Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»
Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Навчальний стенд із демонстрацією використання Apache Spark у вирішенні Big Data задач»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1811»

_____ /Руслан КОТЕНКО/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /доц. Олександр ІВАНОВ/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Олена КУРОПЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»
Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis
on the topic: «Training stand with demonstration of use Apache Spark in solving Big Data tasks»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group ПЗ1811: /Ruslan KOTENKO/

Scientific Supervisor: /Oleksandr IVANOV/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____/Вадим ГОРЯЧКІН/
(підпис)

Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра
студенту Котенку Руслану Олеговичу
1. Тема роботи: «Навчальний стенд із демонстраією використання Apache Spark у вирішенні Big Data задач»
Керівник роботи: Іванов Олександр Петрович, доцент
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: __.__.2022 р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати): збір вимог до програмного забезпечення, опис аналогів, перелік термінів та скорочень, опис архітектури проекту, зовнішнє проектування, вимоги до функціональних характеристик, внутрішнє проектування та написання програми, відлагодження системи, опис інфраструктури, запуск проекту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): діаграми класів, прецедентів, компонентів. Схема інфраструктури стенду.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми з урахуванням її актуальності та практичної цінності	01.03.2022 - 15.03.2022 рр.	
2	З'ясування об'єкта предмета, мети та завдання дослідження	16.03.2022 - 31.03.2022 рр	
3	Відбір основних теоретичних джерел та їх опрацювання	01.04.2022 - 15.04.2022 рр.	
4	Укладання попереднього плану з керівником	16.04.2022 - 30.04.2022 рр	
5	Розробка та опис поставленої задачі, згідно з обраними технологіями	01.05.2022 - 15.06.2022 рр.	
6	Подання кваліфікаційної роботи до кафедри	20.06.2022 р.	
7	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	21.06.2022 р.	

Студент _____ Руслан КОТЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ доц. Олександр ІВАНОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка складається з 8 розділів:

- вступ – в даному розділі описується сутність стенду, причина його створення, актуальність та поверхнево розглядається 3 задачі стенду. Складається з 2 сторінок;
- збір вимог до програмного забезпечення – у цьому розділі описується що є аналогами стенду, які є аналоги, вивчення потреб ринку, проводяться бесіди з цільовими користувачами стенду – студентами та викладачами. Складається з 4 сторінок;
- перелік термінів та скорочень – має список термінів, які використані при описі роботи стенду. Складається з 6 сторінок;
- опис архітектури проекту – у цьому розділі проведений огляд вхідних і вихідних даних, формалізація 3 задач, приводиться опис об'єктно-орієнтованого проектування, проектування інтерфейсу користувача, розробка коду проекту, вибір мови програмування та модулів фреймворку. Складається з 6 сторінок;
- зовнішнє проектування – опис функціонального та експлуатаційного призначення. Складається з 3 сторінок;
- внутрішнє проектування – опис обраних мов, фреймворків, бібліотек, субд для створення програмного продукту. Складається з 3 сторінок;
- відлагодження системи – опис процесу пошуку і усунення помилок. Складається з 3 сторінок.
- опис інфраструктури – включає в себе обґрунтування вибору інструментів для докеризації проекту, процесу докеризації а також – інструкції з використання стенду. Складається з 5 сторінок;
- список веб ресурсів – включає в себе список використаних веб ресурсів. Складається з 1 сторінки;
- додатки – містять рисунки з описом вхідних даних і текст програми;

Кількість таблиць: 2 штук.

Кількість рисунків: 12 штук.

Ключові слова: Big Data, Map Reduce, Apache Spark (Core, SQL, MLlib), Spark Context, SparkConfiguration, SparkSession, Spark In Memory Processing, RDD, Dataframe and Dataset, Tuple, кластеризація, алгоритм K-середніх, Java 11, Spring Boot, Maven, MongoDB, MongoDB-Spark-Connector, Bson, URI, Docker, Docker Compose.

ЗМІСТ

ВСТУП

Big Data – сфера програмування, яка вирішує проблеми обробки (ще кажуть - аналітики), зберігання на передачі великих об'ємів даних різних типів та форматів. Для вирішення задач цієї сфери використовуються різні бібліотеки до популярних мов та фреймворки. Найпопулярніші мови програмування Big Data – Java, Scala та Python, кожна з них підтримує роботу з двома основними фреймворками – Hadoop та Apache Spark.

За інформацією одного з сайтів статистики (statista.com) дохід від ринку технологій великих даних з 2012 по 2022 зріс майже у шість разів, це вказує на загальне збільшення необхідності у спеціалістах з цих технологій. 10 років назад ця сфера була доступна лише розробникам, які мають певний досвід (рівень Middle, Senior або вище). Але зараз із-за збільшення попиту на ринку вакансій в Україні можна побачити вакансії для початківців (рівень Trainee/Intern та Junior). Вивчивши ці вакансії на (devspace.com) та (https://jobs.dou.ua/) бачимо, що у кожній з них у графі «Вимоги» числився фреймворк Apache Spark. Доступних україномовних ресурсів для його вивчення в інтернеті було мало.

З вище сказаного, що дійсно корисним програмним продуктом був би демонстраційний стенд для навчання студентів, які хотіли б спробувати себе у якості Big Data розробника та вивчити можливості фреймворку Apache Spark.

Стенд організовано у вигляді вирішення трьох задач за допомогою різних модулей фреймворку Spark Core, SQL та Spark MLlib. Програма, яка використовує Spark виконана у вигляді http-сервера, його можна запустити, завантаживши docker-образ з репозиторія, на який буде посилання у цій роботі. Сервер написаний на Java, з використанням веб-фреймворку Spring Boot, також у одній з задач використовується документна NoSql база даних – MongoDb.

Задача пошуку слів у тексті є класичним прикладом вирішення задач на початку вивчання технологій великих даних. Саме тому її було обрано для двох з трьох задач. Перша задача читає з файлу логи роботи програми на Spring Boot та виводить кількість помилок та попереджень та, окремо, повідомлення цих стрічок з помилками та попередженнями. У другій задачі планувалося налаштувати потоки даних (в цьому випадку - твітів) з Twitter API, але для цього потрібно було отримати доступ до цього API. Після декількох спроб та вивчення умов отримання доступу (Twitter rules and policies) мені було відмовлено. Тому читання потоків даних з API було замінено на читання даних з файлу. Сутність другої задачі полягає у пошуку всіх хештегів, які є у твітах з хештегом #WARINUKRAINE. Кожній з цих двох задач відповідає ендпоінт у веб-додатку, тобто, можна відкрити браузер, ввести певний url, та отримати результат обробки даних Spark-ом.

Третя задача складається з двох ендпоінтів. Перший відповідає за створення даних, а другий за їх обробку. При запиті на перший ендпоінт Spark з'єднується з

MongoDb, яка в той момент буде запущена docker-ом, та записує 10000 документів у базу даних. При запиті на другий, ці дані читає фреймворк та за допомогою алгоритму К-середніх кластеризує дані на 5 кластерів.

Акцент в інфраструктурі проекту був зроблений на доступність цього стенду на різних системах, а саме Windows та дистрибутивах Linux. Під налаштуванням інфраструктури проекту мається на увазі його підготовка до використання іншими користувачами Для цього у запуску бази даних та самого jar-файлу проекту використовується docker compose – програмне забезпечення для автоматизації розгортання та управління програмами в середовищах з підтримкою контейнеризації.

Важливо згадати, що є відміни між Apache Spark та Spark Java Framework. Перший – «механізм» для виконання задач великих даних та роботи з машинним навчанням локально або на кластерах, а другий – простий фреймворк для створення http-сервісів (аналог більш відомого у світі Java – Spring MVC).

1. ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У якості методів збору вимог були обрані:

– бесіди з потенційними користувачами стенду, а саме – студентами та викладачами 121-ї спеціальності «Інженерія програмного забезпечення»;
– вивчення ринку вакансій на сайтах (https://jobs.dou.ua/) та (devspace.com) для формування розуміння, що саме треба знати молодим спеціалістам Big Data;
– вивчення документації фреймворку Apache Spark (https://spark.apache.org/docs/latest/) для формування того, які саме задачі, із необхідних ринку, можна за допомогою нього вирішувати.

Функціональне призначення – навчальний стенд має:

– демонструвати вирішення різних типів Big Data задач;
– демонструвати код вирішення цих задач, у якості прикладу використання Apache Spark;
– мати можливість запускатись на різних системах, а саме – Windows 10 (або версією вище) та Ubuntu 16.04 (або версією вище).

Експлуатаційне призначення – за допомогою навчального стенду можна:

– за допомогою програмного продукту виконується підготовка з дисципліни «Бази даних»;
– код стенду можна використовувати в якості прикладу вирішення задач на Apache Spark Java API.

Етапність формування функціональних та експлуатаційних вимог представлена у табл. 1.

Таблиця 1.1 Етапи створення вимог

Назва етапу	Зміст етапу
Етап підготовки	Розробка алгоритму зі збору вимог, створення списку студентів та викладачів для бесіди, створення списку сайтів для вивчення ринку вакансій, створення списку посилань на веб-сторінки документації.
Етап виконання	Процес збору інформації з бесід, сайтів з вакансіями, документації Spark
Етап створення результатів	Аналіз зібраної інформації та створення функціональних та експлуатаційних вимог

1.1 Опис аналогів

Аналогами тренувального стенду для демонстрації вирішення задач Big Data є приклади репозиторіїв з готовими задачами на GitHub (найбільший веб-сервіс для хостингу IT-проектів та їхньої спільної розробки).

Приклади аналогів:

- вирішення Apache Spark на Python [5]
- вирішення Apache Spark на Java [6]
- вирішення Apache Spark на Scala [7]
- вирішення Apache Spark на R (мова програмування для статистичної обробки даних та роботи з графікою, а також вільне програмне середовище обчислень з відкритим вихідним кодом) [8]

Збірка Spark у виконавчі файли на кожній з представлених мов

Spark збирається за допомогою Apache Maven. Щоб зібрати Spark у виконавчий пакет та його приклади, запустіть:

```
./build/mvn -DskipTests clean package
```

Інтерактивна оболонка Scala

Найпростіший спосіб почати використовувати Spark – це оболонка Scala:

```
./bin/spark-shell

scala> spark.range(1000 * 1000 * 1000).count()
```

Інтерактивна оболонка Python

В якості альтернативи Python, можно використовувати оболонку Python:

```
./bin/pyspark
```

Команда має повернути 1,000,000,000.

```
>>> spark.range(1000 * 1000 * 1000).count()
```

Spark також постачається з кількома прикладами програм у каталозі прикладів. Щоб запустити одну з них, натисніть ./bin/run-example <class> [params]. Наприклад:

```
./bin/run-example SparkPi
```

Можете встановити змінну оточення MASTER під час запуску прикладів, щоб надіслати приклади в кластер. Це може бути URL mesos:// або spark://, "yarn" для запуску на YARN, та "local" для локального запуску з одним потоком або "local[N]" для локального запуску з N потоками. Також можно використати скорочене ім'я класу, якщо клас знаходиться у пакеті examples. Наприклад:

```
MASTER=spark://host:7077 ./bin/run-example SparkPi
```

Висновки до розділу 1

На етапі формування вимог було описано те, яким має бути навчальний стенд. Методом аналізу ринку вакансій та веб-ресурсів, що готують молодих інженерів була зібрана така інформація:

- Однією з найперспективніших та найактуальніших технологій у сфері великих даних є Apache Spark.
- Відсутні доступні україномовні ресурси із прикладами коду цього фреймворку.

