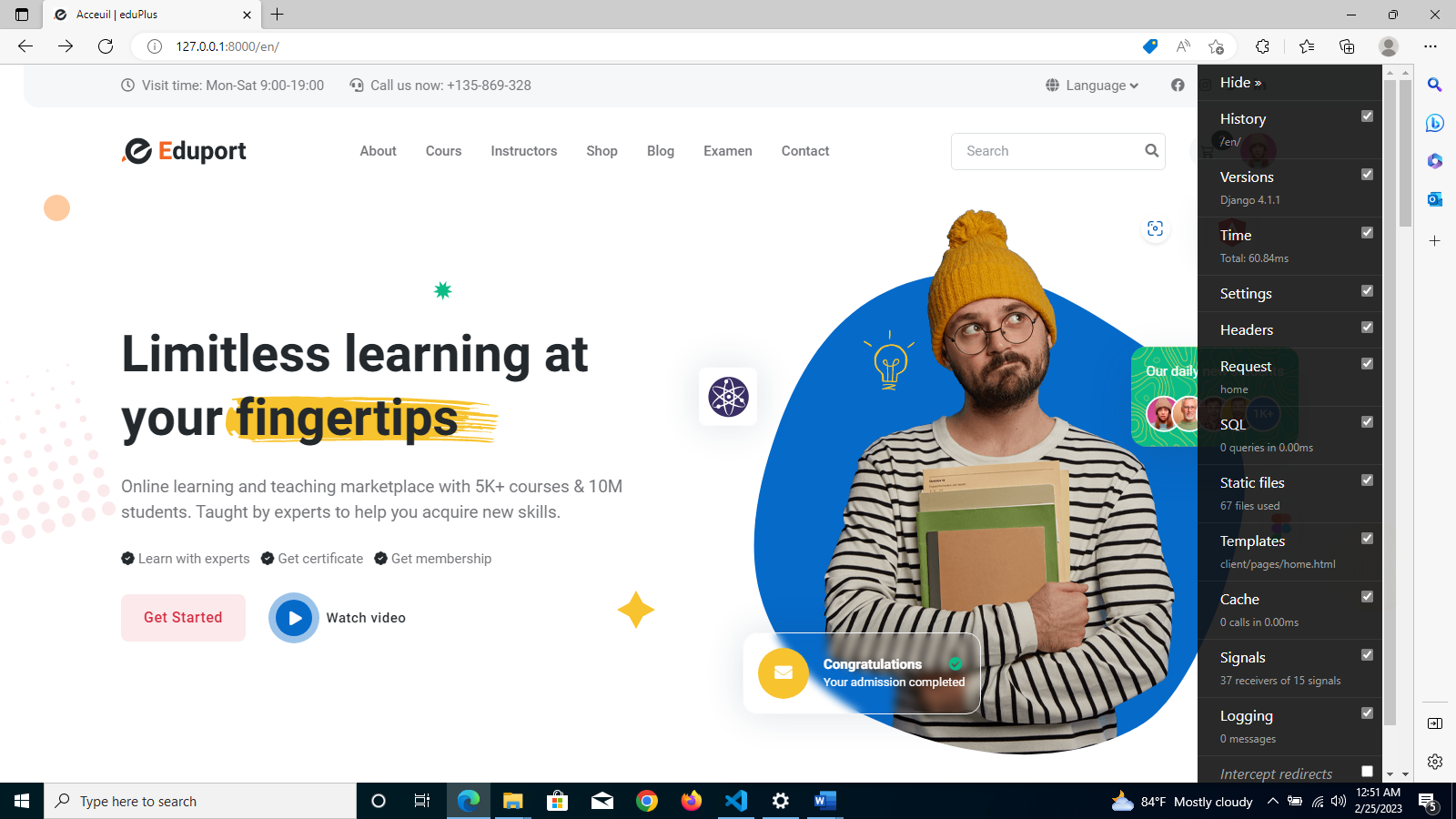
   # ...

   '127.0.0.1',

   # ...

]

Cette variable définira quelle URL doit être déboguée et sur quel débogage doit être affiché.



# Bibliothèque D'importation Et D'exportation Django

**La bibliothèque django-import-export**  fournit une **application import\_export**  pour votre projet Django. Vous pouvez utiliser l'  application import\_export  avec Django Admin. Il ajoutera simplement **des boutons d'importation**  et **d'exportation** entièrement fonctionnels  au**tableau de bord d'administration** .

