Prikazi zasnovani na klasama

U ovom odeljku ćete naučiti prikaze zasnovane na klasama tako što ćete napraviti aplikaciju za listu zadataka koja omogućava korisnicima da se registruju, prijavljuju, resetuju lozinke, kreiraju profile i upravljaju sopstvenim zadacima.

Sadržaj

Todo aplikacija

U ovom tutorijalu ćete naučiti kako da kreirate projekat Todo aplikacije, uključujući:

- Kreiranje virtuelno okruženje
- Instalirate Django paket
- Napravite novi projekat
- Dodate statičkih datoteka
- Podesite šablone
- Napravite aplikaciju sa obavezama
- Kreirate model zadatka i primenite migracije

Kreiranje virtuelnog okruženja

Pokrenite sledeću komandu u shell-u da biste kreirali virtuelno okruženje koristeći ugrađeni `venv` modul:

```
python3 -m venv venv
```

Aktivirajte `venv` virtuelno okruženje pomoću sledeće komande:

```
venv\scripts\activate
```

Instaliranje Django paketa

Pošto je Django paket treće strane, potrebno ga je instalirati pomoću sledeće `pip` komande:

```
pip install django
```

Kreiranje novog Todo_list projekta

Da biste kreirali novi projekat `todo_list`, koristite `startproject` komandu:

```
django-admin startproject todo_list
```

Dodavanje static direktorijuma projektu

Kreirajte `static` direktorijum unutar direktorijuma projekta:

```
mkdir static
```

Podesite `STATICFILES_DIRS` i `STATIC_URL` na `static` direktorijum u `settings.py` datoteci projekta kako bi Django mogao da pronađe statičke datoteke projekta:

```
STATIC_URL = 'static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
```

Kreirajte tri direktorijuma `js`, `css` i `images` unutar `static` direktorijuma:

```
cd static
mkdir css images js
```

Direktorijum `static` će izgledati ovako:

```
├── static
|   ├── css
|   ├── images
|   └── js
```

Na kraju, kopirajte `style.css` datoteku i `feature.jpg` sliku iz datoteke za preuzimanje u `css` i `images` direktorijume.

Podešavanje šablona projekta

Kreirajte `templates` direktorijum unutar direktorijuma projekta:

```
mkdir templates
```

Kreirajte `base.html` šablon unutar `templates` direktorijuma sa sledećim sadržajem:

```
{%load static %}

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

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{% static 'css/style.css' %}" />
    <title>Todo List</title>
  </head>

  <body>
    <header class="header">
      <div class="container">
      </div>
    </header>
    <main>
      <div class="container">
      </div>
    </main>
    <footer class="footer">
      <div class="container">
        <p>&copy; Copyright {% now "Y" %} by <a href="https://
www.pythontutorial.net">Python Tutorial</a></p>
      </div>
    </footer>
  </body>

</html>
```

Šablon `base.html` koristi `style.css` datoteku iz `static/css` direktorijuma.

Konfigurišite direktorijum `TEMPLATES` u datoteci `settings.py` na `templates` tako da Django može da pronađe `base.html` šablon.

```
TEMPLATES = [
  {
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [BASE_DIR / 'templates' ],
    'APP_DIRS': True,
    'OPTIONS': {
      'context_processors': [
        'django.template.context_processors.debug',
        'django.template.context_processors.request',
        'django.contrib.auth.context_processors.auth',
        'django.contrib.messages.context_processors.messages',
      ],
    },
  },
]
```

Kreirajte `home.html` šablon unutar `templates` direktorijuma:

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

{%load static %}

{%block content%}

  <section class="feature">
    <div class="feature-content">
      <h1>Todo</h1>
      <p>Todo helps you more focus, either work or play.</p>
      <a class="btn btn-primary cta" href="#">Get Started</a>
    </div>
    <img src="{%static 'images/feature.jpg'%}" alt="" class="feature-image">
  </section>

{%endblock content%}
```

Kreiranje Todo aplikacije

Kreirajte `todo` aplikaciju u `todo_list` projektu koristeći `startapp` komandu:

```
django-admin startapp todo
```

Registrujte `todo` aplikaciju u `settings.py` projekta `todo_list` tako što ćete je dodati na `INSTALLED_APPS` listu:

```
INSTALLED_APPS = [
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'todo',
]
```

Kreirajte `templates` direktorijum unutar `todo` direktorijuma aplikacije:

```
cd todo
mkdir templates
```

Kreirajte `todo` direktorijum unutar `templates` direktorijuma. Naziv direktorijuma mora biti isti kao i naziv aplikacije.

```
cd templates
mkdir todo
```

Definišite `home()` funkciju prikaza unutar `views.py` aplikacije zadataka koja prikazuje `home.html` šablon:

```
from django.shortcuts import render

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

Kreirajte `urls.py` datoteku u `todo` aplikaciji i definišite rutu koja mapira na početnu URL adresu `views.home()` funkciju prikaza:

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

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

Uključite `urls.py` aplikacije `todo` u `urls.py` projekta:

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

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

Pokrenite Django razvojni server iz `todo_list` direktorijuma:

```
python manage.py runserver
```

Konačno, otvorite http://127.0.0.1:8000/ u veb pregledaču, videćete Home stranicu Todo aplikacije.

Kreiranje Task modela

Definišite `Task` model u `models.py` aplikacije `todo`:

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

class Task(models.Model):
  title = models.CharField(max_length=255)
  description = models.TextField(null=True, blank=True)
  completed = models.BooleanField(default=False)
  created_at = models.DateTimeField(auto_now_add=True)
  user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True)

  def __str__(self):
    return self.title

  class Meta:
    ordering = ['completed']
```

Registrujte `Task` model u `admin.py` aplikaciji `todo` kako biste mogli da ga upravljate modelom na administratorskoj stranici.

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

admin.site.register(Task)
```

Izvršite migracije pokretanjem `makemigrations` komande:

```
python manage.py makemigrations

Migrations for 'todo':
  todo\migrations\0001_initial.py
  - Create model Task
```

Primenite migracije na bazu podataka:

```
python manage.py migrate

Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, todo
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
  Applying todo.0001_initial... OK
```

Kreirajte korisnika `superuser` izvršavanjem `createsuperuser` komande:

```
python manage.py createsuperuser

Username: john
Email address:
Password:
Password (again):
Superuser created successfully.
```

Ponovo pokrenite Django razvojni server:

```
python manage.py runserver
```

Prijavite se na administratorsku stranicu i kreirajte tri zadatka.

Sadržaj

ListView

U ovom tutorijalu ćete naučiti kako da koristite `ListView` klasu za prikazivanje `Todo` liste aplikacije `Todo`.

U prethodnim tutorijalima ste naučili kako da napravite aplikaciju koristeći prikaze zasnovane na funkcijama.

Prikazi zasnovani na funkcijama su jednostavni i fleksibilni. U ranijim verzijama, Django je podržavao samo prikaze zasnovane na funkcijama. Kasnije je Django dodao podršku za prikaze zasnovane na klasama koji vam

omogućavaju da definišete prikaze pomoću klasa.

Prikazi zasnovani na klasama su alternativni način implementacije prikaza. Oni ne zamenjuju prikaze zasnovane na funkcijama. Međutim, imaju neke prednosti u poređenju sa prikazima zasnovanim na funkcijama:

- Organizuju kod povezan sa HTTP metodama kao što su GET i POST koristeći odvojene metode, umesto uslovnog grananja u istoj funkciji.
- Iskorišćavaju višestruko nasleđivanje da bi kreirali klase prikaza koje se mogu ponovo koristiti.

Koristićemo prikaze zasnovane na klasama da bismo izgradili `Todo` aplikaciju.

ListView prikaz

Da biste prikazali listu objekata, definišete klasu koja nasleđuje `ListView` klasu. Na primer, sledeće definiše `TaskList` klasu u `views.py` aplikaciji `todo`:

```
from django.shortcuts import render
from django.views.generic.list import ListView
from .models import Task

class TaskList(ListView):
  model = Task
  context_object_name = 'tasks'
# ...
```

`TaskList` je prikaz zasnovan na klasi koji nasleđuje od `ListView` klase. U `TaskList` klasi definišemo sledeće atribute:

- `model` određuje model iz koga će objekti biti prikazani. U ovom primeru koristimo `Task` model. Interno, Django će praviti upit za sve objekte iz `Task` modela (`Task.objects.all()`) i prosleđivati ih šablonu.
- `context_object_name` određuje naziv promenljive liste objekata modela u šablonu. Podrazumevano, Django koristi `object_list` . Međutim, naziv `object_list` je prilično generički. Stoga, poništavamo `context_object_name` postavljanjem vrednosti na `tasks` .

Po konvenciji, `TaskList` klasa će učitati `todo/task_list.html` šablon. Ime šablona prati ovu konvenciju:

```
appname/modelname_list.html
```

Ako želite da podesite drugačije ime, možete koristiti `template_name` atribut. U ovom tutorijalu ćemo koristiti podrazumevano ime šablona, koje je `task_list.html` .

