Razvoj web aplikacija pomoću okruženja Django

Ivezić, Nikolina

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Nikolina Ivezić

RAZVOJ WEB APLIKACIJA POMOĆU OKRUŽENJA DJANGO

Diplomski rad

Voditelj rada:
doc. dr. sc. Zvonimir Bujanović

Zagreb, rujan, 2015.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Web programeri su u ranim danima morali svaku web-stranicu pisati ručno. Ažuriranje tih web-stranica je značilo mijenjanje svake stranice posebno. Kako je broj web-stranica rastao postalo je jasno da je ažuriranje zahtjevno, dugotrajno i u konačnici neodrživo. Skupina poduzetničkih hakera u NCSA-i (The National Center for Supercomputing Applications) je riješila taj problem stvaranjem okruženja za izradu sučelja koje omogućava dinamičko generiranje HTML-a. Ovo okruženje su nazvali the Common Gateway Interface (CGI) i njime započinje prva generacija dinamičnih web-stranica. Međutim, i CGI je imao svoje mane, činio je ponovnu upotrebu kôda teškom, koristio je mnogo ponavljajućeg kôda i programerima koji ga prvi put koriste bio je teško razumljiv.

Druga generacija započela je razvojem PHP-a (Hypertext Preprocessor). PHP je riješio većinu mana koje je CGI imao i postao i dan danas najpopularniji alat za razvoj dinamičnih web-stranica. PHP-ov kôd se jednostavno ugrađuje u HTML i, za nekoga tko poznaje HTML, učenje PHP-a je veoma brzo. No i PHP ima svojih mana. S obzirom da je jednostavan za korištenje, često dolazi do ponavljajućeg i neurednog kôda. Nadalje, PHP malo brine o zaštiti programera od sigurnosnih propusta. Većina programera je naučila o sigurnosti podosta kasno, najčešće nakon napada na web-stranice.

Frustracije slične ovima su dovele do razvoja treće generacije dinamičnih web-stranica. Najpopularniji od njih su Django i Ruby on Rails. Oni dopuštaju razvoj dinamične i zanimljive Web stranice u kratkom vremenu, omogućuju prečace za česte zadatke (kao što su prijava korisnika ili unos podataka pomoću formi) i jasne konvencije kako riješiti sigurnosne probleme. Obzirom da slijede Model-Template-View arhitekturu prisiljavaju korisnika da koristi osnovne obrasce te arhitekture. Time se smanjuje ponavljajući kôd, ali i povećava sigurnost same stranice.

U ovom radu bit će pobliže objašnjeno što je Django, od čega se sastoji, kako se koristi te njegove prednosti i mane. Na jednostavnim primjerima bit će objašnjeno kako stvoriti funkcionalnu web-stranicu, najčešće greške te kako ih otkloniti. Rad će pratiti razvoj web-stranice koja služi za vođenje znanstvenih članaka na Prirodoslovno–matematičkom fakultetu.

Poglavlje 1

Uvod u Django

1.1 Povijest Djanga

The World Online udruženje, koje je odgovorno za proizvodnju i održavanje nekoliko lokalnih stranica s vijestima, napredovalo je u razvoju okruženja koje može zadovoljiti novinarske rokove. Novinari i upravljački tim su zahtijevali da se dijelovi i cijele stranice izgrade intenzivno brzo, često u samo nekoliko dana ili sati. Zbog toga su Simon Willison i Adrian Holovaty konstruirali web-razvojni okvir. To je bio jedini način da izrade aplikacije unutar tih ekstremnih rokova.

U ljeto 2005. godine kada su dovoljno razvili okruženje, tim koji je sad već uključivao i Jacob Kaplan-Moss-a, odlučio je objaviti okruženje kao otvoreni softver (eng. open source). Objavili su ga u srpnju 2005. godine i nazvali ga Django, prema poznatom jazz gitaristu Djangu Reinhardtu.

Django je dobro uspostavljen projekt s nekoliko stotina tisuća korisnika i suradnika diljem svijeta. Oba originalna programera još uvijek pružaju središnje smjernice za razvoj okruženja. Obzirom da je Django rođen u novinarskom okruženju posebno je dobro prilagođen za sadržajne stranice (eng. content sites) koje nude dinamičke informacije vođene bazom podataka. No, to ne znači da Django nije upotrebljiv za ostale vrste dinamičnih web-stranica.

Django je nastao iz stvarnog kôda. Dakle, nije nastao kao dio akademske vježbe ili komercijalni proizvod pa je fokusiran na rješavanje problema u web-okruženju s kojima su se susreli i sami programeri. Posljedica toga je Djangovo poboljšanje iz dana u dan. Cilj održavatelja Djanga je ušteda vremena, proizvodnja stranica koje je lako održavati i koje imaju dobre performanse pod opterećenjem. Ako ništa drugo, programeri su motivirani svojim vlastitim sebičnim željama da si skrate vrijeme i uživaju u svom poslu.

1.2 Arhitekture MVC i MTV

Model-Viewer-Controller (MVC) je arhitekturni obrazac koji služi za implementiranje korisničkog sučelja. On dijeli dani softver u tri dijela da bi razdvojio originalnu reprezentaciju informacija od načina na koji su informacije predstavljene korisniku. Obzirom da je MVC obrazac, određene arhitekture koje koriste taj obrazac mogu varirati. Prvo ćemo objasniti kako se ti dijelovi opisuju u tradicionalnoj definiciji. Centralni dio je model (eng. model). On zabilježava ponašanje aplikacije u terminima same domene problema koja je neovisna o korisničkom sučelju. Direktno pristupa podacima, logici i pravilima aplikacije. Drugi dio je pogled (eng. viewer). Pogled je svaki izlazni rezultat aplikacije kao što su dijagram, tablica ili tekst. Omogućeno je stvaranje više pogleda jedne informacije. Treći dio je kontrolor (eng. controller) koji prima ulaz i prevodi ga u izlaz razumljiv modelu ili pogledu. Kontrolor je zapravo posrednik između modela i pogleda u oba smjera.

Model-Template-View je vrsta MVC arhitekture korištena za razvoj web-stranica. On razdvaja različite dijelove web-stranice: prikaz, pristup podacima i logiku web-stranice. MTV omogućava neovisnu izgradnju web-stranica, povećava sigurnost sustava te pojednostavljuje održavanje sustava.

Model

Model (eng. model) definira oblike i odnose podataka u bazama podataka. Model u Django okruženju je klasa (eng. class) napisana u programskom jeziku Python, određuje varijable i metode pridružene određenim tipovima podataka te ima značenje tablice u bazi podataka. Pridružene varijable imaju značenje stupca u tablici, a metode definiraju relacije među varijablama. Model je usko povezan s bazom podataka i pogledom. Od baze podataka model dohvaća tražene podatke i prosljeđuje ih pogledu. Model nema saznanja o postojanju predložaka i funkcija izvedenih u pogledu. Na taj način je baza podataka izdvojena od preostala dva dijela sustava.

Pogled

Namjena pogleda (eng. view) je odrediti koji će podaci biti prikazani, odnosno, koji će podaci biti dohvaćeni iz baze podataka i prikazani pomoću pogleda u web-pregledniku. U Django okruženju, prilikom stvaranja web-stranice za svaku pojedinu aplikaciju kreira se zasebna datoteka pogleda. Datoteka pogleda sastoji se od funkcija napisanih u programskom jeziku Python. Za svaku stranicu napisana je posebna funkcija koja upravlja izvođenjem upita. Osim mogućnosti postavljanja upita modelu za dohvatom podataka, ima mogućnost implementacije slanja e-mailova, autentifikacije, provjere ulaznih parametara i mnoge druge. Pogled ne zna kako su podaci prikazani u web-pregledniku. Posao pogleda je dohvatiti tražene podatke i proslijediti ih višem sloju koji će ih prikazati u pregledniku.

Predložak

Predložak (eng. template) je sloj arhitekture MTV-a usko povezan s web-preglednikom. Predložak je HTML stranica s dodatnim strukturama koje omogućavaju prikaz podataka koji su proslijeđeni od pogleda. Zadaća predloška je sadržaj primljen od pogleda organizirati i ugraditi u HTML kôd koji će se prikazati u web-pregledniku. Datoteka predloška ima dodatna ograničenja nemogućnosti upisivanja naredbi u programskom jeziku Python. Time onemogućuje miješanje funkcija pojedinih slojeva te pruža dodatnu sigurnost web-stranici. Dodatne strukture koje omogućuju prikaz podataka proslijeđenih od pogleda su oznake, ugrađeni filter i ugrađene programske strukture, no o tome detaljnije u poglavlju 3.

1.3 Instalacija

Django je „samo" Python kôd pa je potrebno prvo provjeriti da li je instaliran Python. Sva objašnjenja se odnose na Linuxovu distribuciju Ubuntu 13.04. Svejedno je koju verziju Pythona ćemo koristiti od 2.5 do 3.3 jer ih Django sve podržava. Razlike su neznatne jer sve što je potrebno za Django se u tim verzijama nije mijenjalo. Slike koje prikazuju izgled rješenja ili isječci kôda se odnose na Python verziju 2.7.5. Upute i potrebne datoteke za instalaciju Python-a za sve operacijske sustave nalaze se na [5].

Sljedeći korak je instalacija Djanga. Posljednja verzija je 1.7 objavljena u rujnu 2014. godine i radi sa svim verzijama Pythona. Obzirom da je zadnja verzija, time je i najažurnija te podržava sve mogućnosti koje Django trenutno pruža. Verzije manje od 1.6 rade samo s verzijama Pythona od 2.5 do 2.7 uključivo. Više verzije Pythona rade sa Djangom samo eksperimentalno. Postoje dvije vrste Django verzije, zadnje službeno izdanje te razvojno izdanje. U svrhu razvoja web-stranica potpuno je svejedno koje izdanje se koristi, no drugo izdanje služi za poboljšanje samog Djanga u koji se i sami možemo uključiti. Za naše potrebe je dovoljno imati samo zadnje službeno izdanje. Slike koje prikazuju izgled rješenja ili isječci kôda se odnose na Django verziju 1.7. Upute i potrebne datoteke za instalaciju Djanga za sve operacijske sustave nalaze se na [1].

Ukoliko pišemo web-stranice koje koriste baze podataka tada je potrebno instalirati željenu bazu podataka na računalo. Django podržava rad sa četiri baze podataka: PostgreSQL, SQLite3, MySQL i Oracle. Nakon odabira baze podataka potrebno je instalirati tu bazu podataka. Zatim je potrebno instalirati Python biblioteku za odabranu bazu podataka. To nije potrebno za SQLite3 bazu jer je biblioteka već uključena u instalaciju baze podataka. Slike koje prikazuju izgled rješenja ili isječci kôda se odnose na bazu podataka SQLite3. Upute i potrebne datoteke za instalaciju odabrane baze podataka za sve operacijske sustave nalaze se na [2], [3], [4] i [6].

1.4 Stvaranje Django projekta

Na početku razvoja web-stranice potrebno je stvoriti direktorij u kojem će se nalaziti svi kôdovi. Direktorij se može nalaziti bilo gdje na računalu. U PHP-u se kôd stavlja u direktorij */var/www* tj. u web-serverov korijen dokumenata (eng. web-server directory root). To se ne preporuča u Djangu jer postoji mogućnost da će korisnici vidjeti izvorni kôd preko weba, a to nikako ne želimo.

Da bismo stvorili Django projekt potrebno je ući u stvoreni direktorij preko terminala i pokrenuti naredbu *startproject*

```
nikolina@ubuntu:~/django$ django-admin.py startproject primjer
nikolina@ubuntu:~/django$ cd primjer
nikolina@ubuntu:~/django/primjer$ ls
primjer      manage.py
nikolina@ubuntu:~/django/primjer$ cd primjer
nikolina@ubuntu:~/django/primjer/primjer$ ls
__init__.py   settings.py   settings.py  urls.py  wsgi.py
```

Naredba će stvoriti projekt s imenom *primjer*. Stvoreni direktorij sadrži datoteku *manage.py* i novi ugnježđeni direktorij koji se zove isto kao i izvorni. Taj direktorij ćemo zvati konfiguracijski direktorij projekta radi lakšeg razumijevanja. *manage.py* je skripta koja pruža niz naredbi za pokretanje i održavanje projekta u Djangu. U konfiguracijskom direktoriju se nalaze datoteke *__init__.py*, *settings.py*, *urls.py* i *wsgi.py*. *__init__.py* je prazna Python skripta koja javlja interpreteru da je direktorij Python paket. *settings.py* je datoteka u kojoj se nalaze sve postavke projekta u Djangu. *urls.py* je datoteka u koju se spremaju URL obrasci (eng. pattern) projekta. *wsgi.py* je Python skripta koja pomaže pokrenuti Djangov razvojni server (eng. Django developement server, runserver) i razviti projekt u produkcijsku okolinu.