Les données de modèle peuvent être importées depuis  CSV, XML, Excel, JSON,  etc. Vous pouvez également exporter des données vers plusieurs formats de fichiers tels que  CSV, XLS, TSV, ODS, JSON, YAML  et  HTML .

## Installation De La Bibliothèque Django-Import-Export

Pour installer la bibliothèque django-import-export, exécutez simplement :

pip install django-import-export

### Ajouter L'application Import\_export Au Projet  Settings.Py

Maintenant, vous devez ajouter  l'application import\_export  dans votre projet  settings.py  sous la  liste d'attributs **INSTALLED\_APPS**  . Notez que le "-" est remplacé par "\_" en raison des règles de nomenclature de Python.

    INSTALLED\_APPS = [

        ...

        'import\_export',

    ]

### Exemple De Projet De Django Import Export

[Remarque : le projet complet se trouve à la fin de la page en téléchargement gratuit.]

Nous allons utiliser un exemple de classe de modèle appelé  **Contact**

eduPLus\contact\models.py

from django.db import models

import uuid

# Create your models here.

# Contact

class Contact(models.Model):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False)

    nom=models.CharField(blank=True,null=True,max\_length=100,name='nom',verbose\_name='Nom')

    email=models.EmailField(blank=True,null=True,max\_length=100,name='email',verbose\_name='Email')

    message=models.TextField(blank=True,null=True,verbose\_name='Message',name='message')

    created=models.DateField(auto\_now\_add=True,blank=True,null=True,verbose\_name='Create date')

    updated=models.DateTimeField(auto\_now=True,verbose\_name='Update date')

    class Meta:

        ordering=['-updated']

        verbose\_name="Contact"

    def \_\_str\_\_(self) -> str:

        return self.nom

Maintenant, importez et héritez simplement  de la classe **ImportExportModelAdmin**  dans  admin.py

#### Admin.Py

from django.contrib import admin

from .models import \*

from import\_export.admin import ImportExportModelAdmin

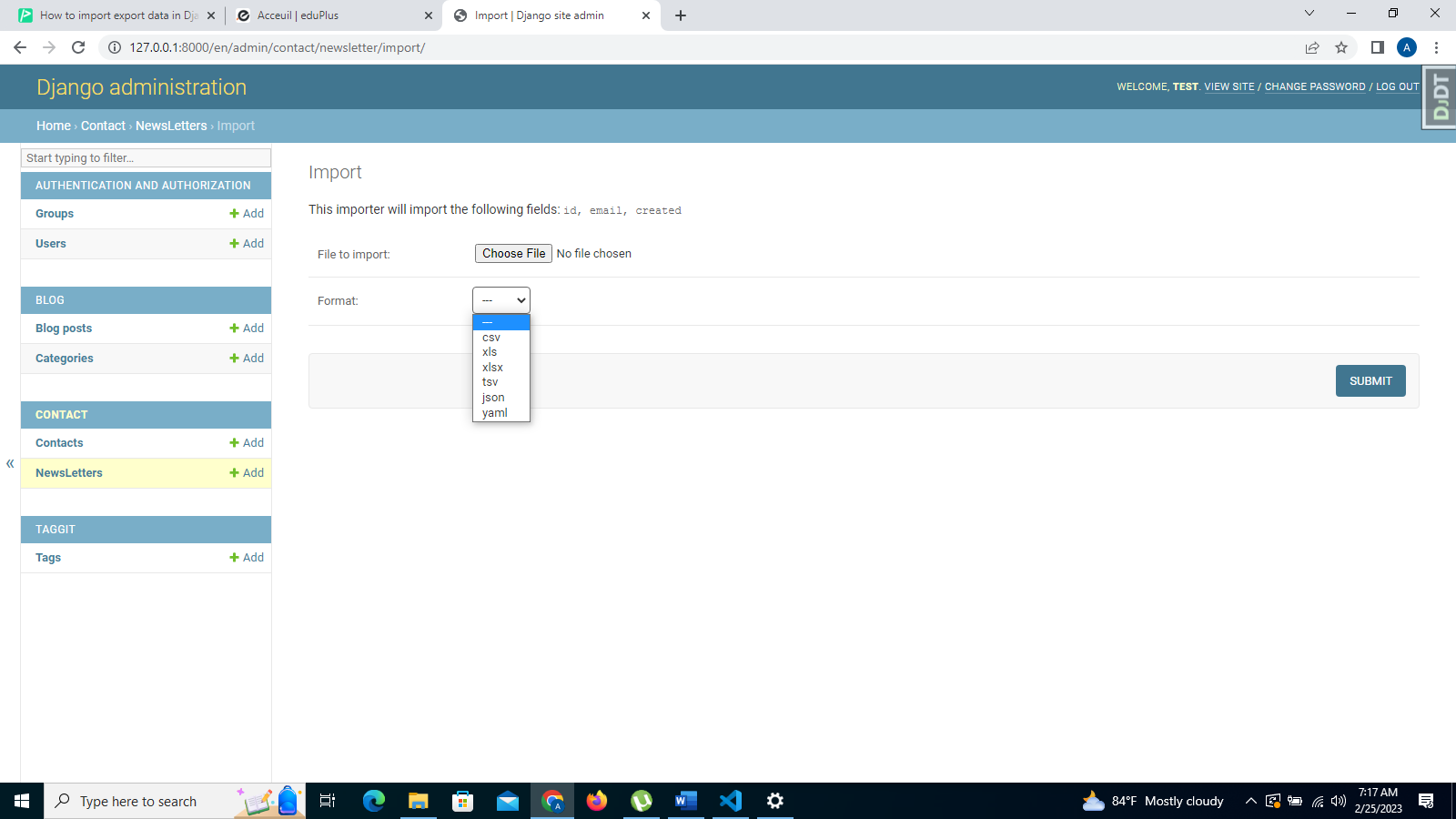
# Register your models here.