Методом бесід з потенційними користувачами стенду були сформовані такі вимоги до стенду:

- Стенд має демонструвати вирішення різних типів Big Data задач.
- Код вирішення цих задач, у якості прикладу використання Apache Spark.
- Мати можливість запускатись на різних системах, а саме – Windows 10 (або версією вище) та Ubuntu 16.04 (або версією вище).

2. ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

Big Data (Великі дані) - це різноманітні дані, які надходять зі швидкістю, що постійно зростає, і обсяг яких постійно зростає. Три основні властивості великих даних – різноманітність, висока швидкість надходження та великий обсяг. Великі дані - це більш об’ємні та складніші набори даних, особливо з нестандартних джерел. Розмір наборів цих даних настільки великий, що традиційні програми обробки не можуть з ними впоратися. Але ці величезні обсяги даних можна використовувати для вирішення бізнес-завдань, які раніше здавались надто складними.

Приклади використання великих даних:

1. *Розробка програмних продуктів.*

Такі компанії як Netflix та Procter & Gamble використовують великі дані для прогнозування споживчого попиту. Вони створюють передиктивні моделі для нових продуктів та послуг, класифікуючи ключові атрибути попередніх або існуючих продуктів та моделюючи взаємозв'язок між цими атрибутами та комерційним успіхом пропозицій.

2. *Передиктивне (пов'язане із прогнозуванням) керування обслуговуванням.*

Фактори, які дозволяють прогнозувати збої механіки, можуть ховатися в надрах структурованих даних, таких як рік, марка і модель обладнання, або неструктурованих даних, таких як записи журналів, дані датчиків, повідомлення про помилки і відомості про температуру двигуна. Проаналізувавши індикатори можливих проблем до їх виникнення, компанії можуть підвищити економічну ефективність технічного обслуговування і максимально продовжити термін служби запчастин та обладнання.

3. *Якість обслуговування клієнтів.*

Боротьба за замовників у самому розпалі. Великі дані дозволять Вам отримати корисні відомості із соцмереж, інформації про відвідування веб-сайтів та інших джерел, таким чином підвищивши якість взаємодії з клієнтами та зробивши свої пропозиції максимально корисними.

4. *Виявлення несанкціонованого доступу та дотримання нормативних вимог.*

Коли справа стосується безпеки, йдеться не просто про пару хакерів: проти вас виступають цілі команди досвідчених фахівців. Нормативні вимоги та стандарти безпеки постійно змінюються. Великі дані дозволяють визначати шаблони, характерні для шахраїв, та збирати значні обсяги даних, щоб прискорити подання нормативної звітності.

5. *Машинне навчання.*

Сьогодні машинне навчання – одна з найпопулярніших тем для. І дані - особливо великі дані - є однією з причин цієї популярності. Сьогодні ми можемо навчати

машини замість того, щоб програмувати їх. Саме доступність великих даних зробила це можливим.

6. *Впровадження інновацій.*

Великі дані дозволяють виявляти взаємозалежності між користувачами, установами та компаніями, обробляти ці взаємозалежності та визначати нові способи застосування отриманих відомостей. Використовуйте результати досліджень даних, щоб підвищити ефективність фінансових рішень та планування. Вивчайте тенденції та бажання покупців, щоб випускати нові продукти та послуги. Впровадьте динамічне ціноутворення. Можливості справді безмежні.

Apache Spark – це фреймворк для обробки даних, який може швидко виконувати завдання обробки дуже великих наборів даних, а також може розподіляти завдання обробки даних між кількома комп'ютерами, самостійно або разом з іншими розподіленими обчислювальними засобами. Spark також знімає частину тягаря програмування цих завдань з плечей розробників за допомогою простого у використанні API, який абстрагує більшу частину важкої роботи розподілених обчислень та обробки великих даних.

Apache Spark Core – базовий механізм загального виконання для платформи Spark, на якому побудовані всі інші функції. Він надає обчислювальні можливості в пам'яті для забезпечення швидкості, узагальнену модель виконання для підтримки широкого спектру додатків, а також API Java, Scala та Python для спрощення розробки.

Apache Spark SQL – модуль фреймворку, який забезпечує вбудовану підтримку SQL для Spark і спрощує процес запитів даних, що зберігаються як у RDD (розподілені набори даних Spark), так і у зовнішніх джерелах. Spark SQL зручно розмиває межі між RDD і реляційними таблицями. Об'єднання цих потужних абстракцій дозволяє розробникам легко змішувати команди SQL, які запитують зовнішні дані, зі складною аналітикою, і все це в одній програмі.

Apache Spark MLlib — це бібліотека машинного навчання (ML) Spark. Його мета зробити практичне машинне навчання масштабованим і простим.

SparkSession – це точка входу для функціональності Spark. Найважливішим кроком будь-якої програми драйвера Spark є створення SparkSession. Це дозволяє програмі Spark отримати доступ до Spark Cluster за допомогою диспетчера ресурсів. Менеджером ресурсів може бути один із цих трьох – Spark Standalone, YARN, Apache Mesos.

RDD – це незмінна розподілена колекція елементів ваших даних, яка була основною для користувачів Spark, у перших версіях. По суті, RDD —, розділених між вузлами вашого кластера, яка може працювати паралельно з низькорівневим API, який пропонує перетворення та дії.

DataFrame — це інтегрована структура даних із простим у використанні API для спрощення розподіленої обробки великих даних. DataFrame доступний для мов програмування загального призначення, таких як Java, Python і Scala. Це розширення Spark RDD.

DataSet— це структура даних у SparkSQL, яка є строго типізованою і є відображенням реляційної схеми. Він представляє структуровані запити з декодерами реляційних схем. Це розширення Dataframe API. Dataset забезпечує як безпеку типу, так і об'єктно-орієнтований інтерфейс програмування. Можна використовувати Dataset в своєму коді, починаючи зі Spark 1.6. На рис. 2.1 можна побачити відмінності трьох основних структурах даних у Apache Spark.

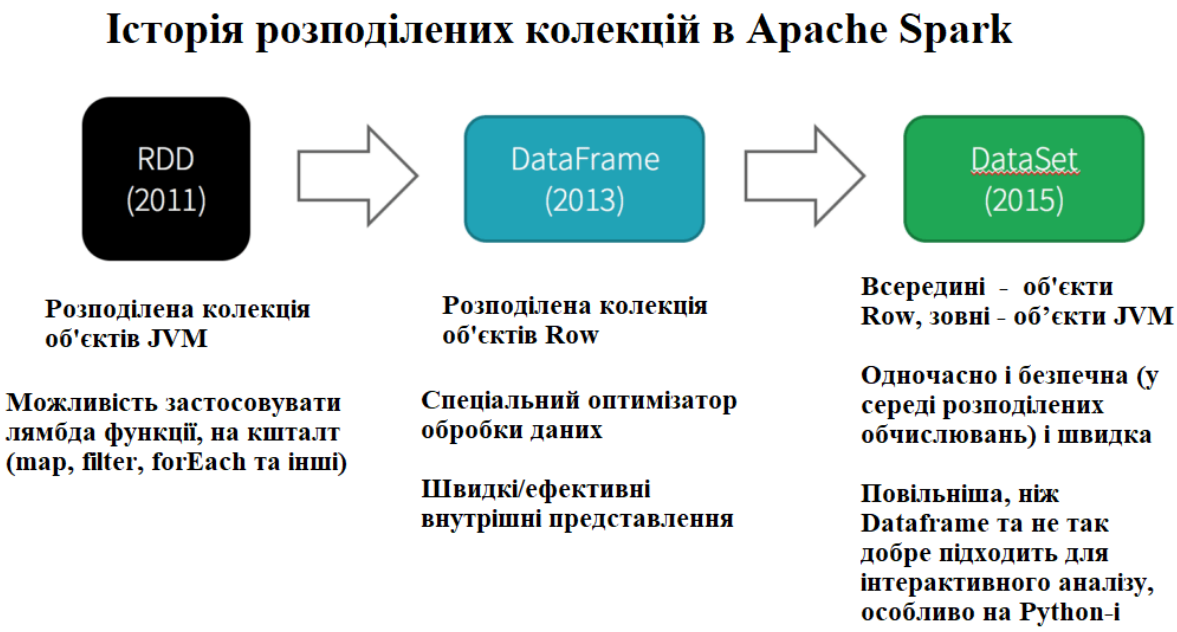


Рисунок 2.1 Історія розвитку колекцій

Tuple (або кортеж) – це впорядкований набір значень. Є у мовах програмування, таких як Scala, Python, Lisp, Prolog та інших. Роздільником для кожного значення часто є кома (залежно від правил конкретної мови). Зазвичай кортеж як тип даних використовується для передачі рядка параметрів з однієї програми в іншу та для представлення набору атрибутів значень у реляційній базі даних. У деяких мовах кортежі можуть бути вкладені в інші кортежи в дужках або дужках або інших роздільниках. Кортежи можуть містити суміш інших типів даних.

Кластеризація даних – це процес, який розбиває набір даних на групи таким чином, що точки даних у групі мають високу схожість у порівнянні один з одним, але дуже відрізняються від точок в інших групах.

Алгоритм кластеризації K-середніх обчислює центроїди і повторює, поки не буде знайдено оптимальний центроїд. Заздалегідь відомо, скільки існує кластерів. Він також відомий як алгоритм плоскої кластеризації. Кількість кластерів, знайдених із даних методом, позначається літерою «K» у K-середніх. У цьому

методі точки даних приписуються кластерам таким чином, щоб сума квадратів відстаней між точками даних і центроїдом була якомога меншою. Важливо відзначити, що зменшення різноманітності всередині кластерів призводить до більшої кількості ідентичних точок даних у тому самому кластері.

JDK (Java development kit) 11— це реалізація мови Java з відкритим вихідним кодом версії 11 платформи Java SE (Standard Edition). JDK 11 став загальнодоступним 25 вересня 2018 року.

Spring Framework — це фреймворк для створення веб-додатків, який надає контейнер для ін'єкції залежностей, з декількома зручними шарами (думайте: доступ до бази даних, проксі, аспектно-орієнтоване програмування, RPC (Remote procedure call), веб-фреймворк MVC (Model, View, Controller)). Це допомагає створювати програму Java швидше та зручніше. На рис. 2.2 зображено модулі фреймворку Spring.

