Končno poročilo

|  |  |
| --- | --- |
| ***Predmet*** | |
| *Naziv predmeta:* | **Storitve v mobilnih komunikacijah** |
| *Študijski program:* | MAG Telekomunikacije |
| *Nosilec predmeta:* | Andrej Žgank |
| *Asistent:* | Danilo Zimšek |
| *Študijsko leto:* | 2019/2020 |

|  |  |
| --- | --- |
| **Študent(ka)** | |
| *Vpisna številka:* | E5036871 |
| *Priimek in ime:* | Enej Unterajter |
| *Datum:* | 20.1.2020 |

Vsebina

[Kazalo slik 3](#_Toc30553462)

[Uvod 4](#_Toc30553463)

[Razvojna okolja in programski jeziki 5](#_Toc30553464)

[Flask 5](#_Toc30553465)

[PyCharm 5](#_Toc30553466)

[Android Studio 5](#_Toc30553467)

[Namestitev 6](#_Toc30553468)

[Test 8](#_Toc30553469)

[Izdvedba projekta PyCharm in Flask 9](#_Toc30553470)

[Nalaganje slike v mapo strežnika 9](#_Toc30553471)

[Koda ki smo jo uporabili za nalaganje slike na strežnik 11](#_Toc30553472)

[Prikaz slike na strežniku 14](#_Toc30553473)

[Koda ki smo jo uporabili za prikaz slike na strežniku 17](#_Toc30553474)

[Končna izvedba strežnika 19](#_Toc30553475)

[Koda končnega strežnika 21](#_Toc30553476)

[Android aplikacija 26](#_Toc30553477)

[Koda Android aplikacije 29](#_Toc30553478)

[Navodila za uporabo 30](#_Toc30553479)

[Primeri 34](#_Toc30553480)

# Kazalo slik

[Slika 1:prikaz razvojnega okolja PyCharm 6](#_Toc30553481)

[Slika 2:namestitev Flask 7](#_Toc30553482)

[Slika 3:ustvarjen projekt z datoteko in uspešno prevedenim odsekom kode 8](#_Toc30553483)

[Slika 4:prikaz aktivnega strežnika 9](#_Toc30553484)

[Slika 5:prikaz izgleda naše spletne strani 9](#_Toc30553485)

[Slika 6:če datoteka ni izbrana 9](#_Toc30553486)

[Slika 7:če izberemo napačno datoteko 10](#_Toc30553487)

[Slika 8:če izberemo pravilno datoteko 10](#_Toc30553488)

[Slika 9:uspešno nalaganje datoteke 10](#_Toc30553489)

[Slika 10:prikaz datoteke v strežniški mapi 10](#_Toc30553490)

[Slika 11:prikaz prazne mape pred zagonom programa 1 14](#_Toc30553491)

[Slika 12:prikaz prazne mape pred zagonom programa 2 14](#_Toc30553492)

[Slika 13:uspešno uploadana slika na strežnik 14](#_Toc30553493)

[Slika 14:prikaz slike na spletni strani 15](#_Toc30553494)

[Slika 15:prikaz slike v strežniški mapi 15](#_Toc30553495)

[Slika 16;uspešno pošiljanje slike 15](#_Toc30553496)

[Slika 17:prikaz naložene slike 16](#_Toc30553497)

[Slika 18:prikaz namestitve PyTorch 19](#_Toc30553498)

[Slika 19:pregled končnega strežnika zaznave objektov 20](#_Toc30553499)

[Slika 20:prikaz aktivnega strežnika 20](#_Toc30553500)

[Slika 21:osnovni prikaz aplikacije 26](#_Toc30553501)

[Slika 22:prikaz vrnjene slike s strežnika 27](#_Toc30553502)

[Slika 23:prikaz slik na strežniku 28](#_Toc30553503)

[Slika 24:iskanje slike na spletu 30](#_Toc30553504)

[Slika 25:prenos slike na telefon 30](#_Toc30553505)

[Slika 26:vnos IP naslova strežnika in vrat 31](#_Toc30553506)

[Slika 27:izbira željene slike 32](#_Toc30553507)

[Slika 28:izbira gumba connect 32](#_Toc30553508)

# Uvod

Računalniški objekti za prepoznavo objektov se danes dosti pogostaje uporabljajo, prav tako pa so vedno bolj napredni. To predstavlja pomemben korak naprej na področju znanosti. Še ne nekaj let nazaj, je bilo nemogoče da bi računalnik lahko zaznal oz. prepoznal posamezne podobe/objekte, predvsem če so si podobe med seboj podobne (pes – mačka,…). Danes napredni sistemi lahko podobo že razloćijo s kar 99% natančnostjo. Prepoznavni objekti so tako čedalje bolj učinkoviti ne glede na zunajne spremembe (npr. pri gibanju, bo prepoznava lahko sledila tudi pri večji hitrosti).

V našem projektu smo tako poskušali narediti prepoznavo objektov. Za projekt smo uporabili aplikacijsko ogrodje Flask skupaj s python programskim jezikom. Za razvoj smo uporabili razvojno okolje PyCharm. Za razvoj aplikacije smo uporabili razvojno okolje Android.

Želeli smo ustvariti projekt, pri katerem iz mobilnega telefona z opercijskim sistemom Android pošljemo na strežnik določeno sliko, strežnik sliko obdela, jo prepozna in vrne rezultat prepoznave.

# Razvojna okolja in programski jeziki

## Flask

Flask je preprosto spletno ogrodje in je zasnovano za hiter in enostaven razvoj, s katerim pa lahko razvijamo tudi kompleksnejše aplikacije. Flask je spletno ogrodje napisano v jeziku Python. Imenovano je tudi mikroogrodje, ker ne zahteva posebnih orodij ali knjižic. Primeri spletnih strani ki uporabljajo Flask ogrodje so Pinterest, Linkedin, uradna spletna stran Flaska,…. Začel se je kot preprosto ogrodje, kasneje pa je postal eden izmed najpupolarnejših ogrodij za spletne aplikacije v programskem jeziku Python. Flask je postal popularen med python entuziasti, okoli leta 2016 pa je postalo popularno spletno razvijalno ogrodje za python in je bilo izglasovano za najbolj popularno ogrodje za python razvijanje v letu 2018.

Flask omogoča predloge, vendar jih ne vsiljuje, tako se razvijalec lahko odločči za orodja in knjižice katere želi uporabiti. V ogrodju Flask najdemo tudi veliko razširitev, katere lahko prispevajo vsi uporabniki, tako se lahko doda tudi nova funkcionalnost.

Flask omogoča razvoj serverjev in razhroščevanje, RESTful zahteve, Junja2 predloge, varne piškotke (na sejni strani odjemalca), unicode, dokumentacijo, Googlov aplikacijski engine, nabor končnic za željene ukaze.

## PyCharm

PyCharm je integrirano razvojno okolje (IDE), ki se uporablja za računalniško programiranje, predvsem v programskem jeziku Python. Razvit je s strani Češkega podjetja JetBrains in omogoča analizo kode, grafični razhroščevalnik, integrirano testiranje in omogoča spletni razvoj.

Zaradi svoje priljubljenosti ga tako lahko namestimo na katerikoli operacijski sistem (Windows, MacOS in Linux).

Lastnosti ki jih omogoča okolje je pomoč pri kodiranju in analizi (preverjanje sintaktičnih napak, hitri popravki,…, navigiranje po projektu (preklapljanje med datotekami,…), omogoča spletne okvirje (kot je tudi Flask), integriran python razhroščevalnik, Googlov aplikacijski engine,….

V okolje lahko vključimo tudi API-je, razvijalec lahko napiše svoje vtičnike in s tem razširi dodatno delovanje PyCharm okolja. Večina drugih vtičnikov (ki niso ustvarjeni s strani JetBrains podjetja tudi delujejo v okolju, tako je na voljo več kot 1000 vtičnikov.

PyCharm je bil razvit leta 2010 in je odprtokodno okolje, omogoča pa tudi nakup različnih licenc.

