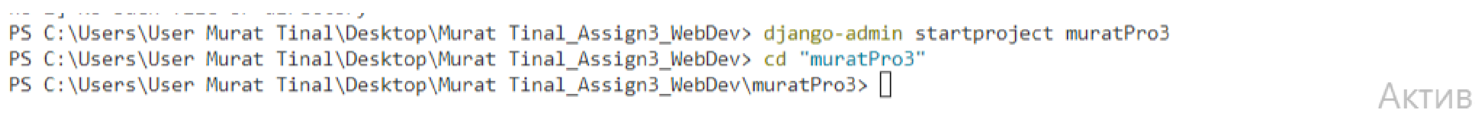
Assignment 3, web app dev

Murat Tinal

23MD0442

07.11.24

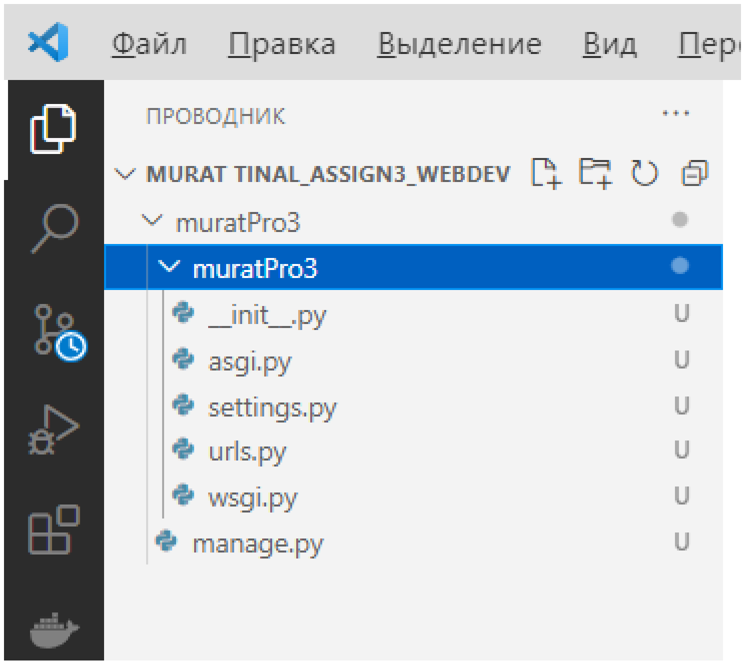
1. Django Models



```
PS C:\Users\User Murat Tinal\Desktop\Murat Tinal_Assign3_WebDev> django-admin startproject muratPro3
PS C:\Users\User Murat Tinal\Desktop\Murat Tinal_Assign3_WebDev> cd "muratPro3"
PS C:\Users\User Murat Tinal\Desktop\Murat Tinal_Assign3_WebDev\muratPro3> 
```

Fig 1.

In *Fig 1*, I started a new Django project called “muratPro3”. Then, I moved into its directory.



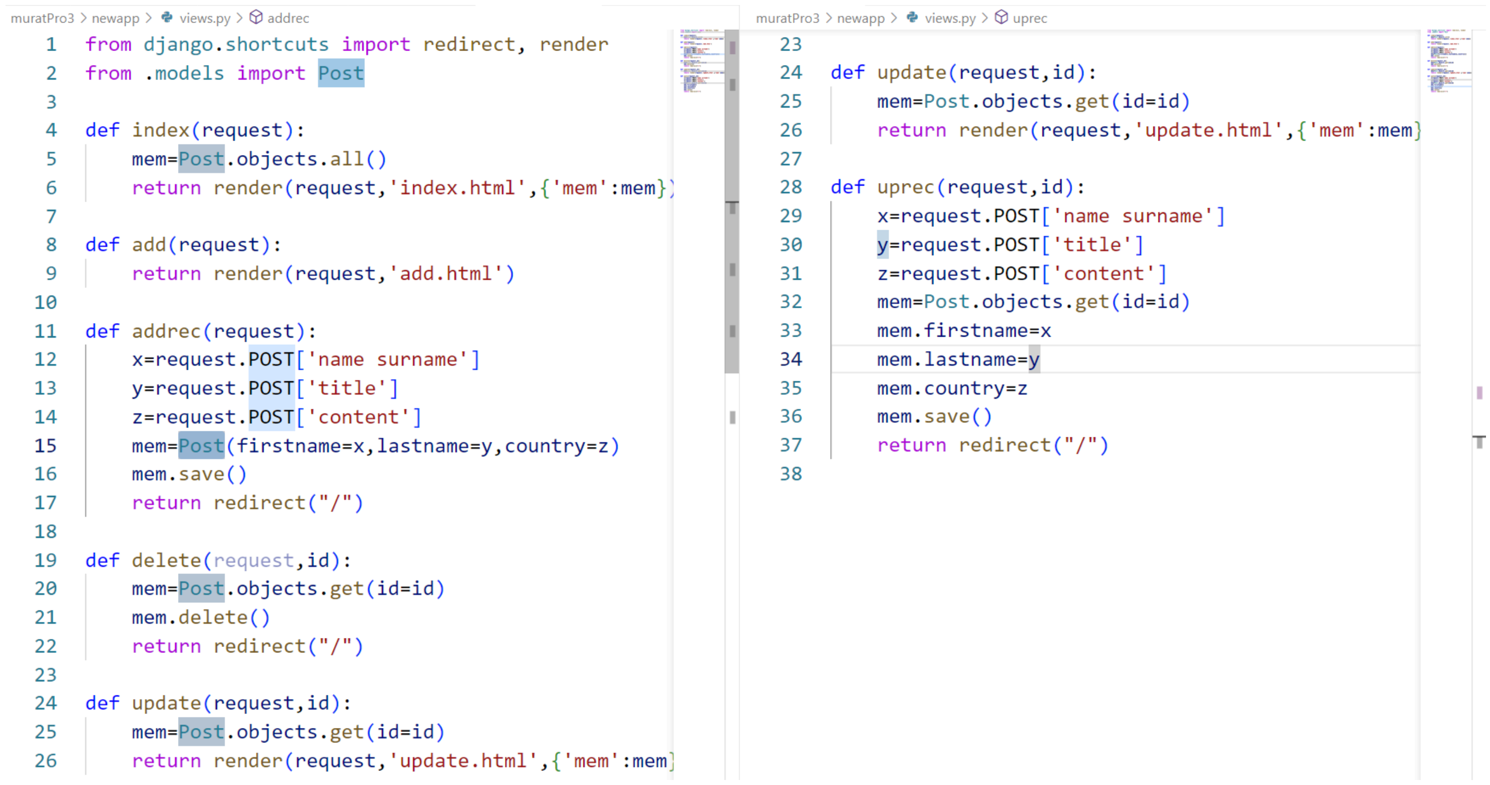
```
ПРОВОДНИК
∨ MURAT TINAL_ASSIGN3_WEBDEV
  ∨ muratPro3
    ∨ muratPro3
        __init__.py      U
        asgi.py          U
        settings.py      U
        urls.py          U
        wsgi.py          U
      manage.py          U
```

Fig 2.