ListView ruta

Promenite `ListView` rutu u `urls.py` aplikacije `todo` na sledeći način:

```
from django.urls import path
from .views import home, TaskList

urlpatterns = [
  path('', home, name='home'),
  path('tasks/', TaskList.as_view(), name='tasks'),
]
```

U ovom kodu, mapiramo URL adresu `tasks/` na rezultat metode `as_view()` klase `TaskList`. Imajte na umu da možete navesti atribute klase `TaskList` u `as_view()` metodi. >Na primer, možete proslediti ime šablona `as_view()` metodi na sledeći način:

```
> path('tasks/', TaskList.as_view(template_name='mytodo.html'), name='tasks'),
```

Metoda `as_view()` ima argumente koji odgovaraju atributima klase `TaskList`.

ListView šablon

Definišite `task_list.html` u `templates/todo` direktorijumu aplikacije `Todo`:

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

{%block content%}

<div class="center">
  <h2>My Todo List</h2>
  {% if tasks %}
  <ul class="tasks">
    {% for task in tasks %}
      <li><a href="#" class="{% if task.completed%}completed{%endif%}">{{
task.title }}</a>
        <div  class="task-controls">
          <a href="#"><i class="bi bi-trash"></i> </a>
          <a href="#"><i class="bi bi-pencil-square"></i></a>
        </div>
      </li>
    {% endfor %}
  {% else %}
    <p>🎉 Yay, you have no pending tasks!</p>
  {% endif %}
  </ul>
</div>

{%endblock content%}
```

Šablon `task_list.html` proširuje `base.html` šablon projekta. U `task_list.html` šablonu, iteriramo kroz `tasks` `QuerySet` i prikazujemo svaki od njih kao stavku na listi.

Takođe, dodajemo `completed` css klasu tagu `a` ako je zadatak završen. Ova css klasa će dodati liniju stavci.

Ako je `tasks` prazan, prikazuje se poruka da nema zadataka na čekanju.

ListView link

Izmenite `base.html` šablon da biste uključili `My Tasks` vezu u navigaciju:

```
{%load static %}

<!DOCTYPE html>
<html lang="en">
  <head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="{% static 'css/style.css' %}" />
    <title>Todo List</title>
  </head>

  <body>
    <header class="header">
    <div class="container">
      <a href="{%url 'home'%}" class="logo">Todo</a>
      <nav class="nav">
        <a href="{%url 'home'%}"><i class="bi bi-house-fill"></i> Home</a>
        <a href="{% url 'tasks' %}"><i class="bi bi-list-task"></i> My Tasks</a>
      </nav>
    </div>
    </header>
    <main>
    <div class="container">
     {%block content %}
     {%endblock content%}
    </div>
    </main>
    <footer class="footer">
    <div class="container">
       <p>© Copyright {% now "Y" %} by <a href="https://
www.pythontutorial.net">Python Tutorial</a></p>
    </div>
    </footer>
  </body>
</html>
```

Ako otvorite URL adresu: http://128.0.0.1:8000/tasks/ videćete listu zadataka.

Rezime ListView

- Napravite prikaz zasnovan na klasi koji prikazuje listu objekata nasleđivanjem iz `ListView` klase.

Sadržaj

DetailView

U ovom tutorijalu ćete naučiti kako da koristite `DetailView` klasu za prikazivanje detalja objekta.

DetailView prikaz

`DetailView` vam omogućava da definišete prikaz zasnovan na klasi koji prikazuje detalje objekta. Da biste koristili `DetailView` klasu, definišete klasu koja nasleđuje `DetailView` klasu.

Na primer, sledeće definiše `TaskDetail` prikaz zasnovan na klasi koji prikazuje detalje `Todo` aplikacije:

```
from django.shortcuts import render
from django.views.generic.list import ListView
from django.views.generic.detail import DetailView
from .models import Task

class TaskDetail(DetailView):
  model = Task
  context_object_name = 'task'
#...
```

Kako ovo funkcioniše?

U `TaskDetail` klasi definišemo sledeće atribute:

- `model` određuje klasu objekta koji će biti prikazan.
- `context_object_name` određuje ime objekta u šablonu. Podrazumevano, Django koristi `object` kao ime objekta u šablonu. Da bismo to učinili očiglednijim, umesto toga koristimo `task` kao ime objekta.

Podrazumevano, `TaskDetail` klasa će učitati šablon sa imenom `task_detail.html` iz `templates/todo` aplikacije.

Ako želite da koristite drugačije ime šablona, možete koristiti `template_name` atribut u `TaskDetail` klasi.

DetailView šablon

Napravite `task_detail.html` šablon u `templates/todo` direktorijumu pomoću sledećeg koda:

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

{%block content%}

<article class="task">
  <header>
    <h2>{{ task.title }}</h2>
    <span class="badge {% if task.completed %}badge-completed{% else %}badge-
pending{%endif%}">
    {% if task.completed %} Completed {%else%} Pending {%endif%}
    </span>
  </header>
  <p>{{task.description}}</p>
</article>

{%endblock content%}
```

Šablon `task_detail.html` proširuje `base.html` šablon.

Šablon `task_detail.html` koristi `task` kao objekat i prikazuje atribute zadatka, uključujući naslov, status (završen ili ne) i opis.

DetailView ruta

Definišite rutu koja mapira URL adresu koja prikazuje zadatak sa rezultatom metode `as_view()` klase `TaskDetail`:

```
from django.urls import path
from .views import home, TaskList, TaskDetail

urlpatterns = [
  path('', home, name='home'),
  path('tasks/', TaskList.as_view(),name='tasks'),
  path('task/<int:pk>/',TaskDetail.as_view(),name='task'),
]
```

URL prihvata ceo broj kao ID (ili primarni ključ, pk) zadatka. `TaskDetail` će uzeti ovaj `pk` parametar, izabrati zadatak iz baze podataka prema ID-u, konstruisati objekat `Task` i proslediti ga šablonu.

DetailView šablon - izmene

Izmenite `task_list.html` šablon da biste uključili vezu do svakog zadatka na listi zadataka koristeći `url` oznaku:

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

{%block content%}

<div class="center">
  <h2>My Todo List</h2>
  {% if tasks %}
  <ul class="tasks">
    {% for task in tasks %}
      <li><a href="{% url 'task' task.id %}" class="{% if task.completed%}
completed{%endif%}">{{ task.title }}</a>
        <div  class="task-controls">
          <a href="#"><i class="bi bi-trash"></i> </a>
          <a href="#"><i class="bi bi-pencil-square"></i></a>
        </div>
      </li>
    {% endfor %}
  {% else %}
    <p>🎉 Yay, you have no pending tasks!</p>
  {% endif %}
  </ul>
</div>

{%endblock content%}
```

Kada kliknete na link svake oznake, bićete preusmereni na stranicu sa detaljima zadatka.

Pokrenite Django dev server

```
python manage.py runserver
```

i otvorite listu zadataka: http://127.0.0.1:8000/tasks/.

Ako kliknite na zadatak, npr. Learn Python, bićete preusmereni na stranicu sa detaljima zadatka "Learn Python".

Rezime DetailView

- Koristite `DetailView` za prikaz detalja objekta.

Sadržaj

CreateView

U ovom tutorijalu ćete naučiti kako da koristite `CreateView` klasu za definisanje prikaza zasnovanog na klasi koji kreira zadatak za aplikaciju `Todo`.

CreateView prikaz

Klasa `CreateView` vam omogućava da kreirate prikaz zasnovan na klasi koji prikazuje obrazac za kreiranje objekta, ponovno prikazivanje obrasca sa greškama u validaciji i čuvanje objekta u bazi podataka.

Da biste koristili `CreateView` klasu, definišete klasu koja nasleđuje od nje i dodate joj neke atribute i metode.

Na primer, sledeći primer koristi `CreateView` klasu za definisanje prikaza zasnovanog na klasi koji prikazuje obrazac za kreiranje novog zadatka u aplikaciji `Todo`:

```
# ..
from django.views.generic.edit import CreateView
from django.contrib import messages
from django.urls import reverse_lazy
from .models import Task

class TaskCreate(CreateView):
  model = Task
  fields = ['title','description','completed']
  success_url = reverse_lazy('tasks')

  def form_valid(self, form):
    form.instance.user = self.request.user
    messages.success(self.request, "The task was created successfully.")
    return super(TaskCreate,self).form_valid(form)