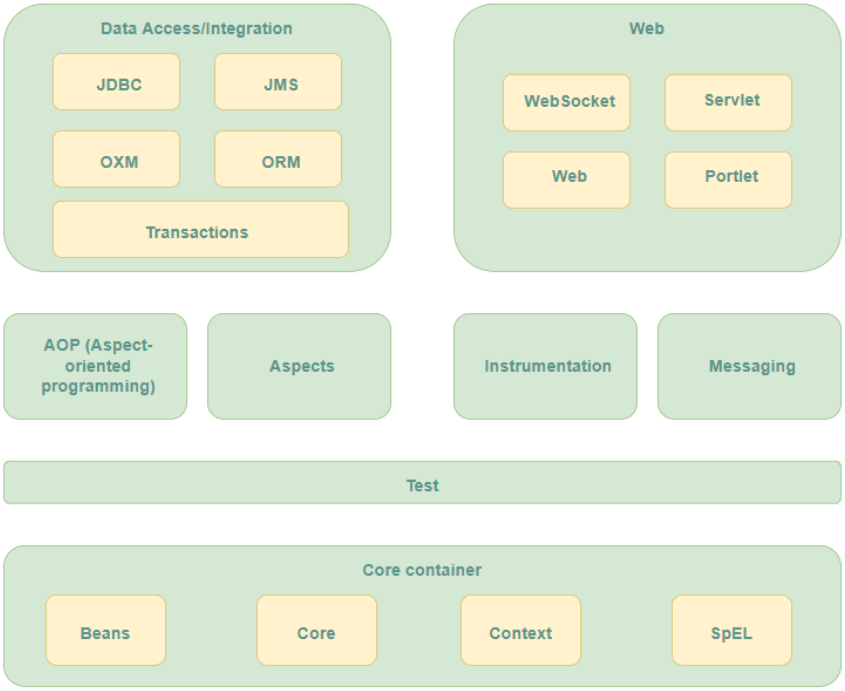


Рисунок 2.2 Модулі Spring

Spring Boot — це фреймворк Java з відкритим кодом, який був розроблений Pivotal у 2014 році і спрощує завдання розгортання корпоративних веб-додатків, написаних на Java. Це проект, побудований на основі фреймворку Spring, який забезпечує ефективний спосіб налаштування та запуску програм. Spring Boot складається з Spring Framework і вбудованих серверів. Оскільки він використовує конвенцію щодо конфігураційного програмного забезпечення, він не вимагає конфігурації XML.

Spring Boot був створений для досягнення наступних цілей:

- Спрощення процесу розробки додатків на Spring.
- Уникнення застарілої XML конфігурації у Spring.
- Скорочення часу розробки, за рахунок зменшення кількості необхідних операторів імпорту.

Apache Maven – це інструмент керування збіркою для Java. Спрощена модель конфігурації на основі XML від Maven дозволяє розробникам швидко описувати або розуміти контури будь-якого проекту на основі Java. Maven також підтримує розробку на основі тестування, довгострокове обслуговування проекту, його декларативна конфігурація та широкий спектр плагінів роблять його популярним варіантом для CI/CD.

MongoDB — це NoSQL база даних з відкритим вихідним кодом. Як нереляційна база даних, вона може обробляти структуровані, напівструктуровані та неструктуровані дані. Він використовує нереляційну, документно-орієнтовану модель даних і неструктуровану мову запитів. MongoDB використовує формат зберігання документів під назвою BSON, який є двійковою формою JSON (JavaScript Object Notation), яка може вмістити більше типів даних.

BSON – це двійкове кодування серіалізації JSON-подібних документів. Як і JSON, BSON підтримує вбудовування документів і масивів в інші документи та масиви.

*MongoDB Spark Connector (*далі - *MSC)* - це модуль, який забезпечує інтеграцію між MongoDB і Apache Spark. За допомогою MSC ви маєте доступ до всіх бібліотек Spark для використання з наборами даних MongoDB: набори даних для аналізу за допомогою SQL (отримавши переваги від автоматичного виведення схеми), потокової передачі, машинного навчання та графічних API.

URI (Uniform Resource Identifier)— це послідовність символів, що використовуються для ідентифікації певного ресурсу. Це дає можливість для взаємодії представлення ресурсу по мережі за допомогою певних протоколів.

Thymeleaf - це сучасний механізм шаблонів Java на стороні сервера як для веб-середовища, так і для автономного середовища. Основне використання Thymeleaf – впровадження шаблонів у робочий процес розробки – HTML, який може правильно відображатись у браузерах.

Docker — це відкрита платформа для контеризація, доставки та запуску програм. Docker дає змогу відокремити програми від інфраструктури, щоб можна було швидко користуватись програмним забезпеченням. За допомогою Docker можна керувати своєю інфраструктурою так само, як і програмами.

Docker Compose — це інструмент, який був розроблений, щоб допомогти визначати багатоконтейнерні додатки та використовувати їх разом, у одному загальному «контейнері контейнерів». За допомогою Compose ми можемо створити

файл YAML, щоб визначати служби, і за допомогою однієї команди можемо все розгорнути або зруйнувати.

Висновки до розділу 2

Опис деяких термінів RDD, DataFrame, Dataset побудований за допомогою сервісу (databricks.com). У даному розділі освітлені основні терміни та скорочення. Окрім термінів самого фреймворку описані також інші інструменти: мова імплементації проекту, інструмент контейнеризації, NoSQL скбд.

3. ОПИС АРХІТЕКТУРИ ПРОЕКТУ

Тренувальний стенд розроблений у вигляді веб-додатку. У якості фреймворку реалізації було обрано – Spring Boot + Spring MVC.

MVC (Model View Controller) – це одна з класичних архітектур побудови веб-додатків. Складається з трьох «рівнів»:

- **Model** або рівень даних, відповідає за збереження даних, у стенді рівень представлений у вигляді POJO (Plain Old Java Objects);
- **View** або рівень перегляду. Він фактично створює для користувача інтерфейс користувача або інтерфейс користувача. У стенді представлений у вигляді файлів шаблонізатору Thymeleaf;
- **Controller** рівень, який забезпечує взаємозв'язок між рівнями перегляду та даних, він діє як посередник. Контролер не повинен турбуватися про обробку логіки даних, він просто вказує моделі, що робити. Отримавши дані від моделі, він обробляє їх, а потім бере всю цю інформацію, яку надсилає до представлення та пояснює, як це представити користувачеві. У стенді всі контролери представлені у якості класів у пакеті ***src/main/java/com/diploma/demo/controller***.

Інші UML діаграми описують функціональність системи, діаграми компонентів використовуються для моделювання компонентів, які допомагають створити цю функціональність. Саме тому, у якості візуалізації схеми імплементації функціоналу обраний цей вид діаграм.

Архітектура фізичної імплементації проекту (у коді) зображена на діаграмах компонентів (рис. 3.1-3.4). Кожна з діаграм побудована навколо основного контролеру Синім кольором позначені java-класи, зеленим – конфігураційні файли, червоним – файли з вхідними даними, жовтим – шаблони Thymeleaf сторінок.

Висновки до розділу 3

У цьому розділі описана архітектура проекту, фізична реалізація класів, шаблонів та конфігураційних файлів представлена у діаграмах компонентів, кожна з яких відповідає одній з задач стенду.

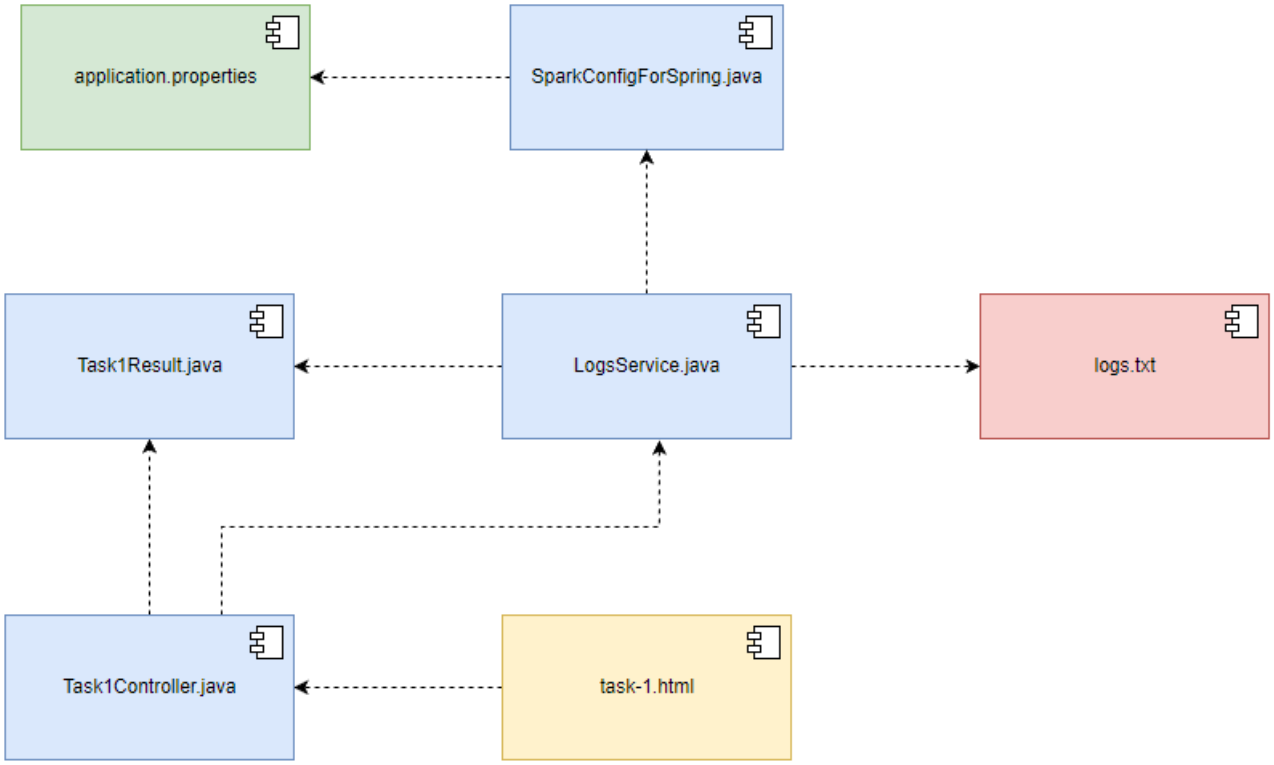


Рисунок 3.1 Діаграма компонентів для першої задачі, побудована навколо LogsService та Task1Controller

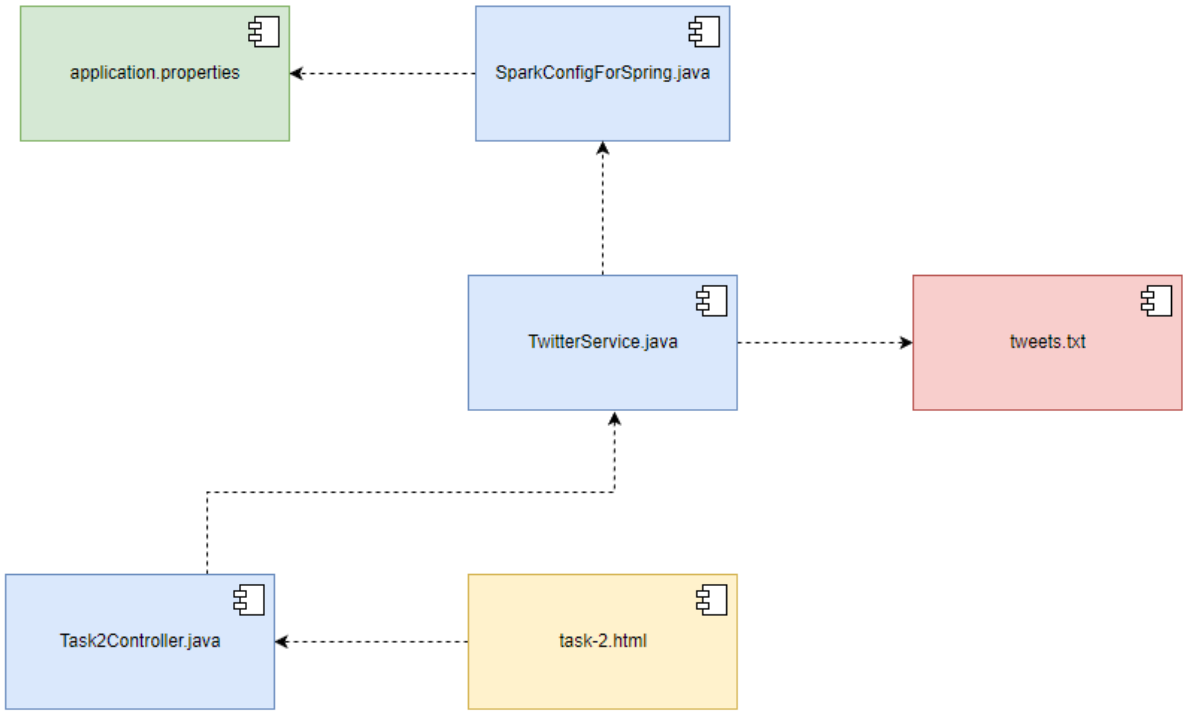


Рисунок 3.2 Діаграма компонентів для другої задачі, побудована навколо TwitterService та Task2Controller

