**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Научный руководитель,  Доцент департамента больших данных и информационного поиска, заведующая лабораторией «Научно-учебная лаборатория биоинформатики» факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. С. Попцова  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия», канд. техн. наук, профессор департамента программной инженерии факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **ПРОГРАММА ДЛЯ ИССЛЕДОВАНИЯ ПРИМЕНИМОСТИ МЕТОДОВ КЛАСТЕРИЗАЦИИ И СНИЖЕНИЯ РАЗМЕРНОСТИ ДЛЯ АВТОМАТИЧЕСКОГО СРАВНЕНИЯ ЭКСПЕРИМЕНТОВ ОДНОКЛЕТОЧНОГО СЕКВЕНИРОВАНИЯ**  **СЕРВЕР**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.13-01 12 01-1-ЛУ** | | |
|  |  | |
| Исполнитель  Студент группы БПИ 182  \_\_\_\_\_\_\_\_\_\_\_/И. С. Егоров. /  «\_\_\_»\_\_\_\_\_\_\_\_\_\_2020 г. | |
|  | | |
|  | |  |

**Москва 2020**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.13-01 12 01-1-ЛУ |  | |  | |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **ПРОГРАММА ДЛЯ ИССЛЕДОВАНИЯ ПРИМЕНИМОСТИ МЕТОДОВ КЛАСТЕРИЗАЦИИ И СНИЖЕНИЯ РАЗМЕРНОСТИ ДЛЯ АВТОМАТИЧЕСКОГО СРАВНЕНИЯ ЭКСПЕРИМЕНТОВ ОДНОКЛЕТОЧНОГО СЕКВЕНИРОВАНИЯ**  **СЕРВЕР**  **Текст программы**  **RU.17701729.04.13-01 12 01-1**  **Листов 19** | | | | |
|  | |  | | |
|  | | |
|  | | | | |
|  | | | |  |

**Москва 2020**

**Содержание**

[**1. Текст программы** 4](#_Toc40016094)

[**1.1** **10x\_mtx\_cutter.py** 4](#_Toc40016095)

[**1.2** **database\_prepare.py** 5](#_Toc40016096)

[**1.3** **lda\_theme\_proportion.py** 8](#_Toc40016097)

[**1.4** **learn\_new\_model.py** 9](#_Toc40016098)

[**1.5** **bio/\_\_init\_\_.py** 10](#_Toc40016099)

[**1.6** **bio/admin.py** 10](#_Toc40016100)

[**1.7** **bio/apps.py** 10](#_Toc40016101)

[**1.8** **bio/models.py** 11](#_Toc40016102)

[**1.9** **bio/test.py** 11](#_Toc40016103)

[**1.10** **bio/urls/py** 11](#_Toc40016104)

[**1.11** **bio/views.py** 11](#_Toc40016105)

[**1.12** **bio\_settings/setting/\_\_init\_\_.py** 14](#_Toc40016106)

[**1.13** **bio\_settings/asgi.py** 14](#_Toc40016107)

[**1.14** **bio\_settings/setting.py** 15](#_Toc40016108)

[**1.15** **bio\_settings/urls.py** 18](#_Toc40016109)

[**1.16** **bio\_settings/wsgi.py** 18](#_Toc40016110)

# **Текст программы**

Текст программы доступен к просмотру по ссылке: <https://github.com/Igor-SeVeR/ReductionMethodsForComparingUnicellularSequencingExperiments-Coursework2/tree/master/djagnoBioInformaticsProject>

## **10x\_mtx\_cutter.py**

import sys

import random

#Uncomment when working on prod

path\_to\_files = sys.argv[1]

path\_to\_data = sys.argv[2]

file = open(path\_to\_data + "/matrix.mtx")

input\_str = file.readline()

input\_str = file.readline()

arr = input\_str.split(" ")

file.close()

number\_of\_barcodes = int(arr[1])

number\_of\_genes = int(arr[0])

#Working with barcodes

print("Starting barcode reducing!")

barcodes = open(path\_to\_files + "/barcodes.tsv", "r")

number\_of\_barcodes = len(barcodes.readlines())

barcodes.close()

