



Distance Learning System

Django framework

Web Application Development

Uvod u Django

<https://www.djangoproject.com/>



- Django je Python okvir za kreiranje **web** aplikacija
- Django aplikacije arhitektonski koriste **MVT** (derivat MVC-a)
- Django je besplatan i otvorenog koda
- ...

Preuzimanje i aktivacija

- Django nije uključen u standardnu Python biblioteku, pa ga treba preuzeti iz eksternog izvora

pip install django

- Provera ispravnosti i verzije

python -m django version

Menadžment alati

<https://docs.djangoproject.com/en/3.0/ref/django-admin/>

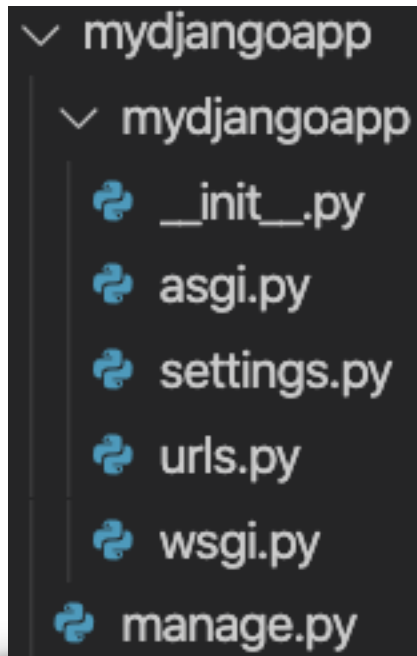
- Kao i drugi okviri, i Django uključuje menadžment alat **django-admin**
- Django-admin se može koristiti za kreiranje projekata ili njegovih delova, migraciju baze, kreiranje korisnika i slično
- Alat se može koristiti direktno, ili posredno putem **manage.py** skripte ili komande **python -m django**
- Manage.py skripta se generiše automatski prilikom kreiranja django projekta i prilikom startovanja učitava podešavanja tekućeg projekta, pa se preporučuje njeno korišćenje kada se radi na pojedinačnim projektima

Kreiranje projekta

Postoji nekoliko fajlova neophodnih za funkcionisanje Django projekta

Ove fajlove možemo kreirati ručno ili pomoću alata **django-admin**

```
django-admin startproject mydjangoapp
```



Startovanje servera i aplikacije

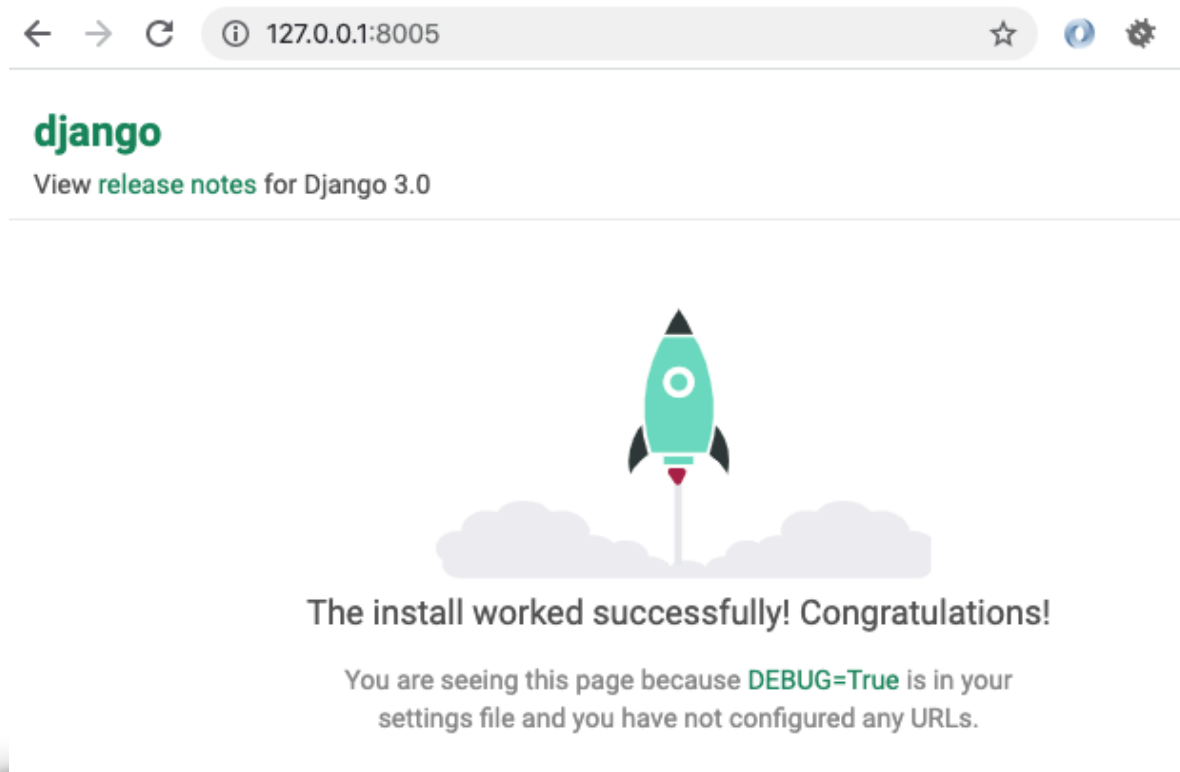
- Kreirani projekat može se odmah startovati i testirati pomoću ugrađenog development servera

```
python manage.py runserver 8005
```

```
Django version 3.0.2, using settings 'mydjangoapp.settings'  
Starting development server at http://127.0.0.1:8005/  
Quit the server with CONTROL-C.
```