# other classes & functions
```

U `TaskCreate` klasi definišemo sledeće atribute i metode:

- `model` određuje klasu objekta koji treba kreirati (`Task`).
- `fields` je lista polja koja se prikazuju na obrascu. U ovom primeru, obrazac će prikazivati `title`,

`description` i `completed` atribute modela `Task`.

- `success_url` je ciljni URL na koji će Django preusmeriti nakon što se zadatak uspešno kreira. U ovom primeru, preusmeravamo na listu zadataka koristeći `reverse_lazy()` funkciju. `reverse_lazy()` prihvata ime prikaza i vraća URL.
- `form_valid()` je metoda koja se poziva nakon uspešnog slanja forme. U ovom primeru, postavljamo korisnika na trenutno prijavljenog korisnika, kreiramo fleš poruku i vraćamo rezultat `form_valid()` metode nadklase.

Podrazumevano, `CreateView` klasa koristi `task_form.html` šablon iz `templates/todo` sa sledećom konvencijom imenovanja:

```
modelname_form.html
```

Ako želite da koristite drugi šablon, možete da zamenite podrazumevani šablon koristeći `template_name` atribut u `TaskCreate` klasi.

CreateView šablon

Kreirajte `task_form.html` u `templates/todo` direktorijumu pomoću sledećeg koda:

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

{%block content%}

<div class="center">
  <form method="post" novalidate class="card">
     {%csrf_token %}

     <h2>Create Task</h2>
    {% for field in form %}
      {% if field.name == 'completed' %}
        <p>
          {{ field.label_tag }}
          {{ field }}
        </p>
        {% if field.errors %}
          <small class="error">{{ field.errors|striptags  }}</small>
        {% endif %}
      {% else %}
        {{ field.label_tag }}
        {{ field }}
        {% if field.errors %}
          <small class="error">{{ field.errors|striptags  }}</small>
        {% endif %}
      {% endif %}
    {% endfor %}

    <div class="form-buttons">
      <input type="submit" value="Save" class="btn btn-primary"/>
      <a href="{%url 'tasks'%}" class="btn btn-outline">Cancel</a>
    </div>
  </form>
</div>

{%endblock content%}
```

U `task_form.html`, polja obrasca prikazujemo ručno. Ako želite da automatski generišete obrazac, možete koristiti jedan od sledećih atributa:

```
{{ form.as_p }}   # render the form as <p>
{{ form.as_div }} # render the form as <div>
{{ form.as_ul }}  # redner the form as <ul>
```

CreateView ruta

Dodajte rutu u `urls.py` aplikacije `todo` mapiranjem URL-a sa rezultatom metode `as_view()` klase `TaskCreate`:

```
from django.urls import path
from .views import home, TaskList, TaskDetail, TaskCreate

urlpatterns = [
  path('', home, name='home'),
  path('tasks/', TaskList.as_view(),name='tasks'),
  path('task/<int:pk>/', TaskDetail.as_view(),name='task'),
  path('task/create/', TaskCreate.as_view(),name='task-create'),
]
```

Fleš poruka i dodavanje CreateView linka

Izmenite `base.html` šablon projekta na:

- Prikažite fleš poruke.
- Dodajte New Task vezu u navigaciju.

```
{%load static %}

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

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{% static 'css/style.css' %}" />
    <title>Todo List</title>
  </head>

  <body>
    <header class="header">
      <div class="container">
        <a href="{%url 'home'%}" class="logo">Todo</a>
        <nav class="nav">
          <a href="{% url 'home'%}"><i class="bi bi-house-fill"></i> Home</a>
          <a href="{% url 'tasks' %}"><i class="bi bi-list-task"></i> My Tasks</a>
          <a href="{% url 'task-create' %}"><i class="bi bi-plus-circle"></i>
Create Task</a>
        </nav>
      </div>
    </header>
    <main>
      <div class="container">
        {% if messages %}
          {% for message in messages %}
            <div class="alert alert-{{message.tags}}">
              {{message}}
            </div>
          {% endfor %}
        {% endif %}

        {%block content %}
        {%endblock content%}
      </div>
    </main>
    <footer class="footer">
      <div class="container">
        <p>© Copyright {% now "Y" %} by <a href="https://
www.pythontutorial.net">Python Tutorial</a></p>
      </div>
    </footer>
  </body>

</html>
```

Pokrenite Django dev server i otvorite URL adresu http://127.0.0.1:8000/task/create/, videćete obrazac za kreiranje novog `Todo` objekta.

Unesite naslov i opis i kliknite na dugme "Save", bićete preusmereni na stranicu sa listom zadataka sa

porukom.

Rezime CreateView

- Koristite klasu `CreateView` da definišete prikaz zasnovan na klasi koji kreira objekat.

UpdateView

U ovom tutorijalu ćete naučiti kako da koristite `UpdateView` klasu za kreiranje prikaza zasnovanog na klasi koji uređuje postojeći objekat.

UpdateView prikaz

Klasa `UpdateView` vam omogućava da kreirate prikaz zasnovan na klasi koji:

- Prikazuje formu za uređivanje postojećeg objekta.
- Ponovo prikazuje formu ako sadrži greške u validaciji.
- Čuva izmene objekta u bazi podataka.

Forma se generiše automatski iz modela objekta, osim ako eksplicitno ne navedete klasu forme.

Da bismo demonstrirali `UpdateView` klasu, kreiraćemo prikaz koji uređuje zadatak `Todo` aplikacije.

Da bismo to uradili, modifikujemo `views.py` aplikacije `todo` i definišemo `TaskUpdate` klasu koja nasleđuje tu `UpdateView` klasu ovako:

```
# ...
from django.views.generic.edit import CreateView, UpdateView
from django.contrib import messages
from django.urls import reverse_lazy
from .models import Task

class TaskUpdate(UpdateView):
  model = Task
  fields = ['title','description','completed']
  success_url = reverse_lazy('tasks')

  def form_valid(self, form):
    messages.success(self.request, "The task was updated successfully.")
    return super(TaskUpdate,self).form_valid(form)
# ...
```

U `TaskUpdate` klasi definišite sledeće atribute i metode:

- `model` određuje klasu objekta koji treba uređivati. Zato u ovom primeru određujemo Task kao model.
- `fields` je lista koja određuje polja forme. U ovom primeru koristimo polja za naslov, opis i popunjena polja.
- `success_url` je ciljni URL (lista zadataka) na koji će Django preusmeriti nakon što se zadatak uspešno ažurira.
- `form_valid()` metod se poziva nakon uspešnog slanja forme. U ovom primeru, kreiramo fleš poruku i vraćamo rezultat metode `form_valid()` nadklase.

Podrazumevano, `TaskUpdate` klasa koristi `task_form.html` šablon iz `templates/todo` direktorijuma. Imajte

na umu da klase `CreateView` i `UpdateView` dele isto ime šablona.

Ako želite da koristite drugačije ime šablona, možete ga navesti pomoću `template_name` atributa.

UpdateList šablon

Izmenite `task_form.html` šablon koji prikazuje `UpdateTask` naslov ako je promenljiva zadatka dostupna u šablonu (režim uređivanja) ili `CreateTask` ako nije (režim kreiranja).

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

{%block content%}

<div class="center">
<form method="post" novalidate class="card">
  {%csrf_token %}
  <h2>{% if task %} Update {%else %} Create {%endif%} Task</h2>
  {% for field in form %}
    {% if field.name == 'completed' %}
      <p>
        {{ field.label_tag }}
        {{ field }}
      </p>
      {% if field.errors %}
        <small class="error">{{ field.errors|striptags  }}</small>
      {% endif %}
    {% else %}
      {{ field.label_tag }}
      {{ field }}
      {% if field.errors %}
        <small class="error">{{ field.errors|striptags  }}</small>
      {% endif %}
    {% endif %}
  {% endfor %}

  <div class="form-buttons">
    <input type="submit" value="Save" class="btn btn-primary"/>
    <a href="{%url 'tasks'%}" class="btn btn-outline">Cancel</a>
  </div>
</form>
</div>

{%endblock content%}
```

UpdateList ruta

Definišite rutu u `urls.py` aplikaciji `todo` koja mapira URL adresu sa rezultatom metode `as_view()` klase `TaskUpdate`:

```
from django.urls import path
from .views import (
  home,
  TaskList,
  TaskDetail,
  TaskCreate,
  TaskUpdate
)

urlpatterns = [
  path('', home, name='home'),
  path('tasks/', TaskList.as_view(),name='tasks'),
  path('task/<int:pk>/', TaskDetail.as_view(),name='task'),
  path('task/create/', TaskCreate.as_view(),name='task-create'),
  path('task/update/<int:pk>/', TaskUpdate.as_view(),name='task-update'),
]
```

UpdateList link

Izmenite `task_list.html` šablon da biste uključili link za uređivanje za svaki zadatak na listi zadataka:

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

{%block content%}

<div class="center">
  <h2>My Todo List</h2>
  {% if tasks %}
  <ul class="tasks">
    {% for task in tasks %}
      <li><a href="{% url 'task' task.id %}" class="{% if task.completed%}
completed{%endif%}">{{ task.title }}</a>
        <div  class="task-controls">
          <a href="#"><i class="bi bi-trash"></i> </a>
          <a href="{%url 'task-update' task.id %}"><i class="bi bi-pencil-
square"></i></a>
        </div>
      </li>
    {% endfor %}
  {% else %}
    <p>🎉 Yay, you have no pending tasks! <a href="{%url 'task-create'%}">Create
Task</a></p>
  {% endif %}
  </ul>
</div>

{%endblock content%}
```

Ako uredite zadatak sa liste obaveza dodavanjem tri zvezdice (***) naslovu i označite zadatak kao završen.

