**Задание для кандидата на должность Big Data Engineer.**

**Уровень**: junior.

**Список технологий**: Hadoop, Spark, Kafka, Docker, REST-API, SQL.

**Цель**: создание REST-сервиса по принятию решения об одобрении или отклонении заявок на получение рассрочек на смартфоны.

**Описание**: мы являемся разработчиком ПО в финансовой организации, которая решила заняться продажей смартфонов и выдачей на них рассрочек (на 12 месяцев). Нашей задачей является создание сервиса по принятию решения о выдаче клиенту рассрочки на смартфон и подготовка соответствующей инфраструктуры.

Наша база данных имеет следующие **таблицы**:

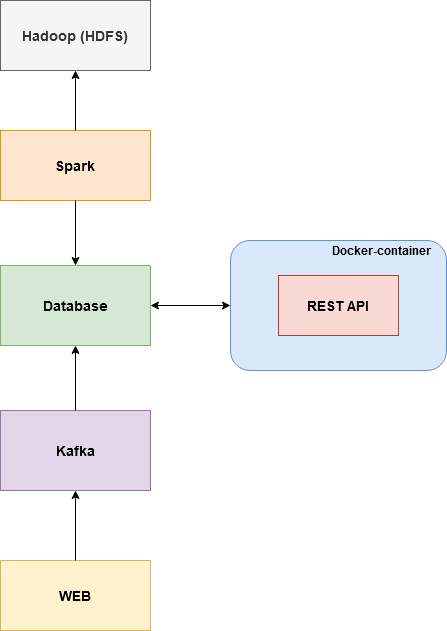
Таблица “черный список” (blacklist)

|  |  |
| --- | --- |
| ctn | event\_time |
| 7051231234 | 2020-10-01 12:00:01 |
| 7059879876 | 2020-10-01 12:00:01 |

Таблица “смартфоны” (smartphones)

|  |  |  |  |
| --- | --- | --- | --- |
| smartphone\_id | smartphone\_brand | smartphone\_model | smartphone\_price |
| 1 | Apple | iPhone X | 200000 |
| 2 | Samsung | Galaxy S9 | 180000 |
| 3 | Samsung | A40 | 70000 |

**Схема архитектуры проекта в рамках данного задания:**



**Сценарий принятия решения** об одобрении или отказе в рассрочке выглядит следующим образом:

1. Принимаем входные данные через POST-запрос нашего REST-сервиса (API).

Пример запроса и входных данных:

*POST http://localhost:5000/get\_result*

*{*

*“ctn”: “7051231234”,*

*“iin”: “900101123456”,*

*“smartphoneID”: 2,*

*“income”: 120000,*

*“creationDate”: “2020-10-01T12:30:00”*

*}*

, где:

ctn – номер телефона клиента,

iin – ИИН клиента,

smartphoneID – код желаемого смартфона,

income – средний ежемесячный доход клиента в тенге,

creationDate – дата создания заявки.

1. Далее сервис должен проверить номер “ctn” на наличие в черном списке.

Если “ctn” был найден в черном списке, то сервис должен вернуть следующий ответ:

*{*

*“result”: “Отказ”,*

*“reason”: “Клиент находится в черном списке”*

*}*

Если клиента не оказалось в черном списке, то переходим к следующему шагу сценария.

1. На данном этапе необходимо принять решение о выдаче рассрочки на основе дохода клиента и стоимости смартфона.

Для этого необходимо получить стоимость смартфона из нашей базы данных на основе параметра “smartphoneID”. Далее необходимо вычислить величину ежемесячного платежа. Для этого нам достаточно поделить полную стоимость на 12.

