

DJANGO

Framework

- Framework ili okvir za razvoj je skup unapred napisanih kodova koji pruža temelj za razvoj softvera. U kontekstu web developmenta, framework omogućava brzu i strukturiranu izgradnju web aplikacija obezbeđujući kolekciju generičkih funkcionalnosti i komponenti koje se mogu prilagoditi za specifične zadatke.
- Frameworki pomažu u ubrzavanju procesa programiranja uz pomoć unapred definisanih šabloni i funkcija, omogućujući programerima da se fokusiraju na jedinstvene aspekte svojih aplikacija umesto na osnovne, rutinske delove.
- Django je Python web framework koji olakšava brz i čist razvoj dinamičkih web sajtova. Osigurava alate potrebne za brzu izradu robustnih web aplikacija.

Arhitektura Django frameworka

Django koristi specifičnu implementaciju **MVC (Model-View-Controller)** obrasca koju naziva MVT.



MVT je korisan zbog:

Model: Definiše strukturu baze podataka. Modeli su Python klase koje definiraju polja i ponašanje podataka koje će biti korišćeni. Svaki model odgovara jednoj tabeli u bazi podataka.

View: Prikazuje podatke koristeći modele, preuzima ono što korisnik zahteva iz baze, i prosledjuje to šablonu. Views u Django-u su funkcije ili klase koje primaju web zahtev i vraćaju web odgovor.

Template: Šablon koji Django koristi za prikaz informacija u formatu koji korisnik može da čita. Templates su HTML fajlovi koji dozvoljavaju Python-like izraze za dinamičko generisanje web sadržaja.

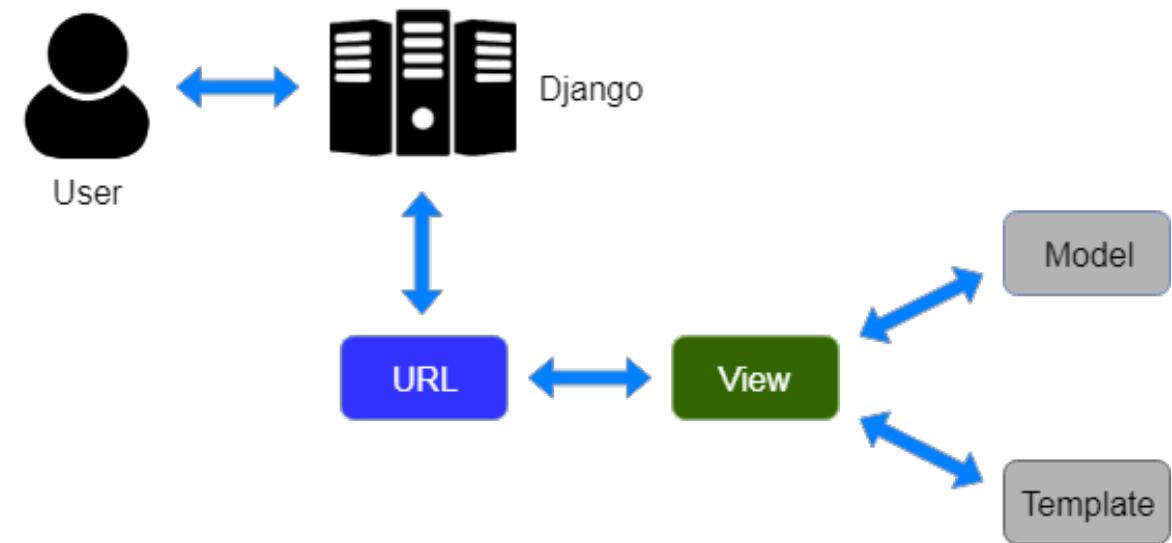
Razdvajanja celina: Odvajanje modela, prikaza, i šablonu čini kod čistijim, organizovanijim i lakšim za održavanje.

Fleksibilnost: Različiti developeri mogu raditi na modelu, priazu, i šablonima nezavisno jedni od drugih.

Ponovne upotrebe koda: Komponente su modularne, što znači da se mogu lako ponovo upotrebiti kroz različite delove aplikacije ili čak u različitim projektima.

Prikaz protoka u Django aplikaciji

- 1. Zahtev od Korisnika:** Korisnik pristupa aplikaciji kroz web pregledač, šaljući zahtev.
- 2. URL Router:** Django koristi URL paterne da odredi koji view treba da obradi zahtev.
- 3. View:** View obrađuje logiku aplikacije, komunicira sa modelom kako bi preuzeo podatke.
- 4. Model:** Model vrši upite prema bazi podataka i vraća rezultate view-u.
- 5. Template:** View prosleđuje podatke u template, koji formira HTML stranicu.
- 6. Odgovor Korisniku:** HTML stranica se vraća korisniku kao odgovor.

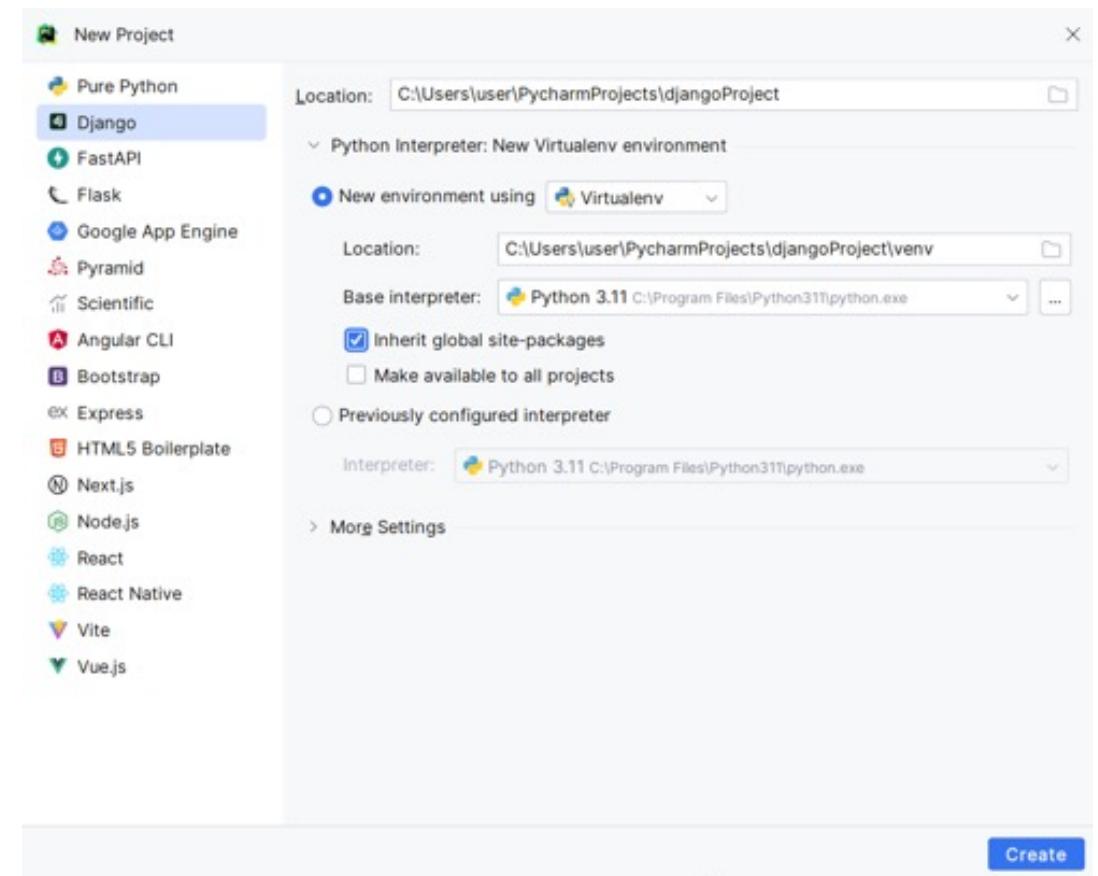


Instalacija i podešavanje Django-a

- Preduslovi:
 - Python – Django zahteva verziju 3.6 ili noviju
 - Pip – menadžer paketa za Django
- Instalacija:
 - U terminal ukucati komandu: `pip install django` .
Ova komanda preuzima i instalira najnoviju stabilnu verziju Django-a iz Python Package Index (PyPI).
- Virtuelno okruzenje:
 - Preporučuje se korišćenje virtuelnog okruženja za izolaciju zavisnosti projekta.
 - Kreiranje virtuelnoj okruženja: `python -m venv myvenv`
 - Aktivacija virtuelnoj okruženja:
 - Windows: `myvenv\Scripts\activate`
 - MacOS i Linux: `source myvenv/bin/activate`

Kreiranje novog Django projekta

- Prikazan je proces kreiranja Django projekta u PyCharm-u.
- Podešavanja uključuju:
 - **Lokacija projekta:** Gde će se projektni fajlovi čuvati na sistemu.
 - **Konfiguracija interpretera:** Odabir verzije Pythona koji će se koristiti i konfiguracija putanje za virtualno okruženje.
 - **Nasleđivanje globalnih paketa:** Opcija koja omogućava da se globalno instalirani paketi koriste unutar virtualnog okruženja, ako je potrebno.



Pregled početnog Django projekta

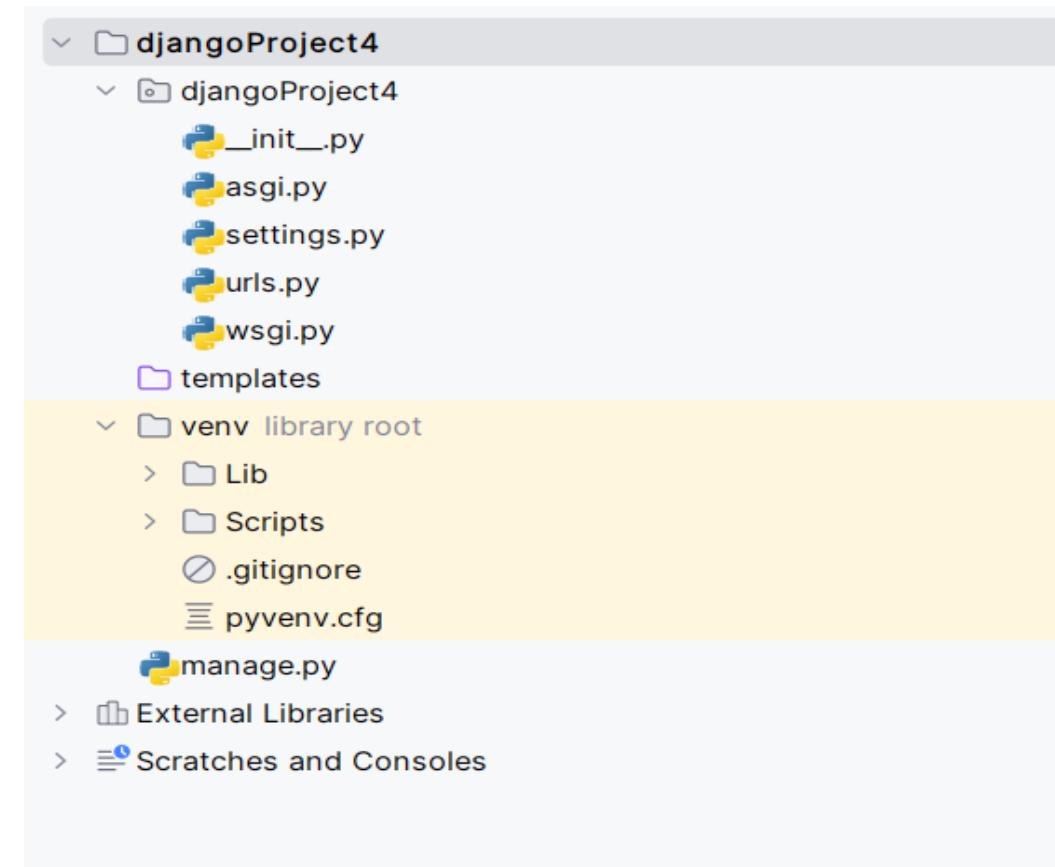
- Kada se kreira novi Django projekt, automatski se generiše određena struktura direktorijuma i fajlova koji su osnova za svaku Django aplikaciju.

Komponente koje obuhvata Django projekt:

- **manage.py** - skript koji omogućava interakciju sa projektom, poput pokretanja servera, kreiranja novih aplikacija, izvršavanja migracija i mnogih drugih. Dostupne komande proveriti sa: *python manage.py*

Projektni direktorijum :

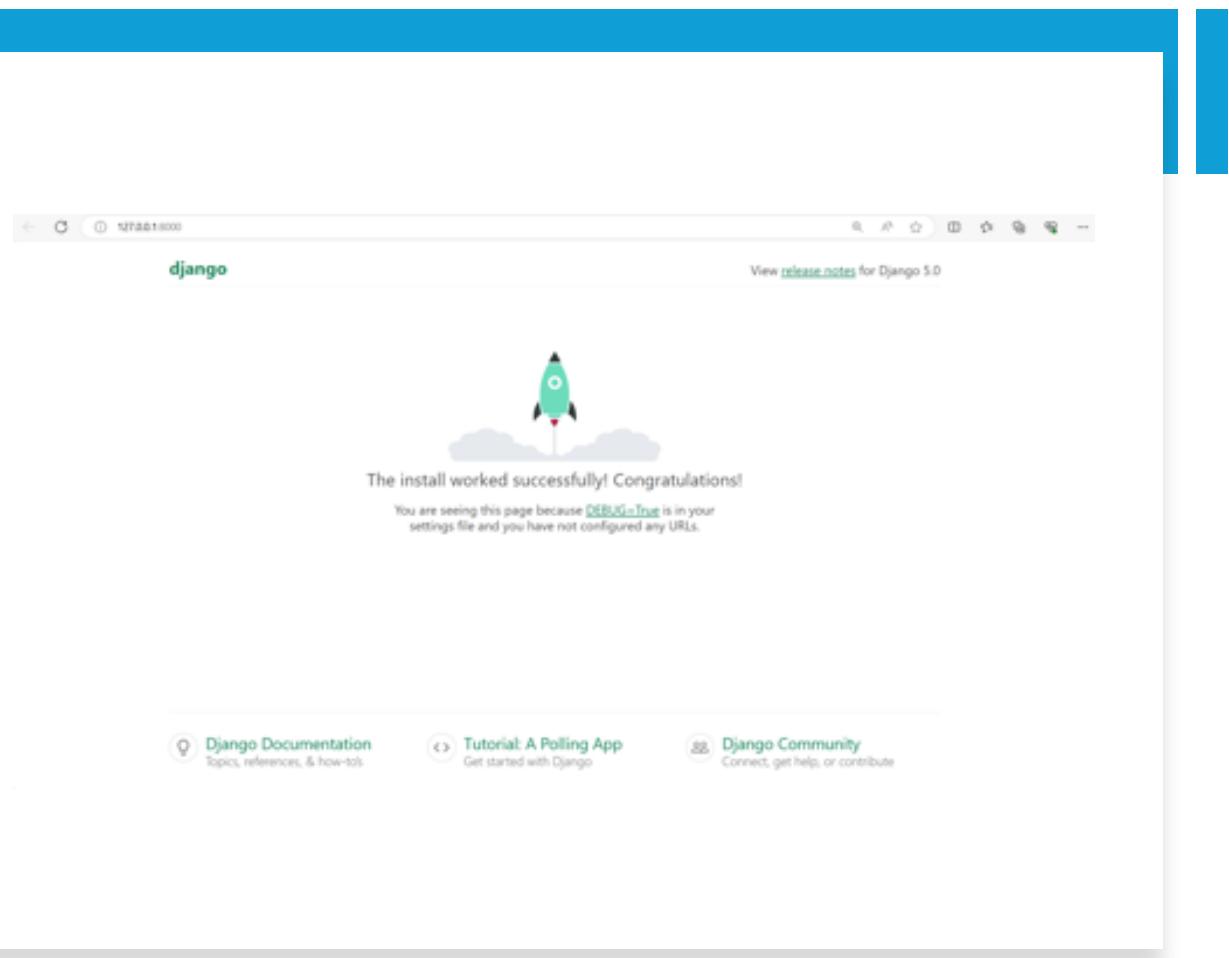
- **init.py** – obično prazan, pravi Python paket od direktorijuma u kojem se nalazi.
- **settings.py** – konfiguracije za projekt: podešavanje baze podataka, konfiguracija statickih i medijskih fajlova, middlewares, instalirane aplikacije, template konfiguracije...
- **urls.py** – definiše URL šeme za projekt
- **wsgi.py i asgi.py** – omogućavaju Django aplikaciji komunikaciju sa web serverom



