Laborator 14: Django, Apache Spark & Kafka

Introducere

Pentru o imagine de ansamblu asupra framework-ului Django, vezi:

- https://docs.djangoproject.com/en/3.0/
- https://buildmedia.readthedocs.org/media/pdf/django/latest/django.pdf
- „Django 3 by example" - Antonio Melé
- „Django 3 web development cookbook" (4th edition) - Aidas Bendoraitis şi Jake Kronika

Pentru documentaţia framework-ului Apache Spark în Python, vezi:

- https://spark.apache.org/docs/2.4.5/api/python/
- https://spark.apache.org/docs/2.4.5/rdd-programming-guide.html
- https://spark.apache.org/docs/2.4.5/submitting-applications.html
- https://spark.apache.org/docs/2.4.5/sql-getting-started.html
- https://spark.apache.org/docs/2.4.5/streaming-programming-guide.html (în special transformările pe DStream-uri şi operaţiile de output pe DStream-uri)

Exemple

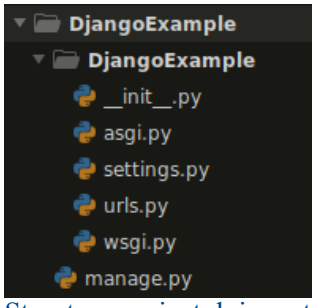
Instalare Django şi PySpark:

```
python3 -m venv env
source env/bin/activate
pip3 install Django==3.0.6 django-taggit==1.3.0 pyspark
```

Crearea proiectului Django:

```
django-admin startproject DjangoExample
```

ATENŢIE: Pentru a evita conflictele, proiectele nu se denumesc după module Python built-in sau module Django.



Structura proiectului creat

- ***manage.py*** (wrapper peste ***django-admin.py***) - utilitar de tip linie de comandă folosit pentru a interacţiona cu proiectul
- DjangoExample/ - directorul proiectului, care conţine următoarele fişiere:
  - › ***__init__.py*** - fişier gol care marchează faptul că Python va trata acest director ca un modul Python
  - › ***asgi.py*** - configurarea pentru execuţia proiectului ca ASGI[1]
  - › ***settings.py*** - setări şi configurări pentru proiect (conţine şi câteva setări implicite)
  - › ***urls.py*** - maparea URL-urilor cu view-uri

[1] ASGI = Asynchronous Server Gateway Interface

› ***wsgi.py*** - configurarea pentru execuţia proiectului ca WSGI[2]

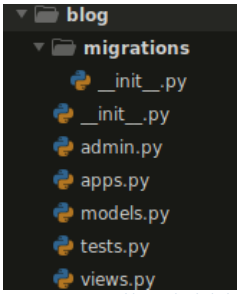
Setările proiectului - settings.py

- `DEBUG` - variabilă booleană care activează/dezactivează modul de debug. Dacă este setată pe True, Django va afişa pagini de erori detaliate când apare o excepţie şi nu este tratată. În producţie, trebuie dezactivat modul de debug, altfel sunt expuse date sensibile legate de proiect).
- `ALLOWED_HOSTS` - nu se aplică când modul de debug este activ, sau când se execută testele. În producţie, după ce se dezactivează modul de debug, trebuie adăugat domeniul în această listă.
- `INSTALLED_APPS` - setare care trebuie modificată pentru toate proiectele. Această setare precizează ce aplicaţii sunt active pentru acest site. Implicit, Django include următoarele aplicaţii:
  - › `django.contrib.admin` - un site de administrare
  - › `django.contrib.auth` - un framework de autentificare
  - › `django.contrib.contenttypes` - un framework pentru gestionarea tipurilor de conţinut
  - › `django.contrib.sessions` – un framework de sesiune
  - › `django.contrib.messages` – un framework de mesaje
  - › `django.contrib.staticfiles` – un framework pentru gestionarea fişierelor statice
- `MIDDLEWARE` - o listă care conţine middleware-urile care vor fi executate
- `ROOT_URLCONF` - indică locaţia fişierului urls.py (în cazul proiectului curent: DjangoExample.urls)
- `DATABASES` - dicţionar care conţine setările pentru toate bazele de date utilizate în proiect. Întotdeauna există o setare default în acest dicţionar. Configurarea implicită utilizează o bază de date SQLite3.
- `TIME_ZONE = 'Europe/Bucharest'`
- `USE_TZ` - activează/dezactivează suportul pentru timezone