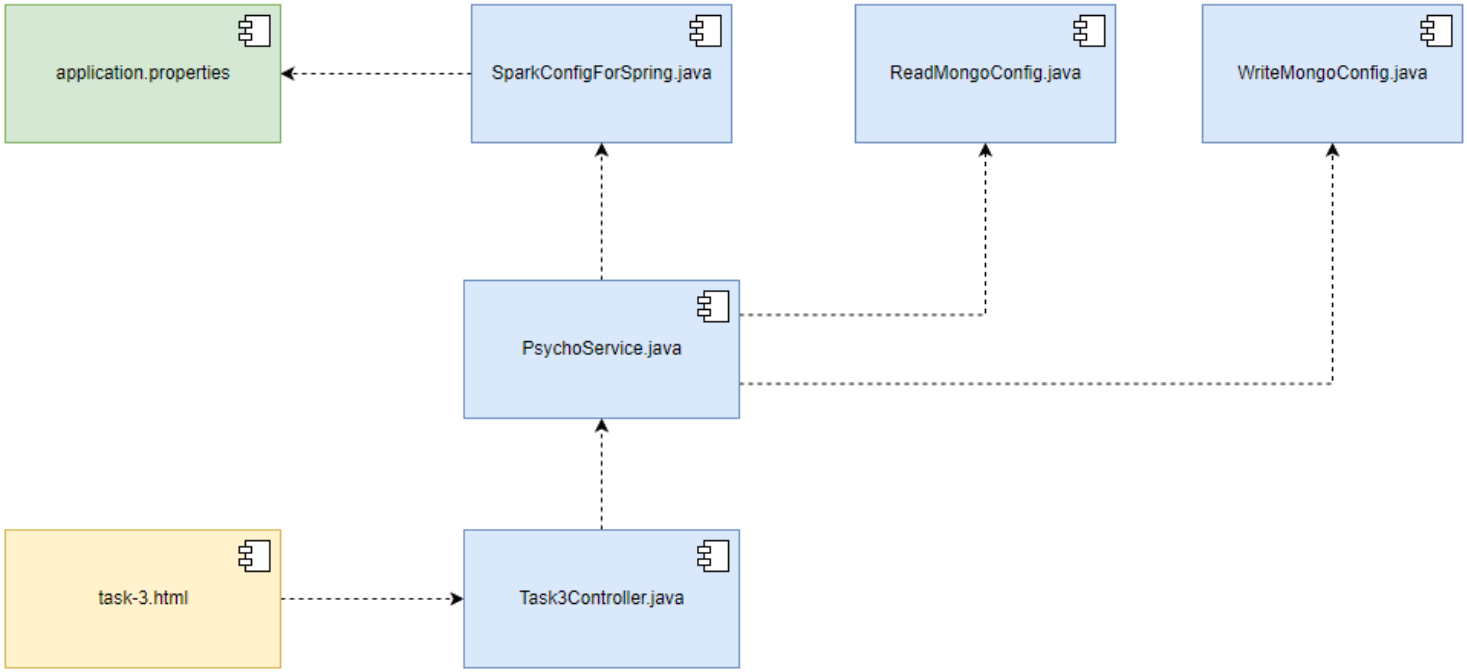


Рисунок 3.3 Діаграма компонентів для прешої частини (обробка даних) третьої задачі, побудована навколо PsychoService та Task3Controller

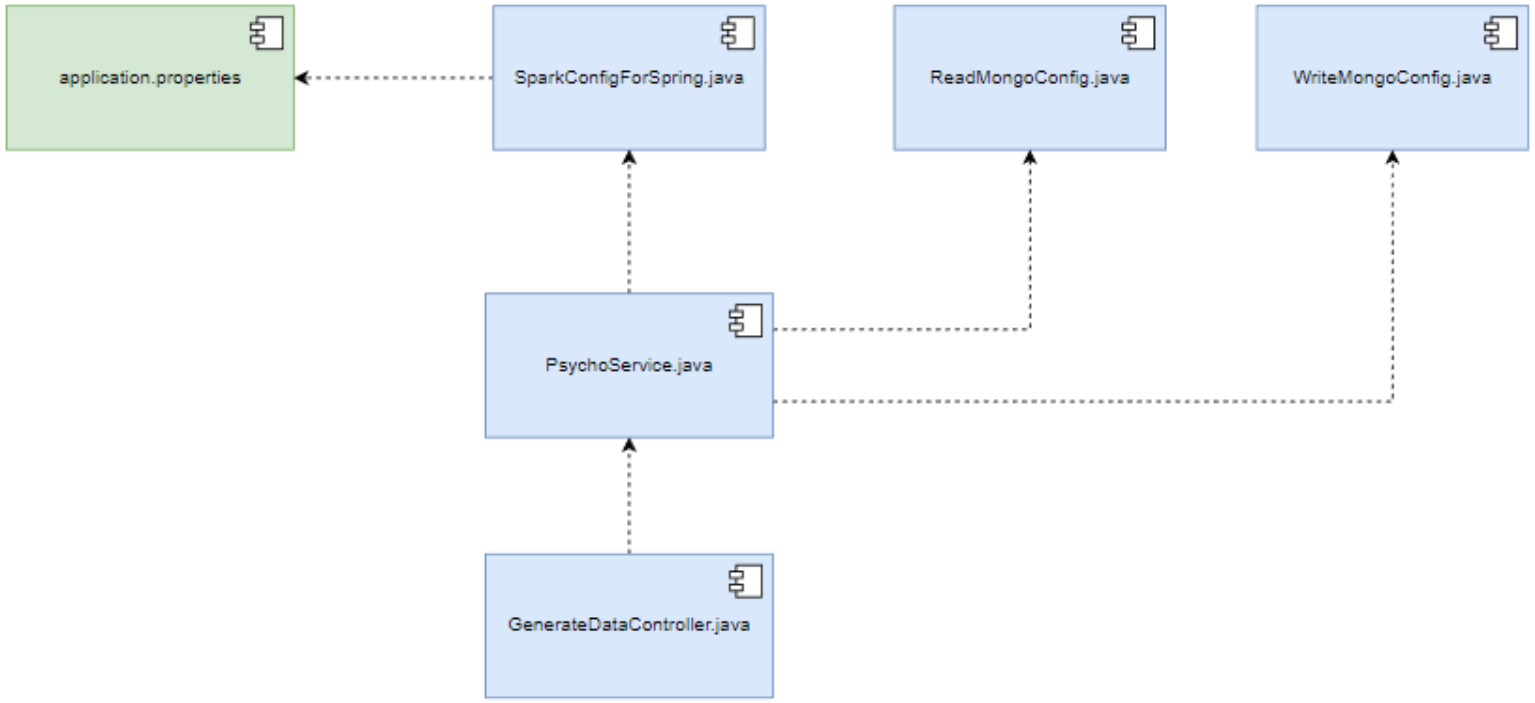


Рисунок 3.4 Діаграма компонентів для другої частини (генерація тестових даних) третьої задачі, побудована навколо PsychoService та GenerateDataController

4. ЗОВНІШНЄ ПРОЕКТУВАННЯ

Навчальний стенд із демонстрацією використання Apache Spark у вирішенні Big Data задач, має демонструвати роботу модулів фреймворку: Spark Core, SparkSQL, Spark MLlib. У якості задач обрано три: обробка логів серверу, обробка даних соціальної мережі Twitter, обробка результатів тесту на тип особистості BigFive.

Студент, використовуючи вирішення цих задач, може розглянути приклади роботи різних модулів фреймворку Apache Spark.

Навчальний стенд є безкоштовним і призначений для використання в навчанні студентів курсу «Бази даних».

Функціональне призначення – програмний продукт має демонструвати вирішення різних типів Big Data задач.

Експлуатаційне призначення – за допомогою програмного продукту виконується підготовка з дисципліни «Бази даних».

4.1 Вимоги до функціональних характеристик

Програмний продукт повинен демонструвати вирішення таких Big Data задач:

– Обробка логів серверу, пошук кількості стрічок з помилками та попередженнями. З використанням Spark Core.
– Обробка даних соціальної мережі Twitter, пошук кількості хештегів, пов’язаних з війною в Україні за допомогою Spark Core. Потрібно брати тільки перші 100 повідомлень користувача.
– Обробка результатів тесту на тип особистості BigFive за допомогою модуля Spark MLlib.

Вимоги до вхідних даних

Задача обробки логів серверу. На вхід подаються текстові файли серверу (див. додаток А):

– файл складається з текстових рядків, розділених \n;
– рядки зі словами ERROR – зараховуються до помилок;
– рядки зі словами WARN – зараховуються до попереджень;

Задача обробки даних соціальної мережі Twitter, окреме повідомлення є:

– текстовий рядок;
– хєштег – підстрока без роздільників, яка починається із символа «#».

Задача обробки результатів тесту на тип особистості. Вхідними даними будуть JSON файли, кожен з яких буде містити об’єкт з 5 параметрами (див. додаток з описом вхідних даних).

Вимоги до вихідних даних

Задача обробки логів серверу:

– кількість рядків з попередженнями має бути цілим числом;
– кількість рядків з помилками має бути цілим числом.

Задача обробки даних соціальної мережі Twitter:

– кількість хєштегів має бути цілим числом.

Задача обробки результатів тесту на тип особистості:

– номер кластеру має бути від 0 до 4;
– кожен з п’яти параметрів тесту має бути цілим числом від 1 до 5.

Вихідними даними є

Задача обробки логів серверу:

– кількість рядків з помилками;
– кількість рядків з попередженнями.

Задача обробки даних соціальної мережі Twitter:

– кількість хєштегів;
– назва хештегу.

Задача обробки результатів тесту на тип особистості:

– Результати кластеризації тестів алгоритмом к-середніх.

Опис функціональних можливостей представлений у діаграмі прецедентів (рис. 4.1). Акторами на даній діаграмі є цільові користувачі – студенти та викладачі.