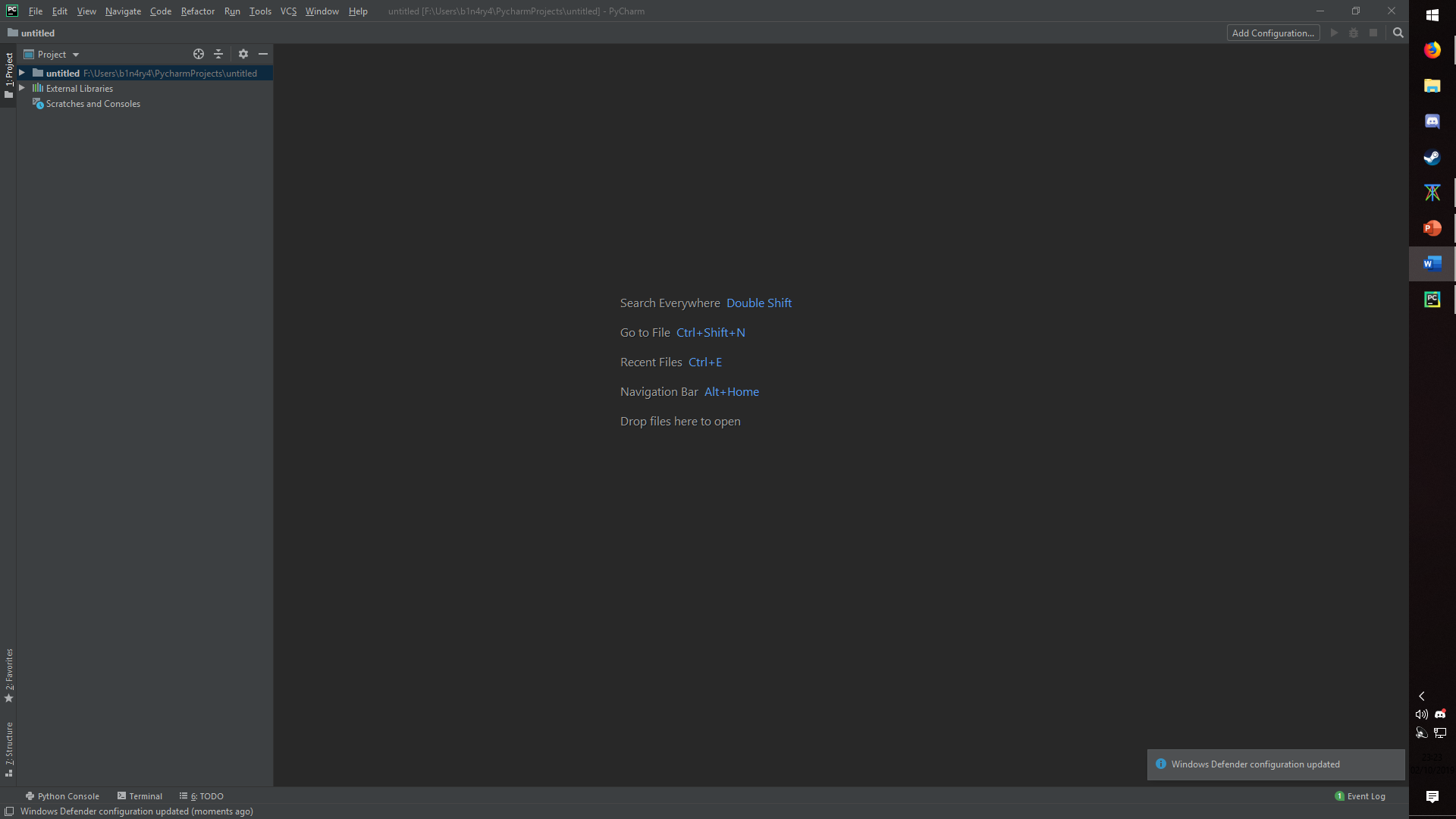
## Android Studio

Android studio je uradno integrirano razvojno okolje (IDE) podjedja Google za Android operacijski sistem in je tudi namenjeno Android razvoju. Namestitev je možna na različne operacijske sisteme (Windows, MacOS in Linux). Android studio je nadomestilo za Eclipse Android razvojno orodje, katero se je primarno uporabljalo prej za razvoj Android aplikacij. Kot predhodna verzija je izšla leta 2013, uradno pa je izšel leta 2014, kjer je tudi decembra 2014 dokačal stabilno verzijo 1.0. Primarno se za razvoj uporablja programski jezik Java, omogoča pa razvoj tudi v programskem jeziku C++ in drugih.

# Namestitev

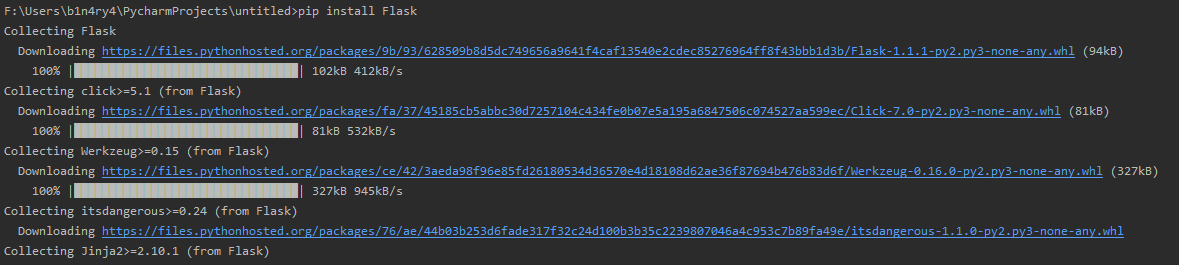
Ker bomo za svoj projekt uporabljali programski jezik python na operacijskem sistemu Microsoft Windows, moramo za začetek namestiti python. Python namestimo preko uradne spletne strani (https://www.python.org/). Prenesemo datoteko in jo namestimo z dvoklikom na računalnik.

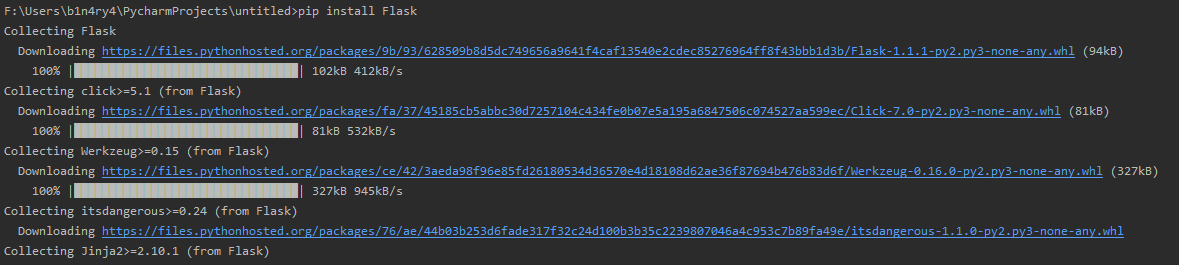
Nato iz uradne spletne strani prenesemo Python IDE PyCharm (https://www.jetbrains.com/pycharm/). Prenesemo datoteko (»download«) in jo z dvoklikom poženemo. Gremo skozi začetni čarovnik in dokončamo namestitev razvojnega okolja. Ko okolje uspešno namestimo ga odpremo in ustvarimo nov projekt.



Slika :prikaz razvojnega okolja PyCharm

Nato namestimo Flask, v spodnjem levem kotu odpremo terminal in namestimo Flask z ukazom: »pip install Flask« (po potrebi je potrebno nastaviti tudi spremeljivke okolja (F:\Python37-32\Scripts).

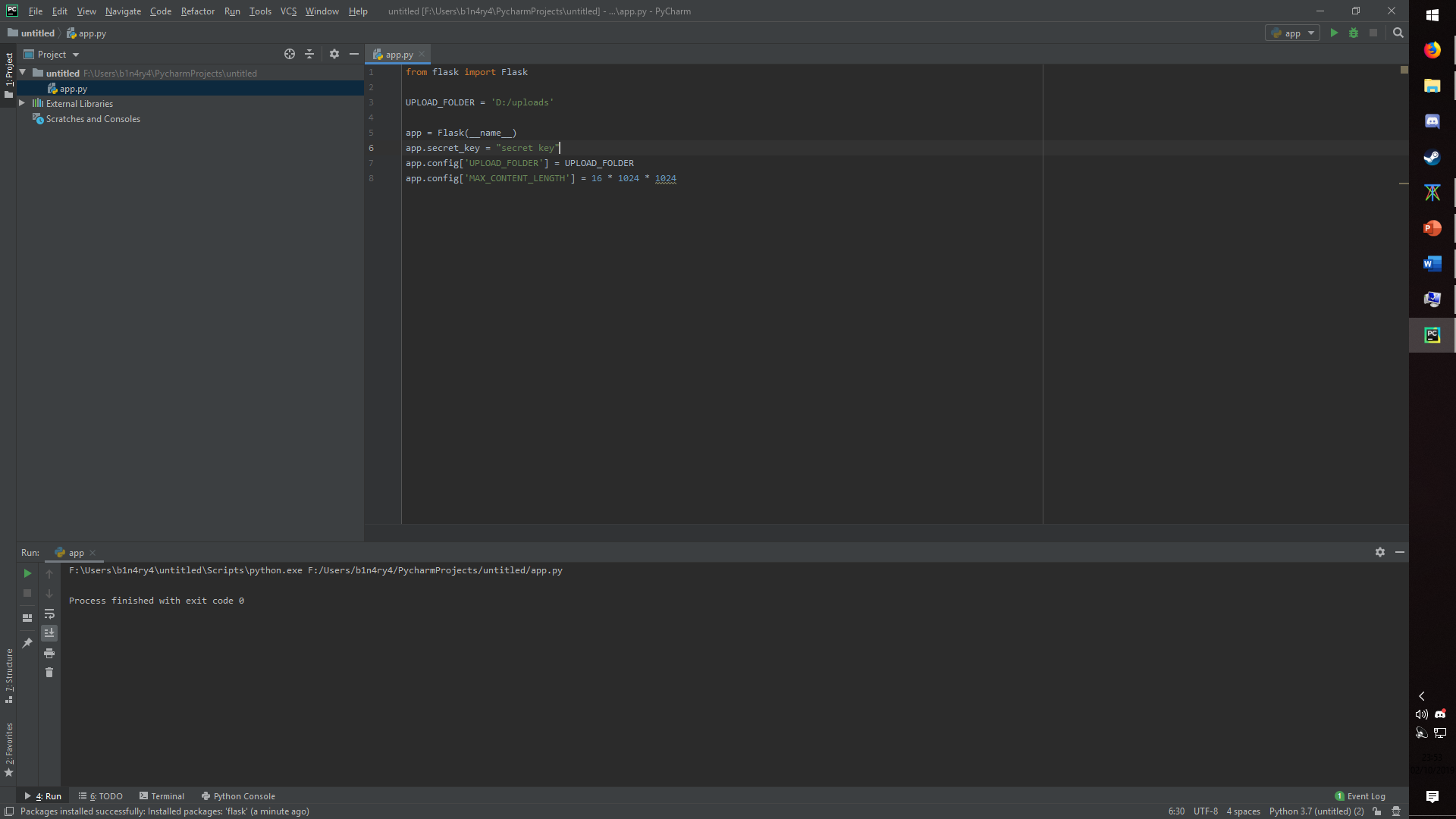




Slika :namestitev Flask

# Test

V svojem projektu ustvarimo nov python file (desni miškin klik na projekt, new, python file) in ga poljubno poimenujemo (v našem primeru app, datoteka avtomatsko pridobi končnico .py). V novo ustvarjeno datoteko napišemo poljuben odsek kode in ga poženemo. Na spodnji sliki je prikazan nov projekt, z datoteko app.py in odsekom kode, katerega smo uspešno pognali.

