STRUČNI UVID

Django 3 kroz primere

Izradite snažne i pouzdane Python veb aplikacije od početka

III izdanje

Antonio Melé

kompjuter biblioteka

Packt

Prevod III izdanja

Django 3
kroz primere

Izradite snažne i pouzdane Python
veb aplikacije od početka

Antonio Melé

kompjuter
biblioteka

Packt›

Izdavač:

kompjuter biblioteka

Obalskih radnika 4a, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Antonio Melé

Prevod: Biljana Tešić

Lektura: Miloš Jevtović

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2020.

Broj knjige: 529

Izdanje: Prvo

ISBN: 978-86-7310-552-9

Django 3 By Example

Third Edition

Antonio Melé

ISBN 978-1-83898-195-2

O AUTORU

Antonio Melé je generalni tehnički direktor (CTO) u londonskoj fintech kompaniji „Nucoro“, koja obezbeđuje vodeću tehnološku platformu za izradu digitalnih rešenja za upravljanje finansijama. Antonio razvija Django projekte za klijente iz nekoliko industrija od 2006. godine, a 2009. godine je osnovao „Zenx IT“, razvojnu kompaniju specijalizovanu za izradu digitalnih proizvoda. Radi kao CTO i tehnološki konsultant za više novoosnovanih tehnoloških preduzeća i upravlja razvojnim timovima koji izrađuju projekte za velika digitalna preduzeća. Stekao je diplomu magistra iz računarskih nauka na Univerzitetu „Pontificia Comillas“. Otac ga je inspirisao za strast prema računarima i programiranju. Više informacija o Meléu možete saznati na njegovom veb sajtu https://antoniomele.es/.

O RECENZENTIMA

Jake Kronika, viši softverski inženjer, sa skoro 25 godina iskustva, koristi Python od 2005, a Django od 2007. godine. Razvijajući se istovremeno sa veb razvojnim prostorom, njegov skup veština, osim servera, obuhvata HTML5, CSS3 i JavaScript (ECMAScript 6) na frontendu, plus Python, Django, Node.js, PHP, Rubi on Rails i još mnogo štošta.

Trenutno vodeći softverski arhitekta i vođa razvojnog tima, Kronika sarađuje sa kvalifikovanim dizajnerima, poslovnim interesnim grupama i programerima širom sveta kako bi planirao i implementirao robusne veb aplikacije. U slobodno vreme obezbeđuje i čitav spektar veb usluga kao jedini vlasnik firme „Gridline Design and Development“, radi na projektima za unapređenje svoje kuće i uživa u trenucima provedenim sa suprugom i sa njihovo dvoje dece.

Kronika je koautor knjige „Django 2 Web Development Cookbook (third edition)”, objavljene u oktobru 2018. godine, i knjige „Django 3 Web Development Cookbook (fourth edition)“, objavljene u martu 2020. godine. Osim toga, bio je tehnički recenzent za nekoliko drugih naslova izdavačke kuće „Packt“, uključujući:

***Web** Development with Django Cookbook – Second Edition (2016)*

Developing Responsive Web Applications with AJAX and jQuery (2014)

jQuery Tools UI Library (2012)

jQuery UI 1.8: The User Interface Library for jQuery (2011)

DjangoJavaScriptIntegration: AJAX andjQuery (2011)

> *“Zahvaljujem se mojoj supruzi Veroniki za sve ono što čini da bi me podržala. Bez nje, ja bih bio delić osobe koja jesam i postigao bih samo mali procenat onoga što sam mogao.*
>
> *Takođe se zahvaljujem mom menadžeru Raviu za njegovu stalnu podršku i savete. Njegovi saveti su bili od velikog značaja u uspesima u mojoj karijeri.”*

David Stanek se profesionalno bavi razvojem softvera već više od 22 godine. Trenutno uživa u izradi distribuiranih sistema pomoću tehnologija kubernetes i cloud. Python je njegov jezik izbora više od 19 godina, ali uživa i u pisanju na Go jeziku i drugim specijalizovanim jezicima. Poslednjih godina učestvovao je u raznim cloud projektima i uživa u izazovima pokretanja aplikacija na većem broju računara. Najviše voli da piše veb servise u Pythonu, da ih raspoređuje na infrastrukturi „orkestriranoj“ pomoću Terraforma i da automatizuje njihove poslovne procese pomoću Ansiblea.

Stanek provodi veći deo svog radnog vremena tako što izrađuje projekte otvorenog koda i razvija svoje tehničke veštine. Uživa u podučavanju, čitanju tehničkih knjiga i slušanju raznih podcastova i audio-knjiga. Kada ne radi, uživa da provodi vreme sa svojom lepom suprugom, četvoro divne dece i malim jazavičarom.

Više informacija o Davidu Staneku možete saznati na njegovom veb sajtu (http://dstanek.com/).

Predgovor

Django je moćan Python veb radni okvir koji podstiče brz razvoj i čist, pragmatičan dizajn, a istovremeno obezbeđuje relativno jednostavnu krivu učenja. To ga čini privlačnim i za početnike i za profesionalne programere.

Ova knjiga će vas voditi kroz celokupni proces razvoja profesionalnih veb aplikacija pomoću radnog okvira Django. Knjiga ne samo da obuhvata najrelevantnije aspekte ovog radnog okvira, već ćete pomoću nje naučiti i kako da integrišete druge popularne tehnologije u svoje Django projekte.

U ovoj knjizi ćemo predstaviti kreiranje aplikacija u realnom svetu, rešavanje uobičajenih problema i primenu najbolje prakse, koristeći pristup „korak po korak", koji je lako slediti.

Kada pročitate ovu knjigu, dobro ćete razumeti kako Django funkcioniše i kako možete da kreirate praktične, napredne veb aplikacije.

KOME JE NAMENJENA OVA KNJIGA

Ova knjiga je namenjena programerima koji poznaju Python i koji žele da nauče Django na pragmatičan način. Možda ste potpuno novi u radnom okviru Django ili možda ga već malo poznajete, ali želite da dobijete maksimum iz njega. Ova knjiga će vam pomoći da savladate najrelevantnije oblasti ovog radnog okvira, tako što ćete izrađivati praktične projekte „od nule". Da biste čitali ovu knjigu, morate da poznajete koncepte programiranja. Podrazumeva se da imate neko prethodno znanje o HTML-u i JavaScriptu.

ŠTA OBUHVATA OVA KNJIGA

U Poglavlju 1, „Izrada aplikacije za blog“, upoznaćete radni okvir, koristeći aplikaciju za blog. Kreiraćete osnovne modele bloga, prikaze (views), šablone (templates) i URL-ove za prikazivanje postova na blogu. Naučićete kako da izradite QuerySets pomoću Django objektnog-relacionog mapera (ORM – Object-Relational Mapper) i kako da konfigurišete Django administratorski sajt.

U Poglavlju 2, „Poboljšanje bloga pomoću naprednih funkcija“, naučićete kako da upravljate obrascima i ModelFormsima, da šaljete e-poštu pomoću Djangoa i da integrišete nezavisne aplikacije. Primenićete sistem za komentare na svoje blogove i omogućićete korisnicima da dele postove pomoću e-pošte. Ovo poglavlje će vas takođe voditi kroz proces kreiranja sistema za označavanje.

U Poglavlju 3, „Proširenje aplikacije za blog“, istražujemo kako se kreiraju prilagođene oznake šablona i filteri. Takođe će biti prikazano kako se koristi radni okvir mape sajta (sitemap) i kako se kreira RSS feed za postove. Završićete svoju aplikaciju za blog izradom pretraživača koji ima mogućnost PostgreSQL-ovog pretraživanja punog teksta.

U Poglavlju 4, „Izrada društvenog veb sajta“, objašnjeno je kako se kreira društveni veb sajt. Koristićete Django radni okvir za autentifikaciju da biste kreirali prikaz korisničkih naloga. Takođe ćete naučiti kako da kreirate prilagođeni model korisničkog profila i kako da ugradite društvenu autentifikaciju u svoj projekat, koristeći glavne društvene mreže.

U Poglavlju 5, „Deljenje sadržaja na veb sajtu“, naučićete kako da transformišete svoju društvenu aplikaciju u veb sajt za merenje popularnosti slika. Definisaćete veze tipa „više prema više“ i kreiraćete AJAX aktivni obeleživač (bookmarklet) u JavaScriptu koji ćete integrisati u svoj projekat. U ovom poglavlju će biti prikazano kako se generiše umanjeni prikaz slika i kako se kreiraju prilagođeni dekoratori za prikaze.

U Poglavlju 6, „Praćenje korisničkih radnji“, prikazano je kako se izrađuje sistem praćenja za korisnike. Završićete veb sajt za merenje popularnosti slika, tako što ćete kreirati aplikaciju za tok korisničkih aktivnosti. Naučićete kako da optimizujete QuerySets i koristićete signale. Na kraju ćete integrisati Redis u svoj projekat da biste izbrojali prikaze slika.

U Poglavlju 7, „Izrada internet prodavnice“, istražujemo kako se kreira internet prodavnica. Izradićete modele kataloga i kreiraćete korpu za kupovinu, koristeći Django sesije. Izradićete kontekstni procesor za korpu za kupovinu i naučićete kako da primenite slanje asinhronih obaveštenja korisnicima koji koriste Celery.

U Poglavlju 8, „Upravljanje plaćanjem i narudžbenicama“, objašnjeno je kako možete da integrišete platni mrežni prolaz u svoju prodavnicu. Takođe ćete prilagoditi administratorski sajt za izvoz narudžbenica u CSV datoteke i dinamički ćete generisati PDF fakture.

U Poglavlju 9, „Proširenje prodavnice“, naučićete kako da kreirate sistem za kupone za primenu popusta na porudžbine. Takođe ćemo prikazati kako da dodate internacionalizaciju svom projektu i kako da prevedete modele. Na kraju ćete kreirati mehanizam za preporuku proizvoda pomoću Redisa.

Poglavlje 10, „Izrada platforme za elektronsko učenje“, vodiće vas kroz kreiranje platforme za elektronsko učenje. U projektu ćete da dodate fixture (skup podataka), da koristite nasleđivanje modela, da kreirate polja prilagođenih modela, da koristite prikaze zasnovane na klasama i da upravljate grupama i dozvolama. Takođe ćete da kreirate sistem za upravljanje sadržajem i da upravljate skupovima obrazaca (formsets).

U Poglavlju 11, „Renderovanje i keširanje sadržaja“, biće prikazano kako možete da kreirate sistem za registraciju učenika i kako da upravljate upisom učenika na kurseve. Renderovaćete različite sadržaje kursa i naučićete kako da koristite radni okvir keš memorije.

U Poglavlju 12, „Izrada API-a“, naučićete postupak izrade RESTful API-a za svoj projekat, koristeći Django REST radni okvir.

U Poglavlju 13, „Izrada servera za ćaskanje“, objasnićemo kako se koriste Django kanali za kreiranje servera za ćaskanje u realnom vremenu za učenike. Naučićete kako da primenite funkcije koje se oslanjaju na asinhronu komunikaciju pomoću WebSocketsa.

U Poglavlju 14, „Akcija“, biće prikazano kako da podesite proizvodno okruženje, koristeći uWSGI, NGINX i Daphne. Naučićete kako da obezbedite okruženje pomoću HTTPS-a. U ovom poglavlju je takođe objašnjeno kako se kreiraju prilagođeni posrednički softver i prilagođene komande za upravljanje.

IZVUCITE MAKSIMUM IZ OVE KNJIGE

Čitalac treba:

da poseduje dobro poznavanje rada u Pythonu

da poznaje HTML i JavaScript

da pročita delove od 1 do 3 uputstva u zvaničnoj dokumentaciji za Django na adresi https://docs.djangoproject.com/en/3.0/intro/tutorial01/.