class ContactAdmin(ImportExportModelAdmin,admin.ModelAdmin):

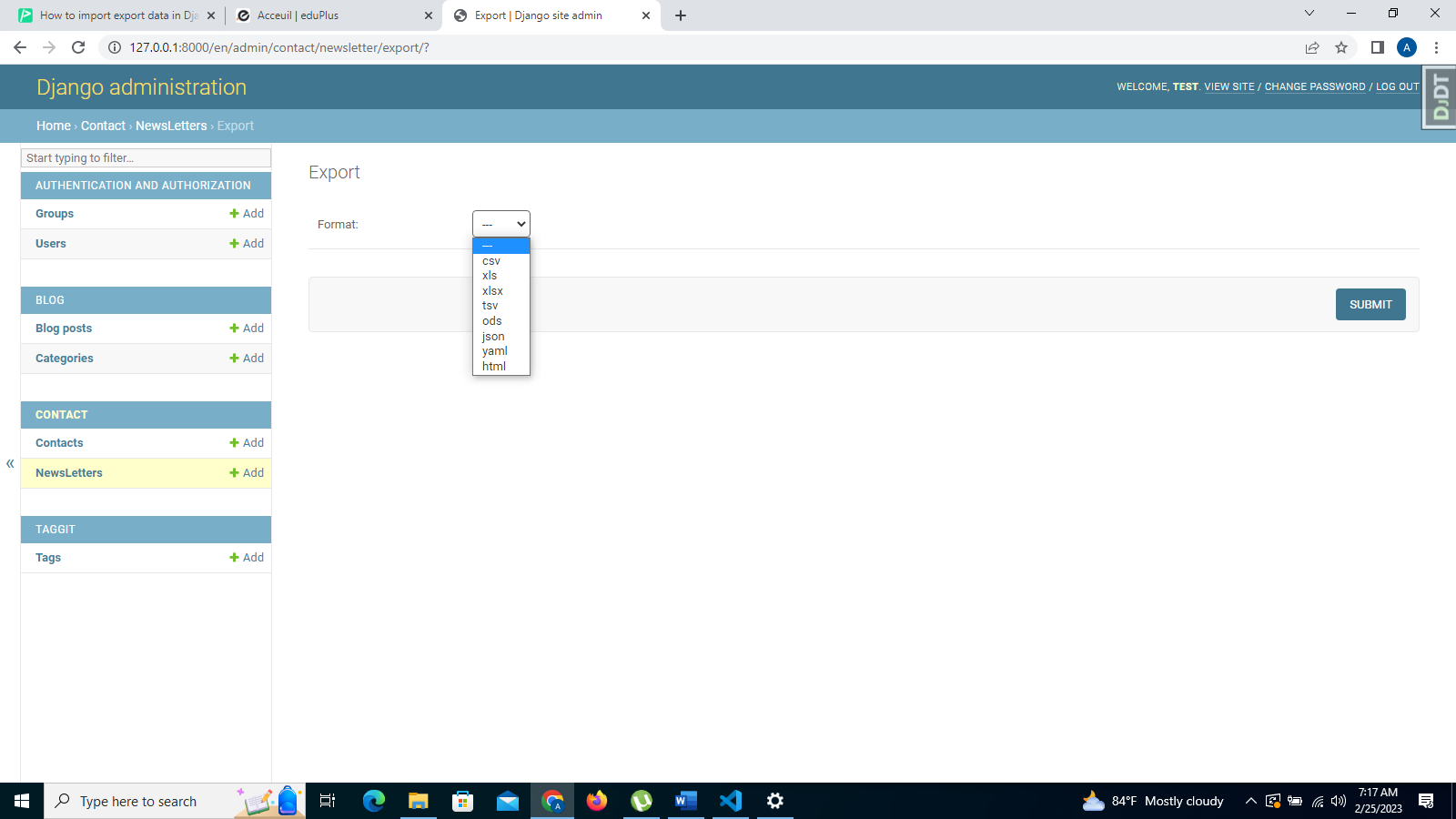
    list\_display=('nom','email','message','updated')