Pregled početnog Django projekta

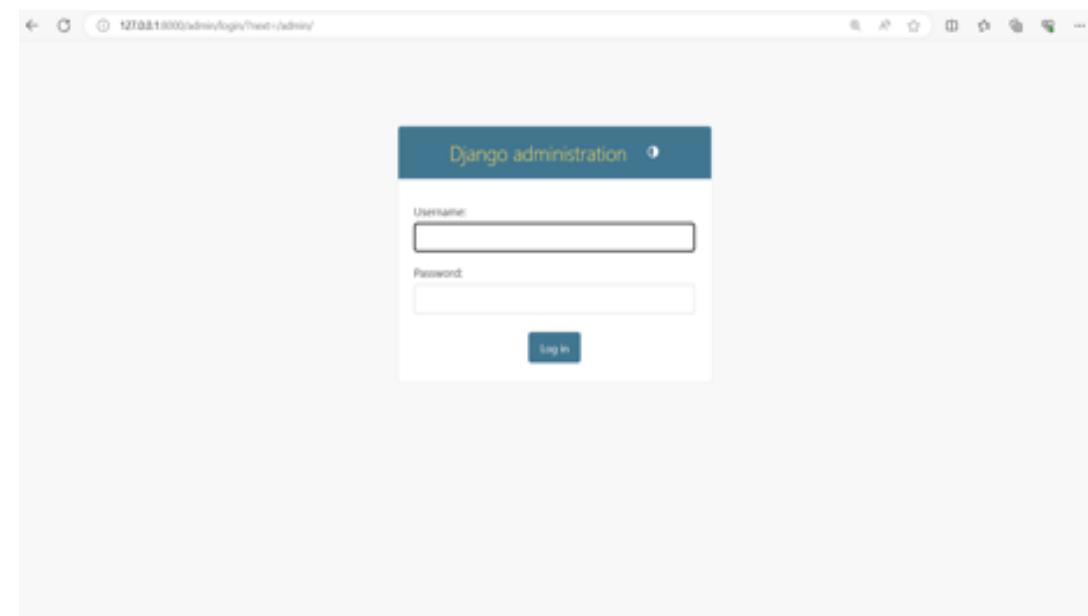
- Pokretanje razvojnog servera: *python manage.py runserver*.

Ova komanda pokreće lokalni razvojni server koji omogućava pristup projektu preko web pregledača.



Pregled početnog Django projekta

Django admin: Django pruža moćan administrativni interfejs koji se može koristiti za upravljanje podacima u aplikaciji odmah nakon instalacije. Dostupan je preko '/admin' putanje nakon pokretanja razvojnog servera.



Kreiranje Django aplikacije

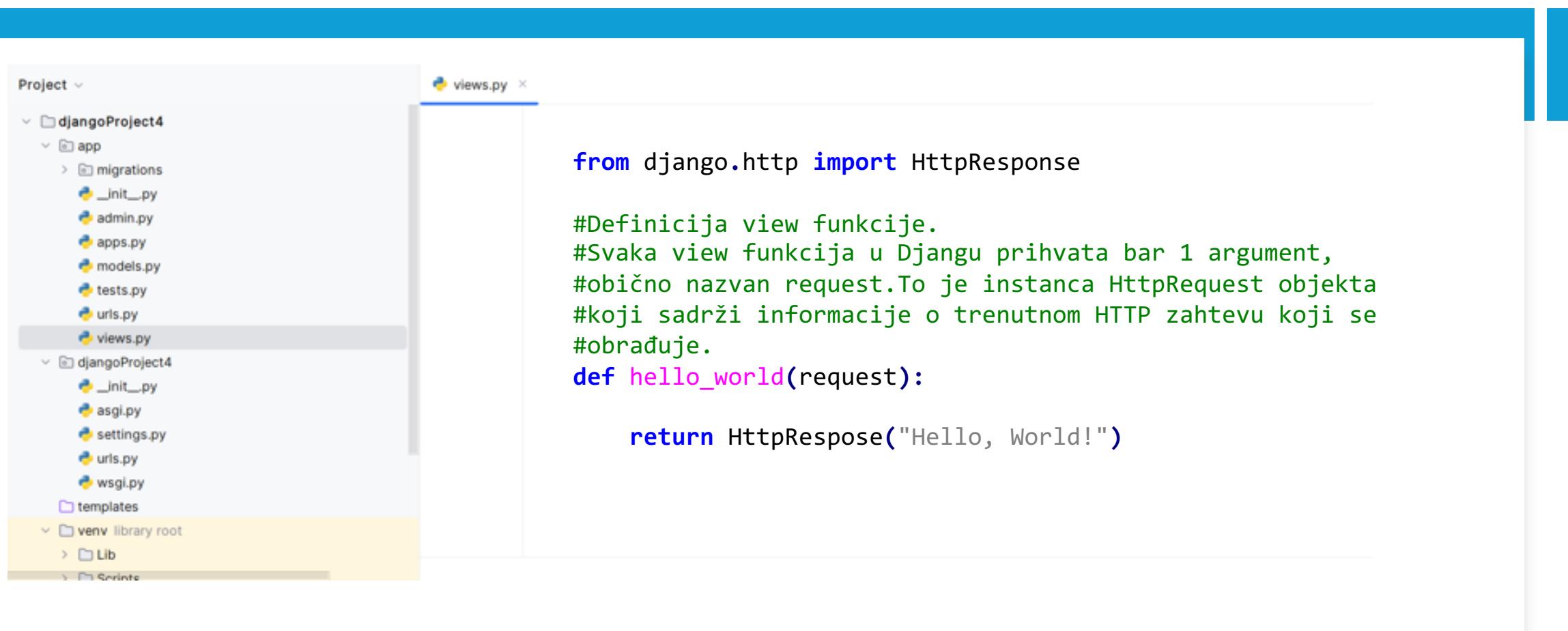
- **Inicijalizacija:** Django aplikacija se kreira unutar postojećeg Django projekta koristeći sledeću komandu: `python manage.py startapp app`.

Ova komanda kreira novi direktorijum ‘app’ unutar projekta koji sadrži početnu strukturu aplikacije.

- **Osnovna Struktura Aplikacije ‘app’:**

- **migrations/**: Direktorijum koji sadrži migracije za modele. Migracije su automatski generisani fajlovi koji čuvaju promene napravljene na modelima, omogućavajući ažuriranje šeme baze na kontrolisan način. Svaka migracija ima svoj identifikacioni broj i opisuje promene koje se primenjuju na bazu.
- **init.py**: Označava direktorijum kao Python paket.
- **admin.py**: Koristi se za registraciju modela unutar Django admin interfejsa.
- **apps.py**: Sadrži konfiguraciju same aplikacije.
- **models.py**: Definiše strukturu baze podataka (modeli).
- **tests.py**: Fajl za testiranje komponenti aplikacije.
- **views.py**: Definiše prikaze koji obrađuju zahteve i vraćaju odgovore.

Hello World aplikacija



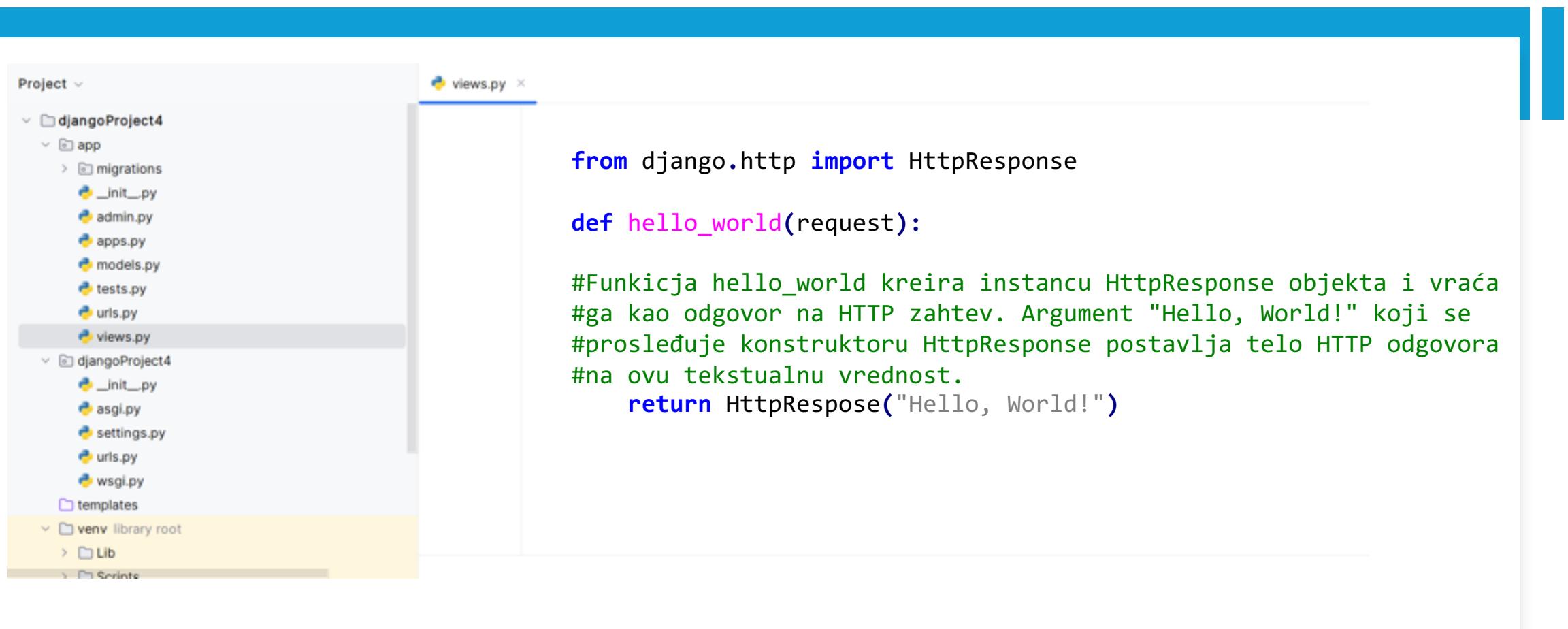
The screenshot shows a code editor interface with a blue header bar. On the left is a project tree titled "Project". It lists a "djangoProject4" folder containing an "app" folder with files like migrations, __init__.py, admin.py, apps.py, models.py, tests.py, urls.py, and views.py. Below "app" are files __init__.py, asgi.py, settings.py, urls.py, wsgi.py, and templates. Under "djangoProject4" are __init__.py, asgi.py, settings.py, urls.py, wsgi.py, and venv. The "venv" folder has subfolders library root, Lib, and Scripts. A yellow highlight covers the bottom of the project tree and the bottom of the code editor window. The main editor area has a tab titled "views.py". The code in "views.py" is:

```
from django.http import HttpResponse

#Definicija view funkcije.
#Svaka view funkcija u Django prihvata bar 1 argument,
#obično nazvan request. To je instanca HttpRequest objekta
#koji sadrži informacije o trenutnom HTTP zahtevu koji se
#obrađuje.
def hello_world(request):

    return HttpResponse("Hello, World!")
```