Da bismo mogli prikazati našu stranicu potrebno je pokrenuti Djangov razvojni server. On automatski ponovno učitava naš kôd što olakšava razvoj jer nema nepotrebnih ručnih pokretanja procesa. Da bismo pokrenuli razvojni server u direktoriju projekta (*primjer*) potrebno je startati naredbu

```
python manage.py runserver
```

Time dobijemo sljedeći ispis:

```
Performing system checks...
System check identified no issues (0 silenced).
You have unapplied migrations; your app may not work properly until they are
applied.
Run 'python manage.py migrate' to apply them.

August 19, 2015 - 12:12:10
Django version 1.7, using settings 'primjer.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

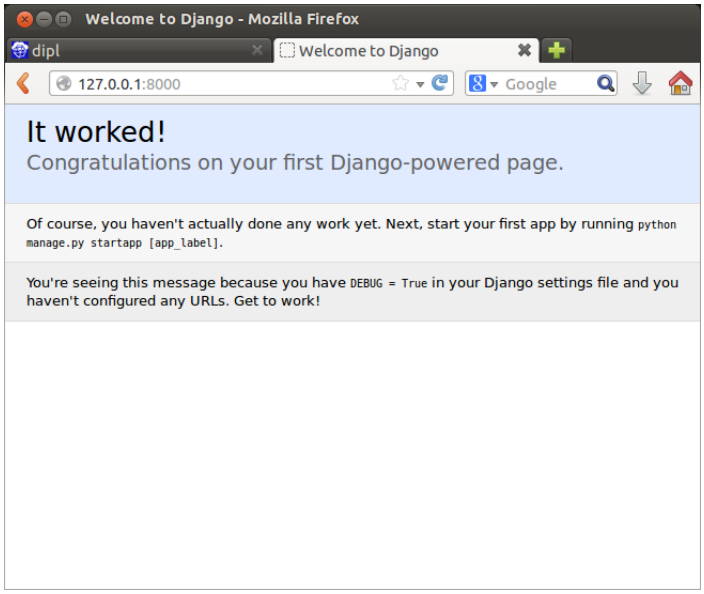
Ukoliko se javi greška upozorava nas da nismo postavili migracije na bazi podataka. Migracija u Djangu je objekt pomoću kojeg se pohranjuju promjene koje su napravljenje na modelima. Naredbom

```
python manage.py migrate
```

primijenimo sve migracije na bazu i time se stvori datoteka *db.sqlite3* te dobijemo sljedeći ispis:

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

Nakon toga ponovno pokrenemo naredbu *runserver*. Otvorimo web-preglednik na adresi `http://127.0.0.1:8000/` i dobijemo prikaz kao na slici 1.1. Adresu na kojoj se nalazi web-aplikacija dobijemo u ispisu nakon pokretanja naredbe *runserver*.



Slika 1.1: Početna stranica projekta

1.5 Stvaranje Django aplikacije

Projekt u Djangu je kolekcija konfiguracija i aplikacija i oni skupa daju web-stranicu. Ovaj pristup, koji koristi više manjih aplikacija u jednoj web-stranici, se koristi jer se na

taj način već stvorene manje aplikacije mogu ubaciti u drugi projekt u Djangu. To se najčešće koristi za generičke aplikacije, kao što su blogovi, i zbog toga stvaranje stranica zahtjeva minimalan napor. Jedna Django aplikacija odrađuje jedan zadatak, odnosno, svaka aplikacija daje drugu funkcionalnost i uklopljene čine jednu web-stranicu. Postoje stvorene generičke aplikacije koje se mogu samo skinuti s weba i uklopiti u projekt, no mi ćemo ovdje pokazati kako stvoriti vlastitu aplikaciju. Sad kad smo stvorili projekt razvijamo jednu aplikaciju, pod nazivom *osnove*, naredbom `startapp`:

```
nikolina@ubuntu:~/django/primjer$ python manage.py startapp osnove
nikolina@ubuntu:~/django/primjer$ cd osnove
nikolina@ubuntu:~/django/primjer/osnove$ ls
admin.py   migrations    urls.py    models.py   tests.py    views.py
__init__.py
```

Naredba stvara novi direktorij *osnove* u korijenskom direktoriju projekta i sadrži sljedećih pet datoteka : *__init__.py*, *tests.py*, *views.py*, *admin.py*, *models.py* te direktorij *migrations*. *__init__.py* ima istu ulogu kao istoimena datoteka nastala pri stvaranju projekta. *tests.py* je datoteka u koju se sprema niz test funkcija za testiranje našeg kôda. *views.py* je datoteka u kojoj definiramo niz funkcija koje primaju klijentove upite i vraćaju odgovore na te upite, odnosno u njoj definiramo funkcije pogleda. *admin.py* datoteka služi za registriranje modela koji se smiju koristiti u aplikaciji. *models.py* datoteka se koristi za pohranu definiranih modela, u njoj se definiraju atributi i veze među podacima. *views.py* i *models.py* su dvije osnovne datoteke koje ćemo koristiti kod razvoja svake aplikacije. U direktorij *migrations* se automatski spremaju sve migracije na bazi podataka.

Nakon što smo stvorili aplikaciju, projektu je potrebno reći koje smo nove aplikacije dodali. Uđemo u *settings.py* u konfiguracijskom direktoriju i u varijablu `INSTALLED_APPS` navedemo nove aplikacije koje smo dodali i koje želimo uključiti u naš projekt. Varijabla `INSTALLED_APPS` je tuple (posebna vrsta liste u Python-u) stringova i potrebno je iza posljednjeg elementa staviti zarez!

```
1 INSTALLED_APPS = (
2     'django.contrib.admin',
3     'django.contrib.auth',
4     'django.contrib.contenttypes',
5     'django.contrib.sessions',
6     'django.contrib.messages',
7     'django.contrib.staticfiles',
8     'osnove',
9 )
```

Poglavlje 2

Pogledi

2.1 Pogledi i povezivanje s URL-om

Nakon što smo stvorili aplikaciju želimo da ta aplikacija ima neku funkcionalnost. Sada krećemo razvijati dijelove MTV arhitekture i međusobno ih povezivati da bismo dobili cjelovitu web-stranicu. Započnimo jednostavnim primjerom. Želimo stvoriti web-stranicu koja prikazuje tekst „Ovo je stranica za prikaz clanaka!". Da bismo prikazali najjednostavniji tekst na web-stranici potrebne su nam dvije stvari: sadržaj, tj. sami tekst koji će se prikazati te URL, npr. `http://www.example.com/tekst/` gdje će se taj tekst prikazati. U Djangu su te dvije stvari razdvojene. Sadržaj se zadaje pomoću pogleda, točnije pomoću funkcije pogleda (eng. view function), a URL se zadaje u datoteci *urls.py*. Ovako izgleda pripadna funkcija pogleda spremljena u datoteci *views.py*.

```
1 from django.http import HttpResponse
2
3 def index(request):
4     return HttpResponse("Ovo je stranica za prikaz clanaka!")
```

Svaka funkcija pogleda prima barem jedan parametar tipa `HttpRequest` koji se dogovorno zove `request`. Taj objekt sadrži informacije o trenutnom zahtjevu za ovaj pogled koji postavlja klijent. U ovom jednostavnom primjeru mi ne koristimo parametar `request`, ali on uvijek mora biti prvi parametar u funkciji pogleda. Svaka funkcija pogleda vraća objekt tipa `HttpResponse`. Ovdje on sadrži `string` (sadržaj) koji šaljemo klijentu koji je zatražio taj pogled.

Projekt još uvijek ne zna za naš pogled, moramo mu točno reći da se taj pogled aktivira na određenom URL-u (Uniform Resource Locator). Za to trebamo stvoriti datoteku *urls.py* u direktoriju naše aplikacije. Ona predstavlja „sadržaj" naše stranice. Django to prevodi ovako: „Za određeni URL pozovi zadani kôd odnosno pogled". Primijetimo da ovdje stvaramo novu datoteku. Dobro je razdvajati URL-ove za svaku aplikaciju. Iako postoji stvorena datoteka

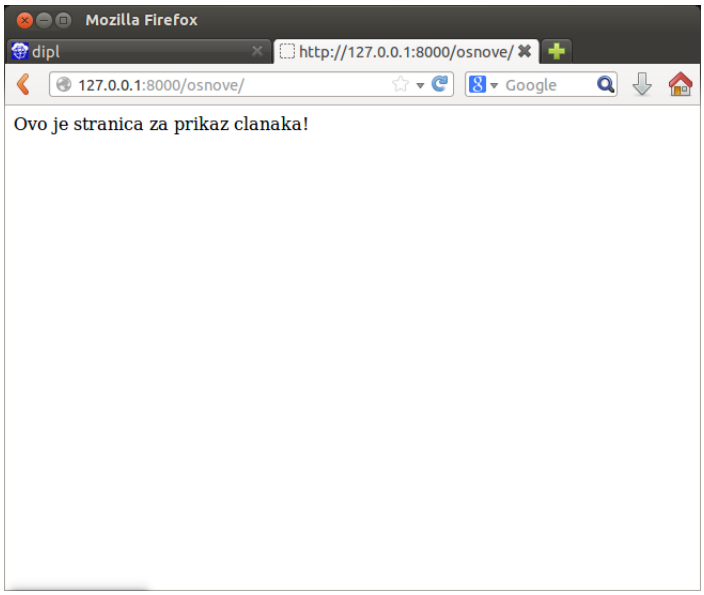
urls.py u konfiguracijskom direktoriju, bolje rješenje je za svaku aplikaciju stvoriti vlastitu datoteku. Podsjetimo se, već smo naglasili da želimo individualnost aplikacija. Unutar datoteke *urls.py* potrebno je dodati varijablu `urlpatterns` koja označava preslikavanje između URL-ova i pogleda. U našem slučaju želimo da se pozove funkcija `index`:

```
1 from django.conf.urls import patterns, url
2 from osnove import views
3
4 urlpatterns = patterns('',
5         url(r'^$', views.index, name='index'),
6 )
```

Varijabla `urlpatterns` je također tuple i iza posljednjeg elementa mora slijediti zarez! Nadalje, prosljeđujemo funkciju pogleda URL-u kao objekt bez poziva funkcije, odnosno prosljeđujemo funkcije kao bilo koje druge varijable.

Obzirom da smo razdvojili popis URL-ova za svaku aplikaciju, sada novostvorenu datoteku trebamo dodati u originalnu datoteku *urls.py* našeg projekta koja se nalazi u konfiguracijskom direktoriju. U varijablu `urlpatterns` potrebno je dodati popis novih URL-ova:

```
1 url(r'^osnove/', include('osnove.urls')),
```



Slika 2.1: Prikaz teksta proslijeđenog iz pogleda

Sada ponovno otvorimo preglednik na adresi `http://127.0.0.1:8000/osnove` i vidjet ćemo da se tekst prikazao kao na slici 2.1.

2.2 Regularni izrazi

URL-ovi su zapisani pomoću regularnih izraza. Ispred svakog regularnog izraza nalazi se znak `'r'` koji označava da je regularni izraz koji slijedi „raw string" tj. u njegovom izrazu ne treba interpretirati znak `'\'`. U Pythonu taj znak u kombinaciji s drugim određenim znakovima označava posebne znakove, npr. `'\n'`. Tablica 2.1 navodi najčešće regex znakove koje ćemo koristiti za regularne izraze.

Simbol	Značenje
$	Kraj stringa
^	Početak stringa
.	Bilo koji znak
\w	Bilo koje slovo ili znamenka
\d	Bilo koja znamenka
[A–Z]	Bilo koje veliko slovo od A do Z
[a–z]	Bilo koje malo slovo od a do z
[A–Za–z]	Bilo koje (veliko ili malo) slovo od A do Z
+	Jedan ili više znakova prethodnog izraza
*	Nijedan ili više znakova prethodnog izraza
?	Jedan ili nijedan znak prethodnog izraza
{1,3}	Između jedan i tri znaka prethodnog izraza
[^]	Bilo koji znak osim navedenih u uglatim zagradama
( )	Grupiranje znakova u izraz

Tablica 2.1: Regex znakovi

2.3 Princip labavog spajanja

Labavo spajanje (eng. loose coupling) je pristup u razvoju softvera u kojemu komponente imaju vrlo malo znanja o drugim odvojenim komponentama. Ako su dvije komponente labavo spojene tada će promjene u jednoj imati malo ili nimalo utjecaja na drugu komponentu. Primjer ovog pristupa je datoteka *urls.py*. U Djangovoj aplikaciji URL definicije i funkcije pogleda ih koje pozivaju su labavo spojeni. Odluka na kojem URL-u će biti određena funkcija pogleda i sama implemenatacija funkcije su dvije zasebne komponente. Time je omogućeno da mijenjamo funkciju pogleda što neće utjecati na URL definiciju ili obrnuto.

Na primjer, recimo da našu trenutnu funkciju pogleda, `index`, želimo staviti na drugi URL, npr. `http://127.0.0.1:8000/osnove/index`. Samo ćemo promijeniti definiciju URL-a u datoteci *urls.py* u:

```
1 url(r'^index/', views.index, name='index'),
```

bez da brinemo o samoj funkciji pogleda. Obrnuto, ako želimo promijeniti funkciju pogleda tako da se prikaže drugi sadržaj, promjena neće utjecati na URL za koji je funkcija vezana. Nadalje, ako želimo da se isti tekst (odnosno, ista funkcija pogleda) veže za više različitih URL-ova, tada samo dodamo definicije URL-ova za koje želimo vezati tu funkciju, bez da mijenjamo kôd funkcije pogleda.

Poglavlje 3

Predlošci

3.1 Povezivanje pogleda s predloškom

Django koristi predloške (eng. template) za jednostavnije oblikovanje dizajna stranice te za odvajanje logike aplikacije od prezentacijskih koncepata. Za stvaranje predožaka potrebno je stvoriti direktorij u kojem će se nalaziti svi predlošci, stvorimo ga u direktoriju projekta pod nazivom *templates*. U njemu ćemo stvoriti direktorij *osnove*, što sugerira da ćemo razdvajati predloške za svaku aplikaciju. Zatim je potrebno projektu reći gdje se nalaze svi predlošci. U datoteci *settings.py* potrebno je definirati tuple stringova `TEMPLATE_DIRS`:

```
1 TEMPLATE_PATH = os.path.join(BASE_DIR, 'templates')
2 TEMPLATE_DIRS = (
3     TEMPLATE_PATH,
4 )
```

Varijabla koja se nalazi u *settings.py* pod nazivom `BASE_DIR` označava koji je apsolutni put direktorija projekta. Koristeći tu varijablu ne moramo razmišljati koji je put do bilo koje datoteke našeg projekta. Sada ćemo stvoriti novi predložak imena *index.html* i spremiti ga u prethodno stvoreni direktorij sa sljedećim kôdom:

```
1  <html>
2      <head>
3          <title>Clanci</title>
4      </head>
5
6      <body>
7          <h1>Ovdje cete moci pronaci</h1>
8          popis <strong>{{ message }}</strong><br />
9          <a href="/osnove/">Pocetna stranica</a><br />
10     </body>
```

```
11 </html>
```

Vidimo da je kôd vrlo sličan klasičnom HTML-u, no ima još nekih dodataka kao što je {{ `message` }} . Ovo je Djangova varijabla predloška (eng. template variable), no o tome detaljnije u odjeljku 3.2. Sada je potrebno dodati funkciju pogleda u datoteci *views.py* kako bi se prikazao naš predložak na sljedeći način:

```
1 from django.shortcuts import render
2 def introduction(request):
3     dictionary = {'message': "svih znanstvenih clanaka"}
4     return render(request, 'osnove/index.html', dictionary)
```

Konstruiramo rječnik koji koristimo s predloškom, ključ je varijabla predloška, a vrijednost je string čiju vrijednost želimo da poprimi varijabla `message`. Rekli smo da svaka funkcija pogleda vraća `HttpResponse`, no mi ovdje moramo učitati stvoreni predložak, povezati ga s funkcijom pogleda, a zatim vratiti `HttpResponse`. Zbog toga koristimo funkciju `render()` koja predstavlja prečac za te radnje. Ne smijemo zaboraviti u datoteku *urls.py* dodati URL na kojem želimo da se prikaže predložak sa pripadnom funkcijom pogleda. Rezultat izgleda kao na slici 3.1.

Slika 3.1: Primjer korištenja predloška

U primjeru vidimo da je link prema početnoj web-stranici koja je također dio našeg projekta eksplicitno napisan. No, da ne bismo svaki put morali tražiti koji je određeni URL neke stranice i da li je on mapiran u varijabli `urlpatterns` ovdje nam pomaže `url`

oznaka. Ona se nalazi, kao i ostale oznake jezika predložaka, unutar {% i %} znakova. Dakle, prethodni kôd:

```
1 <a href="/osnove/">Pocetna stranica</a>
```

možemo zamijeniti sljedećim:

```
1 <a href="{% url 'index' %}">Pocetna stranica</a>
```

jer smo svakom novododanom URL-u inicijalizirali opcionalni parametar `name`. To će biti posebno korisno kada budemo imali veliki broj zadanih URL-ova kako stranica bude rasla.

Korištenjem `url` oznake se također može prosljeđivati parametar u funkciju pogleda na način da se parametar navede nakon naziva URL-a unutar oznake. To će nam biti potrebno kad budemo koristili slug-ove, no o tome u odjeljku 5.3.

3.2 Djangov jezik predložaka

Iako predlošci podosta liče na HTML jezik, da bismo ih koristili u Djangu koristit ćemo i Djangov jezik predložaka. To je posebna sintaksa koju koristimo dok razvijamo Django aplikaciju i omogućuje različite manipulacije podacima koji su proslijeđeni predlošku. Osnovni dijelovi jezika predložaka su varijabla predloška, oznaka predloška i filter.

Varijabla predloška (eng. template variable) zapisana je unutar znakova {{ i }}. Toj varijabli je pridružena određena vrijednost.

Oznaka predloška (eng. template tag) zapisana je unutar znakova {% i %}. U ovom poglavlju dajemo popis osnovnih oznaka koje ćemo koristiti u razvoju te njihova objašnjenja.

if/else

Koristi se u kombinaciji s varijablom predloška i ako je varijabla `True` tada se prikazuje sav sadržaj između {% `if` %} i {% `endif` %} oznaka. Oznaka `else` je opcionalna. Ukoliko je uključena, ako je varijabla kombinirana s `if` oznakom `True`, prikazuje se sav sadržaj između {% `if` %} i {% `else` %} oznaka. Inače, prikazuje se sav sadržaj između {% `else` %} i {% `endif` %} oznaka. Oznaka `if` također prihvaća `and`, `or` i `not` operatore za testiranje višestrukih varijabli, odnosno negiranje varijable, no ne dopušta korištenje `and` i `or` operatora u istoj rečenici. Dopuštena je ponovljena upotreba istog operatora u jednoj rečenici. Bitna stvar je da iza svake {% `if` %} oznake mora doći {% `endif` %} oznaka, inače će Django baciti `TemplateSyntaxError` iznimku. Slijedi primjer pravilne upotrebe `if` oznake.

```
1 {% if varijabla %}
2     <strong>Varijabla ima valjanu vrijednost!</strong>
3 {% else %}
```

```
4       <strong>Varijabla je prazna ili jednaka nuli!</strong>
5   {% endif %}
```

for

Sljedeća često korištena oznaka je `for` koja simulira klasičnu for-petlju u Pythonu. Sintaksa je {% for X in Y %} gdje je Y polje po kojem se prolazi, a X je element polja Y. Svaki prolazak kroz petlju znači ispis cijelog sadržaja između {% for %} i {% endfor %} oznaka. Oznaka `for` može sadržavati dodatne parametre kao što je `reverse`, što znači da petlja prolazi kroz cijelo polje unatrag. Djangov jezik predložaka dopušta ugniježđene for-petlje. Obzirom da se često na početku provjerava da li je lista po kojoj prolazi petlja prazna, Django je uveo posebnu oznaku koja se koristi u tom slučaju, {% empty %}. Oznaka `for` ne podržava bilo koju varijaciju tipične naredbe `break`, tj. nije moguće prekinuti petlju prije nego se ona do kraja izvrti. Ako to želimo, potrebno je postaviti drugačiji uvjet na petlju. Sa svakom `for` oznakom dobivamo pristup varijabli predloška pod nazivom `forloop` koja daje informacije o trenutnom napretku u petlji kao što su: broj dosadašnjih prolaza kroz petlju, broj preostalih prolaza kroz petlju, da li je trenutni prolazak prvi prolazak kroz petlju, da li je trenutni prolazak zadnji prolazak kroz petlju te referenca na roditeljsku petlju ukoliko je petlja ugniježđena. Iza svake {% for %} oznake mora se nalaziti {% endfor %} oznaka, inače će Django opet izbaciti iznimku. Slijedi primjer korištenja `for` oznake.

```
1   {% for l in lista %}
2       {% if forloop.first %}
3           <strong>Ovo je prvi prolaz kroz petlju!</strong>
4           {{ l }}
5       {% else %}
6           {{ l }}
7   {% empty %}
8           <p>Lista je prazna!</p>
9   {% endfor %}
```

ifequal

Oznaka `ifequal` omogućuje uspoređivanje dviju vrijednosti i ispisivanje sadržaja ako su oni jednaki. Iza oznaka također moraju slijediti završne oznake, tj. {% endifequal %} oznaka. Argumenti u oznakama mogu biti stringovi umjesto varijabli predloška (npr. `'marko'` ili `"marko"`). Drugi tipovi kao što su riječnici, liste ili Boolean vrijednosti se ne mogu uspoređivati ovim oznakama. U tom slučaju jednakosti je potrebno testirati {% if %} oznakom. Ove oznake podržavaju opcionalnu {% else %} oznaku koja prikazuje sadržaj u slučaju nejednakosti vrijednosti.

```
{% ifequal string_prvi string_drugi}
    <p>Stringovi su jednaki.</p>
{% endifequal %}
```

Komentar

Djangov jezik dopušta komentiranje koristeći oznaku {# #}. Sve što se nalazi unutar te oznake neće biti ispisano na dotičnoj web-stranici. Na ovaj način nije moguće komentirati više linija kôda. Za komentiranje više linija kôda koristi se {% comment %} oznaka iza koje mora slijediti {% endcomment %} oznaka.

```
{# Ovo je jedan komentar #}
{% comment %}
    Ovo je jos jedan komentar.<br />
    ...i jos jedan komentar.
{% endcomment %}
```

Filter

Filteri unutar predloška su zapisani u obliku {{ varijabla|filter }}. Oni služe za promjenu teksta prije samog prikazivanja. Primjer filtera je {{ name|lower }} koji varijablu name mijenja tako da sva slova pretvori u mala slova. Filteri se mogu ulančavati, odnosno moguće je koristiti nekoliko uzastopnih filtera koji će se primijenjivati na istu vrijednost varijable. Najčešće korišteni filteri su: addslashes – dodaje jedan znak '\' prije svakog znaka '\', date – formatira date ili datetime objekt u format koji je zadan i length – vraća duljinu vrijednosti varijable.

3.3 Bazni predložak

Kod pisanja predložaka imamo puno ponavljajućeg kôda. Svaki predložak mora imati osnovne elemente kao što su naslov ili zaglavlje odnosno podnožje za koje očekujemo da će se pojavljivati u istom obliku na bilo kojem predlošku. Nerealno je očekivati da ćemo svaki puta kopirati isti dio kôda zadužen za to i zbog toga se Django pobrinuo za sve. Najjednostavniji način je da stvorimo bazni predložak iz kojeg će se nasljeđivati svi ostali predlošci u našoj aplikaciji. U baznom predlošku potrebno je stvoriti kostur za standardnu stranicu sa svim potrebnim sadržajima, neka se zove *base.html*. U baznom predlošku označimo sve što će se moći redefinirati u naslijeđenim predlošcima (laički rečeno, sve ono što ćemo ipak moći promijeniti u naslijeđenim predlošcima). Jedan od primjera je {% block tijelo %} koji također mora završavati pripadnom {% endblock %} oznakom.

On označava da se sve unutar tog bloka (mislimo na tijelo stranice) može mijenjati. Dijelovi u naslijeđenom predlošku koji se redefiniraju također moraju biti unutar {% block %} i {% endblock %} oznaka. Neka datoteka *base.html* izgleda ovako:

```
<html>
    <head>
        <title>Clanci {% block naslov %}{% endblock %}</title>
    </head>

    <body>
      <div>
          {% block lista %}{% endblock %}
      </div>
        <div>
            {% block tijelo %}{% endblock %}
        </div>
        <hr />
        <div>
          <a href="{% url 'index' %}">Pocetna stranica</a>
        </div>
    </body>
</html>
```

Moramo uzeti u obzir sve dijelove, odnosno blokove koji će nam se ponavljati na stranicama da ne bismo imali ponavljajući kôd. Prije početka moramo razmisliti kako će nam izgledati kostur stranice, da li će imati zaglavlje, podnožje, što će se u njemu nalaziti, da li će imati izbornik sa strane. Kada odlučimo što ćete imati onda stvaramo bazni predložak jer su to sve dijelovi koji se pojavljuju na svim stranicama. Tako ćemo izbjeći ponavljanje kôda i gubljenje vremena. Ako prepravimo *index.html* tako da nasljeđuje bazni predložak, izgledat će ovako:

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

{% block naslov %}- Uvod{% endblock %}
{% block lista %}
    Ovo ce se nalaziti sa strane kao izbornik!
{% endblock %}
{% block tijelo %}
    <h1>Ovdje cete moci pronaci</h1>
    popis <strong>{{ message }}</strong><br />
{% endblock %}
```

Sada će sve ostale stranice koje razvijamo imati taj kostur. Naravno, možemo imati i nekoliko baznih predložaka, npr. drugačiji izgled za svaku aplikaciju, zbog toga je ključno

navesti {% extends %} oznaku.

3.4 Statični mediji u projektu

Naša web-stranica, kako smo je do sada stvorili, je prilično siromašna. CSS (Cascading Style Sheets), JavaScript i slike su osnovne datoteke (eng. static media file) koje možemo uključiti u našu web-stranicu da bismo stvorili ljepši stil ili dodali dinamičko ponašanje. Ove datoteke se dodaju drugačije od običnih HTML dokumenata pa im posvećujemo posebno poglavlje.

Slike

Potrebno je stvoriti direktorij u koji ćemo spremiti sve statične medije, najbolje u korijenskom direktoriju projekta, pod nazivom *static*. Zbog jednostavnijeg snalaženja stvorit ćemo poddirektorij *images* da bismo kasnije razdvojili vrste statičnih medija. Odaberemo neku sliku koju želimo umetnuti u web-stranicu i spremimo je u taj direktorij pod imenom *book.jpg*. Zatim, kao i kod stvaranja predložaka, moramo projektu reći za naš direktorij, tj. reći mu put do njega. U *settings.py* dodamo sljedeće linije kôda:

```
1 STATIC_PATH = os.path.join(BASE_DIR,'static')
2 STATIC_URL = '/static/'
3 STATICFILES_DIRS = (
4     STATIC_PATH,
5 )
```

Varijabla STATIC_URL je vjerojatno već dodana, no ukoliko nije moramo je dodati sami. U preglednik upišemo adresu http://127.0.0.1:8000/static/images/book.jpg i zadana slika će se prikazati. No, to naravno nije zanimljivo. Sada ćemo prethodno stvoreni predložak izmijeniti kako bi prikazivao sliku zajedno s tekstom na web-stranici. Na početku datoteke je potrebno dodati sljedeći kôd:

```
1 {% load staticfiles %}
```

Time obavještavamo predložak da ćemo koristiti statičke medije. Zatim na mjesto gdje želimo da se prikaže slika dodamo sljedeći kôd:

```
1 <img src="{% static 'images/book.jpg' %}" alt="Knjiga" />
```

Primijetimo logiku oznake koja je vrlo slična HTML jeziku. Razlika je u zadavanju URL-a. Stvorena stranica izgleda kao na slici 3.2.

