

Django v5 Tutorial

Table of Contents

Install and Setup Django Environment with Base Template.....	4
Create model from a template.....	10
Add messages and display models with pagination.....	13
Edit and delete models.....	17
Register and login new users.....	21
Create Users App 	22
generic models, forms and views.....	23
Implementing auth_views and registerUserView.....	24
Implementing templates.....	25
Secure Views 	26
Profile Model    	28
Implement form and view.....	30
Update profile template.....	32
Implement form, url and profileView.....	33
Update user's password using custom email.....	36
Zoho mail.....	37
DNS: NameCheap, Hostinger & GoDaddy.....	38
Zoho Admin Console mailadmin.zoho.com mailadmin.zoho.eu.....	39
Update environment variables for email-host-user.....	39
profile temp.....	40
late + add new: password; reset, done, confirm and complete.....	41
Create, edit and delete user's models only.....	44
Search for Users' Models 	48
Queryset / Lookups / .filter(field_lookup) / Object-Relational Mapping.....	50
Contact Us View (Support)  	60
email_client_and_support_team.....	61
contactFormView.....	63
contact url and template.....	64
Secure Forms (reCAPTCHA) 	65
Amazon Web Services (s3 buckets)  	68
Create General purpose S3 Bucket at Europe Region: eu-west-2.....	69
Connect to AWS S3 via django-storages.....	71
Profile +  /recycle_pics AWS S3 Buckets  	73

Filter NSFW (not safe for work) Profile 📸 images 🍑	77
Signals.....	79
Utils (Utilities).....	79
Class-based views: list, form, detail, create, update and delete.....	80
Types of Class-Based Views.....	81
Advantages of CBVs.....	81
Key Components of CBVs.....	81
Example - Import class-based-views.....	82
Example - UpdateView.....	83
Example - ListView.....	85
HTMX ☕️🔥	86
Infinite Scroll - Update index.html AND Extension models_list.html.....	91
Infinite Scroll - Update indexView.....	94
Infinite Scroll - Models relevency message.....	98
Chat Rooms 💬💥📝⚡ made with: HTMX + Dafne ASGI WebSockets.....	99
Terminology.....	100
Create Chat App 💬📝.....	102
Let's implement CUSTOM {% strip %} template tag (OPTIONAL).....	108
Include - HyperScript (OPTIONAL).....	109
Chat with Django Channels 💬📝.....	111
Some differences between Views & Consumers.....	113
POST message via WebSocket Channel.....	114
Broadcast message to Consumers with Channel Layers.....	115
Online Status (Offline 🔍 5 Online 🔍).....	117
Optimize Chat app with Valkey TO improve server performance.....	120
Chat with Valkey Channels.....	121
Private Chat Rooms 💬📝🔒	122
Usecase example of chat room models with related_name property.....	123
Get or create private chatroom (utility).....	124
Update room View.....	125
Create chat with View.....	126
Update urlpatterns.....	126
My chats Dropdown.....	127
Update index template for one-to-one chatrooms.....	129
Install Stripe 💳💸🏦	131
Initialize API Keys.....	132
PCI and Data Protection Compliance 🔐.....	133
Payment System Vat Tax 💰.....	134
Stripe's Customer 📜.....	135
Buy Plan with Stripe's Card Element UI 💳.....	138
DonateView - Modifiable Payment Element UI 💳.....	148
Create DonateView - Template + Payment Element.....	149
Synchronously mounted, custom donation field.....	153

Stripe CSS and field_error.html extension.....	157
Proceed transaction, with modified by client, amount.....	163
Stripe loader extension and exemption of asterisks (*).	167
Custom Billing Address.....	170
Dynamic Select Images with Flag Icons	171
Install django-countries.....	172
Translate country names with Polish locale messages.....	176
Disclude blocked countries.....	179
Custom Postalcde Field.....	181
Enhance field_error.html extension + translate validity.....	184
Organize billing address fields country & postal_code.....	186
Initialize custom billing address as main address line.....	189
Error – 400 , 403 , 404 and 500	191
Serve staticfiles in Production with WhiteNoise	193
How Django Handles Static Files.....	193
Install WhiteNoise.....	194
Update backend of staticfiles.....	194
Account Model	195
Organize models.py and create Account model.....	195
Register Account model in admin panel.....	198
Update get_or_create_stripe_customer() utility.....	199
Update views in app_name app and profileView in users app.....	200
Migrate users' stripe data FROM Profile TO new Account model.....	201
Verify changes in admin panel.....	203
Remove stripe data FROM Profile model and its admin panel.....	204
2FA (Two Factor Authentication)	205
Install PyOTP + QRCode + Twilio and PhoneNumberField.....	207
Update the Main <form> at Profile Template.....	208
Enable MFA.....	210
Setup environment variables AND configure settings.py.....	211
Register Twilio Phone Number.....	215
Send SMS via Twilio Phone Number.....	217
Resolve ERROR 21608 for Trial Twilio Account.....	218
Create modals.....	219
Password Modal.....	223
QR Code Modal.....	223
Email Modal.....	225
SMS Modal.....	228
Modals Functionality.....	230
Enable MFA View.....	232
Request OTP View.....	236
Disable MFA.....	239
Authenticate users with MFA.....	243

Author of the Documentation: **McRaZick** (Gabriel Książek)
Contact: McRaZick@mail.com

Install and Setup Django Environment with Base Template

Create project environment

```
mkdir project_folder && cd project_folder
```

Virtual Environment (venv) to isolate pip packages

 .venv

```
python -m venv .venv      # Create .venv folder
```

```
.venv/Scripts/activate      # Windows          to exit, type: deactivate  
source .venv/bin/activate   # Linux           OR use buttons: ctrl + c
```

Environment Variables

 .env

```
echo 'DEBUG=True'    > .env    # create .env file (with variable DEBUG=True)  
echo 'SECRET_KEY=...' >> .env    # add $SECRET_KEY (make it a strong password)  
source .env          # Linux          (imports $VARIABLES)
```

I will use the **load_dotenv** function from the **python-dotenv** library in **django_project/settings.py** to automatically load environment variables from the .env file.

In production, it is generally recommended to avoid using python-dotenv. Instead, rely on the hosting platform's (Heroku / Railway) built-in mechanisms for managing environment variables to avoid conflicts.

.gitignore

ignores specified directories when pushing the project to platforms such as: GitHub and GitLab

```
echo '  
# Environments  
.env  
.venv  
env/  
venv/  
ENV/  
env.bak/  
venv.bak/  
  
# Node.js  
node_modules/  
' > .gitignore
```

```
echo '\n  
# Ignore SQLite database files  
*.sqlite  
*.sqlite3  
*.db  
*.db3' >> .gitignore  
  
# Append more paths to .gitignore  
# In this instance, ignore SQLite DB
```

```
# creates/writes .gitignore file (This is basic .gitignore, learn more here)
```

requirements.txt

register, downloaded via: pip (package manager),  libraries in this text file

```
echo > requirements.txt    # create empty file
```

```
# write/create file with installed pip packages (Use only while in  (venv) )  
pip freeze > requirements.txt
```

💻 (venv) @ project_folder

 Django Framework

```
pip install django
django-admin startproject my_website .      # use snake_case naming convention
```

💻 (venv) @ project_folder

 manage.py

```
python manage.py startapp app_name          # create app
python manage.py migrate                   # update database
python manage.py createsuperuser           # create admin user
python manage.py runserver localhost:8000   # run/start project

# in the terminal, press ctrl and c buttons simultaneously, to stop the server
```

Don't use these commands because we haven't initialized app with static folder

```
# in production, collect all /static files TO /staticfiles (STATIC_ROOT)
```

```
python manage.py collectstatic
```

```
# use the below commands in terminal after modifying apps' models.py
```

```
python manage.py makemigrations # prepare any modifications for the database
python manage.py migrate       # update database with these new modifications
```

💻 (venv) @ project_folder

 pip library

```
pip install
    python-dotenv
    B django-bootstrap5 django-crispy-forms crispy-bootstrap5
    S sass sass-langs
```

```
python manage.py sass                  ← Don't use this code yet!
app_name/static/app_name/scss/         app's .scss files will be automatically
app_name/static/app_name/css/ --watch  converted to .css & saved at static/css
```

```
pip freeze > requirements.txt # update requirements.txt
```

💻 (venv) @ project_folder

 media & staticfiles

```
mkdir media staticfiles
```

```
# media      - for serving user's data: images, recordings, videos etc...
# staticfiles - for serving website's assets: css, js etc... in compressed form
```

project_folder/my_website/settings.py

settings.py

```
import os

from dotenv import load_dotenv # Remove this code in production
load_dotenv() # Remove this code in production

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

SECRET_KEY = os.getenv('SECRET_KEY')

DEBUG = os.getenv('DEBUG', 'False').lower() in ['true', '1']
```

```
INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    # Add libraries above apps
    'crispy_forms',
    'crispy_bootstrap5',
    'django_bootstrap5',
    'django_sass',
    ...
    # Add apps below
    'app_name.apps.AppNameConfig',
]
```

```
app_name > 🐍 apps.py
1   from django.apps import AppConfig
2
3
4   class AppNameConfig(AppConfig):
5       default_auto_field = 'django.db.models.BigAutoField'
6       name = 'app_name'
```

```
TEMPLATES = [
    ...
    'DIRS': [os.path.join(BASE_DIR, 'templates')],
    ...
]
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

```
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATIC_URL = '/static/'
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'
```

```
• B CRISPY_ALLOWED_TEMPLATE_PACKS = "bootstrap5" # Includes styled forms in B5
• B CRISPY_TEMPLATE_PACK = "bootstrap5" # Includes styled forms in B5
```

```
(venv) @ project_folder/app_name
```

```
# Update tree structure of app_name (Linux)

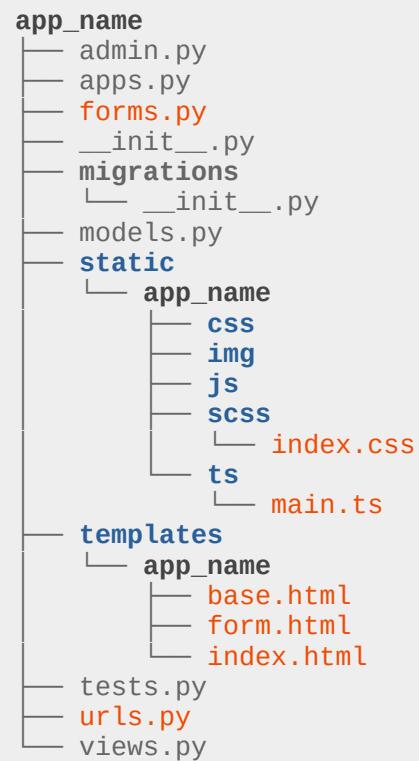
app_name=$(basename "$PWD")

touch forms.py urls.py

static="static/$app_name"
mkdir -p "$static/{scss,css,ts,js,img}"
touch "$static/scss/index.scss"
touch "$static/ts/main.ts"

temps="templates/$app_name"
mkdir -p $temps
touch "$temps/base.html"
touch "$temps/form.html"
touch "$temps/index.html"

tree .
```



```
project_folder/my_website/urls.py
```



Function views

1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path('', views.home, name='home')

Class-based views

1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

```
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
```

```
urlpatterns = [
    ... # imports URLs from my_website/app_name/urls.py
    path('', include('app_name.urls')),
]

if settings.DEBUG:
    # Serve static files during development
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
    # Serve media files during development
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

project_folder/app_name/urls.py

duck urls.py

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

urlpatterns = [
    path('', views.indexView, name='index'),
]
```

project_folder/app_name/views.py

duck views.py

```
from django.shortcuts import render

def indexView(request):
    return render(request, 'app_name/index.html')
```

project_folder/app_name/templates/app_name/index.html

</> index.html

```
{% extends 'app_name/base.html' %}
{% load static %}

{% block title %}Index Page{% endblock %}

{% block head %}
    <link rel="stylesheet" href="{% static 'app_name/css/index.css' %}">
{% endblock %}

{% block content %}
    <h2>Welcome to the Index Page!</h2>
{% endblock %}
```

```
{# this is inline comment #}

{% comment "comment name" %}
This is multi-line comment
{% endcomment %}
```

project_folder/app_name/templates/app_name/base.html

</> base.html

```
B  {% load django_bootstrap5 %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}{% endblock %}</title>
    B  {% bootstrap_css %}
        {% block head %}{% endblock %}
</head>
<body>
    <main class="container p-5">{% block content %}{% endblock %}</main>
    B  {% bootstrap_javascript %}
        {% block base %}{% endblock %}
</body>
</html>
```

Create Model with
Form Post Request from Template
TO update our Model ↗

project_folder/app_name/models.py

```
from django.db import models
from django.utils.translation import gettext_lazy as _

class ModelName(models.Model):
    id = models.AutoField(primary_key=True)

    title = models.CharField(
        max_length=100,
        blank=False,
        verbose_name=_('My Title')
    )

    COUNTRIES = [
        ('GB', 'Great Britain'),
        ('PL', 'Polska'),
        ('DE', 'Deutschland'),
    ]

    country = models.CharField(
        max_length=100,
        blank=False,
        default=COUNTRIES[1][1], # → Polska
        choices=COUNTRIES,
        verbose_name=_('Select a country')
    )
```

IGNORE BELOW CODE! It's just an example

```
class Model(models.Model):
    name = models.CharField()
    context = models.TextField()

    def __str__(self):
        return f"Model(name={self.name},context={self.context})"
```

In settings.py the **DEFAULT_AUTO_FIELD** allows you to specify the default type of auto-incrementing primary key field that will be used for models that do not explicitly define a primary key.
[DEFAULT_AUTO_FIELD @ djangoproject.com](#)

```
DEFAULT_AUTO_FIELD =
'django.db.models.BigAutoField'
```

(venv) @ project_folder manage.py

! IMPORTANT NOTE !

You should **ALWAYS** execute:

```
python manage.py makemigrations
python manage.py migrate
```

when you **create & update** models.py to initialize the model table in database

String/Text	char_field = models.CharField(max_length=100) text_field = models.TextField() url_field = models.URLField()
Numbers	integer_field = models.IntegerField() positive_integer_field = models.PositiveIntegerField() decimal_field = models.DecimalField(max_digits=5, decimal_places=2)
Time	date_field = models.DateField() time_field = models.TimeField() date_time_field = models.DateTimeField(auto_now_add=True)
Other	email_field = models.EmailField() ← max 254 chars by default boolean_field = models.BooleanField(default=False) file_field = models.FileField(upload_to='uploads/') image_field = models.ImageField(upload_to='images/')

project_folder/my_website/admin.py

admin.py

```
from django.contrib import admin
from .models import ModelName

# Register your models here TO display and manage them at the .../admin/ page
admin.site.register(ModelName)
```

project_folder/app_name/forms.py

forms.py

```
from django import forms
from .models import ModelName

class ModelForm(forms.ModelForm):
    class Meta:
        model = ModelName
        fields = ['title', 'country']
```

Select fields from ModelName
for your <form> rendered @ form.html

project_folder/app_name/views.py

views.py

```
from django.shortcuts import render, redirect
from .forms import ModelForm

def formView(request):
    if request.method == 'POST':
        form = ModelForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('index')
    else:
        form = ModelForm()

    return render(request, 'app_name/form.html', {'form': form})
```

<form method='POST'> onSubmit
Verifies posted by client data
Creates new model - ModelName
Sends user to URL path name='index'

project_folder/app_name/urls.py

urls.py

```
urlpatterns = [
    path('', views.indexView, name='index'),
    path('form/', views.formView, name='form'), # ← Add this path
]
```

project_folder/app_name/templates/app_name/form.html

</> form.html

```
{% extends 'app_name/base.html' %}
{% load static %}
{% load crispy_forms_tags %}

{% block title %}Form Page{% endblock %}

{% block content %}
    <form action="{% url 'form' %}" enctype="multipart/form-data" method="POST">
        {% csrf_token %} # Secures our data when sending it to the backend server
        B {{ form|crispy }} # Formats form in bootstrap5 & supports error events
        <button type="submit">Create Model</button>
    </form>
{% endblock %}
```

Legend:

- {} → inputs
- {} → labeled inputs
- {} → bootstrap inputs
- {} → this input
- {} → form.field_name.value
- {} → form.field_name.label
- {} → form.field_name.label_tag
- {} → form.field_name.help_text
- {} → form.field_name.errors

NOTE: My first `view` is called: `formView` (for simplicity)
However, you should rename it to `createModelView` (for semanticity)

Client lacks indication of **Model's** creation!

Let's add **MESSAGES** & display **Models**
from **Database** with **Pagination**

(Output **Models** at  Index Page)



messages - are pre-initialized by default in Django >=5



settings.py

```
INSTALLED_APPS = [
    ...
    'django.contrib.messages',
    # Add libraries above apps
    'crispy_forms',
    ...
]
```

```
MIDDLEWARE = [
    ...
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    ...
]
```



project_folder/app_name/views.py



views.py

```
from django.shortcuts import render, redirect
from django.contrib import messages
from .forms import ModelForm

def formView(request):
    if request.method == 'POST':
        form = ModelForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, 'Model created successfully!')
            return redirect('index')
    else:
        messages.error(request, 'Something went wrong!')
    else:
        form = ModelForm()

    return render(request, 'app_name/form.html', {'form': form})
```

```
messages.success()
messages.error()
messages.warning()
messages.info()
messages.debug()
```



project_folder/app_name/templates/app_name/base.html

</> base.html

```
B  {% load django_bootstrap5 %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}{% endblock %}</title>
    B  {% bootstrap_css %}
    {% block head %}{% endblock %}
</head>
<body>
    B  {% if messages %}B  {% bootstrap_messages %}{% endif %}
    <main name="container p-5">{% block content %}{% endblock %}</main>
    B  {% bootstrap_javascript %}
    {% block base %}{% endblock %}
</body>
</html>
```

```
    {{ messages }}
    {{ message }}      → 'My Message'
    {{ message.tags }} → success, error, info...
```

```

from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
from django.shortcuts import render, redirect
from django.contrib import messages
from .forms import ModelForm
from .models import ModelName


def indexView(request):

    # ASCENDING:(a-z) - order_by('field')
    # DESCENDING:(z-a) - order_by('-field')
    model_list = ModelName.objects.all().order_by('title')

    # Paginate the queryset
    paginator = Paginator(model_list, 5)      # Show 5 models per page
    page_number = request.GET.get('page', 1) # Get 'page' from request's query:
                                            # localhost:port/?page=0-9 (number)

    try:
        models = paginator.page(page_number)
    except PageNotAnInteger:
        # If page is not an integer, deliver the first page
        models = paginator.page(1)
    except EmptyPage:
        # If page is out of range (e.g., 9999), deliver the last page
        models = paginator.page(paginator.num_pages)

    return render(request, 'app_name/index.html', {'models': models})

```

```

{% extends 'app_name/base.html' %}
{% load static %}

{B} {% load django_bootstrap5 %}

{% block title %}Index Page{% endblock %}

{% block head %}
    <link rel="stylesheet" href="{% static 'app_name/css/index.css' %}">
{% endblock %}
                    ... (more on next page)

```

project_folder/app_name/templates/app_name/index.html </> index.html

```
{% block content %}

<h2>Welcome to the Index Page!</h2>

{% if models %}
    <ul class="list-group pb-3">
        {% for model in models %}
            <li class="list-group-item">
                <h2>{{ model.title }}</h2>
                <p>{{ model.country }}</p>
                <small class="text-muted">{{ model.created_at }}</small>
            </li>
        {% endfor %}
    </ul>
    <!-- PAGINATION -->
{% endif %}

{% endblock %}
```

Pagination Example_1 django-bootstrap5

```
<div class="pagination">
    <span class="links">

        {% if models.has_previous %}
            <a href="?page=1">&laquo; First</a>
            <a href="?page={{ models.previous_page_number }}">Previous</a>
        {% endif %}

        <span class="current">
            Page {{ models.number }} of {{ models.paginator.num_pages }}
        </span>

        {% if models.has_next %}
            <a href="?page={{ models.next_page_number }}">Next</a>
            <a href="?page={{ models.paginator.num_pages }}">Last &raquo;</a>
        {% endif %}

    </span>
</div>
```

Pagination Example_2 django-bootstrap5

```
{% bootstrap_pagination models %}
```



Edit & Delete Models

project_folder/app_name/urls.py

urls.py (BOTH)

```
# Add below paths    integer (\d+), argument: model_id (it's a custom name)

path('edit-model/<int:model_id>', views.editModelView, name='edit_model'),
path('delete-model/<int:model_id>', views.deleteModelView, name='delete_model'),

# URL example - localhost:8080/delete-model/2
```

project_folder/app_name/views.py

views.py (DELETE)

```
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from .forms import ModelForm
from .models import ModelName
```

```
def deleteModelView(request, model_id):
    model_instance = get_object_or_404(ModelName, pk=model_id)
    model_instance.delete() # ← Removes Model

    messages.warning(request, f'Deleted model: {model_instance.title}')

    return redirect('index') # CALLS path(... name='index') at urls.py
```

project_folder/app_name/templates/app_name/index.html </> index.html (BOTH)

```
{% block content %}
...
{% for model in models %}
    <li class="list-group-item">
        <h2>{{ model.title }}</h2>
        <p>{{ model.country }}</p>
        <small class="text-muted">{{ model.created_at }}</small>
        <section class="d-flex justify-content-between">
            <a href="{% url 'edit_model' model.id %}">
                <button class="btn btn-secondary">Edit</button>
            </a>
            <a href="{% url 'delete_model' model.id %}">
                <button class="btn btn-danger">Delete</button>
            </a>
        </section>
    </li>
{% endfor %}
...
{% endblock %}
```

```

from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from .forms import ModelForm
from .models import ModelName


def editModelView(request, model_id):
    model_instance = get_object_or_404(ModelName, pk=model_id)

    if request.method == 'POST':
        form = ModelForm(request.POST, instance=model_instance)

        if form.is_valid():
            form.save() # ← Updates Model
            messages.success(request, f'Saved model as: {model_instance.title}')

            return redirect('index')
        else:
            messages.error(request, f'Model not saved: {model_instance.title}')
    else:
        form = ModelForm(instance=model_instance)

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

```

```

...
{% block content %}
<form action="{% url 'form' %}" enctype="multipart/form-data" method="POST">
    {% csrf_token %} # Secures our data when sending it to the backend server
    B {{ form|crispy }} # Formats form in bootstrap5 & supports error events
    <button type="submit">Create Model</button>
</form>

<form action="{% if form.instance.id %}{% url 'edit_model' form.instance.id %}"
      {% else %}{% url 'form' %}{% endif %}" enctype="multipart/form-data"
      method="POST">
    {% csrf_token %}
    B {{ form|crispy }}

    <button class="btn btn-success" type="submit">
        {% if form.instance.id %}
        <span>Update Model</span>
        {% else %}
        <span>Create Model</span>
        {% endif %}
    </button>
</form>
{% endblock %}

```

Register and Login new Users



Create Users App

💻 (venv) @ project_folder

🐍 manage.py

```
python manage.py startapp users
```

Register Users App and it's **Urls** to Django Website

📝 project_folder/my_website/settings.py

🐍 settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    # Add libraries above apps
    'crispy_forms',
    'crispy_bootstrap5',
    ...
    # Add apps below
    'app_name.apps.AppNameConfig',
    'users.apps.UsersConfig',
]

...
LOGIN_REDIRECT_URL = 'index' # Redirects to path(... name='index') on
                            # successful form POST from auth_views.LoginView
LOGOUT_REDIRECT_URL = 'login' # Redirects to path(... name='login') on
                            # successful form POST from auth_views.LogoutView
```

📝 project_folder/my_website/urls.py

🐍 urls.py

```
...
urlpatterns = [
    ...
    path('', include('app_name.urls')),

    # imports URLs from my_website/users/urls.py
    path('users/', include('users.urls')),
]
```

Django has build-in **generic Models, Forms & Views** for Users App

User	(Build-in Model)
username, first_name, last_name, email, password (password: encrypted / hashed) groups, user_permissions, is_staff, is_active, is_superuser ← Boolean (True False) last_login, date_joined ← Date & Time fields	
UserCreationForm	(Build-in Form)
password1, password2 ← temporary fields used to create password	
auth_views	(Build-in Class View)

project_folder/users/forms.py	(register users) forms.py
from django import forms from django.contrib.auth.models import User from django.contrib.auth.forms import UserCreationForm	
<pre>class UserRegisterForm(UserCreationForm): email = forms.EmailField(max_length=255) first_name = forms.CharField(max_length=35, label='Forename') last_name = forms.CharField(max_length=35, label='Surname') class Meta: model = User fields = ['username', 'first_name', 'last_name', 'email', 'password1', 'password2']</pre>	
<p>User model's build-in fields: email, first_name & last_name, are overwritten in this form</p>	

project_folder/users/urls.py

(users) 🐍 urls.py

```
from django.urls import path
from django.contrib.auth import views as auth_views # ← (Build-in Class-View)
from . import views

urlpatterns = [
    path('login', auth_views.LoginView.as_view(template_name='users/form.html'),
         name='login'),
    path('logout', auth_views.LogoutView.as_view(), name='logout'),
    path('register', views.registerUserView, name='register'),
]
```

project_folder/users/views.py

(users) 🐍 views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login
from django.contrib import messages
from .forms import UserRegisterForm

def registerUserView(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        if form.is_valid():

            # Register the user, (and returns user instance)
            user = form.save()
            messages.success(request, f'Created account: {user.username}')

            # Authenticate the user
            raw_password = form.cleaned_data.get('password1')
            user = authenticate(username=user.username, password=raw_password)

            # Sign-in new user
            if user is not None:
                login(request, user)
                messages.success(request, f'Logged-in: {user.username}')
                return redirect('index')
            else:
                messages.error(request, 'Failed to login user')
                return redirect('login')

        else:
            messages.error(request, 'Invalid credentials!')
    else:
        form = UserRegisterForm()

    return render(request, 'users/form.html', {'form': form, 'registry': True})
```

 project_folder/users/templates/users/form.html </> form.html

```

{% extends 'app_name/base.html' %}
{% load static %}
{% load crispy_forms_tags %}

{% block title %}
    {% if registry %}Register{% else %}Login{% endif %} Page
{% endblock %}



---


{% block content %}

    {% if user.is_authenticated and not registry %}
        <div class="p-3 mb-5 bg-light rounded">
            <p>You are already logged-in as: <strong>{{ user }}</strong>
            <br>would you like to sign-in to another account?</p>
        </div>
    {% endif %}



---


<form method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    B {{ form|crispy }}
    <button type="submit" class="btn btn-success">
        <span>% if registry %}Register{% else %}Login{% endif %}</span>
    </button>
</form>



---


<section class="float-end">
    {% if not registry %}
        <p>Don't have an account? <a href="{% url 'register' %}">register now</a></p>
    {% else %}
        <p>Already have an account? <a href="{% url 'login' %}">login now</a></p>
    {% endif %}
</section>

{% endblock %}

```

 project_folder/app_name/templates/app_name/index.html </> index.html

```

{% block content %}
    ...
<a href="{% url 'form' %}">add model</a>

<form action="{% url 'logout' %}" method="POST">
    {% csrf_token %}
    <button type="submit" class="btn btn-warning">logout</button>
</form>

{% endblock %}

```

NOTES

`authenticate()` from `django.contrib.auth`
adds property: `is_authenticated` to `user`

Hence: `{% if user.is_authenticated %}`

`auth_view.LogoutView` requires `POST` method, due to security reasons

Hence: `<form action="{% url 'logout' %}" method="POST">` ✓

instead of: `` ✗

Secure Views 🔒

Anyone can access **views** to create, edit & delete **models**. Let's secure them!

 project_folder/app_name/views.py

 views.py

```
def formView(request):  
  
    if not request.user.is_superuser:  
        messages.error(request, 'you must be logged-in as a super user')  
        return redirect('index') # redirect client to URL path(... name='index') at  urls.py  
  
    if request.method == 'POST':  
        form = ModelForm(request.POST)  
        if form.is_valid():  
            form.save()  
            return redirect('index')  
    else:  
        form = ModelForm()  
  
    return render(request, 'app_name/form.html', {'form': form})
```

In above example, only users with special privilege may access a **formView()**
This approach is ideal for a website with **single-user** in mind



Let's secure the **view** for **multi-user-based** platform

 project_folder/app_name/views.py

 views.py

```
from django.contrib.auth.decorators import login_required  
  
@login_required(login_url='login')  
def formView(request):  
  
    if request.method == 'POST':  
        form = ModelForm(request.POST)  
        if form.is_valid():  
            form.save()  
            return redirect('index')  
    else:  
        form = ModelForm()  
  
    return render(request, 'app_name/form.html', {'form': form})
```

decorator: `@login_required(login_url='login')` redirects client to
URL path(... name='login') at  urls.py

 project_folder/my_website/settings.py

 settings.py

```
LOGIN_URL = 'login'
```

NOT logged-in users trying to access to restricted view: `@login_required`
are redirected to → URL path(... name='login') by default.

```
@login_required(login_url='login')
@login_required
def formView(request):
    ...
```

Profile Model



💻 (venv) @ project_folder

pip library

```
pip install pillow django-resized  
pip freeze > requirements.txt # update requirements.txt
```

📝 project_folder/my_website/settings.py

🐍 settings.py

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    ...  
    # Add libraries above apps  
    'django_resized',  
    'crispy_forms',  
    ...  
]
```

📝 project_folder/users/models.py

(profile users) 🐍 models.py

```
from django.db import models  
from django.contrib.auth.models import User  
from django_resized import ResizedImageField
```

```
class Profile(models.Model):
```

```
    user = models.OneToOneField(User, on_delete=models.CASCADE)  
    phone_number = models.CharField(max_length=15)  
  
    image = ResizedImageField(  
        size=[300, 300],  
        crop=['middle', 'center'],  
        quality=75,  
        force_format='JPEG',  
  
        # 📂 /profile_pics  
        upload_to='profile_pics',  
        default='default_profile.jpg'  
)
```



models.CASCADE → Delete this model IF its referenced model is deleted

📁 /profile_pics is created at 📁 /media → 📁 /media/profile_pics

1. Register model: **Profile** in → 🐍 users/admin.py &

2. Using 🖥 terminal: makemigrations + migrate them to the database → 🏟

3. Place: 📸 **default_profile.jpg** inside 📁 /media directory

 project_folder/users/forms.py

(profile users)  forms.py

```
from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from .models import Profile

class ProfileForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ['phone_number']
...
```

 project_folder/users/views.py

(profile users)  views.py

```
...
from .forms import UserRegisterForm, ProfileForm

def registerUserView(request):
    if request.method == 'POST':
        form_user = UserRegisterForm(request.POST)
        form_profile = ProfileForm(request.POST)

        if form.is_valid() and form_profile.is_valid():

            # Register the user, (and returns user instance)
            user = form_user.save()

            # Create Profile model for new user
            profile = form_profile.save(commit=False)
            profile.user = user
            profile.save()

            messages.success(request, f'Created account: {user.username}')

            # Authenticate the user
            raw_password = form_user.cleaned_data.get('password1')
            ...

    ...
    else:
        form_user = UserRegisterForm()
        form_profile = ProfileForm()

    context = {
        'registry': True,
        'register_view_forms': {
            'user': form_user,
            'profile': form_profile
        }
    }

    return render(request, 'users/form.html', context)
```

 project_folder/users/templates/users/**form.html** </> form.html

```
...
{% block content %}
    ...
    <form method="POST" enctype="multipart/form-data">
        {% csrf_token %}
        B {{ register_view_forms.profile|crispy }}
        B {{ register_view_forms.user|crispy }}
        B {{ form|crispy }} # ← auth_views.LoginView ← URL path(... name='login')
    ...
{% endblock %}
```

Update Profile Details Page



 project_folder/users/forms.py

(profile users)  forms.py

```
class UserUpdateForm(forms.ModelForm):

    email = forms.EmailField(max_length=255)
    first_name = forms.CharField(min_length=4, max_length=35, label='Forename')
    last_name = forms.CharField(min_length=4, max_length=35, label='Surname')

    confirm_password = forms.CharField(
        widget=forms.PasswordInput,
        help_text='Surname'
    )

    class Meta:
        model = User
        fields = [
            'username',
            'email',
            'first_name',
            'last_name',
            'confirm_password'
        ]
```

 project_folder/users/urls.py

(profile users)  urls.py

```
from django.urls import path
from django.contrib.auth import views as auth_views
from . import views
```

```
urlpatterns = [
    path('login', auth_views.LoginView.as_view(template_name='users/form.html'),...name='login'),
    path('logout', auth_views.LogoutView.as_view(), name='logout'),
    path('register', views.registerUserView, name='register'),
    path('profile/<int:user_id>', views.profileView, name='profile'),
]
```

```

from django.contrib.auth.decorators import login_required
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import authenticate, login
from django.contrib.auth.hashers import check_password
from django.contrib.auth.models import User
from django.contrib import messages
from .forms import UserRegisterForm, UserUpdateForm, ProfileForm
from .models import Profile

user_instance : User Model with ID → user_id

@login_required
def profileView(request, user_id):
    user_instance = get_object_or_404(User, pk=user_id)

    if request.method == 'POST' and request.user.id == user_instance.id:
        request.user.id : logged-in user's ID

        form_user = UserUpdateForm(
            request.POST,
            instance=user_instance
        )
        form_profile = ProfileForm(
            request.POST,
            request.FILES,
            instance=user_instance.profile
        )
        confirm_password : value of <input name="confirm_password">

        if form_user.is_valid() and form_profile.is_valid():
            confirm_password = form_user.cleaned_data.get('confirm_password')

            if check_password(confirm_password, user_instance.password):
                form_user.save()
                form_profile.save()
                messages.success(request, 'Successfully updated profile')
            else:
                messages.error(request, 'Invalid password!')
                form_user.add_error('confirm_password', 'Invalid password!')
        else:
            messages.error(request, 'Invalid credentials!')

        .add_error() → form|crispy → RED <input name="confirm_password"> + Error

    else:
        form_user = UserUpdateForm(instance=user_instance)
        form_profile = ProfileForm(instance=user_instance.profile)

    context = {
        'profile_user': user_instance,
        'forms': {
            'user': form_user,
            'profile': form_profile
        }
    }

    return render(request, 'users/profile.html', context)

```

 project_folder/users/templates/users/profile.html </> profile.html

```
{% extends 'app_name/base.html' %}  
{% load static %}  
{% load crispy_forms_tags %}  
{% block title %}Profile Page{% endblock %}  
{% block content %}  
<div class="container">  
    <div class="row">  
        <aside class="col-md-4">  
            <div class="p-3 bg-secondary text-white">  
                  
            <div class="p-3 bg-light">  
                {% if user.id == profile_user.id %}  
                    <form action="{% url 'profile' profile_user.id %}"  
                          enctype="multipart/form-data"  
                          method="POST"  
                    >  
                        {% csrf_token %}  
                        {{ forms.profile|crispy }}  
                        {{ forms.user|crispy }}  
                        <button class="btn btn-warning" type="submit">Update Profile</button>  
                    </form>  
                {% else %}  
                    {% for field in forms.user %}  
                        {% if field.label|lower != "confirm password" %} <!-- Disclude: confirm_password --&gt;<br/>                            <div class="mb-3">  
                                <label for="{{ field.id_for_label }}" class="form-label">{{ field.label }}</label>  
                                <p class="bg-white p-3" id="{{ field.id_for_label }}>{{ field.value }}</p>  
                            </div>  
                        {% endif %}  
                    {% endfor %}  
                    {% for field in forms.profile %}  
                        {% if field.label|lower != "image" %} <!-- Disclude: image --&gt;<br/>                            <div class="mb-3">  
                                <label for="{{ field.id_for_label }}" class="form-label">{{ field.label }}</label>  
                                <p class="bg-white p-3" id="{{ field.id_for_label }}>{{ field.value }}</p>  
                            </div>  
                        {% endif %}  
                    {% endfor %}  
                {% endif %}  
            </div>  
        </main>  
    </div>  
{% endblock %}
```

 project_folder/app_name/templates/app_name/index.html </> index.html

```
...
<a href="{% url 'form' %}" class="float-start">add model</a>

<a href="{% if request.user.is_authenticated %}{% url 'profile' request.user.id %}{% else %}{% url 'login' %}{% endif %}" class="float-end">
    <button type="submit" class="btn btn-primary ms-3">profile</button>
</a>

<form action="{% url 'logout' %}" method="POST" class="float-end">
    {% csrf_token %}
    <button type="submit" class="btn btn-warning">logout</button>
</form>

...
```



Update User's password using custom Email





Zoho Mail

Features ▾ Pricing Enterprise Resources ▾ Partner with us ▾ Contact Us

TRY NOW



Forever Free Plan
Up to five users, 5GB/User, 25MB attachment limit.
Web access and free mobile apps*.
Email hosting for single domain.

SIGN UP NOW



Contact Sales
Get custom plans for bulk users and enterprises.

CONTACT US



Flexible pricing
Mix and match different plans for different users in your organization.

EXPLORE NOW

*IMAP/ POP/ Active Sync are not included in the free plan.

Email is required for this stage
Hence, I will be using Zoho mail's

Forever Free Plan

(you can also create and manage additional emails via admin user)

admin@... , noreply@... , support@...

However, you may need to buy a **domain** for custom emails



OR



HOSTINGER

Hostinger provides domains and can host websites

I personally recommend **NameCheap** service as it's easy to manage

A screenshot of the Namecheap web interface. The top navigation bar includes links for Domains, Hosting, WordPress, Email, Apps, Security, Transfer to Us, Help Center, and Account. On the left is a sidebar with links for Dashboard, Expiring / Expired, Domain List (which is highlighted in teal), Hosting List, Private Email, SSL Certificates, Apps, and Profile. The main content area shows the 'Domains → Details' page for the domain 'cocosoft.club'. It features tabs for Domain, Products, Sharing & Transfer, and Advanced DNS (which is active). Below these are sections for DNS TEMPLATES and HOST RECORDS, each with a 'Actions' dropdown and a 'Filters' dropdown. A table at the bottom lists DNS records with columns for Type, Host, Value, and TTL. The message 'No Records Found' is displayed. At the bottom left of this section is a red arrow pointing to a red '+ ADD NEW RECORD' button.