Hello World aplikacija



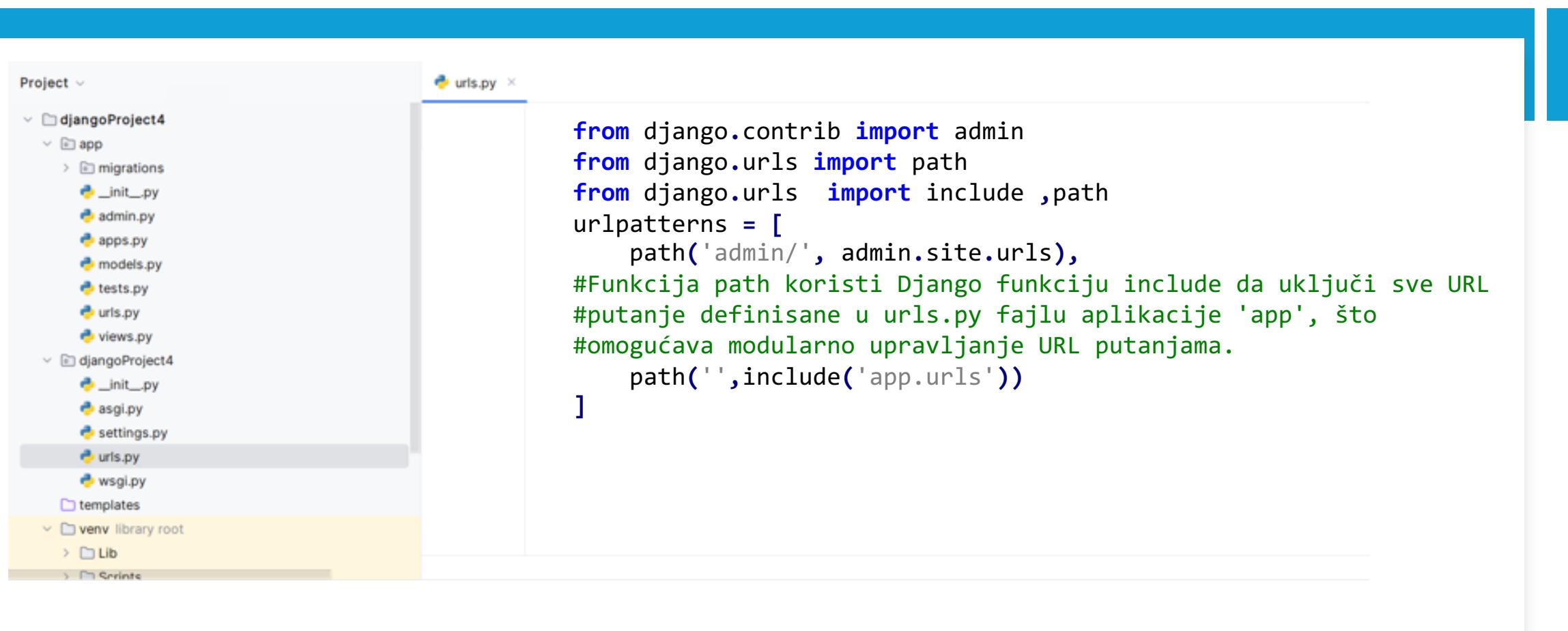
The screenshot shows a code editor interface with a blue header bar. On the left, there is a project navigation sidebar titled "Project" which lists the structure of a Django project named "djangoProject4". The "app" directory contains files like __init__.py, migrations, admin.py, apps.py, models.py, tests.py, urls.py, and views.py. The "djangoProject4" directory contains __init__.py, asgi.py, settings.py, urls.py, wsgi.py, and templates. A yellow highlight covers the bottom of the sidebar and the bottom of the main editor area. The main editor window has a tab titled "views.py" with the following Python code:

```
from django.http import HttpResponse

def hello_world(request):

    #Funkcija hello_world kreira instancu HttpResponse objekta i vraća
    #ga kao odgovor na HTTP zahtev. Argument "Hello, World!" koji se
    #prosleđuje konstruktoru HttpResponse postavlja telo HTTP odgovora
    #na ovu tekstualnu vrednost.
    return HttpResponse("Hello, World!")
```

Hello World aplikacija



The screenshot shows a code editor interface with a sidebar labeled "Project". The sidebar lists the following files and folders:

- djangoProject4 (selected)
- app
 - migrations
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - tests.py
 - urls.py
 - views.py
- djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py (highlighted)
 - wsgi.py
- templates
- venv library root
 - Lib
 - Scripts

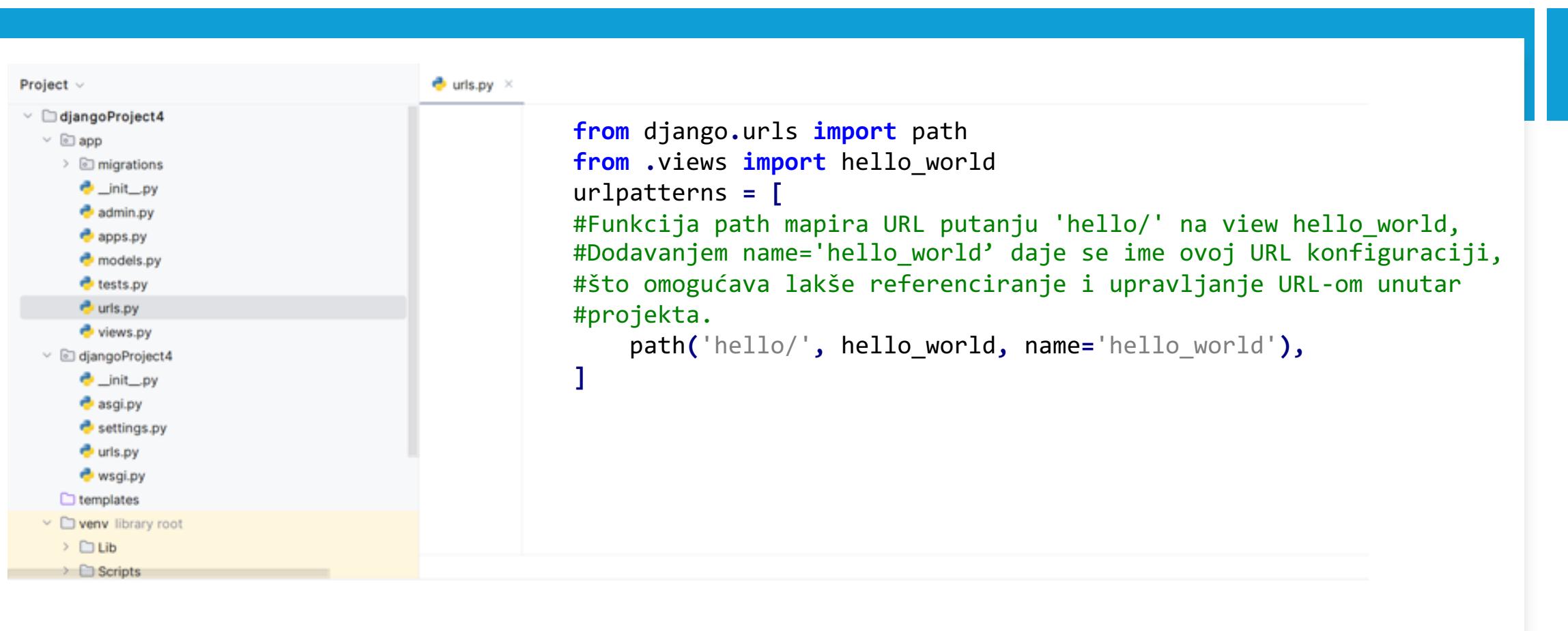
The main editor window displays the contents of the "urls.py" file for the "app" application. The code is as follows:

```
from django.contrib import admin
from django.urls import path
from django.urls import include ,path
urlpatterns = [
    path('admin/', admin.site.urls),
#Funkcija path koristi Django funkciju include da uključi sve URL
#putanje definisane u urls.py fajlu aplikacije 'app', što
#omogućava modularno upravljanje URL putanjama.
    path('' ,include('app.urls'))
]
```

A green annotation is present in the code, explaining the purpose of the "include" function:

#Funkcija path koristi Django funkciju include da uključi sve URL
#putanje definisane u urls.py fajlu aplikacije 'app', što
#omogućava modularno upravljanje URL putanjama.

Hello World aplikacija



The screenshot shows a code editor interface with a blue header bar. On the left, there is a project navigation sidebar titled "Project" showing the structure of a Django project named "djangoProject4". Inside "djangoProject4", there is an "app" folder containing files like migrations, __init__.py, admin.py, apps.py, models.py, tests.py, urls.py (which is currently selected and highlighted in grey), and views.py. Below "djangoProject4" are files __init__.py, asgi.py, settings.py, urls.py, and wsgi.py. There is also a "templates" folder and a "venv library root" folder containing Lib and Scripts. A yellow rectangular selection highlights the "venv library root" folder.

The main editor area has a tab titled "urls.py" with the following Python code:

```
from django.urls import path
from .views import hello_world
urlpatterns = [
    #Funkcija path mapira URL putanju 'hello/' na view hello_world,
    #Dodavanjem name='hello_world' daje se ime ovoj URL konfiguraciji,
    #što omogućava lakše referenciranje i upravljanje URL-om unutar
    #projekta.
    path('hello/', hello_world, name='hello_world'),
]
```

The code uses color-coded syntax highlighting where blue represents keywords like "from", "import", and "path", green represents comments, and purple represents the function name "hello_world".

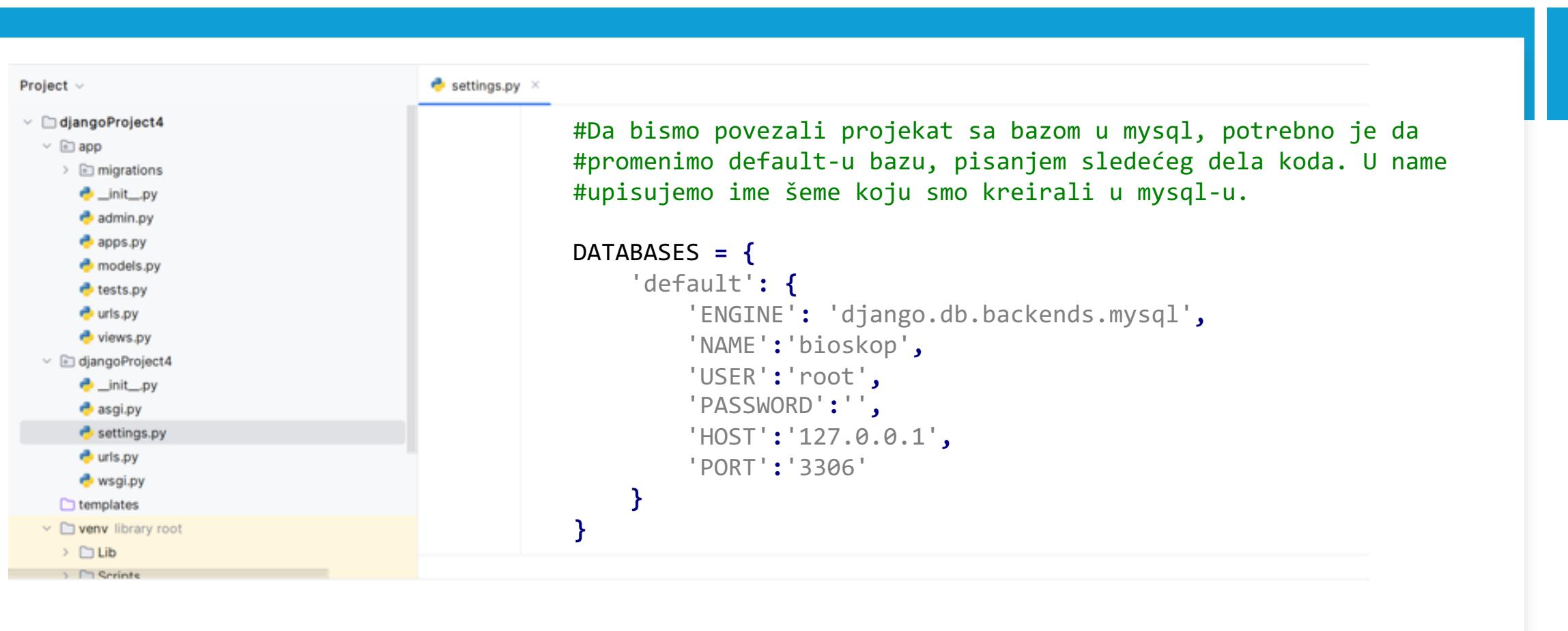
Hello World aplikacija

The screenshot shows a code editor interface with a sidebar displaying the project structure. The project contains a main directory 'djangoProject4' which includes an 'app' folder and a 'djangoProject4' folder. The 'app' folder contains files like migrations, __init__.py, admin.py, apps.py, models.py, tests.py, urls.py, and views.py. The 'djangoProject4' folder contains __init__.py, asgi.py, settings.py (which is currently selected), urls.py, and wsgi.py. There is also a templates folder and a venv library root. A yellow highlight covers the bottom of the sidebar and the bottom of the main editor area.

settings.py

```
INSTALLED_APPS = [  
    #Prepoznavanje i učitavanje aplikacije app kao deo projekta.  
    # AppConfig je ime klase unutar apps.py aplikacije app.  
    'app.apps.AppConfig',  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

Prikaz podataka iz baze



The screenshot shows a code editor interface with a blue header bar. On the left is a sidebar titled "Project" showing the directory structure of a Django project named "djangoProject4". The "settings.py" file is selected and highlighted with a grey background. The main editor area contains Python code for configuring a MySQL database connection.

```
#Da bismo povezali projekat sa bazom u mysql, potrebno je da
#promenimo default-u bazu, pisanjem sledećeg dela koda. U name
#upisujemo ime šeme koju smo kreirali u mysql-u.

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'bioskop',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': '127.0.0.1',
        'PORT': '3306'
    }
}
```

Prikaz podataka iz baze



The screenshot shows a code editor interface with a sidebar and a main workspace.

Project Sidebar:

- djangoProject4
 - app
 - migrations
 - __init__.py
 - admin.py
 - apps.py
 - models.py** (highlighted)
 - tests.py
 - urls.py
 - views.py
 - djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py
 - templates
 - venv library root
 - Lib
 - Scripts

Main Workspace (models.py):

```
from django.db import models

#Podatke iz baze smo dobili uz pomoć komande python.exe manage.py
#inspectdb. Filmovi predstavlja ime tabele unutar baze a ime,
#reziser i godina predstavljaju kolone unutar tabele u bazi.
class Filmovi(models.Model):
    ime = models.CharField(max_length=45, blank=True, null=True)
    reziser = models.CharField(max_length=45, blank=True,
null=True)
    godina = models.CharField(max_length=45, blank=True,
null=True)

    class Meta:
        managed = False
        db_table = 'filmovi'
```

Prikaz podataka iz baze

The screenshot shows a code editor interface with a blue header bar. On the left is a tree view of a Django project named "djangoProject4". The "app" directory contains files: __init__.py, migrations, admin.py, apps.py, models.py, tests.py, urls.py, and views.py (which is currently selected). The root directory "djangoProject4" contains __init__.py, asgi.py, settings.py, urls.py, wsgi.py, and templates. A yellow bar highlights the "venv library root" and "Lib" folder under it. The main editor window is titled "views.py" and contains the following Python code:

```
from django.shortcuts import render
from .models import Filmovi

def prikaz(request):
    #Dohvataju se svi objekti iz Filmovi modela. Filmovi je Django
    #model, i objects.all() je QuerySet metoda koja vraća sve instance
    #Filmovi modela iz baze podataka.
    filmovi=Filmovi.objects.all()
    context={'filmovi':filmovi}
    return render(request,'app/filmovi.html',context)
```

Prikaz podataka iz baze

The screenshot shows a code editor interface with a sidebar and a main code area. The sidebar on the left displays the project structure:

- Project
- djangoProject4
 - app
 - migrations
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py
 - templates
 - venv library root
 - Lib
 - Scripts

Django .object menadžer

- Menadžeri su interfejs kroz koji se Django modeli vrše upite nad svojim podacima.
- Svi upiti na bazi, kao što su dohvatanje, filtriranje i sortiranje podataka, izvršavaju se kroz menadžere.

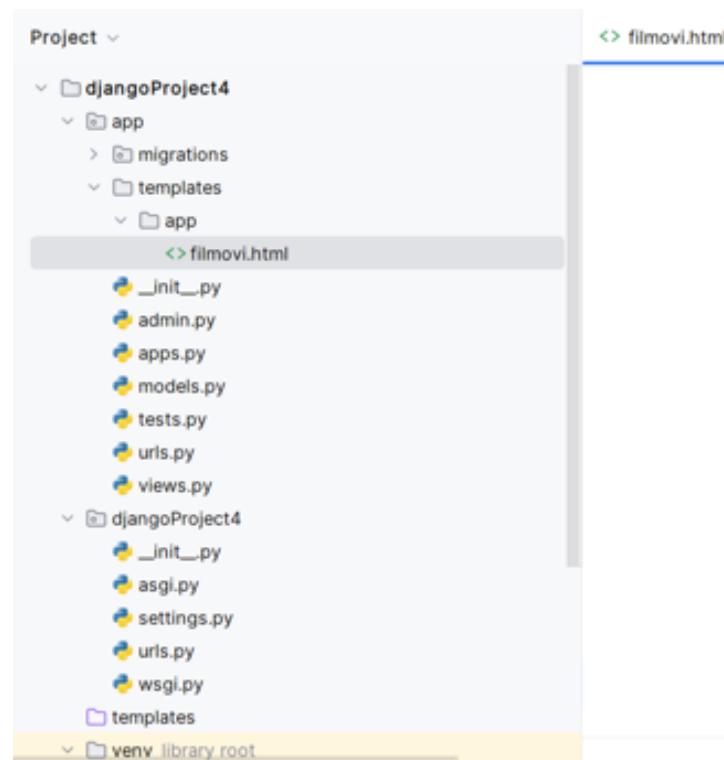
Primeri metoda:

- `.all()` - Vraća QuerySet koji sadrži sve objekte modela
- `.filter()` - Vraća QuerySet koji uključuje objekte koji zadovoljavaju određene kriterijume
- `.exclude()` - Vraća QuerySet koji isključuje objekte koji zadovoljavaju određene kriterijume.
- `.get()` - Vraća tačno jedan objekt koji zadovoljava kriterijume. Ako nema takvog objekta ili ih ima više, baca izuzetak.
- `.create()` - Kreira novi objekt, sačuva ga u bazi i vrati taj objekt.
- `.count()` - Vraća broj objekata koji odgovaraju upitu.

Django .object menadžer - Primeri

```
Filmovi.objects.all()  
Filmovi.objects.filter(reziser='Nolan')  
Filmovi.objects.exclude(godina='2020')  
Filmovi.objects.get(id=1)  
Filmovi.objects.create(ime='Inception', reziser='Nolan', godina='2010')  
Filmovi.objects.filter(reziser='Nolan').count()
```

Prikaz podataka iz baze



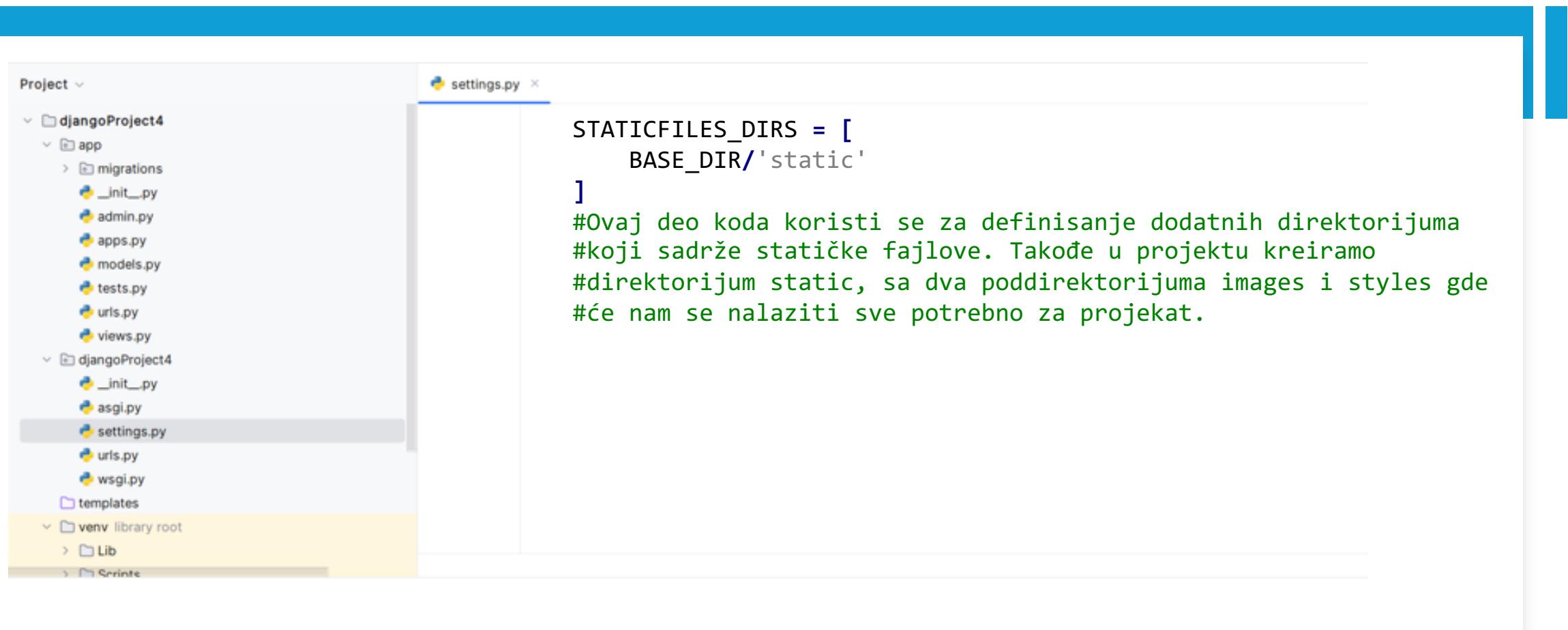
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Filmovi</title>
</head>
<body>
<ul>
    {% for film in filmovi %}
        <li>Ime:{{ film.ime }} reziser:{{ film.reziser }}</li>
    {% endfor %}
</ul>
</body>
</html>
```

Ako html fajl kreiramo u ovom folderu,
bitno je da se u `settings.py` u delu
TEMPLATES, DIRS promeni na:
`'DIRS': []`

Django Template Tagovi

- Django template tagovi su specijalne sintakse u Django template sistemima koji omogućuju izvršavanje određenih programskih logika direktno unutar HTML šablonu.
 - Tagovi se koriste za kontrolu toka, manipulaciju promenljivima, učitavanje statičkih fajlova, nasleđivanje šablonu i još mnogo toga.
1. Kontrola toka:
 - `{% if %}`, `{% else %}`, `{% endif %}` - Uslovi koji kontrolisu koji deo šablonu će biti prikazan.
 - `{% for %}` - Petlja koja se koristi za iteraciju kroz liste ili querysetove.
 - `{% empty %}` - Specifično za `{% for %}` petlju, prikazuje sadržaj ako je lista prazna.
 2. Učitavanje statičkih stranica:
 - `{% load static %}` - Omogućava učitavanje statičkih resursa kao što su CSS i JavaScript fajlovi.
 - `{% static "path/to/file" %}` - Koristi se nakon `{% load static %}` za specifikaciju putanje do statičkog resursa.
 3. Nasleđivanje šablonu:
 - `{% extends "base.html" %}` - Definiše osnovni šablon od kojeg trenutni šablon nasleđuje.
 - `{% block content/title %}` i `{% endblock %}` - Definiše blokove koji se mogu prebrisati u nasleđenim šablonima.

Definisanje static foldera

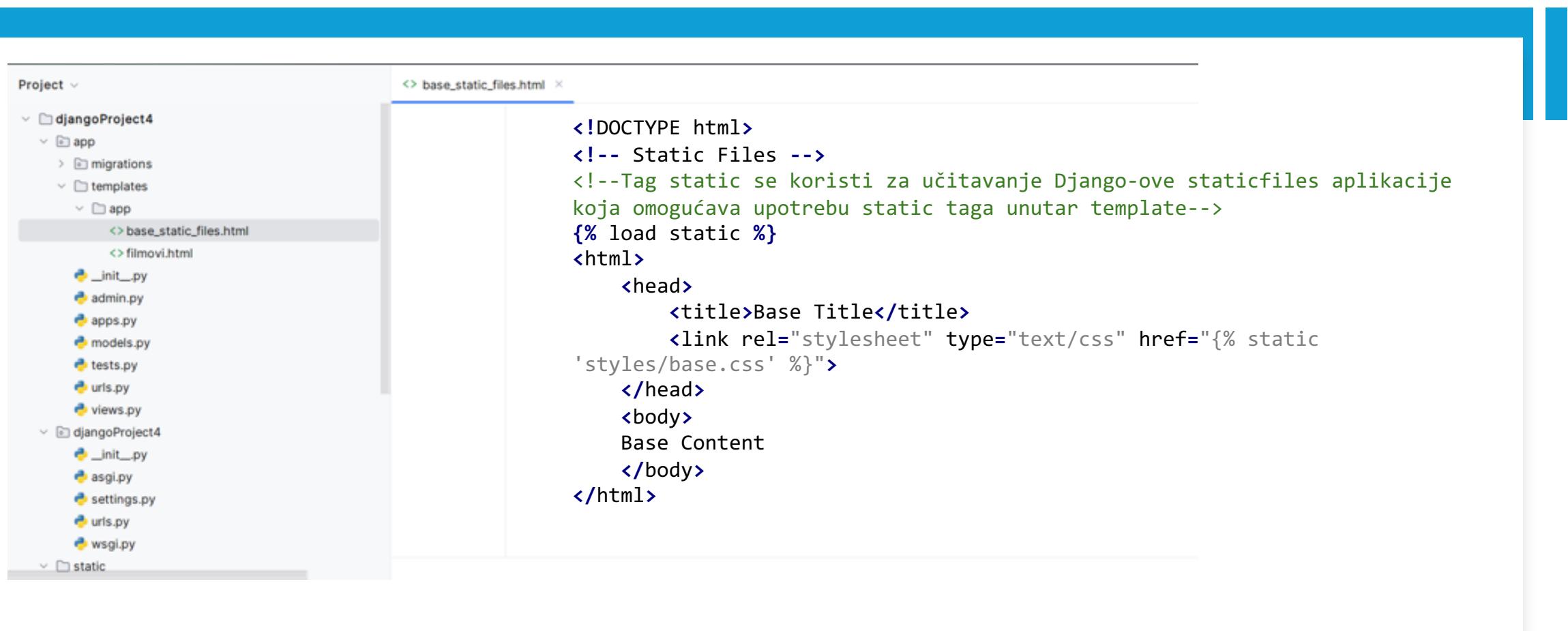


The screenshot shows a code editor interface with a blue header bar. On the left is a 'Project' sidebar listing the structure of a Django project named 'djangoProject4'. The 'settings.py' file is selected and highlighted with a grey background. The main editor area contains the following Python code:

```
STATICFILES_DIRS = [
    BASE_DIR/'static'
]
#Ovaj deo koda koristi se za definisanje dodatnih direktorijuma
#koji sadrže statičke fajlove. Takođe u projektu kreiramo
#direktorijum static, sa dva poddirektorijuma images i styles gde
#će nam se nalaziti sve potrebno za projekat.
```

The code defines the `STATICFILES_DIRS` setting, which is a list containing the path `BASE_DIR/'static'`. A multi-line comment explains the purpose of this setting: it is used to define additional directories that contain static files. It also notes that in the project, we create a directory named `static`, which contains two sub-directories `images` and `styles` where all necessary files for the project will be located.

Korišćenje static fajlova



The screenshot shows a code editor interface with a sidebar and a main editor area.

Project Sidebar:

- djangoProject4
 - app
 - migrations
 - templates
 - app
 - base_static_files.html
 - filmovi.html
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py
 - static

Main Editor Area (base_static_files.html):

```
<!DOCTYPE html>

<!-- Tag static se koristi za učitavanje Django-ove staticfiles aplikacije
koja omogućava upotrebu static taga unutar template--&gt;
{% load static %}

&lt;html&gt;
    &lt;head&gt;
        &lt;title&gt;Base Title&lt;/title&gt;
        &lt;link rel="stylesheet" type="text/css" href="{% static
'static/base.css' %}"&gt;
    &lt;/head&gt;
    &lt;body&gt;
        Base Content
    &lt;/body&gt;
&lt;/html&gt;</pre>
```

Korišćenje static fajlova

The screenshot shows a code editor interface with a sidebar and a main editor area.

Project Sidebar:

- djangoProject4
 - app
 - migrations
 - templates
 - app
 - base_static_files.html
 - filmovi.html
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py
 - static

Main Editor Area:

```
<!DOCTYPE html>
<!-- Static Files --&gt;
{% load static %}

&lt;html&gt;
    &lt;head&gt;
        &lt;title&gt;Base Title&lt;/title&gt;
    &lt;!--Linkuje CSS fajl koji se nalazi u static folderu aplikacije--&gt;
    &lt;link rel="stylesheet" type="text/css" href="{% static
'static/base.css' %}"&gt;
    &lt;/head&gt;
    &lt;body&gt;
        Base Content
    &lt;/body&gt;
&lt;/html&gt;</pre>
```

Nasleđivanje template-a

The screenshot shows a code editor interface with a sidebar and a main editor area. The sidebar on the left displays the project structure:

- Project
- djangoProject4
 - app
 - migrations
 - templates
 - app
 - base.html
 - base_static_files.html
 - filmovi.html
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py

