ИНФРАСТРУКТУРА ЭЛЕКТРОННОГО ПРАВИТЕЛЬСТВА

# СМЭВ QL сервер

Версия 1.4.0

Листов 130

Москва, 2025

# СМЭВ QL СЕРВЕР

**СОДЕРЖАНИЕ**

# 1 Назначение СМЭВ QL сервера

## 1.1 О системе

СМЭВ QL сервер - это компонент взаимодействия с витриной данных, который реализует её типовое API согласно внутренней спецификации СМЭВ QL.

Основным назначением СМЭВ QL сервера является обработка REST-запросов на получение данных от Потребителей и формирование оптимальных (простых) SQL-запросов к витрине для получения запрашиваемых данных.

Для Потребителя взаимодействие со СМЭВ QL сервера равнозначно виду информационного обмена - Обмен с использованием регламентированных запросов типа «Rest-сервис».

Основной принцип работы СМЭВ QL сервера заключается в следующем:

1. СМЭВ QL принимает на вход REST-запрос, который содержит перечень запрашиваемых ресурсов, условий выбора данных, требуемые атрибуты ответа и пр. После чего определяет данные каких объектов и из каких источников необходимо извлечь. При этом определение источников происходит на основании заранее описанных моделей данных, которые хранятся на сервере в файлах вида `model.yaml` (что извлекать) и `source.yaml` (откуда извлекать).
2. Далее формирует, в определенной последовательности (на основании плана выполнения запросов), столько SQL-запросов к источникам данных, сколько ресурсов было запрошено в исходном запросе от клиента. При этом запросы могут выполняться как последовательно, в случае если в запросе ресурсы имеют вложенность (иерархию) или параллельно, если ресурсы указаны на одном уровне. Это позволяет значительно упростить sql-выражения и уменьшить сложность запроса к БД.
3. Обрабатывает результаты выполнения SQL-запросов по мере их поступления от источников.
4. Затем формирует и передает комплексный ответ клиенту.

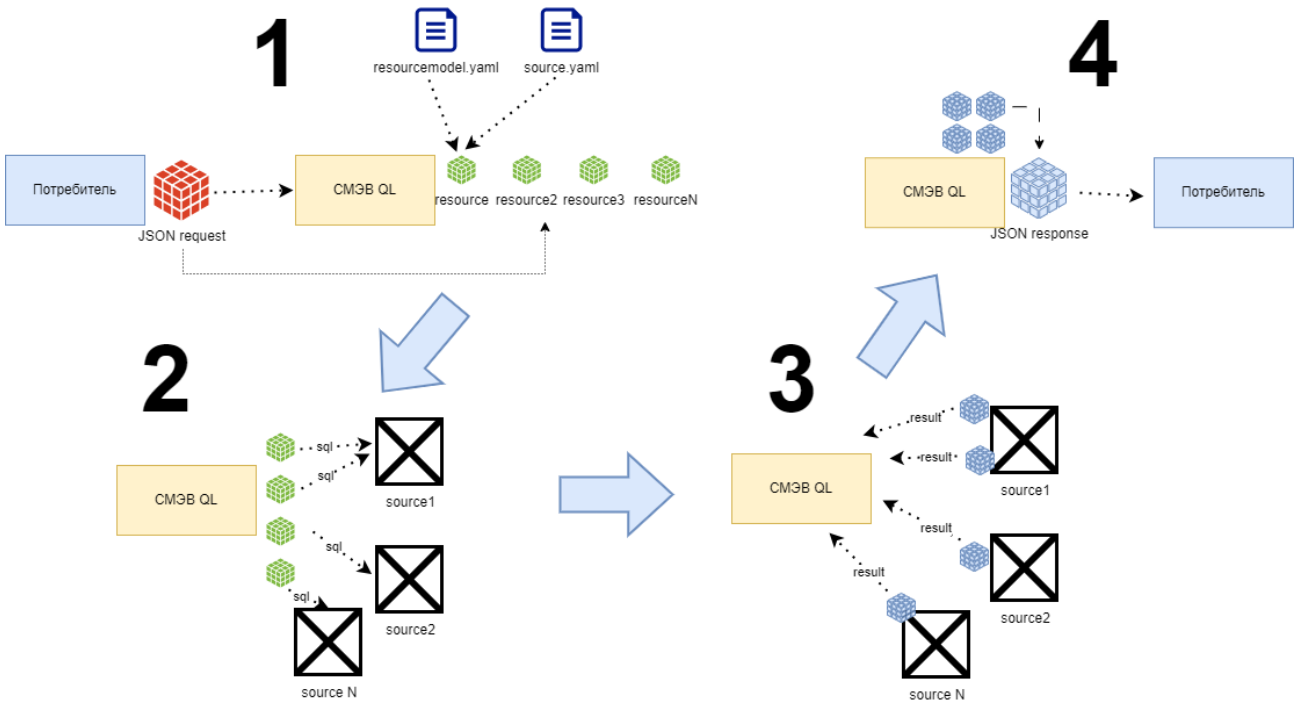



Рисунок - 1.3 Схема СМЭВ QL Сервера

## 1.2 Цели СМЭВ QL сервера

Целями создания СМЭВ QL сервера являются:

1. Повышение скорости предоставления ответов от витрины данных Поставщика по сравнению с типовым видом взаимодействия Агент-Витрина.
2. Защита витрины данных Поставщика от неоптимальных запросов.
3. Сокращение объёма ответов.
4. Сокращение количества передаваемых запросов от Потребителей.
5. Повышение скорости развития услуг ЕПГУ.

## 1.3 Задачи СМЭВ QL сервера

Основные задачи СМЭВ QL сервера:

1. Формирование API и модели данных витрины .
2. Приём REST-запросов от Потребителей через Агент СМЭВ4.
3. Формирование простых SQL-запросов к витрине.
4. Формирование распределенных запросов к нескольким витринам.
5. Формирование и передача ответа Потребителю.
6. Проверка и формирование цифровых подписей ответов.
7. Описание и исполнение при вызове модели машины состояний.
8. Нотификация подписчиков при изменении данных витрины.
9. Предоставление внешним клиентам OpenAPI для управления.

## 1.4 Место СМЭВ QL сервера в ИТ-ландшафте

СМЭВ QL сервер взаимодействует со следующими компонентами СМЭВ4:

1. Агент СМЭВ4.
2. Сервис исполнения запросов ядра витрины данных.
3. Сервер криптографии (Notarius).

4. Сервис формирования документов.

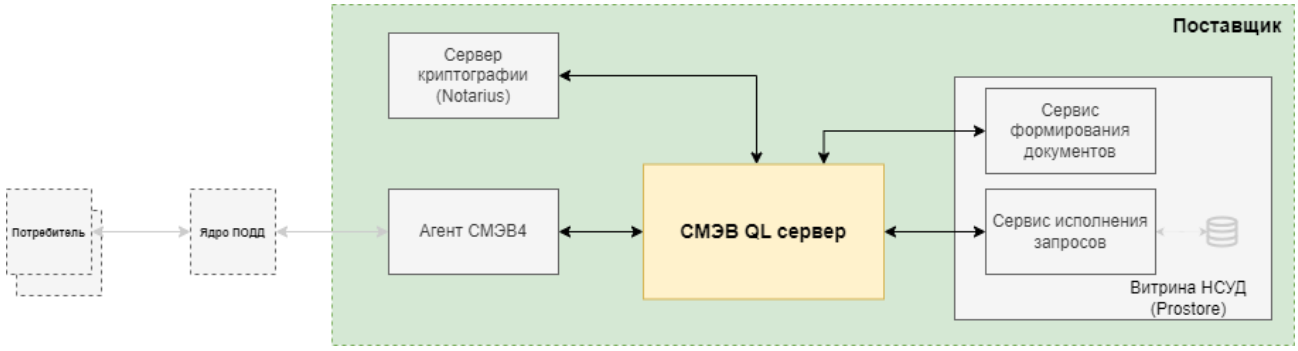



Рисунок - 1.4 Схема взаимодействия СМЭВ QL Сервера с компонентами СМЭВ4

## 1.5 Язык и синтаксис

### 1.5.1 Моделирование

Для моделирования документного слоя данных в спецификации выбран язык разметки YAML.

### 1.5.2 Запросы и ответы

Для написания запросов, а также в качестве сериализатора ответов, спецификация определяет использование JSON.

## 1.6 Типизация

Фактические типы данных наследуют типы данных JSON (включая NULL):

- string;
- number;
- object;
- array;
- boolean;
- null.

### 1.6.1 Типы данных в модели и приведение типов

В описании модели допускается указание фактического типа данных атрибута ресурса **вторым элементом массива type**. Указание является опциональным, по умолчанию подразумевается неограниченный **STRING**.

Пример из описания модели:

```
fields:
  id:
    name: Идентификатор записи
    type:
      - number
      - SHORT
    length: 20
    nullable: not NULL
    key: PRIMARY
```

В качестве второго уточняющего типа следует применять типы НСУД:

- STRING;
- DOUBLE;
- FLOAT;

- BOOLEAN;
- BYTE (не поддерживается на витрине);
- BINARY;
- BIG_DECIMAL (не поддерживается на витрине);
- LONG;
- INTEGER;
- SHORT;
- DATE;
- TIME;
- TIMESTAMP.

### 1.7 Моделирование данных

Модели данных описываются в формате `YAML` в папке проекта `models` согласно спецификации СМЭВ QL.

Структура базовой модели приведена в Базовая модель данных.

Структура базовой модели приведена в Модель данных.

> **Примечание:**
>
> Заливка данных через через модуль REStful-Uploader и DATA-Uploader не предусматривают параллельную заливку в датамарты вместе с другими инструментами. Параллельная заливка данных в те же датамарты вручную или средствами ETL приведет к конфликту в работе с дельтами и к ошибкам соответственно.

### 1.8 Метрики

Для обеспечения возможности сбора информации о работе СМЭВ QL Сервера реализован набор метрик, обеспечивающий формирование показателей:

- время исполнения входящих запросов;
- количество успешных / не успешных выполнений входящих запросов;
- время исполнения исходящих запросов или обращений к СПО;
- количество успешных / не успешных выполнений исходящих запросов или обращений к СПО.

## 2 Быстрый старт

### 2.1 Создание и конфигурация

Создать новое приложение СМЭВ QL Сервера командой:

```
java -jar smevql-server-all.jar new <new-app-name>
```

Данная команда создаст структуру папок сервера и исполняемый файл `smevql` внутри `<new-app-name>`.

### 2.2 Запуск и управление

Запуск СМЭВ QL Сервера осуществляется командой:

```
./smevql start -e <environment>
```

- `environment` - задается название среды разработки. Без указания окружения сервер будет запущен в `development`.

В момент запуска приложения выполняются проверки наличия и корректности

заполнения конфигурационного файла и файлов моделей.

Типовые ошибки представлены ниже:

**Тип ошибки**: Некорректный формат или отсутствие файла модели

**Пример лог-записи**: {«@timestamp»:»2024-01-10T16:25:55.460659+03:00»,»@version»:»1»,»message»:»Ошибка старта сервера»,»logger_name»:»ru.gov.digital.smevql.server.RequestHandler»,»thread_name»:»main»,«level»:»ERROR»,»level_value»:40000,»stack_trace»:»java.lang.RuntimeException: Ошибка парсинга модели данных из файла

**Тип ошибки**: Некорректно заполнен конфигурационный файл

**Пример лог-записи**: {«@timestamp»:»2024-01-10T16:27:01.202248+03:00»,»@version»:»1»,»message»:»Ошибка старта сервера»,»logger_name»:»ru.gov.digital.smevql.server.RequestHandler»,»thread_name»:»main»,«level»:»ERROR»,»level_value»:40000,»stack_trace»:»net.mamoe.yamlkt.YamlDecodingException:Top-level decoder: Yaml Block Map: bad indentation, baseIndent=0, newIndent=2n uri: smevql/api/v1

Остановка СМЭВ QL Сервера осуществляется командой:

```
./smevql stop
```

Перезапуск СМЭВ QL Сервера осуществляется командой:

```
./smevql restart
```

## 2.3 Работа с сервером

### 2.3.1 Генераторы

Генераторы создают папки и файлы-шаблоны с начальными значениями. Для запуска генератора можно использовать полную команду `./smevql generate` или короткий алиас `./smevql g`.

Новый пустой источник генерируется командой:

```
./smevql g source <source-name>
```

Пример источника на основе Prostore:

```
prostore_source:
type: rest
version: '1.0'
adapter: prostore
protocol: http
host: smevql-dtm-prostore01.ru-central1.internal
port: 9090
path: api/v1/datamarts/query?format=json
headers:
    - content-type: application/json
threads-count: 4
connection-timeout: 30
```

Новая модель генерируется командой:

```
./smevql g model <model-name>
```

Пример модели:

```
resources:
- mo: *base_model
    name: Медицинская организация
    description: Логическая таблица "Медицинская организация"
    fields:
```

```
    <<: *default_fields
    parent_id:
        <<: *ds
        name: parent_id
    update_ts:
        <<: *dts
        name: update_ts
    address:
        <<: *ds
        name: address
    address_fias_guid:
        <<: *ds
        name: address_fias_guid
    enabled:
        <<: *ds
        name: enabled
    name:
        <<: *ds
        name: name
    region_okato:
        <<: *ds
        name: region_okato
    create_ts:
        <<: *dts
        name: create_ts
    id:
        <<: *pks
        name: id
    rmis_id:
        <<: *ds
        name: rmis_id
    phone:
        <<: *ds
        name: phone
    connections:
    has_many: []
    belongs_to:
    - attachment:
        primary_key: [ mo_id ]
        foreign_key: [ id ]
    - resource:
        primary_key: [ mo_id ]
        foreign_key: [ id ]
    extract:
    source:
        - name: prostore
        table: misdm02.mo
- profilecode_resource: *base_model
- resource: *base_model
- observation: *base_model
- book: *base_model
- slot: *base_model
- monitoring: *base_model
- referral: *base_model
- attachment: *base_model
- patient: *base_model
- service: *base_model
- unaccessible_period: *base_model
```

Из существующего Prostore модель генерируется командой:

```
./smevql schema-gen test -h localhost -p 9090 -d demo_view
```

- `test` - имя директории, куда будет выгружена модель;
- `-h localhost -p 9090` - это хост и порт Prostore;
- `-d demo_view` - витрина (схема).

### 2.4 Сборка проекта

Собрать проект можно с помощью gradle:

```
./gradlew clean build
```

## 3 Основные понятия СМЭВ QL

### 3.1 Ресурс

Ресурс представляет собой объект (таблица) хранилища Поставщика, данные которого могут быть получены или изменены Потребителем при запросе к СМЭВ QL серверу. Ресурс описывается в модели данных СМЭВ QL, а так же указывается в исходном запросе от Потребителя.

Пример определения ресурса в модели данных СМЭВ-QL:

```
resources:
  - tickets: *base_model
    name: Билеты
    fields: "id", "passengerid", "tripid", "number", "bycard", "price", "sold"
```

Пример определения ресурса в запросе Потребителя:

```
"query": {
       "tickets": {
           "conditions": {
              "number": 1
           },
           "attributes": ["id", "passengerid", "tripid", "number", "bycard", "price",
"sold"]
       }
```

### 3.2 Машина состояний

СМЭВ QL содержит встроенную машину состояний (или стэйт-машина) для изменения ресурсов витрин данных. Одновременно с этим стейт машина может в качестве подтверждения перехода состояния использовать внешний источник.

Карта состояний и переходов машины состояний описывается в виде YAML-файла `state.yaml` располагаемого в папке `states/<имя-модели>/<x.x версия модели>` инстанса СМЭВ QL сервера.

### 3.3 Базовая модель данных

Базовая модель данных - это описание типовых элементов и форматов данных, которые могут использоваться (наследоваться) при создании пользовательских моделей (Модель данных).

Базовая модель не описывает схему данных витрины и не участвует в логике СМЭВ QL при обработке запросов. Хранится в файловой системе СМЭВ QL сервера по пути: `models>base>1.0>model.yaml`

### 3.4 Модель данных

Модель данных (пользовательская модель данных, custom-модель) - описание

метаданных витрины, извлекаемых при обращении потребителя к СМЭВ QL серверу. Содержит названия ресурсов (таблиц), атрибутов ресурсов, связи с другими моделями, ограничения на использование ресурсов для потребителей, а так же источники хранения ресурсов.

Каждая модель представляет из себя YAML-файл, которая может ссылаться на базовую модель данных (Базовая модель данных) для оптимизации описания.

Хранится в виде YAML-файла `model.yaml` по пути: `models>custom>1.0>model.yaml` инстанса СМЭВ QL сервера.

### 3.5 Распределённый запрос

При обращении Потребителя к СМЭВ QL серверу, исполнение запроса (получение данных по запросу) может происходить с участием одного или нескольких источников (поставщиков данных). В случае если для получения данных шло обращение к двум и более источникам, то такой запрос называется распределенным.

Описание источников получения данных задается на уровне Модель данных в блоке `extract:` sources_by_conditions. При этом Потребитель всегда обращается к одному СМЭВ QL, от которого ожидает получение всех данных о всех источников.

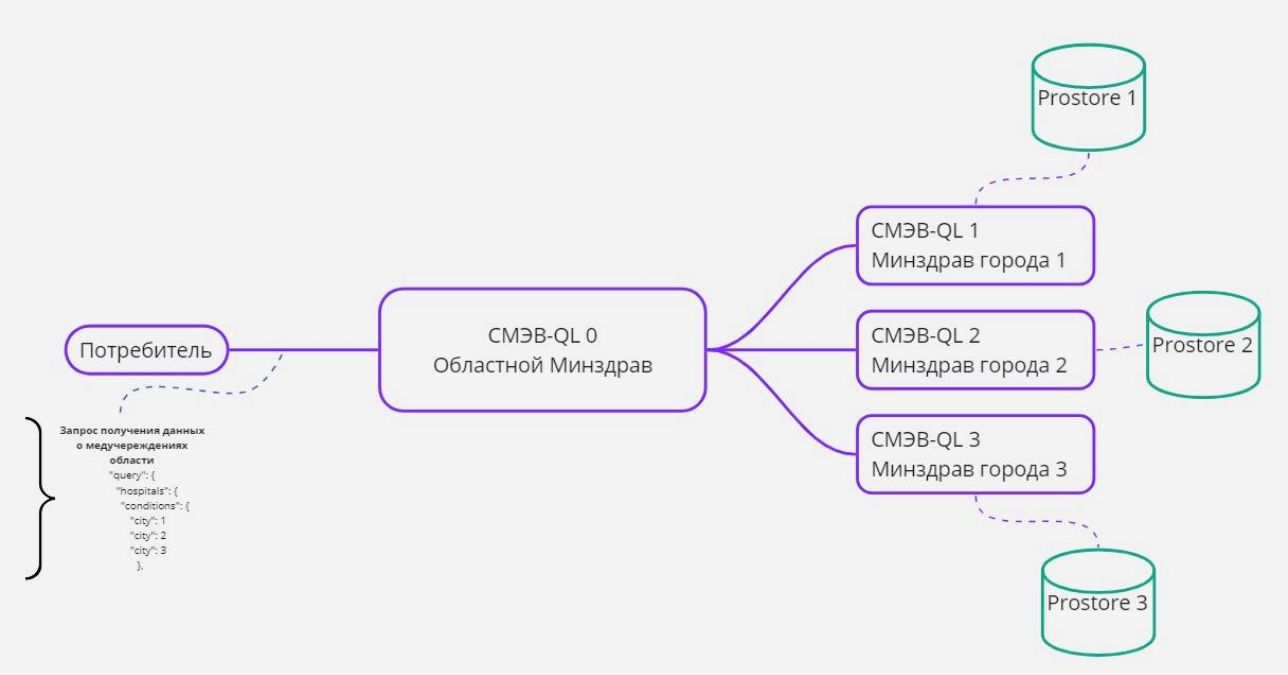



Рисунок - 3.10 Пример схемы распределенного запроса

## 4 Функции СМЭВ QL Сервера

### 4.1 Администрирование и конфигурирование

#### 4.1.1 Создание СМЭВ QL Сервера

Данная функция позволяет создать рабочий экземпляр СМЭВ QL сервера с помощью команды, выполняемой утилитой.

Таблица 4.10 Настраиваемые параметры

| Параметр | Описание | Где задается |
|---|---|---|
| app name | Имя создаваемого экземпляра СМЭВ QL сервера. Параметр является обязательным | в командной строке `java -jar smevql-server-all.jar new <new-app-name>` |

| | | |
|---|---|---|
| | для заполнения | |

**Предварительное состояние:**

1. Развернут дистрибутив СМЭВ QL сервер.
2. Запущена консоль утилиты для работы со СМЭВ QL.

**Результирующее состояние:**

1. Создана первичная структура папок и файлов СМЭВ QL.
2. Ошибка выполнения команды.

**Сценарий выполнения**

1) Администратор сервера в консоли вводит команду создания нового экземпляра СМЭВ QL с указанием имени:
   ```
   java -jar smevql-server-all.jar new <new-app-name>
   ```
2) Система выполняет создание экземпляра СМЭВ QL сервера с заданным именем:
   - если во время выполнения команды возникла ошибка, то сценарий завершается с результирующим состоянием 2;
   - иначе, команда выполнена успешно, создана первичная структура папок и файлов, завершение сценария с результирующим состоянием 1.

**Результирующее состояние:**

1. СМЭВ QL сервер успешно запущен, создана первичная структура папок и файлов.
2. Ошибка выполнения команды.

**Первичная структура папок и файлов**

Таблица 4.11 Первый уровень вложенности корневой папки **smevql**

| **Название** | **Тип** | **Описание** | **Уровень вложенности** |
|---|---|---|---|
| application.yaml | yaml-file | Файл с описанием настроек СМЭВ QL сервер (подробнее см. Конфигурирование сервера) | 1 |
| credentials.yaml | yaml-file | Файл с описанием представления СМЭВ QL сервер (подробнее см. Конфигурирование сервера) | 1 |
| logs | folder | Папка с лог-файлами СМЭВ-QL сервер | 1 |
| models | folder | Папка с описанием моделей данных | 1 |
| readme.md | md-file | read-файл в формате markdown, описывает основные функции СМЭВ QL | 1 |
| smevql.jar | jar-file | Jar-файл с кодом СМЭВ QL | 1 |
| sources | folder | Папка с описанием источников данных | 1 |
| states | folder | Папка с описанием машин состояний | 1 |
| openapi.yaml | yaml-file | Файл с первичной спецификацией Open API СМЭВ QL | 1 |

Таблица 4.12 Содержимое папки **models**

| **Название** | **Тип** | **Описание** | **Уровень вложенности** |
|---|---|---|---|
| base | folder | Папка с версиями описания структуры базовой модели данных | 2 |
| – current (symlink) | symlink | Ссылка на папку модели по умолчанию | 3 |
| – 1.0 | folder | Папка с версией базовой модели данных | 3 |
| – model | yaml-file | Файл с описанием базовой модели данных | 4 |

Содержимое папки **sources**

По умолчанию, пустая папка, содержимое заполняется в процессе создания моделей источников.

Таблица 4.13 Содержимое папки **logs**

| Название | Тип | Описание | Уровень вложенности |
|---|---|---|---|
| environment_name.log | log-file | Лог-файл СМЭВ QL | 2 |

Содержимое папки **states**:

По умолчанию, пустая папка, содержимое заполняется в процессе создания машин-состояний.

### 4.1.2 Конфигурирование СМЭВ-QL сервер

Пример конфигурации файла `application.yml` для **СМЭВ QL сервера** см. в разделе Конфигурирование сервера Руководства администратора.

### 4.1.3 Запуск, остановка, перезапуск приложения СМЭВ QL сервер

Примеры команд и их описание приведено в Быстрый старт.

### 4.1.4 OpenAPI СМЭВ-QL

#### 4.1.4.1 Общее описание

Обращение потребителей данных или иных внешних систем к СМЭВ QL серверу происходит путем вызова REST-методов. Первичная спецификация Open API СМЭВ QL сервера поставляется в исходном дистрибутиве, которая в дальнейшем может обновляться на основании модели данных и модели машины-состояний.

Спецификация Open API хранится в файловой системе СМЭВ QL сервера по пути: `smevql>openapi.yaml`

#### 4.1.4.2 Описание спецификации

Таблица 4.14 Описание спецификации

| Группа | Метод | Назначение |
|---|---|---|
| States (работа с моделью машины состояний) | GET/states | Получить описание всех моделей машин состояний |
| | GET/states/{model} | Получить описание конкретной модели машины состояний |
| | POST/states/{model}/{event} | Изменить статус (состояние) машины состояний. Спецификация метода строится на основании заполненной карты машины состояний. |
| Model (работа с моделью данных) | GET/model | Получить описание всех моделей данных |
| | GET/model/{model} | Получить описание конкретной модели данных |
| | GET/model/{model}/{version} | Получить описание конкретной версии модели данных |
| | GET/server/indexes/required | Получить список рекомендованных для создания индексов полей. Строится на используемых в связях моделей (connections) и блоке `conditions.allowed` моделей |
| Sources (работа с моделью источников) | GET/sources | Получить описание модели источников |

| | | |
|---|---|---|
| Data<br>(работа с данными поставщика) | GET/data/{token} | Запрос данных в асинхронном режиме по токену |
| | GET/certificaties | Получить сертификат для проверки цифровой подписи на стороне клиента |
| | POST/data | Ключевой запрос получения данных поставщика. Схема запроса по умолчанию не содержит объекты витрины, а обновляется отдельной процедурой на основании заполненной Модель данных |
| Push<br>(управление рассылками на изменения данных витрины) | POST/push/consumer/create | Зарегистрировать нового потребителя на получение уведомлений при изменении данных витрины |
| | GET/push/consumer | Получить список всех потребителей, зарегистрированных на получение уведомлений при изменении данных витрины |
| | GET/push/consumer/resources/{agent_target} | Получить список ресурсов, изменения которых, отслеживает заданный потребитель |
| | GET/push/consumer/agent_targets/{resource} | Получить список потребителей, которые отслеживают изменения заданного ресурса |
| | POST/push/consumer/delete/resource | Удалить отслеживание изменений конкретного ресурса для заданного потребителя |
| | POST/push/consumer/delete/ | Удалить отслеживание изменений всех ресурсов для заданного потребителя |
| Прочие служебные методы | GET/ping | Проверить сетевую доступность СМЭВ QL сервер |

Подробное описание всех эндпоинтов в разделе Описание эндпоинтов.

#### 4.1.4.3 Обновление OpenAPI на основании изменений модели данных

Для применения изменений модели (или применения новой модели) данных в Open API СМЭВ QL необходимо выполнить следующие действия:

1. Администратор системы выполняет команду запуска или перезапуска приложения (подробнее см. Быстрый старт)
2. Система определяет актуальную версию модели (старшая по номеру или версия явно указанная в `current symlink`).
3. Система дополняет схемы OpenAPI атрибутами из модели.
4. Система сохраняет обновленные данные Open API только в оперативной памяти (локальный файл `openapi.yaml` не меняется).

Для того, чтобы изменения модели были применены в локальном файле `openapi.yaml`, необходимо выполнить следующие действия:

1. Администратор системы выполняет в консоли команду генерации OpenAPI:

```
yaml g openapi model
```

2. Система определяет актуальную версию модели данных (старшая по номеру или версия явно указанная в `current symlink`)
3. Система дополняет схемы OpenAPI атрибутами из модели.
4. Система обновляет локальный файл `openapi.yaml`.

### 4.1.5 Регистрация OpenAPI СМЭВ QL в СМЭВ4

Для возможности использования СМЭВ QL сервер, необходимо выполнить регистрацию OpenAPI в СМЭВ4 через ЕИП НСУД и выдать права доступа потребителям на

выполнение запросов вида - Обмен с использованием регламентированных запросов типа «Rest-сервис» в системе ЛК УВ.

При регистрации REST-интерфейса потребуется указать следующие ключевые атрибуты:

1. Мнемоника информационной системы поставщика данных.
2. Указание префикса url REST-интерфейса СМЭВ QL сервер, например: `smevql/api/v1`
3. Приложить заполненный файл openapi.yaml (подробнее см. OpenAPI СМЭВ-QL)

## 4.2 Работа с моделями

### 4.2.1 Создание базовой модели

#### 4.2.1.1 Общее описание

Базовая модель данных автоматически генерируется при создании СМЭВ QL и не должна редактироваться вручную.

Базовая модель служит для унификации описания типов данных и позволяет значительно сокращать описание атрибутов ресурса в Модель данных.

Пример описания атрибута в модели данных без применения базовой модели:

```yaml
id:
  name: Идентификатор
  type:
    - string
    - STRING
  length: 0
  nullable: null
  key: NONE
```

Пример описания атрибута в модели данных с применением базовой модели:

```yaml
id: *ds
```

#### 4.2.1.2 Общий сценарий выполнения

1. Администратор сервера в консоли вводит команду создания экземпляра СМЭВ QL сервер:

```
java -jar smevql-server-all.jar new (new-app-name)
```

2. Система создаёт новую версию папки и файла базовой модели с заполненным содержимым - `model.yaml`

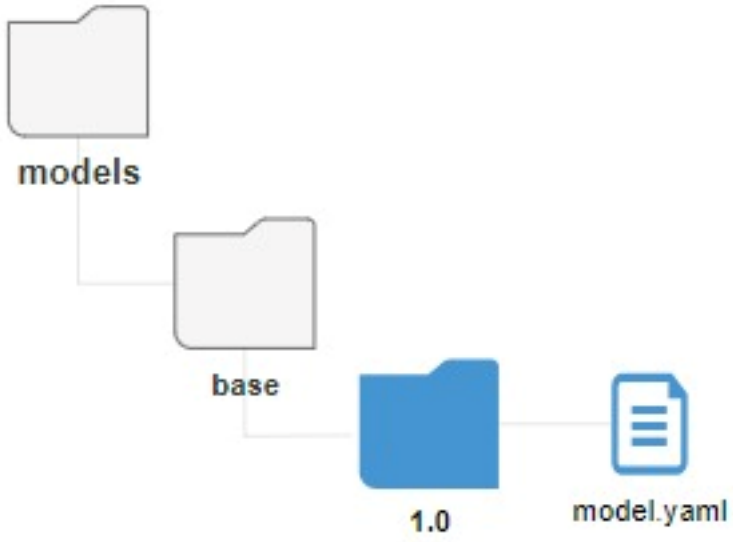



Рисунок - 4.18 Создание базовой модели

#### 4.2.1.3 Описание базовой модели

```yaml
default_string: &ds
  name: Строка
  type:
      - string
      - STRING
  length: 0
  nullable: NULL
  key: NONE
  source: NONE

default_number: &dn
  name: Число
  type:
      - integer
      - INTEGER
  length: 0
  nullable: NULL
  key: NONE
  source: NONE

primary_key: &pk
  name: Ключ
  type:
      - string
      - STRING
  length: 0
  nullable: NULL
  key: PRIMAY
  source: NONE

static_value: &sv
  name: Статическое значение
  type:
      - string
      - STRING
  length: 0
  nullable: NULL
  key: NONE
```

```
  source: STATIC //зарезервированное значение
  default_value: Значение //статическое значение без извлечения

env_value: &ev
  name: Значение из окружения
  type:
      - string
      - STRING
  length: 0
  nullable: NULL
  key: NONE
  source: ENV //зарезервированное значение
  default_value: Значение если нет переменной окружения
  env_key: ключ_переменной_окружения

base_model: &base_model
default_fields: &default_fields
```

### 4.2.2 Генерация модели данных

#### 4.2.2.1 Общее описание

Данная функция позволяет сгенерировать новую пустую Модель данных.

#### 4.2.2.2 Сценарий выполнения

1. Администратор сервера в консоли вводит команду генерации новой модели данных с обязательным указанием названия модели:

```
./smevql g model <model-name>
```

2. Система проверяет, что имя модели уникально:
    a. если имя модели не уникально (модель данных с таким именем уже существует в файловой системе СМЭВ QL), то Система выводит соответствующее сообщение об ошибке.
    b. иначе, имя модели уникально, переход к выполнению следующего шага.
3. Система создаёт папку с указанным наименованием и файл `model.yaml`
4. Далее Администратор сервера вручную заполняет модель данных требуемыми значениями.