Here in *Fig 2*, you can see folders in the directory "muratPro3" that were created for the project. These folders include settings and configurations for my Django project.

In Fig 3, I imported “redirect” and “render” to handle page navigation and display. I also imported the “Member” model to work with member data. I created an “index” function to show all members by retrieving them from the database and sending them to the "index.html" template. The “add” function opens the “add.html” template where I can add a new member’s details. In the “addrec” function, I retrieve form data, create a new “Member” object, save it to the database, and then redirect to the homepage. I wrote a “delete” function to find a member by ID, delete them from the database, and redirect to the homepage. The “update” function finds a member by ID and displays their info in the "update.html" template for editing. Finally, the “uprec” function updates the member's details with the new form data and saves it, then redirects to the homepage.

2. Django Views



muratPro3 > newapp > views.py > addrec

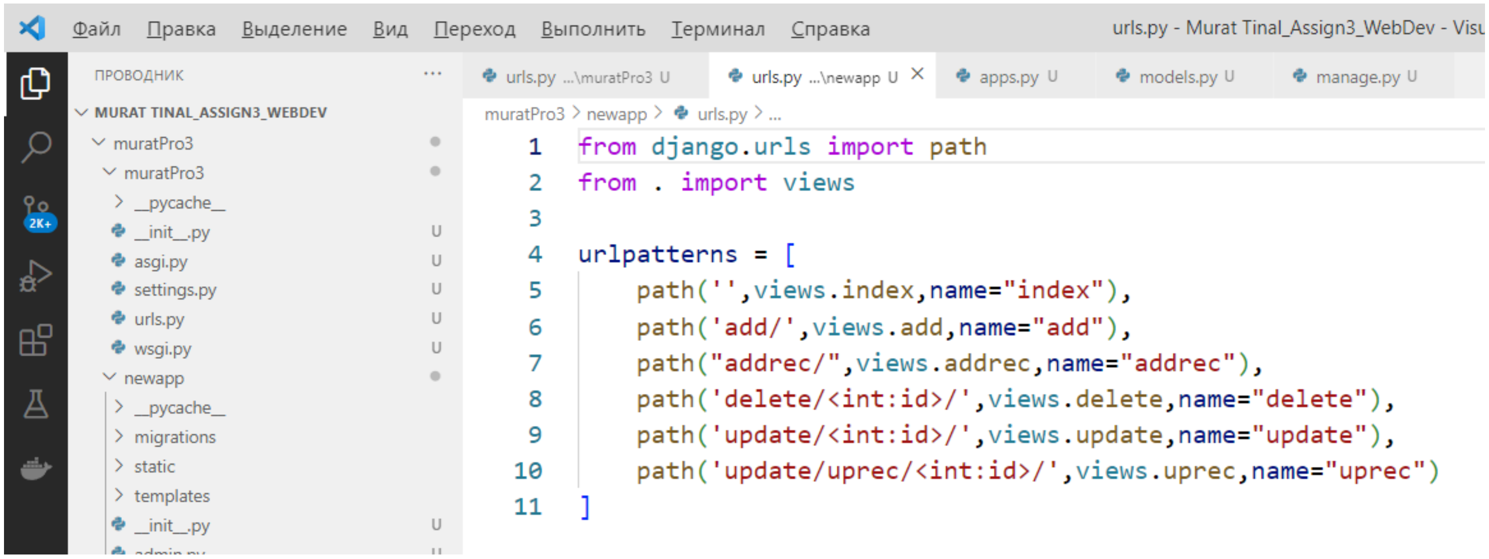
```python
from django.shortcuts import redirect, render
from .models import Post

def index(request):
    mem=Post.objects.all()
    return render(request,'index.html',{'mem':mem})

def add(request):
    return render(request,'add.html')

def addrec(request):
    x=request.POST['name surname']
    y=request.POST['title']
    z=request.POST['content']
    mem=Post(firstname=x,lastname=y,country=z)
    mem.save()
    return redirect("/")

def delete(request,id):
    mem=Post.objects.get(id=id)
    mem.delete()
    return redirect("/")

def update(request,id):
    mem=Post.objects.get(id=id)
    return render(request,'update.html',{'mem':mem}
```

muratPro3 > newapp > views.py > uprec

```python
def update(request,id):
    mem=Post.objects.get(id=id)
    return render(request,'update.html',{'mem':mem}

def uprec(request,id):
    x=request.POST['name surname']
    y=request.POST['title']
    z=request.POST['content']
    mem=Post.objects.get(id=id)
    mem.firstname=x
    mem.lastname=y
    mem.country=z
    mem.save()
    return redirect("/")
```

Fig 3.

