

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

OPERACIJSKI SUSTAVI 2

Sigurnost Django web razvojnog okruzenja

Melita Mihaljevic

Voditelj: *Marin Golub*

Zagreb, Prosinac, 2007.

Sadržaj

1. Uvod.....	4
2. Django web razvojno okruženje.....	5
2.1 Razvoj Django web okruženja.....	5
2.2 Svojstva Django razvojnog okruženja.....	5
2.2.1 MTV.....	6
2.2.2 Konfiguracijska datoteka URL.....	7
2.2.3 Sučelje za administraciju.....	7
2.2.4 Django razvojni poslužitelj.....	7
2.3 Izvođenje web aplikacije izgrađene pomoću Django.....	8
2.4 Programski jezik Python.....	8
3. Razvoj web aplikacija pomoću Django razvojne okoline.....	10
3.1 Postavljanje Django web razvojnog okruženja.....	10
3.1.1 Instalacija potrebnih alata.....	10
3.1.2 Instalacija Django alata.....	11
3.2 Stvaranje projekta i web aplikacije pomoću Django.....	13
3.2.1 Alat za administriranje(eng.admin tool).....	13
3.2.2 Stvaranje projekta.....	13
3.2.3 Dodavanje aplikacija u projekt.....	14
3.2.4 Uređivanje postavki.....	15
3.3 Stvaranje modela.....	17
3.4 Konzolna aplikacija shell.....	19
3.5 Administatorska stranica.....	20
3.6 Kreiranje sheme URL-ova.....	22
3.7 Stvaranje pogleda.....	24
3.8 Stvaranje predloška.....	24
4. Sigurnost web aplikacija stvorenih pomoću Django.....	27
4.1 SQL injection.....	27
4.2 XSS.....	28
4.3 CSRF.....	28

4.4 Napadi otimanja sjednice.....	29
4.5 Umetanje zaglavlja e-maila (E-mail header injection).....	30
4.6 Zaobilaženje direktorija (Directory traversal).....	30
5. Praktični rad.....	32
5.1 Stvaranje blog aplikacije.....	32
6. Zaključak.....	41
7. Literatura.....	42

1. Uvod

Krajem 20-og, početkom 21. stoljeća sve veća potreba korisnika za pristupom raznolikom sadržaju putem Interneta dovela je do ubrzanog razvoja internetskih tehnologija. Internetske tehnologije su spoj mrežne infrastrukture i programskih rješenja koja omogućuju komunikaciju putem mreže Internet. Potreba za uvođenjem dinamičkog sadržaja u web aplikacije dovela je do razvoja složenih internetskih tehnologija. U nastavku seminara web aplikacijama smatraju se aplikacije koje omogućuju prikaz dinamičkog sadržaja.

Dinamički sadržaj podrazumijeva podatke na web stranici koji se mijenjaju za vrijeme korištenja web stranice. Promjene su zasnovane na podacima iz baze podataka i korisnikovim zahtjevima za dohvatom informacija. Promjene na web stranici mogu biti: promjene u strukturi HTML stranice, uključivanje novog sadržaja u postojeću web aplikaciju dohvaćanjem sadržaja iz baze podataka ili generiranjem sadržaja pomoću složenih funkcija. Postupak izgradnje dinamičkih web stranica složen je i dugotrajan posao. Današnje web aplikacije su velike i složene te je njihova izgradnja podložna pogreškama te kao takve izgrađene web aplikacije su nesigurne. Nesigurnost web aplikacija očituje se u mogućnosti napada na njih, dohvaćanje i promjenu sadržaja ili potpuno onemogućavanje dostupnosti web aplikacije.

Radi pojednostavljenja izgradnje složenih dinamičkih web aplikacija razvijena su web razvojna okruženja (eng. *web application framework*). Web razvojna okruženja su programska okruženja koja podržavaju razvoj dinamičkih web stranica, web aplikacija i web poslužitelja. Web razvojna okruženja sastoje se od niza aplikacija koja omogućuju brži i jednostavniji razvoj složenih sustava. Osim niza aplikacija koje omogućavaju jednostavnu i intuitivnu izgradnju web aplikacije, često u sebi imaju ugrađene mehanizme za prevenciju sigurnosnih napada. Najčešće korišten mehanizam je onemogućavanje kombiniranja programskog koda koji opisuje logiku web aplikacije, kôda koji opisuje prikaz i dijela koji pristupa bazi podataka. Pojedini dijelovi izgrađuju se međusobno nezavisno te se na taj način mogućnost napada na web aplikacije svodi na minimum.

Danas postoji velik broj web razvojnih okruženja kao na primjer ASP.NET, Django, Ruby on Rails i drugi. U okviru seminara obrađen je Django, web razvojno okruženje napisano u programskom jeziku Python. Prikazan je način izgradnje web aplikacija i navedene su prednosti i mane prilikom korištenja Django razvojnog okruženja. U prvom dijelu seminara opisano je Django razvojno okruženje te je na jednostavnom primjeru prikazana izgradnja web aplikacije pomoću Djanga. Drugi dio daje osvrt na sigurnost web aplikacija izgrađenih pomoću Djanga. U posljednjem dijelu seminara prikazana je izgrađena blog web aplikacija.

2. Django web razvojno okruženje

Django je slobodno dostupno web razvojno okruženje napisano u programskom jeziku Python. U sljedećim poglavljima opisana su svojstva Django web razvojnog okruženja, arhitektura te na primjeru prikazan razvoj web aplikacije pomoću Django.

2.1 Razvoj Django web okruženja

Django¹ web razvojno okruženje je razvijeno od World Online dijela tvrtke World Company u Lawrence, Kansasu. Glavna zadaća tvrtke je uređivanje vijesti (*online* novine) na Internetu, te razvoj web aplikacija. Razvoj Django započeo je 2003. godine kada su razvojni programeri World Online-a Adrian Holovaty i Simon Willson odlučili prestati koristiti programski jezik PHP za razvoj web aplikacija i počeli koristiti programski jezik Python. Od programskog jezika PHP odustali su zbog činjenice da je kod nepregledan za velike sustave, razvoj aplikacija je dugotrajan i podložen pogreškama. Počeli su razvijati web razvojno okruženje koje bi im omogućilo brz i jednostavan razvoj web aplikacija. Razvojno okruženje je kontinuirano razvijano 2 godine. Razvijen je "*from scratch*", tj. nisu korišteni već postojeći Python moduli.

Za razvoj Django razvojne okoline zaslužni su: Adrian Holovaty, Kacob Kaplan-Moss, Simon Willson i Wilson Miner. Egzistencija Django razvojnog okruženja zasniva se na slobodno dostupnim projektima kao što su Apache, Python, PostgreSQL.

2.2 Svojstva Django razvojnog okruženja

Django razvojno okruženje sastoji se od skupa alata koji pojednostavljaju razvoj web aplikacija i olakšavaju njihovo održavanje. Django alati pružaju niz svojstava:

- **autentikacija i autorizacija** (mogućnost dodavanja korisnika, grupa i dozvola nad izgrađenom web aplikacijom i/ili cijelim projektom)
- **komunikacija s bazom podataka** (podržane su najkorištenije baze podataka, za komunikaciju s bazom podataka nije potrebno postavljati SQL upite nego se oni generiraju iz programskog koda napisanog u programskom jeziku Python)
- **sučelje za administraciju** (u obliku administratorske web stranice omogućava dodavanje, izmjenu i brisanje elemenata u bazi podataka putem web preglednika)
- **konfiguriranje URL-ova** (pomoću regularnih izraza, pruža način povezivanja traženog URL-a i programskog koda koji odgovara traženom URL-u.)

1 Django je dobio ime prema romskom jazz gitaristu Djangu Reinhardt koji se smatra za najboljeg gitarista svih vremena

- **razvojni poslužitelj** (testiranje web aplikacija bez potrebe za konfiguriranjem web poslužitelja)
- **"Flat pages"** – (HTML kod spremljen u bazi te se prosljeđuje izgrađenom predlošku i prikazuje u web pregledniku.
- **odvajanje logike web aplikacije od prikaza** i pristupa bazi podataka (primjena izgleda web aplikacije ne utječe na logičku strukturu web aplikacije)

Posljednje svojstvo odvajanja logike, prikaza i pristupa bazi omogućava arhitektura na temelju koje je izgrađena Django razvojna okolina, a to je arhitektura MTV (eng. *model-template-view*). U nastavku su opisana osnovna svojstva Django web razvojnog okruženja.

2.2.1 MTV

Model-Template-View je arhitektura koji izdvaja različite dijelove web aplikacije – prikaz, pristup podacima i logiku web aplikacije. Važnost MTV strukture je u tome što omogućava neovisnu izgradnju web aplikacija, povećava sigurnost izgrađenog sustava, te pojednostavljuje održavanje sustava.

Model definira oblike i odnose podataka u bazama podataka. Model u Django web razvojnoj okolini je klasa (eng. *class*) napisana u programskom jeziku Python koja određuje varijable i metode pridružene određenim tipovima podataka te ima značenje tablice u bazi podataka. Pridružene varijable imaju značenje retka 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 (eng. *view*). Model nema saznanja o postojanju predloška i funkcija izvedenih u pogledu. Na taj način je baza podataka izdvojena od preostala dva dijela sustava.

Namjena **pogleda** je odrediti koji podaci će biti prikazani, tj. koji podaci će biti dohvaćeni iz baze podataka i prikazani pomoću pogleda u internet pregledniku.