#### 4.2.2.3 Описание модели данных

| Элемент модели | Описание | Пример |
|---|---|---|
| – name: | Название модели данных | smevql |
| version: | Номер версии модели данных | 1.0 |
| resources: | Описание ресурсов модели данных | см. Пример блока resources |
| <resource_name>: *base_model | Техническое название ресурса. Должно быть уникальным в рамках модели данных, т.к. используется для указания связей | |
| name: | Название ресурса на русском языке | |
| description: | Дополнительное описание ресурса, обычно указывается его бизнес-смысл | |
| fields: | Список полей ресурса. Для каждого поля указывается:<br>– название поля;<br>– тип данных (возможные типы данных определены в | |

| | | |
|---|---|---|
| | Базовая модель данных);<br>– опционально источник данных (source) в том случае, если он отличается от источника данных всего ресурса;<br>– опционально ограничение на вывод данного поля в результатах запроса (Пример блока fields:). | |
| connections: | Описание связей между ресурсами по заданным ключам | см. Пример блока connections |
| belongs_to: | | |
| has_many: | | |
| conditions: | Описание ограничений на использование полей в пользовательских запросах | см. Пример блока conditions |
| allowed: | Если заполнено, то поиск разрешен только по этим полям и полям с ключами | |
| denied: | Если заполнено, то поиск запрещен по этим ключам | |
| always: | Наличие условий в блоке `always` должно ко всем запросам ресурса добавлять эти условия, если указаны в запросе, то перетирать | |
| extract: | Указание источника данных для ресурса | см. Пример блока extract |
| name: | Название (мнемоника) источника данных | |
| table: | Название таблицы в схеме БД источника | |
| conditions: | Массив условий применения источника на основании значений полученных в запросе атрибутов.<br>Все условия внутри источника действуют через **логическое И (AND)**.<br>Если запрос подходит под два и более источников , то данные **извлекаются и объединяются из всех подходящих**. | см. Пример блока conditions |
| fetch: | Режим получения данных (синхронный или асинхронный). Содержит атрибуты:<br>– policy - режим работы, поддерживается режимы `sync \| async \| async_on_timeout` (значение по умолчанию sync);<br>– timeout - таймаут получения данных клиента (значение по умолчанию PT20S);<br>– store_timeout - время хранения временных данных для асинхронного режима работы (значение по умолчанию PT15M);<br>– retries - число попыток получения данных (значение по | см. Пример блока fetch |

| | | |
|---|---|---|
| | умолчанию 1). | |

#### 4.2.2.4 Пример блока `fields`

```yaml
fields:
    <<: *default_fields
    first_name: *ds
    last_name:
        <<: *ds
        guard: [last_name]
    snils:
        <<: *ds
        guard: [last_name first_name snils]
```

#### 4.2.2.5 Пример блока `resources`

```yaml
resources:
# slots — техническое название модели, по нему производятся все связи
- slots: *base_model
# значение name — название модели на русском языке
    name: Слоты
# description - дополнительное описание ресурса
    description: таблица с указанием доступных и занятых временных интервалов
# fields — список полей модели
    fields:
# список полей может включать перечень полей из default_fields
    <<: *default_fields
# поля по-умолчанию наследуются от ds (default_string) из базовой модели
    id: *ds
    resource_id: *ds
# у поля может быть переопределен source (по-умолчанию у каждого поля источник всей модели)
    type: *ds
        source:
        field: tag_type
    age: *ds
        source:
        field: tag_age
    visitTime: *ds
    duration: *ds
    status: *ds
    create_ts: *ds
    update_ts: *ds
    update_ts: *ds
```

#### 4.2.2.6 Пример блока `connections`

```yaml
connections:
  belongs_to:
    - resource:
        primary_key: [ id ]
        foreign_key: [ resource_id ]
  has_many:
    - book:
        primary_key: [ id ]
        foreign_key: [ slot_id ]
    - unaccessible_period:
        primary_key: [ resource_id, type ]
        foreign_key: [ resource_id, type ]
```

#### 4.2.2.7 Пример блока `conditions`

```yaml
conditions:
    allowed: [id, name] # если заполнено, то поиск разрешен только по этим полям и полям с ключами
    denied: [snils] # если заполнено, то поиск запрещен по этим ключам
    always: # наличие условий в блоке always должно ко всем запросам ресурса добавлять эти условия, если указаны в запросе, то перетирать
    - region: ["=", "77"]
    - blocked: ["=", true]
```

#### 4.2.2.8 Пример блока `extract`

```yaml
extract:
  source:
  - name: prostore
    table: misdm.slots
```

#### 4.2.2.9 Пример блока `conditions`

```yaml
extract:
  source:
    - name: prostore1
      table: misdm.slots
      conditions:
        # попадание в промежуток
        - range:
            field: age
            from: 0
            to: 2
        - eq:
            field: color
            not: "blue"
    - name: prostore2
      table: misdm.39slots
      conditions:
        # ограничения по (не)равенству
        - eq:
            field: resource_id
            is: 1
    - name: prostore3
      table: misdm.39slots
      conditions:
        # ограничения по наличию в источнике
        - eq:
            field: snils
            extract:
            source: redis
            table: default_table
            key: resource_hashed_id
            algorithm: md5
            # select count(*) > 0 from offices.offices where resource_hashed_id = ?
            # параметр: md5(snils)
            is: true
    - name: prostore_default
      table: misdm.39slots
      conditions:
        - fallback: true
```

#### 4.2.2.10 Пример блока `fetch`

```
fetch:
  policy: sync | async | async_on_timeout # на данном этапе реализуется только sync
  store_timeout: PT15M
  timeout: PT20S
  retries: 2
```

### 4.2.3 Автоматическое создание модели данных на основе схемы БД

#### 4.2.3.1 Общее описание

Данная функция позволяет создать заполненную модель данных, на основе схемы БД подключенного источника (на основе схемы данных Prostore).

#### 4.2.3.2 Сценарий выполнения

**Предварительные условия:**

1. Настроена модель источника данных типа Prostore (см. Создание модели источников).
2. Создана базовая модель (см. Создание базовой модели).

**Ход выполнения:**

1. Администратор сервера в консоли вводит команду генерации модели данных на основе схемы БД источника Prostore:

```
./smevql schema-gen -d <название схемы>
```

2. Система проверят наличие базовой модели:
    a. если базовая модель отсутствует, то Система выводит соответствующее сообщение об ошибке.
    b. иначе, переход к выполнению следующего шага.
3. Система проверяет, что имя модели (название схемы) уникально:
    a. если имя модели не уникально (модель данных с таким именем уже существует в файловой системе СМЭВ QL), то Система выводит соответствующее сообщение об ошибке.
    b. иначе, имя модели уникально, переход к выполнению следующего шага.
4. Система создаёт папку с указанным наименованием и файл `model.yaml`
5. Система заполняет содержимое файла model.yaml по следующим правилам:
    a. Каждая таблица в схеме БД - это отдельный ресурс в модели данных.
    b. Каждый атрибут таблицы в схеме БД - это элемент блока fields: соответствующего ресурса.
    c. Блок `connections:` - пустой.
    d. Блок `conditions:` - пустой.
    e. Блок `extract:` - `name: <название источника из модели источника>`, `table: <название таблицы из БД>`

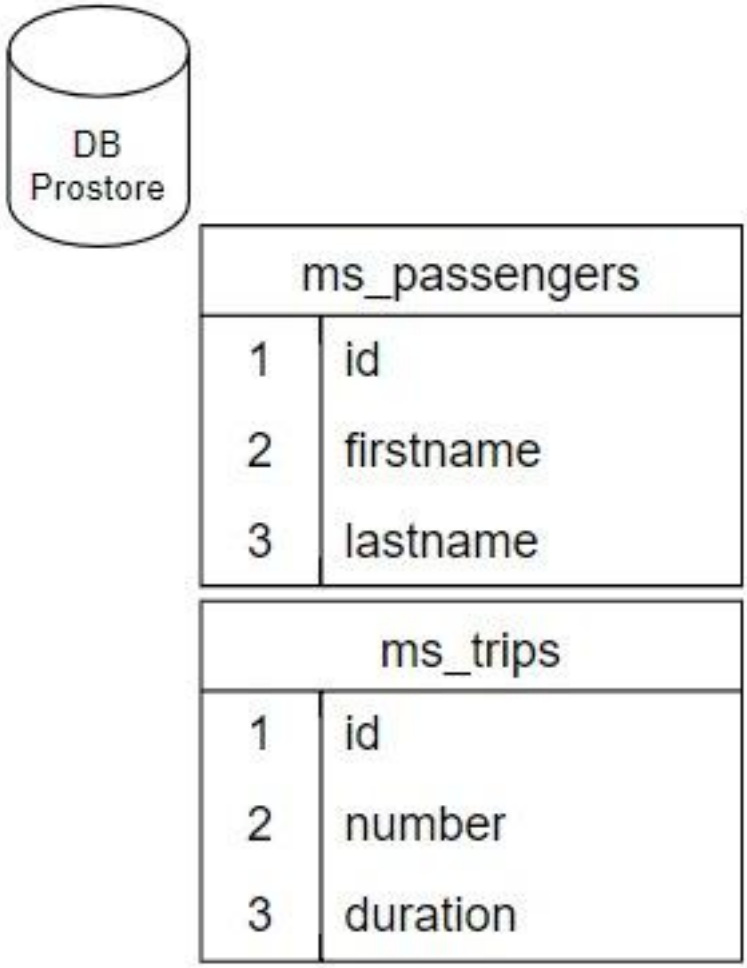



Рисунок - 4.19 Схема в БД Prostore

Сгенерированная модель данных:

```
- name: smevql
    version: 1.0
    data:
      resources:
        - passenger: *base_model
          name: Пассажиры
          description: Логическая таблица "Пассажиры"
          fields:
            id:
              !!merge <<: *ds
              name: ИД пассажира
            firstname:
              !!merge <<: *ds
              name: Имя
            lastname:
              !!merge <<: *ds
              name: Фамилия
           connections:
           extract:
            source:
              - name: prostore
                table: ms.passengers
          - trip: *base_model
            name: Рейс
            description: Логическая таблица "Рейс"
```

```
    fields:
     !!merge <<: *default_fields
     id:
       !!merge <<: *ds
       name: ИД рейса
     number:
       !!merge <<: *pkn
       name: Номер
     duration:
       !!merge <<: *dtm
       name: В пути
    connections:
    extract:
     source:
       - name: prostore
         table: ms.trips
```

### 4.2.4 Создание новой версии модели данных

#### 4.2.4.1 Общее описание

СМЭВ QL может одновременно использовать несколько разных моделей данных с разным наименованием.

Однако в рамках одного наименования модели (папки с названием модели) может быть активна только одна версия модели данных.

Версия модели данных характеризуется уникальным номером и наличием отдельной папки с данным номером в файловой системе СМЭВ QL.

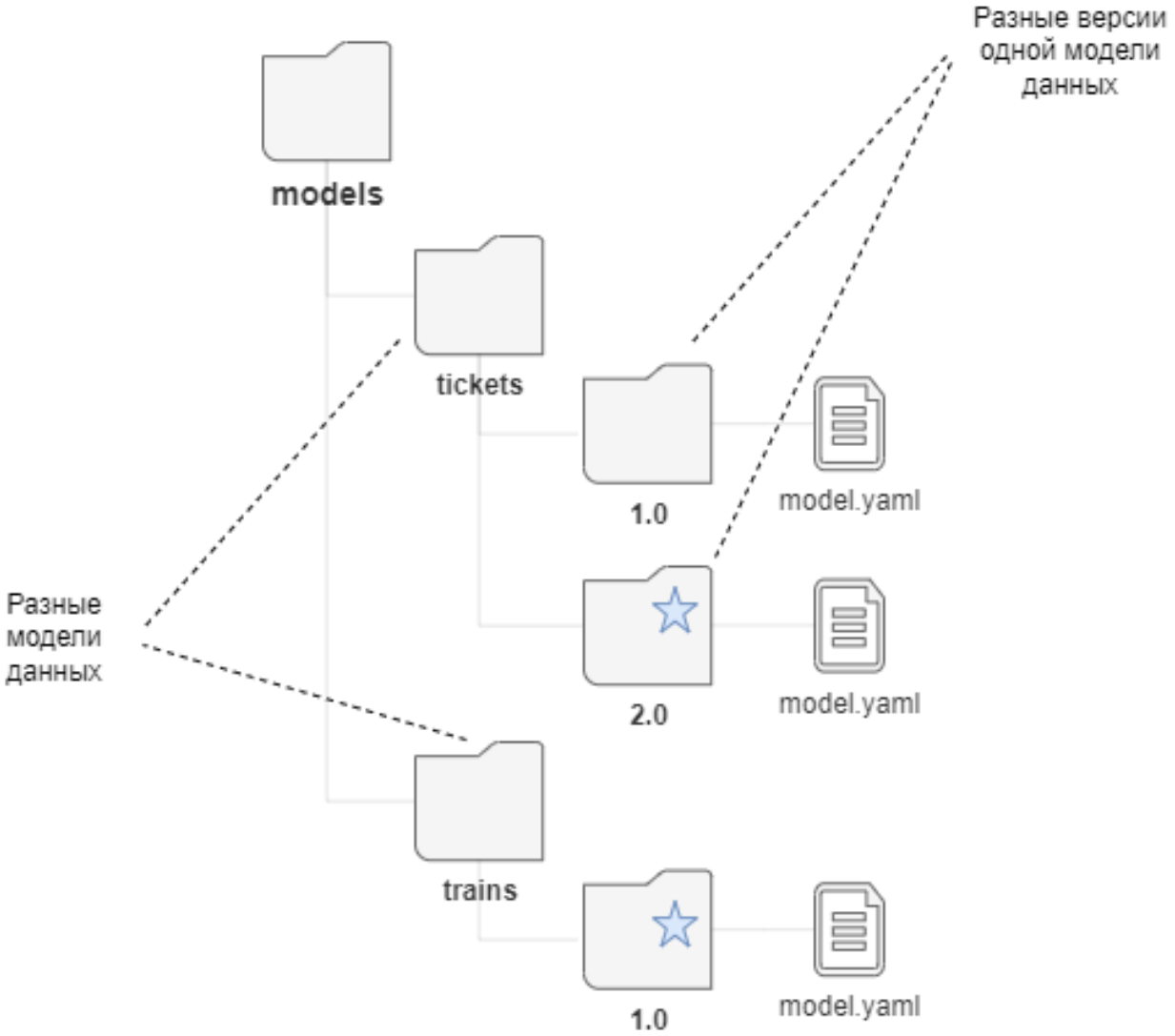



Рисунок - 4.20 Разные версии одной модели

Модели данных считываются, валидируются и загружаются в память из папки `models` при запуске СМЭВ QL сервера.

По умолчанию используется версия `model`, на которую ссылается `symlink current`, при его отсутствии по умолчанию считается **старшая по номеру** версия.

Модели и версии, **начинающиеся с подчеркивания (_) НЕ загружаются**, они находятся в стадии проектирования.

#### 4.2.4.2 Сценарий выполнения

**Предварительные условия:**

1. Подготовлена модель данных с учетом всех необходимых изменений (файл `model.yaml`).

**Ход выполнения:**

1. Администратор системы выбирает папку с названием модели данных для которой необходимо создать новую версию.
2. Администратор системы вручную создает новую папку с указанием нового номера версии (обычно это новое целочисленное значение).
3. Администратор системы вручную переносит, заранее подготовленный файл model.yaml, в папку с новым номером версии.

### 4.2.5 Проверка валидности модели данных

#### 4.2.5.1 Общее описание

Валидация модели данных выполняется при каждом запуске СМЭВ QL сервер и в случае возникновения ошибок, приложение не будет запущено.

Помимо этого можно предварительно проверить корректность заполнения модели данных путем вызова специальной команды в консоли приложения.

#### 4.2.5.2 Сценарий выполнения

1. Администратор сервера в консоли вводит команду проверки валидности модели данных:
    a. если требуется проверить все модели данных, то выполняется команда: `./smevql test model all`
    b. если требуется проверить конкретные модели, то выполняется команда (с указанием через запятую) наименований моделей: `./smevql test model <model-name1>, <model-name2>`
2. Система проверяет корректность заполнения моделей данных и в случае наличия ошибок выведет соответствующее сообщение.

### 4.2.6 Маппинг типов данных СМЭВ QL - Prostore

| **Логический тип Prostore** | **Описание** | **Логический тип СМЭВ QL** | **Тип базовой модели СМЭВ QL** |
|---|---|---|---|
| BOOLEAN | Логический (булевый) тип | BOOLEAN | &db, &pkb |
| CHAR (n) | Строка ограниченной длины (n символов). Размерность строки обязательна | STRING | &ds, &dst, &denv, &pks |
| VARCHAR [(n)] | Строка ограниченной длины (n символов). Размерность строки опциональна | STRING | &ds, &dst, &denv, &pks |
| LINK | Строка неограниченной длины. Предназначена для ссылочных полей | BINARY | &dbn, &pkbn |
| UUID | Строка ограниченной длины (36 символов) | STRING | &ds, &dst, &denv, &pks |
| INTEGER, алиас — INT32 | Целое число фиксированной длины со знаком в диапазоне от `-2147483648` до `2147483647` | SHORT, INTEGER | &pksh, &dsh, &pkn, &dn |
| BIGINT, алиас — INT64 | Целое число фиксированной длины со знаком в диапазоне [$-2^{63}$, $2^{63}-1$] [1] | LONG | &pkl, &dl |
| DOUBLE | Число с плавающей запятой с двойной точностью | DOUBLE | &dd, &pkd |
| FLOAT | Число с плавающей запятой | FLOAT | &pkf, &df |
| DATE | Дата (без времени суток) | DATE | &ddt, &pkdt |
| TIME, TIME (p) | Время (без даты).<br>Значение p задает точность отображаемого времени.<br>Возможные значения: от 0 (секунды) до 6 (микросекунды).<br>Значение по умолчанию — 6. Количество микросекунд находится в диапазоне от 0 до 86399999999 | TIME | &dtm, &pktm |
| TIMESTAMP, | Дата и время. | TIMESTAMP | &pkts, &dts |

[1] $-2^{63}$ = -9 223 372 036 854 775 808, $2^{63}-1$ = 9 223 372 036 854 775 807

| | | | |
|---|---|---|---|
| TIMESTAMP (p) | Значение p задает точность отображаемого времени.<br>Возможные значения: от 0 (секунды) до 6 (микросекунды).<br>Значение по умолчанию — 6 | | |
| | Не поддерживается в Prostore | BYTE | &dbt, &pkbt |
| | Не поддерживается в Prostore | BIG_DECIMAL | &dbd, &pkbd |

## 4.3 Работа с источниками данных

### 4.3.1 Создание модели источников

#### 4.3.1.1 Общее описание

Данная функция позволяет добавлять новую версию описания модели источников данных СМЭВ-QL.

В качестве источников данных для СМЭВ-QL сервер могут выступать:

1. REST-интерфейс витрины данных (Prostore);
2. Другой СМЭВ-QL сервер

Модель источников данных СМЭВ-QL хранится в файловой системе СМЭВ-QL сервера по пути: `sources/custom/1.0/source.yaml`

Допускается описание всех источников в рамках одной модели.

#### 4.3.1.2 Общий сценарий выполнения

1. Администратор сервера в консоли вводит команду генерации новой пустой модели источника:

```
./smevql g source <source-name>
```

2. Система создаёт новую версию папки и файла модели источника с пустыми значениями - `source.yaml`
3. Администратор сервера открывает на редактирование файл модели источника `source.yaml` и заполняет параметры необходимыми значениями.

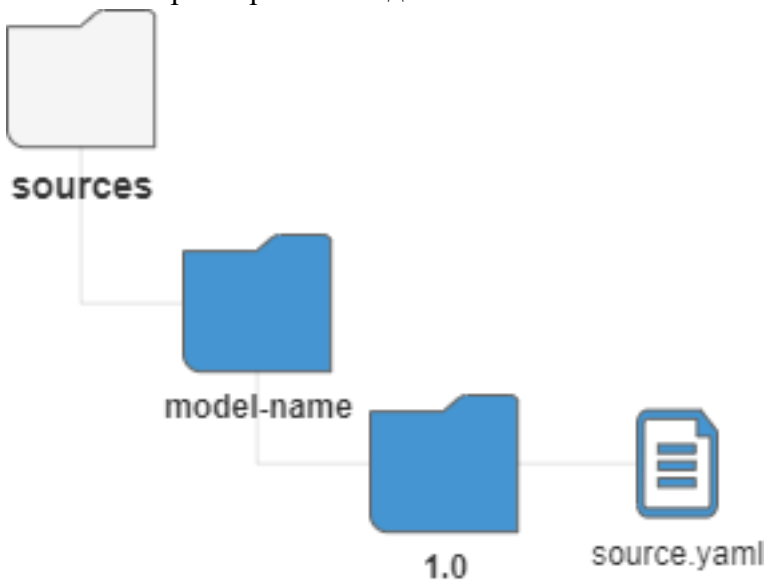



Рисунок - 4.21 Создание модели источников данных

#### 4.3.1.3 Структура source.yaml

Описание источника данных в формате YAML имеет следующую структуру:

Таблица 4.15 Описание источника данных

| Параметр | Описание |
|---|---|
| Наименование | Наименование источника данных.<br>В рамках файла `source.yaml` источник всегда имеет название с постфиксом `_source`.<br>При этом именем источника в моделях данных считается часть **без** учета данного постфикса.<br>То есть источник `egrn_source` в модели данных `model.yaml` необходимо указывать как `egrn`. |
| type | Тип интеграционного взаимодействия. На текущий момент поддерживается только rest |
| version | |
| adapter | Тип источника данных. Может принимать значения:<br>– prostore<br>– smevql |
| protocol | Указание протокола передачи данных. На текущий момент поддерживается только HTTP |
| host | Хост-адрес сервера источника данных |
| port | Порт сервера источника данных |
| path | Путь для вызова REST-метода API источника данных |
| template | Шаблон передачи REST-запроса к источнику |
| payload-path | |
| headers | Значения по умолчанию для заголовка REST-запроса к источнику |
| threads-count | |
| connection-timeout | Тайм-аут ожидания ответа от сервера источника |

Описание источника данных в формате YAML имеет следующую структуру:

```yaml
prostore_source:
  type: rest
  version: 1.0
  adapter: prostore
  protocol: http
  host: localhost
  port: 9090
  path: api/v1/datamarts/query?format=json
  template: '{ "query": "%{request}", "queryId": "%{request_id}" }'
  payload-path: result
  headers:
      - accept: application/json
      - content-type: application/json
  threads-count: 10
  connection-timeout: 0

smevql_server_source:
  type: rest
  version: 1.0
  adapter: smevql
  protocol: https
  host: localhost
  port: 9091
  path: api/query?format=json
```

```
  template: '{ "query": "%{request}" }'
  payload-path: result
  headers:
    - accept: application/json
    - content-type: application/json
```

### 4.3.2 Создание новой версии модели источников

#### 4.3.2.1 Общее описание

СМЭВ QL может одновременно использовать несколько разных моделей источников с разным наименованием.

Однако в рамках одного наименования модели (папки с названием модели) может быть активна только одна версия модели источников.

Версия модели источников характеризуется уникальным номером и наличием отдельной папки с данным номером в файловой системе СМЭВ QL.

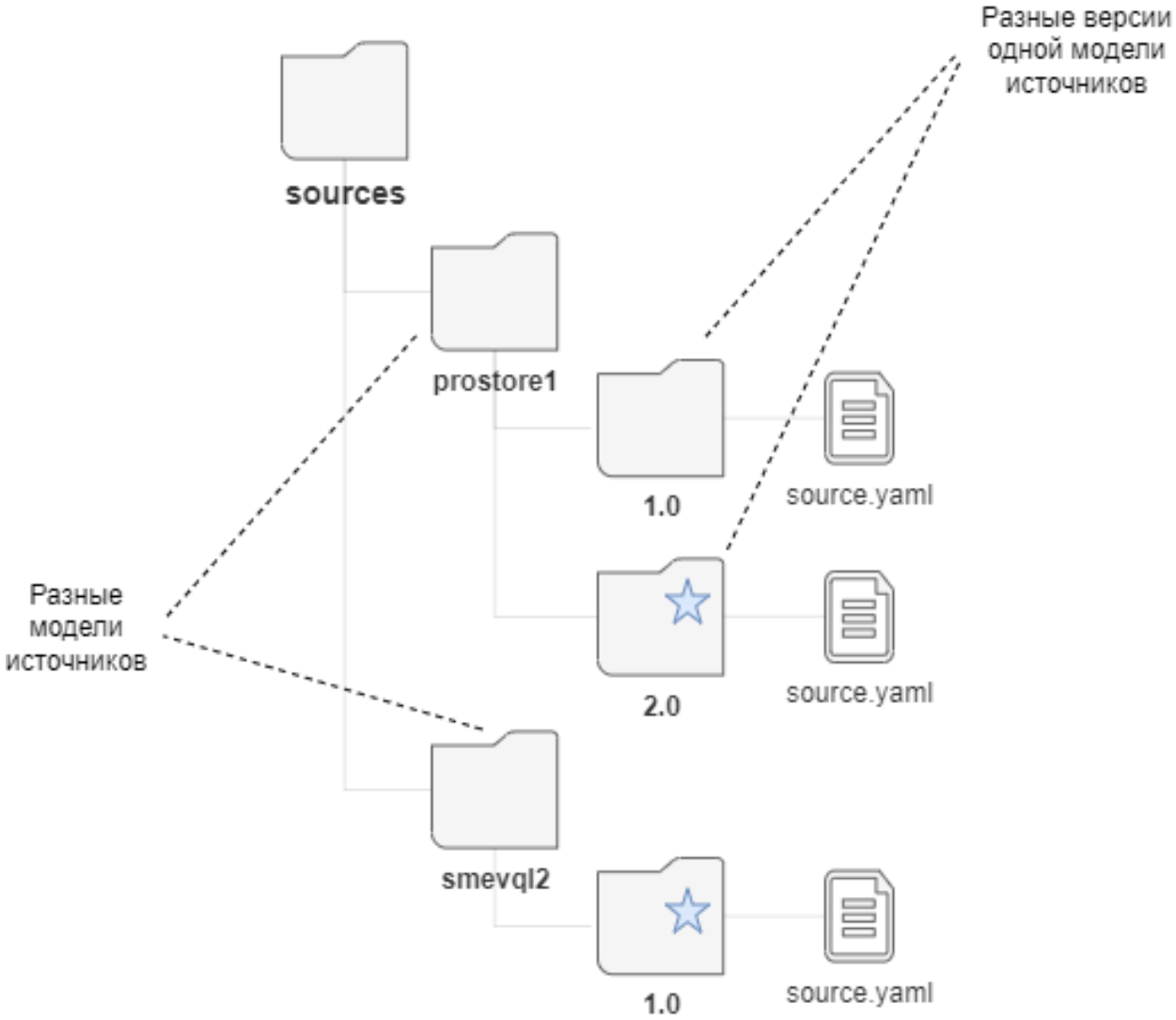



Рисунок - 4.22 Создание новой версии модели источников данных

Источники данных считываются, валидируются и загружаются в память из папки `sources` при запуске СМЭВ QL сервера.

По-умолчанию используется версия источника, на которую ссылается `symlink current`, при его отсутствии по-умолчанию считается **старшая** версия.

Источники и версии, **начинающиеся с подчеркивания (_) НЕ загружаются**, они

находятся в стадии проектирования.

#### 4.3.2.2 Сценарий выполнения

**Предварительные условия:**

1. Подготовлена модель источников с учетом всех необходимых изменений (файл `source.yaml`).

**Ход выполнения:**

1. Администратор системы выбирает папку с названием модели источников для которой необходимо создать новую версию.
2. Администратор системы вручную создает новую папку с указанием нового номера версии (обычно это новое целочисленное значение).
3. Администратор системы вручную переносит, заранее подготовленный файл source.yaml, в папку с новым номером версии.

### 4.3.3 Проверка доступности источников

#### 4.3.3.1 Общее описание

Данная функция позволяет проверить сетевую доступность источников, указанных в модели.

#### 4.3.3.2 Сценарий выполнения

1. Администратор сервера в консоли вводит команду проверки доступности источников:
    a. если требуется проверить все источники, то выполняется команда: `./smevql test source all`
    b. если требуется проверить конкретный источник, то выполняется команда (с указанием через запятую) наименований источников: `./smevql test source <source-name1>, <source-name2>`
2. Система отправляет проверочный запрос к источнику и дожидается ответа в течении заданного таймаута:
    a. если ответ не пришёл в течение заданного таймаута, то Система выводит соответствующее сообщение об ошибке, источник не доступен.
    b. иначе, ответ пришёл в течение заданного таймаута, источник доступен.

## 4.4 Работа с картами машин состояний

### 4.4.1 Создание карты машины-состояний

#### 4.4.1.1 Общее описание

Карта машины состояний (Машина состояний) описывает её статусы и события, при вызове которых осуществляется переход к следующему статусу.