Crearea unei aplicaţii de tip blog

Într-un terminal deschis în directorul DjangoExample (în care se află fişierul *manage.py*) se va executa comanda:

```
python3 manage.py startapp blog
```



Structura aplicaţiei blog

- `admin.py` - pentru înregistrarea modelelor pentru a fi incluse în site-ul de administrarea Django.
- `apps.py` - configuraţia principală pentru aplicaţia blog
- `migrations` - folder care conţine migrările bazei de date a aplicaţiei. Migrările permit ca Django să urmărească schimbările în model şi să sincronizeze baza de date corespunzător.

---

[2] WSGI = Web Server Gateway Interface

- `models.py` - modelele de date ale aplicaţiei; toate aplicaţiile Django au acest fişier *models.py,* dar poate fi lăsat şi gol.
- `tests.py` - teste pentru aplicaţie
- `views.py` - logica aplicaţiei; fiecare view primeşte o cerere HTTP, o procesează şi returnează răspunsul.

blog/models.py

Observaţie: *slug* este un câmp ce se intenţionează a fi utilizat în URL-uri. Acesta este o etichetă scurtă care conţine doar litere, numere, sau cratime.

```
from django.db import models
from django.utils import timezone
from django.urls import reverse
from django.contrib.auth.models import User


class PublishedManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(status='published')


class Post(models.Model):
    STATUS_CHOICES = (
        ('draft', 'Draft'),
        ('published', 'Published'),
    )
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250, unique_for_date='publish')
    author = models.ForeignKey(User,
                               on_delete=models.CASCADE,
                               related_name='blog_posts')
    _body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    status = models.CharField(max_length=10,
                              choices=STATUS_CHOICES,
                              default='draft')

    def set_body(self, body):
        self._body = body
        self._body_changed = True

    def get_body(self):
        return self._body

    body = property(get_body, set_body)

    objects = models.Manager()  # The default manager.
    published = PublishedManager()  # Our custom manager.

    class Meta:
```

```
        ordering = ('-publish', )

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('blog:post_detail',
                       args=[
                           self.publish.year, self.publish.month,
                           self.publish.day, self.slug
                       ])

    def save(self, *args, **kwargs):
        if getattr(self, '_body_changed', True):
            # TODO - Kafka Producer
            pass
        super(Post, self).save(*args, **kwargs)
```

Activarea aplicaţiei (DjangoExample/settings.py):

```
# ...
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'blog.apps.BlogConfig',
]
# ...
```

blog/templates/blog/post/detail.html

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

{% block title %}{{ post.title }}{% endblock %}

{% block content %}
  <h1>{{ post.title }}</h1>
  <p class="date">
    Published {{ post.publish }} by {{ post.author }}
  </p>
  {{ post.body|linebreaks }}
{% endblock %}
```

blog/templates/blog/post/list.html

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

{% block title %}My Blog{% endblock %}
```

```
{% block content %}
  <h1>My Blog</h1>
  {% for post in posts %}
    <h2>
      <a href="{{ post.get_absolute_url }}">
        {{ post.title }}
      </a>
    </h2>
    <p class="date">
      Published {{ post.publish }} by {{ post.author }}
    </p>
    {{ post.body|truncatewords:30|linebreaks }}
  {% endfor %}
  {% include "pagination.html" with page=page_obj %}
{% endblock %}
```

blog/templates/blog/base.html

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}{% endblock %}</title>
  <link href="{% static "css/blog.css" %}" rel="stylesheet">
</head>
<body>
  <div id="content">
    {% block content %}
    {% endblock %}
  </div>
  <div id="sidebar">
    <h2>My blog</h2>
    <p>This is my blog.</p>
  </div>
</body>
</html>
```

blog/templates/pagination.html