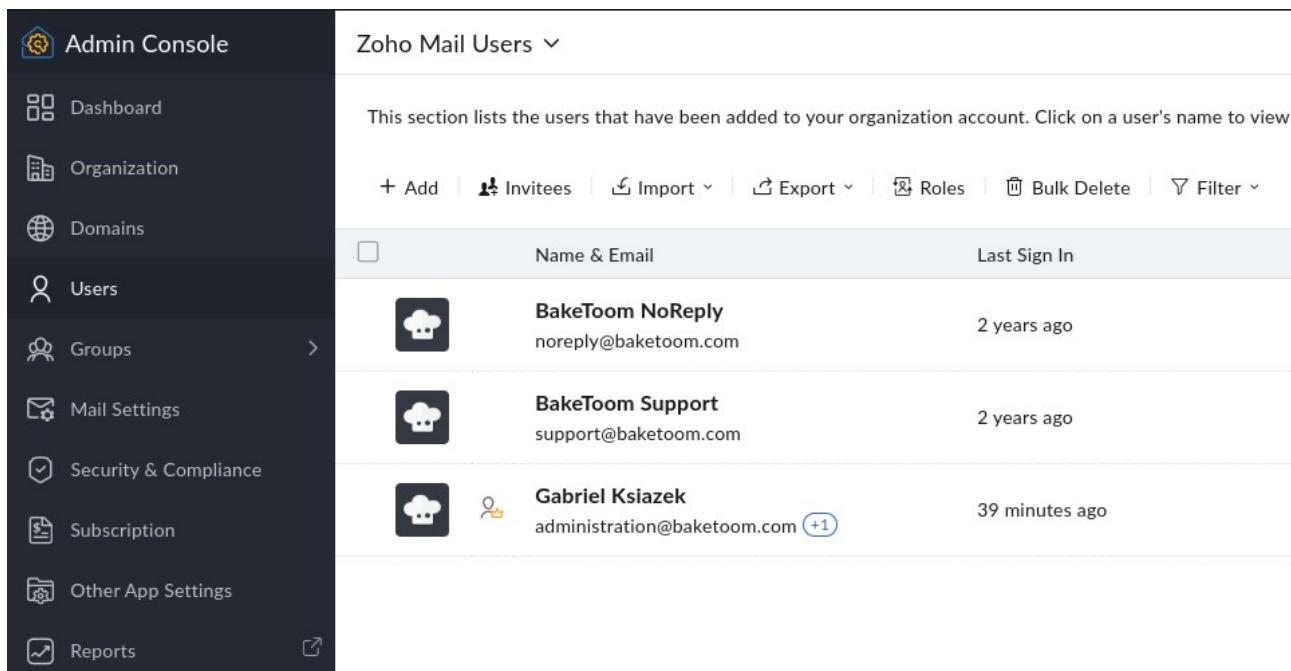
Domain can be linked to
Email & Web-Hosting, Services



Zoho Admin Console

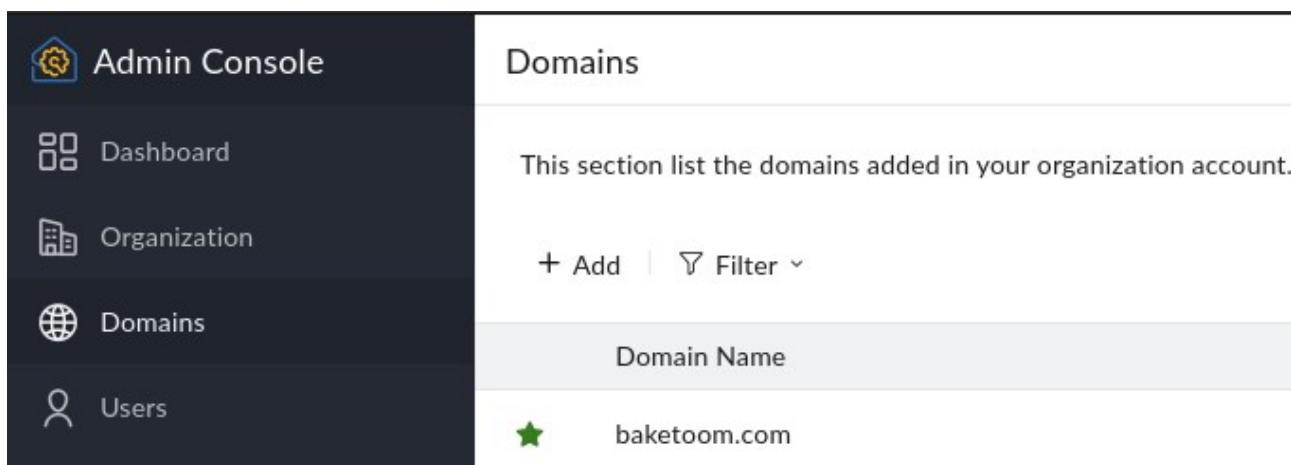
mailadmin.zoho.com | mailadmin.zoho.eu

Email Hosting Guide (link at dashboard)
<https://www.zoho.com/mail/help/adminconsole/email-hosting-setup.html>



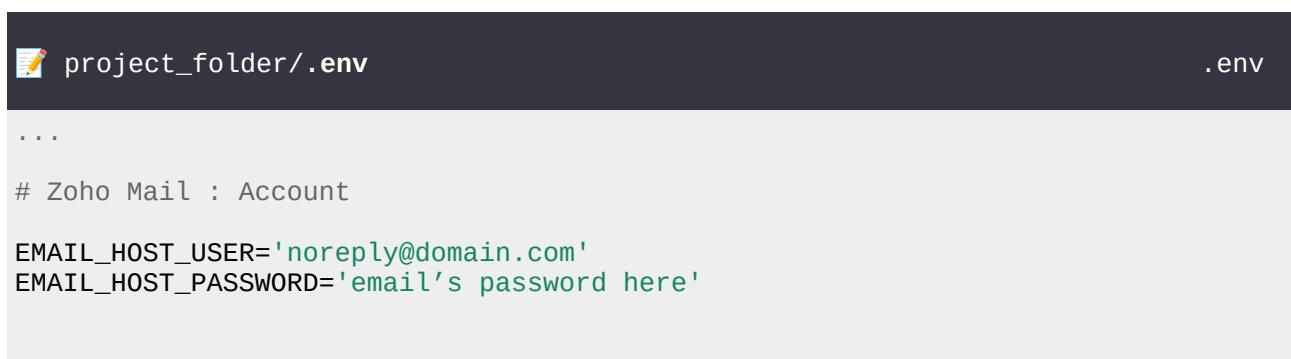
The screenshot shows the 'Zoho Mail Users' section of the Zoho Admin Console. The left sidebar includes links for Dashboard, Organization, Domains, Users, Groups, Mail Settings, Security & Compliance, Subscription, Other App Settings, and Reports. The main content area displays a list of users with columns for Name & Email and Last Sign In. Three users are listed: 'BakeToom NoReply' (noreply@baketoom.com), 'BakeToom Support' (support@baketoom.com), and 'Gabriel Ksiazek' (administration@baketoom.com). A '+ Add' button and a 'Filter' dropdown are also present.

	Name & Email	Last Sign In
	BakeToom NoReply noreply@baketoom.com	2 years ago
	BakeToom Support support@baketoom.com	2 years ago
	Gabriel Ksiazek administration@baketoom.com <small>+1</small>	39 minutes ago



The screenshot shows the 'Domains' section of the Zoho Admin Console. The left sidebar includes links for Dashboard, Organization, Domains, and Users. The main content area displays a list of domains with a 'Domain Name' column. One domain, 'baketoom.com', is listed with a star icon next to it. A '+ Add' button and a 'Filter' dropdown are also present.

Domain Name
baketoom.com



```
project_folder/.env .env
...
# Zoho Mail : Account

EMAIL_HOST_USER='noreply@domain.com'
EMAIL_HOST_PASSWORD='email's password here'
```

project_folder/my_website/settings.py

settings.py

```
...  
  
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'  
EMAIL_HOST = 'smtp.zoho.eu'  
EMAIL_PORT = 587  
EMAIL_USE_TLS = True  
EMAIL_USE_SSL = False # Avoid SSL because it's less secure  
  
EMAIL_HOST_USER = os.getenv('EMAIL_HOST_USER')  
EMAIL_HOST_PASSWORD = os.getenv('EMAIL_HOST_PASSWORD')  
DEFAULT_FROM_EMAIL = f'(Domain) <{EMAIL_HOST_USER}>'  
  
# DEFAULT_FROM_EMAIL : (Domain) <noreply@domain.com>
```

project_folder/users/urls.py

(profile users) urls.py

```
from django.urls import path  
from django.contrib.auth import views as auth_views  
from . import views  
  
  
urlpatterns = [  
    path('login', auth_views.LoginView.as_view(template_name='users/form.html'),  
         name='login'),  
    path('logout', auth_views.LogoutView.as_view(), name='logout'),  
    path('register', views.registerUserView, name='register'),  
  
    path('password-reset/',  
        auth_views.PasswordResetView.as_view(  
            template_name='users/password_reset.html'),  
        name='password_reset'),  
  
    path('password-reset/done/',  
        auth_views.PasswordResetDoneView.as_view(  
            template_name='users/password_reset_done.html'),  
        name='password_reset_done'),  
  
    path('password-reset-confirm/<uidb64>/<token>/',  
        auth_views.PasswordResetConfirmView.as_view(  
            template_name='users/password_reset_confirm.html'),  
        name='password_reset_confirm'),  
  
    path('password-reset-complete/',  
        auth_views.PasswordResetCompleteView.as_view(  
            template_name='users/password_reset_complete.html'),  
        name='password_reset_complete'),  
]  
]
```

 project_folder/users/templates/users/profile.html </> profile.html

```
...
<form action="{% url 'profile' profile_user.id %}"
      enctype="multipart/form-data"
      method="POST"
>
    {% csrf_token %}
{B} {{ forms.profile|crispy }}
{B} {{ forms.user|crispy }}
    <button class="btn btn-warning" type="submit">Update Profile</button>
    <div class="row justify-content-between">
        <div class="col-auto">
            <button class="btn btn-warning" type="submit">Update
Profile</button>
        </div>
        <div class="col-auto">
            <a href="{% url 'password_reset' %}" class="btn btn-link">change
password</a>
        </div>
    </div>
</form>
...

```

 project_folder/users/templates/users/password_reset.html </>

```
{% extends 'app_name/base.html' %}
{% load static %}
{% load crispy_forms_tags %}
{% block title %}Reset Password{% endblock %}

{% block content %}

<form action="{% url 'password_reset' %}"
      enctype="multipart/form-data"
      method="POST"
>
    {% csrf_token %}
{B} {{ form|crispy }}
    <button type="submit" class="btn btn-warning">
        <span>Request Password Change</span>
    </button>
    <a href="javascript:history.back()" class="float-end">
        <span>Return to previous page</span>
    </a>
</form>
{% endblock %}
```



project_folder/users/templates/users/password_reset_done.html

</>

```
{% extends 'app_name/base.html' %}  
{% load static %}  
{% block title %}Reset Password{% endblock %}  
  
{% block content %}  
  
<div class="mx-auto" style="max-width: 750px">  
    <section class="bg-light p-3 mb-3">  
        <h1 class="fs-3">Request sent</h1>  
        <p>An email with instructions to reset your password has been sent to  
the email address you entered.</p>  
    </section>  
    <section class="d-flex justify-content-between">  
        <a href="{% url 'password_reset' %}">  
            <button type="submit" class="btn btn-warning">  
                <span>Resend request</span>  
            </button>  
        </a>  
        <a href="{% url 'index' %}">  
            <button type="submit" class="btn btn-primary">  
                <span>Continue</span>  
            </button>  
        </a>  
    </section>  
</div>  
  
{% endblock %}
```

 project_folder/users/templates/users/password_reset_confirm.html </>

```
{% extends 'app_name/base.html' %}  
{% load static %}  
{% load crispy_forms_tags %}  
{% block title %}Reset Password{% endblock %}  
{% block content %}  
  
{% if form %}  
<form method="POST" enctype="multipart/form-data">  
    {% csrf_token %}  
    B {{ form|crispy }}  
    <button type="submit" class="btn btn-warning">  
        <span>Change Password</span>  
    </button>  
</form>  
{% else %}  
<div class="mx-auto" style="max-width: 750px">  
    <section class="bg-light p-3 mb-3">  
        <h1 class="fs-3">Request sent</h1>  
        <p>ERROR 404 - Something went lost in the dust of the universe.</p>  
    </section>  
</div>  
{% endif %}  
  
{% endblock %}
```

 project_folder/users/templates/users/password_reset_complete.html </>

```
{% extends 'app_name/base.html' %}  
{% load static %}  
{% block title %}Reset Password{% endblock %}  
{% block content %}  
  
<div class="mx-auto" style="max-width: 750px">  
    <section class="bg-light p-3 mb-3">  
        <h1 class="fs-3">Success!</h1>  
        <p>Your password has been updated successfully.</p>  
    </section>  
    <a href="{% url 'index' %}">  
        <button type="submit" class="btn btn-primary">  
            <span>Continue to main page</span>  
        </button>  
    </a>  
</div>  
  
{% endblock %}
```

Models are modifiable by
all logged-in users

Let's Create, Edit & Delete
User's Models only!



project_folder/app_name/models.py

models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.utils.translation import gettext_lazy as _

class ModelName(models.Model):
    id = models.AutoField(primary_key=True)
    creator = models.ForeignKey(User, on_delete=models.CASCADE)

    title = models.CharField(
        max_length=100,
    ...

```

(venv) @ project_folder

manage.py # IGNORE BELOW CODE

(Reminder Example)

! IMPORTANT NOTE !

You should **ALWAYS** execute:

```
python manage.py makemigrations
python manage.py migrate
```

when you **create & update** models.py
to initialize the model table in database

```
class Model(models.Model):
    name = models.CharField()
    context = models.TextField()

    def __str__(self):
        return f"Model(name={self.name},context={self.context})"
```

In settings.py - The **DEFAULT_AUTO_FIELD** allows you to specify the default type of auto-incrementing primary key field that will be used for models that do not explicitly define a primary key.
[DEFAULT_AUTO_FIELD @ djangoproject.com](#)

```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

(venv) @ project_folder

manage.py

Sometimes, you may have to **remove migrations**: `NNNN_initial.py` from app_name
you can also make migrations with `--fake` flag to resolve conflicts (- AVOID!)

```
python manage.py migrate --fake           ← Avoid using: --fake OR --fake-initial
python manage.py migrate --fake-initial Used for integrating existing db schema
```

```
python manage.py migrate --fake app_name zero   ← Use these commands when you
python manage.py makemigrations app_name          can't fix conflicts and want
python manage.py migrate app_name                to preserve your database data
```

(venv) @ project_folder

manage.py (shell)

We also have old instances of model: `ModelName`.
Update their **creator field** manually **OR delete** them all:

```
python manage.py shell
(shell) >> from app_name.models import ModelName
(shell) >> ModelName.objects.all().delete()
(shell) >> exit()
```

```
...
from django.contrib.auth.decorators import login_required
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from .forms import ModelForm
from .models import ModelName



---



@login_required
def formView(request):
    if request.method == 'POST': # CreateModelView
        form = ModelForm(request.POST)
        if form.is_valid():
            form.save()
            model_instance = form.save(commit=False)
            model_instance.creator = request.user
            model_instance.save()

            messages.success(request, 'Model created successfully!')
            return redirect('index')
        else:
            messages.error(request, 'Failed to create new model!')
    else:
        form = ModelForm()

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


...

```

```
# (OPTIONAL EXAMPLE) - declare custom functions in views.py
#   use: user_instance == model_instance.creator      # (for simplicity)
# or use: is_model_creator(model_instance, user_instance) # (for semanticity)
```

```
def is_model_creator(model_instance, user_instance):
    return user_instance == model_instance.creator # → True | False
```

```
from django.core.exceptions import ValidationError
from django.db.models import Model

def is_model_creator(model_instance=None, user_instance=None):
    if model_instance is None or user_instance is None:
        raise ValueError("Both 'model_instance' and 'user_instance' must be provided")

    if not isinstance(model_instance, Model):
        raise ValidationError(f"{model_instance} is not a valid Django model instance")

    if not hasattr(model_instance, 'creator'):
        raise ValidationError(f"{model_instance} does not have a 'creator' field")

    return user_instance == model_instance.creator
```

```
...  
  
@login_required  
def editModelView(request, model_id):  
    model_instance = get_object_or_404(ModelName, pk=model_id)  
  
    if not is_model_creator(model_instance, request.user):  
        messages.error(request, 'You are not the owner of this model')  
        return redirect('index')  
  
    if request.method == 'POST':  
  
        ...  
  
@login_required  
def deleteModelView(request, model_id):  
    model_instance = get_object_or_404(ModelName, pk=model_id)  
  
    if not is_model_creator(model_instance, request.user):  
        messages.error(request, 'You are not the owner of this model')  
        return redirect('index')  
  
    model_instance.delete()  
    messages.warning(request, f'Deleted model: {model_instance.title}')  
  
    return redirect('index')
```

Search for Users' Models 

 project_folder/app_name/templates/app_name/index.html </> index.html

```
<form method="POST" action="{% url 'index' %}{% if request.GET %}?{% for key, value in request.GET.items %}{{ key }}={{ value }}{% if not forloop.last %}&{% endif %}{% endfor %}{% endif %}">

    {% csrf_token %}
    <div class="input-group py-3">

        <input type="search"
            placeholder="Search"
            class="form-control rounded-start"
            name="search_query">

        <button type="submit" class="btn btn-outline-primary">search</button>

    </div>
</form>
```

Include Search bar 

 project_folder/app_name/views.py

 views.py

```
...

def indexView(request):

    model_list = ModelName.objects.all()

    if request.method == 'POST':
        search = request.POST.get('search_query')
        if search:
            model_list = model_list.filter(title__icontains=search)

#localhost:8000/?country_code=PL&creator=McRaZick → request.GET.items()

    for key, value in request.GET.items():
        if key == 'country_code':                      # PL
            model_list = model_list.filter(country=value)
        elif key == 'creator_id':                     #
            model_list = model_list.filter(creator=value)
        elif key == 'creator':                         # McRa...
            model_list = model_list.filter(creator__username__icontains=value)
    ...

title, country & creator are fields of model: ModelName @ app_name.models.py
```

Django Lookup for ORM (Object-Relational Mapping):
`.filter(field__lookup)`

Queryset Lookups

Category	Regexp Equivalent	Lookup
Exact match	<code>^value\$</code>	<code>field__exact="value"</code>
Case-insensitive Exact match	<code>(?i)^value\$</code>	<code>field__iexact="value"</code>
Contains	<code>.*value.*</code>	<code>field__contains="value"</code>
Case-insensitive Contains	<code>(?i).*value.*</code>	<code>field__icontains="value"</code>
In		<code>field__in=[value1, value2, ...]</code>
>		<code>field__gt=value</code>
>=		<code>field__gte=value</code>
<		<code>field__lt=value</code>
<=		<code>field__lte=value</code>
Starts with	<code>^value.*</code>	<code>field__startswith="value"</code>
Case-insensitive Starts with	<code>(?i)^value.*</code>	<code>field__istartswith="value"</code>
Ends with	<code>.*value\$</code>	<code>field__endswith="value"</code>
Case-insensitive Ends with	<code>(?i).*value\$</code>	<code>field__iendswith="value"</code>
Range (From → To)		<code>field__range=(start_value, end_value)</code>

The **code** we've implemented is decent but it **is also faulty**. For example:

When client uses pagination buttons, we will only receive:
`localhost:port/?page=N` in our URL.

By adding query parameter: `&country_code=GB`,
and heading to the next page, using template's pagination button,

we expect our URL to become: `localhost:port/?page=N&country_code=GB`.

However, we are redirected to: `localhost/?page=N+1`, instead.

This is an **issue** because additional **filters** will **not be applied**.

Another issue is with the `<form action ...>`. It's cool that we can include any query parameter and also use a search `<button>` to also filter the models' title.

However, this is **impractical**, because if client decides to make a new search, then the additional filters will not be removed. It's more practical to include additional filters using radio buttons and checkboxes.

 project_folder/app_name/templates/app_name/index.html </> index.html

```
<form method="POST" action="{% url 'index' %}{% if request.GET %}?{{ for key, value in request.GET.items }}{{ key }}={{ value }}{% if not forloop.last %}&{% endif %}{% endfor %}{% endif %}">
    ...
</form>
```

Remove the above code &
Use django_bootstrap5 library's pagination, with `url` argument like so:

 {B} {% bootstrap_pagination models url=request.build_absolute_uri %}

 project_folder/app_name/views.py

 views.py

```
from django.urls import reverse
from django.http import HttpResponseRedirect

...
def indexView(request):

    model_list = ModelName.objects.all()

    if request.method == 'POST':
        search = request.POST.get('search_query')
        if search:
            # Note: request.GET is immutable, hence we need to create a mutable copy.
            query_params = request.GET.copy() # query_params is a mutable copy
            query_params['search'] = search # add search parameter to request copy

            new_url = reverse('index') + "?" + query_params.urlencode()

            # return back to the previous render, with updated url
            return HttpResponseRedirect(new_url)

    model_list = model_list.filter(title__icontains=search)

    for key, value in request.GET.items():
        if key == 'search':
            model_list = model_list.filter(title__icontains=search)
        elif key == 'country_code':
            if key == 'country_code':
                model_list = model_list.filter(country=value)
    ...
```

`.urlencode()` method converts special characters to %XX format:

Space	()	%20 or +
Exclamation Mark	(!)	%21
Double Quote	(")	%22
Hash	(#)	%23
Dollar	(\$)	%24
Percent	(%)	%25
Ampersand	(&)	%26
Plus	(+)	%2B
Equals	(=)	%3D
Question Mark	(?)	%3F

There is more...

`reverse` - is used to IMMEDIATELY get the url.

(from my_website/urls.py file) - often used in - function_based views

`reverse_lazy` - is used to get the url later when needed.

(from my_website/urls.py file) - often used in - class_based views

This is a better solution, however, we haven't implemented our filters just yet. We can include the query parameters into the URL with JavaScript. Or do that in the backend.

I will implement a solution in the backend out of preference and to demonstrate view's control flow of request in custom function, and show other cool perks :)

 project_folder/app_name/templates/app_name/index.html </> index.html

```
...
<form action="{% url 'index' %}" enctype="multipart/form-data" method="POST">
    {% csrf_token %}
    <div class="input-group py-3">
        <input type="search" name="search_query" class="form-control rounded-start" placeholder="Search">
        <button type="submit" class="btn btn-outline-primary">search</button>
    </div>

    <div id="filters">
        <h3 class="fs-4">Include filters in your search</h3>
        <ul class="list-inline list-unstyled py-2">

            <li class="list-inline-item">
                <label for="creator_empty">no creator:</label>
                <input type="radio" name="creator" id="creator_empty" checked>
            </li>
            <li class="list-inline-item">
                <label for="creator_username">creator username:</label>
                <input type="radio" name="creator" id="creator_username">
            </li>
            <li class="list-inline-item">
                <label for="creator_id">creator id:</label>
                <input type="radio" name="creator" id="creator_id">
            </li>

            <li class="mt-2">
                <input type="text" name="creator_username_value" id="creator_value" class="form-control mt-2" placeholder="Enter model creator's username" disabled hidden>
                <input type="text" name="creator_id_value" id="creator_value" class="form-control mt-2" placeholder="Enter model creator's ID" disabled hidden>
            </li>
            <li class="my-2">
                <label for="country_code">country code:</label>
                <input type="checkbox" name="country_code" id="country_code">
                <input type="text" name="country_code_value" id="country_code_value" class="form-control mt-2" placeholder="Enter model's country code" disabled>
            </li>

        </ul>
    </div>
</form>
...
```

```
...
{% block base %}
    <script name="search.js" src="{% static 'app_name/js/search.js' %}" defer></script>
{% endblock %}
```

 project_folder/app_name/static/app_name/js/search.js

 search.js

```
// location - localhost:port/?param1=a&param2=b...
// location.search - param1=a&param2=b...

// Get search parameters into Map Object with their values
const search = document.location.search.slice(1).split('&');

// searchMap - Map(param1 -> a, param2 -> b...)
const searchMap = new Map(search.map(s => {
    const [key, value] = s.split('=');
    return [key, decodeURIComponent(value)];
}));

// searchMap - Map(param1 -> a, param2 -> b...)
const filters = document.querySelector('#filters');
```

```
function showOnlySpecifiedCreatorInput(input_id, value) {
    /* Summary of this function:
     */
    / Hides all creator inputs,
    / Shows creator input specified by input_id, and
    / It optionally updates creator input's value if provided
    */

    let creatorInputs = filters.querySelectorAll('#creator_value');
    [...creatorInputs].map(input => {
        input.disabled = true;
        input.hidden = true;
    });

    if (input_id === undefined) return;

    let input = filters.querySelector(`#creator_value[name=${input_id}]`);
    input.disabled = false;
    input.hidden = false;

    if (value != undefined) input.value = value;
}
```

```

// hides all #creator_value inputs at #filters
showOnlySpecifiedCreatorInput();

// Populate filters if location.search has expected parameters
for (const [key, value] of searchMap) {
    switch (key)
    {
        case 'search':
            document.querySelector('input[name=search_query]').value =
searchMap.get('search');
            break;

        case 'creator':
            filters.querySelector('#creator_username').checked = true;
            // shows #creator_value[name=creator_username_value] and update its value
            showOnlySpecifiedCreatorInput('creator_username_value', value);
            break;

        case 'creator_id':
            filters.querySelector('#creator_id').checked = true;
            // shows #creator_value[name=creator_username_value] and update its value
            showOnlySpecifiedCreatorInput('creator_id_value', value);
            break;

        case 'country_code':
            filters.querySelector('#country_code').checked = true;
            filters.querySelector('#country_code_value').disabled = false;
            filters.querySelector('#country_code_value').value = value;
            break;
    }
}

```

```

// Make radio filters for #creator_value inputs work
const radios = filters.querySelectorAll('input[name=creator]');
[...radios].map(radio => {
    radio.addEventListener('change', () => {
        if (radio.checked && radio.id != 'creator_empty') {
            // shows #creator_value input of filter option: #radio.id
            showOnlySpecifiedCreatorInput(radio.id + '_value');
        } else {
            // hides all filter: #creator_value, inputs
            showOnlySpecifiedCreatorInput();
        }
    });
});

```

```
// Toggle country code input's active state
let checkbox = filters.querySelector('#country_code');
checkbox.addEventListener('change', () => {
    if (checkbox.checked) {
        filters.querySelector('#country_code_value').disabled = false;
    } else {
        filters.querySelector('#country_code_value').disabled = true;
    }
});
```

 project_folder/app_name/views.py

 views.py

```
...
def indexView(request):
    model_list = ModelName.objects.all()

    if request.method == 'POST':
        search = request.POST.get('search_query')
        if search:
            # Note: request.GET is immutable, hence we need to create a mutable copy.
            query_params = request.GET.copy() # query_params is a mutable copy
            query_params['search'] = search # add search parameter to request copy

            new_url = reverse('index') + "?" + query_params.urlencode()
            new_url = reverse('index') + "?" + request_search_filter_params(request)
            # return back to the previous render, with updated url
            return HttpResponseRedirect(new_url)
```

```
for key, value in request.GET.items():
    if key == 'search':
        model_list = model_list.filter(title__icontains=search)
    elif key == 'country_code':
        model_list = model_list.filter(country=value)
...

```

NOTE: `request_search_filter_params()` is a custom function
which is defined at the next page

```
...  
  
def is_model_creator(model_instance, user_instance):  
    return model_instance.creator == user_instance
```

```
def request_search_filter_params(request):
```

```
    """
```

```
    Converts posted form arguments, to permitted search url parameters.  
    Additional parameters are filtered from the request copy: GET_copy.
```

```
    Returns:
```

```
        GET_copy (QueryDict) with valid parameters
```

```
    """
```

```
# Note: requests are immutable, hence we need to create a mutable copy
```

```
GET_copy = request.GET.copy()      # GET_copy is a mutable copy
```

```
POST_copy = request.POST.copy() # POST_copy is a mutable copy
```

```
# Remove 'creator_id_value' if 'creator_username_value' exists
```

```
if request.POST.get('creator_id_value') and request.POST.get('creator_username_value'):  
    POST_copy.pop('creator_id_value')
```

```
# Add parameters to copy of request GET
```

```
for key, value in POST_copy.items():
```

```
    if key == 'search_query':
```

```
        GET_copy['search'] = value
```

```
    elif key == 'creator_id_value':
```

```
        try: # Remove invalid 'creator_id' parameter
```

```
            GET_copy['creator_id'] = int(value)
```

```
        except ValueError:
```

```
            messages.error(request, 'Creator\'s ID must be an Integer!')
```

```
    elif key == 'creator_username_value':
```

```
        GET_copy['creator'] = value
```

```
    elif key == 'country_code_value':
```

```
        GET_copy['country_code'] = value
```

```
    return GET_copy.urlencode()
```

```
...
```

```
Place request_search_filter_params function somewhere above indexView
```

Right, although this code might work initially, there are a couple of issues.

I decided to implement faulty code as an example of desception of a naming convention: `urlencode()`

```
GET_copy = request.GET.copy() → QueryDict (from django.http import QueryDict)
```

```
QueryDict('param1=Hello&param2=World') → <param1: ['Hello'], param2: ['World']>
```

We can't just pass this data in cases were: `QueryDict('param=Hello World')`

Because it includes unsupported characters upon return, like a space character.

Because unsupported string characters are present in our `GET_copy` `QueryDict`. `urlencode()` is thus required to avoid errors. However, its encoding changes URL:

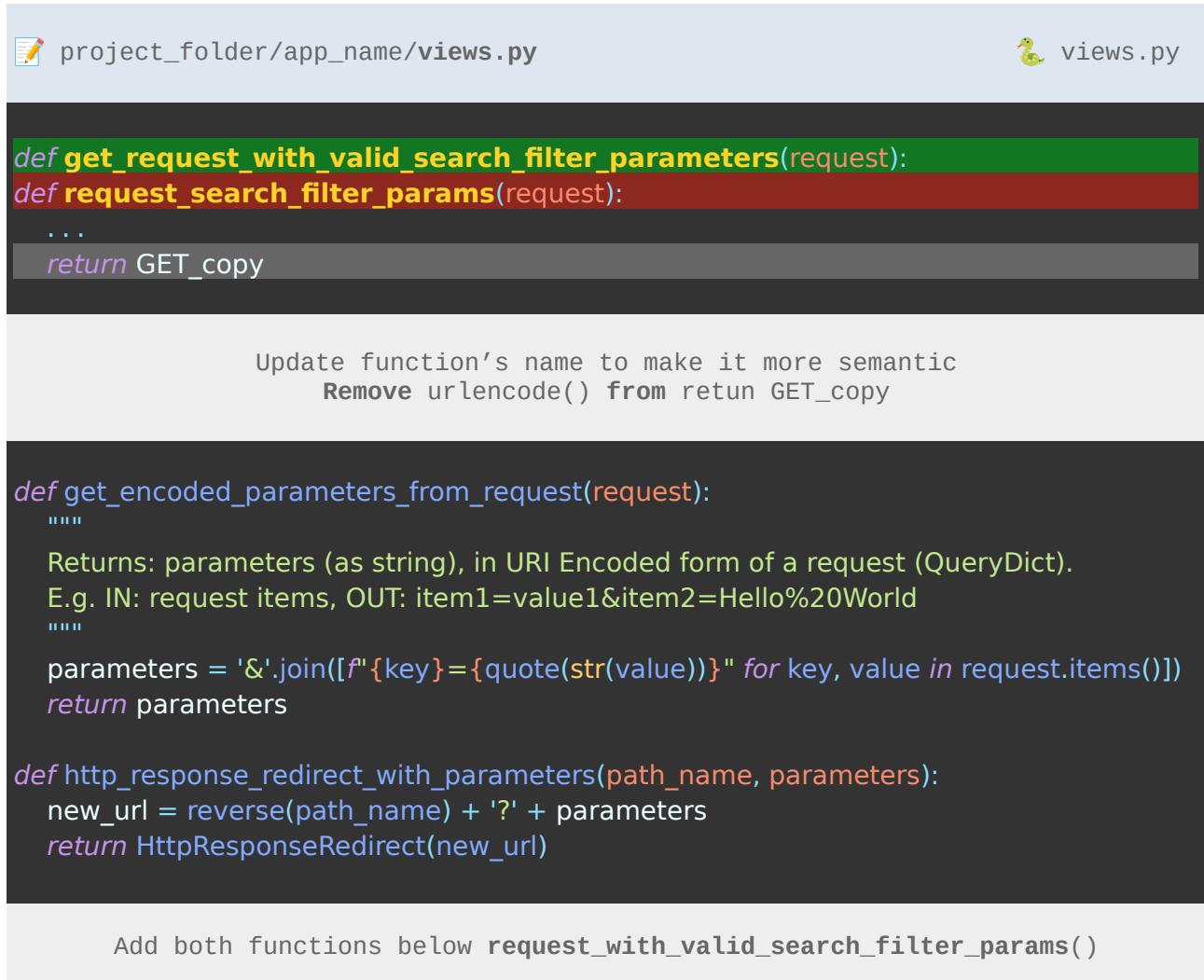
```
new_url = localhost:port/?param>Hello+World.
```

This is not a recognized URL encoding by JavaScript's `decodeURIComponent()`

Hence, the `+` in `?param`, will be ignored, introducing this “plus” character in our template form's `<input>` data.

It gets worse when clients POSTS back this received param data to the server.

Let's fix it



The screenshot shows a code editor with two tabs: `project_folder/app_name/views.py` and `views.py`. The `views.py` tab is active, displaying the following code:

```
def get_request_with_valid_search_filter_parameters(request):
    def request_search_filter_params(request):
        ...
        return GET_copy
```

Below the code, there is a note: "Update function's name to make it more semantic" and "Remove `urlencode()` from return `GET_copy`".

Further down, the code continues:

```
def get_encoded_parameters_from_request(request):
    """
    Returns: parameters (as string), in URI Encoded form of a request (QueryDict).
    E.g. IN: request.items(), OUT: item1=value1&item2>Hello%20World
    """
    parameters = '&'.join([f'{key}={quote(str(value))}' for key, value in request.items()])
    return parameters
```

```
def http_response_redirect_with_parameters(path_name, parameters):
    new_url = reverse(path_name) + '?' + parameters
    return HttpResponseRedirect(new_url)
```

At the bottom of the code block, there is a note: "Add both functions below `request_with_valid_search_filter_params()`".

```

from urllib.parse import quote

Import quote (there is also unquote in cause you'd need it)

...
def indexView(request):

    model_list = ModelName.objects.all()

    # Handle POST request (search form submission)
    if request.method == 'POST':
        R = get_request_with_valid_search_filter_parameters(request)
        params = get_encoded_parameters_from_request(R)
        # Redirect to indexView with search filter parameters in URI
        return http_response_redirect_with_parameters('index', params)
    if request.method == 'POST':
        search = request.POST.get('search_query')
        if search:
            new_url = reverse('index') + "?" + request_search_filter_params(request)
            # return back to the previous render, with updated url
            return HttpResponseRedirect(new_url)

```

Update indexView's - if request.method == 'POST':

WARNING!

Test this search and especially the pagination.

This code might be faulty because I have documented the solution too soon, after couple of improvements.

IF the code is faulty, debug it and improve it. OR,
head to the Section - **Django HTMX - Infinite Scroll**

Contact Us View
(Support)



```
...
```

```
class ContactUsForm(forms.Form):  
  
    TOPIC_CHOICES = [  
        ('', ' --- Select a topic --- '),  
        ('auth_failure', 'I can\'t access my account'),  
        ('auth_failure', 'I can\'t log in with my email address'),  
        ('auth_failure', 'I can\'t log in with my username'),  
        ('auth_failure', 'I can\'t log in with my TFA code'),  
        ('impersonation', 'Someone is impersonating me'),  
        ('report_user', 'User violates terms and conditions'),  
        ('bug_report', 'A feature on a website is broken'),  
        ('other', 'Other')  
    ]  
  
    email = forms.EmailField(label='Email', max_length=255)  
  
    subject = forms.ChoiceField(  
        label='Subject',  
        choices=TOPIC_CHOICES,  
        required=True  
    )  
  
    message = forms.CharField(  
        label='Message',  
        widget=forms.Textarea,  
        initial=''+  
            'Hello, my name is: Forename Surname\n\n'+  
            'I would like to report the following issue:',  
        max_length=10000,  
        min_length=250,  
        required=True  
    )
```

NOTE the use of: `forms.Form` , instead of , `forms.ModelForm`

`ModelForm` uses sub-class `Meta` to link `Model` and it's Fields

There is no need for a model. Hence, the use of a normal `Form`

```
...
from django.core.mail import send_mail, EmailMultiAlternatives
from django.conf import settings
from django.utils.html import escape
...
...
def email_client_and_support_team(email, subject, message):
    # Escape HTML tags to treat them as plain text
    escaped_message = escape(message)

    # Replace newline characters with <br> for the HTML content
    html_message = escaped_message.replace('\n', '<br>')

    recipient_message = (
        '<p style="color: rgb(95,95,95);font-family: "Indeed Sans","Noto Sans", Helvetica , Arial , sans-serif;font-size: 14.0px;font-weight: normal;line-height: 24.0px;margin: 0;padding: 0;direction: ltr;">'
        '<strong>You have sent the following message to our support team:</strong><br><br>'
        f'{html_message}<br><br>'
        '<strong>Our support team will get back to you as soon as possible.</strong><br>'
        '<strong>Feel free to ignore this email if it wasn\'t you.</strong>'
        '</p>'
    )

    # Send HTML email to recipient
    email_to_recipient = EmailMultiAlternatives(
        subject=f"Support Request: {subject}",
        body=recipient_message,
        from_email=settings.EMAIL_HOST_USER,
        to=[email]
    )

    # Send email to recipient
    email_to_recipient.attach_alternative(recipient_message, "text/html")
    email_to_recipient.send()

    # Send email to support team
    send_mail(
        subject=f'Support Request: {subject}',
        message=f'{message}\n\nRequest from (Email): {email}',
        from_email=settings.EMAIL_HOST_USER,
        recipient_list=['support@walentynki.site']
    )
```

Let's create `contactFormView` below `email_client_and_support_team`

NOTE: `email_client_and_support_team` is our custom function at  `views.py`

 project_folder/app_name/`views.py`

 `views.py`

```
...
from .forms import ModelForm, ContactUsForm
...

...
def contactFormView(request):
    if request.method == 'POST':
        form = ContactUsForm(request.POST)

        if form.is_valid():
            email = form.cleaned_data['email']
            subject = form.cleaned_data['subject']
            message = form.cleaned_data['message']

            email_client_and_support_team(email, subject, message)

            messages.success(request, 'Message sent successfully!')
            messages.info(request, f'We will contact you shortly at: {email}')

            # Reset some fields to their default values
            initial_data = form.cleaned_data.copy()
            initial_data['subject'] = ContactUsForm.TOPIC_CHOICES[0][0]
            initial_data['message'] = ContactUsForm.base_fields['message'].initial

            form = ContactUsForm(initial=initial_data)
        else:
            messages.error(request, 'Failed to send message!')

    else:
        form = ContactUsForm()

    return render(request, 'app_name/contact.html', {'form': form})
```

 project_folder/app_name/urls.py

 urls.py

```
urlpatterns = [  
    ...  
    path('contact/', views.contactFormView, name='contact'),  
]
```

 project_folder/app_name/templates/app_name/contact.html </> contact.html

```
{% extends 'app_name/base.html' %}  
  
{% load static %}  
{% load crispy_forms_tags %}  
{% block title %}Contact Page{% endblock %}  
  
-----  
  
{% block content %}  
<form action="{% url 'contact' %}" method="POST">  
    {% csrf_token %}  
    B {{ form|crispy }}  
    <button type="submit" class="btn btn-warning">  
        <span>Send Message</span>  
    </button>  
</form>  
{% endblock %}
```

Secure Forms (reCAPTCHA)

Frequent requests from the server's bot (noreply@domain.com) may lead the external host provider, such as **Zoho**, to stop processing these requests.

Consequently, temporarily disable the EMAIL_HOST provider, classify the bot's messages as spam, and ensure that these messages do not land in recipients' inboxes.

The use of **reCAPTCHA** deters spammers from abusing our **Forms**, significantly minimizing such incidents.

Let's employ it!



I'm not a robot 
reCAPTCHA
Privacy - Terms

Create Google Account, and hit **Get started** at
<http://google.com/recaptcha/about>

Privacy is built-in to reCAPTCHA. Read the blog to discover how we protect your data.

reCAPTCHA bot protection and online fraud prevention

Protect against fraud and abuse with modern bot protection and fraud prevention platform

Uplevel your online fraud protection capabilities with a frictionless solution that protects your website and mobile apps against the most sophisticated targeted and scaled attacks.

[Get started](#)

[Manage reCAPTCHA](#)

Manage existing reCAPTCHAS at
<https://www.google.com/recaptcha/admin>

I'm going to use
reCAPTCHA v2 checkBox

Copy Public
& Private keys

Etykleta 
django-test
11/50

Typ reCAPTCHA 
 Na podstawie wyniku (v3) Weryfikuj żądania na podstawie wyniku
 Zadanie (v2) Weryfikuj żądania w oparciu o zadanie
 Pole wyboru „Nie jestem robotem” Weryfikuj żądania, używając pola wyboru „Nie jestem robotem”
 Niewidoczna plakietka reCAPTCHA Weryfikuj żądania w tle

Domeny 

+ 127.0.0.1 + localhost

ADD: 127.0.0.1 AND localhost TO Domains

Użyj tego klucza witryny w kodzie HTML wyświetlonym użytkownikom

 KOPIUJ KLUCZ
WITRYNY

dfKHSDKJEduhjdJDhfberISDUWJ

Użyj tego tajnego klucza do komunikacji między Twoją witryną a serwerem

 KOPIUJ TAJNY
KLUCZ

hgfhkjkrtdSJFHerJHDsuerlUSDhej

(venv) @ project_folder

 pip library

```
pip install  django-recaptcha
pip freeze > requirements.txt # update requirements.txt
```

project_folder/.env

.env

```
# Google reCAPTCHA
RECAPTCHA_PUBLIC_KEY='public key here'
RECAPTCHA_PRIVATE_KEY='private key here'
```

project_folder/my_website/settings.py

 settings.py

```
INSTALLED_APPS = [
    ...
    # Add libraries above apps
    'django_recaptcha',
]

...
RECAPTCHA_PUBLIC_KEY = os.getenv('RECAPTCHA_PUBLIC_KEY')
RECAPTCHA_PRIVATE_KEY = os.getenv('RECAPTCHA_PRIVATE_KEY')
```

```
from django import forms
from django_recaptcha.fields import ReCaptchaField
```

```
class MyForm(forms.Form):
    captcha = ReCaptchaField()
```

Just include and initialize **captcha** field to protect your **Form**

As of Feb/2025, there are 3 types of reCAPTCHA:

- **ReCaptchaV2Checkbox**
- **ReCaptchaV2Invisible**
- **ReCaptchaV3**

```
from django_recaptcha.widgets import ReCaptchaV2Checkbox
ReCaptchaField(widget=ReCaptchaV2Checkbox) # ← is set by default
```

<p>Google reCAPTCHA</p> <p>Etykietka </p> <p>baketoom.com</p> <p>Typ reCAPTCHA: Pole wyboru „wersja 2”</p> <p>Klucze reCAPTCHA </p> <p>Domeny </p> <ul style="list-style-type: none"> X baketoom.com X www.baketoom.com X baketoom.herokuapp.com X web-production-1cb3.up.railway.app + Dodaj domenę, np. example.com <p>Właściciele</p> <ul style="list-style-type: none"> X administration@baketoom.com X mcrazick@mail.com + Wpisz adresy e-mail 	<p>To ensure that your Django website fully supports reCAPTCHA in a production environment, it is highly recommended to register any domains that your website operates on.</p> <p>Google reCAPTCHA</p> <p>Pole wyboru „wersja 2” baketoom.com </p> 
--	---

learn more: [django_recaptcha](#), [google docs recaptcha](#), [advanced recaptcha docs](#)

Amazon Web Services (s3 buckets)



Managing user image storage on the server side becomes impractical as platform scales.