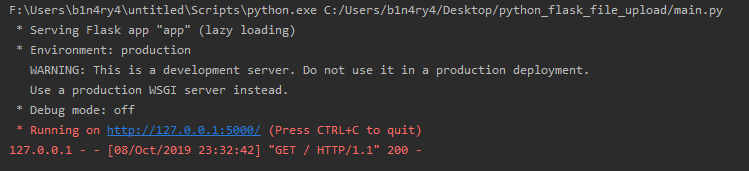


Slika :ustvarjen projekt z datoteko in uspešno prevedenim odsekom kode

# Izdvedba projekta PyCharm in Flask

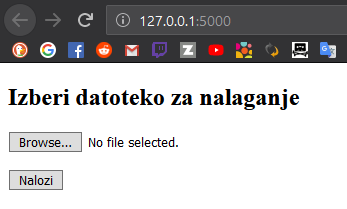
Najprej smo pri projektu naredili osnovno spletno stran, katero smo naredili v upload.html, ter osnovna opozorila v primeru da ne izberemo datoteke, izberemo napačno datoteko, izberemo pravilno datoteko,…. Naš cilj je bil, da preko naše preproste ustvarjene spletne strani naložimo v mapo našega strežnika.

Na spodnjih slikah je prikazano delovanje strežnika:



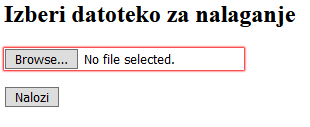
Slika :prikaz aktivnega strežnika

## Nalaganje slike v mapo strežnika



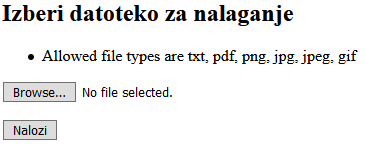
Slika :prikaz izgleda naše spletne strani

V primeru da datoteke ne izberemo:



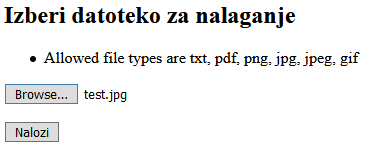
Slika :če datoteka ni izbrana

V primeru da izberemo napačno datoteko:



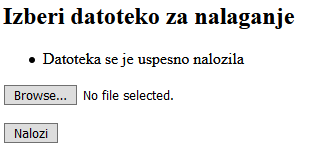
Slika :če izberemo napačno datoteko

V primeru da izberemo pravilno datoteko:



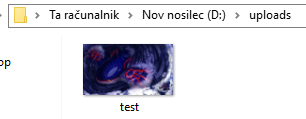
Slika :če izberemo pravilno datoteko

Potem lahko datoteko uspešno naložimo:



Slika :uspešno nalaganje datoteke

Katero nato tudi najdemo v naši mapi strežnika:



Slika :prikaz datoteke v strežniški mapi

## Koda ki smo jo uporabili za nalaganje slike na strežnik

app.py

from flask import Flask #import flask class  
  
UPLOAD\_FOLDER = 'D:/uploads' #upload path  
  
app = Flask(\_\_name\_\_) #\_\_name\_\_ ker gre za 1 modul, za več modulov \_\_main\_\_  
app.secret\_key = "secret key" #za varnost odjemalca, brez varnosti flask ne omogoča dostopa  
app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER #določimo path za upload  
app.config['MAX\_CONTENT\_LENGTH'] = 16 \* 1024 \* 1024 # max 16mb in velikost

main.py

import os #os library

from app import app #app object

from flask import Flask, flash, request, redirect, render\_template #app flask

from werkzeug.utils import secure\_filename

ALLOWED\_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif']) #dovoljene končnice

def allowed\_file(filename): #preveri če je končnica pravilna

return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

@app.route('/')

def upload\_form():

return render\_template('upload.html')

@app.route('/', methods=['POST'])

def upload\_file():

if request.method == 'POST': #če ima post zahteva datoteko

if 'file' not in request.files:

flash('No file part') #izpiše če ni

return redirect(request.url)

file = request.files['file']

if file.filename == '': #če ni izbrane datoteke za nalaganje

flash('No file selected for uploading')

return redirect(request.url)

if file and allowed\_file(file.filename): #če je datoteka izbrana in ustreza

filename = secure\_filename(file.filename) #pretvori datoteko

file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename)) #shranjevanje datoteke

flash('Datoteka se je uspesno nalozila') #feedback

return redirect('/')

else:

flash('Allowed file types are txt, pdf, png, jpg, jpeg, gif')

return redirect(request.url)

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

upload.html

<!doctype html> <!–– deklaracija dtd ––>

<title>Uploader</title> <!–– naslov dokumenta, se prikaže v zavihku ––>

<h2>Izberi datoteko za nalaganje</h2> <!–– header, string, ime ––>

<p> <!–– parahraph ––>

{% with messages = get\_flashed\_messages() %} {# MAGIC funkcija #}

{% if messages %} {# funkcija #}

<ul class=flashes>

{% for message in messages %}

<li>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

{% endwith %} {# MAGIC end #}

</p> <!–– parahraph end ––>

<form method="post" action="/" enctype="multipart/form-data"> <!–– post-request http, enctype-enkripcija za post ––>

<dl> <!–– description list ––>

<p> <!–– parahraph ––>

<input type="file" name="file" autocomplete="off" required> <!–– omogoča uporabniku izbiro datoteke (browse) \*multiple\* za več datotek ––>

</p>

</dl>

<p> <!–– parahraph ––>

<input type="submit" value="Nalozi"> <!–– submit btn ––>

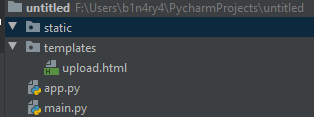
</p>

</form>

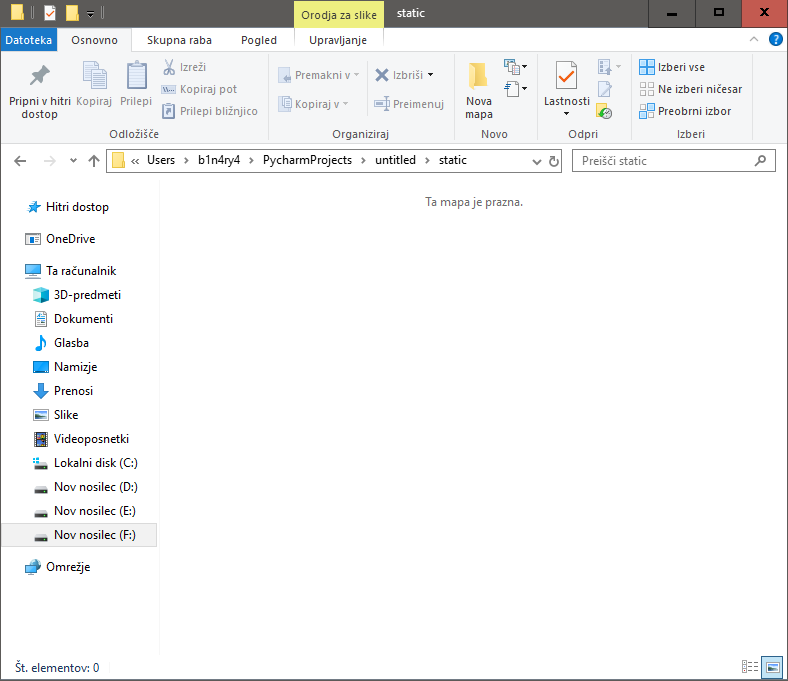
## Prikaz slike na strežniku

V nadaljevanju smo želeli, da se slika ki smo jo naložili, tudi prikaže na naši spletni strani. Skušali smo narediti, da ko pošljemo sliko v mapo strežnika (s pritiskom na gumb »submit«) se ta tudi prikaže na zaslonu spletne strani.

Ustvarili smo nov direktorij in spremenili »path«, ko pritisnemo gumb submit, ta shrani sliko v novo kreirano mapo v direktoriju imenovano static.

