Предисловие

Мы стремимся реализовать распределенную систему для взлома хэша под кодовым именем **CrackHash**. Непосредственно взлом хэша будем реализовывать через простой перебор словаря сгенерированного на основе алфавита (brute-force). В общих чертах система должна работать по следующей логике:

1. В рамках системы существует менеджер, который принимает от пользователя запрос, содержащий MD-5 хэш некоторого слова, а также его максимальную длину.
2. Менеджер обрабатывает запрос: генерирует задачи в соответствии с заданным числом воркеров (вычислительных узлов) на перебор слов составленных из переданного им алфавита. После чего отправляет их на исполнение воркерам через очередь RabbitMq.
3. Каждый воркер принимает задачу, вычисляет собственный диапазон в котором нужно проверять слова, генерирует и вычисляет их хэш. Находит слова у которых он совпадает, и результат работы возвращает менеджеру через очередь.

Task 1. Services Implementation

В рамках первой лабораторной работы необходимо реализовать приложения менеджера и воркера, а также организовать простое их взаимодействие через HTTP.

Примечание:
Настройка очередей и producer-ов/listener-ов для них в рамках данной задачи не предполагается!

Общие требования к сервисам:

1. Реализация приложений предполагается на языке Java (11+) с использованием фреймворка Spring Boot.
2. Для сборки рекомендуется использовать Gradle (6+).
3. Для развертывания сервисов необходимо использовать **docker-compose**, конфигурация с одним воркером.
4. Запросы между менеджером и воркером необходимо передавать внутри сети docker-compose по протоколу HTTP, используя в качестве доменов имена сервисов.

Требования к менеджеру

1. Менеджер должен предоставлять клиенту **REST API** в формате **JSON** для взаимодействия с ним.

   Запроса на взлом хэша **(слово abcd):**

   ```
   POST /api/hash/crack
   Request body:
   {
       "hash":"e2fc714c4727ee9395f324cd2e7f331f",
       "maxLength": 4
   }
   ```

В ответ менеджер должен отдавать клиенту идентификатор запроса, по которому тот сможет обратится за получением ответа.

```
Response body:
{
    "requestId":"730a04e6-4de9-41f9-9d5b-53b88b17afac"
}
```

2. Для получения результатов менеджер должен представлять следующее API.

   `GET /api/hash/status?requestId=730a04e6-4de9-41f9-9d5b-53b88b17afac`

   Ответ, если запрос еще обрабатывается.

```
Response body:
{
    "status":"IN_PROGRESS",
    "data": null
}
```

   Ответ, если ответ готов.

```
Response body:
{
   "status":"READY",
   "data": ["abcd"]
}
```

3. В качестве алфавита менеджер должен использовать строчные латинские буквы и цифры (ограничимся ими в целях экономии времени на вычисления).
4. Взаимодействие между менеджером и воркерами должно быть организовано в формате **XML**. Поэтому необходимо сгенерировать модель запроса менеджера к воркеру на основе **xsd**-схемы. Далее запросы для воркеров заполнять в соответствии с моделью.
5. Перед отправкой задач воркерам менеджер должен сохранить в оперативной памяти информацию о них с привязкой к запросу клиента в статусе **IN_PROGRESS** под идентификатором, который после должен быть выдан пользователю. При получении ответов от всех воркеров менеджер должен перевести статус запроса в **READY**. По истечению таймаута запрос должен перевестись в статус **ERROR**.
6. Для работы с состоянием запросов использовать потокобезопасные коллекции.
7. Взаимодействие с воркером организовать по протоколу HTTP с помощью Rest Template. Для этого в воркере необходимо реализовать контроллер для обработки запросов от менеджера, принимающий запрос по следующему пути:
   `POST /internal/api/worker/hash/crack/task`

Требования к воркерам:

1. Взаимодействие между воркером и менеджером также должно быть организовано в формате **XML**. Поэтому необходимо сгенерировать модель запроса ответа воркера на основе **xsd**-схемы. Ответ для менеджера заполнять в соответствии с моделью.
2. Взаимодействие с менеджером организовать по протоколу HTTP с помощью Rest Template. Для этого в менеджере необходимо реализовать контроллер для обработки ответов от воркера по следующему пути:

`PATCH` **`/internal/api/manager/hash/crack/request`**

3. Для генерации слов на основе полученного алфавита можно использовать библиотеку combinatoricslib. Она позволяет генерировать перестановки с повторениями заданной длины на основе заданного множества. Ключевое условие здесь, чтобы воркер не держал в памяти все сгенерированные комбинации т.к. при увеличении максимальной длины последовательности их банально станет очень много. Для этого библиотека предоставляет итератор по множеству комбинаций.
4. Расчет диапазона слов необходимо производить на основе значений PartNumber и PartCount из запроса от менеджера. Необходимо поделить всё пространство слов поровну между всеми воркерами.

Полезные ссылки:

1. Пример базового сервиса на Spring
   https://spring-projects.ru/guides/rest-service/
2. База про XSD
   https://www.codeguru.com/java/xsd-tutorial-xml-schemas-for-beginners/
3. XSD спецификация
   https://www.w3.org/TR/xmlschema11-1/
4. Пример генерации JAXB моделей на основе xsd
   https://spring.io/guides/gs/producing-web-service/
5. Генерация последовательностей
   https://github.com/dpaukov/combinatoricslib
6. Rest Template
   https://docs.spring.io/spring-android/docs/current/reference/html/rest-template.html
   https://www.baeldung.com/rest-template
7. JAXB + Rest Template
   https://stackoverflow.com/questions/41288036/how-do-i-use-jaxb-annotations-with-spring-resttemplate