```
<div class="pagination">
  <span class="step-links">
    {% if page.has_previous %}
      <a href="?page={{ page.previous_page_number }}">Previous</a>
    {% endif %}
    <span class="current">
      Page {{ page.number }} of {{ page.paginator.num_pages }}.
    </span>
    {% if page.has_next %}
      <a href="?page={{ page.next_page_number }}">Next</a>
    {% endif %}
  </span>
</div>
```

blog/static/css/blog.css

```
body {
    margin:0;
    padding:0;
    font-family:helvetica, sans-serif;
}

a {
    color:#00abff;
    text-decoration:none;
}

h1 {
    font-weight:normal;
    border-bottom:1px solid #bbb;
    padding:0 0 10px 0;
}

h2 {
    font-weight:normal;
    margin:30px 0 0;
}

#content {
    float:left;
    width:60%;
    padding:0 0 0 30px;
}

#sidebar {
    float:right;
    width:30%;
    padding:10px;
    background:#efefef;
    height:100%;
}

p.date {
    color:#ccc;
    font-family: georgia, serif;
    font-size: 12px;
    font-style: italic;
}

/* pagination */
.pagination {
    margin:40px 0;
    font-weight:bold;
}

/* forms */
label {
```

```
    float:left;
    clear:both;
    color:#333;
    margin-bottom:4px;
}
input, textarea {
    clear:both;
    float:left;
    margin:0 0 10px;
    background:#ededed;
    border:0;
    padding:6px 10px;
    font-size:12px;
}
input[type=submit] {
    font-weight:bold;
    background:#00abff;
    color:#fff;
    padding:10px 20px;
    font-size:14px;
    text-transform:uppercase;
}
.errorlist {
    color:#cc0033;
    float:left;
    clear:both;
    padding-left:10px;
}

/* comments */
.comment {
    padding:10px;
}
.comment:nth-child(even) {
    background:#efefef;
}
.comment .info {
    font-weight:bold;
    font-size:12px;
    color:#666;
}
```

blog/views.py

```
from django.shortcuts import render, get_object_or_404
from django.core.paginator import Paginator, EmptyPage,
PageNotAnInteger
from django.views.generic import ListView
from .models import Post


def post_list(request):
```

```
    object_list = Post.published.all()
    paginator = Paginator(object_list, 3)  # 3 posts in each page
    page = request.GET.get('page')
    try:
        posts = paginator.page(page)
    except PageNotAnInteger:
        # If page is not an integer deliver the first page
        posts = paginator.page(1)
    except EmptyPage:
        # If page is out of range deliver last page of results
        posts = paginator.page(paginator.num_pages)
    return render(request, 'blog/post/list.html', {
        'page': page,
        'posts': posts
    })


def post_detail(request, year, month, day, post):
    post = get_object_or_404(Post,
                             slug=post,
                             status='published',
                             publish__year=year,
                             publish__month=month,
                             publish__day=day)
    return render(request, 'blog/post/detail.html', {'post': post})


class PostListView(ListView):
    queryset = Post.published.all()
    context_object_name = 'posts'
    paginate_by = 3
    template_name = 'blog/post/list.html'
```

blog/admin.py

```
from django.contrib import admin
from .models import Post


@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'slug', 'author', 'publish', 'status')
    list_filter = ('status', 'created', 'publish', 'author')
    search_fields = ('title', 'body')
    prepopulated_fields = {'slug': ('title', )}
    raw_id_fields = ('author', )
    date_hierarchy = 'publish'
    ordering = ('status', 'publish')
```

blog/urls.py

```
from django.urls import path
```

```
from . import views

app_name = 'blog'

urlpatterns = [
    # post views
    # path('', views.post_list, name='post_list'),
    path('', views.PostListView.as_view(), name='post_list'),
    path('<int:year>/<int:month>/<int:day>/<slug:post>/',
         views.post_detail,
         name='post_detail'),
]
```

DjangoExample/urls.py