PREUZIMANJE DATOTEKA SA PRIMERIMA KODA

Datoteke sa primerima koda za ovu knjigu možete da preuzmete sa našeg sajta:

https://bit.ly/3hY8eG5

Kada je datoteka preuzeta, raspakujte ili ekstrahujte direktorijum, koristeći najnoviju verziju:

- WinRAR/7-Zip za Windows
- Zipeg/iZip/UnRarX za Mac
- 7-Zip/PeaZip za Linux

PREUZMITE KOLORNE SLIKE

Takođe smo obezbedili PDF datoteku sa kolornim slikama snimaka ekrana/dijagrama upotrebljenih u ovoj knjizi. Možete da je preuzmete na adresi:

https://bit.ly/2BGiQbP

UPOTREBLJENE KONVENCIJE

Postoji veliki broj konvencija teksta koje su upotrebljene u ovoj knjizi.

Code InText - Ukazuje na reči koda u tekstu, nazive tabela baze podataka, nazive direktorijuma, nazive datoteka, ekstenzije datoteka, nazive putanje, skraćene URL-ove, korisnički unos i Twitter postove. Evo i primera: „Uredite datoteku .pyaplikacije shop.“

Blok koda je prikazan na sledeći način:

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

admin.site.register(Post)
```

Kada želimo da vam skrenemo pažnju na određeni deo bloka koda, relevantne linije ili stavke, pišemo zadebljanim slovima:

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

Unos u komandnoj liniji napisan je na sledeći način:

```
python manage.py runserver
```

Zadebljana slova - Ukazuju na novi termin, važnu reč ili reči koje vidite na ekranu. Na primer, reči u menijima ili okvirima za dijalog prikazane su u tekstu zadebljanim slovima. Evo i primera: „Popunite obrazac i kliknite na dugme Save."

Napomene ili važna obaveštenja prikazani su ovako.

Saveti i trikovi su prikazani ovako.

POVRATNE INFORMACIJE

Povratne informacije od naših čitalaca su uvek dobrodošle.

Osnovne povratne informacije - Ako imate bilo kakva pitanja o bilo kom aspektu ove knjige, pošaljite nam e-poruku na adresu kombib@gmail.com i u naslovu napišite naziv knjige.

Štamparske greške - Iako smo preduzeli sve mere da bismo obezbedili tačnost sadržaja, greške su moguće. Ako pronađete grešku u ovoj knjizi, bili bismo zahvalni ako biste nam to prijavili. Posetite stranicu knjige:

https://bit.ly/37Y0OOu

i u polje ya komentar unesite detalje.

Piraterija - Ako pronađete ilegalnu kopiju naših knjiga u bilo kojoj formi na Internetu, molimo vas da nas o tome obavestite i da nam pošaljete adresu lokacije ili naziv veb sajta. Kontaktirajte sa nama na adresi kombib@gmail.com i pošaljite nam link ka sumnjivom materijalu.

RECENZIJA

Kada pročitate i upotrebite ovu knjigu, napišite svoje mišljenje na sajtu sa kojeg ste je poručili. Potencijalni čitaoci tada mogu da upotrebe vaše mišljenje da bi se opredelili za kupovinu ove knjige, mi u „Packtu“ i “Kompjuter biblioteci” možemo da znamo šta mislite o našim proizvodima, a naši autori mogu da vide povratne informacije o svojoj knjizi.

Postanite član Kompjuter biblioteke

Kupovinom jedne naše knjige stekli ste pravo da postanete član Kompjuter biblioteke. Kao član možete da kupujete knjige u pretplati sa 40% popusta i učestvujete u akcijama kada ostvarujete popuste na sva naša izdanja. Potrebno je samo da se prijavite preko formulara na našem sajtu.
Link za prijavu: http://bit.ly/2TxekSa

Skenirajte QR kod
registrujte knjigu
i osvojite nagradu

1

Izrada aplikacije za blog

Django je moćan Python veb radni okvir sa relativno jednostavnom krivom učenja. U kratkom vremenskom periodu možete lako kreirati jednostavne veb aplikacije. Django je takođe robustan i skalabilan radni okvir koji se može koristiti za izradu veb aplikacija na više računara pomoću kompleksnih zahteva i integracija. To ga čini privlačnim i za početnike i za profesionalne programere.

U ovoj knjizi naučićete kako da izradite kompletne Django projekte spremne za upotrebu. Ako još niste instalirali Django, otkrićete kako se to radi u prvom delu ovog poglavlja.

U ovom poglavlju je opisano kako se kreira jednostavna aplikacija za blog pomoću radnog okvira Django. Svrha ovog poglavlja je da vam pomogne da steknete opštu ideju kako radni okvir funkcioniše, da razumete način na koji različite komponente međusobno funkcionišu i da razumete veštine za lako kreiranje Django projekata sa osnovnom funkcionalnošću. Upoznaćete kreiranje kompletnog projekta. Različite komponente radnog okvira biće detaljno istražene u ovoj knjizi.

Ovo poglavlje obuhvata sledeće teme:

- instalacija Djangoa
- kreiranje i konfigurisanje Django projekta
- kreiranje Django aplikacije
- dizajniranje modela i generisanje migracija modela
- izrada administratorskog sajta za modele

- upotreba QuerySetova i menadžera
- izrada prikaza, šablona i URL-ova
- dodavanje numerisanja stranica u listi prikaza
- korišćenje Djangovih prikaza koji se zasnivaju na klasama

INSTALIRANJE DJANGOA

Ako ste već instalirali Django, možete preskočiti ovaj odeljak i preći direktno na odeljak „Kreiranje prvog projekta". Django se isporučuje kao Python paket, pa se može instalirati u svako Python okruženje. Ako još niste instalirali radni okvir Django, u nastavku sledi brzi vodič za njegovu instalaciju za lokalni razvoj.

Django 3 nastavlja „put" obezbeđivanja novih funkcija, uz održavanje osnovnih funkcionalnosti radnog okvira. Verzija 3.0 prvi put uključuje podršku za **Asynchronous Server Gateway Interface (ASGI)**, što Django čini potpuno asinhronim. Django 3.0 takođe uključuje zvaničnu podršku za sistem MariaDB, nova ograničenja za izuzeće u PostgreSQL-u, poboljšanja izraza filtera i enumeraciju za izbor modela polja, ali i druge nove funkcije.

Django 3.0 podržava Python 3.6, 3.7 i 3.8. U primerima u ovoj knjizi koristićemo Python 3.8.2. Ako koristite Linux ili macOS, verovatno imate instaliran Python. Ako koristite Windows, program za instalaciju Pythona možete preuzeti na adresi https://www.python.org/downloads/windows/.

Ako niste sigurni da li je Python instaliran na vašem računaru, to možete da proverite tako što ćete upisati `python` u komandno okruženje. Ako vidite nešto poput sledećeg koda, znači da je Python instaliran na vašem računaru:

```
Python 3.8.2 (v3.8.2:7b3ab5921f, Feb 24 2020, 17:52:18)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

Ako je instalirana verzija Pythona koja je starija od verzije 3.6 ili ako Python nije instaliran na vašem računaru, preuzmite Python 3.8.2 sa adrese https://www.python.org/downloads/, a zatim ga instalirajte.

Pošto ćete koristiti Python 3, ne morate da instalirate bazu podataka. Ova verzija Pythona se isporučuje sa ugrađenom SQLite bazom podataka. SQLite je laka baza podataka koju možete koristiti pomoću Djangoa za programiranje. Ako planirate da primenite aplikaciju u proizvodnom okruženju, trebalo bi da koristite baze podataka sa svim funkcijama, kao što su PostgreSQL, MySQL ili Oracle. Više informacija o tome kako pokrenuti bazu podataka pomoću Djangoa možete pronaći na adresi https://docs.djangoproject.com/en/3.0/topics/install/#database-installation.

Kreiranje izolovanog Python okruženja

Od verzije 3.3 Python se isporučuje sa bibliotekom `venv`, koja obezbeđuje podršku za kreiranje lakih virtuelnih okruženja. Svako virtuelno okruženje ima svoj Python binarni sistem i može imati nezavisni skup instaliranih Python paketa u svojim direktorijumima veb sajtova. Korišćenje Python modula `venv` za kreiranje izolovanih Python okruženja omogućava korišćenje različitih verzija paketa za različite projekte, što je daleko praktičnije od instaliranja Python paketa na celom sistemu. Još jedna prednost korišćenja biblioteke `venv` je što vam neće biti potrebne nikakve administrativne privilegije za instaliranje Python paketa.

Kreirajte izolovano okruženje pomoću sledeće komande:

```
python -m venv my_env
```

Ova komanda će kreirati direktorijum `my_env/`, uključujući i vaše Python okruženje. Sve Python biblioteke koje instalirate dok je vaše virtuelno okruženje aktivno biće smeštene u direktorijumu `my_env/lib/python3.8/site-packages`.

Da biste aktivirali virtuelno okruženje, pokrenite sledeću komandu:

```
source my_env/bin/activate
```

Komandni odzivnik sadrži naziv aktivnog virtuelnog okruženja u zagradama na sledeći način:

```
(my_env)laptop:~ zenx$
```

Okruženje možete bilo kada deaktivirati pomoću komande `deactivate`. Više informacija o biblioteci `venv` možete pronaći na stranici https://docs.python.org/3/library/venv.html.

Instaliranje Djangoa pomoću paketa pip

Sistem upravljanja paketima `pip` je preferirani metod za instaliranje Djangoa. Python 3.8 se isporučuje sa unapred instaliranim paketom `pip`, a uputstva za instalaciju ovog paketa možete pronaći na adresi https://pip.pypa.io/en/stable/installing/.

Da biste instalirali Django pomoću paketa `pip`, pokrenite sledeću komandu u odzivniku komandnog okruženja:

```
pip install "Django==3.0.*"
```

Django će biti instaliran u Python direktorijumu `site-packages` vašeg virtuelnog okruženja.

Sada proverite da li je Django uspešno instaliran. Pokrenite `python` u terminalu, uvezite Django i proverite njegovu verziju na sledeći način:

```
>>> import django
>>> django.get_version()
'3.0.4'
```

Ako dobijete izlaz, kao što je `3.0.X`, znači da je Django uspešno instaliran na vašem uređaju.

Django se može instalirati na nekoliko drugih načina. Kompletan vodič za instalaciju možete pronaći na adresi https://docs.djangoproject.com/en/3.0/topics/install/.

KREIRANJE PRVOG PROJEKTA

Vaš prvi projekat Django biće kreiranje kompletnog bloga. Django obezbeđuje komandu koja omogućava da kreirate početnu strukturu datoteke. Pokrenite sledeću komandu iz vašeg komandnog okruženja:

```
django-admin startproject mysite
```

Ova komanda će kreirati Django projekat `mysite`.

Nemojte imenovati projekte po ugrađenim Python ili Django modulima da biste izbegli „sukobe“ naziva.

Sada ćete videti strukturu generisanih projekata:

```
mysite/
    manage.py
    mysite/
      __init__.py
      asgi.py
      wsgi.py
      settings.py
      urls.py
```

Ove datoteke su sledeće:

- `manage.py` - Ovo je uslužni program komandne linije, koji se koristi za interakciju sa vašim projektom. To je tanak omotač oko alatke `django-admin.py`. Ne morate da uređujete ovu datoteku.
- `mysite/` - Ovo je direktorijum projekta koji se sastoji od sledećih datoteka:
  - `__init__.py` - Ovo je prazna datoteka koja ukazuje Pythonu da treba da tretira direktorijum `mysite` kao Python modul.
  - `asgi.py` - Ovo je konfiguracija za pokretanje vašeg projekta kao ASGI, tj. novi Python standard za asinhrone veb servere i aplikacije.
  - `settings.py` - Ovo označava postavke i konfiguraciju za vaš projekat i sadrži početna podrazumevana podešavanja.
  - `urls.py` - Ovde su smešteni uzorci URL-ova. Svaki URL koje je ovde definisan mapiran je u prikaz.
  - `wsgi.py` - Ovo je konfiguracija za pokretanje vašeg projekta kao aplikacije **WebServer Gateway Interface (WSGI)**.

