Лабораторная 3. Веб-сервис с асинхронной обработкой запросов

Цель: на языке высокого уровня (Java, C#, Python, Go и др. – на выбор обучающегося) реализовать веб-сервис, принимающий на обработку CSV-файлы и *реализующий их асинхронную (отложенную) обработку*. Отложенное выполнение задач необходимо для того, чтобы обеспечить быстрый отклик в веб-приложениях и должно применяться во всех потенциально длительных операциях (посылка писем, обработка файлов и т.д.)

Веб-сервис должен предоставлять веб-API, обеспечивающий загрузку в сервис таблицы в виде CSV-файла для ее обработки и получения результата в виде N элементов из данного файла с максимальным значением значения данных по определенному полю данного CSV-файла (TOP-N). Первая строка CSV-файла содержит заголовки соответствующих столбцов таблицы. В качестве примера можно использовать данные со страницы https://www3.epa.gov/otaq/tcldata.htm (например, https://www3.epa.gov/fueleconomy/testcars/database/16tstcar.csv).

Задачи:

№	Задача	**Баллы**
1.	Реализовать веб-сервис, предоставляющий по адресу www.**example**.com/top веб-API для загрузки файлов с параметрами field и count - поле, по которому нужно сортировать записи и количество верхних записей соответственно. В ответ на запрос должна возвращаться ссылка на результирующий файл, в который будут добавлены результаты обработки.	5
2.	Реализовать метод /upload , обеспечивающий загрузку, асинхронную обработку записей из CSV-файла и добавление их в БД. По запросу GET /top?field=FIELD&count=N должен возвращаться JSON-ответ, содержащий верхние N записей по полю FIELD.	5
3*	Обеспечить распределенную обработку CSV-файла в задании 1. Файл должен разбиваться на части, и отдаваться на независимую обработку набору исполнителей. Каждый из исполнителей осуществляет независимый разбор файла и находит N верхних записей для своего сегмента, затем результаты обработки объединяются и берутся верхние N записей от результата (реализовать концепцию "MAP-REDUCE" в ручном режиме).	**3***

* дополнительные задания за дополнительные баллы.

Методические указания

Для реализации данного задания, необходимо воспользоваться очередью, обеспечивающей асинхронный обмен сообщениями между компонентами системы. Как одно из возможных решений, можно воспользоваться системой Redis: http://redis.io. Для выполнения лабораторной работы можно как самостоятельно развернуть данную систему, так и воспользоваться готовым облачным решением, например http://redistogo.com/.

Для реализации веб-сервиса, легче всего воспользоваться одним из готовых фреймворков для вашего языка программирования. Для языков Python, Ruby и Go можно рекомендовать такие платформы как Flask, Sinatra и Martini соответственно (https://realpython.com/blog/python/python-ruby-and-golang-a-web-Service-application-comparison/).

Рассмотрим пример реализации простейшего веб-сервиса, реализующего асинхронный процесс обработки загруженных файлов, на базе фреймворка Spark для языка Java.

1. Для реализации асинхронного обмена и отложенного задания необходимо завести аккаунт на http://redistogo.com/ (внизу есть маленькая кнопочка free plan), после чего вам выдадут connection string вида redis://redistogo:34534987987@angelfish.redistogo.com:10649/

2. Создадим новый Java-проект с использованием системы Maven, который будет состоять из двух независимых компонентов: веб-сервиса и обработчика.

3. Для работы с проектом, нам потребуется внедрить в проект системы для работы с платформой Redis (`jedis`) и веб-сервис Spark. Это можно реализовать посредством платформы Maven, указав соответствующие зависимости в конфигурационном файле pom.xml. Рассмотрим блок dependencies которые необходимо добавить в pom.xml (полный код pom.xml доступен вот тут: https://gist.github.com/skayred/0c0a9dc57ad8f9fa9744eea49f1fb50a):

```
<dependencies>
    <dependency>
      <groupId>redis.clients</groupId>
      <artifactId>jedis</artifactId>
      <version>2.8.1</version>
    </dependency>
```

```
    <dependency>
      <groupId>com.sparkjava</groupId>
      <artifactId>spark-core</artifactId>
      <version>2.3</version>
    </dependency>
</dependencies>
```

Добавление данных строк в pom.xml и пересборка Maven-проекта загрузит необходимые для реализации проекта зависимости.

4. Рассмотрим исходный код веб-сервиса (https://gist.github.com/skayred/22bd36990558069ef65f505f8201e711):

Сервер обеспечивает обработку POST запросов по адресу "\upload", принимающих один параметр "file". Полученный в запросе файл сохраняется во временном хранилище, после чего его имя передается в очередь для дальнейшей обработки.

```
package ru.susu;


import redis.clients.jedis.Jedis;
import static spark.Spark.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.nio.*;
import java.nio.file.*;
import java.util.*;


public class AppServer {
    public static void main( String[] args ) {
        // Соединяемся с Редисом
        Jedis jedis = new Jedis("АДРЕС_РЕДИСА");


        // Тут мы используем фремворк Spark, следующий вызов
вешает слушателя на метод POST
        post("/upload", "multipart/form-data", (request,
response) -> {
                // Получаем файл из ПОСТа и сохраняем на диск
                String location =
"/home/username/projects/service/";          // the directory
location where files will be stored
                long maxFileSize = 100000000;       // the
maximum size allowed for uploaded files
                long maxRequestSize = 100000000;    // the
```

```
maximum size allowed for multipart/form-data requests
                int fileSizeThreshold = 1024;       // the size
threshold after which files will be written to disk

                MultipartConfigElement multipartConfigElement =
new MultipartConfigElement(
location, maxFileSize, maxRequestSize, fileSizeThreshold);

request.raw().setAttribute("org.eclipse.jetty.multipartConfig",
multipartConfigElement);

                String fName =
request.raw().getPart("file").getSubmittedFileName();

                Part uploadedFile =
request.raw().getPart("file");
                Path out = Paths.get(location + fName);
                try (final InputStream in =
uploadedFile.getInputStream()) {
                    Files.copy(in, out);
                    uploadedFile.delete();
                }
                // cleanup
                multipartConfigElement = null;
                uploadedFile = null;

                // После сохранения файла отправляем путь к
нему обработчику
                jedis.rpush("queue", out.toString());

                return "OK";
            });
    }
}
```

5. Реализуем исполнителя

Исполнитель получает адрес файла, который необходимо обработать, после чего выводит содержимое данного файла.

```
package ru.susu;


import redis.clients.jedis.Jedis;
import java.util.*;
import java.io.*;
import java.nio.file.*;
```

```
public class App {
    public static void main( String[] args ) {
        // Соединяемся с Редисом
        Jedis jedis = new Jedis("АДРЕС_РЕДИСА");
        List<String> messages = null;
        // Ждем сообщения из очереди
        while(true){
          System.out.println("Waiting for a message in the
queue");
          messages = jedis.blpop(0,"queue");
          String payload = messages.get(1);
          // Выполняем задачу
          processFile(payload);
        }
    }


    // Выводим содержимое файла в консоль
    private static void processFile(String filename) {
        try {
            System.out.println(readFile(filename));
        } catch (IOException e) {
        }
    }


    static String readFile(String path) throws IOException {
        byte[] encoded = Files.readAllBytes(Paths.get(path));
        return new String(encoded);
    }
}
```

Вам предлагается самостоятельно выполнить задания 1-2 лабораторной работы на основе представленного примера.