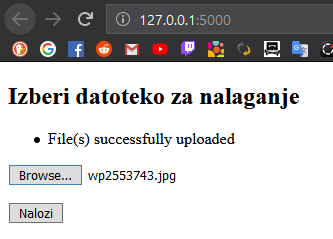


Slika :prikaz prazne mape pred zagonom programa 1



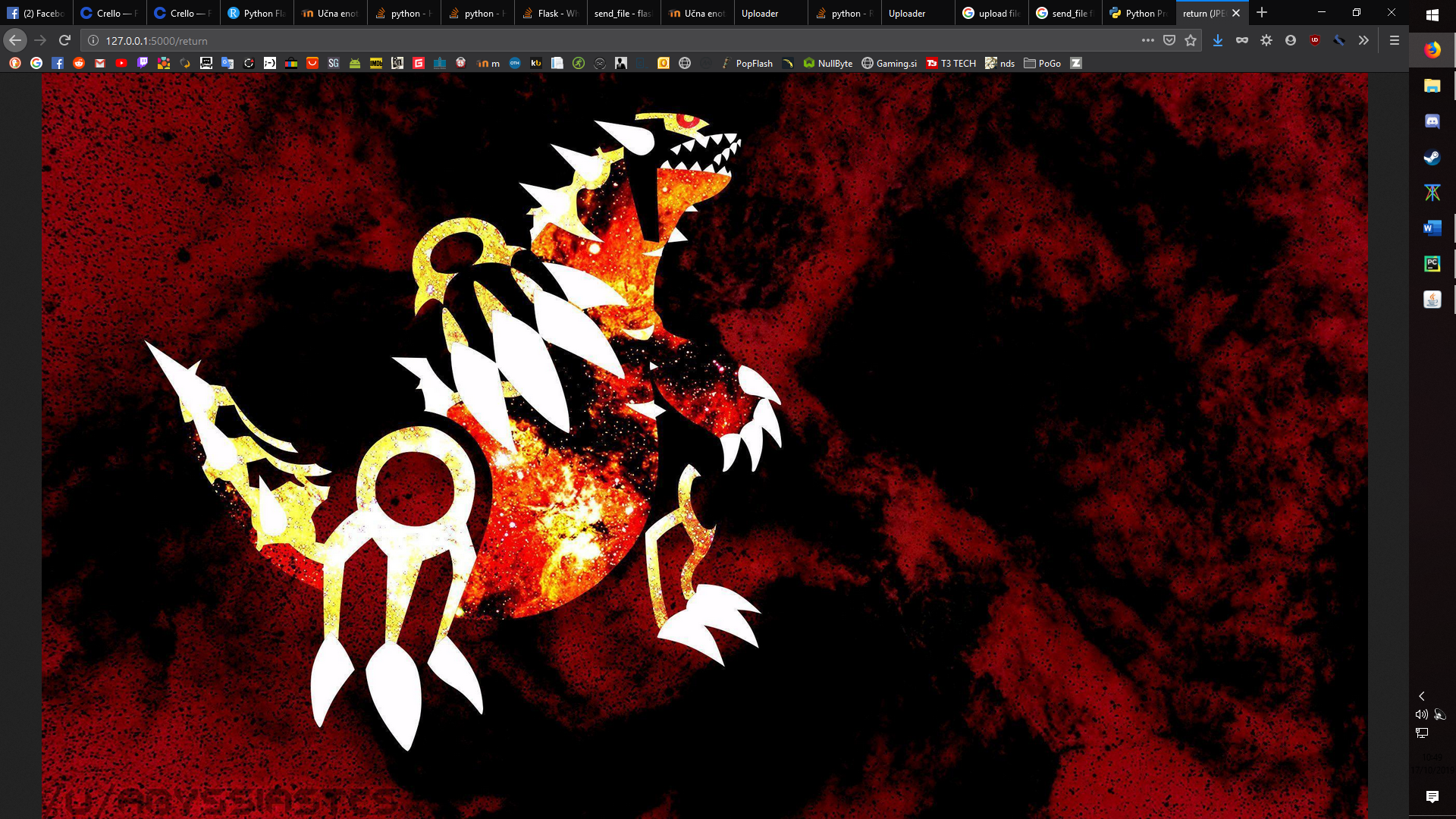
Slika :prikaz prazne mape pred zagonom programa 2

Med zagonom programa izberemo sliko in pritisnemo na gumb »submit«



Slika :uspešno uploadana slika na strežnik

Slika se nam po pritisku na »submit« naloži in na spletni strani se prikaže slika:



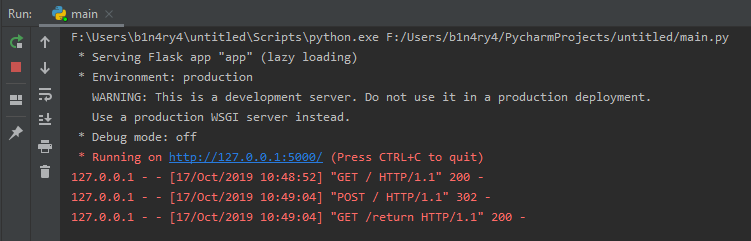
Slika :prikaz slike na spletni strani

Slika se nam tudi pojavi v naši strežniški mapi:



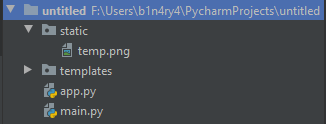
Slika :prikaz slike v strežniški mapi

V terminalu vidimo da se je slika uspešno poslala:



Slika ;uspešno pošiljanje slike

Poslano datoteko (sliko) pa lahko vidimo tudi v mapah našega projekta:



Slika :prikaz naložene slike

## Koda ki smo jo uporabili za prikaz slike na strežniku

V kodi so označeni odseki ki so bili dodani/spremenjeni z rdečim komentarjem.

app.py

from flask import Flask #import flask class  
  
UPLOAD\_FOLDER = 'F:/Users/b1n4ry4/PycharmProjects/untitled/static' #upload path \*spremenjena iz prejšnje oddaje zaradi lažjega nadaljevanja projekta\*  
  
app = Flask(\_\_name\_\_) #\_\_name\_\_ ker gre za 1 modul, za več modulov \_\_main\_\_  
app.secret\_key = "secret key" #za varnost odjemalca, brez varnosti flask ne omogoča dostopa  
app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER #določimo path za upload  
app.config['MAX\_CONTENT\_LENGTH'] = 16 \* 1024 \* 1024 # max 16mb

main.py

import os # os library  
from app import app # app object  
from flask import Flask, flash, request, redirect, render\_template, \  
 send\_file # app flask \*dodan send\_file, omogoča pošiljanje/vračanje slik\*  
from werkzeug.utils import secure\_filename  
  
ALLOWED\_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif']) # dovoljene končnice  
  
  
def allowed\_file(filename): # preveri če je končnica pravilna  
 return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS  
  
  
@app.route('/')  
def upload\_form():  
 return render\_template('upload.html')  
  
  
@app.route('/', methods=['POST'])  
def upload\_file():  
 if request.method == 'POST': # če ima post zahteva datoteko  
 if 'file' not in request.files:  
 flash('No file part') # izpiše če ni  
 return redirect(request.url)  
 file = request.files['file']  
 if file.filename == '': # če ni izbrane datoteke za nalaganje  
 flash('No file selected for uploading')  
 return redirect(request.url)  
 if file and allowed\_file(file.filename): # če je datoteka izbrana in ustreza  
 file.filename = "temp.png" # shranimo sliko pod določeno ime (spremenimo ime), da vračamo vedno sliko z istim imenom  
 filename = secure\_filename(file.filename) # pretvori datoteko  
 file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename)) # shranjevanje datoteke  
 flash('Datoteka se je uspesno nalozila') # feedback  
 return redirect('/return') #ob pritisku na gumb submit preusmeri na localhost/return-kjer je slika  
 else:  
 flash('Allowed file types are txt, pdf, png, jpg, jpeg, gif')  
 return redirect('/')   
  
@app.route('/return') # v zavihku return  
def retutn\_file(): # vrne datoteko oz. pošlje  
 return send\_file('F:/Users/b1n4ry4/PycharmProjects/untitled/static/temp.png') # v tem direktoriju  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run()

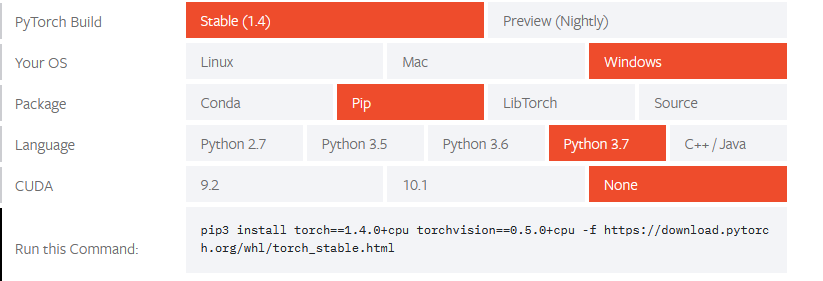
upload.html

\*upload.html je ostal nespremenjen\*

<!doctype html> <!–– deklaracija dtd ––>  
<title>Uploader</title> <!–– naslov dokumenta, se prikaže v zavihku ––>  
<h2>Izberi datoteko za nalaganje</h2> <!–– header, string, ime ––>  
<p> <!–– parahraph ––>  
 {% with messages = get\_flashed\_messages() %} {# MAGIC funkcija #}  
 {% if messages %} {# funkcija #}  
 <ul class=flashes>  
 {% for message in messages %}  
 <li>{{ message }}</li>  
 {% endfor %}  
 </ul>  
 {% endif %}  
 {% endwith %} {# MAGIC end #}  
</p> <!–– parahraph end ––>  
<form method="post" action="/" enctype="multipart/form-data"> <!–– post-request http, enctype-enkripcija za post ––>  
 <dl> <!–– description list ––>  
 <p> <!–– parahraph ––>  
 <input type="file" name="file" autocomplete="off" required> <!–– omogoča uporabniku izbiro datoteke (browse) \*multiple\* za več datotek ––>  
 </p>  
 </dl>  
 <p> <!–– parahraph ––>  
 <input type="submit" value="Nalozi"> <!–– submit btn ––>  
 </p>  
</form>

# Končna izvedba strežnika

Pri končni izvedbi strežnika smo uporabili pytorch. Po navodilih smo namestili pytorch iz njihove uradne spletne strani: <https://pytorch.org/>. Na spletni strani izberemo »*get started«* in nato izberemo željene parametre (*build, OS, package, verzijo python, CUDA*). V spodnji vrstici se nam izpiše ukaz, katerega poženemo v razvojnem programu PyTorch.