U Django razvojnoj okolini, prilikom stvaranja web aplikacije za svaku pojedinu web aplikaciju kreira se zasebna datoteka pogleda. Datoteka pogleda sastoji se od funkcija napisanih u programskom jeziku Python. Za svaki "tip" stranice napisana je posebna funkcija koja upravlja izvođenjem upita zadane web stranice. Osim mogućnosti postavljanja upita model sloju za dohvatom podataka ima mogućnost implementacije slanja e-mailova, autentikacije, 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 (eng. *template*) je sloj arhitekture MTV usko povezan s web preglednikom. Predložak je HTML stranica s dodatnim strukturama koje omogućavaju prikaz podataka koji su proslijeđeni od sloja pogled. Zadaća predloška je sadržaj primljen od pogled sloja organizirati i ugraditi u HTML kôd koji će se prikazati u internet pregledniku.

Dodatne strukture koje omogućuju prikaz podataka proslijeđenih od pogleda su:

- **tagovi** – varijable kojima se pridjeljuje vrijednost prilikom učitavanja predloška u internet preglednik
- **ugrađeni-filter** (npr. prikaz svog teksta malim slovima neovisno o prikazu u bazi podataka)
- **ugrađene programske strukture**: `if()` i `for()`

Datoteka predložaka ima dodatna ograničenja nemogućnosti upisivanja naredbi u programskom jeziku Python. Zadano ograničenje onemogućava miješanje funkcija pojedinih slojeva te pruža dodatnu sigurnost web aplikaciji.

2.2.2 Konfiguracijska datoteka URL

URL konfiguracijska datoteka povezuje URL-ove stranica koje će se prikazati i poglede koji se pozivaju prilikom zahtjeva za određenim URL-om. Pročita URL koji je korisnik zatražio, pronalazi odgovarajući pogled i šalje mu potrebne varijable. URL-ovi su zapisani pomoću regularnih izraza što daje apsolutnu kontrolu nad njima u svakom dijelu web aplikacije onemogućujući neovlašten pristup podacima.

2.2.3 Sučelje za administraciju

Radi pojednostavljenja pohranjivanja, izmjene i brisanje podataka iz baze podataka, Django razvojno okruženje pruža mogućnost korištenja sučelja za administraciju. Sučelje za administraciju je web stranica koju je potrebno dodati u popis URL-ova. Da bi bilo moguće koristiti sučelje za administraciju potrebno je pokrenuti server koji posluhuje web aplikaciju. U datoteku modula potrebno je za svaku klasu dodati mogućnost uređivanja baze putem sučelja za administraciju. Sučelje za administraciju moguće je nadograditi dodatnim opcijama kao što su filteri za prikaz podataka i sortiranja.

2.2.4 Django razvojni poslužitelj

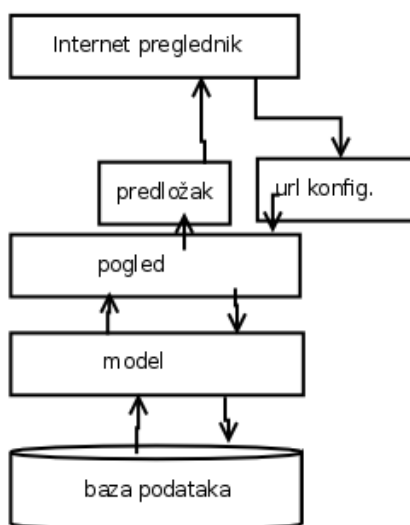
Django razvojni poslužitelj je web poslužitelj napisan u programskom jeziku Python. Uključen je u Django razvojnu radi olakšavanja ispitivanja

ispravnosti rada izgrađenih web aplikacija bez potrebe za konfiguriranjem web poslužitelja.

Django razvojni poslužitelj isključivo služi u razvojne svrhe. Kada je aplikacija u potpunosti izgrađena potrebno je iskonfigurirati web poslužitelj na kojem će se nalaziti web aplikacija. Primjer takvog web poslužitelja je Apache.

2.3 Izvođenje web aplikacije izgrađene pomoću Django

Slika 2.1 prikazuje izvođenje web aplikacije izgrađene pomoću Django web razvojnog okruženja. Korisnik u preglednik unese URL stranice kojoj želi pristupiti. Konfiguracijskoj URL datoteci šalje se zahtjev sa zadanom adresom. U konfiguracijskoj datoteci na temelju URL-a pronalazi se odgovarajući pogled. Odgovarajućom funkcijom u pogledu šalje se upit modelu za traženim podacima. Model sloj pristupa bazi podataka i dohvaća tražene podatke te ih prosljeđuje pogledu. Sloj pogleda šalje podatke u obliku varijabli, predlošku (*template*), evaluira se HTML datoteka i prikazuje u internet pregledniku.



Slika 2.1: Izvođenje web aplikacije izgrađene pomoću djanga

2.4 Programski jezik Python

Django web razvojno okruženje je zasnovano na programskom jeziku Python. Python je prevedeni programski jezik (eng. *interpreted*) kojeg je razvio Guido van Rossum 1990. godine. Python se temelji na nekoliko programskih paradigmi: funkcijska, imperativna i objektno-orijentirana. Po

svojoj sintaksi sličan je funkcijskim jezicima te koristi listu kao osnovni tip podataka. Mogućnost kreiranja klasa i sposobnost nasljeđivanja svojstva su objektno orijentiranog jezika. Imperativna paradigma očituje se u slijednom izvođenju programskog koda.

Razlozi zbog kojih je Python odabran kao jezik za implementaciju Django su upravo ti što je Python prevedeni jezik. Programski kod nije potrebno prevoditi, nego je odmah moguće vidjeti rezultate izvođenja. Nije potrebno brinuti se o deklaraciji tipova podataka, Python ima interaktivno sučelje u kojem se može provjeriti ispravnost podataka, veliki skup biblioteka različitih funkcionalnosti (biblioteke za rad s matricama, matematički, biblioteke koje pružaju korištenje kriptografskih algoritama i mnoge druge).

Biblioteke se u programskom jeziku Python nazivaju modulima. Django razvojno okruženje implementirano je kao Python modul. Moduli se učitavaju na sljedeće načine:

```
>>>import {ime modula}  
>>>from {ime biblioteke} import {ime modula}
```

Prvi način učitava modul do kojeg je moguće doći u jednom koraku pretraživanja modula. Primjer je Django modul:

```
>>> import django
```

Drugi način učitava iz zadane biblioteke /modula jedan od ugnježđenih modula:

```
>>> from django import db
```

3. Razvoj web aplikacija pomoću Django razvojne okoline

U ovom poglavlju na nekoliko jednostavnih primjera opisan je razvoj web aplikacija pomoću Django. U prvom dijelu opisano je postavljanje Django razvojnog okruženja, instalacija potrebnih alata, dok je u drugom dijelu opisano kreiranje projekta i web aplikacija pomoću Django.

3.1 Postavljanje Django web razvojnog okruženja

Postavljanje Django web razvojnog okruženja provodi se u 2 koraka. Prvi korak je instalacija potrebnih alata za pokretanje i rad Django. Drugi korak je instalacija Django razvojnog okruženja.

3.1.1 Instalacija potrebnih alata

Za postavljanje Django razvoje okoline potrebno je instalirati Python, najmanju verziju 2.3, te postaviti bazu podataka. Podržane baze podataka su: MySQL, PostgreSQL, Oracle. Dodatno je potrebno instalirati podršku Python programskog jezika za pristup bazi. Opcionalni alat je Apache web poslužitelj. Izgrađenu web aplikaciju moguće je pokrenuti i ispitati bez podešenog Apache web poslužitelja te o njemu u nastavku seminara neće biti govora. Odabrana baza podataka koja je instalirana je PostgreSQL. Django razvojnu okolinu moguće je koristiti na operacijskim sustavima Linux i Windows. Svi primjeri i naredbe u seminaru odnose se na operacijski sustav Linux, distribuciju Ubuntu.

Instalacija i postavljanje baze podataka PostgreSQL provodi se u nekoliko koraka. Prvi korak je instalacija PostgreSQL baze i pripadnih paketa. Baza PostgreSQL nalazi se u sklopu alata svake Linux distribucije. Osim PostgreSQL paketa potrebno je instalirati i python modul za rad s bazom podataka:

```
#apt-get install postgresql-8.2
#apt-get install python-pygresql
```

Stvaranje i pokretanje baze obavljaju sljedeće naredbe:

```
# su - postgres
postgres@user:~$ createuser
Enter name of role to add: ime
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) y
Shall the new role be allowed to create more new roles? (y/n) y
CREATE ROLE
```

```
postgres@user:~$ createdb primjer_app
CREATE DATABASE
```

Drugi korak je stvaranje baze podataka. Prije kreiranja baze podataka potrebno je kreirati korisnika koji će imati ovlasti nad bazom podataka. Određuje se treba li korisnik imati ovlasti super korisnika ili ne. Nakon što je kreiran korisnik pristupa se kreiranju baze podataka naredbom `createdb`. Baza koja je stvorena je `primjer_app`.

Pregled sadržaja stvorene baze podataka obavlja se naredbama :

```
# psql primjer_app
primjer_app=# \dt
No relations found.
```

U ispisu naredbe saznaje se da je kreirana baza podataka, međutim u bazi nema dodanih relacija. Dodatni paket koji omogućava interakciju PostgreSQL baze i programskog jezika Python je `python-psycopg2`:

```
# apt-get install python-psycopg2
```

Postavljanjem baze podataka instalirani su dodatni paketi te se pristupa instalaciji Django alata.

3.1.2 Instalacija Django alata

Za instalaciju Django alata potrebno je dohvatiti programski kôd. Programski kôd je moguće dohvatiti na 3 načina: sa službene stranice projekta na kojoj se nalazi posljednja stabilna verzija, preko sustava za praćenje verzije programskog kôda (eng. *subversion*) putem kojeg se dohvaća najnovija verzija kôda ili koristeći `python-django` paket dostupan u alatima Linux-a.