Карта состояний и переходов машины состояний описывается в виде YAML-файла `state.yaml` и хранится в файловой системе СМЭВ QL сервера по пути: `states><model-name>><version number>>state.yaml`

#### 4.4.1.2 Структура карты машины состояний

| Элемент | Описание |
|---|---|
| model: | Название карты машины состояний (модели стэйт-машины) |
| states: | Массив возможных состояний |
| state: | Название состояния |
| attributes: | Массив атрибутов, описывающих состояние. Каждый атрибут имеет следующие |

| | |
|---|---|
| | свойства:<br>– `name` - название атрибута<br>– `value` - значение атрибута |
| initial: | Признак исходного состояния.<br>Может быть определен только у одного состояния в рамках одной карты машины состояний. |
| delete: | Признак, указывающий что при переходе в данное состояние будут удалены данные в соответствии с условиями переданными в блоке `conditions` исходного запроса от клиента P``OST/states/{model}/{event}``.<br>При этом значения атрибутов для такого статуса не применимы, они игнорируются.<br>Состояние с данным признаком считается конечным и не может использоваться в разделе from при описании событий. |
| static: | Признак, при установке которого не меняется статус, а только передается указанная в описании события часть запроса в блоке confirm к связанному источнику. |
| events: | Список событий изменения состояний.<br>Каждое событие вызывается отдельным методом `POST/states/{model}/{event}`<br>**Важно! В любой модели машины состояний по умолчанию присутствует event с типом init, хотя он и не описан в явном виде.<br>Такое событие является обязательным и при его вызове в витрину добавляется новая запись и устанавливается статус исходного состояния (initial)** |
| event: | Название события |
| from: | Массив состояний из которых возможен вызов события |
| to: | В какое состояние переходит стэйт-машина после наступления события |
| hooks: | Массив связанных событий, которые будут выполняться после перехода в заданное состояние.<br>На текущий момент в рамках hook возможен только вызов событий других стэйт-машин.<br>Каждый hook имеет следующие свойства:<br>– `model` - название карты машины состояний<br>– `event` - название события |
| confirm: | Вызов внешнего REST-метода.<br>Данное действие выполняется до осуществления перехода в рамках вызванного события.<br>Блок содержит следующие свойства:<br>**Описание запроса:**<br>– `source` - название внешнего источника (мнемоника вызываемого источника)<br>– `endpoint` - эндпоинт вызываемого REST-метода<br>– `method` - тип вызываемого REST-метода<br>– `body` - указание того, какое содержимое body необходимо передать при выполнении REST-запроса внешнему источнику. Возможные значения: `full`, `state`, `conditions`, `payload`, `credentials`.<br>Источником для содержимого является соответствующий блок данных в body исходного запроса от клиента (`POST/states/{model}/{event}`)<br>**Описание ответа:**<br>– `accept` - описание условия ответа, при соблюдении которого |

| | |
|---|---|
| | вызов внешнего REST-метода считается успешным.<br>Если данное условие (получение заданного ответа) не выполняется, то переход стэйт-машины завершается ошибкой. Блок содержит следующие свойства:<br>○ `jsonpath` - получаемый в ответе атрибут;<br>○ `eq` - значение получаемого атрибута.<br>– `enrich` - указание необходимости обновления данных на основании полученного ответа.<br>Блок содержит следующие свойства:<br>○ `allow` - признак, указывающий о необходимости обновления данных (`true` - обновлять; `false` - только извлекать, без обновления ресурса);<br>○ `jsonpath` - указание того, из какой части ответа брать данные.<br>При отсутствии объекта по пути `jsonpath` в `confirm` запросе возвращается ошибка 501 «Отсутствует объект при обращении по jsonpath», выполнение запроса прерывается.<br>○ `attributes` - массив атрибутов, которые необходимо взять из ответа.<br>Если массив пустой `[]`, то берутся все атрибуты из указанного `jsonpath`.<br>При несоответствии атрибутов в блоке `attributes` и в `jsonpath` - в **payload** вернутся только те атрибуты, по которым есть соответствия с фиксацией в логах событий уровня WARN о несоответствующих атрибутах: `Отсутствует атрибут *attribute_name" в $.update.payload`.<br>В случае, если в **payload** присутствуют атрибуты, аналогичные тем, что возвращаются при выполнении блока `enrich`, обновляются данные в соответствии со значениями вернувшимися при выполнении блока `enrich`.<br>В этом случае в логах фиксируется запись `"Изменение значений атрибутов: *attribute_name" payload в соответствии со значениями блока enrich"`.<br>Атрибуты **payload** ответа **confirm** обогащают **payload** основного запроса.<br>Обновление данных в витрине в случае обогащения **payload** дополнительными атрибутами происходит независимо от значения updatable. |
| updatable: | Признак, указывающий, что необходимо обновить ресурс в рамках вызываемого события |
| updatable_attributes: | Массив атрибутов, которые необходимо обновить в рамках вызываемого события. Если массив пустой `[]`, то обновляются все атрибуты. Источником данных для обновления является блок `payload` исходного запроса от клиента (`POST/states/{model}/{event}`) |

Пример структуры карты машины состояний:

```yaml
model: slot # имя модели
states: # массив состояний объекта
  - state: available # название состояния available
      attributes: # массив атрибутов, описывающих состояние
      - name: status # состояние определятся значением атрибута status
          value: AVAILABLE # значение атрибута для описываемого состояния
      initial: true
  - state: booked
      attributes:
      - name: status
```

```
        value: RECORDED
  - state: reserved
      attributes:
      - name: status
        value: RESERVED
  - state: cancelled
      attributes:
      - name: status
        value: CANCELED
  - state: blocked
      attributes:
      - name: status
        value: BLOCKED
  - state: deleted
      delete: true
  events: # список событий изменения состояний, из них создаются методы API
  - event: book # создает метод POST /states/slot/book
      from: # массив состояний из которых возможен вызов события
      - available
      - reserved
      to: booked # в какое состояние переводится объект после события
      hooks: # массив связанных событий
      - model: book # после перевода надо вызвать событие init для модели book
        event: init
      confirm:
        source: rmis_rest # названия источника
        endpoint: /booking/book
        method: post
        body: payload # что включать в тело запроса
(full|state|conditions|payload|credentials)
        accept: # условие принятия
          jsonpath: $.status.statusCode
          eq: 0 # ожидаем, что statusCode будет равен 0
        enrich:
        allow: true // по умолчанию false
        attributes: [] // пустой массив — все, а если перечислены, то только эти
        jsonpath: $.update.payload // ожидается что по этому адресу объект с атрибутами
и значениями
  - event: reserve
      from: available
      to: reserved
      updatable: true // по умолчанию false для всех, кроме init, возможность изменять
запись при переводе статуса
      updatable_attributes: [] // массив атрибутов, которые можно обновлять, пустой —
можно все
  - event: block
      from: available
      to: blocked
  - event: cancel
      from:
      - available
      - reserved
      - booked
      - blocked
      to: cancelled
      hooks:
      - model: book
        event: cancel
  - event: delete
      from:
      - available
      - reserved
```

```
    - booked
    - blocked
    - cancelled
    to: deleted
    hooks: # массив связанных событий
    - model: book # после перевода надо вызвать событие delete для модели book
      event: delete
```

#### 4.4.1.3 Сценарий выполнения

1. Администратор системы в произвольном текстовом редакторе создает и заполняет файл (`state.yaml`) с описанием карты машины состояний.
2. Администратор системы создает папку с названием модели стэйт-машины в рамках каталога `states`.
3. Администратор системы создает папку с указанием номера версии модели стэйт-машины в рамках каталога с названием стэйт-машины.
4. Администратор системы вручную переносит, заранее подготовленный файл `state.yaml`, в папку с новым номером версии.
5. Для возможности вызова API методов новой карты машины состояния необходимо обновить openAPI СМЭВ QL (см. Обновление OpenAPI на основании изменений модели данных)

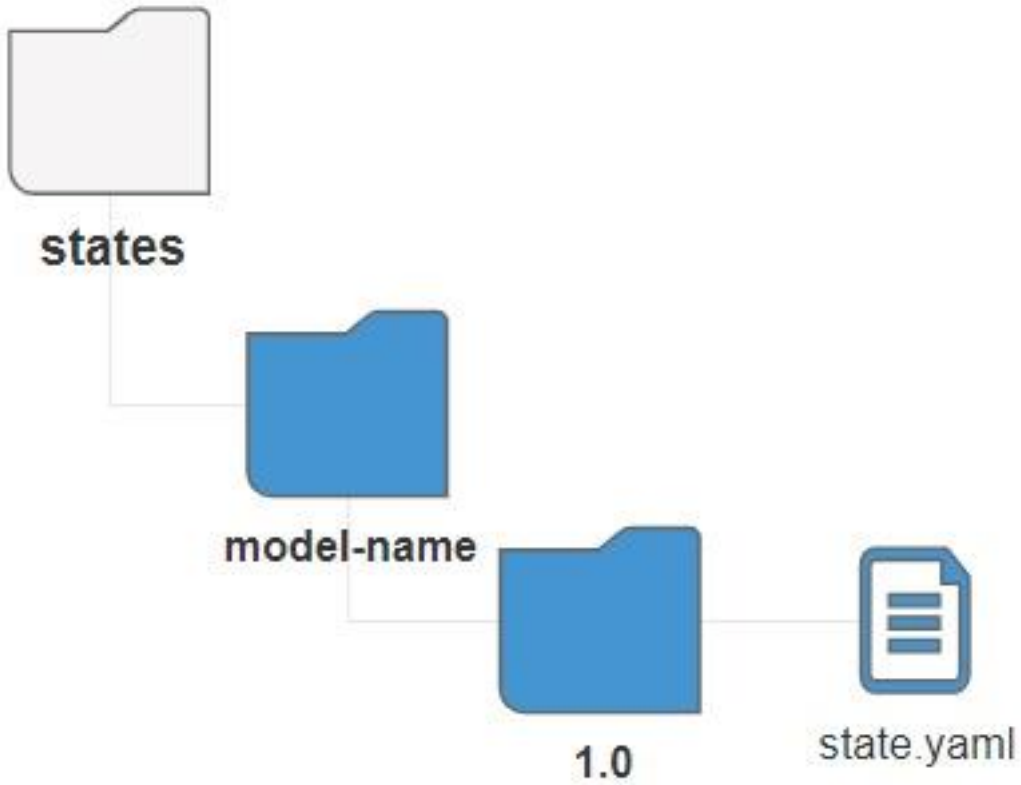



Рисунок - 4.23 Создание карты машины-состояний

### 4.4.2 Создание новой версии карты машины-состояний

#### 4.4.2.1 Общее описание

СМЭВ QL может одновременно использовать несколько разных моделей стэйт-машин с разным наименованием.

Однако в рамках одного наименования модели (папки с названием модели) может быть активна только одна версия модели стэйт-машины.

Версия модели стэйт-машины характеризуется уникальным номером и наличием отдельной папки с данным номером в файловой системе СМЭВ QL.

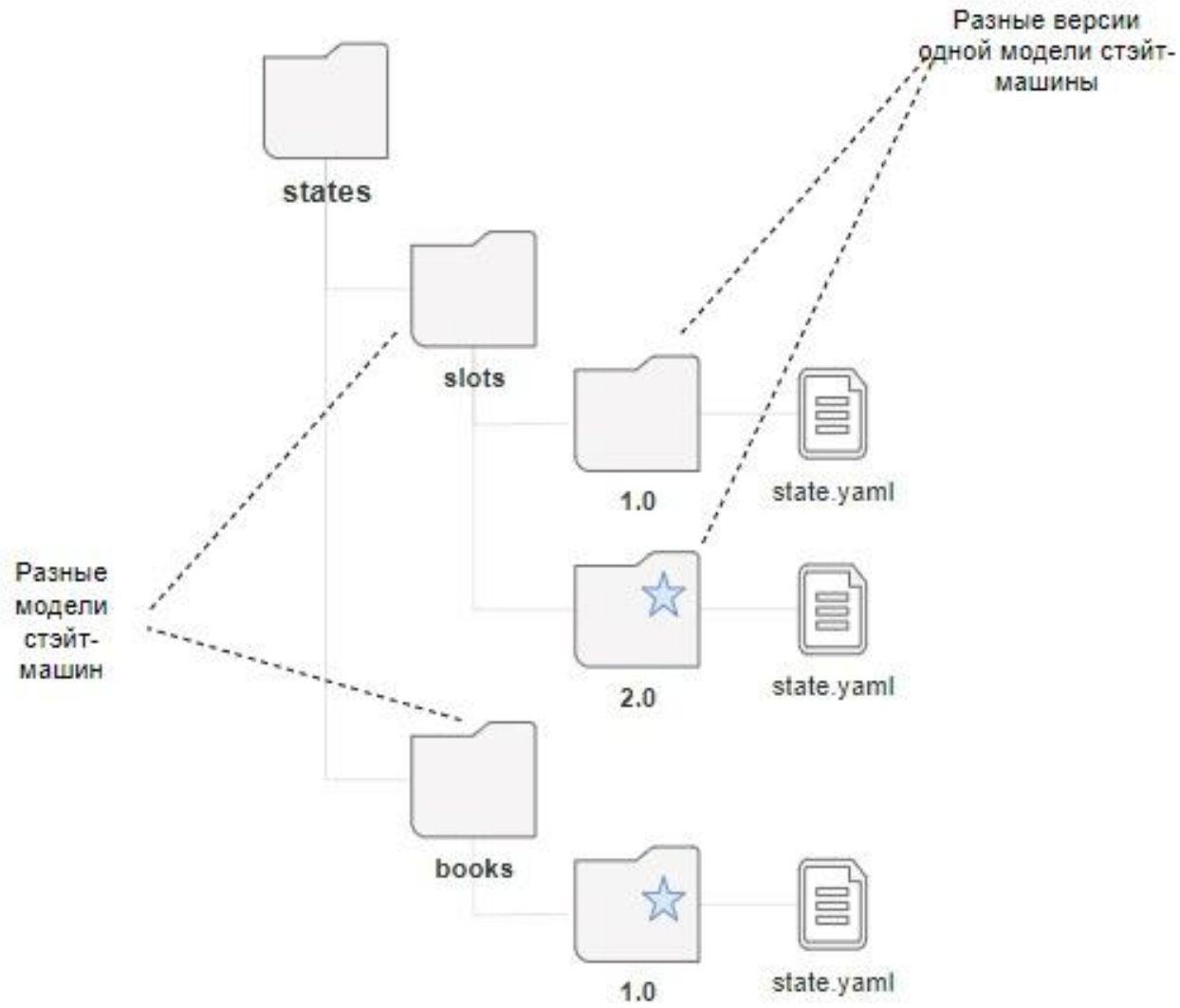



Рисунок - 4.24 Создание новой версии карты машины-состояний

Карты машин состояний считываются, валидируются и загружаются в память из папки `states` при запуске СМЭВ QL сервера.

По-умолчанию используется версия стэйт-машины, на которую ссылается `symlink current`, при его отсутствии по-умолчанию считается **старшая** версия.

Карты машин состояний и версии, **начинающиеся с подчеркивания (_) НЕ загружаются**, они находятся в стадии проектирования.

#### 4.4.2.2 Сценарий выполнения

**Предварительные условия:**

1. Подготовлена модель стэйт-машины с учетом всех необходимых изменений (файл `state.yaml`).

**Ход выполнения:**

1. Администратор системы выбирает папку с названием стэйт-машины для которой

необходимо создать новую версию.

2. Администратор системы вручную создает новую папку с указанием нового номера версии (обычно это новое целочисленное значение).
3. Администратор системы вручную переносит, заранее подготовленный файл state.yaml, в папку с новым номером версии.
4. Администратор перезапускает сервер (Быстрый старт).

## 4.5 Обработка запроса к витрине

### 4.5.1 Запрос получения данных из витрины (POST/data)

#### 4.5.1.1 Общее описание

Для выполнения запроса к СМЭВ QL серверу на получение данных витрины (или нескольких витрин, если запрос распределенный (Раздел 3.5)) потребитель должен вызвать опубликованный метод `POST/data`.

В рамках запроса необходимо передать в теле JSON-объект заданного формата.

POST /data содержит подробное описание.

### 4.5.2 Запрос изменения данных витрины через события машины состояний (POST/states/{model}/{event})

Для выполнения запроса к СМЭВ QL сервер на изменение данных витрины через машину состояний потребитель должен вызвать опубликованный метод `POST/states/{model}/{event}`, где `model` - название карты машины состояний, а `event` - событие карты машины состояний.

В рамках запроса необходимо передать в теле JSON-объект заданного формата.

В рамках вызова событий машины состояний можно выполнять следующие действия:

- Изменить статус машины состояния без обновления данных витрины;
- Обновить данные витрины (добавить новые, изменить текущие);
- Удалить данные витрины;
- Вызывать событие другой машины состояния;
- Добавить/обновить/удалить данные другой витрины.

POST /states/{model}/{event} содержит подробное описание.

### 4.5.3 Обработка запроса получения данных витрины

#### 4.5.3.1 Общее описание

При поступлении запроса от Потребителя на получение данных к СМЭВ QL сервер запускается процесс его обработки, который можно условно разделить на 4 этапа:

1. Проверка запроса и доступов.
2. Формирование плана исполнения запроса.
3. Исполнение плана запроса.
4. Передача ответа потребителю.

Для возможности получения данных Потребителем должны быть соблюдены следующие предварительные условия:

1. Модель данных СМЭВ QL сервера успешно создана.
2. Зарегистрирован метод получения данных витрины `POST/data`.
3. Потребитель обладает правами на выполнение REST-запроса.

#### 4.5.3.2 Проверка запроса и доступов

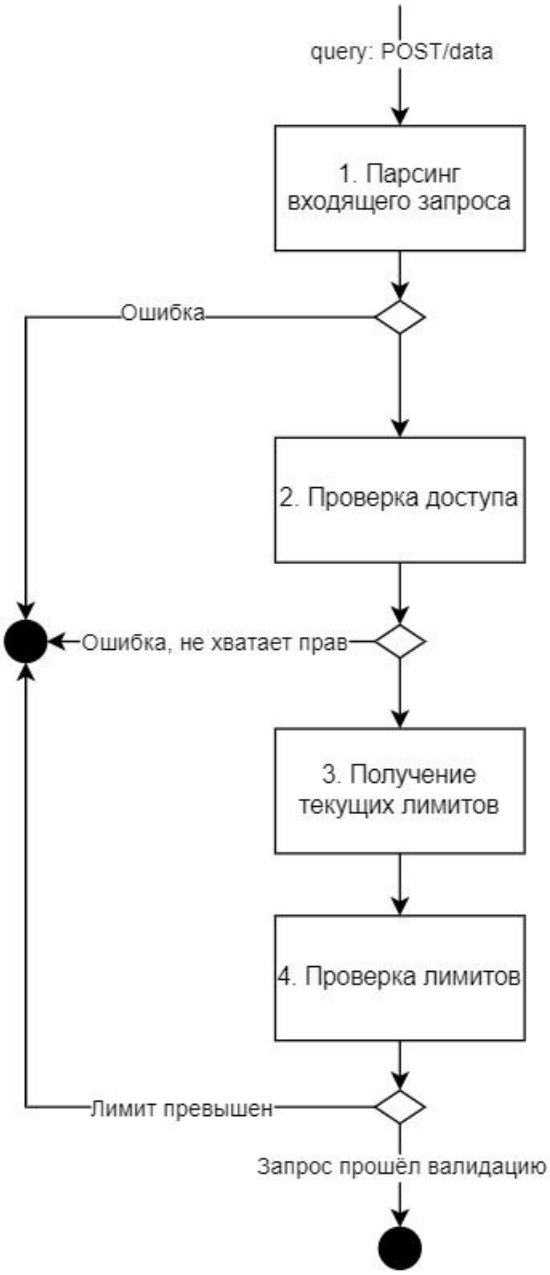



Рисунок - 4.25 Проверка запроса и доступов

**Запускающее событие:**

– Получен запрос получения данных витрины (вызван метод `POST/data`)

**Ход выполнения:**

1. СМЭВ QL (далее Система) проверяет тело запроса на соответствие заданной схеме (Обработка запроса к витрине):
    a. если запрос не соответствует заданной схеме, то Система передаёт соответствующее сообщение об ошибке, процесс завершается
    b. иначе, формат запроса корректен, переход к выполнению следующего шага.
2. Система проверяет доступ ИС Потребителя к данным витрины, для этого сравнивает полученный в запросе идентификатор системы-потребителя (блок `credentials->system`) с перечнем запрещенных и разрешенных, который задается в конфигурационном файле

приложения `application.yaml` (`black_list` и `white_list`):

   a. если доступ запрещен, то Система передаёт соответствующее сообщение об ошибке, процесс завершается
   b. иначе, доступ разрешен, переход к выполнению следующего шага.

3. Система получает из блока персистентности Redis текущие значения лимитов на выполнение запросов.
4. Система проверяет превышение установленных лимитов на выполнение запроса Потребителя к заданным ресурсам. Для этого сравнивает текущие полученные значения на шаге 3 с настройками лимитов, определенных в конфигурационном файле приложения `application.yaml` (`limits`):

   a. если допустимый лимит превышен, то Система передаёт соответствующее сообщение об ошибке, процесс завершается
   b. иначе, лимиты не превышены, переход к выполнению этапа «Формирование плана исполнения запроса».

#### 4.5.3.3 Формирование плана исполнения запроса

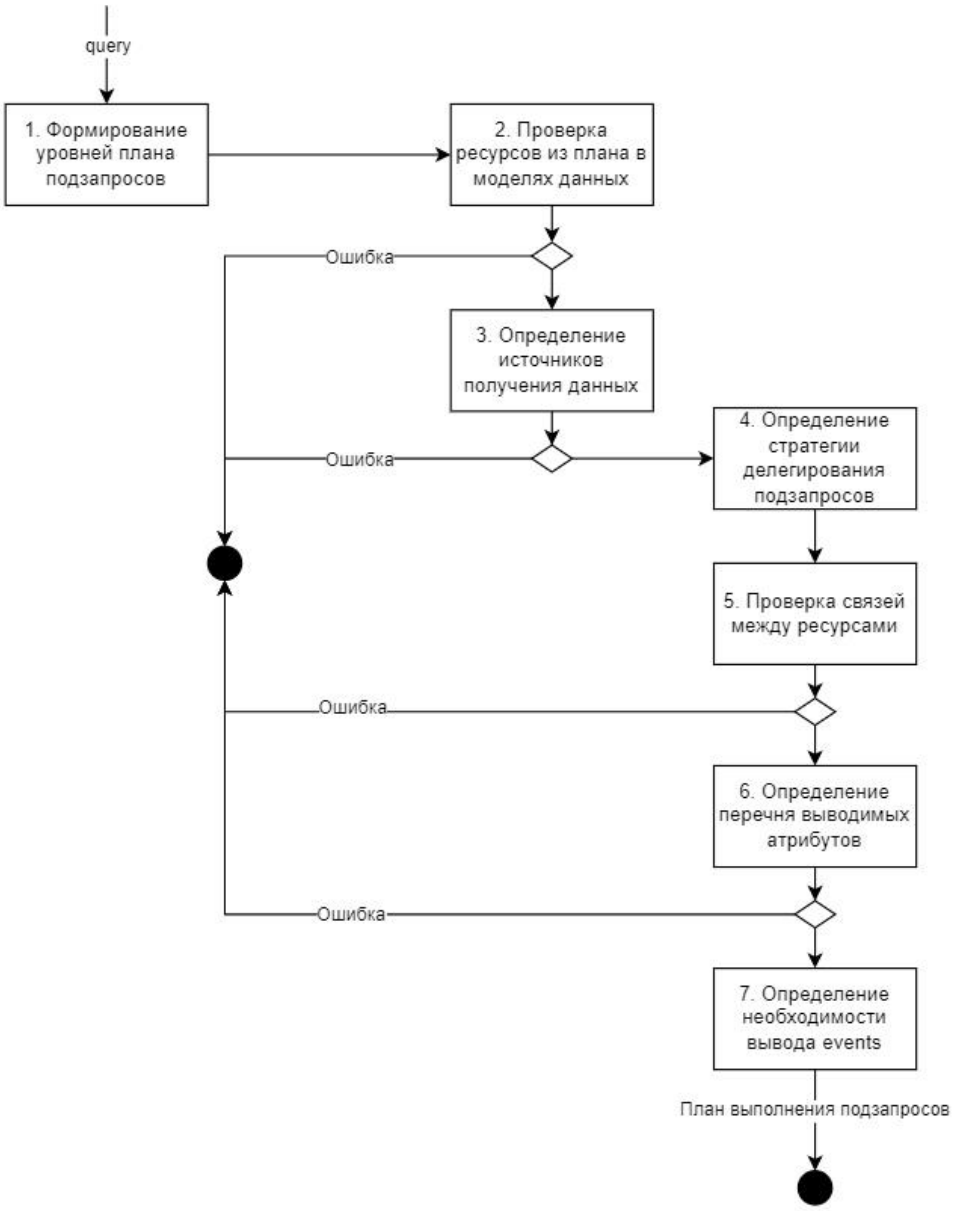



Рисунок - 4.26 Формирование плана исполнения запроса

**Запускающее событие:**

- Запрос получения данных витрины прошел необходимые проверки

**Ход выполнения:**

1. Система на основании вложенности ресурсов, заданной в запросе (query) определяет уровни плана выполнения запроса и sql-выражения. Формирование плана запроса основывается на внешнем объединении данных в сторону основной сущности.

Порядок формирования плана запроса:

- на основе query определяется основная запрашиваемая сущность
- на основе query определяются вспомогательные сущности
- на основе query определяются conditions к основной сущности
- на основе query определяются conditions к вспомогательным сущностям
- на первом уровне плана формируется запрос к основной сущности на основе:

- запрошенных атрибутов
- данных модели
- условий фильтрации

Отмечаются поля, составляющие PK.

Первичные ключи, даже отсутствующие в атрибутах запроса к серверу должны быть в запросе к источнику.

– формируется запрос к вспомогательным сущностям на основе:
  - запрошенных атрибутов
  - данных модели
  - условий фильтрации
  - с добавлением фильтрации по PK основной или предшествующей сущности

Первичные ключи, даже отсутствующие в атрибутах запроса к серверу должны быть в запросе к источнику, за исключением терминальных запросов.

Пример уровней плана на основе запроса:

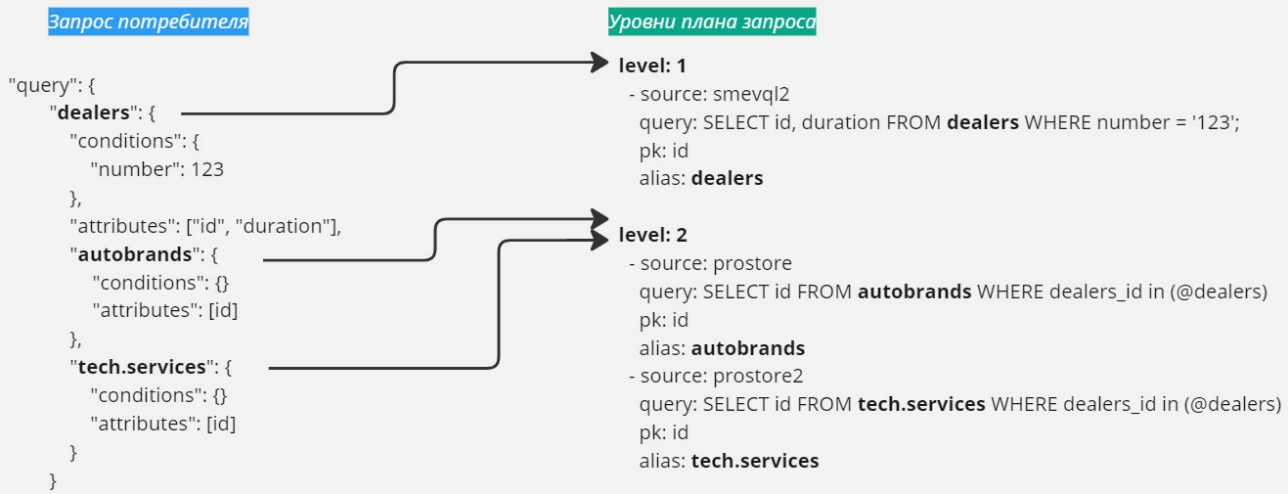



Рисунок - 4.27 Формирование плана исполнения запроса

2. Система проверяет, что все ресурсы из плана запроса определены в модели данных:
   a. если хотя бы один ресурс не описан в модели данных, то Система передаёт соответствующее сообщение об ошибке, процесс завершается
   b. иначе, все ресурсы описаны в модели данных, переход к выполнению следующего шага.
3. Система проверяет, что для каждого ресурса определён источник получения данных. Для этого по каждому ресурсу определяет его источник в модели данных (блок `extract:`), в т.ч. с учетом условий применения источника (`extract->conditions:`) и описан ли источник в модели source:
   a. если хотя бы один источник данных не определен в модели данных или не описан в модели источников, то Система передаёт соответствующее сообщение об ошибке, процесс завершается
   b. иначе, для каждого ресурса определен источник данных, переход к выполнению следующего шага.
4. Система определяет стратегию делегирования подзапросов (подзапросов более низкого уровня). Для этого определяет значение в параметре `request→strategy` конфигурационного файла приложения `application.yaml` (делегирование возможно только источнику smevql):

a. если значение `atomic` - то последовательность выполнения подзапросов соответсвует иерархии уровней, определенной на шаге 1.

b. иначе установлено значение delegate - делегирование подзапроса со всеми дочерними элементами другому СМЭВ QL. В таком случае, все делегированные подзапросы исключаются из плана запроса данного СМЭВ QL.

5. Система проверяет, что в модели данных (блок connections:) существуют и корректно заполнены связи между всему основными и подчиненными ресурсами:
    a. если хотя бы одна связь между основным и подчиненным ресурсом не задана или некорректно описана, то то Система передаёт соответствующее сообщение об ошибке, процесс завершается
    b. иначе, связи между ресурсами заполнены корректно, переход к выполнению следующего шага.
6. Система проверяет, что перечень атрибутов, запрашиваемый Потребителем не запрещен на уровне модели данных. Для этого проверяет заполнение блока `guard:` в модели данных и определяет соблюдены ли условия по передачи данных атрибутов в запросе (блок `conditions`):
    a. если хотя бы один атрибут запрещен для передачи Потребителю (guard заполнен в модели, а требуемый conditions не заполнен в запросе), то Система передаёт соответствующее сообщение об ошибке, процесс завершается
    b. иначе, все требуемые атрибуты разрешены для передачи Потребителю, переход к выполнению следующего шага.
7. Система на основании блока `fetch→show_events` определяет возможность обогащения передаваемых атрибутов данными о возможных событиях (event) машины состояний. Текущий процесс завершается и вызывается исполнение плана запроса.

#### 4.5.3.4 Исполнение плана запроса

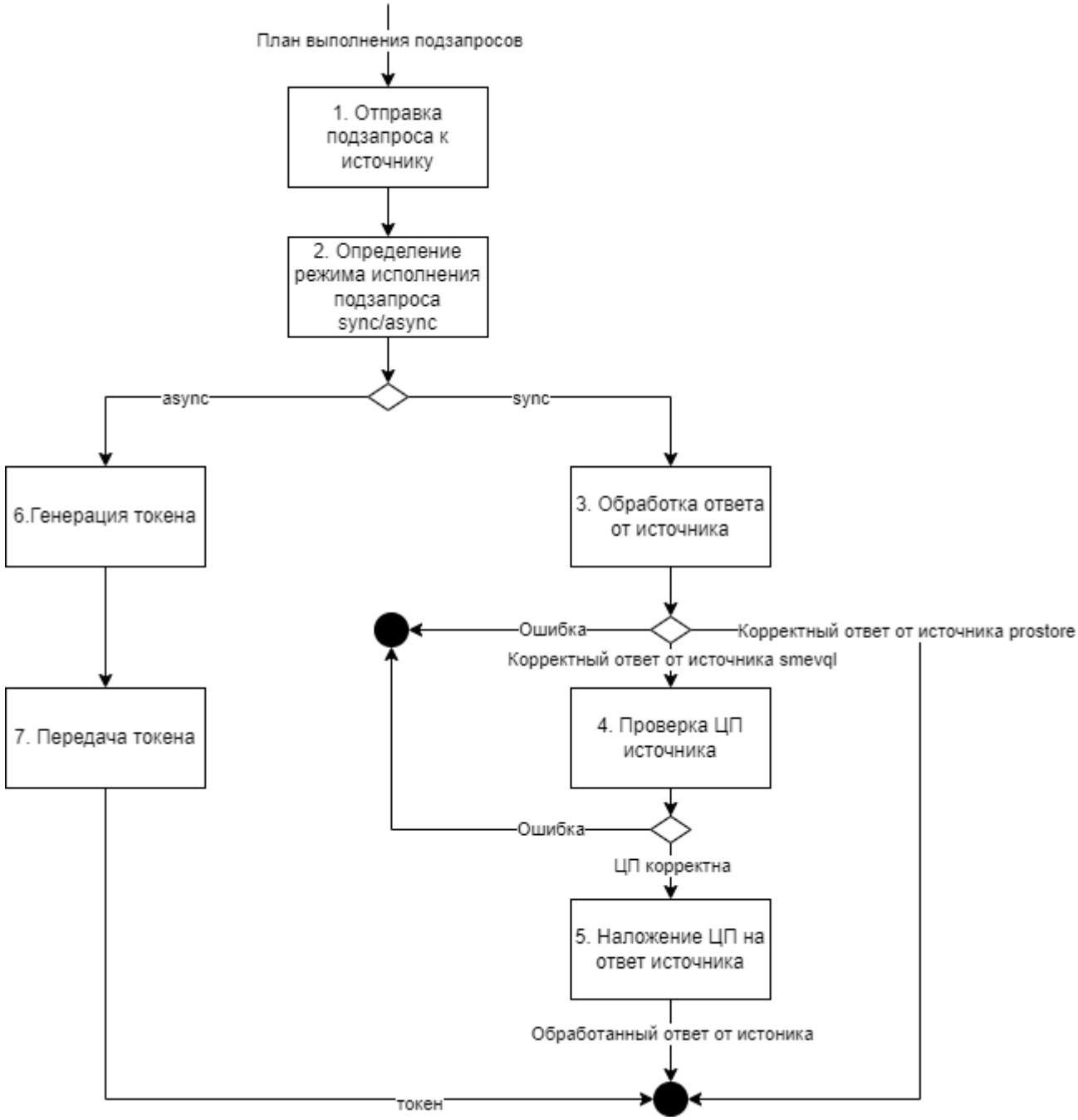



Рисунок - 4.28 Исполнение плана запроса

**Запускающее событие:**

– Подготовлен план выполнения запроса

Ход выполнения:

Указанная последовательность выполняется для каждого подзапроса из плана. При этом сначала параллельно выполняются подзапросы первого уровня, затем подзапросы второго уровня на основе данных полученных на первом уровне и т.д. по всем уровням иерархии плана запроса.

В качестве ответа от Источника могут быть сами запрашиваемые данные (если обращение к источнику шло в синхронном режиме) или токен (если обращение к источнику шло в асинхронном режиме).

