MTV web framework for Python

Radni okvir za veb (eng. *Web framework*)

- Objektno orijentisan (OO) sistem konstruisan kako bi ga programeri proširili na takav način da obezbede funkcionalnosti koje im se zahtevaju.

- Najbolje programerske prakse su već ugrađene u samom radnom okviru.

Separation of concerns (SoC)

- Razdvajanje odgovornosti
- Monolitna aplikacija → Raslojena aplikacija
- Svaki sloj i komponenta rade samo jednu stvar i ništa osim toga
  - Bolja čitljivost programskog koda
  - Bolji uslovi za testiranje koda
- Primenjujemo razne projektne uzorke:
  - MVC (*Model-View-Controller*)
  - MTV (*Model-Template-View*)

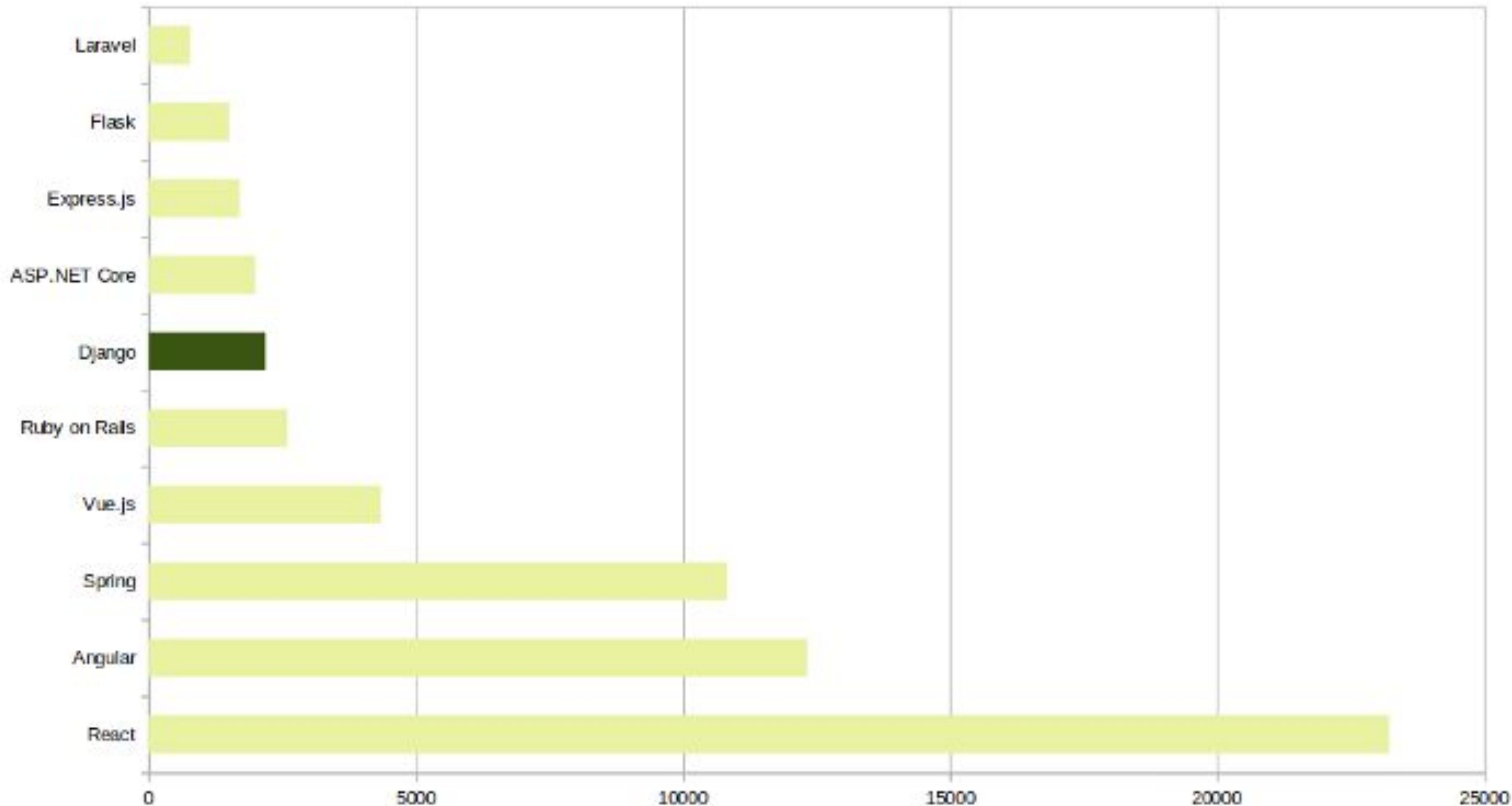
Princip dizajna u veb programiranju

- Celu veb aplikaciju najčešće delimo u 3 celine:
  - Stil i prezentacija: vizuelni izgled veb aplikacije, korisnički interfejs (UI)
  - Poslovna (biznis) logika: način na koji se veb aplikacija ponaša kao odgovor na interakciju od strane korisnika (korisničke radnje)
  - Sadržaj: stvarni podaci koji se prezentuju, najčešće čitanjem iz neke baze podataka (podaci o studentima, predmetima, vesti, itd.)

