

Workshop: Petstagram

This document contains the fifth and final part of the Petstagram Workshop. First, we will add a functionality to send each newly registered user a **greeting email**. Then, we will add **pagination on the profile details page**. Next, we will **test the app**. And finally, we will **deploy using Docker and Amazon**.

The full project description can be found in the [Workshop Description Document](#).

You can directly dive into the app here: <https://softuni-petstagram.azurewebsites.net/>

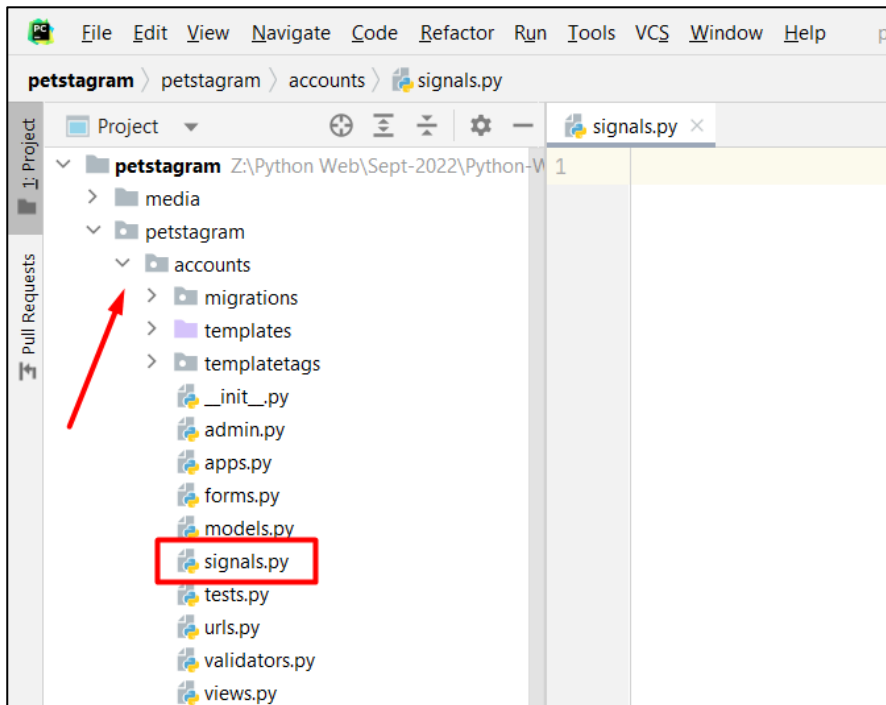
1. Workshop - Part 5.1

Send "Registration greetings" Email

When **new user is registered** on the site, we want them to receive an **automatic greetings email**. We have the **template in this workshop resources**. It is an email template and we should **only change the username** (of the receiver) on it:



To implement the functionality we can **use a signal**. Let us add a **signals.py** file in our **accounts** app:



We want to **invoke the signal just after the profile is saved**. The signal checks if the **user is created**, and if so - **creates the email with the given template and sends it to them**:

```
from django.core.mail import send_mail
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.template.loader import render_to_string
from django.utils.html import strip_tags

from petstagram import settings
from petstagram.accounts.models import PetstagramUser

@receiver(signal=post_save, sender=PetstagramUser)
def send_greeting_email(sender, instance, created, **kwargs):
    if created:
        subject = "Registration greetings"
        html_message = render_to_string('email-greeting.html', {'profile': instance})
        plain_message = strip_tags(html_message)
        to = instance.email
        send_mail(
            subject=subject,
            message=plain_message,
            from_email=settings.EMAIL_HOST_USER,
            recipient_list=[to], html_message=html_message
        )
```

In the **setting.py** file, we should add the **email credentials**. In this workshop, we will be using **Gmail account**. To use it we need to set that **the email uses TLS**, the host is "**smtp.gmail.com**", the Gmail port SMTP port is **587**, the email host user is **our email address**, the password is an **automatically generated password by Gmail** when we add

a "third-party apps with account access":

```
settings.py
128 # https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field
129
130 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
131
132 MEDIA_URL = 'media/'
133 MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
134
135 AUTH_USER_MODEL = 'accounts.PetstagramUser'
136
137 EMAIL_USE_TLS = True
138 EMAIL_HOST = 'smtp.gmail.com'
139 EMAIL_HOST_USER = 'no.reply.petstagram@gmail.com'
140 EMAIL_HOST_PASSWORD = "gghidvelyurzdxcfvghbn"
141 EMAIL_PORT = 587
```

Add Pagination

The next thing we want to add to our app is a pagination on the profile details page. We want to **show only 2 images per page**. Open the `account/views.py` file and we can **add the pagination** to our context:

```
views.py
1 from django.core.paginator import Paginator
2 from django.urls import reverse_lazy
3 from django.views import generic as views
4 from django.contrib.auth import views as auth_views
5
6 from petstagram.accounts.forms import PetstagramUserCreateForm, LoginForm, PetstagramUserEditForm
7 from petstagram.accounts.models import PetstagramUser
8
9
10 class UserDetailsView(views.DetailView):
11     model = PetstagramUser
12     template_name = 'accounts/profile-details-page.html'
13
14     def get_context_data(self, **kwargs):
15         context = super().get_context_data(**kwargs)
16         total_likes_count = sum(p.like_set.count() for p in self.object.photo_set.all())
17         photos = self.object.photo_set.all()
18         paginator = Paginator(photos, 2)
19         page_number = self.request.GET.get('page') or 1
20         page_obj = paginator.get_page(page_number)
21
22         context.update({
23             'total_likes_count': total_likes_count,
24             'paginator': paginator,
25             'page_number': page_number,
26             'page_obj': page_obj,
27         })
28
29     return context
```

Now, let us open the **profile-page-page.html** template and add the pagination code:

```
...
<!-- Start Last Uploaded Photos Section -->
<div class="pet-photos">

    <!-- Link to Last Uploaded Pet Photo -->
    {% for photo in page_obj.object_list %}
        <a href="{% url 'photo-details' photo.id %}">
            <!-- Pet Photo -->
            
        </a>
    {% endfor %}

    <div class="pagination">
        <span class="step-links">
            {% if page_obj.has_previous %}
                <a href="?page=1">&laquo; first</a>
                <a href="?page={{ page_obj.previous_page_number }}">previous</a>
            {% endif %}

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

            {% if page_obj.has_next %}
                <a href="?page={{ page_obj.next_page_number }}">next</a>
                <a href="?page={{ page_obj.paginator.num_pages }}">last
                &raquo;</a>
            {% endif %}
        </span>
    </div>
</div>
```

2. Workshop - Part 5.2

Testing

Test the forms, models and views in the **petstagram/photos** app (but not any libraries or functionality provided as part of Python or Django).

3. Workshop - Part 5.3

Deploy

Deploy the project using AWS and Docker.