1. Система отправляет подзапрос к источнику с указанным в плане sql-выражением. Так же на этом шаге Система обновляет счетчики лимитов в Redis, при этом количество увеличивается на величину подзапросов к источникам данных.
2. Система ожидает ответ и определяет режим обработки подзапроса к источнику ресурса. Для этого определяет значение параметра `fetch->policy` в модели данных ресурса:
    a. если установлено значение `sync` или `async_on_timeout` и ответ пришёл в течение заданного таймаута, то режим обработки подзапроса синхронный, переход к выполнению шага 2
    b. иначе, установлено значение async или async_on_timeout и исчтоник не отвечает в в течение заданного таймаута, то режим обработки подзапроса асинхронный, переход к выполнению шага 6.
3. Система обрабатывает полученный ответ от источника или обрабатывает ситуацию, когда ответ не получен:
    a. если ответ получен и он некорректного формата или ответ не получен в заданный таймаут, то Система передаёт соответствующее сообщение об ошибке, процесс завершается
    b. если, ответ корректен и получен в заданный таймаут, а источником данных является prostore, то данный процесс завершается и вызывается выполнение передачи ответа потребителю (см. Передача ответа потребителю).
    c. иначе, ответ корректен и получен в заданный таймаут, а источником данных является smevql, переход к выполнению следующего шага.
4. Система проверяет, что ответ от источника подписан цифровой подписью (далее ЦП). Для этого извлекает данные из полученного ответа (блок `signature`) и отправляет данные ЦП на проверку в модуль Notarius. После чего дожидается результатов проверки ЦП от Notarius:
    a. если ЦП не прошла успешную проверку в Notarius или блок signature не заполнен источником, то ответ считается невалидным, Система передаёт соответствующее сообщение об ошибке, процесс завершается
    b. иначе, ЦП прошла успешную проверку в Notarius, ответ валиден, переход к выполнению следующего шага.
5. Система накладывает свою ЦП на ответ источника. Для этого отправляет запрос в модуль Notarius и ожидает получение ответа. После чего ответ подписан ЦП (заполнен блок partials в ответе для потребителя). Процесс завершается после получения всех ответов от всех источников и вызывается выполнение передачи ответа потребителю (см. Передача ответа потребителю).
6. Система генерирует токен (по которому в дальнейшем Потребитель сможет отдельно запросить данные) и записывает его в блоке персистентности Redis.
7. Система в качестве временного ответа от источника подставляет токен (получение данных по токену см. Асинхронное получение данных клиентом). Процесс завершается после получения всех ответов от всех источников и вызывается выполнение передачи ответа потребителю.

#### 4.5.3.5 Передача ответа потребителю


Рисунок - 4.29 Передача ответа потребителю

**Запускающее событие:**

– Получены все ответы от источников и/или токены.

**Ход выполнения:**

1. Система формирует комплексный ответ на основе полученных ответов от источников и/или токенов. Формат ответа в таблице ниже:

| **Элемент** | **Описание** | **Пример** |
|---|---|---|
| response | Объект, который содержит данные запрашиваемых ресурсов и/или токенов для асинхронного получения данных ресурсов | Пример блока response |
| <resource_name> | Название ресурса, массив данных которого был получен в источнике.<br>Если запрос содержал подчиненные ресурсы, то данные таких ресурсов будут вставлены в массив объекта основного ресурса.<br>Если данные ресурса должны быть получены отдельно в асинхронном режиме, то ответ будет содержать блок `async`, который имеет следующие атрибуты:<br>– `token` - токен для асинхронного обращения<br>– `source` - источник данных<br>– `reason` - тип асинхронного обращения, `async` - | Пример блока async |

| | | |
|---|---|---|
| | всегда асинхронное или `async_on_timeout` - асинхронное при превышении таймаута<br>– `timeout` - таймаут обращения к источнику<br>– `request_in` - время, по прошествии которого можно отправить повторный запрос получения данных<br>– `retries_left` - оставшееся количество попыток получения данных, если равен 0, то данные запросить невозможно | |
| credentials | Объект, содержащий общие данные запроса и информация о потребителе | Пример блока credentials |
| response | Объект описывает общие параметры ответа. Имеет следующие атрибуты:<br>– `id` - идентификатор ответа<br>– `sub_id` - идентификатор подзапроса<br>– `started_at` - время начала формирования ответа<br>– `finished_at` - время окончания формирования ответа | |
| system | Объект описывает параметры системы-потребителя данных. Имеет следующие атрибуты:<br>– `mnemonic` - мнемоника системы<br>– `instance_id` - идентификатор экземпляра системы<br>– `user_id` - идентификатор пользователя | |
| request | Объект описывает общие параметры запроса. Имеет следующие атрибуты:<br>– `id` - идентификатор запроса<br>– `sub_id` - идентификатор подзапроса<br>– `name` - имя запроса<br>– `purpose_id` - идентификатор события, инициировавшего запрос<br>– `audit` - возможность аудита<br>– `audit_id` - идентификатор аудита. Заполняется только, если audit = true<br>– `audit_token` - токен аудита. Заполняется только, если audit = true | |
| signature | Объект, содержащий данные ЦП по каждому ответу от каждого источника, а так же ЦП комплексного ответа | |
| complex | ЦП комплексного ответа, имеет следующие атрибуты:<br>– `digest` - краткий обзор подписи<br>– `signature` - цифровая подпись | |
| partials | Массив ЦП по каждому полученному ответу от источников данных. Содержит следующие атрибуты:<br>– `digest` - краткий обзор подписи<br>– `signature` - цифровая подпись<br>– `response_id` - идентификатор ответа от | |

| | | |
|---|---|---|
| | источника<br>– `jsonpath` - название ресурса | |

#### 4.5.3.6 Пример блока `response`

```
"response": {
       "ticket": [
           {
               "number": 1,
               "price": 701.0252675821603,
               "trip": [
                   {
                       "id": "f72f4157-e11b-4fe1-86a2-babaeb3ccd21",
                       "duration": "07:31:03"
                   }
               ]
           }
       ]
   }
```

#### 4.5.3.7 Пример блока `async`

```
"async": {
       "token": "11111-22222-333333-4444-55555",
       "source": "region81",
       "reason": "async_on_timeout",
       "timeout": "500ms",
       "request_in": "10s",
       "retries_left": "9"
   }
```

#### 4.5.3.8 Пример блока `credentials`

```
"credentials": {
       "response": {
           "id": "fc7953cb-99ad-4a78-b49d-c3dce2c4bb89",
           "sub_id": "a8744e12-49ce-4524-861d-0e3b9a763d14",
           "started_at": "2023-02-13 07:55:23 +0300",
           "finished_at": "2023-02-13 07:55:23 +0300"
       },
       "system": {
           "mnemonic": "117bed7f-1c07-4079-a446-1161588db4e5",
           "instance_id": "ccb4a88f-f44b-43e7-8a97-3e45c8345e90",
           "user_id": "5ed38461-0907-486a-930a-7b443482932c"
       },
       "request": {
           "id": "df5a0073-c6be-4d8c-8eb2-9b2f4188a429",
           "sub_id": "0cdb59ce-224b-4118-8da1-c5ea08a5d955",
           "name": "driver_data",
           "purpose_id": "ed1170f1-3caa-4985-aa38-c9c5a190b770",
           "audit": "false",
           "audit_id": "fc1048fe-323d-4eeb-92df-5710b3d1d100",
           "audit_token": "39e47aac-45d2-44c1-8c26-2d9b28b1703b"
       }
   },
   "signature": {
       "complex": {
           "digest": "985925e62ce3494a4e73f20676f1506ef64380f0",
           "signature": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9"
       },
       "partials": [
```

```
            {
                "digest": "985925e62ce3494a4e73f20676f1506ef64380f0",
                "signature": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9",
                "response_id": "fc7953cb-99ad-4a78-b49d-c3dce2c4bb89",
                "jsonpath": "$.response.office"
            }
        ]
    }
}
```

### 4.5.4 Асинхронное получение данных клиентом

#### 4.5.4.1 Общее описание

Разные источники с разной скоростью передают данные и по скорости передачи данных их условно можно разделить на «быстрые» и «медленные».

При взаимодействии с «быстрыми» источниками обмен клиента (потребителя) со СМЭВ QL происходит в синхронном режиме в таком случае запрашиваемые данные передаются с разу в момент обращения, а при взаимодействии с «медленными» источниками обмен происходит в асинхронном режиме, в таком случае данные передаются с определенным временным лагом. Указание типа взаимодействия, синхронное или асинхронное, определяется на уровне модели данных в блоке `fetch->policy`.

Для потребителя наличие в ответе блока `async` с токеном вместо данных означает, что данные необходимо получить отдельным запросом, вызвав метод `GET/data/{token}`.

#### 4.5.4.2 Передача асинхронных данных от источника в СМЭВ QL

**Предварительные условия:**

СМЭВ QL отправил источнику запрос получения данных (подробнее см. Обработка запроса получения данных витрины) и сохранил токен.

**Ход выполнения:**

1. Источник подготавливает и передает данные в асинхронном режиме.
2. СМЭВ QL записывает полученные данные в блоке персистентности в спейсе responses с временем жизни `store_timeout`, где в качестве ключа выступает ранее сформированный токен.

Пример записи асинхронных данных в блоке персистентности:

ключ (токен): `11111-22222-333333-4444-55555`

Значение (данные от источника)

```
{"referral": [
  {
    "to_post_name": "врач-терапевт участковый",
    "end_date": null,
    "speciality_id": "119",
    "to_mo_id": "325",
    "mo": [
            {
              "id": "325",
              "name": "ГКБ 222; Кардиология"
            }
          ]
    }
  ]
}
```

#### 4.5.4.3 Запрос асинхронных данных потребителем

**Предварительные условия:**

– Потребитель получил ответ от СМЭВ QL с блоком async

**Ход выполнения:**

1. Потребитель, спустя рекомендованное время (параметр `async->request_in` в ответе), отправляет запрос получения асинхронных данных. Для этого вызывает метод `GET/data/{token}`, где в качестве `{token}` передаёт значение, полученное ранее в ответе (`async→token`).
2. СМЭВ QL сервер проверяет наличие данных по токену:
    a. если данные ещё не поступили от источника, то обновляет счётчик обращений от клиента и формирует ответ вида:

```
{
    "response": {},
    "asynced": {
    "referral": [
        {
        "async":{
        "source": "region12",
        "reason": "receiving_data",
        "timeout": "10s",
        "request_in": "10s",
        "retries_left": "5"
            }
        }
        ]
    }
}
```

b. иначе, данные были получены от источника, Система формирует ответ с данными вида:

```
{
    "response": {
      "referral": [
        {
      "to_post_name": "врач-терапевт участковый",
      "end_date": null,
      "speciality_id": "119",
      "to_mo_id": "325",
      "mo": [
              {
                "id": "325",
                "name": "ГКБ 222; Кардиология"
              }
            ]
        }
    ]
  },
```

3. СМЭВ QL сервер подписывает ответ ЦП с помощью Notarius и передаёт потребителю подписанный ответ.

| Примечание: |
| --- |
| Потребитель продолжает запрашивать данные с рекомендованным интервалом пока не получит данные или пока количество попыток `retries_left` не будет равно 0. |

### 4.5.5 Обработка запроса изменения данных витрины через машину состояний

#### 4.5.5.1 Общее описание

Машина состояний СМЭВ QL сервер имеет определенное количество состояний (`states`), переход между которыми осуществляется посредством вызова событий (`events`).

В зависимости типа вызываемого события выполняется разная логика по обновлению данных витрины.

#### 4.5.5.2 Основной сценарий выполнения

**Запускающее событие:**

Потребитель вызвал событие посредством выполнения запроса POST/states/{model}/{event}

Схема процесса:

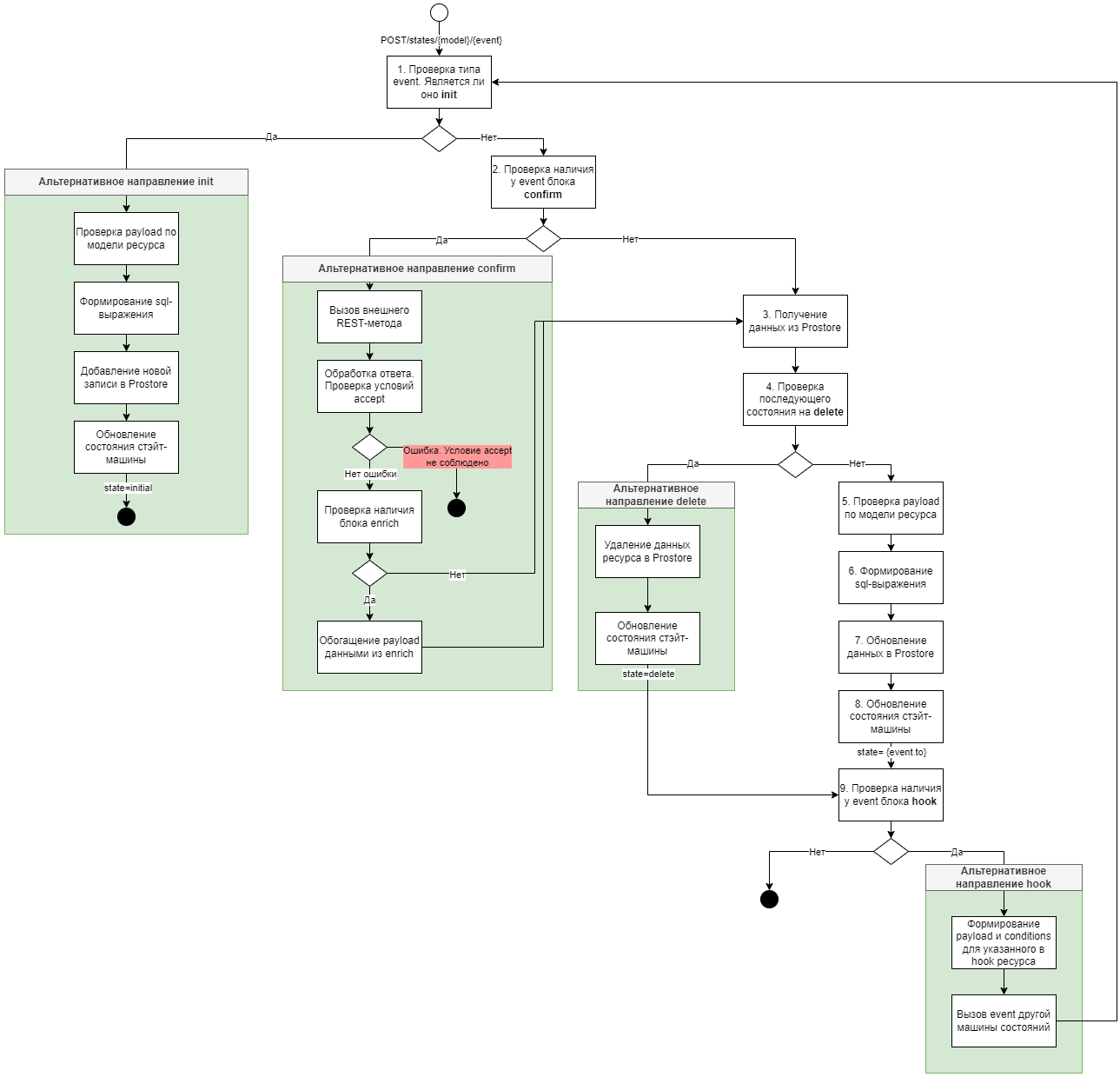



Рисунок - 4.30 Обработка запроса изменения данных витрины через машину состояний

**Ход событий:**

1. Система проверяет тип вызываемого события:
    a. если событие с типом init (`POST/states/{model}/init`), то вызывается альтернативное направление Альтернативное направление для событий с типом init;
    b. иначе, событие не init, переход к выполнению следующего шага.
2. Система проверяет наличие блока `confirm` у вызванного события:
    a. если событие содержит блок `confirm`, то вызывается альтернативное направление Альтернативное направление для событий с блоком confirm;
    b. иначе, событие не содержит блок confirm, переход к выполнению следующего шага.
3. Система получает данные из Prostore для ресурса, вызванной стэйт-машины и сохраняет их в оперативной памяти для последующего обращения. В качестве критерия используются:
    – Условия в блоке `conditions` исходного запроса
    – Значения `attributes:` из текущего состояния стэйт-машины

    Количество записей, получаемых из Prostore ограничено настройкой `max_updated_rows` в конфигурационном файле приложения `application.yaml`. После получения данных, осуществляется переход к выполнению следующего шага.
4. Система проверяет в какое состояние должна перейти стэйт-машина после выполнения вызванного события. Для этого у вызванного события читается параметр `to:`
    a. если стэйт-машина должна перейти в состояние с признаком `delete = true`, то вызывается альтернативное направление Альтернативное направление при переходе в состояние delete;
    b. иначе, стэйт-машина должна перейти в состояние с признаком delete = false (неуказанный признак delete у события приравнивается к значению false), переход к выполнению следующего шага.
5. Система выполняет проверку наличия атрибутов, заданных в блоке `payload` исходного запроса, в модели данных ресурса. Атрибуты отсутствующие в модели данных ресурса игнорируются (не участвуют в последующем обновлении данных витрины). После данной проверки, осуществляется переход к выполнению следующего шага.
6. Система формирует SQL-выражение (определяет какие атрибуты должны быть обновлены) для обновления данных витрины (для выполнения операции upsert в prostore) с учетом следующих правил:
    – Атрибуты из `payload` исходного запроса применяются только, если у вызванного события признак `updatable: true`;
    – Атрибуты из `payload` исходного запроса ограничиваются блоком `updatable_attributes:` (если он указан) вызванного события;
    – Атрибуты, полученные при выполнении `confirm` (см. Альтернативное направление для событий с блоком confirm) из блока `enrich`, применяются всегда;
    – Атрибуты, заданные в блоке `attributes:` текущего состояния стэйт-машины, применяются всегда, за исключением случая, когда у текущего состояния установлен признак `static: true`.

    После формирования SQL-выражения, осуществляется переход к выполнению следующего шага.
7. Система обновляет данные ресурса в Prostore (операция upsert) значениями атрибутов,

определенных на шаге 6 и только для тех записей, которые были получены на шаге 3. После обновления данных осуществляет переход к выполнению следующего шага. Опционально: если в конфигурационном файле `application.yaml`, параметр `state_machine_enabled: true`, то вызывает процесс передачи уведомления об изменении данных витрины.

8. Система переводит стэйт-машину в состояние, указанное в параметре to: вызванного события. После этого осуществляет переход к выполнению следующего шага.
9. Система проверят наличие блока `hooks` у вызванного события:
    a. если событие содержит блок `hooks`, то вызывается альтернативное направление Альтернативное направление для событий с блоком hooks
    b. иначе, обработка события считается выполненной, завершение основного направления.

#### 4.5.5.3 Альтернативное направление для событий с типом init

Данное альтернативное направление запускается из шага 1 основного направления, когда было вызвано событие стэйт-машины с типом `init`. Данный тип события предполагает, что в витрину данных будут добавлены новые записи.

1. Система выполняет проверку наличия атрибутов, заданных в блоке `payload` исходного запроса, в модели данных ресурса. Атрибуты отсутствующие в модели данных ресурса игнорируются (не участвуют в последующем обновлении данных витрины). После данной проверки, осуществляется переход к выполнению следующего шага.
2. Система формирует SQL-выражение для добавления новых записей ресурса с учетом атрибутов, отобранных на шаге 1 данного альтернативного направления. После этого переходит к выполнению следующего шага.
3. Система добавляет новые записи ресурса в Prostore (операция upsert). После этого осуществляет переход к выполнению следующего шага.
4. Система переводит стэйт-машину в состояние, у которого задан признак `initial: true`. После чего завершается выполнение данного альтернативного и основного сценария.

    Опционально: если в конфигурационном файле `application.yaml`, параметр `state_machine_enabled: true`, то вызывает процесс передачи уведомления об изменении данных витрины (Передача уведомления при вызове события машины-состояний).

#### 4.5.5.4 Альтернативное направление для событий с блоком confirm

Данное альтернативное направление запускается из шага 2 основного направления, когда вызванное событие имеет блок `confirm`.

Такое событие предполагает обращение к внешнему источнику с целью передачи ему и получения от него дополнительных данных.

1. Система формирует и передаёт REST-запрос внешнему источнику. Формирование запроса и определение адресата происходит на основании параметров, указанных в блоке `confirm` запускаемого события: `source`, `endpoint`, `method`, `body`. После этого переходит к выполнению следующего шага.
2. Система обрабатывает полученный ответ. Для этого читает условия проверки ответа, заданное в блоке `confirm->accept` вызванного события:
    a. если ответ не соответствует условиям, заданным в accept, то Система передаст соответствующую ошибку Потребителю. Завершение данного альтернативного и основного направлений.
    b. иначе, ответ соответствует условиям, заданным в accept или блок accept не

задан, переход к выполнению следующего шага.

3. Система проверяет наличия блока `enrich` в описании вызванного события:
    a. если блок `enrich` отсутствует, то выполняется переход к шагу 3 основного направления. Данное альтернативное направление завершается.
    b. иначе, блок `enrich` задан, переход к выполнению следующего шага.
4. Система извлекает данные из полученного на шаге 2 ответа и сохраняет их для последующего использования. Далее выполняется переход к шагу 3 основного направления. Данное альтернативное направление завершается.

#### 4.5.5.5 Альтернативное направление при переходе в состояние delete

Данное альтернативное направление запускается из шага 4 основного направления, когда было вызвано событие после выполнения которого стэйт-машина должна перейти в состояние с признаком `delete`. Данный тип события предполагает, что в должны быть удалены данные ресурса из витрины.

1. Система удаляет записи ресурса, полученные на шаге 3 основного направления.
2. Система переводит стэйт-машину в состояние, у которого задан признак `delete: true`. После чего осуществляет переход к выполнению шага 9 основного направления. Данное альтернативное направление завершается.

Опционально: если в конфигурационном файле `application.yaml`, параметр `state_machine_enabled: true`, то вызывает процесс передачи уведомления об изменении данных витрины.

#### 4.5.5.6 Альтернативное направление для событий с блоком hooks

Данное альтернативное направление запускается из шага 9 основного направления, когда вызванное событие имеет блок `hooks`. Такое событие предполагает вызов другого события машины состояния, таким образом формируя вложенные вызовы событий. Количество уровней вложенных вызовов ограничено настройкой `max_nested_event` в конфигурационном файле приложения `application.yaml`. Если в блоке hooks определены несколько переходов (допускается указание массива), то описанные действия выполняются для каждого перехода.

1. Система формирует REST-запрос для вызова другого события стэйт-машины по следующим правилам:
    – URL метода `POST/states/{model}/{event}`: `{model}` - берется из параметра `model` блока `hooks`; `{event}` - берется из параметра `event` блока `hooks`
    – conditions: Заполняется идентификаторами записей нового вызываемого ресурса (название ресурса соответствует названию model). Берутся записи исходного ресурса, полученные на шаге 3 основного направления, далее по связям в модели данных определяются идентификаторы записей для нового ресурса;
    – payload: Строится на основе payload исходного запроса, выбираются атрибуты объекта, название которого совпадает с новым вызываемым ресурсом.
2. Система передаёт REST-запрос для вызова события стэйт-машины. Переход к выполнению шага 1 основного направления, но уже с данными другого события и ресурса. Завершение данного альтернативного направления.

## 4.6 Уведомления при изменении данных витрины (push-сервис)

### 4.6.1 Регистрация получателя уведомлений

#### 4.6.1.1 Общее описание

Данная функция позволяет зарегистрировать в СМЭВ QL сервер нового получателя уведомлений при изменении данных ресурса.

Помимо самого факта уведомления об изменении данных, можно получать определенные атрибуты ресурса, которые будут переданы в рамках нотификации.

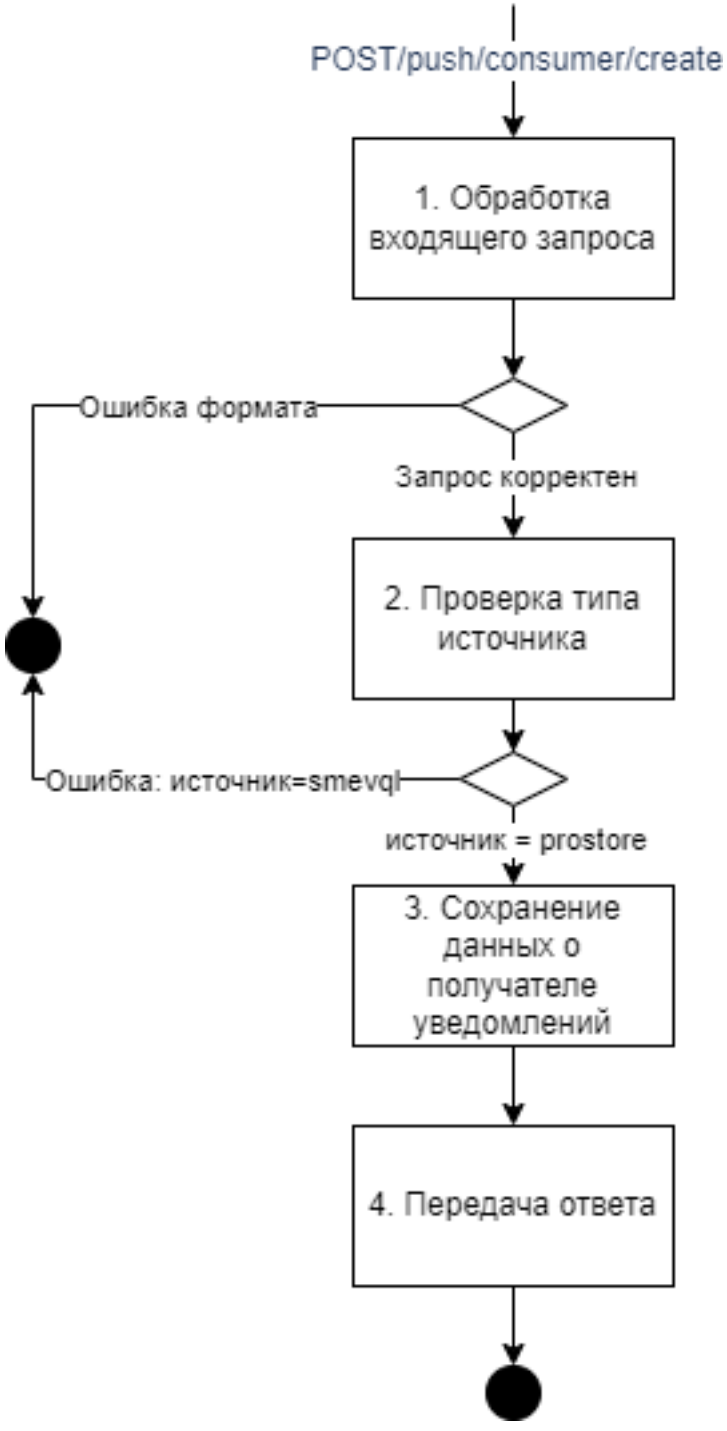



Рисунок - 4.31 Регистрация получателя уведомлений

**Предварительные условия:**

– Потребитель (получатель уведомлений) вызвал запрос на регистрацию получения уведомлений `POST/push/consumer/create`

**Ход выполнения:**

1. СМЭВ QL сервер (далее Система) проверяет входящий запрос на соответствие заданной схеме (Описание запроса POST/push/consumer/create):
    a. если формат запроса некорректен, то Система возвращает соответствующую ошибку и завершает исполнение сценария.
    b. иначе, запрос корректного формата, переход к выполнению следующего шага.
2. Система проверяет к какому типу относится источник данных заданного в запросе ресурса:
    a. если тип источника = smevql, то Система возвращает соответствующую ошибку и завершает исполнение сценария.
    b. иначе, тип источника = prostore, переход к выполнению следующего шага.
3. Система сохраняет данные о получателе уведомлений в `redis hash` с названием `push:consumers:[agent_target]:[resource]`, значение представляется в виде json:

```
{
  "bag": ["sample_field1", "sample_field2"],
  "started_at": "2022-01-01 00:00:00",
  "finished_at": "2022-12-31 23:59:59"
}
```

4. Система передаёт ответ с указанием ключа:

```
{
  "created_consumer": {
  "key": "push:consumers:epgu:books"
  }
}
```

Завершает выполнение сценария.

#### 4.6.1.2 Описание запроса POST/push/consumer/create

Таблица 4.16 Описание запроса `POST/push/consumer/create`

| Элемент | Описание |
|---|---|
| create_consumer | Объект, содержащий информацию о получателе уведомлений и отслеживаемом ресурсе |
| agent | Объект, содержащий информацию об агентах СМЭВ4. Содержит следующие атрибуты:<br>– `source` - мнемоника агента поставщика данных;<br>– `target` - мнемоника агента получателя уведомлений |
| resource | Название ресурса, изменения данных которого необходимо отслеживать |
| bag | Массив атрибутов ресурса, которые необходимо передавать получателю в момент уведомления об изменении данных |
| started_at | Время начала отслеживания изменений |
| finished_at | Время окончания отслеживания изменений |

Пример:

```
{
    "create_consumer": {
        "agent": {
            "source": "zdrav777",
            "target": "epgu"
        },
        "resource": "books",
        "bag": ["sample_field1", "sample_field2"],
        "started_at": "2022-01-01 00:00:00",
        "finished_at": "2022-12-31 23:59:59"
    }
}
```

### 4.6.2 Удаление получателя уведомлений

#### 4.6.2.1 Общее описание

Для того, чтобы прекратить получение уведомлений при изменении ресурса, необходимо вызвать один из следующих методов СМЭВ QL:

– Удаление получателя уведомлений на конкретный ресурс: `POST/push/consumer/delete/resource`

– Удаление получателя уведомлений на все ресурсы: `POST/push/consumer/delete`

#### 4.6.2.2 Сценарий выполнения

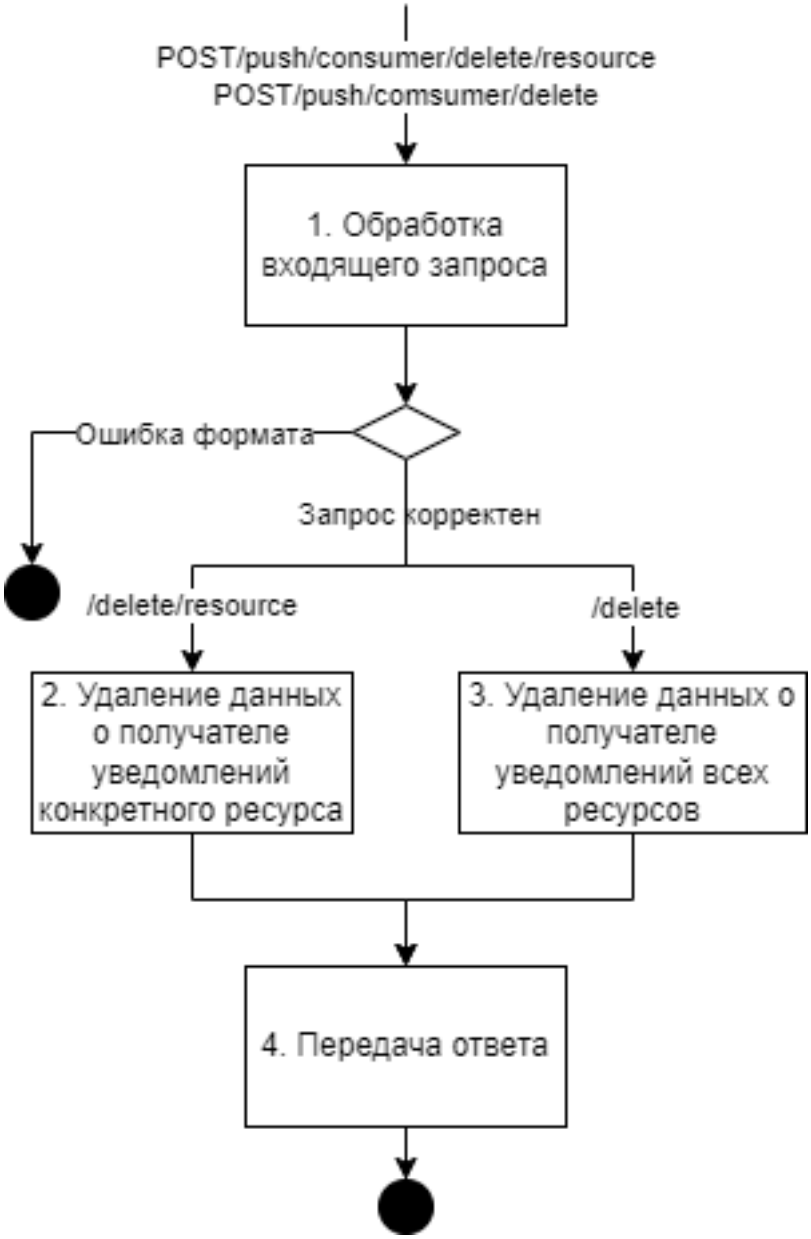



Рисунок - 4.32 Удаление получателя уведомлений

**Предварительные условия:**

- Потребитель (получатель уведомлений) вызвал запрос на удаление получения уведомлений POST/push/consumer/delete или POST/push/consumer/delete/resource

**Ход выполнения:**

1. Система проверяет входящий запрос на соответствие заданной схеме (Описание запроса):
    a. если формат запроса некорректен, то Система возвращает соответствующую ошибку и завершает исполнение сценария.
    b. иначе, запрос `POST/push/consumer/delete/resource` корректного формата, переход к выполнению шага 2 или запрос `POST/push/consumer/delete` корректного формата, переход к выполнению шага 3.
2. Система удаляет данные о получателе уведомлений на конкретный ресурс из redis:
    a. hash с названием `push:consumers:[agent_target]:[resource]`;

b. ссылку на hash из `set` с названием `push:consumers:by:agent_target:[agent_target]`;
c. ссылку на hash из `set` с названием `push:consumers:by:resource:[resource]`;
d. Переходит к выполнению шага 4.

3. Система удаляет данные о получателе уведомлений на все ресурсы из redis:
   a. получает список hash из `set` с названием `push:consumers:by:agent_target:[agent_target]`, далее по всем hash:
      – удаляет из `set push:consumers:by:resource:[resource]` имя `hash`;
      – удаляет `set` с названием `push:consumers:by:agent:target:[agent_target]`;
      – удаляет hash;
   b. Переходит к выполнению шага 4.
4. Система передаёт ответ с указанием удаленных ключей:

```
[
  {
    "deleted_consumer": {
    "key": "push:consumers:epgu:books"
    }
  },
...
]
```

5. Завершает выполнение сценария.

#### 4.6.2.3 Описание запроса

Таблица 4.17 Описание запроса на удаление получателя уведомлений

| Элемент | Описание |
|---|---|
| delete_consumer | Объект, содержащий информацию о получателе уведомлений и отслеживаемом ресурсе |
| agent | Объект, содержащий информацию об агентах СМЭВ4. Содержит следующие атрибуты:<br>– `source` - мнемоника агента поставщика данных;<br>– `target` - мнемоника агента получателя уведомлений |
| resource | Название ресурса, при изменении данных которого должны перестать приходить уведомления для заданного получателя. Заполняется только в запросе `POST/push/consumer/delete/resource` |

Пример:

```
{
    "delete_consumer": {
        "agent": {
            "source": "zdrav777",
            "target": "epgu"
        },
        "resource": "books"
    }
}
```

### 4.6.3 Запрос списка получателей уведомлений

#### 4.6.3.1 Общее описание

Для того, чтобы запросить список получателей уведомлений, необходимо вызвать один из следующих методов СМЭВ QL:

– Получатели уведомлений конкретного ресурса: `GET/push/consumer/agent_targets/{resource}`;
– Все получатели уведомлений данного СМЭВ QL: `GET/push/consumer`.

#### 4.6.3.2 Сценарий выполнения

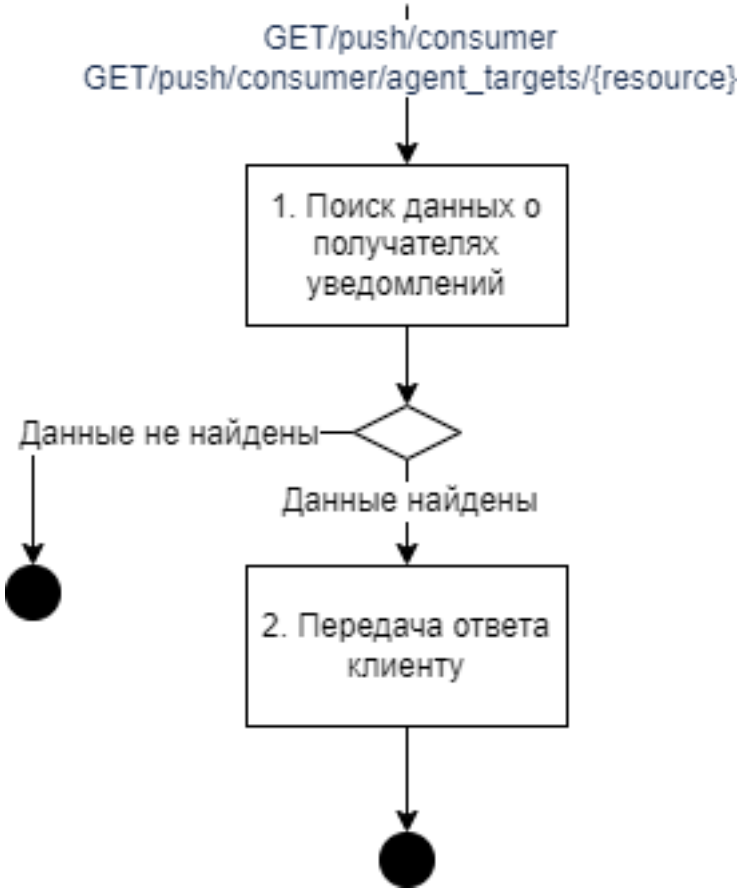



Рисунок - 4.33 Запрос списка получателей уведомлений

**Предварительные условия:**

– Клиент вызвал запрос списка получателей уведомлений `GET/push/consumer` или `GET/push/consumer/agent_targets/{resource}`.

**Ход выполнения:**

1. Система выполняет поиск данных о получателях уведомлений в redis. При этом, если был вызван метод `GET/push/consumer`, то Система получает все имена `hash` из `set push:consumers` и значения всех `hash`, а если был вызван метод `GET/push/consumer/agent_targets/{resource}`, то Система получает имена `hash` из `set push:consumers:by:resource:[resource]` и значения `hash` в рамках указанного ресурса.
    a. если данные не были найдены, то Система возвращает соответствующую ошибку и завершает исполнение сценария.
    b. иначе, данные были найдены, переход к выполнению следующего шага.
2. Система формирует и передаёт ответ клиенту с указанием:
    a. перечня всех получателей уведомлений, если был вызван метод `GET/push/consumer`.
    b. перечня получателей уведомлений при изменении конкретного ресурса, если был вызван метод `GET/push/consumer/agent_targets/{resource}`.

Формат ответа не меняется в зависимости от вызванного метода, отличается только

количество передаваемых данных. Пример ответа:

```
[
  {
    "agent": {
       "target": "epgu"
       },
    "key": "push:consumers:epgu:books"
    "resource": "books",
    "bag": ["sample_field1", "sample_field2"],
    "started_at": "2022-01-01 00:00:00",
    "finished_at": "2022-12-31 23:59:59"
  },
...
]
```

### 4.6.4 Запрос данных об отслеживаемых ресурсах

#### 4.6.4.1 Общее описание

Данная функция позволяет получить перечень отслеживаемых ресурсов заданным потребителем.

Для этого клиенту необходимо вызвать метод `GET/push/consumer/resources/{agent_target}`, где в качестве `{agent_target}` передать значение мнемоники Агента Потребителя.

#### 4.6.4.2 Сценарий выполнения

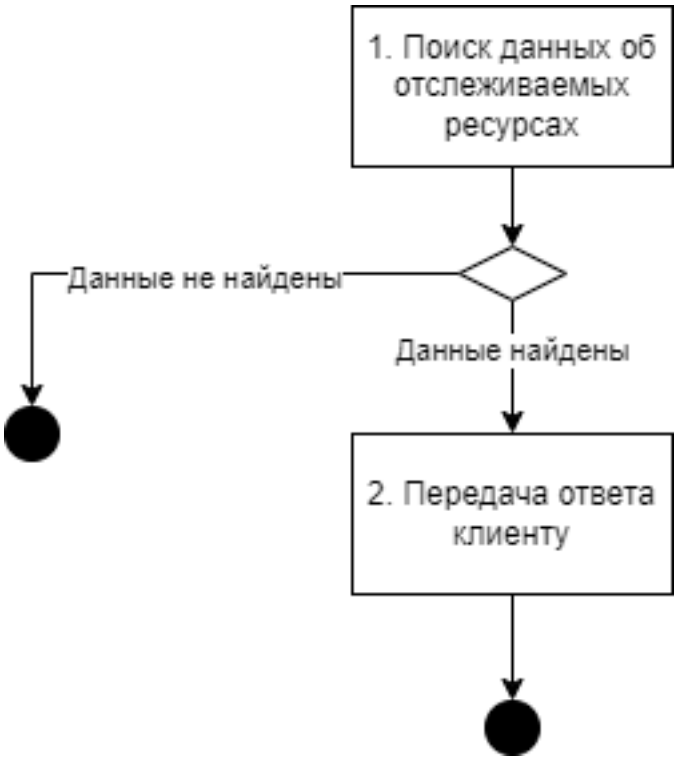



Рисунок - 4.34 Запрос данных об отслеживаемых ресурсах

**Предварительные условия:**

– Клиент вызвал запрос списка отслеживаемых ресурсов `GET/push/consumer/resources/{agent_target}`.

**Ход событий:**

1. Система выполняет поиск данных об отслеживаемых ресурсах в redis: получает имена

`hash` из `set push:consumers:by:agent_target:[agent_target]` и значения `hash`.

a. если данные не были найдены, то Система возвращает соответствующую ошибку и завершает исполнение сценария.
b. иначе, данные были найдены, переход к выполнению следующего шага.

2. Система формирует и передаёт ответ клиенту. Завершает выполнение сценария.

Пример ответа:

```
[
  {
    "agent": {
       "target": "epgu"
       },
    "key": "push:consumers:epgu:books"
    "resource": "books",
    "bag": ["sample_field1", "sample_field2"],
    "started_at": "2022-01-01 00:00:00",
    "finished_at": "2022-12-31 23:59:59"
  },
...
]
```

### 4.6.5 Передача уведомления при вызове события машины-состояний

#### 4.6.5.1 Общее описание

Передача уведомлений об изменении данных витрины средствами СМЭВ QL сервер возможна при выполнении определенного перехода машины состояний (Обработка запроса изменения данных витрины через машину состояний).

Формирование и передача уведомления происходит в следующих случаях:

- Добавление новой записи в Prostore (для событий с типом `init`);
- Удаление данных в Prostore (при переходе в состояние с типом `delete`);
- Обновление данных в Prostore.

#### 4.6.5.2 Сценарий выполнения

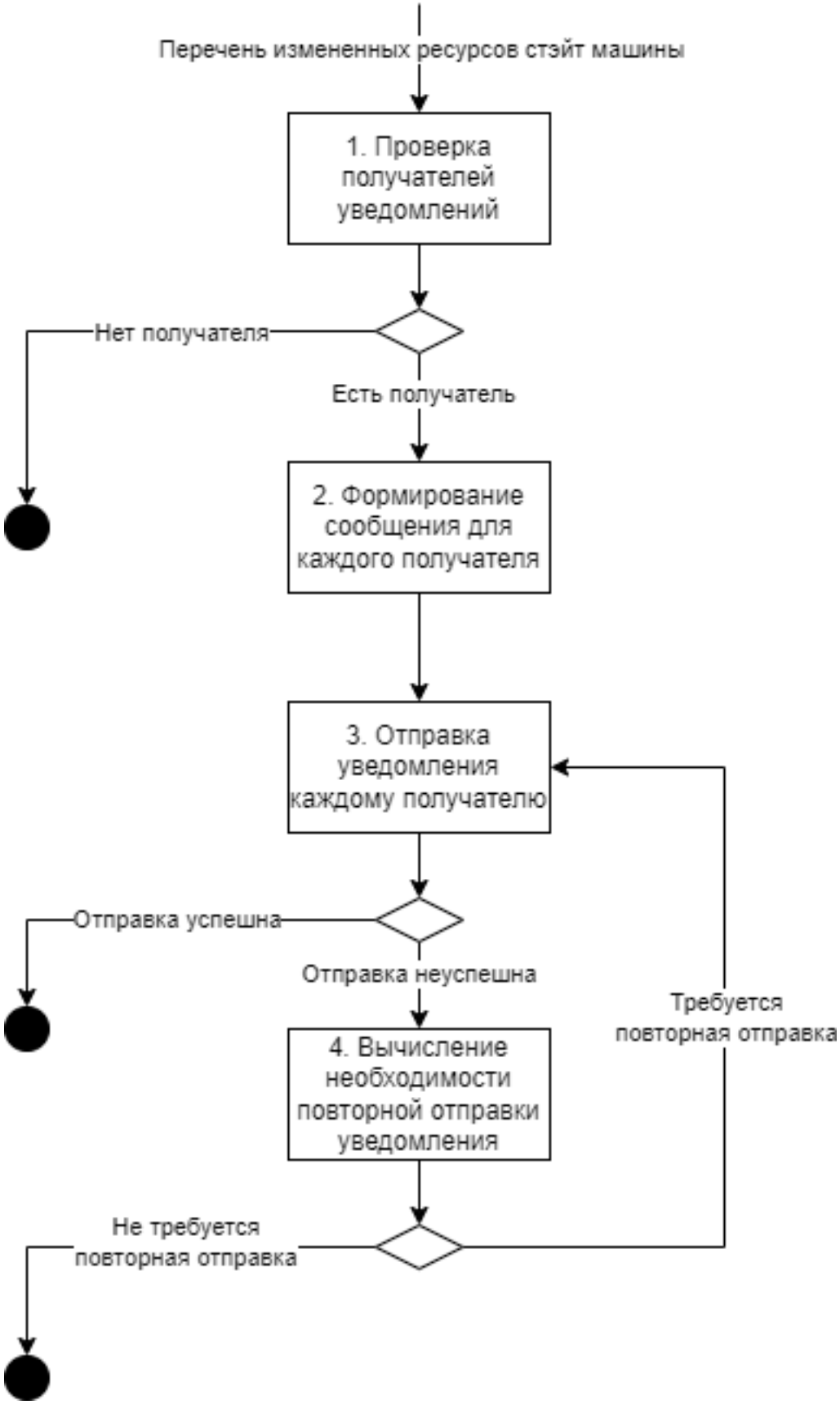



Рисунок - 4.35 Передача уведомления при вызове события машины-состояний

**Предварительные условия:**

- Стэйт машина передала перечень измененных (новые, обновлённые, удалённые) записей ресурсов;
- Получатель зарегистрировал REST-метод (`POST/api/v1/smevql/notify`) для приёма уведомлений.

**Ход событий:**

Данные шаги выполняются в цикле для каждой записи измененного ресурса.

1. Система проверяет есть ли зарегистрированные получатели уведомлений при изменении заданного ресурса. Для этого читает из redis `set` с названием `push:consumers:by:resource` ссылки на зарегистрированных получателей, для каждой ссылки: читает из redis hash данные получателя; проверяет активность получателя на текущий момент.
    a. если не найдено ни одного активного получателя, то передача уведомления не требуется, завершение сценария для данной записи измененного ресурса.
    b. иначе, найден один или более активных получателей, переход к выполнению следующего шага.
2. Система формирует сообщение (содержимое уведомления) для каждого получателя, определенного на шаге 1 (Описание формата уведомления). При необходимости (если при регистрации получения уведомлений (Регистрация получателя уведомлений) был заполнен раздел **bag**) получает дополнительные атрибуты измененного ресурса из витрины. Далее переходит к выполнению следующего шага.
3. Система отправляет уведомления параллельно каждому получателю (отправка уведомления выполняется методом post на адрес агента) и дожидается ответа:
    a. если ответа не поступило в течение заданного таймаута или в ответе получена ошибка (400-500), то Система фиксирует попытку отправки (накопление счётчика) и переходит к выполнению следующего шага.
    b. иначе, уведомление было успешно отправлено получателю, завершение сценария для данной записи измененного ресурса.
4. Система определяет требуется ли выполнить повторную отправку уведомления. Для этого сравнивает текущее значение счётчика попыток отправки уведомления с параметром, заданным в конфигурационном файле `application.yaml push->retry->max_attempts`:
    a. если максимальное количество попыток отправки уведомления превышено, то повторная отправка не требуется, Система сохраняет соответствующее сообщение об ошибке в лог, завершение сценария для данной записи измененного ресурса.
    b. иначе, требуется повторная попытка отправки уведомления, Система возвращается к выполнению шага 3.

#### 4.6.5.3 Описание формата уведомления

Таблица 4.18 Описание формата уведомления

| Элемент | Описание |
|---|---|
| agent | Объект, содержащий информацию об агентах СМЭВ4 |
| source | мнемоника агента поставщика данных |
| target | мнемоника агента получателя уведомлений |
| payload | Сутевые данные уведомления |
| type | Тип уведомления.<br>Заполняется значение `entity`, это означает, что уведомление инициировано событием |

| | |
|---|---|
| | стэйт машины |
| source | Мнемоника источника данных измененного ресурса |
| entity | Данные, полученные из стэйт машины. Содержит следующие атрибуты:<br>– `resource` - название измененного ресурса<br>– `entity_id` - идентификатор записи измененного ресурса<br>– `state` - состояние стэйт машины<br>– `updated_at` - время изменения записи ресурса<br>– `bag` - перечень атрибутов измененного ресурса со значениями |
| push_id | Идентификатор уведомления |

Пример:

```
{
    "notification": {
        "agent": {
            "source": "zdrav777",
            "target": "epgu"
        },
        "payload": {
          "type": entity
          "source": "prostore",
          "entity": {
            "resource": "books",
            "entity_id": 25,
            "state": "received",
            "updated_at": "2023-11-23 12:25:31",
            "bag": {
            "sample_field1": "sample_value"
            }
          }
        },
        "push_id": "UUID"

    }
}
```

### 4.6.6 Регистрация метода для получения уведомлений на стороне клиента

Для того, чтобы потребитель имел возможность получать уведомления об изменении данных витрины, ему необходимо:

1. Развернуть на своей стороне REST-сервис, в котором реализовать метод `POST/api/v1/smevql/notify`;
2. Настроить REST-сервис для взаимодействия с Агентом СМЭВ4 Потребителя;
3. Зарегистрировать REST-сервис в СМЭВ4;
4. Добавить критерии доступа (права поставщику данных) для запросов к развернутому REST-сервису;
5. Выполнить процедуру регистрации получения уведомлений в СМЭВ QL.

Таблица 4.19 Спецификация запроса

| **Элемент** | **Описание** |
|---|---|
| agent | Объект, содержащий информацию об агентах СМЭВ4 |
| source | мнемоника агента поставщика данных |
| target | мнемоника агента получателя уведомлений |

| | |
|---|---|
| payload | Сутевые данные уведомления |
| type | Тип уведомления.<br>На текущий момент всегда заполняется значение `entity`, это означает, что уведомление инициировано событием стэйт машины |
| source | Мнемоника источника данных измененного ресурса |
| entity | Данные, полученные из стэйт машины. Содержит следующие атрибуты:<br>– `resource` - название измененного ресурса<br>– `entity_id` - идентификатор записи измененного ресурса<br>– `state` - состояние стэйт машины<br>– `updated_at` - время изменения записи ресурса<br>– `bag` - перечень атрибутов измененного ресурса со значениями |
| push_id | Идентификатор уведомления |

Приммер:

```
{
    "notification": {
        "agent": {
            "source": "zdrav777",
            "target": "epgu"
        },
        "payload": {
          "type": entity
          "source": "prostore",
          "entity": {
            "resource": "books",
            "entity_id": 25,
            "state": "received",
            "updated_at": "2023-11-23 12:25:31",
            "bag": {
            "sample_field1": "sample_value"
            }
          }
        },
        "push_id": "UUID"

    }
}
```

### 4.6.7 Передача уведомления на основе сообщения топика Prostore

#### 4.6.7.1 Общее описание

При изменении данных витрины, Prostore формирует событие типа WRITE_OK и передаёт его в системный топик `status.event`.

Примеры сообщений системного топика:

```
//пример сообщения для версионированной таблицы
key: {"datamart":"foo","datetime":"2024-02-05
09:52:40.65","event":"WRITE_OK","eventLogId":"f5706095-9dc9-445f-bc93-11683676e3b3"}
value: {"entity":"cat","cn":1,"ts":1707126760643000,"rowsAffected":1}

//пример сообщения для не версионированной таблицы
key: {"datamart":"foo","datetime":"2024-02-05
14:34:23.119","event":"WRITE_OK","eventLogId":"ebb8d74c-d0c2-4f2c-9177-872d1bcb64c6"}
value: {"entity":"catp","cn":null,"ts":null,"rowsAffected":1}
```

Данная функция позволяет передать уведомление об изменении данных витрины на

основе событий системного топика Prostore.

#### 4.6.7.2 Сценарий выполнения

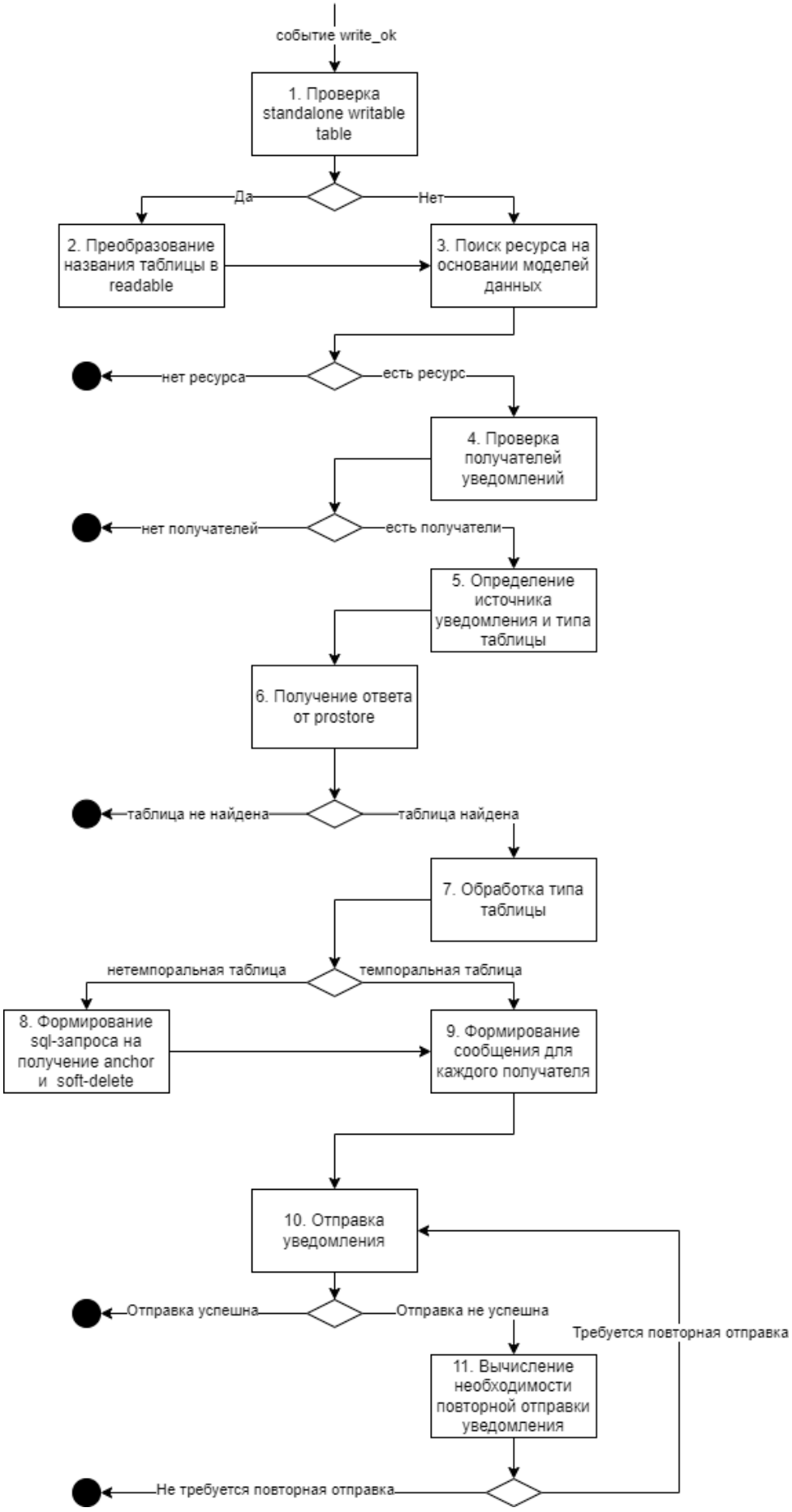


Рисунок - 4.36 Схема передачи уведомления на основе сообщения топика Prostore

**Предварительные условия:**

- Получатель зарегистрировал REST-метод `POST/api/v1/smevql/notify` для приёма уведомлений;
- В топике `status.event` появилось сообщение типа `WRITE_OK`.

**Ход событий:**

1. СМЭВ QL (далее Система) проверяет изменение данных выполнено в таблице вида standalone:
    a. если да, то переходит к выполнению следующего шага
    b. иначе, изменения не в `standalone` таблице, переход к выполнению шага 3.
2. Система определяет `readable` название `standalone-таблицы`. Для этого берет, полученное в сообщении название `writable`, и ищет соответствующее ему readable название в блоке `standalone-tables` конфигурационного файла `application.yaml`. Далее переходит к выполнению следующего шага.
3. Далее Система выполняет поиск ресурса, к которому относится измененная таблица, на основе загруженных моделей данных:
    a. если ресурс не найден, то Система сохраняет соответствующее сообщение в лог и завершает выполнение сценария.
    b. иначе, ресурс определен, переход к выполнению следующего шага.
4. Система проверяет есть ли зарегистрированные получатели уведомлений при изменении заданного ресурса. Для этого читает из redis **set** с названием `push:consumers:by:resource` ссылки на зарегистрированных получателей, для каждой ссылки: читает из redis **hash** данные получателя; проверяет активность получателя на текущий момент.
    a. если не найдено ни одного активного получателя, то передача уведомления не требуется, завершение сценария для данной записи измененного ресурса.
    b. иначе, найден один или более активных получателей, переход к выполнению следующего шага.
5. Затем формирует запрос к Prostore для получения данных о существовании и типе изменённой таблицы. После чего переходит к выполнению следующего шага.
6. Система получает ответ с информацией об измененной таблице:
    a. если таблица не существует в Prostore, то Система сохраняет соответствующее сообщение в лог и завершает выполнение сценария.
    b. иначе, таблица существует в Prostore, переход к выполнению следующего шага.
7. Система обрабатывает значение полученного типа таблицы:
    a. если таблица нетемпоральная (включая proxy-таблицы) , то переход к выполнению следующего шага.
    b. иначе, таблица темпоральная, переход к выполнению шага 9.
8. Система формирует sql-запрос к Prostore для получения даты последнего изменения (**anchor**) и даты удаления (**soft-delete**) записи. После чего переходит к выполнению следующего шага.
9. Система формирует сообщение (содержимое уведомления) для каждого получателя, определенного на шаге 4 (формат уведомления см. Таблица 4.20). При этом если при регистрации получения уведомлений был заполнен раздел **bag**, то он не формируется. Далее переходит к выполнению следующего шага.

10. Система отправляет уведомления параллельно каждому получателю (отправка уведомления выполняется методом `POST/api/v1/smevql/notify` на адрес агента) и дожидается ответа:
    a. если ответа не поступило в течение заданного таймаута или в ответе получена ошибка (400-500), то Система фиксирует попытку отправки (накопление счётчика) и переходит к выполнению следующего шага.
    b. иначе, уведомление было успешно отправлено получателю, завершение сценария для данной записи измененного ресурса.
11. Система определяет требуется ли выполнить повторную отправку уведомления. Для этого сравнивает текущее значение счётчика попыток отправки уведомления с параметром, заданным в конфигурационном файле `application.yaml push->retry->max_attempts`:
    a. если максимальное количество попыток отправки уведомления превышено, то повторная отправка не требуется, Система сохраняет соответствующее сообщение об ошибке в лог, завершение сценария для данной записи измененного ресурса.
    b. иначе, требуется повторная попытка отправки уведомления, Система возвращается к выполнению шага 10.

Таблица 4.20 Описание формата уведомления

| **Элемент** | **Описание** |
|---|---|
| agent | Объект, содержащий информацию об агентах СМЭВ4 |
| source | мнемоника агента поставщика данных |
| target | мнемоника агента получателя уведомлений |
| payload | Сутевые данные уведомления |
| type | Тип уведомления.<br>Заполняется значение **batch**, это означает, что уведомление инициировано событием системного топика Prostore |
| source | немоника источника данных измененного ресурса |
| batch | Данные, полученные из сообщения системного топика Prostore.<br>Содержит следующие атрибуты:<br>– `resource` - название измененного ресурса<br>– `updated_at` - время изменения события<br>– `sys_cn` - номер изменения<br>– `anchor` - время изменения записи в БД. Для темпоральных таблиц всегда `null`<br>– `soft-delete` - время удаления записи в БД. Для темпоральных таблиц всегда `null` |
| push_id | Идентификатор уведомления |

Пример:

```
{
    "notification": {
        "agent": {
            "source": "zdrav777",
            "target": "epgu"
        },
        "payload": {
        "type": batch
        "source": "prostor",
        "batch": {
            "resource": "books",
            "updated_at": "2023-11-23 12:25:31",
            "sys_cn": 134,
            "anchor": null,
            "soft-delete": null
        }
        },
        "push_id": "UUID"
    }
```

# 5 Компонентная модель СМЭВ QL сервера

## 5.1 Общее представление

СМЭВ QL сервер представляет собой бэк-энд сервис, состоящий из определенного набора функциональных модулей, файловой системы (набор папок и yaml-файлов) и персистентного хранилища, реализованного на Reddis. Для обращения к СМЭВ QL сервер используется набор методов OpenAPI.

## 5.2 Схема

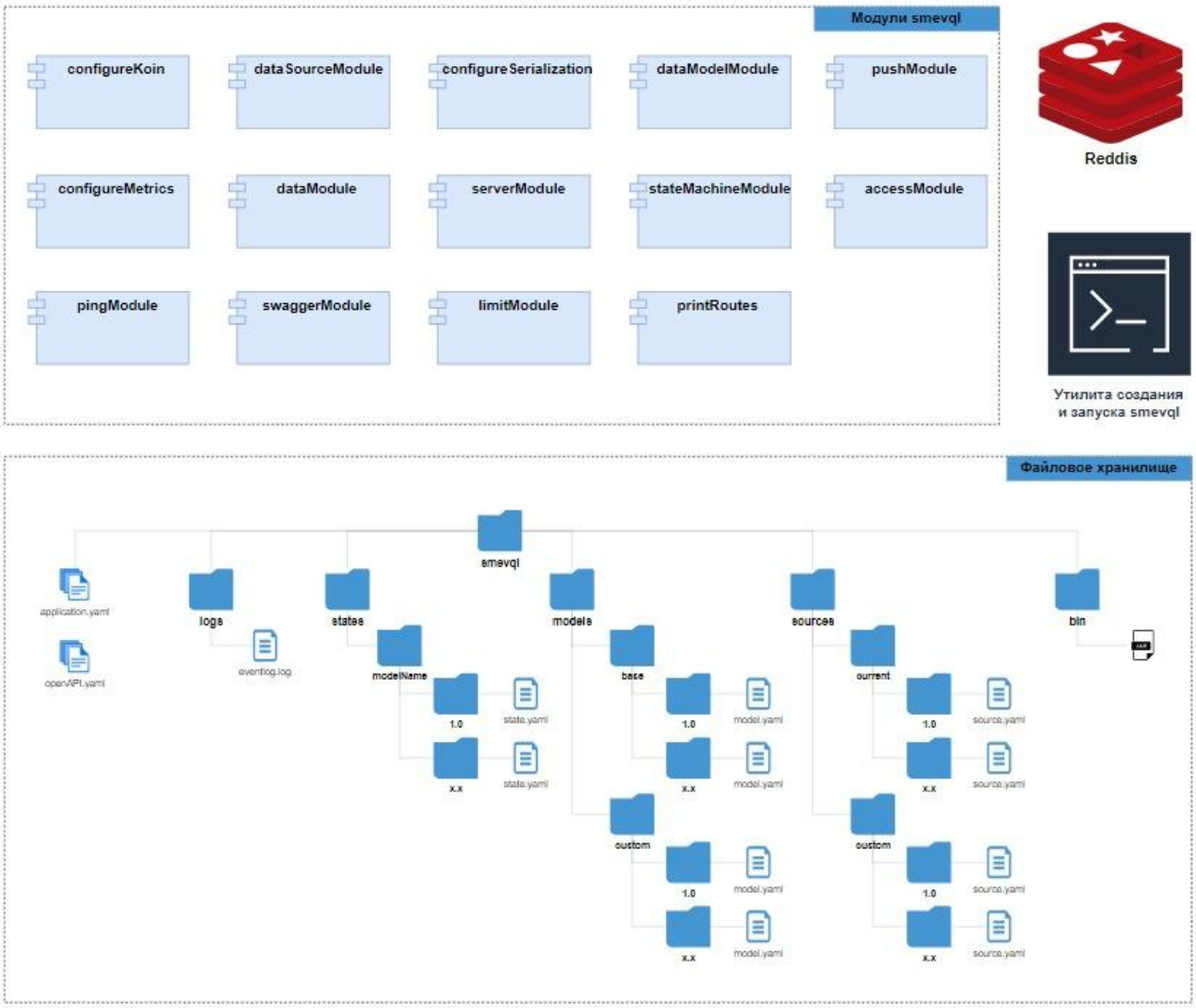



Рисунок - 5.16 Компонентная модель СМЭВ QL сервера

## 5.3 Описание компонентов

Таблица 5.2 Описание компонентов СМЭВ QL сервера

| **Компонент** | **Описание** |
|---|---|
| Утилита создания и запуска smevql | Утилита, позволяющая в терминальном режиме запускать команды для создания и запуска экземпляра СМЭВ QL сервера, а так же выполнять операции по работе с ключевыми сущностями СМЭВ QL (генерация моделей данных, подключение источников данных и т.д.).<br>Перечень доступных команд:<br>1. Создание нового экземпляра СМЭВ-QL сервера: `java -jar smevql-server-all.jar new`<br>2. Добавление источника данных: `./smevql g source`<br>3. Добавление модели данных: `./smevql g model`<br>4. Запуск приложения: `./smevql start -e`<br>5. Генерация openAPI: `yaml g openapi` |
| Reddis | Хранилище персистентных данных СМЭВ QL сервера. |
| Модули smevql | Набор функциональных модулей СМЭВ QL сервера, реализующих его логику работы. |

| | |
|---|---|
| | Перечень и функциональное назначение модулей в Таблица 5.3 |
| Файловое хранилище | Структура папок и файлов СМЭВ QL сервера. Хранит настройки и описание моделей ключевых сущностей: модель данных, описание источников данных, машины состояний и пр. |

Таблица 5.3 Модули СМЭВ QL

| | **Модуль** | **Функциональное назначение** |
|---|---|---|
| 1 | configureKoin | |
| 2 | configureSerialization | |
| 3 | configureMetrics | |
| 4 | accessModule | Управление доступами потребителей данных. Конфигурирование «чёрных» и «белых» списков |
| 5 | serverModule | |
| 6 | pingModule | Проверка доступности СМЭВ QL сервера |
| 7 | limitModule | Управление лимитами входящих запросов от потребителей данных |
| 8 | dataSourceModule | Управление моделями источников данных |
| 9 | dataModelModule | Управление моделями данных |
| 10 | dataModule | Обработка входящих запросов |
| 11 | stateMachineModule | Управление моделями, запуск и исполнение машин-состояний |
| 12 | pushModule | Управление подписчиками на изменение данных витрины и передача уведомлений |
| 13 | swaggerModule | Создание и предоставление внешнему клиенту описания openAPI спецификации |
| 14 | printRoutes | Запись путей вызова обработчиков |

# 6 Конфигурирование сервера

Конфигурирование СМЭВ QL сервера выполняется путем изменения параметров настроек, определенных в файлах `credentials.yaml` и `application.yaml`.

- **application.yaml** - конфигурирует поведение сервера;
- **credentials.yaml** - конфигурирует представление сервера.

## 6.1 Конфигурация файла application.yml

```yaml
ktor:
  deployment:
    port: "$PORT:8080"
  application:
    modules:
      - ru.gov.digital.smevql.ApplicationKt.mainModule

sources:
  directory: "$SOURCES_DIR:sources"
models:
  directory: "$MODELS_DIR:models"
states:
  directory: "$STATES_DIR:states"

swagger:
  file: smevql-openapi.yaml # путь к файлу openapi спецификации
  servers:
    - "http://127.0.0.1:8080/smevql/api/v1"
```

```yaml
storage: # Блок параметров хранения информации
  adapter: redis # На текущий момент поддерживается только redis
  pool: # Настройка подключений к redis
    - host: 127.0.0.1
      port: 6379
  max-pool-size: 20 # Максимальный размер пула соединений
  user: "" # Пользователь для подключения к redis
  password: "" # Пароль

access: # Блок настроек доступа к выполнению операций чтения данных и операций стейтмашины. Допускается задание черного или белого списка
  black-list: [ ] # Указывает список потребителей, для которых доступ запрещен
  white-list: [ ] # Указывает список потребителей, для которых доступ разрешен

request:
  strategy: delegate # Стратегия исполнения запросов delegate|atomic
  timeout: 20s # Таймаут исполнения запросов
  base-path: smevql/api/v1 # Префикс для всех роутов
  max-nested-level: 5 # Предельная вложенность запрашиваемых ресурсов
  pagination:
    default: 100 # Количество элементов на странице по умолчанию
    max: 1000 # Максимальное количество элементов на странице
  logging:
    long:
      duration: 20s # Продолжительность исполнения запросов к источникам, выше которой необходимо производить логирование
      percentage: 100 # Процент логирования длительных запросов к источникам. Допустимо использовать вещественные числа, например, 0.1 - только каждый тысячный долгий запрос
  limits: # Блок параметров конфигурации лимитов
    enabled: true # флаг влюкчения проверок лимитов
    total: # Параметры по всем запросам
      value: 500000 # Общее число запросов в период времени
      period: 1D # Период лимитирования
    mnemonic: # Блок настроек лимитирования по потребителям
      value: 5000 # Число запросов в период времени
      period: 1D # Период лимитирования
    purpose: # Блок настроек лимитирования по целям
      value: 5000 # Число запросов в период времени
      period: 1D # Период лимитирования
    user: # Блок настроек лимитирования по пользователям
      value: 1000 # Число запросов в период времени
      period: 1D # Период лимитирования
    records-ttl:
      day: 1M # Период хранения дневной статистики
      week: 1M # Период хранения статистики за неделю
      month: 1Y # Период хранения статистики за месяц
      year: 2Y # Период хранения статистики за год
  async: # Блок настроек асинхронного выполнения запросов
    request-in: 10s # Значение интервала опроса получения данных асинхронного результата. Выдается в качестве значения атрибута request_in на запрос получения данных
    read-timeout: 5m # Таймаут вычитывания асинхронных данных из источников. Используется для источников с типом smevql, если была возвращена информация по асинхронным результатам

delta:
  commit-interval: 60s # Интервал фиксации дельты источника при изменении данных
  force-commit-interval: 30m # Интервал принудительной фиксации дельт всех источников

state:
  max-nested-event: 5 # Максимально допустимая вложенность связанных переходов стейт
```

```
машины
  max-updated-rows: 1 # Максимальное количество обновляемых строк при событии стейт машины

index-recommendations: # Рекомендации по аналитике
  enabled: true # Флаг включения формирования рекомендаций
  period: 7D  # Период формирования. 7 дней
  concurrency: 10 # Количество параллельных корутин сохранения статистики

# Массив описания standalone таблиц
standalone-tables: [ ]
# Пример описания
# standalone-tables:
#  - readable-table: "misdm05.readable_book"
#    writable-table: "misdm05.writable_book"
#    anchor: "update_at"
#    soft-delete: "delete_at"

# Массив описания proxy таблиц
proxy-tables: [ ]
# Пример описания
# proxy-tables:
#  - table: "misdm05.notebook"
#    anchor: "update_at"
#    soft-delete: "delete_at"

push: # Настройки отправки push уведомлений
  notification-path: "/{target}/push/notify" # Шаблон пути агента, на который необходимо отправлять нотификации
  state-machine-enabled: false # Признак публикации нотификаций на основе событий стейт машины
  status-prostore-enabled: true # Признак публикации нотификаций на основе событий статусов Простора
  prostore:
    status-event-topic:
      topic: "$PS_STATUS_EVENT_TOPIC:status.event"
      property:
        bootstrap.servers: "$PS_KAFKA:localhost:9092"
        group.id: smevql-server-status-event
        auto.offset.reset: earliest
  retry:
    max-attempts: 3 # Количество попыток отправки нотификации
    min-period: 5s # Минимальный период ожидания перед повторной попыткой
    max-period: 10s # Максимальный период ожидания перед повторной попыткой

storage-queue:
  host: "$QUEUE_HOST:localhost"
  port: "$QUEUE_PORT:5432"
  database: "$QUEUE_DATABASE:smevql"
  schema: "$QUEUE_SCHEMA:smevqlqueue"
  user: "$QUEUE_USER:"
  password: "$QUEUE_PASSWORD:"

environment: "$ENVIRONMENT:dev"

agent: # Параметры конфигурирования агента СМЭВ4
  host: 127.0.0.1 # Хост агента
  port: 8171 # Порт приема api-gateway запросов
  mnemonic: "" # Мнемоника агента

signature: # Блок настроек механизма подписания
  enabled: true # Признак включения подписания и проверки подписи
```

```
  validate-enabled: false # Признак предоставления дополнительного URL проверки
подписи
  algorithm: "GOST3410_2012_256" # Алгоритм формирования подписи
  serial-number: "" # Серийный номер ключа подписи
  alias: "" # Алиас ключа. Заполняется либо серийный номер, либо алиас
  notarius: # Блок настроек сервиса Notaris, используемый для подписания
    host: localhost # Хост, на котором доступен Notaris
    port: 8080 # Порт, на котором доступен Notaris
```

Настройка конфигурации **СМЭВ QL сервера** осуществляется путем редактирования параметров настроек в файле `application.yml`, в котором могут быть настроены следующие секции:

- `sources` - определение директории хранения источников;
- `models` - определение директории хранения моделей;
- `states` - определение директории хранения состояний;
- `swagger` - настройка файла openapi спецификации;
- `storage` - управление подключением к внутреннему хранилищу данных СМЭВ-QL;
- `access` - блок настроек доступа к выполнению операций чтения данных и операций стейт-машины;
- `request` - блок настроек исполнения запросов;
- `delta` - управление принудительными коммитами дельт витрин данных;
- `state` - установка ограничений в машинах-состояний;
- `index_recommendations` - рекомендации по аналитике;
- `push` - настройки отправки push уведомлений; - `environment` - определяет значение среды разработки;
- `agent` - параметры конфигурирования Агента СМЭВ4;
- `signature` - блок настроек механизма подписания.

### 6.1.1 Секция storage

В секции `storage` настраиваются параметры хранения информации, например:

```
storage: # Блок параметров хранения информации
  adapter: redis # На текущий момент поддерживается только redis
  pool: # Настройка подключений к redis
    - host: 127.0.0.1
      port: 6379
  max_pool_size: 20 # Максимальный размер пула соединений
  user: "" # Пользователь для подключения к redis
  password: "" # Пароль
```

**Параметры конфигурации**

- `adapter` - система хранения данных, на текущий момент поддерживается только redis;
- `pool` - указание host и port для подключения к хранилищу данных;
- `host` - адрес хоста;
- `port` - порт хоста;
- `max_pool_size` - максимальный размер пула соединений;
- `user` - имя пользователя для авторизации в системе хранения данных;
- `password` - пароль для авторизации в системе хранения данных.

### 6.1.2 Секция access

В секции `access` настраивается доступ к выполнению операций чтения данных и операций стейт-машины.

Допускается задание черного или белого списка.

Например:

```
access: # Блок настроек доступа к выполнению операций чтения данных и операций стейтмашины. Допускается задание черного или белого списка
  black_list: [ ] # Указывает список потребителей, для которых доступ запрещен
  white_list: [ ] # Указывает список потребителей, для которых доступ разрешен
```

**Параметры конфигурации**

- `black_list` - перечень мнемоник ИС Потребителей, которым запрещен доступ к СМЭВ QL. **Не заполняется, если заполнен white_list!**
- `white_list` - перечень мнемоник ИС Потребителей, которым разрешен доступ к СМЭВ QL. **Не заполняется, если заполнен black_list!**

### 6.1.3 Секция request

В секции `request` хранятся настройки исполнения запросов, например:

```
request:
  timeout: 20s # Таймаут исполнения запросов
  base_path: smevql/api/v1 # Префикс для всех роутов
  max_nested_level: 5 # Предельная вложенность запрашиваемых ресурсов
  pagination:
      default: 100 # Количество элементов на странице по умолчанию
      max: 1000 # Максимальное количество элементов на странице
  logging:
      long:
      duration: 20s # Продолжительность исполнения запросов к источникам, выше которой необходимо производить логирование
      percentage: 100 # Процент логирования длительных запросов к источникам. Допустимо использовать вещественные числа, например, 0.1 - только каждый тысячный долгий запрос
  limits: # Блок параметров конфигурации лимитов
      total: # Параметры по всем запросам
      value: 500000 # Общее число запросов в период времени
      period: 1D # Период лимитирования
      mnemonic: # Блок настроек лимитирования по потребителям
      value: 5000 # Число запросов в период времени
      period: 1D # Период лимитирования
      purpose: # Блок настроек лимитирования по целям
      value: 5000 # Число запросов в период времени
      period: 1D # Период лимитирования
      user: # Блок настроек лимитирования по пользователям
      value: 1000 # Число запросов в период времени
      period: 1D # Период лимитирования
      records_ttl:
      day: 1M # Период хранения дневной статистики
      week: 1M # Период хранения статистики за неделю
      month: 1Y # Период хранения статистики за месяц
      year: 2Y # Период хранения статистики за год
  async: # Блок настроек асинхронного выполнения запросов
      request_in: 10s # Значение интервала опроса получения данных асинхронного результата. Выдается в качестве значения атрибута request_in на запрос получения данных
      read_timeout: 5m # Таймаут вычитывания асинхронных данных из источников. Используется для источников с типом smevql, если была возвращена информация по
```

*асинхронным результатам*

**Параметры конфигурации**

- `timeout` - таймаут исполнения запросов;
- `base_path` - префикс для роута запросов. Например, `smevql/api/v1`;
- `max_nested_level` - предельная вложенность запрашиваемых ресурсов;
- `pagination` - управление количеством элементов при ответе. Содержит следующие атрибуты:
  - `default` - количество элементов на странице по умолчанию;
  - `max` - максимальное количество элементов на странице;
- `logging` - управление логированием «долгих» запросов. Содержит следующие атрибуты:
  - `duration` - продолжительность исполнения запросов к источникам, выше которой необходимо производить логирование. Например: `20s`;
  - `percentage` - процент логирования длительных запросов к источникам. Допустимо использовать вещественные числа, например, `0.1` - только каждый тысячный долгий запрос;
- `limits` - установка ограничений на количество запросов. Содержит следующие атрибуты:
  - `total` - общее допустимое количество запросов к серверу:
    - `value` - целочисленное значение количества запросов, например: `1000`;
    - `period` - на какой период устанавливается ограничение, например: 1D (на сутки)
  - `mnemonic` - допустимое для одного потребителя данных количество запросов к серверу:
    - `value` - целочисленное значение количества запросов, например: `100`;
    - `period` - на какой период устанавливается ограничение, например: `1D` (на сутки);
  - `purpose` - допустимое количество запросов к одному ресурсу (таблице, объекту)
    - `value` - целочисленное значение количества запросов, например: `100`;
    - `period` - на какой период устанавливается ограничение, например: `1D` (на сутки);
  - `user` - допустимое количество запросов для пользователя:
    - `value` - целочисленное значение количества запросов, например: `100`;
    - `period` - на какой период устанавливается ограничение, например: `1D` (на сутки);
  - `records_ttl` - настройка хранения статистики по лимитам:
    - `day: 1M` - период хранения дневной статистики;
    - `week: 1M` - период хранения статистики за неделю;
    - `month: 1Y` - период хранения статистики за месяц;
    - `year: 2Y`- период хранения статистики за год;
- `async` - блок настроек асинхронного выполнения запросов. Содержит следующие атрибуты:
  - `request_in` - значение интервала опроса получения данных асинхронного результата. Выдается в качестве значения атрибута request_in на запрос получения данных;

- `read_timeout` - таймаут вычитывания асинхронных данных из источников. Используется для источников с типом smevql, если была возвращена информация по асинхронным результатам

### 6.1.4 Секция delta

В секции `delta` настраивается управление принудительными коммитами дельт витрин данных.

Например:

```
delta:
  commit_interval: 60s # Интервал фиксации дельты источника при изменении данных
  force_commit_interval: 30m # Интервал принудительной фиксации дельт всех источников
```

**Параметры конфигурации**

- `commit_interval` - интервал фиксации дельты источника при изменении данных, например 60s;
- `force_commit_interval` - интервал принудительной фиксации дельт всех источников, например 30s.

Если `commit_interval: 0` и `force_commit_interval: 0`, то для добавления/изменения/удаления данных в Prostore не используется механизм открытия и закрытия дельт, а данные меняются прямым запросом.

При этом возникают следующие ограничения:

- в бекап текущей реализации данные, софрмированные вне дельт, не попадают;
- в результаты запросов к материализованным представлениям данные, софрмированные вне дельт, не попадают;
- в рамках подписок данные, софрмированные вне дельт, не передаются.

### 6.1.5 Секция state

В секции `state` устанавливаются ограничения в стейт машинах.

Например:

```
state:
  max_nested_event: 5 # Максимально допустимая вложенность связанных переходов стейт машины
  max_updated_rows: 1 # Максимальное количество обновляемых строк при событии стейт машины
```

**Параметры конфигурации**

- `max_nested_event` - максимально допустимая вложенность связанных переходов стейт машины, например 5;
- `max_updated_rows` - максимальное количество обновляемых строк при событии стейт машины, например 1.

### 6.1.6 Секция index_recommendations

В секции `index_recommendations` настраиваются рекомендации по аналитике, например:

```
index_recommendations: # Рекомендации по аналитике
  period: 7D  # Период формирования. 7 дней
```

**Параметры конфигурации**

- `period` - устанавливается период формирования аналитики, например 7 дней.

### 6.1.7 Секция standalone-tables

В секции `standalone-tables` настраиваются данные standalone таблиц.

Например:

```
standalone-tables: [ ]
# Пример описания
# standalone-tables:
#  - readable-table: "misdm05.readable_book"
#    writable-table: "misdm05.writable_book"
#    anchor: "update_at"
#    soft-delete: "delete_at"
```

**Параметры конфигурации**

- `readable-table` - название readable таблицы;
- `writable-table` - название writable таблицы;
- `anchor` - название атрибута характеризующего дату и время последнего изменения данных в нетемпоральной таблице;
- `soft-delete` - название атрибута характеризующего дату и время удаления данных в нетемпоральной таблице.

### 6.1.8 Секция proxy-tables

В секции `proxy-tables` настраиваются данные прокси-таблиц.

Например:

```
# Массив описания proxy таблиц
  proxy-tables: [ ]
  # Пример описания
  # proxy-tables:
  #  - table: "misdm05.notebook"
  #    anchor: "update_at"
  #    soft-delete: "delete_at"
```

**Параметры конфигурации**

- `table` - название прокси таблицы;
- `anchor` - название атрибута характеризующего дату и время последнего изменения данных в нетемпоральной прокси таблице;
- `soft-delete` - название атрибута характеризующего дату и время удаления данных в нетемпоральной прокси таблице.

### 6.1.9 Секция push

В секции `push` содержатся настройки сервиса формирования push-уведомлений.

Например:

```yaml
push: # Настройки отправки push уведомлений
  notification-path: "/{target}/push/notify" # Шаблон пути агента, на который необходимо отправлять нотификации
  state-machine-enabled: false # Признак публикации нотификаций на основе событий стейт машины
  status-prostore-enabled: true # Признак публикации нотификаций на основе событий статусов Простора
  prostore:
    status-event-topic:
      topic: "$PS_STATUS_EVENT_TOPIC:status.event"
      property:
        bootstrap.servers: "$PS_KAFKA:localhost:9092"
        group.id: smevql-server-status-event
        auto.offset.reset: earliest
  retry:
    max-attempts: 3 # Количество попыток отправки нотификации
    min-period: 5s # Минимальный период ожидания перед повторной попыткой
    max-period: 10s # Максимальный период ожидания перед повторной попыткой
```

**Параметры конфигурации**

- `notification_path` - шаблон пути агента, на который необходимо отправлять нотификации;
- `state_machine_enabled` - признак публикации нотификаций на основе событий стейт машины;
- `retry` - настройки попыток отправок нотификаций.

### 6.1.10 Секция agent

В секции `agent` указываются параметры подключения к Агенту СМЭВ4 поставщика, например:

```yaml
agent: # Параметры конфигурирования агента СМЭВ4
  host: 127.0.0.1 # Хост агента
  port: 8171 # Порт приема api-gateway запросов
  mnemonic: "" # Мнемоника агента
```

**Параметры конфигурации**

- `host` - хост агента;
- `port` - порт приема api-gateway запросов;
- `mnemonic` - мнемоника агента СМЭВ4.

### 6.1.11 Секция signature

В секции `signature` настраивается управление механизмом цифровой подписи. Например:

```yaml
signature: # Блок настроек механизма подписания
  enabled: true # Признак включения подписания и проверки подписи
  validate_enabled: false # Признак предоставления дополнительного URL проверки подписи
  algorithm: "GOST3410_2012_256" # Алгоритм формирования подписи
  serial_number: "" # Серийный номер ключа подписи
  alias: "" # Алиас ключа. Заполняется либо серийный номер, либо алиас
  notarius: # Блок настроек сервиса Notaris, используемый для подписания
      host: localhost # Хост, на котором доступен Notaris
      port: 8080 # Порт, на котором доступен Notaris
```

**Параметры конфигурации**

- `enable` - включение (true), отключение (false) подписи ответов и проверки

подписей других источников;

- `validate_enabled` - признак предоставления дополнительного URL проверки подписи (доступность вызова REST-метода проверки подписи);
- `algorithm` - алгоритм формирования подписи. На текущий момент доступен только GOST3410_2012_256;
- `serial_number` - серийный номер ключа подписи;
- `alias` - алиас ключа, заполняется либо серийный номер, либо алиас;
- `notarius` - настройки подключения (host и port) к модулю криптографии notarius.

## 6.2 Конфигурация файла credentials.yml

```yaml
version: 1.0.0
system:
    mnemonic: smev_ql_mnemonic
    instance: smev_ql_instance
```

**Параметры конфигурации**

- `version` - номер версии СМЭВ QL;
- `mnemonic` - мнемоника СМЭВ QL, по этому параметру осуществляется идентификация СМЭВ QL сервер во внешних системах и СМЭВ4;
- `instance` - наименование инстанса СМЭВ QL.

## 6.3 Общий сценарий выполнения

1. Администратор системы открывает на редактирование нужный файл (`credentials.yaml` и/или `application.yaml`) настроек СМЭВ QL сервер и меняет требуемые параметры.
2. Перезапускает приложение для применения новых настроек. Для этого открывает консоль утилиты работы со СМЭВ QL и выполняет команду:

```
./smevql restart
```

# 7 Стейт-машина СМЭВ QL

СМЭВ QL содержит встроенную машину состояний для изменения объектов модели внутри витрин данных. Одновременно с этим Стейт машина может, в качестве подтверждения перехода состояния, использовать внешний источник (например ИС Электронной очереди).

## 7.1 Конфигурирование Стейт-машины

Карта состояний и переходов описывается в формате YAML-файла `state.yaml` располагаемого в папке `states/<имя-модели>/<x.x версия модели>` инстанса СМЭВ QL Сервера.

Описание формата и правил карты состояний:

```yaml
model: slot # имя модели
states: # массив состояний объекта
- state: available # название состояния available
    attributes: # массив атрибутов, описывающих состояние
    - name: status # состояние определятся значением атрибута status
        value: AVAILABLE # значение атрибута для описываемого состояния
    initial: true
- state: booked
    attributes:
    - name: status
        value: RECORDED
- state: reserved
    attributes:
```

```
    - name: status
        value: RESERVED
- state: cancelled
    attributes:
    - name: status
        value: CANCELED
- state: blocked
    attributes:
    - name: status
        value: BLOCKED
events: # список событий изменения состояний, из них создаются методы API
- event: book # создает метод POST /states/slot/book
    from: # массив состояний из которых возможен вызов события
    - available
    - reserved
    to: booked # в какое состояние переводится объект после события
    hooks: # массив связанных событий
    - model: book # после перевода надо вызвать событие init для модели book
        event: init
    confirm:
        source: rmis_rest # названия источника
        endpoint: /booking/book
        method: post
        body: payload # что включать в тело запроса
(full|state|conditions|payload|credentials)
        accept: # условие принятия
            jsonpath: $.status.statusCode
            eq: 0 # ожидаем, что statusCode будет равен 0
- event: reserve
    from: available
    to: reserved
- event: block
    from: available
    to: blocked
- event: cancel
    from:
    - available
    - reserved
    - booked
    - blocked
    to: cancelled
    hooks:
    - model: book
        event: cancel
```

### 7.2 Удаление записи через Стейт-машину

Статус удаления отмечается флагом `delete: true`, соответствует удалению записи по conditions. Значения атрибутов для такого статуса не применимы, при наличи атрибутов необходимо выдать предупреждение в лог и проигнорировать их.

Статус с признаком `delete` конечный по смыслу в графе состояний, он не может использоваться в разделе `from` любых событий. Проверок на эту тему не делаем, даже если кто то и пропишет из него переход, то условие никогда не выполнится.

### 7.3 Передача данных без изменения статуса (статичный ивент)

Можно передавать данные к связанному источнику из блока `confirm` с использованием флага статичного запроса при создании стейта: `static: true`.

Созданный с указанием стейта с флагом `static: true` ивент не меняет статус модели, а

только передает указанную в описании ивента часть запроса в блоке confirm к связанному источнику.

Пример state ивента:

```
model: slot # имя модели
states: # массив состояний объекта
  - state: static
      static: true
events: # список событий изменения состояний, из них создаются методы API
  - event: state
    from:
      - available
      - reserved
      - booked
      - blocked
      - cancelled
      - deleted
    to: static # в какое состояние переводится объект после события
    confirm:
        source: rmis_rest # названия источника
        endpoint: /booking/book
        method: post
        body: payload # что включать в тело запроса
(full|state|conditions|payload|credentials)
        accept: # условие принятия
            jsonpath: $.status.statusCode
            eq: 0 # ожидаем, что statusCode будет равен 0
```

### 7.4 Обновление объектов через Стейт-машину

Через Стейт-машину можно обновлять записи в витрине. Для этого при конфигурировании карты состояний необходимо задать значение у event `updatable: true`. При этом если требуется дать возможность обновлять только часть атрибутов, то ограничить этот список можно перечислив атрибуты в массиве ``updatable_attribute``s.

```
- event: reserve
  from: available
  to: reserved
  updatable: true // по умолчанию false для всех, кроме init, возможность изменять
запись при переводе статуса
  updatable_attributes: [] // массив атрибутов, которые можно обновлять, пустой —
можно все
```

Данные для обновления будут браться из блока `payload` запроса на смену состояния. Для события `init` такое конфигурирование не требуется — по умолчанию обновления разрешены для всех атрибутов из `payload`.

### 7.5 Обогащение payload запроса дополнительными атрибутами

Можно обогащать `payload` запроса к Витрине. Для этого необходимо выставить в блоке `enrich` раздела `confim` значение `allow: true`, с указанием `jsonpath` для получения атрибутов для обогащения.

Пример ивента с блоком enrich:

```
events: # список событий изменения состояний, из них создаются методы API
  - event: book # создает метод POST /states/slot/book
    from: # массив состояний из которых возможен вызов события
      - available
      - reserved
```

```
    to: booked # в какое состояние переводится объект после события
    hooks: # массив связанных событий
      - model: book # после перевода надо вызвать событие init для модели book
        event: init
    confirm:
        source: rmis_rest # названия источника
        endpoint: /booking/book
        method: post
        body: payload # что включать в тело запроса
(full|state|conditions|payload|credentials)
        accept: # условие принятия
            jsonpath: $.status.statusCode
            eq: 0 # ожидаем, что statusCode будет равен 0
        enrich:
          allow: true // по умолчанию false
          attributes: [] // пустой массив — все, а если перечислены, то только эти
          jsonpath: $.update.payload // ожидается что по этому адресу объект с атрибутами
и значениями
```

При выставлении атрибута `allow: true` в блоке `enrich`, при выполнении запроса блока `confirm` происходит обращение к объекту, указанному в переменной jsonpath с возвратом атрибутов, перечисленных в массиве `atttributes` (пустой массив означает отсутствие ограничений). Извлеченные атрибуты возвращаются в `payload` ответа `confirm` запроса. Пример возможного `confirm` ответа от ИС приведен ниже:

```
{
  "status": "OK",
  "update": {
    "payload": {
      "position": "Тест1",
      "rvr_cv_id": "8888888-AEA2-4EF7-95C4-B29A4A5E990",
      "rvr_candidate_id": "99999999-AEA2-4EF7-95C4-B29A4A5E990"
    }
  }
}
```

Атрибуты `payload` ответа `confirm` запроса обогащают `payload` основного запроса. Обновление данных в витрине в случае обогащения payload дополнительными атрибутами происходит независимо от значения `updatable`.

При отсутствии объекта по пути jsonpath в confirm запросе возвращается ошибка `501 "Отсутствует объект при обращении по jsonpath"`, выполнение запроса прерывается.

При несоответствии атрибутов в блоке `attributes` и в `jsonpath` - в `payload` вернутся только те атрибуты, по которым есть соответствия с фиксацией в логах событий уровня WARN о несоответствующих атрибутах: `"Отсутствует атрибут *attribute_name" в $.update.payload"`.

В случае, если в `payload` присутствуют атрибуты, аналогичные тем, что возвращаются при выполнении блока `enrich`, обновляются данные в соответствии со значениями вернувшимися при выполнении блока enrich. В этом случае в логах фиксируется запись `"Изменение значений атрибутов: *attribute_name" payload в соответствии со значениями блока enrich"`.

### 7.6 Методы API Стейт-машины

При наличии заполненных состояний машины СМЭВ QL Сервер генерирует API с набором HTTP-методов, отвечающих за изменение и просмотр состояний объектов:

1. `GET /states` — получить карту переходов;
2. `GET /states/<model-name>` — получить карту переходов конкретной модели;

3. `POST /states/<model-name>/<event-name>` — выполнить переход состояний для модели.

Запрос выполнения перехода:

```
POST /states/slot/book
{
    "state": {
        "conditions": {
            "id": "d9e70331-b4c0-4e96-96b6-322ac75e5188" # slot_id
        },
        "payload": {
            "bookId":"82dcac12-0a29-4fff-b9a7-8dfc84f7853d",
            "patient_Id":"23453456",
            "booking_type":"APPOINTMENT",
            "caseNumber":"73367196",
            "preliminaryReservation": false,
            "email":"email@gmail.com",
            "mobilePhone":"89150000102",
            "referral_id":"102111",
            "cards_id":"102"
        }
    },
    "credentials": {
        "system": {
            "mnemonic": "117bed7f-1c07-4079-a446-1161588db4e5",
            "instance_id": "ccb4a88f-f44b-43e7-8a97-3e45c8345e90",
            "user_id": "5ed38461-0907-486a-930a-7b443482932c"
        },
        "request": {
            "id": "df5a0073-c6be-4d8c-8eb2-9b2f4188a429",
            "sub_id": "0cdb59ce-224b-4118-8da1-c5ea08a5d955",
            "name": "request_name",
            "purpose_id": "ed1170f1-3caa-4985-aa38-c9c5a190b770",
            "audit": false,
            "audit_id": "fc1048fe-323d-4eeb-92df-5710b3d1d100",
            "audit_token": false
        }
    }
}
```

## 7.7 Спецификация интерфейса Стейт-машины

```
openapi: 3.0.0
x-stoplight:
id: 5i2oag6m5eq3v
info:
title: SmevQLStateMachine
version: '1.0'
description: ''
servers:
- url: 'http://localhost:3000'
paths:
/states:
    parameters: []
    get:
    summary: Get models
    tags: []
    responses:
        '200':
        description: ''
        content:
            application/x-yaml:
```

```
            schema:
                $ref: '#/components/schemas/Models'
            examples: {}
            application/xml:
            schema:
                type: object
                properties: {}
            multipart/form-data:
            schema:
                type: object
                properties: {}
            text/html:
            schema:
                type: object
                properties: {}
    operationId: get-models
    description: Retrieve the information of models
'/states/{model}':
    parameters:
    - schema:
        type: string
        name: model
        in: path
        required: true
    get:
    summary: Get model
    tags: []
    responses:
        '200':
        description: Model Found
        content:
            application/x-yaml:
            schema:
                $ref: '#/components/schemas/Model'
            examples: {}
        '400':
        description: Bad Request
        '404':
        description: Model Not Found
    operationId: get-model
    description: Retrieve the information of model
    parameters: []
'/states/{model}/{event}':
    post:
    summary: State change
    operationId: post-state
    responses:
        '200':
        description: State Updated
        content:
            plain/text:
            schema:
                type: string
            examples: {}
        '400':
        description: Bad request
        '404':
        description: Not Found
    requestBody:
        content:
        application/json:
            schema:
```

```
            $ref: '#/components/schemas/StateUpdate'
            examples: {}
        description: Post the necessary fields for the API to create a new user.
    description: Update state
    parameters: []
    parameters:
    - schema:
        type: string
        name: model
        in: path
        required: true
    - schema:
        type: string
        name: event
        in: path
        required: true
components:
schemas:
    State:
    type: object
    x-stoplight:
        id: 89f55561cae04
    properties:
        state:
        type: string
        attributes:
        type: array
        minItems: 1
        items:
            $ref: '#/components/schemas/Attribute'
        initial:
        type: boolean
    Model:
    type: object
    x-stoplight:
        id: b96a73db1e1b2
    properties:
        model:
        type: string
        states:
        type: array
        minItems: 1
        items:
            $ref: '#/components/schemas/State'
        events:
        type: array
        items:
            $ref: '#/components/schemas/Event'
    Events:
    title: Events
    x-stoplight:
        id: qdvbkmgs9xkli
    type: array
    items:
        $ref: '#/components/schemas/Event'
    States:
    $ref: '#/components/schemas/State'
    x-stoplight:
        id: wab1w6ro30nrl
    Event:
    title: Event
    x-stoplight:
```

```
    id: sr024y2v7khum
type: object
properties:
    event:
    type: string
    from:
    type: string
    to:
    type: string
Models:
title: ModelSet
x-stoplight:
    id: e7de590d788a7
type: array
items:
    $ref: '#/components/schemas/ModelInstance'
ModelInstance:
type: object
properties:
    model:
    type: string
    states:
    $ref: '#/components/schemas/States'
    events:
    $ref: '#/components/schemas/Events'
ModelSet:
type: array
items:
    $ref: '#/components/schemas/Model'
Attribute:
title: Attribute
x-stoplight:
    id: 3kiqu047734tc
type: object
properties:
    name:
    type: string
    value:
    type: string
description: ''
StateUpdate:
title: StateUpdate
x-stoplight:
    id: 2iyfifo0q1dt2
type: object
properties:
    state:
    type: object
    properties:
        conditions:
        type: object
        payload:
        type: object
    credentials:
    type: object
    properties:
        system:
        type: object
        properties:
            mnemonic:
            type: string
            instance_id:
```

```
            type: string
            user_id:
            type: string
        request:
        type: object
        properties:
            id:
            type: string
            format: uuid
            sub_id:
            type: string
            format: uuid
            name:
            type: string
            purpose_id:
            type: string
            format: uuid
            audit:
            type: boolean
            audit_id:
            type: string
            format: uuid
            audit_token:
            type: string
```

**Пример реализации: Спецификация интерфейса Стейт-машины РМИС (OpenAPI)**

```
openapi: 3.0.0
x-stoplight:
id: 5i2oag6m5eq3v
info:
title: SmevQLStateMachine
version: '1.0'
description: ''
servers:
- url: 'http://localhost:3000'
paths:
/states:
    parameters: []
    get:
    summary: Get models
    tags: []
    responses:
        '200':
        description: ''
        content:
            application/x-yaml:
            schema:
                $ref: '#/components/schemas/Models'
            examples: {}
            application/xml:
            schema:
                type: object
                properties: {}
            multipart/form-data:
            schema:
                type: object
                properties: {}
            text/html:
            schema:
                type: object
```

```
            properties: {}
    operationId: get-models
    description: Retrieve the information of models
'/states/{model}':
    parameters:
    - schema:
        type: string
        name: model
        in: path
        required: true
    get:
    summary: Get model
    tags: []
    responses:
        '200':
        description: Model Found
        content:
            application/x-yaml:
            schema:
                $ref: '#/components/schemas/Model'
            examples: {}
        '400':
        description: Bad Request
        '404':
        description: Model Not Found
    operationId: get-model
    description: Retrieve the information of model
    parameters: []
'/states/{model}/{event}':
    post:
    summary: State change
    operationId: post-state
    responses:
        '200':
        description: State Updated
        content:
            plain/text:
            schema:
                type: string
            examples: {}
        '400':
        description: Bad request
        '404':
        description: Not Found
    requestBody:
        content:
        application/json:
            schema:
            $ref: '#/components/schemas/StateUpdate'
            examples: {}
        description: Post the necessary fields for the API to create a new user.
    description: Update state
    parameters: []
    parameters:
    - schema:
        type: string
        name: model
        in: path
        required: true
    - schema:
        type: string
        name: event
```

```
        in: path
        required: true
components:
schemas:
    State:
    type: object
    x-stoplight:
        id: 89f55561cae04
    properties:
        state:
        type: string
        attributes:
        type: array
        minItems: 1
        items:
            $ref: '#/components/schemas/Attribute'
        initial:
        type: boolean
    Model:
    type: object
    x-stoplight:
        id: b96a73db1e1b2
    properties:
        model:
        type: string
        states:
        type: array
        minItems: 1
        items:
            $ref: '#/components/schemas/State'
        events:
        type: array
        items:
            $ref: '#/components/schemas/Event'
    Events:
    title: Events
    x-stoplight:
        id: qdvbkmgs9xkli
    type: array
    items:
        $ref: '#/components/schemas/Event'
    States:
    $ref: '#/components/schemas/State'
    x-stoplight:
        id: wab1w6ro30nrl
    Event:
    title: Event
    x-stoplight:
        id: sr024y2v7khum
    type: object
    properties:
        event:
        type: string
        from:
        type: string
        to:
        type: string
    Models:
    title: ModelSet
    x-stoplight:
        id: e7de590d788a7
    type: array
```

```
    items:
        $ref: '#/components/schemas/ModelInstance'
    ModelInstance:
    type: object
    properties:
        model:
        type: string
        states:
        $ref: '#/components/schemas/States'
        events:
        $ref: '#/components/schemas/Events'
    ModelSet:
    type: array
    items:
        $ref: '#/components/schemas/Model'
    Attribute:
    title: Attribute
    x-stoplight:
        id: 3kiqu047734tc
    type: object
    properties:
        name:
        type: string
        value:
        type: string
    description: ''
    StateUpdate:
    title: StateUpdate
    x-stoplight:
        id: 2iyfifo0q1dt2
    type: object
    properties:
        state:
        type: object
        properties:
            conditions:
            type: object
            properties:
                id:
                type: string
                format: uuid
            payload:
            type: object
            properties:
                book_id:
                type: string
                format: uuid
                patient_id:
                type: string
                booking_type:
                type: string
                case_number:
                type: string
                preliminary_reservation:
                type: boolean
                email:
                type: string
                mobile_phone:
                type: string
                referral_id:
                type: string
                cards_id:
```

```
            type: string
        credentials:
        type: object
        properties:
            system:
            type: object
            properties:
                mnemonic:
                type: string
                instance_id:
                type: string
                user_id:
                type: string
            request:
            type: object
            properties:
                id:
                type: string
                format: uuid
                sub_id:
                type: string
                format: uuid
                name:
                type: string
                purpose_id:
                type: string
                format: uuid
                audit:
                type: boolean
                audit_id:
                type: string
                format: uuid
                audit_token:
                type: string
```

### 7.7.1 Выполнение операций обновления данных в витрине

Инсерты в витрину выполняются в порядке поступления запросов.

Каждый экземпляр СМЭВ QL сервера ведет нарастающий счетчик инсертов.

Каждый экземпляр СМЭВ QL сервера создает один поток управления дельтами, который выполняет:

1. Периодически (период конфигурируемая величина, по умолчанию 60 сек) проверяет значение счетчика числа инсертов, если значение счетчика более 0
   - Выполняет открытие и закрытие дельты с флагом immediate, ошибки открытия и закрытия дельты игнорируются. Попытка закрытия дельты выполняется независимо от успешности открытия дельты.
   - Обнуляет значение счетчика.
2. Периодически (период конфигурируемая величина, по умолчанию 30 мин)
   - Выполняет открытие и закрытие дельты с флагом immediate, ошибки открытия и закрытия дельты игнорируются. Попытка закрытия дельты выполняется независимо от успешности открытия дельты.

Чтение данных сервером СМЭВQL выполняется с применением AS OF <maxLong>.

### 7.7.2 Обновление объектов через Стейт-машину

Через Стейт-машину можно обновлять записи в витрине. Для этого при конфигурировании карты состояний необходимо задать значение у **event** `updatable: true`.

При этом если требуется дать возможность обновлять только часть атрибутов, то ограничить этот список можно перечислив атрибуты в массиве `updatable_attributes`.

```
- event: reserve
  from: available
  to: reserved
  updatable: true // по умолчанию false для всех, кроме init, возможность изменять запись при переводе статуса
  updatable_attributes: [] // массив атрибутов, которые можно обновлять, пустой — можно все
```

Данные для обновления будут браться из блока **payload** запроса на смену состояния.

Для события init такое конфигурирование не требуется - по умолчанию обновления разрешены для всех атрибутов из **payload**.

## 8 Запросы

Запросы к серверу выполняют методом POST и содержат в теле JSON-объект, состоящий из обязательных блоков:

- Блок Credentials;
- Блок Query;

Также пространстве методов server находятся методы, помогающие эксплуатации корректно конфигурировать хранилища данных относительно модели СМЭВ QL.

### 8.1 Блок Credentials

```
"credentials":{
    "system":{
        "mnemonic":"117bed7f-1c07-4079-a446-1161588db4e5",
        "instance_id":"ccb4a88f-f44b-43e7-8a97-3e45c8345e90",
        "user_id":"5ed38461-0907-486a-930a-7b443482932c"
    },
    "request":{
        "id":"df5a0073-c6be-4d8c-8eb2-9b2f4188a429",
        "sub_id":"0cdb59ce-224b-4118-8da1-c5ea08a5d955",
        "name":"driver_data",
        "purpose_id":"ed1170f1-3caa-4985-aa38-c9c5a190b770",
        "audit":"false",
        "audit_id":"fc1048fe-323d-4eeb-92df-5710b3d1d100",
        "audit_token":"39e47aac-45d2-44c1-8c26-2d9b28b1703b"
    },
    "signature":{
        "digest": null,
        "signature": null
    }
}
```

### 8.2 Блок Query

```
{
    "query": {
        "office": {
            "conditions": {
                "phone": "(347) 246-53-00"
            },
            "attributes": [
                "id",
                "phone",
                "name"
```

```
            ],
            "cabinet": {
                "conditions": {
                    "available": true
                },
                "attributes": [
                    "number",
                    "name",
                    "seats"
                ],
                "online_room": {
                    "conditions": {
                        "public": true,
                        "software": "zoom"
                    },
                    "attributes": [
                        "url"
                    ]
                },
                "parking": {
                    "conditions": {
                        "free": true,
                        "available": true
                    },
                    "attributes": [
                        "number",
                        "floor"
                    ]
                }
            }
        }
    }
}
```

### 8.2.1 Условия фильтрации Conditions

Объединение условий только по and (MVP)

Операции сравнения (op):

- = (по умолчанию)
- >
- >=
- <
- <=
- in
- like (на перспективу)

Условия сравнения применимы к численным типам, датам, временам и таймштампам.

Варианты определения условий фильрации:

По равенству

```
{
    "query": {
        "office": {
            "conditions": {
                "phone": "(347) 246-53-00"
            }
        }
    }
}
```

На основе сравнения, краткая запись

```
{
    "query": {
        "office": {
            "conditions": {
                "area": [">","130"]
            }
        }
    }
}
```

На основе сравнения, полная запись

```
{
    "query": {
        "office": {
            "conditions": {
                "area": {
                    "op": ">",
                    "value": "130"
                }
            }
        }
    }
}
```

Комплексное условие

```
{
    "query": {
        "office": {
            "conditions": {
                "area": [">","130"],
                "floor": ["in", [1, 2]]
            }
        }
    }
}
```

**OR (ИЛИ) в условиях**

В **conditions** СМЭВ QL запроса поддерживается возможность указывать логический ИЛИ через зарезервированное слово **or**.

В блок **or** необходимо передать массив объектов, содержащих условия в свою очередь объединённые логическим **И** (**AND**).

Все условия, находящиеся на одном уровне с **or** группируются через логическое **И** (**AND**), как при обычном СМЭВ QL Запросе.

Пример:

```
"conditions": {
"lastname": "П",
"middlename": "И",
"birthdate": "2021-11-29 00:00:00",
"or": [
    {
        "vin": "в1"
    },
    {
        "vin2": "в2",
        "model": "bmw"
    }
],
"fetch": {
    "order": [["id", "ASC"], ["number", "DESC"]], // ASC default
    "page": [2, 10] // limit 10 offset 10
}
}
```

Условия описанного выше запроса (без учета fetch) соберутся в следующую конструкцию:

```
...
WHERE
        (lastname = 'П' AND middlename = 'И' AND birthdate = '2021-11-29 00:00:00')
    OR
        (vin ='в1')
    OR
        (vin2='в2' AND model = 'bmw')
...
```

### 8.2.2 Сортировка и пагинация

В блоке **conditions** опционально можно добавить блок **fetch**, в котором указывать условия сортировки и выбора страниц.

Пример блока:

```
"conditions": {
"lastname": "П",
"middlename": "И",
"birthdate": "2021-11-29 00:00:00",
"fetch": {
    "order": [["id", "ASC"], ["number", "DESC"]], // ASC default
    "page": [2, 10] // limit 10 offset 10
}
}
```

Если порядок сортировки не указан, то применяется ASC.

Если не указаны страницы, то по умолчанию всегда устанавливается первая страница с лимитом, равными параметру **default** блока **pagination** в файле конфигураций `application.yaml`.

Если запрашивается элементов на страницу больше, чем значение max, то должно использоваться дефолтное значение.

```
pagination:
    default: 100 //количество элементов на странице по умолчанию
    max: 1000 //максимальное количество элементов на странице
```

## 8.3 Эксплуатационные запросы

В пространстве методов server находятся методы, помогающие эксплуатации корректно конфигурировать хранилища данных относительно модели СМЭВ QL.

Метод возврата списка обязательных для создания индексов: `GET server/indexes/required`

В ответе должен возвращаться JSON с полями, используемыми в связях моделей (connections) и блоке conditions.allowed моделей, если они определены (conditions)

```
{
    "server": {
        "mnemonic": "#mnemonic",
        "instance": "#instance"
    },
    "indexes": {
        "connections": [{
                "source": "#source_name2",
                "table": "#table_name2",
                "fields": ["#field_name1", "#field_name2"]
            },
            {
                "source": "#source_name2",
                "table": "#table_name2",
                "fields": ["#field_name1", "#field_name2"]
            }
        ],
        "conditions": [{
                "source": "#source_name2",
                "table": "#table_name2",
                "fields": ["#field_name1", "#field_name2"]
            },
            {
                "source": "#source_name2",
                "table": "#table_name2",
                "fields": ["#field_name1", "#field_name2"]
            }
        ]
    }
}
```

## 8.4 Обработка запросов

### 8.4.1 Логирование мета-данных

У каждого запроса логируются данные из блока `credentials` в `info` и выше.

Формат строки лога:

```
{
    "level": "info",
    "time": "2000-01-01T01:01:01.111Z",
    "name": "#{smevql_server_name}.request",
    "system": {
        "mnemonic": "117bed7f-1c07-4079-a446-1161588db4e5",
        "instance_id": "ccb4a88f-f44b-43e7-8a97-3e45c8345e90",
        "user_id": "5ed38461-0907-486a-930a-7b443482932c"
    },
    "request": {
        "id": "df5a0073-c6be-4d8c-8eb2-9b2f4188a429",
        "sub_id": "0cdb59ce-224b-4118-8da1-c5ea08a5d955",
        "name": "driver_data",
        "purpose_id": "ed1170f1-3caa-4985-aa38-c9c5a190b770",
        "audit": "false",
        "audit_id": "fc1048fe-323d-4eeb-92df-5710b3d1d100",
        "audit_token": "39e47aac-45d2-44c1-8c26-2d9b28b1703b"
    }
}
```


Для каждого входящего запроса логируется каждый отправленный запрос к источнику:


```
{
    "level": "info",
    "time": "2000-01-01T01:01:01.111Z",
    "name": "#{smevql_server_name}.request",
    "system": {
        "mnemonic": "117bed7f-1c07-4079-a446-1161588db4e5",
        "instance_id": "ccb4a88f-f44b-43e7-8a97-3e45c8345e90",
        "user_id": "5ed38461-0907-486a-930a-7b443482932c"
    },
    "request": {
        "id": "df5a0073-c6be-4d8c-8eb2-9b2f4188a429",
        "sub_id": "0cdb59ce-224b-4118-8da1-c5ea08a5d955",
        "name": "driver_data",
        "purpose_id": "ed1170f1-3caa-4985-aa38-c9c5a190b770",
        "audit": "false",
        "audit_id": "fc1048fe-323d-4eeb-92df-5710b3d1d100",
        "audit_token": "39e47aac-45d2-44c1-8c26-2d9b28b1703b"
    },
    "source_request": {
        "id": "1cdb59ce-224b-4118-8da1-c5ea08a5d955",
        "source": "#{source_name}"
    }
}
```


Передача идентификаторов:

1. `source_request.id` передается в `queryId` тела запроса в Простор.
2. В `x-request-id` передается склейка `{request.id};{request.sub_id};{source_request.id}`

Логирование ответа


```
{
    "level": "info",
    "time": "2000-01-01T01:01:01.111Z",
    "name": "#{smevql_server_name}.response",
    "system": {
        "mnemonic": "117bed7f-1c07-4079-a446-1161588db4e5",
        "instance_id": "ccb4a88f-f44b-43e7-8a97-3e45c8345e90",
        "user_id": "5ed38461-0907-486a-930a-7b443482932c"
    },
    "request": {
```

```
        "id": "df5a0073-c6be-4d8c-8eb2-9b2f4188a429",
        "sub_id": "0cdb59ce-224b-4118-8da1-c5ea08a5d955",
        "name": "driver_data",
        "purpose_id": "ed1170f1-3caa-4985-aa38-c9c5a190b770",
        "audit": "false",
        "audit_id": "fc1048fe-323d-4eeb-92df-5710b3d1d100",
        "audit_token": "39e47aac-45d2-44c1-8c26-2d9b28b1703b"
    },
    "source_request": {
        "id": "1cdb59ce-224b-4118-8da1-c5ea08a5d955",
        "source": "#{source_name}"
    },
    "response": {
        "duration": "1ms",
        "code": 200,
        "result": "ok"
    }
}
```

### 8.4.2 Построение плана

Входные данные:

– загруженные модели, описывающие атрибутный состав сущностей, связи между сущностями и источники;
– запрос поступивший к серверу.

Необходимо определить набор запросов и порядок их исполнения СМЭВ QL сервером, обеспечить параллельное исполнение независимых запросов.

Вид плана запроса:

```
plan:
level: 1
    - source: prostore1
    query: SELECT id, phone, name FROM office WHERE phone = '(347) 246-53-00';
    pk: id
    alias: offices
level: 2
    - source: prostore2
    query: SELECT id, number, name, seats FROM cabinet WHERE office_id in (@offices) AND
available = 'true';
    pk: id
    alias: cabinets
    - source: prostore1
    query: SELECT id, number, floor FROM parking WHERE office_id in (@offices) free =
'true' AND available = 'true';
    pk: number, flor
    alias: parkings
level: 3
    - source: vostok7
    query: SELECT url FROM online_room WHERE cabinet_id in (@cabinets) AND public =
'true' AND software = 'zoom';
    pk: null
    alias: online_rooms
```

Сначала параллельно выполняются запросы первого уровня, затем запрос второго уровня на основе данных полученных на первом уровне.

Фомирование плана запроса основывается на внешнем объединении данных в сторону основной сущности.

Порядок формирования плана запроса:

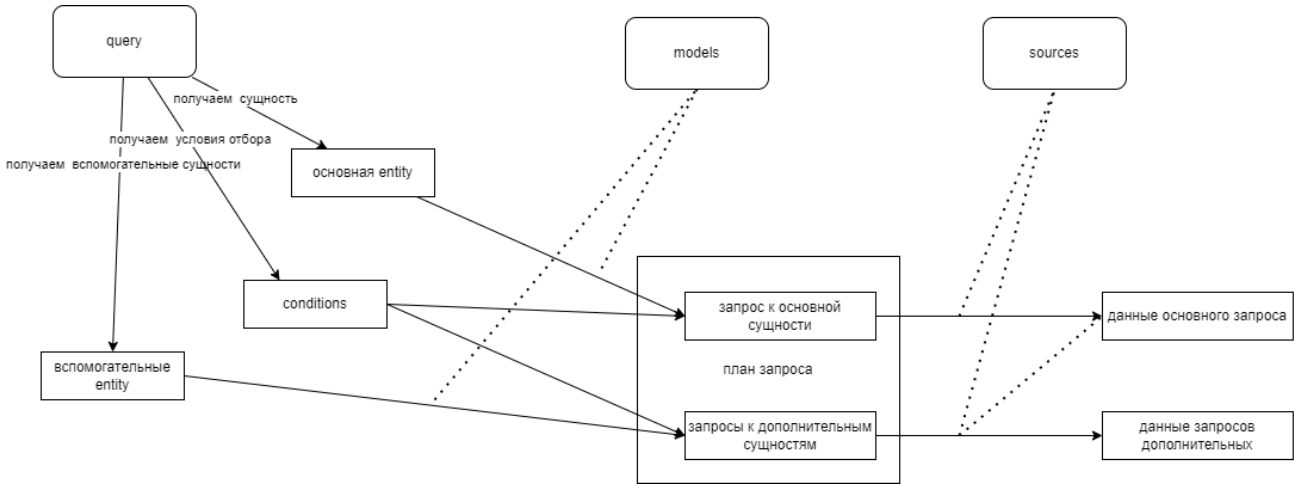
- на основе query определяется основная запрашиваемая сущность;
- на основе query определяются вспомогательные сущности;
- на основе query определяются conditions к основной сущности;
- на основе query определяются conditions к вспомогательным сущностям;
- на первом уровне плана формируется запрос к основной сущности на основе:
  - запрошенных атрибутов;
  - данных модели;
  - условий фильтрации.

Отмечаются поля, составляющие PK.

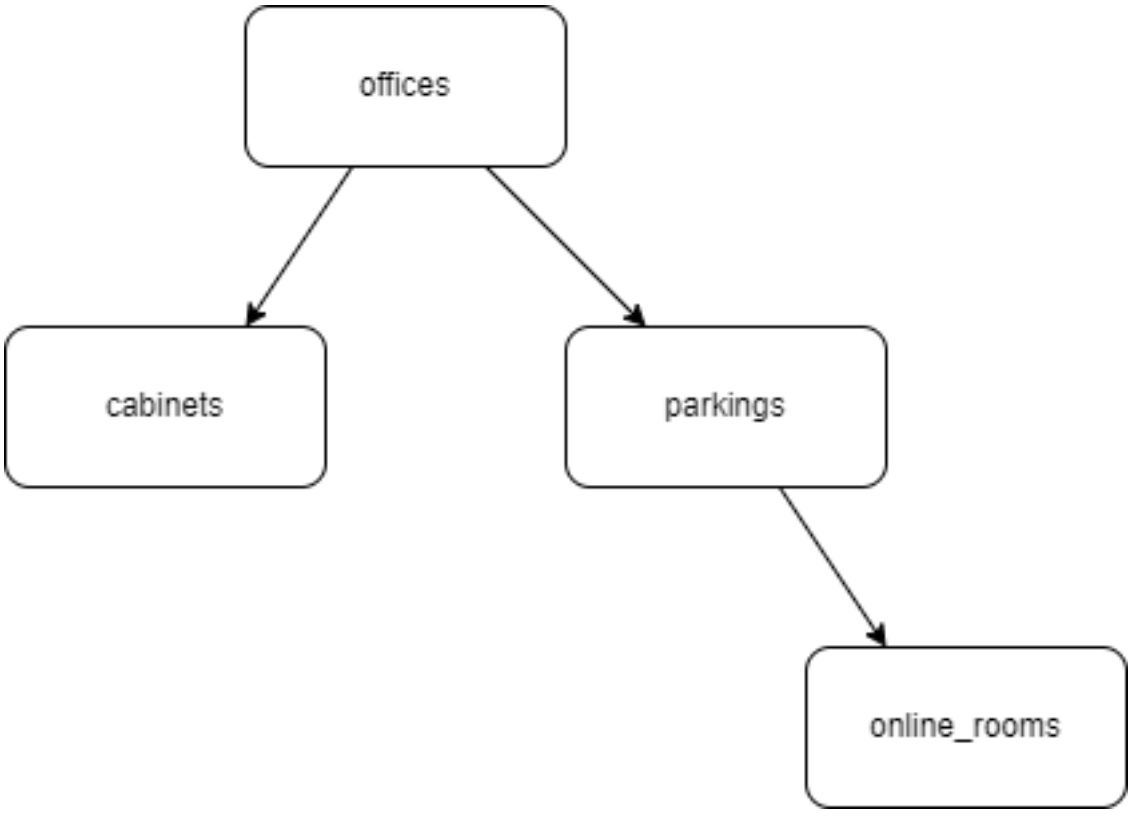
Первичные ключи, даже отсутствующие в атрибутах запроса к серверу должны быть в запросе к источнику.

- формируется запрос к вспомогательным сущностям на основе:
  - запрошенных атрибутов;
  - данных модели;
  - условий фильтрации;
  - с добавлением фильрации по PK основной или предшествующей сущности.

Первичные ключи, даже отсутствующие в атрибутах запроса к серверу должны быть в запросе к источнику, за исключением терминальных запросов.




План запроса соответсвует иерархии построенной от основной сущности на основе связей (modelconnections).




## 9 Ответы

Ответы СМЭВ QL сервера представляют из себя объект JSON, схема которого определяется составом запроса.

Например, для запроса:

```
{
"query":{
    "people":{
        "conditions":{
            "age":"<35"
        },
        "attributes":[
            "name",
            "phone",
            "vsu_code"
        ],
        "military_office":{
            "attributes":[
            "address"
            ]
        }
    }
},
"credentials":{

}
}
```

Ответ будет следующим:

```
{
"response":{
    "people":[
        {
```

```
            "name":"Иван",
            "phone":"+79011001010",
            "vsu_code":"1025",
            "military_office":[
            {
                "address":"г.Москва, ул.Угрешко"
            }
            ]
        },
        {
            "name":"Пётр",
            "phone":"+79022002020",
            "vsu_code":"1026",
            "military_office":[
            {
                "address":"г.Москва, Хилков переулок"
            }
            ]
        }
    ]
},
"credentials":{

}
}
```

# 10 Описание эндпоинтов

## 10.1 GET /ping

Метод проверяет доступность сервера

```
GET /ping
```

URL

```
http://localhost:8080/smevql/api/v1/ping
```

Запрос отправляется без параметров

Пример запроса:

```
curl -X 'GET' ¥
  'http://localhost:8080/smevql/api/v1/ping' ¥
  -H 'accept: plain/text'
```

Пример ответа:

```
pong
```

Таблица 10.1 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| pong | string | Текстовый ответ при успешном отклике сервера |

## 10.2 GET /states

Метод выводит состояния стейт-машины на данный момент

```
GET /states
```

URL

```
http://localhost:8080/smevql/api/v1/states
```

Запрос отправляется без параметров

Пример запроса:

```
curl -X 'GET' ¥
  'http://localhost:8080/smevql/api/v1/states' ¥
  -H 'accept: application/json'
```

Пример ответа:

```
[
  {
    "model": "book",
     "states": [
        {
            "state": "appointment",
            "attributes": [
                {
                    "name": "type",
                    "value": "APPOINTMENT"
                }
            ],
            "initial": true,
            "delete": false,
            "static": false
        },
        {
            "state": "cancel",
            "attributes": [
                {
                    "name": "type",
                    "value": "CANCEL"
                }
            ],
            "initial": false,
            "delete": false,
            "static": false
        },
        {
            "state": "observation",
            "attributes": [
                {
                    "name": "type",
                    "value": "D_OBSERVATION"
                }
            ],
            "initial": false,
            "delete": false,
            "static": false
        },
        {
            "state": "cancelled",
            "attributes": [
                {
                    "name": "type",
                    "value": "Cancel"
                }
            ],
            "initial": false,
            "delete": false,
            "static": false
```

```
        },
        {
            "state": "deleted",
            "attributes": [],
            "initial": false,
            "delete": true,
            "static": false
        }
    ],
    "events": [
        {
            "event": "book",
            "from": [
                "appointment"
            ],
            "to": "observation",
            "hooks": [],
            "confirm": null,
            "updatable": false,
            "updatable_attributes": []
        },
        {
            "event": "observation",
            "from": [
                "appointment"
            ],
            "to": "observation",
            "hooks": [],
            "confirm": null,
            "updatable": false,
            "updatable_attributes": []
        },
        {
            "event": "cancel",
            "from": [
                "appointment",
                "observation"
            ],
            "to": "cancel",
            "hooks": [],
            "confirm": null,
            "updatable": false,
            "updatable_attributes": []
        },
        {
            "event": "delete",
            "from": [
                "appointment",
                "observation",
                "cancel"
            ],
            "to": "deleted",
            "hooks": [],
            "confirm": null,
            "updatable": false,
            "updatable_attributes": []
        }
    ]
  ,
]
```

Таблица 10.2 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| model | string | имя модели |
| state | string | название состояния |
| attributes | array | набор атрибутов, описывающих состояние |
| name | string | имя атрибута |
| value | string | значение атрибута для описываемого состояния |
| initial | boolean | возможность создания записи |
| events | array | список событий изменения состояний |
| event | string | событие |
| from | string | массив состояний из которых возможен вызов события |
| to | string | в какое состояние переводится объект после события |

## 10.3 GET /states/{model}

Возвращает информацию о модели

```
GET /states/{model}
```

URL

```
http://localhost:8080/smevql/api/v1/states/{model}
```

Таблица 10.3 Параметры запроса

| Параметр | Тип данных | Описание |
|---|---|---|
| model | string | имя модели |

Пример запроса:

```
curl -X 'GET' ¥
  'http://localhost:8080/smevql/api/v1/states/{model}' ¥
  -H 'accept: application/json'
```

Пример ответа:

```
[
  {
    "model": "book",
      "states": [
          {
              "state": "appointment",
              "attributes": [
                  {
                      "name": "type",
                      "value": "APPOINTMENT"
                  }
              ],
              "initial": true,
              "delete": false,
              "static": false
          },
          {
              "state": "cancel",
              "attributes": [
```

```
                {
                    "name": "type",
                    "value": "CANCEL"
                }
            ],
            "initial": false,
            "delete": false,
            "static": false
        },
        {
            "state": "observation",
            "attributes": [
                {
                    "name": "type",
                    "value": "D_OBSERVATION"
                }
            ],
            "initial": false,
            "delete": false,
            "static": false
        },
        {
            "state": "cancelled",
            "attributes": [
                {
                    "name": "type",
                    "value": "Cancel"
                }
            ],
            "initial": false,
            "delete": false,
            "static": false
        },
        {
            "state": "deleted",
            "attributes": [],
            "initial": false,
            "delete": true,
            "static": false
        }
    ],
    "events": [
        {
            "event": "book",
            "from": [
                "appointment"
            ],
            "to": "observation",
            "hooks": [],
            "confirm": null,
            "updatable": false,
            "updatable_attributes": []
        },
        {
            "event": "observation",
            "from": [
                "appointment"
            ],
            "to": "observation",
            "hooks": [],
            "confirm": null,
            "updatable": false,
```

```
            "updatable_attributes": []
        },
        {
            "event": "cancel",
            "from": [
                "appointment",
                "observation"
            ],
            "to": "cancel",
            "hooks": [],
            "confirm": null,
            "updatable": false,
            "updatable_attributes": []
        },
        {
            "event": "delete",
            "from": [
                "appointment",
                "observation",
                "cancel"
            ],
            "to": "deleted",
            "hooks": [],
            "confirm": null,
            "updatable": false,
            "updatable_attributes": []
        }
    ]
  ,
]
```

Таблица 10.4 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| model | string | имя модели |
| state | string | название состояния |
| attributes | array | набор атрибутов, описывающих состояние |
| name | string | имя атрибута |
| value | string | значение атрибута для описываемого состояния |
| initial | boolean | возможность создания записи |
| events | array | список событий изменения состояний |
| event | string | событие |
| from | string | список состояний из которых возможен вызов события |
| to | string | в какое состояние переводится объект после события |

## 10.4 POST /states/{model}/{event}

Метод обновляет данные состояния модели по указанному событию

Метод

```
POST /states/{model}/{event}
```

URL

```
http://localhost:8080/smevql/api/v1/states/{model}/{event}
```

Таблица 10.5 Параметры запроса

| Параметр | Тип данных | Описание |
|---|---|---|
| model | string | имя модели |
| event | string | событие |

В теле запроса нужно заполнить поля:

```
{
  "state": {
    "conditions": {
      "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    },
  },
  "credentials": {
    "system": {
      "mnemonic": "string",
      "instance_id": "string",
      "user_id": "string"
    },
    "request": {
      "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "sub_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "name": "string",
      "purpose_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "audit": true,
      "audit_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "audit_token": "string"
    }
  }
}
```

Таблица 10.6 Параметры запроса

| Параметр | Тип данных | Описание |
|---|---|---|
| state | string | название состояния |
| conditions | string | указывается условие фильтрации записей (опционально) |
| credentials | array | системные реквизиты |
| mnemonic | string | мнемоника |
| instance_id | string | идентификатор экземпляра |
| user_id | string | идентификатор пользователя |
| request | array | параметры запроса |
| id | string | идентификатор запроса |
| sub_id | string | идентификатор подзапроса |
| name | string | имя запроса |
| purpose_id | string | идентификатор события, породившего запрос |
| audit | boolean | признак необходимости запроса аудита |
| audit_id | string | идентификатор аудита |
| audit_token | string | токен аудита |

Пример запроса:

```
curl -X 'POST' ¥
  'http://localhost:8080/smevql/api/v1/states/{model}/{event}' ¥
```

```
  -H 'accept: plain/text' ¥
  -H 'Content-Type: application/json' ¥
  -d '{
  "state": {
    "conditions": {
      "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    },
    "payload": {
      "book_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "patient_id": "string",
      "booking_type": "string",
      "case_number": "string",
      "preliminary_reservation": true,
      "email": "string",
      "mobile_phone": "string",
      "referral_id": "string",
      "cards_id": "string"
    }
  },
  "credentials": {
    "system": {
      "mnemonic": "string",
      "instance_id": "string",
      "user_id": "string"
    },
    "request": {
      "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "sub_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "name": "string",
      "purpose_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "audit": true,
      "audit_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "audit_token": "string"
    }
  }
}'
```

Пример ответа:

```
{
  "response": {
    "state": "booked"
},
```

Таблица 10.7 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| state | string | обновленный статус события |

## 10.5 GET /model

Метод выводит массив моделей сервера

```
GET /model
```

URL

```
http://localhost:8080/smevql/api/v1/model
```

Запрос отправляется без параметров

Пример запроса:

```
curl -X 'GET' ¥
  'http://localhost:8080/smevql/api/v1/model' ¥
  -H 'accept: application/json'
```

Пример ответа:


```
"resources": {
    "ticket1": {
        "name": "ticket",
        "description": "ticket",
        "version": "1.0",
        "available_versions": [
            "1.0"
        ],
        "fields": {
            "id": {
              "name": "id",
               "type": [
                   "string",
                   "STRING"
               ],
               "length": 0,
               "nullable": "NULL",
               "key": "NONE"
            },
            "passengerid": {
                "name": "passengerid",
                "type": [
                    "string",
                    "STRING"
                ],
                "length": 0,
                "nullable": "NULL",
                "key": "NONE"
            },
            "tripid": {
                "name": "tripid",
                "type": [
                    "string",
                    "STRING"
                ],
                "length": 0,
                "nullable": "NULL",
                "key": "NONE"
            },
            "number": {
                "name": "number",
                "type": [
                    "number",
                    "LONG"
                  ],
                  "length": 0,
                  "nullable": "NULL",
                  "key": "PRIMARY"
            },
            "bycard": {
                "name": "bycard",
                "type": [
                    "boolean",
                    "BOOLEAN"
                ],
                "length": 0,
```

```
            "nullable": "NULL",
            "key": "NONE"
        },
        "price": {
            "name": "price",
            "type": [
              "number",
              "DOUBLE"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        },
        "sold": {
            "name": "sold",
            "type": [
                "string",
                "TIMESTAMP"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        }
    },
    "connections": {
        "has_many": [],
        "belongs_to": [
            {
                "passenger1": {
                    "primary_key": [
                        "id"
                    ],
                    "foreign_key": [
                        "passengerid"
                    ]
                }
            },
            {
                "trip1": {
                    "primary_key": [
                        "id"
                    ],
                    "foreign_key": [
                        "tripid"
                    ]
                }
            }
        ]
    },
    "restrictions": {
        "personal_data": []
    },
    "conditions": {
        "allowed": [],
        "denied": [],
        "always": []
    },
    "extract": {
        "source": [
            {
                "name": "prostore",
                "table": "smevql_test.ticket1",
```

```
                "conditions": [
                    {
                        "eq": {
                            "field": "sold",
                            "extract": {
                                "source": "prostore",
                                "table": "smevql_test.ticket1",
                                "key": "sold",
                                "is": true
                            },
                            "isFallback": false
                        }
                    }
                ]
            }
        ]
    }
},
```

Таблица 10.8 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| name | string | название модели |
| version | string | текущая версия модели |
| available_versions | string | доступные версии модели |
| fields | string | список полей модели |

## 10.6 GET /model/{model}

Метод выводит данные выбранной модели

```
GET /model/{model}
```

URL

```
http://localhost:8080/smevql/api/v1/model/{model}
```

Запрос отправляется без параметров

Пример запроса:

```
curl -X 'GET' ¥
  'http://localhost:8080/smevql/api/v1/model/{model}' ¥
  -H 'accept: application/json'
```

Пример ответа:

```
[
  "ticket1": {
      "name": "ticket",
      "description": "ticket",
      "version": "1.0",
      "available_versions": [
          "1.0"
      ],
      "fields": {
          "id": {
            "name": "id",
             "type": [
                 "string",
                 "STRING"
             ],
```

```
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        },
        "passengerid": {
            "name": "passengerid",
            "type": [
                "string",
                "STRING"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        },
        "tripid": {
            "name": "tripid",
            "type": [
                "string",
                "STRING"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        },
        "number": {
            "name": "number",
            "type": [
                "number",
                "LONG"
              ],
              "length": 0,
              "nullable": "NULL",
              "key": "PRIMARY"
        },
        "bycard": {
            "name": "bycard",
            "type": [
                "boolean",
                "BOOLEAN"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        },
        "price": {
            "name": "price",
            "type": [
               "number",
               "DOUBLE"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        },
        "sold": {
            "name": "sold",
            "type": [
               "string",
                "TIMESTAMP"
            ],
            "length": 0,
            "nullable": "NULL",
```

```
            "key": "NONE"
        }
    }
  }
]
```

Таблица 10.9 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| name | string | название модели |
| version | string | текущая версия модели |
| available_versions | string | доступные версии модели |
| fields | string | список полей модели |

## 10.7 GET /model/{model}/{version}

Метод выводит версию выбранной модели

```
GET /model/{model}/{version}
```

URL

```
http://localhost:8080/smevql/api/v1/model/{model}/{version}
```

Запрос отправляется без параметров

Пример запроса:

```
curl -X 'GET' ¥
  'http://localhost:8080/smevql/api/v1/model/{model}/{version}' ¥
  -H 'accept: application/json'
```

Пример ответа:

```
[
  "ticket1": {
      "name": "ticket",
      "description": "ticket",
      "version": "1.0",
      "available_versions": [
          "1.0"
      ],
      "fields": {
          "id": {
             "name": "id",
              "type": [
                  "string",
                  "STRING"
              ],
              "length": 0,
              "nullable": "NULL",
              "key": "NONE"
          },
          "passengerid": {
              "name": "passengerid",
              "type": [
                  "string",
                  "STRING"
              ],
              "length": 0,
              "nullable": "NULL",
              "key": "NONE"
```

```
        },
        "tripid": {
            "name": "tripid",
            "type": [
                "string",
                "STRING"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        },
        "number": {
            "name": "number",
            "type": [
                "number",
                "LONG"
              ],
              "length": 0,
              "nullable": "NULL",
              "key": "PRIMARY"
        },
        "bycard": {
            "name": "bycard",
            "type": [
                "boolean",
                "BOOLEAN"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        },
        "price": {
            "name": "price",
            "type": [
               "number",
               "DOUBLE"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        },
        "sold": {
            "name": "sold",
            "type": [
               "string",
                "TIMESTAMP"
            ],
            "length": 0,
            "nullable": "NULL",
            "key": "NONE"
        }
    }
  }
]
```

Таблица 10.10 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| name | string | название модели |
| version | string | текущая версия модели |

| | | |
|---|---|---|
| available_versions | string | доступные версии модели |
| fields | string | список полей модели |

## 10.8 GET /sources

Метод выводит массив источников сервера

```
GET /sources
```

URL

```
http://localhost:8080/smevql/api/v1/sources
```

Запрос отправляется без параметров

Пример запроса:

```
curl -X 'GET' ¥
  'http://localhost:8080/smevql/api/v1/sources' ¥
  -H 'accept: application/json'
```

Пример ответа:

```
{
    "prostore": [
        {
            "version": "1.0",
            "adapter": "prostore",
            "protocol": "http",
            "host": "prostore",
            "port": 9090,
            "path": "api/v1/datamarts/query?format=json",
            "headers": [
                {
                    "content-type": "application/json"
                }
            ],
            "threads-count": 4,
            "connection-timeout": 30,
            "type": "rest"
        }
    ],
    "smevql": [
        {
            "version": "1.0",
            "adapter": "smevql",
            "protocol": "http",
            "host": "smevql-server",
            "port": 8080,
            "path": "data?format=json",
            "headers": [
                {
                    "content-type": "application/json"
                }
            ],
            "threads-count": 4,
            "connection-timeout": 30,
            "type": "rest"
        }
    ],
    "csv-uploader": [
        {
            "version": "1.0",
```

```
            "adapter": "confirm",
            "protocol": "http",
            "host": "csv-uploader",
            "port": 8080,
            "path": "version?format=json",
            "headers": [
                {
                    "content-type": "application/json"
                }
            ],
            "threads-count": 4,
            "connection-timeout": 30,
            "type": "rest"
        }
    ]
}
```

Таблица 10.11 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| version | string | версия |
| adapter | string | имя адаптера |
| protocol | string | протокол передачи данных |
| host | string | хост сервера |
| port | int | порт сервера |
| path | string | путь к источнику |
| headers | array | заголовки |
| threads-count | int | кол-во потоков |
| connection-timeout | int | время соединения |
| type | string | тип взаимодействия с источником |

## 10.9 POST /data

Запрашивает данные ресурсов в синхронном режиме

```
POST /data
```

URL

```
http://localhost:8080/smevql/api/v1/data
```

При запросе необходимо указать Headers:

- Key: content-type
- Value: application/json

Тело запроса должно содержать два обязательных блока:

Блок `query` - описание требуемых ресурсов и условий фильтрации данных; Блок `credentials` - общие данные запроса и информация о потребителе.

Пример блока `query`:

```
{
    "query": {
        "office": {
            "conditions": {
                "phone": "(347) 246-53-00"
            },
            "attributes": [
```

```
                "id",
                "phone",
                "name"
            ],
            "cabinet": {
                "conditions": {
                    "available": true
                },
                "attributes": [
                    "number",
                    "name",
                    "seats"
                ],
                "online_room": {
                    "conditions": {
                        "public": true,
                        "software": "zoom"
                    },
                    "attributes": [
                        "url"
                    ]
                },
                "parking": {
                    "conditions": {
                        "free": true,
                        "available": true
                    },
                    "attributes": [
                        "number",
                        "floor"
                    ]
                }
            }
        }
    }
}
```

Пример блока `credentials`:

```
"credentials":{
    "system":{
        "mnemonic":"117bed7f-1c07-4079-a446-1161588db4e5",
        "instance_id":"ccb4a88f-f44b-43e7-8a97-3e45c8345e90",
        "user_id":"5ed38461-0907-486a-930a-7b443482932c"
    },
    "request":{
        "id":"df5a0073-c6be-4d8c-8eb2-9b2f4188a429",
        "sub_id":"0cdb59ce-224b-4118-8da1-c5ea08a5d955",
        "name":"driver_data",
        "purpose_id":"ed1170f1-3caa-4985-aa38-c9c5a190b770",
        "audit":"false",
        "audit_id":"fc1048fe-323d-4eeb-92df-5710b3d1d100",
        "audit_token":"39e47aac-45d2-44c1-8c26-2d9b28b1703b"
    },
    "signature":[
    {
        "digest": null,
        "signature": null,
        "jsonpath": "/response"
    }
    ]
}
```

Таблица 10.12 Параметры запроса

| Параметр | Тип данных | Описание | Примечание |
|---|---|---|---|
| query | array | атрибут запроса данных внутри указывается название сущности, данные которой нужно получить | |
| conditions | string | указывается условие фильтрации записей (опционально) | – Правила задания условий<br>– Варианты определения условий фильтрации |
| fetch | array | опциональный блок условий | – Подробное описание блока fetch |
| attributes | array | Атрибуты, значения которых необходимо получить при выполнении запроса | |
| credentials | object | Объект, содержащий общие данные запроса и информация о потребителе | |
| mnemonic | string | мнемоника системы | |
| instance_id | string | идентификатор экземпляра системы | |
| user_id | string | идентификатор пользователя | |
| request | object | Объект описывает общие параметры запроса | |
| id | string | идентификатор запроса | |
| sub_id | string | идентификатор подзапроса | |
| name | string | имя запроса | |
| purpose_id | string | идентификатор события, инициировавшего запрос | |
| audit | boolean | возможность аудита | |
| audit_id | string | идентификатор аудита. Заполняется только, если audit = true | |
| audit_token | string | токен аудита. Заполняется только, если audit = true | |
| signature | object | Объект, описывает параметры ЭЦП | |
| digest | string | дайджест подписи (не заполняется) | |
| signature | string | подпись (не заполняется) | |

Пример запроса:

```
curl -X 'POST' ¥
  'http://localhost:8080/smevql/api/v1/data' ¥
  -H 'accept: application/json' ¥
  -H 'Content-Type: application/json' ¥
  -d '{
  "query": {
      "ticket": {
      "conditions": {},
      "attributes": [
          "id",
          "price"
      ]
    }
```

```
  },
  "credentials": {
    "system": {
      "mnemonic": "3fa45f64-5717-4562-b3fc-2c963f66afa6",
      "instance_id": "3fa45f64-5717-4562-b3fc-2c963f66afa6",
      "user_id": "3fa45f64-5717-4562-b3fc-2c963f66afa6"
    },
    "request": {
      "id": "6fa45f64-5717-4562-b3fc-2c963f66afa6",
      "sub_id": "7fa45f64-5717-4562-b3fc-2c963f66afa6",
      "name": "query",
      "purpose_id": "8fa45f64-5717-4562-b3fc-2c963f66afa6",
      "audit": true,
      "audit_id": "9fa45f64-5717-4562-b3fc-2c963f66afa6",
      "audit_token": "b3fc"
    },
    "signature": {
      "digest": "digest",
      "signature": "signature"
    }
  }
}'
```

Пример ответа:

```
{
    "response": {
        "ticket": [
            {
                "price": 298.8082301905419,
                "id": "3424f5b3-e337-4d0d-a046-a1c491ab3300"
            },
            {
                "price": 67.79323025646744,
                "id": "ad14fce2-04f5-45b8-b430-3ff9efb5d4ca"
            }
        ]
    },
    "credentials": {
        "system": {
            "mnemonic": "3fa45f64-5717-4562-b3fc-2c963f66afa6",
            "instance_id": "4fa45f64-5717-4562-b3fc-2c963f66afa6",
            "user_id": "5fa45f64-5717-4562-b3fc-2c963f66afa6"
        },
        "request": {
            "id": "6fa45f64-5717-4562-b3fc-2c963f66afa6",
            "sub_id": "7fa45f64-5717-4562-b3fc-2c963f66afa6",
            "name": "query",
            "purpose_id": "8fa45f64-5717-4562-b3fc-2c963f66afa6",
            "audit": true,
            "audit_id": "9fa45f64-5717-4562-b3fc-2c963f66afa6",
            "audit_token": "b3fc"
        },
        "signature": {
            "digest": "digest",
            "signature": "signature"
        },
        "response": {
            "id": "ec26f676-5931-11ee-9044-eb1795d841ef",
            "sub_id": "ec26f677-5931-11ee-9044-eb1795d841ef",
            "started_at": "2023-09-22 13:22:24.456 +0300",
            "finished_at": "2023-09-22 13:22:24.530 +0300"
```

```
        }
    }
}
```

Таблица 10.13 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| response | object | объект с описанием ресурсов, содержащих массивы данных запросов |
| resourse_data | array | массивы данных, запрашиваемых ресурсов (имя объекта) |

### 10.9.1 Правила задания условий

1. Объединение условий только по **and**
2. Операции сравнения:
   - `=` (по умолчанию);
   - `>`;
   - `>=`;
   - `<`;
   - `<=`;
   - `in`.
3. Условия сравнения применимы к численным типам, датам, временам и таймштампам.

### 10.9.2 Варианты определения условий фильтрации

1. Простое равенство

```
{
    "query": {
        "office": {
            "conditions": {
                "phone": "(347) 246-53-00"
            }
        }
    }
}
```

2. Сравнение краткое

```
{
    "query": {
        "office": {
            "conditions": {
                "area": [">","130"]
            }
        }
    }
}
```

3. Сравнение полное

```
```

{

**«query»: {**

**«office»: {**

**«conditions»: {**

**«area»: {**

«op»: «>», «value»: «130»

}

}

}

}

}

4. Комплексное условие

```
{
    "query": {
        "office": {
            "conditions": {
                "area": [">","130"],
                "floor": ["in", [1, 2]]
            }
        }
    }
}
```

В conditions СМЭВ QL запроса поддерживается возможность указывать логический ИЛИ через зарезервированное слово **or**.

В блок or необходимо передать массив объектов, содержащих условия в свою очередь объединённые логическим **И (AND)**.

Все условия, находящиеся на одном уровне с **or** группируются через логическое И (AND), как при обычном СМЭВ QL Запросе.

Пример:

```
"conditions": {
  "lastname": "П",
  "middlename": "И",
  "birthdate": "2021-11-29 00:00:00",
  "or": [
      {
          "vin": "в1"
      },
      {
          "vin2": "в2",
          "model": "bmw"
      }
  ],
  "fetch": {
      "order": [["id", "ASC"], ["number", "DESC"]], // ASC default
      "page": [2, 10] // limit 10 offset 10
  }
}
```

Условия описанного выше запроса (без учета fetch) соберутся в следующую конструкцию:

```
WHERE
        (lastname = 'П' AND middlename = 'И' AND birthdate = '2021-11-29 00:00:00')
    OR
        (vin ='в1')
    OR
        (vin2='в2' AND model = 'bmw')
```

### 10.9.3 Подробное описание блока fetch

В блоке `conditions` опционально можно добавить блок `fetch`, в котором указывать:

- условия сортировки (**order**);
- условия выбора страниц (**page**);
- условия выборки доступных для каждого ресурса событий машины состояния (**show_events**);
- применение локализованных переменных (**localize**);
- условия выбора версии данных на указанную дату, диапазон дат, дельту, диапазон дельт (**for_system_time**, **for_system_time_started**, **for_system_time_finished**).

```
"conditions": {
  "lastname": "П",
  "middlename": "И",
  "birthdate": "2021-11-29 00:00:00",
  "fetch": {
      "order": [["id", "ASC"], ["number", "DESC"]], // ASC default
      "page": [2, 10] // limit 10 offset 10
  }
}
```

Если порядок сортировки не указан, то применяется ASC.

Если не указаны страницы, то по умолчанию всегда устанавливается первая страница с лимитом, равными параметру `default` блока `pagination` в файле конфигураций `application.yaml`.

Если запрашивается элементов на страницу больше, чем значение max, то должно использоваться дефолтное значение.

В блок `fetch` можно добавить атрибут выборки доступных евентов (events стэйт-машины (Создание карты машины-состояний) `show_events`):

```
"conditions": {
  "lastname": "П",
  "middlename": "И",
  "birthdate": "2021-11-29 00:00:00",
  "fetch": {
      "order": [["id", "ASC"], ["number", "DESC"]], // ASC default
      "page": [2, 10] // limit 10 offset 10,
      "show_events": true // false default
  }
}
```

Если данный атрибут указан и возвращаемый объект управляется стейт-машиной, то его атрибуты дополняются доступными евентами в атрибуте `available_state_events`:

```
"response": {
       "office": [
           {
               "id": "10459",
               "phone": "(347) 246-53-00",
               "name": "ГБУЗ РБ Поликлиника № 50 г.Уфа, Кабинет вакцинации COVID-
19",
```

```
            "available_state_events": [ "book", "cancel" ]
        },
        {
            "id": "10461",
            "phone": "(347) 246-53-00",
            "name": "ГБУЗ РБ Поликлиника № 50 г.Уфа, Кабинет вакцинации COVID-
19",
            "available_state_events": [ "book" ]
        },
        {
            "id": "6344",
            "phone": "(347) 246-53-00",
            "name": "ГБУЗ РБ Поликлиника № 50 г.Уфа, отделение 26",
            "available_state_events": [ "reserve", "cancel" ]
        }
    ]
}
```

Если евентов нет, то выводится пустой массив. Если объект не управляется стейт-машиной, то директива просто игнорируется.

В блоке локализованных переменных `localize` можно задать массив произвольных именованных переменных.

Данные переменные можно указывать в качестве критерия отбора данных для других ресурсов.

При этом получение данных ресурса с критерием включающим локализованную переменную выполняется после получения данных ресурса, где эта переменная была определена.

Пример определения локализованных переменных:

```
"query": {
  "mdm": {
    "conditions": {
      "eduejd": "00650abd-dde7-4a3b-a44e-475285ff276f",
      "fetch": {
        "localize": [
          "eduejd01",
          "eduejd02"
        ]
      }
```

Пример указания локализованной переменной в качестве критерия отбора данных другого ресурса:

```
"students_eduejd01": {
    "conditions": {
        "id": "@mdm.eduejd01",
    },
```

> Примечание:
>
> Использование локализованных переменных допускается только для ресурсов, находящихся на одном уровне в запросе.

Если необходимо получить данные конкретной версии (версия может характеризоваться временной отметкой или номером дельты), то в блоке `fetch` указывается соответствующий системный параметр:

- `for_system_time` - получение текущих данных витрины на заданную отметку версии. Возможны следующие варианты определения отметки версии:

- Номер дельты (целое число), например: `"for_system_time": 231`
- Время (timestamp), например:`"for_system_time": "2024-02-09 12:05:03"`

– `for_system_time_started` - получение новых/измененных данных витрины с заданным диапазоном версий. Возможны следующие варианты определения отметки диапазона версий:
  - Диапазон номеров дельт, например: `"for_system_time_started": [210,215]`
  - Диапазон времени, например: `"for_system_time_started": ["2024-02-09 12:00:00","2024-02-09 15:59:59"]`
– `for_system_time_finished` - получение удалённых данных витрины с заданным диапазоном версий. Возможны следующие варианты определения отметки диапазона версий:
  - Диапазон номеров дельт, например: `"for_system_time_finished": [210,215]`
  - Диапазон времени, например: `"for_system_time_finished": ["2024-02-09 12:00:00","2024-02-09 15:59:59"]`
– `for_system_time_cn` - получение текущих данных витрины на заданный номер операции (CN, целое число), например: `"for_system_time_cn": 114`;
– `for_system_time_started_cn` - получение новых/ измененных данных витрины с заданным диапазоном номеров операций, например: `"for_system_time_started_cn": [110,115]`;
– `for_system_time_finished_cn` - получение удалённых данных витрины с заданным диапазоном номеров операций, например: `"for_system_time_finished_cn": [110,115]`.

Общие правила работы с системными параметрами:

1. СМЭВ QL фиксирует время получения запроса для подстановки в запросы к ресурсам у которых не указана отметка.
2. Указание отметок времени может быть на уровне родительского ресурса, так и на уровне вложенного ресурса.
3. При наличии отметок и на уровне родительского и на уровне вложенного ресурса, приоритет имеет более близкая к ресурсу отметка.
4. При чтении данных ресурса из нескольких источников, фильтрация на основе отметки ресурса распространяется на все таблицы.
5. Для таблиц типа стенделон и прокси Простор игнорирует конструкцию `for system_time`.
   a. Для не версионированных таблиц запросы с `for system_time as off` возвращают актуальное значение без учета отметки
   b. Для не версионированных таблиц запросы с `for system_time started/finished` возвращают пустое множество без учета отметок диапазона
6. При наличии отметок на уровне запроса, она распространяется на все ресурсы, для которых метка не определена.
7. Совмещение в одном запросе различных режимов получения данных (добавленных и удаленных), например 1 ресурс добавленные записи, а 2 ресурс удаленные записи недопустимо, должен вернуться код 409 «Недопустимое сочетание условий выборки данных». Остальные сочетания режимов получения данных допустимы.
8. Метки времени на ресурсы сиблинги не распространяются.

Примеры запросов с указанием временных меток:

**получение данных офисов и кабинетов на таймштамп**

```
{
    «query»: {
        «office»: {
            «conditions»: {
                «phone»: «(347) 246-53-00», «fetch»: {
                    «for_system_time»: «2024-02-09 12:05:03»
                }
            }, «attributes»: [
                «id», «phone», «name»
            ], «cabinet»: {
                «conditions»: {
                    «available»: true
                }, «attributes»: [
                    «number», «name», «seats»
                ]
            }
        }
    }
}
```

**получение данных офисов на 120 дельту и кабинетов на 45 дельту**

```
{
    "query": {
        "office": {
            "conditions": {
                "phone": "(347) 246-53-00",
                "fetch": {
                "for_system_time": 120
                }
            },
            "attributes": [
                "id",
                "phone",
                "name"
            ],
            "cabinet": {
                "conditions": {
                    "available": true,
                "fetch": {
                "for_system_time": 45
                }
            },
                "attributes": [
                    "number",
                    "name",
                    "seats"
                ]
            }
        }
    }
}
```

**получение данных офисов и кабинетов добавленных с 210 по 215 дельту**

```
{
    "query": {
        "office": {
            "conditions": {
                "phone": "(347) 246-53-00",
                "fetch": {
                "for_system_time_started": [210,215]
                }
            },
            "attributes": [
                "id",
                "phone",
                "name"
            ],
            "cabinet": {
                "conditions": {
                    "available": true
                },
                "attributes": [
                    "number",
                    "name",
                    "seats"
                ]
            }
        }
    }
}
```

**получение данных офисов и кабинетов удаленных за диапазон времени**

```
{
    "query": {
        "office": {
            "conditions": {
                "phone": "(347) 246-53-00",
                "fetch": {
                "for_system_time_finished": ["2024-02-09 12:00:00","2024-02-09 15:59:59"]
                }
            },
            "attributes": [
                "id",
                "phone",
                "name"
            ],
            "cabinet": {
                "conditions": {
                    "available": true
                },
                "attributes": [
                    "number",
                    "name",
                    "seats"
                ]
            }
        }
    }
}
```

**получение данных офисов на 231 дельту и всех актуальных кабинетов (сиблинг)**

```
{
```

```
    "query": {
        "office": {
            "conditions": {
                "phone": "(347) 246-53-00",
                "fetch": {
                "for_system_time": 231
                }
            },
            "attributes": [
                "id",
                "phone",
                "name"
            ]
        },
        "cabinet": {
                "conditions": {
                    "available": true
                },
                "attributes": [
                    "number",
                    "name",
                    "seats"
                ]
        }
    }
}
```

### 10.10 GET /server/indexes/required

Формирует рекомендации по индексам на основании описания моделей

```
GET /server/indexes/required
```

URL

```
http://localhost:8080/smevql/api/v1/server/indexes/required
```

Запрос отправляется без параметров

Пример запроса:

```
curl -X 'GET' ¥
  'http://localhost:8080/smevql/api/v1/server/indexes/required' ¥
  -H 'accept: application/json'
```

Пример ответа:

```
{
  "server": {
      "mnemonic": "00000000-d759-11ed-84c6-47c59cf9ecf6",
      "instance": "00000001-d759-11ed-84c6-47c59cf9ecf6"
  },
  "indexes": {
      "connections": [
          {
              "source": "prostore",
              "table": "smevql_test.ticket1",
              "fields": [
                  "passengerid",
                  "tripid"
              ]
          }
      ],
      "conditions": [
```

```
        {
            "source": "prostore",
            "table": "smevql_test.passenger1",
            "fields": [
                "lastname",
                "firstname",
                "code"
            ]
        },
        {
            "source": "prostore",
            "table": "mvd.vehicleregdata",
            "fields": [
                "vehiclevin",
                "vehiclevin2"
            ]
        }
      ]
    }
  }
}
```

Таблица 10.14 Описание ответа

| Параметр | Тип данных | Описание |
|---|---|---|
| server | string | данные сервера |
| mnemonic | string | мнемоника |
| instance | string | экземпляр сервера |
| indexes | string | данные индекса |
| connections | array | массив связей |
| source | string | название источника |
| table | string | название таблицы |
| fields | string | список полей модели |
| conditions | array | массив условий фильтрации записей |

# 11 Ошибки

Ошибки выводятся в блоке **response**:

```
{
"response": {
    "errors": [
    {
        "error": "Запрещен вывод атрибутов без переданного guard",
        "code": "401"
    }
    ]
},
"credentials": {

}
}
```

## 11.1 Базовые ошибки СМЭВ Ql

### 1XX Ошибки разбора запроса

– 101 Запрос не должен называться „errors“ — Неправильное название

запроса „errors“

**2XX Ошибки модели**

- 201 Неизвестный атрибут — У ресурса не найден атрибут с соотв. именем
- 202 Неизвестный ресурс — Ресурс с соотв. именем не зарегистрирован в модели данных
- 203 Неизвестная связь — Не найдена связь между двумя ресурсами
- 204 Неправильная связь — Размеры ключей не соответствуют для соединения одного ресурса с другим

**3XX Ошибки источников**

- 301 Неизвестный источник — Не найдено описание источника данных с соотв. именем
- 302 Неизвестный адаптер — Не найден адаптер с соотв. именем

**4XX Ошибки доступов и ограничений**

- 401 Запрещен вывод атрибутов без переданного guard — Среди запрашиваемых атрибутов есть атрибут с невыполненными ограничениями в блоке guard (не переданы в запросе атрибуты из guard)
- 402 Недостаточно атрибутов для выбора источника — Среди атрибутов фильтрации нет атрибутов, необходимых для выбора источника
- 403 Запрещенные атрибуты для поиска — Среди атрибутов фильтрации есть атрибуты, указанные в блоке denied модели
- 404 Атрибуты для поиска не разрешены — Среди атрибутов фильтрации есть атрибуты, которые не указаны в разрешающем блоке allowed модели
- 405 Попытка переопределения фиксированных условий поиска — Среди атрибутов фильтрации есть атрибуты, которые пытаются переопределить фиксированные ограничения поиска в блоке always модели

**9XX Прочие ошибки**

- 901 Непредвиденная ошибка — Непредвиденная ошибка

**СОДЕРЖАНИЕ**