Далее проверяем: если величина ежемесячного платежа превышает 20% от среднего ежемесячного дохода клиента (параметр “income”), то возвращаем ответ об отказе:

*{*

*“result”: “Отказ”,*

*“reason”: “Клиент не имеет достаточный уровень дохода”*

*}*

Если же величина ежемесячного платежа меньше 20% от среднего ежемесячного дохода клиента, то возвращает ответ об одобрении:

*{*

*“result”: “Одобрение”*

*}*

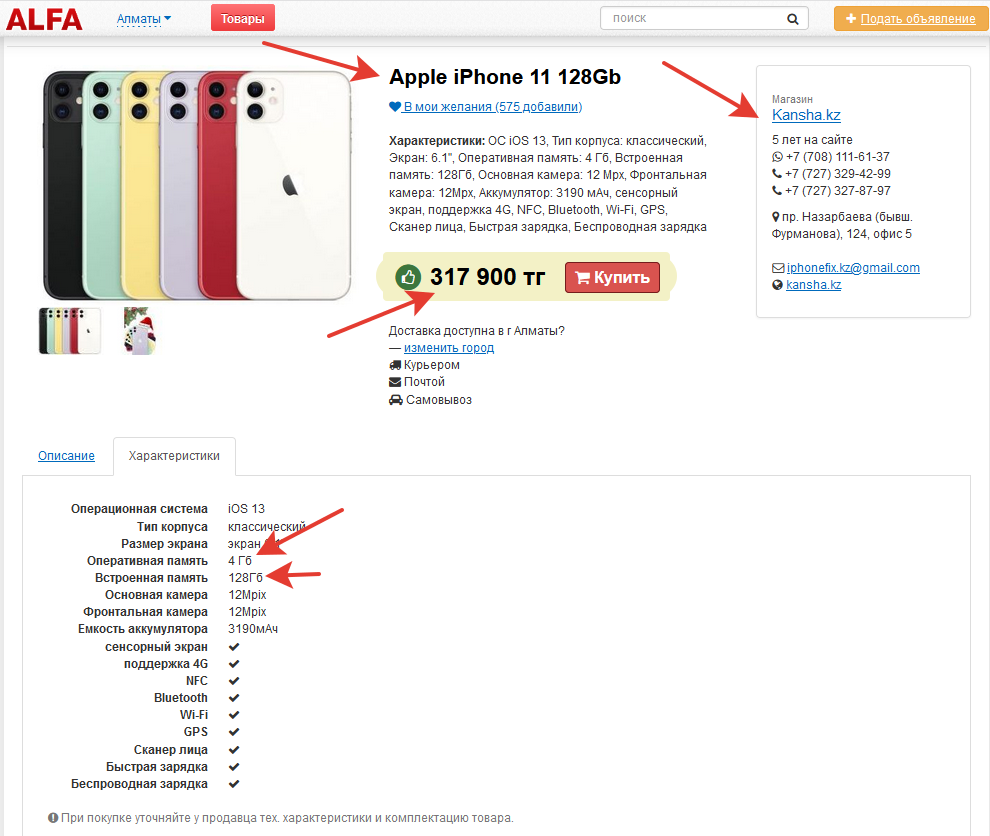
Разработку полного сервиса можно поделить на следующие шаги:

1. Парсинг web-ресурса.
2. Работа со spark, hadoop и базами данных.
3. Работа с бинарными файлами.
4. Работа с kafka.
5. Разработка API и работа с Docker.

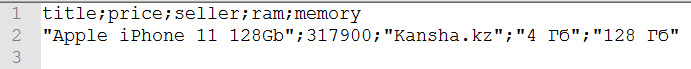
**Часть первая. Парсинг WEB-ресурсов.**

Необходимо написать скрипт парсинга ресурса alfa.kz. В частности, нужно получить данные по объявлениям из следующей ссылки: <https://alfa.kz/phones/telefony-i-smartfony>

В рамках задания достаточно распарсить 5-10 разных моделей смартфонов. Необходимы следующие параметры: название смартфона, стоимость, имя продавца, размер оперативной памяти, размер встроенной памяти.



Результат парсинга необходимо записать в csv-файл.



**Часть вторая. Загрузка данных в Hadoop (hdfds) и таблицу “smartphones”.**

На данном этапе необходимо начать настройку и подготовку необходимой инфраструктуры. А именно установка Hadoop (hdfs), spark и любой реляционной базы данных (например, postgresql).