Slika :prikaz namestitve PyTorch

Za zagon smo še morali prenesti in namestiti dve datoteki:

* <http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar>
* <http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar>

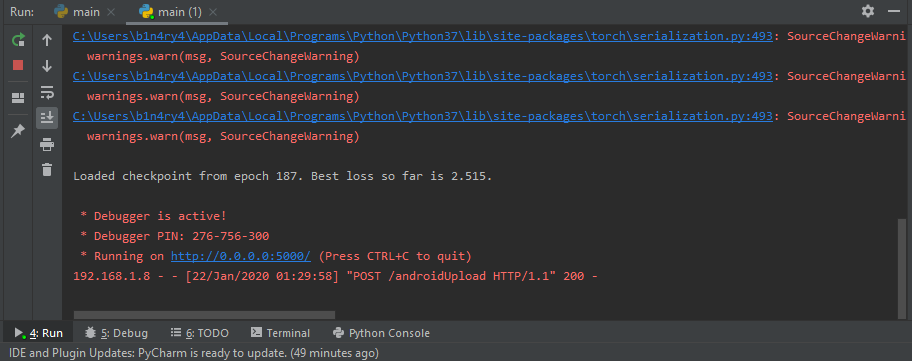
V omenjenih mapah najdemo vse potrebne datoteke

V projektu smo uporabili Pascal Visual Object Classes VOC, kateri vsebuje 20 različnih podob:

* Letalo
* Kolo
* Ptica
* Ladja
* Plastenka
* Avtobus
* Avto
* Mačka
* Stol
* Krava
* Jedilna miza
* Pes
* Konj
* Motorno kolo
* Oseba
* Ovca
* Kavč
* Vlak
* Monitor
* Lončnica



Slika :pregled končnega strežnika zaznave objektov



Slika :prikaz aktivnega strežnika

## Koda končnega strežnika

main.py

import os # os library  
  
import torch  
from pip.\_vendor.chardet import detect  
  
from app import app # app object  
from flask import Flask, flash, request, redirect, render\_template, send\_from\_directory # app flask  
from werkzeug.utils import secure\_filename  
import base64  
from flask import send\_file  
from torchvision import transforms  
from utils import \*  
from PIL import Image, ImageDraw, ImageFont  
  
device = torch.device("cuda" if torch.cuda.is\_available() else 'cpu')  
  
checkpoint = './BEST\_checkpoint\_ssd300.pth.tar'  
checkpoint = torch.load(checkpoint)  
start\_epoch = checkpoint['epoch'] + 1  
best\_loss = checkpoint['best\_loss']  
print('\nLoaded checkpoint from epoch %d. Best loss so far is %.3f.\n' % (start\_epoch, best\_loss))  
model = checkpoint['model']  
model = model.to(device)  
model.eval()  
  
resize = transforms.Resize((300, 300))  
to\_tensor = transforms.ToTensor()  
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],  
 std=[0.229, 0.224, 0.225])  
  
ALLOWED\_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif']) # dovoljene končnice  
PUBLIC\_DIR = app.config['UPLOAD\_FOLDER']  
  
  
def allowed\_file(filename): # preveri če je končnica pravilna  
 return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS  
  
  
@app.route('/')  
def upload\_form():  
 image\_names = [f for f in os.listdir(PUBLIC\_DIR) if  
 (f.endswith('png') or f.endswith('jpg') or f.endswith('jpeg') or f.endswith('gif'))]  
  
 print(image\_names)  
 return render\_template("upload.html", image\_names=image\_names)  
  
  
@app.route('/<filename>')  
def send\_image(filename):  
 return send\_from\_directory(PUBLIC\_DIR, filename)  
  
  
@app.route('/androidUpload', methods=['GET', 'POST'])  
def upload\_image():  
 encoded\_img = request.form['base64'] # 'base64' is the name of the parameter used to post image file  
 filename = request.form['ImageName'] # 'ImageName' is name of the parameter used to post image name  
 img\_data = base64.b64decode(encoded\_img) # decode base64 string back to image  
 image\_path\_name = os.path.join(app.config['UPLOAD\_FOLDER']) + "/" + filename  
 with open(image\_path\_name, 'wb') as f: # "w"rite and "b"inary = wb  
 f.write(img\_data)  
 original\_image = Image.open(image\_path\_name, mode='r')  
 original\_image = original\_image.convert('RGB')  
 detect(original\_image, image\_path\_name, min\_score=0.2, max\_overlap=0.5, top\_k=200)  
 return send\_file(image\_path\_name)  
  
  
@app.route('/', methods=['POST'])  
def upload\_file():  
 if request.method == 'POST': # če ima post zahteva datoteko  
 if 'file' not in request.files:  
 flash('No file part') # izpiše če ni  
 return redirect(request.url)  
 file = request.files['file']  
 if file.filename == '': # če ni izbrane datoteke za nalaganje  
 flash('No file selected for uploading')  
 return redirect(request.url)  
 if file and allowed\_file(file.filename): # če je datoteka izbrana in ustreza  
 filename = secure\_filename(file.filename) # pretvori datoteko  
 img\_path = os.path.join(app.config['UPLOAD\_FOLDER']) + "/" + filename  
 file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename)) # shranjevanje datoteke  
 flash('Datoteka se je uspesno nalozila') # feedback  
 original\_image = Image.open(img\_path, mode='r')  
 original\_image = original\_image.convert('RGB')  
 detect(original\_image, img\_path, min\_score=0.2, max\_overlap=0.5, top\_k=200)  
 return redirect('/response?file=' + filename)  
 else:  
 flash('Allowed file types are txt, pdf, png, jpg, jpeg, gif')  
 return redirect(request.url)  
  
  
@app.route('/response')  
def return\_file():  
 return send\_file(app.config['UPLOAD\_FOLDER'] + request.args.get('file'))  
  
  
def detect(original\_image, filename, min\_score, max\_overlap, top\_k, suppress=None):  
 *"""  
 Detect objects in an image with a trained SSD300, and visualize the results.* ***:param*** *original\_image: image, a PIL Image* ***:param*** *filename: name of image* ***:param*** *min\_score: minimum threshold for a detected box to be considered a match for a certain class* ***:param*** *max\_overlap: maximum overlap two boxes can have so that the one with the lower score is not suppressed via Non-Maximum Suppression (NMS)* ***:param*** *top\_k: if there are a lot of resulting detection across all classes, keep only the top 'k'* ***:param*** *suppress: classes that you know for sure cannot be in the image or you do not want in the image, a list* ***:return****: annotated image, a PIL Image  
 """* # Transform  
 image = normalize(to\_tensor(resize(original\_image)))  
  
 # Move to default device  
 image = image.to(device)  
  
 # Forward prop.  
 predicted\_locs, predicted\_scores = model(image.unsqueeze(0))  
  
 # Detect objects in SSD output  
 det\_boxes, det\_labels, det\_scores = model.detect\_objects(predicted\_locs, predicted\_scores, min\_score=min\_score,  
 max\_overlap=max\_overlap, top\_k=top\_k)  
  
 # Move detections to the CPU  
 det\_boxes = det\_boxes[0].to('cpu')  
  
 # Transform to original image dimensions  
 original\_dims = torch.FloatTensor(  
 [original\_image.width, original\_image.height, original\_image.width, original\_image.height]).unsqueeze(0)  
 det\_boxes = det\_boxes \* original\_dims  
  
 # Decode class integer labels  
 det\_labels = [rev\_label\_map[l] for l in det\_labels[0].to('cpu').tolist()]  
  
 # If no objects found, the detected labels will be set to ['0.'], i.e. ['background']  
 # in SSD300.detect\_objects() in model.py  
 if det\_labels == ['background']:  
 # Just return original image  
 return original\_image  
  
 # Annotate  
 annotated\_image = original\_image  
 draw = ImageDraw.Draw(annotated\_image)  
 font = ImageFont.truetype("./arial.ttf", 15)  
  
 # Suppress specific classes, if needed  
 for i in range(det\_boxes.size(0)):  
 if suppress is not None:  
 if det\_labels[i] in suppress:  
 continue  
  
 # Boxes  
 box\_location = det\_boxes[i].tolist()  
 draw.rectangle(xy=box\_location, outline=label\_color\_map[det\_labels[i]])  
 draw.rectangle(xy=[l + 1. for l in box\_location], outline=label\_color\_map[  
 det\_labels[i]]) # a second rectangle at an offset of 1 pixel to increase line thickness  
 # draw.rectangle(xy=[l + 2. for l in box\_location], outline=label\_color\_map[  
 # det\_labels[i]]) # a third rectangle at an offset of 1 pixel to increase line thickness  
 # draw.rectangle(xy=[l + 3. for l in box\_location], outline=label\_color\_map[  
 # det\_labels[i]]) # a fourth rectangle at an offset of 1 pixel to increase line thickness  
  
 # Text  
 text\_size = font.getsize(det\_labels[i].upper())  
 text\_location = [box\_location[0] + 2., box\_location[1] - text\_size[1]]  
 textbox\_location = [box\_location[0], box\_location[1] - text\_size[1], box\_location[0] + text\_size[0] + 4.,  
 box\_location[1]]  
 draw.rectangle(xy=textbox\_location, fill=label\_color\_map[det\_labels[i]])  
 draw.text(xy=text\_location, text=det\_labels[i].upper(), fill='white',  
 font=font)  
 del draw  
  
 annotated\_image.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename)) # Shranjevanje datoteke  
 # return redirect('/response?file=' + annotated\_image)  
 # return annotated\_image  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host="0.0.0.0", port=5000, debug=True)

upload.html

<!doctype html> <!–– deklaracija dtd ––>  
<title>Uploader</title> <!–– naslov dokumenta, se prikaže v zavihku ––>  
<h2>Izberi datoteko za nalaganje</h2> <!–– header, string, ime ––>  
<p> <!–– parahraph ––>  
 {% with messages = get\_flashed\_messages() %} {# MAGIC funkcija #}  
 {% if messages %} {# funkcija #}  
 <ul class=flashes>  
 {% for message in messages %}  
 <li>{{ message }}</li>  
 {% endfor %}  
 </ul>  
 {% endif %}  
 {% endwith %} {# MAGIC end #}  
</p> <!–– parahraph end ––>  
<form method="post" action="/" enctype="multipart/form-data"> <!–– post-request http, enctype-enkripcija za post ––>  
 <dl> <!–– description list ––>  
 <p> <!–– parahraph ––>  
 <input type="file" name="file" autocomplete="off" required> <!–– omogoča uporabniku izbiro datoteke (browse) \*multiple\* za več datotek ––>  
 </p>  
 </dl>  
 <p> <!–– parahraph ––>  
 <input type="submit" value="Nalozi"> <!–– submit btn ––>  
 </p>  
</form>