admin.site.register(Contact,ContactAdmin)

#### Importation



#### Exportation



Envoi de mail

eduplus\settings.py

# email configs

EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL\_HOST = str(os.getenv('EMAIL\_HOST'))

EMAIL\_USE\_TLS = True

EMAIL\_PORT = str(os.getenv('EMAIL\_PORT'))

EMAIL\_HOST\_USER = str(os.getenv('EMAIL\_USER'))

EMAIL\_HOST\_PASSWORD = str(os.getenv('EMAIL\_PASSWORD'))

ADMIN\_EMAIL = str(os.getenv('ADMIN\_EMAIL'))

SUPPORT\_EMAIL = str(os.getenv('SUPPORT\_EMAIL'))

DEFAULT\_FROM\_EMAIL = ADMIN\_EMAIL

SERVER\_EMAIL = ADMIN\_EMAIL

.env

#E-MAIL SETTINGS

EMAIL\_HOST='smtp.titan.email'

EMAIL\_HOST\_USER='contact@sonomou.tech'

EMAIL\_HOST\_PASSWORD='xxxxxxxxxxxxxxxxxxx'

EMAIL\_PORT= 465

EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL\_USE\_SSL=True

ADMIN\_EMAIL = "contact@sonomou.tech"

SUPPORT\_EMAIL = "contact@sonomou.tech"

DEFAULT\_FROM\_EMAIL = ADMIN\_EMAIL

SERVER\_EMAIL = ADMIN\_EMAIL

Gestion de l’authentification

Dans accounts\models.py

from django.db import models

from django.contrib.auth.models import AbstractUser

from PIL import Image

from django.contrib.auth.models import BaseUserManager

import uuid

# Create your models here.

class UserProfileManager(BaseUserManager):

    def create\_user(self,email,password=None,\*callback\_args, \*\*callback\_kwargs):

        if not email:

            raise ValueError('Desolé, veuillez saisir un email')

        email=self.normalize\_email(email)

        user=self.model(email=email)

        user.set\_password(password)

        # user=self.model(username=username,password=password)

        user.save(using=self.\_db)

        return user

    def create\_superuser(self,email,password):

        user=self.create\_user(email,password)

        user.is\_staff=True

        user.is\_superuser=True

        user.is\_active=True

        user.save(using=self.\_db)

        return user

class Profession(models.Model):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False)

    title=models.CharField(verbose\_name='Title',max\_length=200)

    description=models.TextField(null=True,blank=True,)

    created=models.DateField(auto\_now\_add=True,blank=True,null=True,verbose\_name='Create date')

    updated=models.DateTimeField(auto\_now=True,verbose\_name='Update date')

    class Meta:

        verbose\_name="Profession"

    def \_\_str\_\_(self) -> str:

        return self.title

class Specialite(models.Model):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False)

    title=models.CharField(verbose\_name='Title',max\_length=200)

