UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

DAVID FRLIC

Razvoj spletnih aplikacij z odprtokodnim ogrodjem

DIPLOMSKO DELO NA
VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00319/2012

Datum: 03.09.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DAVID FRLIC**

Naslov: **RAZVOJ SPLETNIH APLIKACIJ Z ODPRTOKODNIM OGRODJEM**

DEVELOPMENT OF WEB APPLICATIONS WITH OPEN SOURCE FRAMEWORK

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Proučite odprtokodno ogrodje DJANGO, ki je ogrodje za razvoj spletnih aplikacij. Z uporabo ogrodja DJANGO razvijte spletno aplikacijo in na podlagi izkušenj pri razvoju izdelajte primerjavo med klasičinm razvojem spletnih aplikacij in razvojem spletnih aplikacij z uporabo ogrodja.

Mentor:

doc. dr. Rok Rupnik

Dekan:

prof. dr. Nikolaj Zimic

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

I Z J A V A   O   A V T O R S T V U

diplomskega dela

Spodaj podpisani David Frlic,

z vpisno številko 63070270,

sem avtor diplomskega dela z naslovom:

Razvoj spletnih aplikacij z odprtokodnim ogrodjem

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  doc. dr. Roka Rupnika,

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 17.09.2012 Podpis avtorja:

ZAHVALA

Zahvaljujem se mentorju doc. dr. Roku Rupniku za pomoč in nasvete pri izdelavi diplomske naloge. Predvsem pa se zahvaljujem staršem, ki so mi študij omogočili, me spodbujali in podpirali. Zahvaljujem se tudi svojemu dekletu Jani za vso njeno pomoč in potrpežljivost.

KAZALO VSEBINE

Povzetek 1
Abstract 2
1. UVOD 3
2. OSNOVNA ZGRADBA IN DELOVANJE DJANGA 5
 2.1. Priprava 5
 2.1.1. Namestitev 5
 2.1.2. Priprava podatkovne baze 7
 2.2. Začetek izdelave spletne strani 7
 2.2.1. Projekt 7
 2.2.2. Aplikacija znotraj projekta 8
 2.2.3. Razvojni spletni strežnik 8
 2.3. Glavne nastavitve projekta in aplikacij znotraj projekta 9
 2.4. Veljavne spletne povezave za naš projekt in z njimi povezani klici funkcij 11
 2.5. Kontroler (ang. controller) 13
 2.6. Predloge (ang. templates) 14
 2.6.1. Značke v predlogah 15
 2.6.2. Filtri v predlogah 18
 2.7. Model 20
 2.7.1. Definiranje modela 20
 2.7.2. Tipi atributov v modelu 22
 2.7.3. Generiranje/brisanje tabel v podatkovni bazi 23
 2.7.4. Delo z modeli 25
3. PREDNOSTI IN SLABOSTI UPORABE OGRODIJ MVC PRI IZDELAVI SPLETNIH APLIKACIJ 28
 3.1. Primerjava programske kode 29
 3.2. Podatki in baza podatkov 32
 3.3. Administracijsko okolje 34
 3.4. Varnost spletnih aplikacij 36
 3.5. Primerjava zmogljivosti v produkcijskem okolju 38
4. ZAKLJUČEK 40
VIRI 41

KAZALO SLIK

Slika 1: Osnovni princip koncepta MVC 3
Slika 2: Rezultat pri uspešni namestitvi Djanga...5
Slika 3: Ubuntu Software Center..6
Slika 4: Osnovna datotečna struktura projekta..7
Slika 5: Osnovna datotečna struktura aplikacije znotraj projekta 8
Slika 6: Izgled naše spletne strani 9
Slika 7: Model v Djangu...21
Slika 8: Izgled generiranega modela 21
Slika 9: Tabele v podatkovni bazi 24
Slika 10: Primer krajše kode ob uporabi ogrodja 30
Slika 11: Primer lepšega internetnega naslova spletne strani pri uporabi ogrodja 30
Slika 12: Prikaz napake v spletni aplikaciji ob uporabi ogrodja Django......................................31
Slika 13: Objektno-relacijska preslikava..32
Slika 14: Primer programiranja z uporabo ogrodja in brez njega...34
Slika 15: Administracijsko okolje v Djangu...36
Slika 16: Vrivanje stavkov SQL...37
Slika 17: Uporaba delovnega pomnilnika (v kB) med različnimi ogrodji 38
Slika 18: Primerjava obremenjenosti procesorja strežnika pri uporabi različnih vtičnikov spletnega strežnika...39

KAZALO TABEL

Tabela 1: Nekaj osnovnih posebnih znakov v regularnih izrazih ……………………………….……12
Tabela 2: Nekaj osnovnih metapodatkov, zapisanih v zahtevi HTPP..……………………….....14
Tabela 3: Uporaba operatorjev v znački *for* …………………………………………….... 16
Tabela 4: Vgrajeni filtri v Djangu ……………………………………………………………19
Tabela 5: Tipi osnovnih atributov v modelu …………………………………………………22
Tabela 6: Tipi osnovnih argumentov atributov v modelu …………………………………... 23
Tabela 7: Tipi atributov v modelu, ki predstavljajo povezavo med modeli …..…………….... 23
Tabela 8: Nekaj osnovnih definiranih besed za filtriranje iskanja ………………………….. 27

Povzetek

Diplomska naloga predstavlja sistem za hiter razvoj spletnih aplikacij in ga primerja s klasičnim razvojem spletnih aplikacij. Diplomska naloga je predstavljena v dveh delih.

V prvem delu je predstavljen Django. To je odprtokodno visokonivojsko Pythonovo ogrodje za hiter razvoj spletnih aplikacij. Obravnavana sta osnovna zgradba in delovanje tega ogrodja. Predstavljena je tudi t. i. arhitektura *Model-View-Controler* (v nadaljevanju MVC), ki jo Django tudi uporablja. Predstavljene bodo samo osnove Djanga, ne pa tudi naprednejše možnosti, ki jih sistem omogoča. Predstavitev ogrodja je pomembna predvsem zaradi lažjega razumevanja hitrega razvoja spletnih aplikacij z ogrodji MVC. Za Django sem se odločil tudi zaradi njegove razširjenosti, dobre dokumentacije in dobrega poznavanja programskega jezika Python.

V drugem delu je prikazana primerjava razvoja spletnih aplikacij z ogrodji MVC, med katera spada tudi Django, s klasičnim programiranjem. Osnovni in glavni cilj uporabe ogrodij je povečati kakovost in hitrost razvoja z uporabo vgrajenih orodij in metodologij, ki temeljijo na obstoječem jeziku.
Ogrodja imajo že veliko vgrajenih funkcij in orodij, ki programerjem olajšajo delo. To so npr. avtentikacija, avtorizacija in registracija uporabnikov, administracija spletne aplikacije, varnost in pravice uporabnikov, varnost spletne aplikacije, delo s spletnimi obrazci (angl. *forms*), delo s podatki in validacija podatkov. Na trgu obstaja veliko različnih ogrodij MVC. Cilj te diplomske naloge ni primerjava ogrodij MVC med sabo, ampak le prednosti in slabosti njihove uporabe.

Ključne besede: Django, modularen način razvoja, koncept MVC, spletne aplikacije, hitri razvoj spletnih aplikacij, Python

Abstract

In the diploma work the system for rapid development of web applications is presented and compared to the standard development of web applications. It consists of two parts.

In the first part Django is described. This is a high-level Python Web framework that encourages rapid development of web applications. Basic structure and function of this framework are investigated. The so called Model-View-Controler (MVC) architecture used by Django is also described. Only basic and not advanced features of Django are presented here. The presentation of this framework is important due to easier understanding of web application rapid progress through MVC framework. Django was chosen for its wide range, good documentation and our knowledge of Python programming language.

In the second part the development of web applications with MVC framework, including Django, is compared with the standard programming. The basic and main aim of using frameworks is to intensify their quality and development speed through the usage of built-in tools and methodologies which are based on the existent language.

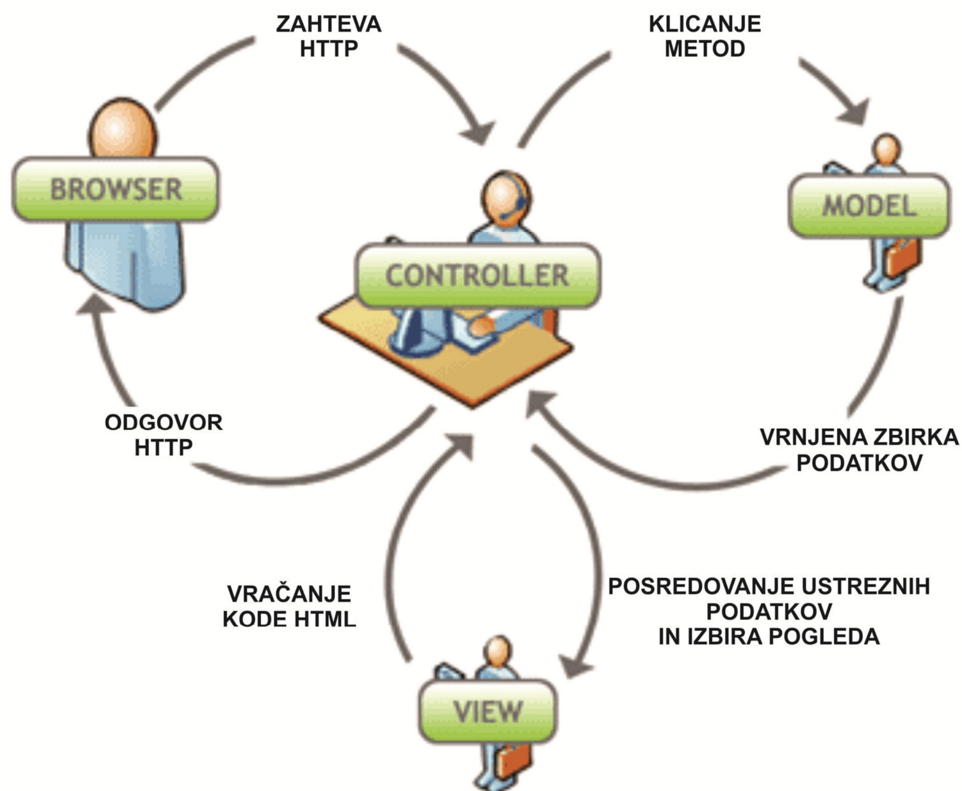
There are many built-in functions and tools already integrated into the frameworks, which make work easier for programmers. These are for instance authentication , authorization and registration of users, administration of web application, safety and users' rights, web application safety, work with web forms, data work and data validation. There are many MVC frameworks in the market. However, the aim of this diploma is not to compare different MVC frameworks to one another, but to present the advantages and disadvantages of using them.

Key words: Django, modular development, MVC concept, web applications, rapid development of web applications, Python

1. UVOD

V današnjem času oz. v informacijski dobi so zahteve po vse bolj naprednih in prilagodljivih internetnih straneh vse večje. Razvijanje takih internetnih strani za programerje postaja vse bolj kompleksno in zahtevno. Zato so se na trgu pojavile nove rešitve in tehnologije. Nekatere tehnologije so dobrodošle in inovativne, druge pa so le izvedene iz že obstoječih tehnologij. Sistem oz. ogrodje Django, ki sem ga izbral za temo diplomske naloge, temelji na arhitekturi *Model-View-Controller* (v nadaljevanju MVC).

MVC pomeni, da je sama zgradba programa oz. spletne aplikacije razdeljena na tri sklope oziroma plasti. To so model, pogled in kontroler. Model predstavlja naše podatke. Predstavlja zgradbo naših podatkov, atribute in relacije med njimi. Skrbi za logiko za dostopanje do podatkov, manipulacijo nad podatki in shrambo podatkov v neko določeno zbirko podatkov.
Pogled skrbi za ustrezen prikaz podatkov. Podatke dobi od kontrolerja. Sam pogled določa, kateri podatki in na kakšen način so vidni uporabniku. Podatki se lahko ločijo glede na pooblastila uporabnika. Kontroler skrbi za povezavo med modelom in pogledom. Kontroler določa konsistentnost podatkov (pravilna oblika oz. format), preverja avtentikacijo in avtorizacijo uporabnikov. Sam kontroler podatke dobi od modela in na podlagi tega se odloči, kateri podatki bodo prikazani in kateri pogled bo izbran za prikaz podatkov.
Osnovni princip koncepta MVC prikazuje Slika 1.



Slika 1: Osnovni princip koncepta MVC

Django je napisan v programskem jeziku Python. Uporablja se pri programiranju oz. povsod, tudi v nastavitvah Djanga. Zgodovina Djanga se je začela leta 2003, ko sta programerja spletnih strani, Adrian Holovaty in Simon Willison, začela uporabljati Python za izdelavo spletnih strani. Zaposlena sta bila v podjetju The World Company. To podjetje je takrat upravljalo in vzdrževalo kar nekaj večjih lokalnih novičarskih spletnih strani (LJWorld.com, Lawrence.com, KUsports.com …). Ker je postajala zahteva po hitrem spreminjanju, dodelavi in programiranju novih spletnih strani vedno večja, sta takrat začela razvijati ogrodje, ki jima je vse to omogočalo.
Leta 2005 je bilo to ogrodje že razvito do te točke, da je vse to lahko omogočalo. Takrat sta se Adrian Holovaty in Simon Willison skupaj z Jacobom Kaplanom - Moosom odločila, da to ogrodje izdajo pod licenco BSD. Izdali so ga julija 2005 in poimenovali po francoskem džezovskem kitaristu Djangu Reinhartu.
Danes je Django dobro uveljavljen odprtokodni projekt z več deset tisoč sodelujočimi uporabniki. Njegova poglavitna prednost oz. cilj je olajšanje razvoja kompleksnih in podatkovnih spletnih strani.

Zadnja trenutna različica je 1.4. Da je Django dosegel izdajo različice 1.0, je potreboval več kot tri leta. Razvijalci ohranjajo združljivost za nazaj. To pomeni, da bodo naše izdelane spletne strani pri posodobitvi Djanga na različice 1.5, 1.6 in 1.9 delovale brez poseganja v kodo naše spletne strani. Ko pa bo Django dosegel različico 2.0, naša spletna stran verjetno ne bo več delovala brez poseganja v programsko kodo. Sodeč po trenutnih izdajah različic lahko predvidevamo, da bo do različice 2.0 minilo še kar nekaj časa.

