BackEnd Workshop-4

Clarusway

Subject: Django project with a pizza ordering website.

Learning Goals

- Practice Django forms,
- Using submitted data,
- Adding models
- Using the ModelForms class
- Working with widgets
- Accepting files and multiple forms on a page
- Customizing formsets
- Using local validation
- Delivering errors responsibly
- Styling with CSS

Introduction

In this workshop, we will create a Django app named "Nandia's Garden" which will let users to order pizza.

Most websites require the use of forms to receive data from users, so it is crucial to know how to safely collect and handle data while maintaining a user-friendly experience on your website. In this course, learn how to use Django to create forms.

See the details of the project:

Nadia's Garden Webpage

Models:

In this project, you need to create a Size model, and a Pizza model with fields:

- Size:
    - title
- Pizza:
    - topping1
    - topping2
    - size (ForeignKey)

Forms :

In this project, you need to create a PizzaForm using Pizza model. And also, you need to create a MultiplePizzaForm to order mored than one pizza at a time.

Views:

In this project, you need to create four views:

- home
- order
- pizzas
- edit_order

Templates:

In this project, you need to create four templates:

- home.html
- order.html
- pizzas.html
- edit_order.html

And one optional template for base.html.

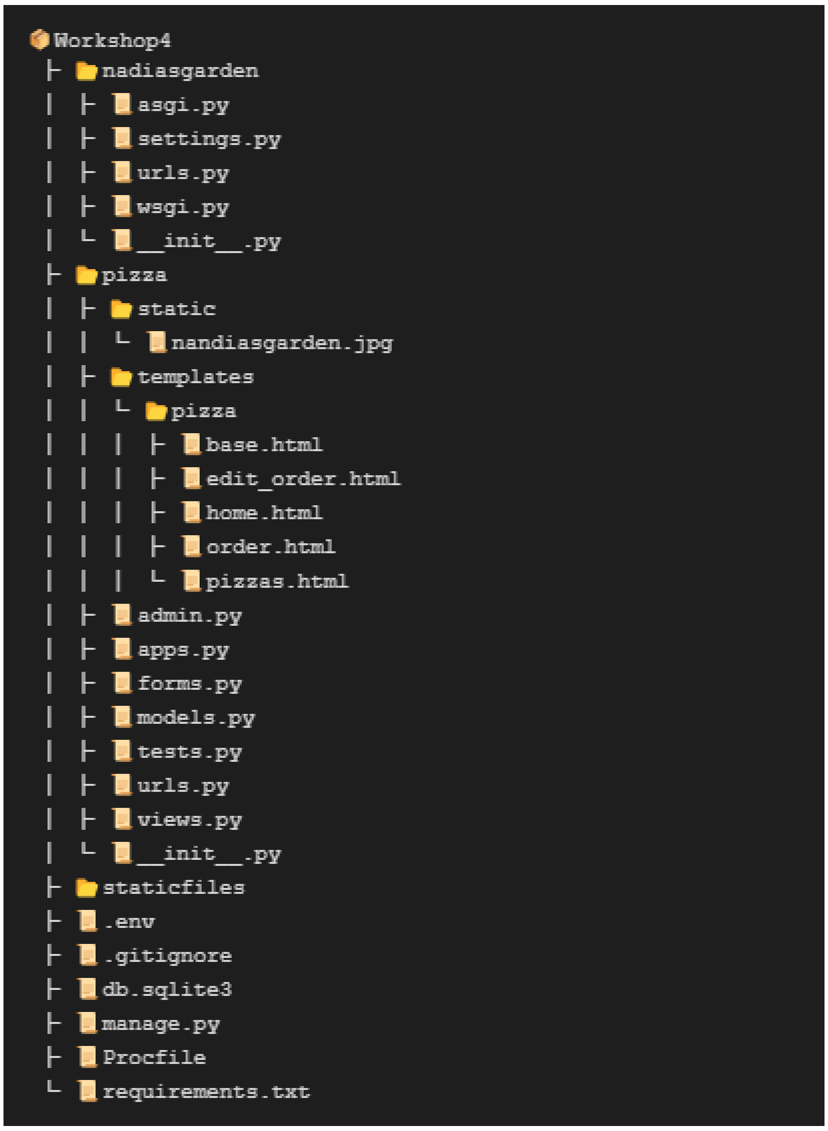
URLs:

In this project, you need to create four urls:

- '' : root path will return home page.
- 'order/' : page to order pizza, showing pizza form.
- 'order/<int:pk>/' : editing order.
- 'pizzas/' : to order more than one pizza.