Slika 3.2: Prikaz slike unutar stranice

CSS

Jedna od najvažnijih stvari u izgradnji web-stranica je njihov stil. On pridonosi dojmu korisnika. Zbog toga će CSS biti jedna od stvari koje ćemo sigurno koristiti u svom projektu. CSS je stilski jezik koji se rabi za opis prezentacije dokumenta. U početku razvoja CSS se pisao unutar samog HTML dokumenta, no ubrzo su programeri shvatili da je bolje rješenje odvajati ga u zasebnu datoteku. Na taj način se koristi i danas. Potrebno je prvo stvoriti direktorij *css* unutar direktorija *static* i u njemu datoteku *style.css*. U njoj definiramo stilove koje koristimo u svojoj stranici. Svaki element može biti stiliziran. CSS za određeni element opisuje kako će biti izražen na ekranu. To radimo tako da pridružimo vrijednosti različitim svojstvima povezanih s elementom. Postoji mnogo svojstava koje možemo koristiti. Popis svojstava se nalazi na [7]. Koristeći CSS možemo zadati fontove, boje, pozadine, obrube te poziciju elementa.

Potrebno je još spomenuti CSS selektore. Oni se koriste za mapiranje određenog stila sa HTML elementom. Osnovni CSS selektori su: element selektor, identifikacijski selektor i klasni selektor.

Element selektori se odnose na određenu HTML oznaku npr. `<body>`, `<h1>`, `<h2>`, `<h3>`, `<p>` i `<div>`. Stil se primjenjuje na sve instance određene oznake.

Identifikacijski selektor (eng. ID) se koristi za mapiranje jedinstvenog elementa u dokumentu. Svaki element može biti jedinstven ako mu zadamo njegov `id` atribut. Ovaj tip selektora počinje # simbolom prije imena.

Klasni selektor (eng. class) je sličan identifikacijskom selektoru, no s klasnim selektorom možete obuhvatiti više od jednog elementa. Pripadnost klasi se određuje postavljanjem `class` atributa. Ovaj tip selektora počinje `.` simbolom prije imena.

Nakon što smo odredili stilove elemenata potrebno je datoteku uklopiti u projekt. Unutar baznog predloška potrebno je dodati sljedeće linije kôda:

```
1 {% load staticfiles %}
2 <link rel="stylesheet" type="text/css" href="{% static
3     'css/style.css' %}" />
```

Na slici 3.3 možete vidjeti jedan primjer korištenja CSS-a za stiliziranje naše dosadašnje stranice.

Slika 3.3: Umetnje CSS-a u stranicu

3.5 Server statičnih medija

Sad smo naučili kako prikazati statične medije u projektu. No, ponekad želimo da korisnici mogu staviti statične medije na našu stranicu, npr. ako žele staviti svoju profilnu sliku prilikom registracije. Za to nam je potreban server na kojem će se pohranjivati mediji koje korisnik doda.

Prvo ćemo stvoriti novi direktorij *media* u direktoriju projekta. U datoteku *urls.py* je potrebno dodati sljedeće linije kôda da bi se moglo pristupati varijablama koje se nalaze u postavkama našeg projekta.

```
from django.conf import settings

if settings.DEBUG:
    urlpatterns += patterns(
        'django.views.static',
        (r'^media/(?P<path>.*)',
        'serve',
        {'document_root': settings.MEDIA_ROOT}), )
```

Zatim je potrebno modificirati datoteku *settings.py* kako bi projekt znao put do servera:

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Sada je server spreman za dodavanja medija. Često se kombinira sa `FileField` tipom atributa unutar modela za dodavanja datoteka na stranicu (mi ćemo ga koristiti za dodavanja članaka u PDF-u).

Poglavlje 4

Modeli

4.1 Modeli i baze podataka

U suštini, model je objekt koji opisuje neku tablicu podataka. Umjesto direktnog rada s tablicama baza podataka preko SQL-a, upiti se izršavaju pozivanjem metoda objekta. Prije rada s modelima potrebno je namjestiti bazu podataka. Nakon što se stvori projekt, Django automatski dodaje rječnik pod nazivom `DATABASES` koji se nalazi u *settings.py*. U tom rječniku je navedeno koja baza podataka se koristi (ovdje SQLite3). Ključevi koji se nalaze u tom rječniku su: `engine`, `name`, `user`, `password`, `host` i `port`, no potrebno je inicijalizirati samo `engine` i `name`.

Nakon toga dolazi definiranje modela. Oni se definiraju u datoteci *models.py* koja se nalazi u direktoriju pripadne aplikacije. Svaki novi model predstavlja klasu koja mora naslijediti klasu `django.db.models.Model`. Slijedi primjer modela:

```
1  class Autor(models.Model):
2      ime = models.CharField(max_length=128, unique=True)
3      web = models.URLField()
4      email = models.EmailField()
5      def __unicode__(self):
6          return self.ime
7
8  class Clanak(models.Model):
9      autor = models.ManyToManyField(Autor)
10     naslov = models.CharField(max_length=128)
11     detalji = models.TextField(blank=True)
12     pdf = models.FileField(upload_to='pdf', blank=True)
13     def __unicode__(self):
14         return self.naslov
```

Potrebno je navesti listu atributa i njihove tipove sa eventualnim dodatnim parametrima. Svaki model predstavlja jednu tablicu u bazi podataka, a svaki atribut predstavlja jedan stupac te tablice. Stvaranje tablica i ažuriranje u bazi, obzirom na definicije, Django obavlja sam. Za to se koriste migracijski alati (eng. migration tools). Oni služe zapravo da, ukoliko napravimo promjenu u modelu, možemo ažurirati bazu bez da ju prvo brišemo. Prvo treba inicijalizirati bazu, a to smo napravili kod stvaranja projekta, naredbom

```
python manage.py migrate
```

Svaki put kad stvaramo novi model ili mijenjamo postojeći model potrebno je registrirati te promjene, a to se izvodi pomoću naredbe `makemigrations` za dotičnu aplikaciju, npr.

```
python manage.py makemigrations osnove
```

Time se stvaraju nove migracije. Da bi se migracije primijenile na bazi potrebno je startati naredbu `migrate`. Svaki put kad se mijenja postojeći model treba izvesti obje naredbe!

```
nikolina@ubuntu:~/web/primjer$ python manage.py makemigrations osnove
Migrations for 'osnove':
  0001_initial.py:
    - Create model Autor
    - Create model Clanak
nikolina@ubuntu:~/web/primjer$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, contenttypes, osnove, auth, sessions
Running migrations:
  Applying osnove.0001_initial... OK
nikolina@ubuntu:~/web/primjer$
```

Sljedeća stvar je stvaranje superkorisnika koji će upravljati bazom naredbom

```
python manage.py createsuperuser
```

Ispis izgleda ovako:

```
nikolina@ubuntu:~/django/primjer$ python manage.py createsuperuser
Username (leave blank to use 'nikolina'): root
Email address: nivezic91@gmail.com
Password:
Password (again):
Superuser created successfully.
```

4.2 Popis tipova atributa i osnovnih parametara modela

Sada ćemo navesti sve tipove atributa modela koje će nam biti potrebne u razvoju aplikacije u tablici 4.1. Parametri označeni u `[]` zagradama su opcionalni, ostali su obavezni. Parametar `**options` označava parametre koji su dostupni svim tipovima, a njih ćemo navesti u tablici 4.2. Django također definira polja koja predstavljaju veze među modelima, navodimo ih u tablici 4.3. Detalji o tipovima, vezama i parametrima nalaze se na [1].

Tip	Parametri	Značenje tipa
`AutoField`	`**options`	`IntegerField` koji se automatski povećava za 1 prema dosadašnjim ID-evima
`BigIntegerField`	`[**options]`	Sličan `IntegerField`-u, ali podržava brojeve od −9223372036854775808 do 9223372036854775807
`BinaryField`	`[**options]`	Za spremanje sirovih binarnih podataka
`BooleanField`	`**options`	Prihvaća `True` ili `False` vrijednost
`CharField`	`max_length=None, [**options]`	Polje za stringove
`CommaSeparatedIntegerField`	`max_length=None, [**options]`	Polje integera odvojenih zarezom
`DateField`	`[auto_now=False, auto_now_add=False, **options]`	Datum, u Python-u reprezentiran `datetime.date` instancom
`DateTimeField`	`[auto_now=False, auto_now_add, **options]`	Datum i vrijeme, u Python-u reprezentiran `datetime.datetime` instancom
`DecimalField`	`max_digits=None, decimal_places=None, [**options]`	Decimalni broj s fiksnom točkom, u Python-u reprezentiran `Decimal` instancom
`EmailField`	`[max_length=75, **options]`	`CharField` koji provjerava da li je email adresa valjana
`FileField`	`[upload_to=None, max_length=100, **options]`	Polje za datoteku
`FilePathField`	`path=None, [mathc=None, recursive=False, max_length=100, **options]`	`CharField` ograničen na nazive datoteka u određenom direktoriju

Tip	Parametri	Značenje tipa
`FloatField`	`[**options]`	Floating-point broj, u Python-u reprezentiran `float` instancom
`ImageField`	`[upload_to=None, height_field=None, width_field=None, max_length=100, **options]`	Nasljeđuje sve atribute i metode od `FileField`, ali također provjerava je li objekt valjana slika
`IntegerField`	`[**options]`	Integer, vrijednosti u rasponu od −2147483648 do 2147483647
`GenericIPAdressField`	`[protocol=both, unpack_ipv4=False, **options]`	IPv4 ili IPv6 adresa formata `192.0.2.30` ili `2a02:42fe::4`
`NullBooleanField`	`[**options]`	Kao `BooleanField`, no dopušta da vrijednost bude `Null`
`PositiveIntegerField`	`[**options]`	Pozitivni integer, vrijednosti od 0 do 2147483647
`PositiveSmallIntegerField`	`[**options]`	Kao `PositiveIntegerField`, ali prima vrijednosti od 0 do 32767
`SlugField`	`[max_length=50, **options]`	Slug je skraćena oznaka za nešto, većinom korištena za URL, slično `CharField`-u
`SmallIntegerField`	`[**options]`	Kao `IntegerField`, ali vrijednosti od −32768 do 32767
`TextField`	`[**options]`	Veliko polje za unos teksta
`TimeField`	`[auto_now=False, auto_now_add=False, **options]`	Vrijeme, u Python-u reprezentiram `datetime.time` instancom
`URLField`	`[max_length=200, **options]`	`CharField` za URL

Tablica 4.1: Tipovi atributa modela

Parametar	Zadana vrijednost
null	False
blank	False
choices	
column	ime polja
db_index	False
db_tablespace	DEFAULT_INDEX_TABLESPACE
default	
editable	True
error_messages	
help_text	
primary_key	
unique	
unique_for_date	
unique_for_month	
unique_for_year	
verbose_name	ime polja gdje su znakovi _ zamijenjeni razmakom
validators	

Tablica 4.2: Opcionalni parametri

Polje	Parametri	Značenje polja
ForeignKey	Othermodel, [**options]	veza mnogo : jedan
ManyToManyField	Othermodel, [**options]	veza mnogo : mnogo
OneToOneField	Othermodel, [**options]	veza jedan : jedan

Tablica 4.3: Polja povezivanja

4.3 Povezivanje modela i pogleda

Nakon što smo naučili kako funkcioniraju osnovni dijelovi Django projekta prema MTV arhitekturi i kako se pogled povezuje s predloškom, vrijeme je da vidimo kako se podaci iz baze mogu koristiti preko pogleda. Osnovni koraci su: uvoz potrebnih modela koje ćemo

koristiti u datoteku *views.py*, u pogledu zatražiti podatke koje želimo prikazati, proslijediti podatke koje želimo prikazati predlošku, postaviti predložak na način na koji želimo da se prikaže te mapirati URL.

Prvo mijenjamo datoteku *views.py*:

```
from osnove.models import Autor, Clanak

def lista_autora(request):
    lista = Autor.objects.order_by('ime')
    dictionary = {'authors': lista}

    return render(request, 'osnove/autori.html', dictionary)
```

Sljedeći je korak urediti predložak kako želimo da se naši podaci prikažu. Ovo je jedan primjer:

```
{% extends 'osnove/base.html' %}
{% load staticfiles %}

{% block naslov %}- Popis autora{% endblock %}

{% block tijelo %}
  {% if authors %}
        <h1>Popis autora:</h1>
        <ul>{% for author in authors %}
        <li>{{author.ime}}</li>
        {% endfor %}</ul>
    {% else %}
        <strong>Trenutno nemate autora u bazi.</strong>
    {% endif %}
{% endblock %}
```

Potrebno je jos mapirati URL na kojem želimo da se prikaže naša stranica, samo dodamo sljedeće u `urlpatterns`

```
url(r'^autori/$', views.lista_autora, name='autori')
```

Sad odemo na adresu `http://127.0.0.1:8000/osnove/autori` u pregledniku. Prikaz je na slici 4.1.