2. OSNOVNA ZGRADBA IN DELOVANJE DJANGA

2.1. Priprava

2.1.1. Namestitev

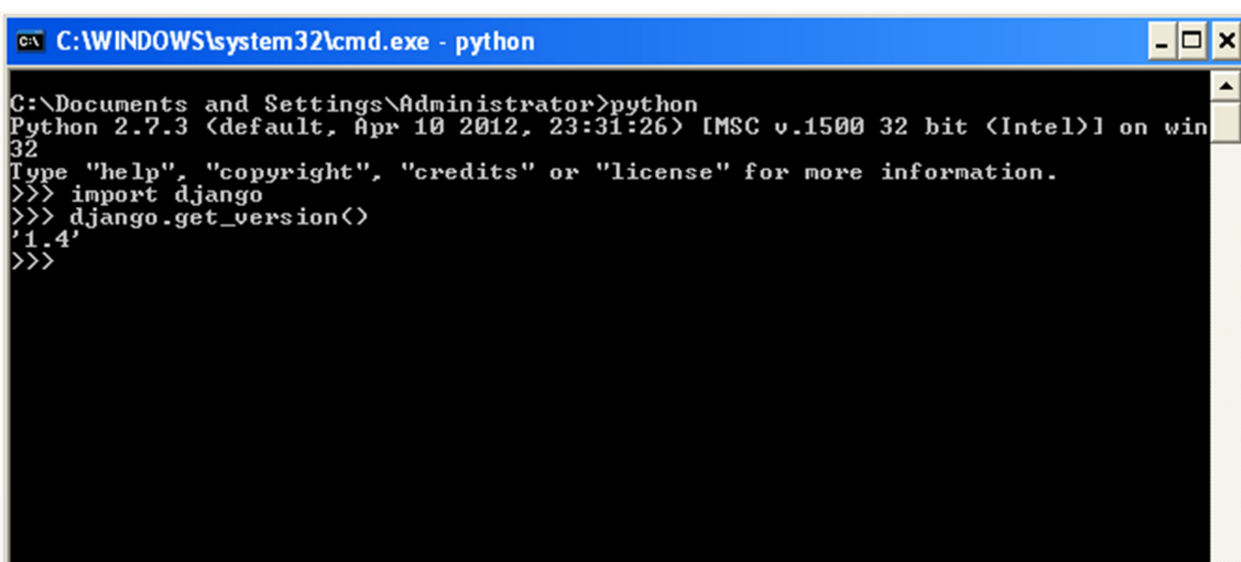
Ker je Django v celoti napisan v programskem jeziku Python, bo deloval v vseh operacijskih sistemih, v katerih deluje Python. Deluje celo na nekaterih pametnih telefonih. Django za svoje delovanje potrebuje Python različico 2.6 ali 2.7. Zaenkrat še ne podpira različice 3.0. V tej diplomski nalogi bodo predstavljene namestitev trenutne različice Djanga 1.4 in trenutne delovne izdaje Pythona 2.7.3 na operacijski sistem Windows 7 in Ubuntu 12.04 lts.

Za namestitev Pythona na operacijski sistem Windows se odpravimo na njihovo uradno stran *www.python.org* in z nje prenesemo binarno datoteko Python 7.3 Windows Installer.
Program namestimo z dvakratnim klikom na to datoteko in sledimo navodilom. Po končani namestitvi moramo v sistemsko spremenljivko *Path* dodati pot do Pythonovih izvršnih datotek. Dodati moramo *C:\Python27\;C:\Python27\Scripts.,* če smo seveda inštalirali Python v privzeto mapo. Več o dodajanju v sistemsko spremenljivko na [1].

Za namestitev Djanga pa moramo z njihove uradne spletne strani *https://www.djangoproject.com* prenesti namestitveno datoteko. To datoteko razširimo. Odpremo konzolno okno in se premaknemo v mapo, v katero smo razširili to datoteko. Zaženemo ukaz:

```
python setup.py install.
```

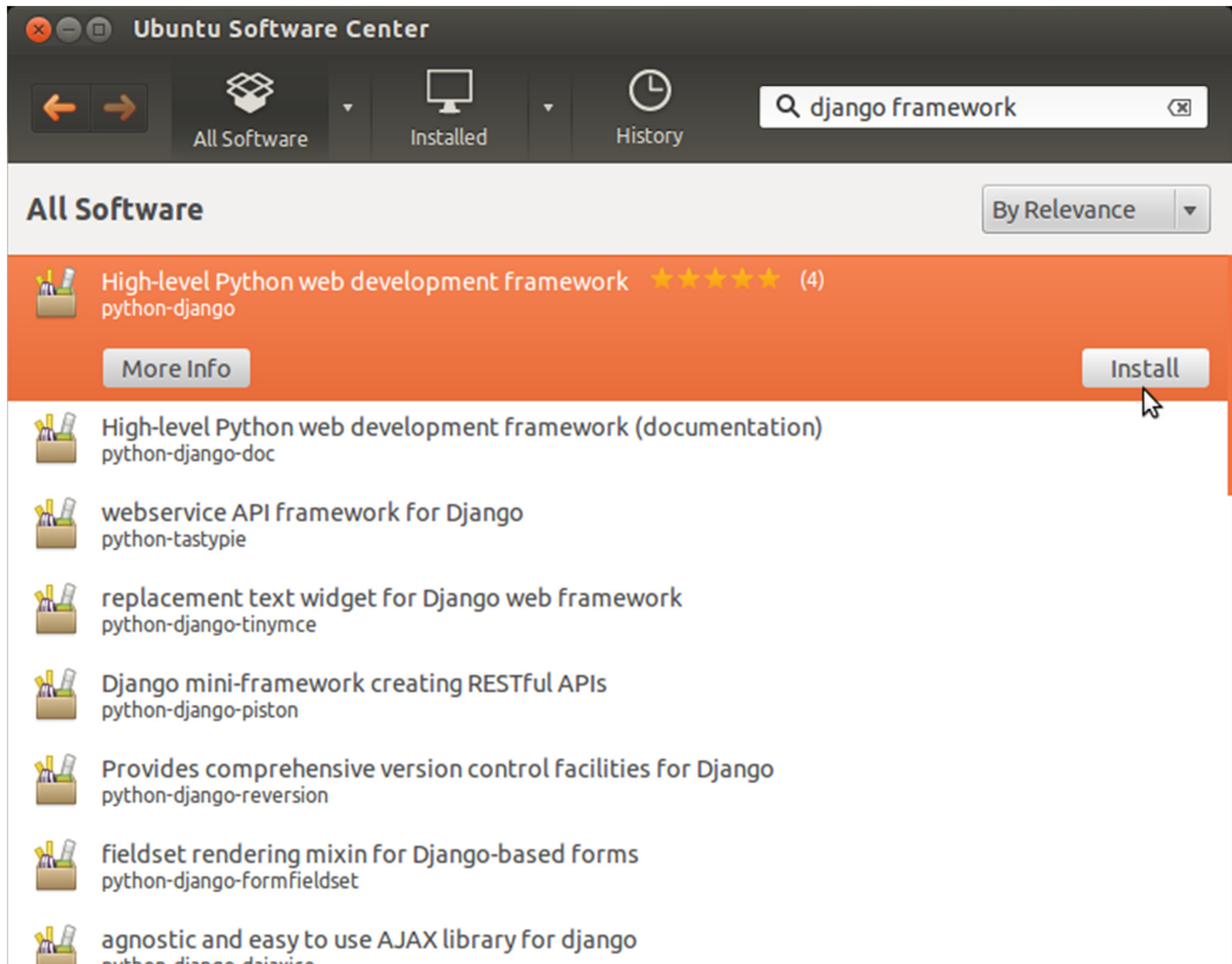
Vse, kar ta namestitev naredi, je to, da v mapo *c:\Python27\Lib\site-packages\Django* namesti potrebne datoteke oz. knjižnice. Pri posodabljanju Djanga v novejšo verzijo je potrebno to mapo izbrisati, da ne pride do razlik v programski kodi. Če se je Django pravilno namestil, moramo v konzolnem oknu pri spodnjem postopku (Slika 2) dobiti enak rezultat.



Slika 2: Rezultat pri uspešni namestitvi Djanga

V operacijskem sistemu Linux pa je Python že nameščen. Django bi lahko namestili z aplikacijo Ubuntu Software Center , kjer bi ga poiskali in namestili s klikom na gumb Install. Namestila bi se različica 1.3.1. Za namestitev različice 1.4 pa moramo ravno tako z njihove strani prenesti namestitveno datoteko in namestiti ročno.

Vse, kar ta namestitev naredi, je, da v mapo */usr/lib/python2.7/dist-packages/django* namesti potrebne datoteke oz. knjižnice. Pri posodabljanju Djanga v novejšo verzijo je potrebno to mapo izbrisati, da ne pride do razlik v kodi.



Slika 3: Ubuntu Software Center

2.1.2. Priprava podatkovne baze

Če hočemo uporabljati Django samo za prikaz dinamičnih strani, ne potrebujemo podatkovne baze. Za razvoj spletnih strani, ki shranjujejo oz. berejo shranjene podatke, pa potrebujemo podatkovno bazo. Za povezavo s podatkovnimi bazami moramo namestiti ustrezne Pythonove knjižnice. Podprte so štiri različne podatkovne baze:

- PostgreSQL: Obstaja več knjižnic za povezavo s to podatkovno bazo. Najbolj razširjena je knjižnica Psycopg2. Dobimo jo na naslovu: *http://initd.org/psycopg/*.
- SQLite3: Pri uporabi različice Python 2.5 ali več nam ni potrebno namestiti ničesar. Vse knjižnice so že vključene.
- MySQL: Django zahteva MySQL različice 4.0 ali več. Potrebno je namestiti tudi knjižnico MySQLdb. Dobimo jo na naslovu *http://sourceforge.net/projects/mysql-python/*.
- Oracle: Django zahteva Oracle Database Server različice 9i ali več. Namestiti je potrebno knjižnico cx_Oracle. Dobimo jo na naslovu *http://cx-oracle.sourceforge.net/*.

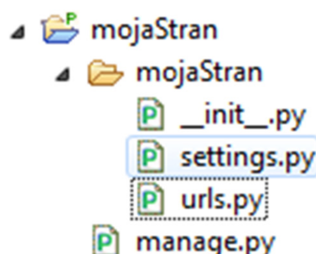
2.2. Začetek izdelave spletne strani

2.2.1. Projekt

Django vsako spletno stran poimenuje projekt. Projekt je zbirka nastavitev za Django, zbirka nastavitev za podatkovno bazo in zbirka nastavitev za aplikacije znotraj spletne strani. Poglejmo si primer ustvarjanja projekta mojaStran v konzolnem oknu oz. terminalu (odvisno od operacijskega sistema):

```
django-admin.py startproject mojaStran.
```

Ukaz ustvari spodnje mape in datoteke:



Slika 4: Osnovna datotečna struktura projekta

Opis ustvarjenih datotek:

- *__init__.py*: zahtevana datoteka, da Python celotno mapo obravnava kot paket (skupino modulov); datoteka je prazna;
- *manage.py*: program, napisan v programskem jeziku Python, ki služi za interakcijo s tem projektom na več načinov;
- *settings.py*: datoteka, v kateri so shranjene vse nastavitve za ta projekt in aplikacije znotraj projekta;

- *urls.py*: datoteka, v kateri so definirane veljavne spletne povezave za naš projekt in z njimi povezani klici funkcij.

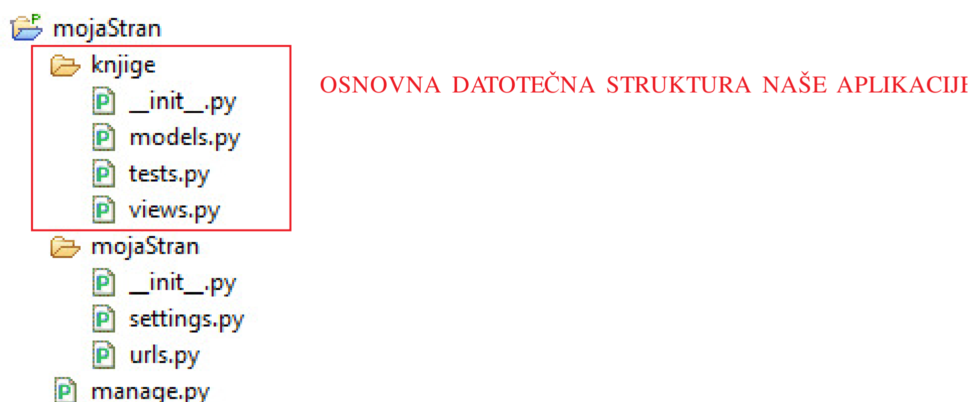
2.2.2. Aplikacija znotraj projekta

Aplikacija je skupek programerske kode znotraj projekta, ki vključuje kontroler, modele in poglede. Aplikacija naj predstavlja zaključeno celoto nekega logičnega dela spletne strani. Enostavne spletne strani imajo lahko samo eno aplikacijo. Za večje in obsežnejše spletne strani pa je priporočljivo, da posamezne logične dele spletne strani ločimo v svoje aplikacije. Primer aplikacij so npr. CMS (sistem za upravljanje vsebine), forum in sistem za registracijo uporabnikov. Aplikacije znotraj projektov so prenosljive. To pomeni, da jih lahko prekopiramo oz. prenesemo in uporabimo v drugem projektu.

Poglejmo si primer ustvarjanja aplikacije *knjige* v konzolnem oknu oz. terminalu (odvisno od operacijskega sistema). Nahajati se moramo znotraj projekta oz. v mapi, kjer se nahaja program *manage.py*:

```
manage.py startapp knjige.
```

Ukaz ustvari spodnje mape in datoteke znotraj projekta.



