Параметры приложения - Общая информация

Параметры приложения

Оглавление

- Назначение
- Архитектура
- Параметры конфигурации
  - Основные
- Права и роли
- REST API
- Примеры

Параметры приложения — это фиксированные значения параметров системы, хранящиеся в БД и доступные программно через `AppSettingService`. Модуль подключается как библиотека и работает внутри процесса вашего приложения.

> **Модуль подключается как библиотека и работает внутри процесса вашего приложения.**

> **Быстрый старт** — см. раздел Подключение к проекту.

Назначение

С помощью параметров приложения можно изменять поведение системы под конкретные нужды. Реакция на параметр настраивается и связывается с определённым параметром программистом, после чего значение можно получить через пользовательский интерфейс или программно.

Тип значения параметров приложения может быть текстовым, числовым, логическим, временным, объектом (JSON) или массивом (JSON-список). Примеры параметров: текущий период работы, текущая дата работы, текущий план счетов, максимальное число полей и т.д.

В дополнение предусмотрены **три среза (уровня)** параметров. По умолчанию все три среза `GLOBAL` — глобальные, т.е. параметр по умолчанию является глобальным. Ограничений на наименование среза нет. Срезы с одинаковым наименованием рекомендуется задавать на одном уровне, например, первый срез для всех параметров = `USER` или `GLOBAL`.

Пример

Параметр `workHours` (рабочие часы):

- глобальный со значением 8 — все срезы = `GLOBAL`
- для сотрудника `PIOTR` со значением 4 — первый срез `USER` со значением `PIOTR`, остальные срезы `GLOBAL`
- для сотрудника `IVAN` со значением 6 — первый срез `USER` со значением `IVAN`, остальные срезы `GLOBAL`

Архитектура

Модуль состоит из следующих подмодулей:

Модуль	Назначение
`application-setting-service`	Бизнес-логика, DAO, кеширование
`application-setting-pojo`	DTO, enum'и, валидация
`application-setting-gate`	JAX-RS ресурсы (user + admin). При подключении автоматически регистрирует REST-эндпоинты `/as/v3` и `/as/v3/admin`
`application-setting-client`	Необязательный REST-клиент (Java). Нужен только сервису, который обращается к параметрам другого приложения по HTTP
`application-setting-base-auth`, `application-setting-cedar-auth`, `application-setting-basic-auth`	Провайдеры авторизации: базовый интерфейс, Cedar, Keycloak

Модуль **всегда подключается как библиотека**: параметры хранятся в БД приложения, выделение в отдельный сервис не предусмотрено. При подключении `application-setting-gate` автоматически регистрируются REST-эндпоинты `/as/v3` и `/as/v3/admin`, предназначенные для **внешнего взаимодействия** (service-to-service, UI) без дополнительных ограничений.

Эндпоинты регистрируются автоматически и **не отключаются** — это не опция сборки. Вызывать их не обязательно: внутри процесса работайте через `AppSettingService`. См. Способы доступа.

Параметры конфигурации

Основные

Кеширование включено по умолчанию (`app-setting.cache.enabled=true`), провайдер — `noop` . Если кеширование не нужно, дополнительных настроек не требуется; чтобы отключить — задайте `app-setting.cache.enabled=false` .

Существуют следующие параметры конфигурации:

Название	Описание	Тип	Значение по умолчанию	Возможные значения
`app-setting.prefix`	Префикс, используемый при хранении параметров	String	`setting`	
`app-setting.cache.provider`	Тип кеширования значений параметров	String	`noop`	`noop` , `caffeine` , `redis` , `custom` (см. ниже)
`app-setting.cache.enabled`	Флаг, позволяющий отключить кеширование	Boolean	`true`	`true` , `false`
`app-setting.cache.ttl`	Время хранения значений в кеше	Duration	Без ограничений	

Провайдеры кеширования

Название	Описание
`noop`	Без кеширования (по умолчанию). Аналогично случаю, когда блок `cache` не используется
`caffeine`	Кеширование в память с помощью Quarkus Caffeine
`redis`	Кеширование в Redis

Название	Описание
`custom`	Кастомное кеширование, настраивается вручную (реализуйте `ICacheService`)

> **Примечание**
> Для провайдеров `redis` и `custom` могут понадобиться дополнительные настройки, например, зависимость для сервиса Redis и его адрес.

Права и роли

Для работы с параметрами через REST API требуется аутентификация. Подробное описание провайдеров авторизации, ролей и примера Cedar-политики (включая обязательное действие `entityRestriction`) — см. раздел Права и роли.

REST API

При подключении `application-setting-gate` автоматически регистрируются REST-эндпоинты для работы с параметрами через HTTP.

> Подробное описание REST API, авторизации и примеры — см. раздел REST API.

Примеры

Готовые примеры использования модуля в проекте (через `AppSettingService`, REST API и REST-клиент), лучшие практики и типовые ошибки — см. раздел Примеры.

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Подключение к проекту

Подключение к проекту

Оглавление

- Быстрый старт
  - 1. Добавить зависимости в pom.xml
  - 2. Подключить Liquibase
  - 3. Создать параметр в БД
  - 4. Использовать в коде
  - 5. Использовать REST API
- Настройка

Быстрый старт

1. Добавить зависимости в `pom.xml`

Основная зависимость (обязательна)

```
<dependency>
    <groupId>com.bivgroup.lib</groupId>
    <artifactId>application-setting-gate</artifactId>
    <version>${biv-platform.version}</version>
</dependency>
```

> Модуль `application-setting-gate` транзитивно подключает `application-setting-service`,
> который содержит основной класс `AppSettingService`. Для работы с параметрами достаточно подключить только `application-setting-gate`.

Авторизация (выбрать один)

```
<!-- Авторизация через Cedar (рекомендуется для production) -->
<dependency>
    <groupId>com.bivgroup.lib</groupId>

```

```
    <artifactId>application-setting-cedar-auth</artifactId>
    <version>${biv-platform.version}</version>
</dependency>

<!-- Авторизация через Keycloak (альтернатива Cedar) -->
<!--
<dependency>
    <groupId>com.bivgroup.lib</groupId>
    <artifactId>application-setting-keycloack-auth</artifactId>
    <version>${biv-platform.version}</version>
</dependency>
-->
```

> Без одного из этих модулей приложение не запустится с ошибкой `Unsatisfied dependency for type AuthContext`.