#Working with genes

genes = open(path\_to\_files + "/genes.tsv", "r")

genes\_data = genes.readlines()

number\_of\_genes\_in\_file = len(genes\_data) - 1

genes.close()

genes = open(path\_to\_files + "/genes.tsv", "w")

if (number\_of\_genes\_in\_file >= number\_of\_genes):

genes.writelines(genes\_data[1:number\_of\_genes + 1])

else:

genes.writelines(genes\_data[1:number\_of\_genes\_in\_file + 1])

for i in range (number\_of\_genes - number\_of\_genes\_in\_file):

genes.write("ENSG00001111159 AAAAAA" + '\n')

genes.close()

#Reducing\_the\_matrix

def cut\_line(line):

numbers = line.split()

first\_number = int(numbers[0])

second\_number = int(numbers[1])

if (first\_number > number\_of\_genes or second\_number > number\_of\_barcodes):

return False

return True

matrix = open(path\_to\_files + "/matrix.mtx", "r")

data1 = matrix.readline()

data2 = matrix.readline()

cnt = 0

data = []

next\_line = matrix.readline()

while (next\_line):

if (cut\_line(next\_line)):

cnt += 1

data.append(next\_line)

next\_line = matrix.readline()

matrix.close()

matrix = open(path\_to\_files + "/matrix.mtx", "w")

matrix.writelines(data1)

matrix.writelines(str(number\_of\_genes) + ' ' + str(number\_of\_barcodes) + ' ' + str(cnt) + '\n')

matrix.writelines(data)

matrix.close()

## **database\_prepare.py**

import sys

# Library to save LDA model

import joblib

# Library to read files format

import scanpy as sc

# Library to build pyLDAvis graph

import pyLDAvis

from pyLDAvis import sklearn as sklearn\_lda

from sklearn.feature\_extraction.text import CountVectorizer

# Path to database

path\_to\_database = sys.argv[1]

# Path to lda model

path\_to\_lda\_model = sys.argv[2]

# Path to data, on which lda model was learnt

path\_to\_data = sys.argv[3]

# Path to data, on which lda model was learnt

path\_to\_output\_data = sys.argv[4]

file\_name = sys.argv[5]

# Numer of genes, to get from each model topic

number\_of\_genes\_in\_topics = 200

# Opening database

data\_base = open(path\_to\_database, "r")

# Reading database and parsing it into dictionary

lines\_in\_database = data\_base.readline()

lines\_in\_database = data\_base.readline()

base\_dict = {}

data = []

while (lines\_in\_database):

data.append(lines\_in\_database.split('\t'))

lines\_in\_database = data\_base.readline()

for i in range(len(data)):

cell\_type = data[i][4]

cell\_name = data[i][5]

base\_dict[cell\_name, cell\_type] = []

for i in range(len(data)):

genes\_array = data[i][8].replace(" ", "").split(',')

for j in range(len(genes\_array)):

genes\_array[j] = genes\_array[j].replace("[", "")

genes\_array[j] = genes\_array[j].replace("]", "")

cell\_type = data[i][4]

cell\_name = data[i][5]

for j in range(len(genes\_array)):

base\_dict[cell\_name, cell\_type].append(genes\_array[j])

# End of reading database

# Readind lda model

lda\_model = joblib.load(path\_to\_lda\_model)

model\_data = sc.read\_10x\_mtx(path\_to\_data + '/')

# Getting top-genes for each theme

genes\_names = []

for row in sc.get.var\_df(model\_data).index:

genes\_names.append(row)

genes\_by\_theme = []

for topic\_idx, topic in enumerate(lda\_model.components\_):

topic\_genes\_names = ""

topic\_genes\_names += " ".join([genes\_names[i]

for i in topic.argsort()[:-number\_of\_genes\_in\_topics - 1:-1]])

genes\_by\_theme.append(topic\_genes\_names.split(" "))

# Genes per topic are now in genes\_by\_theme array

answer = []

for i in range(lda\_model.n\_components):

genes\_intersection = {}

step\_answer = []

for elem in (base\_dict):

cnt = 0

for k in range(len(base\_dict[elem])):

for j in range(len(genes\_by\_theme[i])):

if (base\_dict[elem][k] == genes\_by\_theme[i][j]):

cnt += 1

name\_and\_type = []

name\_and\_type.append(elem[0])

name\_and\_type.append(elem[1])

genes\_intersection[float(cnt) / float(len(base\_dict[elem]))] = name\_and\_type

sorted\_values = sorted(genes\_intersection, reverse=True)

number = 0

for j in range(5):

step\_answer.append(genes\_intersection[sorted\_values[j]])

answer.append(step\_answer)

file = open(path\_to\_output\_data + "/" + file\_name + ".txt", "w")

file.write("Here are predicted cells by genes significance in each topic." + '\n')

for i in range(len(answer)):

file.write("Topic " + str(i) + '\n')

for j in range(len(answer[i])):

file.write("Cell name: " + answer[i][j][0] + "; Cell type: " + answer[i][j][1] + '\n')

file.close()