Далее при помощи Spark (на scala или python) необходимо написать скрипт, который будет работать с csv-файлом из предыдущего этапа. Скрипт должен сделать следующее:

* Считать csv-файл из первого этапа и преобразовать данные этого файла в spark-dataframe.
* Разделить поле title на два отдельных – smartphone\_brand и smartphone\_model. Например, “Apple iPhone 11 128Gb” разделится на брэнд “Apple” и модель “iPhone 11 128 Gb”.
* Найти средние стоимости моделей смартфонов. Например, на первом шаге мы спарсили два разных объявления по продаже “iPhone 11 128 Gb”. Стоимость смартфона в первом объявлении равна 300000 тг, а во втором – 310000 тг. Скрипт должен объединить две эти записи в одну со стоимостью 305000 тг. При этом мы удаляем поля seller, ram и memory.
* Каждой уникальной записи dataframe’а необходимо присвоить свой порядковый smartphone\_id, начиная с 1.
* Финальный dataframe необходимо записать в Hadoop (hdfs) в формате parquet в целях долгосрочного хранения данных, а также в таблицу “smartphones” реляционной базы данных для работы с API.

|  |  |  |  |
| --- | --- | --- | --- |
| smartphone\_id | smartphone\_brand | smartphone\_model | smartphone\_price |
| 1 | Apple | iPhone X | 200000 |
| 2 | Samsung | Galaxy S9 | 180000 |
| 3 | Samsung | A40 | 70000 |

**Часть третья. Парсинг бинарных файлов.**

Необходимо разработать скрипт, который преобразует данные из файла, содержащего информацию в бинарном формате, в привычный читабельный вид.

Название файла: data.bin

Бинарный файл содержит в себе данные об абонентах, которых стоит внести в наш черный список.

Запись об одном абоненте весит 21 байт. В файле закодированы следующие поля по порядку:

* Номер записи – целое число, 4 байта, big endian.
* Номер телефона – string, символы utf-8, занимает 10 байт.
* Год – целое число, 2 байта, little endian
* Номер месяца – целое число, 1 байт, little endian
* День – целое число, 1 байт, little endian
* Час – целое число, 1 байт, little endian
* Минута – целое число, 1 байт, little endian
* Секунда– целое число, 1 байт, little endian

В результате преобразования фала data.bin должны получиться следующие данные:

|  |  |  |
| --- | --- | --- |
| Номер | Номер телефона | Время события |
| 1 | 7051231234 | 2020-10-04 12:01:29 |
| 2 | 7771231235 | 2020-02-25 03:10:03 |
| 3 | 7051231236 | 2019-05-17 16:55:40 |
| 4 | 7771233377 | 2020-12-05 05:30:41 |
| 5 | 7051232255 | 2021-07-09 20:32:50 |

Результат нужно записать в csv-файл.

**Часть четвертая. Заполнение таблицы “blacklist”.**

В рамках данного этапа нам необходимо заполнить таблицу “blacklist” нашей реляционной базы данных (та же база, в которой у нас хранится таблица “smartphones”).

Для заполнения таблица нужно задействовать технологию Kafka. Для этого нам необходимо развернуть сервис kafka, создать в нём новый топик blacklist и написать два отдельных скрипта:

* Первый скрипт считывает данные из csv-файла и отправляет все записи по отдельности в наш kafka-топик.
* Второй скрипт считывает записи из нашего kafka-топика и заносит их в таблицу “blacklist” нашей базы данных.

Также в базе данных необходимо создать индекс на поле “ctn” таблицы “blacklist”.

**Часть пятая. Разработка REST-API и развертывание контейнера.**

Необходимо разработать REST-API, который будет работать согласно сценарию принятия решения, описанному на второй странице данного документа.

Можно использовать любой предпочитаемый язык программирования или framework (flask, django, spring, node.js и др).

Результат каждого запроса необходимо логировать в нашу базу данных.

Возможный вариант таблицы логов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | url | ctn | result | log\_timestamp |
| 1 | http://localhost:5000/get\_result | 7051231234 | Отказ | 2020-10-01 14:00:00 |
| 2 | http://localhost:5000/get\_result | 7051231235 | Одобрение | 2020-10-01 16:10:20 |

Важно! В самом конце необходимо обернуть весь API в docker-контейнер.