Danas

- Radni okviri postoje i za veb, ali i desktop aplikacije
- Neki su komercijalni, neki otvorenog koda
  - Java Web: JSP, Struts2, Grails, Play, Vaadin, **JSF, Spring,**...
  - Java Desktop: Swing, Griffon, **JavaFX,**...
  - Java Script: **Angular 2+**, **React**, Vue.js, Ember.js, **Node.js**, Meteor, Backbone.js,...
  - .NET Web: ASP .Net, **ASP MVC,**...
  - .NET Desktop: **WPF**, Windows Forms
  - Php: **Codeigniter, Laravel**, Symfony, CakePHP, Yii, Zend,…
  - Python: **Django**, Flask
  - Ostali: Ruby on Rails, Xamarin,...
  - CSS: **Bootstrap,** Gumby, Foundatin, YAML...
  - Ima ih još MNOGO!
- Odabir? Pametno odabrati, loš izbor košta!!!

Popularnost radnih okvira



Odabir tehnologije / radnog okvira

- Programski jezik: Java, Scala, Groovy, PHP, Ruby, Python, C#, JavaScript,...
  - Šta članovi tima dobro poznaju?
  - Šta nije teško za učenje?
  - Šta je produktivno za programiranje?
    - Tipiziranost, postojeće biblioteke...
- Platforma: JVM, PHP, RVM, .NET,...
  - Šta je robustno?
  - Šta je skalabilno?
  - Koja je namena i kakva je planirana upotreba aplikacije? Koliko brzo treba „izbaciti“ prvu verziju?
- Radni okvir: utiču prethodne dve odluke
  - Šta članovi timova znaju?
  - Kakva je kriva učenja?
  - Koliko je podržan (zajednica, biblioteke, dodaci)?
  - Koliko je produktivno za programiranje?
  - Iskustva drugih?
  - Kakva je namena aplikacije? (nema *one-fits-all* rešenja)