Pristupanje aplikaciji

Ukoliko je aplikacija ispravno napravljena i startovana pomoću development servera, ona će biti dostupna putem web pregledača



Rutiranje

- Podrazumevano, fajl/modul sa rutama aplikacije je **urls.py**
- Promenljiva urlpatterns sadrži listu svih ruta koje je aplikacija u stanju da prepozna

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

- Tokom izgradnje aplikacije, unutar ove liste naći će se različite rute za različite sekcije

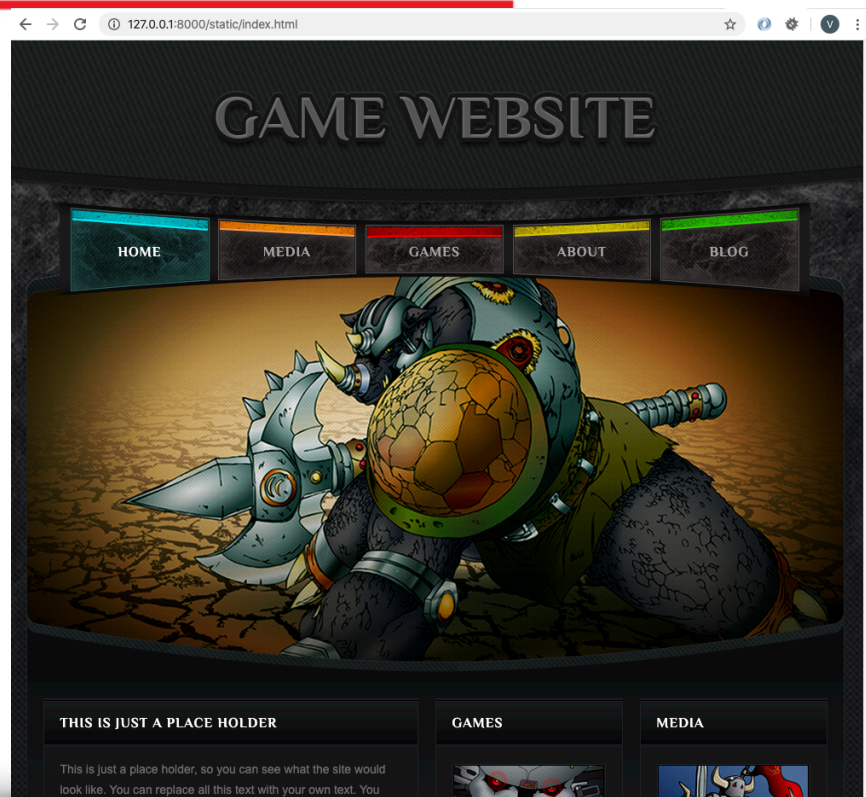
```
urlpatterns = [  
    path('users', views.all_users),  
    path('user/<int:id>', views.user_by_id),  
    path('search/<int:category>/<str:name>', views.search_products)  
]
```