```
from django.contrib import admin
from django.urls import path, include

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

Crearea şi aplicarea migrărilor

```
python3 manage.py makemigrations blog
python3 manage.py sqlmigrate blog 0001
python3 manage.py migrate
```

Crearea unui superuser pentru site-ul de administrare (deja inclus în proiect)

```
python3 manage.py createsuperuser

# Username (leave blank to use 'student'):
# Email address: student@ac.tuiasi.ro
# Password: studentpw
# Password (again): studentpw
# This password is too short. It must contain at least 8 characters.
# Bypass password validation and create user anyway? [y/N]: y
# Superuser created successfully.
```

Pornirea server-ului

Se execută în terminal comanda `python3 manage.py runserver` apoi se deschide în browser următorul URL: http://127.0.0.1:8000/admin/

După ce se adaugă câteva postări pe blog, se navighează la URL-ul http://127.0.0.1:8000/blog/

Exemplu de creare a unui flux de date direct cu Kafka în python

Se va instala pyspark cu comanda: `pip3 install pyspark`

Se va descărca *spark-streaming-kafka-0-8-assembly_2.11-2.4.5.jar* de la adresa: https://repo1.maven.org/maven2/org/apache/spark/spark-streaming-kafka-0-8-assembly_2.11/2.4.5/spark-streaming-kafka-0-8-assembly_2.11-2.4.5.jar în folder-ul proiectului.

```
from pyspark.streaming.context import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
from pyspark.context import SparkContext
import os


if __name__ == '__main__':
    os.environ['PYSPARK_SUBMIT_ARGS'] = '--jars ./spark-streaming-
kafka-0-8-assembly_2.11-2.4.5.jar pyspark-shell'
    sparkContext = SparkContext(master="local", batchSize=0)
    ssc = StreamingContext(sparkContext, batchDuration=1)

    kafkaParams = {"bootstrap.servers": "localhost:9092"}
    kafkaStream = KafkaUtils.createDirectStream(ssc=ssc,
topics=["blog"], kafkaParams=kafkaParams)
    kafkaStream.foreachRDD(lambda rdd: print(rdd.collect()))

    ssc.start()
    ssc.awaitTermination()
```

ATENŢIE: Calea specificată în variabila de mediu este una relativă la folder-ul în care se află scriptul de mai sus.

Exemplu de streaming pe fişiere text

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
import os


if __name__ == '__main__':
    os.environ['PYSPARK_PYTHON'] = '/usr/bin/python3'
    sc = SparkContext("local", "Text File Streaming")
    ssc = StreamingContext(sc, 1)

    ROOT_DIR = os.path.abspath(os.path.dirname(__file__))
    path = "file:///" + os.path.join(ROOT_DIR, 'resources/text')
    stream = ssc.textFileStream(path)

    stream.foreachRDD(lambda rdd: print(rdd.collect()))

    ssc.start()
    ssc.awaitTermination()
```

Exemplu de Spark RDD

```
from pyspark import SparkContext, SparkConf, StorageLevel
import os
```

```
import re


if __name__ == '__main__':
    ''' configurare variabila de mediu cu versiunea python utilizata
pentru spark '''
    os.environ['PYSPARK_PYTHON'] = '/usr/bin/python3'
    # configurarea Spark
    spark_conf = SparkConf().setMaster("local").setAppName("Spark
Example")
    # initializarea contextului Spark
    spark_context = SparkContext(conf=spark_conf)

    items = ["123/643/7563/2134/ALPHA", "2343/6356/BETA/2342/12",
"23423/656/343"]
    # paralelizarea colectiilor
    distributed_dataset = spark_context.parallelize(items)  # RDD

    ''' 1) spargerea fiecarui string din lista intr-o lista de
substring-uri si reunirea intr-o singura lista
    2) filtrarea cu regex pentru a pastra doar numerele
    3) conversia string-urilor filtrate la int prin functia de mapare
    4) sumarea tuturor numerelor prin functia de reducere '''
    sum_of_numbers = distributed_dataset.flatMap(lambda item:
item.split("/"))\
        .filter(lambda item: re.match("[0-9]+", item))\
        .map(lambda item: int(item))\
        .reduce(lambda total, next_item: total + next_item)
    print("Sum of numbers =", sum_of_numbers)