Prvi način instalacije Djanga je dohvaćanjem kôda preko sustava za praćenje verzije programskog kôda:

```
# svn co http://code.djangoproject.com/svn/django/trunk/
django_src
# cd django_src
# python setup.py install
```

`django_src` je ime direktorija u koji se sprema kod Django alata. Instalacija se pokreće izvršavanjem python programa `setup.py` s argumentom `install`.

Drugi način je instalacija stabilne verzije Djanga. Sa službenih stranica projekta dohvaća se Django programski kôd. Kôd je zapakiran te ga je potrebno otpakirati i pokrenuti instalaciju:

```
#tar xzvf Django-*.tar.gz
#cd Django-*
#python setup.py install
```

Treći način je odabir Django alata kao python modula dostupnog kao paketa u Linuxu:

```
# apt-get install python-django
```

Provjera je li Django alat uspješno instaliran vrši se otvaranjem python konzole i učitavanjem modula django ključom riječi import:

```
#python
>>> import django
```

Ključnom riječi naredbom dir provjeravamo metode na raspolaganju s django modulom.

```
>>>dir(django)
['VERSION', '__builtins__', '__doc__', '__file__',
 '__name__', '__path__']
```

U ispisu nisu navedeni podmoduli Djanga koji se učitavaju na sljedeći način:

```
>>> from django import forms
```

Ukoliko se pri učitavanju podmodula pojavi greška :

```
File "/var/lib/python-
support/python2.5/django/conf/__init__.py", line 52, in
_import_settings
raise EnvironmentError, "Environment variable %s is
undefined." % ENVIRONMENT_VARIABLE
EnvironmentError: Environment variable
DJANGO_SETTINGS_MODULE is undefined.
```

Potrebno je dodatno omogućiti varijable okoline:

```
>>>import os
>>>os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'
```

Nakon što su uspješno postavljeni Django alat i baza podataka moguće je stvoriti projekt web aplikacije.

3.2 Stvaranje projekta i web aplikacije pomoću Django

3.2.1 Alat za administriranje(*eng.admin tool*)

Alat za administriranje je alat Django razvojne okoline koji omogućuje stvaranje novog projekta, održavanje projekta, stvaranje potrebnih relacija u bazi podataka, sinkronizaciju projekta s bazom podataka, te pokretanje poslužitelja za izgrađene web aplikacije.

Pokreće se naredbom:

```
#django-admin.py action [options]
```

Najčešće korištene akcije za stvaranje i pokretanje web aplikacije su:

```
adminindex [modelmodule ...] Prints the admin-index template snippet for the given model module name(s).  
startapp [appname] Creates a Django app directory structure for the given app name in the current directory.  
startproject [projectname] Creates a Django project directory structure for the given project name in the current directory.  
runserver [--noreload] [optional port number, or ipaddr:port] Starts a lightweight Web server for development.  
syncdb Creates the database tables for all apps in INSTALLED_APPS whose tables haven't already been created.
```

3.2.2 Stvaranje projekta

Projekt se stvara naredbom `django-admin.py` uz akciju `startproject` i zadano ime projekta.

```
# django-admin.py startproject primjer_app
```

Stvoren je projekt pod imenom `primjer_app`. Pokretanjem zadane naredbe stvoren je direktorij s imenom `primjer_app` u kojem se nalaze konfiguracijske datoteke potrebne za izgradnju web aplikacije pomoću Django web razvojnog okruženja. Stvorene konfiguracijske datoteke su:

```
__init__.py  
manage.py  
settings.py  
urls.py
```

`__init__.py` je prazna datoteka koja označava da je direktorij Python paket. U svakom direktoriju aplikacije nalazi se zadana datoteka.
`manage.py` je konzolni alat koji omogućava interakciju s Django projektom, kao što su pokretanje servera ili osvježavanje baze podataka.
`settings.py` je konfiguracijska datoteka postavki stvorenog Django

projekta.

urls.py je datoteka koja spaja URL-ove i poglede.

3.2.3 Dodavanje aplikacija u projekt

Za dodavanje nove web aplikacije potrebno je nalaziti se u direktoriju projekta. Dodavanje nove aplikacije u projekt obavlja se naredbom:

```
# python manage.py startapp primjer1
```

Stvoren je direktorij primjer1 u kojem su generirane datoteke za model i pogled:

```
__init__.py  
models.py  
views.py
```

models.py je datoteka u koju se dodaju modeli web aplikacije.

views.py je datoteka u koju se dodaju pogledi .

Poslužitelj na kojem je pokrenuta zadana aplikacija pokreće se :

```
# python manage.py runserver
```

Ukoliko je poslužitelj uspješno pokrenut ispisat će se:

```
Validating models...  
0 errors found.  
Django version 0.95.1, using settings 'primjer_app.settings'  
Development server is running at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```

Provjereni su modeli, nema grešaka te je pokrenut poslužitelj na adresi 127.0.0.1:8000. Moguće je eksplicitno navesti pristupnu točku na kojoj je pokrenut poslužitelj. Učitavanjem u internet preglednik prikazuje se poruka o ispravno pokrenutom poslužitelju.

Na slici 3.1 prikazan je uspješno pokrenut poslužitelj. Na zadanoj adresi nije izgrađena web aplikacija te u konfiguracijskoj datoteci URL ne postoji povezanost između zadane adrese i pogleda te se ispisuje poruka da je poslužitelj uspješno pokrenut.

It worked!

Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Here's what to do next:

- If you plan to use a database, edit the DATABASE_* settings in `crypto_site/settings.py`.
- Start your first app by running `python crypto_site/manage.py startapp [appname]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

Slika 3.1: Uspješno pokrenut poslužitelj

3.2.4 Uređivanje postavki

Nakon uspješno pokrenutog poslužitelja potrebno je promijeniti postavke u datoteci `settings.py`. Postavke koje je moguće urediti su: postavke baze podataka, dostupne aplikacije Django web razvojnog okruženja, direktoriji u kojima se nalaze predlošci, direktoriji u kojima se nalazi medijski sadržaj i drugi.

Postavke baze započinju ključnom riječi `DATABASE`. `ENGINE` označava vrstu baze podataka kojoj se pristupa. U primjerima u seminaru radi se o bazi podataka PostgreSQL. `NAME` je ime kreirane baze. `USER` označava korisnika koji pristupa bazi podataka, a `PASSWORD` lozinku korisnika. `HOST` i `PORT` popunjavaju se ukoliko se baza podataka ne nalazi fizički na računalu na kojem je pokrenuta Django aplikacija:

```
DATABASE_ENGINE = 'postgresql'
DATABASE_NAME = 'primjer_app'
DATABASE_USER = 'ime'
DATABASE_PASSWORD = ''
DATABASE_HOST = ''
DATABASE_PORT = ''
```

Postavke koje označavaju instalirane dodatne aplikacije Django web razvojnog okruženja označene su strukturom `INSTALLED_APPS`. U strukturi su navedena imena Django aplikacija koje su dostupne svakom projektu stvorenom pomoću Django. Navedene aplikacije su sustav za autentikaciju, okruženje za content tipove, za omogućavanje sjednice i uređivanje većeg broja web stranica putem unutar Django projekta:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
)
```

Za svaku aplikaciju navedenu u strukturi potrebno je stvoriti tablicu u bazi podataka. Stvaranje tablica u bazi podataka obavlja se naredbom :

```
python manage.py syncdb
```

Opcija syncdb koristi se i za stvaranje tablica za modele pojedinih aplikacija u Django projektu.

Pokretanjem gore navedene naredbe stvorene su tablice za autentikaciju korisnika, za održavanje sjednice te tablica sa postavkama Django stranice . Kako je navedeno u prikazu izvršavanja naredbe ,za autentikaciju korisnika zadane su tablice koje određuju grupu kojoj pripadaju korisnici i ovlasti koje imaju. Za preostale aplikacije stvorena je po jedna tablica.

```
Creating table auth_message
Creating table auth_group
Creating table auth_user
Creating table auth_permission
Creating many-to-many tables for Group model
Creating many-to-many tables for User model
Creating table django_content_type
Creating table django_session
Creating table django_site
```

Pri stvaranju Django sustava za autentikaciju potrebno je stvoriti super korisnika (root).

```
You just installed Django's auth system, which means you don't
have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'root'):
E-mail address: ime.prezime@gmail.com
Password:
Password (again):
Superuser created successfully.
```

Nakon što je stvaranje super korisnika uspješno izvršeno dodane su ovlasti zadanom korisniku. Super korisnik ima ovlasti dodavanja podataka, promjena i brisanja podataka, dodavanja grupa , promjena, i brisanja te dodavanja korisnika, te ima mogućnost promjena ovlasti.

```
Adding permission 'message | Can add message'
Adding permission 'message | Can change message'
Adding permission 'message | Can delete message'
Adding permission 'group | Can add group'
Adding permission 'group | Can change group'
Adding permission 'group | Can delete group'
Adding permission 'user | Can add user'
[...]
Adding permission 'permission | Can add permission'
```



```
Adding permission 'permission | Can change permission'
Adding permission 'permission | Can delete permission'
[...]
```

Nakon što su podešene postavke i stvorene osnovne tablice koje omogućuju upravljanje Django projektom, sljedeći korak je stvaranje web aplikacija. Prvi korak stvaranja web aplikacije je kreiranje modela.