Statički fajlovi

(wab-ex06 mydjangoapp)

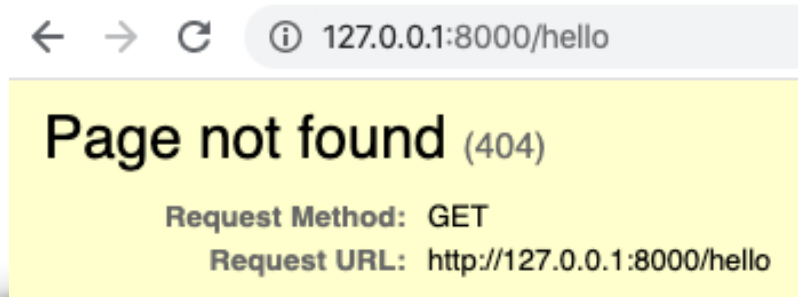
- Da bi aplikacija mogla da hostuje statičke fajlove u developerskom režimu, u modulu/fajlu settings.py postavljaju se promenljive **STATIC_URL** i **STATICFILES_DIRS**
- **STATIC_URL** predstavlja url putanju statičkih resursa
- **STATICFILES_DIR** predstavlja putanju do statičkih resursa na fajl sistemu

```
STATIC_URL = '/static/'  
STATICFILES_DIRS = [  
    os.path.join(BASE_DIR, "static"),  
]
```



Pogledi

- Django **pogledi** u MVT šablonu odgovaraju **kontrolerima** u MVC šablonu
- Pogledi sadrže funkcionalnosti koje su povezane sa definisanim **rutama**
- Django sadrži predefinisane poglede za standardne operacije (obrade grešaka i statičke fajlove)



Korisnički definisani pogledi

- Da bismo kreirali sopstveni pogled, treba povežemo funkciju sa odgovarajućom rutom
- Funkcija vraća objekat HttpResponse

```
from django.http import HttpResponse

urlpatterns = [
    path('hello', lambda request: HttpResponse("Hey man, how are you?"))
]
```

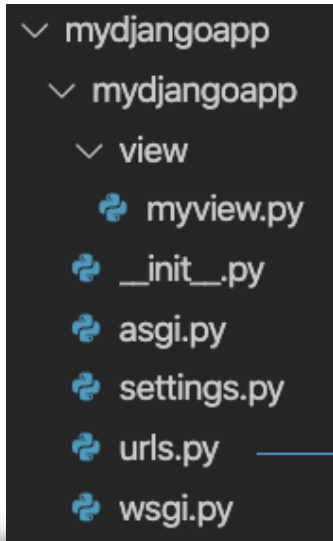


127.0.0.1:8000/hello

Hey man, how are you?

Fajlovi pogleda

- Najčešće se pogledi tematski grupišu, tako da jedan fajl sadrži jednu ili više funkcija koje se odnose na jednu sekciju aplikacije ili kompletnu aplikaciju



```
from django.http import HttpResponse  
def welcome(request):  
    return HttpResponse("Hello and welcome!")
```

```
import mydjangoapp.view.myview as myview  
  
urlpatterns = [  
    path('welcome', myview.welcome)  
]
```

Parametrizacija

- Parametrizacija pogleda se može rešiti:
 - Pomoću URL parametara (url encoded query string)
`http://someexamplesite?a=2&b=3`
 - Pomoću POST parametara (parametri poslati kroz telo zahteva)
a=2&b=3
 - Pomoću parametara putanje (modifikovana putanja)
`http://someexamplesite/2/3`
- Sve pomenute parametrizacije mogu da se kombinuju

Parametrizacija putem url-a

- Svaka funkcija pogleda dobija automatski request objekat
- Ovaj request objekat, između ostalog, sadrži i parsirane get (url) i post parametre zahteva

```
def add(request):  
    a = int(request.GET.get("a"))  
    b = int(request.GET.get("b"))  
    return HttpResponse(f"Result is: {a+b}")
```



127.0.0.1:8000/add?a=2&b=3

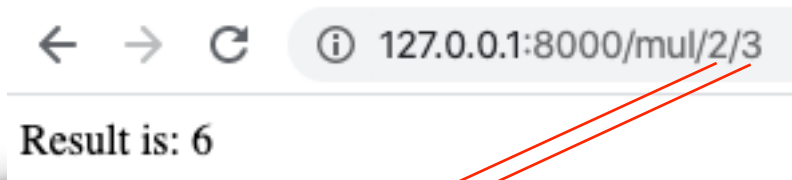
Result is: 5

Path parametrizacija

- Da bi se delovi putanje tretirali kao parametri potrebno je definisati posebnu putanju sa predviđenim prostorom za parametre

```
path('mul/<int:a>/<int:b>', mathview.mul)
```

- Parametri će zatim biti automatski prosleđeni pogledu ukoliko zahtev odgovara putanji



```
def mul(request, a, b):  
    return HttpResponse(f"Result is: {a*b}")
```

Django šabloni

- Najčešće se HTML kod ne emituje direktno iz pogleda, već se umesto toga, učitava šablon
- Šablon (Template) u MVT-u odgovara pogledu u MVC-u
- Django ima sistem automatskog učitavanja šablona
- Django ima sistem parsiranja šablona (**DTL** - Django template Language)
- Šabloni omogućavaju ponovnu upotrebljivost HTML-a
- Django već sadrži šablone za standardne slučajeve (master detail i slično)

Template i Context

- Dve klase za generisanje sadržaja pomoću šablona su Template i Context.

```
from django.template import Template  
from django.template import Context
```



```
t = Template("Hello {{username}}")  
c = Context({"username": "Peter"})  
output = t.render(c)
```



Hello Peter

Podešavanje Django šablona

- Šabloni su fajlovi sa html ekstenzijom i oni treba da se nalaze u jednom ili više direktorijuma
- Ove direktorijume pridružiti aplikaciji unutar fajla settings.py

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR,"templates")],  
        'APP_DIRS': True,  
        'OPTIONS': {
```



Upotreba Django šablona

- Ispravno definisani fajl šablona može se učitati na više načina

```
import django.template.loader
```

```
def welcome(request):  
    return HttpResponse(  
        django.template.loader.render_to_string("hello.html")  
    )
```

```
from django.shortcuts import render
```

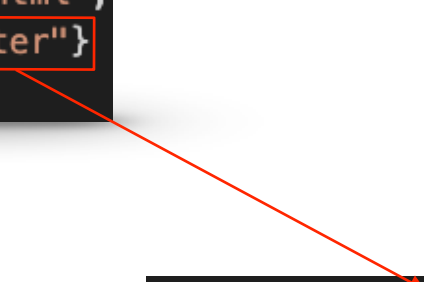
```
def welcome(request):  
    return render(request, "hello.html")
```

Parametrizacija šablona

- Šabloni najčešće sadrže predefinisana mesta za dinamičke delove
- Ovi dinamički delovi popunjavaju se na osnovu parametara poslatih šablonu prilikom kreiranja

```
def welcome(request):  
    return render(  
        request, "hello.html",  
        {"username": "Peter"}  
    )
```

- Prosleđeni parametri, dostupni su u šablonu kroz DTL tagove



```
Hello {{username}}
```

Procesori sadržaja

- Prilikom aktivacije, django učitava različite procesore sadržaja

```
'context_processors': [  
    'django.template.context_processors.debug',  
    'django.template.context_processors.request',  
    'django.contrib.auth.context_processors.auth',  
    'django.contrib.messages.context_processors.messages',  
],
```

- Procesori sadržaja dodaju promenljive na raspolaganje u kontekst

Django Template Language

<https://docs.djangoproject.com/en/2.2/ref/templates/language/>

- DTL poznaje promenljive (izraze), tagove (direktive) i filtere
- Tagove obično kontrolišemo ponašanje šablona, a prepoznajemo ih po oznaci %

Uslovno prikazivanje
promenljive

```
{% if loggedin %}  
Hello {{username}}  
{% endif %}
```

Učitavanje eksternog
šablona

```
{% include "abc.html" %}
```

Iteracija liste

```
{% for user in users %}  
{{user}}  
{% endfor %}
```

Promenljiva




Nasleđivanje šablona

- Nasleđivanjem šablona, omogućavamo koncept šablon strane, u koju se učitavaju različiti sadržaji

masterpage.html

```
My master<br>{% block my_content %}{% endblock %}
```



```
{% extends "masterpage.html" %}
{% block my_content %}
Hello from partial!
{% endblock %}
```

Filteri

- Pomoću filtera, vršimo dodatnu manipulaciju nad sadržajem u trenutku njegovog procesiranja
- Filteri se označavaju vertikalnom crtom (pipe) u okviru izraza

```
{{ "Hello world" | upper }}
```

HELLO WORLD

Korisnički definisani filteri i tagovi

<https://docs.djangoproject.com/en/2.2/howto/custom-template-tags/>

- Osim ugrađenih, moguće je koristiti i korisnički definisane filtere i tagove
- U tom slučaju:
 - aplikacija mora biti registrovana u listi aplikacija (settings.py)
 - mora postojati paket templatetags unutar direktorijuma aplikacije
 - Unutar fajla mora biti učitani i instancirani template library objekti

```
INSTALLED_APPS = [  
    'mydjangoapp',
```

```
    mydjangoapp  
        templatetags  
            __init__.py
```

```
from django import template  
register = template.Library()
```

Korisnički definisani filteri i tagovi

- Filteri i tagovi se registruju metodama ili dekoratorima

```
@register.filter
def greet(value):
    return f"Hello {value}"
```

ili

```
def greet(value):
    return f"Hello {value}"

register.filter("greet",greet)
```

```
{% load custom_tags %}
{{ "Peter" | greet }}
```



Hello Peter

Korisnički definisani tagovi

- Po istom principu kao filtere, moguće je definisati i tagove
- Parametri tag funkcija se automatski mapiraju sa atributa taga

```
@register.simple_tag
def mytag(color):
    return django.utils.html.format_html(f'''
    <div style='padding:4px;background-color:{color};color:white;'>
    Hello
    </div>''')
```

`{% mytag color="blue" %}`



Hello

Rad sa modelom

<https://docs.djangoproject.com/en/3.0/ref/databases/>

- Django sadrži sistem za objektno relaciono mapiranje (ORM) za rad sa podacima
- Bazu podataka treba postaviti unutar fajla settings.py

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'sakila',  
        'USER': 'root',  
        'PASSWORD': '',  
        'HOST': 'localhost',  
        'PORT': '3306',  
    }  
}
```

Kreiranje model klasa i migracija

- Klase modela su entiteti koje nameravamo da koristimo u aplikaciji
- Ove klase nasleđuju klasu **django.db.models.Model**
- Prilikom kreiranja klase, kolone/polja se definišu kroz objekte modela

<https://docs.djangoproject.com/en/3.0/ref/models/fields/>

```
class GameSkin(models.Model):  
    skin_name = models.CharField(max_length=50)  
    skin_price = models.DecimalField(max_digits=6, decimal_places=2)  
    skin_hero = models.CharField(max_length=50)
```

- Nakon kreirane model klase, generiše se migracioni fajl komandom makemigrations
- Kreirane migracije izvršavamo nad bazom komandom migrate

```
python3 manage.py makemigrations
```

```
Migrations for 'mydjangoapp':  
mydjangoapp/migrations/0002_gameskin.py  
- Create model GameSkin
```

```
python3 manage.py migrate
```

```
Running migrations:  
Applying mydjangoapp.0002_gameskin... OK
```

Manipulacija podacima

<https://docs.djangoproject.com/en/3.0/topics/db/queries/>

- Registrovane klase modela nasleđuju standardne setove CRUD operacija i omogućavaju laku manipulaciju podacima

Kreiranje i čuvanje objekta

```
skin = appmodel.GameSkin(  
    skin_name="Union Jack Fiddlesticks",  
    skin_price=12.5,  
    skin_hero="Fiddlesticks"  
)  
skin.save()
```