Slika 5: Osnovna datotečna struktura aplikacije znotraj projekta

Opis ustvarjenih datotek:

- *__init__.py:* zahtevana datoteka, da Python celotno mapo obravnava kot paket (skupino modulov); datoteka je prazna;
- *models.py:* datoteka, v kateri je definiran model naše aplikacije;
- *tests.py:* datoteka, s katero lahko avtomatsko testiramo oz. iščemo hrošče v naši kodi;
- *views.py:* datoteka, v kateri so definirani pogledi naše aplikacije; na začetku je datoteka prazna.

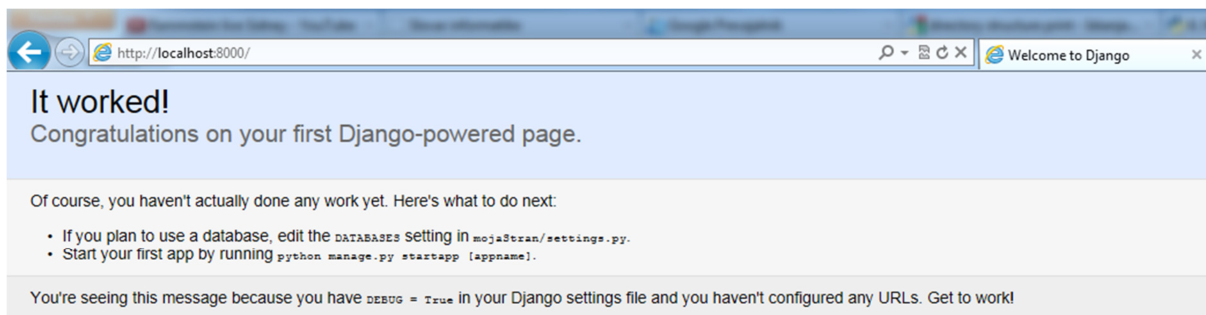
2.2.3. Razvojni spletni strežnik

V ogrodje Django je vključen tudi spletni strežnik. Uporabi se ga lahko samo v fazi razvijanja oz. testiranja, nikakor pa ni predviden za produkcijske namene. Dobra lastnost tega razvojnega spletnega strežnika je seveda ta, da nam pri razvoju ni potrebno nastavljati in nameščati drugih spletnih strežnikov. Ta vključeni spletni strežnik spremlja našo programsko kodo in se med njenim spreminjanjem sam osvežuje. S tem skrajšamo čas, saj nam za vsako

spremembo ni potrebno spletnega strežnika ponovno zaganjati. Opozori nas tudi na razne napake v programski kodi. Ko so morebitne napake odpravljene, ga poskusimo zagnati.
V konzolnem oknu se moramo nahajati v mapi, ki vsebuje datoteko *manage.py*. Spletni strežnik zaženemo z ukazom:

```
manage.py runserver.
```

Odpremo brskalnik in spletni naslov *http://localhost:8000*. Če je vse pravilno, bi se nam morala odpreti spletna stran (Slika 4).



Slika 6: Izgled naše spletne strani

2.3. Glavne nastavitve projekta in aplikacij znotraj projekta

V datoteki *settings.py*, ki se ustvari ob ustvarjanju projekta, so shranjene vse nastavitve projekta in aplikacij znotraj projekta. Ko ustvarimo projekt, so v datoteki že vpisane osnovne nastavitve. Poglejmo si datoteko oz. nekaj osnovnih nastavitev.

```
# Razhroščevalni način. Vse napake se prikažejo na spletni strani.
# V produkciji je potrebno to nastavitev prestaviti na false.
DEBUG = True
TEMPLATE_DEBUG = DEBUG

# Seznam naslovnikov, katerim Django pošlje opozorila, če pride do
# napak pri izvajanju. Pošiljanje deluje samo, če je debug nastavljen
# na false.
ADMINS = (
    # ('Ime Priimek ', 'e-poštni naslov'),
)

# Seznam naslovnikov, na katere Django pošlje opozorila, če
# obstajajo pokvarjene povezave URL ali pa če povezave ne obstajajo.
MANAGERS = ADMINS

# Nastavitve za podatkovno bazo
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.', # Dodati eno izmed možnosti:
            # 'postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': '',                      # Ime zbirke podatkov
        'USER': '',                      # Uporabniško ime za dostop
```

```
        'PASSWORD': '',                   # Geslo za dostop
        'HOST': '',                       # Strežnik
        'PORT': '',                       # Vrata, s katerimi
            # komunicira strežnik. Lahko je prazno.
    }
}

# Nastavitev časovnega pasu
TIME_ZONE = 'Europe/Ljubljana'

# Nastavitev privzetega jezika
LANGUAGE_CODE = 'sl-si'


# Identifikator projekta - uporablja se, če mora ena podatkovna baza
# upravljati vsebino za več spletnih strani oz. projektov.
SITE_ID = 1

# Ali naj bo vključena internacionalizacija?
USE_I18N = True

# ALi naj bo vključena lokalizacija oz. krajevna prilagoditev (npr.
# prikaz številk in datumov)?
USE_L10N = True

# Datotečna pot do statičnih datotek (npr. slike, css, Javascript)
STATIC_ROOT = ''

# Naslov URL do statičnih datotek (npr. slike, css, Javascript)
STATIC_URL = '/static/'

# Naključni niz se avtomatsko ustvari pri ustvarjanju projekta.
# Uporablja se kot skriti ključ v razpršilnem algoritmu.
SECRET_KEY = '7ks90ghi!3padyv4!m5$2janxcl1*!0y6o-^-7+zu@7#7j-*&amp;b'

# Nalagalniki predlog
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader',
    'django.template.loaders.eggs.Loader',
)

# Razredi middleware
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
)

# Pot do datoteke, kjer so definirane veljavne spletne povezave za
# naš projekt in z njimi povezani klici funkcij
ROOT_URLCONF = 'mojaStran.urls'

# Pot do predlog
TEMPLATE_DIRS = (
    # npr "/home/html/django_predloge"
)
```

```
# Aplikacije, ki se bodo uporabljale znotraj projekta. Dodati moramo
# tudi lastne aplikacije. Dodana je prej ustvarjena aplikacija
# knjige.

INSTALLED_APPS = (
    'knjige',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
)

# Beleženje napak in ukrepanje ob napakah
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        }
    },
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler'
        }
    },
    'loggers': {
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': True,
        },
    }
}
```

2.4. Veljavne spletne povezave za naš projekt in z njimi povezani klici funkcij

Za veljavne spletne povezave skrbi datoteka *urls.py*, ki se nahaja znotraj projekta. Da ravno ta datoteka skrbi za to, je definirano v datoteki *setings.py*:

```
ROOT_URLCONF = 'mojaStran.urls'.
```

Zgornji primer nakazuje, da so veljavne spletne povezave definirane v mapi *mojaStran*, v datoteki *urls.py*.

Za povezovanje spletnega naslova s funkcijo (napisano v Pythonu) se uporablja regularne izraze (angl. *regular expression, regex, regexp*). Regularni izrazi so pameten način iskanja (in menjanja) besedila. Regularni izrazi so lahko tako uporabni kot zapleteni. Za iskanje besedila se uporablja posebne znake, ki se ujemajo z delčki besedila. Tabela 1 predstavlja nekaj osnovnih posebnih znakov.

Posebni znak	Opis	Primer
. (pika)	ujemanje s katerim koli posameznim znakom (črka, številka, simbol)	A.C se ujema z ABC, AAC,AKC, AUC …
? (vprašaj)	ujemanje z nobenim ali enim od prejšnjih elementov	A?C se ujema z AC, AAC
* (zvezdica)	ujemanje z nobenim ali večjim številom prejšnjih elementov	A*C se ujema z AC, AAC, AAAAC …
^ (znak »manjka«)	zahteva, da so podatki na začetku niza	^test se ujema z nizom test, ne pa tudi z nizom 1test
$ (dolar)	zahteva, da so podatki na koncu niza	test$ se ujema z nizom test, ne pa tudi z nizom test1
\d	ujemanje s številko	A\dC se ujema z A4C, A5C …, ne pa tudi z ABC
\w	ujemanje s črko, številko ali _ (podčrtaj)	A\wC se ujema z A4C, AHC, A_C…
^$	zahteva, da je niz prazen	»«

Tabela 1: Nekaj osnovnih posebnih znakov v regularnih izrazih

Primer vsebine datoteke *urls.py*:

```
# Naložimo potrebne pakete.
from django.conf.urls import patterns, url

# Privzeta funkcija, če Django ne najde veljavne povezave
handler404='knjige.views.neveljavenNaslov'

urlpatterns = patterns('',
     url(r'^$', 'knjige.views.pozdravljenSvet'),        # primer 1
     url(r'^datum/$', 'knjige.views.datum'),            # primer 2
     url(r'^sestej/(\d+)/$', 'knjige.views.sestejZ10'), # primer 3
)
```

Postopek primerjave veljavnega spletnega naslova in klic ustrezne programske funkcije:

1. Ko uporabnik v brskalnik vpiše spletni naslov naše spletne strani, Django shrani ta naslov v neki niz in po potrebi doda končni znak / . Primer je npr. *http://<moja domena>.si/datum/*.
2. Iz tega niza odstrani naslov naše domene (npr. *http://<naslov naše domene>.si/*). V nizu ostane samo še besedilo, ki je desno od celotnega spletnega naslova (npr. *datum/*).
3. Django ta niz primerja z uporabo regularnih izrazov z vsakim prvim argumentom vseh vpisanih programskih funkcij url v datoteki *urls.py*.
4. Ko najde prvo ujemanje, kliče funkcijo, ki se nahaja kot drugi argument te funkcije url (npr. *'knjige.views.datum'*). Uporabniku se prikaže spletna stran. Vsebina spletne strani je vrnjena vrednost te funkcije, napisane v programskem jeziku Python.

Primer 1 znotraj naše datoteke prikazuje regularni izraz ^$. Ta regularni izraz se ujema samo takrat, ko uporabnik odpre našo spletno stran brez podstrani (npr. *http://<moja domena>.si*). Kliče se funkcija *pozdravljenSvet*. Ta funkcija pa se nahaja v mapi (oz. aplikaciji) *knjige* v datoteki *views.py*.

Primer 2 znotraj naše datoteke prikazuje regularni izraz *^datum/$*. Ta regularni izraz se ujema samo takrat, ko uporabnik odpre podstran *datum/* na naši spletni strani (*npr. http://<moja domena>.si/datum/*). Kliče se funkcija *datum*.

Primer 3 znotraj naše datoteke prikazuje regularni izraz *^sestej/(\d+)$*. *\d+* predstavlja katero koli naravno število, *(\d+)* pa pomeni, da se to naravno število pošlje kot argument klicani funkciji. Primer je npr. *http://moja domena>.si/sestej/45/.*
Kliče se funkcija *sestejZ10*, ki kot drugi argument prejme število 45, shranjeno kot niz (ang. *string*). Kot prvi argument funkcija vedno prejme objekt *HttpResponse*. Več o tem v poglavju 1.5.

Če uporabnik zahteva neveljavno spletno povezavo, spletni strežnik prikaže stran z besedilom »A server error occurred. Please contact the administrator«. Ta vrsta napake je poznana kot napaka 404. V datoteki *urls.py* je zato spremenljivki *handler404* vpisana funkcija, ki se vedno kliče ob neveljavni spletni povezavi.

2.5. Kontroler (ang. *controller*)

Kontroler predstavlja eno ali več funkcij, napisanih v programskem jeziku Python. Funkcije lahko kot vhod sprejmejo tudi argumente, vračajo pa vrednosti oz. kodo HTML. Privzeto je predpostavljeno, da so zapisane znotraj aplikacij v datoteki *views.py*. Ni pa to nujno, lahko so tudi v drugih datotekah znotraj aplikacij. Spodaj je prikazan primer funkcij, napisanih v programskem jeziku Python. Prva v brskalniku prikaže besedilo *»Hello world«*, druga pa trenutni datum. Tretja kot drugi argument sprejme niz *stevilo*, vrne pa seštevek tega števila z 10.

```
# Naložimo potrebne pakete.
from django.http import HttpResponse
import datetime

# Definiramo funkcijo pozdravljenSvet.
def pozdravljenSvet(request):
    return HttpResponse("<html><body> Hello world </body></html>")

# Definiramo funkcijo datum.
def datum(request):
    datum=datetime.date.today()
    return HttpResponse("<html><body>"+str(datum)+"</body><html>")

# Definiramo funkcijo sestejZ10.
def sestejZ10(request,stevilo):
    sestevek=10+int(stevilo)
    return HttpResponse("<html><body>"+str(sestevek)+"</body><html>")
```

Glavne funkcije oz. funkcije, ki se kličejo neposredno, vedno kot prvi argument že prejmejo objekt *HttpRequest*, vračajo pa objekt *HttpResponse*. To velja za vse funkcije, razen naših pomožnih. *HttpRequest* je generiran takrat, ko se zahteva neka določena spletna stran. Vsebuje metapodatke o zahtevi http. Nekaj osnovnih metapodatkov je predstavljenih v Tabeli 2. Več o tem pa na [2].

Metapodatek	Opis
path	niz, v katerem je shranjena celotna absolutna pot do zahteve spletne strani (iz niza je odstranjen naslov naše spletne strani)
method	metoda http, ki je bila uporabljena v zahtevi http
GET	seznam vseh argumentov GET
POST	seznam vseh argumentov POST
META	seznam vseh metapodaktov v glavi zahtevka http (npr. ime gostitelja, naslov ip)
user	trenutno prijavljeni uporabnik
USER_AGENT	ime brskalnika, ki dostopa do spletne strani

Tabela 2: Nekaj osnovnih metapodatkov, zapisanih v zahtevi HTTP

2.6. Predloge (ang. *templates*)

V poglavju 1.5 smo pogledali, kako funkcije neposredno vračajo kodo HTML. Ta način ima kar nekaj pomanjkljivosti:

- zapleteno kodiranje;
- vsaka sprememba v grafični podobi spletne strani zahteva spreminjanje kode Python;
- pisanje kode in oblikovanje grafične podobe spletne strani sta čisto ločeni zadevi – oblikovalcem grafične podobe spletnih strani ne bi bilo treba popravljati kode Python, ponavadi je tudi ne poznajo.
- bolj učinkovito in hitreje je, če programerji lahko kodirajo, medtem pa oblikovalci istočasno že oblikujejo spletno stran.