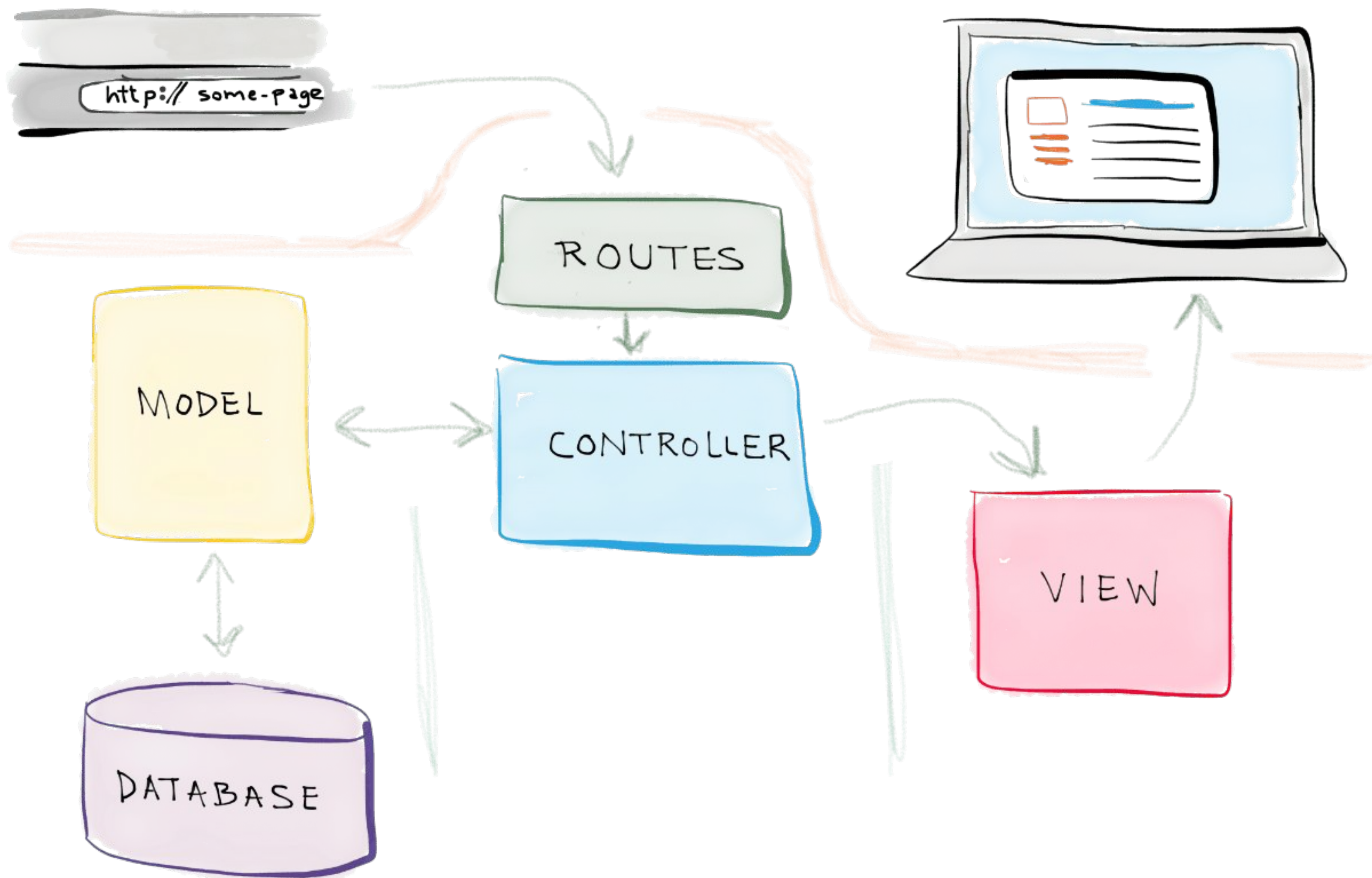
Dogovor umesto konfigurisanja

- Princip *Convention over configuration* je danas jako popularan
- Projekti najčešće imaju identičnu, logičnu strukturu
- Ranije – sve kroz konfiguracione fajlove
  - Java aplikacije: kroz serije XML fajlova, od Java 1.5 kroz anotacije
- Danas: usvojena je razumna pretpostavka
- Aplikacija se strukturirana po unapred definisanim pravilima, uz mogućnost konfigurisanja neočekivanih odluka (*override*)
- Radni okvir forsira određenu strukturu da bi kod obavljao ono što želite
- Posledica: svi učesnici projekta znaju kako su unutar projekta razvrstane komponente; lako je nastaviti tuđi započeti rad, lako pronaći mesto na kome se pojavljuje defekat (*bug*)
- Don’t reinvent the wheel!

MVC ideja

- Razdvojiti model od prezentacije
  - Generisanje dinamičkog HTML-a: **view**
  - Manipulacija podacima: **controller**
  - Podaci koji se razmenjuju između C i V: **model**
    - Tu „pripada“ i dobavljanje podataka iz neke DB
- Grupisati tešku poslovnu logiku na jedno mesto, veb orijentisane stvari na drugo (*controller*)

Projektni uzorak MVC



Projektni uzorak MVC

- Model
  - Podaci kojima se manipuliše kroz aplikaciju
  - Domen problema (Ljude, Knjige,...)
  - Služi da nosi informacije između *Controller*-a i *View*-a
  - Zadužen da perzistira podatke (sama komunikacija sa DB)
  - Zadužen da dobavlja podatke iz baze podataka
- View
  - Komponenta koja prikazuje podatke (*model*) na određeni način
- Controller
  - Komponenta koja prima zahteve (korisnika), pokreće određenu poslovnu logiku, odlučuje šta se čuva u bazi podataka i kada

Dodatak: Services

- Ideja: ne opteretiti kontrolere business logikom
- Kontroler odlučuje koji view prikazati, i koje podatke tada prikazivati, ili kome proslediti izvršavanje
- Ne bi trebalo da sadrži čitav proces i donošenje odluka oko domenskog problema
- To (mozganje) ostaviti servisima
  – Posebne klase koje sadrže kod koji ostvaruje business logiku, ali bez svesti o tome da je business logika potrebna web aplikaciji
- Tada kontroler samo “diriguje”