Generisana datoteka `settings.py` sadrži postavke projekta, uključujući osnovnu konfiguraciju za upotrebu baze podataka SQLite3 i listu zvanu `INSTALLED_APPS` koja sadrži uobičajene Django aplikacije koje su podrazumevano dodate u vaš projekat. Razmotrićemo ove aplikacije kasnije u odeljku „*Postavke projekta*“.

Django aplikacije sadrže datoteku `models.py`, u kojoj su definisani modeli podataka. Svaki model podataka mapira se u tabelu baze podataka. Da biste dovršili podešavanje projekta, morate da kreirate tabele povezane sa modelima aplikacija navedenih u listi `INSTALLED_APPS`. Django uključuje sistem migracije za kreiranje tih tabela.

Otvorite komandno okruženje (shell) i pokrenite sledeće komande:

```
cd mysite
python manage.py migrate
```

Primetićete izlaz koji se završava sledećim linijama:

```
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 sessions.0001_initial... OK
```

Prethodne linije su migracije baze podataka koje primenjuje Django. Primenom migracija kreiraju se tabele za početne aplikacije u bazi podataka. Više informacija o komandi upravljanja `migrate` saznaćete u odeljku *„Kreiranje i primena migracija“* u ovom poglavlju.

Pokretanje razvojnog servera

Django se isporučuje sa lakim veb serverom za brzo pokretanje koda, bez potrebe da se troši vreme na konfigurisanje proizvodnog servera. Kada pokrenete razvojni server Django, on i dalje proverava promene u vašem kodu i učitava se automatski, pa ne morate ponovo da ga ručno unesete nakon promene koda. Međutim, možda razvojni server neće primetiti neke radnje, poput dodavanja novih datoteka vašem projektu, tako da ćete u tim slučajevima morati ručno da ga pokrenete.

Pokrenite razvojni server, tako što ćete ukucati sledeću komandu iz osnovne fascikle vašeg projekta:

```
python manage.py runserver
```

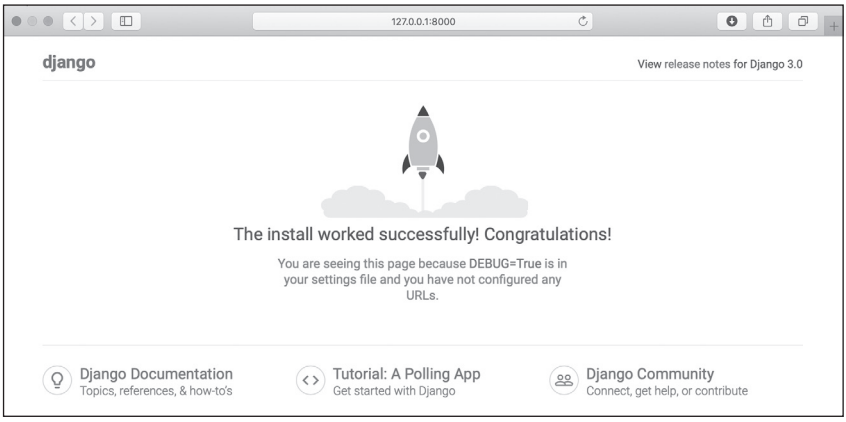
Trebalo bi da vidite nešto ovako:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

January 01, 2020 - 10:00:00
Django version 3.0, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Sada otvorite `http://127.0.0.1:8000/` u vašem pregledaču. Trebalo bi da vidite stranicu na kojoj je navedeno da je projekat uspešno pokrenut, kao što je prikazano na sledećem snimku ekrana.



Slika 1.1 Podrazumevana stranica Django razvojnog servera

Na prethodnom snimku ekrana prikazano je da je Django pokrenut. Ako pogledate konzolu, videćete `GET` zahtev koji izvršava vaš pregledač:

```
[01/Jan/2020 17:20:30] "GET / HTTP/1.1" 200 16351
```

Svaki HTTP zahtev evidentira razvojni server u konzoli. Svaka greška koja se dogodi tokom pokretanja razvojnog servera takođe će biti prikazana u konzoli.

Možete pokrenuti Django razvojni server na prilagođenom hostu i portu ili ukazati Djangou da treba da učita određenu datoteku postavki na sledeći način:

```
python manage.py runserver 127.0.0.1:8001 \--settings=mysite.settings
```

Kada morate da koristite više okruženja koja zahtevaju različite konfiguracije, možete da kreirate različitu datoteku postavki za svako okruženje.

Zapamtite da je ovaj server namenjen samo za razvoj i nije pogodan za upotrebu u proizvodnji. Da biste Django rasporedili u proizvodnom okruženju, trebalo bi da ga pokrenete kao WSGI aplikaciju pomoću veb servera, kao što su Apache, Gunicorn ili uWSGI, ili kao ASGI aplikaciju pomoću servera, kao što su Uvicorn ili Daphne. Više informacija o načinu kako se implementira Django pomoću različitih veb servera možete pronaći na adresi https://docs.djangoproject.com/en/3.0/howto/deployment/wsgi/.

U Poglavlju 14, „*Akcija*“, objašnjeno je kako možete da podesite proizvodno okruženje za svoje Django projekte.

Postavke projekta

Sada ćemo otvoriti datoteku `settings.py` i pogledaćemo konfiguraciju projekta. Postoji nekoliko postavki koje Django uključuje u ovu datoteku, ali one su samo deo svih dostupnih Django postavki. Sve postavke i njihove podrazumevane vrednosti možete videti na stranici https://docs.djangoproject.com/en/3.0/ref/settings/.

Sledeće postavke vredi pogledati:

- `DEBUG` je Bulova vrednost koja uključuje i isključuje režim debagovanja u projektu. Ako je ovaj režim postavljen na vrednost `True`, Django će prikazati detaljne stranice o greškama kada vaša aplikacija generiše neuobičajeni izuzetak. Kada pređete na proizvodno okruženje, morate ga postaviti na vrednost `False`. Nikada nemojte pokrenuti sajt kada je uključen režim `DEBUG`, jer ćete izložiti javnosti poverljive podatke u vezi sa projektom.
- `ALLOWED_HOSTS` se ne primenjuje dok je režim debagovanja uključen ili kada su testovi pokrenuti. Kada pokrenete svoj sajt i postavite `DEBUG` na vrednost `False`, moraćete da ovoj postavci dodate svoj domen/host koji će opslužiti vaš Django sajt.
- `INSTALLED_APPS` je postavka koju ćete morati da uređujete za sve projekte. Ova postavka ukazuje Djangou koje aplikacije su aktivne za vaš sajt. Django podrazumevano uključuje sledeće aplikacije:
  - `django.contrib.admin` - administratorski sajt
  - `django.contrib.auth` - radni okvir za autentifikaciju
  - `django.contrib.contenttypes` - radni okvir za upravljanje tipovima sadržaja
  - `django.contrib.sesions` - radni okvir sesije
  - `django.contrib.messages` - radni okvir za razmenu poruka
  - `django.contrib.staticfiles` - radni okvir za upravljanje statičkim datotekama.
- `MIDDLEWARE` je lista koja sadrži posrednički softver koji treba da se izvrši.
- `ROOT_URLCONF` ukazuje na Python modul u kojem su definisani osnovni uzorci URL-a vaše aplikacije.
- `DATABASES` je rečnik koji sadrži postavke za sve baze podataka koje se koriste u projektu. Uvek mora postojati podrazumevana baza podataka. Podrazumevana konfiguracija koristi SQLite3 bazu podataka.
- `LANGUAGE_CODE` definiše podrazumevani kod jezika za ovaj Django sajt.

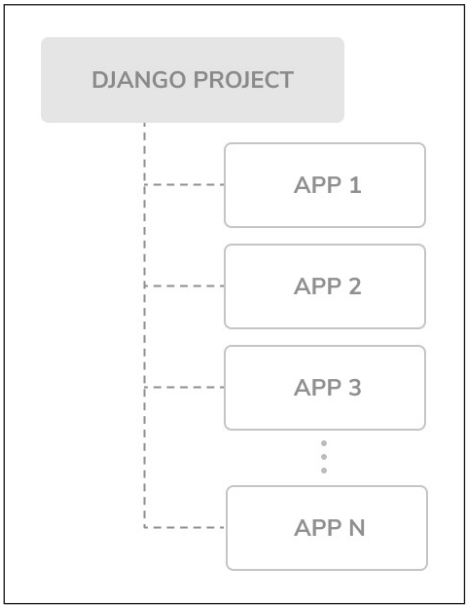
- `USE_TZ` ukazuje Djangou da treba da aktivira/deaktivira podršku za vremensku zonu. Django se isporučuje sa podrškom za datum i vreme za koje se prepoznaje vremenska zona. Ova postavka je postavljena na vrednost `True` kada se kreira novi projekat pomoću komande za upravljanje `startproject`.

Ne brinite ako ne razumete dobro ono što ovde vidite. U sledećim poglavljima ćete upoznati različite postavke Djangoa.

Projekti i aplikacije

U ovoj knjizi ćete se stalno susretati sa terminima **projekat** i **aplikacija**. U Djangou se projekat smatra Django instalacijom sa nekim postavkama. Aplikacija je grupa modela, prikaza, šablona i URL-ova. Aplikacije komuniciraju sa radnim okvirom da bi obezbedile neke specifične funkcionalnosti i mogu se ponovo upotrebiti u različitim projektima. O projektu možete razmišljati kao o svom veb sajtu, koji sadrži nekoliko aplikacija, kao što su blog, wiki ili forum, a koje se mogu koristiti i u drugim projektima.

Na sledećem dijagramu prikazana je struktura Django projekta.



Slika 2.1 Struktura Django projekta/aplikacije

Kreiranje aplikacije

Sada ćete da kreirate vašu prvu Django aplikaciju. Kreirajte aplikaciju za blog „od nule“. Iz osnovnog direktorijuma projekta pokrenite sledeću komandu:

```
python manage.py startapp blog
```

Ova komanda će kreirati osnovnu strukturu aplikacije koja izgleda ovako:

```
blog/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

Ove datoteke su sledeće:

- `admin.py` - Ovde registrujete modele da biste ih uključili u Django administratorski sajt; korišćenje ovog sajta nije obavezno.
- `apps.py` - Ova datoteka uključuje glavnu konfiguraciju aplikacije `blog`.
- `migrations` - Ovaj direktorijum će sadržati migracije baze podataka vaše aplikacije. Migracije omogućavaju Djangou da prati promene vašeg modela i da, u skladu s tim, sinhronizuju bazu podataka.
- `models.py` - Ova datoteka uključuje modele podataka vaše aplikacije. Sve Django aplikacije moraju da imaju datoteku `models.py`, ali ova datoteka može biti prazna.
- `test.py` - Ovde možete dodati testove za svoju aplikaciju.
- `views.py` - Logika vaše aplikacije je smeštena u ovoj datoteci; svaki prikaz prima HTTP zahtev, obrađuje ga i vraća odgovor.

DIZAJNIRANJE ŠEME PODATAKA BLOGA

Dizajniranje šeme podataka bloga ćete započeti definisanjem modela podataka za vaš blog. Model je Python klasa koja ima potklase `django.db.models.Model`, u kojima svaki atribut predstavlja polje baze podataka. Django će kreirati tabelu za svaki model koji je definisan u datoteci `models.py`. Kada kreirate model, Django će vam obezbediti praktičan API za lako pretraživanje objekata u bazi podataka.

Prvo morate da definišete model `Post`. U datoteku `models.py` aplikacije `blog` dodajte sledeće linije:

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

class Post(models.Model):
    STATUS_CHOICES = (
        ('draft', 'Draft'),
        ('published', 'Published'),
    )
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250,
                            unique_for_date='publish')
    author = models.ForeignKey(User,
                               on_delete=models.CASCADE,
                               related_name='blog_posts')
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    status = models.CharField(max_length=10,
                               choices=STATUS_CHOICES,
                               default='draft')

    class Meta:
        ordering = ('-publish',)

    def __str__(self):
        return self.title
```

Ovo je vaš model podataka za postove na blogu. Sada ćemo prikazati polja koja ste upravo definisali za ovaj model:

- `title` - Ovo je polje za naslov posta. Ovo polje je `CharField`, koje se prevodi u kolonu `VARCHAR` u SQL bazi podataka.
- `slug` - Ovo je polje koje se koristi u URL-ovima. Slug je kratka oznaka koja sadrži samo slova, brojeve, donje crte ili crtice. Polje `slug` ćete koristiti da biste kreirali lepe SEO-friendlyURL-ove za svoje blogove. U ovo polje ste dodali parametar `unique_for_date` da biste mogli da kreirate URL-ove za postove, koristeći datum objavljivanja (`publish`) i `slug`. Django će sprečiti da više postova ima isti slug za određeni datum.