As of now, images accumulate in the hosting-service storage under the following directory:
📁 `/media/profile_pics`, restricting the website's growth potential, straining server performance while serving clients, and significantly increasing costs. This is because not all hosting platforms are designed to handle frequent resource updates.

Let's utilize a service specializing in resource management, like AWS S3 buckets, to accommodate our platform's media assets, such as, user's images.

Create Account At - <https://aws.amazon.com/s3/>

The image shows two side-by-side screenshots of the AWS sign-in process. On the left, the 'IAM user sign in' page is displayed, featuring fields for Account ID, IAM username, and Password, along with options for Show Password and Remember this account. On the right, the 'Create an AWS Account' page is shown, prompting the user to choose between being a Root user or an IAM user, providing a Root user email address, and agreeing to the AWS Customer Agreement. Both pages include links for 'Create a new AWS account' and 'Sign in using root user email'.

Sign In Create an AWS Account

For future, login using **Root** user email

IAM user sign in ⓘ

Account ID (12 digits) or account alias

IAM username

Password

Show Password Having trouble?

Sign in

Sign in using root user email

Create a new AWS account

Remember this account

Root user

Account owner that performs tasks requiring unrestricted access. [Learn more](#)

IAM user

User within an account that performs daily tasks. [Learn more](#)

Root user email address

username@example.com

Next

By continuing, you agree to the [AWS Customer Agreement](#) or other agreement for AWS services, and the [Privacy Notice](#). This site uses essential cookies. See our [Cookie Notice](#) for more information.

New to AWS?

Create a new AWS account

© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved

Create General purpose S3 Bucket at Europe Region: eu-west-2

Official Tutorial

The screenshot shows the AWS S3 console interface. At the top, there are buttons for 'Copy ARN', 'Empty', 'Delete', and a prominent orange 'Create bucket' button. Below this, a section titled 'General purpose buckets (1)' shows a single bucket entry. The bucket name is 'myWebsite-appName-files', and it is located in the 'Europe (London) eu-west-2' region. A search bar labeled 'Find buckets by name' is also visible.

(Note: Bucket name must not contain uppercase characters, this is just example)

The screenshot shows the contents of the 'myWebsite-appName-files' bucket. It lists four objects: a folder named '/profile_pics/' and a file named 'default_profile.jpg'. The file is a JPEG image. The bucket is accessible via the URL eu-west-2.console.aws.amazon.com/s3. A note on the left suggests creating a folder named '/profile_pics' and including a default profile picture for newly registered users. A 'NOTE:' section provides instructions for adding files to the bucket.

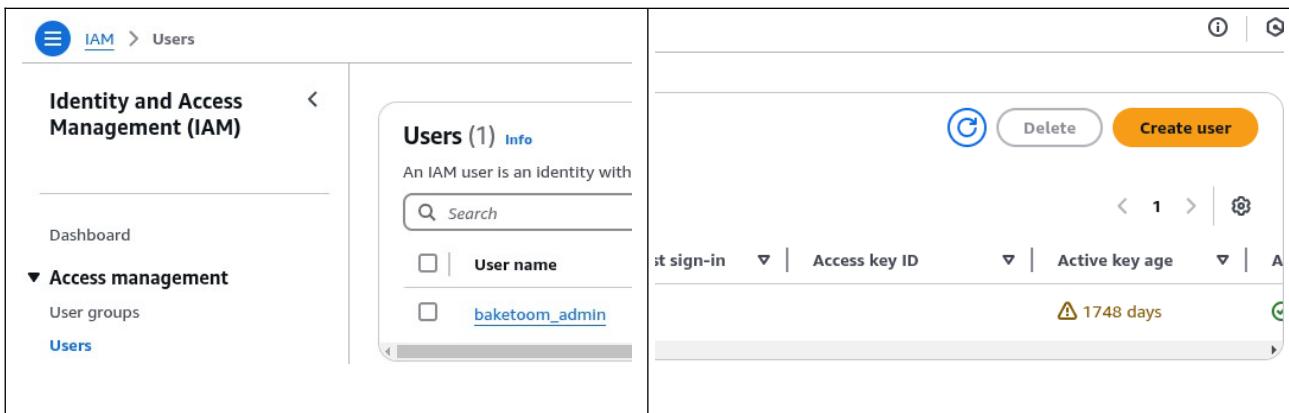
In future, once you login as Root user. You can access S3 bucket's by hitting the green S3 bucket icon, under section: **Recently visited**

Console Home Info

The screenshot shows the AWS Console Home interface. Under the 'Recently visited' section, there is a link to the S3 service. Under the 'Applications' section, it shows the region as 'Europe (London)' and the current region as 'eu-west-2 (Current Region)'.

Head to AWS IAM (Identity and Access Management):
<https://us-east-1.console.aws.amazon.com/iamv2>

Under **Access management**, head to **Users**, and hit **Create user**

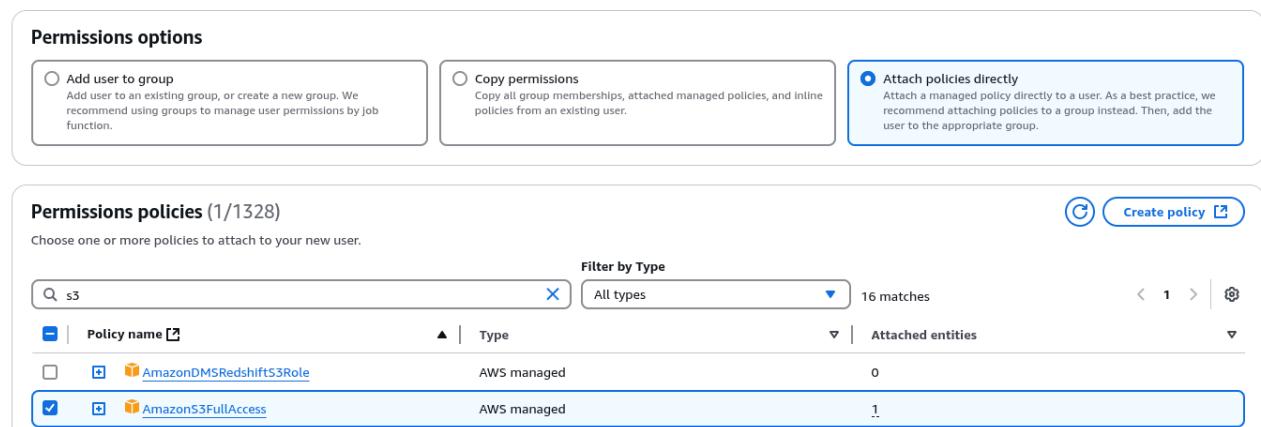


The screenshot shows the AWS IAM 'Users' page. On the left, there's a sidebar with 'Identity and Access Management (IAM)' selected. The main area shows 'Users (1)'. A table lists one user: 'User name' (baketoom_admin). To the right, there are buttons for 'Delete' and 'Create user'. Below the table, there are filters for 'Last sign-in', 'Access key ID', and 'Active key age', with a note indicating an active key has been active for 1748 days.

Attach policies: **AmazonS3FullAccess**, to the new user

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)



This screenshot shows the 'Permissions options' section where 'Attach policies directly' is selected. Below it, the 'Permissions policies (1/1328)' list shows a search bar for 's3', a filter for 'Policy name', and a table listing policies. The policy 'AmazonS3FullAccess' is selected, indicated by a checked checkbox.

Access the user you've just created, and hit: **Create access key**



The screenshot shows the 'Access key 1' creation page. It features a table with two rows. The first row contains 'User name' (baketoom_admin) and 'Access key ID'. The second row contains 'Secret access key'. To the right, there's a note about using the access key to access AWS resources, and two radio button options: 'Application running outside AWS' (selected) and 'Other'.

Connect to AWS S3 via django-storages

```
project_folder/.env .env  
...  
# Amazon Web Services : s3 bucket for serving media files  
  
AWS_ACCESS_KEY_ID='AKXURHQ7EYDHMSJDALTE'  
AWS_SECRET_ACCESS_KEY='YdfudsfhSJShfje4Jsd328ASDhsjWieuSdr58ap'  
AWS_STORAGE_BUCKET_NAME='mywebsite-appname-files'
```

Retrieve access keys Info

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new one.

Access key

Secret access key

AKXURHQ7EYDHMSJDALTE

***** Show

```
(venv) @ project_folder pip library  
  
pip install dj django-storages boto3  
pip freeze > requirements.txt # update requirements.txt
```

Follow this librarie's documentation: [Amazon S3 \(django-storages\)](#)

```
project_folder/my_website/settings.py settings.py  
  
INSTALLED_APPS = [  
    ...  
    # Add libraries above apps  
    'storages',  
]  
  
# AWS S3 Bucket CONFIG  
AWS_ACCESS_KEY_ID = os.getenv('AWS_ACCESS_KEY_ID')  
AWS_SECRET_ACCESS_KEY = os.getenv('AWS_SECRET_ACCESS_KEY')  
AWS_STORAGE_BUCKET_NAME = os.getenv('AWS_STORAGE_BUCKET_NAME')  
  
AWS_S3_REGION_NAME = 'eu-west-2'  
AWS_S3_FILE_OVERWRITE = False  
AWS_DEFAULT_ACL = None  
DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'  
  
(...more below on next page)
```

```
# Include STORAGES dictionary only if you're in Django version >= 4.2

STORAGES = {
    "default": {
        "BACKEND": DEFAULT_FILE_STORAGE,
        "OPTIONS": {
            "access_key": AWS_ACCESS_KEY_ID,
            "secret_key": AWS_SECRET_ACCESS_KEY,
            "bucket_name": AWS_STORAGE_BUCKET_NAME,
        },
    },
    "staticfiles": {
        "BACKEND": "django.contrib.staticfiles.storage.StaticFilesStorage",
        "OPTIONS": {
            "location": STATIC_ROOT,
        },
    },
}
```

(venv) @ project_folder

(InteractiveConsole)

```
>>> import boto3
>>> s3 = boto3.resource('s3')
>>> for bucket in s3.buckets.all():
...     print(bucket.name)
...
baketoom-recipes-files
mywebsite-appname-files
```

manage.py (shell)

```
python manage.py shell
>>> import boto3
>>> s3 = boto3.resource('s3')
>>> for bucket in s3.buckets.all():
...     print(bucket.name)
```

← Should display buckets from AWS
IF it does, it means it works

Now, create a user and update its profile picture. The newly added image should land in S3 bucket, instead of local /media directory.

[profile_pics/](#)

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of objects.

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	mcr.jpeg	jpeg	February 18, 2025, 22:40:12 (UTC+00:00)	7.3 KB	Standard
<input type="checkbox"/>	Screenshot_fro_m_2024-05-16_02-50-10.jpeg	jpeg	February 18, 2025, 22:28:32 (UTC+00:00)	18.7 KB	Standard

Profile + 📁 /recycle_pics

AWS S3 Buckets 

Users can upload an image to set their profile picture. The issue is that every time a user uploads an image, it gets automatically added with a unique hashed name if there are duplicates.

Consequently, the 📁 [/profile_pics](#) directory in our S3 bucket will quickly fill up with images. This design flaw makes it difficult to identify and remove unused images, which can lead to limited storage space and increased expenses for maintaining the bucket.

We could re-design the model to replace the user's profile picture in the S3 bucket with the newly uploaded picture, which would be the most efficient solution. However, I propose a different approach.

We will create a new folder in our S3 bucket called 📁 [/recycle_pics](#). This folder will store users' unused profile pictures once they update their profile picture in the 📁 [/profile_pics](#) folder of the S3 bucket.

This setup allows us to monitor the images used. In the future, we could even attach user IDs to these images and make the folder private. This would help us ensure compliance with our terms and services, particularly regarding the uploading of NSFW images. These files could also be used as evidence for law enforcement when requested in reasonable timeframe. Additionally, this approach will be useful if a user wishes to retrieve their past data.

The 📁 [/recycle_pics](#) folder can be maintained with different policies and configured to clear the pictures periodically and automatically directly via AWS.

Head to the next page

to learn how to implement these solutions, and

feel free to integrate one that suits your organization's needs

 project_folder/users/models.py

(profile users)  models.py

```
from django.conf import settings

from botocore.exceptions import ClientError
import boto3

s3 = boto3.client(
    's3',
    aws_access_key_id=settings.AWS_ACCESS_KEY_ID,
    aws_secret_access_key=settings.AWS_SECRET_ACCESS_KEY,
    region_name=settings.AWS_S3_REGION_NAME
)

def is_profile_pic(image_key):
    try:
        s3.head_object(
            Bucket=settings.AWS_STORAGE_BUCKET_NAME,
            Key=f"profile_pics/{image_key.split('/')[-1]}"
        )
        return True
    except ClientError as e:
        if e.response['Error']['Code'] == '404':
            return False
        else:
            raise e
```

Include the above imports and function: `is_profile_pic()`, as they are required for:

The functions: `remove_profile_pic` and `recycle_profile_pic`

```

def remove_profile_pic(image_key):

    if is_profile_pic(image_key):

        # Delete the old image from the original location
        s3.delete_object(
            Key=f"profile_pics/{image_key.split('/')[-1]}",
            Bucket=settings.AWS_STORAGE_BUCKET_NAME
        )



---


def recycle_profile_pic(image_key):

    if is_profile_pic(image_key):

        profile_pic_key = f"profile_pics/{image_key.split('/')[-1]}"
        recycle_pic_key = f"recycle_pics/{image_key.split('/')[-1]}"

        # Copy the image to the recycle_pics folder
        s3.copy_object(
            Key=recycle_pic_key,
            Bucket=settings.AWS_STORAGE_BUCKET_NAME,
            CopySource={
                'Key': profile_pic_key,
                'Bucket': settings.AWS_STORAGE_BUCKET_NAME
            }
        )
        # Delete the image from the original location
        s3.delete_object(
            Key=profile_pic_key,
            Bucket=settings.AWS_STORAGE_BUCKET_NAME
        )

    else:
        message = f'Image {image_key} not found in bucket: '
        message += f'{settings.AWS_STORAGE_BUCKET_NAME}'
        message += ', skipping copy and delete.'
        print(message)

```

NOTE: Implement one of the above functions

remove_profile_pic OR recycle_profile_pic

because you don't need both

```

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    phone_number = models.CharField(max_length=15)
    image = ResizedImageField(
        size=[300, 300],
        crop=['middle', 'center'],
        quality=75,
        force_format='JPEG',
        upload_to='profile_pics',
        default='default_profile.jpg'
    )

    def save(self, *args, **kwargs):
        try:
            this = Profile.objects.get(user=self.user)
            # Only move the old image if there is an existing image
            # and it's different from the new image
            if this.image and this.image.name != self.image.name:
                recycle_profile_pic(this.image.name)
                remove_profile_pic(this.image.name)
        except Profile.DoesNotExist:
            pass # This is a new profile, so no need to move any image

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

```

Include `recycle_profile_pic()` OR `remove_profile_pic()` only

mywebsite-appname-files Info

Objects | **Properties** | **Permissions**

Objects (3)

(C)

Objects are the fundamental entities stored in Amazon S3 in your bucket. For others to access your objects, you must grant them permission.

<input type="checkbox"/>	Name	Type
<input type="checkbox"/>	default_profile.jpg	jpg
<input type="checkbox"/>	profile_pics/	Folder
<input type="checkbox"/>	recycle_pics/	Folder

IF you decided to use `recycle_profile_pic()`

THEN

You have to add new folder at AWS S3 Bucket:

 /recycle_pics

Filter NSFW (not safe for work) Profile images



💻 (venv) @ project_folder

📚 pip library

```
pip install requests  
pip freeze > requirements.txt # update requirements.txt
```

📝 project_folder/my_website/settings.py

🐍 settings.py

Requests is just a pip library, it is not specifically a Django package
Hence, **NO NEED** to install it in the **INSTALLED_APPS**

```
# DeepAI for NSFW image detection
```

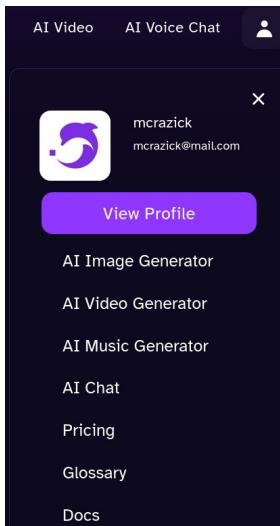
```
DEEPAI_API_KEY = os.getenv('DEEPAI_API_KEY')  
DEEPAI_NSFW_DETECTOR_URL = 'https://api.deepai.org/api/nsfw-detector'  
DETECT_NSFW_IMAGES = os.getenv('DETECT_NSFW_IMAGES', 'False').lower() in ['true', '1']
```

📝 project_folder/.env

.env

```
# Deepai for NSFW
```

```
DETECT_NSFW_IMAGES='True'  
DEEPAI_API_KEY='DEEPAI_API_KEY'
```



Head to the deepai.org, create an account → View Profile

I will continue with a free plan account

Scroll Down to API Keys, and copy your API into

DEEPAI_API_KEY at .env file

NOTE: Free Plan has limited number of API requests per month
Reconsider upgrading plan in future, like to: Pay as you Go

```

📝 project_folder/users/utils.py          (profile users) 🐍 utils.py

import requests
from django.core.files.uploadedfile import UploadedFile
from django.conf import settings

def is_image_nsfw(image_file: UploadedFile):

    # Push image to DeepAI API, to get NSFW score ( deepai.org/docs#Python )
    response = requests.post(
        url=settings.DEEPAI_NSFW_DETECTOR_URL,
        files={'image': image_file.open('rb')},
        headers={'api-key': settings.DEEPAI_API_KEY}
    )

    result = response.json()

    # Check for status and handle errors
    if result.get('status') != 'success':
        message = "API request failed with status:"
        message += f" {result.get('status')} -"
        message += f" {result.get('error', 'No error message provided')}"
        raise ValueError(message)

    nsfw_score = result.get('output', {}).get('nsfw_score')
    if nsfw_score is None:
        raise ValueError("Unexpected response format: 'output' or 'nsfw_score' key missing.")

    # Adjust the threshold as needed (0.5 = 50%)
    return nsfw_score > 0.5

```

```

📝 project_folder/users/signals.py          (profile users) 🐍 signals.py

from django.db.models.signals import pre_save
from django.dispatch import receiver
from .models import Profile
from .utils import is_image_nsfw

@receiver(pre_save, sender=Profile)
def check_nsfw_image(sender, instance, **kwargs):

    if instance.image:
        if is_image_nsfw(instance.image):
            raise ValueError('NSFW images are not allowed.')

```

```
from django.apps import AppConfig
from django.conf import settings

class UsersConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'users'

    def ready(self):
        if settings.DETECT_NSFW_IMAGES:
            from users.signals import check_nsfw_images
```

Register your signals at apps.py of your app

Signals

In Django, signals are a way to allow certain events to notify other parts of your application when certain actions have occurred. They help in decoupling your application by allowing different components to communicate with each other without direct dependencies. Common signals include actions like model instance saving, deleting, or creating, where you can attach custom behavior to these events.

Utils (Utilities)

Django provides various utilities to assist with common tasks, making development more efficient. These utilities are helper functions or modules that simplify operations such as text processing, timezone handling, cryptographic operations, and more. They enhance readability and maintainability of your code by abstracting complex or repetitive tasks into reusable functions.

NOTE - We've created our own, custom:

signal: `check_for_nsfw_image()`, and **util**: `is_image_nsfw()` function

And custom setting options, to manage our NSFW functionality:

`DEEPAI_API_KEY`
`DEEPAI_NSFW_DETECTOR_URL`
`DETECT_NSFW_IMAGES`

Class-based views:

list, form, detail,
create, update and delete

Class-based views (CBVs) in Django provide a more structured and object-oriented approach to handling HTTP requests compared to function-based views. They offer a way to encapsulate and reuse view logic by defining views as classes. Here's an overview of CBVs:

Types of Class-Based Views

- **Base Views:** The simplest form, used as a foundation for more complex views.
Example: **View**, **TemplateView**. (@ [django/views/generic/base.py](#))
- **Generic Views:** Pre-built views that provide common functionality. Example:
 1. **ListView:** Displays a list of objects.
 2. **DetailView:** Displays a single object.
 3. **CreateView:** Handles the creation of a new object.
 4. **UpdateView:** Handles updating an existing object.
 5. **DeleteView:** Handles the deletion of an object.

Advantages of CBVs

- **Code Reuse:** By using inheritance, common functionality can be shared across multiple views.
- **Organization:** Encapsulating logic within a class provides a clean and organized structure.
- **Extensibility:** Easily extend and customize functionality by overriding class methods.

Key Components of CBVs

- **Mixin Classes:** Reusable classes that provide specific functionality and can be combined to create complex behavior.
- **HTTP Methods:** Class-based views have methods for handling different HTTP requests (GET, POST, etc.).
- **Attributes and Methods:** Use attributes (like **template_name**, **model**) and methods (like **get_context_data**, **form_valid**) to customize view behavior.

Example - Import class-based-views

class-based views are under **django.contrib.auth.views** module:

 project_folder/app_name/views.py

 views.py

```
from django.views.generic import (
    ListView,
    UpdateView,
    DetailView,
    ...
)
```

 project_folder/app_name/urls.py

 urls.py

```
path('pathname/', MyClassBasedView.as_view(), name='path-name')
```

NOTE:

MyClassBasedView is a **class**. Hence, use **as_view()** to convert it to a supported format.

We've used some build-in views before, with some mixins, when creating user
Register and Login new Users

I decided to introduce class-based views at this stage so that we can understand the usecase of functional-views and to experience the struggles of its inclusion, all in order to appreciate the class-based views.

They can become complicated when we decide to overwrite their behaviour without understanding their inherited classes. Hence, I advice to study Django's codebase of class-based views at: <https://github.com/django/django/tree/main/django/views/generic>

Example - UpdateView

This means, we don't have to constantly write, for example:

```
def updateUserView(request, user_id):
    user_instance = get_object_or_404(User, pk=user_id)

    if request.method == 'POST':
        form = MyForm(request.POST, instance=user_instance)
        if form.is_valid():
            form.save()
            message.success(request, 'Updated profile')
            redirect('index')
        else:
            message.error(request, 'Failed to update profile')
    else:
        form = MyForm(instance=user_instance)

    context = {'form': form}
    return render(request, 'template.html', context)
```

Instead, we can do:

```
from django.urls import reverse_lazy

class UpdateUserView(UpdateView):
    model = User
    form_class = MyForm
    template_name = 'template.html'
    success_url = reverse_lazy('index')
    # error_url = reverse_lazy('index')
    context_object_name = 'context'

    def get_object(self, queryset=None):
        return get_object_or_404(User, pk=self.kwargs['user_id'])

    def form_valid(self, form):
        messages.success(self.request, 'Updated profile')
        return super().form_valid(form)

    def form_invalid(self, form):
        messages.error(self.request, 'Failed to update profile')
        return super().form_invalid(form)
```

Notice how a class-based view is comprehensive and semantic. (Eliminates boiler-plate)

(**NOTE:** `class` in django is in **PascalCase**)

(**NOTE:** `kwargs`, stands for - keyword arguments)

{'form': form} is created and passed automatically.

Use: `get_context_data`, to update context data:

```
...  
  
def get_context_data(self, **kwargs):  
    context = super().get_context_data(**kwargs)  
  
    context['custom_message'] = 'custom data for template.html'  
    context['extra_info'] = 'another custom data for template.html'  
  
    # IF URI = localhost:port/?param1=Hello&param2=World  
    # Extract query parameters from the URL  
    param1 = self.request.GET.get('param1') # ← Hello  
    param2 = self.request.GET.get('param2') # ← World  
  
    return context
```

`super()` : A Python feature used to call inherited methods from its parent class

Example - ListView

```
from django.views.generic import ListView
from django.urls import reverse_lazy
from .models import ModelName
from .forms import FormName


class ModelNameListView(ListView):
    model = ModelName
    template_name = 'template.html'
    context_object_name = 'context'
    # paginate_by = 10

    def get_queryset(self):
        queryset = super().get_queryset()
        form = FormName(self.request.GET or None)

        if form.is_valid():
            # Get <input name="my_input"> values from template's form
            search_query = form.cleaned_data.get('search_query')
            country_code = form.cleaned_data.get('country_code')
            creator_id = form.cleaned_data.get('creator_id')
            creator = form.cleaned_data.get('creator')

            if search_query:
                queryset = queryset.filter(title__icontains=search_query)
            if country_code:
                queryset = queryset.filter(country=country_code)
            if creator_id:
                queryset = queryset.filter(creator=creator_id)
            if creator:
                queryset = queryset.filter(creator__username__icontains=creator)

        return queryset


    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['form'] = FormName(self.request.GET or None)
        return context
```



💻 (venv) @ project_folder



```
pip install django-htmx  
pip freeze > requirements.txt # update requirements.txt
```

</> htmx

📝 project_folder/my_website/settings.py

🐍 settings.py

```
INSTALLED_APPS = [  
    # Add libraries above apps  
    'django_htmx',  
]
```

```
MIDDLEWARE = [  
    'django.middleware.common.CommonMiddleware',  
    'django_htmx.middleware.HTMXMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
]
```

💻 (venv) @ project_folder/app_name/static/app_name/js (app_name static/js)

```
wget https://unpkg.com/htmx.org@latest/dist/htmx.min.js
```

This Linux command will **download** the HTMX to current terminal location

OR download manually from <https://unpkg.com/browse/htmx.org@latest/dist/>
argument @latest is not always the most recent available version.

📝 project_folder/app_name/templates/app_name/base.html

</> base.html

```
{% load django_bootstrap5 %}  
{% load static %}  
<!DOCTYPE html>  
  <html lang="en">  
    <head>  
      <meta charset="UTF-8">  
      <meta name="viewport" content="width=device-width, initial-scale=1.0">  
      <title>{% block title %}{% endblock %}</title>  
      {% bootstrap_css %}  
      <script src="{% static 'app_name/js/lib/htmx.min.js' %}" defer></script>  
      {% block head %}{% endblock %}  
    </head>  
  <body>
```

Official HTMX Documentation: <https://htmx.org/docs/>

Please read the documentation of HTMX, I will show basic options only:

<code>hx-post & hx-get</code>	form's attribute: <code>action=""</code> with <code>method=""</code>
<code>hx-trigger</code>	<code>submit, click, keyup, keydown, load, change, focus, blur, resize, revealed, mouseenter, mouseleave,</code> <code>delay</code> (optional) → click delay:1s (or delay:500ms) <code>throttle</code> (optional) → click throttle:1s (waits after request)
<code>hx-target</code>	<code>document.querySelector</code> → <code>#id, .class [attribute="value"] ...</code> Server response is send directly to the target
<code>hx-swap</code>	<code>beforebegin, beforeend, afterbegin, afterend, innerHTML, outerHTML, none</code> Server response replaces the target's content
<code>hx-swap-oob</code> (out of band)	Server side only feature. Has same keywords as <code>hx-swap</code> . (The use case of <code>hx-swap-oob</code> is demonstrated in the <code>chat</code> app) The <code>hx-swap-oob</code> attribute replaces the content of any target with the same ID as the HTML element. The target element must have an ID for the <code>hx-swap-oob</code> operation to work.

Example

```
<div id="target"></div>

<input type="button" ...
      hx-post="/url-path"
      hx-trigger="click delay:1s"
      hx-target="#target"
      hx-swap="innerHTML">
```

Avoid adding multiple `<elements>` with `hx` attribute inside the `<form>` because the responses may repeat unnecessarily! Also, use `delay` as it minimizes spam responses (throttling), improving server's performance. Do this instead:

```
<form
  hx-post="/url-path"
  hx-trigger="keyup delay:500ms"
  hx-target="#target"
  hx-swap="innerHTML">
  <input type="text" name="forename">
  <input type="text" name="surname">
</form>
```

Response is made to the server for each input updation.

Infinite Scroll - Update index.html AND Extension models_list.html

(Continuation of - Search for users' models 🔎)

project_folder/app_name/templates/app_name/index.html </> index.html

```
<form action="{% url 'index' %}" enctype="multipart/form-data" method="POST"
  hx-post="{% url 'index' %}"
  hx-target="#models"
  hx-trigger="keyup delay:500ms"
  hx-swap="innerHTML"
  hx-include="[name='search_query'], [name='creator_value'],
  [name='country_code_value']"
>
```

...

```
</form>      NOTE: <form> doesn't need - action, nor method with HTMX
```

```
<hr class="pt-3" We will include them anyways, to show additional functionality:
  parse request with URL parameters on <button type="submit">
{%- if models %}
  <ul id="models" class="list-group pb-3">
    {% include 'app_name/extensions/models_list.html' %}
  </ul>
{%- endif %}
```

Update <form> with action to indexView, and include models_list.html below it

NOTE: models_list.html is defined at the next page

NOTE: app_name/extensions/models_list.html (optional path)

```
<a href="{% url 'form' %}" class="float-start">add model</a>
<a href="{% if request.user.is_authenticated %}{% url 'profile' request.user.id %}{%
else %}{% url 'login' %}{% endif %}" class="float-end">
  <button type="submit" class="btn btn-primary ms-3">profile</button>
</a>
<form action="{% url 'logout' %}" method="POST" class="float-end">
  {% csrf_token %}
  <button type="submit" class="btn btn-warning">logout</button>
</form>

<form action="{% url 'index' %}" enctype="multipart/form-data" method="POST" ...
```

It's better to hoist the other options, since we gonna scroll infinitely

 project_folder/app_name/templates/app_name/extensions/models_list.html </>

```
{% if models %}  
    {% for model in models %}  
        <li class="list-group-item">  
            <h2>{{ model.title }}</h2>  
            <p>{{ model.country }}</p>  
            <small class="text-muted">{{ model.created_at }}</small>  
            <section class="d-flex justify-content-between" aria-label="model-options">  
                <a href="{% url 'edit_model' model.id %}">  
                    <button class="btn btn-secondary" aria-label="edit-model">Edit</button>  
                </a>  
                <a href="{% url 'delete_model' model.id %}">  
                    <button class="btn btn-danger" aria-label="delete-model">Delete</button>  
                </a>  
            </section>  
        </li>  
    {% endfor %}  
{% endif %}
```

```
{% if models %}  
    <div class="loader mx-auto my-5"  
        hx-trigger="revealed"  
        hx-swap="outerHTML"  
        hx-get="{% url 'index' %}?page={{ next_page_number }}{% if  
search_filter_params %}&{{ search_filter_params }}{% endif %}">  
        </div>  
    {% else %}  
        <p class="text-center mx-auto my-5">No more models to load</p>  
    {% endif %}
```

 project_folder/app_name/static/app_name/css/extensions/model_list.css



```
#models li {  
    animation: linear 0.5s fadeln;  
}  
  
@keyframes fadeln {  
    0% {  
        opacity: 0;  
    }  
    100% {  
        opacity: 100;  
    }  
}
```

Fade-In animation when **models** are included in **indexView**

```

Custom Loader / Spinning Circle .spinner-border in bootstrap5

.loader {
    width: 48px;
    height: 48px;
    border-radius: 50%;
    position: relative;
    animation: rotate 1s linear infinite;
}

.loader::before {
    content: "";
    box-sizing: border-box;
    position: absolute;
    inset: 0px;
    border-radius: 50%;
    border: 5px solid gray;
    animation: prixClipFix 2s linear infinite;
}

@keyframes rotate {
    100% {
        transform: rotate(360deg);
    }
}

@keyframes prixClipFix {
    0% {
        clip-path: polygon(50% 50%, 0 0, 0 0, 0 0, 0 0, 0 0);
    }
    25% {
        clip-path: polygon(50% 50%, 0 0, 100% 0, 100% 0, 100% 0, 100% 0);
    }
    50% {
        clip-path: polygon(50% 50%, 0 0, 100% 0, 100% 100%, 100% 100%, 100% 100%);
    }
    75% {
        clip-path: polygon(50% 50%, 0 0, 100% 0, 100% 100%, 0 100%, 0 100%);
    }
    100% {
        clip-path: polygon(50% 50%, 0 0, 100% 0, 100% 100%, 0 100%, 0 0);
    }
}

```

 project_folder/app_name/templates/app_name/index.html </> index.html

```

{% block head %}
    <link rel="stylesheet" href="{% static 'app_name/css/extensions/model_list.css' %}">
{% endblock %}

```

Infinite Scroll - Update indexView

project_folder/app_name/views.py

views.py

Update `indexView`. You may even scrap it and replace it with the following code

```
def indexView(request):
```

```
    model_list = ModelName.objects.all()
```

```
    context = {}
```

```
    R = request.GET
```

```
    # Handle POST request (search form submission)
```

```
    if request.method == 'POST':
```

```
        R = get_request_with_valid_search_filter_parameters(request)
```

```
        params = get_encoded_parameters_from_request(R)
```

```
        request.session['search_filter_params'] = params
```

```
        # Redirect to indexView with search filter parameters in URI
```

```
        return http_response_redirect_with_parameters('index', params)
```

```
    # Retrieve search parameters from session
```

```
    if 'search_filter_params' in request.session:
```

```
        context['search_filter_params'] = request.session['search_filter_params']
```

NOTE: `context` and `R`. Hoisting them, allows us to apply changes at any time

NOTE: `indexView()` is called, when POST or GET request is sent via template

Therefore, `R` will be overwritten at - if `request.method == 'POST'`. However, updated `R` has no effect on the rest of the codebase due to a `return` keyword

We can remove the `return http_response_redirect_with_parameters()` and the code will still work, however, the client will no longer be able to parse the whole website with the new request containing search filter query parameters. (localhost:port/?param1=Hello%20World¶m2=Hi)

NOTE: `request.session['search_filter_params']`. Think of it as a cookie, it saves value locally on a client

This value, can be accessed at later point, such as when we call `indexView()` again. Hence, if '`search_filter_params`' in `request.session`:

`search_filter_params` is set in `context`, so that it can be sent to the `models_list.html` - hx-get request

Django automatically initializes an expiry event for sessions.

Session is removed upon browser's termination. (For example; when client closes the browser)

```

# Get all models ordered by title
model_list = ModelName.objects.all().order_by('title')

# Filter search_models based on request parameters
search_models = model_list

for key, value in R.items():
    if not value:
        continue
    if key == 'search':
        search_models = search_models.filter(title__icontains=value)
    elif key == 'country_code':
        search_models = search_models.filter(country=value)
    elif key == 'creator_id':
        search_models = search_models.filter(creator=value)
    elif key == 'creator':
        search_models = search_models.filter(creator_username__icontains=value)

# Get other_models (irrelevant to search)
search_models_ids = search_models.values_list('id', flat=True)
other_models = model_list.exclude(id__in=search_models_ids)

```

Update filtered model to search_models

Models that are NOT relevant to the search, will be stored in **other_models**
 we will render them, after all **search_models** in **models_list.html** template extension

```

paginator = Paginator(model_list, 5) # Show 5 models per page
page_number = request.GET.get('page', 1) # Get 'page' from request's query:

try:
    models = paginator.page(page_number)
except PageNotAnInteger:
    # If page is not an integer, deliver the first page
    models = paginator.page(1)
except EmptyPage:
    # If page is out of range (e.g., 9999), deliver
    models = paginator.page(paginator.num_pages)

```

Remove old model paginator

```
# Paginate 'search_models' and 'other_models'  
paginator_search = Paginator(search_models, 5)  
paginator_other = Paginator(other_models, 5)
```

```
# Get page number from request  
try:  
    page_number = int(request.GET.get('page', 1))  
except ValueError:  
    page_number = 1
```

Paginate - **search_models**, and **other_models**

NOTE: page_number - localhost:port/?**page=N** - page_number = **N**

hx-get in **div.loader**, at **models_list.html** template extension, will request 5 models from **indexView()**

```
# Start of - Determine which paginator to use -----  
  
if page_number <= paginator_search.num_pages:  
    # Get relevant to search results. (models = search_models)  
    models = paginator_search.get_page(page_number)  
    models_type = 'search_relevant'  
  
elif page_number <= paginator_search.num_pages + paginator_other.num_pages:  
    # Get irrelevant to search results. (models = other_models)  
    page_number_other = page_number - paginator_search.num_pages  
    models = paginator_other.get_page(page_number_other)  
    models_type = 'search_irrelevant'  
  
else:  
    models = False  
    models_type = False  
    next_page_number = False  
  
# END of - Determine which paginator to use -----
```

Prepare **models** for template's context data.

NOTE: IF page_number exceeds total page count of paginated search_models,
THEN other_models will be served until their page count is also exceeded.

```

# Update context
context.update({
    'models': models,
    'models_type': models_type,
    'next_page_number': page_number + 1,
})

# Render the appropriate template
template = 'app_name/extensions/models_list.html' if request.htmx else
'app_name/index.html'

return render(request, 'app_name/index.html', {'models': models})
return render(request, template, context)

```

NOTE: `page_number` is incremented, and served as `next_page_number`

NOTE: `models_list.html` extension, makes a HTMX response using context data:
`localhost:port/?page={{ next_page_number }}&{{ search_filter_params }}`

NOTE: IF request has been made by the HTMX,
THEN `models_list.html` extension template, is returned back to the client.

Great Job!

That was a lot, though minor improvement is needed. Let's show the client their search result relevancy.

Infinite Scroll - Models relevency message

project_folder/app_name/views.py

views.py

```
def indexView(request):
...
elif page_number <= paginator_search.num_pages + paginator_other.num_pages:
    # Get irrelevant to search results. (models = other_models)
    page_number_other = page_number - paginator_search.num_pages
    models = paginator_other.get_page(page_number_other)
    models_type = 'search_irrelevant'

    # One time message - End of relevant search results
    if page_number == paginator_search.num_pages + 1:
        context.update({'start_other_models_message': True})

else:
    models = False
    models_type = False
...
```

Add following statement for a one time message, at `indexView()`

project_folder/app_name/templates/app_name/extensions/models_list.html </>

```
{% if models and start_other_models_message %}
<p class="text-center bg-dark my-4 mx-auto w-100 p-3 fs-4 text-white">
    Irrelevant models will start to appear
</p>
{% endif %}

...
```

Add the following statement at the top of the models_list.html extension

Chat Rooms    

made with: HTMX + Dafne ASGI WebSockets

This is the most complicated part of the documentation. It's best to proceed with 3 months of active Django development experience. Additionally, this section is based of a following

YouTube tutorial - [Chat App with Django Channels](#) by Andreas Jud
(Accessed: 10 March 2025)

Please watch the tutorial before commiting to this chat-app development.