База данных

Таблица `cmn_app_setting` создаётся ченджсетом из модуля `database`. Для этого добавьте зависимости:

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-liquibase</artifactId>
</dependency>

<dependency>
    <groupId>com.bivgroup.lib</groupId>
    <artifactId>database</artifactId>
    <version>${biv-platform.version}</version>
</dependency>
```

2. Подключить Liquibase

В ресурсах проекта (например, `src/main/resources/db/`) создайте основной Liquibase-файл `changeLog.xml` и включите в него общий ченджсет платформы:

```
<?xml version="1.1" encoding="UTF-8" standalone="no"?>
<databaseChangeLog
    xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"
    xmlns:pro="http://www.liquibase.org/xml/ns/pro"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog-ext
        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd
        http://www.liquibase.org/xml/ns/pro
```

```
        http://www.liquibase.org/xml/ns/pro/liquibase-pro-latest.xsd
        http://www.liquibase.org/xml/ns/dbchangelog
        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd">
    <include file="db/platformChangeLog.xml"/>
</databaseChangeLog>
```

В `application.properties` включите миграцию при старте:

```
quarkus.liquibase.migrate-at-start=true
```

> Таблица `cmn_app_setting` (а также sequence и уникальный индекс) создаются автоматически ченджсетом `001_app_settings_init.xml` из модуля `database`, который включается через `db/platformChangeLog.xml`. **Создавать таблицу вручную не нужно.**

3. Создать параметр в БД

```
INSERT INTO cmn_app_setting (id, key, value, type, scope_level_1, scope_id_1, s
VALUES (1, 'myKey', 'myValue', 'string', 'GLOBAL', null, 'GLOBAL', null, 'GLOB
```

4. Использовать в коде

> Примеры использования `AppSettingService` (инъекция, получение, создание, обновление, удаление, срезы, параметры по ролям, чтение типов) — см. раздел Примеры.

> Полное описание методов `AppSettingService` (получение, создание, обновление, удаление, срезы, типы значений, кеширование) — см. раздел Использование через AppSettingService.

5. Использовать REST API

При подключении `application-setting-gate` автоматически регистрируются REST-эндпоинты:

- `POST`, `PATCH`, `DELETE /as/v3/` — получение, обновление, удаление по ключу + срезы;
- `GET /as/v3/admin/{id}`, `POST /as/v3/admin/list`, `PUT /as/v3/admin/`, `PATCH /as/v3/admin/{id}`, `DELETE /as/v3/admin/{id}` — полный CRUD по ID (требуется роль `GridViewAdmin`).

Эндпоинты регистрируются автоматически и **не отключаются**, но вызывать их не обязательно. Внутри процесса работайте через `AppSettingService`; извне эндпоинты вызывают фронтенд/UI напрямую по HTTP, а другой **Java**-сервис — через необязательный REST-клиент `application-setting-client` (см. Способы доступа). Модуль при этом остаётся библиотекой в вашем приложении, а не отдельным сервисом.

> Примеры запросов к REST API — см. раздел Примеры.

> Подробное описание REST API, авторизации и примеры — см. раздел REST API.

Настройка

После подключения необходимо:

1. **Права доступа** — см. Права и роли
2. **Кеширование** (опционально) — см. Параметры конфигурации

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Примеры

Примеры

Оглавление

- Сквозной сценарий
- Работа через AppSettingService
  - Инъекция
  - Получение параметра
  - Параметры текущего пользователя
  - Пользовательские параметры через DTO
  - Создание параметра
  - Обновление параметра
  - Удаление параметра
  - Получение списка параметров
  - Параметры по ролям (срезы MANAGER / EMPLOYEE / GLOBAL)
- Работа через REST API
- Работа через REST-клиент
- Срезы и каскадный поиск
- Кеш
  - Создание и обновление параметров в разных срезах
  - Инвалидация кеша
- Типы значений
- Лучшие практики
- Частые ошибки

В этом разделе собраны примеры использования модуля в проекте: программно через `AppSettingService`, по HTTP через REST API, через REST-клиент, а также лучшие практики и типовые ошибки.

Сквозной сценарий

Типовая последовательность подключения и использования:

1. Добавить зависимости (`application-setting-gate` + auth-модуль + `database`) и подключить Liquibase — см. Подключение к проекту.
2. Создать параметр в БД (SQL-вставкой или через Admin API) — см. Создать параметр в БД.
3. Читать/изменять параметр в коде через `AppSettingService` (см. ниже).
4. При необходимости отдавать параметры наружу через REST API (см. ниже); другому **Java**-сервису можно обращаться через необязательный REST-клиент.
5. Выдать права на сущность `CmnAppSetting` — см. Права и роли.

Работа через AppSettingService

Инъекция

```
import com.bivgroup.lib.appsetting.service.AppSettingService;
import com.bivgroup.lib.appsetting.pojo.AppSettingType;
import jakarta.inject.Inject;

@Inject
AppSettingService appSettingService;
```

Получение параметра

```
// Без срезов (глобальный параметр)
String appName = appSettingService.getAppSetting("appName", AppSettingType.STR

// С одним срезом (для пользователя PIOTR)
Integer hours = appSettingService.getAppSetting("workHours", AppSettingType.NUI

// С двумя срезами
String value = appSettingService.getAppSetting("someKey", AppSettingType.STRIN

// С тремя срезами
String value3 = appSettingService.getAppSetting("someKey", AppSettingType.STRI
        "PROJECT", "456", "USER", "PIOTR", "ROLE", "ADMIN");
```

Если нужен полный объект `AppSetting` (с идентификатором, типом, описанием):

```
import com.bivgroup.lib.appsetting.pojo.AppSetting;
import com.bivgroup.lib.appsetting.pojo.get.GetAppSettingRequest;

AppSetting setting = appSettingService.getAppSetting(
        new GetAppSettingRequest("workHours", AppSettingType.NUMBER, "USER", "I
```

```
Object rawValue = setting.getValue();
AppSettingType type = setting.getType();
```

Параметры текущего пользователя

Типичный сценарий — хранить значения в срезе `USER` с идентификатором текущего пользователя и читать/обновлять их без админ-прав (методы «по ключу»):

```
import io.quarkus.security.identity.SecurityIdentity;
import com.bivgroup.lib.appsetting.pojo.update.UpdateAppSettingRequest;