Zaradi teh razlogov je veliko bolje, da ločimo grafično podobo od kode Python oz. ločimo programsko logiko od vsebine. To lahko storimo z Djangovimi predlogami (angl. *Django's template sytem*). Predloge so datoteke s končnico html, ki vsebujejo kodo HTML.

Namesto, da bi kontroler vrnil celotno kodo HTML z objektom *HttpResponce*, jo vrne s funkcijo *render_to_response*:

```
return render_to_response("detajl.html",{'ključ':'vrednost'}).
```

Funkcija kot prvi argument sprejme ime datoteke s kodo HTML. Django poišče to datoteko v eni izmed map, ki smo jih nastavili v nastavitveni datoteki *settings.py*, pod nastavitvijo *template_dirs*.
Kot drugi argument pa sprejme seznam »ključ : vrednost« elementov. Ključ predstavlja ime nove spremenljivke, ki se nato uporablja v predlogi. Vrednost pa predstavlja vrednost te nove spremenljivke. Vrednosti so lahko nizi (angl. *strings*), cela števila (angl. *integers*), realna števila (angl. *decimal numbers*), datumi (angl. *dates*), časi (angl. *time*) in logične vrednosti (angl. *boolean*).

Django torej v predlogo prenese novo spremenljivko z določeno vrednostjo. Nove spremenljivke v predlogi kličemo tako, da njihova imena zapišemo med dvojne oglate oklepaje. Če je v predlogi navedena spremenljivka, ki funkcije ni podala oz. vrnila, Django to

enostavno spregleda oz. ne javi napake. Posamezna spremenljivka se lahko v predlogi pojavi tudi večkrat.
Primer:

Datoteka views.py:

```
# Naložimo potrebne pakete.
from django.shortcuts import render_to_response

# Definiramo funkcijo pozdravljenSvet.
def pozdravljenSvet(request):
      seznam={'niz1':'Hello world.' , 'niz2':'Primer predloge.'}
      return render_to_response("stran.html",seznam)
```

Datoteka stran.html:

```
<html> <body>
      {{niz}}  <br /> {{niz2}}
</body> </html>
```

Izgled v brskalniku:

```
Hello world.
Primer predloge.
```

2.6.1. Značke v predlogah

V predlogah lahko uporabljamo tudi značke. Značke se uporabljajo za manipulacijo nad spremenljivkam. Vgrajenih je že veliko značk, kljub temu pa nam Django dopušča izdelavo lastnih. Več o tem na [3]. V tem poglavju bo predstavljenih samo nekaj najosnovnejših značk.

Značka *if/else* prestavlja pogojni stavek v programskih jezikih. Uporablja se za preverjanje, ali ima spremenljivka vrednost *true* ali *false*. Pri tem se upošteva, da je *true* vse, kar ni je *false*. Za *false* pa se upoštevajo:

- prazen seznam elementov,
- prazen niz,
- številska vrednost 0,
- objekt *None*,
- logična vrednost *false*.

Primer:

```
{% if ali_je_danes_sobota %}
      Danes je sobota
{% else %}
      Danes ni sobota
{% endif %}
```

Odvisno od vrednosti spremenljivke *ali_je_danes_petek* se na naši spletni strani prikaže ustrezno besedilo. Django dopušča tudi primerjanje dveh vrednosti z operatorji enakosti (==), neenakosti (!=), manjši (<), večji (>), manjši ali enak (<=), večji ali enak (>=). V znački lahko

uporabimo tudi operatorje, kot so logični ali (*or*), logični in (*and*) ali pa logični ne (*not*). V posamezni znački lahko uporabimo tudi več operatorjev.

```
{% if ali_je_danes_sobota and ali_je_danes_nedelja %}
      Vikend
{% else %}
      Delavni dan
{% endif %}

{% if temperatura > 30 %}
      To besedilo se izpiše le, če je vrednost spremenljivke
      temperatura večja od 30
{% endif %}
```

Značka *for* predstavlja zanko *for* v programskih jezikih. Ta zanka omogoča sprehod skozi vse elemente določenega seznama. V Tabeli 3 so prikazani operatorji, ki jih lahko uporabimo.

Operator	Opis
reverse	Omogoča, da se skozi seznam sprehodimo v nasprotnem vrstnem redu. Primer: *{% for dan in dni_v_tednu reverse %}*.
forloop.counter	To spremenljivko se lahko uporabi v zanki *for*. Vrne pa nam zaporedno številko elementa. Šteti začne z 1.
forloop.counter0	Enako kot *forloop.counter*, vendar začne šteti pri 0.
forloop.revcounter	Ta spremenljivka prikaže število elementov, skozi katere se še nismo sprehodili. Spremenljivka ima vrednost 1, ko se sprehodimo skozi zadnji element.
forloop.revcounter0	Enako kot *forloop.revcounter*, vendar ima vrednost 0, ko se sprehodimo skozi zadnji element.
forloop.first	To je spremenljivka, ki vrne logično vrednost *True*, samo če je to prvi element.
forloop.last	To je spremenljivka, ki vrne logično vrednost *True,* samo če je to zadnji element.

Tabela 3: Uporaba operatorjev v znački *for*

Primer:

```
# Na seznamu dni_v_tednu so shranjena vsa imena dni v tednu.
# V zanki for ustvarimo novo poljubno spremenljivko z imenom dan.
{% for dan in dni_v_tednu %}
      {{dan}}: To je dan z zaporedno številko {{forloop.counter}}.
      <br />
{% endfor %}

Izgled v brskalniku:

Ponedeljek: To je dan z zaporedno številko 1.
Torek: To je dan z zaporedno številko 2.
....
Nedelja: To je dan z zaporedno številko 7.
```

Značka *ifequal/ifnotequal* predstavlja primerjavo dveh vrednosti. Django v predlogi omogoča primerjavo spremenljivk z nizi, celimi števili in realnimi števili. Skozi leta je to značko nadomestila značka *if* z operatorjem enakosti.

```
{% ifequal spremenljivka1 'niz' %}
    Besedilo se prikaže samo, če sta vrednosti spremenljivke in
    niza enaki.
{% endifequal %}

{% ifequal spremenljivka1 1.3 %}
    Besedilo se prikaže samo, če ima vrednost spremenljivke 1.3
{% else %}
    Besedilo se prikaže samo, če vrednost spremenljivke ni 1.3
{% endifequal %}
```

Z značko (*in*) »vsebuje« lahko preverimo, če vrednost obstaja na seznamu. Lahko pa se uporabi tudi za preverjanje, če je določen niz vsebovan v drugem nizu ali spremenljivki.

```
{% if 'world' in 'Hello world' %}
    To besedilo se prikaže, saj niz 'Hello world' vsebuje
    niz 'world'
{% endif %}

{% if 'petek' in seznam_dni %}
    Besedilo se izpiše, če je niz 'petek' v seznamu
{% endif %}
```

Komentarji so v predlogah dovoljeni. Django pozna enovrstične in večvrstične komentarje.

```
{# enovrstični komentar #}

{% comment %}
    več
    vrstični
    komentar
{% endcomment %}
```

Značka *include* »vključi« omogoča, da v trenutno predlogo vključimo drugo predlogo. Vključena predloga lahko vsebuje tudi spremenljivke. Ta značka je uporabna pri obsežnejših spletnih straneh, kjer se koda, ki se večkrat ponavlja, shrani kot druga predloga To predlogo nato kličemo iz ostalih predlog. S tem se predvsem izognemo podvajanju pisanja kode oz. lažje spreminjamo že obstoječo. Primer uporabe so npr. glave in noge spletnih strani.

```
{% include 'glava.html' %}
```

Django nam omogoča tudi t. i. dedovanje vsebine predlog. Naredimo neko osnovno zgradbo datoteke HTML, v katero vnesemo tudi bloke (angl. *block*) brez vsebine. Nato iz svoje predloge kličemo osnovno predlogo. Django naloži vsebino te datoteke, namesto blokov pa naloži našo želeno vsebino. Če uporabimo dedovanje, nam Django ne dovoli pisanja kode drugam razen v bloke. Primer prikazuje, kako naložimo predlogo *predloga.html.*

Primer:

```
Datoteka osnovnaPredloga.html:

<html><body>
{%block naslov%} {% endblock %}
<br />
{%block vsebina %} {% endblock %}
</body><html>
```

```
Datoteka predloge.html:

{% extends "osnovnaPredloga.html" %}

{%block naslov%}
To je naslov
{% endblock %}

{%block vsebina %}
To je vsebina
{% endblock %}
```

```
Izgled kode html, ki se naloži v brskalnik:

<html><body>
To je naslov
<br />
To je vsebina
</body></html>
```

2.6.2. Filtri v predlogah

Filtri omogočajo enostavno spreminjanje vrednosti spremenljivke, tik preden se prikaže v brskalniku. Za filtre se uporablja znak | (*pipe character*). S tem ne spreminjamo vrednosti spremenljivke, ampak samo prikaz. Django ima že vgrajenih veliko filtrov, kljub temu pa nam Django dopušča izdelavo lastnih. Več o tem na [3]. Filtri lahko kot vhod sprejmejo tudi en argument. V tem poglavju bo predstavljenih samo nekaj najosnovnejših filtrov. Ogledali pa si bomo tudi izdelavo lastnega filtra.

Primer uporabe filtra:

```
# Filter capfirst nam v nizu vrednost spremeni prvo črko
# v veliko začetnico.
{{ vrednost|capfirst }}

# Filter cut nam iz niza vrednost odstrani vse znake, ki jih podamo
# kot argument. V tem primeru so to presledki.
{{ vrednost|cut:" " }}

# Primer uporabe več kot enega filtra
{{ vrednost|capfirst|cut:" " }}
```

Filter	Opis
add	Izbrani vrednosti prišteje število, ki ga podamo kot argument. Primer: {{ vrednost|add: " " }}
default	Če je vrednost spremenljivke *false* ali pa ta ne obstaja, se namesto nje vpiše besedilo, ki je podano kot argument. Primer: {{ vrednost|default: "vrednost" }}
first	Vrne prvi element s seznama. Primer: {{ vrednost|first }}
last	Vrne zadnji element s seznama. Primer: {{ vrednost|last }}
random	Vrne naključni element s seznama. Primer: {{ vrednost|random }}
floatformat	Zaokroževanje realnega števila. Več o argumentih na [4]. Primer: {{ vrednost|first }}
escape	Pretvori znake (<,>,', ", &) v ustrezne znake HTML.
length	Vrne število elementov v seznamu. Primer: {{ vrednost|length }}
rjust	Na konec niza doda presledke. Doda pa jih toliko, da potem celoten niz vsebuje število znakov, kot je navedeno v argumentu. Primer: {{ vrednost|rjust: "10" }}
ljust	Na začetek niza doda presledke. Doda jih toliko, da potem celoten niz vsebuje število znakov, kot je navedeno v argumentu. Primer: {{ vrednost|ljust: "10" }}
lower	Vse velike črke v nizu spremeni v male. Primer: {{ vrednost|lower }}
upper	Vse male črke v nizu spremeni v velike. Primer: {{ vrednost|upper }}
linebreaksbr	Vse znake za novo vrstico (\n) spremeni v znak HTML za novo vrstico ( ).
safe	Privzeto zaradi varnosti Django ne dopušča vračanja oznak HTML. Vse oznake kode HTML enostavno prikaže kot besedilo. S tem filtrom omogočimo uporabo oznak kode HTML.
striptags	Odstrani vse oznake kode HTML.
time	Spremeni obliko prikazovanja časa.
truncatechars	Če je niz predolg, odstrani znake in doda … . Maksimalno dolžino določimo z argumentom.
truncatewords	Z argumentom določimo maksimalno število besed v nizu. Ko je število doseženo doda … .
wordcount	Vrne število besed iz niza.

Tabela 4: Vgrajeni filtri v Djangu

Izdelava lastnega filtra je zelo enostavna. V aplikaciji naredimo mapo z imenom *templatetags*. V to mapo dodamo datoteko *__init__.py,* in datoteko s poljubnim imenom s končnico *.py* (npr. *mojiFiltri.py*). Filtri so napisane funkcije, ki vračajo neko vrednost. Uporabo lastnih filtrov v predlogi je potrebno v sami predlogi omogočiti z značko *load <ime naše datoteke.py>*.

Primer:

Datoteka templatetags\mojiFiltri.py:

```
# Dodamo potrebne pakete.
from django import template

# Funkcija vrne niz, ki mu doda . (piko).
# Če niz že vsebuje piko, je ne doda, ampak vrne samo niz.
def dodajPiko(vrednost):
    if not vrednost.endswith("."):
        return vrednost+"."
    return vrednost

# To funkcijo registriramo kot lastni filter.
register=template.Library()
register.filter('dodajPiko',dodajPiko)
```

Datoteka predloge.html:

```
{% load mojiTagi %}

{{ 'To je poved'|dodajPiko }}
<br />
{{ 'To je poved.'|dodajPiko }}
```

Izgled html kode, ki se naloži v brskalnik:

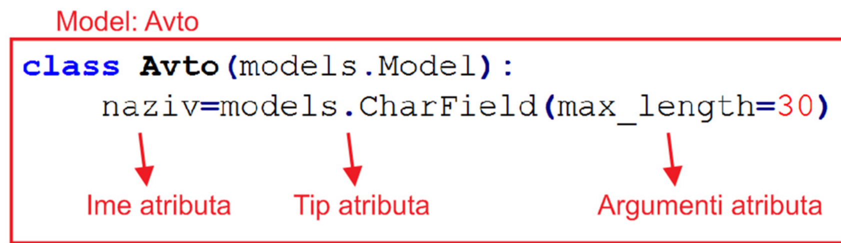
```
To je poved.
<br />
To je poved.
```

2.7. Model

Kot je bilo že omenjeno, model predstavlja naše podatke. Predstavlja zgradbo podatkov, atribute in relacije med njimi. Skrbi za logiko dostopanja do podatkov, manipulacijo nad podatki in shrambo podatkov v neko določeno zbirko podatkov. V poglavju 1.1.3 je prikazano, kako namestimo potrebne Pythonove knjižnice, v poglavju 1.3 pa, kako izberemo in nastavimo podatkovno bazo v datoteki *settings.py*.