3.3 Stvaranje modela

Prvi korak razvoja aplikacija pomoću Django razvojnog okruženja je stvaranje modela. Stvaranjem modela definiraju se klase i varijable unutar klase koji se koriste i relacije među varijablama. Stvorene varijable u bazi podataka predstavljaju stupac tablice koja je predstavljena klasom u kojoj se nalaze zadane varijable.

Stvaranje modela vrši se editiranjem datoteke `models.py` koja je smještena u direktoriju izgeneriranom posebno za svaku aplikaciju projekta. Inicijalno su u datoteci `models.py` uključeni modeli iz `django.db` modula. Django modeli koje je moguće koristiti za opis objekata korištenih u aplikaciji su:

```
>>> from django.db import models
>>> dir(models)
['ADD', 'AdminOptions', 'AutoField', 'BLANK_CHOICE_DASH',
'BLANK_CHOICE_NONE', 'BOTH', 'BooleanField', 'CHANGE',
'CharField', 'CommaSeparatedIntegerField', 'DateField',
'DateTimeField', 'EmailField', 'Field', 'FieldDoesNotExist',
'FileField', 'FilePathField', 'FloatField', 'ForeignKey',
'GenericForeignKey', 'GenericRel', 'GenericRelation',
'HORIZONTAL', 'IPAddressField', 'ImageField',
'ImproperlyConfigured', 'IntegerField', 'LazyDate', 'Manager'
[...]]
```

Način stvaranja modela opisan je pomoću primjera koji je opisan u nastavku. Aplikacija `primjer1` sastoji se od dva modela. Prvi model sadrži podatke o korisniku: ime, prezime i datum rođenja. Drugi model sadrži podatke o ispitima koje je korisnik položio: predmet i ocjena. Ime, prezime i ime predmeta su objekti znakovnog tipa, datum rođenja je tipa `datum`, a ocjena je cjelobrojnog tipa.

Stvorena model datoteka nakon unesenih prethodno navedenih podataka je sljedeća:

```
from django.db import models
# Create your models here.
class Korisnik(models.Model):
```

```

ime = models.CharField(maxlength=50)
prezime= models.CharField(maxlength=50)
datum_rodjenja= models.DateTimeField()

class Ispiti(models.Model):

    korisnik = models.ForeignKey(Korisnik)
    predmet = models.CharField(maxlength = 50)
    ocjena = models.IntegerField()

```

U primjeru su modeli Korisnik i Ispit povezani. Model Ispit definira varijablom korisnik koja je tipa strani ključ povezanost sa modelom Korisnik.

Nakon što su stvoreni modeli u datoteku postavki potrebno je dodati novu aplikaciju koja se koristi. U strukturu INSTALLED_APPS dodaje se ime aplikacije, 'primjer_app.primjer1'. Nakon dodavanja aplikacije, potrebno je stvoriti tablice na temelju zadanog koda u datoteci models.py. Prvi korak je na temelju programskog koda u Pythonu generirati SQL naredbe za stvaranje tablica u bazi podataka. To se obavlja pokretanjem skripte manage.py uz akciju sql i ime aplikacije:

```
# python manage.py sql primjer1
```

Ispisuje se izgenerirani SQL kod:

```

BEGIN;
CREATE TABLE "primjer1_korisnik" (

    "id" serial NOT NULL PRIMARY KEY,
    "ime" varchar(50) NOT NULL,
    "prezime" varchar(50) NOT NULL,
    "datum_rodjenja" timestamp with time zone NOT NULL

);
CREATE TABLE "primjer1_ispiti" (

    "id" serial NOT NULL PRIMARY KEY,
    "korisnik_id" integer NOT NULL REFERENCES
"primjer1_korisnik" ("id"),
    "predmet" varchar(50) NOT NULL,
    "ocjena" integer NOT NULL

);
COMMIT;

```

Generiran je SQL kod koji stvara tablice primjer1_korisnik i primjer1_ispit, automatski su dodani primarni ključevi ("id"), ograničenja da pojedine vrijednosti u tablici ne mogu biti NULL, te je varijabla korisnik tablice ispit referenciran na "id" tablice korisnik.

Nakon generiranja SQL koda potrebno je stvoriti tablice u bazi podataka pokretanjem skripte `manage.py` uz naredbu `syncdb`.

```
# python manage.py syncdb
```

U bazi podataka stvorene su tablice `primjer1_korisnik` i `primjer1_ispit` te su dodane ovlasti nad bazom podataka:

```
Creating table primjer1_korisnik
Creating table primjer1_ispiti
Adding permission 'korisnik | Can add korisnik'
Adding permission 'korisnik | Can change korisnik'
Adding permission 'korisnik | Can delete korisnik'
Adding permission 'ispiti | Can add ispiti'
Adding permission 'ispiti | Can change ispiti'
Adding permission 'ispiti | Can delete ispiti'
```

U bazi podataka moguće je vidjeti dodane tablice:

```
primjer_app=# \dt

List of relations

Schema | Name | Type | Owner
-----
[...]
public | primjer1_ispiti | table | root
public | primjer1_korisnik | table | root
```

Uređivanje baze podataka moguće je na više načina. U nastavku su objašnjena 2 načina: upravljanje bazom preko konzolne aplikacije `shell` i korištenjem administratorske stranice.

3.4 Konzolna aplikacija `shell`

Konzolna aplikacija `shell` omogućuje interakciju s bazom podataka, dohvaćanje podataka, unošenje podataka. Pokreće se pozivanjem skripte `manage.py` uz opciju `shell`:

```
#python manage.py shell
```

Izvršavanjem zadane naredbe pokrenuta je interaktivna konzola. U konzolu je potrebno učitati module aplikacije čijim tablicama se pristupa na sljedeći način:

```
>>> from primjer_app.primjer1.models import Korisnik, Ispiti
```

Moduli koji su učitani nalaze se u projektu `primjer_app`, aplikacije `primjer1` u datoteci `models.py`. Moduli koji su učitani su `Korisnik` i `Ispiti`.

Ispis svih elemenata koji se nalaze u pojedinoj tablici vraća naredba `objects.all()`. Ukoliko je tablica prazna naredba vraća praznu listu:

```
>>> Korisnik.objects.all()
[]
```

Dodavanje korisnika u bazu podataka obavlja se upisivanjem vrijednosti varijabli u strukturu `korisnik`:

```
>>>k=Korisnik(ime="ivo",prezime="ivic",datum_rodjenja=
datetime.datetime(1985,6,5,5,0))
>>> k.save()
```

Popunjena je struktura `Korisnik` s imenom, prezimenom i datumom rođenja. `k` je instanca klase `Korisnik` sa zadanim parametrima. Spremanje u bazu podataka postiže se funkcijom `save()`.

Pristup pojedinom elementu instance (stupcu u tablici) postiže se imenovanjem instance klase i pozivanjem elementa na način:

```
>>> k.prezime
'ivic'
```

Moguće je i promijeniti vrijednost pojedinog elementa u bazi podataka pozivanjem elementa kao u gornjem primjeru, instanciranjem na novu vrijednost i spremanjem u bazu podataka:

```
>>> k.prezime="horvat"
>>> k.save()
>>> k.prezime
'horvat'
```

Uređivanje baze (unošenje podataka i brisanje podataka iz već postojećih tablica) podataka putem konzolne aplikacije `shell` nepregledno je i složeno. Aplikacija koja omogućuje jednostavnije korištenje baze podataka je administratorska stranica.

3.5 Administratorska stranica

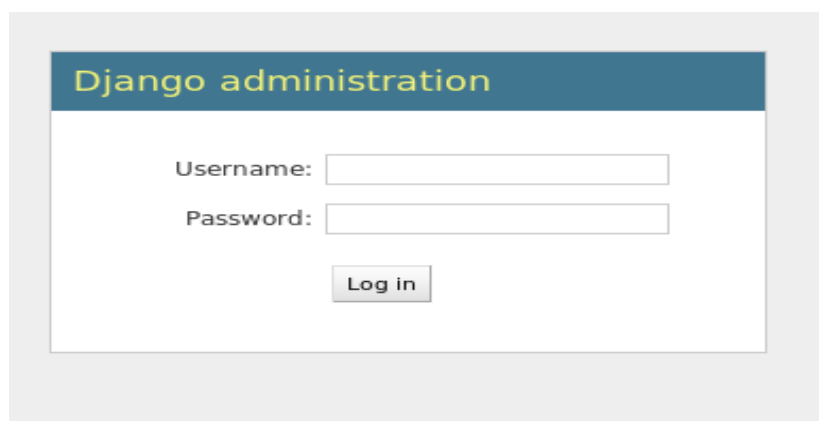
Administratorska stranica omogućava jednostavno uređivanje baze podataka putem web preglednika. Administratorsku stranicu potrebno je aktivirati. Aktivacija se provodi u nekoliko koraka. Prvi korak je u datoteku postavki u strukturu `INSTALLED_APPS` dodavanje `django.contrib.admin` te nakon toga pokrenuti sinkronizaciju s bazom podataka i stvaranje potrebnih tablica.