Terminology

Client	In this context, a browser prompting requests to the server
Request	Method: POST, GET, UPDATE, DELETE, HTMX , WS etc... made by the client , which then is sent to the server
Response	Content-Type: text / html / json / xml / csv / application etc... made by the server , which then serves it to the client
WebSocket	Communication protocol. Provides consistent connection between client & server
Web Server Gateway Interface (WSGI)	Handles requests and responses , synchronously . (like procedurally, where a task must finish before another can start) Is configured by default @ settings.py
Asynchronous Server Gateway Interface (ASGI)	Handles requests and responses , synchronously & asynchronously . (like concurrently, where tasks are being proccesed idependently) And is requiried for processing WebSocket connections. Can be configured as default @ settings.py
	<p>Like WSGI, it relies on a: request response cycle</p> <p>Additionally includes a: handshake, (hashed keys) to establish a secure connection between client & server</p>

Channel	Django extension for handling real-time functionality & protocols like WebSockets				
Channel Layers	<p>Enables communication between different parts of an app & Handles, asynchronous tasks and real-time features for: consumers</p> <ul style="list-style-type: none"> - Has to be configured @ settings.py - Can broadcast message to multiple consumers <p>For production-based applications, initialize Channel Layers with in-memory data structure, like:</p> <table border="1"> <tr> <td>Redis</td> <td>Valkey</td> </tr> <tr> <td>Redis is a multi-purpose tool: a cache, a database (RAM storage with RDB snapshots), and a message broker (publish/subscribe). Once open-source, it was later acquired, and license changes diminished user trust in it.</td> <td>A Redis version predating the policy changes. Many Redis open-source developers shifted to Valkey, ensuring a trustworthy, open-source alternative. Numerous companies impacted by Redis policies now financially back Valkey.</td> </tr> </table>	Redis	Valkey	Redis is a multi-purpose tool: a cache , a database (RAM storage with RDB snapshots), and a message broker (publish/subscribe). Once open-source, it was later acquired, and license changes diminished user trust in it.	A Redis version predating the policy changes. Many Redis open-source developers shifted to Valkey, ensuring a trustworthy, open-source alternative . Numerous companies impacted by Redis policies now financially back Valkey.
Redis	Valkey				
Redis is a multi-purpose tool: a cache , a database (RAM storage with RDB snapshots), and a message broker (publish/subscribe). Once open-source, it was later acquired, and license changes diminished user trust in it.	A Redis version predating the policy changes. Many Redis open-source developers shifted to Valkey, ensuring a trustworthy, open-source alternative . Numerous companies impacted by Redis policies now financially back Valkey.				
Consumer	Similar to a Django View , but it supports asynchronous communication and real-time events.				
Routing	<p>Django Channels - routing.py, is like urls.py, where we can define paths(), in this context for ASGI protocols. We will use this file to define WebSocket routes/paths only.</p> <p>urls.py defines URL routing only for HTTP requests</p>				
Out of Band (OOB)	<p>Security testing technique that focuses on identifying and detecting security vulnerabilities through external interactions, such as; <u>blind routing code executions</u></p> <p>HTMX includes OOB, for example: hx-swap-oob However, it is just a naming convention.</p> <p>In this context, it focuses on updating parts of the DOM that are not the direct target of the request.</p>				
Ignore this note					

Create Chat App

```
█ (venv) @ project_folder
```

 manage.py

```
python manage.py startapp chat
```

Register Chat App and it's URLs to Django Website

 project_folder/my_website/settings.py settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    # Add libraries above apps
    'crispy_forms',
    'crispy_bootstrap5',
    ...
    # Add apps below
    'app_name.apps.AppNameConfig',
    'chat.apps.ChatConfig',
]
```

```
app_name > 🐍 apps.py
1  from django.apps import AppConfig
2
3
4  class AppNameConfig(AppConfig):
5      default_auto_field = 'django.db.models.BigAutoField'
6      name = 'app_name'
```

 project_folder/my_website/urls.py urls.py

```
urlpatterns = [
    ...
    path('', include('app_name.urls')),

    # imports URLs from my_website/users/urls.py
    path('chat/', include('chat.urls')),
]
```

 project_folder/chat/urls.py urls.py

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

urlpatterns = [
    path('', views.roomView, name='room'),
]
```

```

from django.db import models

class Room(models.Model):
    name = models.CharField(max_length=128, unique=True)

    def __str__(self):
        return self.name

class Message(models.Model):
    room = models.ForeignKey(
        Room,
        related_name='messages',
        on_delete=models.CASCADE
    )
    author = models.ForeignKey('auth.User', on_delete=models.CASCADE)
    message = models.CharField(max_length=1000)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-created_at']

    def __str__(self):
        return f'{self.author.username}: {self.message}'

```

NOTE: User = 'auth.User', and
does NOT require: django.contrib.auth.models import User

NOTE: class Meta - ordering includes: "-", this will make a DESCENDING order

related_name in Message models.ForeignKey(...)
allows you to access all Message instances of related Room instances

room = get_object_or_404(...)
room.messages → All instance_message of instance_room

project_folder/chat/admin.py

(Register Room & Message) admin.py

```
from django.contrib import admin
from .models import Room, Message
```

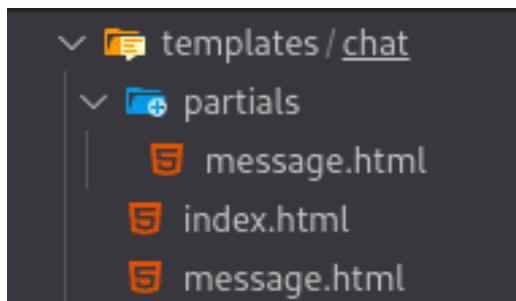
```
# Register your models here.
admin.site.register(Room)
admin.site.register(Message)
```

project_folder/chat/forms.py

(Chat index.html <input>) forms.py

```
from .models import Message
from django import forms
```

```
class MessageForm(forms.ModelForm):
    class Meta:
        model = Message
        fields = ['message']
```



CREATE:

templates/chat/partials/message.html

templates/chat/message.html

templates/chat/index.html

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
from django.http import HttpResponseRedirect
from django.utils import timezone

from .forms import MessageForm
from .models import Room


@login_required
def roomView(request):

    room = get_object_or_404(Room, name='public')
    form = MessageForm()

    if request.method == 'POST':
        form = MessageForm(request.POST)

        if form.is_valid():
            message = form.save(commit=False)
            message.author = request.user
            message.room = room
            message.save()

            context = {
                'msg': message,
                'user': message.author,
                'today': timezone.now().date().strftime("%Y-%m-%d")
            }
            return render(request, 'chat/partials/message.html', context)

    context = {
        'form': form,
        'user': request.user,
        'room_name': room.name,
        'room_messages': room.messages.all()[:30],
        'today': timezone.now().date().strftime("%Y-%m-%d")
    }

    return render(request, 'chat/index.html', context)
```

Create **Room** with name “**public**” in the **database** (localhost:port/admin)

 project_folder/chat/templates/chat/index.html </> index.html

```
{% extends 'app_name/base.html' %}  
{% load static %}  
{% load crispy_forms_tags %}  
  
{% block content %}  
  
<h1>{{ room_name|title }}</h1>  
<hr>  
  
{% if request.user.is_authenticated %}  
  
<section id="chat-messages" class="overflow-auto" style="height: 50vh">  
    {% if room_messages %}  
        {% for msg in room_messages reversed %}  
            {% include 'chat/message.html' %}  
        {% endfor %}  
    {% endif %}  
</section>  
  
<hr class="my-5">  
  
<form id="chat-form"  
      hx-post="{% url 'room' %}"  
      hx-trigger="submit delay:500ms"  
      hx-target="#chat-messages"  
      hx-swap="beforeend">  
    {% csrf_token %}  
    {{ form|crispy }}  
</form>  
  
{% endif %}  
  
{% endblock %}
```

NOTE: `reversed` in - `{% for msg in room_messages ... %}` - iterates backwards

NOTE: custom variable - `msg`, is usable in `templates/chat/partials/message.html`

```
 project_folder/chat/templates/chat/partials/message.html    </> message.html  
  
{% include 'chat/message.html' %}
```

```
 project_folder/chat/templates/chat/message.html      </> message.html  
  
<message>  
  
  <div style="width: clamp(350px, 32vw, 500px)" class="msg-container  
    {% if user == msg.author %} text-white bg-info ms-auto  
    {% else %} bg-light me-auto  
    {% endif %} text-white my-4 p-3">  
  
    <p class="msg-body fw-bold">{{ msg.message }}</p>  
  
    <label class="msg-author fw-bold  
      {% if user == msg.author %} text-end{% else %} w-100{% endif %}">  
      {% if user == msg.author %} Me{% else %} {{ msg.author }}{% endif %}  
    </label>  
  </div>  
  
  <div class="msg-date  
    {% if user == msg.author %} text-end{% else %} text-start{% endif %}">  
    {% if today == msg.created_at|date:"Y-m-d" %}  
      {{ msg.created_at|date:"H:i" }}  
    {% else %}  
      {{ msg.created_at|date:"d M Y, H:i" }}  
    {% endif %}  
  </div>  
  
</message>
```

NOTE: Additional spaces are added to the `class` attributes because of the way this code is spaced out with the **IF-ELSE** statements.

The HTML template works. However, extra spacing may obfuscate the `class` attribute, worsening debugging experience in developer tools

We can enhance the template response by stripping off these extra, unnecessary space characters.

Let's implement **CUSTOM** `{% strip %}` template tag (OPTIONAL)

```
📝 project_folder/app_name/templatetags/strip_spaces.html      🐍 strip_spaces.py

from django import template
import re # regular expression

register = template.Library()

class StripSpacesNode(template.Node):
    def __init__(self, nodelist):
        self.nodelist = nodelist

    def render(self, context):
        output = self.nodelist.render(context)
        return re.sub(r'\s+', ' ', output).strip()

@register.tag(name='strip')
def do_strip_spaces(parser, token):
    nodelist = parser.parse(('endstrip',))
    parser.delete_first_token()
    return StripSpacesNode(nodelist)
```

Create new directory, `templatetags` inside app: `app_name`, and

Include empty file named: `__init__.py` inside `templatetags` directory

NOTE: `@register.tag(name='strip')` → `{% strip %}`, and
`parser.parse(('endstrip',))` → `{% endstrip %}`

You can create **templatetags** folder in any of the apps, like **chat**. I decided to implement it in **app_name**, because it is the main app.

Once the template tag is registered, it is then accessible across all applications of the **project_folder** directory, regardless of which application contains the **templatetags** folder.

```
📝 project_folder/chat/templates/chat/message.html      </> message.html

{% load strip_spaces %}
{% strip %}
<message>...</message>
{% endstrip %}
```

Include - HyperScript

(OPTIONAL)

```
project_folder/chat/templates/chat/index.html      </> index.html

...
<hr class="my-5">
<form
  hx-post="{% url 'room' %}"
  hx-trigger="submit delay:500ms"
  hx-target="#chat-messages"
  hx-swap="beforeend"
  _="on htmx:afterRequest reset() me"    ← clears <input> after submit
>
  {% csrf_token %}
  {{ form|crispy }}
</form>

...
NOTE: "_" is HyperScript. Include CDN @ app_name/templates/app_name/base.html
<script name="hyperscript.js"
  src="{% static 'app_name/js/lib/hyperscript.min.js' %}" defer></script>
```

HyperScript is compatible with **HTMX** because both have been created by the same developer: “Carson Gross”.

HyperScript is a JavaScript library that provides different approach to programming. Like jQuery, it minimizes lengthy JavaScript syntax into a more compact base.

HyperScript can be used in a script tag: `<script name="text/hyperscript">`

The HyperScript syntax is not yet fully adopted by the large AI models (As of 11/March/2025).

Therefore, you should follow the official documentation instead, if you plan on using this library.

```

...
{% block base %}

<script name="chat-messages.js" type="text/javascript">



---


    const chatForm = document.getElementById('chat-form');
    const chatMessages = document.getElementById('chat-messages');

    const scrollToBottom = () => {
        chatMessages.scrollTop = chatMessages.scrollHeight;
    }



---


    const observer = new MutationObserver(scrollToBottom);
    observer.observe(chatMessages, { childList: true });



---


    document.addEventListener('htmx:afterRequest', scrollToBottom);
    scrollToBottom();



---


</script>
{% endblock %}

```

IF you are **NOT** using **HyperScript**, **THEN** extend htmx:afterRequest EventListener

```

...
const observer = new MutationObserver(scrollToBottom);
observer.observe(chatMessages, { childList: true });

document.addEventListener('htmx:afterRequest', () => {
    chatForm.reset();
    scrollToBottom();
});

scrollToBottom();

</script>
{% endblock %}

```

NOTE: This script is essential to scroll the client to the newest message.
NOTE: `hx-swap="beforeend scroll:bottom"` can be used to display newest message.

However, we will use **WebSockets** to support real-time conversations.
`hx-swap` unfortunately, is not a viable option when using **WebSockets**.

Chat with Django Channels

💻 (venv) @ project_folder

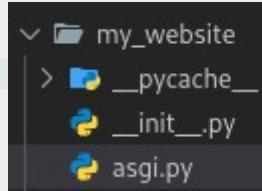
📚 pip library

```
pip install -U 'channels[daphne]'  
pip freeze > requirements.txt # update requirements.txt
```

📝 project_folder/my_website/settings.py

🐍 settings.py

```
INSTALLED_APPS = [  
    'daphne', # Add to the top  
    'django.contrib.admin',  
    ...  
]
```



```
# WSGI_APPLICATION = 'my_website.wsgi.application' ← comment out the WSGI setup  
ASGI_APPLICATION = 'my_website.asgi.application'
```

📝 project_folder/my_website/asgi.py

🐍 asgi.py

```
...  
For more information on this file, see  
https://docs.djangoproject.com/en/5.1/howto/deployment/asgi/  
"""
```

```
import os
```

```
from django.core.asgi import get_asgi_application  
from channels.auth import AuthMiddlewareStack  
from channels.routing import ProtocolTypeRouter, URLRouter  
from channels.security.websocket import AllowedHostsOriginValidator  
from chat import routing # routing.py in chat application
```

```
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'my_website.settings')
```

```
application = get_asgi_application()  
application = ProtocolTypeRouter({  
    "http": get_asgi_application(),  
    "websocket": AllowedHostsOriginValidator(  
        AuthMiddlewareStack(URLRouter(routing.websocket_urlpatterns))  
    )  
})
```

Update ASGI application to rout WebSocket connections

project_folder/my_website/routing.py

🐍 routing.py

```
from django.urls import path
from .consumers import *

websocket_urlpatterns = [
    path('ws/chatroom/<str:room_name>', ChatroomConsumer.as_asgi()),
]
```

Think of **routing** AS urls.py - These are **WebSocket** connections instead of **HTTP**

project_folder/my_website/consumers.py

🐍 consumers.py

```
from channels.generic.websocket import WebsocketConsumer
from django.shortcuts import get_object_or_404
from django.template.loader import render_to_string
from django.utils import timezone

from .models import Message, Room
import json

class ChatroomConsumer(WebsocketConsumer):

    def connect(self):
        self.user = self.scope['user']
        self.room_name = self.scope['url_route']['kwargs']['room_name']
        self.room = get_object_or_404(Room, name=self.room_name)
        self.accept()

    def receive(self, text_data):
        text_data_json = json.loads(text_data)

        message = Message.objects.create(
            room=self.room,
            author=self.user,
            message=text_data_json['message']
        )
        context = {
            'msg': message,
            'today': timezone.now().date().strftime("%Y-%m-%d"),
            'user': self.user
        }

        html = render_to_string('chat/partials/message.html', context)
        self.send(text_data=html)
```

NOTE: WebSocket Views/Consumers **DO NOT** have `request` Object

NOTE: Consumers uses different functions for responding back to the client(s)

NOTE: We **can't use**: `{% url 'websocket_pathname' %}`
because it is handled by HTTP Django Channel

Use `self.scope` object

to retrieve data, such as, user or path's parameters: `path('chat/<str:parameter>', ...)`

Some differences between **Views & Consumers**

views.py	consumers.py
<code>HttpResponse()</code> sends response data to client: HTML, XML, JSON...	<code>render_to_string(template, context)</code> similar to <code>render()</code> in views.py, and does NOT require <code>request</code> object
<code>render(request, template, context)</code> renders templates <code>{% if ... %}{% else %}</code> , before sending them using <code>HttpResponse()</code> with <code>context</code> data	
<code>def myView(request, parameter)</code> to retrieve data, such as path's params: <code>path('chat/<str:parameter>', views.myView, name...)</code>	<code>self.scope['url_route']['kwargs']['ID']</code> to retrieve data, such as path's params: <code>path('ws/<int:ID>', myConsumer.as_asgi())</code>
<code>form.cleaned_data['input_name']</code> is used to get POST data like <form>'s <input> when <code>form.is_valid()</code>	<code>json.loads(text_data)</code> at <code>def receive()</code> like <code>form.cleaned_data</code> , is used to get POST data. <code>json.loads(text_data)['input_name']</code>
<code>return HttpResponse()</code> <code>return render()</code>	<code>self.send(text_data=render_to_string())</code>

 project_folder/app_name/templates/app_name/base.html

</> base.html

NOTE: HTMX does NOT support WebSockets by default

Hence, include HTMX WebSockets Extension from:

<https://v1.hmx.org/extensions/web-sockets/>

```

...
<script name="htmx.js" src="{% static 'app_name/js/lib/htmx.min.js' %}" defer></script>
<script name="htmx-websockets.js" src="{% static 'app_name/js/lib/htmx_websockets.js' %}" defer></script...
<script name="hyperscript.js" src="{% static 'app_name/js/lib/hyperscript.min.js' %}" defer></script>
{% block head %}{% endblock %}
...

```

POST message via WebSocket Channel

project_folder/chat/templates/chat/index.html </> index.html

```

...
<form id="chat-form"
      hx-post="{% url 'room' %}"
      hx-trigger="submit delay:500ms"
      hx-target="#chat-messages"
      hx-swap="beforeend">
  {% csrf_token %}
  {{ form|crispy }}
</form>
...
...
addEventListener('htmx:afterRequest') addEventListener('htmx:wsAfterSend')
...

```

```

...
<form id="chat-form"
      hx-ext="ws"
      ws-connect="/ws/chatroom/public/"
      ws-send>
  {% csrf_token %}
  {{ form|crispy }}
</form>
...
...

```

NOTE: "public" at ws-connect, is the created **Room** model with name="public"

This form POSTs <input> message TO → **ChatroomConsumer** at customers.py

project_folder/chat/templates/chat/partials/message.html </> message.html

```

<div id="chat-messages" hx-swap-oob="beforeend">
  {% include 'chat/message.html' %}
</div>

```

NOTE: hx-swap-oob (out of band)
is used only with a server-response. Hence, the use of a message.html partial

It targets any DOM element with id: #chat-messages, and in this context;

It appends, the wrapped innerHTML, at the end of the target

Broadcast message to Consumers with Channel Layers

```
project_folder/my_website/settings.py           settings.py

CHANNEL_LAYERS = {
    'default': { 'BACKEND': 'channels.layers.InMemoryChannelLayer' }
}

InMemoryChannelLayer is NOT suitable in production!
Use it in DEBUG=True ONLY. For production USE Redis or Valkey
```

```
project_folder/my_website/consumers.py          consumers.py

...
from channels.generic.websocket import WebsocketConsumer # ← AsyncWeb...
from asgiref.sync import async_to_sync    # ← you can also import sync_to_async
...

class ChatroomConsumer(WebSocketConsumer):

    def connect(self):
        self.user = self.scope['user']
        self.room_name = self.scope['url_route']['kwargs']['room_name']
        self.room = get_object_or_404(Room, name=self.room_name)

        async_to_sync(self.channel_layer.group_add)(
            self.room_name, self.channel_name
        )

        self.accept()
...
```

NOTE: `async_to_sync()` → Allows asynchronous calls in synchronous functions

NOTE: `group_add()` → Connects `channel` to the `group = room_name`

Alternatively, you can replace: `WebSocketConsumer` TO `AsyncWebSocketConsumer`

```
class ChatroomConsumer(AsyncWebSocketConsumer):

    async def connect(self):
        await self.user = self.scope['user']
        await self.room_name = self.scope['url_route']['kwargs']['room_name']
        await self.room = get_object_or_404(Room, name=self.room_name)
        await self.channel_layer.group_add(self.room_name, self.channel_name)
        await self.accept()
```

```

def disconnect(self, close_code):
    async_to_sync(self.channel_layer.group_discard)(
        self.room_name, self.channel_name
    )

```

ADD `disconnect()` method TO LEAVE the `group` when the `channel` DISCONNECTS

```

def receive(self, text_data):
    text_data_json = json.loads(text_data)

    message = Message.objects.create(
        room=self.room,
        author=self.user,
        message=text_data_json['message']
    )
    context = {
        'msg': message,
        'today': timezone.now().date().strftime("%Y-%m-%d"),
        'user': self.user
    }

    html = render_to_string('chat/partials/message.html', context)
    self.send(text_data=html)

    event = {
        'type': 'message_handler',
        'message_id': message.id
    }
    async_to_sync(self.channel_layer.group_send)(
        self.room_name, event
    )

```

NOTE: `group_send()` → sends `response` TO all connected `Consumers`
Create `message_handler()` method below `receive()` method

```

def message_handler(self, event):
    message_id = event['message_id']
    message = Message.objects.get(id=message_id)
    context = {
        'msg': message,
        'today': timezone.now().date().strftime("%Y-%m-%d"),
        'user': self.user
    }

    html = render_to_string('chat/partials/message.html', context)
    self.send(text_data=html)

```

Online Status (Offline 5 Online)

 project_folder/chat/**models.py**

(Room & Message)  **models.py**

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

class Room(models.Model):
    name = models.CharField(max_length=128, unique=True)
    users_online = models.ManyToManyField(
        User,
        related_name='online_in_rooms',           ← user_instance.online_in_rooms
        blank=True
    )

    def __str__(self):
        return self.name

class Message(models.Model):
    room = models.ForeignKey(
        Room,
        related_name='messages',
        on_delete=models.CASCADE
    )
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    message = models.CharField(max_length=1000)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-created_at']

    def __str__(self):
        return f'{self.author.username}: {self.message}'
```

Update DATABASE: python manage.py makemigrations **THEN** python manage.py migrate

project_folder/my_website/consumers.py

consumers.py

```
...
```

```
class ChatroomConsumer(WebSocketConsumer):  
  
    def connect(self):  
        ...  
  
        # ADD User TO online list, AND UPDATE online count  
        if self.user not in self.room.users_online.all():  
            self.room.users_online.add(self.user)  
            self.update_online_count()  
  
        self.accept()  
  
  
    def disconnect(self, close_code):  
        async_to_sync(self.channel_layer.group_discard)(  
            self.room_name, self.channel_name  
        )  
        # REMOVE User FROM online list, AND UPDATE online count  
        if self.user in self.room.users_online.all():  
            self.room.users_online.remove(self.user)  
            self.update_online_count()  
  
  
    def update_online_count(self):  
        online_count = self.room.users_online.count()  
        event = {  
            'type': 'online_count_handler',  
            'online_count': online_count  
        }  
        async_to_sync(self.channel_layer.group_send)(  
            self.room_name, event  
        )  
  
  
    def online_count_handler(self, event):  
        online_count = event['online_count']  
        partial = 'chat/partials/online_count.html'  
  
        html = render_to_string(partial, {'online_count': online_count})  
        self.send(text_data=html)
```

```
project_folder/chat/templates/chat/partials/online_count.html </>

{% with online_count|add:"-1" as online_count %}
{% load strip_spaces %}
{% strip %}

<div id="online-status" hx-swap-oob="outerHTML">

    <span class="icon">
        {% if online_count > 0 %}●
        {% else %}○
        {% endif %}
    </span>

    <span class="online-count
        {% if online_count > 0 %} text-success
        {% else %} text-secondary
        {% endif %}>
        {% if online_count > 0 %} {{ online_count }}
        {% else %} 0
        {% endif %} Online
    </span>

</div>
{% endstrip %}
{% endwith %}
```

```
project_folder/chat/templates/chat/index.html
```

...

```
<h1>{{ room_name|title }}</h1>
<hr>
<div id="online-status" class="text-primary">
    <span class="spinner-grow spinner-grow-sm"></span>
    <span role="status">Connecting...</span>
</div>
<hr>
```

...

(Next step requires a break... Take a break from the **Chat** app!)

Optimize Chat app with Valkyrie TO improve server performance

In this tutorial, Valkyrie is used as a, **Message Broker**:

Message Broker

A software component that facilitates communication between different applications or services by translating, routing, and managing messages. It acts as an intermediary, ensuring that messages are delivered reliably and efficiently, even if the sender and receiver are not directly connected or use different protocols.

Redis: While primarily an in-memory data store;

It can act as a lightweight message broker using its **Pub/Sub** (Publish/Subscribe) feature. It allows messages to be published to **channels** and received by subscribers in real-time.

(AI Generated: Copilot @ copilot.microsoft.com, 18/Mar/2025)

Django + Redis, Architecture

Client requests **App FOR** a resource

THEN App searches for the resource **IN Redis**

IF resource **IS FOUND IN Redis**

THEN Redis sends resource to **App** which passes it to the **Client**

ELSE IF resource **IS NOT FOUND IN Redis**

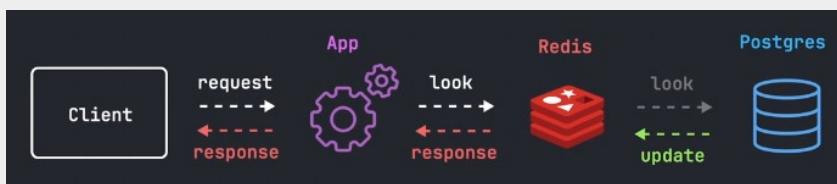
THEN Redis searches for the resource **IN Database**

IF resource **IS FOUND IN Database**

THEN Redis FETCHES resource **AND** sends it to **App → Client**

NOTE: Application in this context is: **chat**

NOTE: Fetching resource to **Redis**, allows us to access it when needed directly from **Redis**, instead of the **Database**



Chat with Valkey Channels

(venv) @ project_folder

 pip library

```
pip install channels-valkey
pip freeze > requirements.txt # update requirements.txt
```



 project_folder/my_website/settings.py

 settings.py

```
CHANNEL_LAYERS = {}

if DEBUG:
    CHANNEL_LAYERS = {
        'default': { 'BACKEND': 'channels.layers.InMemoryChannelLayer' }
    }
else:
    CHANNEL_LAYERS = {
        'default': {
            'BACKEND': 'channels_valkey.pubsub.ValkeyPubSubChannelLayer',
            'CONFIG': {
                'hosts': [(os.getenv('VALKEY_HOST_URL'))],
            },
        },
    }
```

NOTE:

InMemoryChannelLayer is **NOT** suitable in **production!** Hence, the integration of **Valkey**

"BACKEND": "channels_valkey.core.ValkeyChannelLayer",
is the **original** layer, and implements channel and group handling itself.

"BACKEND": "channels_valkey.pubsub.ValkeyPubSubChannelLayer",
is newer and leverages Valkey Pub/Sub for message dispatch. This layer is currently at **Beta** status,
meaning it **may be subject to breaking** changes whilst it matures.

 project_folder/.env

.env

```
DEBUG='False' ← Set the DEBUG to False, in order to, test the Valkey channel
NOTE: You should always set DEBUG to False in production
```

```
...
```

```
# Railway App | deploy valkey in railway.com
```

```
VALKEY_HOST_URL='create redis app at railway.com, and paste redis_url here'
```

Private Chat Rooms



💻 (venv) @ project_folder

📚 pip library

```
pip install shortuuid
pip freeze > requirements.txt # update requirements.txt
```

Short UUID generates random string of unique set of characters.
It's not a standard build-in library hence, we have to install it.

📝 project_folder/chat/[models.py](#)

(Room & Message) 🐍 [models.py](#)

```
from django.db import models
from django.contrib.auth.models import User
import shortuuid
```

```
class Room(models.Model):
    name = models.CharField(
        max_length=128,
        unique=True,
        unique=shortuuid.uuid)
    users_online = models.ManyToManyField(
        User,
        related_name='online_in_rooms',
        blank=True
    )
    members = models.ManyToManyField(
        User,
        related_name='chat_rooms',
        blank=True
    )
    is_private = models.BooleanField(default=False)

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

Update DATABASE: python manage.py makemigrations **THEN** python manage.py migrate

Usecase example of chat room models with related_name property

In your Django models, `related_name` defines the reverse relationship name when accessing related objects.

Here's how it works in your case:

```
users_online = models.ManyToManyField(User,  
related_name='online_in_rooms')
```

This means a `User instance` can access all rooms where they are currently online using `user.online_in_rooms.all()`.

```
members = models.ManyToManyField(User, related_name='chat_rooms')
```

A `User instance` can access all rooms they are a member of using `user.chat_rooms.all()`.

```
room = models.ForeignKey(Room, related_name='messages',  
on_delete=models.CASCADE)
```

A `Room instance` can access all messages associated with it using `room.messages.all()`.

Get or create private chatroom (utility)

```
project_folder/chat/utils.py utils.py

from .models import Room

def get_or_create_chatroom_for_users(User_A, User_B):
    """
    This utility searches for an existing private chatroom between
    the provided user instances. If chatroom is not found,
    then a new private chatroom is created for those users.
    """

    if User_A == User_B:
        print('Cannot create chatroom for the same user!')
        return None

    my_chat_rooms = User_A.chat_rooms.filter(is_private=True)
    room_instance = None

    # GET private chatroom OF User_A and User_B
    if my_chat_rooms.exists():
        for room in my_chat_rooms:
            if User_B in room.members.all():
                room_instance = room
                break

    # Otherwise, create new private chatroom for them
    if room_instance is None:
        room_instance = Room.objects.create(is_private=True)
        room_instance.members.add(User_B, User_A)

    return room_instance
```

```
project_folder/chat/views.py views.py

from django.shortcuts import render, redirect, get_object_or_404
from django.contrib import messages
from django.contrib.auth.models import User
from django.contrib.auth.decorators import login_required
from django.http import Http404 # Replaced HttpResponseRedirect with HttpResponse
from django.utils import timezone

from .utils import get_or_create_chatroom_for_users
from .forms import MessageForm
from .models import Room
```

Update room View

```
... ← imports

@login_required
def roomView(request, room_name='public'): #added room_name parameter

    room = get_object_or_404(Room, name=room_name) #set name to room_name
    form = MessageForm()
    other_user = None

    # GET other_user IF room is private

    if room.is_private:
        members = room.members.all()

        if request.user not in members:
            raise Http404()

        for member in members:
            if member != request.user:
                other_user = member
                break

    ... ← if request.htmx:

        context = {
            'form': form,
            'user': request.user,
            'other_user': other_user,
            'room_name': room.name,
            'room_messages': room.messages.all()[:30],
            'today': timezone.now().date().strftime("%Y-%m-%d")
        }

        return render(request, 'chat/index.html', context)
```

NOTE: roomView provides context of **public** chatroom (**Room**) by default

NOTE: **other_user** is defined only when a provided chatroom (**Room**) **is_private**

Create chat with View

```
... ← roomView()

@login_required
def chatWithView(request, username):

    logged_in_user = request.user
    chat_with_user = User.objects.get(username=username)

    room_instance = get_or_create_chatroom_for_users(
        logged_in_user, chat_with_user
    )

    if room_instance is None:
        messages.error(request, 'You cannot chat with yourself!')
        return redirect('public-chatroom')

    return redirect('chatroom', room_instance.name)
```

NOTE: `chatWithView` redirects to path(..., `name='chatroom'`) which is `roomView`

NOTE: `roomView`'s parameter - `room_name='public'` becomes `room_instance.name`

Update urlpatterns

```
📝 project_folder/chat/urls.py                                     🐍 urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.roomView, name='public-chatroom'), #renamed path-name
    path('room/<str:room_name>', views.roomView, name='chatroom'),
    path('with-user/<username>', views.chatWithView, name='chat-with'),
]
```

Summary: '`chatroom`' **path** requires - the chatroom's name. For example: '`public`' (default). Accessing '`chat-with`' **path** requires - the NOT logged-in user's username.

- IF there is already existing room between NOT logged-in user & the logged-in user. THEN client is sent TO '`chatroom`' **path** with that room's name AS `room_name` arg.
- ELSE new room is created for those users with, generated by `shortuuid`, name instead before the client is sent TO '`chatroom`' **path**.

My chats Dropdown

Let's create a dropdown with available chat rooms that the logged-in user is a part of. Use `{% include %}` tag to implement it in needed templates, such as user's profile.

In future, consider using DynamicSelect instead TO include images of private chatrooms.

```
project_folder/chat/templates/chat/includes/my_chats.html </>

{% load strip_spaces %}



127


```

 project_folder/users/templates/users/profile.html (users profile) </>

```

{% extends 'app_name/base.html' %}
{% load static %}
{% load crispy_forms_tags %}

{% block title %}Profile Page{% endblock %}

{% block content %}






{% if profile_user != request.user %}

Direct Message

{% else %}
{% include 'chat/includes/my_chats.html' with
  button_class="btn-secondary border-top rounded-0 w-100"
  ul_class="w-100"
%}
{% endif %}

...


```

IF profile **IS** that of a logged-in user **THEN** show chats dropdown.

ELSE show button **TO** start a direct chat **WITH** the owner of that profile.

NOTE: `{% include %}` tag must be fully inline.

The above example contains organized `{% include %}` template tag.
However, this would not work in real Django application anytime soon.

This is because a template tag including argument **with** requires the variables to be set inline within the tag's scope.

Hence the unexpected error when linebreaks are present in an `{% include %}` tag

Update index template for one-to-one chatrooms

```
project_folder/chat/templates/chat/index.html           </> index.html

{% extends 'app_name/base.html' %}

{% load static %}

{% load crispy_forms_tags %}

{% block content %}

{% include 'chat/includes/my_chats.html' with div_class="float-end" %}

{%- if other_user %}

    <a class="float-start" href="{% url 'profile' other_user.id %}">
        
    </a>
    <h2>{{ other_user.username }}</h2>

{%- else %}

    <h1>{{ room_name|title }}</h1>
    <hr>
    <div id="online-status" class="text-primary">
        <span class="spinner-grow spinner-grow-sm"></span>
        <span role="status">Connecting...</span>
    </div>
    <hr>

{%- endif %}

<hr>
```

NOTE: {%- if other_user %} tag contains previously implemented code. & <hr> was removed

```
{% if request.user.is_authenticated %}

...

<form id="chat-form"
    hx-ext="ws"
    ws-connect="/ws/chatroom/public/"
    ws-connect="/ws/chatroom/{{ room_name }}/"
    ws-send>
    {% csrf_token %}
    {{ form|crispy }}
</form>

{%- endif %}
```


Install Stripe



💻 (venv) @ project_folder

📚 pip library

```
pip install stripe  
pip freeze > requirements.txt # update requirements.txt
```



- We will explore **Payment Element** ([stripe UI elements](#)) after **Card Element**
- **Payment Element** and **Card Element** differences: [click me](#)
- **Dummy Cards** used for Testing: <https://docs.stripe.com/testing#cards>
- **Stripe API**: <https://docs.stripe.com/api>

CLIENT LIBRARIES



```
$ pip install stripe
```

[STRIPE-PYTHON](#)

📝 project_folder/app_name/views.py

(Stripe API Example) 🐍 views.py

```
import stripe  
stripe.api_key = 'secret key (sk)' # https://dashboard.stripe.com/test/apikeys  
  
@login_required  
def paymentView(request):  
  
    basket_instance = get_object_or_404(BasketModel, owner=request.user)  
    if request.method == 'POST':  
  
        # Below code makes the purchase.  
        # The successful transaction is saved at your stripe account.  
  
        customer = stripe.Customer.create(  
            email=request.POST['email'],  
            name=request.POST['username'],  
            source=request.POST['stripeToken'])  
    )  
    charge = stripe.Charge.create(  
        customer,  
        amount=int(basket_instance.totalCost * 100), # amount=500 → $5.00  
        currency='USD',  
        description='My description'  
    )
```

NOTE: Stripe's attribute: 'amount', can't accept float-type-numbers (decimals)

Stripe API works only in certified with SSL & TLS domains (<https://> protocol)

Initialize API Keys

```
project_folder/.env .env

# Stripe API Keys

STRIPE_PUBLIC_KEY='pk_live_FldsSDFjkasdASd...'
STRIPE_SECRET_KEY='sk_live_dSDlasdWEpdI3Iw...'

STRIPE_PUBLIC_TEST_KEY='pk_test_oREsdkAWEj...'
STRIPE_SECRET_TEST_KEY='pk_test_SDerEjWEs0...'

API KEYS: https://dashboard.stripe.com/test/apikeys
```

```
project_folder/my_website/settings.py settings.py

# Stripe Payment System API keys

STRIPE_PUBLIC_KEY =
os.getenv('STRIPE_PUBLIC_TEST_KEY') if DEBUG else os.getenv('STRIPE_PUBLIC_KEY')

STRIPE_SECRET_KEY =
os.getenv('STRIPE_SECRET_TEST_KEY') if DEBUG else os.getenv('STRIPE_SECRET_KEY')

NOTE: STRIPE_PUBLIC_KEY      Are custom variables out of preference
      STRIPE_SECRET_KEY    DO NOT USE TEST_KEYs in production!
```

Follow Stripe's best practices:

<https://docs.stripe.com/payments/payment-element/best-practices>

Study the below article before implementing Stripe API

<https://support.stripe.com/questions/prohibited-and-restricted-businesses-list-faqs>

PCI and Data Protection Compliance



To ensure compliance with **PCI DSS**, **Stripe** helps safeguard sensitive payment data and adheres to local regulations. By handling and storing sensitive payment information, **Stripe** provides tokens for secure payment processing, protecting both clients and the company.

It is essential to transparently **inform clients** about where their data is stored and processed. **Stripe's terms and policies must be included on payment forms**, allowing clients to review and agree before proceeding with any transactions.

For additional protection and verification, implementing 3D Secure (e.g., **two-factor authentication**) is recommended. This helps prevent fraud and protects the company from liability for chargebacks due to fraudulent claims.

We must comply with **GDPR**, the **Data Protection Act 2018**, **RODO**, and similar laws to build trust, protect client data, and enhance our credibility. Clients should know where their data is stored, how it is processed, and that no unnecessary information is collected beyond what is required for our services.

To align with data protection regulations, **users** must be allowed to **deactivate** and **permanently delete** their data. This should be handled within a reasonable timeframe (30–90 days), during which payments can be processed, and checks for unlawful activities can be completed if requested by local authorities. Beyond this timeframe, all user data must be removed when possible, respecting clients' rights and fostering trust.

Finally, collaborating with legal advisors or consultants specializing in payment and data protection laws can ensure comprehensive compliance and anticipate potential challenges, reducing risks for both the company and its clients.

Payment System Vat Tax

VAT Compliance:

(Make OWN RESEARCH! Info may change)

If your business exceeds the VAT threshold in the UK but your clients are based in France, the VAT rules depend on the nature of your transactions:

- Business-to-Business (B2B):** If you're providing services or goods to VAT-registered businesses in France, the "place of supply" is considered to be France. In this case, the reverse charge mechanism often applies, meaning your French clients account for the VAT on their end. You wouldn't need to register for VAT in France, but you must ensure proper documentation, such as your client's VAT number, and include the reverse charge statement on your invoices.
- Business-to-Consumer (B2C):** If you're selling to individual consumers in France, you may need to register for VAT in France. This is because the "place of supply" rules for B2C transactions often require VAT to be charged in the customer's country. The threshold for VAT registration in France is typically much lower than in the UK, or it may not exist at all for cross-border sales.

If you're selling digital services, you might also need to use the VAT One Stop Shop (OSS) to simplify VAT compliance across EU countries. To comply with VAT rules, **you may need to collect multiple pieces of evidence** to determine the customer's location. For example

- IP Address:** Approximate location of the customer.
- Billing Address:** Provided during checkout.
- Card Issuing Country:** Based on the card's BIN.

If your, registered in UK online service-based business, exceeds the VAT threshold (e.g. £90,000 turnover), you would need to register for VAT in the relevant country (e.g. in France, if your primary online customers are from this country) & remit taxes accordingly.

Practical Steps

- Collect Evidence:**
Use **Stripe**'s (an online service that we will use to create checkout form) tools to gather IP address, billing address, and card issuing country.
- Automate VAT Calculation:**
Integrate a VAT calculation tool to automatically apply the correct tax rate based on the customer's location.
- Consult a Tax Professional:**
Ensure compliance with international VAT rules to avoid penalties.

Stripe's Customer



We will use **stripe**'s API, to create a **Customer**

The **Customer** is going to be our registered **User**

We will also use stripe's API - **Payment Intent** TO mount **Card Element UI** in our <form>

It is important to use stripe's UI elements, in order to, NOT store client's sensitive payment data, on our server. Stripe's UI elements will give us the TOKENIZED client's payment method, which we can use to make payments.

project_folder/users/utils.py

utils.py

```
from django.core.files.uploadedfile import UploadedFile
from django.conf import settings

import requests, stripe
stripe.api_key = settings.STRIPE_SECRET_KEY


def get_or_create_stripe_customer(user):
    stripe_customer_id = None

    if hasattr(user, 'profile'):
        # Retrieve the user's Stripe customer ID from its profile
        stripe_customer_id = getattr(user.profile, 'stripe_customer_id', None)

    if stripe_customer_id is None:
        # Create a new Stripe customer
        customer = stripe.Customer.create(
            name=user.username,
            email=user.email,
            metadata={'user_id': str(user.id)})
        stripe_customer_id = customer.id

    if hasattr(user, 'profile'):
        # Update the user's profile with the new Stripe customer ID
        user.profile.stripe_customer_id = stripe_customer_id
        user.profile.save()

    return stripe_customer_id
```

project_folder/users/models.py

🐍 models.py

```
from django.db import models
from django.contrib.auth.models import User
from django_resized import ResizedImageField
from django.conf import settings

from .utils import get_or_create_stripe_customer

from botocore.exceptions import ClientError
import boto3


class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    stripe_customer_id = models.CharField(
        max_length=255, blank=True, null=True, unique=True, editable=False)
    phone_number = models.CharField(max_length=15)
    image = ResizedImageField(
        size=[300, 300],
        crop=['middle', 'center'],
        quality=75,
        force_format='JPEG',
        upload_to='profile_pics',
        default='default_profile.jpg'
    )

    def save(self, *args, **kwargs):
        try:
            this = Profile.objects.get(user=self.user)

            # Create Stripe customer for new Profile. (used for payments)
            if not hasattr(this, 'stripe_customer_id'):
                self.stripe_customer_id = get_or_create_stripe_customer(self.user)

            # Only move the old image if there is an existing image and it's...
            if this.image and this.image.name != self.image.name:
                #recycle_profile_pic(this.image.name) ← AWS S3 bucket
                remove_profile_pic(this.image.name) ← AWS S3 bucket

        except Profile.DoesNotExist:
            # This is a new profile, so no need to move any image.
            pass

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

python manage.py makemigrations users → python manage.py migrate users

 project_folder/users/views.py

 views.py

```
...  
from django.conf import settings  
from .models import Profile  
from .forms import UserRegisterForm, UserUpdateForm, ProfileForm  
  
import stripe  
stripe.api_key = settings.STRIPE_SECRET_KEY  
  


---

  
@login_required  
def profileview(request, user_id):  
    user_instance = get_object_or_404(User, pk=user_id)  
    ...  
  
    if form_user.is_valid() and form_profile.is_valid():  
        confirm_password = form_user.cleaned_data.get('confirm_password')  
        if check_password(confirm_password, user_instance.password):  
            form_user.save()  
            form_profile.save()  
            stripe.Customer.modify(  
                user_instance.profile.stripe_customer_id,  
                name=user_instance.username,  
                email=user_instance.email  
            )  
            ...  
    ...
```

User's Stripe Customer's data: `name` and `email`, is also being updated

 project_folder/users/admin.py

 admin.py

```
from django.contrib import admin  
from .models import Profile  
  


---

  
class ProfileAdmin(admin.ModelAdmin):  
    readonly_fields = ('stripe_customer_id',)  
    list_display = [field.name for field in Profile._meta.fields]  
  


---

  
# Register your models here.  
admin.site.register(Profile, ProfileAdmin) ← Added ProfileAdmin
```

Buy Plan with Stripe's Card Element UI

Pay Now

£25.00

Plan*

Standard: £25

 4000 0027 6000 3184 03 / 59 342 ZIP

Powered by [Stripe](#). By proceeding, you agree to [Terms of Service](#) and [Privacy Policy](#).