- `author` - Ovo polje definiše vezu tipa „više prema jedan“, što znači da korisnik piše svaki post, a može da napiše bilo koji broj postova. Za ovo polje Django će kreirati spoljni ključ u bazi podataka pomoću primarnog ključa povezanog modela. U ovom slučaju se oslanjate na model `User` Django sistema za autentifikaciju. Parametar `on_delete` određuje ponašanje koje treba usvojiti kada se referentni objekat izbriše. Ovo nije specifično za Django; to je SQL standard. Koristeći `CASCADE`, vi određujete da baza, kada bude izbrisan referentni korisnik, izbriše i sve povezane komentare na blogu. Sve moguće opcije možete pogledati na stranici https://docs.djangoproject.com/en/3.0/ref/models/fields/#django.db.models. Odredite naziv reverzne veze, od korisnika (`User`) do posta (`Post`), pomoću atributa `related_name`. To će vam omogućiti jednostavan pristup povezanim objektima. O tome ćete više detalja saznati kasnije.
- `body` - Ovo je „telo“ posta. Ono je tekstualno polje koje se prevodi u kolonu `TEXT` u SQL bazi podataka.
- `publish` - Ovaj tip podataka datetime označava kada je post objavljen. Vi koristite Djangov metod za vremensku zonu `now` kao podrazumevanu vrednost. Ovo polje vraća tekući datum i vreme u formatu koji prepoznaje vremensku zonu. Ovo možete smatrati verzijom standardnog Python metoda `datetime.now`, koji prepoznaje vremensku zonu.
- `created` - Ovaj tip podataka datetime označava kada je post kreiran. Pošto ovde koristite `auto_now_add`, datum će biti automatski memorisan prilikom kreiranja objekta.
- `updated` - Ovaj tip podataka datetime označava kada je poslednji put post ažuriran. Pošto ovde koristite `auto_now`, datum će biti automatski ažuriran prilikom memorisanja objekta.
- `status` - Ovo polje prikazuje status posta. Koristite parametar `choices`, tako da vrednost ovog polja može biti postavljena samo na jedan od zadatih izbora.

Django se isporučuje sa različitim tipovima polja koja možete da koristite za definisanje svojih modela. Sve tipove polja možete pronaći na stranici https://docs.djangoproject.com/en/3.0/ref/models/fields/.

Klasa `Meta` unutar modela sadrži metapodatke. Kada postavljate upit u bazu podataka, ukazujete Djangou da treba da sortira rezultate prema polju `publish` opadajućim redosledom. Vi određujete opadajući redosled upotrebom negativnog prefiksa, pa će se prvi put pojaviti postovi koji su nedavno objavljeni.

Metod `__str__()` je podrazumevani objekat koji mogu čitati ljudi. Django će ga koristiti na mnogim mestima - na primer, na administratorskom sajtu.

Ako koristite Python 2.x, imajte na umu da se u Pythonu 3 svi znakovni nizovi izvorno smatraju Unicodeom; stoga, koristimo samo metod `__str__()`, a metod `__unicode__()` je zastareo.

Aktiviranje aplikacije

Da bi Django mogao da prati vašu aplikaciju i da kreira tabele baza podataka za svoje modele, morate da aktivirate aplikaciju. Da biste to učinili, uredite datoteku `settings.py` i dodajte `blog.apps.BlogConfig` u postavku `INSTALLED_APPS`. To bi trebalo da izgleda ovako:

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

Klasa `BlogConfig` je vaša konfiguracija aplikacije. Sada Django „zna“ da je vaša aplikacija aktivna za ovaj projekat i moći će da učita svoje modele.

Kreiranje i primena migracija

Pošto sada imate model podataka za svoje postove na blogu, biće vam potrebna tabela baza podataka. Django se isporučuje sa sistemom migracije koji prati izmene izvršene u modelima i omogućava im da se šire u bazu podataka. Kao što je već napomenuto, komanda `migrate` primenjuje migracije na sve aplikacije navedene u postavci `INSTALLED_APPS` i sinhronizuje bazu podataka sa trenutnim modelima i postojećim migracijama.

Prvo ćete morati da kreirate početnu migraciju za svoj `Post` model. U osnovnom direktorijumu vašeg projekta pokrenite sledeću komandu:

```
python manage.py makemigrations blog
```

Trebalo bi da dobijete sledeći izlaz:

```
Migrations for 'blog':
    blog/migrations/0001_initial.py
        - Create model Post
```

Django je upravo kreirao datoteku `0001_initial.py` unutar direktorijuma `migrations` aplikacije `blog`. Možete otvoriti tu datoteku da biste videli kako se prikazuje migracija. Migracija određuje zavisnosti od drugih migracija i operacija koje se izvršavaju u bazi podataka da bi bila sinhronizovana sa izmenama modela.

Sada ćemo pogledati SQL kod koji će Django izvršiti u bazi podataka da bi bila kreirana tabela za vaš model. Komanda `sqlmigrate` prihvata nazive migracija i vraća njihov SQL bez izvršenja. Da biste ispitali SQL izlaz svoje prve migracije, pokrenite sledeću komandu:

```
python manage.py sqlmigrate blog 0001
```

Izlaz bi trebalo da izgleda ovako:

```
BEGIN;
--
-- Create model Post
--
CREATE TABLE "blog_post" ("id" integer NOT NULL PRIMARY KEY
AUTOINCREMENT, "title" varchar(250) NOT NULL, "slug" varchar(250) NOT
NULL, "body" text NOT NULL, "publish" datetime NOT NULL, "created"
datetime NOT NULL, "updated" datetime NOT NULL, "status" varchar(10)
NOT NULL, "author_id" integer NOT NULL REFERENCES "auth_user" ("id")
DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "blog_post_slug_b95473f2" ON "blog_post" ("slug");
CREATE INDEX "blog_post_author_id_dd7a8485" ON "blog_post" ("author_id");
COMMIT;
```

Tačan izlaz zavisi od baze podataka koju koristite. Prethodni izlaz je generisan za SQLite. Kao što možete videti u izlazu, Django generiše nazive tabela kombinovanjem naziva aplikacije i naziva modela ispisanog malim slovima (`blog_post`), ali možete odrediti i prilagođeni naziv baze podataka za vaš model u `Meta` klasi modela, tako što ćete koristiti atribut `db_table`.

Django automatski kreira primarni ključ za svaki model, ali možete ga i izmeniti, tako što ćete navesti `primary_key=True` u jednom od polja vašeg modela. Podrazumevani primarni ključ je kolona `id`, koja se sastoji od celog broja, koji se automatski inkrementira. Ova kolona odgovara polju `id` koje se automatski dodaje vašim modelima.

Sinhronizujte vašu bazu podataka sa novim modelom. Za primenu postojećih migracija pokrenite sledeću komandu:

```
python manage.py migrate
```

Dobićete izlaz koji se završava sledećom linijom:

```
Applying blog.0001_initial... OK
```

Upravo ste primenili migracije za aplikacije koje su navedene u postavci `INSTALLED_APPS`, uključujući vašu aplikaciju `blog`. Nakon primene migracija, baza podataka odražava trenutni status vaših modela.

Ako uređujete datoteku `model.py` da biste dodali, uklonili ili promenili polja postojećih modela ili ako dodate nove modele, moraćete da kreirate novu migraciju pomoću komande `makemigrations`. Migracija će omogućiti Djangou da prati izmene modela. Zatim ćete morati da primenite tu migraciju pomoću komande `migrate` da biste bazu podataka uskladili sa vašim modelima.

KREIRANJE ADMINISTRATORSKOG SAJTA ZA MODELE

Pošto ste definisali model `Post`, kreiraćete jednostavan administratorski sajt za upravljanje postovima na blogu. Django se isporučuje sa ugrađenim interfejsom za administraciju koji je veoma koristan za uređivanje sadržaja. Django sajt se dinamički izrađuje čitanjem metapodataka vašeg modela i obezbeđivanjem interfejsa koji je namenjen za uređivanje sadržaja. Možete ga koristiti kao gotov proizvod, konfigurišući način na koji želite da se vaši modeli prikazuju.

Aplikacija `django.contrib.admin` je već uključena u postavku `INSTALLED_APPS`, tako da je ne morate dodavati.

Kreiranje superkorisnika

Prvo treba da kreirate korisnika koji će upravljati administratorskim sajtom. Pokrenite sledeću komandu:

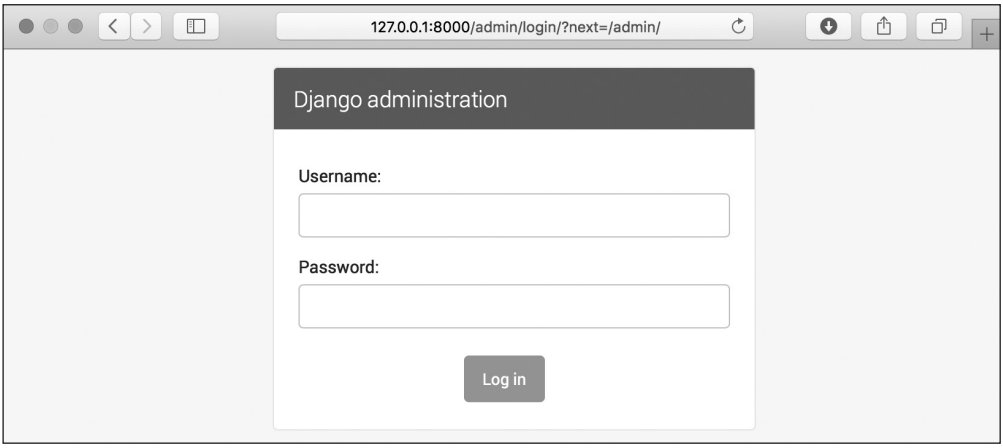
```
python manage.py createsuperuser
```

Videćete sledeći izlaz; unesite željeno korisničko ime, e-poštu i lozinku na sledeći način:

```
Username (leave blank to use 'admin'): admin
Email address: admin@admin.com
Password: ********
Password (again): ********
Superuser created successfully.
```

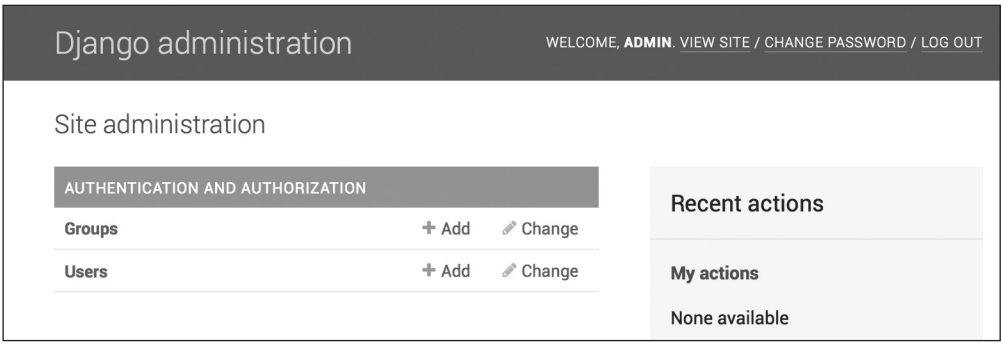
Django administratorski sajt

Sada pokrenite razvojni server pomoću komande `python manage.py runserver` i otvorite stranicu `http://127.0.0.1:8000/admin/` u vašem pretraživaču. Trebalo bi da vidite stranicu za prijavljivanje administratora, kao što je prikazano na sledećem snimku ekrana.