    description=models.TextField(null=True,blank=True,)

    created=models.DateField(auto\_now\_add=True,blank=True,null=True,verbose\_name='Create date')

    updated=models.DateTimeField(auto\_now=True,verbose\_name='Update date')

    class Meta:

        verbose\_name="Specialite"

    def \_\_str\_\_(self) -> str:

        return self.title

class User(AbstractUser):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False)

    is\_student = models.BooleanField(default=False)

    is\_teacher = models.BooleanField(default=False)

    email = models.EmailField(db\_index=True, unique=True)

    username = None

    last\_name = models.CharField(max\_length=250, blank=True, null=True)

    first\_name = models.CharField(max\_length=250, blank=True, null=True)

    midle\_name = models.CharField(max\_length=250, blank=True, null=True)

    photo = models.ImageField(upload\_to='users/photos/',blank=True, null=True)

    about = models.TextField(blank=True, null=True)

    twitter = models.CharField(

        blank=True, null=True, name='twitter', verbose\_name="Twitter", max\_length=200)

    facebook = models.CharField(

        blank=True, null=True, name='facebook', verbose\_name="Facebook", max\_length=200)

    instagram = models.CharField(

        blank=True, null=True, name='instagram', verbose\_name="Instagram", max\_length=200)

    linkdin = models.CharField(

        blank=True, null=True, name='linkdin', verbose\_name="Linkdin", max\_length=200)

    youtube = models.CharField(

        blank=True, null=True, name='youtube', verbose\_name="Linkdin", max\_length=200)

    telephone= models.CharField(null=True,blank=True,unique=True,max\_length=12)

    profession=models.ForeignKey(Profession,on\_delete=models.SET\_NULL,blank=True,null=True,related\_name="fk\_profession")

    specialite=models.ForeignKey(Specialite,on\_delete=models.SET\_NULL,blank=True,null=True,related\_name="fk\_specialite")

    pays= models.CharField(null=True,blank=True,unique=True,max\_length=12)

    is\_active = models.BooleanField(default=True)

    is\_staff = models.BooleanField(default=False)

    # A timestamp representing when this object was created.

    created\_at = models.DateTimeField(auto\_now\_add=True)

    # A timestamp reprensenting when this object was last updated.

    updated\_at = models.DateTimeField(auto\_now=True)

    USERNAME\_FIELD = 'email'

    REQUIRED\_FIELDS = []

    def \_\_str\_\_(self):

        if self.is\_student:

                type\_ = 'Student'

        elif self.is\_teacher:

            type\_ = 'Teacher'

        else:

            type\_ = 'None'

        return f'{type\_}: {self.email}'

    def save(self, \*args, \*\*kwargs):

        super().save(\*args, \*\*kwargs)

        if self.photo:

            pic = Image.open(self.photo.path)

            if pic.width > 256:

                pic.thumbnail((256, pic.height / (pic.width / 265)))

                pic.save(self.photo.path)

    def get\_photo\_url(self):

        if self.photo:

            return self.photo.url

        else:

            if self.is\_student:

                return '/media/users/students/student-default.png'

            elif self.is\_teacher:

                return '/media/users/teachers/teacher-default.png'

            else:

                return '/media/users/default.png'

    def get\_photo\_name(self):

        if self.photo:

            return self.photo.name

        else:

            if self.is\_student:

                return 'student-default.png'

            elif self.is\_teacher:

                return 'teacher-default.png'

            else:

                return 'default.png'

    def get\_profile(self):

        if self.is\_student:

            return Student.objects.get(user=self)

        elif self.is\_teacher:

            return Teacher.objects.get(user=self)

        else:

            return None

    def get\_shortname(self):

        return f'{self.first\_name[0:1]}.{self.last\_name}'

    def get\_fullname(self):

        return f'{self.first\_name} {self.last\_name}'

    objects= UserProfileManager()

class Student(models.Model):

    user = models.OneToOneField(User, on\_delete=models.CASCADE, primary\_key=True, unique=True)

    interested\_category=models.TextField(blank=True,null=True,verbose\_name="Interested category")

    def \_\_str\_\_(self) -> str:

        return f'{self.user.email}'

class Teacher(models.Model):

    user = models.OneToOneField(User, on\_delete=models.CASCADE, primary\_key=True, unique=True)

    def \_\_str\_\_(self) -> str:

        return f'{self.user.email}'