Kliknite na dugme "Save" i videćete da su naslov i status zadatka ažurirani:

Rezime UpdateView

- Definišite novu klasu koja nasleđuje `UpdateView` klasu da biste kreirali prikaz zasnovan na klasi koji uređuje postojeći objekat.

Sadržaj

DeleteView

U ovom tutorijalu ćete naučiti kako da koristite `DeleteView` klasu za definisanje prikaza zasnovanog na klasi koji briše postojeći objekat.

Izgradnja `DeleteView` klasu

Klasa `DeleteView` vam omogućava da definišete prikaz zasnovan na klasi koji prikazuje stranicu za potvrdu i briše postojeći objekat.

Ako je metod HTTP zahteva `GET`, `DeleteView` će prikazati stranicu za potvrdu. Međutim, ako je zahtev `POST`, `DeleteView` prikaz će obrisati objekat.

Da biste koristili `DeleteView` klasu, definišete klasu koja nasleđuje od nje i dodajete atribute i metode da biste poništili podrazumevana ponašanja.

Na primer, sledeće definiše `TaskDelete` klasu koja briše zadatak za aplikaciju `Todo`:

```
#...
from django.views.generic.edit import DeleteView, CreateView, UpdateView
from django.contrib import messages
from django.urls import reverse_lazy

from .models import Task

class TaskDelete(DeleteView):
  model = Task
  context_object_name = 'task'
  success_url = reverse_lazy('tasks')

  def form_valid(self, form):
    messages.success(self.request, "The task was deleted successfully.")
    return super(TaskDelete,self).form_valid(form)

#...
```

`TaskDelete` klasa ima sledeće atribute:

- `model` određuje klasu modela (`Task`) koja će biti obrisana.
- `context_object_name` određuje ime objekta koje će biti prosleđeno šablonu. Podrazumevano, `DeleteView` klasa koristi object kao ime. Međutim, možete zameniti ime pomoću `context_object_name` atributa.
- `success_url` je URL adresa na koju će biti preusmereno nakon što se objekat uspešno obriše.
- `form_valid()` metoda se poziva kada se objekat uspešno obriše. U ovom primeru kreiramo fleš poruku.

Podrazumevano, `DeleteView` klasa koristi `task_confirmation_delete.html` šablon ako ga eksplicitno ne

navedete.

DeleteView šablon

Napravite novi `task_confirm_delete.html` šablon datoteke u `templates/todo` aplikaciji pomoću sledećeg koda:

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

{%block content%}
<div class="center">
  <form method="post" class="card">
    {% csrf_token %}
    <h2>Delete Task</h2>
    <p>Are you sure that you want to delete "{{task}}"?</p>
    <p class="form-buttons">
      <input type="submit" class="btn btn-primary" value="Delete">
      <a href="{% url 'tasks'%}" class="btn btn-outline">Cancel</a>
    </p>
  </form>
</div>

{%endblock content%}
```

Ovaj kod proširuje `task_confirm_delete.html` šablon `base.html` i sadrži obrazac koji briše zadatak.

DeleteView ruta

Definišite novu rutu u `urls.py` koja mapira URL adresu koja briše zadatak sa rezultatom metode `as_view()` klase `TaskDelete` prikaza:

```
from django.urls import path
from .views import (
  home,
  TaskList,
  TaskDetail,
  TaskCreate,
  TaskUpdate,
  TaskDelete
)

urlpatterns = [
  path('', home, name='home'),
  path('tasks/', TaskList.as_view(),name='tasks'),
  path('task/<int:pk>/', TaskDetail.as_view(),name='task'),
  path('task/create/', TaskCreate.as_view(),name='task-create'),
  path('task/update/<int:pk>/', TaskUpdate.as_view(),name='task-update'),
  path('task/delete/<int:pk>/', TaskDelete.as_view(),name='task-delete'),
]
```

DeleteView link

Izmenite `task_list.html` šablon da biste dodali vezu koja briše zadatak svakom zadatku na listi zadataka:

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

{%block content%}

<div class="center">
  <h2>My Todo List</h2>
  {% if tasks %}
  <ul class="tasks">
    {% for task in tasks %}
      <li><a href="{% url 'task' task.id %}" class="{% if task.completed%}
completed{%endif%}">{{ task.title }}</a>
        <div  class="task-controls">
          <a href="{%url 'task-delete' task.id %}"><i class="bi bi-trash"></i> </a>
          <a href="{%url 'task-update' task.id %}"><i class="bi bi-pencil-
square"></i></a>
        </div>
      </li>
    {% endfor %}
  {% else %}
    <p>🎉 Yay, you have no pending tasks! <a href="{%url 'task-create'%}">Create
Task</a></p>
  {% endif %}
  </ul>
</div>

{%endblock content%}
```

Ako kliknete na dugme za brisanje da biste obrisali zadatak sa liste, dobićete stranicu za potvrdu brisanja.

Klikom na dugme "Delete" obrisaćete zadatak iz baze podataka i vratiti ga na listu zadataka.

Rezime DeleteView

Koristite `DeleteView` klasu da definišete prikaz zasnovan na klasi koji briše postojeći objekat.

Sadržaj

LoginView

U ovom tutorijalu ćete naučiti kako da koristite `LoginView` za kreiranje stranice za prijavu za `Todo` aplikaciju.

`LoginView` vam omogućava da prikažete formu za prijavu i obradite akciju prijave. Koristićemo LoginView klasu da kreiramo stranicu za prijavu za `Todo` aplikaciju.

Users aplikacije

Aplikacija `users` će imati sledeće funkcionalnosti:

- Prijava / Odjava korisnika
- Registrovanje korisnika
- Resetovanje lozinke korisnika

U ovom tutorijalu ćemo se fokusirati na funkcije `Prijava / Odjava`.

Koristite `startapp` komandu za kreiranje `users` aplikacije:

```
django-admin startapp users
```

Zatim, registrujte `users` aplikaciju u `settings.py` projektu:

```
INSTALLED_APPS = [
  #...
  'users'
]
```

Zatim, kreirajte `urls.py` u `users` aplikaciji sa sledećim kodom:

```
from django.urls import path

urlpatterns = []
```

Nakon toga, uključite `urls.py` aplikaciju `user` su `urls.py` projekat:

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

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

Konačno, kreirajte `templates` direktorijum i `users` direktorijum unutar `templates` direktorijuma u `users` aplikaciji za čuvanje šablona.

LoginView strana

Sledeći kod definiše klasu koja nasleđuje od klase `LoginView` u datoteci `views.py`.

```
from django.contrib.auth.views import LoginView
from django.urls import reverse_lazy
from django.contrib import messages

class MyLoginView(LoginView):
  redirect_authenticated_user = True

  def get_success_url(self):
    return reverse_lazy('tasks')

  def form_invalid(self, form):
    messages.error(self.request,'Invalid username or password')
    return self.render_to_response(self.get_context_data(form=form))
```

Klasa `MyLoginView` ima sledeće atribute i metode:

- `redirect_authenticated_user` je podešeno na `True` da bi naložilo Django-u da preusmeri korisnike nakon što se uspešno prijave. Podrazumevano, `redirect_authenticated_user` je `False`, što isključuje preusmeravanje.
- `get_success_url()` vraća URL adresu za preusmeravanje nakon što se korisnici uspešno prijave.
- `form_invalid()` se poziva kada prijava ne uspe. U `form_invalid()`, kreiramo fleš poruku i ponovo prikazujemo obrazac za prijavu.

LoginView ruta

Izmenite `views.py` datoteku da biste definisali rutu za `LoginView` stranicu:

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

urlpatterns = [
  path('login/', MyLoginView.as_view(),name='login'),
]
```

LoginView šablon

Kreirajte `login.html` šablon u `templates/users` direktorijumu pomoću sledećeg koda:

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

{%block content%}

<div class="center">
  <form method="post" class="card" novalidate>
    {% csrf_token %}
    <h2 class="text-center">Log in to your account</h2>
    {% for field in form %}
      {{ field.label_tag }}
      {{ field }}
      {% if field.errors %}
        <small>{{ field.errors|striptags }}</small>
      {% endif %}
    {% endfor %}

    <input type="submit" value="Login" class="btn btn-primary full-width">
    <hr>
    <p class="text-center">Forgot your password <a href="#">Reset Password</a></p>
    <p class="text-center">Don't have a account? <a href="#">Join Now</a></p>
  </form>
</div>

{%endblock content%}
```

Ako otvorite URL adresu za prijavu http://127.0.0.1:8000/login/ videćete obrazac za prijavu.