Slika 1.3 Ekran stranice za prijavljivanje administratora

Prijavite se, koristeći akreditive korisnika koji ste kreirali u prethodnom koraku. Videćete indeksnu stranicu administratorskog sajta, kao što je prikazano na sledećem snimku ekrana.



Slika 1.4 Indeksna stranica Django administratorskog sajta

Modeli `Group` i `User` koje možete videti na prethodnom snimku ekrana su deo Django radnog okvira za autentifikaciju koji se nalazi u aplikaciji `django.contrib.auth`. Ako kliknete na stavku **Users**, videćete prethodno kreiranog korisnika.

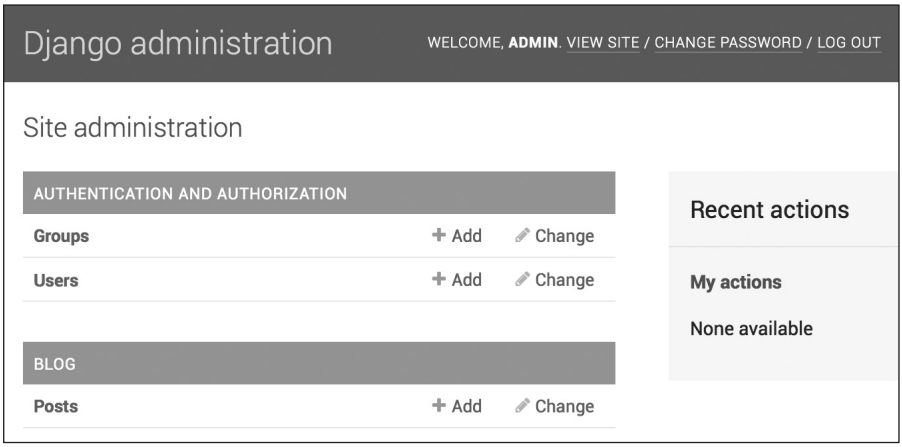
Dodavanje modela administratorskom sajtu

Sada ćete dodati vaše modele bloga na administratorski sajt. Uredite datoteku `admin.py` aplikacije `blog` na sledeći način:

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

admin.site.register(Post)
```

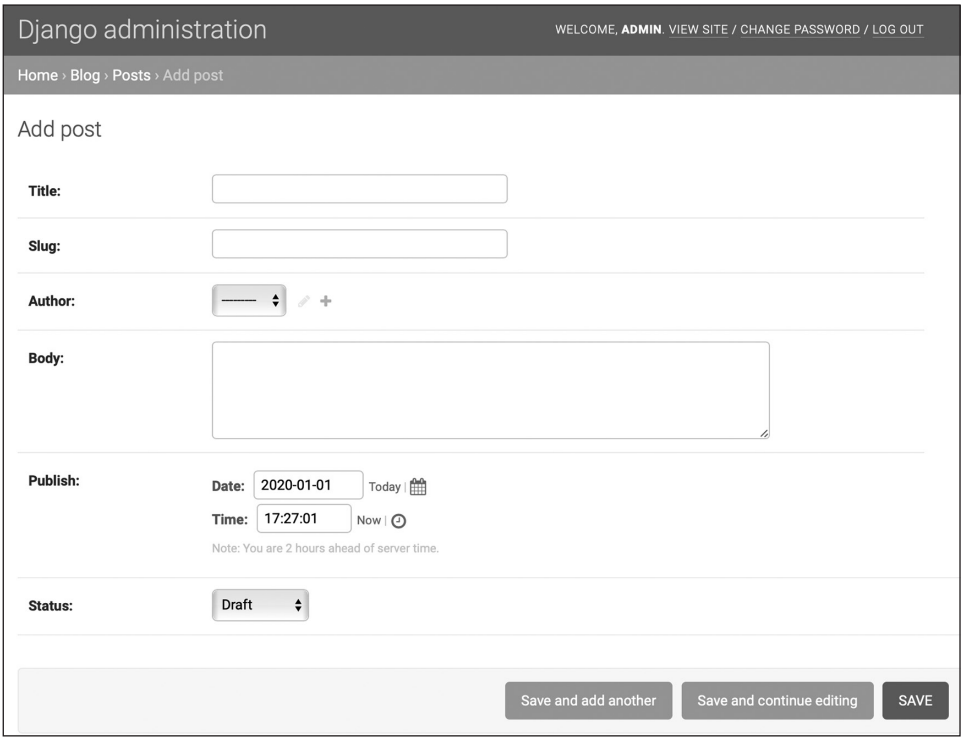
Ponovo učitajte administratorski sajt u svom pregledaču. Vaš model `Post` bi trebalo da bude prikazan na sajtu kao na sledećoj slici.



Slika 1.5 Model Post aplikacije blog uključen u indeksnu stranicu Django sajta

Ovo je bilo lako, zar ne? Kada registrujete model na Django administratorskom sajtu, dobijate korisnički interfejs koji je generisan introspekcijom vaših modela i koji omogućava da na jednostavan način pregledate, uređujete, kreirate i brišete objekte.

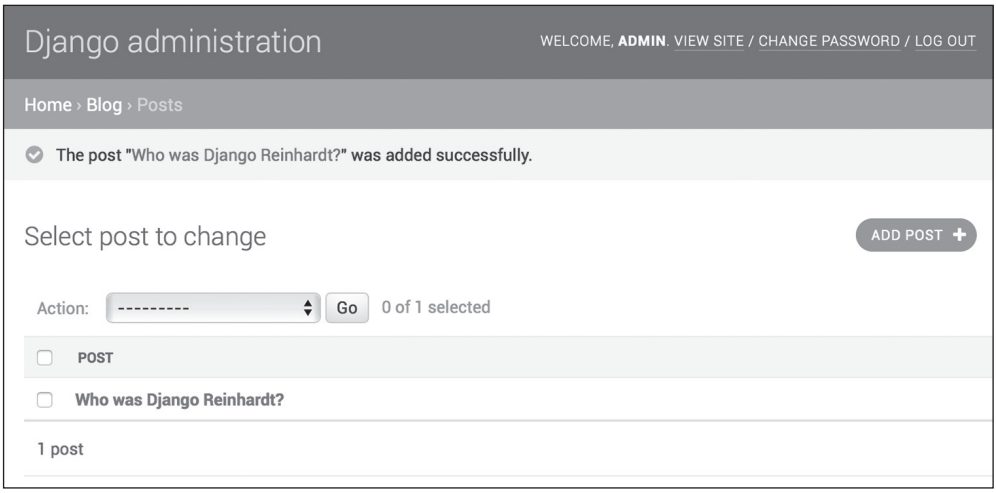
Kliknite na link **Add** pored stavke **Posts** da biste dodali novi post. Primetićete obrazac koji je Django dinamički generisao za vaš model, kao što je prikazano na sledećem snimku ekrana.



Slika 1.6 Django obrazac za uređivanje administratorskog sajta za model Post

Django koristi različite vidžete za obrasce za svaki tip polja. Čak se i složena polja, kao što su `DateTimeField` polja, prikazuju pomoću jednostavnog interfejsa, kao što je JavaScript data picker.

Popunite obrazac i kliknite na dugme **SAVE**. Trebalo bi da budete preusmereni na stranicu liste postova sa uspešno dodatom porukom i na post koji ste upravo kreirali, kao što je prikazano na sledećem snimku ekrana.



Slika 1.7 Prikaz liste Django administratorskog sajta za model Post sa uspešno dodatom porukom

Prilagođavanje načina za prikazivanja modela

Sada ćemo pogledati kako se prilagođava administratorski sajt. Uredite datoteku `admin.py` aplikacije `blog` i izmenite je na sledeći način:

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

@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'slug', 'author', 'publish', 'status')
```

Ukazaćete Django administratorskom sajtu da je vaš model registrovan na sajtu pomoću prilagođene klase koja nasleđuje klasu `ModelAdmin`. U ovu klasu možete uključiti informacije o načinu kako prikazati model na sajtu i kako komunicirati sa tim modelom.

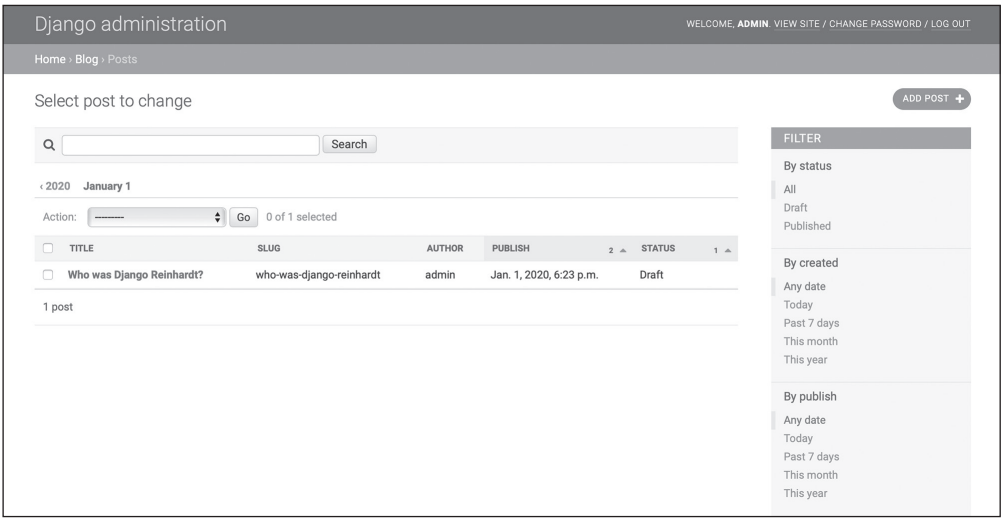
Atribut `list_display` omogućava da postavite polja vašeg modela koja želite da prikažete na administratorskoj stranici liste objekata. Dekorator `@admin.register()` izvršava istu funkciju kao i funkcija `admin.site.register()` koju ste zamenili, tako što ste registrovali klasu `ModelAdmin` koju taj dekorator ukrašava.

Sada ćemo prilagoditi model `admin` sa još nekim opcijama pomoću sledećeg koda:

```
@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'slug', 'author', 'publish', 'status')
    list_filter = ('status', 'created', 'publish', 'author')
```

```
    search_fields = ('title', 'body')
    prepopulated_fields = {'slug': ('title',)}
    raw_id_fields = ('author',)
    date_hierarchy = 'publish'
    ordering = ('status', 'publish')
```

Vratite se na pregledač i ponovo učitajte stranicu liste postova. Sada će stranica lista postova izgledati kao na sledećoj slici.



Slika 1.8 Prilagođeni prikaz liste Django administratorskog sajta za model Post

Možete videti da su polja prikazana na stranici liste postova ona polja koja ste naveli u atributu `list_display`. Stranica liste sada uključuje desnu bočnu traku koja omogućava filtriranje rezultata prema poljima koja su uključena u atribut `list_filter`.

Traka za pretraživanje pojavila se na stranici liste, zato što ste definisali listu polja za pretraživanje pomoću atributa `search_fields`. Odmah ispod trake za pretraživanje nalaze se linkovi za navigaciju za kretanje kroz hijerarhiju datuma; ovo je definisano pomoću atributa `date_hierarchy`. Takođe možete videti da su postovi raspoređeni prema kolonama **`STATUS`** i **`PUBLISH`**. Odredili ste podrazumevane kriterijume sortiranja pomoću atributa `ordering`.