@Inject
SecurityIdentity securityIdentity;

String user = securityIdentity.getPrincipal().getName();

// Прочитать параметр для текущего пользователя
Integer hours = appSettingService.getAppSetting("workHours", AppSettingType.NUI

// Обновить параметр для текущего пользователя (тип определяется автоматически
AppSetting updated = appSettingService.updateAppSetting(
        new UpdateAppSettingRequest("workHours", 6, null, "USER", user, null))
```

Для динамического определения типа по Java-классу или системному имени:

```
AppSettingType byClass = AppSettingType.getByObjectClass(Integer.class); // NUI
AppSettingType byName   = AppSettingType.getBySysName("number");          // NU
```

Пользовательские параметры через DTO

Если у пользователя есть группа однотипных настроек, их удобно описать отдельным DTO: имена полей совпадают с ключами параметров, а тип каждого поля определяется автоматически через `AppSettingType.getByObjectClass`.

```
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;

import java.util.Date;
import java.util.List;

@Data
@JsonIgnoreProperties(ignoreUnknown = true)
public class UserAppSettingGetResult {

```

```
    private String notDeleteString;

    private Number notDeleteNumber;

    private Boolean notDeleteBoolean;

    private Date notDeleteDate;

    private Object notDeleteObject;

    private List<Object> notDeleteArray;
}
```

Сервис обходит поля DTO через reflection и читает/обновляет их в срезе `USER` с идентификатором текущего пользователя:

```
import com.bivgroup.lib.appsetting.pojo.AppSetting;
import com.bivgroup.lib.appsetting.pojo.AppSettingType;
import com.bivgroup.lib.appsetting.pojo.get.GetAppSettingRequest;
import com.bivgroup.lib.appsetting.pojo.update.UpdateAppSettingRequest;
import com.bivgroup.lib.appsetting.service.AppSettingService;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Inject;
import jakarta.ws.rs.core.SecurityContext;

import java.lang.reflect.Field;

@ApplicationScoped
public class UserSettingsService {

    @Inject
    SecurityContext securityContext;

    @Inject
    AppSettingService appSettingService;

    public UserAppSettingGetResult getUserAppSetting() throws IllegalAccessExc
        String user = securityContext.getUserPrincipal().getName();
        UserAppSettingGetResult result = new UserAppSettingGetResult();
        for (Field field : result.getClass().getDeclaredFields()) {
            field.setAccessible(true);
            AppSetting appSetting = appSettingService.getAppSetting(new GetApp
                    field.getName(), AppSettingType.getByObjectClass(field.get
            field.set(result, appSetting.getValue());
        }
        return result;
    }

    public void updateUserAppSetting(UserAppSettingUpdateRequest request) thro
```

```
        String user = securityContext.getUserPrincipal().getName();
        for (Field field : request.getClass().getDeclaredFields()) {
            field.setAccessible(true);
            appSettingService.updateAppSetting(new UpdateAppSettingRequest(
                    field.getName(), field.get(request), null, "USER", user, nu
        }
    }
}
```

При обновлении тип можно не указывать (`null`) — он определяется автоматически по значению. Подход удобен для UI, когда нужно отдать или принять сразу группу пользовательских настроек.

Создание параметра

`createAppSetting` доступен **только администраторам** (`GridViewAdmin` / `gridViewAdmin`).

```
import com.bivgroup.lib.appsetting.pojo.create.CreateAppSettingRequest;

// Глобальный параметр
AppSetting created = appSettingService.createAppSetting(
        new CreateAppSettingRequest("workHours", 8, AppSettingType.NUMBER, "Pa

// Параметр в срезе USER/PIOTR
AppSetting createdForUser = appSettingService.createAppSetting(
        new CreateAppSettingRequest("workHours", 4, AppSettingType.NUMBER, "USI
```

Тип значения может быть указан явно (`AppSettingType`) либо определён автоматически по значению.

Обновление параметра

```
import com.bivgroup.lib.appsetting.pojo.update.UpdateAppSettingRequest;

// Обновление по ключу + срезы
AppSetting updated = appSettingService.updateAppSetting(
        new UpdateAppSettingRequest("workHours", 6, AppSettingType.NUMBER, "USI

// Обновление по идентификатору (только администраторы)
AppSetting updatedById = appSettingService.updateAppSetting(1L,
        new UpdateAppSettingRequest("workHours", 6, AppSettingType.NUMBER, "Pa
```

Удаление параметра

```
import com.bivgroup.lib.appsetting.pojo.delete.DeleteAppSettingRequest;
import com.bivgroup.lib.appsetting.pojo.delete.DeleteAppSettingResponse;

// Удаление по ключу + срезы
DeleteAppSettingResponse deleted = appSettingService.deleteAppSetting(
        new DeleteAppSettingRequest("workHours", "USER", "PIOTR"));

// Удаление по идентификатору (только администраторы)
DeleteAppSettingResponse deletedById = appSettingService.deleteAppSetting(1L);
```

Получение списка параметров

```
import com.bivgroup.lib.appsetting.pojo.get.GetAppSettingsByScopeRequest;
import com.bivgroup.lib.entitydata.services.data.DataResponseObjectList;

GetAppSettingsByScopeRequest request = GetAppSettingsByScopeRequest.builder().l

DataResponseObjectList list = appSettingService.getAppSettingsByScopes(request
```

Метод возвращает постраничный список (`DataResponseObjectList`) и используется, в том числе, для отображения через журнал гибких таблиц.

Параметры по ролям (срезы `MANAGER` / `EMPLOYEE` / `GLOBAL`)

Срезы не ограничены `USER` и `GLOBAL` : можно хранить разные значения для ролей, филиалов, проектов и т.п. Пример — список виджетов дашборда: свой для менеджера и сотрудника, с общим значением в `GLOBAL` .