Ako unesete važeće korisničko ime i lozinku, uspešno ćete se prijaviti. U suprotnom, dobićete poruku da ste uneli nevažeće korisničko ime ili lozinku.

Dodajte LOGIN_URL u `settings.py` projekat:

```
LOGIN_URL = 'login'
```

Ako pokušate da pristupite stranici koja zahteva prijavu, Django će koristiti `LOGIN_URL` za preusmeravanje. Ako ne dodate `LOGIN_URL` u `settings.py`, Django će koristiti podrazumevani URL za prijavu, koji je `accounts/login/`.

Logout url

`LogoutView` odjavljuje korisnika i prikazuje poruku. Koristićemo `LogoutView` da kreiramo link za odjavu.

Za razliku od `LoginView` klase, možete koristiti `LogoutView` klasu direktno u `urls.py`. Na primer, možete izmeniti `views.py` da biste kreirali rutu za URL adresu za odjavu:

```
from django.urls import path
from .views import MyLoginView
from django.contrib.auth.views import LogoutView

urlpatterns = [
  path('login/', MyLoginView.as_view(),name='login'),
  path('logout/', LogoutView.as_view(next_page='login'),name='logout'),
]
```

Login / Logout linkovi

Ako se korisnik prijavi, zaglavlje prikazuje početnu stranicu, moje zadatke, novi zadatak i vezu za odjavu. Kada se korisnik odjavi, zaglavlje prikazuje veze početna stranica, prijava i pridruži se sada.

Da biste to postigli, izmenite base.htmlšablon projekta na sledeći način.

```
{%load static %}
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{% static 'css/style.css' %}" />
    <title>Todo List</title>
  </head>
  <body>
    <header class="header">
      <div class="container">
        <a href="{%url 'home'%}" class="logo">Todo</a>
        <nav class="nav">
          <a href="{%url 'home'%}"><i class="bi bi-house-fill"></i> Home</a>
          {% if request.user.is_authenticated %}
              <a href="{% url 'tasks' %}"><i class="bi bi-list-task"></i> My
Tasks</a>
              <a href="{% url 'task-create' %}"><i class="bi bi-plus-circle"></i>
Create Task</a>
              <a href="#">Hi {{request.user | title}}</a>
              <a href="{% url 'logout' %}" class="btn btn-outline">Logout</a>
          {% else %}
            <a href="{% url 'login' %}" class="btn btn-outline">Login</a>
            <a href="#" class="btn btn-primary">Join Now</a>
          {% endif %}
          </nav>
      </div>
    </header>
    <main>
      <div class="container">
        {% if messages %}
        {% for message in messages %}
          <div class="alert alert-{{message.tags}}">
               {{message}}
             </div>
             {% endfor %}
        {% endif %}

       {%block content %}
       {%endblock content%}
      </div>
    </main>
    <footer class="footer">
      <div class="container">
        <p>© Copyright {% now "Y" %} by <a href="https://
www.pythontutorial.net">Python Tutorial</a></p>
      </div>
    </footer>
  </body>
</html>
```

Imajte na umu da ako se korisnik prijavi, `request.user.is_authenticated` vraća `True`. Stoga, možete koristiti ovo svojstvo da proverite da li je korisnik prijavljen ili ne.

Iako niste prijavljeni, i dalje možete upravljati listom zadataka, kao što je pregled, dodavanje, uređivanje i brisanje zadataka. Da biste zaštitili ove stranice, koristićete `LoginRequiredMixin` klasu.

Da biste to uradili, modifikujete `views.py` aplikaciju `todo` i koristite `LoginRequiredMixin` klasu na sledeći način:

```python
from django.shortcuts import render
from django.views.generic.list import ListView
from django.views.generic.detail import DetailView
from django.views.generic.edit import CreateView, UpdateView, DeleteView
from django.contrib import messages
from django.urls import reverse_lazy
from django.contrib.auth.mixins import LoginRequiredMixin

from .models import Task

class TaskDelete(LoginRequiredMixin, DeleteView):
  model = Task
  context_object_name = 'task'
  success_url = reverse_lazy('tasks')

  def form_valid(self, form):
    messages.success(self.request, "The task was deleted successfully.")
    return super(TaskDelete,self).form_valid(form)

class TaskUpdate(LoginRequiredMixin, UpdateView):
  model = Task
  fields = ['title','description','completed']
  success_url = reverse_lazy('tasks')

  def form_valid(self, form):
    messages.success(self.request, "The task was updated successfully.")
    return super(TaskUpdate,self).form_valid(form)

class TaskCreate(LoginRequiredMixin, CreateView):
  model = Task
  fields = ['title','description','completed']
  success_url = reverse_lazy('tasks')

  def form_valid(self, form):
    form.instance.user = self.request.user
    messages.success(self.request, "The task was created successfully.")
    return super(TaskCreate,self).form_valid(form)

class TaskDetail(LoginRequiredMixin, DetailView):
  model = Task
  context_object_name = 'task'

class TaskList(LoginRequiredMixin,ListView):
  model = Task
  context_object_name = 'tasks'

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

Ako se niste prijavili i pokušali ste da pristupite zaštićenoj stranici, Django će vas preusmeriti na stranicu za prijavu. Na primer: http://127.0.0.1:8000/task/create Django će vas preusmeriti na stranicu za prijavu koristeći LOGIN_URL konfigurisano u `settings.py`: http://127.0.0.1:8000/login/?next=/task/create/.

Rezime LoginView

- Koristite `LoginView` klasu da biste kreirali stranicu za prijavu.
- Koristite `LogoutView` klasu da odjavite korisnika.
- Koristite `LoginRequiredMixin` klasu da zaštitite stranicu.

Sadržaj

FormView

U ovom tutorijalu ćete naučiti kako da koristite `FormView` klasu za kreiranje registracionog formulara za aplikaciju `Todo`.

Kreiranje forme za prijavu

Napravite `forms.py` datoteku u `users` aplikaciji i definišite `RegisterForm` klasu na sledeći način:

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

class RegisterForm(UserCreationForm):
  email = forms.EmailField(max_length=254)

  class Meta:
    model = User
    fields = ('username',  'email', 'password1', 'password2', )
```

RegisterForm koristi model `User` i prikazuje polja korisničko `username`, `email`, `password1` i `password2`.

FormView klasa

Definišite `RegisterView` klasu u `views.py` aplikacije `users.py`:

```
from django.urls import reverse_lazy
from django.views.generic.edit import FormView
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import login
from django.contrib.auth.models import User
from .forms import RegisterForm

class RegisterView(FormView):
  template_name = 'users/register.html'
  form_class = RegisterForm
  redirect_authenticated_user = True
  success_url = reverse_lazy('tasks')

  def form_valid(self, form):
    user = form.save()
    if user:
      login(self.request, user)

    return super(RegisterView, self).form_valid(form)
```

Klasa `RegisterView` nasleđuje klasu `FormView` i ima sledeće atribute i metode:

- `template_name` određuje ime šablona za prikazivanje forme za prijavu.
- `form_class` određuje formu (RegisterForm) koji se koristi u šablonu.
- `redirect_authenticated_user` je postavljeno na `True` da bi se korisnik preusmerio nakon autentifikacije.
- `success_url` određuje URL adresu za preusmeravanje nakon što se korisnik uspešno registruje. U ovom primeru, preusmerava korisnika na Task listu.
- `form_valid()` metoda se poziva nakon uspešnog slanja formulara. U ovom primeru, čuvamo `User` model i automatski prijavljujemo korisnika.

RegisterView ruta

Definišite rutu koja mapira URL adresu za registraciju `register/` sa rezultatom metode `as_view()` klase `RegisterView` u `urls.py` aplikacije `users`:

```
from django.urls import path
from django.contrib.auth.views import LogoutView
from .views import MyLoginView, RegisterView

urlpatterns = [
  path('login/', MyLoginView.as_view(),name='login'),
  path('logout/', LogoutView.as_view(next_page='login'),name='logout'),
  path('register/', RegisterView.as_view(),name='register'),
]
```

RegisterView šablon

Napravite datoteku `register.html` u `templates/users` direktorijumu aplikacije `users` sa sledećim kodom:

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

{%block content%}
  <div class="center">
    <form method="post" novaldiate class="card">
      {% csrf_token %}
    <h2 class="text-center">Create your account</h2>
    {% for field in form %}
        {{ field.label_tag }}
        {{ field }}
        {% if field.errors %}
          <small class="error">{{ field.errors|striptags  }}</small>
        {% endif %}
    {% endfor %}

    <input type="submit" value="Register" class="btn btn-primary full-width">
    <hr>
    <p class="text-center">Already have an account? <a href="{% url
'login'%}">Login Here</a></p>
    </form>
  </div>