app.py

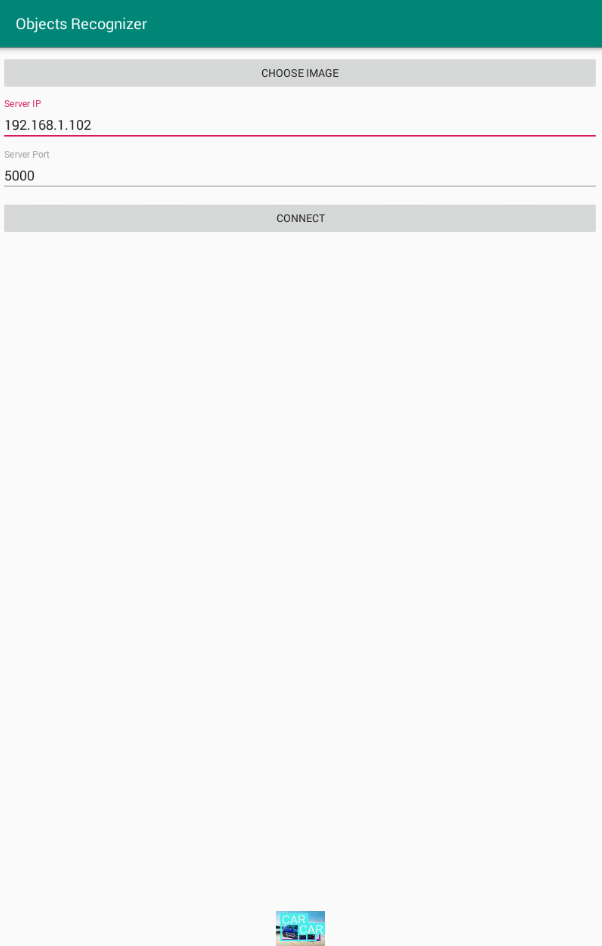
from flask import Flask #import flask class  
  
UPLOAD\_FOLDER = 'C:/Users/b1n4ry4/Desktop/merged/static' #upload path  
  
app = Flask(\_\_name\_\_) #\_\_name\_\_ ker gre za 1 modul, za več modulov \_\_main\_\_  
app.secret\_key = "secret key" #za varnost odjemalca, brez varnosti flask ne omogoča dostopa  
app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER #določimo path za upload  
app.config['MAX\_CONTENT\_LENGTH'] = 16 \* 1024 \* 1024 # max 16mb

# Android aplikacija

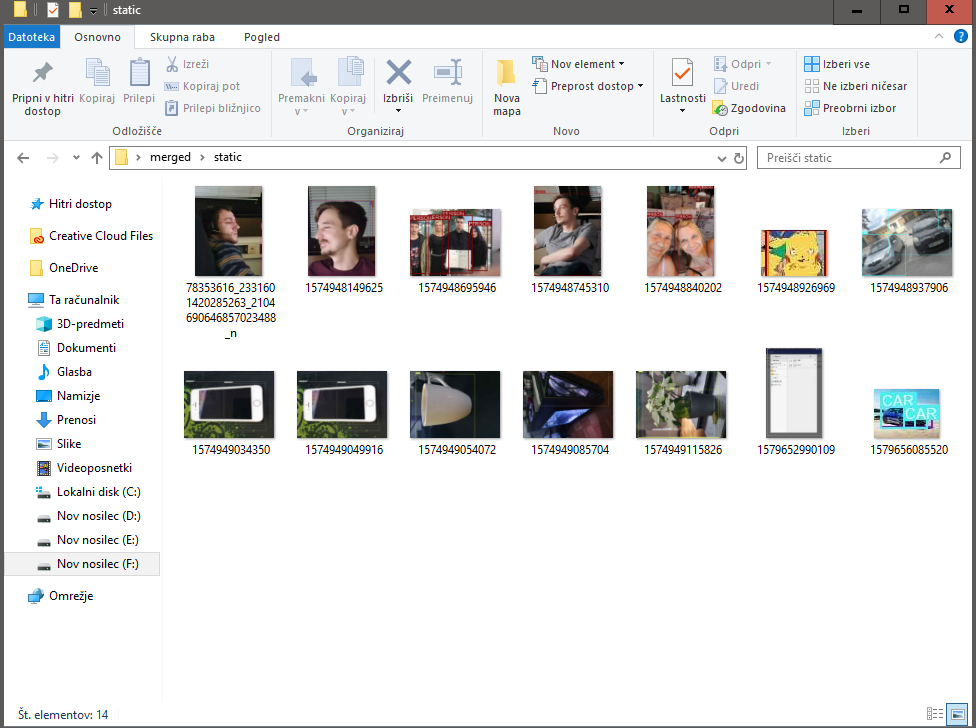
Uporabnik mora ob prvem zagonu aplikacije sprejeti pravice za dostop do galerije. Nato se uporabniku prikaže osnovni zaslon aplikacije:



Slika :osnovni prikaz aplikacije



Slika :prikaz vrnjene slike s strežnika



Slika :prikaz slik na strežniku

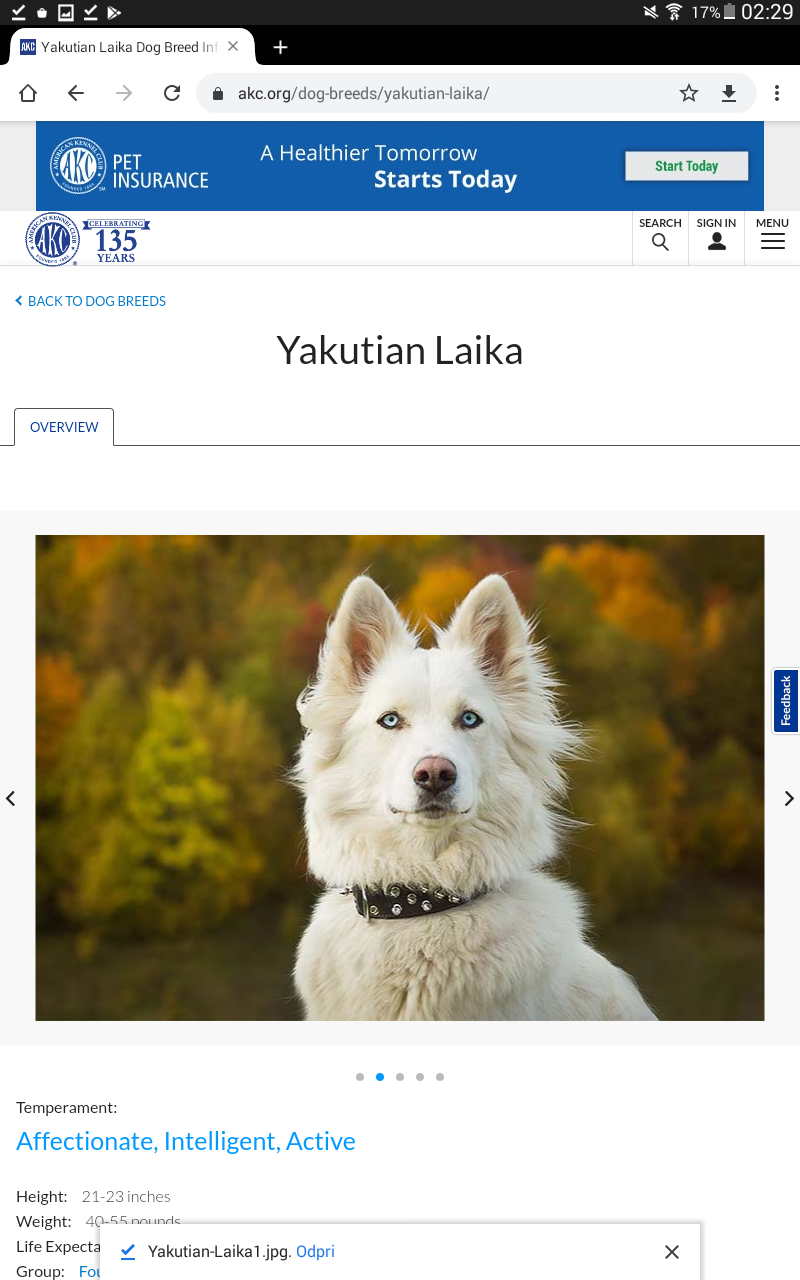
## Koda Android aplikacije

BitmapFactory.Options options = new BitmapFactory.Options(); //sliko pretvorimo v *byte za pošiljanje na strežnik*  
options.inSampleSize = 4;  
options.inPurgeable = true;  
Bitmap bitmap = BitmapFactory.*decodeFile*(selectedImagePath,options);  
ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();  
bitmap.compress(Bitmap.CompressFormat.*JPEG*,40,byteArrayOutputStream);  
byte[] byteImage\_photo = byteArrayOutputStream.toByteArray();  
String encodedImage =Base64.*encodeToString*(byteImage\_photo,Base64.*DEFAULT*);

protected Void doInBackground(Void... voids) { //pošiljanje slike na strežnik  
 try {  
 ArrayList<NameValuePair> nameValuePairs = new ArrayList<>();  
 nameValuePairs.add(new BasicNameValuePair("base64", stringBytes));  
 nameValuePairs.add(new BasicNameValuePair("ImageName", System.*currentTimeMillis*() + ".jpg"));  
 try {  
 HttpClient httpclient = new DefaultHttpClient();  
 HttpPost httppost = new HttpPost("http://" + serverIp + ":" + portNumber + "/androidUpload");  
 httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));  
 HttpResponse response = httpclient.execute(httppost);  
   
 if (response.getStatusLine().getStatusCode() == 200) {  
 HttpEntity entity = response.getEntity();  
 if (entity != null) {  
 InputStream inputStream = entity.getContent();  
 ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();  
 int bufferSize = 1024;  
 byte[] buffer = new byte[bufferSize];  
 int len;  
 try {  
 while ((len = inputStream.read(buffer)) != -1) {  
 byteArrayOutputStream.write(buffer, 0, len);  
 }  
 byteArrayOutputStream.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 byte[] b = byteArrayOutputStream.toByteArray();  
 bitmap = BitmapFactory.*decodeByteArray*(b, 0, b.length);  
 }  
 }  
 } catch (Exception e) {  
 Log.*e*("ERROR", "Error in http connection " + e.toString());  
 }  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 return null;  
}