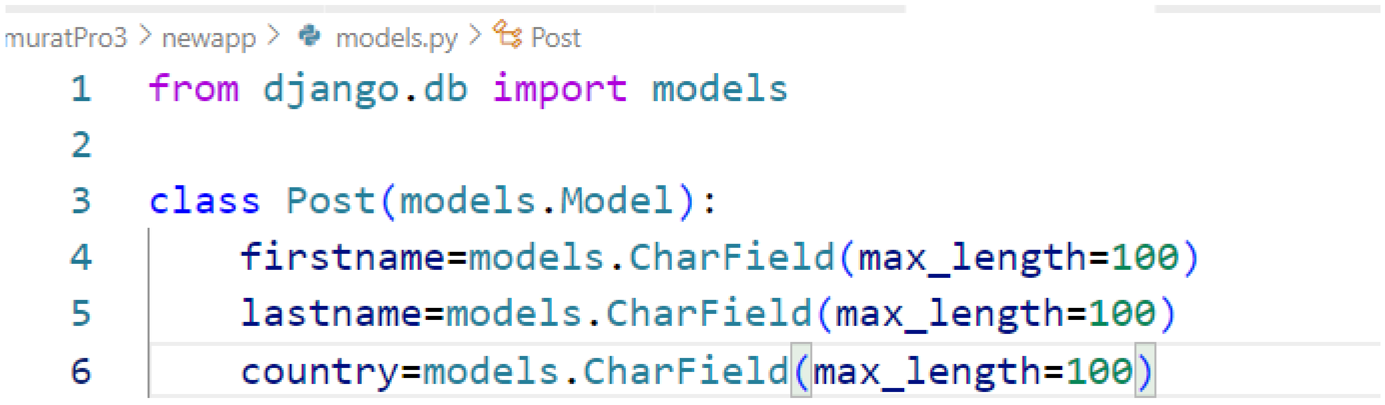
In Fig 4, I imported “path” from “django.url”` and the “views” module. I defined URL patterns for different pages in the app. The routes include paths for displaying members, and adding, deleting, and updating records. Each path is linked to a specific view function in “views.py”.



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

urlpatterns = [
    path('',views.index,name="index"),
    path('add/',views.add,name="add"),
    path("addrec/",views.addrec,name="addrec"),
    path('delete/<int:id>/',views.delete,name="delete"),
    path('update/<int:id>/',views.update,name="update"),
    path('update/uprec/<int:id>/',views.uprec,name="uprec")
]
```

Fig 4.

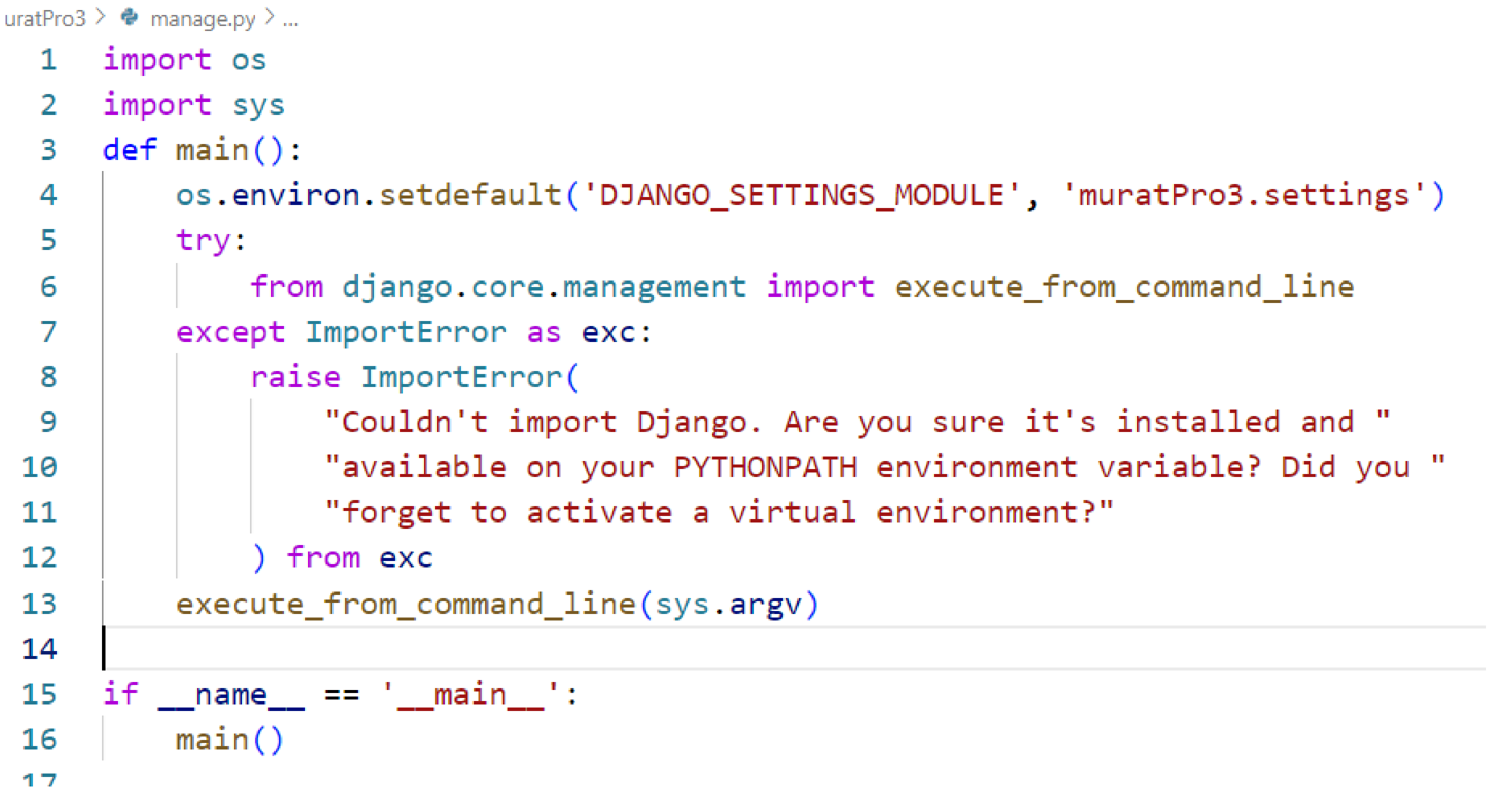
.


muratPro3 > newapp > models.py > Post