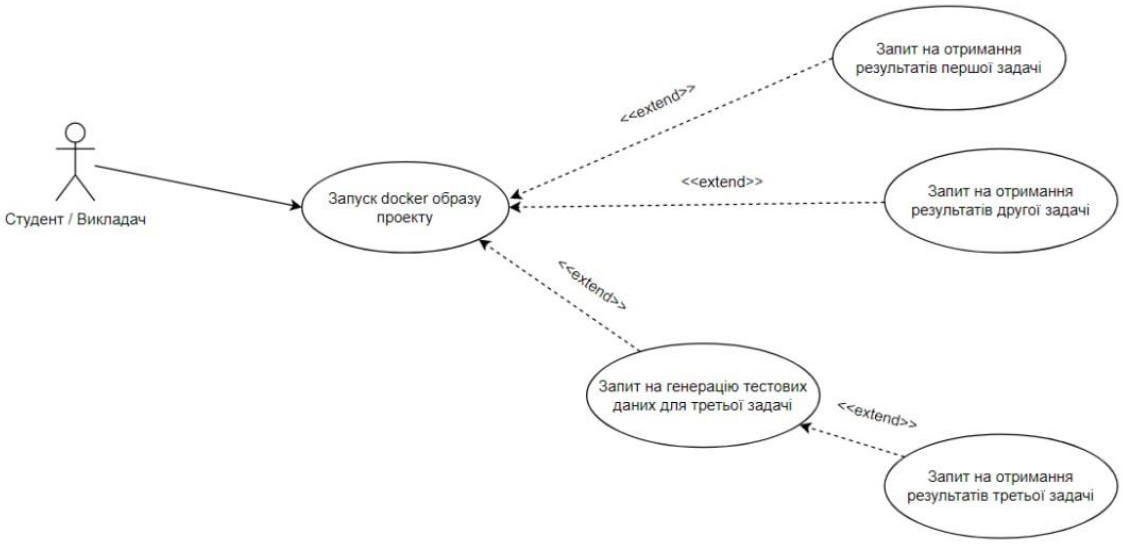


Рисунок 4.1 Діаграма прецедентів (use case) проекту

Висновки до розділу 4

У даному розділі описані функціональні та експлуатаційні вимоги. Описана суть задач та вимоги до вхідних і вихідних даних. Додатково, у якості опису функціональних можливостей додана діаграма прецедентів.

5. ВНУТРІШНЄ ПРОЕКТУВАННЯ ТА НАПИСАННЯ ПРОГРАМИ

Як було зазначено вище, проект написаний із використанням патерну MVC. Структура коду проекту має вид стандартного проекту з використанням Spring Boot та Java. Опис пакетів (папок з класами Java, конфігураційними файлами та шаблонами):

- /src/main/java/com/diploma/demo/ - загальний пакет, який містить весь Java-код проекту;
- /src/main/resources/ - загальний пакет, який знає всі шаблони та файли конфігурації;
- /src/main/resources/templates/ - пакет шаблонів;
- /src/main/java/com/diploma/demo/config/ - пакет конфігураційних файлів на Java;
- /src/main/java/com/diploma/demo/controller/ - пакет контролерів;
- /src/main/java/com/diploma/demo/model/ - пакет моделей;
- /src/main/java/com/diploma/demo/service/ - пакет сервісів, сервіси мають всю логіку, яку використовують контролери.

При проектуванні акцент було зроблено саме на внутрішній устрій стенду, а не на форми, тому форми виведення результатів зроблені за допомогою звичайного html, шаблонізатору thymeleaf та примітивних css стилей.

Загальні зв’язки класів показані на діаграмі класів (рис. 5.1).

Висновки до розділу 5

У цьому розділі було розглянуто внутрішнє проектування стенду, розглянуті пакети коду, в яких знаходиться імплементація. У якості схеми фізичної реалізації задач стенду представлена діаграма класів.

Рисунок 5.1. Діаграма класів навчального стенду

6. ВІДЛАГОДЖЕННЯ СИСТЕМИ

У якості методу знаходження і усунення проблем було обране відлагодження (debugging). Відлагодження програми полягає в тестуванні вручну за допомогою тестового набору, при роботі з яким було допущено помилку. Незважаючи на ефективність, метод не вдасться використовувати для великих програм або програм зі складними обчисленнями. Програма стенду не є складною або великою, саме тому було обрано цей метод.

Java Exceptions (винятки) – механізм, який дозволяє виявляти події, що відбуваються підчас неправильного потоку інструкцій програми.

Для більш детального опису того, які саме Java Exceptions виникли та як виплавлені дивіться у таблицю 6.1.

Таблиця 6.1 - Список винятків, які були відлагоджені під час ручного тестування.

Номер помилки	Назва винятку	Опис ситуації, при якому він виник	Опис дій по усуненню проблеми
1	2	3	4
1	ClassCastException	Спроби невірного перетворення типу RDD у тип Dataset у TwitterService.java	Використання окремого вбудованого в Java механізму перетворення типів
2	IllegalArgumentException	Спроба передати у один із методів PsychoService.java параметр типу-обгортки	Заміна параметру типу обгортки на самий тип параметру
3	MongoSocketOpenException	Спроба під’єднатися до docker контейнеру з mongodb по хосту, який використовувався при локальному тестуванні системи - localhost	Хостом при приєднанні до Docker Image є назва цього Image, отже було localhost було замінено на mongodb у конфігураційному файлі application.propeties
4	InvalidPathException	Спроба зчитати файл logs.txt та tweets.txt без зміни шляху до них.	Під час створення інфраструктури файли були переміщені у корень папки докер контейнеру та не

			знаходились там, де при вони знаходились при локальному тестуванні. Заміна шляху.
5	NullPointerException	Виникла при спробі звернутись до даних ендпоінту третьої задачі, який пов’язаний з обробкою даних, без їх попереднього створення іншим ендпоінтом.	Створення попередження про те, що потрібно виконувати ендпоінти третьої задачі послідовно.

Висновки до розділу 6

У цьому розділі було розглянуто відлагодження або debugging стенду. Також представлена таблиця зі списком винятків, які були відлагоджені під час цього процесу.

7. ОПИС ІНФРАСТРУКТУРИ

Під інфраструктурою маються на увазі ті інструменти, за допомогою яких у проекті є можливість розгортати цей самий проект у різних середовищах та різних комп'ютерах.

Проблему сумісності з іншими середовищами було вирішено використанням контейнеризатору Docker та його модуля для багатокомпонентних програм Docker Compose.

У Docker існує таке поняття як Image. Docker Image — це шаблон для створення контейнерів Docker. Це виконуваний пакет, який містить все необхідне для запуску програми: код, середовище виконання, бібліотеки, змінні середовища та файли конфігурації.

Інфраструктура проекту описується у двох файлах:

– docker-compose.yml – файл з набором команд до Docker Compose, описом всіх Image, які використовуються у проекті;
– Dockerfile – набір команд для створення із виконуваного jar-архіву тренувального стенду Docker Image, який потім Compose-ом буде покладений у загальну інфраструктуру.

Загальну схему інфраструктури стенду можна побачити на рисунку 7.1.

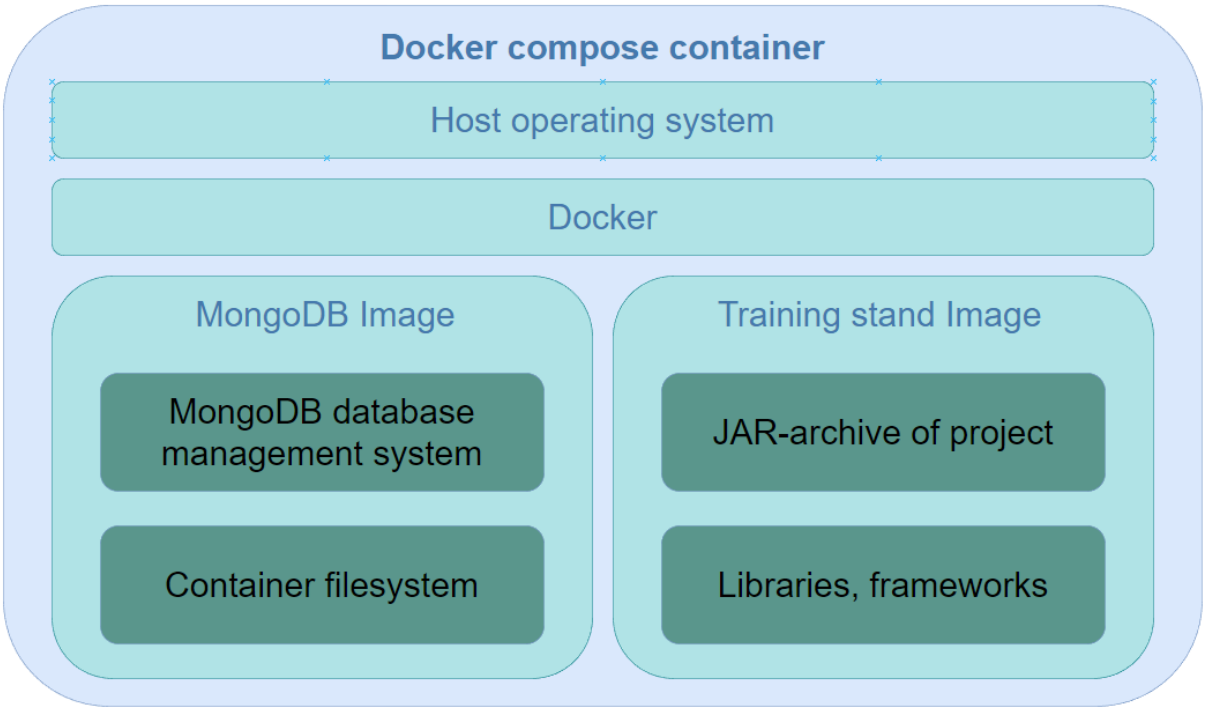


Рисунок 7.1. Організація інфраструктури проекту.

7.1 Запуск проекту

1. Встановити на Ваш комп’ютер Docker, в залежності від операційної системи, що на ньому встановлена:

-інструкція для Windows
https://docs.docker.com/desktop/windows/install/
-інструкція для Ubuntu
https://docs.docker.com/engine/install/ubuntu/

2. Необхідно завантажити образ проекту з Docker Hub (це сервіс, який надає Docker для пошуку образів контейнерів) виконавши команду в консолі

docker pull kotero/training-stand:latest

3. Також, для запуску команд Docker Compose потрібно встановити необхідні плагіни:

-для Windows
https://docs.docker.com/compose/install/compose-plugin/#install-compose-on-windows-server
Якщо Ви завантажили Docker Desktop для Windows, завантажувати окремо його не треба, цей плагін в нього входить.
-для Ubuntu
https://docs.docker.com/compose/install/compose-plugin/#installing-compose-on-linux-systems

4. Перейти за допомогою терміналу вашої ос (Windows PowerShell або Ubuntu Terminal) у папку, у якій знаходиться Docker Image, який Ви завантажили в першому пункті.
5. Запустити Image за допомогою команди терміналу

docker-compose up

Додатковий приклад:
https://docs.docker.com/compose/gettingstarted/

6. Зараз веб-додаток стенду та MongoDB запущені, ми можемо посилати запити на сервер та дивитись їх результати.
7. Відкрити будь-який браузер. Та у рядку URL ввести:

– для отримання результатів задачі 1
http://localhost:8080/api/get-warnings-and-errors
– для отримання результатів задачі 2
http://localhost:8080/api/get-trending-hashtags
– перейти за посиланням та відправити запит за таким URL:
http://localhost:8080/api/generate-tests

Це згенерує тестові данні у базі даних для коректного запиту до ендпоінту третьої задачі. Без виконання цього пункту, наступний працювати не буде.

– для отримання результатів завдання 3 введіть у стрічці браузера:

http://localhost:8080/api/get-aggregated-tests

Приклади виконання запитів при запущеному в Docker стенді зображені на рисунках 7.2-7.4.