{%endblock content%}
```

Forma `users/register.html` je za `RegisterForm` klasu koju smo definisali u `forms.py` datoteci.

RegisterForm link

Izmenite `login.html` šablon dodavanjem linka za registraciju:

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

{%block content%}
  <div class="center">
    <form method="post" class="card" novalidate>
      {% csrf_token %}
    <h2 class="text-center">Log in to your account</h2>
    {% for field in form %}
        {{ field.label_tag }}
        {{ field }}
        {% if field.errors %}
          <small>{{ field.errors|striptags  }}</small>
        {% endif %}
    {% endfor %}

1    <input type="submit" value="Login" class="btn btn-primary full-width">
    <hr>
    <p class="text-center">Forgot your password <a href="#">Reset Password</a></p>
    <p class="text-center">Don't have a account? <a href="{%url 'register'%}">Join
Now</a></p>
  </form>
</div>

{%endblock content%}
```

Takođe, izmenite `base.html` šablon dodavanjem linka za registraciju u navigaciju i na početnu stranicu:

```
{%load static %}
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{% static 'css/style.css' %}" />
    <title>Todo List</title>
  </head>

  <body>
    <header class="header">
      <div class="container">
        <a href="{%url 'home'%}" class="logo">Todo</a>
        <nav class="nav">
          <a href="{%url 'home'%}"><i class="bi bi-house-fill"></i> Home</a>
          {% if request.user.is_authenticated %}
                <a href="{% url 'tasks' %}"><i class="bi bi-list-task"></i> My
Tasks</a>
                <a href="{% url 'task-create' %}"><i class="bi bi-plus-circle"></i>
Create Task</a>
             <a href="#">Hi {{request.user | title}}</a>
               <a href="{% url 'logout' %}" class="btn btn-outline">Logout</a>
          {% else %}
        <a href="{% url 'login' %}" class="btn btn-outline">Login</a>
              <a href="{% url 'register' %}" class="btn btn-primary">Join Now</a>
          {% endif %}
         </nav>
      </div>
    </header>
    <main>
      <div class="container">
        {% if messages %}
      {% for message in messages %}
      <div class="alert alert-{{message.tags}}">
           {{message}}
        </div>
        {% endfor %}
    {% endif %}

       {%block content %}
       {%endblock content%}
      </div>
    </main>
    <footer class="footer">
      <div class="container">
        <p>© Copyright {% now "Y" %} by <a href="https://
www.pythontutorial.net">Python Tutorial</a></p>
      </div>
    </footer>
  </body>
```

```
</html>
```

`home.html`

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

{%load static %}

{%block content%}
  <section class="feature">
    <div class="feature-content">
      <h1>Todo</h1>
      <p>Todo helps you more focus, either work or play.</p>
      <a class="btn btn-primary cta" href="{% url 'register' %}">Get Started</a>
    </div>
    <img src="{%static 'images/feature.jpg'%}" alt="" class="feature-image">
  </section>
{%endblock content%}
```

Ako otvorite URL adresu za registraciju: http://127.0.0.1:8000/register/ videćete formular za registraciju.

Nakon uspešne registracije, bićete automatski prijavljeni.

Međutim, imamo jedan problem. Jane može da pregleda, ažurira i briše zadatke drugih korisnika.

Da biste ovo popravili, potrebno je da filtrirate zadatke koji pripadaju trenutno prijavljenom korisniku u svim klasama dodavanjem `get_queryset()` metoda u klase `TaskList`, `TaskDetail`, `TaskCreate`, `TaskUpdate`, `TaskDelete`.

```python
from django.shortcuts import render
from django.views.generic.list import ListView
from django.views.generic.detail import DetailView
from django.views.generic.edit import CreateView, UpdateView, DeleteView
from django.urls import reverse_lazy
from django.contrib.auth.mixins import LoginRequiredMixin
from django.contrib import messages
from .models import Task

class TaskList(LoginRequiredMixin, ListView):
  model = Task
  context_object_name = 'tasks'

  def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context['tasks'] = context['tasks'].filter(user=self.request.user)
    return context

class TaskDetail(LoginRequiredMixin, DetailView):
  model = Task
  context_object_name = 'task'

  def get_queryset(self):
    base_qs = super(TaskDetail, self).get_queryset()
    return base_qs.filter(user=self.request.user)

class TaskUpdate(LoginRequiredMixin, UpdateView):
  model = Task
  fields = ['title','description','completed']
  success_url = reverse_lazy('tasks')

  def form_valid(self, form):
    messages.success(self.request, "The task was updated successfully.")
    return super(TaskUpdate,self).form_valid(form)

  def get_queryset(self):
    base_qs = super(TaskUpdate, self).get_queryset()
    return base_qs.filter(user=self.request.user)

class TaskDelete(LoginRequiredMixin, DeleteView):
  model = Task
  context_object_name = 'task'
  success_url = reverse_lazy('tasks')

  def form_valid(self, form):
    messages.success(self.request, "The task was deleted successfully.")
    return super(TaskDelete,self).form_valid(form)

  def get_queryset(self):
    base_qs = super(TaskDelete, self).get_queryset()
    return base_qs.filter(user=self.request.user)
```

Sada, ako se prijavite kao Jane, videćete praznu listu obaveza.

Ako kreirate novi zadatak, videćete samo taj zadatak na listi obaveza.

Ako pokušate da pristupite Joe zadacima dok ste prijavljeni kao Jane, dobićete grešku 404.

Rezime RegisterForm

- Koristite `FormView` da biste kreirali prikaz koji prikazuje formu.

Sadržaj

Resetovanje lozinke

U ovom tutorijalu ćete naučiti kako da implementirate funkciju resetovanja lozinke koja omogućava korisnicima da resetuju svoje lozinke koristeći imejl adrese.

- Ako korisnik klikne na vezu `Password reset` na obrascu za prijavu, Django prikazuje obrazac koji omogućava korisniku da unese adresu e-pošte za prijem linka za resetovanje lozinke.
- Django koristi `PasswordResetView` prikaz da bi prikazao ovaj obrazac.
- Korisnik unosi adresu e-pošte i klikne na dugme `Submit`.
- Django šalje imejl na unetu imejl adresu i prikazuje poruku koja upućuje korisnika da proveri prijemno sanduče.
- Django koristi `PasswordResetDoneView` klasu za renderovanje ovog obrasca.
- Korisnik otvara prijemno sanduče i klikne na vezu za resetovanje lozinke.
- Na kraju, korisnik unosi novu lozinku i klikne na dugme `Password reset`.
- Django prikazuje poruku potvrde.
- Django koristi `PasswordResetCompleteView` za rukovanje ovom stranicom.

Implementacija resetovanja korisničke lozinke za aplikaciju Todo

Izmenite `views.py` aplikacije `users` da biste mapirali URL za resetovanje lozinke sa odgovarajućim prikazima zasnovanim na klasi.

```
from django.urls import path
from .views import MyLoginView, RegisterView

from django.contrib.auth.views import (
  LogoutView,
  PasswordResetView,
  PasswordResetDoneView,
  PasswordResetConfirmView,
  PasswordResetCompleteView
)

urlpatterns = [
  path('login/',
MyLoginView.as_view(redirect_authenticated_user=True),name='login'),
  path('logout/', LogoutView.as_view(next_page='login'),name='logout'),
  path('register/', RegisterView.as_view(),name='register'),
  path('password-reset/', PasswordResetView.as_view(template_name='users/
password_reset.html'),name='password-reset'),
  path('password-reset/done/', PasswordResetDoneView.as_view(template_name='users/
password_reset_done.html'),name='password_reset_done'),
  path('password-reset-confirm/<uidb64>/<token>/',
PasswordResetConfirmView.as_view(template_name='users/
password_reset_confirm.html'),name='password_reset_confirm'),
  path('password-reset-
complete/',PasswordResetCompleteView.as_view(template_name='users/
password_reset_complete.html'),name='password_reset_complete'),
]
```

Resetovanje korisničke lozinke

Mapirajte URL adresu za resetovanje lozinke `password-reset/` sa rezultatom metode `as_view()` klase `PasswordResetView`. `PasswordResetView` prikaz zasnovan na klasi koristi `users/password_reset.html` šablon za prikazivanje forme.

Kreirajte `password_reset.html` šablon u `templates/users` direktorijumu:

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

{%block content%}

<div class="center">
  <form method="post" class="card">
    {% csrf_token %}
    <h2 class="text-center">Reset Password</h2>
    {% for field in form %}
      {{ field.label_tag }}
      {{ field }}
      {% if field.errors %}
        <small>{{ field.errors|striptags  }}</small>
      {% endif %}
    {% endfor %}
    <div class="form-buttons">
      <input type="submit" value="Send" class="btn btn-primary">
      <a href="{%url 'login' %}"  class="btn btn-outline">Cancel</a>
    </div>
  </form>
</div>

{%endblock content%}
```

Resetovanje korisničke lozinke završeno

Mapirajte `password-reset/done/` sa rezultatom metode `as_view()` klase `PasswordResetDoneView`. `PasswordResetDoneView` klasa koristi `password_reset_done.html` šablon za prikazivanje stranice:

```
path('password-reset/done/', PasswordResetDoneView.as_view(template_name='users/
password_reset_done.html'),name='password_reset_done'),
```

Kreirajte `reset_password_done.html` šablon u `templates/users` direktorijumu:

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

{%block content%}

<div class="center card">
  <h2>Reset Password</h2>
  <p>Please check your inbox and follow the instruction to reset your password.</p>
</div>

{%endblock content%}
```

PasswordResetConfirmView

Mapirajte `password-reset-confirm/<uidb64>/<token>/` URL sa rezultatom `as_view()` klase `PasswordResetConfirmView`.