[Proceed with payment](#)

project_folder/app_name/forms.py

🐍 forms.py

```
class PlanForm(forms.Form):
    UNIT = '£'
    PLANS = {
        'basic': 12.25, # Hence, £12.25
        'standard': 25,
        'premium': 50
    }
    OPTION = [
        ('', '--- Select a plan ---'),
        ('basic', f'Basic: {UNIT}{PLANS.get("basic")}' ),
        ('standard', f'Standard: {UNIT}{PLANS.get("standard")}' ),
        ('premium', f'Premium: {UNIT}{PLANS.get("premium")}' )
    ]
    plan = forms.ChoiceField(
        label='Plan',
        choices=OPTION,
        required=True
    )
```

project_folder/app_name/views.py

🐍 views.py

```
...
from django.http import HttpResponseRedirect, JsonResponse ← Added JsonResponse

from django.views.generic import ListView
from users.utils import get_or_create_stripe_customer
from .forms import ModelForm, ContactUsForm, PlanForm ← Added PlanForm
from .models import ModelName

import stripe, json
stripe.api_key = settings.STRIPE_SECRET_KEY



---


@login_required
def buyPlanView(request):

    context = {
        'form': PlanForm(),
        'plans': PlanForm.PLANS,
        'payment_unit': PlanForm.UNIT,
        'display_price': 0, # initial price to be displayed
        'stripe_public': settings.STRIPE_PUBLIC_KEY
    }

    return render(request, 'app_name/buy_plan.html', context)
```

```

@login_required
def buyPlanPaymentIntentView(request):

    form = PlanForm(request.POST)

    # Validate plan from POST request
    if request.method == 'POST' and form.is_valid():
        plan = request.POST.get('plan')
    else:
        return JsonResponse(
            {'error': 'Invalid form, try refreshing the page'}, status=400)

    if plan not in PlanForm.PLANS.keys():
        return JsonResponse(
            {'error': 'Selected plan is not registered, please report this issue'}, status=400)

    # Retrieve the user's Stripe customer ID from its profile
    stripe_customer_id = get_or_create_stripe_customer(request.user)
    # Stripe's amount is in pennies. Hence, amount=500 → £5.00
    amount = int(PlanForm.PLANS.get(plan) * 100)

    intent = stripe.PaymentIntent.create(
        amount=amount,
        currency="gbp",
        customer=stripe_customer_id,
        payment_method_types=["card"],
        description=f"Purchased '{plan}' plan",
        metadata={"purchased_plan": str(plan)}
    )

    return JsonResponse({'client_secret': intent.client_secret})

```

NOTE: `client_secret` is required for security reasons, and to proceed a charge

```

def paymentSuccessView(request):
    return render(request, 'app_name/payment_success.html')

```

project_folder/app_name/urls.py

urls.py

```
urlpatterns = [
    ...
    path('buy-plan/', views.buyPlanView, name='buy_plan'),
    path('buy-plan/intent/', views.buyPlanPaymentIntentView,
name='buy_plan_intent'),
    path('payment-success/', views.paymentSuccessView,
name='payment_success'),
]
```

project_folder/app_name/buy_plan.html

</> buy_plan.html

```
{% extends 'app_name/base.html' %}
{% load static %}
{% load crispy_forms_tags %}
{% block title %}Checkout Page{% endblock %}

{% block head %}
<style>

form * {
    font-family: "Helvetica Neue", Helvetica, sans-serif;
}

.stripe-agreement {
    text-align: center;
    font-size: 0.85rem;
    color: #333;
    padding: 15px 20px;
    background-color: #f9fafb;
    border-top: 1px solid #ddd;
}
.stripe-agreement a {
    color: #0070ba;
    text-decoration: none;
}
.stripe-agreement a:hover {
    text-decoration: underline;
}
.stripe-agreement p {
    margin: 0;
}

</style>
<script name="stripe.js" src="https://js.stripe.com/v3/"></script> ← Stripe API in JavaScript
{% endblock %}
```

```
{% block content %}
```

```
<h1>Pay Now</h1>
<hr>
<h2 id="total-cost" class="display-3 fw-bold text-primary mb-0">
{{ payment_unit }}{{ display_price|floatformat:2 }}
</h2>
```

```
<form id="payment-form"
      action="{% url 'buy_plan' %}"
      method="POST"
      data-secret="{{ client_secret }}">
  {% csrf_token %}
  {{ form|crispy }}
  <div id="card-element" class="form-control p-3">
    <!-- Elements will create form elements here -->
  </div>
```

```
<section class="stripe-agreement mt-3">
  <p>
    Powered by
    <a href="https://stripe.com" target="_blank">Stripe</a>.
    By proceeding, you agree to
    <a href="https://stripe.com/legal" target="_blank">Terms of Service</a>
    and
    <a href="https://stripe.com/privacy" target="_blank">Privacy Policy</a>.
  </p>
</section>
```

```
<button type="submit" id="submit-button" class="btn btn-primary fw-bold my-3 p-3">
  Proceed with payment
</button>
<div id="card-errors" role="alert" class="text-danger">
  <!-- Display error message to your customers here -->
</div>

</form>
```

```
{% endblock %}
```

```
DO NOT RENAME: payment-form, card-element and card-errors ← (stripe elements)
```

```

{%- block base %}



---


<script name="select-plan-form" type="text/javascript">

const objPlans = {{ plans|safe }};
let elmTotalCost = elmSelectPlan = undefined;

document.addEventListener('DOMContentLoaded', () => {

    elmTotalCost = document.getElementById('total-cost');
    elmSelectPlan = document.querySelector('select[name="plan"]');
    elmSelectPlan.classList.add('p-3');

    elmSelectPlan.addEventListener('change', (event) => {
        const totalCost = objPlans[event.target.value] || 0;
        elmTotalCost.innerHTML = `{{ payment_unit }}${totalCost.toFixed(2)}`;
    });
});

</script>

```

```
<script name="stripe-payment-form" type="text/javascript">
```

```
// Initialize Stripe with your publishable key
const stripe = Stripe('{{ stripe_public }}');

// Create an instance of Elements
const elements = stripe.elements();
```

```
// Options for customizing the Card Element styles
```

```
const style = {
  base: {
    color: "#32325d",
    fontFamily: "'Helvetica Neue', Helvetica, sans-serif",
    fontSmoothing: "antialiased",
    fontSize: "20px",
    "::placeholder": { color: "#aab7c4", },
  },
  invalid: {
    color: "#fa755a",
    iconColor: "#fa755a",
  },
};
```

```
// Create an instance of the Card Element & Add the Card Element to the page
```

```
const card = elements.create("card", { style });
card.mount("#card-element");
```

```

const form = document.getElementById("payment-form");
const errorElement = document.getElementById("card-errors");



---


// Handle real-time validation errors
card.on("change", (event) => {
  errorElement.textContent = (event.error) ? event.error.message : "";
});



---


// Handle form submission
form.addEventListener("submit", async (event) => {
  event.preventDefault();

  const formData = new FormData();
  formData.append("plan", elmSelectPlan.value);

  // Send the selected plan to the backend
  const response = await fetch("{% url 'buy_plan_intent' %}", {
    method: "POST",
    headers: {
      "X-CSRFToken": "{{ csrf_token }}",
    },
    body: formData,
  });

```

```

  const data = await response.json();

  if (!response.ok) {
    // Handle errors returned from the backend
    errorElement.textContent = data.error;
  }

```

```

  const clientSecret = data.client_secret;

```

NOTE: card.on("change") - adds realtime validation errors for `.card-errors`

NOTE: `response`, sends `request` with `plan` input's value TO `buyPlanIntentView`

NOTE: IF `plan` passes a validation at the backend, THEN we get `clientSecret`

NOTE: `clientSecret` is required TO `make/confirm` the payment

```

// Confirm the card payment
const { error } = await stripe.confirmCardPayment(clientSecret, {
  payment_method: {
    card: card,
  },
});

if (error) {
  // Handle errors from the stripe card payment
  errorElement.textContent = error.message;
} else {
  // The card has been paid successfully
  // Reset form inputs
  card.clear();
  form.reset();
  // Redirect to the success page
  window.location.href = "{% url 'payment_success' %}";
}

});


```

```

</script>
{% endblock %}

```

NOTE: `card` is a stripe element, mounted as: `#card-element <input>`

NOTE: `client` is redirected TO `payment_success.html` template on a successful payment transaction



project_folder/app_name/payment_success.html

</> payment_success.html

```
{% extends 'app_name/base.html' %}  
{% block head %}  
<style>  
body {  
    text-align: center;  
    padding: 40px 0;  
}  
h1 {  
    color: #88B04B;  
    font-family: "Nunito Sans", "Helvetica Neue", sans-serif;  
    font-weight: 900;  
    font-size: 40px;  
    margin-bottom: 10px;  
}  
p {  
    color: #404F5E;  
    font-family: "Nunito Sans", "Helvetica Neue", sans-serif;  
    font-size: 20px;  
    margin: 0;  
}  
i {  
    color: #9ABC66;  
    font-size: 100px;  
    line-height: 200px;  
    margin-left: -15px;  
}  
.card {  
    padding: 60px;  
    display: inline-block;  
    border: 0;  
    margin: 0 auto;  
    animation: pop-in 1s;  
}  
.btn-success {  
    animation: pop-in 1s, 1s scale-up 1s ease;  
}  
@keyframes pop-in {  
    0% { opacity: 0; transform: scale(0.1); }  
    100% { opacity: 1; transform: scale(1); }  
}  
@keyframes scale-up {  
    0% { transform: scale(1); }  
    50% { transform: scale(1.5); }  
    100% { transform: scale(1); }  
}  
</style>  
{% endblock %}
```

```
{% block content %}  
<div class="card">  
    <div style="  
        border-radius: 200px;  
        height: 200px;  
        width: 200px;  
        background: #F8FAF5;  
        margin: 0 auto;">  
        <i class="checkmark">✓</i>  
    </div>  
    <h1 class="mt-5">Success</h1>  
    <p>We have successfully received your purchase</p>  
    <a href="{% url 'index' %}">  
        <button class="btn btn-success my-3">  
            Continue to  
            <strong>Home Page</strong>  
        </button>  
    </a>  
</div>  
{% endblock %}
```

DonateView - Modifiable Payment Element UI

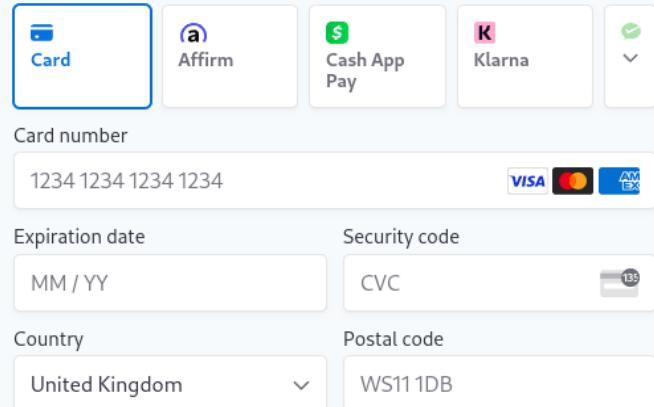
[Payment Element](#) is a **recommended** implementation of stripe.

Similar to **Card Element**, **Payment Element** must be created in backend to get essential key: **client_secret** ([used only in frontend](#))

intent has property **intent_id**, which can be used to modify the Payment Intent.
([NEVER send intent_id to the frontend](#))

Many payment method types, such as; **paypal**, **p24**, **klarna**... require a **Hook**, to verify transaction state of a third-party service.
([To ease the tutorial, DonateView will include default payment method only](#))

Payment Element Form is mounted from the **frontend** and requires **client_secret**, and the state of new **Payment Element** is added on stripe, even if client hasn't finished their form.



Backend View	Frontend Template
<pre>intent = stripe.PaymentIntent.create(# amount of 100 → \$1.00 amount=100, currency="pln", # ← Polish złoty #customer=stripe_customer_id, payment_method_types=["card", "paypal", "p24",], description="Donation", #metadata={ "key": value })</pre> <pre>context = { 'client_secret': intent.client_secret, 'stripe_public': settings.STRIPE_PUBLIC_KEY }</pre>	<pre>const stripe = Stripe('{{ stripe_public }}'), const options = { clientSecret: '{{ client_secret }}', appearance: { theme: 'tabs', /* stripe tabs night */ }, }; const elements = stripe.elements(options);</pre> <pre>const paymentElementOptions = { layout: { type: 'tabs', // accordion tabs auto defaultCollapsed: false, }, }; const paymentElement = elements.create('payment', paymentElementOptions); paymentElement.mount('#payment-element');</pre> <pre>let url = window.location.origin + "{% url 'payment_success' %}" const {error} = await stripe.confirmPayment({ elements, // If form is valid → redirect client confirmParams: { return_url: url, }, }); if(error) { /* confirmPayment failed, print error */ }</pre>

Create DonateView - Template + Payment Element

project_folder/app_name/views.py views.py

```
@login_required
def donateView(request):

    min_value = 5 # 5 -> 5,00zł

    context = {
        'locale': 'pl',
        'payment_unit': 'zł',
        'display_price': min_value, # 5 -> 5,00zł
        'stripe_public': settings.STRIPE_PUBLIC_KEY
    }

    # Retrieve the user's Stripe customer ID from its profile
    stripe_customer_id = get_or_create_stripe_customer(request.user)

    intent = stripe.PaymentIntent.create(
        amount=int(min_value * 100), # 5 -> 0.05zł
        currency="pln",
        customer=stripe_customer_id,
        payment_method_types=["card"],
        description=f"Donation",
    )

    context.update({
        'client_secret': intent.client_secret
    })

    return render(request, 'app_name/donate.html', context)
```

NOTE: `donateView` is made for Polish customers

project_folder/app_name/urls.py urls.py

```
urlpatterns = [
    ...
    path('donate/', views.donateView, name='donate'),
]
```

 project_folder/chat/templates/chat/donate.html

</> donate.html

```
{% extends 'app_name/base.html' %}  
{% load static %}  
{% load crispy_forms_tags %}  
{% block title %}Strona Dotacji{% endblock %}  
  
{% block head %}  
<script name="stripe.js" src="https://js.stripe.com/v3/"></script>  
{% endblock %}  
  
{% block content %}  
<h1>Wesprzyj Nas</h1>  
<hr>  
<h2 id="total-cost" class="display-3 fw-bold text-primary mb-0">  
    <span class="value">{{ display_price|floatformat:2 }}</span>{{ payment_unit }}  
</h2>  
  
<form id="payment-form" data-secret="{{ client_secret }}">  
    {% csrf_token %}  
    <div id="payment-element">  
        <!-- Elements will create form elements here -->  
    </div>  
  
<section class="stripe-agreement mt-3">  
    <p>  
        Obsługiwane przez  
        <a href="https://stripe.com" target="_blank">Stripe</a>.  
        Kontynuując, akceptujesz  
        <a href="https://stripe.com/legal" target="_blank">Warunki korzystania z usługi</a>  
        i  
        <a href="https://stripe.com/privacy" target="_blank">Politykę prywatności</a>.  
    </p>  
</section>  
  
<button type="submit" id="submit-button" class="btn btn-primary fw-bold my-3 p-3">  
    Kontynuuj płatność  
</button>  
<div id="error-message" role="alert" class="text-danger">  
    <!-- Display error message to your customers here -->  
</div>  
</form>  
{% endblock %}
```

```

{% block base %}

<script name="stripe-payment-form" type="text/javascript">

/* --- 3. Collect payment details --- */

// https://docs.stripe.com/payments/checkout/customization/appearance?payment-ui=embedded-components#all-rules

// Set your publishable key: remember to change this to your live publishable key in
// production
// See your keys here: https://dashboard.stripe.com/apikeys
const stripe = Stripe('{{ stripe_public }}');

const options = {
  clientSecret: '{{ client_secret }}',
  locale: '{{ locale }}',
  // Fully customizable with appearance API.
  appearance: {
    theme: 'tabs', // stripe | tabs | night (NOTE: variables modify the theme)
    variables: {
      colorPrimary: '#0570de',
      colorBackground: '#ffffff',
      colorText: '#30313d',
      colorDanger: '#df1b41',
      fontFamily: 'Ideal Sans, system-ui, sans-serif',
      spacingUnit: '6px',
      borderRadius: '4px',
    },
  },
};

// 
```

```

// Set up Stripe.js and Elements to use in checkout form,
// passing the client secret obtained in a previous step
const elements = stripe.elements(options);

// Create Payment Element
const paymentElementOptions = {
  layout: {
    type: 'tabs', // accordion | tabs | auto
    defaultCollapsed: false,
  }
};

const paymentElement = elements.create('payment', paymentElementOptions);

// Mount/Show the Payment Element
paymentElement.mount('#payment-element');

```

```

/* --- 4. Submit the payment to Stripe --- */

const form = document.getElementById('payment-form');
const errorElement = document.querySelector('#error-message');

// Handle form submission
form.addEventListener("submit", async (event) => {
    event.preventDefault();

    const {error} = await stripe.confirmPayment({
        // Elements` instance that was used to create the Payment Element
        elements,
        confirmParams: {
            return_url: window.location.origin + "{% url 'payment_success' %}",
        },
    });

    if (error) {
        // This point will only be reached if there is an immediate error when
        // confirming the payment. Show error to your customer (for example, payment
        // details incomplete)
        errorElement.textContent = error.message;
    }
    else {
        // Your customer will be redirected to your `return_url`. For some payment
        // methods like iDEAL, your customer will be redirected to an intermediate
        // site first to authorize the payment, then redirected to the `return_url`.
    }
});

</script>
{% endblock %}

```

 project_folder/app_name/templates/app_name/base.html </> base.html

```

{% load django_bootstrap5 %}
{% load static %}
<!DOCTYPE html>
<html lang="{% if locale %}{{ locale }}{% else %}en{% endif %}">
...

```

Including **Payment Element** is straightforward, but modifying it presents challenges regarding security. Let's integrate a **custom form** with a **donation field**, allowing clients to specify their own amount.

Synchronously mounted, custom donation field

```
project_folder/app_name/forms.py
```

```
forms.py
```

```
class DonateForm(forms.Form):
    locale = 'pl' # code e.g. 'pl', 'gb', 'de'
    UNIT = 'zł'

    min_value = 5 # 5 -> 500 groszy (pennies)
    max_value = 100

    donation = forms.DecimalField(
        label='Kwota donacji',
        initial=min_value,
        min_value=min_value,
        max_value=max_value,
        decimal_places=2,
        required=True,
        widget=forms.NumberInput(attrs={
            'placeholder': f'{min_value} do {max_value} zł',
            'aria-required': 'true',
            'title': '',
        })
    )
```

NOTE: The `amount` property of the `Payment Intent` object must be defined and cannot be `0`. We've set it to `500` (`5.00 zł`) at the backend in `donateView`.

To maintain consistency and prevent conflicts, let's initialize the `Payment Intent`'s `amount` using the `min_value` from `DonateForm` before serving it to the client.

We will do the same for the `UNIT` of the currency, and for the `locale` to translate `Payment Element`.

Additionally, restrict client from providing absurd amount of money to mitigate a probable mistake.

NOTE: `decimal_places` is primarily used for visual clarity and streamlined backend processing.

To ensure transparency, the client must be clearly informed of the exact amount they are about to pay before finalizing the transaction.

 project_folder/app_name/views.py

 views.py

```
...
from .forms import (
    ModelForm,
    ContactUsForm,
    PlanForm,
    DonateForm
)
```

```
...
@login_required
def donateView(request):
```

```
    form = DonateForm()
    min_value = form.fields['donation'].min_value

    context = {
        'form': form,
        'locale': DonateForm.locale,
        'payment_unit': DonateForm.UNIT,
        'display_price': min_value, # 5 -> 5,00 zł
        'stripe_public': settings.STRIPE_PUBLIC_KEY
    }
```

```
    # Retrieve the user's Stripe customer ID from its profile
    stripe_customer_id = get_or_create_stripe_customer(request.user)
```

```
    intent = stripe.PaymentIntent.create(
        amount=int(min_value * 100), # 5 -> 0,05 zł
        currency='pln',
        customer=stripe_customer_id,
        payment_method_types=['card'],
        description=f'Donation',
    )
```

```
    context.update({
        'client_secret': intent.client_secret
    })
```

```
    return render(request, 'app_name/donate.html', context)
```

NOTE: imported forms are wrapped inside a tuple to tidy up the codebase

 project_folder/app_name/templates/app_name/donate.html </> donate.html

```
...
<form id="payment-form" data-secret="{{ client_secret }}">
    {% csrf_token %}
    {{ form|as_crispy }}
    <div id="payment-element">
        <!-- Elements will create form elements here -->
    </div>
...

```

The **donation** field should only be displayed once the **Payment Element** is fully mounted. However, currently, it loads beforehand, which is unintuitive.

Suppose the stripe API **fails to load the Payment Element**. In that case, the client should know that the form is currently unavailable.

In this tutorial, I will only show how to mount both forms at the same time

```
/* --- Show custom <input> "Donation" --- */

// NOTE: This is NOT part of Stripe's documentation

// Mount Donation (Field), once paymentElement is ready
paymentElement.on('ready', () => {

    // Mount/Show the crispy_form divs
    const crispyFormDivs = document.querySelectorAll('[id^="div_id_"]');

    [...crispyFormDivs].map(div => {
        // Add class mounted
        div.classList.add('mounted');
    });
});

// Mount/Show the Payment Element
paymentElement.mount('#payment-element');
```

NOTE: elements with id: "div_id_..." are generated by crispy forms |filter

```
{% block head %}
<style>[id^="div_id_"]:not(.mounted) { display: none; }</style>
<script name="stripe.js" src="https://js.stripe.com/v3/"></script>
{% endblock %}
```

```

...
/* --- Show custom <input> "Donation" --- */

// NOTE: This is NOT part of Stripe's documentation

// Update #total-cost (label) when price is changed on input[name="donation"]
const labelTotal = document.getElementById('total-cost'),
  valueTotal = labelTotal.querySelector('.value'),
  parentDonation = document.querySelector('#div_id_donation'),
  inputDonation = parentDonation.querySelector('input[name="donation"]');

inputDonation.addEventListener('input', () => {
  let value = parseFloat(inputDonation.value);

  value = isNaN(value) ? 0 : value;
  valueTotal.innerText = value.toFixed(2).replace('.', ',');

  inputDonation.classList.contains('Input--invalid')
    ? labelTotal.classList.add('text-danger')
    : labelTotal.classList.remove('text-danger');
});

// Mount Donation (Field), once paymentElement is ready
...

```

Summary: added event listener for **donation** input TO update **#total-cost** (Label)

NOTE: **.Input--invalid** is stripe's CSS style & **.text-danger** is Bootstrap v5 CSS

NOTE: This code does not work yet because it relies on **.Input--invalid** class, which is toggled by **field_error.html** extension which we're about to implement

Let's update our donation field to support these stripe styles

```

{% block head %}
<link name="stripe.css" rel="stylesheet" href="{% static 'app_name/css/stripe.css' %}">
<style>

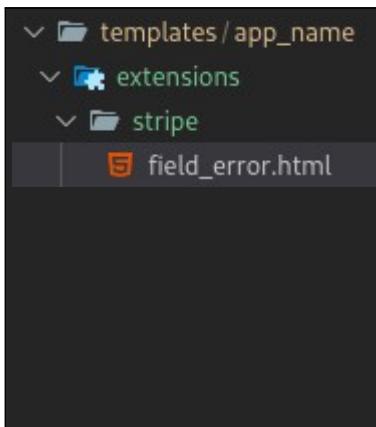
[id^="div_id_"]:not(.mounted) { display: none; }

.form-control {
  height: 56.39px !important;
}

</style>
<script name="stripe.js" src="https://js.stripe.com/v3/"></script>
{% endblock %}

```

Stripe CSS and field_error.html extension



Wesprzyj Nas

0,00 zł

Kwota donacji

5 do 100 zł

To pole jest wymagane.

This extension implements essential, real-time validation, feedback for the custom fields. However, it relies on stripe CSS and ID naming convention: **#Field-Type**, **.p-FieldError...**

We will implement only the essential styles. All of these styles were imported directly from generated Payment Element's <iframe>, some of them were also modified.

project_folder/app_name/static/app_name/css/stripe.css

stripe.css

```
:root {
    --p-spacing1: 6px;
    --p-spacing3: 18px;
    --borderRadius: 4px;
    --colorText: #30313d;
    --colorDanger: #df1b41;
    --colorBackground: #ffffff;
    --p-colorBackgroundDeemphasize10: #e6e6e6;
    --colorIconLoadingIndicator: #999999;
    --fontSizeSm: 0.93rem;
    --c-inputPaddingRight: 18px;
}
form * {
    font-family: "SF Pro Text", -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto", "Helvetica Neue", "Ubuntu", sans-serif;
    color: var(--colorText);
}
input.Input::placeholder {
    color: gray !important;
    opacity: 1;
}
.Input {
    padding: var(--p-spacing3);
    background-color: var(--colorBackground);
    border-radius: var(--borderRadius);
    transition: background 0.15s ease, border 0.15s ease, box-shadow 0.15s ease, color 0.15s ease;
    border: 1px solid var(--p-colorBackgroundDeemphasize10);
    box-shadow: 0px 1px 1px rgba(0, 0, 0, 0.03), 0px 3px 6px rgba(0, 0, 0, 0.02);
}
.Input--invalid {
    color: var(--colorDanger) !important;
    border-color: var(--colorDanger);
    box-shadow: 0px 1px 1px rgba(0, 0, 0, 0.03), 0px 3px 6px rgba(0, 0, 0, 0.02), 0 0 0 1px var(--colorDanger);
}
```

```

.p-Input-input {
  -webkit-animation: native-autofill-out 1ms;
  animation: native-autofill-out 1ms;
  display: block;
  width: 100%;
}

/* Field select */

.p-Select-select {
  padding-right: calc(var(--c-inputPaddingRight) + 1em) !important;
}

/* Label above Field input */

.Label, .form-label {
  /* Note .form-label is from bootstrap5 |crispy */
  margin-bottom: var(--p-spacing1);
  font-size: var(--fontSizeSm);
  transition: transform 0.5s cubic-bezier(0.19, 1, 0.22, 1), opacity 0.5s cubic-bezier(0.19, 1, 0.22, 1);
}

/* Field--Error (label under input) */

.Error {
  margin-top: var(--p-spacing1);
  color: var(--colorDanger);
  font-size: var(--fontSizeSm);
}

.AnimateSinglePresencelItem {
  transition: transform 0.35s ease-in-out;
}
.AnimateSinglePresencelItem > p {
  opacity: 1;
  transition: opacity 0.35s ease-in-out;
}
.AnimateSinglePresencelItem > p.Field--hide {
  opacity: 0;
  height: 0;
}
.AnimateSinglePresencelItem:has(p.Field--hide) {
  transform: scale(1.25);
}

/* Stripe terms and policies CSS */

.stripe-agreement {
  text-align: center;
  font-size: 0.85rem;
  color: #333;
  padding: 15px 20px;
  background-color: #f9fafb;
  border-top: 1px solid #ddd;
}
.stripe-agreement a {
  color: #0070ba;
  text-decoration: none;
}
.stripe-agreement a:hover {
  text-decoration: underline;
}
.stripe-agreement p {
  margin: 0;
}

```

As you may have noticed in the example picture, an **Error Label IS translated TO Polish**. For that, we need a custom function: **translateValidity()**

project_folder/app_name/static/app_name/js/translate_validity_pl.js JS

```
// AI Generated @ copilot.microsoft.com 26/Mar/2025 & edited by McRaZick

const validityTranslations = {
    valueMissing: "To pole jest wymagane.",
    typeMismatch: "Wprowadź poprawny typ danych (np. email lub URL).",
    patternMismatch: "Wartość nie pasuje do wzorca.",
    tooShort: "Wprowadź co najmniej {minLength} znaków.",
    tooLong: "Wartość przekracza maksymalną liczbę znaków ({maxLength}).",
    rangeUnderflow: "Wprowadź wartość większą lub równą {min}.",
    rangeOverflow: "Wprowadź wartość mniejszą lub równą {max}.",
    stepMismatch: "Wartość musi być zgodna z krokiem {step}.",
    customError: "To pole zawiera błąd niestandardowy.",
    valid: ""
};

const translateValidity = (validity, input) => {
    for (let [key, message] of Object.entries(validityTranslations)) {
        if (validity[key]) {
            // Replace placeholders dynamically if applicable
            message = message
                .replace("{minLength}", input.minLength || "")
                .replace("{maxLength}", input.maxLength || "")
                .replace("{min}", input.min || "")
                .replace("{max}", input.max || "")
                .replace("{step}", input.step || "");
            return message;
        }
    }
    return "";
};
```

NOTE: The application of a custom error is demonstrated later in the implementation of postal code field

project_folder/app_name/templates/app_name/donate.html </> donate.html

```
...
</style>
<script name="stripe.js" src="https://js.stripe.com/v3/"></script>
<script name="translate-validity-pl.js" src="{% static 'app_name/js/translate_validity_pl.js' %}"></script>
{% endblock %}
...
```



.../templates/app_name/extensions/stripe/field_error.html

</>

```
<div class="AnimateSinglePresence" aria-live="polite">
    <div class="AnimateSinglePresenceItem">
        <p id="Field-{{ id_type }}" class="p-FieldError Error" role="alert" aria-live="polite">
            {{ content }}
        </p>
    </div>
    <script type="text/javascript">
        // private scope wrapper is used to prevent polluting global scope
        (() => {
            const target = document.querySelector("{{ target }}"),
                  script = document.currentScript,
                  parent = script.parentElement,
                  field = parent.querySelector("#Field-{{ id_type }}"),
                  input = ("{{ id_type }}" === 'selectError')
                           ? target.querySelector('select')
                           : target.querySelector('input');

            input.addEventListener('input', () => {
                let validity = input.validity;

                if (!validity.valid) {
                    input.classList.add('Input--invalid');
                    field.classList.remove('Field--hide');
                    field.textContent = translateValidity(validity, input);
                    return;
                }

                // Valid
                input.classList.remove('Input--invalid');
                field.classList.add('Field--hide');
            });
        });

        // Append this .AnimateSinglePresence to Target
        target.appendChild(parent);
        // Remove this script
        script.remove();
    })();
</script>
</div>
```

NOTE: id_type, content, and target. We will import them in `{% include %}` tag

Also, we will NOT be using content in this tutorial. It is there as an option

 project_folder/app_name/templates/app_name/donate.html </> donate.html

```
...
</style>
<script name="stripe.js" src="https://js.stripe.com/v3/"></script>
<script name="translate-validity-pl.js" src="{% static 'app_name/js/translate_validity_pl.js' %}"></script>
{% endblock %}
...


---


...
<form id="payment-form" data-secret="{{ client_secret }}">
    {% csrf_token %}
    {{ form|as_crispy }}
    {{ form.donation|as_crispy_field }}
    <div id="payment-element">
        <!-- Elements will create form elements here -->
    </div>
    {% include 'app_name/extensions/stripe/field_error.html' with id_type="numberError"
    target="#div_id_donation" %}
    <section class="stripe-agreement mt-3">
...

```

NOTE: `|as_crispy` is NOT supported for specific fields. Hence, `|as_crispy_field`

NOTE: `id_type` and `target` are passed to the `field_error.html` extension.

NOTE: You can include many `field_error.html` extensions for many fields,
as long as, the field itself, is an `<input>` or `<select>` element.

NOTE: SET `id_type` TO "selectError" TO target `<select>` element.
Anything else targets the `<input>` element.

The extension will be included in this HTML template before it is served to client. Its `<script>` self-envokes: `((() => {...}))()`, to apply an event listener used for real-time, input validation feedback functionality for Polish clients.

In this case, the validation functionality is applied for the `donation` input, which is located in `target` parent: `#div_id_donation`.

Almost done...

Let's add initial stripe styles to the `donation` field from the **DonateForm**

```

class DonateForm(forms.Form):
    locale = 'pl' # code e.g. 'pl', 'gb', 'de'
    UNIT = 'zl'

    min_value = 5 # 5 -> 500 groszy (pennies)
    max_value = 100

    donation = forms.DecimalField(
        label='Kwota donacji',
        initial=min_value,
        min_value=min_value,
        max_value=max_value,
        decimal_places=2,
        required=True,
        widget=forms.NumberInput(attrs={
            'placeholder': f'{min_value} do {max_value} zł',
            'class': 'p-Input-input Input Input--empty p-',
            'DonationAmountInput-input',
            'inputmode': 'numeric',
            'aria-required': 'true',
            'title': '',
            'id': 'Field-donationAmountInput',
        })
    )

```

NOTE: Defined in widget attribute: “**id**”, is not used in this tutorial.
It’s only an example of stripe fields naming convention.

Now, real-time **feedback should be triggered** when the **donation** input is **invalid**

Take note that the input does not receive the **.text-danger** class when its value is empty unless the **required** attribute is provided

This is because we rely on the **.Input--invalid** class, which is added via **field_error.html**

However, validation from the **field_error.html** only applies when the input is genuinely invalid. By adding the **required** attribute, the input is considered invalid when its value is missing, ensuring proper validation

Stripe API handles POST validation for us on **confirmPayment()**. However, **donation** field originates from **DonateForm** (in **forms.py**). Thereby, it requires custom validation on POST request at the backend TO correctly handle the updation of client’s payment **intent**.

Proceed transaction, with modified by client, amount

```
NOTE: intent_id is required TO update the amount of client's payment intent  
NOTE: intent_id is generated by payment intent  
NOTE: intent_id can't be exposed to other clients
```

Hence, register new property for all users: **stripe_last_intent_id**

```
project_folder/users/models.py (users app) 🐍 models.py  
...  
  
class Profile(models.Model):  
    user = models.OneToOneField(User, on_delete=models.CASCADE)  
    stripe_customer_id = models.CharField(max_length=255, blank=True, null=True, unique=True,  
                                          editable=False)  
    stripe_last_intent_id = models.CharField(max_length=255, blank=True, null=True, unique=True,  
                                             editable=False)  
    ...
```

Display **stripe_last_intent_id** AS read-only in **admin** panel (OPTIONAL)

```
project_folder/users/admin.py (users app) 🐍 admin.py  
  
from django.contrib import admin  
from .models import Profile  
  
  
class ProfileAdmin(admin.ModelAdmin):  
    readonly_fields = (  
        'stripe_customer_id',  
        'stripe_last_intent_id'  
    )  
    list_display = [field.name for field in Profile._meta.fields]  
  
# Register your models here.  
admin.site.register(Profile, ProfileAdmin)
```

Finally, let's update the **amount** of client's payment **intent** via a POST request in the backend, and address additional errors on failed validation of **DonateForm**

```
project_folder/app_name/views.py                                (app_name app) 🐍 views.py

...
@login_required
def donateView(request):
    ...
    request.user.profile.stripe_last_intent_id = intent.id
    request.user.profile.save()

    context.update({
        'client_secret': intent.client_secret
    })

    return render(request, 'app_name/donate.html', context)
...
```

SET client's `stripe_last_intent_id` at `donateView` (TO target client's `intent`)

```
...
@login_required
def donateUpdatePaymentIntentView(request):
    ...
    if request.method != 'POST':
        return JsonResponse({'error': 'Invalid request, expected POST'}, status=400)

    form = DonateForm(request.POST)

    if not form.is_valid():
        return JsonResponse({'error': 'Invalid form', 'form_errors': form.errors}, status=400)

    ...
    donation = form.cleaned_data.get('donation')

    # Note: amount is in groszy (PLN)
    intent = stripe.PaymentIntent.modify(
        id=request.user.profile.stripe_last_intent_id,
        amount=int(donation * 100),
        metadata={'donation': str(donation)}
    )

    return JsonResponse({'success': 'Updated payment intent', 'donation': donation}, status=200)
...
```

Create `donateUpdatePaymentIntentView` (TO update amount of client's `intent`)

project_folder/app_name/urls.py

(app_name app) 🐍 urls.py

```
urlpatterns = [
    ...
    path('donate/', views.donateView, name='donate'),
    path('donate/update-intent/', views.donateUpdatePaymentIntentView,
name='donate_update_intent'),
]
```

Register new path route: **donateUpdatePaymentIntentView**

Utilize HTMX to send POST request and retrieve back a JSON response from the backend

project_folder/app_name/templates/app_name/donate.html </> donate.html

```
...
<button type="submit" id="submit-button" class="btn btn-primary fw-bold my-3 p-3"
hx-post="{% url 'donate_update_intent' %}"
hx-trigger="click delay:500ms"
hx-swap="none">
```

Kontynuuj płatność

```
</button>
<div id="error-message" role="alert" class="text-danger">
    <!-- Display error message to your customers here -->
</div>
...
```

```
...
/* --- 4. Submit the payment to Stripe --- */
```

```
const form = document.getElementById('payment-form');
const errorElement = document.querySelector('#error-message');
```

```
form.addEventListener("submit", async (event) => {
    event.preventDefault();
```

```
document.addEventListener('htmx:afterRequest', async (event) => {
```

```
    // Receive response from the backend
    let responseText = await event.detail.xhr.responseText,
        response = JSON.parse(responseText),
        formErrors = response.form_errors;
```

```
... continues on next page
```

```

// Display error labels for invalid <input>(s) of DonateForm
if (formErrors) {
  const inputNames = [...form.querySelectorAll('input')].map(input => input.name);

  inputNames.forEach(name => {
    if (!formErrors.hasOwnProperty(name)) return;

    let parent = document.querySelector(`#div_id_${name}`),
      field = parent.querySelector('.p-FieldError'),
      input = parent.querySelector(`input[name="${name}"]`);

    field.innerText = formErrors[name][0];
    field.classList.remove('Field--hide');
    input.classList.add('Input--invalid');

  });
}

```

```

if (response.error) {
  // Failure of Payment Intent's modification in the backend.
  // Hence, it is important to prevent further code executions.
  // Otherwise, client may proceed with the initial donation price,
  errorElement.textContent = response.error;
  throw new Error(response.error);
}

```

```

const {error} = await stripe.confirmPayment({
  // Elements` instance that was used to create the Payment Element
  elements,
  confirmParams: {
    return_url: window.location.origin + "{% url 'payment_success' %}",
  },
});

```

```

if (error) {
  // This point will only be reached if there is an immediate error when
  // confirming the payment. Show error to your customer (for example, payment
  // details incomplete)
  errorElement.textContent = error.message;
}

else {
  // Your customer will be redirected to your `return_url`. For some payment
  // methods like iDEAL, your customer will be redirected to an intermediate
  // site first to authorize the payment, then redirected to the `return_url`.
}

});

```

Stripe loader extension and exemption of asterisks (*)



Stripe's API requires time to parse essential elements, necessitating a loading indication. To optimize the user experience. This part of the tutorial demonstrates implementation of a loader for the donation field, ensuring seamless loading feedback during the parsing state.