Project Folder Structure

At the end of the project, the folder sturcture will be like:



```
Workshop4
├ nadiasgarden
| ├ asgi.py
| ├ settings.py
| ├ urls.py
| ├ wsgi.py
| └ __init__.py
├ pizza
| ├ static
| | └ nandiasgarden.jpg
| ├ templates
| | └ pizza
| | | ├ base.html
| | | ├ edit_order.html
| | | ├ home.html
| | | ├ order.html
| | | └ pizzas.html
| ├ admin.py
| ├ apps.py
| ├ forms.py
| ├ models.py
| ├ tests.py
| ├ urls.py
| ├ views.py
| └ __init__.py
├ staticfiles
├ .env
├ .gitignore
├ db.sqlite3
├ manage.py
├ Procfile
└ requirements.txt
```

Code:

Part 1 - Create project and app

1. Create a working directory, with a meaningfull name, and cd to new directory.

2. Create virtual environment as a best practice:

```
python3 -m venv env # for Windows or
python -m venv env # for Windows
virtualenv env # for Mac/Linux or;
virtualenv env -p python3 # for Mac/Linux
```

3. Activate scripts:

```
.\env\Scripts\activate  # for Windows, may need to switch powershell on this
operation.
source env/bin/activate  # for MAC/Linux
```

See the (env) sign before your command prompt.

4. Install django:

```
pip install django  # or
py -m pip install django
```

5. (Optional) See installed python packages:

```
pip freeze

# you will see:

# asgiref==3.5.0
# Django==4.0.4
# sqlparse==0.4.2
# tzdata==2022.1

# If you see lots of things here, that means there is a problem with your virtual
env activation. Activate scripts again!
```

6. Create the requirements.txt on your working directory, send your installed packages to this file, requirements file must be up to date:

```
pip freeze > requirements.txt
# In this project we will not use this file. But this is a standard procedure to
learn.
```

7. Create project:

```
django-admin startproject nadiasgarden .
# With . it creates a single project folder.
# Avoiding nested folders.

django-admin startproject nadiasgarden
# Without . at the end it will create nested folders.
# Another naming as "main":
django-admin startproject main .

# Naming depends on the company, team and the project.
```

Various files has been created inside project folder "nadiasgarden"!

- manage.py - A command-line utility that allows you to interact with your Django project
- __init__.py - An empty file that tells Python that the current directory should be considered as a Python package
- settings.py - Comprises the configurations of the current project like DB connections.
- urls.py - All the URLs of the project are present here
- wsgi.py - This is an entry point for your application which is used by the web servers to serve the project you have created.

8. (Optional) If you created your project wihthout "." at the end, change the name of the project main directory as src to distinguish from subfolder with the same name:

```
# Be careful to use your own folder names.
mv nadiasgarden src
```

9. Let's create the application:

Go to the same level with manage.py file if you create your project with nested folders at step 7:

```
cd nadiasgarden  # or, if you changed;
cd src
```

Start app

```
# Using app name as "pizza"
python manage.py startapp pizza  # or
py -m manage.py startapp pizza

# Another naming:
python manage.py startapp home
py -m manage.py startapp home
```

10. Go to settings.py and add another line as below under INSTALLED_APPS:

```
'pizza',
```

11. Run your project:

```
python manage.py runserver  # or
py -m manage.py runserver
```

Go to http://localhost:8000/ in your browser, and you should see Django rocket, which means you successfully created project.

The install worked successfully! Congratulations!

Part 2 - Create Models

1. Go to models.py and create Size and Pizza models like below:

```
from django.db import models

class Size(models.Model):
    title = models.CharField(max_length=100)

    def __str__(self):
        return self.title  # This is for good visual experimentation!

class Pizza(models.Model):
    topping1 = models.CharField(max_length=100)
    topping2 = models.CharField(max_length=100)
    size = models.ForeignKey(Size, on_delete=models.CASCADE)  # This is for
correlation to the Size class

    def __str__(self):
        return self.size, self.topping1, self.topping2
```

2. See how to use ForeignKey.
3. Discuss the differences between TextField and CharField.
4. Manage migrations:

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

Part 3 - Create Forms

1. Create forms.py under pizza app and create your forms like below:

```
from django import forms
from .models import Pizza, Size


class PizzaForm(forms.ModelForm):

    class Meta:
        model = Pizza
        fields = ['topping1', 'topping2', 'size']
        labels = {'topping1':'Topping 1', 'topping2':'Topping 2'}

class MultiplePizzaForm(forms.Form):
    number = forms.IntegerField(min_value=2, max_value=6)
```

Part 4 - Create Views and URLs

1. Go to views.py under "pizza" directory, and create your views like below:

```
from django.shortcuts import render
from .forms import PizzaForm, MultiplePizzaForm  # referring to newly created
forms.py and our new PizzaForm
from django.forms import formset_factory
from .models import Pizza
from django.contrib import messages

def home(request):
    return render(request, 'pizza/home.html')

def order(request):
    multiple_form = MultiplePizzaForm()
    if request.method == 'POST':
        # filled_form = PizzaForm(request.POST, request.FILES)
        filled_form = PizzaForm(request.POST)
        if filled_form.is_valid():
            created_pizza = filled_form.save()
            created_pizza_pk = created_pizza.id

            size = filled_form.cleaned_data.get('size')
            topping1 = filled_form.cleaned_data.get('topping1')
            topping2 = filled_form.cleaned_data.get('topping2')

            messages.success(request, f'Thanks for ordering! Your {size},
{topping1} and {topping2} pizza is on its way!')

            filled_form = PizzaForm()
            # return render(request, 'pizza/order.html',
{'created_pizza_pk':created_pizza_pk, 'pizzaform':new_form, 'note':note,
'multiple_form':multiple_form})
        else:
            created_pizza_pk = None
            messages.warning(request, 'Pizza order failded, try again!')

        return render(request, 'pizza/order.html',
{'created_pizza_pk':created_pizza_pk, 'pizzaform':filled_form,
'multiple_form':multiple_form})
    else:
        form = PizzaForm()
        return render(request, 'pizza/order.html', {'pizzaform':form,
'multiple_form':multiple_form})

def pizzas(request):
    number_of_pizzas = 2
    filled_multiple_pizza_form = MultiplePizzaForm(request.GET)
    if filled_multiple_pizza_form.is_valid():
        number_of_pizzas = filled_multiple_pizza_form.cleaned_data.get('number')
    PizzaFormSet = formset_factory(PizzaForm, extra=number_of_pizzas)
    formset = PizzaFormSet()
    if request.method == "POST":
        filled_formset = PizzaFormSet(request.POST)
        if filled_formset.is_valid():
            for form in filled_formset:
                form.save()
            messages.success(request, 'Pizzas have been ordered!')

        else:
            messages.warning(request, 'Order was not created, please try again')


        return render(request, 'pizza/pizzas.html', {'formset':formset})
    else:
        return render(request, 'pizza/pizzas.html', {'formset':formset})

def edit_order(request, pk):
    pizza = Pizza.objects.get(pk=pk)
    form = PizzaForm(instance=pizza)
    if request.method == 'POST':
        filled_form = PizzaForm(request.POST,instance=pizza)
        if filled_form.is_valid():
            filled_form.save()
            form = filled_form
            messages.success(request, 'Order has been updated.')

            return render(request, 'pizza/edit_order.html',
{'pizzaform':form,'pizza':pizza})
    return render(request, 'pizza/edit_order.html',
{'pizzaform':form,'pizza':pizza})
```

2. Discuss using formsets.

3. Include URL path of the new app to the project url list, go to urls.py and add:

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('pizza.urls'))
]
```

4. Add urls.py file under pizza directory and add:

```
from django.urls import path
from .views import home, order, pizzas, edit_order

urlpatterns = [
    path('', home, name='home'),  # This is for home page
    path('order/', order, name='order'),  # This is for ordering page
    path('pizzas', pizzas, name='pizzas'),
    path('order/<int:pk>', edit_order, name='edit_order'),
]
```

Part 5 - Create Templates

1. Create templates/pizza folder and under that folder, create base.html file as a best practice. Create this base template including bootstrap like below:

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css"
integrity="sha384-
GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMp0DgiMDm4iYMj70gZWKYbI706tWS"
crossorigin="anonymous">
```

```
    <title>Nandia's Garden</title>
  </head>
  <nav class="navbar navbar-expand-lg navbar-dark" style="background-color:
#238a44">
    <div class="container">
      <a class="navbar-brand" href="{% url 'home' %}">Nandia's Garden</a>
      <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav">
          <li class="nav-item active">
            <a class="nav-link" href="{% url 'order' %}">Order Pizza</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>

  <body>

    {% block body %}
    {% endblock %}


    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.6/umd/popper.min.js"
integrity="sha384-
wHAiFfRlMFy6i5SRaxvfOCifBUQy1xHdJ/yoi7FRNXMRBu5WHdZYu1hA6ZOblgut"
crossorigin="anonymous"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/js/bootstrap.min.js"
integrity="sha384-
B0UglyR+jN6CkvvICOB2joaf5I4l3gm9GU6Hc1og6Ls7i6U/mkkaduKaBhlAXv9k"
crossorigin="anonymous"></script>
  </body>
</html>
```