```
import com.bivgroup.lib.appsetting.pojo.AppSetting;
import com.bivgroup.lib.appsetting.pojo.AppSettingType;
import com.bivgroup.lib.appsetting.pojo.get.GetAppSettingRequest;
import com.bivgroup.lib.appsetting.service.AppSettingService;
import com.bivgroup.lib.serviceexception.HandledServiceException;

private static final String GLOBAL = "GLOBAL";
private static final String MANAGER = "MANAGER";
private static final String EMPLOYEE = "EMPLOYEE";

@Inject
AppSettingService appSettingService;

private List<Map<String, Object>> getWidgetConfigs(RoleEnum role) throws Handl
    GetAppSettingRequest request = new GetAppSettingRequest();
    request.setKey("mobile_widget_list");
    request.setType(AppSettingType.ARRAY);
```

```
    String scopeLevel = determineScopeLevel(role);
    request.setScopeLevel1(scopeLevel);
    if (!GLOBAL.equals(scopeLevel)) {
        request.setScopeId1(role.getCaption()); // например, "wf-teamlead"
    }
    request.setScopeLevel2(GLOBAL);
    request.setScopeLevel3(GLOBAL);

    AppSetting appSetting = appSettingService.getAppSetting(request);
    return (List<Map<String, Object>>) appSetting.getValue();
}

private String determineScopeLevel(RoleEnum role) {
    return role == null ? GLOBAL
            : getManagerRoles().contains(role) ? MANAGER
            : RoleEnum.EMPLOYEE.equals(role) ? EMPLOYEE
            : GLOBAL;
}
```

Сложные типы читаются как обычные Java-коллекции: `ARRAY` — `List`, `OBJECT` — `Map`:

```
GetAppSettingRequest request = new GetAppSettingRequest();
request.setKey("mobile_widget_vacation");
request.setType(AppSettingType.OBJECT);
request.setScopeLevel1(GLOBAL);

AppSetting appSetting = appSettingService.getAppSetting(request);
Map<String, Object> widget = (Map<String, Object>) appSetting.getValue();
```

Если для выбранного среза записи нет, сервис вернёт значение из `GLOBAL` — см. Срезы и каскадный поиск.

Работа через REST API

Полное описание всех эндпоинтов (тела запросов/ответов, коды, права) — в разделе REST API. Все запросы требуют заголовок `Authorization: Bearer <token>`.

Типовой сценарий — прочитать параметр текущего пользователя (User API, по ключу + срезы):

```
curl -X POST https://your-service.com/as/v3/ \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/json" \
  -d '{"key": "workHours", "scopeLevel1": "USER", "scopeId1": "PIOTR", "scopeL
```

Создание параметра (Admin API, по ID, требуется роль `GridViewAdmin`):

```
curl -X PUT https://your-service.com/as/v3/admin/ \
  -H "Authorization: Bearer <admin-token>" \
  -H "Content-Type: application/json" \
  -d '{"key": "maxRetries", "value": 3, "type": "number", "scopeLevel1": "GLOB
```

Работа через REST-клиент

`application-setting-client` — **необязательная** зависимость. Она нужна только в одном сценарии: ваш **Java**-сервис обращается к параметрам **другого** приложения по HTTP. Внутри своего приложения используйте `AppSettingService`; фронтенд/UI вызывают REST API напрямую, без этого клиента. Подробнее — см. Способы доступа.

Подключите зависимость:

```
<dependency>
    <groupId>com.bivgroup.lib</groupId>
    <artifactId>application-setting-client</artifactId>
    <version>${biv-platform.version}</version>
</dependency>
```

Клиент предоставляет два интерфейса: `AppSettingClient` (User API `/as/v3`) и `AppSettingAdminClient` (Admin API `/as/v3/admin`). Оба помечены `@RegisterRestClient(configKey = "app-setting-api")` и пробрасывают токен через `@AccessToken`, поэтому в конфигурации указывается только базовый URL приложения-носителя (без пути):

```
quarkus.rest-client.app-setting-api.url=https://your-service.com
```

```
import jakarta.inject.Inject;
import org.eclipse.microprofile.rest.client.inject.RestClient;
import com.bivgroup.lib.appsetting.client.AppSettingClient;
import com.bivgroup.lib.appsetting.pojo.AppSetting;
import com.bivgroup.lib.appsetting.pojo.AppSettingType;
import com.bivgroup.lib.appsetting.pojo.get.GetAppSettingRequest;
import com.bivgroup.lib.appsetting.pojo.update.UpdateAppSettingRequest;

@Inject
@RestClient
AppSettingClient appSettingClient;

// Получить параметр
```

```
AppSetting setting = appSettingClient.get(
    new GetAppSettingRequest("workHours", AppSettingType.NUMBER, "USER", "PIOTI
);

// Обновить параметр
AppSetting updated = appSettingClient.update(
    new UpdateAppSettingRequest("workHours", 6, AppSettingType.NUMBER, "USER",
);
```

`AppSettingAdminClient` (требует роль `GridViewAdmin`) добавляет операции по ID: `list`, `create`, `get(id)`, `update(id, ...)`, `delete(id)`.

Срезы и каскадный поиск

Если для запрошенного среза значения нет, сервис последовательно ищет менее конкретные срезы:

1. Точное совпадение по всем указанным срезам
2. Срез 3 заменяется на `GLOBAL`
3. Срезы 2 и 3 заменяются на `GLOBAL`
4. Все срезы = `GLOBAL`

Это позволяет задать общее значение в `GLOBAL` и переопределять его только там, где нужно.

Кеш

> **Важно.** По умолчанию провайдер кеширования — `noop` (`app-setting.cache.provider=noop`), т.е. кеша нет. Приведённые ниже ключи и поведение инвалидации применимы только когда кеш включён: `app-setting.cache.enabled=true` и `app-setting.cache.provider` = `caffeine`, `redis` или `custom`. Подробнее — см. Параметры конфигурации.

Префикс ключа (`setting` в примерах) определяется параметром `app-setting.prefix`.

Создание и обновление параметров в разных срезах

Вызваны два запроса «Создать параметр»:

```
{
    "key": "workHours",
    "value": 8,
    "scopeLevel1": "GLOBAL",
    "scopeId1": null,
```

```
    "description": "Рабочие часы"
}
```

и

```
{
    "key": "workHours",
    "value": 4,
    "scopeLevel1": "USER",
    "scopeId1": "PIOTR",
    "description": "Рабочие часы Пети"
}
```

В БД будут созданы две записи.

Затем три запроса «Получить параметр»:

```
{ "key": "workHours", "scopeLevel1": "GLOBAL" }
```

```
{ "key": "workHours", "scopeLevel1": "USER", "scopeId1": "PIOTR" }
```

```
{ "key": "workHours", "scopeLevel1": "USER", "scopeId1": "IVAN" }
```

В кеше будут три записи:

Ключ кеша	Значение
`setting:workHours:GLOBAL:null:GLOBAL:null:GLOBAL:null`	`8`
`setting:workHours:USER:PIOTR:GLOBAL:null:GLOBAL:null`	`4`
`setting:workHours:USER:IVAN:GLOBAL:null:GLOBAL:null`	`8` (GLOBAL, т.к. для IVAN нет записи в БД)

Схематично:

```
setting
└── workHours
    ├── GLOBAL:null:GLOBAL:null:GLOBAL:null → 8
```

```
    ├── USER:PIOTR:GLOBAL:null:GLOBAL:null → 4
    └── USER:IVAN:GLOBAL:null:GLOBAL:null → 8 (fallback на GLOBAL)
```

Инвалидация кеша

- Если обновить параметр `workHours` для среза `USER/PIOTR` — удалится только ключ `setting:workHours:USER:PIOTR:GLOBAL:null:GLOBAL:null`
- Если обновить параметр `workHours` для среза `GLOBAL` — удалятся **все** ключи, начинающиеся на `setting:workHours`

Типы значений

Ниже для наглядности тип указан явно (`type`), хотя он может определяться автоматически по значению.

string — строковый тип

```json
{
  "key": "appName",
  "value": "MyApp",
  "type": "string",
  "scopeLevel1": "GLOBAL"
}
```

number — числовой тип

```json
{
  "key": "workHours",
  "value": 8,
  "type": "number",
  "scopeLevel1": "GLOBAL",
  "description": "Рабочие часы"
}
```

boolean — булевый тип

```json
{
  "key": "notificationsEnabled",
  "value": true,
  "type": "boolean",
  "scopeLevel1": "GLOBAL"
}
```

date — временной тип

```
{
  "key": "contractDate",
  "value": "2024-03-15T00:00:00+03:00",
  "type": "date",
  "scopeLevel1": "PROJECT",
  "scopeId1": "456",
  "description": "Дата подписания договора"
}
```

> **Формат** `date` — ISO-8601 со смещением таймзоны, например `2000-01-01T00:00:00+03:00`.

object — объект

```
{
  "key": "userPreferences",
  "value": {"theme": "dark", "language": "ru"},
  "type": "object",
  "scopeLevel1": "USER",
  "scopeId1": "FIODOR"
}
```

array — массив