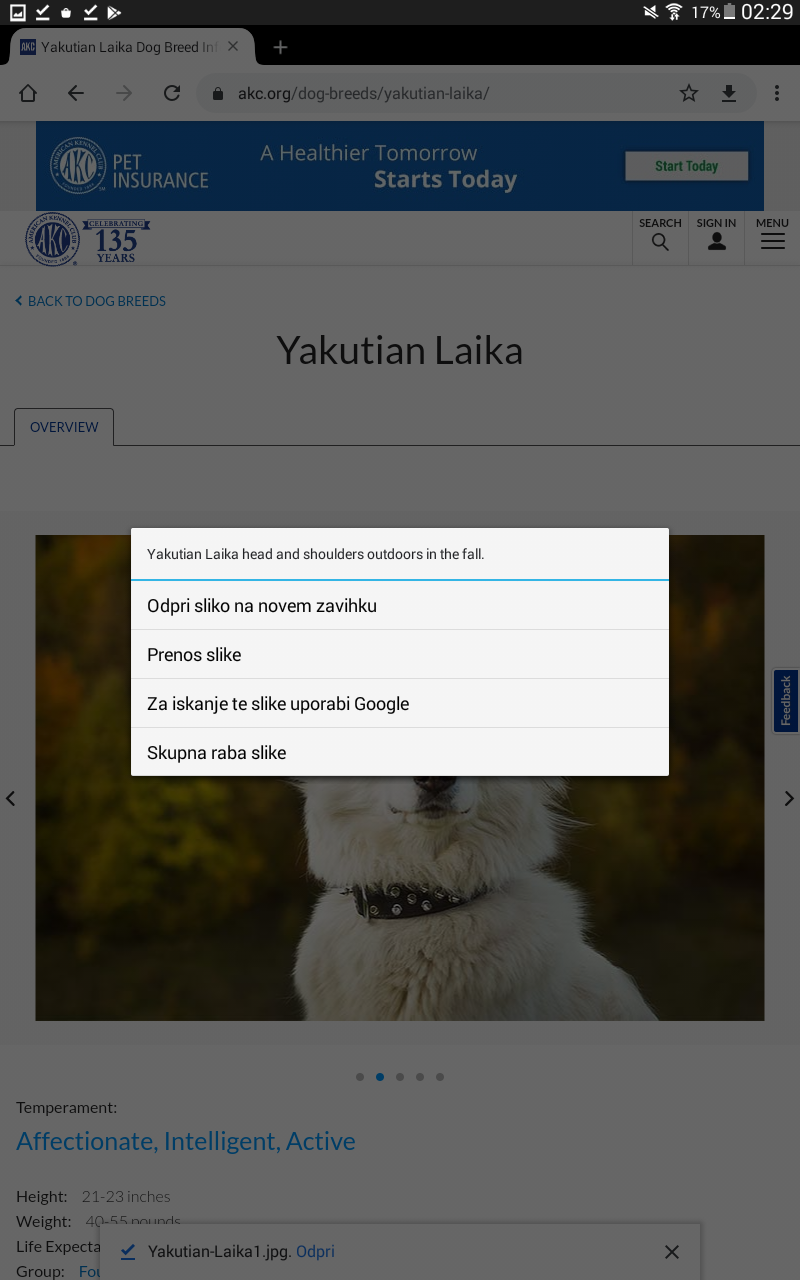
# Navodila za uporabo

Če še željene slike nimamo na mobilnem telefonu oz. v galeriji, jo poiščemo na spletu:



Slika :iskanje slike na spletu

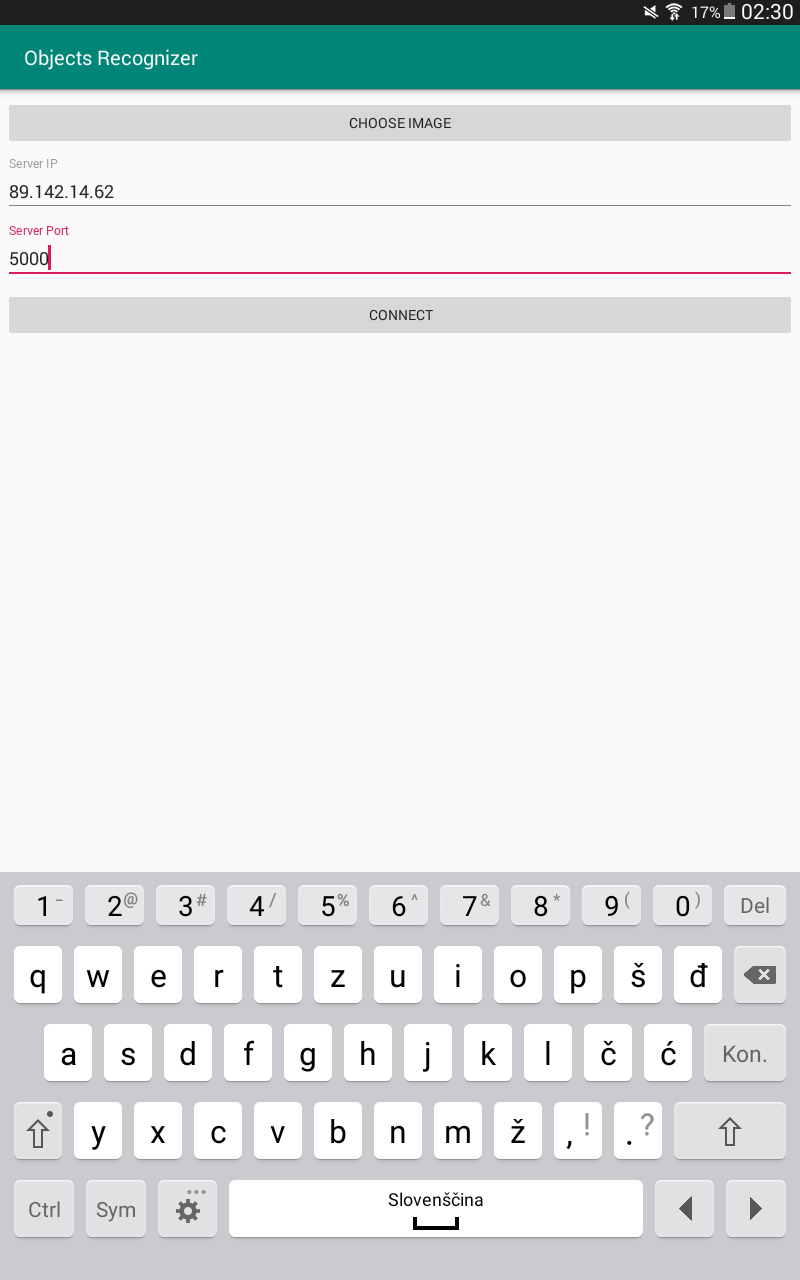
Sliko nato prenesemo na svoj telefon:



Slika :prenos slike na telefon

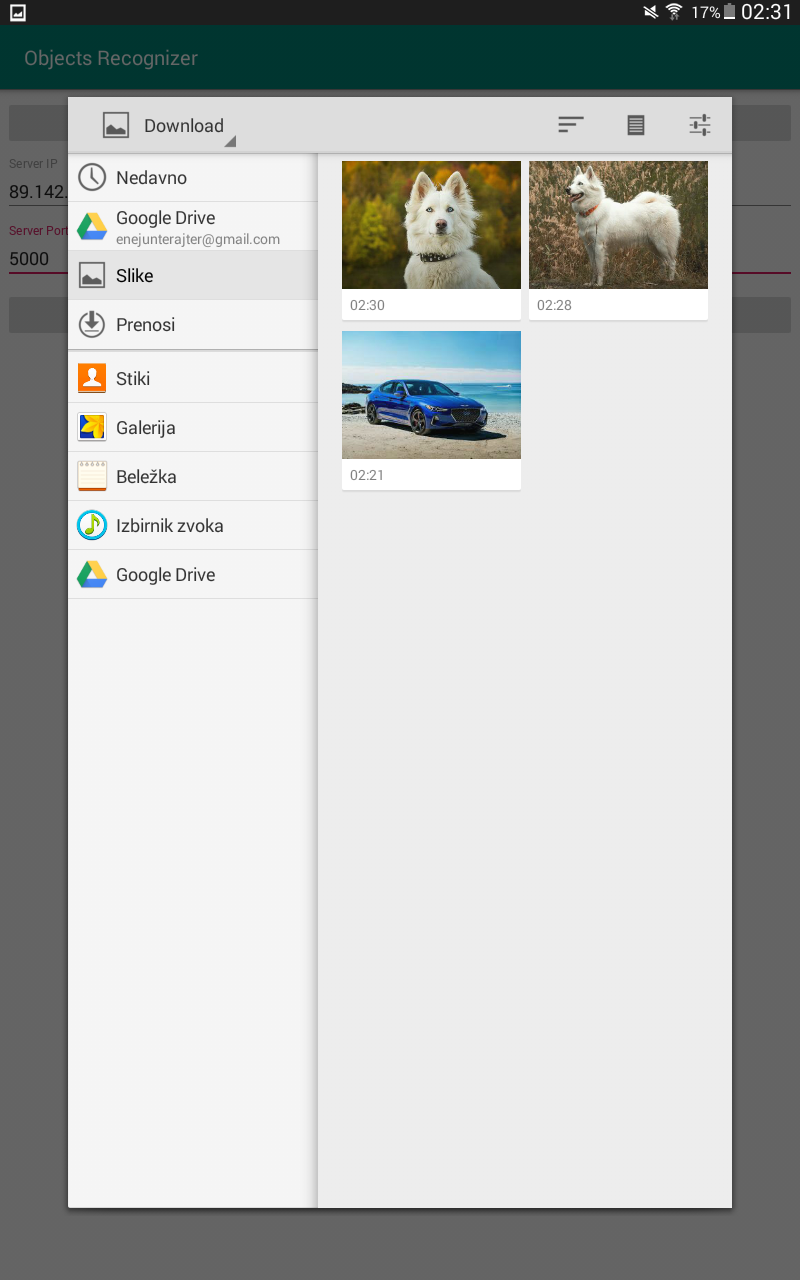
Nato odpremo Android aplikacijo, katero namestimo na svoj telefon. Na telefon lahko prenesemo .apk, v nastavitvah telefona moramo odobriti »unknown sources« in aplikacijo namestimo. Drug način je, da celoten projekt namestimo v Android studio, nato telefon preko USB kabla priključimo na računalnik, v nastavitvah mobilnega telefona v možnosti za razvijalce omogočimo »USB debbuging« in nato projekt v Android Studio poženemo, aplikacija se nam tako namesti na mobilni telefon, kjer jo lahko poženemo.