4.4 Povezivanje modela i predloška

Do sada smo upoznali osnovne oznake predloška. No, Django nam omogućava da stvorimo svoje vlastitite oznake. Zamislimo da želimo urediti stranicu tako da se svi naši

Slika 4.1: Prikaz podataka iz baze

autori koje korisnici mogu pogledati nalaze na dijelu stranice u zasebnom bloku, ali na svakoj stranici. Ako bismo koristili bazni predložak imali bismo previše ponavljajućeg kôda u svakom pogledu jer listu autora ne možemo proslijediti direktno baznom predlošku. Lakši način je da stvorimo vlastite oznake predloška.

U svojoj aplikaciji stvorimo direktorij *templatetags* i u njemu dvije nove datoteke, *__init__.py* koja je prazna te *extra_tags.py* u koju ćemo upisati sljedeći kôd:

```
1 from django import template
2 from osnove.models import Autor
3 register = template.Library()
4
5 @register.inclusion_tag('osnove/aut.html')
6 def autori_lista():
7     return {'autori': Auhor.objects.all()}
```

Stvorili smo pogled koji pronalazi listu autora i prosljeđuje je *aut.html* predlošku. Sada je potrebno stvoriti taj predložak na sljedeći način:

```
1 {% if autori %}
2     <ul>{% for a in autori %}
3         <li>{{ a.ime }}</li>
4         {% endfor %}
5     </ul>
6 {% else %}
```

```
7       <li><strong>Nemate autora u bazi.</strong></li>
8   {% endif %}
```

U baznom predlošku pristupamo toj oznaci predloška tako da je prvo učitamo naredbom:

```
1   {% load extra_tags %}
```

a zatim ga dodamo u stranicu sljedećim kôdom u baznom predlošku:

```
1   {% block lista %}
2       {% autori_lista %}
3   {% endblock %}
```

Izgled stranice je sada prikazan na slici 4.2. Moramo ponovno pokrenuti server svaki put kad dodamo nove vlastite oznake predloška da bi se one mogli registrirati! Također, možemo stvarati i vlastite parametrizirane oznake predloška.

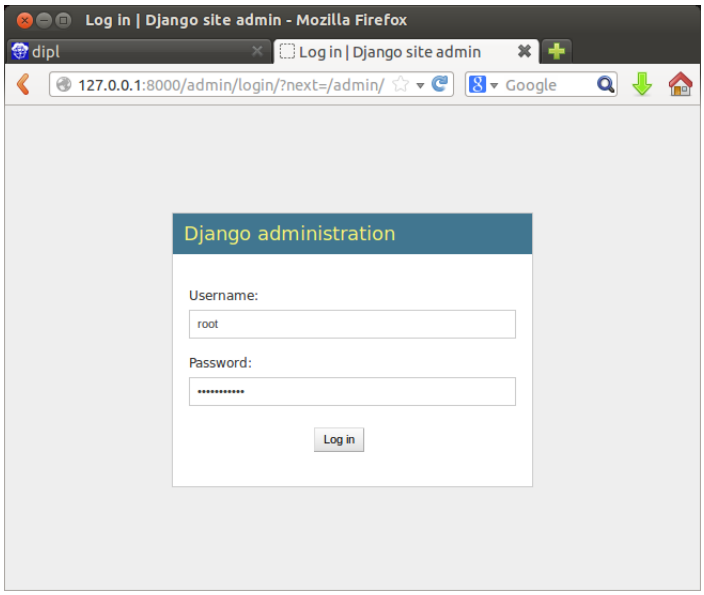
Slika 4.2: Vlastita oznaka predloška

Poglavlje 5

Ostale mogućnosti Djanga

5.1 Administracija

Django pruža jednostavno sučelje administracije koje omogućava pretraživanje i uređivanje podataka u bazi. Na slici 4.2 je prikazano sučelje administracije na stranici `http://127.0.0.1:8000/admin/`.



Slika 5.1: Sučelje administracije

Prijava je pomoću korisničkog imena i lozinke koje smo stvorili kad smo stvarali superkorisnika. U sučelje administracije je potrebno dodati modele da bismo ih na taj način mogli uređivati. Otvorimo *osnove/admin.py* i dodamo sljedeći kôd:

```
from django.contrib import admin
from osnove.models import Autor, Clanak

admin.site.register(Autor)
admin.site.register(Clanak)
```

Na taj način preko sučelja možemo dodavati retke u tablicama koje smo stvorili. Datoteka *admin.py* se može prilagoditi da se preko sučelja može mijenjati i model, tj. njegovi atributi ili tipovi.

5.2 Populacijska skripta

Populacijska skripta (eng. population script) je skripta koja pomaže u testiranju baze. Ona popunjava bazu umjesto nas testnim podacima. Da bismo stvorili populacijsku skriptu stvorimo novu datoteku *populate_article.py* u korijenskom direktoriju projekta i dodamo sljedeći kôd:

```
import os
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
    'primjer.settings')
import django
django.setup()
from osnove.models import Autor, Clanak

def populate():
    prvi_aut = add_auth(name="Marko Markovic",
        web="www.example.com/autor1",
        email = "autor1@gmail.com")
    drugi_aut = add_auth(name="Ivan Juric",
        web="www.example.com/autor2",
        email = "autor2@gmail.com")

    add_art(auth=[prvi_aut],
        title="Znanstveni clanak 1")
    add_art(auth=[prvi_aut, drugi_aut],
        title="Znanstveni clanak 2")
    add_art(auth=[drugi_aut],
        title="Znanstveni clanak 3")

    for c in Autor.objects.all():
        for p in Clanak.objects.filter(autor=c):
            print "- {0} - {1}".format(str(c), str(p))
```

```
26
27 def add_art(auth, title):
28     p = Clanak.objects.get_or_create(naslov=title)[0]
29     for a in auth:
30             p.autor.add(a)
31     return p
32
33 def add_auth(name, web, email):
34     c = Autor.objects.get_or_create(ime=name, web=web,
35         email=email)[0]
36     return c
37
38 if __name__ == '__main__':
39     print "Pokrenuta populacijska skripta..."
40     populate()
```

Populacijsku skriptu izvršavamo na sljedeći način:

```
nikolina@ubuntu:~/web/primjer$ python populate_clanci.py
Pokrenuta populacijska skripta...
- Marko Markovic - Znanstveni clanak 1
- Marko Markovic - Znanstveni clanak 2
- Ivan Juric - Znanstveni clanak 2
- Ivan Juric - Znanstveni clanak 3
```

Spremljene promjene u bazi možemo provjeriti tako da odemo na stranicu administracije. Tamo možemo i ručno promijeniti ili dodati podatke. Populacijska skripta je najjednostavniji način testiranja kôda i toplo preporučamo njeno korištenje.

5.3 Slug polje

Želimo napraviti da svaki autor ima svoju stranicu, tj. da se na njoj prikazuju detalji tog autora. Prvo treba postaviti URL-ove za te stranice. To ćemo raditi intuitivno, npr. ako netko izabere autora Marko Marković tada će URL biti „osnove/autori/Marko Marković".

No, to nije ispravan URL pa stvaramo čisti URL (eng. clean URL) na način da dodamo slug polje u model `Autor`: prvo uvezemo funkciju `slugify` iz Djanga koja zamjenjuje prazna mjesta s povlakama, velika slova s malima te sva slova pretvara u slova engleske abecede (npr. ako imamo autora Marko Marković koji ima više riječi, u URL-u će funkcija to zamijeniti u marko-markovic). Zatim redefiniramo postojeću `save()` metodu u modelu `Autor` koja poziva funkciju `slugify` i popunjava polje `slug` pri spremanju instance modela. Ovako izgleda prerađeni model:

```
1 from django.template.defaultfilters import slugify
2 class Autor(models.Model):
```

```
3       ime = models.CharField(max_length=128, unique=True)
4       web = models.URLField()
5       email = models.EmailField()
6       slug = models.SlugField()
7       def save(self, *args, **kwargs):
8           self.slug = slugify(self.ime)
9           super(Autor, self).save(*args, **kwargs)
10
11      def __unicode__(self):
12          return self.ime
```

Ne zaboravimo pokrenuti naredbe za promjenu tablica u bazi koristeći migracijske alate! Nakon što imamo definirane URL-ove za određenog autora potrebno je dodati funkciju pogleda koja će za odabranog autora pronaći sve detalje koje želimo prikazati o autoru te stvoriti predložak koji će prikazivati web-stranicu o detaljima autora. Ovako izgleda pripadna funkcija pogleda koja prima parametar `autor_slug` pomoću kojeg stvaramo upit bazi podataka:

```
1  def autor(request, autor_slug):
2      try:
3          autor = Autor.objects.get(slug=autor_slug)
4
5      except Autor.DoesNotExist:
6          pass
7
8      return render(request, 'osnove/autor.html',
9          {'autor': autor})
```

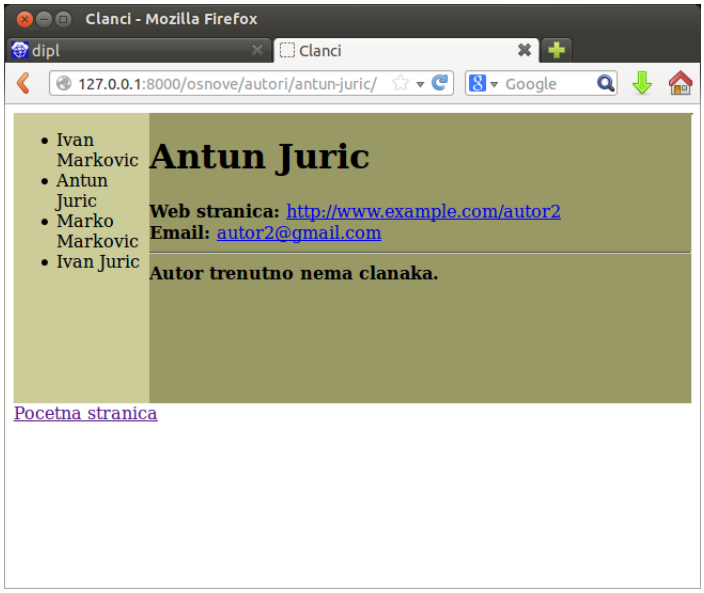
Zatim treba dodati URL na kojem će se prikazati stranica o detaljima autora s parametrom `author_slug` koji prosljeđujemo funkciji pogleda:

```
1  url(r'^autori/(?P<autor_slug>[\w\-]+)/$', views.autor,
2      name='autor'),
```

Ovaj URL prihvaća sve vrijednosti varijable `autor_slug` koje se sastoje od barem jednog slova `a-z`, `A-Z`, znamenke `0-1` ili znaka `-`.

Potrebno je ponovno pokrenuti populacijsku skriptu da bi se popunilo `slug` polje za do sada dodane autore. Ako želimo da se u sučelju administracije `slug` polje automatski prilagođava kako mijenjate ime i prezime autora tada trebamo u datoteci *admin.py* prethodni kôd zamijeniti sljedećim linijama:

```
1  class AutorAdmin(admin.ModelAdmin):
2      prepopulated_fields = {'slug':('ime',)}
3  admin.site.register(Autor, AutorAdmin)
```



Slika 5.2: Stranica određenog autora

Na ovaj način možemo stvarati zasebne stranice o svakom podatku iz baze bez da pišemo svaki URL posebno, odnosno da se to radi automatski i logički. Pogledajmo sada kako izgleda naša stranica na slici 5.2.

5.4 Forme

Web-forme sakupljaju informacije od korisnika i šalju povratne informacije natrag njima. One omogućavaju prikaz HTML forme, kao što su `textfield` ili `datepicker`, provjeru poslane informacije po zadanim pravilima, ponovni prikaz forme u slučaju validacijskih grešaka te pretvorbu poslanih podataka u relevantni Python tip. Da bismo radili s formama najprije je potrebno stvoriti datoteku *forms.py* u direktoriju aplikacije u koje će se spremati klase vezane uz forme. Nužno je stvoriti `ModelForm` za svaki model koji želimo prikazati kao formu, prilagoditi formu po želji, prilagoditi pogled koji će raditi s formom (prikazivanje forme, spremanje podataka iz forme, označavanje grešaka), prilagoditi predložak koji će prikazivati formu i dodati URL za pogled ako smo stvorili novi pogled. Stvaranje nove datoteke *forms.py* nije potrebno, sav kôd se može nalaziti u datoteci *models.py*, ali ovo čini kôd čišćim i lakšim.

Klasa `ModelForm` pomaže stvaranju forme od već postojećeg modela. Slijedi primjer stvaranja forme za naš model koja će korisniku služiti za dodavanje novog autora.

```
1 from django import forms
```

```
2  from osnove.models import Autor, Clanak
3
4  class AutorForm(forms.ModelForm):
5      ime = forms.CharField(max_length=128,
6          help_text="Unesite ime autora")
7      web = forms.URLField(max_length=200,
8          help_text="Unesite stranicu autora", required=False)
9      email = forms.EmailField(help_text="Unesite email autora",
10         required=True)
11
12     class Meta:
13         model = Autor
14         fields = ('ime', 'web', 'email',)
```

Vidimo da klasa sadrži ugniježđenu klasu za povezivanje s postojećim modelom i opisuje što će se prikazati u polju, a što ne. Definiramo vrste forme koja će se prikazati, npr. dopušta samo unos teksta ili znamenaka te da li je vidljiva i da li je korisnik može mijenjati. Ako za neki atribut modela ne navedemo vrstu i svojstva, Django će svejedno stvoriti formu sa svojstvima koja su zadana za taj tip atributa. Nakon toga potrebno je preraditi datoteku *views.py* za rad s formama na način da dodamo novu funkciju pogleda.

```
1  from osnove.forms import AutorForm
2  from django.http import HttpResponseRedirect
3  from django.core.urlresolvers import reverse
4  def dodaj_autora(request):
5      if request.method == 'POST':
6          form = AutorForm(request.POST)
7
8          if form.is_valid():
9              form.save(commit=True)
10
11             return HttpResponseRedirect(reverse('uvod'))
12         else:
13             print form.errors
14     else:
15         form = AutorForm()
16     return render(request, 'osnove/dodaj_autora.html',
17         {'form': form})
```

U kôdu se provjerava da li je metoda zahtjeva POST. Ukoliko želimo prikazati formu tada je metoda GET, a ukoliko želimo dohvatiti podatke iz forme tada je metoda POST. Django automatski provjerava da li su podaci valjani, odnosno da li odgovaraju tipu postavljenom u klasi AutorForm. Ukoliko postoji greška Django će ih ispisati. Ako su svi podaci točni

spremaju se u bazu podataka funkcijom `save()`, a zatim, koristeći funkciju `reverse()`, korisnik se preusmjerava na početnu stranicu. Zatim je potrebno napraviti novi predložak za povezivanje s funkcijom pogleda.

```
1  {% extends 'osnove/base.html' %}
2
3  {% block tijelo %}
4          <form id="autor_form" method="post" action="/osnove/
               dodaj_autora/">
5
6              {% csrf_token %}
7              {% for hidden in form.hidden_fields %}
8                  {{ hidden }}
9              {% endfor %}
10
11             {% for field in form.visible_fields %}
12                 {{ field.errors }}
13                 {{ field.help_text }} </br>
14                 {{ field }}
15             <br /><br />
16             {% endfor %}
17             <input type="submit" name="submit" value="Stvori
                   autora" />
18         </form>
19 {% endblock %}
```

Vidimo da je metoda forme POST i da se svi podaci prosljeđuju URL-u *osnove/dodaj_autora*. Također vidimo novu oznaku {% csrf_token %}. To je Cross-site request forgery token koji pomaže osigurati HTTP POST akciju. On je nužno potreban u Django okruženju, inače bi korisnik pri korištenju forme mogao dobiti greške te spriječava napad na stranicu.

Unutar forme imamo dvije petlje, jedna prolazi skrivenim poljima, a druga vidljivim poljima. Potreba za vidljivim poljima je jasna. No skrivena polja su također bitna jer je HTTP protokol koji ne pamti nikakve podatke o transakcijama (eng. stateless protocol) i zbog toga ne podržava različite zahtjeve pa je neke dijelove teško implementirati. Da bi zahtjevi bili isti stvorena su skrivena polja koja se prosljeđuju klijentu i kasnije vraćaju nazad serveru. Ona naravno nisu vidljiva klijentu. Nakon toga mapiramo URL sa novom funkcijom pogleda i vidimo forme koje smo zadali na slici 5.3. Detaljni opis formi, vrsta i svojstava nalazi se na [1].

Slika 5.3: Korištenje formi za dodavanje autora

5.5 Registracija i prijava

Na većini web-stranica je danas potrebno registriranje, odnosno prijava (eng. register i login). Django u sebi ima ugrađen mehanizam za prijavu. Za to ćemo koristiti `auth` aplikaciju iz paketa `django.contrib.auth`. Potrebno je provjeriti da li se u *settings.py* nalazi `django.contrib.auth` i `django.contrib.contenttypes`. Ako ne, potrebno ih je dodati u `INSTALLED_APPS`. Lozinke kod prijave se spremaju PBKDF2 algoritmom, što povećava njihovu sigurnost. Opcionalno mogu se zadati hash funkcije da bismo povećali razinu sigurnosti. Jednostavno dodamo tuple stringova `PASSWORD_HASHERS` u kojemu navedemo funkcije koje želimo koristiti. Django će uvijek koristiti prvu navedenu funkciju, no ukoliko želimo da ih koristi sve potrebno je instalirati `Bcrypt`. Ako ne navedemo hash funciju Django će koristiti zadanu funkciju `PBKDF2PasswordHasher`.

Osnovni objekt koji ćemo koristiti u sustavu prijave je `User` i nalazi se u biblioteci `django.contrib.auth.models.User`. Jedna instanca objekta `User` predstavlja jednu osobu koja pristupa web-stranici. Sadrži pet primarnih atributa: `username`, `password`, `email_address`, `first_name`, `surname` te druge atribute kao `is_active`. Popis ostalih atributa nalazi se u službenoj dokumetaciji Djanga na [1]. Ukoliko želimo staviti dodatne atribute (koji nisu pruženi u `User` modelu) moramo stvoriti novi model povezan s `User`. Zamislimo da svakom korisniku želimo dodati `URLField`, atribut koji će sadržavati web-stranicu korisnika. Dodat ćemo novi model koji izgleda ovako:

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

```
2
3 class UserProfile(models.Model):
4     user = models.OneToOneField(User)
5     website = models.URLField(blank=True)
6
7     def __unicode__(self):
8         return self.user.username
```

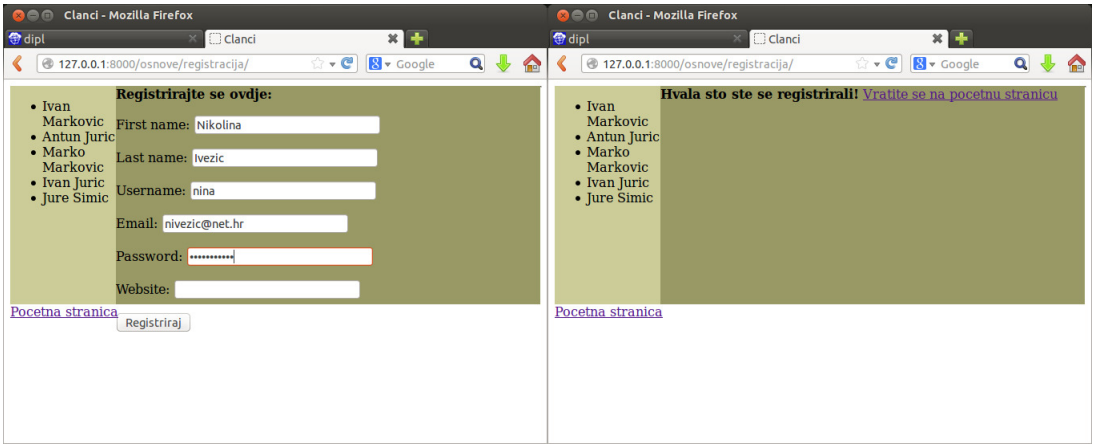
Najprije je potrebno uvesti model `User` da bismo ga mogli koristiti. Uočimo da prvo povezujemo novostvoreni model sa postojećim, `User`, vezom jedan prema jedan. Stvaranje novog modela tako da naslijedimo model `User` se ne preporuča jer je moguće da će druge aplikacije također morati koristiti model `User`. Zatim je potrebno promijeniti datoteku *admin.py* da bismo novostvoreni model uključili u sučelje administracije. Ne smijemo zaboraviti ažurirati bazu podataka nakon stvaranja svakog novog modela!

Sada smo spremni promijeniti našu web-stranicu kako bi omogućila registriranje novim korisnicima. To ćemo, naravno, napraviti koristeći novi pogled i predložak. Prvo ćemo stvoriti dvije nove forme za modele `User` i `UserProfile` u datoteci *forms.py*. Stvaramo novu funkciju pogleda koja će provjeriti jesu li unešeni podaci valjani. Moramo stvoriti predložak koji će prikazati naš pogled i, na kraju, mapirati željeni URL.

```
1  from django.core.urlresolvers import reverse
2  from django.contrib.auth import authenticate, login
3  from osnove.forms import UserForm, UserProfileForm
4
5  def registracija(request):
6      registered = False
7      if request.method == 'POST':
8          user_form = UserForm(data=request.POST)
9          profile_form = UserProfileForm(data=request.POST)
10         if user_form.is_valid() and profile_form.is_valid():
11             user = user_form.save()
12             user.set_password(user.password)
13             user.save()
14             profile = profile_form.save(commit=False)
15             profile.user = user
16             profile.save()
17             registered = True
18         else:
19             print user_form.errors, profile_form.errors
20     else:
21         user_form = UserForm()
22         profile_form = UserProfileForm()
```

```
23      return render(request, 'osnove/registracija.html',
24              {'user_form': user_form, 'profile_form':
25              profile_form, 'registered': registered} )
```

Slika 5.4 prikazuje stranice za registraciju.



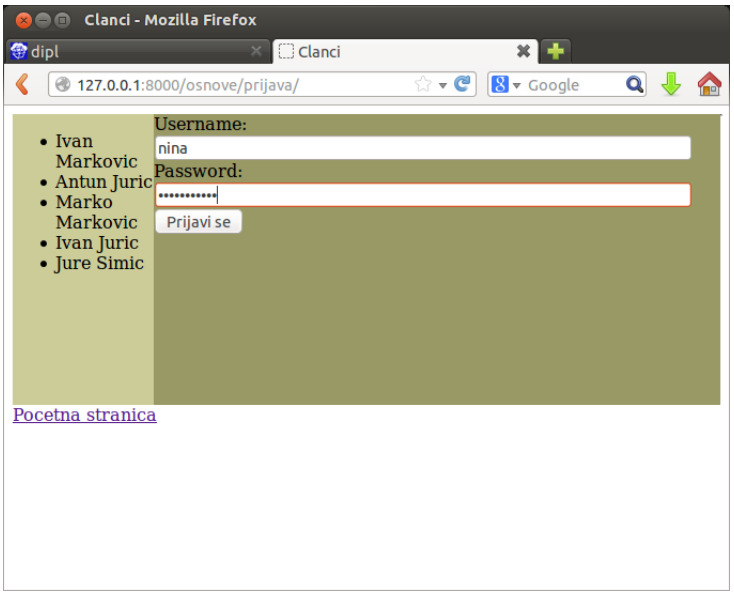
Slika 5.4: Registracija

Nakon što se korisnik uspješno registrira, stranica mora imati mogućnost prijave i odjave. Sada stvaramo novu funkciju pogleda te novi predložak za prijavu i odjavu korisnika. U Djangu postoji funkcija `logout()` koja se brine za odjavu korisnika i, ako korisnik želi pristupiti pogledu kojem ne smije, odbija mu pristup.

```
1  from django.contrib.auth import logout
2
3  def prijava(request):
4      if request.method == 'POST':
5          username = request.POST.get('username')
6          password = request.POST.get('password')
7          user = authenticate(username=username,
8              password=password)
9          if user:
10             if user.is_active:
11                 login(request, user)
12                 return HttpResponseRedirect(reverse('uvod'))
13             else:
14                 return HttpResponse("Deaktiviran racun!")
15         else:
16             print "Invalid login details:
17                 {0}, {1}".format(username, password)
```

```
18                return HttpResponse("Netocni podaci.")
19        else:
20            return render(request, 'osnove/prijava.html')
21
22    def odjava(request):
23        logout(request)
24        return HttpResponseRedirect(reverse('uvod'))
```

Nakon što dodamo URL za prijavu na slici 5.5 možemo vidjeti kako izgleda sustav za prijavu na našoj stranici.



Slika 5.5: Prijava

Ograničavanje pristupa

Nakon što smo stvorili stranice za registraciju i prijavu korisnika potrebno je određenim korisnicima ograničiti pristup, tj. ukoliko korisnik nije prijavljen, ne smije moći pristupiti svim stranicama. U Djangu postoje dva načina kako ograničiti pristup. Prvi je direktno ispitivanjem `request` objekta i provjerom je li korisnik valjano prijavljen naredbom

```
1 if request.user.is_authenticated():
```

Drugi način je koristeći Pythonov decorator. Decorator koji ćemo mi koristiti je `login_required()` koji samo dodamo na bilo koji pogled za koji želimo da korisnik bude prijavljen. Da bismo mogli koristiti Pythonov decorator potrebno ga je uvesti, a nalazi

se u biblioteci `django.contrib.auth.decorators`. Sada ćemo ograničiti neprijavljenog korisnika da ne može dodati autora.

```
from django.contrib.auth.decorators import login_required

@login_required
def dodaj_autora(request):
    if request.method == 'POST':
        form = AutorForm(request.POST)

        if form.is_valid():
            form.save(commit=True)

            return HttpResponseRedirect(reverse('uvod'))
        else:
            print form.errors
    else:
        form = AutorForm()
    return render(request, 'osnove/dodaj_autora.html',
        {'form': form})
```

5.6 JQuery

JQuery je posebna vrsta JavaScript biblioteke s namjenom da bude nadogradnja osnovnog JavaScript-a. JQuery pojednostavljuje njegovu sintaksu i omogućava bolju interakciju između JavaScript-a i drugih programskih jezika namjenjenih razvoju web-stranica. Vrlo jednostavno se može uklopiti u Django aplikaciju.