```python
from django.db import models

class Post(models.Model):
    firstname=models.CharField(max_length=100)
    lastname=models.CharField(max_length=100)
    country=models.CharField(max_length=100)
```

Fig 5.

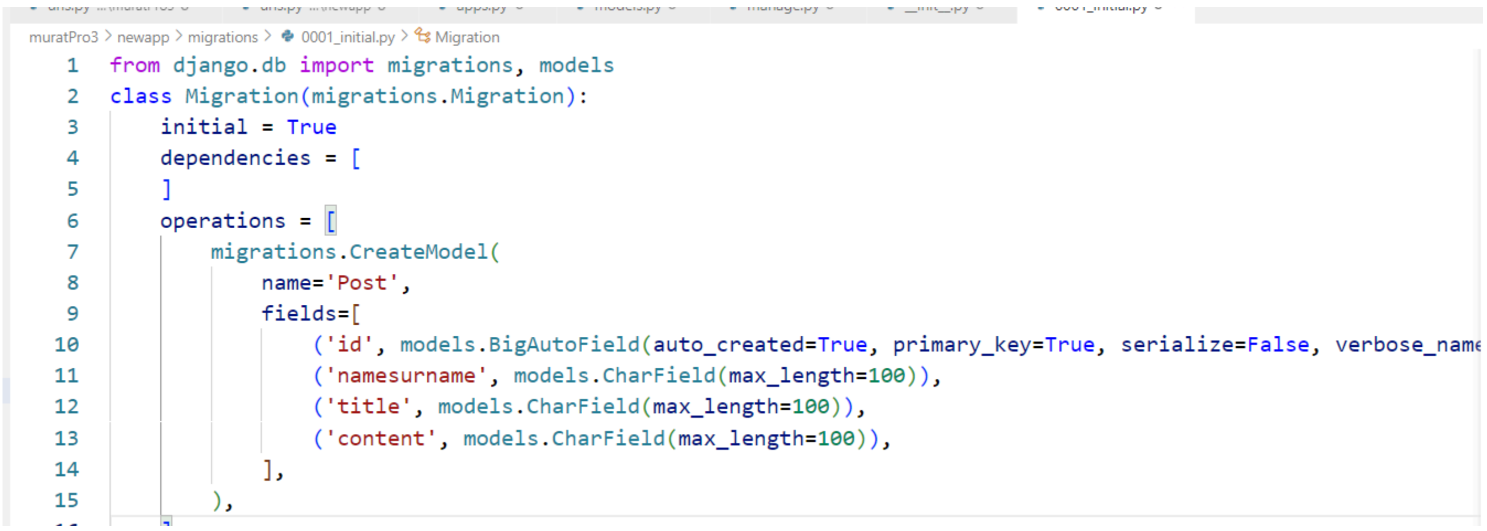
In Fig 5, I created a “Post” model with three fields: “namesurname”, “title”, and “content”. Each field is a “CharField” with a maximum length of 100 characters. This model will be used to store member information in the database.


uratPro3 > manage.py > ...

```python
import os
import sys
def main():
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'muratPro3.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()
```

Fig 6.

Fig 6 sets the environment variable “DJANGO_SETTINGS_MODULE” to point to the “muratPro3.settings” file. If Django is not found, it raises an error and suggests checking if it's installed and if the virtual environment is activated. Finally, it executes Django's command-line utility using “execute_from_command_line” with the provided arguments.

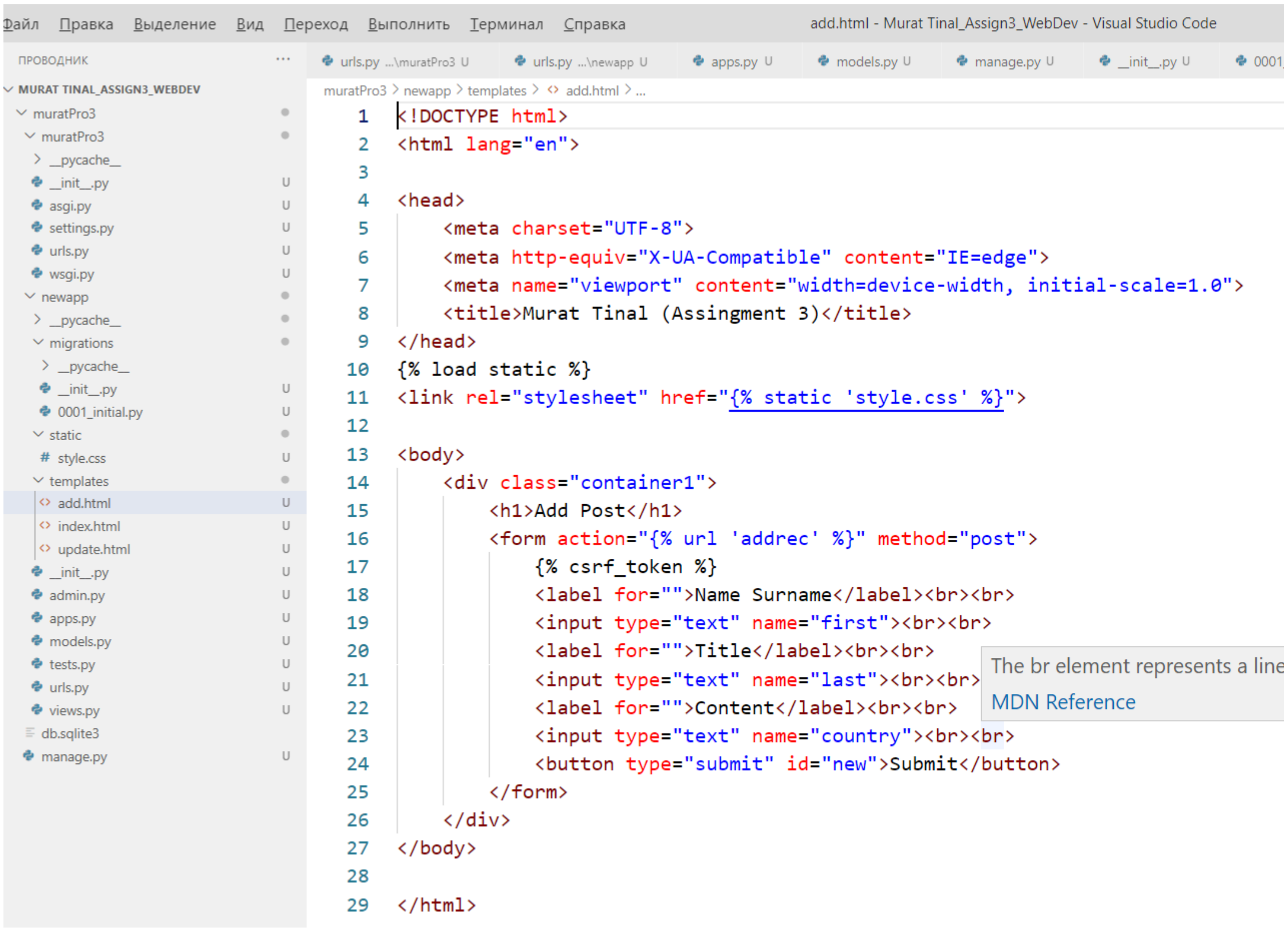

muratPro3 > newapp > migrations > 0001_initial.py > Migration