NOTE: Labels for Payment Element fields do not include an asterisk (*).

NOTE: Django fields and Crispy Forms, by default, include an asterisk for required fields.

```
.../templates/app_name/extensions/stripe/extended_mounting.html </>

<div id="mounting" style="margin-bottom: 18px !important;">
    <div style="">
        <div style="height: 12.8204px !important; background-color: rgba(47, 48, 60, 0.04) !important; position: relative !important; overflow: hidden !important; will-change: transform !important; border-radius: 4px !important; width: 65.1407px !important; box-sizing: border-box !important; margin-bottom: 10.2735px !important; box-shadow: rgba(0, 0, 0, 0.03) 0px 1px 1px 0px, rgba(0, 0, 0, 0.02) 0px 3px 6px 0px !important;">
            <div class="loader" style="position: absolute !important; top: 0px !important; left: 0px !important; height: 100% !important; width: 50% !important; transform: translateX(200%); will-change: transform !important; background: linear-gradient(to right, rgba(47, 48, 60, 0), rgba(47, 48, 60, 0.05) 50%, rgba(47, 48, 60, 0)) !important; transition: transform 3000ms;"></div>
        </div>
        <div style="height: 56.3906px !important; padding: 20.1367px !important; box-sizing: border-box !important; background-color: rgb(255, 255, 255) !important; position: relative !important; overflow: hidden !important; border-radius: 4px !important; box-shadow: rgba(0, 0, 0, 0.03) 0px 1px 1px 0px, rgba(0, 0, 0, 0.02) 0px 3px 6px 0px !important; border-width: 1px !important; border-color: rgb(230, 230, 230) !important; border-style: solid !important;">
            <div style="height: 12.8204px !important; background-color: rgba(47, 48, 60, 0.04) !important; position: relative !important; overflow: hidden !important; will-change: transform !important; border-radius: 4px !important; width: 65.1407px !important; box-sizing: border-box !important;">
                <div class="loader" style="position: absolute !important; top: 0px !important; left: 0px !important; height: 100% !important; width: 50% !important; transform: translateX(200%); will-change: transform !important; background: linear-gradient(to right, rgba(47, 48, 60, 0), rgba(47, 48, 60, 0.05) 50%, rgba(47, 48, 60, 0)) !important; transition: transform 3000ms;"></div>
            </div>
        </div>
    </div>
</div>
```



```

... function refreshLoaderAnimation() {...} is above

// Select the target node
const targetNode = document.getElementById('payment-element');

// Create an instance of MutationObserver
const observer = new MutationObserver((mutationsList) => {
    mutationsList.forEach((mutation) => {
        const privateStripeLoader = targetNode.querySelector('.__PrivateStripeElementLoader');
        if (!privateStripeLoader) return;

        // Start the loader animation at #mounting
        refreshLoaderAnimation();
        setInterval(refreshLoaderAnimation, 3500);
        document.getElementById('mounting').classList.add('mounted');
        observer.disconnect();
    });
});

observer.observe(targetNode, {childList: true, subtree: true });
...

```

Observe: #payment-element FOR .__PrivateStripeElementLoader

This observer .mounts (displays) **Extension:** extended_mounting.html (loader)
when .__PrivateStripeElementLoader IS parsed in: #payment-element

```

...
// Mount Donation (Field), once paymentElement is ready
paymentElement.on('ready', () => {

    // Remove #mounting with .loader(s)
    document.getElementById('mounting').remove();
    clearInterval(refreshLoaderAnimation);

    // Mount>Show the crispy_form divs
    const crispyFormDivs = document.querySelectorAll('[id^="div_id_"]');

    [...crispyFormDivs].map(div => {
        // Remove '*' from label
        let label = div.querySelector('label');
        label.innerText = label.innerText.replace('*', '');
        // Add class mounted
        div.classList.add('mounted');
    });
});

```

Interval: **refreshLoaderAnimation** IS unrequired WHEN **paymentElement** IS parsed

Let's also REMOVE: the asterisks (*) FROM: the additional fields' <label>

```
class DonateForm(forms.Form):
    locale = 'pl' # code e.g. 'pl', 'gb', 'de'
    UNIT = 'zl'

    min_value = 5 # 5 -> 500 groszy (pennies)
    max_value = 100

    donation = forms.DecimalField(...)

    class Meta:
        # Disclude the '*' suffix from the fields
        # NOTE: This does not work for crispy forms
        # Because crispy filters alter the, altered by Meta class, fields
        label_suffix = ''
```

NOTE: This is an **OPTIONAL SOLUTION** for REMOVING **asterisk** (and other suffixes) FROM the fields' <label>.

NOTE: Crispy filters, such as (form|as_crispy & form.field|as_crispy_field) include asterisk (*) FOR fields containing **required** attribute.

Now, additional **loading feedback should be displayed** **only** during the parsing phase of the Payment Element

The initial appearance of the loader extension seems to be synchronous with the loaders generated by the Stripe API in the Payment Element Form. However, it isn't perfect on a slow internet connection and during intensive throttling.

Additionally, Stripe is an external service, which may be prone to change in the future, adding unnecessary maintenance requirements in terms of frontend design. Therefore, it's best not to include additional loaders entirely.

Dynamic Select Images with Flag Icons



The main functionality is successfully implemented. However, the default country field in Stripe API lacks visual distinction, as it does not display country flags. This makes it harder for users to quickly identify and select their desired country option.

Flag Icons: <https://github.com/lipis/flag-icons/>

Download Flags from: <https://flagicons.lipis.dev/>

Dynamic Select Images created by “David Adams” (JS Library)
<https://codeshack.io/dynamic-select-images-html-javascript/>

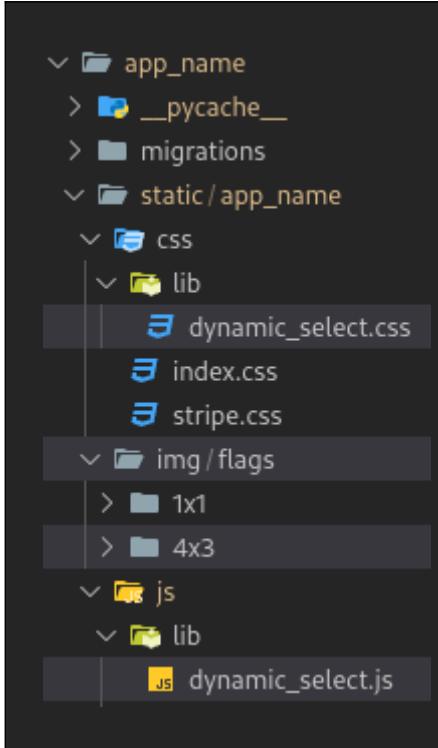
A screenshot of a dynamic select dropdown menu. The title "Country" is at the top. The dropdown contains four items: "United States" (selected), "United States", "United Kingdom", and "Japan". Each item is represented by a small flag icon followed by the country name.

This tutorial utilizes **Dynamic Select Images** with additional “Accessibility” functionality.

Dynamic Select Images created by “David Adams” and modified by “McRaZick”
<https://github.com/gubrus50/dynamic-select-images-js>

A screenshot of a dynamic select dropdown menu titled "Select country". It shows a list of countries with their flags and names. The country "Poland" is selected and highlighted with a yellow background. Other countries listed include Moldova, Romania, Hungary, Czech Republic, Slovakia, Lithuania, and Estonia. To the left of the dropdown, there is a form field labeled "Kraj" with several entries, each showing a flag and a country name: "Polska", "Pitcairn", "Polinezja Francuska", "Polska", and "Portugalia".

<https://jsfiddle.net/sat6h1r4/>



Install dynamic_select.css

<https://raw.githubusercontent.com/gubrus50/dynamic-select-images-js/refs/heads/main/DynamicSelect.css>

Install dynamic_select.js

<https://raw.githubusercontent.com/gubrus50/dynamic-select-images-js/refs/heads/main/DynamicSelect.js>

Install flag-icons-main.zip (4x3)

<https://flagicons.lipis.dev/>

Install django-countries

Countries and territories evolve over time, with some gaining international recognition while others remain disputed due to geopolitical factors. Stripe's API includes an object listing the countries where its services are available. However, making frequent requests for this data at scale is inefficient. Maintaining a custom list of countries presents its own challenges, requiring regular updates.

A more practical approach is to use the django-countries library, which provides an up-to-date list of recognized countries and territories. While some entries in this dataset may lack international recognition, we can filter them out if necessary. Additionally, Stripe's API can validate country data when processing a POST request on **confirmPayment()**, ensuring only supported locations are accepted.

```
(venv) @ project_folder pip library
```

```
pip install django-countries
pip freeze > requirements.txt # update requirements.txt
```

 project_folder/app_name/forms.py

 forms.py

```
from django_recaptcha.fields import ReCaptchaField
from django_countries import countries

from django import forms
from .models import ModelName


class DonateForm(forms.Form):
    locale = 'pl' # code e.g. 'pl', 'gb', 'de'
    UNIT = 'zł'

    min_value = 5 # 5 -> 500 groszy (pennies)
    max_value = 100

    donation = forms.DecimalField(...)

    # CHOICES = [('PL', 'Poland'), ('GB', 'Great Britain') ...]
    country = forms.ChoiceField(
        label='Kraj',
        choices=[(code, name) for code, name in countries],
        widget=forms.Select(attrs={
            'class': 'Input p-Select-select',
            'inputmode': 'text',
            'aria-required': 'true',
            'autocomplete': 'billing country',
            'title': '',
            'id': 'Field-countryInput',
        }),
        required=True
    )

    class Meta:
        # Disclude the '*' suffix from the fields
        # NOTE: This does not work for crispy forms
        # Because crispy filters alter the, altered by Meta class, fields
        label_suffix = ''
```

NOTE: choices=[] list is populated with django_countries.
However, it should contain country names in Polish.

 project_folder/app_name/templates/app_name/donate.html </> donate.html

...

```
.dynamic-select img {  
    border-radius: 5px;  
    border: 1px solid #d4d7da;  
}  
  
</style>  
<script name="stripe.js" src="https://js.stripe.com/v3/"></script>  
<script name="DynamicSelect.js" src="{% static 'app_name/js/lib/dynamic_select.js'%}"></script>  
<script name="translate-validity-pl.js" src="{% static 'app_name/js/translate_validity_pl.js'%}"></script>  
{% endblock %}  
...
```

Insert "dynamic_select.js" FROM {% static %} libraries directory

```
...  
<form id="payment-form" data-secret="{{ client_secret }}">  
    {% csrf_token %}  
    {% include 'app_name/extensions/stripe/extended_mounting.html' %}
```

```
    {{ form.donation|as_crispy_field }}  
    <div id="payment-element">  
        <!-- Elements will create form elements here --&gt;<br/>    </div>  
    {{ form.country|as_crispy_field }}
```

```
    {% include 'app_name/extensions/stripe/field_error.html' with  
id_type="numberError" target="#div_id_donation" %}  
    {% include 'app_name/extensions/stripe/field_error.html' with  
id_type="selectError" target="#div_id_country" %}  
    <section class="stripe-agreement mt-3">  
    ...
```

Include country field AND provide real-time validation feedback

```

// Mount Donation & Billing Address (Fields), once paymentElement is ready
paymentElement.on('ready', () => {

    // --- Include flag-icons TO <select> country (using DynamicSelect)

    const __flags = "{% static 'app_name/img/flags/4x3' %}",
        country = document.querySelector('#div_id_country > select'),
        options = country.querySelectorAll('option');

    [...options].map(option => { // NOTE: 'xx' is empty/white flag
        let countryCode = (option.value || 'xx').toLowerCase();
        option.dataset.img = __flags + `/${countryCode}.svg`;
    });
}

// https://github.com/gubrus50/dynamic-select-images-js
new DynamicSelect('#div_id_country > select', {
    class: 'form-control px-0',
    selectedStyle: 'border: 0',
    onChange: function updateAndValidatePostalCode(value, text, option) {

        // Hide invalid field of <dynamic-select> and danger highlight
        let input = document.querySelector('#div_id_country input');
        if (input && input.value.length) {

            let parent = document.querySelector('#div_id_country'),
                select = parent.querySelector('dynamic-select'),
                field = parent.querySelector('.p-FieldError');

            select.classList.remove('Input--invalid');
            field.classList.add('Field--hide');
        }
    }
});

// Remove #mounting with .loader(s)
... previously implemented code
});

```

NOTE: new DynamicSelect creates new instance of <dynamic-select> element
 Learn about DS: <https://github.com/gubrus50/dynamic-select-images-js>

Now you should be able to see a country field with flags.

Let's **include translated countries** TO the **country** field.

Translate country names with Polish locale messages

```
(venv) @ project_folder (Linux)

└── project_folder
    ├── .venv
    ├── app_name
    └── chat
        └── locale/pl/LC_MESSAGES
            ├── django.mo
            └── django.po

← Create tree for locale folder
mkdir -p locale/pl/LC_MESSAGES

← Create messages django.po template for locale PL
django-admin makemessages -l pl

NOTE: django.po file IS USED TO generate django.mo
      django.mo is USED FOR translation purposes.
```

DOCS - Internalization & localization:
<https://docs.djangoproject.com/en/5.1/topics/i18n/>

```
project_folder/my_website/settings.py          settings.py

LOCALE_PATH = [
    os.path.join(BASE_DIR, 'locale')
]

LANGUAGES = [
    ('en', 'English'),
    ('pl', 'Polish'),
]

LANGUAGE_CODE = 'en-us'

USE_I18N = True

NOTE: LANGUAGE_CODE defines the global language setting.
      However, only donateView is specifically designed for Polish clients.
      Therefore, LANGUAGE_CODE remains set to English.
```

Edit file **django.po** to translate countries (**remove # ,fuzzy** at .po TO indicate finalized file).

OR - Download template: [django5-tutorial - PL locale - django.po countries and territories](#)

NOTE: providing both **msgid** with same value results in compilers failure.

project_folder/locale/pl/LC_MESSAGES/django.po	abc django.po
INVALID	VALID
msgid "Poland" msgstr "Polska"	msgid "Netherlands" msgstr "Holandia"
msgid "Poland" msgstr "Polonia"	msgid "Holland" msgstr "Holandia"

(venv) @ project_folder (Linux)

```
# (All) Recommended when manage.py is initialized.  
python manage.py compilemessages  
  
# (All) Recommended when manage.py is NOT initialized.  
django-admin compilemessages  
  
# (Linux) This option provides a detailed output in terminal, in case something goes wrong.  
msgfmt django.po -o django.mo  
  
GENERATE django.mo file used for translating
```

project_folder/.gitignore .gitignore

```
# Optional ignore rule for any file with domain .mo at locale/ directory.  
locale/*.mo  
  
EXCLUDE django.mo files when deploying a project to platforms like GitHub.  
These files, similar to compiled SASS/SCSS (output → CSS), do not need to  
be stored in the repository since they can be generated as needed during  
development or when the project is served live in a production environment.
```

 project_folder/app_name/forms.py

 forms.py

```
from django_recaptcha.fields import ReCaptchaField
from django.utils.translation import activate, gettext_lazy as _
from django_countries import countries

from django import forms
from .models import ModelName


class DonateForm(forms.Form):
    ...
    # CHOICES = [('PL', 'Poland'), ('GB', 'Great Britain') ...]
    country = forms.ChoiceField(
        label='Kraj',
        choices=[],
    ...
    class Meta:
        # Disclude the '*' suffix from the fields
        # NOTE: This does not work for crispy forms
        # Because crispy filters alter the, altered by Meta class, fields
        label_suffix = ''

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        # Generate country list in the correct language
        activate(self.locale)
        # Manually translate country names using Django's gettext
        self.fields['country'].choices = [
            (code, _(name)) # Wrap country name in gettext_lazy
            for code, name in countries
        ]

        # Select initial county (currently selected)
        self.fields['country'].initial = self.locale.upper()
```

NOTE: activate() initializes the locale, so that gettext_lazy:
_() can translate the country names.

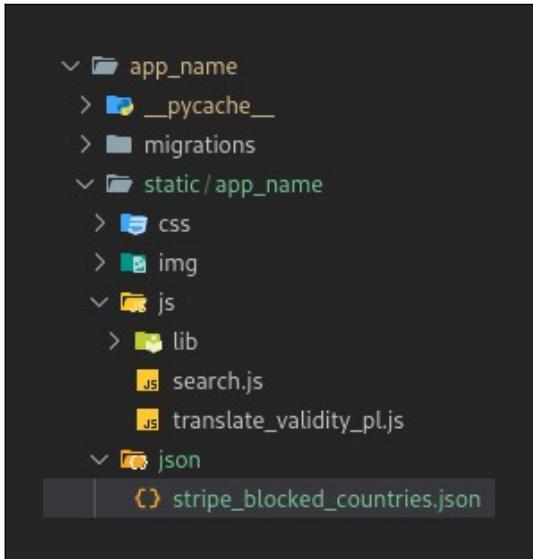
Stripe has **blocked** some countries. Let's disclude them from the custom country field.

source: <https://foundeck.com/blog/stripe-for-unsupported-countries/>

main source: <https://support.stripe.com/questions/sanctions-on-russia-and-belarus>

Both of the sources have been accessed: 19/April/2025, and might be inaplicable in future

Disclose blocked countries

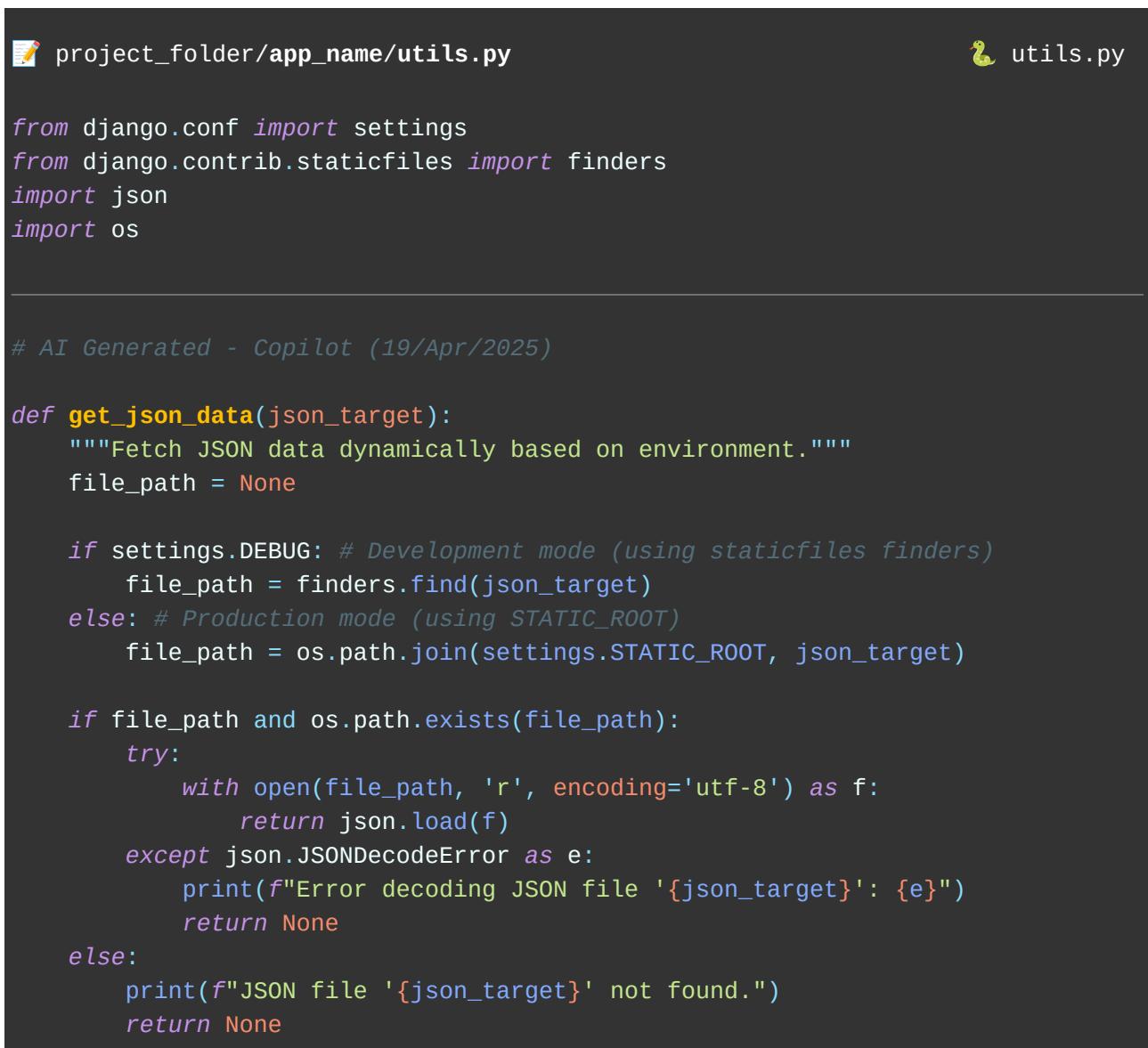


```
app_name
├── __pycache__
├── migrations
└── static
    └── app_name
        ├── css
        ├── img
        ├── js
        │   ├── lib
        │   ├── search.js
        │   └── translate_validity_pl.js
        └── json
            └── stripe_blocked_countries.json
```

Install stripe_blocked_countries.json

https://github.com/gubrus50/django5-tutorial/blob/main/project_folder/app_name/static/app_name/json/stripe_blocked_countries.json

Create utility



```
project_folder/app_name/utils.py
```

```
utils.py
```

```
from django.conf import settings
from django.contrib.staticfiles import finders
import json
import os

# AI Generated - Copilot (19/Apr/2025)

def get_json_data(json_target):
    """Fetch JSON data dynamically based on environment."""
    file_path = None

    if settings.DEBUG: # Development mode (using staticfiles finders)
        file_path = finders.find(json_target)
    else: # Production mode (using STATIC_ROOT)
        file_path = os.path.join(settings.STATIC_ROOT, json_target)

    if file_path and os.path.exists(file_path):
        try:
            with open(file_path, 'r', encoding='utf-8') as f:
                return json.load(f)
        except json.JSONDecodeError as e:
            print(f"Error decoding JSON file '{json_target}': {e}")
            return None
    else:
        print(f"JSON file '{json_target}' not found.")
        return None
```

project_folder/app_name/forms.py

🐍 forms.py

```
from django_recaptcha.fields import ReCaptchaField
from django.utils.translation import activate, gettext_lazy as _
from django_countries import countries

from django import forms
from .models import ModelName
from .utils import get_json_data

# Get SET of abbrevs OF stripe blocked countries JSON - {"AB", "CD", "EF" ...}
BLOCKED_COUNTRIES = {entry["abbrev"] for entry in
get_json_data("app_name/json/stripe_blocked_countries.json")}
```

```
class DonateForm(forms.Form):
...
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

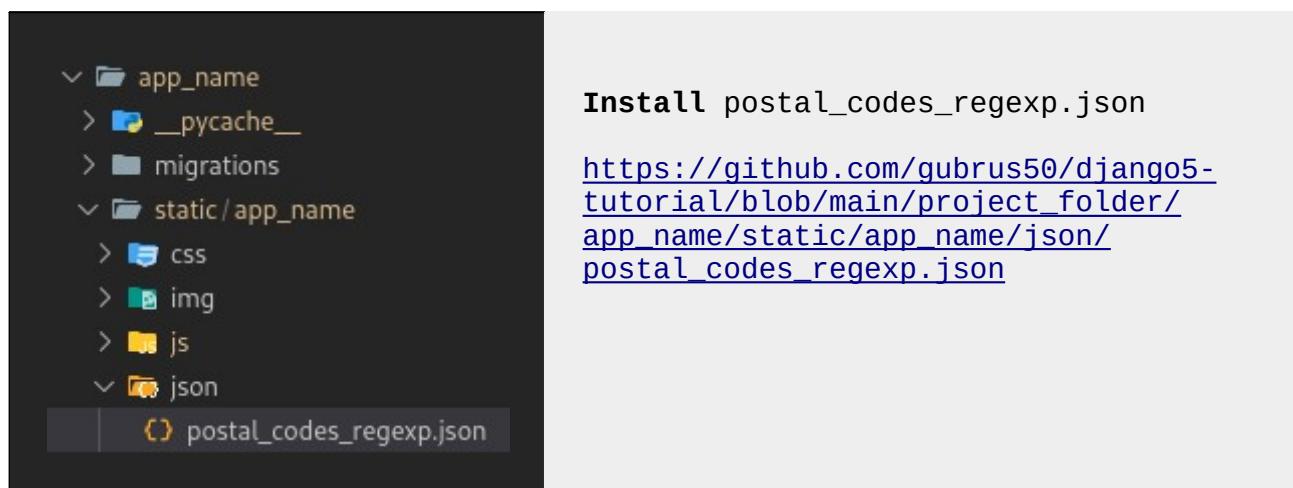
        # Generate country list in the correct language
        activate(self.locale)
        # Manually translate country names using Django's gettext
        self.fields['country'].choices = [
            (code, _(name)) # Wrap country name in gettext_lazy
            for code, name in countries if code not in BLOCKED_COUNTRIES
        ]

        # Select initial county (currently selected)
        self.fields['country'].initial = self.locale.upper()
```

Some **countries require postal code** to complete payments,

Let's implement custom postal code field.

Custom Postalcode Field



Install postal_codes_regex.json

https://github.com/gubrus50/django5-tutorial/blob/main/project_folder/app_name/static/app_name/json/postal_codes_regex.json

project_folder/app_name/forms.py

forms.py

```
class DonateForm(forms.Form):
    ...

    # CHOICES = [('PL', 'Poland'), ('GB', 'Great Britain') ...]
    country = forms.ChoiceField(...)

    postal_code = forms.CharField(
        label='Kod pocztowy',
        max_length=15,
        widget=forms.TextInput(attrs={
            'placeholder': '12-345',
            'class': 'p-Input-input Input-input--empty p-PostalCodeInput-
input',
            'inputmode': 'text',
            'aria-required': 'true',
            'data-country-parent': '#div_id_country',
            'autocomplete': 'billing postal-code',
            'title': '',
            'id': 'Field-postalCodeInput',
        }),
        required=True
    )

    ...

```

NOTE: The dataset **data-country-parent** is required to update the postcode validation expression based on the selected country at **#div_id_country**

Desktop - space between	Mobile - stack
<p>Kraj</p> <div style="display: flex; align-items: center;">  Polska </div> <p>Kod pocztowy</p> <input type="text" value="00-001"/>	<p>Kraj</p> <div style="display: flex; align-items: center;">  Polska </div> <p>Kod pocztowy</p> <input type="text" value="00-001"/>

 project_folder/app_name/templates/app_name/donate.html </> donate.html

...

```
.side-by-side > div {
  flex-grow: 1;
  flex-shrink: 1;
  flex-basis: auto;
  width: 100%;
  transition: all 1.25s cubic-bezier(0.68, -0.55, 0.27, 1.55), margin 0.1s ease-out;
}

.smooth-collapse {
  width: 0px !important;
  opacity: 0;
}
```

```
.smooth-collapse[data-width="0"] {
  margin: 0px !important;
  position: absolute;
}
```

```
/* Desktop - space between */
@media (min-width: 767px) {
  .side-by-side {
    display: flex;
  }
  .side-by-side > div:last-child {
    margin-left: 20px; /* This replaces the spacing div */
  }
}
```

```
/* Mobile - stack */
@media (max-width: 766px) {
  .side-by-side > div:last-child {
    margin-top: 10px; /* Space between stacked containers */
    margin-left: 0;
  }
}
```

```
.dynamic-select img {
  ...
}
```

```

...
</style>
<script name="stripe.js" src="https://js.stripe.com/v3/"></script>
<script name="DynamicSelect.js" src="{% static 'app_name/js/lib/dynamic_select.js' %}"></script>
<script name="translate-validity-pl.js" src="{% static 'app_name/js/translate_validity_pl.js' %}"></script>
<script name="postal-code-validation" type="text/javascript">

let POSTAL_CODES_REGEX;

fetch("{% static 'app_name/json/postal_codes_regex.json' %}")
  .then(response => {
    if (!response.ok) throw new Error(`HTTP error! Status: ${response.status}`);
    return response.json();
})
  .then(data => {
    POSTAL_CODES_REGEX = data;
})
  .catch(error => console.error("Error:", error));

const isCountriesPostalCodeValid = (countryCode, postalCode) => {

  let isValid = false;

  POSTAL_CODES_REGEX.forEach(obj => {
    if (countryCode.toLowerCase() === obj.abbrev.toLowerCase()) {
      // Return if postal-code doesn't match regex
      if (!postalCode.match(new RegExp(obj.postal || ""))) return;
      // Is valid if postal-code matches the regexp-example-length
      if (obj.example) {
        if (obj.example.length == postalCode.length) isValid = true;
        else return;
      }
      // Is valid if postal-code is not empty
      isValid = (postalCode.length > 0);
    }
  });
}

return isValid;
}

</script>
{% endblock %}

```

NOTE: Fetch REQUESTS and PARSES
→ postal_codes_regex.json AS POSTAL_CODES_REGEX IN global scope

It will be also used TO enhance the **country field** <dynamic-select>

isCountriesPostalCodeValid is used TO improve extension: **field_error.html**

Enhance field_error.html extension + translate validity

 project_folder/app_name/static/app_name/js/translate_validity_pl.js JS

// AI Generated @ copilot.microsoft.com 26/Mar/2025 & edited by McRaZick

```
const validityTranslations = {
    valueMissing: "To pole jest wymagane.",
    typeMismatch: "Wprowadź poprawny typ danych (np. email lub URL).",
    patternMismatch: "Wartość nie pasuje do wzorca.",
    tooShort: "Wprowadź co najmniej {minLength} znaków.",
    tooLong: "Wartość przekracza maksymalną liczbę znaków ({maxLength}).",
    rangeUnderflow: "Wprowadź wartość większą lub równą {min}.",
    rangeOverflow: "Wprowadź wartość mniejszą lub równą {max}.",
    stepMismatch: "Wartość musi być zgodna z krokiem {step}.",
    customError: "To pole zawiera błąd niestandardowy.",
    invalidPostalCode: "Wprowadź poprawny kod pocztowy.",
    valid: ""
};

const translateValidity = (validity, input) => {
    for (let [key, message] of Object.entries(validityTranslations)) {
        if (validity[key]) {
            // Replace placeholders dynamically if applicable
            message = message
                .replace("{minLength}", input.minLength || "")
                .replace("{maxLength}", input.maxLength || "")
                .replace("{min}", input.min || "")
                .replace("{max}", input.max || "")
                .replace("{step}", input.step || "");
            return message;
        }
    }
    return "";
};
```

Let's update **field_error.html**



.../templates/app_name/extensions/stripe/field_error.html

</>

```
<div class="AnimateSinglePresence" aria-live="polite">
  <div class="AnimateSinglePresenceItem">
    <p id="Field-{{ id_type }}" class="p-FieldError Error" role="alert" aria-live="polite">
      {{ content }}
    </p>
  </div>
  <script type="text/javascript">
    // private scope wrapper is used to prevent polluting global scope
    (() => {
      const target = document.querySelector("{{ target }}"),
            script = document.currentScript,
            parent = script.parentElement,
            field = parent.querySelector("#Field-{{ id_type }}"),
            input = ("{{ id_type }}" === 'selectError')
              ? target.querySelector('select')
              : target.querySelector('input');

      input.addEventListener('input', () => {
        let validity = input.validity;

        if (!validity.valid) {
          input.classList.add('Input--invalid');
          field.classList.remove('Field--hide');
          field.textContent = translateValidity(validity, input);
          return;
        }
        else if (input.id === 'Field-postalCodeInput') {
          let countryParent = document.querySelector(input.dataset.countryParent),
              countryInput = countryParent.querySelector('input'),
              countryCode = countryInput.value,
              postalCode = input.value;

          // Invalid PostCode
          if (!isCountriesPostalCodeValid(countryCode, postalCode)) {
            input.classList.add('Input--invalid');
            field.classList.remove('Field--hide');
            field.textContent = translateValidity({
              'invalidPostalCode': true
            }, input);
            return;
          }
        }
        // Valid
        input.classList.remove('Input--invalid');
        field.classList.add('Field--hide');
      });
    });
  ...
}
```

Organize billing address fields country & postal_code

```
project_folder/app_name/templates/app_name/donate.html      </> donate.html

...
<form id="payment-form" data-secret="{{ client_secret }}">
    {% csrf_token %}
    {% include 'app_name/extensions/stripe/extended_mounting.html' %}

    {{ form.donation|as_crispy_field }}
    <div id="payment-element">
        <!-- Elements will create form elements here --&gt;
    &lt;/div&gt;
    {{ form.country|as_crispy_field }}
    &lt;div id="billing-address" class="p-0 m-0"&gt;
        &lt;div class="side-by-side"&gt;
            {{ form.country|as_crispy_field }}
            {{ form.postal_code|as_crispy_field }}
        &lt;/div&gt;
        &lt;script type="text/javascript"&gt;
            // Remove &lt;br&gt; nodes from billing-address fields' &lt;label&gt;.
            // And marginate fields vertically like those in Payment Element.
            (() =&gt; {
                let script = document.currentScript,
                    fields = script.parentElement.querySelectorAll('[id^="div_id_"]'),
                    label = null;

                [...fields].map(field =&gt; {
                    field.classList.replace('mb-3', 'my-3');
                    label = field.querySelector('label.form-label');
                    label.innerHTML = label.innerText.trim();
                });

                script.remove();
            })();
        &lt;/script&gt;
    &lt;/div&gt;
    {% include 'app_name/extensions/stripe/field_error.html' with
        id_type="numberError" target="#div_id_donation" %}
    {% include 'app_name/extensions/stripe/field_error.html' with
        id_type="selectError" target="#div_id_country" %}
    {% include 'app_name/extensions/stripe/field_error.html' with
        id_type="postalCodeError" target="#div_id_postal_code" %}
    &lt;section class="stripe-agreement mt-3"&gt;
    ...
</pre>
```

```

// Mount Donation & Billing Address (Fields), once paymentElement is ready
paymentElement.on('ready', () => {
...

```

```

let parentPC = document.querySelector('#div_id_postal_code'),
    inputPC = document.querySelector('#div_id_postal_code > input'),
    fieldPC = document.querySelector('#Field-postalCodeError'),
    limitPC = inputPC.getAttribute('maxlength');

// Set initial maxlength for postal-code input, based on selected country
POSTAL_CODES_REGEX.forEach(obj => {
    if (obj.abbrev.toLowerCase() !== country.value.toLowerCase()) return;
    else if (limitPC)
        inputPC.setAttribute('maxlength', obj.example.length ?? limitPC);
});

```

```

// https://github.com/gubrus50/dynamic-select-images-js
new DynamicSelect('#div_id_country > select', {
    class: 'form-control px-0',
    selectedStyle: 'border: 0',
    onChange: function updateAndValidatePostalCode(value, text, option) {

```

```

        POSTAL_CODES_REGEX.forEach(obj => {
            // Return if country doesn't match
            if (obj.abbrev.toLowerCase() !== value.toLowerCase()) return;
            // Hide & disable #div_id_postal_code IF obj has no postal-code
            if (!obj.postal) {
                inputPC.setAttribute('disabled', '');
                inputPC.setAttribute('aria-required', false);
                parentPC.classList.add('smooth-collapse');
                return;
            }
            // Otherwise, show & enable #div_id_postal_code
            else if (inputPC.hasAttribute('disabled')) {
                inputPC.removeAttribute('disabled');
                inputPC.setAttribute('aria-required', true);
                parentPC.classList.remove('smooth-collapse');
            }
            // Update placeholder and maxlength for postal-code <input>
            inputPC.setAttribute('placeholder', obj.example ?? '');
            if (limitPC) inputPC.setAttribute(
                'maxlength', obj.example.length ?? limitPC
            );

```

... continues on the next page

```

    // Clear postal-code <input>, and hide invalid field
    inputPC.value = '';
    inputPC.classList.remove('Input--invalid');
    fieldPC.classList.add('Field--hide');
  });

  // Hide invalid field of <dynamic-select> and danger highlight
  ... previously implemented code
}

});

// Remove #mounting with .loader(s)
... previously implemented code
});

// Update data-width of #div_id_postal_code (used for transition animation)
setInterval(() => { parentPC.dataset.width = parentPC.clientWidth }, 10);

```

NOTE: Postal code field inherits transition animation FROM **.side-by-side**

NOTE: dataset **width** is used to support **.smooth-collapse** stylesheet class

NOTE: Postal code field should collapse (hide) when property: "postal",
is NOT provided IN **postal_codes_regexp.json** → **POSTAL_CODES_REGEX**

NOTE: As postal code field is collapsing, the country field should expand
gradually and cover the available width of **#billing-address** wrapper

NOTE: Error field OF postal code field IS now corresponding
TO the real-time change OF the country field's value

Let's replace Payment Element's **billing address** with the custom one

Initialize custom billing address as main address line

```
project_folder/app_name/templates/app_name/donate.html      </> donate.html

// Set up Stripe.js and Elements to use in checkout form,
// passing the client secret obtained in a previous step
const elements = stripe.elements(options);

// Create Payment Element
const paymentElementOptions = {
    layout: {
        type: 'tabs', // accordion | tabs | auto
        defaultCollapsed: false,
    },
    fields: {
        billingDetails: {
            address: 'never',
        },
    }
};

const paymentElement = elements.create('payment', paymentElementOptions);

// Mount/Show the Payment Element
paymentElement.mount('#payment-element');
```

NOTE: `billing_details` must be manually provided AT `confirmPayment` OF payment element with `billing_details` option SET to "never"

NOTE: `billing_details` AT `payment_method_data` requires properties: line1, line2, city, state, postal_code, and country

```
document.addEventListener('htmx:afterRequest', async (event) => {
...
    const {error} = await stripe.confirmPayment({
        //`Elements` instance that was used to create the Payment Element
        elements,
        confirmParams: {
            return_url: window.location.origin + "{% url 'payment_success' %}",
            /* IF You've disabled billing_details,
               THEN You must specify payment_method_data manually */
            payment_method_data: {
                billing_details: {
                    address: {
```

```

... confirmParams {

payment_method_data: {
  billing_details: {
    address: {
      // Get, required for Stripe API, billing data
      line1: document.querySelector('#div_id_line_1 > input:not(:disabled)').value || null,
      line2: document.querySelector('#div_id_line_2 > input:not(:disabled)').value || null,
      city : document.querySelector('#div_id_city > input:not(:disabled)').value || null,
      state: document.querySelector('#div_id_state > input:not(:disabled)').value || null,
      postal_code: document.querySelector('#div_id_postal_code > input:not(:disabled)').value || null,
      // Get the appropriate country code (e.g., 'GB' for the United Kingdom)
      country: document.querySelector('#div_id_country input[name="country"]:not(:disabled)').value
        || document.querySelector('#div_id_country > select:not(:disabled)').value
        || null,
    },
  },
},
},
},
...