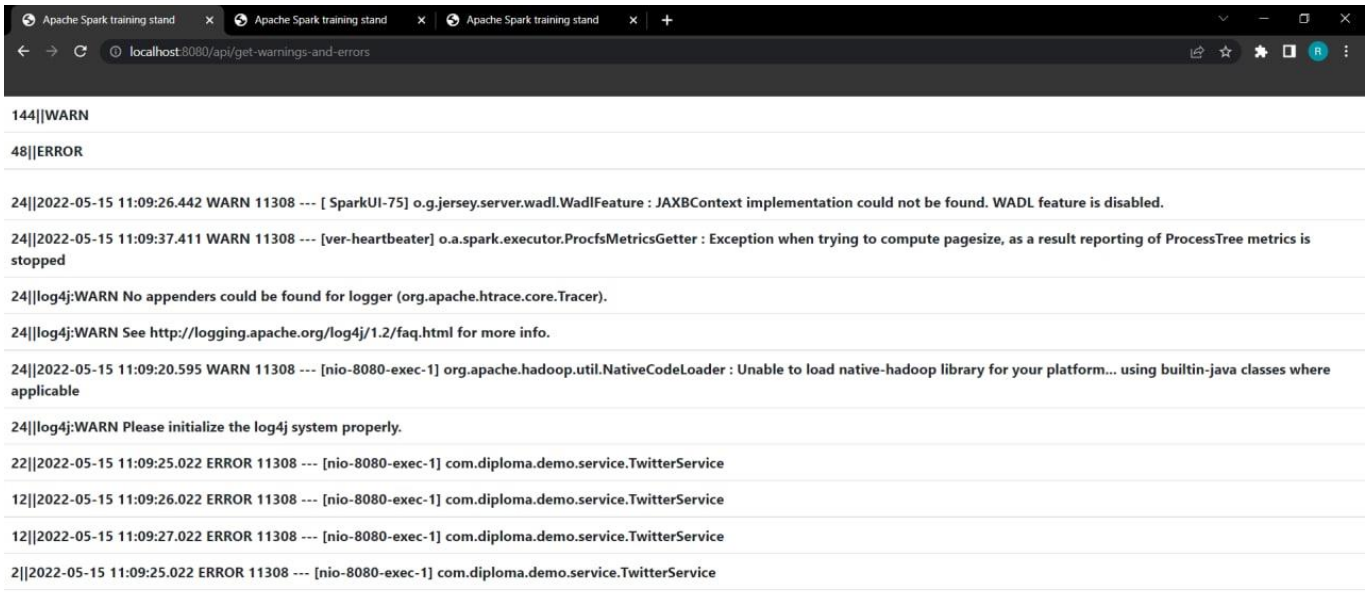


Рисунок 7.2 Результати роботи першого завдання.

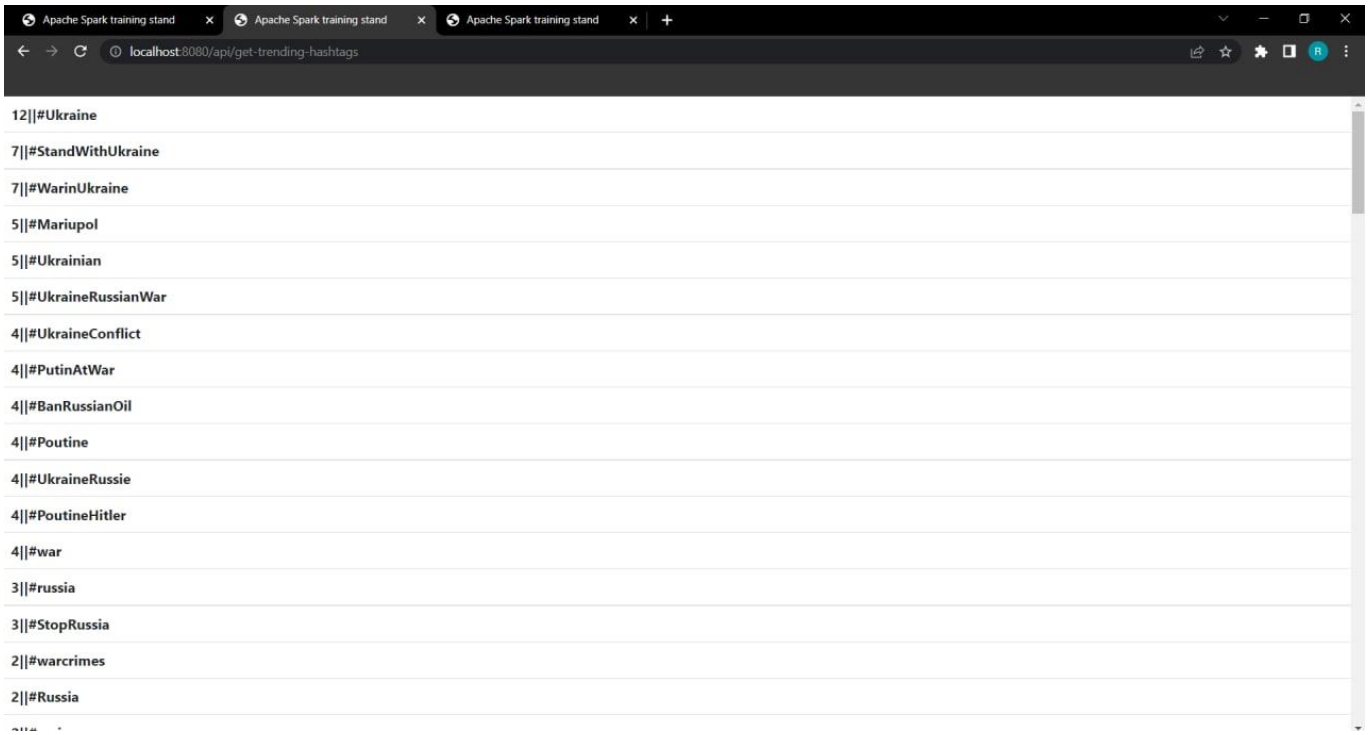
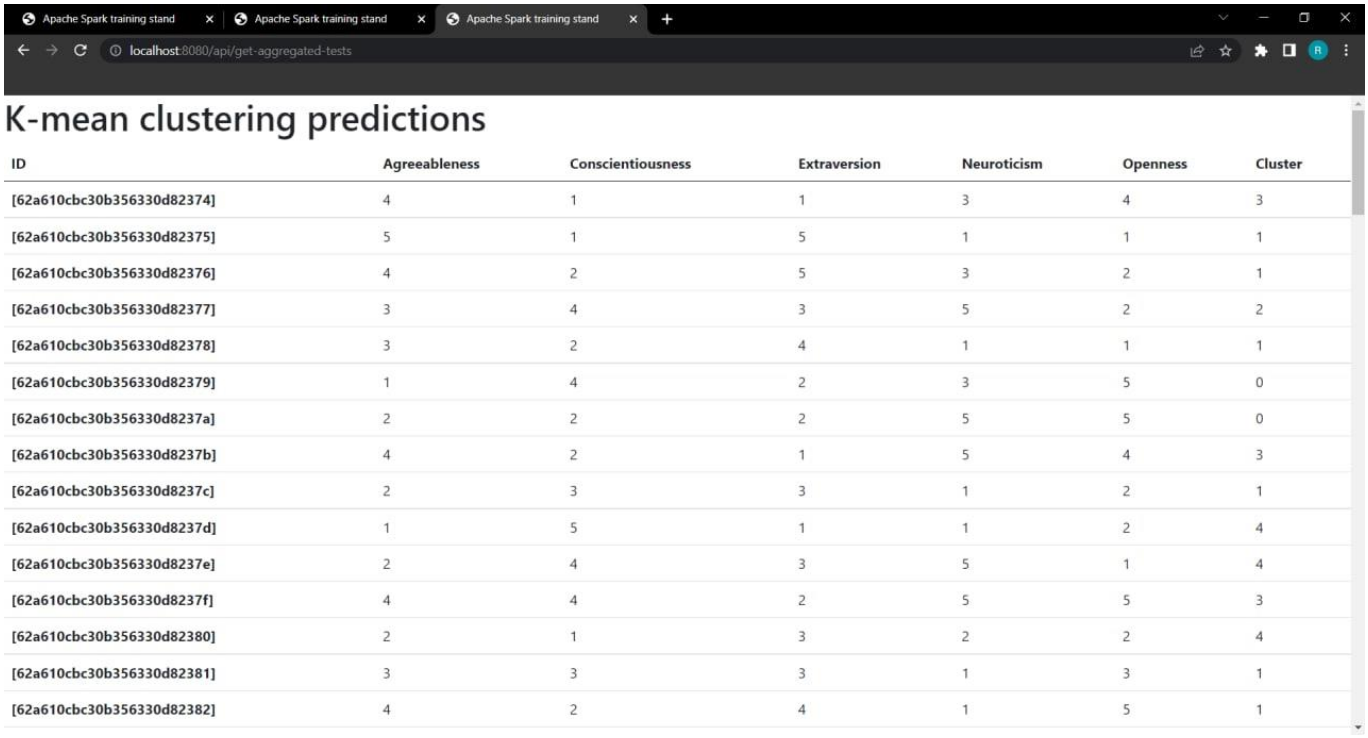


Рисунок 7.3 Результати роботи другого завдання.



ID	Agreeableness	Conscientiousness	Extraversion	Neuroticism	Openness	Cluster
[62a610cbc30b356330d82374]	4	1	1	3	4	3
[62a610cbc30b356330d82375]	5	1	5	1	1	1
[62a610cbc30b356330d82376]	4	2	5	3	2	1
[62a610cbc30b356330d82377]	3	4	3	5	2	2
[62a610cbc30b356330d82378]	3	2	4	1	1	1
[62a610cbc30b356330d82379]	1	4	2	3	5	0
[62a610cbc30b356330d8237a]	2	2	2	5	5	0
[62a610cbc30b356330d8237b]	4	2	1	5	4	3
[62a610cbc30b356330d8237c]	2	3	3	1	2	1
[62a610cbc30b356330d8237d]	1	5	1	1	2	4
[62a610cbc30b356330d8237e]	2	4	3	5	1	4
[62a610cbc30b356330d8237f]	4	4	2	5	5	3
[62a610cbc30b356330d82380]	2	1	3	2	2	4
[62a610cbc30b356330d82381]	3	3	3	1	3	1
[62a610cbc30b356330d82382]	4	2	4	1	5	1

Рисунок 7.4 Результати роботи третього завдання.

Висновки до розділу 7

У цьому розділі була описана інфраструктура проекту, інструменти за допомогою яких вона була створена а також додана інструкція для запуску проекту потенційними користувачами.