DJANGO

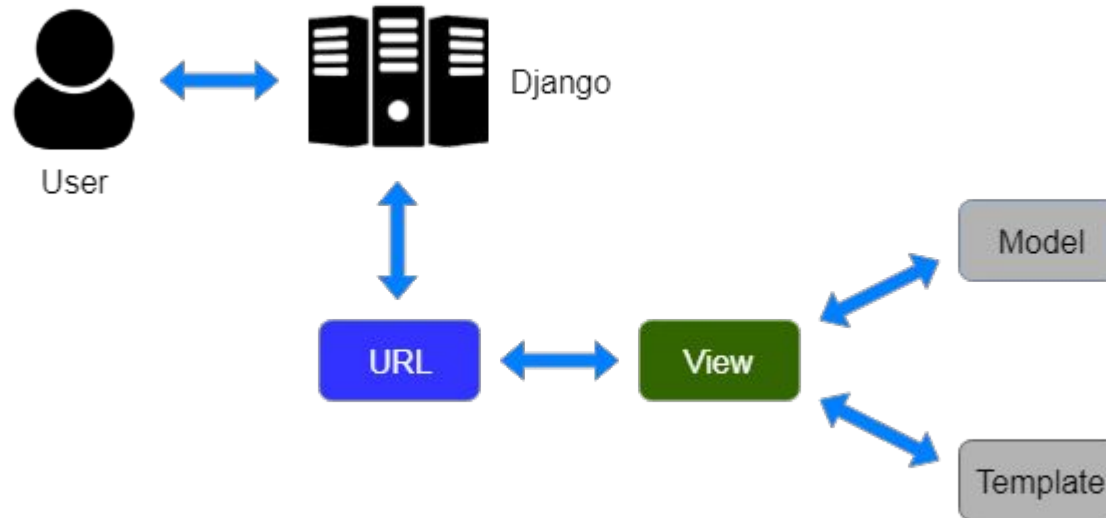
MVT unutar Django-a

- Svaki radni okvir ima sopstvene mehanizme na osnovu kojih ostvaruje razdvajanje rada sa podacima, poslovne logike i prezentacije.

- Django svoje stranice dinamickički renderuje iz template-a, pa Template menja View u Django MVT.

- Controller se u MVT zove View i on se bavi biznis logikom.

MVT i Django



- Zahtev (*request*) ide do servera, a zatim:
  - analizira se URL (*routing*);
  - zahtev se procesira zbog sigurnosnih razloga;
  - na osnovu URL šablona poziva se različita metoda view-a;
  - view izvršava potrebnu obradu uz pomoć modela, biblioteki, pomoćnih funkcija i svih drugih resursa potrebne za obradu specifičnog zahteva;
  - generiše se odgovarajući template koji će korisniku biti prikazan.

Priprema radnog okruženja

- Priprema radnog okruženja moguća je na sledeće načine:
  - Korišćenjem pip

    ```
    pip install django
    ```

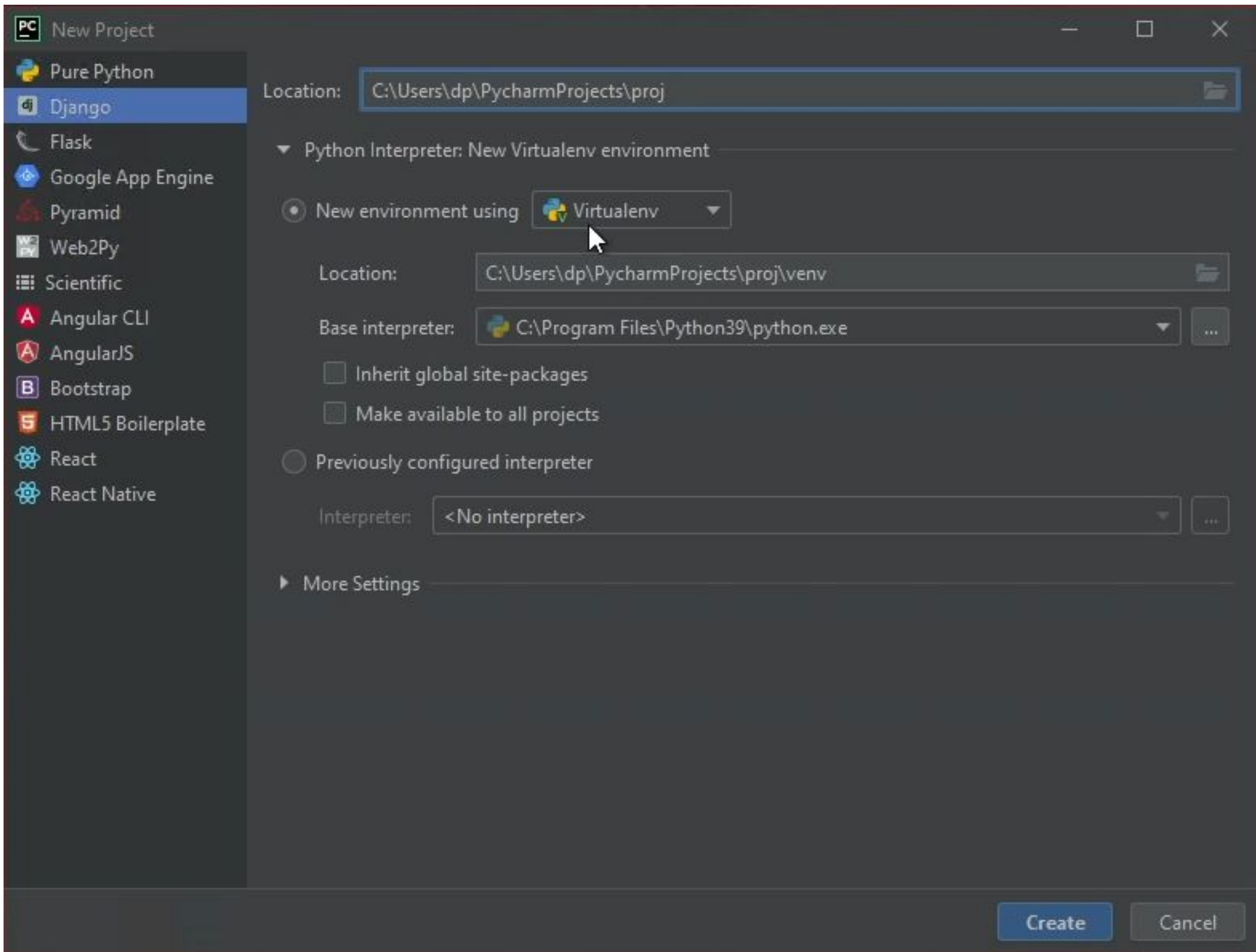
    (obratiti pažnju da verzije djanga prate verzije pythona)
  - Preuzimanje trenutne development verzije sa GIT-a