```

NOTE: Country value of the target: "#div_id_country > select" is used instead WHEN the select element fails to convert into a <dynamic-select> element

Since this is a donation form, collecting address details such as line1, line2, city, and state is unnecessary unless required for specific purposes - such as, shipping a product purchased through the payment element.

You should be able to implement additional fields when needed by defining them first in the **DonateForm** and by attaching additional functionality using the custom-defined extensions

Wesprzyj Nas

5,00zł

Kwota donacji

Bezpieczna finalizacja jednym kliknięciem z Link ▾

Numer karty

Data ważności

Kod bezpieczeństwa

Kraj

Polska

Kod pocztowy

Bezpieczna finalizacja jednym kliknięciem z Link ▾

Numer karty

Data ważności

Kod bezpieczeństwa

Kraj

Polska

Kod pocztowy

Adres - linia 1

Ulica

Adres - linia 2

Numer mieszkania

Miasto

Województwo

Obsługiwane przez Stripe. Kontynuując, akceptujesz [Warunki korzystania z usługi i Politykę prywatności](#).

Obsługiwane przez Stripe. Kontynuując, akceptujesz [Warunki korzystania z usługi i Politykę prywatności](#).

Kontynuuj płatność

Kontynuuj płatność

Error – 400 , 403 , 404 and 500



LearnDjango: <https://learndjango.com/tutorials/customizing-django-404-and-500-error-pages> 22/May/2025

Common HTTP status codes:

Status	Name
400	Bad Request
403	Forbidden
404	Page Not Found
500	Internal Server Error
505	HTTP Version Not Supported

DEBUG=True @ **settings.py** will display a full log of particular issue to help developers.

This option should be disabled in production as it can expose critical information to clients.

Setting DEBUG=False will display build-in templates in case of an error: 400,403,404,500

We can replace these build-in templates with our own by including, for example **404.html** template, in templates directory of the main application (**app_name**).

NOTE: Custom error tempaltes should not be complicated, so minimize the use of template tags. This is especially true for the **500.html** template, in fact, you should make it fully static since failed service of the server may not be able to render the template.

```
project_folder/app_name/templates/404.html </> 404.html

{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>404 - Error</title>
    <link rel="stylesheet" href="{% static 'app_name/css/error.css' %}">
</head>
<body>
    <main>
        <label class="hidden" style="--delay: 6.4s">ERROR</label>
        <h1 class="hidden" style="--delay: 1s">
            <span style="--delay: 0.5s">4</span>
            <span style="--delay: 1s">0</span>
            <span style="--delay: 1.5s">4</span>
        </h1>
        <h2 class="hidden" style="--delay: 4s">Page Not Found</h2>
        <p class="hidden" style="--delay: 6.2s">The page may have been removed, renamed or is temporarily unavailable. Please try again later</p>
    </main>
</body>
</html>
```

```
* { box-sizing: border-box; padding: 0; margin: 0; cursor: default; }
body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: calc(100vh - 100px);
    overflow: hidden;
}
main { max-width: 750px; font-family: arial; color: hsl(240, 6%, 20%); }
label {
    position: absolute;
    font-family: arial;
    font-weight: bold;
    font-size: 2em;
    color: #dad6d6;
}
h1 { font-size: 10em; white-space: nowrap; }
h2 { font-size: 2.5em; }
p { font-size: 1.75em; }

@keyframes fadeIn {
    from { opacity: 0; transform: translateY(20px); }
    to { opacity: 1; transform: translateY(0); }
}
@keyframes gradientAnimation {
    0% { opacity: 0; background-position: 0% 50%; font-size: 11em; }
    25% { opacity: 1; }
    50% { background-position: 200% 50%; font-size: 10em; }
}

.hidden {
    --delay: .5s;
    opacity: 0;
    animation: fadeIn .5s ease-in-out var(--delay) forwards;
}
span {
    --delay: 0s;
    margin: 0;
    padding: 0;
    background: linear-gradient(45deg, #dad3d3, #444241, #c4b4b2);
    background-size: 400% 400%;
    animation: gradientAnimation 5s ease-in-out var(--delay) forwards;
    background-clip: text;
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
}
```

Serve staticfiles in Production with WhiteNoise

You might have encountered a problem when serving the stylesheet from previous part of the documentation while being in production.

Normally, Django's built-in static file handling works fine in development, but when you switch to production via `DEBUG=False`, Django **does not serve static files**—it expects a dedicated static file server like Nginx or an external storage solution (e.g., AWS S3).

You should already have some experience with initializing Amazon S3 bucket. However, there is alternative solution to quickly simulate the production environment locally.

WhiteNoise steps in by allowing Django to serve static files **directly**, without needing a separate web server or storage provider. It compresses files, adds caching headers, and handles efficient delivery, making it ideal for small-to-medium projects or when you don't want to rely on a CDN (Content Delivery Network).

In **development**, Django serves static files directly from the `static/` folders inside each app when `DEBUG=True`. The `STATICFILES_DIRS` setting allows additional locations for static files.

In **production**, Django **does not serve static files by itself** when `DEBUG=False`. Instead, the `STATIC_ROOT` directory is used, where all static files are collected.

How Django Handles Static Files

- Django **searches for static files across multiple locations** using `STATICFILES_FINDERS`, meaning it can pull static assets from individual apps or shared locations.
- If an app (`app_name`, `users`, `chat`) has a `static/` directory, Django treats it as a valid source for assets when you use `{% static 'path/to/resource' %}`.
- However, in **production**, all static files are collected into **one central directory** (`STATIC_ROOT`) via `collectstatic`, ensuring assets from different apps are merged into a single location.

```
(venv) @ project_folder          manage.py
python manage.py collectstatic
```

Install WhiteNoise

💻 (venv) @ project_folder

📚 pip library

```
pip install whitenoise  
pip freeze > requirements.txt # update requirements.txt
```

Official WhiteNoise Documentation for Django:
<https://whitenoise.readthedocs.io/en/latest/django.html>

📚 WhiteNoise

🐍 settings.py

```
INSTALLED_APPS = [  
    'daphne',  
    'whitenoise.runserver_nostatic',  
    ...  
]
```

WhiteNoise app should be installed before 'django.contrib.staticfiles' to avoid conflicts.

```
MIDDLEWARE = [  
    ...  
    'django.middleware.security.SecurityMiddleware',  
    'whitenoise.middleware.WhiteNoiseMiddleware',  
    ...  
]
```

WhiteNoise middleware should be installed after 'django.middleware.security.SecurityMiddleware'

Update backend of staticfiles

```
STORAGES = {  
    "default": {  
        "BACKEND": DEFAULT_FILE_STORAGE,  
        "OPTIONS": {  
            "access_key": AWS_ACCESS_KEY_ID,  
            "secret_key": AWS_SECRET_ACCESS_KEY,  
            "bucket_name": AWS_STORAGE_BUCKET_NAME,  
        },  
    },  
    "staticfiles": {  
        "BACKEND": "django.contrib.staticfiles.storage.StaticFilesStorage",  
        "BACKEND": "whitenoise.storage.CompressedManifestStaticFilesStorage",  
        "OPTIONS": {  
            "location": STATIC_ROOT,  
        },  
    },  
}
```

NOTE: Consider a CDN for high traffic and frequent requests for static files

Account Model



This exercise requires a completion of stripe payment system with the **donate** view, and **3 – 5** registered users.

The User model is extended through the Profile model, which was previously created in the User app. The Profile model is specifically designed to store public user information, including contact details, bio, custom nickname, avatar image, and other non-critical attributes.

To manage the overall account state, an Account model is required. This model should include essential properties such as account status (e.g., disabled or suspended), email verification status, multi-factor authentication settings, and some privileges (staff / admin).

Profile model includes **stripe_customer_id** and **stripe_last_intent_id**. These properties belong to **Account** model. Hence, a need for custom migration

In this part of the documentation we will:

- **Organize** models.py OF the **users** app BY moving AWS S3 definitions TO utils.py.
- **Create Account model** and register it in the **admin** panel.
- **Update** utility get_or_create_stripe_customer().
- **Update** some views in **app_name** application and **profileView()** in **users** app.
- **Migrate** new **Account** model.
- **Create custom migration** to migrate users' stripe data TO their **Account** model.
- **Remove** stripe data from **Profile** model and unregister that data in **Profile's admin panel**.

Organize models.py and create Account model

```
project_folder/users/utils.py (users app) 🐍 utils.py

from django.core.files.uploadedfile import UploadedFile
from django.conf import settings

import requests, stripe, boto3 # ← Added boto3
from botocore.exceptions import ClientError

stripe.api_key = settings.STRIPE_SECRET_KEY

s3 = boto3.client(
    's3',
    aws_access_key_id=settings.AWS_ACCESS_KEY_ID,
    aws_secret_access_key=settings.AWS_SECRET_ACCESS_KEY,
    region_name=settings.AWS_S3_REGION_NAME
)
... more on other page
```

```

def is_profile_pic(image_key):
    ...

def remove_profile_pic(image_key):
    ...

def recycle_profile_pic(image_key):
    ...

... def is_image_nsfw():

```

NOTE: `is_profile_pic` is not used anywhere, so it can be removed.
 Above functions are found in `models.py` in `users` application.
 Move them TO `utils.py`

📝 project_folder/`users/models.py`

(`users` app) 🐍 `models.py`

```

from django.db import models
from django.contrib.auth.models import User
from django_resized import ResizedImageField
from django.conf import settings

from .utils import (
    remove_profile_pic,
    recycle_profile_pic,
    get_or_create_stripe_customer,
)

```

Moved – (ClientError , s3 , is_profile_pic() , remove_profile_pic() , recycle_profile_pic()) TO utils.py

```

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    stripe_customer_id = models.CharField(
        max_length=255, blank=True, null=True, unique=True, editable=False)
    stripe_last_intent_id = models.CharField(
        max_length=255, blank=True, null=True, unique=True, editable=False)

    phone_number = models.CharField(max_length=15)
    image = ResizedImageField(
        size=[300, 300],
        crop=['middle', 'center'],
        quality=75,
        force_format='JPEG',
        upload_to='profile_pics',
        default='default_profile.jpg'
    )

```

```

def save(self, *args, **kwargs):
    try:
        this = Profile.objects.get(user=self.user)

        # Create Stripe customer for new Profile.
        # (Stripe is used for making payments)
        if not hasattr(this, 'stripe_customer_id'):
            self.stripe_customer_id = get_or_create_stripe_customer(self.user)

        # Only move the old image if there is an existing image
        # and it's different from the new image
        if this.image and this.image.name != self.image.name:
            #recycle_profile_pic(this.image.name)
            remove_profile_pic(this.image.name)

```

NOTE: Remember what was your option - (`recycle` | `remove`) profile picture.
Use `remove_profile_pic` TO save space in S3 bucket at Amazon Web Services

```

except Profile.DoesNotExist:
    # This is a new profile, so no need to move any image.
    pass

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

```

```

class Account(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    # State
    is_active = models.BooleanField(default=True)
    is_suspended = models.BooleanField(default=False)
    suspension_end = models.DateTimeField(blank=True, null=True)

    # Security
    has_verified_email = models.BooleanField(default=False)
    mfa_secret = models.CharField(max_length=32, blank=True, null=True)
    mfa_enabled = models.BooleanField(default=False)
    #backup_codes = ""

    # Payment
    stripe_customer_id = models.CharField(
        max_length=255, blank=True, null=True, unique=True, editable=False)
    stripe_last_intent_id = models.CharField(
        max_length=255, blank=True, null=True, unique=True, editable=False)

... continues on next page

```

NOTE: I haven't removed `stripe_customer_id` NOR `stripe_last_intent_id` FROM `Profile` model. We will remove them later as they are required for now

```

def save(self, *args, **kwargs):
    try:
        account_instance = Account.objects.get(user=self.user)

        # Create Stripe customer for new Account.
        # (Stripe is used for making payments)
        if not hasattr(account_instance, 'stripe_customer_id'):
            self.stripe_customer_id = get_or_create_stripe_customer(self.user)

    except Account.DoesNotExist:
        pass

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

```

Register Account model in admin panel

 project_folder/users/admin.py (users app)  admin.py

```

from django.contrib import admin
from .models import Profile, Account # ← Added Account

class ProfileAdmin(admin.ModelAdmin):
    readonly_fields = (
        'stripe_customer_id',
        'stripe_last_intent_id'
    )
    list_display = [field.name for field in Profile._meta.fields]

@admin.register(Account)
class AccountAdmin(admin.ModelAdmin):
    readonly_fields = (
        'stripe_customer_id',
        'stripe_last_intent_id'
    )
    list_display = [field.name for field in Account._meta.fields]

# Register your models here.
admin.site.register(Profile, ProfileAdmin)

```

NOTE: `@admin.register(Account)` = `admin.site.register(Account, AccountAdmin)`. Otherwise we'd have to repeat ourselves. Don't modify `ProfileAdmin` just yet!

Update `get_or_create_stripe_customer()` utility

We have organized models.py, and registered new model: Account.

Let's update `get_or_create_stripe_customer()` utility
TO retreive and register stripe data TO **Account** model, instead of a **Profile** model.

We will need this refurbished function in our custom migration.

```
project_folder/users/utils.py                                (users app) 🐍 utils.py

... is_image_nsfw(): is above

def get_or_create_stripe_customer(user):
    """Retrieves or creates a Stripe customer ID for the given user.
    Args:
        user: Django auth User instance
    Returns:
        str: The Stripe customer ID from the user's Account model
    Note:
        If no Stripe customer ID exists, this function will:
        1. Create a new Stripe customer account
        2. Save the ID to the user's Account model
        3. Return the newly created ID
    """
    stripe_customer_id = None

    if hasattr(user, 'account'):
        # Retrieve the user's Stripe customer ID from its account
        stripe_customer_id = getattr(user.account, 'stripe_customer_id', None)

    if stripe_customer_id is None:
        # Create a new Stripe customer
        customer = stripe.Customer.create(
            name=user.username,
            email=user.email,
            metadata={'user_id': str(user.id)})
        stripe_customer_id = customer.id

    if hasattr(user, 'account'):
        # Update the user's account with the new Stripe customer ID
        user.account.stripe_customer_id = stripe_customer_id
        user.account.save()

    return stripe_customer_id
```

Update **views** in **app_name** app and **profileView** in **users** app

Replace all ...**profile.stripe_**... with ...**account.stripe_**...

project_folder/app_name/views.py (app_name app) 🐍 views.py

```
def donateView(request):
    ...
    intent = stripe.PaymentIntent.create(
        amount=1000,
        currency='PLN',
        payment_method_types=['card'],
        confirm=True
    )
    stripe.Customer.modify(
        user.account.stripe_last_intent_id,
        amount=int(donation * 100),
        metadata={'donation': str(donation)}
    )
    ...
    return render(request, 'app_name/donate.html', {'intent': intent})
```

```
def donateUpdatePaymentIntentView(request):
    ...
    donation = request.POST.get('donation')
    stripe.PaymentIntent.modify(
        id=request.user.account.stripe_last_intent_id,
        amount=int(donation * 100),
        metadata={'donation': str(donation)}
    )
    ...
    return redirect('donate')
```

project_folder/users/views.py (users app) 🐍 views.py

```
def profileView(request, user_id):
    ...
    if check_password(confirm_password, user_instance.password):
        form_user.save()
        form_profile.save()
        stripe.Customer.modify(
            user_instance.account.stripe_customer_id,
            name=user_instance.username,
            email=user_instance.email
        )
        messages.success(request, 'Successfully updated profile')
    else:
        messages.error(request, 'Invalid password!')
        form_user.add_error('confirm_password', 'Invalid password')
```

Update the database with migrations

Migrate users' stripe data FROM Profile TO new Account model

```
(venv) @ project_folder manage.py
```

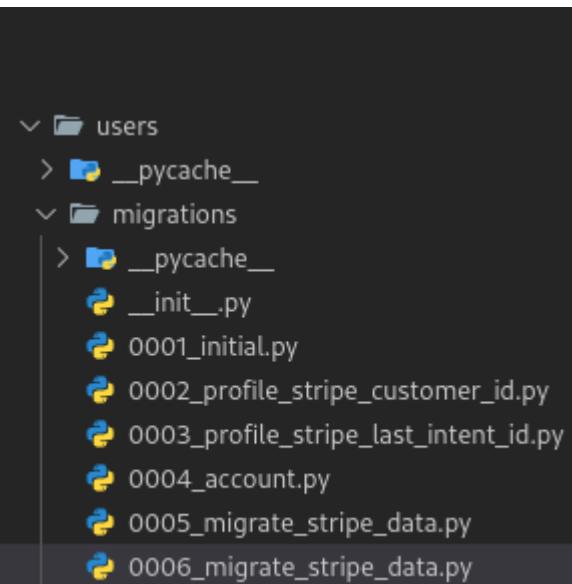
```
python manage.py makemigrations
python manage.py makemigrations -empty users -name migrate_stripe_data
```

NOTE: Ignore my migrations (in a provided example).

Our migrations may differ as I was doing other things for testing purposes. You should focus on the last two migrations in your users migrations folder instead.

The before last migration `NNNN_account.py` is used TO include the **Account** model, so do not remove it.

We will modify the `NNNN_migrate_stripe_data.py` TO move existing users' **stripe data** FROM their **Profile** model TO their **Account** model, which will be assigned and populated via custom definition – `migrate_stripe_data()`.



```
project_folder/users/migrations/NNNN_migrate_stripe_data.py (users app) manage.py
```

```
# Generated by Django N.N.N on YYYY-MM-DD HH:MM
```

```
from django.db import migrations
from users.utils import get_or_create_stripe_customer
```

```
def migrate_stripe_data(apps, schema_editor):
    Profile = apps.get_model('users', 'Profile')
    Account = apps.get_model('users', 'Account')

    for profile in Profile.objects.all():
        # Get or create Account without triggering save()
        account, created = Account.objects.get_or_create(user=profile.user)

        # Migrate existing Stripe data if available
        if profile.stripe_customer_id:
            account.stripe_customer_id = profile.stripe_customer_id
            account.stripe_last_intent_id = profile.stripe_last_intent_id
```

```
# Only create new Stripe customer if profile never had one
elif not account.stripe_customer_id:
    account.stripe_customer_id = get_or_create_stripe_customer(profile.user)

# Save with direct SQL update to bypass model save()
Account.objects.filter(pk=account.pk).update(
    stripe_customer_id=account.stripe_customer_id,
    stripe_last_intent_id=account.stripe_last_intent_id
)
```

```
class Migration(migrations.Migration):

    dependencies = [
        ('users', 'NNNN_account'),
    ]

    operations = [
        migrations.RunPython(migrate_stripe_data),
    ]
```

NOTE: **NNNN_account** should be your **before last** migration @ users/migrations

```
(venv) @ project_folder                                     manage.py
python manage.py migrate
python manage.py runserver localhost:8000
```

Head TO `localhost:8000/admin` (your admin panel) TO verify changes

Verify changes in admin panel

<http://localhost:8000/admin/users/profile/>

Django administration

Home > Users > Profiles

Start typing to filter...

APP_NAME

Model names [+ Add](#)

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

CHAT

Messages [+ Add](#)

Rooms [+ Add](#)

USERS

Accounts [+ Add](#)

Profiles [+ Add](#)

Select profile to change

Action: ----- [Go](#) 0 of 4 selected

<input type="checkbox"/>	ID	USER	STRIPE CUSTOMER ID	STRIPE LAST INTENT ID
<input type="checkbox"/>	5	mcratzick	cus_RzWbO017giui3S	pi_3RSTxGCMc3sguBOi0mW7wemO
<input type="checkbox"/>	4	test3	-	-
<input type="checkbox"/>	3	test5	cus_RzWRuo4UTxx8s2	-
<input type="checkbox"/>	2	test6	cus_RzZD4oKMyKxn4V	-

4 profiles

<http://localhost:8000/admin/users/account/>

Django administration

Welcome, GABRIEL [VIEW SITE](#) / [CHANGE PASSWORD](#)

Home > Users > Accounts

Start typing to filter...

APP_NAME

Model names [+ Add](#)

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

CHAT

Messages [+ Add](#)

Rooms [+ Add](#)

USERS

Accounts [+ Add](#)

Profiles [+ Add](#)

Select account to change

Action: ----- [Go](#) 0 of 4 selected

<input type="checkbox"/>	ID	USER	IS ACTIVE	IS SUSPENDED	SUSPENSION END	HAS VERIFIED EMAIL	MFA SECRET	MFA ENABLED	STRIPE CUSTOMER ID	STRIPE LAST INTENT ID
<input type="checkbox"/>	4	test3	●	○	-	○	-	○	cus_SNUcbXideeqvBD	-
<input type="checkbox"/>	3	mcratzick	●	○	-	○	-	○	cus_RzWbO017giui3S	pi_3RSTxGCMc3sguBOi0mW7wemO
<input type="checkbox"/>	2	test5	●	○	-	○	-	○	cus_RzWRuo4UTxx8s2	-
<input type="checkbox"/>	1	test6	●	○	-	○	-	○	cus_RzZD4oKMyKxn4V	-

4 accounts

MFA ENABLED	STRIPE CUSTOMER ID	STRIPE LAST INTENT ID
✗	cus_SNUcbXideeqvBD	-
✗	cus_RzWbO017giui3S	pi_3RSTxGCMc3sguBOi0mW7wemO
✗	cus_RzWRuo4UTxx8s2	-
✗	cus_RzZD4oKMyKxn4V	-

NOTE: stripe_customer has been successfully created FOR user: "test3"

Remove stripe data FROM Profile model and its **admin** panel

```
project_folder/users/admin.py (users app) 🐍 admin.py

from django.contrib import admin
from .models import Profile, Account

@admin.register(Profile)
class ProfileAdmin(admin.ModelAdmin):
    readonly_fields = (
        'stripe_customer_id',
        'stripe_last_intent_id'
    )
    list_display = [field.name for field in Profile._meta.fields]

@admin.register(Account)
class AccountAdmin(admin.ModelAdmin):
    readonly_fields = (
        'stripe_customer_id',
        'stripe_last_intent_id'
    )
    list_display = [field.name for field in Account._meta.fields]

# Register your models here.
admin.site.register(Profile, ProfileAdmin)
```

```
project_folder/users/models.py (users app) 🐍 models.py

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    stripe_customer_id = models.CharField(
        max_length=255, blank=True, null=True, unique=True, editable=False)
    stripe_last_intent_id = models.CharField(
        max_length=255, blank=True, null=True, unique=True, editable=False)
...
...
```

```
(venv) @ project_folder 🐍 manage.py

python manage.py makemigrations users
python manage.py migrate
```

2FA (Two Factor Authentication)



This exercise requires a completion of **Account model** and be confident with **Django + Python** and **JavaScript + HTMX**

This section of the documentation assumes a solid grasp of previously applied utilities and methodologies, as it emphasizes full-stack development. It highlights frequent asynchronous communication between the client and server, both directly and indirectly.

We will also utilize API from two platforms – **Zoho Mail** & **Twilio**. For serving OTPs

Important Abbreviations

- **MFA (Multi-Factor Authentication)**: Requires multiple forms of verification, such as a password plus a biometric scan.
- **2FA (Two-Factor Authentication)**: A subset of MFA, involving exactly two authentication steps—often combining something you know (password) with something you have (a phone or security key).
3D Secure is an example used in payment authentication.
- **OTP (One-Time Password)**: A single-use code, typically sent via SMS or email, valid for a short period.

We will use OTP based of User's MFA secret. User will receive OTP via Email, SMS or Mobile Authenticator App of User's choice. (I recommend [Microsoft's Authenticator App](#))



The screenshot shows a user profile interface. On the left, there is a large profile picture of a white rose against a blue and purple background. Below the picture is a dark grey bar with the text "My Chats". To the right of the picture is a form with the following fields:

- Image***: A file input field currently set to "profile_pics/aura.jpeg". There is also a "Choose file" button and a message "No file chosen".
- Phone number***: An input field containing "My Phone Number :)" with a clear button (X) to its right.
- Enable MFA**: A toggle switch that is currently turned on (indicated by a grey circle).
- Username***: An input field containing "mcrazick" with a small edit icon to its right.

We're going to create this new togglable button in **Profile** template – **Enable MFA**

Enable MFA

Enable MFA 1/4 X

Enter your password to configure Multi-Factor Authentication

Password

Proceed to the next step

Enable MFA 2/4 X

Download [authenticator app](#) on your mobile device, and use it to **Scan the QR code**

To receive the one-time password (OTP) required to activate **Multi-Factor Authentication**

Enter code

SARY AKF6 BJ55 RIEX MQRU KV2A CUTA P5ZD
Into your authentication app if you're unable to scan the QR code

OTP

Proceed to the next step

Enable MFA 3/4 X

Enter the one-time password (OTP) sent to your email:
mcratzick@mail.com

43
sec

OTP

Proceed to the next step

OTP

Proceed to the next step

Enable MFA 4/4 X

Enter the one-time password (OTP) sent to your mobile via text:
07198392323

to finalize Multi-Factor Authentication

0
sec

OTP

Enable Multi-Factor Authentication

We will utilize **Bootstrap5** library TO create 4 template [Modals](#):

1. password.html
2. otp_qrcode.html
3. otp_email.html
4. otp_sms.html

All inheriting from a **base.html** (Modal template) via **include.html**

We will utilize **HTMX** library to communicate with the backend TO serve a Modal at-a-time, (Based on the current **step <input>**)

Images are available in [Users App - Static](#)

NOTE: Familiarity with the **hx-swap-oob** attribute is essential, as it will be used in conjunction with Django's **{% include %}** and **{% extends %}** template tags

These modals will be **requested** on demand, **and** terminated on **submit + close** event

Install PyOTP + QRCode + Twilio and PhoneNumberField

💻 (venv) @ project_folder

📚 pip library

```
pip install pyotp qrcode twilio "django-phonenumber-field[phonenumberslite]"  
pip freeze > requirements.txt # update requirements.txt
```

NOTE: - **PyOTP** is used to generate OTPs
- **QRCode** is used to generate QR Image
- **Twilio** API. Is used to request OTPs via SMS. [Twilio Docs SMS setup](#)
- **django-phonenumber-field** converts phonenumbers input TO [E.164 Format](#)

📚 django-phonenumber-field

🐍 settings.py

```
INSTALLED_APPS = [  
    ...  
    'storages',  
    'phonenumbers_field',  
    ...  
]
```

Install **phonenumbers_field** app before styling libraries:

- bootstrap,
- crispy,
- sass, etc...

📝 project_folder/users/models.py

(users app) 🐍 models.py

```
...  
from phonenumbers_field.modelfields import PhoneNumberField  
import pyotp
```

```
class Profile(models.Model):  
    user = models.OneToOneField(User, on_delete=models.CASCADE)  
    phone_number = models.CharField(max_length=15)
```

```
class Account(models.Model):  
    user = models.OneToOneField(User, on_delete=models.CASCADE)  
  
    # Contact  
    phone_number = PhoneNumberField(unique=True, null=True)
```

Migrate changes. (I didn't bother migrating phone number's of my users)

IF you had 100 of Users with unique phone numbers at Profile model,
THEN you may want to consider constructing a custom migration.

```

class Account(models.Model):
...
    def save(self, *args, **kwargs):
        try:
            account_instance = Account.objects.get(user=self.user)

            # Create Stripe customer for new Account.
            # (Stripe is used for making payments)
            if not hasattr(account_instance, 'stripe_customer_id'):
                self.stripe_customer_id = get_or_create_stripe_customer(self.user)
            # Create MFA Secret for new Account.
            # (MFA secret is used for generating One-Time password)
            if not hasattr(account_instance, 'mfa_secret'):
                self.mfa_secret = pyotp.random_base32()

        except Account.DoesNotExist:
            pass

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

```

Update the Main <form> at Profile Template

 project_folder/users/forms.py  (users app)  forms.py

```

from django import forms
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm
from .models import Profile, Account

class ProfileForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ['image'] # Removed: 'phone_number', from the list

class AccountForm(forms.ModelForm):
    class Meta:
        model = Account
        fields = ['phone_number']
...

```

```
...  
from .models import Profile  
from .forms import UserRegisterForm, UserUpdateForm, ProfileForm, AccountForm  
...  
  
@login_required  
def profileView(request, user_id):  
    user_instance = get_object_or_404(User, pk=user_id)  
  
    if request.method == 'POST' and request.user.id == user_instance.id:  
  
        form_user = UserUpdateForm(  
            request.POST,  
            instance=user_instance  
        )  
        form_profile = ProfileForm(  
            request.POST,  
            request.FILES,  
            instance=user_instance.profile  
        )  
        form_account = AccountForm(  
            request.POST,  
            instance=user_instance.account  
        )  
  
        if form_user.is_valid() and form_profile.is_valid() and form_account.is_valid():  
            confirm_password = form_user.cleaned_data.get('confirm_password')  
            if check_password(confirm_password, user_instance.password):  
                form_user.save()  
                form_profile.save()  
                form_account.save()  
                ...  
  
            else:  
                form_user = UserUpdateForm(instance=user_instance)  
                form_profile = ProfileForm(instance=user_instance.profile)  
                form_account = AccountForm(instance=user_instance.account)  
  
                context = {  
                    'profile_user': user_instance,  
                    'forms': {  
                        'user': form_user,  
                        'profile': form_profile,  
                        'account': form_account  
                    }  
                }  
    }
```

 project_folder/users/templates/users/profile.html </> profile.html

```
...
{%- if user.id == profile_user.id %}

<form action="{% url 'profile' profile_user.id %}"
      enctype="multipart/form-data"
      method="POST">

    {% csrf_token %}
    {% for form in forms.profile|crispy %}
    {% for form in forms.account|crispy %}
    {% for form in forms.user|crispy %}

<div class="row justify-content-between">
    <div class="col-auto">
        <button class="btn btn-warning" type="submit">Update Profile</button>
    </div>
    <div class="col-auto">
        <a href="{% url 'password_reset' %}" class="btn btn-link">
            change password
        </a>
    </div>
</div>

</form>

{%- else %}
...

```

TASK: User Registration and Phone Number Update

1. **Register a new user** and ensure successful creation.
2. **Verify MFA Setup** in the admin panel, confirming the user has a unique mfa_secret.
3. **Login as the newly created user** to ensure authentication works correctly.
4. **Update the user's phone number**, ensuring that any national number provided is automatically converted to **E.164 format**.
5. **Verify the update** by checking the stored number in the admin panel.

Enable MFA

(Continues on the next page)

Setup environment variables AND configure settings.py

```
📝 project_folder/.env .env

# Website/Company name
COMPANY_NAME='MyCompany'

# Twilio API Keys (We'll get the valid keys after creating some utilities)
TWILIO_ACCOUNT_SID='AUXXXXXXXXXXXXXXXXXXXXXXXXXXXXd091'
TWILIO_AUTH_TOKEN='yixXXXXXXXXXXXXXXXXXXXX6d12'
TWILIO_PHONE_NUMBER='+NNXXXXXXX51'
```

📝 project_folder/users/settings.py (users app) 🐍 settings.py

```
COMPANY_NAME = str(os.environ['COMPANY_NAME'])
```

NOTE: COMPANY_NAME variable is custom made. Hence, independent from libraries

```
# Django Phonenumber Field
```

```
# -- WARNING! -- Data loss may occur when changing the DB format!
# Hence, it is critical to read the below documentation before making any changes.
# https://django-phonenumber-field.readthedocs.io/en/latest/reference.html#phone-number-format-choices
```

```
PHONENUMBER_DB_FORMAT='E164'      # 'E164' by default when unspecified
PHONENUMBER_DEFAULT_FORMAT='E164' # 'E164' by default when unspecified
PHONENUMBER_DEFAULT_REGION='GB'   # 'None' by default when unspecified
```

```
# One-Time Password for Multi-Factor Authentication
```

```
OTP_ISSUER_NAME = COMPANY_NAME
OTP_DEFAULT_INTERVAL = 30 # 30 seconds
OTP_EMAIL_INTERVAL = 180 # 3 minutes
OTP_SMS_INTERVAL = 180 # 3 minutes
```

NOTE: OTP__... variables are custom made and therefore NOT dependent on libraries

```
# Twilio API keys (USE Live credentials API Keys instead of Test credentials)
```

```
TWILIO_ACCOUNT_SID = os.getenv('TWILIO_ACCOUNT_SID')
TWILIO_AUTH_TOKEN = os.getenv('TWILIO_AUTH_TOKEN')
TWILIO_PHONE_NUMBER = os.getenv('TWILIO_PHONE_NUMBER')
```

 project_folder/users/utils.py (users app)  utils.py

```
from django.core.files.uploadedfile import UploadedFile
from django.core.mail import send_mail
from django.conf import settings
from twilio.rest import Client

import io, base64, requests, stripe, boto3, pyotp, qrcode
from botocore.exceptions import ClientError
...

def get_or_create_mfa_secret_for_user(user_instance):
    """
    Generates a random base32 string and saves it as mfa_secret
    in the, attached to the auth user_instance, account model.

    Returns: 'mfa_secret' from user_instance.account model (on success),
             'False' otherwise.
    """

```

```
if hasattr(user_instance, 'account'):
    if not user_instance.account.mfa_secret:
        user_instance.account.mfa_secret = pyotp.random_base32()
        user_instance.account.save()

return user_instance.account.mfa_secret or False
```

```
def get_users_mfa_secret_as_qrcode_base64(user_instance):
```

```
    """

```

```
    Generates a Base64-encoded QR code from an MFA secret.
```

```
This function:
```

- This function ensure the user has a stored MFA secret.
- Converts the MFA secret into an OTP provisioning URI.
- Generates a QR code from the OTP URI.
- Stores the QR code in an in-memory buffer as a PNG.
- Encodes the QR code image into a Base64 data URI format.

```
Args:
```

```
    user_instance: A Django User instance.
```

```
Returns:
```

```
    str: A Base64-encoded PNG data URI suitable for embedding in HTML.
```

```
    """

```

```
mfa_secret = get_or_create_mfa_secret_for_user(user_instance)
```

```
...continues on the next page
```

```

otp_uri = pyotp.totp.TOTP(mfa_secret).provisioning_uri(
    name=user_instance.email,
    issuer_name=settings.OTP_ISSUER_NAME
)

# Convert OTP URI to QR Code as PNG
qr = qrcode.make(otp_uri)      # Convert OTP URI → QR Code
buffer = io.BytesIO()          # Set in-memory buffer to temporarily store the QR Code
qr.save(buffer, format='PNG')   # Convert QR Code → PNG image & store it in the buffer
buffer.seek(0)                  # Move buffer's reading position to the beginning

# Return QR image AS data base64 URI
qrcode_png_base64 = base64.b64encode(buffer.getvalue()).decode('utf-8')
return f'data:image/png;base64,{qrcode_png_base64}'


def generate_otp_for_user(user_instance, interval=settings.OTP_DEFAULT_INTERVAL):
    """
    Generates a time-based one-time password (OTP) using the user's MFA secret

    This function ensure the user has a stored MFA secret,
    generates an OTP using TOTP (based on interval)

    Args:
        user_instance: A Django User instance.
        interval: Natural number (in seconds).

    Returns:
        str: The generated OTP
    """
    mfa_secret = get_or_create_mfa_secret_for_user(user_instance)

    totp = pyotp.TOTP(mfa_secret, interval=interval)
    otp = totp.now()

    return otp

```

...continues on the next page

```
def email_otp_to_user(user_instance):
    """
    Generates a time-based one-time password (OTP)
    using the user's MFA secret and sends it via email.

    This function ensures the user has a stored MFA secret,
    generates an OTP using TOTP (based on OTP_EMAIL_INTERVAL),
    and delivers it to their registered email address.

    Args:
        user_instance: A Django User instance.

    Returns:
        str: The generated OTP (on success). bool: False (on missing otp).
    """

```

```
interval = settings.OTP_EMAIL_INTERVAL

otp = generate_otp_for_user(user_instance, interval=interval)
if not otp:
    return False
```

```
# Send OTP via email
sender_email = settings.EMAIL_HOST_USER
subject = 'Your OTP Code'
message = f'Your OTP code is {otp}. It expires in {interval} seconds.
\n\nPlease do not share this code with anyone!'

send_mail(subject, message, sender_email, [user_instance.email])
return otp
```

```
def sms_otp_to_user(user_instance):
    """
    Generates a time-based one-time password (OTP)
    using the user's MFA secret and sends it via SMS.

    This function ensures the user has a stored MFA secret,
    generates an OTP using TOTP (based on OTP_SMS_INTERVAL),
    and delivers it to their registered in account model:
        - phone number (which should be in E.164 format)
```

Args:

user_instance: A Django User instance.

Returns:

str: The generated OTP (on success). bool: False (on missing otp).

...continues on the next page

```

interval=settings.OTP_SMS_INTERVAL

otp = generate_otp_for_user(user_instance, interval=interval)
if not otp:
    return False

account_sid = settings.TWILIO_ACCOUNT_SID
auth_token = settings.TWILIO_AUTH_TOKEN
client = Client(account_sid, auth_token)

# Both 'from_' and 'to' phone numbers must be of format: E.164
message = client.messages.create(
    body=f'Your OTP code is {otp}. It expires in {interval} seconds.
\n\nPlease do not share this code with anyone!',
    # Twilio Phone Number must support SMS
    from_=str(settings.TWILIO_PHONE_NUMBER),
    to=str(user_instance.account.phone_number)
)

return otp

```

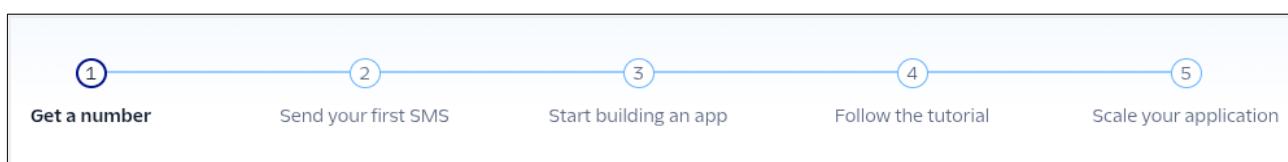
Register Twilio Phone Number



Once you register a Twilio account at: <https://www.twilio.com/try-twilio>

Head to the Twilio console panel at: <https://console.twilio.com>

Follow the steps: 1 – 2



NOTE: Working script is already implemented in **Users** app utils.py - **sms_otp_to_user()**

By scrolling down, you should see **Go to API Keys** link at **Account Info**
<https://console.twilio.com/us1/account/keys-credentials/api-keys>

Head there after you register new Twilio Phone Number, and
Copy the **Live credentials** into your **environment variables**

NOTE: **Test credentials** cannot send **SMS** at least during testing, this was the case, though it may have changed. Nonetheless, consider this a heads-up about a potential issue

Step 1: Get a Twilio phone number

To get started sending or receiving SMS messages with Twilio, you can start by getting a phone number. You can use this number to receive incoming calls or text messaging to any phone or application.

Your trial includes one UK long code phone number

- UK long code phone numbers support two-way SMS to URLs.
- Continue to build and engage with customers at scale by sending to non-US/Canada countries and 1 MPS for US/CANADA.
- To get additional local phone numbers, you must meet [regional routing requirements](#).

[Get phone number](#)

Capabilities

Voice

Receive incoming calls and make outgoing calls.

Fax

Send and receive faxes.

SMS

Send and receive text messages.

Global Routing

Routing ([Regional](#))

Voice and Messaging will be routed to the United States (US1) Region. You can re-route in the number configuration after purchase.

Make sure that your Phone Number of choice:

- Provides SMS functionality TO send text messages: Requested by the user OTP(s)
- Routing is set to United States (US1) as some routers do NOT support all features
- Is situated near your target – clients. IF you operate in UK – Choose an UK Number

The screenshot shows two parts of the Twilio interface. On the left, under 'Select End-User', it asks 'Who will use +44 01234 56789?' with two options: 'Business' (radio button) and 'Individual' (radio button, selected). On the right, a modal titled 'Comply with Regulatory Requirements' appears. It says 'Assign an approved UNITED KINGDOM LOCAL INDIVIDUAL Regulatory Bundle' and provides a link to 'Learn more'. Below that, it says 'Assign approved Bundle' with a text input field. A red arrow points from the 'Create a Regulatory Bundle' link in the modal to the 'Create a Regulatory Bundle' link in the text below it. A red box highlights the 'Create a Regulatory Bundle' link.