```
{
  "key": "allowedRoles",
  "value": ["admin", "user", "moderator"],
  "type": "array",
  "scopeLevel1": "GLOBAL",
  "scopeId1": null
}
```

> **Системное имя типа-массива** — `array` (см. Описание AppSettingType).

Лучшие практики

- **Срезы.** Задавайте единое соглашение об именовании: один и тот же срез на одном и том же уровне для всех параметров (например, срез 1 = `USER` или `GLOBAL`).
- **GLOBAL по умолчанию.** Храните общее значение в `GLOBAL`, а в конкретных срезах задавайте только переопределения — за счёт каскадного поиска это уменьшает число записей.
- **Кеширование.** Для production включайте провайдер `caffeine` или `redis`: по умолчанию используется `noop`, и значения читаются из БД на каждый запрос.

- **Тип значения.** Указывайте `type` явно, если значение может быть неоднозначно разобрано (например, числовая строка), чтобы избежать ошибок конвертации.
- **Доступ из кода.** Внутри процесса используйте `AppSettingService` — это быстрее, чем HTTP. REST API предназначен для внешнего взаимодействия (UI, другие микросервисы).
- **Межсервисное взаимодействие.** Если другой Java-сервис обращается к параметрам по HTTP — используйте необязательный `application-setting-client`, а не ручные HTTP-запросы. Внутри своего приложения клиент не нужен.
- **Авторизация.** Подключите один auth-модуль (`application-setting-cedar-auth` для production или `application-setting-basic-auth` для Keycloak); без него приложение не запустится. См. Права и роли.
- **Cedar.** Помимо CRUD-действий обязательно выдавайте `Action::"entityRestriction"` — иначе запросы завершатся ошибкой `Forbidden`.
- **Схема БД.** Не создавайте таблицу `cmn_app_setting` вручную — она создаётся ченджсетом из модуля `database`. См. Подключение к проекту.

Частые ошибки

Симптом	Причина	Решение
`Unsatisfied dependency for type AuthContext` при старте	Не подключён auth-модуль	Добавить `application-setting-cedar-auth` или `application-setting-basic-auth`
`Forbidden` / «Нет прав на выполнение действия над сущностью CmnAppSetting»	В Cedar не выдано `entityRestriction`	Разрешить `Action::"entityRestriction"` для `Entity::"CmnAppSetting"`
`Forbidden` при create/update/delete по ID	Не выдана роль `GridViewAdmin` (`gridViewAdmin`)	Выдать роль администратора
Значения не кешируются	Провайдер кеша `noop` (по умолчанию)	Задать `app-setting.cache.provider=caffeine` (или `redis`) и `app-setting.cache.enabled=true`

Смотрите также

- Подключение к проекту
- Использование через AppSettingService
- REST API
- Права и роли
- Классы (DTO)

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Параметры приложения - Сервис

Использование через AppSettingService

Оглавление

- Методы
- Динамическое определение типа
- Каскадный поиск срезов
- Типы значений
- Особенности
- Смотрите также

Основной способ работы с параметрами приложения — через сервис `AppSettingService`. Он работает внутри вашего процесса, использует БД и кеш приложения и не требует обращения по HTTP.

> Готовые примеры кода (инъекция, получение, создание, обновление, удаление, список, параметры текущего пользователя и по ролям) — см. раздел Примеры.
>
> Доступ по HTTP из другого Java-сервиса или из UI — см. раздел REST API.

Методы

Перегрузки `getAppSetting` возвращают **типизированное значение** (`<T>`) в соответствии с `AppSettingType`.

Метод	Назначение
`getAppSetting(String key, AppSettingType type)`	Получить значение без (глобальный параметр)
`getAppSetting(String key, AppSettingType type, String scopeLevel1, String scopeId1)`	Получить значение с од срезом
`getAppSetting(String key, AppSettingType type, ..., scopeLevel2, scopeId2)`	Получить значение с дв срезами

Метод	Назначение
`getAppSetting(String key, AppSettingType type, ..., scopeLevel3, scopeId3)`	Получить значение с тр срезами
`getAppSetting(GetAppSettingRequest request)`	Получить полный объе `AppSetting` (id, тип
`getAppSetting(Long id)`	Получить параметр по идентификатору
`createAppSetting(CreateAppSettingRequest request)`	Создать параметр
`updateAppSetting(UpdateAppSettingRequest request)`	Обновить параметр по срезы
`updateAppSetting(Long id, UpdateAppSettingRequest request)`	Обновить параметр по идентификатору
`deleteAppSetting(DeleteAppSettingRequest request)`	Удалить параметр по кл срезы
`deleteAppSetting(Long id)`	Удалить параметр по идентификатору
`getAppSettingsByScopes(GetAppSettingsByScopeRequest request)`	Получить постраничны (`DataResponseObj` в т.ч. для журнала гибк

> Полное описание прав и Cedar-политики — см. раздел Права и роли.

Динамическое определение типа

Тип значения можно определить динамически статическими методами `AppSettingType`:

- `AppSettingType.getByObjectClass(Class<?>)` — по Java-классу;
- `AppSettingType.getBySysName(String)` — по системному имени.

> Пример — см. Пользовательские параметры через DTO.

Каскадный поиск срезов

Если для запрошенного среза значения нет, сервис последовательно ищет менее конкретные срезы (вплоть до `GLOBAL`). Подробнее — см. Срезы и каскадный поиск.

Типы значений

Тип (`AppSettingType`)	Java-класс	Пример значения
`STRING`	`String`	`"MyApp"`
`NUMBER`	`Number`	`8`
`BOOLEAN`	`Boolean`	`true`
`DATE`	`Date`	дата/время
`OBJECT`	`Object`	`Map` /JSON
`ARRAY`	`List`	список/JSON

> Примеры значений каждого типа — см. Типы значений.

Особенности

- Значения кешируются (ленивое заполнение при чтении). Провайдер кеша настраивается параметром `app-setting.cache.provider`. Подробнее — см. Кеш.
- При изменении параметра в конкретном срезе инвалидируется только его ключ кеша; при изменении `GLOBAL`-параметра сбрасываются все ключи с этим `key`.
- При обращении по ключу через Panache в Cedar-режиме дополнительно требуется действие `entityRestriction`.

Смотрите также

- Подключение к проекту
- Права и роли
- Классы (DTO)
- Примеры

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Использование через REST API

REST API

Оглавление

- Обзор
- User API (`/as/v3`)
  - Получить параметр
  - Обновить параметр
  - Удалить параметр
- Admin API (`/as/v3/admin`)
  - Получить параметр по ID
  - Получить список параметров
  - Создать параметр
  - Обновить параметр по ID
  - Удалить параметр по ID
- Авторизация
- Способы доступа
- REST-клиент
- Смотрите также

При подключении модуля `application-setting-gate` автоматически регистрируются два набора REST-эндпоинтов для работы с параметрами приложения через HTTP.

> **Целевое использование**: REST API предназначен для внешнего взаимодействия — его могут вызывать фронтенд, admin-UI, другие сервисы и любые HTTP-клиенты. Модуль при этом остаётся библиотекой в вашем приложении, а не отдельным сервисом. Эндпоинты регистрируются автоматически и **не отключаются** — это не опция сборки. Если вам нужен только программный доступ через `AppSettingService` — просто не вызывайте их.

> Готовые сценарии использования (`curl`) — см. раздел Примеры.

Обзор

Набор	Базовый путь	Назначение	Доступ
User API	`/as/v3`	Получение, обновление, удаление по ключу + срезы	Аутентификация + права CRUD на `CmnAppSetting`
Admin API	`/as/v3/admin`	Полный CRUD (create, list, get/update/delete по ID)	Требуется роль `GridViewAdmin`

Все эндпоинты требуют аутентификации (`@Authenticated`); токен передаётся в заголовке `Authorization: Bearer <token>` . Полные требования по правам и Cedar-политика — в разделе Права и роли.

User API (`/as/v3`)

Эндпоинты для повседневной работы с параметрами. Идентификация параметра — по ключу (`key`) и срезам (`scopeLevel` / `scopeId`).

Метод	Путь	Назначение	Тело запроса	Ответ
POST	`/as/v3/`	Получить параметр	GetAppSettingRequest	AppSetting
PATCH	`/as/v3/`	Обновить параметр	UpdateAppSettingRequest	AppSetting
DELETE	`/as/v3/`	Удалить параметр	DeleteAppSettingRequest	DeleteAppSetti

Получить параметр

```
POST /as/v3/
```

Назначение: получить параметр по ключу и срезам. Если для запрошенного среза значения нет — сработает каскадный поиск вплоть до `GLOBAL` .

Тело запроса: GetAppSettingRequest

```
{
  "key": "workHours",
  "type": "number",

```

```
  "scopeLevel1": "USER",
  "scopeId1": "PIOTR",
  "scopeLevel2": "GLOBAL",
  "scopeLevel3": "GLOBAL"
}
```

Ответ: `AppSetting`

```
{
  "key": "workHours",
  "value": 4,
  "type": "number",
  "scopeLevel1": "USER",
  "scopeId1": "PIOTR",
  "scopeLevel2": "GLOBAL",
  "scopeLevel3": "GLOBAL",
  "description": "Рабочие часы"
}
```

Коды ответа:

Код	Описание
200	OK
400	Ошибки валидации или бизнес-логики
500	Непредвиденная ошибка сервиса

Авторизация: аутентификация + права чтения `CmnAppSetting` (в Cedar — дополнительно `entityRestriction`).

▶ Пример cURL

Обновить параметр

```
PATCH /as/v3/
```

Назначение: обновить значение параметра по ключу и срезам.

Тело запроса: `UpdateAppSettingRequest`

```
{
  "key": "workHours",
  "value": 6,
  "type": "number",
  "scopeLevel1": "USER",
  "scopeId1": "PIOTR",
  "scopeLevel2": "GLOBAL",
  "scopeLevel3": "GLOBAL"
}
```

Ответ: `AppSetting`

```
{
  "key": "workHours",
  "value": 6,
  "type": "number",
  "scopeLevel1": "USER",
  "scopeId1": "PIOTR",
  "scopeLevel2": "GLOBAL",
  "scopeLevel3": "GLOBAL",
  "description": "Рабочие часы"
}
```

Коды ответа:

Код	Описание
200	OK
400	Ошибки валидации или бизнес-логики
500	Непредвиденная ошибка сервиса

Авторизация: аутентификация + права обновления `CmnAppSetting` (в Cedar — дополнительно `entityRestriction`).

▶ Пример cURL

Удалить параметр

```
DELETE /as/v3/
```

Назначение: удалить параметр по ключу и срезам.

Тело запроса: `DeleteAppSettingRequest`

```
{
  "key": "workHours",
  "scopeLevel1": "USER",
  "scopeId1": "PIOTR",
  "scopeLevel2": "GLOBAL",
  "scopeLevel3": "GLOBAL"
}
```

Ответ: `DeleteAppSettingResponse`

```
{
  "text": "Параметр удалён"
}
```

Коды ответа:

Код	Описание
200	OK
400	Ошибки валидации или бизнес-логики
500	Непредвиденная ошибка сервиса

Авторизация: аутентификация + права удаления `CmnAppSetting` (в Cedar — дополнительно `entityRestriction`).

▶ Пример cURL

Admin API (`/as/v3/admin`)

Административные эндпоинты для управления параметрами. Идентификация параметра — по зашифрованному ID (`{id}`). Требуется роль `GridViewAdmin` .

Метод	Путь	Назначение	Тело запроса
GET	`/as/v3/admin/{id}`	Получить параметр по ID	—
POST	`/as/v3/admin/list`	Получить список параметров	`GetAppSettingsByScopeRequest`
PUT	`/as/v3/admin/`	Создать параметр	`CreateAppSettingRequest`
PATCH	`/as/v3/admin/{id}`	Обновить параметр по ID	`UpdateAppSettingRequest`
DELETE	`/as/v3/admin/{id}`	Удалить параметр по ID	—

`{id}` — зашифрованный идентификатор параметра (см. `AppSetting.id`).

> Admin API **не содержит** операций по ключу на корне (`POST` / `PATCH` / `DELETE` `/as/v3/admin/`). Для работы по ключу используйте User API (`/as/v3`).

Получить параметр по ID

```
GET /as/v3/admin/{id}
```

Назначение: получить параметр по зашифрованному идентификатору.

Параметр	Тип	Описание
`id`	String	Зашифрованный идентификатор параметра

Ответ: `AppSetting`

```
{
  "id": "4VOdpW3VA-86pValIlW4-w==",
  "key": "maxRetries",
```

```
  "value": 3,
  "type": "number",
  "scopeLevel1": "GLOBAL",
  "scopeLevel2": "GLOBAL",
  "scopeLevel3": "GLOBAL",
  "description": "Максимальное число повторов"
}
```

Коды ответа:

Код	Описание
200	OK
400	Ошибки валидации или бизнес-логики
500	Непредвиденная ошибка сервиса

Авторизация: аутентификация + роль `GridViewAdmin` .

▶ Пример cURL

Получить список параметров

```
POST /as/v3/admin/list
```

Назначение: получить постраничный список параметров (в том числе для журнала гибких таблиц).

Тело запроса: `GetAppSettingsByScopeRequest`

```
{
  "scopeLevel1": "GLOBAL",
  "scopeLevel2": "GLOBAL",
  "scopeLevel3": "GLOBAL"
}
```

Ответ: `DataResponseObjectList`

```
{
  "list": [
    {
      "id": "4VOdpW3VA-86pValIlW4-w==",
      "key": "maxRetries",
```

```
      "value": 3,
      "type": "number",
      "scopeLevel1": "GLOBAL",
      "scopeLevel2": "GLOBAL",
      "scopeLevel3": "GLOBAL",
      "description": "Максимальное число повторов"
    }
  ],
  "count": 1
}
```

Коды ответа:

Код	Описание
200	OK
400	Ошибки валидации или бизнес-логики
500	Непредвиденная ошибка сервиса

Авторизация: аутентификация + роль `GridViewAdmin` .

▶ Пример cURL

Создать параметр

```
PUT /as/v3/admin/
```

Назначение: создать параметр.

Тело запроса: `CreateAppSettingRequest`

```
{
  "key": "maxRetries",
  "value": 3,
  "type": "number",
  "scopeLevel1": "GLOBAL",
  "description": "Максимальное число повторов"
}
```

Ответ: `AppSetting`

```
{
  "id": "4VOdpW3VA-86pValIlW4-w==",
  "key": "maxRetries",
  "value": 3,
  "type": "number",
  "scopeLevel1": "GLOBAL",
  "scopeLevel2": "GLOBAL",
  "scopeLevel3": "GLOBAL",
  "description": "Максимальное число повторов"
}
```

Коды ответа:

Код	Описание
200	OK
400	Ошибки валидации или бизнес-логики
500	Непредвиденная ошибка сервиса

Авторизация: аутентификация + роль `GridViewAdmin` .

▶ Пример cURL

Обновить параметр по ID

```
PATCH /as/v3/admin/{id}
```

Назначение: обновить параметр по зашифрованному идентификатору.

Параметр	Тип	Описание
`id`	String	Зашифрованный идентификатор параметра

Тело запроса: `UpdateAppSettingRequest`

```
{
  "key": "maxRetries",
  "value": 5,
  "type": "number",

```

```
    "description": "Максимальное число повторов"
  }
```

Ответ: AppSetting

```
{
  "id": "4VOdpW3VA-86pValIlW4-w==",
  "key": "maxRetries",
  "value": 5,
  "type": "number",
  "scopeLevel1": "GLOBAL",
  "scopeLevel2": "GLOBAL",
  "scopeLevel3": "GLOBAL",
  "description": "Максимальное число повторов"
}
```

Коды ответа:

Код	Описание
200	OK
400	Ошибки валидации или бизнес-логики
500	Непредвиденная ошибка сервиса

Авторизация: аутентификация + роль `GridViewAdmin` .

▶ Пример cURL

Удалить параметр по ID

```
DELETE /as/v3/admin/{id}
```

Назначение: удалить параметр по зашифрованному идентификатору.

Параметр	Тип	Описание
`id`	String	Зашифрованный идентификатор параметра

Ответ: DeleteAppSettingResponse