    ```
    git clone
    https://github.com/django/django.git
    ```

Kreiranje projekta

- Django projekat se može inicijalizovati:
  - komandom:
    ```
    django-admin startproject mysite
    ```
  - pravljenjem novog django projekta u pycharm-u

- Korisno je imati i virtual env za eventualno isporučivanje aplikacije zajedno sa svim neophodnim paketima


New Project
Pure Python
Django
Flask
Google App Engine
Pyramid
Web2Py
Scientific
Angular CLI
AngularJS
Bootstrap
HTML5 Boilerplate
React
React Native
Location: C:\Users\dp\PycharmProjects\proj
Python Interpreter: New Virtualenv environment
New environment using
Virtualenv
Location: C:\Users\dp\PycharmProjects\proj\venv
Base interpreter: C:\Program Files\Python39\python.exe
Inherit global site-packages
Make available to all projects
Previously configured interpreter
Interpreter: <No interpreter>
More Settings
Create
Cancel

Struktura projekta

```
proj/
    manage.py          <- skripta za administraciju
    venv/              <- generisani virtual env
    proj/
        __init__.py    <- inicijalizator servera
        settings.py    <- konfiguracija servera
        urls.py        <- URL matching rute
        asgi.py        <- asgi deploy konfiguracija
        wsgi.py        <- wsgi deploy konfiguracija
```

Pokretanje servera

- Django sadrži lokalni server za razvoj koji koristi lokalni Python interpreter.
- Razvojni server je single-threaded i ne sme se koristiti u produkciji

- Pokreće se izvršavanjem komande:

```
python manage.py runserver
```

i tada projektu možete pristupiti i putem linka:
http://localhost:8000/

Režim razvoja aplikacije (eng. *Development mode*)

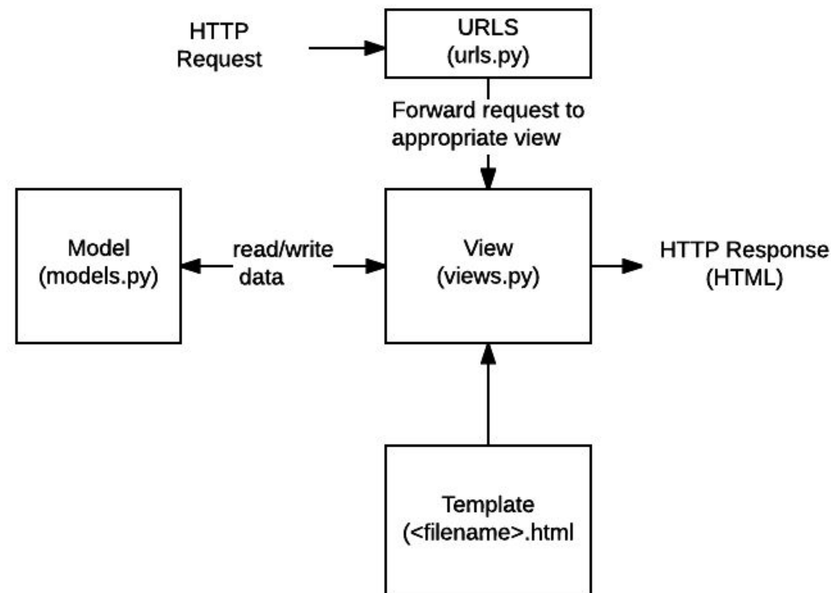
- Projekat je podrazumevano generisan sa debug konfiguracijom.

- Projekat je moguće pokrenuti u produkcionom režimu, ako se u *settings.py* promeni linija
```
DEBUG = False
```

- Razvojni režim omogućava prikazivanje svih nastalih grešaka u pretraživaču, dok se u produkcionom režimu one ne prikazuju.

Kako funkcioniše?

- Prilikom klika na link http://localhost:8000/vesti/vest/3
  - Instancira se `HttpRequest`
  - Server pomoću URL pattern-a pronalazi podudaranje izmedju odgovarajuće metode u View i zadatog URL-a
  - Uz request šalju se i pronadjeni parametri iz zadattog URL-a
    - `<int:br_vesti>` postaje 3



Django aplikacije

- Django projekat predstavlja ceo sajt
- Aplikacije su podmoduli projekta i nasleđuju njegovu konfiguraciju

- Jedan projekat može imati više aplikacija
- Jedna aplikacija može biti iskorišćena u više projekata

- Aplikacija se generiše putem komande:
  - `python manage.py startapp` *`ime_aplikacije`*

Struktura aplikacije

```
polls/
    __init__.py    <- inicijalizacija app-a
    admin.py       <- pristup admina modelima
    apps.py    <- konfiguracija app-a
    migrations/    <- direktorijum migracija
        __init__.py  <- inicijalna migracija
    models.py      <- modul modela
    tests.py       <- testiranje app-a
    views.py       <- modul view-ova
```

VIEW

View

- Pišu se u fajlu views.py kao funkcije jedne aplikacije
- Moguć je i klasni view
- Pozivana funkcija se bira na osnovu patterns iz `urls.py` projekta

- Ulaz funkcije je uvek objekat tipa `HttpRequest`
- View po pravilu ne čuva stanje već se stanje čuva u sesiji

- View pristupa modelu i bavi se poslovnom logikom
- Rezultuje HTML stranicom najčešće generisanom pomoću template-a

Parametri zahteva

- Parametri zahteva se prosleđuju kroz HttpRequest objekat
- Bitna polja HttpRequest:
  - request.method
  - request.GET
  - request.POST
  - request.session
  - request.user
- Kontroler ima polje $request koje će se inicijalizovati prilikom inicijalizacije kontrolera

Parametri zahteva

- $request**->** getGet('ime'), $request**->** getPost('ime') i $request**->** getVar('ime')
  - vraća vrednost elementa iz $_GET , $_POST i $_REQUEST niz
  - vraća null vrednost ukoliko element ne postoji

- $request**->**getMethod()
  - vraća informaciju da li je zahtev POST ili GET

- $request**->**isAJAX()
  - vraća informaciju da li je AJAX zahtev

Vraćanje response-a

- Postoji nekoliko varijacija vraćanja response-a:
  - vratiti prost HttpResponse ili neku njegovu potklasu
  - render template-a
  - redirect na drugu stranicu
  - raised exception koji vraca Http404

```
from django.shortcuts import render, redirect
```

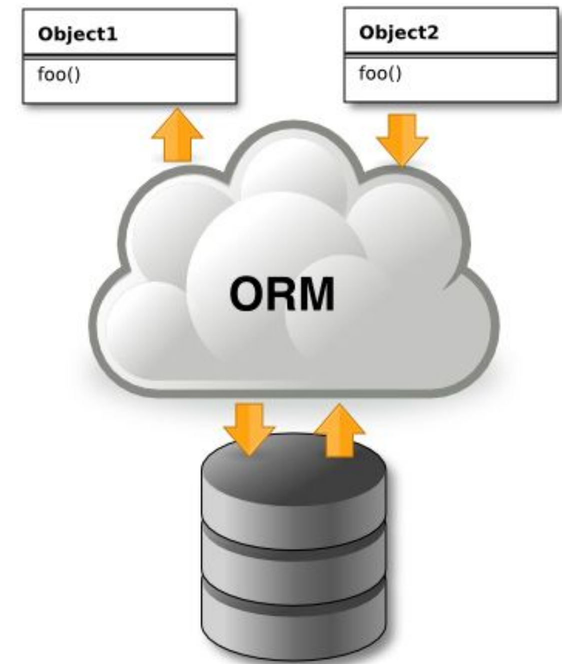
- Svaki slučaj konačno vraća neki tip HttpResponse

- View pristupa modelu i bavi se poslovnom logikom
- Rezultuje HTML stranicom najčešće generisanom iz template-a

MODEL

ORM

- ORM: *Object Relational Mapping*
  - Mehanizam koji omogućuje rad sa objektima na aplikativnom nivou, uz automatski rad sa BP
  - Skup klasa/funkcija koje generišu odgovarajući SQL kod
- Postoje različita rešenja
  - Kod nekih se podrazumeva da je na programeru da promene održava i u BP i u modelu
  - Kod nekih, postoje alati koji generišu tabele i promene u tabelama, na osnovu izmena u klasama modela
  - Kod nekih, neophodno je da se, kroz konfiguracione fajlove specificira kako se model mapira na tabele u BP; drugi analiziraju strukturu tabela i samostalno „shvataju“



Prednosti ORM

- Objektno orijentisani kod, čak i pri pisanju upita
- Bez SQL i preterane brige o valjanosti korisničkih podataka
- Olakšano jedinično testiranje rada aplikacije
  - „mock“-ovati bazu podataka

- Primer:

```
nova_vest = Vest(“naslov”,”sadrzaj”,date.today())
nova_vest.save()
dodata_vest = Vest.objects.get(naslov=’naslov’)
```

Model

- Pišu se u models.py kao klase
- Nasleđuju django.db.models.Model
  - svaki model ima automatski definisan id
- Poseduje Meta potklasu koja sadrži:
  - ime tabele u bazi
  - da li je shema promenljiva iz django-a

- Kod za model moguće je importovati iz baze putem:
  - `python manage.py inspectdb`

Migracija modela

- Model napisan u python-u moguće je migrirati u bazu
  - generisaće se tabele sa nazivima i kolonama definisanim u modelu

- Kod za migraciju se genriše putem komande:
  - `python manage.py makemigrations ime_migracije`
- Migracija se vrši putem komande:
  - `python manage.py migrate ime_migracije`
- Pregled SQL skripte koja vriši migriranje:
  - `python manage.py sqlmigrate ime_migracije`
- Pregled napravljenih migracija:
  - `python manage.py showmigrations ime_migracije`

Upiti

- Klasa modela sadrzi objects nad kojim se mogu izvršiti upiti

```
Vest.objects.all()
Vest.objects.get(pk=1)
q1 = Vest.objects.filter(naslov__contains=term)
q1.exclude(naslov__contains=’[FOTO]’)
Vest.objects.filter(datum__gte=datetime.date(2022,5,6))
    .order_by(‘-datum’) # descending
```

- Za kompleksnije upite moguće je koristiti Q objekte koji imaju preklopljene or, and, not operatore

```
Vest.objects.filter(Q(sadrzaj__icontains=term) |
    Q(naslov__icontains=term))
Vest.objects.filter(Q(sadrzaj__icontains=term) &
    ~Q(autor__username__startswith=’tasha’))
```

TEMPLATE

Template

- Template-i su generični textualni fajlovi (najčešće html) koji se popunjavaju vrednostima iz konteksta View-a
- Templati projekta se čuvaju u root/templates direktorijumu ali svaka aplikacija može imati svoj set template-a u različitom namespace–u
- Vrednosti promenljivih unutar template-a se dinamički odredjuju django template engine-om pre nego što se pošalju korisniku

Template sintaksa

- Sve promenljive unutar template-a potrebno je stavljati u duple zagrade:
  - `{{ naslov }}`
- Promenljive mogu biti klasnog tipa poput rečnika ili nizova:
  - `{{ vest.naslov }}`
  - `{{ niz.0 }}`

- Logičke strukture (tagove) potrebno je pisati poput:
  - `{% if user.is_admin %}Hello, admin.{% endif %}`
  - `{% for vest in vesti %}{{ vest.naslov }}{% endfor %}`
  - `{% block naziv%}{{ vest.autor }}{% endblock %}`

Template generici

- Moguće je zameniti boilerplate kod poput HTML header-a izvođenjem iz generičkih template-a
- `base.html:`

```
<!DOCTYPE html>
<html>
    <head></head>
   <body>
      {% block content %}Base sadrzaj{% endblock %}
   </body>
</html>
```

- `derived.html:`

```
{% extends "base.html" %}
{% block content %}
   <p>Sadržaj izvedenestranice</p>
{% endblock %}
```

Forme

- Nasleđuju django.forms.Form
- Svakom polju forme potrebno je definisati adekvatno polje u klasi forme:

```
class VestForm(forms.Form):
    autor = forms.CharField(label='autor', max_length=32)
    sadrzaj = forms.CharField(label='autor', max_length=99)
    datum = forms.DateField(initial=datetime.date.today)
```

- Objekat forme se može kreirati sa argumentom request.POST gde se adekvatna polja popunjavaju vrednostima iz zahteva

Model forme

- Nasleđuju django.forms.ModelForm
- Implicitno uzima polja modela na koji ukazuje
- Mora biti definisan class Meta

```
class VestForm(forms.modelForm):
    class Meta:
        model = Vest
        fields = [‘naslov’, ‘sadrzaj’] # ‘__all__’
        # exclude = [‘autor’]
```

- Poseduje preklopljenu metodu save koja kreira i cuva objekat modela

Izgled forme

- Formin izgled moguće je menjati programatično:

```
class SearchForm(Form):
    term = forms.CharField(max_length=50, required=True)
    CHOICES = [
        ('mod', 'Moderator'),
        ('default', 'Default'),
    ]
    group = forms.ChoiceField(choices=CHOICES,
required=True, widget=forms.RadioSelect, initial=CHOICES[0])
```

- Forme se dublje mogu menjati preko [crispy frameworka]()

Forme u template-ima

- Forme se mogu direktno ugraditi u template kao objekti:
  - Neophodno je naznačiti post metodu
  - Submit dugme služi kao potvrda slanja forme

```
<form method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Submit">
</form>
```

- Dodavanjem taga `csrf_token` forma se štiti od *cross-site request forgery* napada

AUTENTIKACIJA

Authentication framework

- `django.contrib.auth`
- Django poseduje ugradjenu aplikaciju za autentikaciju
- Moguće je proširiti ugradjene modele

```
class Korisnik(AbstractUser):
    br_objavljenih_vesti = model.IntegerField()
```

- Pre prve migracije neophodno je promeniti ukazani model za autentikaciju u settings.py!

```
AUTH_USER_MODEL = 'vesti.Korisnik'
```

Login

- Ugradjena funkcija authenticate pristupa izabranom autentikacionom modelu u bazi
- Ulogovani user se cuva u session storage-u unutar response

```
user = authenticate(username=username,password=password)
login(request, user)
```

- Objekat user-a se briše iz session storage-a funkcijom logout

```
logout(request)
```

Ograničavanje view-ova

- Moguće je proveriti da li je trenutni korisnik ulogovan unutar view-a

```
request.user.is_authenticated
```

- Objekat user postoji iako korisnik nije ulogovan!
  - AnonymousUser

- View je moguće ograničiti i pomoću dekoratora:

```
@login_required(login_url='login/')
def view(request):
    pass
```

Permisije

- Django generiše permisije za svaki definisan contenttype (model)
- Permisije se mogu dodeliti na nivou korisnika ili na nivou grupe
- Korisnici mogu biti asocirani sa više grupa

- Permisije su samo stringovi, ograničenja se moraju definisati!

```
if request.user.has_perms('vesti.add_vest'):
    pass

@permission_required('vesti.add_vest',raise_exception=True)
def create_vest(request):
    pass
```

Autentikacija u templatima

- U templat-u uvek je moguće pristupiti objektu user, perms, request

```
{% if user.is_authenticated %}
{% if perms.vesti.delete_vest %}
{% for vest in request.session[‘vesti’] %}
```

- Django server neće slati klijentu podatke koji nisu renderovani u template-u
  - Deo logike može biti ovde
  - Sigurnost podataka

TESTIRANJE

Django TestCase

- `django.test.TestCase`
- Django poseduje ugradjen framework za testiranje
- Testovi se pokreću nad privremenim modelom
  - potrebno je inicijalizovati vrednosti

- Testovi su metode `TestCase` i moraju počinjati prefixom `test_`

```
class IndexTest(TestCase):
    def test_show_vesti(self):
        create_group('mod')
        kor = create_mod('tasha')
        create_vest(kor, 'Test vest')
        response = self.client.get("")
        self.assertContains(response, 'Test vest', html=True)
```

Assert

assertFieldOutput
assertFormError
assertFormsetError
assertContains
assertTemplateUsed
assertURLEqual
assertRedirects
assertHTMLEqual
assertXMLEqual
assertInHTML
assertJSONEqual
assertQuerysetEqual
assertNumQueries

- Postoje i not-ovane verzije:
  - assertNotContains
  - assertHTMLNotEqual
  - assertXMLNotEqual
  - assertJSONNotEqual

Fixtures

- Moguće je iskoristiti iste podatke za datu TestCase klasu
- Fixirani podaci se definišu u zasebnom .json fajlu:

```
[{
    "model": "auth.group",
    "pk": 1,
    "fields": {
        "name": "mod",
        "permissions": []
    }
}]
```

- Dodaje se klasi preko:

```
class IndexTest(TestCase):
  fixtures = ["group.json"]
```

MISC

Rad sa fajlovima

- Modelu je moguće pridružiti media fajl
- Potrebno je definisati media folder ili CDN server:

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

- Definisati putanju do fajlova:

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

- Model samo čuva ime fajla -> polje u bazi će biti tipa string

```
class Korisnik(AbstractUser):
    file = models.FileField(upload_to=’files/',null=True)
    pfp = models.ImageField(upload_to='imgs/',null=True)
```

Messages framework

- `django.contrib.messages`
- Django poseduje ugradjenu aplikaciju za poruke
- Moguće je dodati novu poruku unutar view-a

```
messages.add_message(request, messages.INFO, ‘Hello’)
messages.info(request, ‘Hello’)
```

- Ugradjeni tipovi poruka

```
DEBUG, INFO, SUCCESS, WARNING, ERROR
```

- Poruke u template-u

```
{% if messages %}
    {% for message in messages %}{{ message }}{% endfor %}
{% endif %}
```