# Building pyLDAvis graph

data\_for\_pyLDA = []

for row in sc.get.var\_df(model\_data).index:

data\_for\_pyLDA.append(row)

vectorizer\_n = CountVectorizer(lowercase=False, token\_pattern=r"(?u)\w+\.\w+|\w+\-\w+|\w+|\.\w+|\w+\.|\w+\-|\-\w+")

count\_data = vectorizer\_n.fit\_transform(data\_for\_pyLDA)

LDAvis\_prepared = sklearn\_lda.prepare(lda\_model, count\_data, vectorizer\_n)

pyLDAvis.save\_html(LDAvis\_prepared, path\_to\_output\_data + '/' + file\_name + '.html')

## **lda\_theme\_proportion.py**

import sys

#Library to save LDA model

import joblib

#Library to read files format

import scanpy as sc

#Library, which contains LDA model trainer

from sklearn.decomposition import LatentDirichletAllocation as LDA

#Libary for plotting

import pandas as pd

import matplotlib.pyplot as plot

import numpy as np

#Reading data

path\_to\_model = sys.argv[1]

path\_to\_model\_data = sys.argv[2]

path\_to\_users\_data = sys.argv[3]

path\_to\_images = sys.argv[4]

file\_name = sys.argv[5]

#Reading LDA model

lda\_model = joblib.load(path\_to\_model)

model\_data = sc.read\_10x\_mtx(path\_to\_model\_data + '/')

data\_out = model\_data.X

lda\_output = lda\_model.transform(data\_out)

#Creating a bar plot of percentage of cell divisions by topic in trained model

df\_cell\_topic = pd.DataFrame(np.round(lda\_output, 2))

df\_cell\_dominant\_topic = np.argmax(df\_cell\_topic.values, axis=1)

df\_cell\_topic['dominant\_topic'] = df\_cell\_dominant\_topic

number\_of\_dominant\_topics = df\_cell\_topic['dominant\_topic'].value\_counts()

topics = []

for i in range (lda\_model.n\_components):

topics.append(i)

values = []

for i in range (lda\_model.n\_components):

try:

values.append(number\_of\_dominant\_topics[i] / float(data\_out.shape[0]) \* 100)

except Exception:

values.append(0)

# Dictionary loaded into a DataFrame

dataFrame = pd.DataFrame(data={"Topic":topics, "Percentage":values })

# Draw a vertical bar chart

dataFrame.plot.bar(x="Topic", y="Percentage", rot=70, title="")

plot.savefig(path\_to\_images + "/" + file\_name + "1.png", dpi = 300)

#Reading data

data = sc.read\_10x\_mtx(path\_to\_users\_data + '/')

#Applying data on a trained model to get cells by topics

data\_test = data.X

topic\_probability\_scores = lda\_model.transform(data\_test)

df\_cell\_topic\_user = pd.DataFrame(np.round(topic\_probability\_scores, 2))

df\_cell\_dominant\_topic\_user = np.argmax(df\_cell\_topic\_user.values, axis=1)

df\_cell\_topic\_user['dominant\_topic'] = df\_cell\_dominant\_topic\_user

number\_of\_dominant\_topics\_user = df\_cell\_topic\_user['dominant\_topic'].value\_counts()

topics = []

for i in range (lda\_model.n\_components):

topics.append(i)

values = []

for i in range (lda\_model.n\_components):

try:

values.append(number\_of\_dominant\_topics\_user[i] / float(data\_test.shape[0]) \* 100)

except Exception:

values.append(0)

# Dictionary loaded into a DataFrame

dataFrame = pd.DataFrame(data={"Topic":topics, "Percentage":values })

# Draw a vertical bar chart

dataFrame.plot.bar(x="Topic", y="Percentage", rot=70, title="")

plot.savefig(path\_to\_images + "/" + file\_name + "2.png", dpi = 300)

## **learn\_new\_model.py**

import sys

#Library to save LDA model

import joblib

#Library to read files format

import scanpy as sc

#Library, which contains LDA model trainer

from sklearn.decomposition import LatentDirichletAllocation as LDA

#Reading data

lda\_model\_name = sys.argv[1]

path\_to\_files = sys.argv[2]

number\_of\_themes = int(sys.argv[3])

a\_Data = sc.read\_10x\_mtx(path\_to\_files + '/')

path\_to\_model = sys.argv[4] + '/'