Ob prvem zagonu dovolimi aplikaciji dostop do galerije in vnesemo IP našega strežnika ki je vedno enak in port 5000.



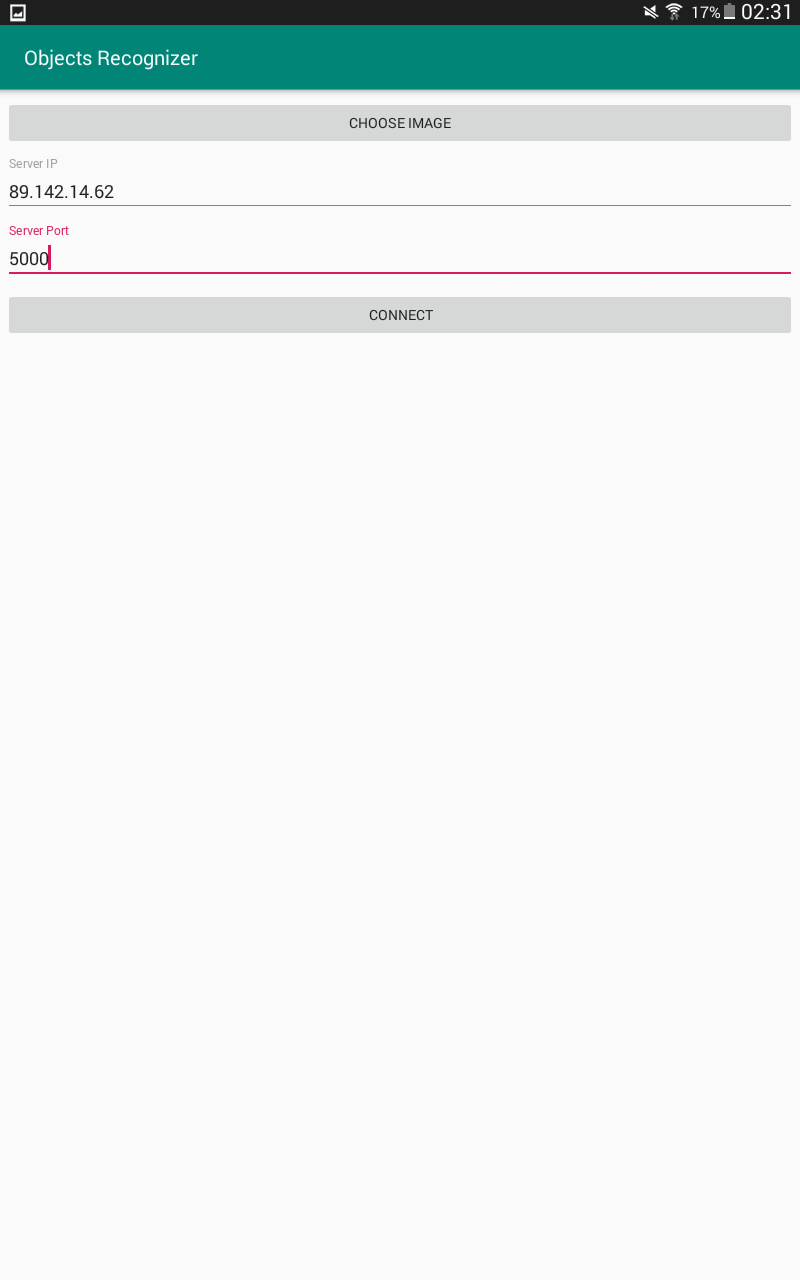
Slika :vnos IP naslova strežnika in vrat

Z gumbom »choose image« nato izberemo željeno sliko:



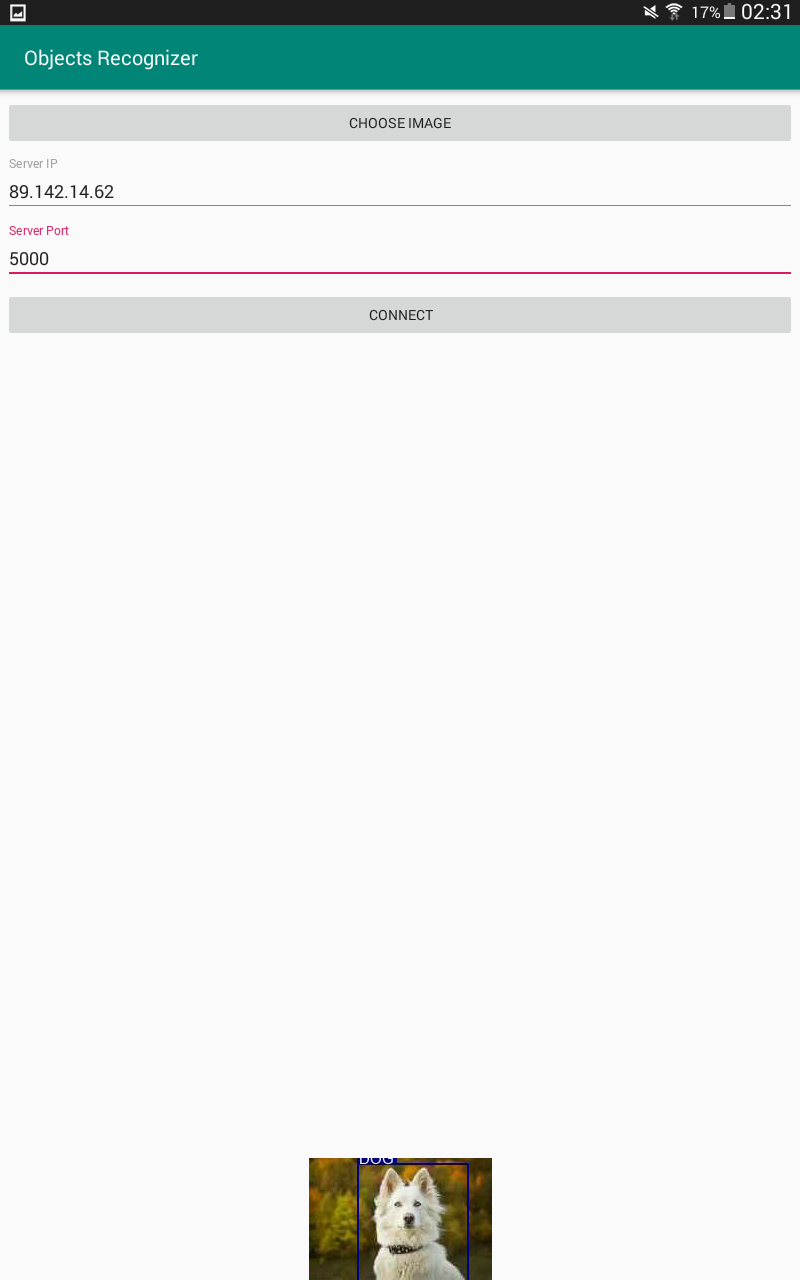
Slika :izbira željene slike

Po izbiri slike, se aplikacija vrne na začetni zaslon, kjer pritisnemu gumb »connect«.



Slika :izbira gumba connect

Nato malo počakamo in v aplikaciji se nam prikaže željeni rezultat:



Nalalagnje na virtualko (deployment)

Za naložitev strežnika na virtualko smo se najprej povezali na njo z ukazom ssh [username@164.8.x.](mailto:username@164.8.x.)x. Nato smo vnesli geslo \*\*\*\*\*\*\*. Najprej smo preverili ali je že nameščen python, pognali smo ukaz python –version kjer se nam je izpisala trenutna nameščena verzija pythona, tako smo ugotovili da je python že nameščen. Delo smo nadaljevali z namestitvijo ogrodja Flask z ukazom pip3 install flask ter preprostega besedilnega urejevalnika z ukazom sudo apt-get install vim, kjer smo še v mapi etc/vsftpd.com dodali vrstico write\_enable=YES. Service smo nato ponovno zagnali, na strežnik smo se še povezali preko programa FileZilla, ki smo ga prenesli iz njihove uradne spletne strani: <https://filezilla-project.org/>. Ko smo program odprli, smo vnesli IP virtualke, uporabniško ime: username in geslo: \*\*\*\*\*\*\*. Celoten projekt smo nato prenesli v mape virtualke. Nato smo še namestili PyTorch s pomočjo ukaza conda. Ker smo ob testu neprestano dobivali napako permission denied, smo še morali na strežnik vpisati ukaz sudo chmod -R 777 /path/to/directory.

Ko se povežemo na virtualko, virtualko zaženemo z ukazoma:

* cd merged
* python3.7 main.py

# Primeri

Na spodnjih slikah še najdemo nekaj primerov delovanja:







