ОПИСАНИЕ АРХИТЕКТУРЫ

СИСТЕМЫ ТЕХНИЧЕСКОГО ДОКУМЕНТООБОРОТА

ФЕНИКС

Автор:	ООО «Цифровая Эпоха»
Дата создания:	01.03.2023
Дата обновления:	07.09.2023
Версия:	2.0

С. Петербург, 2023

ОГЛАВЛЕНИЕ

1 Архитектура системы. Схема

Система Феникс построена в микро-сервисной архитектуре. Принципиальная схема представлены на Рис. 1.

Для работы с системой пользователь открывает в браузере SPA приложение на JS (Vue framework) и взаимодействует с системой посредством сервиса API Gateway, который является единой точкой входа. Аутентификация осуществляется через Keycloack.

На схеме также изображены сервисы прикладного уровня и сервисы платформы, который взаимодействуют через RMQ - message broker.

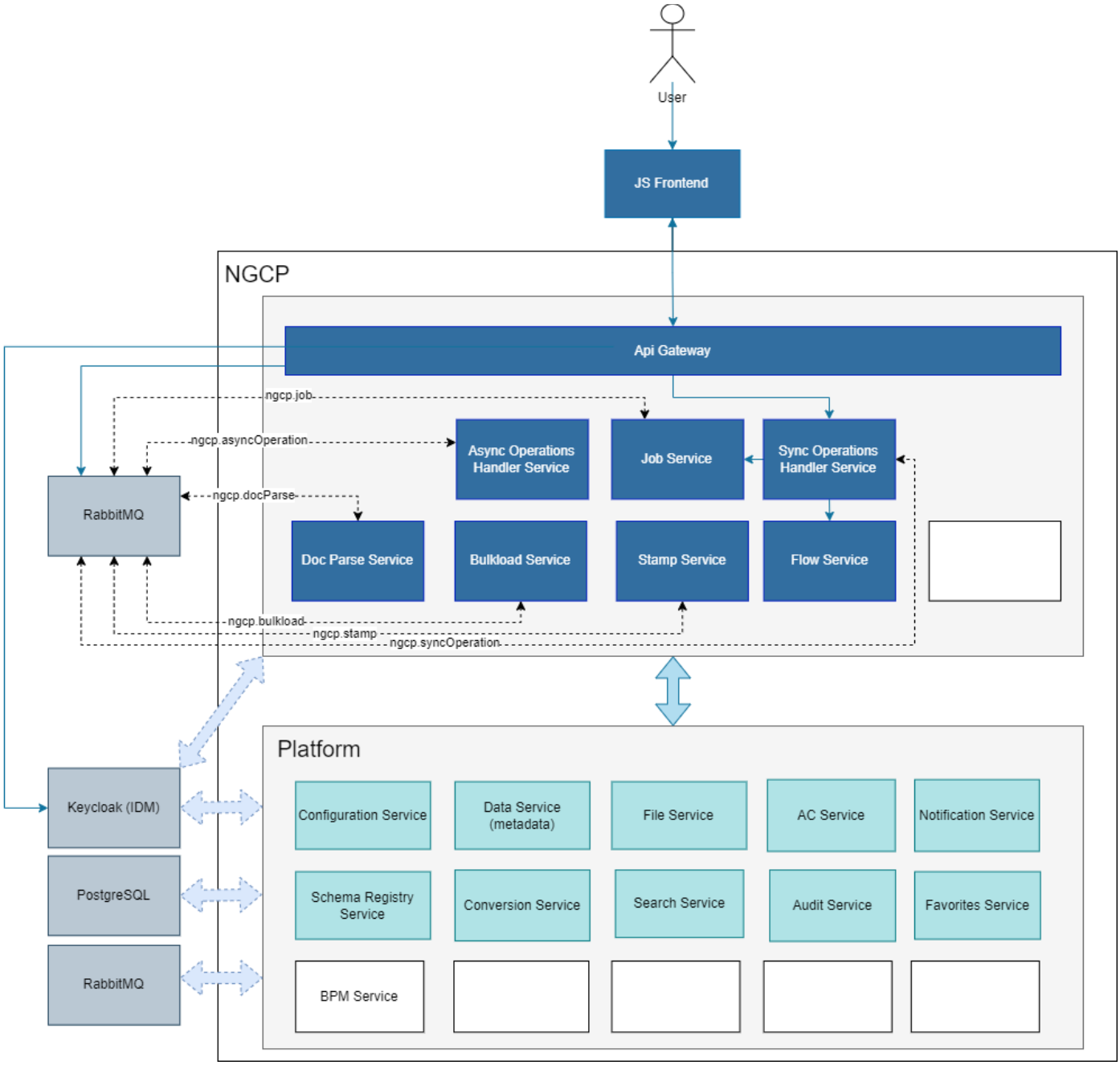


Рис. 1. Архитектура системы Феникс

2 Описание сервиса API Gateway

Сервис обеспечивает единую точку входа для всех клиентов системы:

- Пользовательских интерфейсов (WEB/Десктоп клиенты);
- интеграционных шин;
- аналитических служб

Сервис стоит на границе между внешним миром и демилитаризованным окружением и, условно, выполняет роль брандмауэра верхнего уровня.

3 Технические требования

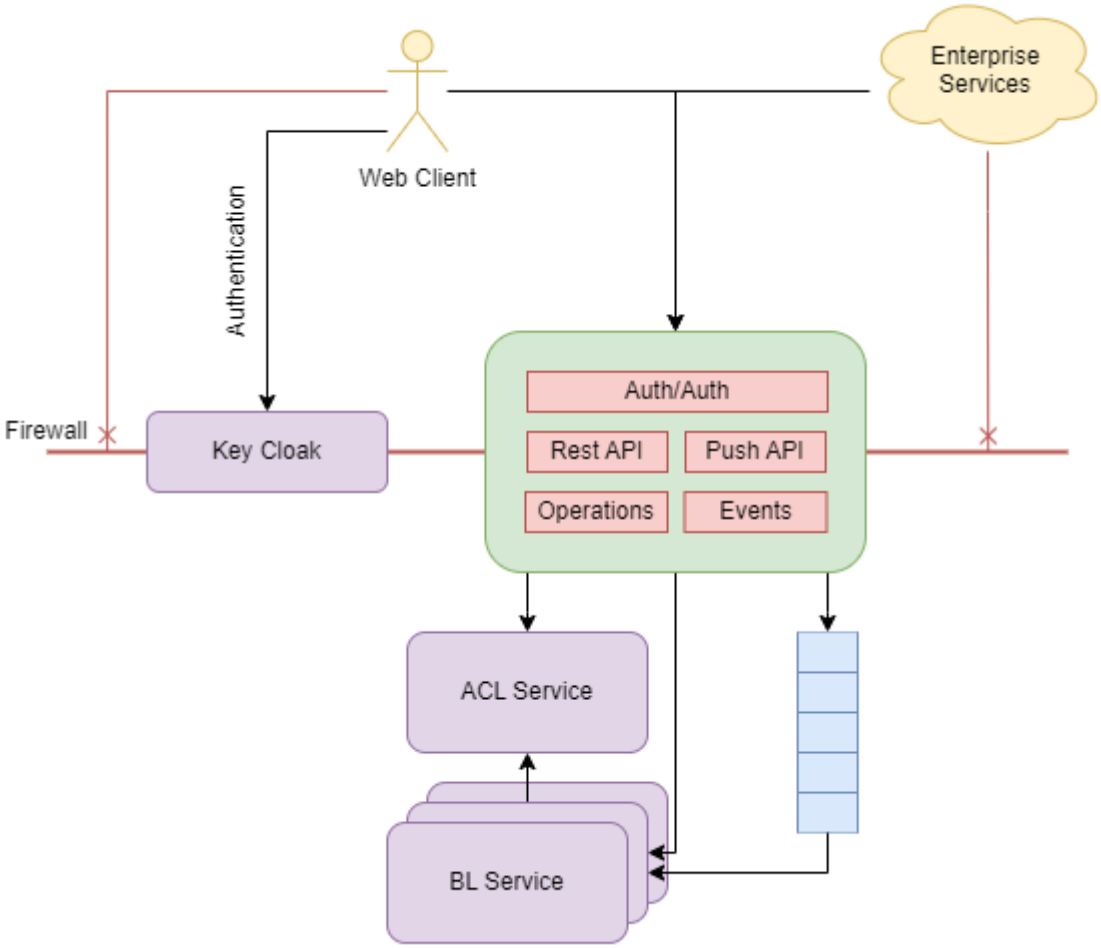
3.1 Общие сведения

Сервис-шлюз является единой точкой доступа ко всем функциям системы. Сервис предоставляет:

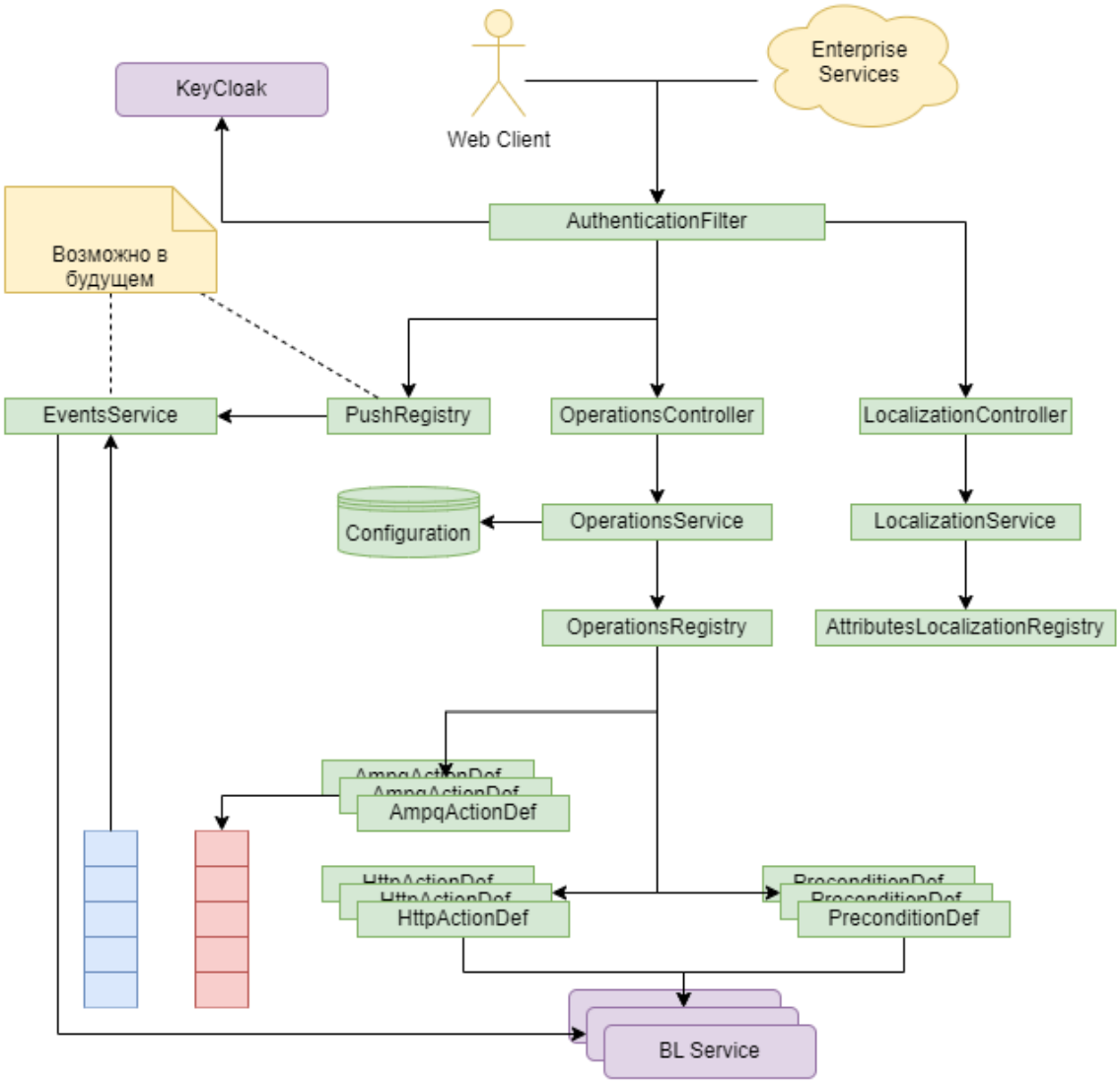
- WEB API для синхронного взаимодействия;
- WEB API для асинхронного взаимодействия;
- WEB API для подписки на события;
- документацию по WEB API для автоматической генерации клиентского кода;
- механизмы аутентификации и авторизации.

3.2 Архитектура

3.2.1 Контекст



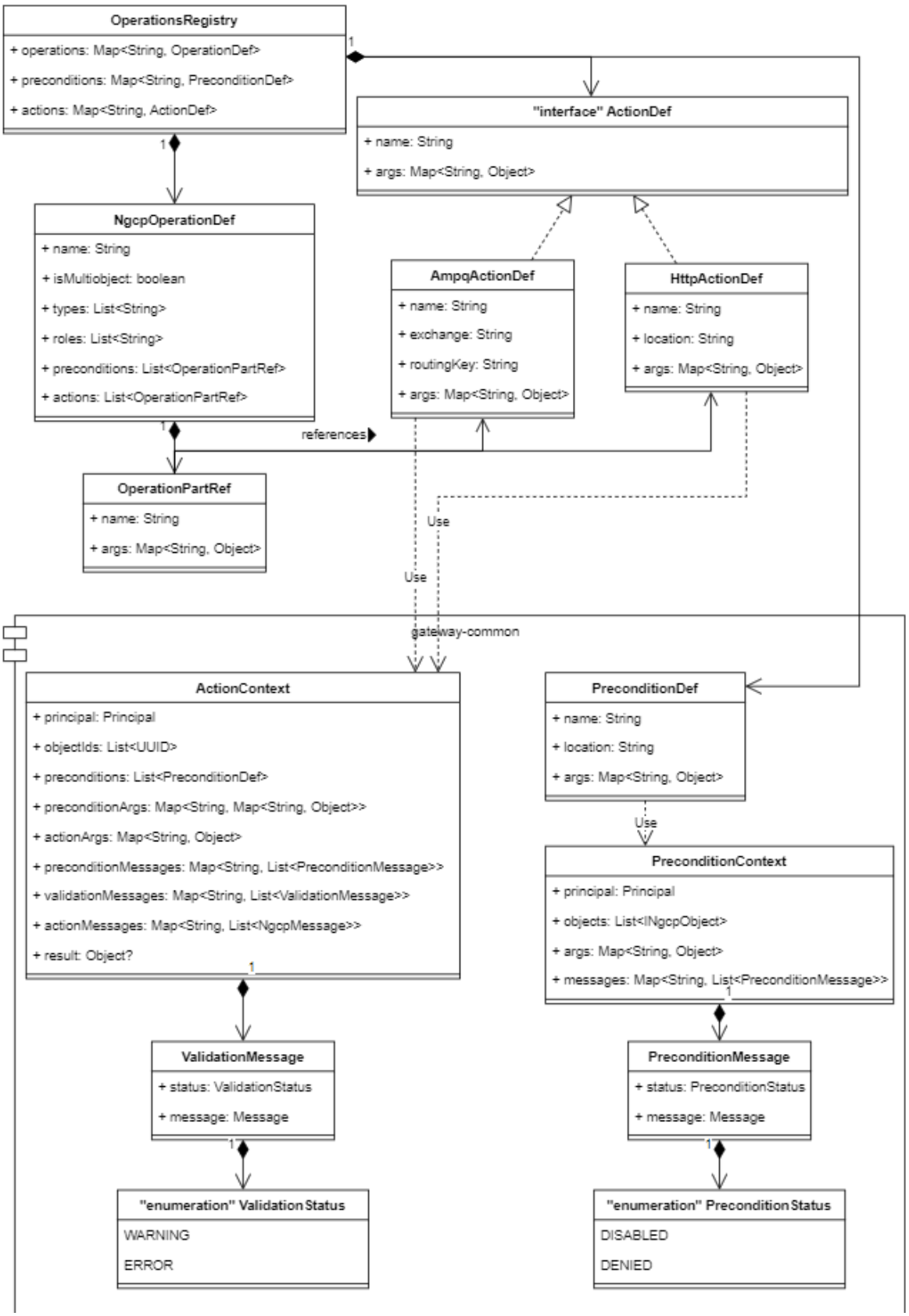
3.2.2 Контейнер



3.2.3 Компоненты

3.2.3.1 Диаграмма классов: операции, предусловия, действия

Диаграмма классов: операции, предусловия, действия



3.2.4 Принципы взаимодействия

Синхронное взаимодействие клиентов с системой производится по протоколу HTTP. В качестве формата для передачи сообщений используется JSON. Информация об API для

создания клиентов генерируется с использованием swagger и springdoc.
Уведомления для клиентов реализуется при помощи Server Sent Events (SSE), поддержка которого встроена в Spring MVC (**Реализовать в будущем**).

3.2.5 Авторизация

3.2.5.1 Общие сведения

Авторизации пользователей производится в 2 этапа.

1. Решается вопрос о доступности того или иного API вызова в зависимости от того:
    - кто производит вызов;
    - что содержится в документах, которые являются контекстом для вызова (см. Авторизация по принципалу и контексту).
2. Решается вопрос о возможности успешно обработать запрос в зависимости от прав на документы, которые являются контекстом для вызова (см. ACL Авторизация).

3.2.5.2 Авторизация по принципалу и контексту

Данные о принципале (например, набор ролей) передается в составе JWT от сервера авторизации Key Cloak. Сервис-шлюз является единственным арбитром при определении доступности того или иного API вызова для пользователя в соответствии с набором ролей. В случае если принципал не проходит авторизацию по контексту, действие проваливается с HTTP кодом 403 Forbidden. Если пользователь ещё не прошел процесс аутентификации, то пользователю возвращается ответ, соответствующий схеме аутентификации (см. Аутентификация).

3.2.5.3 ACL Авторизация

Контроль прав доступа к действиям, а также на документы, производится в ходе обработки запроса теми BL сервисами, которые участвуют в обработке запроса. Если пользователь не авторизован на выполнение тех или иных действий, о чем сообщил один из BL сервисов, пользователю возвращается ответ с HTTP кодом 403 Forbidden.

3.2.6 Аутентификация

3.2.6.1 Общие сведения

Аутентификация производится по протоколу *OpenID Connect*. Аутентификацию производит сервер аутентификации *Keycloak*.

3.2.6.2 Настройка аутентификации Spring Security

В Spring Security встроена поддержка аутентификации по протоколу *OpenID Connect*. В случае использования Spring Boot требуется подключить стартер *org.springframework.boot:spring-boot-starter-oauth2-client*. Пример конфигурации в файле *application.properties*:

```
spring.security.oauth2.client.provider.keycloak.authorization-uri = [authorizationEndpoint]
spring.security.oauth2.client.provider.keycloak.token-uri = [tokenEndpoint]
spring.security.oauth2.client.provider.keycloak.user-info-uri = [userinfoEndpoint]
spring.security.oauth2.client.provider.keycloak.jwk-set-uri = [jwksUri]
spring.security.oauth2.client.provider.keycloak.user-name-attribute = sub

spring.security.oauth2.client.registration.keycloak.client-id = [clientId]
spring.security.oauth2.client.registration.keycloak.client-secret = [clientSecret]
spring.security.oauth2.client.registration.keycloak.redirect-uri = {baseUrl}/login/oauth2/code/{registrationId}
spring.security.oauth2.client.registration.keycloak.authorization-grant-type = authorizationCode
spring.security.oauth2.client.registration.keycloak.scope = openid
```

Copy

Здесь:

- *[authorizationEndpoint]*, *[tokenEndpoint]*, *[userinfoEndpoint]*, *[jwksUri]* - значения из полей JSON объекта, который возвращает сервер Keycloak в ответ на запрос */realms/{realm-id}/.well-known/openid-configuration*.

- *[clientId]* - идентификатор клиента, см. Настройка клиента Keycloak.
- *[clientSecret]* - клиентский пароль, см. Настройка клиента Keycloak.

3.2.6.3 Настройка клиента Key Cloak

С точки зрения Keycloak, API шлюз является клиентом, который запрашивает разрешение на доступ к ресурсам пользователя. Для доступа к Keycloak со стороны API шлюза, API шлюз требуется зарегистрировать в Keycloak. Для этого требуется:

- Войти в интерфейс администрирования.
- Выбрать нужный Realm.
- Перейти в раздел Clients.
- Нажать кнопку *Create client*.
- В разделе *General Settings* заполнить поля:
  - Client type - OpenID Connect;
  - Client ID - идентификатор клиента, например, *ngcp-gw-api-client*.
- В разделе *Capability Config* включить галочку Client Authentication.
- В разделе *Login Settings* заполнить поля:
  - Root URL - базовый адрес API шлюза, например, *https://api-gw.ngcp.local*;
  - Home URL - домашняя страница, например, */index*
  - Valid redirect URIs - допустимые адреса перенаправления для передачи кода аутентификации, например, */login/oauth2/code/*.
  - Valid post logout redirect URIs - допустимые адреса перенаправления, после выполнения Logout.

После создания клиента, пароль можно получить с вкладки *Credentials* в окне настроек клиента.

3.2.6.4 Настройка Keycloak Scopes

Spring Security в процессе аутентификации производит запрос к ресурсу *UserInfo*. Keycloak запрещает доступ к этому ресурсу для клиентов, у которых не задан Scope *openid*. Изначально, такой Scope отсутствует в списке доступных в Keycloak. Требуется создать его вручную. Для этого нужно:

- Войти в интерфейс администрирования.
- Выбрать нужный Realm.
- Перейти в раздел *Client scopes*.
- Нажать кнопку *Create client scope*.
- Заполнить поле *Name = openid* и сохранить Scope.

Далее, нужно назначить клиенту Scope *openid*. Для этого нужно:

- Перейти в раздел *Clients*.
- Открыть на редактирование нужный клиент.
- Перейти на вкладку *Client scopes*.
- Назначить Scope *openid*, при помощи кнопки *Add client scope*.

3.2.6.5 Состав JWT Claims

В результате аутентификации Keycloak передает шлюзу JWT, который можно использовать для авторизации доступа к ресурсам системы, например, операциям. Ниже приведен пример раздела данных (claims) из JWT.

JWT Claims


```
{
  "exp": 1678422267,
  "iat": 1678422207,
  "auth_time": 1678421829,
  "jti": "d48e7ff6-d142-4c0b-9ec6-dc007a78233b",
  "iss": "http://localhost:8980/realms/master",
  "aud": [
    "account",
    "ngcp-api-gw-client"
  ],
  "sub": "048c2cb8-f31f-4c9a-8643-ffeb4b6f1fac",
```

```
 "typ": "Bearer",
 "azp": "ngcp-api-gw-client",
 "nonce": "ZRH4T-jAHOQJBbKSThwtzy0fZrrRVVr5D2AP37WTo3g",
 "session_state": "e458c814-4b40-4ec1-bf6d-ea517fd9e69c",
 "acr": "0",
 "allowed-origins": [
  ""
 ],
 "realm_access": {
  "roles": [
   "create-realm",
   "default-roles-master",
   "offline_access",
   "admin",
   "uma_authorization"
  ]
 },
 "resource_access": {
  "account": {
   "roles": [
    "manage-account",
    "manage-account-links",
    "view-profile"
   ]
  },
  "ngcp-api-gw-client": {
   "roles": [
    "agcc/controller"
   ]
  }
 },
 "scope": "email openid profile",
 "sid": "e458c814-4b40-4ec1-bf6d-ea517fd9e69c",
 "email_verified": false,
 "preferred_username": "login1"
}
```


Copy

Ключевыми данными, которые нужны для авторизации, являются следующие поля:

- *sub* - идентификатор пользователя;
- *preferred_username* - логин пользователя;
- *resource_access.ngcp-api-gw-client.roles* - набор ролей пользователя для клиента *ngcp-api-gw-client*.

3.2.6.6 Управление сессиями

Spring Security отличает сессию приложения от сессии Keycloak. Если время действия JWT истекло, Spring Security автоматически обновит его при помощи refresh token'a. Однако, если при этом время HTTP сессии не истекло, данные аутентификации обновлены не будут. Данную особенность нужно иметь ввиду в случае использования объекта типа *OidcUser*. Объект этого типа можно извлечь из *SecurityContext'a*, или инжектировать при помощи аннотации *@AuthenticationPrincipal* в метод контроллера. Он содержит декодированные данные из JWT. Объект *OidcUser* не обновляется до истечения HTTP сессии, и если время жизни HTTP сессии дольше времени жизни JWT, возникает ситуация, при которой JWT мог уже обновиться, а объект *OidcUser* нет. Например, если у пользователя изменился набор ролей, то в объекте *OidcUser* эти изменения не отразятся до устаревания HTTP сессии, даже если время жизни JWT истекло. Проблему можно обойти, если вручную декодировать JWT, и получать данные из него напрямую.

3.2.7 Операции (operations)

API шлюз предоставляет клиентам доступ к бизнес операциям. Операция (*operation*), это конструкция, состоящая из набора предусловий (*preconditions*) и действий (*actions*).

Предусловия говорят клиенту о том, может ли пользователь выполнить операцию с объектом. Действия составляют содержательную часть операции. Предусловия и действия запускаются в указанном конфигурацией порядке, образуя **цепочку исполнения**. Перед запуском предусловий и действий формируется контекст, который содержит в себе комбинацию статических и динамических параметров, данные принципала, от имени которого запускается цепочка из предусловий или действий, набор сообщений пользователю, результат валидации и результат выполнения. Все изменения, внесенные в контекст предусловием или действием, видят следующие в цепочке предусловия и действия.
В описание операции входят:

- набор имен типов документов, для которых доступна данная операция;
- набор проектных ролей пользователей, которым данная операция доступна, при условии соблюдения всех предусловий.

Описания всех операций хранится в локальной конфигурации сервиса, файл *operations.yaml* в classpath. В описании, для динамического определения значений, допустимо использование SpEL выражений. Это, например, позволит указывать адреса HTTP предусловий без привязки к конкретному окружению.
Операция всегда работает в контексте одного или нескольких объектов.
Если операция должна работать в контексте нескольких объектов, такая операция является мультиобъектной. Мультиобъектные операции маркируются полем `multipleObjects: true`.
Если операция должна работать без объектов в контексте, такая операция называется необъектной. Необъектные операции маркируются полем `noObjects: true`. Если операция является необъектной, список в списке допустимых ролей указываются непроектные роли, тогда как в для объектных операций указываются проектные роли.
Операция не может быть одновременно и мультиобъектной и необъектной.
Пример описания мультибъектной операции

```yaml
---
name: operation1          # имя операции
multipleObjects: true     # является ли операция мультиобъектной
types:                    # список допустимых типов документов
  - Incoming Transmittal
  - Outgonig Transmittal
roles:                    # список допустимых проектных ролей
  - controller
  - contributor
preconditions:            # набор предусловий операции
  - name: precondition1   # имя предусловия
    args:                 # аргументы предусловия
      arg1: value1
      arg2: value2
  - name: precondition2
    args:
      arg1: value1
      arg2: value2
actions:                  # набор действий операции
  - name: action1
    validators:
      - name: validator1
        args:
          isTest: true
    args:                 # аргументы действия
      arg1: value1
      arg2: value2
```

Copy

Пример описания необъектной операции

```yaml
---
name: operation2
noObjects: true    # является ли операция необъектной
```

```
roles:          # для необъектных операций указываются непроектные роли
 - administrator
```

Copy

3.2.8 Статические и динамические параметры

Предусловия и действия обладают статическими и динамическими параметрами. Статические параметры задаются конфигурацией, динамические формируются сразу перед запуском. Перед стартом цепочки предусловий статические и динамические параметры сливаются. Динамические параметры имеют приоритет, и перетирают значения из статических параметров при совпадении имен.

3.2.9 Контекст

Контекст формируется перед запуском цепочки предусловий или действий.
Контекст предусловия хранит в себе следующие данные:

- данные принципала, от имени которого выполняется предусловие;
- набор объектов/документов, которые подлежат валидации;
- параметры запуска, статические и динамические;
- статус выполнения проваленных предусловия: *DISABLED*, *DENIED*;
- сообщения от проваленных предусловий.

Контекст действия хранит в себе следующие данные:

- данные принципала, от имени которого выполняется действие;
- набор идентификаторов объектов/документов, которые подлежат валидации;
- параметры запуска, статические и динамические;
- статус проваленных валидаций: *WARNING*, *ERROR*;
- сообщения проваленных валидации;
- сообщения от проваленных действий.

Если в ходе выполнения предусловия/действия в контекст вносятся изменения, эти изменения видны всем последующим предусловиям/действиям в цепочке.

3.2.10 Предусловия (preconditions)

3.2.10.1 Общие сведения

Предусловия запускаются клиентом для определения доступности той или иной операции. Предусловия не изменяют состояние системы, поэтому могут запускаться без использования блокировок и транзакций.
Предусловия не имеют собственного состояния и работают только на основе принципала и набора объектов которые входят в состав контекста. Благодаря этому все предусловия можно запустить из сервиса в рамках одной транзакции. Например, сервис начинает транзакцию, поднимает нужные объекты, передает их в цепочку предусловий, если предусловия завершились нормально, продолжает свою работу.
Если хотя-бы одно предусловие провалилось (статус выполнения *DENIED*), цепочка выполнения предусловий завершается, статус и сообщения возвращаются клиенту. Элементы интерфейса, отвечающие за данное действие, не отображается.
Если после выполнения всех предусловий среди статусов имеется хотя-бы один *DISABLED*, элемент интерфейса, отвечающий за выполнения действия отображается неактивным, а в подсказку выводятся сообщения которые соответствуют статусам *DISABLED*.
Клиент может предусмотреть режим отладки, который позволит отображать *DENIED* элементы с сообщениями, соответствующими статусу *DENIED*.
Предусловие обладает собственным набором параметров по-умолчанию. Параметры заданные на уровне описания операции, или переданные клиентом при запуске перекрывают значения параметров по-умолчанию.
Описание всех предусловий хранится в сервисе *DWS-03. Управление конфигурациями*.

3.2.11 Запуск HTTP предусловий

В описании предусловия фигурирует URL ссылка на предусловие *location*. Предусловия запускаются при помощи HTTP POST вызова к сервису по адресу, указанному в поле *location*. Контекст преобразуется в JSON объект и передается в теле запроса. После выполнения операции, обновленный контекст также возвращается в виде JSON объекта.
Пример описания предусловия

```
---
name: precondition1
location: http://${preconditionServiceName}/precondition/preconditionName
args:
  - ...
```

Copy

3.2.12 Встроенное предусловие *rolePrecondition*

Перед запуском цепочки предусловий, для необъектных операций производится проверка на доступность действия по совокупности глобальных ролей принципала (см. Определение доступности по глобальным ролям). Данное предусловие неявно представлено во всех конфигурациях.

3.2.13 Встроенное предусловие *projectRolePrecondition*

Перед запуском цепочки предусловий, для объектных операций производится проверка на доступность действия по совокупности проектных ролей принципала (см. Определение доступности по проектным ролям). Данное предусловие неявно представлено во всех конфигурациях.

3.2.14 Встроенное предусловие *doctypePrecondition*

Перед запуском цепочки предусловий, производится проверка на доступность действия по типам документов, которые являются частью контекста операции (см. Определение доступности по типам документов). Данное предусловие неявно представлено во всех конфигурациях и не подлежит настройке.

3.2.15 Встроенное предусловие *isAuthorizedPrecondition*

Похоже что ACS от ДД будет предоставлять функционал, подобный Cerbos. Вероятно будем использовать его. Как только будет понятно что умеет ACS, нужно будет переписать данный раздел.

Перед запуском цепочки предусловий, всегда производится обращение к сервису Cerbos для авторизации операции. Данные принципала и набор объектов передаются сервису, который принимает решение о доступности, или недоступности операции. Данное предусловие неявно представлено во всех конфигурациях и не подлежит настройке.

3.2.16 Действия (actions)

3.2.16.1 Общие сведения

Действия это полезная нагрузка операции. Действия выполняются в соответствии с последовательностью, заданной описанием операции (см. Операции).
Перед выполнением цепочки действий, производится запуск всей цепочки предусловий в рамках единой транзакции, включая встроенное предусловие *isAuthorizedPrecondition* (см. Встроенное предусловие *isAuthorizedPrecondition*).
Первое действие в цепочке ответственно за запуск цепочки предусловий. Остальные действия из цепочки не запускают предусловий.
Отдельный слой валидации операций сознательно опущен, т.к. с точки зрения ACID валидация и выполнение действия не отделимы друг от друга. Поэтому, именно действие ответственно за:

- загрузку объектов, которые подлежат изменению;
- расстановку блокировок на объекты;
- начало и фиксацию транзакции;

- валидацию входящих параметров и состояния объектов;
- выполнение бизнес логики.

Чтобы избежать межсервисной транзакционности, требуется реализовывать операцию:
- одним действием;
- набором идемпотентных действий, невыполнение которых некритично.

В состав каждого действия входит набор валидаторов, и их аргументов.

```
validators:
    - name: validator1
      args:
        isTest: true
```

Copy

Валидаторы исполняются после precondition но перед основным action
Действия бывают 2-х видов:
- синхронные;
- асинхронные.

Если синхронное действие завершило стадию валидации со статусом *WARNING*, оно может, в зависимости от параметров запуска, продолжить выполнение, или завершить всю цепочку с откатом транзакции. Клиент должен предоставлять пользователю возможность пропустить такую валидацию. Типичный сценарий использования следующий:
- пользователь запускает операцию на исполнение;
- одно из действий завершает стадию валидации со статусом *WARNING*;
- действие откатывает транзакцию и передает клиенту сообщение о провале валидации;
- пользователь видит сообщение о том что не удалось завершить операцию и описание того, почему не удалось;
- пользователь принимает решение о том, что проблема не серьезная и ставит галочку *игнорировать предупреждение*;
- пользователь повторно запускает операцию;
- т.к. пользователь явно сообщил о том что валидацию можно игнорировать, действие, после завершения валидации, продолжает выполнение.

Пользователь явно сообщает какое предупреждение можно игнорировать. Если действие реализует различные проверки, и со статусом *WARNING* завершится не та проверка, которую можно игнорировать, действие не может игнорировать этот статус, и завершается с откатом транзакции.
Асинхронные действия также изменяют состояние системы, но по своей природе не могут быть включены в одну транзакцию, т.к. время начала выполнения и завершения действия не определено, в отличии от синхронных действий. Для синхронного действия мы ожидаем что оно запустится *сразу* после вызова и завершится в *разумный срок* или сообщит о том что не удалось завершиться в *разумный срок*. *Сразу* и *разумный срок* -- достаточные определения в контексте системы.
Асинхронные действия всегда игнорируют статус *WARNING* на стадии валидации и продолжают исполнение.

3.2.16.2 Синхронные действия

Синхронные действия запускаются при помощи HTTP POST запроса. Контекст преобразуется в JSON объект и передается в теле запроса. После выполнения операции, обновленный контекст также возвращается в виде JSON объекта.
Пример описания синхронного удаленного действия

```
---
name: action1
location: http://${actionServiceName}/action/actionName
args:
  - ...
```

Copy

3.2.16.3 Асинхронные действия

Асинхронные действия запускаются путем публикации в обменнике сообщения, которое является триггером для запуска асинхронной операции (см. Сервис асинхронных операций). Измененный контекст не передается следующему действию по цепочке. Передача измененного контекста следующему действию невозможно, вследствие неопределенности времени запуска и завершения асинхронных действий. Следующее в цепочке действие получает контекст от последнего выполнившегося синхронного действия.
В описание действия фигурируют имя обменника `exchange` и маршрутный ключ `routingKey`.
Пример описания асинхронного действия

```
---
name: action2
exchange: ""
routingKey: ${asynOpServiceQueueName}
args:
  - ...
```

Copy

3.2.17 Реестр операций *OperationsRegistry*

Реестр операций осуществляет хранение и доступ к операциям, предусловиям и действиям.

3.2.18 Библиотека для реализации действий и предусловий

Требуется разработать разделяемую библиотеку которая определит интерфейсы для реализации предусловий и действий, а также определит структуру контекста и методы доступа к нему.
Библиотека должна реализовывать общий функционал по приему запросов на выполнение предусловий и действий.
Диаграмма классов отображает структуру разделяемой библиотеки (модуль gateway-common).

3.2.19 Реестр локализаций атрибутов, групп и сообщений

3.2.19.1 Общие сведения

API шлюз предоставляет клиентам данные по локализации атрибутов различных типов посредством специального синхронного API вызова. Клиент перед началом работы должен запросить эти данные и сохранить их у себя на время всей сессии.

3.2.19.2 Конфигурационный файл

Данные для реестра локализаций атрибутов хранятся в конфигурационном файле или в сервисе данных, в зависимости от того как сконфигурирован шлюз (см. INgcpConfigService). Вся локализация заносится в один `.properties` файл в кодировке UTF-8, каждая строка которого имеет следующий формат:

```
<type>.<field>.<languageTag> = <text>
```

Copy

Например:

```
project.code.en = Project code
project.code.ru = Код проекта
```

Copy

Здесь:

- *type* - условное имя типа, например *project* представляет проект, а *projectType* тип проекта;
- *field* - поле, локализованное имя которого задает конфигурационный файл.
- *languageTag* - код локали, например *en*, *ru*, *kk-KZ*.

Такой формат удобен потому что:

- *properties* простой формат и хорошо представляет несложные структуры;

- избыточность описания компенсируется тем, что в любой строке понятно какой тип и какую локаль редактируем, благодаря чему можно иметь один большой конфигурационный файл без последствий при редактировании;
- описание всех локализаций в одном файле позволяет не забывать добавлять локализацию для всех поддерживаемых языков;
- хранение всей локализации в одном конфигурационном файле позволяет использовать *INgcpConfigService* для хранения конфигурации в сервисе данных.

3.2.19.3 Загрузчик реестра локализации атрибутов

Загрузчик реестра производит считывание конфигурации, разбор и валидацию. На этапе валидации требуется предусмотреть не только корректность заполнения каждой строки реестра, но и то, что все указанные поля имеют текст для всех поддерживаемых локалей. Набор всех поддерживаемых локалей требуется сформировать из содержимого реестра. Если хотя бы одно поле имеет текст для того или иного *language tag*, то считается что соответствующая локаль поддерживается системой. Если валидация не прошла по одному из указанных критериев, сервис должен вывести в лог сообщение об ошибке и не менять текущую конфигурацию. Если ошибка возникла при старте сервиса, конфигурация считается пустой.

3.2.19.4 Модель данных для хранения и отправки данных реестра локализации

После загрузки реестра локализации атрибутов, реестр хранится в памяти и отправляется клиенту в структуре следующего вида:

```
{
  "project": {
    "objectName": {
      "en": "Project Name",
      "ru": "Имя проекта"
    },
    "code": {
      "en": "Project Code",
      "ru": "Код проекта"
    },
    ...
  },
  "transmittal": {
    "objectName": {
      "en": "Transmittal Name",
      "ru": "Имя трансмиттала"
    },
    "objectNumber": {
      "en": "Transmittal Number",
      "ru": "Номер трансмиттала"
    },
    ...
  },
  ...
}
```

Copy

3.2.20 Схема системных типов

API шлюз предоставляет клиентам частичную схему данных, необходимую для отображения документов, процессов, трансмитталов и т.п. Если тот или иной атрибут объекта не представлен в схеме, то этот атрибут считается специальным и не предназначен для отображения, или должен отображаться/редактироваться специальным образом.
В результате запроса, клиент получает ответ в виде:

```
{
   "document": {
      "objectName": "string",
      "objectNumber": "string",
```

```
      ...
  },
  ...
}
```

Copy

Каждое поле в результате, это условное имя типа системного объекта. Соответствие условного имени и типа:

- `project` - NgcpProject;
- `documentType` - NgcpDocumentType;
- `document` - NgcpDocument;
- `comment` - NgcpComment;
- `transmittalType` - NgcpTransmittalType;
- `transmittal` - NgcpTransmittal;
- `processType` - NgcpProcessType;
- `process` - NgcpProcess;
- `task` - NgcpTask;
- `bulkLoadSheet` - NgcpBulkLoadSheet.

Каждое значение, это соответствие атрибута и типа этого атрибута. Набор допустимых типов атрибутов:

- `string` - простая строка, подставляется как есть;
- `float` - число с плавающей запятой, может быть представлено в формате десятичной дроби, целого числа или экспоненциальном формате, например "13", "13.159", "1.3159e1";
- `integer` - целое число, например "1799";
- `boolean` - логическое значение, например, "false"
- `time` - время, например "16:06:20.481564600";
- `date` - дата, например "2023-06-19";
- `datetime` - дата и время, например "2023-06-19T16:07:54.562619300";
- `instant` - момент времени, количество миллисекунд начиная с UNIX эпохи, например "1687173703371";
- `userId` - идентификатор пользователя системы;
- `projectId` - идентфификатор проекта;
- `documentId` - идентификатор документа;
- `documentTypeId` - идентификатор типа документов;
- `documentTemplateId` - идентификатор шаблона документа.

Массивы и списки задаются постфиксом `[]`, например `string[]` - массив строк.
Далее приведен список специальных атрибутов, которые не попадают в схему и не передаются клиенту:

- для всех типов:
  - additionalAttachments;
  - attrsConfig;
  - content;
- для трансмитталов
  - coverSheet;
  - coverSheetTemplate;
  - commentSheet;
  - commentSheetTemplate;
- для процессов:
  - distribution;
  - deadlines.

Остальные атрибуты извлекаются из типов при помощи рефлексии и отдаются клиенту.
Соответствие типов с датой/временем:

- `NgcpDateOnly` - date;
- `LocalWithTimezoneNgcpDate` - datetime;
- `TimestampNgcpDate` - instant.

Поля, являющиеся ссылками на другие объекты имеют соответствующий тип вида xxxId. Например, все атрибуты author имеют тип userId.

3.2.20.1.1 Пример схемы системных типов

api_typeschema.json

3.3 Rest API

3.3.1 Общие сведения

API шлюз осуществляет синхронное взаимодействие с клиентами посредством протокола HTTP. Далее в таблице приведены запросы которые обрабатывает API шлюз.
HTTP Запросы

Путь	Описание
POST /api/operation/available	Получить список операций, доступных для указанных объектов.
POST /api/operation/{name}	Выполнить операцию с именем *name*.
GET /api/operations	Получить описание операций и их парамтеров (их предусловий, действий, типов, ролей и прочее)
GET /api/locregistry	Получить реестр локализации аттрибутов, групп и сообщений.
GET /api/typeschema	Получить схему системных типов.

3.3.2 Список операций

Запрос на получения списка операций: POST /api/operation/available.
Тело запроса - список объектов типа UUID.
Результат:

HTTP код	Описание	Результат
200	Запрос выполнен успешно	JSON массив операций, доступных/видимых для объектов в соответствии со схемой ответа.
500	Внутренняя ошибка сервиса	ProblemDetail

Обработчик делегирует запрос сервису управления операциями (см. Определение доступности операций).

3.3.3 Выполнить операцию

Запрос на выполнение действий операции: POST /api/operation/{name}.
В тело запроса передается JSON объект, в соответствии со схемой
Результат:

HTTP код	Описание	Результат
200	Запрос выполнен успешно	Ответ в соответствии со схемой.
404	Указанная операция не найдена	-
415	Если клиент ожидает получить JSON, но сервер хочет вернуть содержимое документа	ProblemDetail
500	Внутренняя ошибка сервиса	ProblemDetail

Обработчик делегирует запрос сервису управления операциями (см. Запуск действий).

3.3.4 Получить описание операций

Запрос на получение описания операций: `GET /api/operations`.
Результат:

HTTP код	Описание	Результат
200	Запрос выполнен успешно	JSON описание операций Пример описания операций
500	Внутренняя ошибка сервиса	ProblemDetail

3.3.4.1 Пример описания операций

api_operations.json

3.3.5 Получить реестр локализации атрибутов, групп и сообщений

Запрос на получение реестра локализации: `GET /api/locregistry`.
Результат:

HTTP код	Описание	Результат
200	Запрос выполнен успешно	Реестр локализации атрбутов, см. Модель данных для хранения и отпавки данных реестра локализации
500	Внутренняя ошибка сервиса	ProblemDetail

3.3.6 Получить схему системных типов

Запрос на получение схемы системных типов: `GET /api/typeschema`.
Результат:

HTTP код	Описание	Результат
200	Запрос выполнен успешно	Схема системных типов, см. Схема системных типов
500	Внутренняя ошибка сервиса	ProblemDetail

3.4 Сервис управления операциями *OperationsService*

3.4.1 Общие сведения

Объект *OperationsService* основной объект управления операциями. Он:

- производит загрузку описания всех операций;
- определяет доступность операций для принципала и объектов;
- делегирует запуск действий соответствующим сервисам.

3.4.2 Инициализация *OperationsService*

В ходе инициализации производится подключение к сервису конфигурации и загрузка описания операций. Описание операций используется для инициализации реестра операций, ссылка на который хранится в объекте *OperationsService*.

3.4.3 Загрузка описания операций

Описание операций загружается из YAML файла *operations.yaml*, расположенного в *classpath:/operations.yaml*. Пример конфигурации см. ниже:

Пример конфигурации

```
operations:
  - ... # см. пример конфигурации операций в разделе "Операции"
httpPreconditions:
  - ... # см. пример конфигурации действий в разделе "Предусловия"
httpActions:
  - ... # см. пример конфигурации действий в разделе "Синхронные действия"
ampqActions:
  - ... # см. пример конфигурации действий в разделе "Асинхронные действия"
```

Copy

3.4.4 Определение доступности операций

3.4.4.1 Общие сведения

Входящие аргументы метода: набор идентификаторов документов `objectIds: List<UUID>`.
Метод запрашивает документы по набору идентификаторов из сервиса управления метаданными.
Если операция является мультиобъектной, то все последующие проверки выполняются относительно всех объектов сразу. Если операция не мультиобъектная, то проверки выполняются для каждого объекта индивидуально.
Метод выполняет следующие действия для каждой операции из реестра:

- определяет доступность по типам документов;
- определяет доступность по глобальным ролям если операция необъектная;
- определяет доступность по проектным ролям, если операция объектная;
- определяет доступность по цепочке предусловий.

Если хотя-бы одна проверка провалилась со статусом *DENIED*, операция считается недоступной.
Метод формирует набор имен операций, проверка по которым не провалилась, или провалилась со статусом *DENIED*. Если проверка завершилась со статусом *DISABLED*, сообщения с результатами проверок "цепляются" к имени операции.

Метод возвращает структуру, которая соответствует json схеме Список доступных/видимых операций.

3.4.4.2 Определение доступности по типам документов

Входящие аргументы метода:
- набор объектов `documents: List<INgcpObject>`;
- дескриптор операции `operation: OperationDef`.

Если `documentIds.isEmpty()`:
- если `operation.types.isEmpty()`, операция считается доступной;
- иначе операция считается недоступной.

Для каждой строки `type` из `operation.types` и документа `document` из `documents:
- `type` считается регулярным выражением;
- если `document.docType.objectName` не подходит под регулярное выражение `type`, операция считается недоступной.

Если операция недоступна, вернуть *Map* вида;

```
{
  "DoctypesPrecondition": [{
    "status": "DENIED",
    "message": {
      "key": "MSG_DENIED_BY_DOCTYPE"
    }
  }]
}
```

Copy

Иначе вернуть пустой MultiValueMap.

3.4.4.3 Определение доступности по глобальным ролям

Входящие аргументы метода:
- дескриптор принципала `principal: Principal`;
- дескриптор операции `operation: OperationDef`.

Список ролей принципала `principalRoles: Set<String>` извлечь из `principal`.
Список ролей операции `operationRoles: List<String> := operation.roles`.
Если `roles.isEmpty()`, операция считается доступной.
Для каждой роли операции `operationRole` из `operationRoles`:
- если `principalRoles.contains(operationRole)`, операция считается доступной.

3.4.4.4 Определение доступности по проектным ролям

Входящие аргументы метода:
- дескриптор принципала `principal: Principal`;
- набор объектов `documents: List<INgcpObject>`;
- дескриптор операции `operation: OperationDef`.

Список ролей принципала `principalRoles: Set<String>` извлечь из `principal`.
Список ролей операции `operationRoles: List<String> := operation.roles`.
Если `roles.isEmpty()`, операция считается доступной.
Если `documents.isEmpty()`, операция считается недоступной.
Для каждого документа `document` из `documents`:
- Проект документа `docProject := document.project.objectName`;
- Для каждой роли операции `operationRole` из `operationRoles`:
    - если `principalRoles.contains(docProject + "/" + operationRole)`, операция считается доступной.

Если операция недоступна, вернуть *Map* вида;

```
{
  "DoctypesPrecondition": [{
    "status": "DENIED",
```

```
  "message": {
    "key": "MSG_DENIED_BY_PROJECT_ROLE"
  }
 }]
}
```

Copy

Иначе вернуть пустой Map.

3.4.4.5 Определение доступности по правилам Cerbos

Входящие аргументы метода:

- дескриптор принципала `principal: Principal`;
- набор объектов `documents: List<INgcpObject>`;
- дескриптор операции `operation: OperationDef`.

На основании входных параметров формируется запрос к сервису Cerbos. В запрос включаются следующие данные:

- имя пользователя;
- набор ролей пользователя;
- имя операции;
- сериализованные объекты.

Исходя из этих данных и политик, Cerbos принимает решение о доступности или недоступности операции. Детальную информацию об API и политиках доступа см. в https://docs.cerbos.dev/cerbos/latest/index.html.

3.4.4.6 Определение доступности по цепочке предусловий

Входящие аргументы метода:

- дескриптор принципала `principal: Principal`;
- набор объектов `documents: List<INgcpObject>`;
- дескриптор операции `operation: OperationDef`;
- реестр операций `operationsRegistry: OperationsRegistry`.

Сформировать контекст предусловия `context: PreconditionContext`.
Для каждой ссылки на предусловие `preconditionRef` из `operation.preconditions`:

- найти предусловие `precondition := operationsRegistry.findPrecondition(preconditionRef.name)`;
- применить параметры из `preconditionRef` к контексту `context`, параметры не должны перекрывать существующие в контексте;
- применить параметры предусловия `precondition` к контексту `context`, параметры не должны перекрывать существующие в контексте;
- выполнить предусловие и обновить контекст `context := precondition.execute(context)`;
- если `context.messages.get(precondition.name)` содержит сообщение со статусом *DENIED*, выполнение предусловий завершается.

Вернуть `context.messages`.

3.4.5 Запуск действий

Входящие аргументы метода:

- дескриптор принципала `principal: OidcUser`;
- набор идентификаторов объектов `documents: List<MetaRef<?>>`;
- набор игнорируемых предупреждений `ignoreWarnings: MultiValueMap<String, String>`;
- дескриптор операции `operation: NgcpOperationDef`;
- реестр операций `operationsRegistry: OperationsRegistry`.

Сформировать контекст действия `context: ActionContext`.
Для каждой ссылки на действие `actionRef` из `operation.actions`:

- найти действие `action := operationsRegistry.findAction(actionRef.name)`;
- применить параметры из `actionRef` к контексту `context`, параметры не должны перекрывать существующие в контексте;

- применить параметры действия `action` к контексту `context`, параметры не должны перекрывать существующие в контексте;
- выполнить действие и обновить контекст `context := action.execute(context, ignoreWarnings.get(action.name))`;
- если `context.validationMessages` содержит сообщения, выполнение действий завершается;
- если `context.actionMessages` содержит сообщения, выполнение действий завершается;

Вернуть комбинацию из `context.validationMessages` и `context.actionMessages`. Считается что все `context.actionMessages` имеют статус *ERROR*.

3.5 Класс *PreconditionDef*

Предусловие, которое выполняется посредством HTTP POST запроса по указанному адресу. В качестве тела запроса передается контекст, сериализованный в соответствии с JSON схемой. В ответ возвращается обновленный контекст. Детали в приложении HTTP вызов предусловия.

3.6 Класс *HttpActionDef*

Действие, которое выполняется посредством HTTP POST запроса по указанному адресу. В качестве тела запроса передается контекст, сериализованный в соответствии с JSON схемой. Детали в приложении HTTP вызов действия.

3.7 Класс *AmpqActionDef*

Действие, которое выполняется посредством публикации запроса на выполнение асинхронной операции. Детали в приложении AMPQ вызов действия.
Для настройки подключения к очереди используются стандартные property SpringBoot с ключом вида `spring.rabbitmq.*`, например:

```
spring.rabbitmq.host = 192.168.0.1
spring.rabbitmq.port = 5672
spring.rabbitmq.username = guest1
spring.rabbitmq.password = guest1

spring.rabbitmq.listener.type = direct
spring.rabbitmq.listener.direct.default-requeue-rejected = false
```

Copy

4 Приложения

4.1 HTTP вызов предусловия

Вызов HTTP предусловий осуществляется посредством HTTP запроса вида `POST /path/to/precondition/{precondition}`. В качестве тела запроса передается контекст, сериализованный в JSON в соответствии со схемой. Результатом является обновленный контекст, который возвращает сторонний исполнитель в ответ на HTTP запрос.
Результат HTTP запроса:

HTTP код	Описание	Результат
200	Запрос выполнен успешно	Обновленный контекст предусловия, сериализованный в JSON в соответствии со схемой.
500	Внутренняя ошибка сервиса	

4.2 HTTP вызов действия

Вызов HTTP действий осуществляется посредством HTTP запроса вида `POST /path/to/action/{action}`. В качестве тела запроса передается контекст, сериализованный в JSON в соответствии со схемой. Результатом является обновленный контекст, который возвращает сторонний исполнитель в ответ на HTTP запрос.
Результат HTTP запроса:

HTTP код	Описание	Результат
200	Запрос выполнен успешно	Обновленный контекст действия, сериализованный в JSON в соответствии со схемой.
500	Внутренняя ошибка сервиса	

4.3 AMPQ вызов действия

Вызов асинхронного AMPQ действия производится посредством публикации запроса на выполнение асинхронной операции. Формат запроса описан в CP-02. Сервис обработки асинхронных операций. Имя обменника и маршрутный ключ зависят от того кто будет выполнять операцию. Сразу после публикации, управление передается вызывающему методу.

4.4 JSON схемы

4.4.1 Список доступных/видимых операций

```
{
  "type": "array",
  "items": {
    "type": "object",
    "required": [
      "objectIds",
      "operations"
    ],
    "properties": {
      "objectIds": {
        "type": "array",
        "items": {
          "type": "string",
          "format": "uuid"
        }
      },
      "operations": {
        "type": "array",
        "items": {
          "type": "object",
          "required": [
            "name",
            "status"
          ],
          "properties": {
            "name": {
              "type": "string"
            },
            "status": {
              "enum": [
```

```
                        "AVAILABLE",
                        "DISABLED"
                    ]
                },
                "message": {
                    "$ref": "https://wiki.digitalepoch.ru/e/ru/cp++/archit/common/Internationalization?message-json"
                }
            }
        }
    }
  },
  "examples": [
    [
      {
        "objectIds": ["205c9acb-46bf-42b5-acb5-a5370fe895d9"],
        "operations": [
          {
            "name": "Operation1",
            "status": "AVAILABLE"
          },
          {
            "name": "Operation2",
            "status": "DISABLED",
            "message": {
              "key": "MSG_OPERATION2_DISABLED"
            }
          }
        ]
      },
      {
        "objectIds": ["37c344d3-b283-49e9-81e1-a90fd0f94e60"],
        "operations": [
          {
            "name": "Operation1",
            "status": "AVAILABLE"
          },
          {
            "name": "Operation3",
            "status": "AVAILABLE"
          }
        ]
      }
    ],
    [
      {
        "objectIds": ["205c9acb-46bf-42b5-acb5-a5370fe895d9", "37c344d3-b283-49e9-81e1-a90fd0f94e60"],
        "operations": [
          {
            "name": "MultiobjectOperation1",
            "status": "AVAILABLE"
          },
          {
            "name": "MultiobjectOperation2",
            "status": "DISABLED",
            "message": {
              "key": "MSG_MULTIOBJECT_OPERATION2_DISABLED"
            }
          }
        ]
      }
    ]
  ]
}
```

Copy

4.4.2 Тело запроса на выполнение операции


```
{
    "$schema": "https://json-schema.org/draft/2019-09/schema",
    "$id": "https://wiki.digitalepoch.ru/ru/cp++/archit/CP-XX_ApiGatewayService/TWP?post-operation-json-request",
    "type": "object",
    "required": [
        "objectIds"
    ],
    "properties": {
        "objectIds": {
            "type": "array",
            "items": {
                "type": "object",
                "required": [
                    "schemaId",
                    "objectId"
                ],
                "properties": {
                    "schemaId": {
                        "type": "string",
                        "format": "uuid"
                    },
                    "objectId": {
                        "type": "string",
                        "format": "uuid"
                    }
                }
            }
        },
        "args": {
            "type": "object"
        },
        "ignoreWarnings": {
            "type": "array",
            "items": {
                "type": "object",
                "patternProperties": {
                    ".*": {
                        "type": "array",
                        "items": {
                            "type": "string"
                        }
                    }
                }
            }
        }
    },
    "examples": [
        {
            "objectIds": [
                {
                    "schemaId": "6008abe3-2287-4367-b85d-c6e757d450cb",
                    "objectId": "39f1000f-02b0-45ae-a0ac-9c439a79e0c7"
                }
            ],
            "args": {
                "arg": "value"
            },
            "ignoreWarnings": [
                {
                    "TheAction": [
                        "MSG_ACTION_WARNING_1",
                        "MSG_ACTION_WARNING_2"
                    ]
                }
            ]
```

```
      }
    ]
}
```

Copy

4.4.3 Результат выполнения операции

Пример ответа

```
[
  {
    "objectIds": [
      "b3b62b89-bbde-4f4c-903a-d05a4aa50d3d",
      "ee5bae6f-9eb8-4656-bc62-d01238198d4d"
    ],
    "value": "FAILED",
    "preconditionMessages": {
      "Precondition1": [
        {
          "status": "DISABLED",
          "message": {
            "messageKey": "MSG_PRECONDITION1_DISABLED"
          }
        }
      ]
    },
    "validationMessages": {
      "Action1": [
        {
          "status": "WARNING",
          "message": {
            "messageKey": "MSG_ACTION1_VALIDATION_WARNING"
          }
        }
      ]
    },
    "actionMessages": {}
  }
]
```

Copy

4.4.4 Сериализованное описание предусловия

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://wiki.digitalepoch.ru/ru/cp++/archit/CP-XX_ApiGatewayService/TWP?precondition-def-json",
  "type": "object",
  "required": [
    "name",
    "location"
  ],
  "properties": {
    "name": {
      "type": "string",
      "minLength": 1
    },
    "location": {
      "type": "string",
      "format": "uri"
    },
    "args": {
      "type": "object"
    }
  },
  "examples": [
    {
```

```
            "name": "theName",
            "location": "http://the.precondition/location",
            "args": {
                "arg1": "value1",
                "arg2": "value2"
            }
        }
    ]
}
```

Copy

4.4.5 Сериализованный контекст предусловия

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "https://wiki.digitalepoch.ru/ru/cp++/archit/CP-XX_ApiGatewayService/TWP?precondition-context-json",
    "type": "object",
    "required": [
        "principal",
        "objects",
        "args",
        "messages"
    ],
    "properties": {
        "principal": {
            "type": "string",
            "format": "jwt"
        },
        "objects": {
            "type": "array"
        },
        "args": {
            "type": "object"
        },
        "messages": {
            "type": "object",
            "patternProperties": {
                ".*": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "status": {
                                "enum": ["DISABLED", "DENIED"]
                            },
                            "message": {
                                "$ref": "/archit/common/Internationalization?message-json"
                            }
                        }
                    }
                }
            }
        }
    },
    "examples": [
        {
            "principal":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",
            "objects": [
                {
                    "id": "6d533a95-a8a4-428c-a191-41f0b8b1074d",
                    "project": "ngcp",
                    "doctype": "Design Document"
                }
            ],
```

```
        "args": {
          "arg1": "value1",
          "arg2": "value2"
        },
        "messages": {
          "Precondition1": [
            {
              "status": "DISABLED",
              "message": {
                "key": "MSG_PRECONDITION_DISABLED"
              }
            }
          ],
          "Precondition2": [
            {
              "status": "DENIED",
              "message": {
                "key": "MSG_PRECONDITION_DENIED"
              }
            }
          ]
        }
      }
    ]
}
```

Copy

4.4.6 Сериализованный контекст действия

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://wiki.digitalepoch.ru/ru/cp++/archit/CP-XX_ApiGatewayService/TWP?action-context-json",
  "type": "object",
  "required": [
    "principal",
    "objectIds",
    "preconditions",
    "preconditionArgs",
    "actionArgs",
    "preconditionMessages",
    "validationMessages",
    "actionMessages"
  ],
  "properties": {
    "principal": {
      "type": "string",
      "format": "jwt"
    },
    "objectIds": {
      "type": "array",
      "items": {
        "type": "string",
        "format": "uuid"
      }
    },
    "preconditions": {
      "type": "array",
      "items": {
        "$ref": "/TWP?precondition-def-json"
      }
    },
    "preconditionArgs": {
      "type": "object",
      "patternProperties": {
        ".*": {
          "type": "object"
```

```
            }
        }
    },
    "actionArgs": {
        "type": "object"
    },
    "preconditionMessages": {
        "type": "object",
        "patternProperties": {
            ".*": {
                "type": "array",
                "items": {
                    "type": "object",
                    "required": [
                        "status",
                        "message"
                    ],
                    "properties": {
                        "status": {
                            "enum": ["DISABLED", "DENIED"]
                        },
                        "message": {
                            "$ref": "/archit/common/Internationalization?message-json"
                        }
                    }
                }
            }
        }
    },
    "validationMessages": {
        "type": "object",
        "patternProperties": {
            ".*": {
                "type": "array",
                "items": {
                    "type": "object",
                    "required": [
                        "status",
                        "message"
                    ],
                    "properties": {
                        "status": {
                            "enum": ["WARNING", "ERROR"]
                        },
                        "message": {
                            "$ref": "/archit/common/Internationalization?message-json"
                        }
                    }
                }
            }
        }
    },
    "actionMessages": {
        "type": "object",
        "patternProperties": {
            ".*": {
                "type": "array",
                "items": {
                    "$ref": "/archit/common/Internationalization?message-json"
                }
            }
        }
    },
    "result": {
        "type": "object"
    }
```

```
    },
    "examples": [{
        "principal":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",
        "objectIds": [
            "a1c70c12-467e-42f1-9d3d-931acab8a399",
            "7b9a0ec2-d2b0-417d-a61b-cca3e33c3b61"
        ],
        "preconditions": [{
            "name": "theName",
            "location": "http://the.precondition/location",
            "args": {
                "arg1": "value1",
                "arg2": "value2"
            }
        }],
        "preconditionArgs": {
            "theName": {
                "arg1": "value1",
                "arg2": "value2"
            }
        },
        "actionArgs": {
            "arg1": "value1",
            "arg2": "value2"
        },
        "preconditionMessages": {
            "Precondition1": [{
                "status": "DISABLED",
                "message": {
                    "key": "MSG_PRECONDITION_DISABLED"
                }
            }]
        },
        "validationMessages": {
            "Action1": [{
                "status": "WARNING",
                "message": {
                    "key": "MSG_VALIDATION_WARNING"
                }
            }],
            "Action2": [{
                "status": "ERROR",
                "message": {
                    "key": "MSG_VALIDATION_ERROR"
                }
            }]
        },
        "actionMessages": {
            "Action1": [{
                "key": "MSG_ACTION_ERROR"
            }]
        },
        "result": {
            "schemaId": "cdd8a0c1-5f5e-42ee-b6fa-760e0e7d8db4",
            "objectId": "6c8c3aed-e010-4041-a01f-bd8632cbfe6e"
        }
    }]
}
```

Copy

5 Авторизация асинхронных операций

Каждая операция выполняется в отдельной сессии. Токен запрашивается непосредственно перед началом выполнения операции в рамках процесса инициализации сессии.
В зависимости от конфигурации (параметров запуска операции), операция может быть выполнена либо от имени самого сервиса (сервисной учётной записи), либо от имени пользователя (например, инициировавшего выполнение операции) через механизм имперсонации/повышения привилегий.
Каждый сервис - отдельный openid клиент

6 Приложение. API системы

6.1 Системный API

См. отдельный документ «Феникс. API System.pdf»

6.2 API Управление документами

См. отдельный документ «Феникс. API Управление документами.pdf»

6.3 API Управление задачами

См. отдельный документ «Феникс. API Управление задачами.pdf»

6.4 API Управление комментариями

См. отдельный документ «Феникс. API Управление комментариями.pdf»

6.5 API Управление проектами

См. отдельный документ «Феникс. API Управление проектами.pdf»

6.6 API Управление процессами

См. отдельный документ «Феникс. API Управление процессами.pdf»

6.7 API Управление РМЗ

См. отдельный документ «Феникс. API Управление РМЗ.pdf»

6.8 API Управление списками выбора

См. отдельный документ «Феникс. API Управление списками выбора.pdf»

6.9 API Управление трансмитталами

См. отдельный документ «Феникс. API Управление трансмитталами.pdf»