#Learning new model

data = a\_Data.X

lda\_model = LDA(n\_components=number\_of\_themes, learning\_method='online')

lda\_output = lda\_model.fit\_transform(data)

#Saving new model

joblib.dump(lda\_model, path\_to\_model + lda\_model\_name + '.jl')

## **bio/\_\_init\_\_.py**

## **bio/admin.py**

from django.contrib import admin

# Register your models here.

## **bio/apps.py**

from django.apps import AppConfig

class BioConfig(AppConfig):

name = 'bio'

## **bio/models.py**

from django.db import models

# Create your models here.

## **bio/test.py**

from django.test import TestCase

# Create your tests here.

## **bio/urls/py**

from django.urls import path

from .views import UseBuiltModel, BuildModel

app\_name = "bio"

# app\_name will help us do a reverse look-up latter.

urlpatterns = [

path('get\_mtx\_data', UseBuiltModel.as\_view()),

path('build\_model\_and\_get\_mtx\_data', BuildModel.as\_view())

]

## **bio/views.py**

import os

import zipfile

import subprocess

from django.core.files.base import ContentFile

from django.core.files.storage import default\_storage

from rest\_framework.response import Response

from rest\_framework.views import APIView

from bio\_settings.settings import SERVER\_URL

import sys

#Library to save LDA model

import joblib

#Library to read files format

import scanpy as sc

#Library, which contains LDA model trainer

from sklearn.decomposition import LatentDirichletAllocation as LDA

#Libary for plotting

import pandas as pd

import matplotlib.pyplot as plot

import numpy as np

import pyLDAvis

import random

def clearData(filePath):

folder = filePath

for the\_file in os.listdir(folder):

next\_file\_path = os.path.join(folder, the\_file)

try:

if os.path.isfile(next\_file\_path):

os.unlink(next\_file\_path)

except Exception as e:

print()

def useBloodModel(file):

clearData("media/plots")

clearData("media/OutputData")

z = zipfile.ZipFile("media/" + file, 'r')

z.extractall(path="media/archives")

data = sc.read\_10x\_mtx("media/archives/")

input\_number = open("media/systemFiles/number\_of\_questions.txt", "r")

number = int(input\_number.readline()) + 1

input\_number.close()

input\_number = open("media/systemFiles/number\_of\_questions.txt", "w")

input\_number.write(str(number))

input\_number.close()

file\_name = "model\_data" + str(number)

p = subprocess.Popen("python Scripts/10x\_mtx\_cutter.py media/archives media/TrainData/blood", stdout=subprocess.PIPE, shell=True)

p.wait()

p = subprocess.Popen("python Scripts/lda\_theme\_proportion.py media/models/blood\_lda\_model.jl media/TrainData/blood media/archives media/plots " + file\_name, stdout=subprocess.PIPE, shell=True)

p.wait()

p = subprocess.Popen("python Scripts/database\_prepare.py media/DataBase/Human\_cell\_markers.txt media/models/blood\_lda\_model.jl media/TrainData/blood media/OutputData " + file\_name,

stdout=subprocess.PIPE, shell=True)

p.wait()

return {"Predicted cells by clusters:": 'media/OutputData/' + file\_name + '.txt', "Interactive pyLDAvis graph:": 'media/OutputData/' + file\_name + '.html', "The distribution of barcodes in model.": 'media/plots/' + file\_name +'1.png', "The distribution of barcodes in users data due to model.": 'media/plots/' + file\_name + '2.png'}

def buildModelAndUseIt(file):

clearData("media/plots")

clearData("media/OutputData")

z = zipfile.ZipFile("media/" + file, 'r')

z.extractall(path="media/archives")

data = sc.read\_10x\_mtx("media/archives/")

input\_number = open("media/systemFiles/number\_of\_questions.txt", "r")

number = int(input\_number.readline()) + 1

input\_number.close()

input\_number = open("media/systemFiles/number\_of\_questions.txt", "w")

input\_number.write(str(number))

input\_number.close()

file\_name = "users\_data" + str(number)

p = subprocess.Popen("python Scripts/learn\_new\_model.py users\_lda\_model media/archives 10 media/models", stdout=subprocess.PIPE, shell=True)

p.wait()

p = subprocess.Popen("python Scripts/lda\_theme\_proportion.py media/models/users\_lda\_model.jl media/archives media/archives media/plots " + file\_name, stdout=subprocess.PIPE, shell=True)

p.wait()

p = subprocess.Popen("python Scripts/database\_prepare.py media/DataBase/Human\_cell\_markers.txt media/models/users\_lda\_model.jl media/archives media/OutputData " + file\_name, stdout=subprocess.PIPE, shell=True)

p.wait()

return {"Predicted cells by clusters:": 'media/OutputData/' + file\_name + '.txt', "Interactive pyLDAvis graph:": 'media/OutputData/' + file\_name + '.html', "The distribution of barcodes in model.": 'media/plots/' + file\_name + '1.png'}

def saveFile(file, pathToSave):

data = file.read()

path = default\_storage.save(pathToSave, ContentFile(data))

return path

def formUrlOnServer(pathes):

for key in pathes:

pathes[key] = SERVER\_URL + pathes[key]

return pathes

class UseBuiltModel(APIView):

def post(self, request):

clearData("media/archives")

file = request.FILES['photo']

pathToArch = saveFile(file, 'archives/arch.zip')

pathes = useBloodModel(pathToArch)

pathes = formUrlOnServer(pathes)

print(pathToArch)

return Response(pathes)

class BuildModel(APIView):

def post(self, request):

clearData("media/archives")

file = request.FILES['photo']

pathToArch = saveFile(file, 'archives/arch.zip')

pathes = buildModelAndUseIt(pathToArch)

pathes = formUrlOnServer(pathes)

print(pathToArch)

return Response(pathes)

## **bio\_settings/setting/\_\_init\_\_.py**

## **bio\_settings/asgi.py**

"""

ASGI config for bio\_settings project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see

https://docs.djangoproject.com/en/3.0/howto/deployment/asgi/

"""

import os

from django.core.asgi import get\_asgi\_application

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'bio\_settings.settings')

application = get\_asgi\_application()

## **bio\_settings/setting.py**

"""

Django settings for bio\_settings project.

Generated by 'django-admin startproject' using Django 3.0.5.

For more information on this file, see

https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/3.0/ref/settings/

"""

import os

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'tjp\*cv6np97pabz1j^8w37q4#@5i-60626ivu^uc&t2f7%!1j0'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'corsheaders',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'rest\_framework',

'bio'

]

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

'corsheaders.middleware.CorsMiddleware',

'django.middleware.common.CommonMiddleware',

]

CORS\_ORIGIN\_ALLOW\_ALL = True # If this is used then `CORS\_ORIGIN\_WHITELIST` will not have any effect

CORS\_ALLOW\_CREDENTIALS = True

CORS\_ORIGIN\_WHITELIST = [

'http://localhost:63344',

] # If this is used, then not need to use `CORS\_ORIGIN\_ALLOW\_ALL = True`

CORS\_ORIGIN\_REGEX\_WHITELIST = [

'http://localhost:53344',

]

ROOT\_URLCONF = 'bio\_settings.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [os.path.join(BASE\_DIR, 'templates')]

,

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.debug',

'django.template.context\_processors.media'

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

WSGI\_APPLICATION = 'bio\_settings.wsgi.application'

# Database

# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

# Password validation

# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

# Internationalization

# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_L10N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC\_URL = '/static/'

MEDIA\_DIR = os.path.join(BASE\_DIR, 'media')

MEDIA\_ROOT = MEDIA\_DIR

MEDIA\_URL = '/media/'

SERVER\_URL = 'http://localhost:8000/'

## **bio\_settings/urls.py**

"""bio\_settings URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:

https://docs.djangoproject.com/en/3.0/topics/http/urls/

Examples:

Function views

1. Add an import: from my\_app import views

2. Add a URL to urlpatterns: path('', views.home, name='home')

Class-based views

1. Add an import: from other\_app.views import Home

2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path

2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

"""

from django.conf.urls.static import static

from django.contrib import admin

from django.urls import path, include

from bio\_settings import settings

urlpatterns = [

path('admin/', admin.site.urls),

path('api/', include('bio.urls'))

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

## **bio\_settings/wsgi.py**

"""

WSGI config for bio\_settings project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see

https://docs.djangoproject.com/en/3.0/howto/deployment/wsgi/

"""

import os

from django.core.wsgi import get\_wsgi\_application

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'bio\_settings.settings')

application = get\_wsgi\_application()