```python
from django.db import migrations, models
class Migration(migrations.Migration):
    initial = True
    dependencies = [
    ]
    operations = [
        migrations.CreateModel(
            name='Post',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name
                ('namesurname', models.CharField(max_length=100)),
                ('title', models.CharField(max_length=100)),
                ('content', models.CharField(max_length=100)),
            ],
        ),
```

Fig 7.

3. Django Templates

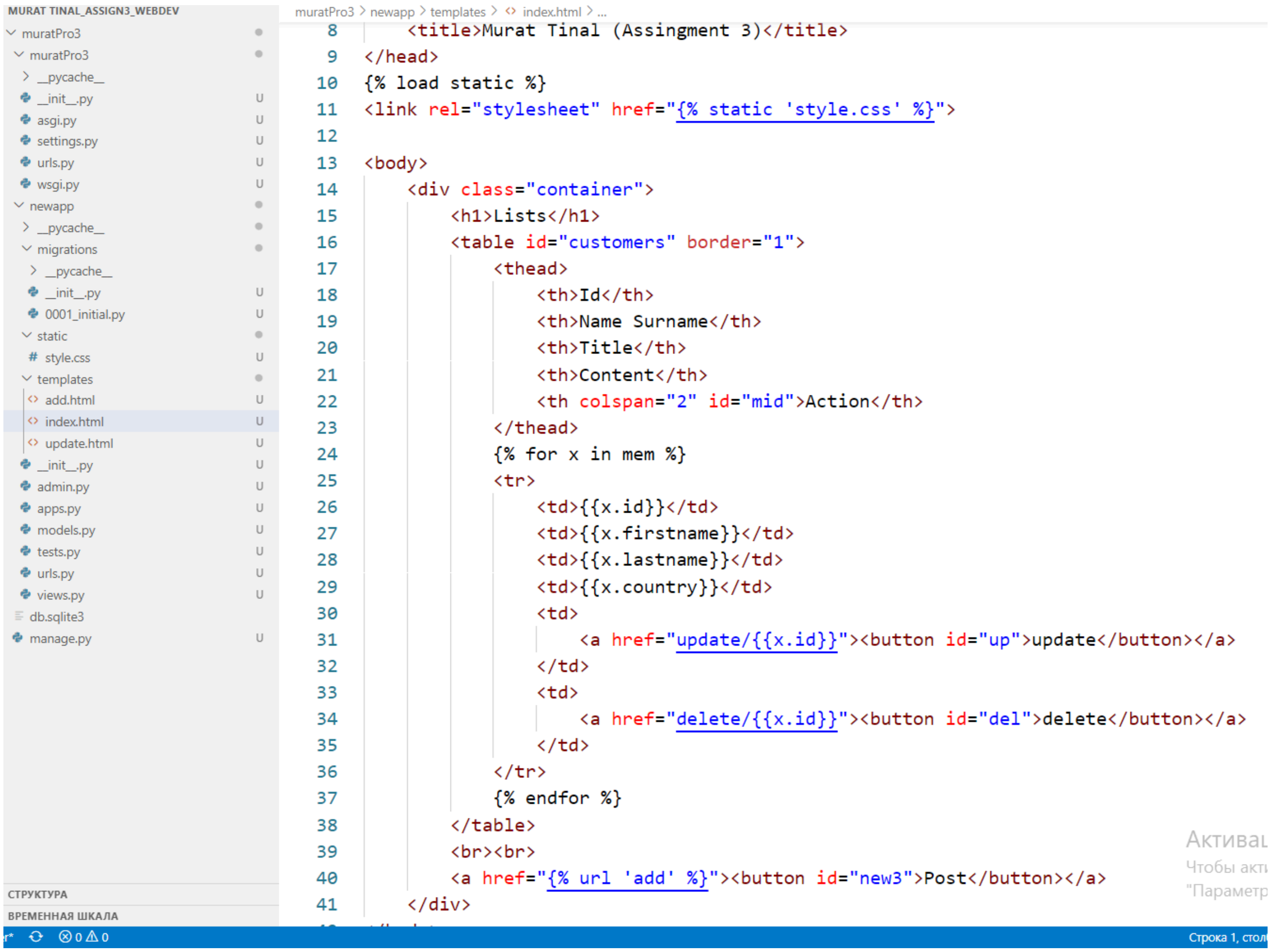
Fig 7 defines a migration for creating the “Post” model in the database. The migration specifies that the model has four fields: “id”, “namesurname”, “title”, and “content”. The “id” field is an auto-incrementing primary key. The migration will be applied to create the table in the database when run.



```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Murat Tinal (Assignment 3)</title>
</head>
{% load static %}
<link rel="stylesheet" href="{% static 'style.css' %}">

<body>
    <div class="container1">
        <h1>Add Post</h1>
        <form action="{% url 'addrec' %}" method="post">
            {% csrf_token %}
            <label for="">Name Surname</label><br><br>
            <input type="text" name="first"><br><br>
            <label for="">Title</label><br><br>
            <input type="text" name="last"><br><br>
            <label for="">Content</label><br><br>
            <input type="text" name="country"><br><br>
            <button type="submit" id="new">Submit</button>
        </form>
    </div>
</body>

</html>
```

Fig 8.

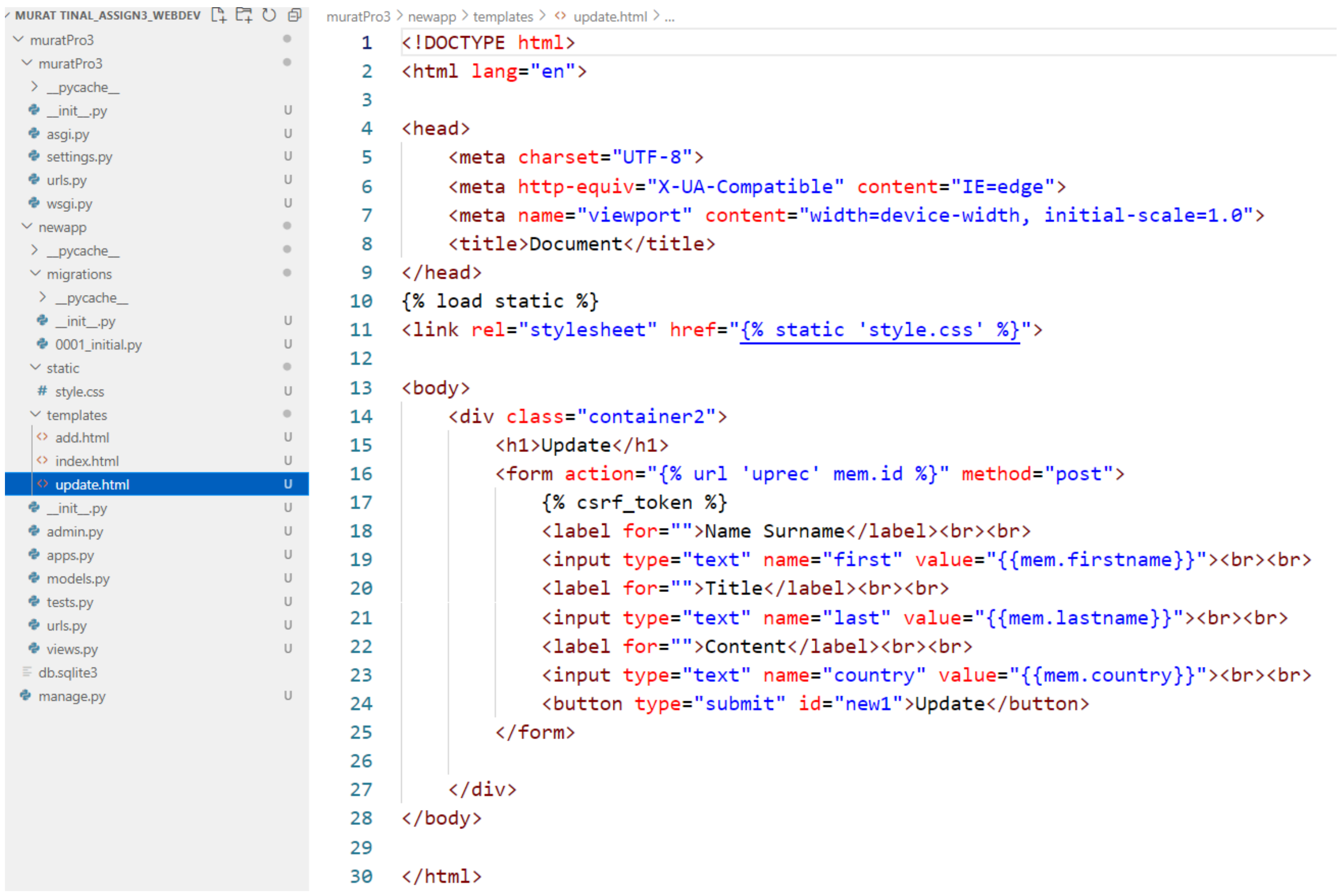
In Fig 8, this code creates an HTML page with a form for adding a new post. The form contains three input fields: one for "Name Surname", one for "Title", and one for "Content". It uses the “POST” method to submit the form data to the “addrec” URL, which is mapped to a view function that handles saving the data in the database. Additionally, the page links to an external CSS file using “{% static 'style.css' %}” to apply styling to the page.



```html
    <title>Murat Tinal (Assingment 3)</title>
</head>
{% load static %}
<link rel="stylesheet" href="{% static 'style.css' %}">

<body>
    <div class="container">
        <h1>Lists</h1>
        <table id="customers" border="1">
            <thead>
                <th>Id</th>
                <th>Name Surname</th>
                <th>Title</th>
                <th>Content</th>
                <th colspan="2" id="mid">Action</th>
            </thead>
            {% for x in mem %}
            <tr>
                <td>{{x.id}}</td>
                <td>{{x.firstname}}</td>
                <td>{{x.lastname}}</td>
                <td>{{x.country}}</td>
                <td>
                    <a href="update/{{x.id}}"><button id="up">update</button></a>
                </td>
                <td>
                    <a href="delete/{{x.id}}"><button id="del">delete</button></a>
                </td>
            </tr>
            {% endfor %}
        </table>
        <br><br>
        <a href="{% url 'add' %}"><button id="new3">Post</button></a>
    </div>
```

Fig 9.

In Fig 9, this code creates an HTML page that displays a list of posts in a table format. The table has columns for “Id”, “Name Surname”, “Title”, and “Content”, with additional columns for "Action" buttons that allow users to update or delete posts. Each row includes buttons that link to the update and delete functions for the specific post, using the post's “id”. A button at the bottom links to the "Add Post" page to create a new post.



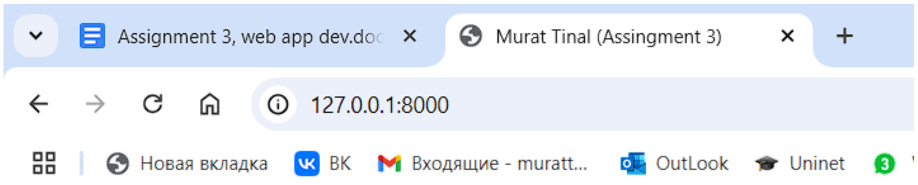
```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
{% load static %}
<link rel="stylesheet" href="{% static 'style.css' %}">

<body>
    <div class="container2">
        <h1>Update</h1>
        <form action="{% url 'uprec' mem.id %}" method="post">
            {% csrf_token %}
            <label for="">Name Surname</label><br><br>
            <input type="text" name="first" value="{{mem.firstname}}"><br><br>
            <label for="">Title</label><br><br>
            <input type="text" name="last" value="{{mem.lastname}}"><br><br>
            <label for="">Content</label><br><br>
            <input type="text" name="country" value="{{mem.country}}"><br><br>
            <button type="submit" id="new1">Update</button>
        </form>