Make sure that your Phone Number of choice:

- Is setup correctly for your needs. I am NOT operating as a business. Since my Django application is a personal project for educational purposes only. Hence, **individual End-User**
- Complies with the **Regulatory Requirements**. You will need to provide documentation, such that proves - Your/Companies address. Best documentation is that issued by the government: Passport and Driving License. You may also provide other supporting documents, such as: Bank Statement or University Associate Card

It might take some time for Twilio to verify documents' integrity. (4 - 6 hours depending on the day and ongoing events like bank holidays). The bundle should arrive to your mail box

Comply with Regulatory Requirements

Assign an approved UNITED KINGDOM LOCAL INDIVIDUAL Regulatory Bundle
Ensure the number +44 131 381 0496 is compliant with local regulations by assigning a Twilio Approved Local Regulatory Bundle for your United Kingdom (GB) number. [Learn more](#)

Assign approved Bundle

United Kingdom: Local - Individual 2025-06-02 at 19:58:22 (redacted)

If you don't have an approved bundle, you can [Create a Regulatory Bundle](#)

[Back](#) [Buy +44 131 381 0496](#)

NOTE: "+441313810496"
(My old Twilio number, and it doesn't support SMS)

Is a phone number in a **E.164** format

Number Purchased

You have purchased:
+44 131 381 0496

Capabilities

- Voice**
Receive incoming calls and make outgoing calls.
- Fax**
Send and receive faxes.

Global Routing

Routing (Regional)
Voice and Messaging are routed to the United States (US1) Region.
You can re-route in the number configuration after purchase.

[Close](#) [Configure +44 131 381 0496](#)

Send SMS via Twilio Phone Number



Step 2: Send your first SMS

• To phone number
+449919929939

The phone number you verified your account with.

• From phone number
+441313810496

Your new Twilio phone number, provisioned in Step 1.

• Body
Hello World!

[Send test SMS](#)

Assume **From Phone Number** - is from my mobile

Request

curl Java Ruby PHP Python C#

Show auth token

```
from twilio.rest import Client

account_sid = 'AUXXXXXXXXXXXXXXXXXXXXXXXXXXXXd091'
auth_token = 'yiXXXXXXXXXXXXXXXXXXXX6d12'
client = Client(account_sid, auth_token)

message = client.messages.create(
    from_='+441313810496',
    body='Hello World!',
    to='+449919929939')

print(message.sid)
```

Response

201 - CREATED - The request was successful. We created a new resource and the response body contains the representation.

NOTE: You may receive **ERROR 21608** initially because:
Trial accounts cannot send messages to unverified numbers.

(You can also inspect **successful** and **failed** messages at - [Twilio SMS logs](#))

Resolve **ERROR 21608** for Trial Twilio Account

Trial Twilio accounts are restricted from sending messages to unverified phone numbers to prevent misuse by malicious actors. As a result, the current Twilio setup is not suitable for production use in our Django app.

To ensure smooth deployment, upgrading the account will be essential when the application goes live. In the meantime, our focus is on testing the Twilio SMS API. Let's resolve this limitation to enhance our development experience.

At **Send your first SMS**, Scroll down to section **Account Info**

The screenshot shows the 'Account Info' section of the Twilio console. It includes fields for 'Account SID' (AUXXXXXXXXXXXXXXXXXXXXXXXd091), 'Auth Token' (redacted), and 'My Twilio phone number' (+441313810496). A warning message states: 'You are in a trial account and can only send messages and make calls to [verified phone numbers](#). Learn more about your [trial account](#)'.

Account Info

Account SID
AUXXXXXXXXXXXXXXXXXXXXXXXd091

Auth Token
..... [Show](#)

⚠ Always store your token securely to protect your account. [Learn more](#)

My Twilio phone number
+441313810496

You are in a trial account and can only send messages and make calls to [verified phone numbers](#). Learn more about your [trial account](#)

[Go to account settings](#)

API Keys
[Go to API Keys](#)

Head to: <https://console.twilio.com/us1/develop/phone-numbers/manage/verified>
By pressing the **verified phone numbers** link

The screenshot shows the 'Verified Caller IDs' section of the Twilio console. It includes a note about UK phone number formatting and a form to add a new caller ID.

Verified Caller IDs

Verify a number that you own to use it as a caller ID or as the 'To' number for outbound calls/messages from the Sandbox Number. The phone numbers you buy from Twilio, or port to Twilio, can always be used as caller IDs. [Learn more about Verified Caller IDs](#)

Add a new Caller ID

Hit the **button:** [Add a new Caller ID](#)

NOTE: I had an issue when I provided Phone Number in UK's National Format: "077XXXXXXXX"

Typically, the platform should handle input data formatting automatically. However, something went wrong in this case

Hence, to avoid the issue - manually remove the leading "0"

Country
UK (+44) United Kingdom - GB

Number
9919929939

Extension

Send verification code via

SMS Call

Give the Twilio server some time to update the registered phone number, and then Head back to the **Send your first SMS** and try sending a text message. Next set [API Keys](#)

project_folder/users/views.py (users app) views.py

```

from .utils import get_users_mfa_secret_as_qrcode_base64

@login_required
def profileView(request, user_id):
    user_instance = get_object_or_404(User, pk=user_id)

    # Generate QR Code image to verify OTP for user's MFA
    qrcode_data_uri = get_users_mfa_secret_as_qrcode_base64(request.user)

    ...

    context = {
        'profile_user': user_instance,
        'qrcode_data_uri': qrcode_data_uri,
        'forms': {
            'user': form_user,
            'profile': form_profile,
            'account': form_account
        }
    }
}

```

This implementation guarantees that every user's account instance is assigned an `mfa_secret`, just like we did when initializing a Stripe customer. Without this, existing users wouldn't be able to setup a Multi-Factor Authentication.

We'll use the `qrcode_data_uri` in one of the modals.
Treat it as a password, it cannot be exposed to malicious actors.

Create modals

```

└── users
    > __pycache__
    > migrations
    > static
    └── templates / users
        └── modals / profile
            dj base.html
            dj include.html
            dj otp_email.html
            dj otp_qrcode.html
            dj otp_sms.html
            dj password.html

```

base.html will be used to extend repeating HTML elements for the modals:
`otp_email.html` | `otp_qrcode.html` | `otp_sms.html` | `password.html`

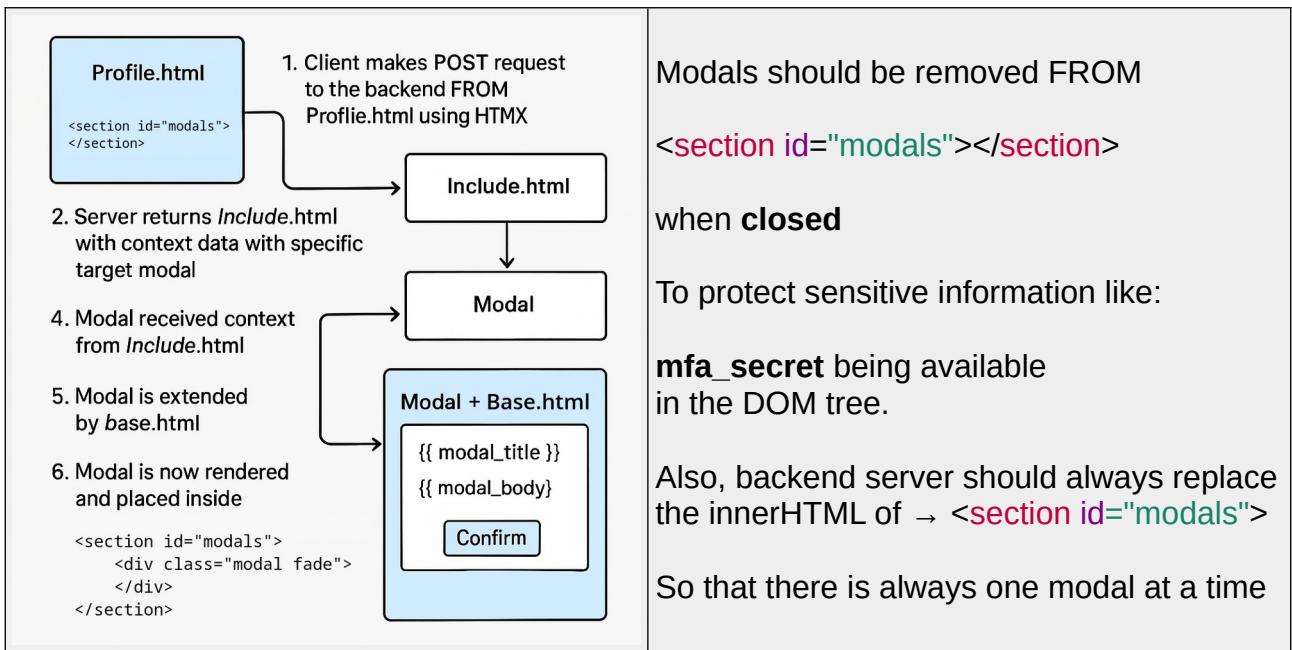
include.html combined with HTMX `hx-swap-oob` will be used as a render() template in the backend to serve specific modals

We will pass context data to serve specific modals via custom view:
`enableMFAView()`

For now, create the tree structure in **Users** app as shown in the example.
And download some image assets from: [github-django5-tutorial-users-images](#)

`authenticator_apps.png` | `mail_to_laptop.png` | `mail_to_phone.png`

And save these assets in `users / static / users / img`



project_folder/users/templates/users/profile.html

profile.html

```

...
{% block content %}
<section id="modals"></section>

<div class="container">
    ...

    <form action="{% url 'profile' profile_user.id %}"
        enctype="multipart/form-data"
        method="POST">

        {% csrf_token %}
        B {{ forms.profile|crispy }}
        B {{ forms.account|crispy }}

        <hr>
        <div class="form-check form-switch mb-3">
            <input class="form-check-input" type="checkbox" id="mfa-flex-switch"
                {% if user.account.mfa_enabled %}checked{% endif %}
                hx-post="{% url 'enable_mfa' %}"
                hx-trigger="click throttle:1s"
                hx-swap="none">
            <label class="form-check-label" for="mfa-flex-switch" title="Multi Factor Authentication">Enable MFA</label>
        </div>
        <hr>
        B {{ forms.user|crispy }}
    
```

Enable MFA

 project_folder/users/templates/users/modals/profile/base.html </>

```
{% load custom_filters %}

<div class="modal fade" id="{{ step|replace:'_|-'}-modal" tabindex="-1"
    data-bs-backdrop="static"
    aria-labelledby="{{ step|replace:'_|-'}-label"
    aria-hidden="true">

    <div class="modal-dialog modal-dialog-centered">
        <div class="modal-content">

            <div class="modal-header">
                <h5 class="modal-title" id="{{ step|replace:'_|-'}-label">{{ title }}</h5>
                <span class="mx-auto text-secondary">{{ page }}</span>
                <button type="button" class="btn-close" onclick="onAnyClosedModal()"
                    data-bs-dismiss="modal" aria-label="Close"></button>
            </div>

            <div class="modal-body">{% block body %}{% endblock %}</div>

            <form id="{{ step|replace:'_|-'}-form" class="modal-footer"
                hx-post="{{ post_url }}"
                hx-trigger="submit throttle:200ms"
                hx-swap="none">
                {% csrf_token %}

                <div class="input-group mb-3">
                    {% block input %}{% endblock %}
                    <input name="step" type="hidden" value="{{ step }}>
                </div>
                <button type="submit" class="btn btn-primary modal-submit-button w-100"
                    hx-post="{{ post_url }}"
                    hx-trigger="click delay:1000ms"
                    hx-swap="none">

                    <span class="spinner-grow spinner-grow-sm visually-hidden"
                        role="status" aria-hidden="true"></span>
                    <span class="button-text">{{ submit }}</span>
                    <strong class="ms-1">{{ submit_boldend }}</strong>
                </span>

                </button>
            </form>
        </div>
    </div>
</div>
```

NOTE: base.html = **Bootstrap5** → Static Modal

{% load custom_filters %} Converts "my_string_name" TO "my-string-name"

{{ variable }} Orange variables are FROM include.html

onAnyClosedModal() → IS function FROM base.html

 project_folder/users/templates/users/profile.html </> profile.html

```
{% block base %}
<script name="main-script" type="text/javascript">

const input_enable_mfa = document.getElementById('mfa-flex-switch');
const onAnyClosedModal = () => input_enable_mfa.checked = false;

</script>
{% endblock %}
```

 project_folder/app_name/templatetags/custom_filters.py (app_name app) 

```
from django import template

register = template.Library()

@register.filter
def replace(value, arg):
    """Replace characters in a string. Usage: {{ string|replace:'_|-'}}"""
    old, new = arg.split('|')
    return value.replace(old, new)
```

NOTE: You may have to restart the server to register the filter: replace()

 project_folder/users/templates/users/modals/profile/include.html </>

```
<section id="modals" hx-swap-oob="innerHTML">
  {% with model_path='users/modals/profile/'|add:step|add:".html" %}
    {% include model_path with
      title=title post_url=post_url step=step page=page submit=submit
      submit_boldend=submit_boldend
    %}
  {% endwith %}
</section>
```

```
 {{ variable }} Red variables are FROM users app view: enableMFAView()
```

NOTE: This view does NOT exist. We'll create it later.

Password Modal

```
project_folder/users/templates/users/modals/profile/password.html      </>

{% extends 'users/modals/profile/base.html' %}

{% load custom_filters %}

{% block body %}
  <p id="{{ step|replace:'_|-'}-instructions">Enter your password to configure
    <strong>Multi-Factor Authentication</strong>
  </p>
  {% endblock %}

  {% block input %}
    <span class="input-group-text">Password</span>
    <input name="password" type="password" class="form-control" aria-label="one-time-
  Password" aria-describedby="{{ step|replace:'_|-'}-instructions" required>
  {% endblock %}
```

QR Code Modal

```
project_folder/users/templates/users/modals/profile/otp_qrcode.html      </>

{% extends 'users/modals/profile/base.html' %}

{% load custom_filters %}

{% load static %}

{% block body %}
  {% if qrcode_data_uri %}
    <div id="{{ step|replace:'_|-'}-instructions">
      <p>Download <a href="#">authenticator app</a> on your mobile device, and use
      it to</p>
      <h5><strong>Scan the QR code</strong></h5>
      <p>To receive the one-time password (OTP) required to activate <strong>Multi-
      Factor Authentication</strong></p>
      <hr>
      <h5><strong>Enter code</strong></h5>
      <span id="mfa-secret" class="text-primary" style="letter-spacing: 0.15em; word-
      spacing: 0.3em;"></span>
    <script type="text/javascript">...continues on the next page</script>
```

```

<script type="text/javascript">
() => {
    // Wrapped with self-invoking function for security reasons:
    // mfa_secret code must not be easily available to prevent fraud
    //
    // "DMSKAJIWESKJ"... -> "DMSK AJIW ESKJ"
    let splitText = text => text.match(/.{1,4}/g).join(' ');
    let mfaSecret = document.getElementById('mfa-secret');

    mfaSecret.innerText = splitText('{{ request.user.account.mfa_secret }}');
    mfaSecret = undefined;

    document.currentScript.remove();
})();
</script>

```

```

<script type="text/javascript">...above script</script>
<p>Into your authentication app if you're unable to scan the QR code</p>
</div>
{ % endif %}

```

```

<div class="d-flex justify-content-between">
    
    
</div>
{ % endblock %}

```

```

{ % block input %}
    <span class="input-group-text">OTP</span>
    <input name="otp_code" type="text" class="form-control" required
        aria-label="one-time-password"
        aria-describedby="{{ step|replace:'_|-'} }-instructions"
        hx-post="{{ post_url }}"
        hx-trigger="input changed delay:250ms"
        hx-swap="none">
{ % endblock %}

```

Email Modal

```
project_folder/users/templates/users/modals/profile/otp_email.html      </>

{% extends 'users/modals/profile/base.html' %}
{% load custom_filters %}
{% load static %}

{% block body %}
    <p id="{{ step|replace:'_|-'}-instructions">Enter the one-time password (OTP) sent to
your email: <br><strong>{{ request.user.email }}</strong></p>
    <hr>
    <div class="d-flex justify-content-between text-center">
        <button type="button" class="btn btn-light w-50 me-4">
            <span class="spinner-border text-secondary visually-hidden" role="status"
aria-hidden="true"></span>
            <span class="button-text">Resend OTP</span>
        </button>
        <p class="count-text text-secondary px-auto m-auto fs-2 w-25">
            <span class="count-down">0</span>
            <br><span class="fs-5">sec</span>
        </p>
        
        <script type="text/javascript">...below script</script>
    </div>
<script type="text/javascript">
() => {

    let script = document.currentScript,
        parent = script.parentElement,

        counter = parent.querySelector('.count-down'),
        countTxt = parent.querySelector('.count-text'),
        button = parent.querySelector('.btn'),

        buttonTxt = button.querySelector('.button-text'),
        spinner = button.querySelector('.spinner-border'),

        timeout = null,
        fallback = null,
        interval = null,
        isIntervalActive = false,
        count = 0;

        ...continues on the next page
}
```

```

function countDown() {
    counter.innerText = count;
    if (count > 0) count -= 1;
    else {
        clearInterval(interval);
        button.removeAttribute('disabled');
        countTxt.classList.add('text-secondary');
        isIntervalActive = false;
    }
}

button.addEventListener('click', () => {
    if (count > 0) return;
    // Request new OTP code AND send it to user's email
    // https://htmx.org/api/#ajax
    htmx.ajax('POST', '{% url "request_otp" "email" %}', {
        swap: 'none',
        headers: {
            'X-CSRFToken': '{{ csrf_token }}',
        }
    });
    timeout = setTimeout(() => {
        spinner.classList.remove('visually-hidden');
        buttonTxt.classList.add('visually-hidden');
    }, 500);

    // Set fallback execution after 15 seconds
    fallback = setTimeout(() => {
        console.warn('No response received in 15 seconds, executing fallback.');
        buttonTxt.classList.remove('visually-hidden');
        spinner.classList.add('visually-hidden');
    }, 15000);
});
}

document.addEventListener('htmx:afterRequest', async (event) => {
    if (event.detail.pathInfo.requestPath !== '{% url "request_otp" "email" %}') return;

    // Receive response from the backend

    let xhr = event.detail.xhr;
    responseText = await xhr.responseText,
    response = '';

    if (xhr.getResponseHeader('Content-Type')?.includes('application/json')) {
        response = JSON.parse(responseText);
    }
    else return;      ...continues on the next page
});

```

```

    // Cancel the timeout & fallback execution if response is received
    clearTimeout(timeout);
    clearTimeout(fallback);

    // Handle response

    if (response.success === 'OTP Sent')
    {
        if (isIntervalActive) return;
        count = 45;

        button.setAttribute('disabled', '');
        countTxt.classList.remove('text-secondary');

        isIntervalActive = true;
        if (interval) clearInterval(interval);
        interval = setInterval(countDown, 1000);

        buttonTxt.classList.remove('visually-hidden');
        spinner.classList.add('visually-hidden');
    }
    else if (response.error) {
        console.error(response.error);
    }

});

script.remove();

})();
</script>

```

```

<script type="text/javascript">...above script</script>
</div>
{%
  endblock %
}
```

```

{%
  block input %
}
<span class="input-group-text">OTP</span>
<input name="otp_code" type="text" class="form-control" required
aria-label="one-time-password"
aria-describedby="{{ step|replace:'_|- ' }}-instructions"
hx-post="{{ post_url }}"
hx-trigger="input changed delay:250ms"
hx-swap="none">
{%
  endblock %
}
```

SMS Modal

project_folder/users/templates/users/modals/profile/otp_sms.html </>

```
{% extends 'users/modals/profile/base.html' %}  
{% load custom_filters %}  
{% load static %}  
  
{% block body %}  
<p id="{{ step|replace:'-'| '-' }}-instructions">  
    Enter the one-time password (OTP) sent to your mobile via text:  
    <br>  
    <span class="phone-number d-block fs-5">  
        <strong>{{ request.user.profile.phone_number }}</strong>  
    </span>  
    <br>to finalize <strong>Multi-Factor Authentication</strong>  
</p>  
<hr>  
<div class="d-flex justify-content-between text-center">  
    <button type="button" class="btn btn-light w-50 me-4">  
        <span class="spinner-border text-secondary visually-hidden" role="status"  
            aria-hidden="true"></span>  
        <span class="button-text">Resend OTP</span>  
    </button>  
    <p class="count-text text-secondary px-auto m-auto fs-2 w-25">  
        <span class="count-down">0</span>  
        <br><span class="fs-5">sec</span>  
    </p>  
      
    <script type="text/javascript">...below script</script>
```

The <script> is similar to that of - [Email Modal](#)

Copy it and replace the following:

```
button.addEventListener('click', () => {  
    if (count > 0) return;  
    // Request new OTP code AND send it to user's email  
    // https://htmx.org/api/#ajax  
    htmx.ajax('POST', '{% url "request_otp" "email" %}', {  
        htmx.ajax('POST', '{% url "request_otp" "sms" %}', {
```

```
document.addEventListener('htmx:afterRequest', async (event) => {  
    if (event.detail.pathInfo.requestPath !== '{% url "request_otp" "email" %}') return;  
    if (event.detail.pathInfo.requestPath !== '{% url "request_otp" "sms" %}') return;
```

```
<script type="text/javascript">...</script>
</div>
{% endblock %}



---


{% block input %}
<span class="input-group-text">OTP</span>
<input name="otp_code" type="text" class="form-control" required
aria-label="one-time-password"
aria-describedby="{{ step|replace:'_|-'} }-instructions"
hx-post="{{ post_url }}"
hx-trigger="input changed delay:250ms"
hx-swap="none">
{% endblock %}
```

NOTE: You can test modals by including them in `profile.html` template using the `{% include %}` tag. However, you'd have to define/remove `{% url %}` tags FOR the `enable_mfa` and `request_otp` paths.

Refer TO [Bootstrap5 - JavaScript Modal Methods](#),
TO manually toggle included modals.

```
var myModalEl = document.querySelector('#myModal');
var modal = bootstrap.Modal.getOrCreateInstancemyModalEl);
modal.toggle();
```

Also, I recommend copying modals FROM a github repository: [django-tutorial](#)
Once we start implementing the `enableMFAView()` and `requestOTPView()`

Modals Functionality

Let's add some event listeners and functions TO:

- Toggle loading animation for the submit <button>,
- Add error highlight to the <input>,
- Hide all modals,
- Dispose modals from the DOM, and
- Clear error highlight from the modal's <input>.

 project_folder/users/templates/users/profile.html </> profile.html

```
{% block base %}

<script name="functions-for-included-modal" type="text/javascript">

const modals = document.getElementById('modals');

function hideModals() {
    document.querySelectorAll('.modal').forEach(modal => {
        const Modal = bootstrap.Modal.getOrCreateInstance(modal);
        Modal.hide();
    });
}

function setInvalidInput(input) {
    input.classList.add('is-invalid');
    input.classList.add('text-danger');
}

function setNormalInput(input) {
    input.classList.remove('is-invalid');
    input.classList.remove('text-danger');
}

function setLoadingButtonForModal(button) {
    button.querySelector('.spinner-grow').classList.remove('visually-hidden');
    button.querySelector('.button-text').innerText = 'loading...';
}

function setNormalButtonForModal(button) {
    button.querySelector('.spinner-grow').classList.add('visually-hidden');
    button.querySelector('.button-text').innerHTML = button.dataset.name;
}

</script>
<script name="main-script" type="text/javascript">

const input_enable_mfa = document.getElementById('mfa-flex-switch');
const onAnyClosedModal = () => input_enable_mfa.checked = false;

</script>
{% endblock %}
```

 project_folder/users/templates/users/modals/profile/include.html </>

```
<section id="modals" hx-swap-oob="innerHTML">
  {%- with model_path='users/modals/profile/'|add:step|add:".html" %}
    {% include model_path with
      title=title post_url=post_url step=step page=page submit=submit
      submit_boldend=submit_boldend
    %}
  {% endwith %}
  <script type="text/javascript">...below script</script>
</section>

<script type="text/javascript">
  // Global access to currently loaded modal element
  var modal = new bootstrap.Modal(
    document.currentScript.parentElement.querySelector('.modal'));

  () => {

    // https://getbootstrap.com/docs/5.0/components/modal/#methods
    // Destroy modal when it is fully closed
    modal._element.addEventListener('hidden.bs.modal', () => {
      modal._element.remove();
      modal.dispose(); // Removes Bootstrap modal_instance
    });

    // Remove inputs error-highlights ON input change AND modal exit
    let input = modal._element.querySelector('input:not([type="hidden"])');
    input.addEventListener('keydown', () =>
    {
      setNormalInput(input);
    });
    let btnClose = modal._element.querySelector('.btn-close');
    btnClose.addEventListener('click', () =>
    {
      setNormalInput(input);
    });

    // Set loading animation for button, when pressed
    let btnSubmit = document.querySelector('.modal-submit-button')
    btnSubmit.dataset.name = btnSubmit.innerHTML.trim();
    btnSubmit.addEventListener('click', () =>
    {
      setLoadingButtonForModal(btnSubmit);
    });
  };
</script>
```

...continues on the next page

```

    // Remove faded background element: .modal-backdrop
    //
    // Note: Bootstrap Models remove .modal-backdrop
    //         by themselves on modal_instance.hide() call.
    //
    // However, the Bootstrap Modal is replaced too quickly
    // for its eventListener to clear the faded element.
    //
    document.querySelectorAll('.modal-backdrop')
    ?.forEach(backdrop => backdrop.remove());
    modal.show();
    document.currentScript.remove();

})();
</script>

```

Take a moment to revisit the `modals`—having a solid grasp of HTMX will be crucial as we move into implementing the views. This final stage of the tutorial may be challenging at first, but the key is to understand how `include.html` interacts with the extended `modal` and how data flows between them.

Important Considerations:

- The HTML elements with `hx-post` attribute triggers all HTMX eventListeners.
 - To manage responses efficiently, determine which HTML element initiated the HTMX request.
 - If a request is unnecessary, handle it promptly to avoid redundant processing and improve performance.
-

Enable MFA View

```

project_folder/users/views.py                                (users app) 🐍 views.py

...
from django.http import HttpResponseBadRequest, JsonResponse
from django.urls import reverse

from .models import Profile
from .forms import UserRegisterForm, UserUpdateForm, ProfileForm, AccountForm
from .utils import (
    get_users_mfa_secret_as_qrcode_base64,
    email_otp_to_user,
    sms_otp_to_user,
)
import stripe, pyotp
...

```

```

@login_required
def enableMFAView(request):
    if request.method != 'POST':
        return HttpResponseBadRequest()
    if request.user.account.mfa_enabled:
        return JsonResponse({'error': 'MFA already enabled'}, status=400)

# Configuration
# Note: changing STEPS order affects modals-display-sequence
STEPS = ['password', 'otp_qrcode', 'otp_email', 'otp_sms']
TITLE = 'Enable MFA'
POST_URL = reverse('enable_mfa')
CONTEXT_BASE = {
    'title': TITLE,
    'post_url': POST_URL,
}

```

NOTE: **STEPS**, **TITLE** & **POST_URL** = variables passed TO the **include.html** modal

```

# Show initial modal
step = request.POST.get('step')
# If the step is invalid or it's the last step and all previous steps were completed,
# it's likely the session data is outdated, so we restart the form.
if step not in STEPS or
(not _all_steps_completed(request, STEPS) and
request.session[f'verified_{STEPS[-1]}'] == True):

    _reset_all_steps(request, STEPS)
    # If valid step and IS NOT a first step
    if step in STEPS and step != STEPS[0]:
        messages.info(request, f'Restarted the session for Enable MFA')

return _render_step_modal(request, STEPS[0], STEPS, CONTEXT_BASE)

```

NOTE: Every **modal.html** contains `<input name="step">`

```

# Validate current step
if not _validate_step(request, step):
    return JsonResponse(
        {'error': _get_validation_error(step), 'step': step}, status=400)

# Mark step as verified
request.session[f'verified_{step}'] = True

```

```
# Handle step-specific post-validation actions
_handle_post_validation(request, step)
```

NOTE: `_handle_post_validation()` performs additional actions following the successful verification of a `step`. E.g. Sending an OTP via a specific method

```
# Check if all steps are completed
if _all_steps_completed(request, STEPS):
    request.user.account.mfa_enabled = True
    request.user.account.save()
    _reset_all_steps(request, STEPS)
    return JsonResponse(
        {'success': 'Enabled MFA', 'step': step}, status=200)

# Handle final step verification failure
if step == STEPS[-1]:
    return JsonResponse(
        {'error': 'Not all steps are verified', 'step': step}, status=400)
```

Enable MFA for logged-in user IF all steps (`modal` forms) are verified.
AND reset them TO "False" IN session storage.

It's crucial to reset these steps since each verified step unlocks access to the next. Some steps contain sensitive information. For instance,

`otp_qrcode.html` exposes `mfa_secret`, which could be exploited by malicious actors to generate OTPs through an authentication app.

```
# Proceed to next step
next_step = STEPS[STEPS.index(step) + 1]
return _render_step_modal(request, next_step, STEPS, CONTEXT_BASE)
```

Take a break

```

def _reset_all_steps(request, steps):
    # Reset verification status
    for step_name in steps:
        request.session[f'verified_{step_name}'] = False



---


def _render_step_modal(request, step, all_steps, base_context):
    is_last_step = all_steps.index(step) == len(all_steps) - 1

    context = {
        **base_context,
        'step': step,
        'page': f'{all_steps.index(step) + 1}/{len(all_steps)}',
        'submit': 'Enable' if is_last_step else 'Proceed to the next step',
        'submit_boldend': 'Multi-Factor Authentication' if is_last_step else '',
    }

    if step == 'otp_qrcode':
        context['qrcode_data_uri'] = get_users_mfa_secret_as_qrcode_base64(request.user)
    return render(request, 'users/modals/profile/include.html', context)

```

Returns `include.html` modal + `context` data FOR the `{% include %}` variables

And `qrcode_data_uri` = (`mfa_secret` as QR image) IF `step` = "otp_qrcode"

```

def _validate_step(request, step):
    if step == 'password':
        password = request.POST.get('password')
        return request.user.check_password(password)
    else:
        otp = request.POST.get('otp_code')
        interval = {
            'otp_email': settings.OTP_EMAIL_INTERVAL,
            'otp_sms': settings.OTP_SMS_INTERVAL
        }.get(step, settings.OTP_DEFAULT_INTERVAL)

        totp = pyotp.TOTP(request.user.account.mfa_secret, interval=interval)
        return totp.verify(otp)

```

NOTE: Every `otp_modal.html` contains `<input name="otp_code">`

NOTE: IF `step` = "otp_email" THEN `interval` = `settings.OTP_EMAIL_INTERVAL`
IF NO MATCH FOUND for `step` THEN `interval` = `settings.OTP_DEFAULT_INTERVAL`

```

def _get_validation_error(step):
    return 'Invalid password' if step == 'password' else 'Invalid OTP'



---


def _handle_post_validation(request, step):
    if step == 'otp_qrcode':
        email_otp_to_user(request.user)
    elif step == 'otp_email':
        sms_otp_to_user(request.user)



---


def _all_steps_completed(request, steps):
    return all(request.session.get(f'verified_{step}', False) for step in steps)

```

Request OTP View

```

@login_required
def requestOTPView(request, method):
    if request.method != 'POST':
        return HttpResponseBadRequest()

    # Allow requests only from this Django project
    allowed_origins = ['http://localhost', 'http://127.0.0.1']

    origin = request.headers.get('Origin') or request.headers.get('Referer')
    if not origin or not any(origin.startswith(o) for o in allowed_origins):
        return JsonResponse({'error': 'Unauthorized request'}, status=403)

    # Send OTP to user via specified method: (email | sms)
    send_otp_via = {
        'email': email_otp_to_user,
        'sms': sms_otp_to_user
    }

    if method in send_otp_via:
        send_otp_via[method](request.user)
        return JsonResponse({'success': 'OTP Sent', 'method': method}, status=200)

    # Return invalid method error
    return JsonResponse({'error': 'Invalid method', 'method': method}, status=400)

```

Security Notice: Allowed Origins Configuration

The `allowed_origins` setting **must** be strictly enforced to prevent unauthorized access, especially in production.

- **Development:** `localhost` may be acceptable.
- **Production:** The setting **must** reflect the custom domain to safeguard authentication security and prevent misuse.

Failure to properly configure this setting in production could allow external requests to abuse third-party services such as **Zoho and Twilio**, leading to unnecessary financial costs. Given that **Twilio operates on a pay-per-request model**, unauthorized access could result in excessive or fraudulent requests, creating serious financial and operational risks.

Critical Endpoint: Request OTP View

The `requestOTPView()` function handles OTP delivery via third-party services like **Zoho and Twilio**.

- **Only this Django application** should have permission to make requests.
- **This restriction is enforced via** `allowed_origins` to ensure security.
- **In production, the setting must reflect the custom domain**, NOT `localhost`.

Important Consideration: Third-Party Authentication

Some platforms allow users to sign in using external services such as **Facebook or Google**.

- If our system supports this functionality, we may need to reassess our approach to `allowed_origins` to accommodate these third-party integrations securely.

```
project_folder/users/urls.py (users app) 🐍 urls.py

urlpatterns = [
    ...
    path('enable-mfa', views.enableMFAView, name='enable_mfa'),
    path('request-otp/<str:method>', views.requestOTPView, name='request_otp'),
]
```



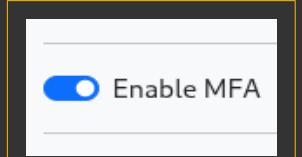
project_folder/users/templates/users/profile.html

</> profile.html

```
{% block base %}  
<script name="functions-for-included-modal" type="text/javascript">...</script>  
<script name="main-script" type="text/javascript">  
  
const input_enable_mfa = document.getElementById('mfa-flex-switch');  
const onAnyClosedModal = () => input_enable_mfa.checked = false;  
  
document.addEventListener('htmx:afterRequest', async (event) => {  
  
    let form = event.detail.target.closest('form');  
    if (!form?.classList.contains('modal-footer')) return;  
  
    // Receive response from the backend  
  
    let xhr = event.detail.xhr;  
    responseText = await xhr.responseText,  
    response = '';  
  
    if (xhr.getResponseHeader('Content-Type')?.includes('application/json')) {  
        response = JSON.parse(responseText);  
    }  
    else return;  
  
    // Handle response  
  
    if (response.error) {  
        console.error(response.error);  
    }  
    if (response.success) {  
        hideModals();  
    }  
    else if (response.error === 'Invalid OTP' || 'Invalid password') {  
        let input = form.querySelector('input:not([type="hidden"])');  
        setInvalidInput(input);  
    }  
  
    // Reset modal's submit button  
  
    let button_submit = form.querySelector('.modal-submit-button');  
    setNormalButtonForModal(button_submit);  
  
});  
  
</script>  
{% endblock %}
```

Disable MFA

project_folder/users/templates/users/profile.html </> profile.html

```
...  
<input class="form-check-input" type="checkbox" id="mfa-flex-switch" name="switch"  
      {% if user.account.mfa_enabled %}checked{% endif %}  
      hx-post="{% url 'enable_mfa' %}"  
      hx-trigger="click throttle:1s"  
      hx-swap="none">  
...  

```

project_folder/users/views.py (users app) views.py

```
@login_required  
def enableMFAView(request):  
    if request.method != 'POST':  
        return HttpResponseBadRequest()  
  
    step = request.POST.get('step')  
    switch = request.POST.get('switch')  
  
    if request.user.account.mfa_enabled:  
        return JsonResponse({'error': 'MFA already enabled'}, status=400)  
    if not request.user.account.mfa_enabled and not switch:  
        return JsonResponse({'info': 'No action required'}, status=200)  
  
    ...  
  
    # Show initial modal  
    step = request.POST.get('step')  
    # --- Show initial modal TO start verifying steps TO Enable MFA ---  
  
    # If the step is invalid or it's the last step and all previous steps were completed,  
    # it's likely the session data is outdated, so we restart the form.  
    if step not in STEPS or  
        ...
```

```

... CONTEXT_BASE

# --- Prompt logged-in user TO Disable MFA ---

if request.user.account.mfa_enabled and not switch:

    if step == 'password':
        # Validate password
        if not _validate_step(request, step):
            return JsonResponse(
                {'error': _get_validation_error(step), 'step': step}, status=400)
        else:
            request.user.account.mfa_enabled = False
            request.user.account.save()
            return JsonResponse(
                {'success': 'Disabled MFA'}, status=200)

    _reset_all_steps(request, STEPS)
    return render(request, 'users/modals/profile/include.html', {
        'title': 'Disable MFA',
        'post_url': reverse('enable_mfa'),
        'step': 'password',
        'submit': 'Disable',
        'submit_boldend': 'Multi-Factor Authentication',
    })
}

# --- Show initial modal TO start verifying steps TO Enable MFA ---

```

NOTE: `password.html` modal includes instructions for **enabling** MFA. However, in this case, we are **disabling** the MFA. Let's update this modal specifically for **disabling** MFA

 project_folder/users/templates/users/modals/profile/password.html </>

```

{% extends 'users/modals/profile/base.html' %}
{% load custom_filters %}

{% block body %}
    <p id="{{ step|replace:'_|-'}-instructions">Enter your password to {{ if submit == 'Disable' }}deactivate{{ else }}configure{{ endif %}}
        <strong>Multi-Factor Authentication</strong>
    </p>
{% endblock %}

{% block input %}
    <span class="input-group-text">Password</span>
    <input name="password" type="password" class="form-control" aria-label="one-time-Password" aria-describedby="{{ step|replace:'_|-'}-instructions" required>
{% endblock %}

```

NOTE: `base.html` modal has an issue.

When the client closes the modal, we assume that the `step` was unverified.
Hence, the `switch` `<input id="mfa-flex-switch">` should be switched;

- **ON** when **MFA is enabled**
- **OFF** when **MFA is disabled**
after the `modal` is `closed / terminated`.

However, function: `onAnyClosedModal()` is triggered from `profile.html` template.
Always turning the `switch` **ON**. Let's update this function, and spice up the submit button.

```
project_folder/users/templates/users/modals/profile/base.html </>

...
<div class="modal-header">
  <h5 class="modal-title" id="{{ step|replace:'_|-'}-label}>{{ title }}</h5>
  <span class="mx-auto text-secondary">{{ page }}</span>
  <button type="button" class="btn-close" onclick="onAnyClosedModal('{{ submit }}')"
    data-bs-dismiss="modal" aria-label="Close"></button>
</div>
...

<button type="submit" class="btn
btn-{{ if submit == 'Disable' %}danger{{ else %}primary{{ endif %}}
modal-submit-button w-100"
  hx-post="{{ post_url }}"
  hx-trigger="click delay:1000ms"
  hx-swap="none">
...
</button>
...
```

```
project_folder/users/templates/users/profile.html </> profile.html

{% block base %}
<script name="functions-for-included-modal" type="text/javascript">...</script>
<script name="main-script" type="text/javascript">

const input_enable_mfa = document.getElementById('mfa-flex-switch');
const onAnyClosedModal = () => input_enable_mfa.checked = false;
const onAnyClosedModal = submit => input_enable_mfa.checked = submit == 'Disable';
```

NOTE: Client cannot make spam requests via the **switch** `<input id="mfa-flex-switch">`

However, a client can spam the toggle animation. Desynchronizing the request and visual feedback of the switch itself. The switch can also bug-out, displaying invalid MFA state.

Let's fix this by throttling the toggle animation of the **switch** element.

```
project_folder/users/templates/users/profile.html      </> profile.html

{% block base %}
<script name="functions-for-included-modal" type="text/javascript">...</script>
<script name="main-script" type="text/javascript">

function throttleElement(element, type='click', delay=500) {
    element.dataset.throttle = false;
    element.addEventListener(type, event => {
        if (element.dataset.throttle == 'true') {
            event.preventDefault(); return;
        }
        element.dataset.throttle = true;
        setTimeout(() => element.dataset.throttle = false, delay);
    });
}

const input_enable_mfa = document.getElementById('mfa-flex-switch');
const onAnyClosedModal = submit => input_enable_mfa.checked = submit == 'Disable';
throttleElement(input_enable_mfa, 'click', parseInt(
    input_enable_mfa.getAttribute('hx-trigger')
    .match(/throttle:(\d+(s|ms))/)[1]
    .replaceAll('ms', '')
    .replaceAll('s', '000'))
));
```

NOTE: This script retrieves the **delay** of **throttle** argument of the **switch** element `<input id="mfa-flex-switch">` specified in the **hx-trigger** attribute

This approach improves developers' experience when updating the **throttle**

Authenticate users with MFA