# la class Competence

VALEURS = (

    (0, "50"),

    (1, "60"),

    (2, "80"),

    (3, "90"),

)

class Competence(models.Model):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False)

    title=models.CharField(verbose\_name='Title',max\_length=200)

    pourcentage=models.IntegerField(choices=VALEURS, default=0,null=True,blank=True,verbose\_name="Pourcentage")

    teacher=models.ForeignKey(Teacher,on\_delete=models.SET\_NULL,blank=True,null=True,verbose\_name='Teacher',related\_name="fk\_teacher\_competence")

    created=models.DateField(auto\_now\_add=True,blank=True,null=True,verbose\_name='Create date')

    updated=models.DateTimeField(auto\_now=True,verbose\_name='Update date')

    class Meta:

        verbose\_name="Competences"

    def \_\_str\_\_(self) -> str:

        return self.title

dans accounts\signals.py

from django.db.models.signals import post\_save

from .models import \*

from django.dispatch import receiver

def save\_profile(sender, instance, created, \*\*kwargs):

    if created:

        if instance.is\_student:

            student = Student(user=instance)

            student.save()

        elif instance.is\_teacher:

            teacher = Teacher(user=instance)

            teacher.save()

post\_save.connect(save\_profile, sender=User)

accounts\apps.py

from django.apps import AppConfig

class AccountsConfig(AppConfig):

    default\_auto\_field = 'django.db.models.BigAutoField'

    name = 'accounts'

    def ready(self):

        import accounts.signals

accounts\admin.py

from django.contrib import admin

from .models import \*

from import\_export.admin import ImportExportModelAdmin

# Register your models here.

@admin.register(User)

class UserAdmin(ImportExportModelAdmin, admin.ModelAdmin):

    list\_display =( 'email','specialite','is\_student','is\_teacher', 'is\_active', 'is\_staff', 'created\_at', 'updated\_at')

    ordering=('updated\_at',)

@admin.register(Profession)

class ProfessionAdmin(ImportExportModelAdmin, admin.ModelAdmin):

    list\_display=('title','description',)

@admin.register(Specialite)

class SpecialiteAdmin(ImportExportModelAdmin, admin.ModelAdmin):

    list\_display=('title','description',)

@admin.register(Student)

class StudentAdmin(ImportExportModelAdmin, admin.ModelAdmin):

    list\_display=('user','interested\_category',)

class CompetenceAdmin(admin.StackedInline):

    model= Competence

class TeacherAdmin(admin.ModelAdmin):

    inlines=[CompetenceAdmin]

admin.site.register(Teacher,TeacherAdmin)

@admin.register(Competence)

class CompetenceAdmin(ImportExportModelAdmin, admin.ModelAdmin):

    list\_display=('title','pourcentage','teacher','updated')

**django-countries**

1. **Installation**

**pip install django-countries**

For more accurate sorting of translated country names, install it with the optional **pyuca** package:

**pip install django-countries[pyuca]**

1. Add **django\_countries** to **INSTALLED\_APPS**

**CountryField**

A country field for Django models that provides all ISO 3166-1 countries as choices.

**CountryField** is based on Django’s CharField, providing choices corresponding to the official ISO 3166-1 list of countries (with a default **max\_length** of 2).

Consider the following model using a **CountryField**:

from django.db import models

from django\_countries.fields import CountryField

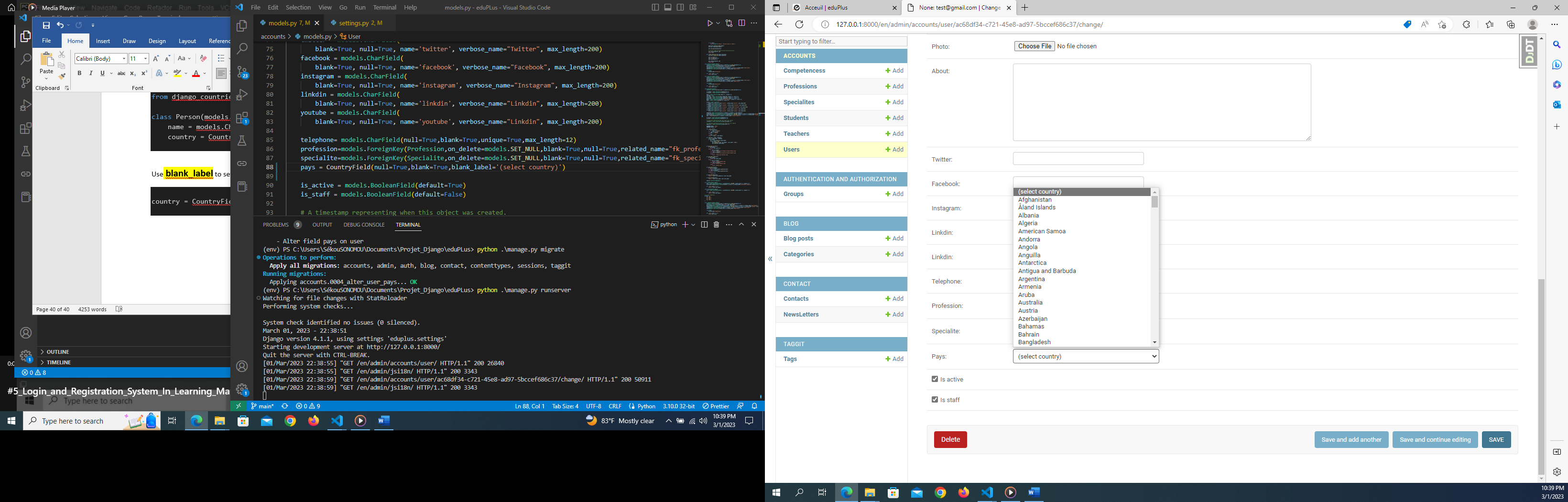
class User(AbstractUser):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False)

    pays = CountryField(null=True,blank=True,blank\_label='(select country)')

Use **blank\_label** to set the label for the initial blank choice shown in forms:

country = CountryField(blank\_label='(select country)')



# django-phonenumber-field

Une bibliothèque Django qui s'interface avec [python-phonenumbers](https://github.com/daviddrysdale/python-phonenumbers) pour valider, imprimer et convertir les numéros de téléphone. python-phonenumbers est un port de la bibliothèque [libphonenumber](https://github.com/google/libphonenumber) de Google , qui gère la gestion des numéros de téléphone d'Android.

## Installation

Installer en utilisant [pip](https://pypi.org/project/phonenumbers/) avec :

pip install phonenumbers

pip install django-phonenumber-field

Et dans le fichier settings.py :

# Application definition

INSTALLED\_APPS = [

    'phonenumber\_field',

]

Utilization

Téléphone par PhoneNumberField

Dans le modèle :

from phonenumber\_field.modelfields import PhoneNumberField

class User(AbstractUser):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False)

    telephone= PhoneNumberField(null=False, blank=False, unique=True)

Téléphone par regexp

class ReceiverForm(forms.ModelForm):

     telephone= forms.RegexField(regex=r'^\+?1?\d{9,15}$',

                                     error\_message = ("Phone number must be entered in the format: '+999999999'. Up to 15 digits is allowed."))

accounts\models.py

from django.core.validators import RegexValidator

class User(AbstractUser):

    uid=models.UUIDField(primary\_key=True,default=uuid.uuid4,editable=False)

    phone\_regex = RegexValidator(regex=r'^\+?1?\d{9,15}$', message="Phone number must be entered in the format: '+999999999'. Up to 15 digits allowed.")

    telephone = models.CharField(validators=[phone\_regex], max\_length=17, blank=True) # Validators should be a list

**PhoneNumberPrefixWidget()**

De plus, attribuez **le widget "PhoneNumberPrefixWidget()"** au champ dans un formulaire personnalisé et attribuez le formulaire personnalisé à l'administrateur comme indiqué ci-dessous :

from django import forms

from phonenumber\_field.widgets import PhoneNumberPrefixWidget

class UserForm(forms.ModelForm):

    class Meta:

        widgets = {

            'telephone': PhoneNumberPrefixWidget(),

        }

@admin.register(User)

