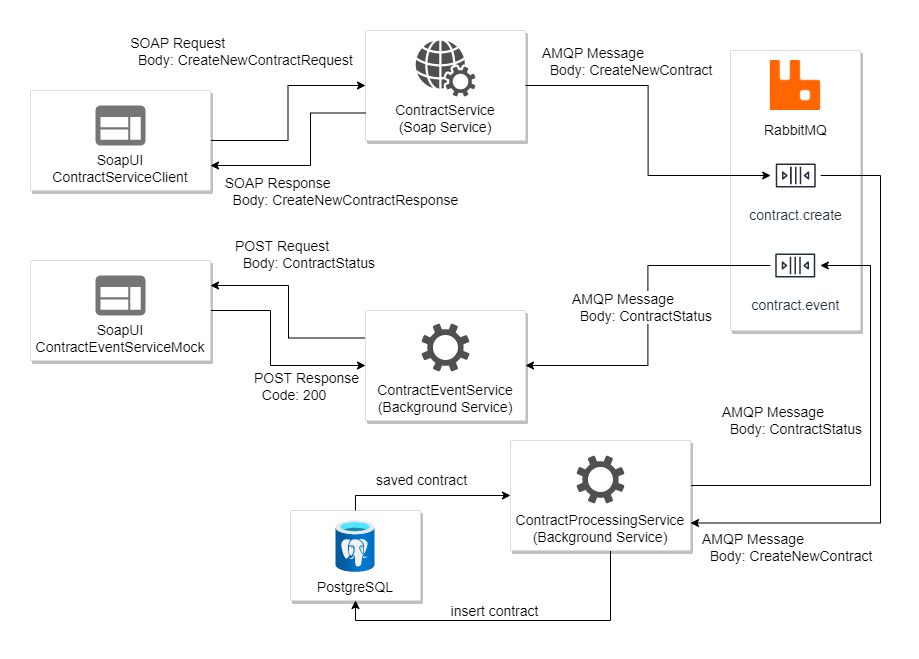
Тестовое задание

## **Продуктовые требования**

Реализация обработки потока данных по схеме:



Требуется реализовать набор микро-сервисов: **ContractService**, **ContractProcessingService** и **ContractEventService**.

**ContractService** принимает запрос на создание договора по протоколу soap, преобразует полученный конверт во внутреннее представление **CreateNewContract** и пересылает сообщение в очередь **contract.create** RabbitMQ, после чего возвращает синхронный ответ со статусом **RequestIsQueued**. Если при отправке в RabbitMQ происходит ошибка (например, если RabbitMQ не доступен), то возвращается ответ со статусом **Error** и человеко-читаемым сообщением об ошибке в поле ErrorMessage.

**ContractProcessingService** слушает очередь **contract.create** в RabbitMQ. Полученные из очереди сообщения записывает в таблицу **contract** БД PostgreSQL и отправляет в очередь contract.event событие об успешной регистрации в виде экземпляра **ContractStatus**. Перед сохранением выполняется проверка на дубликаты: в случае, если в БД уже записан договор с аналогичным **contract\_number** или **id** - в очередь **contract.event** отправляется событие со статусом ошибки (для случая с id - свой код, для случая с contract\_number - свой, описано в маппинге).

Структура таблицы **contract** описана в маппинге.

**ContractEventService** слушает очередь **contract.event** в RabbitMQ. Полученные из очереди сообщения транслирует в тело POST-запроса и отправляет по адресу http:/host:port/status.

### Маппинги

#### Маппинг CreateNewContractRequest во внутреннее представление CreateNewContract

| **CreateNewContract** | **CreateNewContractRequest** | **Описание** |
| --- | --- | --- |
| id | Id |  |
| date\_start | DateStart |  |
| date\_end | DateEnd |  |
| date\_send |  | Текущая дата и время |
| contract\_number | ContractNumber |  |
| contract\_name | ContractName |  |
| client\_api |  | Перечислимое значение **Soap** |
| contractual\_parties | ContractualParties | Маппинг массива. Описание маппинга элемента массива ниже. |

#### Маппинг ContractualParty во внутреннее представление ContractualParty

| **CreateNewContract.ContractualParty** | **CreateNewContractRequest .ContractualParty** |
| --- | --- |
| name | Name |
| bank\_account\_number | BankAccount |
| bik | BankBik |