```
path('password-reset-confirm/<uidb64>/<token>/',
PasswordResetConfirmView.as_view(template_name='users/
password_reset_confirm.html'),name='password_reset_confirm'),
```

Klasa `PasswordResetConfirmView` koristi `password_reset_confirm.html` šablon za prikazivanje stranice. URL adresa za resetovanje lozinke izgleda ovako:

```
http://127.0.0.1:8000/password-reset-confirm/OA/bfwk0g-
d87966e0a694f519bc6f29daa4616b07/
```

Kreirajte `password_reset_confirm.html` šablon u `templates/users` direktorijumu:

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

{%block content%}

<div class="center">
  <form method="post" class="card">
    {% csrf_token %}
    <h2>Password Reset Confirm</h2>

    {% for field in form %}
      {{ field.label_tag }}
      {{ field }}
      {% if field.errors %}
        <small>{{ field.errors|striptags  }}</small>
      {% endif %}
    {% endfor %}

    <div>
      <button type="submit" class="btn btn-primary">Reset Password</button>
    </div>
  </form>
</div>

{%endblock content%}
```

PasswordResetCompleteView

Mapirajte kompletnu URL adresu za resetovanje lozinke sa `as_view()` metodom klase `PasswordResetCompleteView`.

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

{%block content%}

<div class="card center">
  <p>Your password has been changed successfully. Please <a href="{% url 'login'
%}">Login</a></p>
</div>

{%endblock content%}
```

Kreirajte `password_reset_complete.html` šablon u `templates/users` direktorijumu

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

{%block content%}
<div class="center">
  <form method="post" class="card" novalidate>
    {% csrf_token %}
    <h2 class="text-center">Log in to your account</h2>
    {% for field in form %}
      {{ field.label_tag }}
      {{ field }}
      {% if field.errors %}
        <small>{{ field.errors|striptags  }}</small>
      {% endif %}
    {% endfor %}

    <input type="submit" value="Login" class="btn btn-primary full-width">
    <hr>
    <p class="text-center">Forgot your password <a href="{%url 'password-
reset'%}">Reset Password</a></p>
    <p class="text-center">Don't have a account? <a href="{%url 'register'%}">Join
Now</a></p>
  </form>
</div>

{%endblock content%}
```

Konfigurisanje SMTP-a za slanje imejlova iz Djanga

Da biste slali imejlove u Djangu, potreban vam je lokalni Simple Mail Transfer Protocol (SMTP) server ili eksterni SMTP server od dobavljača usluga imejla.

Kada imate SMTPserver, možete dodati njegove informacije u `settings.py` Django projekt<> sa sledećim informacijama:

```
EMAIL_HOST: SMTPserver domaćin.
EMAIL_PORT: SMTP port, podrazumevana vrednost je 25.
EMAIL_HOST_USER: Korisničko ime za SMTPserver.
EMAIL_HOST_PASSWORD: lozinka za SMTPserver.
EMAIL_USE_TLS: da li se koristi bezbedna veza ( TLS ).
```

Na primer, sledeće pokazuje kako se koriste SMTP podešavanja Google servera:

```
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_PORT = 25
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'your_gmail_account@gmail.com'
EMAIL_HOST_PASSWORD = 'your_gmail_password'
```

Imajte na umu da treba da zamenite `EMAIL_HOST_USER` i `EMAIL_HOST_PASSWORD` svojim podacima za Gmail.

Ako nemate lokalni SMTPserver, možete koristiti sledeću vrednost u `settings.py` datoteci:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

Korišćenjem ovog podešavanja, Django će ispisati sve imejlove u konzolu (Shell) umesto da ih šalje. Ovo je veoma pogodno za testiranje bez SMTPservera.

Primer izlaza će izgledati ovako:

```
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
Subject: Password reset on 127.0.0.1:8000
From: webmaster@localhost
To: jane@pythontutorial.net
Date: <date>
Message-ID:
 <167014230656.23040.17412319412157509424@pythontutorial.net>

You're receiving this email because you requested a password reset for your user
account at 127.0.0.1:8000.

Please go to the following page and choose a new password:

http://127.0.0.1:8000/password-reset-confirm/OA/bfwytu-
b9f9b789e9a294eb80d707e448dde1d2/

Your username, in case you've forgotten: jane
Thanks for using our site!
The 127.0.0.1:8000 team
```

Ako želite da prilagodite imejl za resetovanje lozinke, možete da kreirate `password_reset_email.html` u `templates/users` direktorijumu:

```
<p>Hi</p>

<p>You're receiving this email because you requested a password reset for your user
account at {{domain}}/</p>

<p>Please click the following link to reset your password:</p>

{{ protocol }}://{{ domain }}{% url "password_reset_confirm" uidb64=uid token=token
%}

<p>Thanks</p>
<p>Todo App Team</p>
```

I navedite `html_email_template_name` u `as_view()` metodi klase `PasswordResetView`:

```
path('password-reset/',
   PasswordResetView.as_view(
    template_name='users/password_reset.html',
    html_email_template_name='users/password_reset_email.html'
  ),
  name='password-reset'
)
```

Ako resetujete lozinku, Django će koristiti prilagođeni šablon e-pošte.

```
<p>Hi</p>

<p>You're receiving this email because you requested a password reset for your user
account at 127.0.0.1:8000/</p>

<p>Please click the following link to reset your password:</p>

http://127.0.0.1:8000/password-reset-confirm/OA/
bfwzqv-9d6b6777ad40073cfa1d4d4e150fb76f/

<p>Thanks</p>
<p>Todo App Team</p>
```

Rezime resetovanja korisničke lozinke

- Koristite klase `PasswordResetView`, `PasswordResetDoneView`, `PasswordResetConfirmView` i `PasswordResetCompleteView` da biste implementirali funkciju resetovanja lozinke za Django aplikaciju.

Sadržaj

Korisnički profil

U ovom tutorijalu ćete naučiti kako da implementirate korisnički profil u Django aplikacijama.

Korisnički profil se sastoji od podešavanja i informacija povezanih sa korisnikom. U ovom tutorijalu ćete naučiti kako da dozvolite korisnicima da ažuriraju svoje profile u Django aplikacijama.

Instaliranje paketa pillow

Pošto ćemo se baviti slikama, potrebno je da instaliramo `pillow` paket pomoću sledeće `pip` komande:

```
pip install Pillow
```

Konfigurisanje direktorijuma za čuvanje otpremljenih slika

Kreirajte `media` direktorijum u projektu:

```
mkdir media
```

Dodajte `MEDIA_ROOT` i `MEDIA_URL` u `settings.py` projekta:

```
MEDIA_ROOT = BASE_DIR / 'media'
MEDIA_URL = '/media/'
```

`MEDIA_ROOT` navodi direktorijum koji čuva otpremljenu sliku. `MEDIA_URL` navodi URL adresu koja služi za preuzimanje datoteka slika iz `MEDIA_ROOT` direktorijuma.

Dodajte URL adresu koja služi medijskim datotekama urls.pyprojekta na sledeći način:

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

from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
  path('admin/', admin.site.urls),
  path('',include('todo.urls')),
  path('',include('users.urls'))
]

if settings.DEBUG:
  urlpatterns += static(settings.MEDIA_URL,
              document_root=settings.MEDIA_ROOT)
```

Ako DEBUGse nalazi Trueu settings.pydatoteci, Django aplikacija će poslužiti medijske datoteke iz MEDIA_URL.

Kreiranje modela profila

Izmenite models.py aplikaciju usersi definišite Profilemodel na sledeći način:

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

class Profile(models.Model):
  user = models.OneToOneField(User, on_delete=models.CASCADE)

  avatar = models.ImageField(
    default='avatar.jpg', # default avatar
    upload_to='profile_avatars' # dir to store the image
  )

  def __str__(self):
    return f'{self.user.username} Profile'

  def save(self, *args, **kwargs):
    # save the profile first
    super().save(*args, **kwargs)

    # resize the image
    img = Image.open(self.avatar.path)
    if img.height > 300 or img.width > 300:
      output_size = (300, 300)
      # create a thumbnail
      img.thumbnail(output_size)
      # overwrite the larger image
      img.save(self.avatar.path)
```

Svaki korisnik ima profil i svaki profil pripada korisniku. Stoga je odnos između `User` modela i `Profile` modela `jedan na jedan`.