Drugi korak je editiranje datoteke sheme URL-ova `urls.py` koja je detaljno objašnjena u nastavku te odkomentiranje linije:

```
(r'^admin/', include('django.contrib.admin.urls'))
```

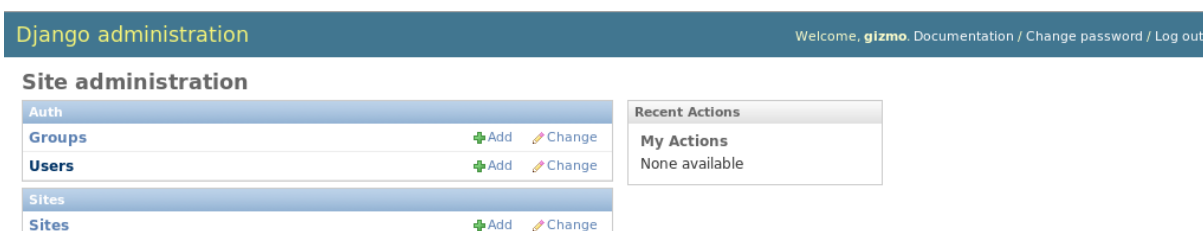
koja omogućava da se administratorska stranica nalazi na adresi `<adresa_servera>/admin`

Posljednji korak je pokretanje poslužitelja skriptom `manage.py` uz akciju `runserver`. Adresa na kojoj je pokrenuta administratorska stranica je: `<ip_adresa_servera>:<port>/admin`. Na slici 3.2 prikazana je početna administratorska stranica nakon otvaranja u web pregledniku. Potrebno je unjeti korisničko ime i lozinku super korisnika koji je stvoren prilikom uređivanja postavki web projekta.



Slika 3.2: Forma za ulaz u administratorsku tranicu

Nakon uspješne prijave u sustav otvara se administratorska stranica koja je prikazana na slici 3.3:



Slika 3.3: Administratorska stranica

Inicijalno su popisi grupa, korisnika i stranica prazni, a administratorska stranica nema dodatnih opcija.

Sljedeći korak je omogućavanje prikaza Korisnika i Ispita u administratorskoj stranici. Da bi se omogućio prikaz u administratorskoj stranici potrebno je promijeniti postavke modula u datoteci `modules.py`. U datoteci `modules.py` u

svaku klasu koju je potrebno prikazati u administratorskoj stranici potrebno je dodati klasu Admin:

```
class Korisnik(models.Model):  
  
    # ...  
    class Admin:  
        pass
```

Nakon spremanja datoteke models.py promjene su vidljive u administratorskoj stranici. Na slici 3.4. prikazane su dodane tablice Korisnik i Ispiti.

Primjer1		
Ispitis	+Add	Change
Korisniks	+Add	Change

Slika 3.4: Prikaz dodatnih tablica Korisnik i Ispiti

Unošenje u bazu podataka vrši se odabirom opcije Add. Odabirom opcije Add otvara se forma za unos podataka prikazana na slici 3.5:

Django administration

Home > Korisniks > Add korisnik

Add korisnik

Ime:

Prezime:

Datum Today

Slika 3.5: Forma za unos podataka o korisniku

Podaci se spremaju u bazu odabirom opcije Save. Osim inicijalnih opcija administratorske stranice moguće su dodatne opcije koje se navode u datoteci models.py za svaku klasu zasebno. Neke od dodatnih opcija su: odabir prikaza stupaca u tablici, vizualno odvajanje logičkih cjelina (datum je posebna cjelina odvojena od imena i prezimena), dodatno pojašnjenje što treba upisati u pojedino polje, selekcija podataka. Korištenjem administratorske stranice omogućuje se jednostavnije upravljanje bazom podataka.

3.6 Kreiranje sheme URL-ova

Nakon stvaranja baze podataka i administratorske stranice, sljedeći korak u razvoju web aplikacija pomoću Djanga je odabir URL-ova web aplikacije. Odabir URL-a web aplikacije uključuje povezivanje odabranog URL-a i

stvarne staze do pogleda koji poslužuje zahtjev generiran korištenjem aplikacije na zadanom url-u.

Kada web aplikacija primi zahtjev za prikazom određene stranice, pretražuje se struktura ROOT_URLCONF datoteke postavki. Modul koji je naveden u zadanoj strukturi učitava te pretražuje varijablu urlpatterns koja je oblika:

```
(regular expression, Python callback function [, optional dictionary])
```

Pretražuje regularne izraze u datoteci urls.py tražeći odgovarajuću stazu. URL-ove u datoteku urls.py potrebno je dodati ručno.

Za zadanu web aplikaciju primjer1 kreirani su URL-ovi:

```
from django.conf.urls.defaults import *
urlpatterns = patterns('',
    (r'^primjer1/', 'primjer_app.primjer1.views.index'),
    (r'^primjer1/korisnik/$', 'primjer_app.primjer1.views.korisnik',
)
)
```

Ukoliko korisnik zatraži stranicu primjer1/ pozvat će se funkcija index pogleda koji je dodijeljen web aplikaciji primjer1 projekta primjer_app. U drugom URL-u navedeno je ukoliko korisnik zatraži stranicu primjer1/korisnik/ pozvat će se funkcija korisnik. Oznaka r na početku zapisa URL-a označava da nije moguće preskočiti niti jedan znak pri čitanju URL-ova.

Za zadane URL-ove nisu još stvoreni pogledi. Ispravnost definiranih URL-ova moguće je provjeriti pokretanjem razvojnog poslužitelja. Nakon pokretanja poslužitelja potrebno je otvoriti web stranicu definiranu u datoteci urls.py:

```
http://127.0.0.1:8000/primjer1/
```

Ukoliko su URL-ovi ispravno definirani ispisat će se pogreška koja govori da ne postoji pogled naveden u urls.py konfiguracijskoj datoteci. U sprotom će se ispisati sintaksna pogreška. Slika 3.6 prikazuje ispis greške pri zahtjevu za stranicom za koju nije stvoren pogled:

ViewDoesNotExist at /primjer1/

Tried index in module primjer_app.primjer1.views. Error was: 'module' object has no attribute 'index'

Request Method: GET
Request URL: http://127.0.0.1:8000/primjer1/
Exception Type: ViewDoesNotExist
Exception Value: Tried index in module primjer_app.primjer1.views. Error was: 'module' object has no attribute 'index'
Exception Location: /var/lib/python-support/python2.5/django/core/urlresolvers.py in get_callback, line 122

Slika 3.6: Obavijest da pogled još nije kreiran

Nakon što su kreirani URL-ovi sintaksno ispravni sljedeći korak stvaranja web aplikacije je stvaranje pogleda.

3.7 Stvaranje pogleda

Nakon stvaranja modela , pokretanja administratorske stranice i stvaranja URL-ova sljedeći korak razvoja web aplikacija je stvaranje pogleda. Pogledi ostvaruju funkcionalnost web aplikacije. Za svaki definirani URL stvorimo jednu funkciju koja predstavlja pogled za pojedinu stranicu.

U datoteku views.py dodaju se funkcije. Datoteka views.py je inicijalno prazna datoteka. Stvoren je pogled index koji predstavlja prvu stranicu web aplikacije :

```
from django.http import HttpResponse
def index(request):

    return HttpResponse("Prva stranica index ")
```

U prvom redu uključen je Django modul HttpResponse. Funkcija index nakon što primi zahtjev vraća HttpResponse.

3.8 Stvaranje predloška

U pogled je moguće unijeti HTML kod koji se izvodi kada korisnik zatraži stranicu, međutim takvo rješenje nije u skladu s arhitekturom MTV te se sav prikaz stvara pomoću predložaka. Predložak je tekstualna datoteka ili string u programskom jeziku Python koji određuju prikaz u web pregledniku.

Sintaksne strukture predloška su : blok tagovi i varijable. Blok tagovi u predlošku obavljaju neku funkciju, na primjer ispis dohvaćenih podataka iz baze, definiranje kontrole toka (if,for). Oznake za blok tagove su {% % } . Primjer blok taga je:

```
{% if logiran %}Prijavljeni ste u sustav!{% else %}Logirajte se.{% endif %}
```


U zadanom primjeru definirani su blok tagovi if logiran, else i endif.

Varijable služe ispisu vrijednosti u predlošku. Oznaka za varijable je: {{ }}. Primjer varijabli je :

```
zovem se{{moje_ime }}. studiram na {{fakultet}}.
```

Django predložak sastoji se od barem dva tekstualna dokumenta. Prvi dokument je osnovni predložak, dok je drugi naslijeđen od osnovnog predloška. Nasljeđivanje predložaka svojstvo je Djanga koje omogućava izgradnju osnovnog predloška(kostur) i neograničenog broja naslijeđenih predložaka te na taj način neograničen broj predložaka koji se mogu prikazati u internet pregledniku na osnovu samo jednog osnovnog predloška. Osnovni predložak sadrži sve potrebne elemente definirane blokovima koje naslijeđeni predlošci nadjačavaju. Korištenjem osnovnog predloška i naslijeđenih predložaka u potpunosti se odvaja logika web aplikacije od prikaza.

Potrebno je kreirati osnovni predložak base.html. Primjer osnovnog predloška je:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">

    <head>

        <title>Moja stranica: {% block title %}stranica{%
endblock %}
        </title>

        {% block extrahead %}{% endblock %}

    </head>
    <body>

        {% block content %}{% endblock %}

    </body>

</html>
```

U predlošku su definirani blokovi title, extrahead i content. U blok title unosi se ime stranice, u blok extrahead moguće je dodati svu funkcionalnost koju je potrebno smjestiti u glavu HTML dokumenta. U blok content sprema se sav sadržaj koji je moguće smjestiti u tijelo HTML dokumenta.

Primjer naslijeđenog predloška je:

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

{%block extrahead %}

<style>

    body {

        font-style: arial;

    } h1 {

        text-align: center;

    } .job .title {

        font-size: 120%;
        font-weight: bold;

    } .job .posted {

        font-style: italic; }

</style >

{%endblock%}
```

4. Sigurnost web aplikacija stvorenih pomoću Django

4.1 SQL injection

SQL injection je napadačka tehnika koja se koristi kako bi se ugrozila sigurnost Web aplikacije koja konstruira SQL izjave iz korisnički unesenih podataka.