#### Маппинг внутреннего представления CreateNewContract на таблицу БД contract

| **contract** | **CreateNewContract** | **Описание колонок таблицы БД** |
| --- | --- | --- |
| id | id | Первичный ключ таблицы.  Тип колонки uuid. |
| date\_start | date\_start | Дата без времени и временной зоны |
| date\_end | date\_end | Дата без времени и временной зоны |
| date\_send | date\_send | Дата и время без временной зоны |
| date\_create |  | Должна заполняться автоматически текущей датой и временем при выполнении вставки в БД.  Дата и время без временной зоны |
| contract\_number | contract\_number | Добавить индекс. |
| contract\_name | contract\_name |  |
| client\_api | client\_api |  |
| contractual\_parties | contractual\_parties | Использовать тип jsonb для хранения json-объекта. В данном случае будет сохраняться массив целиком. |

#### Маппинг созданной записи contract на статус ContractStatus

| **ContractStatus** | **contract** | **Описание** |
| --- | --- | --- |
| id | id |  |
| status |  | Перечислимое значение **Created**, если сохранение прошло успешно и **Error**, если запись с таким id или contract\_number уже существует. |
| date\_create | date\_create | Заполняется, если сохранение выполнено. |
| error\_code |  | Не заполняется, если сохранение выполнено.  Иначе заполняется значениями:  **1** - если запись с таким id уже существует;  **2** - если запись с таким contract\_number уже существует. |

## **Технические требования**

* Используемый стек: Java 8, Maven, Spring Boot, Apache Camel, Spring WS, Spring AMQP, Spring Data JDBC, MapStruct, RabbitMQ, PosgreSQL, Docker;
* Для простоты разработки и проверки, сформировать docker-compose стек с сервисами RabbitMQ и PostgreSQL, настроить их запуск на нестандартных портах и зафиксировать эти порты в файлах конфигурации сервисов;
* Сервис ContractService реализовать на базе Spring WS, используя подход Contract-First и прилагаемую схему ContractService.xsd;
* Сервисы ContractProcessingService и ContractEventService реализовать на базе Spring Boot + Apache Camel;
* Генерация контрактов данных для сервиса ContractService по схеме ContractService.xsd должна быть реализована с использованием плагина jaxb2-maven-plugin. Генерируемые контракты должны генерироваться в директорию со сборкой target, а не в основной проект;
* DTO, пересылаемые через RabbitMQ и в качестве тела ответного POST-запроса должны генерироваться по схемам CreateNewContract.json и ContractStatus.json при помощи плагина jsonschema2pojo-maven-plugin.

Необходимо настроить плагин так, чтобы он генерировал типы дата и дата и время в LocalDate и LocalDateTime соответственно. Генерируемые контракты должны генерироваться в директорию со сборкой target, а не в основной проект;

* В сервисе ContractService отправка сообщения CreateNewContract в RabbitMQ должна производиться средствами Spring AMQP;
* В сервисе ContractProcessingService получение сообщения и отправка статуса в очередь RabbitMQ должна быть полностью реализована при помощи маршрутизации Apache Camel;
* В сервисе ContractProcessingService получение сообщения должно реализовываться при помощи Apache Camel. Отправка POST-запроса любым http-клиентом.
* Маппинг объектов в ContractService реализовать при помощи MapStruct;
* Сервисы ContractProcessingService и ContractEventService должны создать очереди contract.create и contract.event соответственно. Это не то, что должно использоваться в production, но в целях простоты отладки и проверки, воспользоваться для этого опцией autoDeclare у Apache Camel;
* Данные подключений к RabbitMQ, БД, порт поднятия soap-сервиса, адрес отправки POST-запроса, должны быть оформлены в файлах конфигурации соответствующих сервисов - application.yml;
* Приложить скрипт создания БД и таблицы contract БД в директорию с сервисом ContractProcessingService или реализовать автоматическое создание таблицы в БД, если она не существует. Рекомендуется создать отдельный класс dto для работы с таблицей contract БД или использовать plain sql запросы;
* Полный цикл проверки от отправки soap-запроса в ContractService до получения ответного статуса должно быть возможно провести при помощи SoapUI-проекта ContractService, конфигурация которого прилагается в файле ContractService-soapui-project.xml и доступна для импорта в SoapUI версии 5.7.0.

Проект содержит:

1. Клиент для отправки soap-запроса CreateNewContract на адрес <http://localhost:8284/ws>. По этому адресу должен запускаться ContractService. У ContractService по адресу <http://localhost:8284/ws/ContractService/ContractService.wsdl> должна быть доступна схема wsdl сервиса;
2. Мок ContractEventServiceMock, который принимает по адресу <http://localhost:8285/status> POST-запрос статуса и возвращает в ответ код 200.