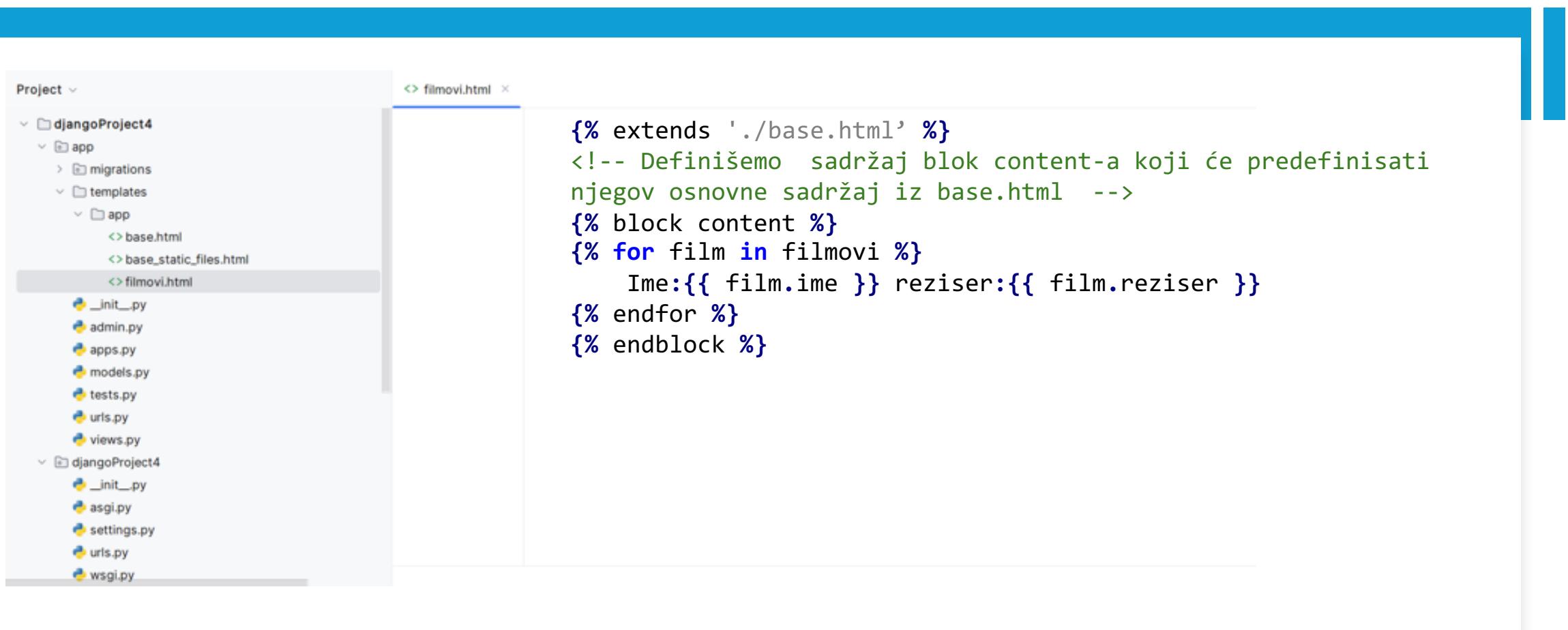
```
<!DOCTYPE html>
<html>
    <head>
        <!-- Blok koji omogućava template-ima koji naslede ovaj da predefinišu sadržaj unutar title taga. Ako nasledni template ne definiše ovaj blok, biće prikazan Base Title.
        Isto važi i za block content--&gt;
        &lt;title&gt;{% block title %}Base Title{% endblock %}&lt;/title&gt;
    &lt;body&gt;
        {% block content %}
            Base Content
        {% endblock %}
    &lt;/body&gt;
&lt;/html&gt;</pre>
```

Nasleđivanje template-a

The screenshot shows a code editor interface with a blue header bar. On the left, there is a project tree labeled "Project". It lists a "djangoProject4" project containing an "app" directory with "migrations", "templates", and "models.py". Inside "templates", there is an "app" directory with "base.html", "base_static_files.html", and "filmovi.html". The "filmovi.html" file is selected and shown in the main editor area. The code in "filmovi.html" is as follows:

```
<!-- Pomoću extends taga govorimo da je ovaj template nasledio  
base.html -->  
{% extends './base.html' %}  
{% block content %}  
    <ul>  
        {% for film in filmovi %}  
            <li> Ime:{{ film.ime }} reziser:{{ film.reziser }}</li>  
        {% endfor %}  
    </ul>  
{% endblock %}
```

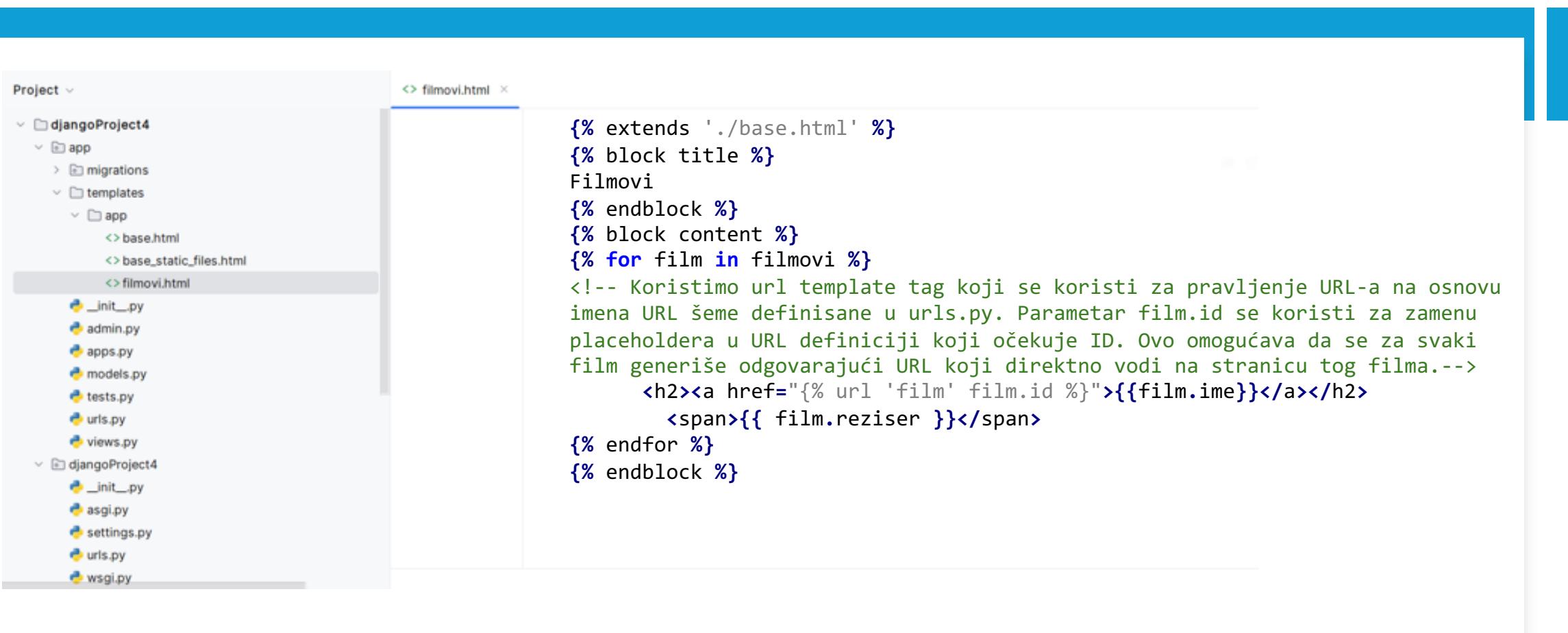
Nasleđivanje template-a



The screenshot shows a code editor interface with a blue header bar. On the left is a project tree titled "Project". It lists a "djangoProject4" folder containing an "app" folder with "migrations", "templates", and another "app" folder. Inside the inner "app" folder are "base.html", "base_static_files.html", and "filmovi.html". Other files like "__init__.py", "admin.py", etc., are also listed. The right pane shows the content of "filmovi.html". The code is written in Django template language:

```
{% extends './base.html' %}  
<!-- Definišemo sadržaj blok content-a koji će predefinisati<br/>njegov osnovne sadržaj iz base.html -->  
{% block content %}  
{% for film in filmovi %}  
    Ime:{{ film.ime }} reziser:{{ film.reziser }}  
{% endfor %}  
{% endblock %}
```

Parametrizovan URL



The screenshot shows a code editor interface with a project structure on the left and a code editor window on the right.

Project Tree:

- djangoProject4
 - app
 - migrations
 - templates
 - base.html
 - base_static_files.html
 - filmovi.html
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py

Code Editor (filmovi.html):

```
{% extends './base.html' %}  
{% block title %}  
Filmovi  
{% endblock %}  
{% block content %}  
{% for film in filmovi %}  
<!-- Koristimo url template tag koji se koristi za pravljenje URL-a na osnovu<br/>imena URL šeme definisane u urls.py. Parametar film.id se koristi za zamenu  
placeholdera u URL definiciji koji očekuje ID. Ovo omogućava da se za svaki  
film generiše odgovarajući URL koji direktno vodi na stranicu tog filma.-->  
    <h2><a href="{% url 'film' film.id %}">{{film.ime}}</a></h2>  
    <span>{{ film.reziser }}</span>  
{% endfor %}  
{% endblock %}
```

Parametrizovan URL

Putanja:

```
path('film/<int:id>', views.film_detalji, name='film')
```

- Ovde, **<int:id>** je dinamički segment u URL putanji koji očekuje ceo broj (**int**). Kada koristite **{% url 'film' film.id %}**, Django zamenuje **<int:id>** sa stvarnom vrednošću **id** iz film objekta.

View :

```
def film(request, id):
```

- Za definiciju view-a koja treba da obradi zahtev koji uključuje dinamičke parametre iz URL-a, potrebno je u definiciji funkcije navesti sve te parametre kako bi ih funkcija mogla ispravno obraditi.

Migracije

- **Migracije** su način na koji Django omogućava izmene u šemi baze podataka potrebe za direktnim pristupanjem ili modifikovanjem baze podataka. One predstavljaju set promena koje se primenjuju na bazu kako bi se modeli u Django projektu sinhronizovali sa strukturom baze podataka.
- **Kako se kreiraju migracije:**
 1. Definisanje ili izmena modela
 2. Kreiranje migracionih fajlova i primena migracija:
 - Korišćenjem komande `python manage.py makemigrations`, Django analizira modele i automatski generiše migracione fajlove. Ovi fajlovi sadrže klase koje Django koristi za ažuriranje baze podataka.
 - Korišćenjem komande `python manage.py migrate`, Django prolazi kroz sve migracione fajlove koji nisu primenjeni i izvršava operacije definisane u njima na bazi podataka.

Primer kreiranja modela

The screenshot shows a code editor interface with a blue header bar. On the left, there is a 'Project' sidebar displaying the directory structure of a Django project named 'djangoProject4'. The 'models.py' file in the 'app' directory is selected and highlighted with a grey background. The main editor area shows the Python code for defining a 'Rezervacije' model.

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

class Rezervacije(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    film = models.ForeignKey(Filmovi, on_delete=models.CASCADE)
    broj = models.IntegerField(max_length=1, blank=True, null=True)

    #Definicija str metode koja je specijalna metoda u Pythonu koja se automatski
    #poziva kada se od objekta zahteva njegova string reprezentacija, na primer,
    #kada se objekat prosledi funkciji print() ili se koristi u string kontekstu.
    def __str__(self):
        return self.film
```

Relacije u Django modelima

1. ForeignKey (Mnogo prema Jedan):

- Povezuje jedan objekat sa mnogim objektima druge klase.
- Najčešće se koristi za definisanje relacija gde jedan objekat "poseduje" ili je "roditelj" više objekata.

2. ManyToManyField (Mnogo prema Mnogo):

- Omogućava relaciju u kojoj objekti jedne klase mogu imati veze sa mnogim objektima druge klase i obrnuto.

3. OneToOneField (Jedan prema Jedan):

- Jedan objekat je direktno povezan sa jednim objektom druge klase.
- Često se koristi za proširenje informacija o modelu, kao što je detaljni profil korisnika.

Django Forme

- Forme u Django-u su Python klase koje se koriste za generisanje i obradu HTML formulara. One su moćan alat za automatizaciju zadataka vezanih za podatke koje korisnik unosi, uključujući njihov prijem, validaciju i obradu.

Glavne funkcije formi:

1. Generisanje HTML formulara:

1. Automatsko kreiranje HTML koda za formu na osnovu definisanih polja u formi.
2. Mogućnost prilagođavanja izgleda formi korišćenjem Django widgeta ili prilagođenog HTML-a.

2. Validacija podataka:

1. Provera da li su podaci koje korisnik unosi ispravni i u skladu sa očekivanjima aplikacije.
2. Automatska validacija tipova podataka, obaveznih polja, maksimalne/minimalne dužine i drugih pravila definisanih u formi.

3. Obrada podataka:

1. Prihvatanje i obrada podataka nakon što korisnik pošalje formu.
2. Mogućnost čuvanja podataka direktno u bazu, ažuriranja postojećih zapisa ili njihove obrade na druge načine.

Django Forme

Tipovi forme:

1. Standardne Forme (`forms.Form`):

- Nezavisne od modela baze podataka.

2. Model Forme (`forms.ModelForm`):

- Direktno povezane sa Django modelima.
- Automatizuju kreiranje formi za unos, izmenu i brisanje podataka u modelima.

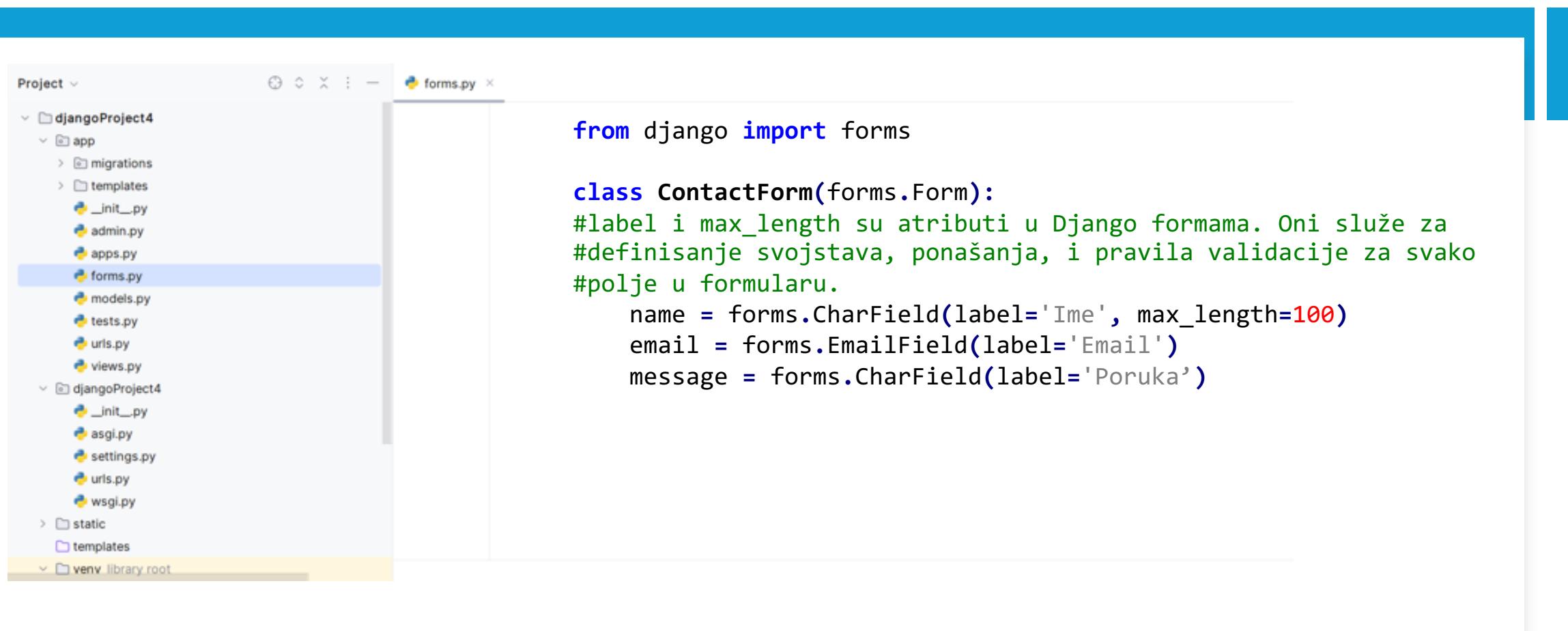
Django Forme

The screenshot shows a code editor interface with a blue header bar. On the left is a sidebar titled "Project" showing the directory structure of a Django project named "djangoProject4". Inside "djangoProject4" is an "app" folder containing files like "migrations", "templates", "__init__.py", "admin.py", "apps.py", and "forms.py" (which is currently selected). Below "app" are "models.py", "tests.py", "urls.py", and "views.py". Further down are "djangoProject4" files ("__init__.py", "asgi.py", "settings.py", "urls.py", "wsgi.py"), and finally "static" and "templates" folders. At the bottom is a "venv" library root. The main editor area has a tab labeled "forms.py" and contains the following Python code:

```
from django import forms

#Definiše se nova klasa ContactForm koja nasleđuje od forms.Form.
#Ovo je osnovna klasa za sve Django forme koji nisu direktno
#povezani sa modelom baze podataka.
class ContactForm(forms.Form):
    name = forms.CharField(label='Ime', max_length=100)
    email = forms.EmailField(label='Email')
    message = forms.CharField(label='Poruka')
```

Django Forme



The screenshot shows a code editor interface with a blue header bar. On the left is a project navigation sidebar titled "Project". It lists several files and folders under "djangoProject4/app": migrations, templates, __init__.py, admin.py, apps.py, forms.py (which is currently selected and highlighted in blue), models.py, tests.py, urls.py, and views.py. Below this, under "djangoProject4", are __init__.py, asgi.py, settings.py, urls.py, and wsgi.py. At the bottom of the sidebar are static and templates folders, and a venv library root entry.

The main editor area has a tab labeled "forms.py". The code inside is:

```
from django import forms

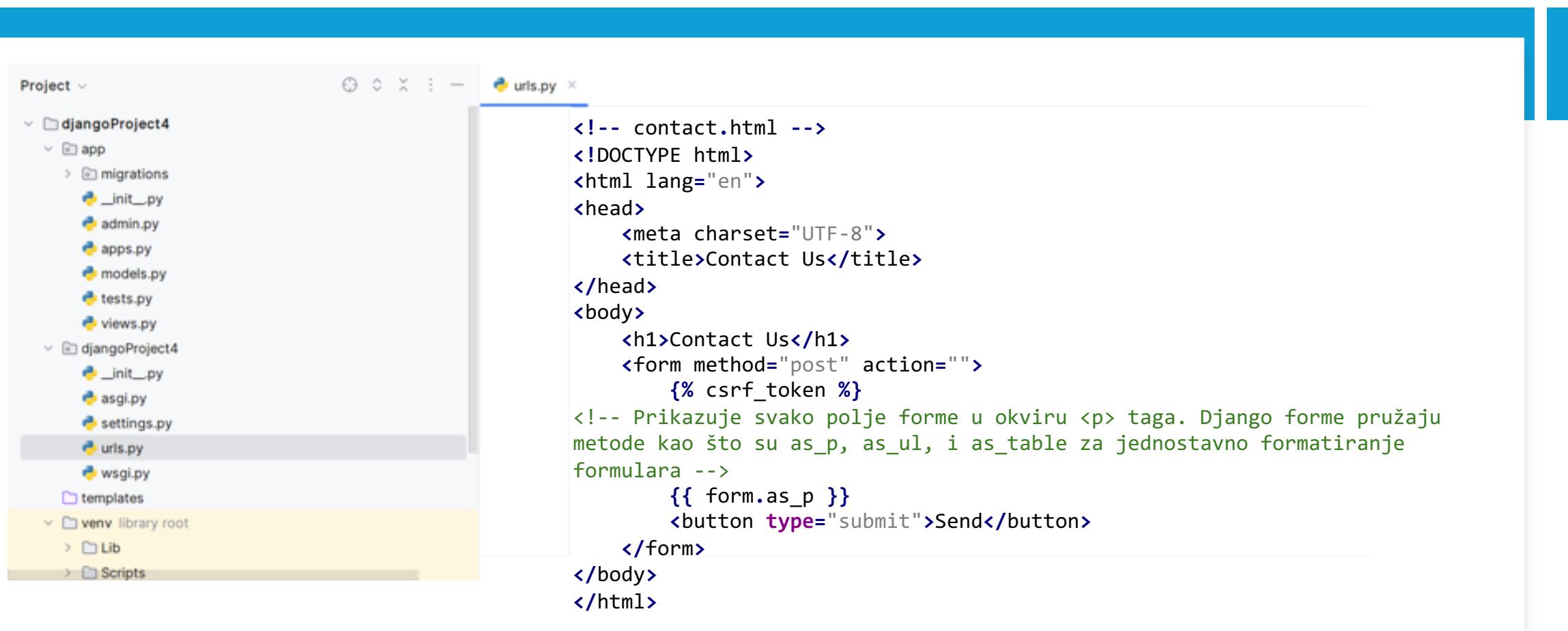
class ContactForm(forms.Form):
    #label i max_length su atributi u Django formama. Oni služe za
    #definisanje svojstava, ponašanja, i pravila validacije za svako
    #polje u formularu.
    name = forms.CharField(label='Ime', max_length=100)
    email = forms.EmailField(label='Email')
    message = forms.CharField(label='Poruka')
```

Django Forme

The screenshot shows a code editor interface with a blue header bar. On the left, there is a sidebar titled "Project" showing the directory structure of a Django project named "djangoProject4". The "urls.py" file in the "app" directory is currently selected and highlighted with a grey background. The main editor area displays the content of "contact.html", which is a Django template. The template includes HTML tags for the document structure, a title, and a form. It also contains Django template tags like `{% csrf_token %}` and `{{ form.as_p }}` to handle CSRF protection and form rendering. A green annotation at the bottom of the template text reads: "Django template tag koji dodaje CSRF token u formular. Ovo je sigurnosna mera koja sprečava CSRF napade-->". The code editor has a light grey background and uses color-coded syntax highlighting for Python and HTML.

```
<!-- contact.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Contact Us</title>
</head>
<body>
    <h1>Contact Us</h1>
    <form method="post" action="">
        <!-- Django template tag koji dodaje CSRF token u formular. Ovo je sigurnosna mera koja sprečava CSRF napade-->
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Send</button>
    </form>
</body>
</html>
```

Django Forme



The screenshot shows a code editor interface with a blue header bar. On the left, there is a sidebar titled "Project" showing the directory structure of a Django project named "djangoProject4". The "urls.py" file is selected in the sidebar, indicated by a grey background. The main editor area displays the content of the "contact.html" template. The template code includes HTML tags like ``, `

`, ``, ``, ``, `Contact Us`, and ``. It also contains an `h1` tag with the text "Contact Us", a form with method="post" and action="", and a CSRF token placeholder. A note in green text at the bottom of the template explains that it shows how each form field is enclosed in a

tag. It also mentions Django's form rendering methods like `as_p`, `as_ul`, and `as_table` for simple styling.

```
<!-- contact.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Contact Us</title>
</head>
<body>
    <h1>Contact Us</h1>
    <form method="post" action="">
        {% csrf_token %}
<!-- Prikazuje svako polje forme u okviru <p> taga. Django forme pružaju metode kao što su as_p, as_ul, i as_table za jednostavno formatiranje formulara -->
        {{ form.as_p }}
        <button type="submit">Send</button>
    </form>
</body>
</html>
```

Django Forme

The screenshot shows a code editor interface with a blue header bar. On the left, there is a 'Project' sidebar displaying the file structure of a Django project named 'djangoProject4'. The 'views.py' file under the 'app' directory is selected and highlighted with a grey background. The main editor area shows the Python code for a view named 'contact'. The code imports necessary modules from Django and defines a function that handles POST requests by creating a 'ContactForm' instance, checking its validity, and rendering a template ('contact.html') with the form object.

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import ContactForm

def contact(request):
    # Proverava da li je metoda zahteva POST, što znači da korisnik šalje podatke
    # formulara.
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            return HttpResponseRedirect('Hvala na poruci!')
    else:
        form = ContactForm()

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

Django Forme

The screenshot shows a code editor interface with a sidebar and a main code area. The sidebar on the left lists project files and folders:

- Project
- djangoProject4
 - app
 - migrations
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py
 - templates
 - venv library root
 - Lib
 - Scritps

Django Forme

The screenshot shows a code editor interface with a blue header bar. On the left, there is a 'Project' sidebar displaying the file structure of a Django project named 'djangoProject4'. The 'views.py' file under the 'app' directory is selected and highlighted with a grey background. The main editor area shows the Python code for a view named 'contact'. The code imports necessary modules from django.shortcuts, django.http, and .forms. It defines a function 'contact' that handles POST requests by creating a 'ContactForm' instance, checking if it's valid, and returning an 'HttpResponse' if valid. If the request is not POST, it creates an empty form. Finally, it renders a template 'contact.html' with the form context.

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        # Provera da li su svi uneti podaci validni (npr. da li su sva obavezna polja
        # popunjena, da li podaci odgovaraju očekivanim formatima itd.).
        if form.is_valid():
            return HttpResponseRedirect('Hvala na poruci!')
    else:
        form = ContactForm()

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

Django Forme

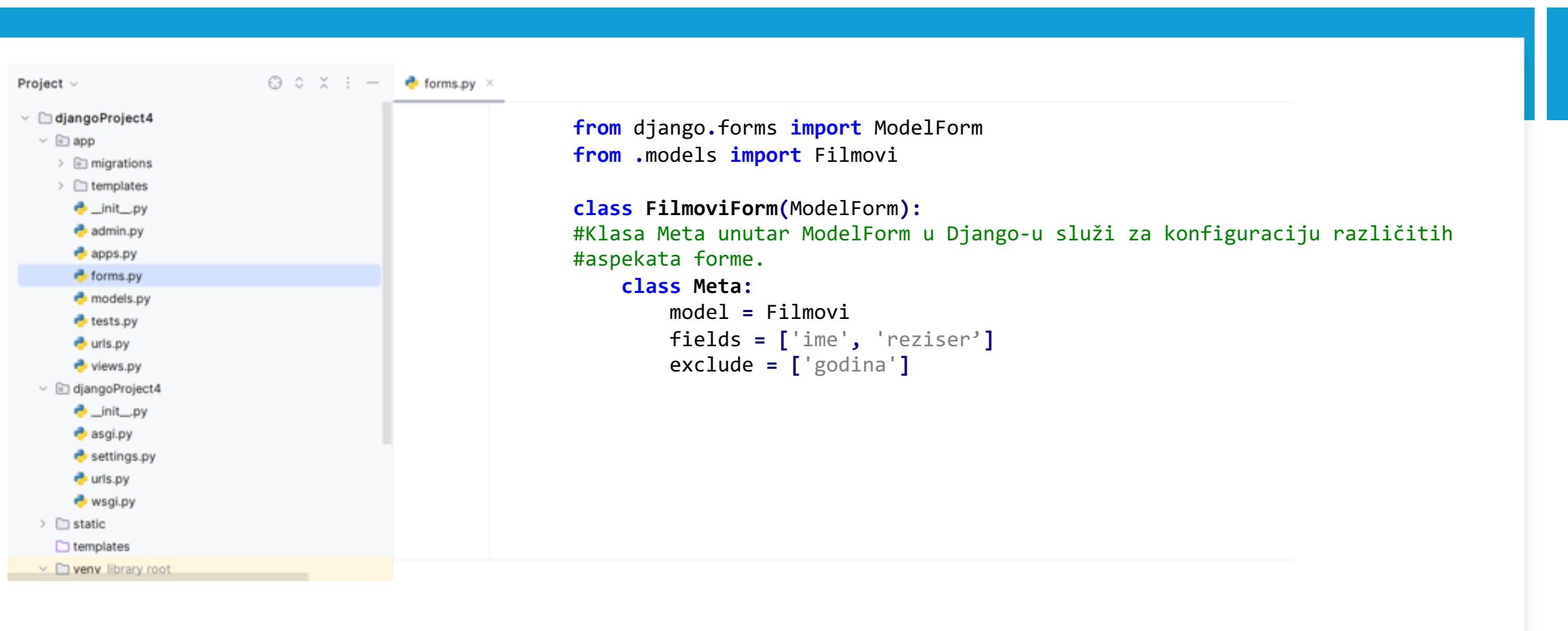
The screenshot shows a code editor interface with a blue header bar. On the left is a 'Project' sidebar listing files and folders for a Django project named 'djangoProject4'. The 'views.py' file is selected and highlighted with a grey bar at the bottom of the sidebar. The main editor area shows the Python code for a view named 'contact'. The code imports necessary modules, defines the 'contact' function, handles POST requests, and renders a template named 'contact.html'.

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
from .forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            return HttpResponseRedirect('Hvala na poruci!')
    else:
        #Pravljenje nove, prazne instance ContactForm bez ikakvih unapred popunjених
        #podataka. Koristi se za inicijalni prikaz formulara na web stranici.
        form = ContactForm()

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

Django Forme

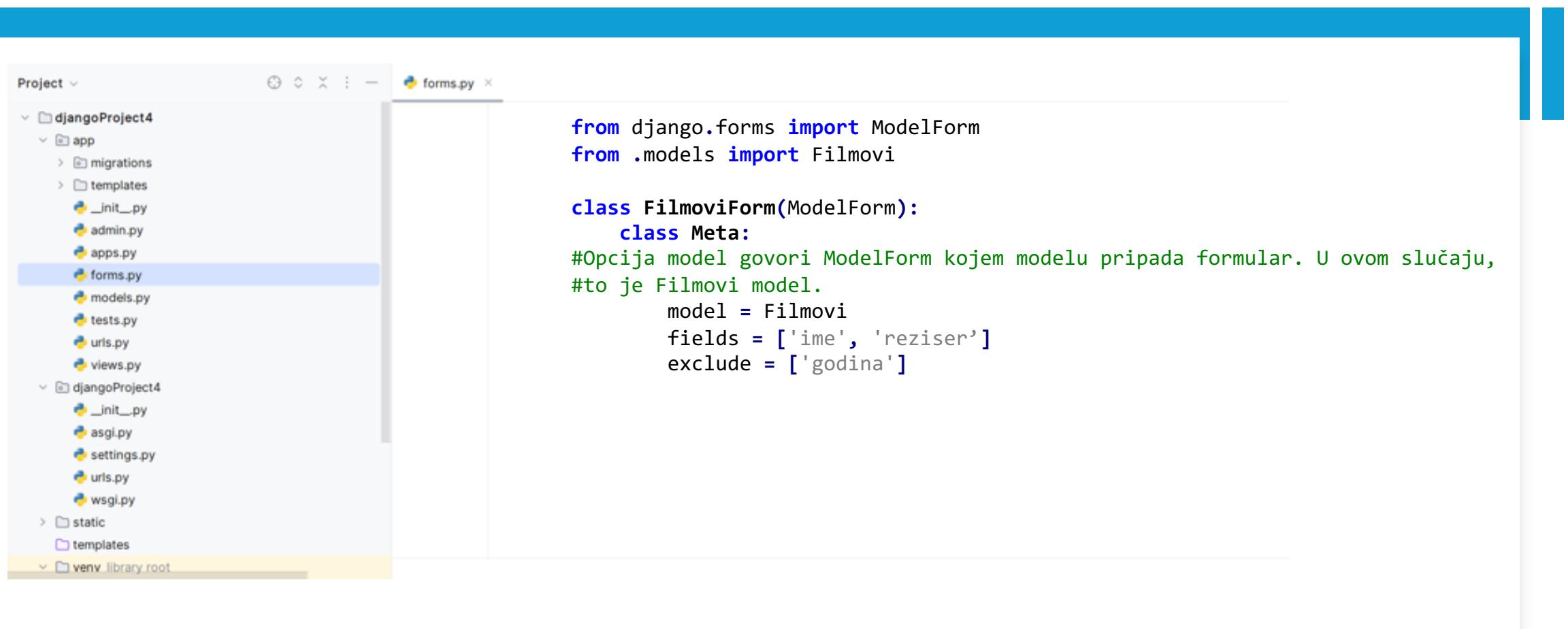


The screenshot shows a code editor interface with a blue header bar. On the left is a project navigation sidebar titled "Project". It lists several files and folders under "djangoProject4/app": migrations, templates, __init__.py, admin.py, apps.py, forms.py (which is currently selected and highlighted in blue), models.py, tests.py, urls.py, and views.py. Below this, there are entries for "djangoProject4" and "venv" (library root). The main editor area has a tab labeled "forms.py". The code inside is as follows:

```
from django.forms import ModelForm
from .models import Filmovi

class FilmoviForm(ModelForm):
    #Klasa Meta unutar ModelForm u Django-u služi za konfiguraciju različitih
    #aspekata forme.
    class Meta:
        model = Filmovi
        fields = ['ime', 'reziser']
        exclude = ['godina']
```

Django Forme



The screenshot shows a code editor interface with a blue header bar. On the left is a project navigation sidebar titled "Project". It lists several files and folders under "djangoProject4/app": migrations, templates, __init__.py, admin.py, apps.py, forms.py (which is currently selected and highlighted in blue), models.py, tests.py, urls.py, and views.py. Below this, under "djangoProject4", are __init__.py, asgi.py, settings.py, urls.py, and wsgi.py. At the bottom of the sidebar are static and templates folders, and a venv library root entry.

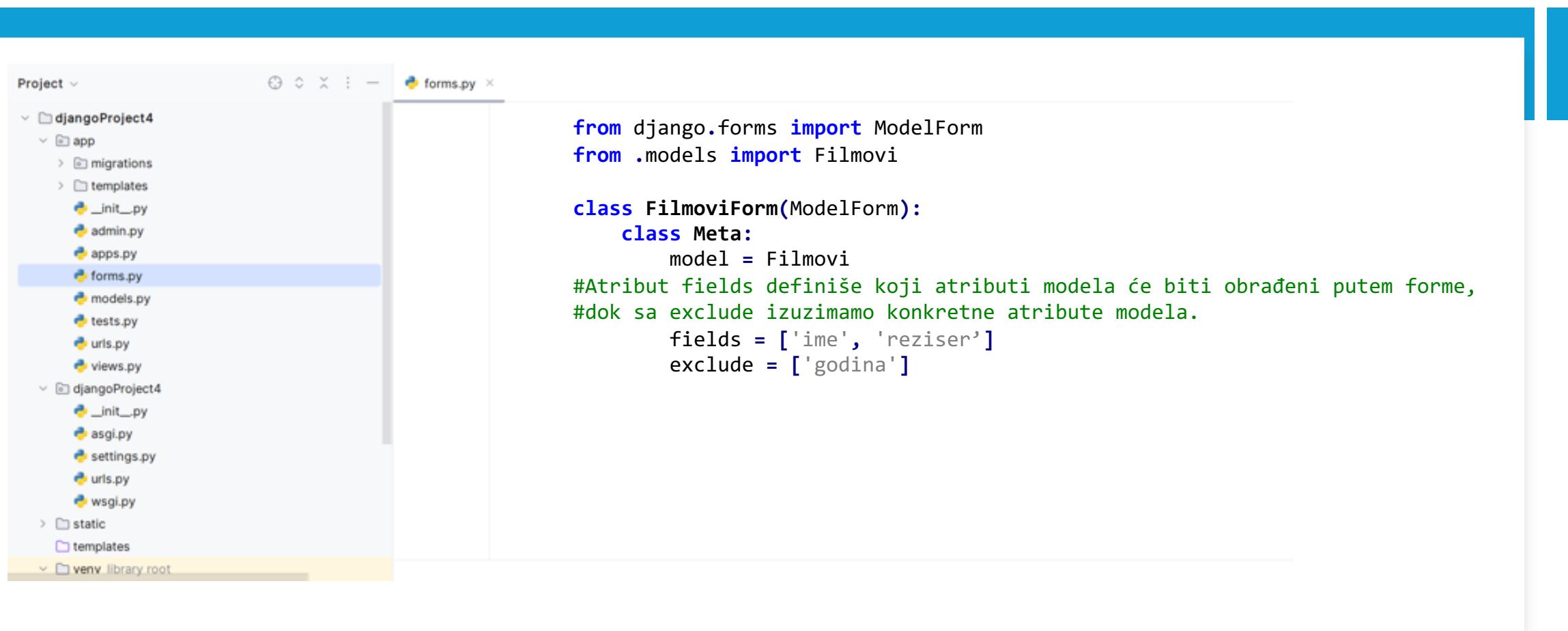
The main editor area has a tab labeled "forms.py". The code inside is:

```
from django.forms import ModelForm
from .models import Filmovi

class FilmoviForm(ModelForm):
    class Meta:
        #Opcija model govori ModelForm kojem modelu pripada formular. U ovom slučaju,
        #to je Filmovi model.
        model = Filmovi
        fields = ['ime', 'reziser']
        exclude = ['godina']
```

A green annotation is present in the code, reading: "#Opcija model govori ModelForm kojem modelu pripada formular. U ovom slučaju, #to je Filmovi model."

Django Forme



The screenshot shows a code editor interface with a blue header bar. On the left is a project navigation sidebar titled "Project". It lists several files and folders under "djangoProject4/app": migrations, templates, __init__.py, admin.py, apps.py, forms.py (which is currently selected and highlighted in blue), models.py, tests.py, urls.py, and views.py. Below this, there are entries for "djangoProject4" and "venv library root". The main editor area has a tab labeled "forms.py". The code inside is:

```
from django.forms import ModelForm
from .models import Filmovi

class FilmoviForm(ModelForm):
    class Meta:
        model = Filmovi
    #Atribut fields definiše koji atributi modela će biti obrađeni putem forme,
    #dok sa exclude izuzimamo konkretnе atribute modela.
        fields = ['ime', 'reziser']
        exclude = ['godina']
```

Django Admin

Django admin je jedan od najmoćnijih i najpopularnijih alata koje Django pruža, služeći kao gotov backend interfejs za upravljanje podacima aplikacije. Omogućava administratorima da lako rukuju podacima poput korisnika, grupe i permisija, kao i modela definisanih u samoj aplikaciji.

- Pre korišćenja Django admina prvo je potrebno izvršiti migracije da bi se postavile tabele u bazi podataka: tabele za autentifikaciju i autorizaciju, tabela za menadžment sesijama, tabela za administraciju i tabele za poruke

- Nakon toga potrebno je kreirati korisnika koji ima sve permisije, odnosno superusera komandom: `python manage.py createsuperuser`.

Username: korisnik

Email address: korisnik@gmail.com

Password:

Password (again):

This password is too short. It must contain at least 8 characters.

This password is too common.

This password is entirely numeric.

Bypass password validation and create user anyway? [y/N]: y

Superuser created successfully.

Django Admin

U Django admin panelu, može se lako upravljati korisnicima, što uključuje:

- Dodavanje novih korisnika:** Klikom na "Add" pored "Users" u sekciji "AUTHENTICATION AND AUTHORIZATION".
- Izmena postojećih korisnika:** Klikom na korisničko ime koje želite izmeniti.
- Brisanje korisnika:** Označavanjem korisnika koje želite obrisati i korišćenjem akcije "Delete selected users".

Django administration

Home · Authentication and Authorization · Users · Add user

Add user

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

Username:

Required: 150-characters or fewer. Letters, digits and @/./~/_, only.

Password:

Your password can't be too similar to your other personal information.

Your password must contain at least 8 characters.

Your password can't be a commonly used password.

Your password can't be entirely numeric.

Password confirmation:

Enter the same password as before, for verification.

[Save](#) [Save and add another](#) [Save and continue editing](#)

Django administration

Home · Authentication and Authorization · Users · korisnik

Change user

korisnik

Username:

Required: 150-characters or fewer. Letters, digits and @/./~/_, only.

Password:

algorithm: pbkdf2_sha256 iterations: 720000 salt: JcHnz----- hash: 3M4wfd-----

Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

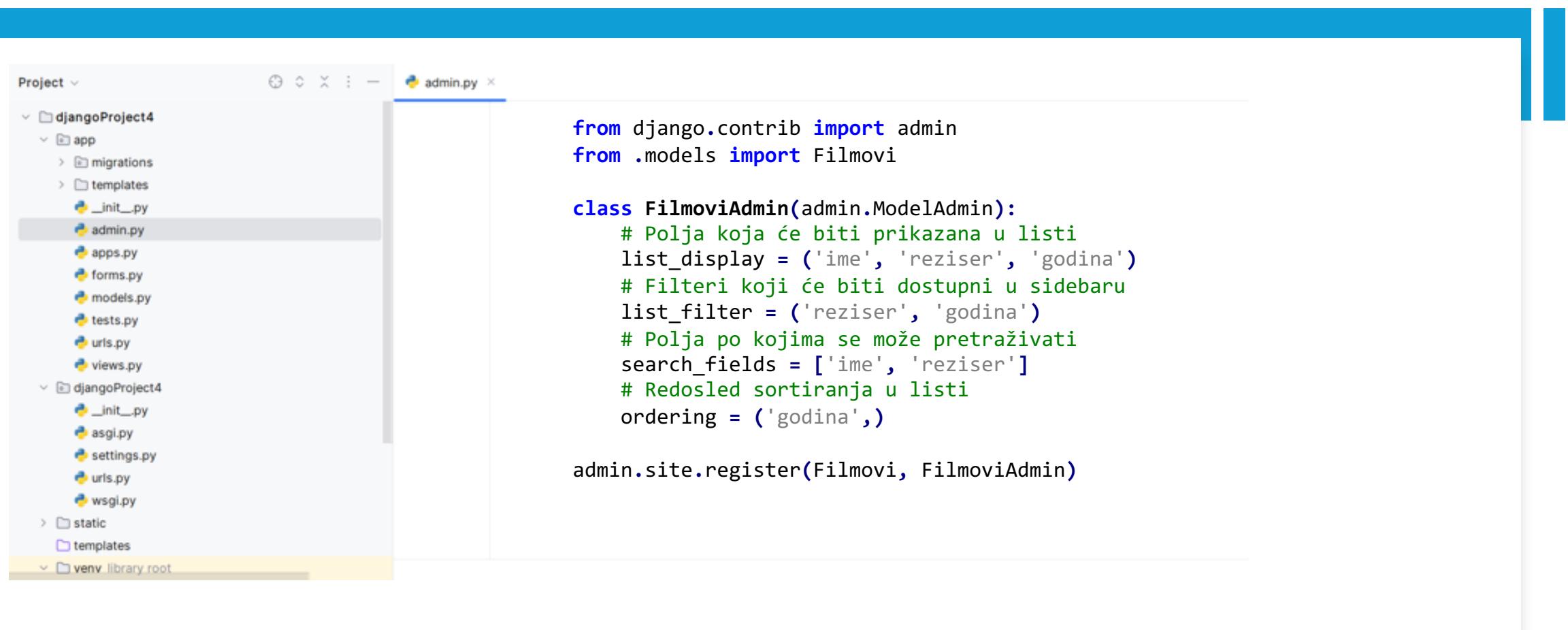
Personal info

First name:

Last name:

Email address:

Django Admin – prikaz modela



The screenshot shows a code editor interface with a sidebar and a main code area. The sidebar on the left displays a project structure:

- Project
- djangoProject4
 - app
 - migrations
 - templates
 - __init__.py
 - admin.py
 - apps.py
 - forms.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py
 - static
 - templates
 - venv library root

The main code area contains Python code for configuring a Django admin model:

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

class FilmoviAdmin(admin.ModelAdmin):
    # Polja koja će biti prikazana u listi
    list_display = ('ime', 'reziser', 'godina')
    # Filteri koji će biti dostupni u sidebaru
    list_filter = ('reziser', 'godina')
    # Polja po kojima se može pretraživati
    search_fields = ['ime', 'reziser']
    # Redosled sortiranja u listi
    ordering = ('godina',)

admin.site.register(Filmovi, FilmoviAdmin)
```

Django Admin – prikaz modela

Django administration

WELCOME, TEODORA. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Filmovi › Filmovis

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

FILMOVI

Filmovis + Add

Select filmovi to change

Action: ----- Go 0 of 10 selected

<input type="checkbox"/> IME	REZISER	GODINA
<input type="checkbox"/> The Godfather	Francis Ford Coppola	1972
<input type="checkbox"/> Schindler's List	Steven Spielberg	1993
<input type="checkbox"/> Forrest Gump	Robert Zemeckis	1994
<input type="checkbox"/> The Shawshank Redemption	Frank Darabont	1994
<input type="checkbox"/> Pulp Fiction	Quentin Tarantino	1994
<input type="checkbox"/> Titanic	James Cameron	1997
<input type="checkbox"/> The Matrix	Lana Wachowski, Lilly Wachowski	1999
<input type="checkbox"/> Fight Club	David Fincher	1999

ADD FILMOVI +

FILTER

Show counts

I By reziser

All

- Christopher Nolan
- David Fincher
- Francis Ford Coppola
- Frank Darabont
- James Cameron
- Lana Wachowski, Lilly Wachowski
- Quentin Tarantino
- Robert Zemeckis
- Steven Spielberg

I By godina

All

- 1972
- 1993

<input type="checkbox"/> IME	REZISER	GODINA
<input type="checkbox"/> The Godfather	Francis Ford Coppola	1972
<input type="checkbox"/> Schindler's List	Steven Spielberg	1993
<input type="checkbox"/> Forrest Gump	Robert Zemeckis	1994
<input type="checkbox"/> The Shawshank Redemption	Frank Darabont	1994
<input type="checkbox"/> Pulp Fiction	Quentin Tarantino	1994
<input type="checkbox"/> Titanic	James Cameron	1997
<input type="checkbox"/> The Matrix	Lana Wachowski, Lilly Wachowski	1999
<input type="checkbox"/> Fight Club	David Fincher	1999

Upravljanje korisnicima u Django

Django pruža ugrađeni sistem za upravljanje korisnicima, koji uključuje autentifikaciju, registraciju, prijavu i odjavu korisnika.