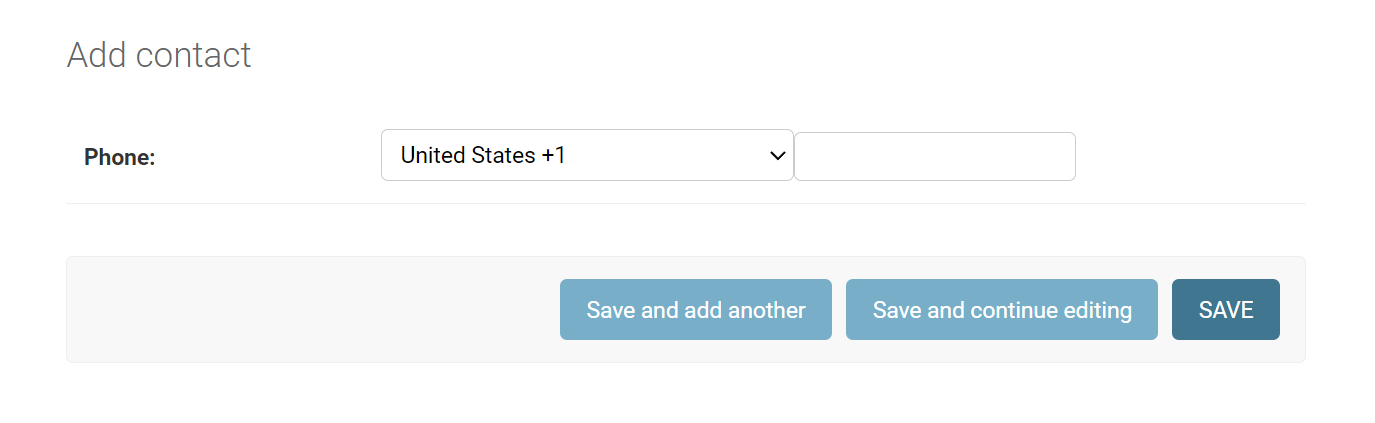
class UserAdmin(ImportExportModelAdmin, admin.ModelAdmin):

    list\_display =( 'email','specialite','is\_student','is\_teacher', 'is\_active', 'is\_staff', 'created\_at', 'updated\_at')

    ordering=('updated\_at',)

    form = UserForm

Maintenant, avec les codes de pays, le champ pour un numéro de téléphone est créé

[](https://i.stack.imgur.com/5m4qy.png)

Django + Heroku

Etape 1: Installez les dépendances et ces packages requis par votre application Django.

..\env\Scripts\activate

pip install django gunicorn whitenoise dj-database-url psycopg2

ÉTAPE 2 : Créez un fichier nommé Procfile et ajoutez-y la ligne suivante ci-dessous.

web: gunicorn eduplus.wsgi --log-file –

Les applications Heroku utilisent un fichier nommé Procfile sans extensions qui déclare les commandes exécutées par l'application au démarrage de l'application.

ETAPE 3: Créer un fichier d' exigences requirements.txt

Ce fichier d'exigences contiendra tous les modules que vous avez utilisés dans votre application pour un fonctionnement correct et oui, vous n'avez pas besoin d'écrire manuellement chaque module que vous avez utilisé, il y a une commande pour cela et c'est juste en dessous.

pip freeze > requirements.txt

python manage.py collectstatic

ETAPE 4 : Créer un fichier d'exécution

Ajoutez dans le dossier racine de vos projets runtime.txt ,ensuite ajoutez le python-3.10.4 suivant

**Settings.py:** Modify your settings.py as per the instructions below:

1.Set debug as False.

DEBUG = False

2. Modify allowed hosts.

ALLOWED\_HOSTS = ['127.0.0.1', '.herokuapp.com']

3. To disable Django’s static file handling and allow WhiteNoise to take over add ‘nostatic’ to the top of your ‘INSTALLED\_APPS’ list.

INSTALLED\_APPS = [

'whitenoise.runserver\_nostatic',

'django.contrib.staticfiles',

# ...

]

4. Add WhiteNoise to the MIDDLEWARE list. The WhiteNoise middleware should be placed directly after the Django SecurityMiddleware (if you are using it) and before all other middleware:

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'whitenoise.middleware.WhiteNoiseMiddleware',

# ...

]

5.  Update your database settings.

import dj\_database\_url

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql\_psycopg2',

'NAME': '<database\_name>',

'USER': '<user\_name>',

'PASSWORD': '<password>',

'HOST': 'localhost',

'PORT': '',

}

}

db\_from\_env = dj\_database\_url.config(conn\_max\_age=500)

DATABASES['default'].update(db\_from\_env)