Da biste definisali odnos `jedan na jedan`, koristite OneToOneField:

```
user = models.OneToOneField(User, on_delete=models.CASCADE)
```

Ako `User` bude obrisan, `Profile` povezano sa `User` se takođe briše. Na ovo ukazuje `on_delete=models.CASCADE` parametr `OneToOneField()`.

Definišite `avatar` polje koje sadrži avatar korisnika:

```
avatar = models.ImageField(
    default='avatar.jpg', # default avatar
    upload_to='profile_avatars' # dir to store the image
)
```

Ako korisnici nisu otpremili avatare, podrazumevana datoteka je `avatar.jpg` datoteka. Takođe, navodimo direktorijum `profile_avatars` koji će čuvati otpremljene avatare.

Imajte na umu da možete dodati još polja modelu `Profile` kao što su adrese, interesovanja itd., ako je potrebno.

Preuzmite `avatar.jpg` datoteku i kopirajte je u `media` direktorijum projekta.

Definišite `__str__()` metod koji vraća string reprezentaciju modela `Profile`:

```
def __str__(self):
   return f'{self.user.username} Profile'
```

Definišite `save()` metod koji čuva profil u bazi podataka, kreira sličicu avatara i čuva ga u navedenom direktorijumu:

```
def save(self, *args, **kwargs):
  # save the profile first
  super().save(*args, **kwargs)

  # resize the image
  img = Image.open(self.avatar.path)
  if img.height > 150 or img.width > 150:
    output_size = (150, 150)
    # create a thumbnail
    img.thumbnail(output_size)

    # overwrite the large image
    img.save(self.avatar.path)
```

Registrujte profil u `admin.py` kako bismo mogli da upravljamo profilom na administratorskoj stranici:

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

admin.site.register(Profile)
```

Primena migracija

Izvršite migracije pokretanjem `makemigrations` komande:

```
python manage.py makemigrations

Migrations for 'users':
   users\migrations\0001_initial.py
    - Create model Profile
```

Drugo, primenite migracije:

```
python manage.py migrate

Operations to perform:
   Apply all migrations: admin, auth, contenttypes, sessions, todo, users
Running migrations:
   Applying users.0001_initial... OK
```

MyProfile prikaza

Definišite `MyProfile` prikaz zasnovan na klasi u `views.py` datoteci:

```
from django.views import View
# ...

class MyProfile(LoginRequiredMixin, View):
  def get(self, request):
    user_form = UserUpdateForm(instance=request.user)
    profile_form = ProfileUpdateForm(instance=request.user.profile)

    context = {
      'user_form': user_form,
      'profile_form': profile_form
    }

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

  def post(self,request):
    user_form = UserUpdateForm(
      request.POST,
      instance=request.user
    )
    profile_form = ProfileUpdateForm(
      request.POST,
      request.FILES,
      instance=request.user.profile
    )

    if user_form.is_valid() and profile_form.is_valid():
      user_form.save()
      profile_form.save()

      messages.success(request,'Your profile has been updated successfully')

      return redirect('profile')
    else:
      context = {
        'user_form': user_form,
        'profile_form': profile_form
      }
      messages.error(request,'Error updating you profile')

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

Definišite `MyView` klasu koja nasleđuje od `View` klase. `MyView` klasa ima metode `get()` i `post()` koje odgovaraju HTTP GET i POST.

Korišćenjem View klase, ne morate imati if-else grananje unutar funkcije prikaza da biste utvrdili da li je HTTP metod GET ili POST.

U `get()` metodi klase `MyProfile`, kreiramo objekte `UserUpdateForm` and `ProfileUpdateForm` i prosleđujemo ih šablonu `profile.html`.

U `post()` metodi takođe kreiramo objekte `UserUpdateForm` and `ProfileUpdateForm` , ali prosleđujemo dodatne podatke iz `request.POST` and `request.FILES` .

Kada su obe forme validne, čuvamo ih u bazi podataka, kreiramo fleš poruku i preusmeravamo korisnika nazad na stranicu profila.

Ako jedan od obrazaca nije važeći, kreiramo poruku o grešci i preusmeravamo korisnika nazad na stranicu profila, a zatim ponovo prikazujemo šablon.

Profile šablona

Napravite `profile.html` u `templates/users` direktorijumu koji proširuje `base.html` šablon:

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

{% block content %}


<div class="center">
  <form method="POST" enctype="multipart/form-data" class="card">
    {% csrf_token %}

    {% if user.profile %}
      <img src="{{user.profile.avatar.url}}" alt="{{ user.username }}"
class="avatar"  accept=".jpg,.jpeg,.png"/>
    {% endif %}

    <h2 class="text-center">{{ user.username | title }}</h2>
    <p class="text-center"><a href="mailto:{{user.email}}">{{user.email}}
    </a></p>
    <hr>
    <label for="email">Email Address:</label>
    <input type="email" id="email" name="email" value="{{user.email}}" />

    <label for="avatar">Avatar:</label>
    <input type="file" name="avatar" id="avatar">

    <button type="submit" class="btn btn-primary full-width">Update Profile
    </button>
  </form>
</div>

{% endblock content %}
```

Ručno kreiranje profila

Prijavite se na administratorski sajt koristeći nalog superkorisnika, videćete `Profiles` ispod `Users` aplikacije.

Kliknite na dugme `Dodaj profil` da biste kreirali novi profil za postojećeg korisnika.

Dodajte novi profil za korisnika "John" i kliknite na dugme "Save".

Django će pokazati da je profil za korisnika "John" uspešno dodat.

Ako se prijavite kao "John" i otvorite stranicu profila http://127.0.0.1:8000/profile/, videćete sledeću stranicu:

Na ovom obrascu možete ažurirati profil promenom adrese e-pošte i otpremanjem novog avatara.

Profila link

Izmenite `base.html` šablon da biste dodali URL profila u zaglavlje:

```
{%load static %}
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{% static 'css/style.css' %}" />
    <title>Todo List</title>
  </head>

  <body>
    <header class="header">
      <div class="container">
        <a href="{%url 'home'%}" class="logo">Todo</a>
        <nav class="nav">
          <a href="{%url 'home'%}"><i class="bi bi-house-fill"></i> Home</a>
          {% if request.user.is_authenticated %}
            <a href="{% url 'tasks' %}"><i class="bi bi-list-task"></i> My Tasks</
a>
            <a href="{% url 'task-create' %}"><i class="bi bi-plus-circle"></i>
Create Task</a>
            <a href="{% url 'profile' %}" title="Update my profile">Hi
{{request.user | title}}</a>
              <a href="{% url 'logout' %}" class="btn btn-outline">Logout</a>
          {% else %}
            <a href="{% url 'login' %}" class="btn btn-outline">Login</a>
            <a href="{% url 'register' %}" class="btn btn-primary">Join Now</a>
          {% endif %}
         </nav>
      </div>
    </header>
    <main>
      <div class="container">
        {% if messages %}
          {% for message in messages %}
            <div class="alert alert-{{message.tags}}">
              {{message}}
            </div>
          {% endfor %}
        {% endif %}

        {%block content %}
        {%endblock content%}
      </div>
    </main>
    <footer class="footer">
      <div class="container">
        <p>© Copyright {% now "Y" %} by <a href="https://
www.pythontutorial.net">Python Tutorial</a></p>
      </div>
    </footer>
```

```
  </body>
</html>
```

Automatsko kreiranje profila

Ako se prijavite kao "Jane" i pristupite stranici profila, dobićete sledeću grešku.

Razlog je taj što nismo kreirali profil za korisnika "Jane". Da bismo rešili ovaj problem, trebalo bi da kreiramo profil kada se korisnik uspešno registruje. Da bismo ovo implementirali, koristićemo nešto što se zove signali u Django-u.

Prvo, kreirajte `signals.py` datoteku u `users` aplikaciji sa sledećim kodom:

```
from django.db.models.signals import post_save
from django.contrib.auth.models import User
from django.dispatch import receiver
from .models import Profile

@receiver(post_save, sender=User)
def create_profile(sender, instance, created, **kwargs):
  if created:
    Profile.objects.create(user=instance)

@receiver(post_save, sender=User)
def save_profile(sender, instance, **kwargs):
  instance.profile.save()
```

U ovoj `signals.py` datoteci:

- Funkcija `create_profile()` kreira novi profil nakon što je `User` objekat kreiran.
- Funkcija `save_profile()` ažurira profil nakon što je `User` objekat sačuvan.

Dekorater `@receiver` povezuje `post_save` događaj modela `User` sa svakom funkcijom.

Uvezite signale u `apps.py` datoteku `users` aplikacije:

```
from django.apps import AppConfig

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

  def ready(self):
    import users.signals
```

U `apps.py` aplikaciji `users`, uvozimo `signals.py` metod klase `UsersConfig`.

Registrujte novog korisnika i proverite profil.

Rezime korisnički profil

- Koristite Django `signal` da biste automatski kreirali profil za korisnika.

Sadržaj