Zatim, kliknite na link **ADD POST**. Ovde ćete takođe primetiti neke promene. Dok kucate naslov novog posta, polje `slug` se automatski popunjava. Ukazali ste Djangou da unapred popuni polje `slug`, tako što ste uneli polje `title`, koristeći atribut `prepopulated_fields`.

Polje `author` sada se prikazuje sa vidžetom za odabir koji može biti mnogo bolji od padajućeg menija za odabir unosa kada imate hiljade korisnika. Ovo se postiže atributom `raw_id_fields` i izgleda kao na sledećoj slici.

Slika 1.9 Vidžet za odabir srodnih objekata za polje author modela Post

Pomoću nekoliko linija koda prilagodili ste način na koji se vaš model prikazuje na administratorskom sajtu. Postoji mnogo načina na koje Django administratorski sajt može da se prilagodi i proširi; o tome ćete saznati više detalja kasnije u ovoj knjizi.

UPOTREBA QUERYSETOVA I MENADŽERA

Sada, kada imate potpuno funkcionalan administratorski sajt za upravljanje sadržajem vašeg bloga, vreme je da naučite kako možete da preuzmete informacije iz baze podataka i kako da komunicirate sa njom. Django se isporučuje sa moćnim API-em za apstrakciju baze podataka, koji omogućava da lako kreirate, preuzimate, ažurirate i brišete objekte. Django **objektno-relativni maper (ORM)** kompatibilan je sa sistemima za upravljanje bazama podataka MySQL, PostgreSQL, SQLite, Oracle i MariaDB. Imajte na umu da bazu podataka vašeg projekta možete da definišete u postavci `DATABASES` u datoteci vašeg projekta `settings.py`. Django može da koristi više baza podataka odjednom, a možete i da programirate rutere za baze podataka da biste kreirali prilagođene šeme usmeravanja.

Nakon što kreirate modele podataka, Django vam obezbeđuje besplatan API za interakciju sa njima. Referencu modela podataka zvanične dokumentacije možete pronaći na adresi https://docs.djangoproject.com/en/3.0/ref/models/.

Django ORM zasnovan je na QuerySetovima. QuerySet je kolekcija upita baze podataka za preuzimanje objekata iz baze podataka. Možete primeniti filtere na QuerySetovima da biste odredili upite na osnovu zadatih parametara.

Kreiranje objekata

Otvorite terminal i pokrenite sledeću komandu da biste otvorili Python komandno okruženje:

```
python manage.py shell
```

Zatim, unesite sledeće linije:

```
>>> from django.contrib.auth.models import User
>>> from blog.models import Post
>>> user = User.objects.get(username='admin')
>>> post = Post(title='Another post',
...             slug='another-post',
...             body='Post body.',
...             author=user)
>>> post.save()
```

Sada ćemo analizirati šta ovaj kod radi. Prvo preuzmite korisnički objekat pomoću korisničkog imena `admin`:

```
user = User.objects.get(username='admin')
```

Metod `get()` omogućava preuzimanje jednog objekta iz baze podataka. Imajte na umu da ovaj metod očekuje rezultat koji odgovara upitu. Ako baza podataka ne vrati rezultate, ovaj metod će generisati izuzetak `DoesNotExist`, a ako baza podataka vrati više od jednog rezultata, biće generisan izuzetak `MultipleObjectsReturned`. Oba izuzetka su atributi klase modela u kojoj se izvršava upit.

Zatim, kreirajte instancu `Post` sa prilagođenim naslovom, URL-om i „telom“ i postavite korisnika kojeg ste prethodno preuzeli kao autora posta:

```
post = Post(title='Another post', slug='another-post', body='Post
body.', author=user)
```

Ovaj objekat se nalazi u memoriji i ne zadržava se u bazi podataka.

Na kraju, objekat `Post` ukladištite u bazu podataka pomoću metoda `save()`:

```
post.save()
```

Prethodna radnja izvršava SQL iskaz `INSERT` u pozadini. Prvo ste videli kako se kreira objekat u memoriji, a zatim kako da ga zadržite u bazi podataka, ali takođe možete da kreirate objekat i da ga unesete u bazu podataka u samo jednoj operaciji pomoću metoda `create()` na sledeći način:

```
Post.objects.create(title='One more post',
                    slug='one-more-post',
                    body='Post body.',
                    author=user)
```

Ažuriranje objekata

Sada promenite naslov posta u nešto drugačije i ponovo sačuvajte objekat:

```
>>> post.title = 'New title'
>>> post.save()
```

Ovoga puta metod `save()` izvršava SQL iskaz `UPDATE`.

Promene koje unesete u objekat ne zadržavaju se u bazi podataka sve dok ne pozovete metod `save()`.

Preuzimanje objekata

Već znate kako da preuzmete jedan objekat iz baze podataka pomoću metoda `get()`. Pristupili ste ovom metodu koristeći metod `Post.objects.get()`. Svaki Django model ima najmanje jednog menadžera, a podrazumevani menadžer zove se `objects`. Preuzmite objekat `QuerySet` pomoću menadžera modela. Da biste preuzeli sve objekte iz tabele, samo koristite metod `all()` u podrazumevanom menadžeru objekata na sledeći način:

```
>>> all_posts = Post.objects.all()
```

Ovako kreirate QuerySet koji vraća sve objekte u bazu podataka. Imajte na umu da ovaj QuerySet još nije izvršen. Django QuerySetovi su „lenji“, što znači da se procenjuju samo kada su na to „prisiljeni“. Ovo ponašanje čini QuerySetove vrlo efikasnim. Ako ne postavite QuerySet na promenljivu, već ga, umesto toga, upišete direktno u komandno okruženje Pythona, SQL iskaz QuerySeta se izvršava zato što ste ga „prisilili“ da ispiše rezultate:

```
>>> all_posts
```

Upotreba metoda filter()

Da biste filtrirali QuerySet, možete koristiti metod menadžera `filter()`. Na primer, možete da preuzmete sve postove objavljene ove godine pomoću sledećeg QuerySeta:

```
>>> Post.objects.filter(publish__year=2020)
```

Takođe možete da filtrirate više polja. Na primer, možete da preuzmete sve postove koje je autor objavio ove godine sa korisničkim imenom `admin`:

```
>>> Post.objects.filter(publish__year=2020, author__username='admin')
```

Ovo je isto kao izrada istog QuerySeta koji ulančava više filtera:

```
>>> Post.objects.filter(publish__year=2020) \
>>>             .filter(author__username='admin')
```

Upiti sa metodima pretraživanja polja kreiraju se pomoću dve donje crte - na primer, `publish__year`, ali isti zapis se koristi i za pristup poljima srodnih modela, kao što je `autor__username`.

Upotreba metoda exclude()

Možete izuzeti određene rezultate iz QuerySeta, koristeći metod menadžera `exclude()`. Na primer, možete da preuzmete sve postove objavljene ove godine čiji naslovi ne počinju rečju `Why`:

```
>>> Post.objects.filter(publish__year=2020) \
>>>             .exclude(title__startswith='Why')
```

Upotreba metoda order_by()

Rezultate možete da rasporedite prema različitim poljima pomoću metoda menadžera `order_by()`. Na primer, možete da preuzmete sve objekte koje su poređani prema naslovu na sledeći način:

```
>>> Post.objects.order_by('title')
```

Rastući redosled se podrazumeva. Opadajući redosled možete označiti prefiksom negativnog znaka na sledeći način:

```
>>> Post.objects.order_by('-title')
```

Brisanje objekata

Ako želite da izbrišete objekat, to možete učiniti iz instance objekta, koristeći metod `delete()`:

```
>>> post = Post.objects.get(id=1)
>>> post.delete()
```

Imajte na umu da će uz objekte biti izbrisane i sve zavisne veze za objekte ForeignKey definisane pomoću upita on_delete koji je postavljen na vrednost CASCADE.

ProcenaQuerySetova

Kreiranje QuerySeta ne uključuje aktivnosti baze podataka dok se on ne proceni. QuerySetovi obično vraćaju drugi QuerySet koji nije procenjen. Možete spojiti onoliko filtera koliko želite u QuerySetu i neće biti pogođena baza podataka, sve dok se QuerySet ne proceni. Kada se proceni QuerySet, on se pretvara u SQL upit u bazu podataka.

QuerySetovi se procenjuju samo u sledećim slučajevima:

- kada ih prvi put ponovite
- kada ih „isečete" - na primer, `Post.objects.all()[:3]`
- kada ih serijalizujete ili keširate
- kada pozovete metod `repr()` ili `len()` u njima
- kada izričito pozovete metod `list()` u njima
- kada ih testirate u iskazu, kao što su `bool()`, `or`, `and` ili `if`

Kreiranje menadžera modela

Kao što je ranije napomenuto, `objects` je podrazumevani menadžer svakog modela koji preuzima sve objekte iz baze podataka. Međutim, takođe možete definisati prilagođene menadžere za svoje modele. Kreiraćete prilagođeni menadžer za preuzimanje svih postova koji imaju status `published`.

Postoje dva načina za dodavanje ili prilagođavanje menadžera za vaše modele: možete da dodate još metoda menadžera postojećem menadžeru ili da kreirate novi menadžer modifikovanjem početnog QuerySeta koji menadžer vraća. Prvi metod obezbeđuje QuerySet API, kao što je `Post.objects.my_manager()`, a drugi metod obezbeđuje `Post.my_manager.all()`. Menadžer će vam omogućiti da preuzmete postove koristeći `Post.published.all()`.

Uredite datoteku `models.py` aplikacije `blog` da biste dodali prilagođeni menadžer:

```
class PublishedManager(models.Manager):
    def get_queryset(self):
        return super(PublishedManager,
                     self).get_queryset()\
                          .filter(status='published')

class Post(models.Model):
    # ...
    objects = models.Manager() # The default manager.
    published = PublishedManager() # Our custom manager.
```

Prvi menadžer koji je deklarisan u modelu postaje podrazumevani menadžer. Možete da koristitie `Meta` atribut `deafult_manager_name` da biste odredili različiti podrazumevani menadžer. Ako ni jedan menadžer nije definisan u modelu, Django automatski kreira podrazumevani menadžer `objects`. Ako deklarišete menadžere za vaš model, ali želite da zadržite i menadžer `objects`, morate da ga eksplicitno dodate vašem modelu. U prethodnom kodu dodati su podrazumevani menadžer `objects` i prilagođeni menadžer `published` za model `Post`.

Metod `get_queryset()` menadžera vraća QuerySet koji će biti izvršen. Vi ćete zameniti ovaj metod da biste dodali vaš prilagođeni filter u konačni QuerySet.

Pošto ste definisali svoj prilagođeni menadžer i dodali ga modelu `Post`; možete ga koristiti za izvršavanje upita. Sada ćemo ovo testirati.

Ponovo pokrenite razvojni server pomoću sledeće komande:

```
python manage.py shell
```

Sada možete da uvezete model `Post` i da preuzmete sve objavljene postove čiji naslov počinje rečju `Who`, tako što ćete izvršiti sledeći QuerySet:

```
>>> from blog.models import Post
>>> Post.published.filter(title__startswith='Who')
```

Da biste dobili rezultate za ovaj QuerySet, obavezno postavite polje `published` na vrednost `True` u objektu `Post` čiji naslov (`title`) počinje rečju `Who`.

IZRADA LISTE I PRIKAZI DETALJA

Pošto sada znate kako se koristi ORM, spremni ste za izradu prikaza (views) aplikacije `blog`. Django view je samo Python funkcija koja prima zahtev i vraća odgovor. Sva logika za povratak željenog odgovora se smešta unutar prikaza.

Prvo ćete kreirati prikaze za svoju aplikaciju, onda ćete definisati uzorak URL-a za svaki prikaz, a na kraju ćete kreirati HTML šablone za prikazivanje podataka koje generišu prikazi. Svaki prikaz će renderovati šablon, tako što će mu proslediti promenljive i vratiti HTTP odgovor sa renderovanim izlazom.