Опис доповнений схемою організації інфраструктури, а інструкція – прикладами виконання запитів на тренувальному стенді.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вивчаємо Spark. Блискавичний аналіз даних: Халдеї Карау, Енді Конвінські, Патрік Венделл, Матей Захар: ДМК Прес, 2015. – 304 с.
2. High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark: Holden Karau, Rachel Warren: 2017. – 358 с.
3. Apache Spark for Data Science Cookbook: Padma Priya Chitturi: 2016. – 392 с.
4. Apache Spark 2.x for Java Developers: Explore big data at scale with Apache Spark 2.x Java API: Surav Gulati, Sumit Kumar: 2017. - 350 с.
5. https://github.com/apache/spark/tree/master/examples/src/main/python
6. https://github.com/apache/spark/tree/master/examples/src/main/java/org/apache/spark/examples
7. https://github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples
8. https://github.com/apache/spark/tree/master/examples/src/main/r

ДОДАТКИ

Опис вхідних даних

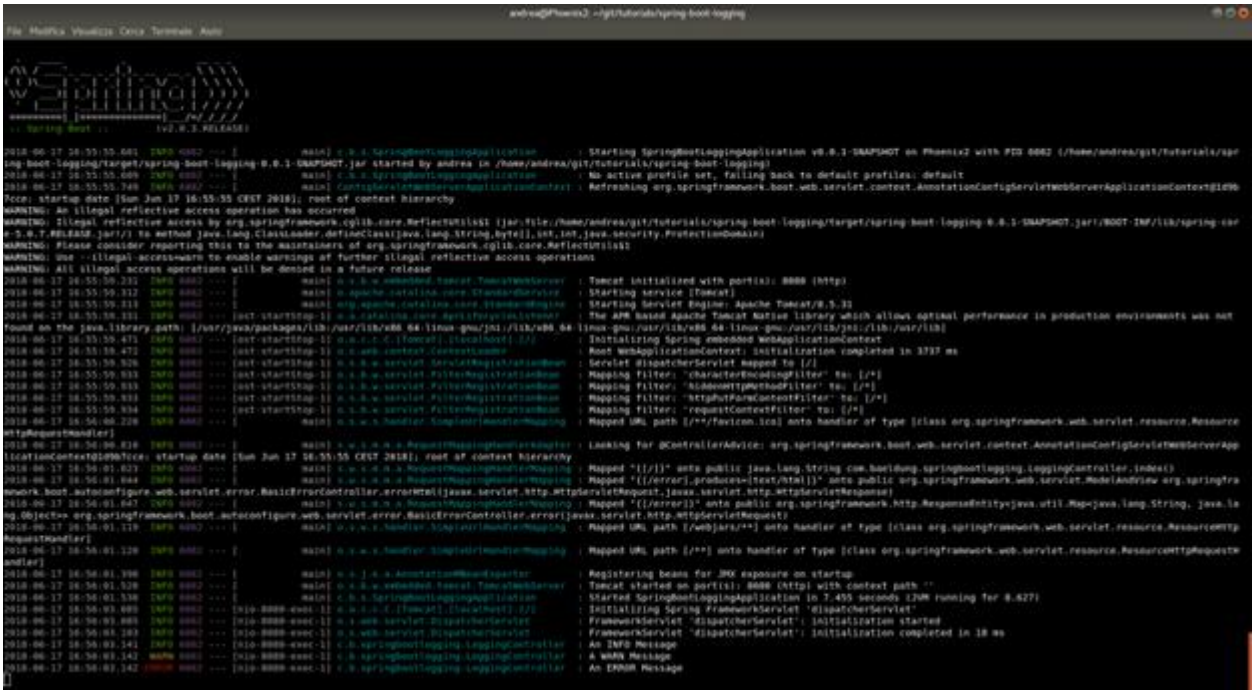


Рисунок А.1 Приклад формату вхідних даних першої задачі

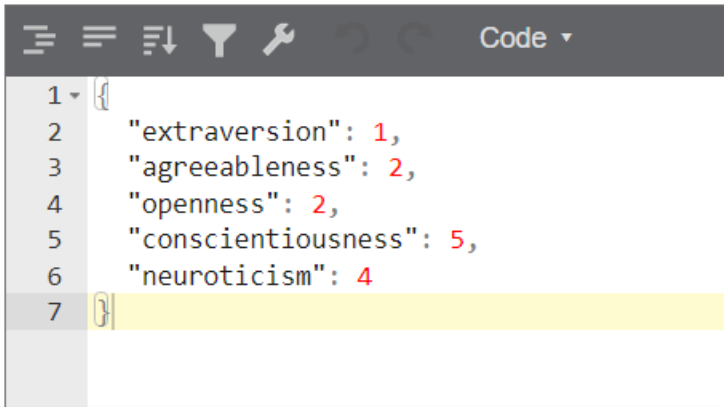


Рисунок А.2 Приклад формату вхідних даних третьої задачі

Текст програми

Окрім цього додатку весь код проекту можна знайти за посиланням https://github.com/kote-ro/diploma-project

Файл Dockerfile

```
FROM maven:3.8-openjdk-11-slim as builder

WORKDIR /app

COPY . .

RUN mvn clean package -DskipTests



FROM openjdk:11

WORKDIR /app

ADD data/* /app/

COPY --from=builder /app/target/diploma-project-0.0.1-SNAPSHOT.jar /app/app.jar

ENTRYPOINT ["java", "-jar","/app/app.jar"]
```

Файл docker-compose.yml

```
version: "3.7"


services:

  mongodb:

    image: "mongo:4.4.3"

    restart: always

    expose:

      - 27017

    ports:

      - 27017:27017

    networks:

      - connector

    environment:

      MONGO_INITDB_ROOT_USERNAME: mongo

      MONGO_INITDB_ROOT_PASSWORD: mongo
```

```
  spark_stand:
    build: .
    restart: always
    networks:
      - connector
    ports:
      - 8080:8080
    depends_on:
      - mongodb
    links:
      - mongodb


networks:
  connector:
```

Файл pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.6.7</version>
                <relativePath/>
        </parent>
        <groupId>com.dimploma</groupId>
        <artifactId>diploma-project</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>diploma-project</name>
        <description>diploma-project</description>
```

```
    <properties>
        <java.version>11</java.version>
    </properties>
    <dependencies>
<!--        Apache Spark-->
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.12</artifactId>
            <version>3.1.3</version>
            <exclusions>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-log4j12</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming_2.12</artifactId>
            <version>3.1.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_2.12</artifactId>
            <version>3.1.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-mllib_2.12</artifactId>
            <version>3.1.3</version>
```

```
            <scope>provided</scope>
        </dependency>
<!--        Spring-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-log4j12</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>slf4j-log4j12</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <version>2.6.7</version>
            <scope>test</scope>
        </dependency>
<!--        Thymeleaf-->
```

```
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-thymeleaf</artifactId>
                        <version>2.3.3.RELEASE</version>
                </dependency>
<!--            Janiono-->
                <dependency>
                        <groupId>org.codehaus.janino</groupId>
                        <artifactId>commons-compiler</artifactId>
                        <version>3.0.8</version>
                </dependency>
                <dependency>
                        <groupId>org.codehaus.janino</groupId>
                        <artifactId>janino</artifactId>
                        <version>3.0.8</version>
                </dependency>
<!--            Mongo Spark Connector-->
                <dependency>
                        <groupId>org.mongodb.spark</groupId>
                        <artifactId>mongo-spark-connector_2.12</artifactId>
                        <version>3.0.1</version>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
```

```
            </plugins>
        </build>

</project>
```

Файл application.properties

```
#Spring Boot configuration
spring.profiles.active=LOCAL
#Mongo configuration

#Spark configuration
spark.uri.local=local[*]
spark.uri.cluster=local[3]
spark.app.name=Apache Spark training stand
spark.mongodb.input.uri=mongodb://mongo:mongo@mongodb:27017/admin.inputData
spark.mongodb.output.uri=mongodb://mongo:mongo@mongodb:27017/admin.outputData
#Task 3 variables
amount.of.tests=1000
lower.value.limit=1
upper.value.limit=5
```

Файл task-1.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Apache Spark training stand</title>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet"
      integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
</head>
<body>
```

```
<div th:if="${errors != null}">
   <table class="table">
      <tr th:each="error : ${errors}">
         <th scope="row" th:text="${error}"></th>
      </tr>
   </table>
</div>
<div th:if="${warnings != null}">
   <table class="table">
      <tr th:each="warning : ${warnings}">
         <th scope="row" th:text="${warning}"></th>
      </tr>
   </table>
</div>
</body>
</html>
```

Файл task-2.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Apache Spark training stand</title>
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet"
      integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
</head>
<body>
<div th:if="${hashtags != null}">
   <table class="table">
      <tr th:each="hashtag : ${hashtags}">
         <th scope="row" th:text="${hashtag}"></th>
```

```
    </tr>
  </table>
</div>
</body>
</html>
```

Файл task-3.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Apache Spark training stand</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet"
      integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
</head>
<body>
<div th:if="${predictions != null}">
<table class="table">
  <h1>K-mean clustering predictions</h1>
  <thead>
  <tr>
    <th scope="col">ID</th>
    <th scope="col">Agreeableness</th>
    <th scope="col">Conscientiousness</th>
    <th scope="col">Extraversion</th>
    <th scope="col">Neuroticism</th>
    <th scope="col">Openness</th>
    <th scope="col">Cluster</th>
  </tr>
  </thead>
  <tbody>
```

```
      <tr th:each="prediction : ${predictions}">
        <th scope="row" th:text="${prediction.id}"></th>
        <td th:text="${prediction.agreeableness}"></td>
        <td th:text="${prediction.conscientiousness}"></td>
        <td th:text="${prediction.extraversion}"></td>
        <td th:text="${prediction.neuroticism}"></td>
        <td th:text="${prediction.openness}"></td>
        <td th:text="${prediction.cluster}"></td>
      </tr>
    </tbody>
</table>
</div>
</body>
</html>
```

Файл DiplomaProjectApplication.java

```
package com.diploma.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.data.mongo.MongoDataAutoConfiguration;
import org.springframework.boot.autoconfigure.mongo.MongoAutoConfiguration;

@SpringBootApplication(exclude = {MongoAutoConfiguration.class, MongoDataAutoConfiguration.class})
public class DiplomaProjectApplication {

	public static void main(String[] args) {
		SpringApplication.run(DiplomaProjectApplication.class, args);
	}
```

}

Файл TwitterService.java

```
package com.diploma.demo.service;


import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;

import org.apache.spark.sql.SparkSession;

import org.springframework.stereotype.Service;

import scala.Tuple2;


import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;


@Service

public class TwitterService {

   private final SparkSession spark;


   public TwitterService(SparkSession spark) {

      this.spark = spark;

   }


   public List<String> findTrendingHashtags(){

      JavaSparkContext sc = JavaSparkContext.fromSparkContext(spark.sparkContext());

      String path = "tweets.txt";


      JavaRDD<String> file = sc.textFile(path);

      List<Tuple2<Integer, String>> words = file

            .flatMap(line -> Arrays.asList(line.split(" ")).iterator())

            .filter(w -> w.startsWith("#") && !w.equals("#WARINUKRAINE"))
```

```
            .mapToPair(w -> new Tuple2<>(w, 1))
            .reduceByKey(Integer::sum)
            .mapToPair(Tuple2::swap)
            .sortByKey(false)
            .collect();

    return words.stream().map(x -> x._1 + "||" + x._2).collect(Collectors.toList());
  }
}
```