    </div>
</body>

</html>
```

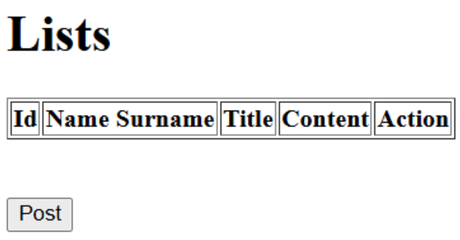
Fig 10.

Fig 10 creates a page for updating an existing post. The form allows users to edit the post's "Name Surname", "Title", and "Content" fields, pre-filled with the current values. The form sends the updated data to the “uprec” URL, where the changes will be processed by the corresponding view. The page includes a submit button to save the updates.

4. Conclusion

In conclusion, this project demonstrates the implementation of a simple BLOG functionality using Django (*Fig 11 - Fig 17*). I created a basic application where users can add, view, update, and delete posts through a web interface. The application uses Django's model, views, and URL routing to interact with the database, and HTML forms for user input. Through this project, I gained hands-on experience with Django's MVC architecture and static file handling. The project provides a foundation for building more complex web applications by utilizing Django's built-in features to manage data efficiently.



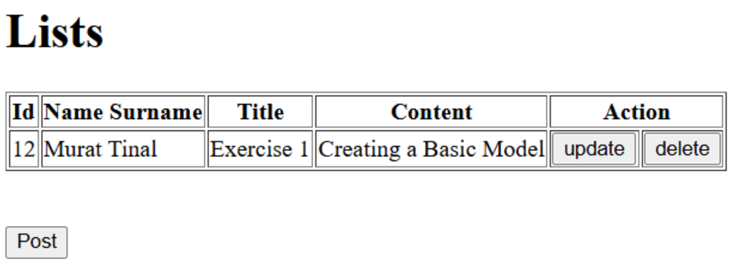


Lists

Id	Name Surname	Title	Content	Action

Post

Fig 11.

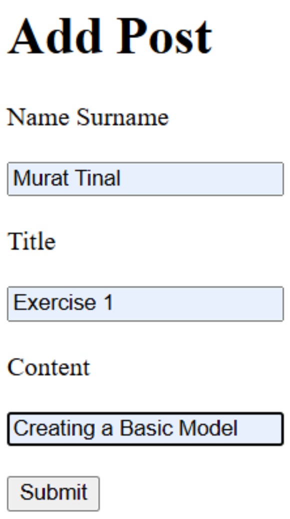


Lists

Id	Name Surname	Title	Content	Action	
12	Murat Tinal	Exercise 1	Creating a Basic Model	update	delete

Post

Fig 13.

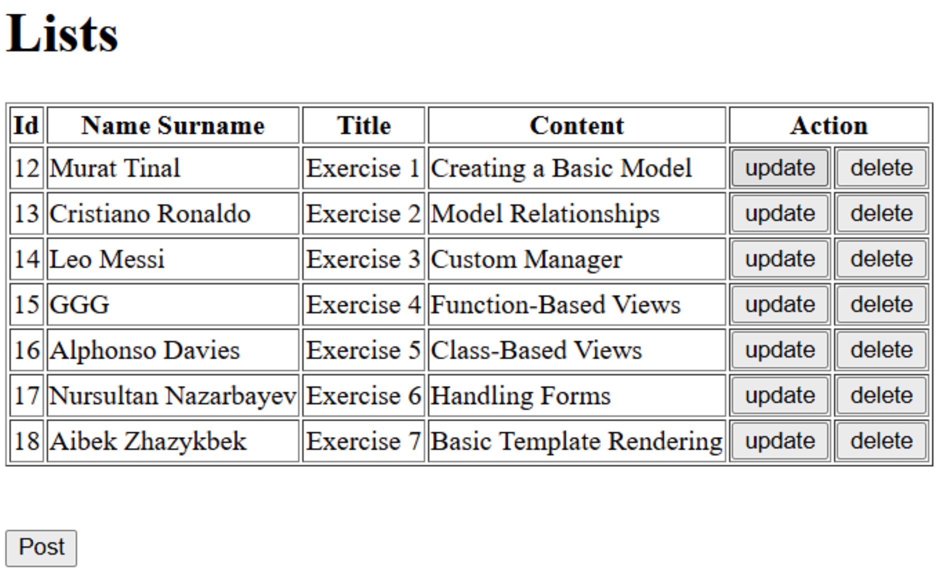


Add Post

Name Surname	Murat Tinal
Title	Exercise 1
Content	Creating a Basic Model

Submit

Fig 12.

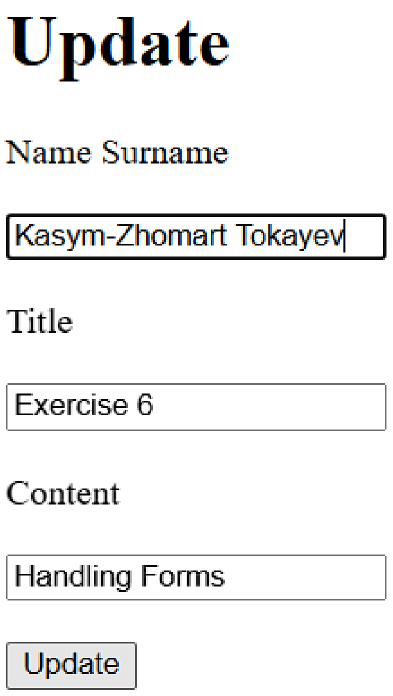


Lists

Id	Name Surname	Title	Content	Action	
12	Murat Tinal	Exercise 1	Creating a Basic Model	update	delete
13	Cristiano Ronaldo	Exercise 2	Model Relationships	update	delete
14	Leo Messi	Exercise 3	Custom Manager	update	delete
15	GGG	Exercise 4	Function-Based Views	update	delete
16	Alphonso Davies	Exercise 5	Class-Based Views	update	delete
17	Nursultan Nazarbayev	Exercise 6	Handling Forms	update	delete
18	Aibek Zhazykbek	Exercise 7	Basic Template Rendering	update	delete

Post

Fig 14.



Update

Name Surname