- Ove funkcionalnosti su deo *django.contrib.auth* modula, koji olakšava implementaciju sigurnih i efikasnih metoda za upravljanje korisničkim sesijama i pristupima.

Koristićemo : *create_user()*, *authenticate()*, *login()*, *logout()*

Django nudi i niz ugrađenih formulara koji su specijalizovani za različite aspekte autentifikacije i upravljanja korisnicima. Formulari kao što su *AuthenticationForm*, *UserCreationForm*, i drugi, olakšavaju implementaciju standardnih operacija kao što su prijava, registracija, i izmena korisničkih podataka. Ovi formulari dolaze sa preddefinisanom logikom i validacijama.

Upravljanje korisnicima – login1

The screenshot shows a code editor interface with a sidebar and a main code area. The sidebar on the left displays the project structure:

- Project
- djangoProject4
 - app
 - migrations
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py
 - templates
 - venv library root
 - Lib
 - Scripts

Upravljanje korisnicima – login1

The screenshot shows a code editor interface with a sidebar and a main code area. The sidebar on the left displays the project structure:

- Project
- djangoProject4
 - app
 - migrations
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - tests.py
 - urls.py
 - views.py
 - djangoProject4
 - __init__.py
 - asgi.py
 - settings.py
 - urls.py
 - wsgi.py
 - templates
 - venv
 - library root
 - Lib
 - Scripts

The main code area shows the content of `views.py`:

```
def login_user(request):
    if request.user.is_authenticated:
        return redirect('home')
    if request.method == 'POST':
        #Ako je zahteva POST, dohvataju se korisničko ime i lozinka iz podataka formulara.
        username = request.POST.get('username')
        password = request.POST.get('password')
        try:
            user = User.objects.get(username=username)
        except:
            mess='Ne postoji korisnik'
        user = authenticate(username=username, password=password)
        if user:
            login(request, user)
            return redirect('home')
        else:
            mess='Neispravna lozinka'
    context = {'mess': mess}
    return render(request, 'filmovi/login.html', context)
```

Upravljanje korisnicima – login1

The screenshot shows a code editor interface with a blue header bar. On the left is a file tree for a Django project named "djangoProject4". The "views.py" file under the "app" directory is selected and highlighted with a yellow background. The code editor window shows the Python code for a login view:

```
def login_user(request):
    if request.user.is_authenticated:
        return redirect('home')
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        try:
            #Pokušava se da se pronađe korisnik sa unetim korisničkim imenom.
            user = User.objects.get(username=username)
        except:
            mess='Ne postoji korisnik'
        user = authenticate(username=username, password=password)
        if user:
            login(request, user)
            return redirect('home')
        else:
            mess='Neispravna lozinka'
    context = {'mess': mess}
    return render(request, 'filmovi/login.html', context)
```

Upravljanje korisnicima – login1

The screenshot shows a code editor interface with a sidebar labeled "Project". The sidebar lists the structure of a Django project named "djangoProject4". It includes an "app" directory containing files like migrations, __init__.py, admin.py, apps.py, models.py, tests.py, urls.py, and views.py. Below the app is another "djangoProject4" directory with __init__.py, asgi.py, settings.py, urls.py, and wsgi.py. There are also "templates", "venv library root", and "Scripts" sections.

The main editor window is titled "views.py" and contains the following Python code:

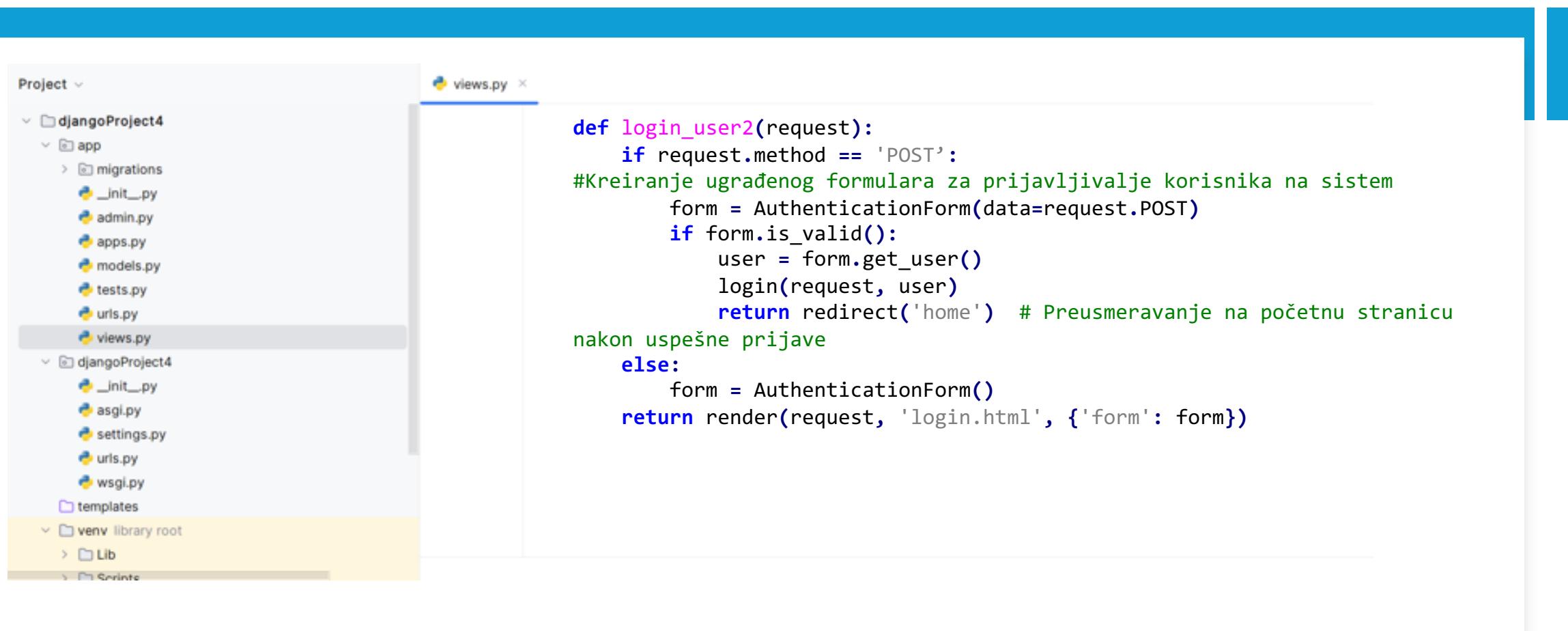
```
def login_user(request):
    if request.user.is_authenticated:
        return redirect('home')
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        try:
            user = User.objects.get(username=username)
        except:
            mess='Ne postoji korisnik'
            # Autentifikacija korisnika pomoću unetog korisničkog imena i lozinke pomoću već
            # implementiranu Django funkcionalnosti, ako postoji takav korisnik, izvršavamo
            # logovanje na sistem i redirektujemo korisnika na home stranicu.
            user = authenticate(username=username, password=password)
            if user:
                login(request, user)
                return redirect('home')
            else:
                mess='Neispravna lozinka'
                context = {'mess': mess}
                return render(request, 'filmove/login.html', context)
```

Upravljanje korisnicima – login 1

The screenshot shows a code editor interface with a blue header bar. On the left is a sidebar titled 'Project' showing the directory structure of a Django project named 'djangoProject4'. The 'templates/app/app' folder contains several files: 'base.html', 'base_static_files.html', 'film_prikaz.html', 'filmovi.html', and 'login.html', which is currently selected and highlighted in grey. The main editor area displays the content of 'login.html'.

```
{% extends './base.html' %}  
{% block title %}Login{% endblock %}  
{% block content %}  
<div>  
    <form method="POST" action="">  
        {% csrf_token %}  
        <label>Username:</label>  
        <input type="text" name="username" placeholder="Enter Username..."/>  
  
        <label>Password:</label>  
        <input type="password" name="password" placeholder="Enter Password..."/>  
  
        <input type="submit" value="Login"/>  
    </form>  
    Click here to <a href="{% url 'register' %}">register</a></p>  
</div>  
{% endblock %}
```

Upravljanje korisnicima – login2



The screenshot shows a code editor interface with a blue header bar. On the left is a project tree titled "Project". It lists several files and folders under "djangoProject4" and "app". The file "views.py" is selected and highlighted with a yellow background. The code editor window has a tab labeled "views.py" at the top. The code itself is written in Python and defines a function named "login_user2". The code is annotated with comments in green. The code logic handles a POST request, creates an AuthenticationForm, checks if it's valid, logs the user in, and redirects them to the home page. If the form is not valid, it renders the "login.html" template with the form.

```
def login_user2(request):
    if request.method == 'POST':
        #Kreiranje ugrađenog formulara za prijavljivalje korisnika na sistem
        form = AuthenticationForm(data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('home') # Preusmeravanje na početnu stranicu
        nakon uspešne prijave
    else:
        form = AuthenticationForm()
    return render(request, 'login.html', {'form': form})
```

Upravljanje korisnicima – login2

The screenshot shows a code editor interface with a blue header bar. On the left is a project navigation pane titled "Project" showing the structure of a Django project named "djangoProject4". The "app" directory contains files like "migrations", "templates", and "base.html". Inside "templates", there is a "login.html" file which is currently selected and shown in the main editor area. The "login.html" file contains the following code:

```
{% extends './base.html' %}  
{% block title %}Login{% endblock %}  
{% block content %}  
    <div>  
        <form method="post">  
            {% csrf_token %}  
            {{ form.as_p }}  
            <button type="submit">Login</button>  
        </form>  
        {% if form.non_field_errors %}  
            <div>  
                {{ form.non_field_errors }}  
            </div>  
        {% endif %}  
    </div>  
{% endblock %}
```

Upravljanje korisnicima – register

The screenshot shows a code editor interface with a blue header bar. On the left is a 'Project' sidebar displaying the directory structure of a Django project named 'djangoProject4'. The 'views.py' file under the 'app' folder is selected and highlighted with a grey background. The main editor area shows the Python code for a 'register_user' view function.

```
def register_user(request):
    #Kreiranje ugrađenog formulara za kreiranje korisnika
    form = UserCreationForm()
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('home')
        else:
            print(form.non_field_errors())
    context = {'form': form}
    return render(request, 'filmovei/register.html', context)
```

Upravljanje korisnicima – register

The screenshot shows a code editor interface with a blue header bar. On the left, there is a 'Project' sidebar displaying the directory structure of a Django project named 'djangoProject4'. The 'templates/app' folder contains several HTML files: base.html, base_static_files.html, film_prikaz.html, filmovi.html, login.html, and register.html. The 'register.html' file is currently selected and shown in the main editor area.

```
{% extends './base.html' %}  
{% block title %}Register{% endblock %}  
{% block content %}  
    <div> <form method="POST" action="">  
        {% csrf_token %}  
        {{form.as_p}}  
        <input type="submit" value="Register"/>  
  
        <p><a href="{% url 'login' %}">Login</a></p>  
    </form>  
</div>  
{% endblock %}
```

Kontrola pristupa

Provera u Template-ima:

```
{% if request.user.is_authenticated %}
```

- Opis: Ovaj template tag omogućava da u HTML template-ima uslovno prikažete sadržaj zavisno od toga da li je korisnik trenutno prijavljen (autentifikovan) ili ne.
- Primena: Koristi se za prikaz ili skrivanje delova web stranice koji su dostupni samo prijavljenim korisnicima.

```
{% if request.user.is_authenticated %}  
    <p> Hello, {{ request.user.username }}!</p>  
{% else %}  
    <p> Login <a href="{% url 'login'  
%}"></a>.</p>  
{% endif %}
```

Kontrola pristupa

Dekorator `@login_required`:

Ograničavanje pristupa view funkcijama

- Opis: Dekorator `@login_required` se primjenjuje na view funkcije kako bi se ograničio pristup samo prijavljenim korisnicima.
- Funkcionalnost: Ako korisnik koji nije priavljen pokuša da pristupi view-u koji je zaštićen ovim dekoratorom, biće preusmeren na stranicu za prijavu.
- Konfiguracija: Moguće je podesiti URL za preusmeravanje za neautentifikovane korisnike modifikovanjem `LOGIN_URL` u Django settings.

```
from django.contrib.auth.decorators import  
login_required  
  
@login_required  
def page(request):  
    return render(request, 'page.html')
```

Prilagođavanje Django User modela

The screenshot shows a code editor with a sidebar displaying the project structure of a Django application named 'djangoProject4'. The 'models.py' file is selected in the sidebar and is currently being edited.

```
Project
  - djangoProject4
    - app
      - migrations
      - __init__.py
      - admin.py
      - apps.py
      - models.py
      - tests.py
      - urls.py
      - views.py
    - djangoProject4
      - __init__.py
      - asgi.py
      - settings.py
      - urls.py
      - wsgi.py
    - templates
    - venv library root
      - Lib
      - Scripts

models.py
```

The code in the 'models.py' file defines a custom user profile model:

```
class UserProfile(models.Model):
    #Kreira se relacija jedan na jedan sa Django korisnikom, pri brisanju
    #korisnika obezbedili smo da se briše i modifikovani model korisnika.
    #Postavljanjem null na True obezbeđujemo postojanje korisnika bez potrebe da
    #odmah kreiramo i novog, modifikovanog korisnika.
    user = models.OneToOneField(User, null=True, on_delete=models.CASCADE)
    phone_number = models.CharField(max_length=15, blank=True)
    bio = models.TextField(blank=True)

    #Obezbeđujemo da se pri ispisu modifikovanog korisnika ispisuje njegovo
    #korisničko ime.
    def __str__(self):
        return self.user.username
```

A callout box with a blue border contains the following text:

Da bi se lakše upravljalo profilima korisnika potrebno je registrovati ovaj model u Django admin panelu:

```
admin.site.register(UserProfile)
```