Файл PsychoService.java

```
package com.diploma.demo.service;

import com.diploma.demo.model.Task3Result;
import com.mongodb.spark.MongoSpark;
import com.mongodb.spark.config.WriteConfig;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.MapFunction;
import org.apache.spark.ml.clustering.KMeans;
import org.apache.spark.ml.clustering.KMeansModel;
import org.apache.spark.ml.feature.VectorAssembler;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Encoders;
import org.apache.spark.sql.Row;
import org.bson.Document;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
```

```
import java.util.logging.Logger;


@Service
public class PsychoService {
    private final JavaSparkContext jsc;
    private final WriteConfig writeConfig;
    private final Logger logger = Logger.getLogger(PsychoService.class.getName());
    @Value("${amount.of.tests}")
    private Integer amountOfTests;
    @Value("${lower.value.limit}")
    private Integer lowerValueLimit;
    @Value("${upper.value.limit}")
    private Integer upperValueLimit;

    public PsychoService(JavaSparkContext jsc, WriteConfig writeConfig) {
        this.jsc = jsc;
        this.writeConfig = writeConfig;
    }

    public void generateData(){
        List<Document> tests = new ArrayList<>();

        for(int i = 1; i <= amountOfTests; i++){
            Document test = initData();
            tests.add(test);
        }

        JavaRDD<Document> documents = jsc.parallelize(tests);
        MongoSpark.save(documents, writeConfig);
```

```
    jsc.close();
}

public List<Task3Result> processData() {
    Dataset<Row> dataset = MongoSpark.load(jsc).toDF();

    VectorAssembler assembler = new VectorAssembler()
            .setInputCols(new String[]{"openness", "conscientiousness", "extraversion", "agreeableness", "neuroticism"})
            .setOutputCol("features");

    dataset = assembler.transform(dataset);

    KMeans kmeans = new KMeans().setK(5).setSeed(1L);
    KMeansModel model = kmeans.fit(dataset);
    Dataset<Row> predictions = model.transform(dataset);

    return beautifyResults(predictions);
}

private Document initData() {
    Integer openness = lowerValueLimit + (int) (Math.random() * upperValueLimit);
    Integer conscientiousness = lowerValueLimit + (int) (Math.random() * upperValueLimit);
    Integer extraversion = lowerValueLimit + (int) (Math.random() * upperValueLimit);
    Integer agreeableness = lowerValueLimit + (int) (Math.random() * upperValueLimit);
    Integer neuroticism = lowerValueLimit + (int) (Math.random() * upperValueLimit);

    Document test = new Document();
    test.append("openness", openness);
    test.append("conscientiousness", conscientiousness);
```

```
        test.append("extraversion", extraversion);
        test.append("agreeableness", agreeableness);
        test.append("neuroticism", neuroticism);
        return test;
    }

    private List<Task3Result> beautifyResults(Dataset<Row> results){
        return results
                .map((MapFunction<Row, Task3Result>) row ->
                        new Task3Result(row.get(0).toString(), row.get(1).toString(), row.get(2).toString(),
                                row.get(3).toString(), row.get(4).toString(), row.get(5).toString(), row.get(7).toString())
                        , Encoders.bean(Task3Result.class))
                .limit(100)
                .collectAsList();
    }

}
```

Файл LogsService.java

```
package com.diploma.demo.service;

import com.diploma.demo.model.Task1Result;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.SparkSession;
import org.springframework.stereotype.Service;
import scala.Tuple2;

import java.util.Arrays;
import java.util.List;
import java.util.logging.Logger;
```

```
import java.util.stream.Collectors;


@Service
public class LogsService {
    private final SparkSession spark;


    private final Logger logger = Logger.getLogger(this.getClass().getName());


    public LogsService(SparkSession spark) {
        this.spark = spark;
    }



    public Task1Result findWarningsAndErrors(){
        JavaSparkContext sc = JavaSparkContext.fromSparkContext(spark.sparkContext());
        String path = "logs.txt";


        JavaRDD<String> file = sc.textFile(path);
        List<Tuple2<Integer, String>> words1 = file
                .flatMap(line -> Arrays.asList(line.split("\n")).iterator())
                .filter(w -> w.contains("ERROR ") || w.contains("WARN "))
                .mapToPair(w -> (w.contains("ERROR ")) ? new Tuple2<>("ERROR ", 1) : new Tuple2<>("WARN ", 1))
                .reduceByKey(Integer::sum)
                .mapToPair(Tuple2::swap)
                .sortByKey(false)
                .collect();


        List<Tuple2<Integer, String>> words2 = file
                .flatMap(line -> Arrays.asList(line.split("\n")).iterator())
                .filter(w -> w.contains("ERROR ") || w.contains("WARN "))
```

```
            .mapToPair(w -> new Tuple2<>(w, 1))
            .reduceByKey(Integer::sum)
            .mapToPair(Tuple2::swap)
            .sortByKey(false)
            .collect();

    List<String> wordsStr1 = words1.stream().map(x -> x._1 + "||" + x._2).collect(Collectors.toList());
    List<String> wordsStr2 = words2.stream().map(x -> x._1 + "||" + x._2).collect(Collectors.toList());

    return new Task1Result(wordsStr1, wordsStr2);
  }
}
```

Файл Task1Result.java

```
package com.diploma.demo.model;

import java.util.List;

public class Task1Result {
  List<String> errors;
  List<String> warnings;

  public Task1Result(List<String> errors, List<String> warnings) {
    this.errors = errors;
    this.warnings = warnings;
  }

  public List<String> getErrors() {
    return errors;
  }
```

```
    public void setErrors(List<String> errors) {
        this.errors = errors;
    }

    public List<String> getWarnings() {
        return warnings;
    }

    public void setWarnings(List<String> warnings) {
        this.warnings = warnings;
    }
}
```

Файл Task3Result.java

```
package com.diploma.demo.model;

import java.io.Serializable;

public class Task3Result implements Serializable {
    private String id;
    private String agreeableness;
    private String conscientiousness;
    private String extraversion;
    private String neuroticism;
    private String openness;
    private String cluster;

    public Task3Result() {
    }

    public Task3Result(String id, String agreeableness, String conscientiousness, String extraversion, String neuroticism, String openness, String cluster) {
```

```
    this.id = id;
    this.agreeableness = agreeableness;
    this.conscientiousness = conscientiousness;
    this.extraversion = extraversion;
    this.neuroticism = neuroticism;
    this.openness = openness;
    this.cluster = cluster;
}

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getAgreeableness() {
    return agreeableness;
}

public void setAgreeableness(String agreeableness) {
    this.agreeableness = agreeableness;
}

public String getConscientiousness() {
    return conscientiousness;
}

public void setConscientiousness(String conscientiousness) {
```

```
    this.conscientiousness = conscientiousness;
}

public String getExtraversion() {
    return extraversion;
}

public void setExtraversion(String extraversion) {
    this.extraversion = extraversion;
}

public String getNeuroticism() {
    return neuroticism;
}

public void setNeuroticism(String neuroticism) {
    this.neuroticism = neuroticism;
}

public String getOpenness() {
    return openness;
}

public void setOpenness(String openness) {
    this.openness = openness;
}

public String getCluster() {
    return cluster;
}
```

```
    public void setCluster(String cluster) {
        this.cluster = cluster;
    }
}
```

Файл Task1Controller.java

```
package com.diploma.demo.controller;

import com.diploma.demo.model.Task1Result;
import com.diploma.demo.service.LogsService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/api")
public class Task1Controller {
    private final LogsService logsService;

    public Task1Controller(LogsService logsService) {
        this.logsService = logsService;
    }

    @GetMapping("/get-warnings-and-errors")
    public String displayTrendingHashtags(Model model){
        Task1Result logs = logsService.findWarningsAndErrors();
        model.addAttribute("errors", logs.getErrors());
        model.addAttribute("warnings", logs.getWarnings());
```

```
        return "task-1";
    }
}
```

Файл Task2Controller.java

```
package com.diploma.demo.controller;


import com.diploma.demo.service.TwitterService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;


import java.util.List;


@Controller
@RequestMapping("/api")
public class Task2Controller {
    private final TwitterService twitterService;


    public Task2Controller(TwitterService twitterService) {
        this.twitterService = twitterService;
    }


    @GetMapping("/get-trending-hashtags")
    public String getTrendingHashtags(Model model){
        List<String> hashtags = twitterService.findTrendingHashtags();
        model.addAttribute("hashtags", hashtags);
        return "task-2";
    }
}
```

Файл Task3Controller.java

```
package com.diploma.demo.controller;


import com.diploma.demo.model.Task3Result;
import com.diploma.demo.service.PsychoService;
import org.apache.spark.sql.Dataset;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;


import java.util.List;


@Controller
@RequestMapping("/api")
public class Task3Controller {
    private final PsychoService psychoService;

    public Task3Controller(PsychoService psychoService) {
        this.psychoService = psychoService;
    }

    @GetMapping("/get-aggregated-tests")
    public String getAggregatedTests(Model model){
        List<Task3Result> predictions = psychoService.processData();
        model.addAttribute("predictions", predictions);
        return "task-3";
    }
```

```
}
```

Файл GenerateDataController.java

```
package com.diploma.demo.controller;


import com.diploma.demo.service.PsychoService;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


@RestController
@RequestMapping("/api")
public class GenerateDataController {
    private final PsychoService psychoService;


    public GenerateDataController(PsychoService psychoService) {
        this.psychoService = psychoService;
    }


    @GetMapping("/generate-tests")
    public void generateTests(){
        psychoService.generateData();
    }
}
```

Файл ReadMongoConfig.java

```
package com.diploma.demo.config;


import com.mongodb.spark.config.ReadConfig;
import org.apache.spark.api.java.JavaSparkContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import java.util.HashMap;
import java.util.Map;

@Configuration
public class ReadMongoConfig {
    private final JavaSparkContext jsc;

    public ReadMongoConfig(JavaSparkContext jsc) {
        this.jsc = jsc;
    }

    @Bean
    public ReadConfig getReadConfig(){
        Map<String, String> readOverrides = new HashMap<>();
        readOverrides.put("collection", "outputData");
        readOverrides.put("readPreference.name", "secondaryPreferred");
        return ReadConfig.create(jsc).withOptions(readOverrides);
    }
}
```

Файл WriteMongoConfig.java

```
package com.diploma.demo.config;

import com.mongodb.spark.config.WriteConfig;
import org.apache.spark.api.java.JavaSparkContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashMap;
import java.util.Map;
```

```
@Configuration
public class WriteMongoConfig {
    private final JavaSparkContext jsc;

    public WriteMongoConfig(JavaSparkContext jsc) {
        this.jsc = jsc;
    }

    @Bean
    public WriteConfig getWriteConfig(){
        Map<String, String> writeOverrides = new HashMap<>();
        writeOverrides.put("collection", "inputData");
        writeOverrides.put("writeConcern.w", "majority");
        return WriteConfig.create(jsc).withOptions(writeOverrides);
    }
}
```

Файл SparkConfigForSpring.java

```
package com.diploma.demo.config;

import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.sql.SparkSession;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;

@Configuration
public class SparkConfigForSpring {
```

```
@Value("${spark.app.name}")
private String appName;

@Value("${spark.uri.local}")
private String masterUriLocal;

@Value("${spark.uri.cluster}")
private String masterUriCluster;

@Value("${spark.mongodb.input.uri}")
private String inputIri;

@Value("${spark.mongodb.output.uri}")
private String outputIri;

@Bean
@Profile("LOCAL")
public SparkSession sparkSessionLocal(){
   return SparkSession.builder()
         .master(masterUriLocal)
         .appName(appName)
         .config("spark.mongodb.input.uri", inputIri)
         .config("spark.mongodb.output.uri", outputIri)
         .getOrCreate();
}

@Bean
@Profile("CLUSTER")
public SparkSession sparkSessionCluster(){
   return SparkSession.builder()
```

```
            .master(masterUriCluster)
            .appName(appName)
            .config("spark.mongodb.input.uri", inputIri)
            .config("spark.mongodb.output.uri", outputIri)
            .getOrCreate();
    }

    @Bean
    @Profile("LOCAL")
    public JavaSparkContext sparkContextLocal(){
        return new JavaSparkContext(sparkSessionLocal().sparkContext());
    }

    @Bean
    @Profile("CLUSTER")
    public JavaSparkContext sparkContextCluster(){
        return new JavaSparkContext(sparkSessionLocal().sparkContext());
    }
}
```