2.7.1. Definiranje modela

Definiranju modela je namenjena datoteka *models.py*. Vsak model je predstavljen kot razred v programskem jeziku Python. Ta napisani razred je podrazred razreda *django.db.models.Model*. Ta razred vsebuje vse potrebno, da naši napisani razredi lahko komunicirajo s podatkovno bazo. Vsak razred v modelu predstavlja fizično podatkovno tabelo v neki zbirki podatkov oziroma v podatkovni bazi. Vsak atribut v razredu predstavlja stolpec v tabeli. Vsak tip atributa predstavlja vrsto oz. tip polja v tabeli.



Slika 7: Model v Djangu

Primer definiranja:

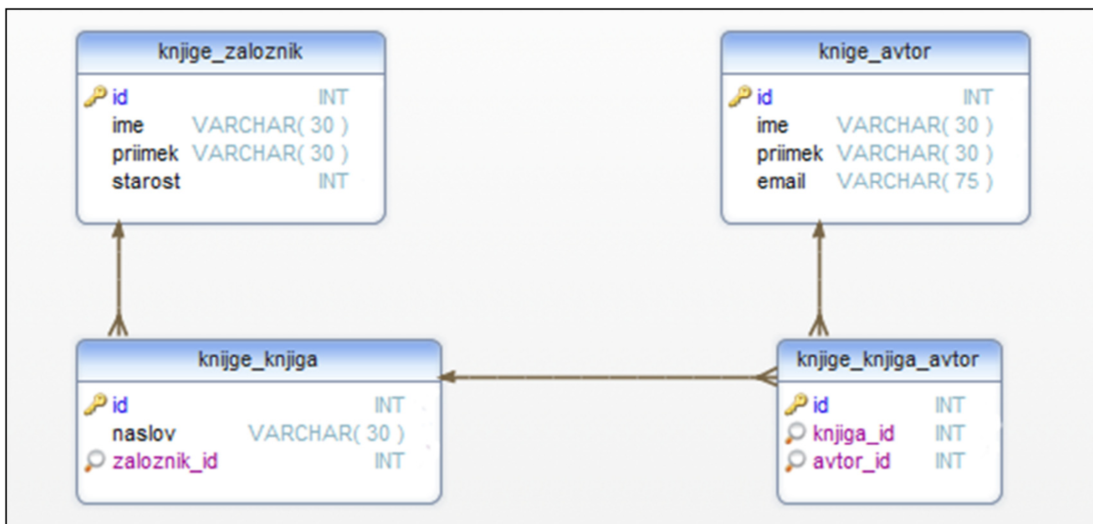
Datoteka models.py:

```
# Naložimo potrebne pakete.
from django.db import models

# Model, ki predstavlja založnika knjige
class Zaloznik(models.Model):
    ime=models.CharField(max_length=30)
    priimek=models.CharField(max_length=30)
    starost=models.IntegerField(blank=True, null=True)

# Model, ki predstavlja avtorja knjige
class Avtor(models.Model):
    ime=models.CharField(max_length=30)
    priimek=models.CharField(max_length=30)
    email=models.EmailField()

# Model, ki predstavlja knjigo
class Knjiga(models.Model):
    naslov=models.CharField(max_length=30)
    zaloznik=models.ForeignKey(Zaloznik)
    avtor=models.ManyToManyField(Avtor)
```



Slika 8: Izgled generiranega modela

2.7.2. Tipi atributov v modelu

Django pozna več vrst atributov. Izbira ustreznega je pomembna, saj se na podlagi tega pravilno generirajo tabele in stolpci v ciljni podatkovni bazi. Kot bomo videli v naslednjih poglavjih, se glede na izbrani tip atributa preverja tudi vsebina, ki jo vnašamo v to polje (npr. preverjanje pravilnosti elektronskega naslova). V Tabeli 5 je prikazanih nekaj osnovnih atributov.

Tip atributa	**Opis**
AutoField	Polje za cela števila (angl. *integer field*), ki se samodejno povečuje. To polje ni mišljeno za vnos podatkov. Uporablja se lahko za primarni ključ v tabeli.
BigIntegerField	Celoštevilsko polje (angl. *integer field*). Omogoča števila od $-(2^{36})$ do (2^{36}).
BooleanField	Polje z logično vrednostjo (*True/False*).
CharField	Polje za nize (angl. *strings*). Na spletni strani je prikazano pod oznako html <*input type='text'*>. To je polje za enovrstično besedilo.
TextField	Polje za dolge nize (angl. *strings*). Razlika med CharFieldom je ta, da je na spletni strani prikazan pod oznako html <*textarea*>. To je polje za večvrstično besedilo.
DateField	Polje za datume
DateTimeField	Polje za kombinacijo datum – čas
TimeField	Polje za čas
DecimalField	Polje za realna števila
FloatField	Polje za realna števila. Več o razliki med *DecimalField* na [5].
EmailField	Polje za elektronske naslove. Preverja se tudi pravilnost naslova.
FileField	Polje za datoteke
ImageField	Polje za slike – enako kot *FileField,* vendar preveri, če je datoteka res slika.
IntegerField	Polje za cela števila
PositiveIntegerField	Polje za cela pozitivna števila
IPAddressField	Polje za vnos naslovov ip. Preverja se pravilnost naslova ip.
SlugField	Polje, v katerem so dovoljene samo črke, številke, pomišljaji in podčrtaji.

Tabela 5: Tipi osnovnih atributov v modelu

Atributom lahko vnesemo tudi argumente. Nekateri argumenti so obvezni, drugi pa ne, saj ima Django določeno že privzeto vrednost. Nekateri so omejeni glede na podatkovno bazo. Tabela 6 prikazuje nekaj najosnovnejših argumentov.

Argument	Opis
null	Z argumentom se določi privzeto shranjevanje praznih vrednosti v podatkovno bazo. Če je nastavljen na *true,* se prazne vrednosti shranijo kot *null*, drugače pa kot privzeta vrednost izbranega podatkovnega tipa. Privzeto je nastavljen na *false*. Ta argument ne velja za nize, saj se prazni nizi vedno shranijo koz niz dolžine 0.
blank	Z argumentom določimo obveznost vnosa podatka. Privzeto je nastavljen na *False*.
db_column	Ime stolpca tabele v podatkovni bazi. Če ta argument ni podan, Django sam vnese ime stolpca po ključu <*ime aplikacije*>_<*ime atributa*>.
db_index	Django temu stolpcu v podatkovni bazi doda index.
primary_key	Z argumentom določimo, ali to polje uporabimo kot primarni ključ v podatkovni bazi.
unique	Z argumentom določimo, ali mora to polje biti enolično (angl. *unique*)
max_length	Največja dolžina (v znakih) vnesenih podatkov

Tabela 6: Tipi osnovnih argumentov atributov v modelu

Django pozna tudi množico atributov, ki predstavljajo povezavo (relacijo) med modeli. Z njimi zelo prihranimo pri času in si tudi olajšamo delo z definiranjem modelov, ki so v kakršni koli povezavi z drugim modelom (npr. knjiga ima lahko več avtorjev). Tabela 7 prikazuje nekaj osnovnih atributov.

Tip atributa	Opis
ForeignKey (tuj ključ)	Polje predstavlja tuj ključ. V podatkovni bazi, v tabeli v okviru ene relacije, je enak primarnemu ključu drugega modela oz. tabele. Kot argument mu obvezno moramo določiti, na kateri model se navezuje. Določimo mu lahko tudi polje, na katerega se navezuje, ime stolpca v podatkovni bazi, obveznost vnosa in dogodek ob brisanju zapisa v drugem modelu. Primer: knjiga ima založnika.
ManyToManyField	Polje povezuje dva modela. V podatkovni bazi, v tabeli v okviru ene relacije, je enak enemu ali več primarnim ključem drugega modela oz. tabele in obratno. V podatkovni povezavi je to definirano z vmesno tabelo. Primer: knjiga ima lahko enega ali več avtorjev, hkrati pa je avtor napisal več knjig.

Tabela 7: Tipi atributov v modelu, ki predstavljajo povezavo med modeli

2.7.3. Generiranje/brisanje tabel v podatkovni bazi

Django iz modela sam generira tabele v podatkovni bazi. Privzeto vsaki tabeli dodeli ime <*ime aplikacije*>_<*ime razreda*> (npr. *knjige_avtor*). Na podlagi izbranih tipov zna stolpcem določiti ustrezni podatkovni tip in druge atribute (primarne ključe, tuje ključe, obvezen vnos …). Če ni definirano drugače, vsakemu modelu dodeli tudi primarni ključ *id*.

Podatkovno bazo generiramo z ukazom:

```
manage.py syncdb.
```

Ukaz generira samo tabele, ki še ne obstajajo v podatkovni bazi. Generira tabele, ki jih Django rabi za svoje delovanje in tabele naših modelov. Ob prvem zagonu ukaza nas vpraša tudi za uporabniško ime, geslo in elektronski naslov super uporabnika (angl. *superuser*) oz. administratorja.

Slika 9: Tabele v podatkovni bazi

Django ob vsaki naknadni spremembi modela ne prilagodi oz. ne spremeni tabele v podatkovni bazi. Tega ne zna oz. zaradi vnesenih podatkov niti noče. Mislim, da je to velika pomanjkljivost. Zna pa to brezplačno orodje South, ki ga lahko naložimo kot razširitev Djanga. Brez tega orodja moramo vse spremembe v podatkovni bazi narediti ročno.

Ukaz *manage.py sqlall <ime aplikacije>* izpiše potrebne stavke sql za generiranje tabel izbrane aplikacije v podatkovni bazi. Zelo uporaben je pri spreminjanju modelov, saj naredimo izpis pred in po spremembi in na ta način na hitro vidimo razlike.

Primer:

```
manage.py sqlall knjige
```

```
BEGIN;
CREATE TABLE `knjige_zaloznik` (
    `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
    `ime` varchar(30) NOT NULL,
    `priimek` varchar(30) NOT NULL,
    `starost` integer
);

CREATE TABLE `knjige_avtor` (
    `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
    `ime` varchar(30) NOT NULL
);
```

```
CREATE TABLE `knjige_knjiga_avtor` (
    `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
    `knjiga_id` integer NOT NULL,
    `avtor_id` integer NOT NULL,
    UNIQUE (`knjiga_id`, `avtor_id`)
);

ALTER TABLE `knjige_knjiga_avtor` ADD CONSTRAINT
`avtor_id_refs_id_7b69579b`
FOREIGN KEY (`avtor_id`) REFERENCES `knjige_avtor` (`id`);

CREATE TABLE `knjige_knjiga` (
    `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
    `naslov` varchar(30) NOT NULL,
    `zaloznik_id` integer NOT NULL
);

ALTER TABLE `knjige_knjiga` ADD CONSTRAINT
`zaloznik_id_refs_id_6880ed51`
FOREIGN KEY (`zaloznik_id`) REFERENCES `knjige_zaloznik` (`id`);

ALTER TABLE `knjige_knjiga_avtor` ADD CONSTRAINT
`knjiga_id_refs_id_11acc4b8`
FOREIGN KEY (`knjiga_id`) REFERENCES `knjige_knjiga` (`id`);

COMMIT;
```

Ukaz *manage.py sqlclear <ime aplikacije>* izpiše potrebne stavke SQL za brisanje tabel izbrane aplikacije v podatkovni bazi. Pogosto se ga uporablja pri spreminjanju modelov, ko je potrebno vse tabele izbrane aplikacije izbrisati.

Primer:

```
manage.py sqlall knjige

BEGIN;
DROP TABLE `knjige_tabela2`;
DROP TABLE `knjige_tabela1`;
COMMIT;
```

2.7.4. Delo z modeli

Django vključuje API (programski vmesnik) za delo z modeli oz. interakcijo s podatkovno bazo. Programski vmesnik nam nudi hitro in enostavno delo s podatki. V ozadju sam sestavi potrebne stavke sql za dostop in spreminjanje podatkov. Delo s podatki je naloga kontrolerja, zato kodiramo v datoteke *views.py*.

Vstavljanje podatkov v podatkovno bazo poteka tako, da generiramo objekt, ki je instanca izbranega razreda. Atributi so naši navedeni atributi definiranega modela. Zapis v bazo se izvede šele ob klicanju medote *save()*.

Primer:

```
# Naložimo naš model.
from knjige.models import Avtor

# Generiramo objekt, ki je instanca razreda Avtor.
prviAvtor=Avtor(ime='Ime avtorja',priimek='Priimek avtorja')

# Poljubno lahko objektu spreminjamo njegove atribute.
prviAvtor.ime='ime'
prviAvtor.email='e-poštni naslov avtorja'



# Generiranje ustreznega stavka SQL in shranjevanje podatkov v
# podakovno bazo
prviAvtor.save()

prviAvtor.ime='Novo ime'

# Generiranje ustreznega stavka SQL in shranjevanje podatkov v
# podatkovno bazo
prviAvtor.save()
```

Django v zgornjem primeru, pri prvem klicanju metode *save()*, izvede stavek SQL *insert*. Pri drugem klicanju pa izvede stavek SQL *update*:

```
INSERT INTO `knjige_avtor` (`ime`, `priimek`, `email`)
VALUES ('ime', 'Priimek avtorja', 'e-postni naslov avtorja')

UPDATE `knjige_avtor`
SET `ime` = 'Novo ime', `priimek` = 'Priimek avtorja', `email` = 'e-
postni naslov avtorja'
WHERE `id` = 9
```

Branje vseh podatkov poteka z metodo *<ime modela>.objects.all()*. Ta metoda vrne seznam vseh zapisov v podatkovni bazi. Branje podatkov pa v večini primerov zahteva branje samo določenih zapisov. To storimo z *<ime modela>.objects.filter()*, kateri kot argument podamo enega ali več kriterijev filtriranja. Dobljeni zapisi se shranijo v seznam *Queryset*. Django dopušča tudi, da predloge te sezname sprejmejo kot novo spremenljivko. V predlogi nato lahko v zanko *for* kličemo atribute izbranega modela. Določimo lahko tudi vrstni red vrnjenih podatkov iz podatkovne baze z metodo *<ime modela>.objects.all().filter()*. Kot parameter pa podamo argument, po katerem želimo razvrstiti vrnjene podatke.

Primer:

```
vsiAvtorji=Avtor.objects.all()
avtorji1=Avtor.objects.filter(ime='Novo ime')
avtorji2=Avtor.objects.filter(priimek__icontains='Novo ime')
```

Privzeto se za filtriranje zapisov pri stavkih SQL uporabi znak = (enakost). Če je naštetih več atributov, pa se v stavku SQL uporabi logični operator in *(and)*. Django omogoča, da za atributom, po katerem iščemo, definiramo besedo, ki nadomesti znak enakosti. V Tabeli 8 je navedenih nekaj teh definiranih besed. Za uporabo logičnega operatorja ali *(or)* pa moramo uporabiti objekt Q. Več o objektu Q na [6].

Definirana beseda	Opis
__contains	Pomeni vsebovalnost. Preverja, če je niz vsebovan v vrednosti atributa. Loči velike in male črke. Primer: ime_contains='niz'
__icontains	Pomeni vsebovalnost. Preverja, če je niz vsebovan v vrednosti atributu. Ne loči velikih in malih črk. Zamenja se z %LIKE% stavkom SQL.
gt (greater than)	Pomeni *večji kot*. Uporablja se pri celih številih in realnih številih.
gte (less than or equal)	Pomeni *večji ali enako kot*. Uporablja se pri celih številih in realnih številih. Primer: starost__gte=3
lt (less than)	Pomeni *manjši kot*. Uporablja se pri celih številih in realnih številih.
lte (less than or equal)	Pomeni *manjši ali enako kot*. Uporablja se pri celih številih in realnih številih.
in »vsebovanost«	Preverja, če so vrednosti s seznama. Primer: starost in [12,34]
__startswith	Preverja, če se niz začne z določenim nizom. Loči velike in male črke. Primer: ime__startswith in [12,34]
__istartswith	Preverja, če se niz začne z določenim nizom. Ne loči velikih in malih črk. Primer: ime__istartswith in [12,34]
__isnull	Preverja vrednost atributa, če je enaka *null*.
__range	Preverja vrednost atributa, če je med določenima vrednostma. Primer: starost__range=(12,24)

Tabela 8: Nekaj osnovnih definiranih besed za filtriranje iskanja

3. PREDNOSTI IN SLABOSTI UPORABE OGRODIJ MVC PRI IZDELAVI SPLETNIH APLIKACIJ

Na trgu obstaja veliko različnih ogrodij MVC (v nadaljevanju ogrodij), pri katerih se za programiranje uporablja različne programske jezike. V tem poglavju bomo predstavili splošne prednosti in slabosti uporabe ogrodij za izdelavo spletnih aplikacij. Vsekakor pa ni cilj primerjava ogrodij med sabo, saj so nekatera primerna za splošno rabo, spet druga samo za določena področja. Nekatera ne potrebujejo niti namestitve, sama konfiguracija in programiranje pa potekata kar v brskalniku. Vsak izmed njih ima svoje prednosti in slabosti. Seznam bolj priljubljenih ogrodij, ki se na trgu za programiranje največ uporabljajo:

- Python: Django, TurboGears, Pylons, Web2Py, Zope, CubicWeb;
- Php: CakePhp, Drupal, Kohana, Zend, Codeigniter, Yii;
- Ruby: Ruby on Rails, Camping, Ramaze, Sinatra, Merb;
- Perl: Catalyst, Dancer, Mason, Egg, ClearPress, Squatting;
- Java: Spring, Turbine, Makumba, JPublish , Vroom;
- Asp.net: Asp.net mwc, Kentico, DotNetNuke, Monorail.

Osnovni in glavni cilj uporabe ogrodij je povečati kakovost in hitrost razvoja z uporabo vgrajenih orodij in metodologij, ki temeljijo na obstoječem jeziku. Pod kakovost štejemo tudi število vrstic, potrebnih za dosego cilja in preglednost kode. Manj vrstic pomeni hitrejši razvoj oz. manj izgube časa. Preglednost kode je dobrodošla, saj s tem hitreje odpravimo napake v programski kodi, lažje vzdržujemo kodo in lažje spreminjamo programsko kodo. Hitrost razvoja je v 21. stoletju še kako pomemben dejavnik, saj se število spletnih aplikacij in spletnih strani naglo povečuje.
Na kratko bi lahko rekli, da je cilj ogrodja pisati kvalitetnejšo kodo hitreje (angl. *write better code, faster*). Velikokrat pa zasledimo tudi DRY (angl. *don't repeat yourself*), kar pomeni »Ne ponavljaj programske kode«.

Ponudniki ogrodij poudarjajo, naj se ogrodje uporabi pri vsaki spletni aplikaciji oz. spletni strani, ki je bolj zapletena kot pa »*Hello World*«. Nobene potrebe ni, da bi nekaj na novo odkrivali oz. programirali. Ogrodja že vsebujejo veliko uporabnih in testiranih komponent oz. orodij, ki se zagotovo potrebujejo v spletni aplikaciji. S tem se izognemo izgubi pomembnega in dragocenega časa, saj nam ni treba iskati drugih knjižnic oz. jih programirati.
Slaba lastnost ogrodij je novo učenje. Naučiti se je treba uporabe ogrodja in načina programiranja. Struktura oz. zgradba kode se razlikuje od programiranja po klasični poti z uporabo skriptnih jezikov. Če pa smo prisiljeni k uporabi ogrodja, ki uporablja nam neznan programski jezik, pa se tega jezika moramo predhodno naučiti.
Za enostavne aplikacije oz. spletne strani je potrebno pretehtati smiselnost uporabe ogrodja, saj s tem ne pridobimo nič, kar bi nam dalo prednost pred klasičnim programiranjem spletnih aplikacij.

Razna podjetja, ki se ukvarjajo z razvijanjem spletnih aplikacij, si sprogramirajo oz. prilagodijo kar svoje lastno ogrodje. Dobra lastnost tega je, da ogrodje vsebuje samo potrebna orodja in knjižnice. S tem vplivamo na samo »težo« ogrodja. Slaba lastnost pa je, da na trgu ljudje takšnega ogrodja ne poznajo, kar ob novih zaposlitvah v podjetju pomeni dodatno učenje. Lastna ogrodja so nastala predvsem pred pojavom številnih prosto dostopnih ogrodij.

3.1. Primerjava programske kode

Datotečna struktura datotek in map je pri vsakemu ogrodju posebej standardizirana oz. določena. To pripomore k določeni urejenosti datotek in map. S tem se izognemo zmedi datotek, saj vedno vemo, kje se mora določena programska koda nahajati. To je dobrodošlo predvsem takrat, ko je zaradi obsežnosti ali kompleksnosti projekta v razvoj vključenih veliko ljudi. Seveda lahko pri klasičnem programiranju ohranimo urejenost datotek, vendar moramo biti prizadevni.

Kot že omenjeno, so ogrodja sestavljena iz treh plasti. To so model, kontroler in pogled. Vsaka plast predstavlja celoto in vsaka plast se nahaja v ločeni datoteki. To pomeni, da v projektu lahko hkrati sodeluje več ljudi. Oblikovalci grafične podobe spletne aplikacije lahko pripravljajo osnovno podobo spletne aplikacije, hkrati pa lahko programerji že pripravljajo strukturo modela oz. podatkovno bazo. Ko sta model in podatkovna baza pripravljena, lahko programerji začnejo programirati, saj je programska koda ločena od podobe oz. izgleda spletne aplikacije. To za oblikovalce predstavlja veliko lažje delo, saj je struktura datoteke brez programske kode preglednejša, hkrati pa oblikovalci kode niti ne poznajo dovolj dobro. Odvečna programska koda jih pri njihovem delu le ovira.
Drugače je pri klasičnem programiranju, saj se vsaka podstran v spletni aplikaciji nahaja v ločeni datoteki ali datotekah. Programer lahko le delno loči programsko kodo in kodo HTML.

Programska koda je zaradi ogrodja preglednejša, čistejša in krajša. Ogrodja imajo že veliko vgrajenih funkcij in orodij, ki programerjem olajšajo delo. Tem pravimo programski vmesniki (angl. *Application programming interface*, krajše API). Nekaj osnovnih možnosti, ki jih ponujajo programski vmesniki v ogrodjih:

- avtentikacija, avtorizacija in registracija uporabnikov,
- administracija spletne aplikacije,
- varnost in pravice uporabnikov,
- varnost spletne aplikacije,
- delo s spletnimi obrazci (angl. *forms*),
- delo s podatki in validacija podatkov,
- delo z datotekami (xml, pdf …),
- podpora za razrede middleware,
- lažje delo z razhroščevanjem oz. iskanjem napak.

Ob pravilni uporabi ogrodja vneseni podatki ohranjajo konsistentnost vse od vnosa do zapisa v podatkovno bazo. Za to poskrbijo že sama ogrodja, le pravilno je treba definirati model. Primer prikazuje Slika 10. Na podlagi modela vsebino spletnega obrazca definira že sam kontroler. V pogledu pa nato ta spletni obrazec samo kličemo. S tem skrajšamo programsko kodo, hkrati pa nam ni potrebno preverjati pravilnosti podatkov. Ogrodje že na podlagi modela ve, da mora npr. polje »starost« imeti celoštevilsko vrednost, elektronski naslov pa mora biti oblike veljavnega elektronskega naslova. Ob neuporabi ogrodja moramo za konsistentnost podatkov poskrbeti sami in lahko se zgodi, da na kaj pozabimo.

```
<form name="registracija" action="/registracija" method="get" >
{{form.as_p}}
<input type="submit" value="Potrdi" />
</form>
```

```
<form name="registracija" action="/registracija" method="get" >
<p>
	<label for="id_name">Ime:</label>
	<input id="id_name" type="text" name="ime" maxlength="100" />
</p><p>
	<label for="id_starost">Starost:</label>
	<input id="id_starost" type="text" name="starost" maxlength="100" />
</p><p>
	<label for="id_email">Email:</label>
	<input id="id_email" type="text" name="email" maxlength="100" />
</p>
<input type="submit" value="Potrdi" />
</form>
```

Ime:

Starost:

Email:

Potrdi

Slika 10: Primer krajše kode ob uporabi ogrodja

Ogrodja imajo že privzeto nastavljeno, da so internetni naslovi spletnih strani prijaznejši in lepši. Primer prikazuje Slika 10. To ni pomembno le za obiskovalca naše spletne aplikacije, ampak tudi za razne spletne iskalnike, kot je npr. Google. Ti internetni naslovi spletnih strani pripomorejo k optimizaciji spletne aplikacije oz. spletne strani. Optimizacija spletne aplikacije je postopek izboljšanja vidljivosti spletnih strani na iskalnikih prek naravnih oz. neplačanih iskalnih rezultatov. Če ne uporabimo ogrodja, to še ne pomeni, da takih internetnih naslovov ne moremo uporabiti. Pomeni pa, da je za nastavitev potrebnega več časa in po potrebi tudi programiranja ali pa uporaba knjižnic drugih programerjev.

<stran>/uporabnik/1

<stran>/uporabnik/getuser.php?id=1

Slika 11: Primer lepšega internetnega naslova spletne strani pri uporabi ogrodja

Kot pri vsakem programiranju tudi v programiranju spletnih aplikacij prihaja do napak.
Za vsakega programerja je pomembno, kako hitro poteka odkrivanje napak v programski kodi oz. v spletni aplikaciji. Napake v programski kodi se pojavljajo predvsem med razvojem spletne aplikacije. Pomembno je, da jih ogrodje prikaže programerju na čim bolj pregleden način in da mu s tem sporoči čim več koristnih informacij. Tako programer že takoj lahko vidi, kje in zakaj je prišlo do napake. Slika 11 prikazuje, na kakšen način napake sporoči ogrodje Django.

localhost:8000

TypeError at /

index() takes exactly 2 arguments (1 given)

Request Method: GET
Request URL: http://localhost:8000/
Django Version: 1.5
Exception Type: TypeError
Exception Value: index() takes exactly 2 arguments (1 given)
Exception Location: c:\Python26\Lib\site-packages\django\core\handlers\base.py in get_response, line 111
Python Executable: C:\Python26\python.exe
Python Version: 2.6.0
Python Path: ['C:\\Users\\david.frlic\\Documents\\eclipse\\testiranjeDjango', 'C:\\Python26\\lib\\site-packages\\distribute-0.6.27-py2.6.egg', 'C:\\Python26\\python26.zip', 'C:\\Python26\\DLLs', 'C:\\Python26\\lib', 'C:\\Python26\\lib\\plat-win', 'C:\\Python26\\lib\\lib-tk', 'C:\\Python26', 'C:\\Python26\\lib\\site-packages']
Server time: pet, 27 Jul 2012 15:48:14 +0200

Traceback Switch to copy-and-paste view

c:\Python26\Lib\site-packages\django\core\handlers\base.py in get_response

```
111.                response = callback(request, *callback_args, **callback_kwargs)
```

▶ Local vars

Request information

GET No GET data
POST No POST data
FILES No FILES data
COOKIES No cookie data
META

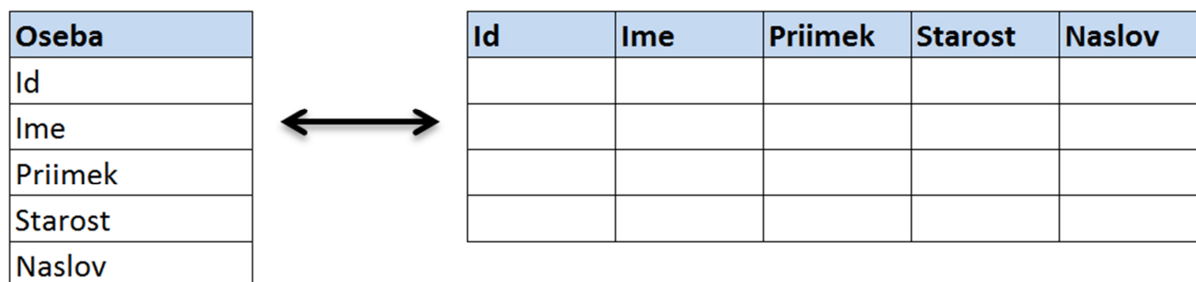
Variable	Value
TMP	'C:\\Users\\DAVID~1.FRL\\AppData\\Local\\Temp'

Slika 12: Prikaz napake v spletni aplikaciji ob uporabi ogrodja Django

Ko spletna aplikacija preide v produkcijsko okolje, je prav tako pomembno beleženje in javljanje napak. Ob napaki se mora prikazati spletna stran, s katero aplikacija sporoči le to, da je na strani prišlo do napake. Obiskovalcu spletne strani se zaradi varnosti oz. zaradi možnosti vdora ne sme prikazati spletna stran, kot prikazuje Slika 11. S tem lahko razkrije preveč informacij. Pomembno pa je, da se napaka zabeleži in da je upravitelj spletne aplikacije obveščen bodisi po elektronski pošti ali kako drugače. Ogrodja privzeto omogočajo javljanje napak, le definirati je potrebno, kdaj in na kakšen način. Pri klasičnem programiranju tega privzeto nimamo omogočeno. Vse je potrebno sprogramirati oz. uporabiti knjižnice drugih oseb. To pa pomeni, da lahko pri programski kodi pride do napak, saj ni testirana kot pri ogrodjih.

3.2. Podatki in baza podatkov

Pri spletnih aplikacijah so podatki in delo s podatki ključnega pomena. Ogrodja (oz. večina ogrodij) uporabljajo t. i. koncept ORM.
Objektno relacijska preslikava (angl. *Object relational mapping*, krajše ORM) je koncept, ki razvijalcem v objektno orientiranih okoljih omogoča, da v svojih aplikacijah upravljajo s podatki na povsem domač način, kar preko objektov, tako kot da bi delali z objektno podatkovno bazo in ne z relacijsko. To omogoča abstrakcijski sloj ORM, ki poskrbi za brezšivno in transparentno preslikovanje iz okolja objektov v relacijsko podatkovno bazo in nazaj.
Ideja objektno-relacijskega preslikovanja se je pojavila še pred večjim razmahom objektno orientiranega programiranja v devetdesetih letih prejšnjega stoletja, a kljub temu je še dolgo ostala le v domeni akademskih projektov in redkih komercialnih sistemov. Pred nekaj leti se je predvsem po zaslugi nekaterih odprtokodnih projektov ORM razširil praktično na vsako platformo, danes pa je tudi integralen del vsakega sodobnega ogrodja za razvoj spletnih aplikacij.



Slika 13: Objektno-relacijska preslikava

Kot je že bilo omenjeno, upravljanje podatkov poteka kar preko objektov. Objekti so definirani v modelu. Pri nekaterih ogrodjih se struktura teh podatkov samodejno preslika v podatkovno bazo. Pri drugih moramo to storiti ročno. To pa storimo tako, da v nastavitvah navedemo, v katera polja se objekt preslikava v podatkovno bazo. Zamenjava podatkovne baze je med razvojem enostavna. Programirati lahko začnemo, še preden se stranka odloči, katero podatkovno bazo bo uporabljala. Prehod na drugo podatkovno bazo pa zahteva samo spremembo v nastavitvah.

Prednosti pri uporabi objektno-relacijske preslikave:
- naravno delo s podatki;
- manj kode, ta je čistejša;
- hitrejši razvoj spletne aplikacije;
- neodvisnost od podatkovne baze;
- lažje vzdrževanje in dograjevanje.

Slabosti pri uporabi objektno-relacijske preslikave:
- če je dostop do baze podatkov neposreden, potem imajo razvijalci nekaj nadzora nad poizvedbami oz. delovanjem – s tem lahko vplivajo na zmogljivost;
- potrebno se je naučiti in razumeti sam koncept ORM, kajti vsaka tehnologija ORM uporablja drugačen programski vmesnik (API).

Pri klasičnem programiranju (brez ogrodja) moramo podatkovno bazo kreirati ročno s stavki SQL ali pa z uporabo raznih aplikacij (npr. PhpMyAdmin). V celotnem procesu razvoja moramo pisati stavke SQL. To pa pomeni veliko ponavljanja in veliko možnosti za razne napake. Slika 14 prikazuje branje zapisa iz podatkovne baze in dodajanje zapisa v podatkovno bazo z uporabo in brez uporabe ogrodja.

PRIMER UPORABE BREZ OGRODJA (PHP)

```
<?php
$username="username";
$password="password";
$database="your_database";

mysql_connect(localhost,$username,$password);
@mysql_select_db($database) or die( "Unable to select database");

//primer 1: Branje zapisa iz podatkovne baze
$result=mysql_query("SELECT name FROM osebe WHERE id=3" );
$row = mysql_fetch_row($result);
$ime = row[0];

//primer 2: Dodajanje novega zapisa
$ime='david'
$priimek='frlic'
$starost=20;
mysql_query("INSERT INTO osebe (name,surname,age) VALUES
        ('".$ime."','".$priimek."',".$starost.")");

mysql_close();
?>
```

PRIMER UPORABE Z OGRODJEM (DJANGO)

```
//primer 1: Branje zapisa iz podatkovne baze
p=Oseba.objects.get(id=3)
ime=p.name


//primer 2: Dodajanje novega zapisa
ime='david'
priimek='frlic'
starost=20;
p=Oseba(name=ime,surname=priimek,age=starost)
p.save()
```

Slika 14: Primer programiranja z uporabo ogrodja in brez njega

3.3. Administracijsko okolje

Vsaka spletna aplikacija z dinamično vsebino rabi administratorja oz. skrbnika. Ta za potrebe vnašanja vsebine, upravljanja spletne aplikacije in upravljanja z uporabniki uporablja administracijsko okolje.

Če je spletna aplikacija pravilno razvita, je z uporabo administracijskega okolja na spletni aplikaciji možno urejanje katere koli vsebine. Nobene potrebe ni po poseganju v samo kodo spletne aplikacije. Omogočeno mora biti dodajanje, spreminjanje in brisanje vsebine kot tudi večpredstavnostne vsebine. Vsebina je shranjena v podatkovni bazi. Koristno pa je tudi, če administracijsko okolje v tej spletni aplikaciji omogoča tudi urejanje menijev.
Omogočeno mora biti menjanje vrstnega reda menijev, dodajanje, brisanje in skrivanje menijev.

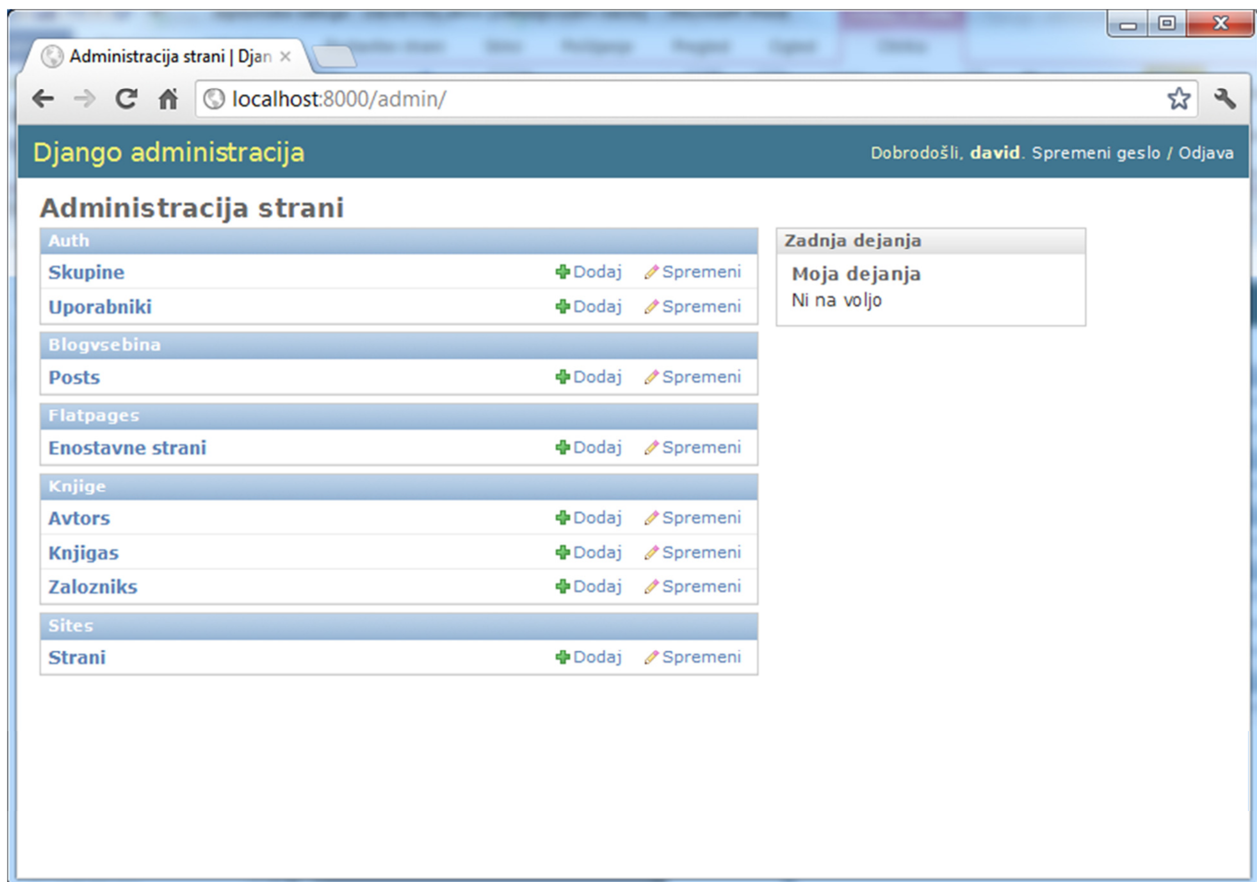
Dobra ogrodja omogočajo samodejno generiranje administracijskega okolja iz našega definiranega modela. To pomeni, da se na podlagi definiranega modela samodejno generira okolje, ki omogoča dodajanje, brisanje in spreminjanje definiranih objektov. Ti lastno definirani objekti pa so lahko npr. vsebina naše spletne aplikacije, košarica v spletni prodajni trgovini ali pa šifrant partnerjev.

Upravljanje spletne aplikacije preko administracijskega okolja je tudi pomembna in dobra lastnost spletnih aplikacij. Možnosti, ki jih ogrodja bolj ali manj omogočajo:

- pregled nad statusom spletne aplikacije (pregled raznih napak …);
- pregled statusa strežnika (npr. obremenjenost procesorja);
- pregled in spreminjanje raznih glavnih nastavitev spletne aplikacije (npr. naslov poštnega strežnika);
- opozarjanje na razne varnostne posodobitve ogrodja;
- pregled nad prijavami uporabnikov.

Upravljanje z uporabniki pomeni, da administrator lahko dodaja, briše in spreminja podatke uporabnikov. Dodajanje uporabnikov lahko poteka tudi preko spletnega obrazca. V nekaterih ogrodjih administrator v nastavitvah lahko vključi ali izključi ta način registracije. Spreminja lahko razne podatke, kot so ime, priimek, uporabniško ime in elektronski naslov. Administrator lahko uporabnikom dodeljuje tudi pravice. To so lahko pravice v spletni aplikaciji ali pa pravice v administratorskem okolju. V administracijskem okolju so to razne pravice nad odseki spletne aplikacije ali pa pravice nad objekti, ki smo jih definirali v samem modelu.

Vse to omogočajo večinoma vsa ogrodja. Nekatera omogočajo več, spet druga manj. Nekatera ogrodja, ki tega ne vključujejo, omogočajo to preko raznih razširitev, vtičnikov ali generatorjev. Pri klasičnem programiranju ni vključeno nič od tega. To pomeni, da je vse potrebno sprogramirati ali pa mora s spletno aplikacijo upravljati programer, ki vse spremembe spremeni v programski kodi. Če skrbnik spletne aplikacije nima tega znanja, mora za ta opravila najeti oz. plačati programerja. Tudi za programerja je z uporabo administratorskega okolja upravljanje veliko lažje. Pri razvoju lastnega administracijskega okolja za določeno spletno aplikacijo pa se podaljšata čas in posledično tudi cena samega razvoja spletne aplikacije.

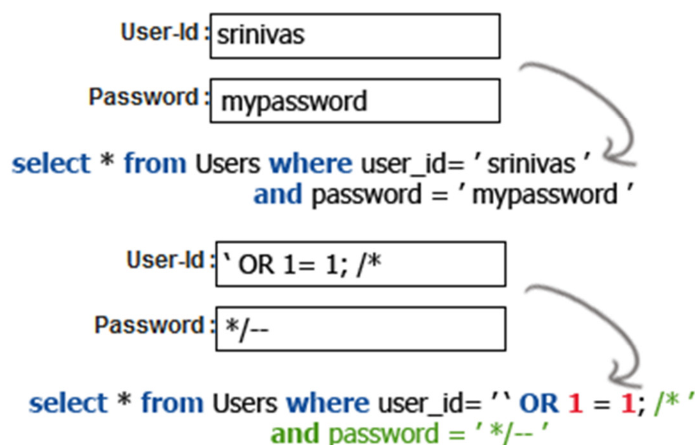


Slika 15: Administracijsko okolje v Djangu

3.4. Varnost spletnih aplikacij

Varnost spletnih aplikacij je pomemben faktor. Vsaka spletna aplikacije bi morala biti čim bolj zavarovana pred spletnimi napadalci oz t. i. hekerji. V praksi se izkaže, da je to zelo težko. Vsak programer se mora zavedati pomena varnosti. Varnost spletne aplikacije bi morala biti prioriteta vsakega programerja. Tukaj govorimo o napakah v programski kodi, ki so rezultat slabega programiranja. Programer bi moral poskrbeti za vse varnostne luknje (napake v programski kodi), kajti spletni napadalec za napad rabi samo eno varnostno luknjo. Spletni napadalci oz. hekerji so ljudje, ki se ukvarjajo z vdori oz. nepooblaščenimi dostopi v računalniška omrežja. Te osebe iz različnih vzrokov zaobidejo varnostne sisteme. Ti razlogi so lahko npr. zaslužek, aktivizem, pridobivanje informacij ali pa iskanje pomanjkljivosti v varnostnih sistemih. Predvsem so za spletne napadalce priljubljena tarča zbirke podatkov, ki so lahko prava zakladnica zaupnih podatkov.

Ogrodja imajo pred klasičnim programiranjem veliko prednost, saj so različni »varnostni mehanizmi« vgrajeni že v samo ogrodje. Poglejmo si nekaj osnovnih ranljivosti, ki jih ogrodja že poznajo in nas pred njim ustrezno zavarujejo.
Vrivanje stavkov SQL (SQL injection) [7]: gre za napad, pri katerem med podatke, ki jih aplikacije prejmejo od uporabnika, napadalec vrine del stavka SQL in s tem spremeni osnovno delovanje ukaza. Primer prikazuje Slika 16 [8].



Slika 16: Vrivanje stavkov SQL

Vrivanje stavkov SQL je mogoče le zaradi napak v spletni aplikaciji. Konkretno gre za pomanjkljivost, ki je vir številnih težav, s katerimi se spopadajo razvijalci spletnih aplikacij: (ne)preverjanje vnosnih podatkov (input/data validation). Torej se uporabnikov vnos ne preverja, temveč se neposredno uporabi za generiranje stavka SQL. Pričakovati od uporabnikov, da bodo v polja (input field) vnašali le dovoljene znake, je seveda utopično.

Napad XSS (angl. *cross site scripting*): je napad na spletno aplikacijo z vrivanjem zlonamerne kode, napisane v skriptnem jeziku. Spletni napadalec lahko kot vhodni podatek pošlje na spletno stran zlonamerno kodo le, če spletna stran ne preverja vnešenih podatkov. Na ta način lahko napadalec ukrade podatke iz uporabniške seje, pokvari vsebino prikazane spletne strani, zažene škodljive programe itd. Ta napad izkorišča uporabnikovo zaupanje v spletno aplikacijo. Rešitev pred tovrstnim napadom je, da so vsi vhodni podatki ustrezno prečiščeni [9].

Ponarejanje spletnih zahtev (angl. *cross site request forgery*): je napad, pri katerem spletni napadalci uporabnikov brskalnik lahko prisilijo v pošiljanje neželenih zahtevkov HTTP z uporabo internetnih spletnih naslovov. Zahteva HTTP se nato izvede v uporabnikovem brskalniku z vsemi trenutnimi pravicami tega uporabnika. S tem se lahko izvedejo različna dejanja kot so nakup v spletni trgovini ali pa spreminjanje in odstranjevanje različnih informacij. Ta napad izkorišča zaupanje spletne aplikacije v uporabnika, ker zahteva HTTP izgleda upravičena in spletna aplikacija težko določa, če jo je uporabnik res nameraval izvesti [10].

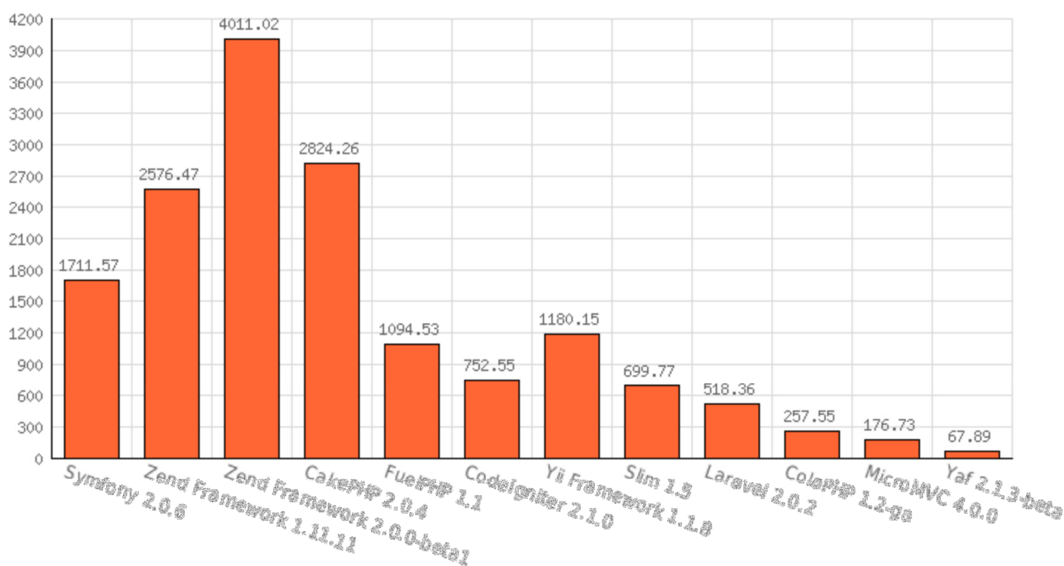
Branje in pisanje v datoteke izven okolja spletne aplikacije (angl. *directory traversal attack*): s tem napadom si spletni napadalci lahko pridobijo vpogled v različne datoteke na strežniku (npr. datoteko z uporabniškimi imeni in gesli na operacijskem sistemu Linux). Napad pa deluje tako, da spletni napadalec v različna polja ali kako drugače vnese pot do želene datoteke. Pred to datoteko mora vnesti enega ali več ponovitev besedila »../«. To besedilo pa pomeni »mapa višje od trenutne v datotečni strukturi«. Če spletna aplikacija predvideva, da v polju, v katerega je spletni napadalec vnesel to besedilo, mora biti navedena pot do datoteke, potem jo tudi odpre [11].

Pri klasičnem programiranju moramo za obrambo proti tem napadom poskrbeti sami. To pa pomeni, da za to porabimo več časa in moramo napisati več vrstic programske kode. Ne

smemo pa tukaj pozabiti omeniti še človeško pozabljivost in neznanje programerjev. Vedeti moramo, da spletni napadalec za napad rabi samo eno varnostno luknjo.

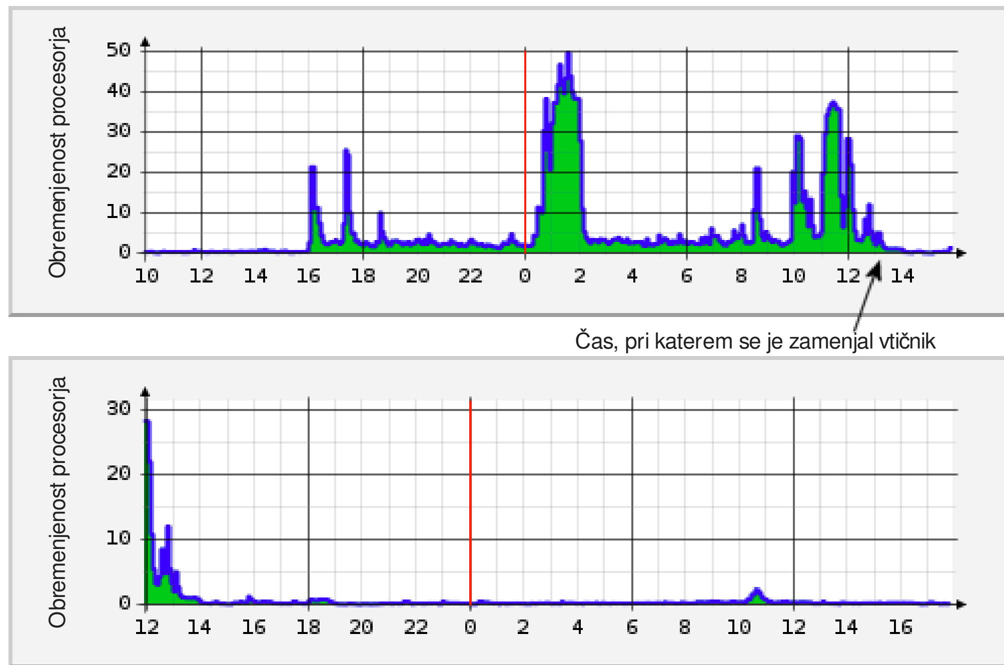
3.5. Primerjava zmogljivosti v produkcijskem okolju

Ko spletna aplikacija preide v produkcijsko okolje, je pomembno, da čim manj obremenjuje strežnik. To pomeni, da mora spletni strežnik, ki poganja to spletno aplikacijo, porabiti čim manj procesorske moči in delovnega pomnilnika (angl. RAM). To je pomemben faktor predvsem pri strežnikih, na katerih se poganja več spletnih aplikacij. Obremenjenost je seveda pogojena z obiskanostjo spletne aplikacije. Na spletu je mogoče najti veliko primerjav med ogrodji. Primer prikazuje Slika 17 [12]. Ta slika prikazuje primerjavo uporabe delovnega pomnilnika med različnimi ogrodji. S slike je razvidno, da lahko prihaja do velikih razlik med ogrodji že pri isti spletni aplikaciji, napisani v različnih ogrodjih. Do razlik prihaja predvsem zaradi zgradbe ogrodij in njihove »teže«, saj različna ogrodja vključujejo različne programske vmesnike in odvečno programsko kodo.
Klasično programiranje ima prednost, saj spletne aplikacije porabijo manj procesorske moči in delovnega pomnilnika. Vzrok je predvsem v »lahkosti« spletne aplikacije, saj je vanjo vključena samo programska koda, ki se je potrebna za pravilno delovanje.



Slika 17: Uporaba delovnega pomnilnika (v kB) med različnimi ogrodji

Do razlik lahko prihaja tudi zaradi različnih kompatibilnosti ogrodij s spletnimi strežniki. Primer prikazuje Slika 18 [13]. Slika prikazuje primerjavo obremenjenosti procesorja strežnika. Za primerjavo se je uporabilo isto spletno aplikacijo, napisano v ogrodju Django, pri uporabi različnega vtičnika. Ta vtičnik je potreben za pogon spletne aplikacije na spletnem strežniku Apache. Primer prikazuje menjave vtičnika *mod_python* z vtičnikom *mod_wsgi*.



Slika 18: Primerjava obremenjenosti procesorja strežnika pri uporabi različnih vtičnikov spletnega strežnika

4. ZAKLJUČEK

V današnjem času oz. v informacijski dobi so zahteve po vse bolj naprednih in prilagodljivih internetnih strani vse večje. Razvoj takih internetnih strani je postal vse bolj kompleksen in zahtevnejši za programerje. Zato so se na trgu pojavile nove rešitve in tehnologije.

V diplomski nalogi sem najprej podrobneje predstavili sistem za hiter razvoj spletnih aplikacij. Predstavil sem Django, ki je odprtokodno visokonivojsko Pythonovo ogrodje za hiter razvoj spletnih aplikacij. Obravnavana sta bila osnovna zgradba in delovanje tega ogrodja, kar je pripomoglo k razumevanju hitrega razvoja spletnih aplikacij. Osnovni in glavni cilj uporabe ogrodij je povečati kakovost in hitrost razvoja. To lahko potrdimo, saj smo spoznali orodja in vgrajene funkcije, ki programerjem ne le olajšajo delo, temveč tudi pospešijo razvoj spletnih aplikacij.

V drugem delu sem predstavil splošne prednosti in slabosti uporabe ogrodij MVC pri razvoju spletnih aplikacij. Za splošne prednosti gre zato, ker bi raziskovanje in testiranje vseh ogrodij MVC presegalo okvire diplomske naloge. Predstavil sem:

- splošne prednosti in slabosti uporabe ogrodij MVC,
- primerjavo programske kode,
- prednosti programskih vmesnikov ogrodij MVC,
- prednosti in slabosti pri delu s podatkovno bazo,
- prednosti pri uporabi administracijskega okolja spletnih aplikacij,
- primerjavo varnosti spletnih aplikacij,
- primerjavo zmogljivosti spletnih aplikacij.

Cilj pa nikakor ni bila primerjava ogrodij MVC med sabo, saj na trgu obstaja veliko različnih ogrodij MVC. Vsako ima svoje prednosti in slabosti. Prišel sem do ugotovitve, da je za razvoj spletnih aplikacij uporaba ogrodij MVC ne le priporočljiva, temveč že skoraj obvezna.

VIRI

[1] Setting Path on Windows. Dostopno na:
http://java.com/en/download/help/path.xml

[2] HttpRequst objekt v Djangu. Dostopno na:
https://docs.djangoproject.com/en/dev/ref/request-response/#django.http.HttpRequest

[3] Custom template tags and filters. Dostopno na:
https://docs.djangoproject.com/en/dev/howto/custom-template-tags

[4] Built in template tags and filters. Dostopno na:
https://docs.djangoproject.com/en/dev/ref/templates/builtins/?from=olddocs

[5] Decimal fixed point and floating point arithmetic in Python. Dostopno na:
http://docs.python.org/library/decimal.html#decimal

[6] Making queries. Dostopno na:
https://docs.djangoproject.com/en/dev/topics/db/queries

[7] Sql injection. Dostopno na:
http://en.wikipedia.org/wiki/SQL_injection

[8] Sql injection example. Dostopno na:
http://0ux.drimateo.uni.me

[9] Cross site scripting. Dostopno na:
http://en.wikipedia.org/wiki/Cross-site_scripting

[10] Cross site request forgery. Dostopno na:
http://en.wikipedia.org/wiki/Csrf

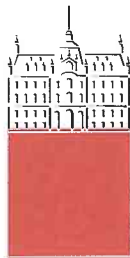
[11] Directory traversal attack. Dostopno na:
http://en.wikipedia.org/wiki/Directory_traversal_attack

[12] Php framework mvc benchmark. Dostopno na:
http://www.ruilog.com/blog/view/b6f0e42cf705.html

[13] mod_python versus mod_wsgi . Dostopno na:
http://collingrady.wordpress.com/2009/01/06/mod_python-versus-mod_wsgi

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si



Št. naloge: 00319/2012

Datum: 03.09.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DAVID FRLIC**

Naslov: **RAZVOJ SPLETNIH APLIKACIJ Z ODPRTOKODNIM OGRODJEM**

DEVELOPMENT OF WEB APPLICATIONS WITH OPEN SOURCE FRAMEWORK

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Proučite odprtokodno ogrodje DJANGO, ki je ogrodje za razvoj spletnih aplikacij. Z uporabo ogrodja DJANGO razvijte spletno aplikacijo in na podlagi izkušenj pri razvoju izdelajte primerjavo med klasičinm razvojem spletnih aplikacij in razvojem spletnih aplikacij z uporabo ogrodja.

Mentor:

doc. dr. Rok Rupnik

Dekan:

prof. dr. Nikolaj Zimic