Navedena nesigurnost očituje se pri prisanju SQL upita direktno u programskom kôdu web aplikacije. Ukoliko želimo onemogućiti čitanje svih e-mail adresa iz baze podataka , stvara se funkcija koja dohvaća e-mail adresu korisnika na osnovu njegovog korisničkog imena:

```
def korisnici(request):  
  
    korisnik = request.GET['ime']  
    sql = "SELECT * FROM korisnik WHERE ime = '%s';" % ime  
    # execute the SQL here...
```

Ukoliko je umjesto korisničkog imena u formu na web stranici upisan niz: ' OR 'a'='a, konstruirani upit pozivanjem funkcije korisnici je:

```
SELECT * FROM korisnici WHERE ime = '' OR 'a' = 'a';
```

Unošenjem nesigurnog SQL koda omogućeno je vraćanje svih stupaca u tablici ključnom riječi OR.

Također, omogućeno je brisanje potpune liste kontakata unošenjem naredbe '; DELETE FROM user_contacts WHERE 'a' = 'a u formu za upit na web stranici. Unošenjem naredbe generira se SQL upit:

```
SELECT * FROM user_contacts WHERE username = ''; DELETE FROM  
user_contacts WHERE 'a' = 'a';
```

Rješenje problema je izbjegavanje kôda prilikom prosljeđivanja unosa SQL upitima. Django u sebi ima ugrađen mehanizam za izbjegavanje kôda. Automatski se izbjegavaju posebni SQL parametri. Za postavljen upit:

```
UserContacts.objects.filter(username="" OR 'a'='a")
```

Django mehanizam izbjegava parametar OR generirajući upit:

```
SELECT * from user_contacts WHERE username = '' OR \'a\'=\'a'
```

Izraz ""\' OR \'a\'=\'a"" je potpuno bezopasan.

4.2 XSS

Cross-site scripting (XSS) je napadačka tehnika koja prisiljava Web aplikaciju da proslijedi napadački izvršni kod korisniku, koji se zatim učitava u korisnikovom Web pregledniku i izvršava.

Primjer nesigurnosti Django web aplikacije je pogled koji čita ime iz GET parametara i prosljeđuje ih stranici hello.html:

```
def Ispisi_pozdrav(request):  
    ime = request.GET.get('ime', 'world')  
    return render_to_response("hello.html", {"ime" : ime})
```

Neka je zadan predložak :

```
<h1>Hello, {{ name }}!</h1>
```

Ukoliko je pozvana stranica s parametrima:

<http://example.com/hello/ime=<i>Pero</i>> ,učitana stranica je oblika:

```
<h1>Hello, <i>Pero</i >!</h1>
```

Napadač ima mogućnost unošenja malicioznog kôda direktno u web stranicu.

Rješenje problema je ,kao i kod SQL injection napada, uvijek izbjegavanje sadržaja koji su unijeli korisnici. Izbjegavanje sadržaja omogućava se promjenom predloška na sljedeći način:

```
<h1>Hello, {{ ime|escape }}!</h1>
```

Korištenjem escape filtera onemogućeno je unošenje programskog kôda u predložak i na taj način onemogućeni su XSS napadi.

4.3 CSRF

CSRF (*Cross Site Request Forgey*) je napadačka tehnika kojom zločudne web stranice prevare korisnika učitavajući URL stranice na koju su se unaprijed autentificirali, koristeći pogodnosti korisnikovig autentifikacijskog statusa.

Django ima ugrađena rješenja zaštite od CSRF oblika napada. U standardnoj biblioteci django.contrib nalazi se paket csrf, ugrađeno rješenje zaštite od CSRF napada.

CSRF napad i zaštita od napada objašnjeni su na primjeru u nastavku. Korisnik se prijavljuje za rad na webmail na stranicu `moj_primjer.com`. Stranica sadrži gumb za odjavu koji preusmjerava korisnika na stranicu `moj_primjer.com/odjava`. Da bi se korisnik odjavio sa sustava potrebno je posjetiti stranicu `moj_primjer.com/odjava`.

Zloćudna stranica prisiljava korisnika na posjećivanje stranice `moj_primjer.com/odjava` uključivanjem zloćudnog skrivenog kôda u stranicu. Kada korisnik zatraži stranicu `moj_primjer.com` automatski je preusmjeren na stranicu `moj_primjer.com/odjava`. Primjer koji je naveden onemogućava uslugu pregleda e-maila, međutim CSRF napadi su u pravilu puno opasniji.

Rješenje problema je korištenje Django alata za onemogućavanje CSFR napada. `django.csrf` paket sadrži modul `middleware.py`. Modul u sebi ima klasu `CsrfMiddleware` koja implementira zaštitu od CSRF napada.

Korištenje Django alata omogućava se dodavanjem klase `CsrfMiddleware` u strukturu `MIDDLEWARE_CLASSES` u datoteku postavki web aplikacije: `'django.contrib.csrf.middleware.CsrfMiddleware'`

Rad alata za prevenciju CSRF napada radi u dva koraka:

- modificira izlazne upite dodajući skriveno polje svim POST formama. Ime skrivenog polja je `csrfmiddlewaretoken`, a vrijednost je hash ID sjednice i tajnog ključa. Alat ne mijenja odgovor ukoliko ne postoji ID sjednice, stoga je ovakav način zaštite nemoguć za upite koji ne koriste sjednicu.
- za sve dolazne POST upite koji imaju postavljen kolačić sjednice, provjerava polje `csrfmiddlewaretoken`. U slučaju da polje nije ispravno korisniku se vraća greška 403 s porukom o detektiranom CSRF napadu: *"Cross Site Request Forgery detected. Request aborted."*

4.4 Napadi otimanja sjednice

Napadi otimanja sjednice je skup napada na sjednicu. Vrste napada su:

- **Napad s čovjekom u sredini** (eng. *man in the middle*) - napadač se ubacuje u komunikaciju.
- **Krivotvorenje sjednice** (eng. *Session forging*) – napadač krivotvori ID sjednice
- **Krivotvorenje kolačića** (eng. *Cookie forging*) – napadač krivotvori sadržaj kolačića
- **Fiksiranje sjednice** (eng. *Session fixation*) – napadač podmeće korisniku krive postavke ili resetiranje ID-ja sjednice.
- **Trovanje sjednice** (eng. *Session poisoning*) – napadač ubacuje potencijalno opasne podatke u sjednicu

Rješenje se svodi na nekoliko principa:

- URL-ovi stranice ne smiju sadržavati podatke o sjednici
- Nije dopušteno spremanje podataka o sjednici direktno u kolačiće. Umjesto toga u kolačiće se sprema ID sjednice koji povezuje podatke o sjednici koji su spremljeni u stražnje aplikacije (*backend*) i kolačiće
- Potrebno korištenje escape filtera ukoliko se podaci sjednice prikazuju putem predloška
- Onemogućiti dohvaćanje ID-ja sjednice

Međutim, ni jedan od navedenih principa ne sprječava napad s čovjekom u sredini. Napade s čovjekom u sredini vrlo je teško detektirati. Ukoliko je korisnicima omogućena prijava za rad na sustav i pregled povjerljivih podataka preporuča se korištenje HTTPS protokola.

4.5 Umetanje zaglavlja e-maila (*E-mail header injection*)

E-mail header injection je napad sličan *SQL injectionu*. *E-mail header injection* napadom preotimaju se web forme i koriste za slanje spam poruka. Većina web formi za slanje e-poruka (npr. kontakt forme) omogućavaju unošenje vlastitog naslova e-poruke. Ukoliko napadač u formu naslova upiše niz: "hello\ncc:spamzrtva@example.com", generira se zaglavlje poruke:

```
To: prava.adresa@example.com
Subject: hello
cc: spamzrtva@example.com
```

Te se na e-mail adresu žrtve spama šalje generirana e-poruka.

Rješenje problema je jednako kao i rješenje problema *SQL injection* napada korištenjem izbjegavanjem kôda.

Dodatno, Django ima ugrađene mehanizme za prevenciju napada umetanjem e-mail zaglavlja: `django.core.mail` funkcija koja ne dopušta umetanje novih redova u zaglavlje putem forme za unos.

4.6 Zaobilaženje direktorija (*Directory traversal*)

Zaobilaženje direktorija jedan je od napada umetanje u kojem napadač vara datotečni sustav u kojem je smještena web aplikacija i pristupa zaštićenim datotekama sustava.

Primjer nesigurnog sustava je aplikacija kojoj funkcija pogleda ima oblik:

```
def dump_file(request):
    filename = request.GET["filename"]
    filename = os.path.join(BASE_PATH, filename)
    content = open(filename).read()
    # ...
```

Ukoliko napadač prosljeđuje funkciji ime datoteke koje sadrži ".." (dvije točke), moguće je pristupiti datotekama iznad BASE_PATH. Potrebno je upisati dovoljno znakova ../../ ... da bi se pristupilo datoteci s lozinkama korisnika npr: ../../../../etc/passwd.

Rješenje problema je verifikacija staze da napadač ne bi mogao odabrati direktorij iznad osnovne staze direktorija (eng. *base path*) .

5. Praktični rad

Za prikaz aplikacija stvorenih pomoću Django web razvojnog okruženja stvorena je jednostavna aplikacija - blog aplikacija koja omogućava pisanje blog članaka, objavu i njihovo administriranje. Blog aplikacija odabrana je zbog toga što se pomoću nje prikazuje kreiranje modela, pogleda i predložaka, kreiranje URL konfiguracijskih datoteka te administriranje izgrađene aplikacije. Također, dan je izgled izgeneriranog html kôda u internet pregledniku.