    ''' seturi de date externe
     setul de date nu este inca incarcat in memorie (si nu se
actioneaza inca asupra lui) '''
    ROOT_DIR = os.path.abspath(os.path.dirname(__file__))
    path = "file:///" + os.path.join(ROOT_DIR, 'resources/data.txt')
    lines = spark_context.textFile(path)

    ''' pentru utilizarea unui RDD de mai multe ori, trebuie apelata
metoda persist: '''
    lines.persist(StorageLevel.MEMORY_ONLY)

    ''' functia de mapare reprezinta o transformare a setului de date
initial (nu este calculat imediat)
     abia cand se ajunge la functia de reducere (care este o actiune)
Spark imparte operatiile in task-uri
     pentru a fi rulate pe masini separate (fiecare masina executand o
parte din map si reduce)
     exemplu cu functii lambda: '''
    total_length0 = lines.map(lambda s: len(s)).reduce(lambda acc, i:
acc + i)

    print("Total length =", total_length0)
```

```
    ''' variabila partajata de tip broadcast
     trimiterea unui set de date ca input catre fiecare nod intr-o
maniera eficienta: '''
    broadcast_var = spark_context.broadcast([1, 2, 3])
    total_length1 = lines.map(lambda s: len(s) +
broadcast_var.value[0]).reduce(lambda acc, i: acc + i)
    print("Sum(line_length + broadcast_val[0])=", total_length1)
    # variabila partajata de tip acumulator
    accumulator = spark_context.accumulator(0)
    spark_context.parallelize([1, 2, 3, 4]).foreach(lambda x:
accumulator.add(x))
    print("Accumulator =", accumulator)

    # oprirea contextului Spark
    spark_context.stop()
```

Exemplu de Spark SQL

```
from pyspark.sql import SparkSession, Row
import os


if __name__ == '__main__':
    ''' configurare variabila de mediu cu versiunea python utilizata
pentru spark '''
    os.environ['PYSPARK_PYTHON'] = '/usr/bin/python3'
    # configurarea si crearea sesiunii Spark SQL
    spark_session = SparkSession\
        .builder\
        .appName("Python Spark SQL example")\
        .config("spark.master", "local")\
        .getOrCreate()

    # initializarea unui DataFrame prin citirea unui json
    ROOT_DIR = os.path.abspath(os.path.dirname(__file__))
    people_json_path = "file:///" + os.path.join(ROOT_DIR,
'resources/people.json')
    df = spark_session.read.json(people_json_path)

    # afisarea continutului din DataFrame la consola
    df.show()

    # Afisarea schemei DataFrame-ului intr-o forma arborescenta
    df.printSchema()

    # Selectarea coloanei nume si afisarea acesteia
    df.select("name").show()

    # Selectarea tuturor datelor si incrementarea varstei cu 1
    df.select(df["name"], df["age"] + 1).show()
```

```
    # Selectarea persoanelor cu varsta > 21 ani
    df.filter(df["age"] > 21).show()

    # Numararea persoanelor dupa varsta
    df.groupBy("age").count().show()

    # Inregistarea unui DataFrame ca un SQL View temporar
    df.createOrReplaceTempView("people")

    # Utilizarea unei interogari SQL pentru a selecta datele
    sqlDF = spark_session.sql("SELECT * FROM people")
    sqlDF.show()

    # Inregistrarea unui DataFrame ca un SQL View global temporar
    df.createGlobalTempView("people")

    ''' Un SQL View global temporar este legat de o baza de date a
sistemului: `global_temp` '''
    spark_session.sql("SELECT * FROM global_temp.people").show()

    # Un view global temporar este vizibil intre sesiuni
    spark_session.newSession().sql("SELECT * FROM
global_temp.people").show()

    # Interoperabilitatea cu RDD-uri
    # Crearea unui RDD de obiecte Person dintr-un fisier text
    people_txt_path = "file:///" + os.path.join(ROOT_DIR,
'resources/people.txt')
    lines = spark_session.sparkContext.textFile(people_txt_path)
    people = lines.map(lambda l: l.split(",")).map(lambda p:
Row(name=p[0], age=int(p[1])))