Kasym-Zhomart Tokayev

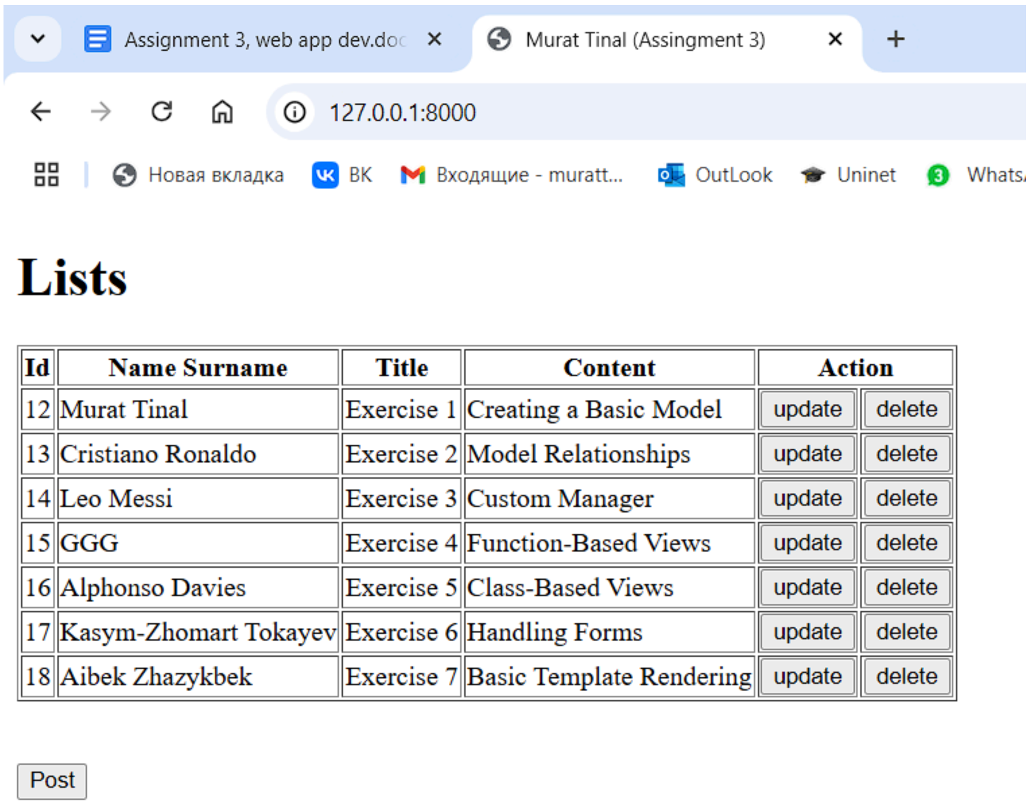
Title

Exercise 6

Content

Handling Forms

Update



Assignment 3, web app dev.doc | Murat Tinal (Assingment 3)

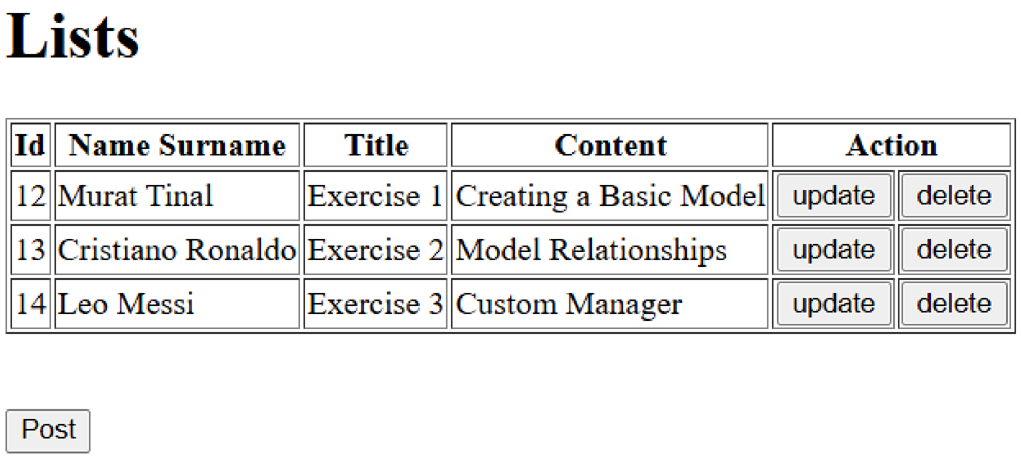
127.0.0.1:8000

Lists

Id	Name Surname	Title	Content	Action	
12	Murat Tinal	Exercise 1	Creating a Basic Model	update	delete
13	Cristiano Ronaldo	Exercise 2	Model Relationships	update	delete
14	Leo Messi	Exercise 3	Custom Manager	update	delete
15	GGG	Exercise 4	Function-Based Views	update	delete
16	Alphonso Davies	Exercise 5	Class-Based Views	update	delete
17	Kasym-Zhomart Tokayev	Exercise 6	Handling Forms	update	delete
18	Aibek Zhazykbek	Exercise 7	Basic Template Rendering	update	delete

Post

Fig 15. **Fig 16.**



Lists

Id	Name Surname	Title	Content	Action	
12	Murat Tinal	Exercise 1	Creating a Basic Model	update	delete
13	Cristiano Ronaldo	Exercise 2	Model Relationships	update	delete
14	Leo Messi	Exercise 3	Custom Manager	update	delete

Post

Fig 17.

In Fig. 11, you can see what my project looks like when it's empty.
In Fig. 12, I added new posts.
In Fig. 13, you can see the post that I added.
In Fig. 14, I added several posts with name, surname, title, and content.
In Fig. 15, I updated "Nursultan Nazarbayev" to "Kasym-Zhomart Tokayev".
In Fig. 16, you can see the updated list of posts.
In Fig. 17, I deleted several posts with their titles and content.

5. References

- https://realpython.com/build-a-blog-from-scratch-django/
- https://djangocentral.com/building-a-blog-application-with-django/
- https://www.youtube.com/watch?v=I8TRkEcw9Mg
- https://dev.to/doridoro/django-several-foreignkey-relations-hj5
- https://medium.com/scalereal/custom-model-managers-in-django-2dac30acdf55
- https://medium.com/@satyarepala/a-comprehensive-guide-to-django-views-understanding-types-and-use-cases-4a02a078ced
- https://medium.com/django-unleashed/mastering-custom-template-tags-in-django-step-by-step-guide-with-real-project-example-4c0184e2bab3

My project (GitHub): https://github.com/kiraakz/Murat_Tinal_WebDev_3