JQuery datoteku spremamo u direktorij *static* zajedno s ostalim statičnim medijima. Da bismo mogli koristi JQuery potrebno je skinuti verziju JQuery biblioteke ili je eksplicitno uključiti u projekt (u *base.html*) sljedećom linijom kôda:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery
   /1.11.1/jquery.min.js" />
<script src="{% static "js/osnove-jquery.js" %}" />
```

Primjer JQuery datoteke vidimo ispod:

```
$(document).ready( function() {
    $("#lista1").click( function(event) {
        $("#ul1").toggle();
        $( "ul.sakrivanje" ).not("#ul1").hide();
    });
});
```

5.7 Prednosti Djanga

Sad kad smo naučili osnove korištenja Djanga i njegove mogućnosti vrijeme je da kažemo nešto o prednostima Djanga u odnosu na prijašnji način razvoja web-stranica. Kada je predstavljena MTV arhitektura dizajneri su bili oduševljeni. Mogli su dizajnirati izgled stranice neovisno o programerima s obzirom da su ti dijelovi odvojeni. Nisu morali brinuti niti o pohrani podataka, niti o upravljanju njima. Mogli su se fokusirati na ono što najbolje rade, uređivanje izgleda. S druge strane, to je i programerima olakšalo posao jer mogu razmišljati samo o kôdu i podacima, a ne o prezentaciji tih podataka.

Uz to je uvedeno programiranje u objektno orijentiranom jeziku Python koji se dobro uklapa u razvoj web-stranica. Koristi čistu i elegantnu sintaksu te sadrži veliku biblioteku paketa koji pokrivaju mnoge funkcionalnosti, od višedretvenosti do sažimanja datoteka. Također podržava rad s različitim web-serverima i bazama podataka, brz je i stabilan te daje dobre performanse.

5.8 Sigurnost Djanga

Django dosta brine o sigurnosti web-stranica. Pod sigurnost mislimo na sprječavanje napada na stranicu od strane zlonamjernih korisnika. U nastavku navodimo najčešće napade na web-stranice i način kako je Django smanjio tu mogućnost. Više o poboljšanju sigurnosti možete pročitati u [8].

XSS

Cross Site Scripting najraširenija je sigurnosni propust na web-stranicamaa, a događa se kad stranica prihvati neprovjerene podatke i pošalje ih pregledniku. To napadačima omogućava da izvršavaju skripte u pregledniku žrtve kada ona posjeti stranicu, čime može preuzeti kontrolu nad korisničkom sesijom, vandalizirati stranicu ili preusmjeriti korisnika na zloćudne stranice. Najčešće se postiže spremanjem zlonamjerne skripte u bazu podataka, odakle će biti prikazana drugim korisnicima, ili navođenjem korisnika na klik linka koji će pokrenuti napadačku skriptu. No, XSS napadi mogu potjecati od bilo kakvog neovlaštenog izvora podataka kao što su kolačići ili web-usluge, kad god podaci nisu provjereni prije uključivanja u stranicu. U ranim danima razvoja Djanga ovakav propust se rješavao na način da se svaka nepouzdana varijabla puštala kroz `escape` filter. On pretvara opasne znakove u bezopasne. No, to je prisiljavalo programera da uz svaku varijablu doda filter, a u velikim projektima se lako moglo dogoditi da neki promakne. U novije vrijeme je uveden Djangov automatski filter, `autoescape`. Zadano je da svaki predložak ima uključenu `autoescape` oznaku. Stoga, korištenjem Djangovog sustava predložaka, zaštićeni smo od većine XSS napada.

CSRF

Cross-site request forgery napad krade autentifikacijske informacije koje se koriste za prijavu na ranjivu stranicu. Kad se to izvrši, napadač može preuzeti kontrolu nad žrtvinom sesijom. Neka je korisnik prijavljen na našu stranicu i šalje nam podatke u formi. Dok je još prijavljen, korisnik ode na zaraženu stranicu koja također nudi formu „normalnog" izgleda korisniku. Ta stranica tada šalje podatke našoj stanici. Međutim, kako naša stranica vjeruje da je prijavljen legitimni korisnik, jako je teško otkriti kad je ovakav napad uspio. Django ima ugrađena rješenja zaštite od CSRF oblika napada. Zbog toga smo u predloške uključivali `csrf_token` koji generira jedinstvenu slučajnu vrijednost u kolačiću i formi. Nakon slanja podataka Django provjerava da li su te dvije vrijednosti jednake. Zaražena stranica ne može doći do vrijednosti iz kolačića pa u slučaju napada, vrijednosti neće biti jednake. No zaštita ima i svoja ograničenja jer je moguće onemogućiti CSRF modul za cijelu aplikaciju ili za određene poglede. Django koristi klasu `django.middleware.csrf.CsrfViewMiddleware` uključenu u `MIDDLEWARE_CLASSES` varijablu u *settings.py* datoteci koja onemogućuje CSRF napade.

SQL injection

SQL Injection je tehnika umetanja kôda koja unosi zloćudne SQL upite i naredbe u polje unosa za izvršavanje nad bazom podataka što dovodi do toga da web-server šalje, odnosno vraća informaciju koju ne bi smio vratiti. Kao rezultat, web-server omogućava pristup informacijama koje bi trebale biti sigurne i izvan dohvata. Na primjer, takvi su podaci korisnička imena i lozinke. Rezultat toga može biti brisanje ili curenje podataka. Django u sebi ima ugrađen mehanizam za izbjegavanje kôda s obzirom da se ne koriste direktni SQL upiti nego ugrađene metode klase `Model` koje spriječavaju takav način napada.

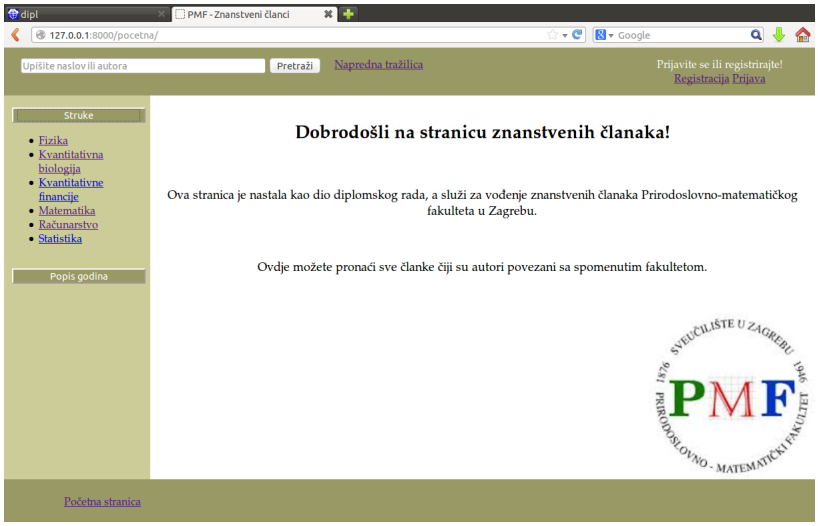
Clickjacking

Clickjacking je vrsta napada gdje zaražena stranica zapakira drugu stranicu u okvir. Rezultat ovakvog napada je da korisnik prevarom radi nenamjerne akcije na ciljnoj stranici. Django omogućava zaštitu od Clickjackinga tako što u varijabli `MIDDLEWARE_CLASSES` ima klasu `django.middleware.clickjacking.XFrameOptionsMiddleware` koja sprječava da se stranica proslijedi u okvir. Moguće je onemogućiti zaštitu u određenim pogledima, no to se ne preporuča.

Poglavlje 6

Web-aplikacija za vođenje znanstvenih članaka

Dio ovog diplomskog rada je web-aplikacija za vođenje znanstvenih članaka. Ona u potpunosti slijedi MTV arhitekturu te koristi sve mogućnosti Djanga koje su do sada opisane. U nastavku će pobliže biti objašnjeno što web-aplikacija omogućuje te kako je to realizirano.



Slika 6.1: Početna stranica

Projekt se sastoji od dvije aplikacije *osnove* i *prijava*. Obje nasljeđuju bazni predložak *base.html* u kojem je definiran kostur i dijelovi svih stranica. Stranica sadrži

zaglavlje koje sadrži tražilicu, link na naprednu tražilicu te linkove za registraciju i prijavu

izbornik koji je smješten na lijevoj strani, sadrži dva gumba „Struke" i „Popis godina", klikom na gumb otvara se popis struka, odnosno godina

podnožje koje sadrži linkove na sve stranice kojima korisnik može pristupiti kao što su početna stranica ili dodavanja članaka

glavni dio koji je predviđen za ispis rezultata traženja i detalja pojedinih članaka, ovisno o pripadnoj funkciji pogleda.

Prikaz početne stranice te izgled baznog predloška vidimo na slici 6.1

6.1 Aplikacija *osnove*

Aplikacija *osnove* je središnji dio projekta. U njoj su definirana četiri modela: `Autor`, `Kategorija`, `Struka` i `Clanak`. Model `Autor` opisuje jednog autora članka i sastoji se od sljedećih atributa:

`ime` - ime i prezime autora

`web` - osobna web-stranica autora, nije obavezan

`email` - email autora

`slug` - atribut objašnjen u odjeljku 5.3.

`Struka` se sastoji od atributa `naziv` i predstavlja jednu struku kojoj članak može pripadati (npr. fizika, matematika). `Kategorija` predstavlja kategoriju u koju članak može biti svrstan (npr. Numerička analiza, Strojno učenje). Povezan je modelom `Struka` vezom jedan prema mnogo. `Clanak` predstavlja jedan članak i u njemu se nalaze osnovni podaci o članku:

`autori` - autori članka

`kategorije` - kategorije kojima članak pripada

`dodao` - korisnik koji je dodao članak

`vrijeme` - datum i vrijeme kada je članak dodan

`naslov` - naslov članka

`detalji` - opis članka

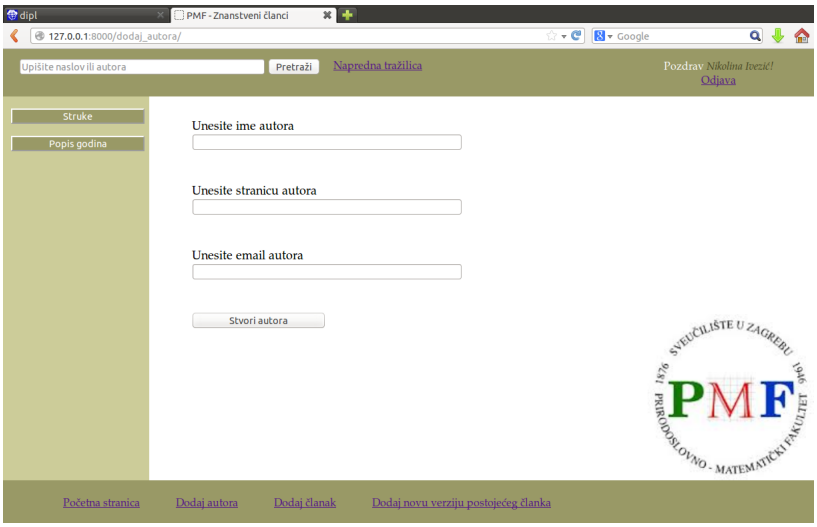
`kljucne_rijeci` - ključne riječi članka

`verzija` - verzija članka

`godina` - godina nastanka članka

`pdf` - apsolutni put do PDF-a članka unutar direktorija *media*.

Povezan je s modelima `Autor` i `Kategorija` vezom mnogo prema mnogo te modelom `User` vezom jedan prema mnogo.

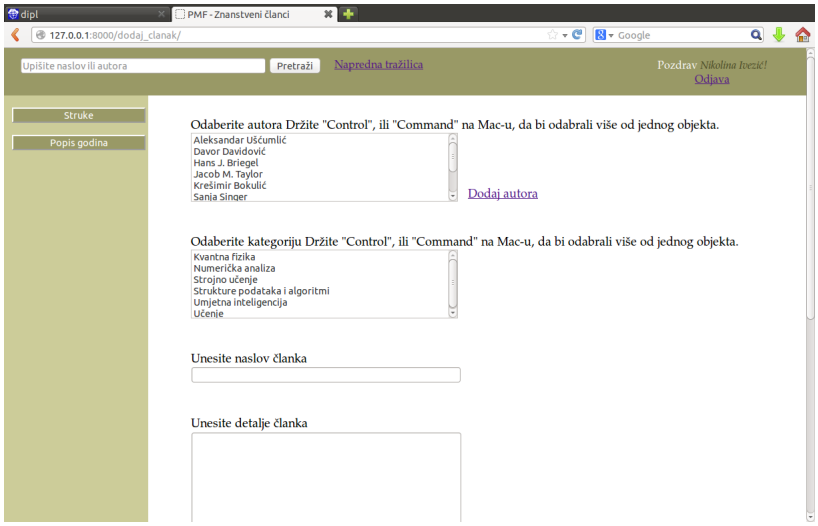


Slika 6.2: Forma za dodavanja autora

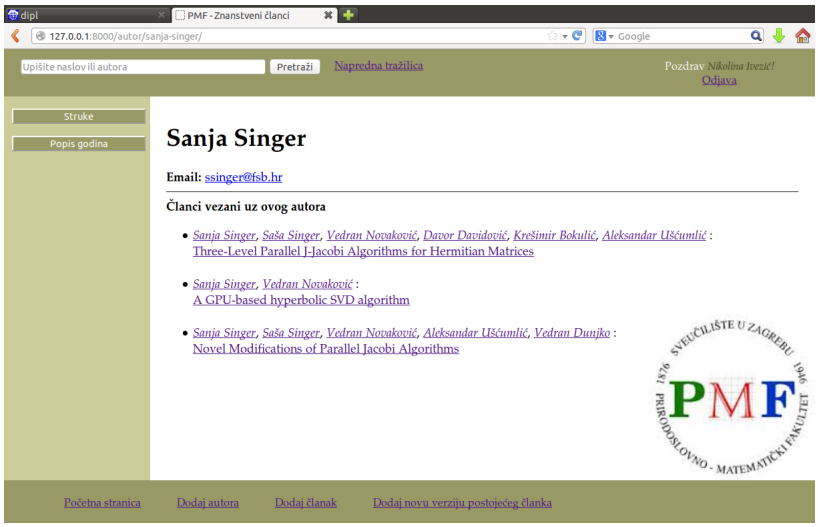
Datoteka *forms.py* se sastoji od dvije forme `AutorForm` i `ClanakForm` koje služe za upisivanje podataka o novom autoru ili članku, prikaz je na slikama 6.2 i 6.3.

Datoteka *views.py* se sastoji od nekoliko funkcija i u nastavku opisujemo njihovu ulogu. Funkcija `pocetna()` služi za prikaz početne stranice s opisom same aplikacije. Sljedeće fukcije primaju parametar `slug`, pronalaze traženu instancu u bazi podataka (npr. funkcija `clanak()` prima parametar `clanak_slug` te u bazi traži dotični članak) i služe za prikaz detalja o toj instanci:

- `autor()`
- `clanak()`
- `kategorija()`
- `struka()`
- `godina()`.



Slika 6.3: Forma za dodavanja članka

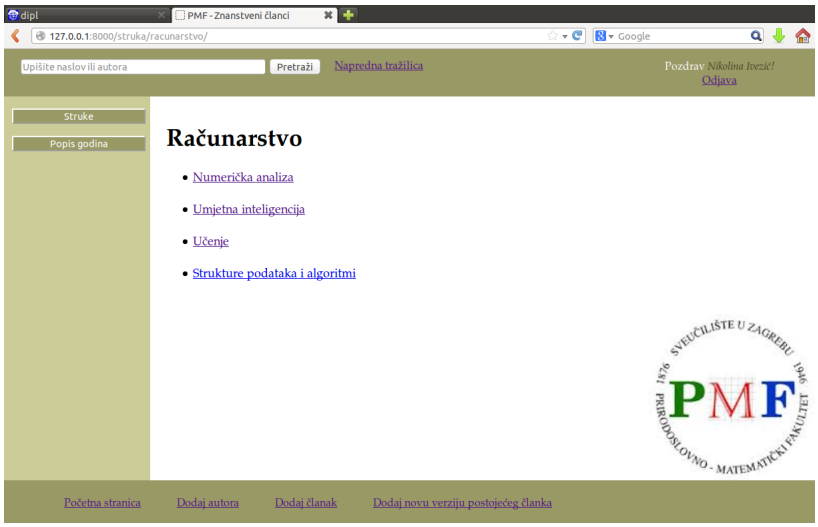


Slika 6.4: Prikaz detalja o autoru

Detalji o autoru podrazumijevaju

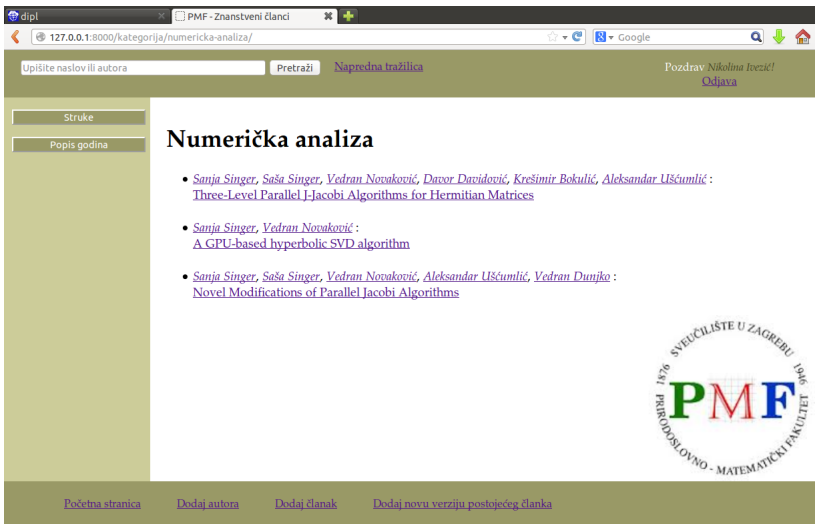
- ime
- web-stranicu (ako postoji)
- email autora
- sve članke koje je on napisao.

Prikaz detalja o autoru je na slici 6.4.



Slika 6.5: Prikaz detalja o struci

Detalji o struci su kategorije koje su s njom povezane. Prikaz je na slici 6.5. Detalji o kategoriji podrazumijevaju članke koji pripadaju toj kategoriji. Prikaz je na slici 6.6.



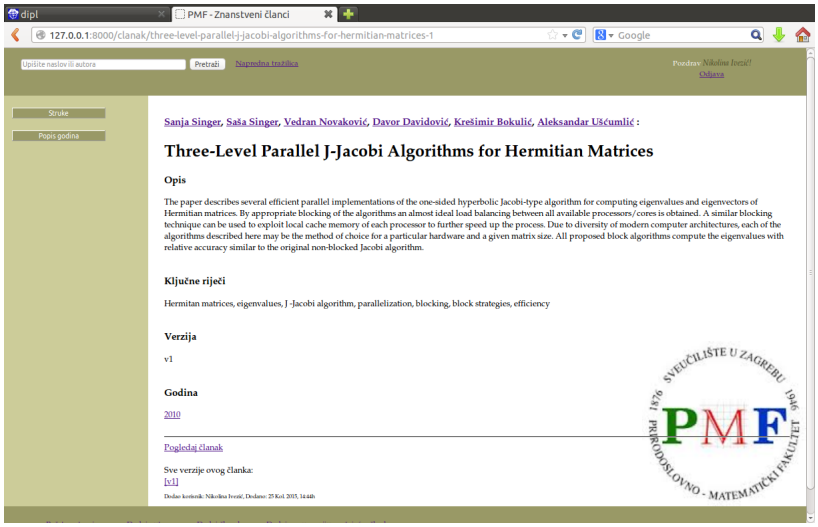
Slika 6.6: Prikaz detalja o kategoriji

Detalji o članku su:

- autori članka

- naslov članka
- opis članka
- godina nastanka članka
- ključne riječi
- verzija članka
- link na članak u PDF formatu
- linkovi na sve verzije tog članka
- korisnik koji je dodao članak i vrijeme dodavanja članka.

Prikaz detalja o članku nalazi se na slici 6.7.

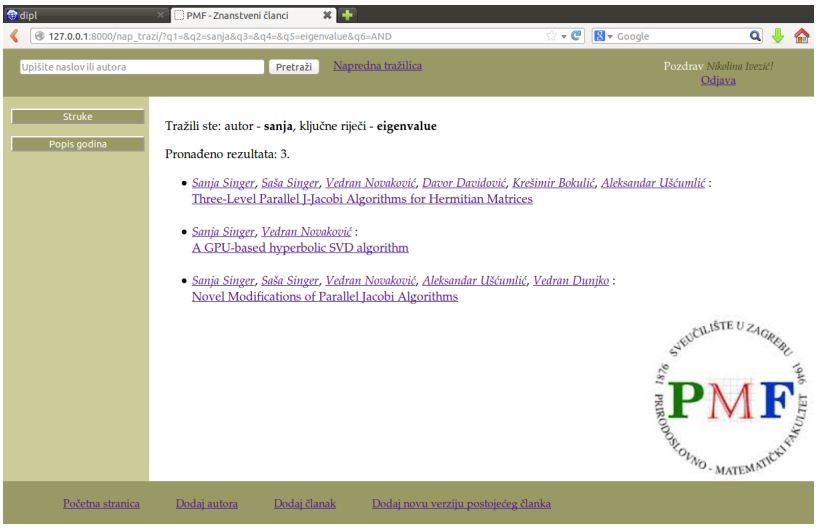


Slika 6.7: Prikaz detalja o članku

Pod detaljima o godini se podrazumijevaju svi članci nastali u dotičnoj godini.

Funkcije `dodaj_autora()`, `dodaj_clanak` i `dodaj_verziju` zahtijevaju prijavljenog korisnika. One dohvaćaju podatke iz forme, provjeravaju njihovu valjanost te ih spremaju u bazu ili ispisuju greške. Funkcija `dodaj_verziju` sprema već postojeći članak s novom verzijom u bazu podataka. Verzija se automatski povećava za 1. Samo korisnik koji je dodao članak može dodati nove verzije u bazu. Nove struke i kategorije može dodati samo superkorisnik.

Funkcija `trazi` dohvaća izraz koji je korisnik upisao u tražilicu koja se nalazi u zaglavlju web-stranice. Ona pretražuje bazu podataka prema imenu autora i naslovu članka. Funkcija `nap_trazi` je povezana s naprednom tražilicom. U njoj korisnik može pretraživati prema imenu autora, nazivu članka, struci, kategoriji i/ili ključnim riječima. Funkcija dohvaća podatke iz forme te pretražuje bazu obzirom na korisnikov upit. Prikaz naprednog pretraživanja je na slici 6.8.



Slika 6.8: Rezultat naprednog pretraživanja

Funkcija `pokazi` prima parametar `clan_slug`, pronalazi traženi članak u bazi te u pregledniku prikazuje PDF članka.

Svaka funkcija pogleda ima nekoliko linija kôda, a veliku funkcionalnost. Sada vidimo koliko je razvoj stranica u Djangu brz i jednostavan.

6.2 Aplikacija *prijava*

Aplikacija *prijava* služi za registriranje i prijavu korisnika. Obzirom da je zasebna aplikacija može se ukolopiti u neki drugi projekt u Djangu ili zamijeniti drugom aplikacijom koja može pružiti više funkcionalnosti.

Korisnik je predstavljen modelom `User` na način koji je opisan u odjeljku 5.5. Dakle, korisnik se može registrirati sa svojim osobnim podacima, prijaviti nakon registracije i odjaviti.

Prijavljeni korisnik može dodavati autora ili članak te nove verzija članka koje je već dodao. Neprijavljeni korisnik može samo pretraživati i čitati članke.

Bibliografija

[1] *Django*, `http://www.djangoproject.com/`, kolovoz 2015.

[2] *MySQL*, `https://www.mysql.com/`, kolovoz 2015.

[3] *Oracle Help Center*, `http://docs.oracle.com/en/`, kolovoz 2015.

[4] *PostgreSQL*, `http://www.postgresql.org/`, kolovoz 2015.

[5] *Python*, `http://www.python.org/`, kolovoz 2015.

[6] *SQLite*, `https://www.sqlite.org/`, kolovoz 2015.

[7] *W3Schools*, `http://www.w3schools.com/cssref/`, kolovoz 2015.

[8] D. Greenfeld i A. Roy, *Two Scoops of Django: Best Practices for Django 1.6*, Two Scoops Press, Corona, 2014.

Sažetak

U ovom radu je predstavljen Django, web-razvojno okruženje iz treće generacije razvoja dinamičnih web-stranica, prvi put objavljen 2005. godine. Osnovna prednost Djanga je korištenje MTV-a, arhitekture kojom se razdvaja prikaz, pristup podacima i logika web-stranice. MTV arhitektura dijeli jednu stranicu u tri dijela: model - zadužen za komunikaciju s bazom podataka, predložak - zadužen za prezentiranje podataka, pogled - zadužen za manipulaciju podacima i prosljeđivanje podataka predlošku. To omogućuje programerima i dizajnerima jednostavniji i brži razvoj web-stranica te povećanje sigurnosti same stranice. Također, omogućava jednostavno korištenje formi i dodavanja svih vrsta statičnih medija. Django aplikacija je napisana u jeziku Python. Danas se Djangom koristi veliki broj korisnika, a predviđa se da će treća generacija kroz neko vrijeme potpuno potisnuti PHP iz uporabe.

Summary

In this paper we introduced Django, a web framework from the third generation of web development. It is released in 2005. Django uses the MTV architecture that separates user interface, data access and request-routing logic. The Django web-page is containing three components: model, which can create, retrieve, update and delete records in the database, template, which describes the design of the page, and view, which is responsible for data manipulation and data forwarding to the template. This allows developers and designers easier and faster development of web-sites, and increases security of the site itself. It also allows simple use of forms and adding all types of static media files. Django project is written in Python. Large number of users use Django, and it is anticipated that the third generation will repress the use of PHP completely.

Životopis

Osobni podaci:

- Prezime i ime: Ivezić Nikolina
- Datum rođenja: 15. prosinca 1991.
- Mjesto rođenja: Požega, Republika Hrvatska
- Državljanstvo: hrvatsko
- Narodnost: Hrvatica

Obrazovanje:

- 2013.-2015. Sveučilište u Zagrebu, Prirodoslovno–matematički fakultet, Zagreb, Republika Hrvatska (diplomski studij - Računarstvo i matematika)
- 2010.-2013. Sveučilište u Zagrebu, Prirodoslovno–matematički fakultet, Zagreb, Republika Hrvatska (preddiplomski studij - Matematika)
- 2006.-2010. Opća gimnazija, Požega, Republika Hrvatska
- 1998.-2006. Osnovna škola Julija Kempfa, Požega, Republika Hrvatska

Zvanje:

- srpanj 2010. „Sveučilišni prvostupnik matematike”, Sveučilište u Zagrebu, Republika Hrvatska

Strani jezici:

- engleski (razumijevanje: odlično; govor i pisanje: vrlo dobro)
- njemački (osnovno)
- talijanski (osnovno)