    ''' Aplicarea unei scheme pe un RDD de bean-uri pentru a obtine
DataFrame '''
    peopleDF = spark_session.createDataFrame(people)
    # Inregistrarea DataFrame-ului ca un view temporar
    peopleDF.createOrReplaceTempView("people")

    # Selectarea persoanelor intre 13 si 19 ani cu o interogare SQL
    teenagersDF = spark_session.sql("SELECT name FROM people WHERE age
BETWEEN 13 AND 19")

    teen_names = teenagersDF.rdd.map(lambda p: "Name: " +
p.name).collect()
    for name in teen_names:
        print(name)
```

Aplicaţii şi teme

Aplicaţii de laborator:

1. Pornind de la exemplul de mai sus, să se creeze un Kafka Producer şi să se modifice aplicaţia astfel încât, pe lângă salvarea în baza de date SQLite a postărilor pe blog, să se trimită într-un topic postările respective serializate cu ajutorul modulului *json*.

2. Utilizând modulul django-taggit (trebuie instalat cu pip3) să se adauge un câmp *tags* în clasa Post (care extinde clasa models.Model) de tipul *TaggableManager*

   Observaţie: aplicaţia *taggit* va fi inclusă în aplicaţiile instalate, apoi se vor reface migrările.

Teme pe acasă:

1. Utilizând Spark Streaming şi Spark RDD, să se creeze un stream direct prin intermediul KafkaUtils[3] şi să se realizeze o analiză (statistică) de sentimente pe baza a două fişiere text ce conţin cuvinte pozitive / negative (încărcate în aplicaţie ca RDD-uri). Practic, vor fi calculate şi comparate două procentaje: cuvinte_pozitive/nr_total_cuvinte * 100, cuvinte_negative/nr_total_cuvinte * 100. Dacă procentajele sunt relativ apropiate (diferenţă maximă de 5%), postarea va fi considerată neutră. În caz contrar, va fi considerată pozitivă, sau negativă, în funcţie de cel mai mare procentaj.

   Observaţie: A fost ataşat la laboratorul curent un set de cuvinte pozitive/negative[4]. Se pot utiliza şi alte seturi, dacă se doreşte. De asemenea, vezi şi exemplul de creare a unui flux direct de date cu Kafka în python.

2. Să se creeze un Kafka Producer care să trimită într-un topic tag-urile calculate (pozitiv, neutru, negativ) pentru fiecare postare în parte. Un Kafka Consumer va prelua tag-urile din topic şi va modifica baza de date SQLite astfel încât fiecare postare să aibă tag-ul asociat. La final, se va forţa o reîncărcare a paginii (nu din cache) pentru a observa rezultatul.

3. Să se facă deployment la aplicaţia Django pe o maşină cloud oferită de Heroku[5] sau într-un container docker[6].

[BONUS]: Utilizând algoritmul K-means din Spark MLlib[7], să se grupeze postările pe blog astfel încât, în funcţie de o postare selectată, să se determine toate postările similare. Pentru aceasta, se vor determina cele mai frecvente 5 cuvinte din fiecare postare (acestea fiind considerate cuvinte cheie), apoi se vor clusteriza postările de pe blog, rezultând 3 centroizi poziţionaţi în funcţie de distanţa cosinus dintre vectorii de cuvinte cheie ai fiecărei postări.

Observaţie: Se va utiliza algoritmul TF-IDF disponibil în modulul scikit-learn[8] pentru a calcula coeficienţii necesari pentru calculul distanţei cosinus.

---

[3] https://spark.apache.org/docs/2.4.5/api/python/pyspark.streaming.html#module-pyspark.streaming.kafka

[4] http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar

[5] https://devcenter.heroku.com/categories/working-with-django

[6] https://dzone.com/articles/how-to-deploy-a-django-application-with-docker

[7] https://spark.apache.org/docs/latest/mllib-clustering.html

[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html