Preuzimanje svih objekata

```
skins = appmodel.GameSkin.objects.all()
```

Preuzimanje objekata po kriterijumu

```
skins = appmodel.GameSkin.objects.filter(skin_price__gt=100)
```

Preuzimanje po primarnom ključu

```
skin = appmodel.GameSkin.objects.get(pk=2)
```

Sortiranje

```
skins = appmodel.GameSkin.objects.order_by("id")
```

```
skins = appmodel.GameSkin.objects.order_by("-id")
```

Paging i limitiranje

```
skins = appmodel.GameSkin.objects.all()[2:4]
```



Dodavanje modela u admin sistem

admin.py

```
from django.contrib import admin
import mydjangoapp.models
admin.site.register(mydjangoapp.models.GameSkin)
```

Rad sa formama

```
from django import forms
class SkinForm(forms.Form):
    skin_name = forms.CharField(label='Skin name')
    skin_price = forms.IntegerField(label="Skin price")
    skin_hero = forms.ChoiceField(
        choices=((1,"Fiddlesticks"),(2,"Garen"),(3,"Vayne"))
    )
```

```
def product(request):
    form = frm.SkinForm(request.POST)
    return render(request,"product.html",{"form":form})
```

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


Autentifikacija i autorizacija

<https://docs.djangoproject.com/en/3.0/topics/auth/default/>

Da li je korisnik ulogovan

```
{{ user.is_authenticated }}
```

Izlogovanje

```
logout(request)
```

Kreiranje korisnika

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

```
user = User.objects.create_user('peter', 'peters@email.com', '123')
user.is_active = False
user.save()
```

Autentifikacija i logovanje

```
res = authenticate(username="peter", password="123")
if res:
    login(request, res)
```

Vežba

Python Development - Create Freelance Marketplace Site

Full Stack Development

Posted 11 minutes ago

🔗 Specialized profiles can help you better highlight your expertise when submitting proposals to jobs like these. [Create a specialized profile.](#)

To Create Freelance Marketplace Website with Pages in .pdf attachment.

To Include:

- Responsive UI
- Messaging System
- Support Ticketing System
- 2 Factor Auth
- Calendar Booking System
- Payment Gateway
- Google API (Search by Location)

💰 **\$400**
Fixed-price

💰 **Intermediate level**
I am looking for a mix of experience and value

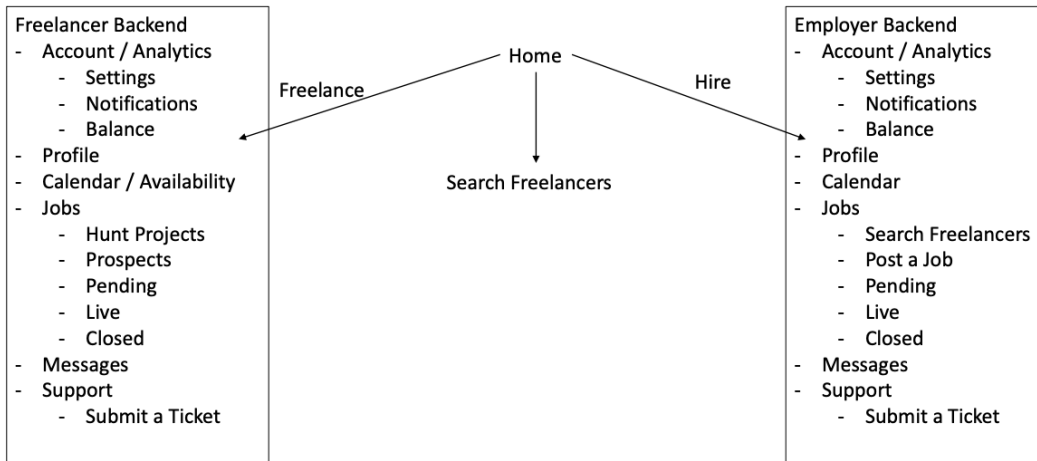
Attachment

📎 [r88s.pdf \(48 KB\)](#)

Project Type: One-time project

You will be asked to answer the following questions when submitting a proposal:

1. Please include examples of similar projects



[FAQ](#) | [Terms of Service](#) | [Privacy Policy](#) | [Contact](#)