Kreiranje liste i prikaza detalja

Prvo treba kreirati view za prikaz liste postova. Uredite datoteku `views.py` aplikacije `blog` da bi izgledala ovako:

```
from django.shortcuts import render, get_object_or_404
from .models import Post

def post_list(request):
    posts = Post.published.all()
    return render(request,
                  'blog/post/list.html',
                  {'posts': posts})
```

Upravo ste kreirali vaš prvi Django prikaz. Prikaz `post_list` prihvata objekat `request` kao jedini parametar. Taj parametar zahtevaju svi prikazi. U ovom prikazu preuzimate sve postove sa statusom `published`, tako što ćete koristiti menadžer `published` koji ste prethodno kreirali.

Konačno, koristite prečicu `render()` koju je obezbedio Django da biste prikazali listu postova sa zadatim šablonom. Ova funkcija preuzima objekat `request`, putanju šablona i kontekstne promenljive da bi bio renderovan zadati šablon. Vraća objekat `HttpResponse` sa prikazanim tekstom (obično sa HTML kodom). Prečica `render()` prihvata kontekst zahteva, tako da je bilo koja promenljiva postavljena pomoću procesora kontekstnog šablona dostupna u zadatom šablonu. Procesori kontekstnog šablona su samo callable tip funkcije koje postavljaju promenljive u kontekst. Saznaćete u Poglavlju 3, *„Proširenje aplikacije zablog“*, kako se koriste ovi procesori.

Sada ćemo kreirati drugi prikaz (view) da bismo prikazali jedan post. Dodajte sledeću funkciju u datoteku `views.py`:

```
def post_detail(request, year, month, day, post):
    post = get_object_or_404(Post, slug=post,
                                   status='published',
                                   publish__year=year,
                                   publish__month=month,
                                   publish__day=day)
    return render(request,
                  'blog/post/detail.html',
                  {'post': post})
```

Ovo je prikaz detalja posta. On prihvata argumente `year`, `month`, `day` i `post` za preuzimanje objavljenog posta sa konkretnim poljima slug i date. Imajte na umu da ste prilikom kreiranja modela `Post` dodali `unique_for_date parameter` u polje `slug`. Ovo osigurava da će za određeni datum postojati samo jedan post sa poljem slug, pa možete preuzeti pojedinačne postove, koristeći polja date i slug. U prikazu detalja koristite prečicu `get_object_or_404()` da biste preuzeli željeni post. Ova funkcija preuzima objekat koji odgovara konkretnim parametrima ili prikazuje izuzetak HTTP 404 (not found) ako nije pronađen ni jedan objekat. Na kraju, koristite prečicu `render()` da biste renderovali preuzeti post pomoću šablona.

Dodavanje uzoraka URL-a za prikaze

Uzorci URL-a omogućavaju da mapirate URL-ove u prikaze. Uzorak URL-a sastoji se od obrasca znakovnog niza, od prikaza i, opciono, od naziva koji dodeljujete URL-u u celom projektu. Django prolazi kroz svaki uzorak URL-a i zaustavlja se na prvom uzorku koji odgovara traženom URL-u. Zatim, uvozi prikaz odgovarajućeg obrasca URL-a i izvršava ga, prosleđujući instancu klase `HttpRequest` i ključne reči ili pozicione argumente.

Kreirajte datoteku `urls.py` u direktorijumu aplikacije `blog` i dodajte joj sledeće linije:

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

app_name = 'blog'

urlpatterns = [
    # post views
    path('', views.post_list, name='post_list'),
    path('<int:year>/<int:month>/<int:day>/<slug:post>/',
         views.post_detail,
         name='post_detail'),
]
```

U prethodnom kodu definišite imenski prostor aplikacije pomoću promenljive `app_name`. To vam omogućava da organizujete URL-ove prema aplikaciji i da koristite njhove nazive kada ih referencirate. Dva različita obrasca definišete pomoću funkcije `path()`. Prvi uzorak URL-a ne prihvata argumente i mapiran je u prikazu `post_list`. Drugi uzorak prihvata sledeća četiri argumenta i mapiran je u prikazu `post_detail`:

- `year` - Potreban je ceo broj.
- `month` - Potreban je ceo broj.
- `day` – Potreban je ceo broj.
- `post` - Može se sastojati od reči i crtica.

Koristite uglaste zagrade za snimanje vrednosti iz URL-a. Svaka vrednost navedena u uzorku URL-a kao `<parameter>` snima se kao znakovni niz. Koristite pretvarače putanje (path converters), kao što su `<int:year>` (da biste posebno uparili i vratili ceo broj) i `<slug:post>` (da biste posebno uparili slug). Možete da vidite sve pretvarače putanje koje obezbeđuje Django na adresi https://docs.djangoproject.com/en/3.0/topics/http/urls/#pathconverters.

Ako vam upotreba metoda `path()` i pretvarača nije dovoljna, umesto njih možete da upotrebite metod `re_path()` za definisanje složenih uzoraka URL-a pomoću Python regularnih izraza. Možete saznati više detalja o definisanju uzoraka URL-a pomoću regularnh izraza na adresi https://docs.djangoproject.com/en/3.0/ref/urls/#django.urls.re_path. Ako ranije niste koristili regularne izraze, možda prvo treba da pogledate članak „Regular Expression HOWTO“ na adresi https://docs.python.org/3/howto/regex.html.

Kreiranje datoteke `urls.py` za svaku aplikaciju je najbolji način da se vaše aplikacije ponovo upotrebe u drugim projektima.

Zatim, treba da uključite uzorke URL-a aplikacije `blog` u glavne uzorke URL-a projekta.

Uredite datoteku `urls.py` koja se nalazi u direktorijumu `mysite` vašeg projekta na sledeći način:

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

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

Novi obrazac URL-a koji je definisan pomoću iskaza `include` odnosi se na uzorke URL-a definisane u aplikaciji `blog`, tako da budu uključeni u putanju `blog/`. Ovi uzorci uključuju imenski prostor `blog`. Imenski prostori moraju biti jedinstveni u celom vašem projektu. Kasnije ćete lako referencirati URL svojih blogova, koristeći imenski prostor praćen dvotačkom i nazivom URL-a - na primer, `blog:post_list` i `blog:post_detail`. Možete saznati više detalja o imenskim prostorima URL-a na adresi https://docs.djangoproject.com/en/3.0/topics/http/urls/#url-namespaces.

Kanonski URL-ovi za modele

Kanonski URL je preferirani URL za resurs. Na vašem sajtu možda postoje različite stranice na kojima se prikazuju postovi, ali postoji jedan URL koji koristite kao glavni za post na blogu. Konvencija u Djangou je da se modelu `get_absolute_url()` doda model koji vraća kanonski URL za objekat.

Možete koristiti URL `post_detail` koji ste definisali u prethodnom odeljku za izradu kanonskog URL-a za objekte `Post`. Za ovaj metod koristićete metod `reverse()`, koji omogućava da kreirate URL-ove prema njihovom nazivu, a zatim da prosledite opcione parametre. Možete saznati više detalja o uslužnim funkcijama URL-ova na adresi https://docs.djangoproject.com/en/3.0/ref/urlresolvers/.

Uredite datoteku `models.py` aplikacije `blog` i dodajte sledeći kod:

```
from django.urls import reverse

class Post(models.Model):
    # ...
    def get_absolute_url(self):
        return reverse('blog:post_detail',
                       args=[self.publish.year,
                             self.publish.month,
                             self.publish.day, self.slug])
```

Koristite metod `get_absolute_url()` u svojim šablonima da biste se povezali sa određenim postovima.

IZRADA ŠABLONA ZA PRIKAZE

Kreirali ste prikaze i uzorke URL-a za aplikaciju `blog`. Uzorci URL-a mapiraju URL-ove u prikaze i prikazi „odlučuju“ koji podaci će biti vraćeni korisniku. Šabloni definišu način na koji će podaci biti prikazani; obično se podaci pišu u HTML-u u kombinaciji sa Django jezikom šablona. Više informacija o Django jeziku šablona možete pronaći na adresi https://docs.djangoproject.com/en/3.0/ref/templates/language/.

Sada ćete u svoju aplikaciju dodati šablone za prikazivanje postova na jednostavan način.

Kreirajte sledeće direktorijume i datoteke u direktorijumu aplikacije `blog`:

```
templates/
    blog/
        base.html
        post/
            list.html
            detail.html
```

Prethodna struktura će biti struktura datoteka za vaše šablone. Datoteka `base.html` sadrži glavnu HTML strukturu veb sajta i deli sadržaj na oblast glavnog sadržaja i na bočnu traku. Datoteke `list.html` i `detail.html` nasledиće datoteku `base.html` za prikaz liste postova i prikaze detalja (tim redosledom).

Django ima moćan jezik šablona koji omogućava da odredite način za prikazivanje podataka. Ovaj jezik se zasniva na *oznakama šablona*, *promenljivim šablona* i *filterima šablona*:

- Oznake šablona kontrolišu prikazivanje šablona i izgledaju kao `{% tag%}`.
- Promenljive šablona se zamenjuju vrednostima kada je šablon prikazan i izgledaju kao `{{variable}}`.
- Filteri šablona omogućavaju da modifikujete promenljive za prikaz i izgledaju kao `{{variable|filter}}`.

Sve ugrađene oznake šablona i filtera možete videti na adresi https://docs.djangoproject.com/en/3.0/ref/templates/builtins/.

Uredite datoteku `base.html` i dodajte sledeći kod:

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}{% endblock %}</title>
  <link href="{% static "css/blog.css" %}" rel="stylesheet">
</head>
<body>
  <div id="content">
    {% block content %}
    {% endblock %}
  </div>
  <div id="sidebar">
    <h2>My blog</h2>
    <p>This is my blog.</p>
  </div>
</body>
</html>
```

Oznaka `{% loadstatic%}` ukazuje Djangou da treba da učita oznake statičkog šablona koje obezbeđuje aplikacija `django.contrib.staticfiles`, koja se nalazi u postavci `INSTALLED_APPS`. Nakon što su učitani, moći ćete da koristite oznaku šablona `{% static%}` u ovom šablonu. Pomoću ove oznake šablona možete uključiti statičke datoteke, kao što su datoteke `blog.css`, koje ćete naći u kodu ovog primera u direktorijumu `static/`. Kopirajte direktorijum `static/` iz koda u ovom poglavlju na istu lokaciju na kojoj i vaš projekat za primenu CSS stilova na šablone. Sadržaj direktorijuma možete pronaći na adresi https://github.com/PacktPublishing/Django-3-by-Example/tree/master/Chapter01/mysite/blog/static.

Postoje dve oznake `{% block%}`. One ukazuju Djangou da želite da definišete blok u toj oblasti. Šabloni koji naslede ovaj obrazac mogu da popune blokove sadržajem. Definisali ste blok koji se zove `title` i blok koji se zove `content`.

Sada ćemo urediti datoteku `post/list.html` da bi izgledala ovako:

```
{% extends "blog/base.html" %}

{% block title %}My Blog{% endblock %}

{% block content %}
  <h1>My Blog</h1>
  {% for post in posts %}
    <h2>
      <a href="{{ post.get_absolute_url }}">
        {{ post.title }}
      </a>
    </h2>
    <p class="date">
      Published {{ post.publish }} by {{ post.author }}
    </p>
    {{ post.body|truncatewords:30|linebreaks }}
  {% endfor %}
{% endblock %}
```

Pomoću oznake šablona `{% extends%}` ukazujete Djangou da nasledi od šablona `blog/base.html`. Zatim, popunjavate sadržajem blokove `title` i `content` osnovnog šablona. Pregledajte komentare i prikažite njihov naslov, datum, ime autora i „telo“, uključujući i link u naslovu za kanonski URL-a posta.

U „telu“ posta primenjujete dva filtera šablona: `truncatewords` skraćuje vrednost na određeni broj reči, a `linebreaks` pretvara izlaz u HTML prekid reda. Možete spojiti onoliko filtera šablona koliko želite; svaki od njih će biti primenjen na izlaz koji je generisao prethodni filter šablona.

Otvorite komandno okruženje i izvršite komandu `python manage.py runserver` da biste pokrenuli razvojni server. Otvorite stranicu `http://127.0.0.1:8000/blog/` u svom pregledaču i videćete da sve dobro funkcioniše. Imajte na umu da morate da imate neke postove sa statusom `Published` da biste ih prikazali na toj stranici. Trebalo bi da vidite nešto kao na sledećoj slici.

Slika 1.10 Stranica za prikaz liste poruka

Uredite datoteku `post/detail.html`:

```
{% extends "blog/base.html" %}

{% block title %}{{ post.title }}{% endblock %}

{% block content %}
  <h1>{{ post.title }}</h1>
  <p class="date">
    Published {{ post.publish }} by {{ post.author }}
  </p>
  {{ post.body|linebreaks }}
{% endblock %}
```

Zatim se možete vratiti u pregledač i kliknuti na jedan od naslova posta da biste pogledali prikaz detalja posta. Trebalo bi da vidite nešto ovako:

Slika 1.11 Stranica za prikaz detalja posta

Pogledajte URL koji bi trebalo da izgleda ovako `/blog/2020/1/1/who-was--djangoreinhardt/`. Dizajnirali ste SEO-friendlyURL-ove za vaše blogove.

DODAVANJE NUMERISANJA STRANICA

Kada počnete da dodajete sadržaj na svoj blog, lako možete doći do tačke u kojoj se u vašoj bazi podataka čuvaju desetine ili stotine postova. Umesto da prikažete sve postove na jednoj stranici, možda ćete želeti da podelite listu postova na više stranica. To se može postići numerisanjem stranica. Možete da definišete broj postova koje želite da budu prikazani po stranici i da pronađete postove koji odgovaraju stranici koju korisnik traži. Django ima ugrađenu klasu pagination koja omogućava lako upravljanje numerisanim podacima.

Uredite datoteku `views.py` aplikacije `blog` da biste uvezli klase Django paginatora i modifikovali prikaz `post_list` na sledeći način:

```
from django.core.paginator import Paginator, EmptyPage,\
                                  PageNotAnInteger

def post_list(request):
    object_list = Post.published.all()
    paginator = Paginator(object_list, 3) # 3 posts in each page
    page = request.GET.get('page')
    try:
        posts = paginator.page(page)
    except PageNotAnInteger:
        # If page is not an integer deliver the first page
        posts = paginator.page(1)
    except EmptyPage:
        # If page is out of range deliver last page of results
        posts = paginator.page(paginator.num_pages)
    return render(request,
                  'blog/post/list.html',
                   {'page': page,
                    'posts': posts})
```

Ovako funkcioniše numerisanje stranica:

1. Klasu `Paginator` instancirate pomoću broja objekata za koje želite da budu prikazani na svakoj stranici.
2. Dobijate parametar `page GET`, koji pokazuje trenutni broj stranice.
3. Objekte za željenu stranicu dobijate pozivanjem metoda `page()` klase `Paginator`.

4. Ako parametar stranice nije ceo broj, preuzimate prvu stranicu rezultata. Ako je ovaj parametar broj veći od broja poslednje stranice rezultata, preuzimate poslednju stranicu.
5. Prosleđujete broj stranice i preuzimate objekte u šablon.

Sada morate da kreirate šablon za prikaz paginatora tako da ga možete uključiti u bilo koji šablon u kojem se koristi numerisanje stranica. U fascikli `templates/` aplikacije `blog` kreirajte novu datoteku i dodelite joj naziv `pagination.html`. Dodajte sledeći HTML kod u datoteku:

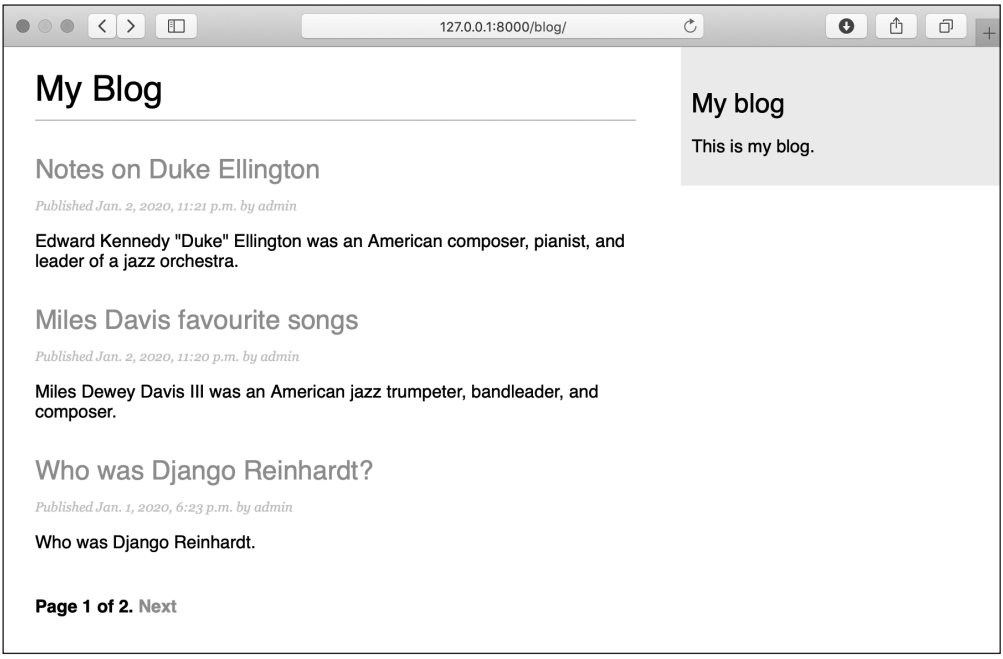
```
<div class="pagination">
  <span class="step-links">
    {% if page.has_previous %}
      <a href="?page={{ page.previous_page_number }}">Previous</a>
    {% endif %}
    <span class="current">
      Page {{ page.number }} of {{ page.paginator.num_pages }}.
    </span>
    {% if page.has_next %}
      <a href="?page={{ page.next_page_number }}">Next</a>
    {% endif %}
  </span>
</div>
```

Šablon za numerisanje stranica očekuje objekat `Page` da bi bili prikazani prethodni i sledeći linkovi, trenutna stranica i ukupan broj stranica rezultata. Sada ćemo se vratiti šablonu `blog/post/list.html` i uključićemo šablon `pagination.html` na dnu bloka `{ % content %}` na sledeći način:

```
{% block content %}
  ...
  {% include "pagination.html" with page=posts %}
{% endblock %}
```

Pošto se objekat `Page` koji prosleđujete šablonu zove `posts`, dodajete šablon za numerisanje stranica u šablon liste postova, prosleđujući parametre da biste svoj šablon ispravno prikazali. Možete da sledite ovaj metod da biste ponovo koristili svoj šablon stranice u numerisanim prikazima različitih modela.

Sada otvorite stranicu `http://127.0.0.1:8000/blog/` u svom pregledaču. Trebalo bi da se vidi stranica na dnu liste postova i trebalo bi da možete da se krećete kroz stranice.



Slika 1.12 Stranica sa listom postova, uključujući i numerisanje stranica

KORIŠĆENJE PRIKAZA ZASNOVANIH NA KLASAMA

Prikazi zasnovani na klasama predstavljaju alternativni način implementacije prikaza kao Python objekata umesto funkcija. Pošto je prikaz callable tip funkcije koja prihvata veb zahtev i vraća veb odgovor, svoje prikaze možete definisati i kao metode klase. Django za to obezbeđuje prikaze zasnovane na klasama. Svi ti prikazi nasleđuju od klase `View`, koja obrađuje otpremanje HTTP metoda i druge uobičajene funkcionalnosti.

Prikazi zasnovani na klasama za neke slučajeve korišćenja obezbeđuju prednosti u odnosu na prikaze koji su zasnovani na funkcijama. Oni imaju sledeće funkcije:

- organizovanje koda koji se odnosi na HTTP metode, kao što su `GET`, `POST` ili `PUT`, u zasebne metode, umesto korišćenja uslovnog grananja

- korišćenje višestrukog nasleđivanja za kreiranje klasa prikaza za višekratnu upotrebu (koje su poznate i kao *miksini*)

Uvod u prikaze zasnovane na klasama možete pogledati na adresi https://docs.djangoproject.com/en/3.0/topics/class-based-views/intro/.

Promenite svoj prikaz `post_list` u prikaz zasnovan na klasama da biste koristili generički prikaz `ListView` koji obezbeđuje Django. Ovaj osnovni prikaz omogućava da prikažete objekte bilo koje vrste.

Uredite datoteku `views.py` vaše aplikacije `blog` i dodajte sledeći kod:

```
from django.views.generic import ListView

class PostListView(ListView):
    queryset = Post.published.all()
    context_object_name = 'posts'
    paginate_by = 3
    template_name = 'blog/post/list.html'
```

Ovaj prikaz zasnovan na klasama je analogan prethodnom prikazu `post_list`. U prethodnom kodu prikazu `ListView` ukazujete da treba da uradi sledeće:

- da koristi određeni QuerySet, umesto da preuzme sve objekte. Umesto da definišete atribut `queryset`, mogli biste da odredite `model = Post` i Django bi automatski izradio generički QuerySet `Post.objects.all()`.
- da koristi kontekstnu promenljivu `posts` za rezultate upita. Podrazumevana promenljiva je `object_list` ako ne odredite ni jedan `context_object_name`.
- da numeriše rezultate, prikazujući tri objekta po stranici
- da koristi prilagođeni šablon za prikaz stranice. Ako ne postavite podrazumevani šablon, `ListView` će koristiti `blog/post_list.html`.

Sada otvorite datoteku `urls.py` vaše aplikacije `blog`, napišite komentar za prethodni uzorak URL-a `post_list` i dodajte novi uzorak URL-a, koristeći klasu `PostListView`, na sledeći način:

```
urlpatterns = [
    # post views
    # path('', views.post_list, name='post_list'),
    path('', views.PostListView.as_view(), name='post_list'),
    path('<int:year>/<int:month>/<int:day>/<slug:post>/',
        views.post_detail,
        name='post_detail'),
]
```

Da bi numerisanje stranica funkcionisalo, morate koristiti objekat desne stranice koji se prosleđuje šablonu. Djangov generički prikaz `ListView` prosleđuje izabranu stranicu promenljivoj `page_obj`, tako da morate da uredite svoj šablon `post/list.html` da biste uključili paginator, koristeći odgovarajuću promenljivu:

```
{% include "pagination.html" with page=page_obj %}
```

Otvorite stranicu `http://127.0.0.1:8000/blog/` u svom pregledaču i proverite da li sve funkcioniše na isti način kao u prethodnom prikazu `post_list`. Ovo je jednostavan primer prikaza zasnovanog na klasama koji koristi generičku klasu koju obezbeđuje Django. Saznaćete više detalja o prikazima zasnovanim na klasama u Poglavlju 10, „*Izrada platforme za elektronsko učenje*“, i u narednim poglavljima.

REZIME

U ovom poglavlju upoznali ste osnove Django veb radnog okvira, tako što ste kreirali jednostavnu aplikaciju za blog. Dizajnirali ste modele podataka i primenili migracije na svoj projekat. Takođe ste kreirali prikaze, šablone i URL-ove za svoj blog, uključujući i objekat numerisanja stranica.

U sledećem poglavlju otkrićete kako možete da poboljšate svoju aplikaciju za blog pomoću sistema za komentare i pomoću funkcija označavanja i kako da omogućite svojim korisnicima da dele postove e-poštom.