U nastavku prikazan je proces izgradnje blog aplikacije te je dan potpuni programski kod zadane aplikacije.

5.1 Stvaranje blog aplikacije

Početni koraci pri stvaranju blog aplikacije su kreiranje baze podataka. U primjeru je stvorena baza podataka blog.db. Sljedeći korak je stvaranje projekta "web" i nove Django aplikacije "blog" te uređivanje datoteke postavki settings.py.

Izgled promijenjenih dijelova datoteke settings.py je:

```
DATABASE_ENGINE = 'postgresql'
DATABASE_NAME = 'blog.db'
DATABASE_USER = 'root'

DATABASE_PASSWORD = ''
DATABASE_HOST = ''
DATABASE_PORT = ''
...
TEMPLATE_DIRS = (
    '/direktorij_u_kojem_se_nalazi/melita_mihaljevic_os2/web/templates/',
)
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',
    'web.blog',
)
```


U datoteci opcija dodane su informacije o bazi podataka koja je stvorena, dodana je staza do direktorija u kojem će biti smješteni predlošci te su uključene mogućnosti korištenja administratorske stranice i aplikacije blog koja je dio projekta web.

Sljedeći korak je uređivanje datoteke url konfiguracija urls.py. Promjene u datoteci urls.py su:

```
urlpatterns = patterns('',
    (r'^blog/', include('web.blog.urls')),
    (r'^admin/', include('django.contrib.admin.urls')),
    (r'^site_media/(.*)$', 'django.views.static.serve',
    {'document_root':
    '/home/gizmo/melita_mihaljevic_os2/web/media'})),
)
```

Prvom linijom definirano je da svi url-ovi koji započinju sa blog se traže u web.blog.urls Na taj način su svi url-ovi koji pripadaju aplikaciji blog preusmjereni u datoteku konfiguracije url-ova bloga te na taj način odvojeni od dugih aplikacija u projektu. Na taj način smanjena je mogućnost pogreške i povećana sigurnost izgrađeni aplikacija u projektu. Druga linija omogućava korištenje administratorske stranice. Posljednja linije omogućava korištenje medijskih sadržaja u aplikaciji. Medijski sadržaji smješteni su u direktoriju media projekta web, a za korištenje medijskog sadržaja u predloške je potrebno navesti url na sadržaj koristeći oblik:

```
/site_media/direktorij/datoteka_ili_slika
```

Nakon uređivanja datoteke url konfiguracija kreirani su modeli u datoteci models.py Izgled modela je:

```
from django.db import models

class Category(models.Model):

    name = models.CharField(max_length= 100)
    slug = models.SlugField(unique= True,
    populate_from=('name',))

    def __str__(self):

        return self.name

    class Meta:

        verbose_name_plural="Categories"
```

```

class Admin:

    pass

class BlogEntry(models.Model):

    title = models.CharField(max_length= 100)
    slug = models.SlugField(unique= True,
    prepopulate_from=('title',))
    date = models.DateTimeField()
    precis = models.TextField(blank=True)
    content = models.TextField(blank= True)
    categories = models.ManyToManyField(Category, blank=True)

    def __str__(self):

        return self.title

    class Meta:

        verbose_name_plural="Blog entries"

    class Admin:

        pass

```

Dvijema klasama definirana je struktura blog aplikacije. Klasom **Category** definirane su kategorije blog članaka, na primjer: računala, programiranje, glazba i drugi. U klasi su definirani ime kategorije name koje je tipa polje znakova maksimalne duljine 100 znakova. Osim imena definirana je poveznica na ime kategorije koje je tipa poveznice pomoću koje se pristupa imenu i pomoću koje se dohvaćaju kategorije i učitavaju kao link. Poveznica se generira iz imena kategorije na način da se riječi povezuju znakom "-" u jedan niz. Unutar klase kategorija definirana je klasa Meta koja služi za definiranje ispisa množine u administratorskoj stranici. Umjesto "categorys" u administratorsku stranicu upisuje se "categories". Također dodana je klasa **Admin** koja omogućuje prikaz kategorija u administratorskoj stranici. Klasom **BlogEntry** definirana je struktura blog članka. Definiran je naslov, poveznica na naslov, datum stvaranja članka ili uređivanja, kratak pregled članka (eng. *precis*) i potpuni sadržaj članka (eng. *content*). Također omogućen je odabir kategorije. Dodana je i mogućnost prikaza u administratorskoj stranici te izmjena ispisa množine.

Izgled administratorske stranice koja je kreirana prikazan je na slici 5.1:

Django administration

Site administration

Auth		
Groups	+ Add	Change
Users	+ Add	Change
Sites		
Sites	+ Add	Change
Blog		
Blog entries	+ Add	Change
Categories	+ Add	Change

Slika 5.1: Stvorena administratorska stranica blog aplikacije

Na slici 5.2 prikazana je forma za unos blog članka pomoću administratorske stranice:

Add blog entry

Title:	<input type="text" value="prvi post"/>
Slug:	<input type="text" value="prvi-post"/>
Date:	Date: <input type="text" value="2008-01-20"/> Today Time: <input type="text" value="19:41:34"/> Now
Precis:	<div>Ovo je prvi post na blogu</div>
Content:	<div></div>

Slika 5.2: Forma za unos blog članaka

Da bi unos bio valjan i da bi se mogao spremiti u bazu i prikazati u internet pregledniku potrebno je ispuniti sva polja osim polja sadržaj koje je opcionalno polje. Ukoliko nisu unesena sva polja ispisat će se poruka o grešci kao što je to prikazano na slici 5.3:

Add blog entry

❌ Please correct the errors below.



Title:

⚠️ This field is required.

Slug:

⚠️ This field is required.

⚠️ This field is required.

Date: **Date:** Today  **Time:** Now 

Precis:

Slika 5.3: Poruka o grešci pri spremanju forme

Potrebno je unijeti kategoriju kojoj pripada blog članak koji je stvoren. Ukoliko ne postoji kategorija potrebno ju je dodati u formi za kategorije:

Add category

Name:

Slug:

Slika 5.4: Dodavanje kategorije

Nakon definiranja modela kreirani su pogledi, uređena lokalna datoteka url konfiguracija blog aplikacije i kreirani su predlošci za prikaz. Stvoreni pogledi pogledi i predlošci omogućuju prikaz blog aplikacije u internet pregledniku na adresi definiranoj u lokalnoj datoteci url konfiguracija. Stvoreni pogledi smješteni su u datoteci views.py:

```
import django.http as http
import models
import django.shortcuts as shortcuts
def summary(request):

    entries = models.BlogEntry.objects.order_by('-date')
    categories = models.Category.objects.order_by('name')
    return shortcuts.render_to_response("blog/summary.html",
                                       dict(entries=entries, categories=categories))

def detail(request, entry_slug):

    entry= \
shortcuts.get_object_or_404(models.BlogEntry, slug=entry_slug)
```

```

    return \
shortcuts.render_to_response("blog/detail.html",dict(entry=entry)
)

def category(request,category_slug):

    category = shortcuts.get_object_or_404(models.Category,
        slug=category_slug)
    return shortcuts.render_to_response("blog/category.html",
        dict(category=category))

```

Pogledom su definirane 3 funkcije, za svaku stranicu koju je moguće otvoriti. Prva funkcija **summary** definira prikaz članaka bloga sortiranih po datumima objave i kategorija sortiranih prema imenu kategorije. Funkcija definira akcije za prikaz glavne stranice bloga. Posljednjom naredbom definira se prikaz bloga pozivajući html datoteku **summary.html** , prosljeđujući objekte unosa članaka i kategorija html datoteci. Funkcijom **detail** definirana je akcija prikaza detalja članaka bloga ukoliko je naveden detaljniji opis članka ,putem stranice **detail.html**. Posljednjom funkcijom **category** definirana je akcija koja poziva stranicu **category.html** te ispisuje zadane kategorije.

U direktoriju **/template/blog** definirane su html datoteke pogleda koje omogućuju prikaz bloga u internet pregledniku. Definirani predlošci su: **summary.html**:

```

<html>

<head>

    <title> Blog </title>

    <link rel="stylesheet" type="text/css"
href="/site_media/images/style.css" />

</head>
<body>

    <div id = "container">
    <div id = "header"> Primjer Bloga </div>
    <div id = "right-column">
    <h2>Categories</h2>

        {%for category in categories%}

            <p><a \
href='/blog/category/{{category.slug}}/'>:{{category.name}}</a></
p>

```

```

        {%endfor%}

</div>
<div id ="content">

    {%for entry in entries %}
    <h2>{{entry.title}}</h2>
    <h3>{{entry.date|date:"Y-n-j H:i"}}</h3>
    {{entry.precis|linebreaks}}
    {%if entry.content%}
    <a href='/blog/{{entry.slug}}/'>More</a>
    {%endif%}
    <hr>
    {%endfor%}

</div>
</div>

</body>

</html>

```

Prvom označenom linijom definirano je pozivanje.css datoteke kojom je definiran dizajn blog stranice. Preostale označene linije definiraju predložak kreiran pomoću Django. Prvim dijelom definiran je prikaz kategorija. Prikazuju se svi objekti tipa category. Drugim dijelom prikazuju se članci bloga. Za svaki članak koji postoji definiran tipom entry ispisuje se naslov, datum kreiranja, kratak sadržaj, ukoliko postoji ispisuje se link na prošireni sadržaj članka bloga. Izgled blog stranice bez uključenog dizajna je prikazan na slici 5.5:

Primjer bloga

drugi post

2008-1-20 20:16

ovo je drugi post

prvi post

2008-1-20 19:45

Ovo je prvi post na blogu

Slika 5.5: Prikaz bloga bez uključenog dizajna

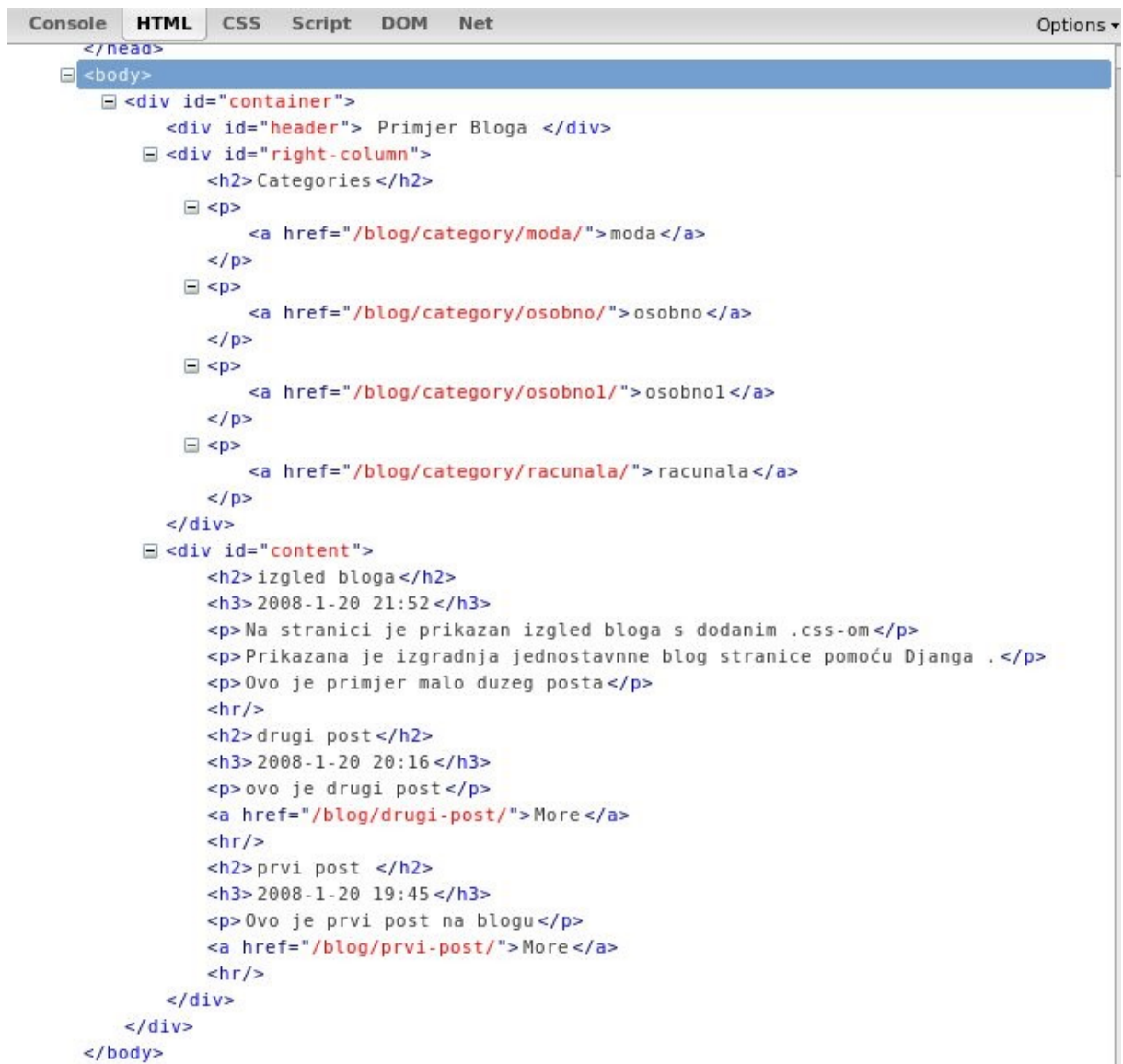
Datoteke predložaka za kategorije i dodatne ispise o člancima bloga analogne su glavnoj stranici bloga te nisu navedene.

Izgled stvorene blog stranice s uključenim dizajnom je :



Slika 5.6: Konačan izgled blog aplikacije

Ukoliko se stranica testira pomoću alata za ispitivanje web aplikacija Firebuga. Izgled izgeneriranog html koda iz predložaka je prikazan na slici 5.7:



```
</head>
<body>
  <div id="container">
    <div id="header"> Primjer Bloga </div>
    <div id="right-column">
      <h2>Categories</h2>
      <p>
        <a href="/blog/category/moda/">moda</a>
      </p>
      <p>
        <a href="/blog/category/osobno/">osobno</a>
      </p>
      <p>
        <a href="/blog/category/osobno1/">osobno1</a>
      </p>
      <p>
        <a href="/blog/category/racunala/">racunala</a>
      </p>
    </div>
    <div id="content">
      <h2>izgled bloga</h2>
      <h3>2008-1-20 21:52</h3>
      <p>Na stranici je prikazan izgled bloga s dodanim .css-om</p>
      <p>Prikazana je izgradnja jednostavne blog stranice pomoću Django .</p>
      <p>Ovo je primjer malo duzeg posta</p>
      <hr/>
      <h2>drugi post</h2>
      <h3>2008-1-20 20:16</h3>
      <p>ovo je drugi post</p>
      <a href="/blog/drugi-post/">More</a>
      <hr/>
      <h2>prvi post </h2>
      <h3>2008-1-20 19:45</h3>
      <p>Ovo je prvi post na blogu</p>
      <a href="/blog/prvi-post/">More</a>
      <hr/>
    </div>
  </div>
</body>
```

Slika 5.7: Prkaz html kôda blog aplikacije otvorene u internet pregledniku

Kako je moguće vidjeti varijable i funkcije zamijenjene su stvaram vrijednostima i prikazane u internet pregledniku. Stvarne vrijednosti dohvaćene su pomoću funkcije pogleda.

Izgrađenim primjerom prikazan je način izgradnje web aplikacija pomoću Django razvojnog okruženja pridržavajući se MTV arhitekture pri stvaranju aplikacije.

6. Zaključak

Sigurnost web aplikacija ponajprije ovisi o arhitekturi na temelju koje je izgrađena i dizajnu. Ručno pisanje koda dug je postupak podložan pogreškama te se zbog toga koristi razvojna okruženja. U seminaru je opisano Django razvojno okruženje te je dan osvrt na sigurnost web aplikacija izgrađenih pomoću Django. Kao praktični rad prikazana je izgrađena blog aplikacija u skladu s Django arhitekturom. Prednosti Django web razvojnog okruženja su:

- Napisan je u programskom jeziku Python što omogućuje korištenje već postojećih modula napisanih u Pythonu za razvoj funkcionalnosti stvorene web aplikacije.
- Slobodno dostupan programski kôd. Moguće je pratiti nove verzije programa te sudjelovati u popravljaju grešaka i dodavanju novih mogućnosti sustavu.
- Arhitektura temeljena na MTV koja odvaja model, logičku strukturu i izgled web aplikacije te na taj način povećava sigurnost.
- Razvojni poslužitelj za testiranje web aplikacija izgrađenih pomoću Django web razvojnog okruženja bez potrebe za konfiguriranjem poslužitelja za testiranje rada izgrađene web aplikacije.
- Ugrađeni sigurnosni mehanizmi za izbjegavanje najpoznatijih napada na web aplikacije, kao što je alat za prevenciju CSRF napada. Django web razvojno okruženje omogućava stvaranje sigurnih web aplikacija.
- Predložci se temelje na Smarty sustavu za izgradnju predložaka. Smarty sustav se koristi prvenstveno za izgradnju predložaka za složene PHP aplikacije te je kao takav prilično popularan. Prelazak na izgradnju predložaka u Django olakšan je za dizajnere predložaka web aplikacija.
- Ugrađena administratorska stranica koja se jednostavno uključuje u izgrađenu web aplikaciju te omogućava njenu administraciju i kontrolu.

Mane Django web razvojnog okruženja su:

- Django web razvojno okruženje je u fazi razvoja te postoje pogreške koje otežavaju izgradnju web aplikacija i ograničavaju funkcionalnost aplikacija izgrađenih pomoću Django.
- Django je napisan u Programskom jeziku Python. Iako je navedena stavka prednost Django web razvojnog okruženja, ona je isto tako i mana zbog toga što dizajneri web aplikacija moraju naučiti programirati u Python programskom jeziku. Do danas su najrazvijenije tehnologije bile .NET, AJAX i programski jezik PHP. Korištenje programskog jezika Python potpuno je novi koncept i zahtjeva dodatnu edukaciju dizajnera web aplikacija.
- Aplikacije izgrađene pomoću Django web razvojnog okruženja imaju problem sa zauzimanjem memorije korištenjem razvojnog poslužitelja.

7. Literatura

- [1] <http://www.djangobook.com/en/beta>
- [2] <http://www.python.org/>
- [3] <http://msdn2.microsoft.com/en-us/library/ms533040.aspx>
- [4] <http://www2.jeffcroft.com/blog/2006/may/02/django-non-programmers/>
- [5] http://en.wikipedia.org/wiki/Dynamic_web_page
- [6] <http://www2.jeffcroft.com/blog/2007/jan/11/django-and-mtv/>
- [7] http://www.djangoproject.com/documentation/request_response/
- [8] <http://www.ibm.com/developerworks/linux/library/l-django/>
- [9] <http://simonwillison.net/2005/Aug/15/request/>
- [10] <http://kurafire.net/log/archive/2006/05/22/django-first-impressions>
- [11] <http://www.amk.ca/python/writing/pycrypt/pycrypt.html>