2. Under templates/pizza folder create home.html as the second template, this will serve home page:

```
{% extends 'pizza/base.html' %}

{% block body %}
    {% load static %}
    <img style="background-size: cover;" src="{% static 'nandiasgarden.jpg' %}"
class="img-fluid" alt="Nandia's Garden">
{% endblock body %}
```

3. Under templates/pizza folder create order.html as the third template, this will enable users to order pizza:

```
{% extends 'pizza/base.html' %}
{% block body %}

<div class="container">
    <h1>Order pizza</h1>

    <div class="messages">
        {% if messages %}
            <div class="">
                {% for message in messages %}
                <p{% if message.tags %} class="text-center alert alert-{{
message.tags }}" {% endif %}>
                {% if message.level == DEFAULT_MESSAGE_LEVELS.ERROR %}Important:{%
endif %}

                {{ message }}
                </p>
                {% endfor %}
                </ul>
            </div>
        {% endif %}
    </div>

    <div style="text-align: center">
        {% if created_pizza_pk %}
        <button><a href="{% url 'edit_order' created_pizza_pk %}">Edit Your
Order</a></button><br>
        {% endif %}
    </div>

    <div>
        <form action="{% url 'order' %}" method="POST">
            {% csrf_token %}
            {{ pizzaform.as_p }}
            <input type="submit" value="Order Pizza">
        </form>

        <br><br>

        Want more than one pizza?

        <form action="{% url 'pizzas' %}", method="GET">
            {{ multiple_form.as_p }}
            <input type="submit" value="Get Pizzas">
        </form>
    </div>
</div>
{% endblock %}
```

4. Under templates/pizza folder create edit_order.html as the third template, this will enable users to edit their orders:

```
{% extends 'pizza/base.html' %}

{% block body %}
<div class="container">

    <div class="messages">
        {% if messages %}
            <div class="">
                {% for message in messages %}
                <p{% if message.tags %} class="text-center alert alert-{{
message.tags }}" {% endif %}>
                {% if message.level == DEFAULT_MESSAGE_LEVELS.ERROR %}Important:{%
endif %}

                {{ message }}
                </p>
                {% endfor %}
                </ul>
            </div>
        {% endif %}
      </div>

    <form action="{% url 'edit_order' pizza.id %}" method="post">
        {% csrf_token %}
        {{ pizzaform.as_p }}
        <input type="submit" value="Edit Order">
    </form>
</div>
{% endblock body %}
```

5. Under templates/pizza folder create pizzas.html as the third template, this will enable users to order more than one pizza at the same time:

```
{% extends 'pizza/base.html' %}


{% block body %}

<div class="container">

  <h1>Order Pizzas</h1>
```

```
    <div class="messages">
      {% if messages %}
          <div class="">
              {% for message in messages %}
              <p{% if message.tags %} class="text-center alert alert-{{ message.tags
}}" {% endif %}>
              {% if message.level == DEFAULT_MESSAGE_LEVELS.ERROR %}Important:{%
endif %}

              {{ message }}
              </p>
              {% endfor %}
              </ul>
          </div>
      {% endif %}
    </div>


    <form action="{% url 'pizzas' %}" method="POST">
      {% csrf_token %}
        {{ formset.management_form }}

        {% for form in formset %}
          {{ form }}
          <br><br>
        {% endfor %}
        <input type="submit" value="Order Pizzas" />
      </form>
</div>

{% endblock body %}
```

6. Run our project again to see our view:

```
python manage.py runserver
py -m manage.py runserver
```

7. Go to http://localhost:8000/ in your browser, and check the functionality of your website. To stop the server use "CTRL + C"

8. If you want to send this project to your Github repo, do not forget to add .gitignore file, and secure your sensitive information like keys storing them locally in .env file and using python-decouple module.

9. Update the requirements.txt file after all your installations!
10. (Optional) Deploy your project to Heroku.

☺ Thanks for Attending ✍

Clarusway