```
{
  "text": "Параметр удалён"
}
```

Коды ответа:

Код	Описание
200	OK
400	Ошибки валидации или бизнес-логики
500	Непредвиденная ошибка сервиса

Авторизация: аутентификация + роль `GridViewAdmin` .

▶ Пример cURL

Авторизация

Все эндпоинты требуют аутентификации (`@Authenticated`). Токен передаётся в заголовке `Authorization: Bearer <token>` . Требования по правам для каждого эндпоинта и Cedar-политика — см. раздел Права и роли.

Эндпоинт	Минимальные права
`POST /as/v3/` (get)	Аутентификация + права чтения `CmnAppSetting`
`PATCH /as/v3/` (update by key)	Аутентификация + права обновления `CmnAppSetting`
`DELETE /as/v3/` (delete by key)	Аутентификация + права удаления `CmnAppSetting`
`GET /as/v3/admin/{id}`	Аутентификация + роль `GridViewAdmin`
`POST /as/v3/admin/list`	Аутентификация + роль `GridViewAdmin`
`PUT /as/v3/admin/`	Аутентификация + роль `GridViewAdmin`
`PATCH /as/v3/admin/{id}`	Аутентификация + роль `GridViewAdmin`
`DELETE /as/v3/admin/{id}`	Аутентификация + роль `GridViewAdmin`

> **Важно (Cedar):** помимо CRUD-действий обязательно выдайте `Action::"entityRestriction"` — иначе запросы завершатся ошибкой `Forbidden`. Подробнее — см. Права и роли.

Способы доступа

REST API — один из способов доступа к параметрам. Выбор зависит от того, откуда нужны данные:

Где нужны параметры	Что использовать	Нужен ли `application-setting-client`
Внутри своего приложения (Java)	`AppSettingService`	Нет
Из фронтенда / UI	REST API по HTTP	Нет
Из другого Java-сервиса по HTTP	REST API + `application-setting-client`	Да

REST-клиент

`application-setting-client` — **необязательная зависимость**. Подключайте её только в одном сценарии: ваш **Java**-сервис обращается к параметрам **другого** приложения по HTTP. Внутри своего приложения клиент не нужен — используйте `AppSettingService`. Фронтенд/UI этот клиент также не используют: они вызывают REST API напрямую по HTTP.

В `application.properties` укажите базовый URL приложения-носителя (без пути — `/as/v3` уже задан аннотацией `@Path` на клиенте):

```
quarkus.rest-client.app-setting-api.url=https://your-service.com
```

> Зависимость, `AppSettingClient` / `AppSettingAdminClient` и пример — см. Работа через REST-клиент.

Смотрите также

- Примеры
- Права и роли
- Классы (DTO)
- Подключение к проекту

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Права и роли

Права и роли

Оглавление

- Авторизация
- Роли
- CRUD-операции с сущностью CmnAppSetting
- Авторизация REST API
  - Требования по эндпоинтам
  - Различия между User API и Admin API

Авторизация

Для работы с параметрами через REST API требуется аутентификация (`@Authenticated`). Токен передаётся через заголовок `Authorization: Bearer <token>` .

Модуль поддерживает два провайдера авторизации через интерфейс `AuthContext` :

Провайдер	Модуль	Механизм
Keycloak	`application-setting-basic-auth`	Проверка ролей через `SecurityIdentity`
Cedar	`application-setting-cedar-auth`	Проверка действий через `UserOperationAuthService`

Роли

Роль / Действие	Описание
`GridViewAdmin` (Keycloak) / `gridViewAdmin` (Cedar)	Администратор параметров приложения. Имеет полный доступ ко всем CRUD-операциям

Роль / Действие	Описание
`UserCanUpdateOwnGridView` (Keycloak) / `updateOwnGridView` (Cedar)	Пользователь может изменять собственные параметры

CRUD-операции с сущностью `CmnAppSetting`

Для просмотра/редактирования параметров приложений в пользовательском интерфейсе необходимо дать права на CRUD-операции с сущностью `CmnAppSetting`.

> **Важно (Cedar):** помимо CRUD-действий обязательно нужно разрешить действие `Action::"entityRestriction"`. В Cedar-режиме каждая операция над сущностью через Panache (чтение/создание/обновление/удаление, а также получение списка через грид) проходит отдельную проверку права `entityRestriction`. Без него запрос завершится ошибкой `Forbidden` («Нет прав на выполнение действия над сущностью CmnAppSetting»), даже если CRUD-действия разрешены.

Пример Cedar-политики (все права):

```
// Разрешить действия с CmnAppSetting (методы «по ключу»)
permit (
    principal,
    action in [Action::"createEntity",Action::"readEntity",Action::"updateEnti
    resource == Entity::"CmnAppSetting"
);

// Разрешить административные действия (create/update/delete/get по ID)
permit (
    principal,
    action,
    resource == UserOperation::"gridViewAdmin"
);
```

Методы `AppSettingService`, работающие **по ключу + срезы** (`getAppSetting(GetAppSettingRequest)`, `updateAppSetting(UpdateAppSettingRequest)`, `deleteAppSetting(DeleteAppSettingRequest)`), требуют только CRUD-действий и `entityRestriction`. Методы, работающие **по ID** (`getAppSetting(Long)`, `createAppSetting`, `updateAppSetting(Long, ...)`, `deleteAppSetting(Long)`), дополнительно требуют разрешения `gridViewAdmin` (`UserOperation::"gridViewAdmin"`).

Если нужно разрешить чтение/редактирование, но без выдачи ошибки для записей, которые не видны пользователю, действие `entityRestriction` можно наложить на конкретную сущность точечно — например, через `@JoinRestriction` / `@WhereRestriction`.

Ограничение можно задать и программно через `@CustomDataRestrictorFor`. Например, скрывать записи в зависимости от значения параметра приложения:

```
import com.bivgroup.lib.appsetting.service.AppSettingService;
import com.bivgroup.lib.appsetting.pojo.AppSettingType;
import com.bivgroup.lib.auth.data.DataRestrictionDTO;
import com.bivgroup.lib.entitydata.services.data.*;

@ApplicationScoped
@CustomDataRestrictorFor(entity = "Branch")
public class BranchRestrictor implements CustomDataRestrictor {

    @Inject
    AppSettingService appSettingService;

    public DataRestrictionDTO makeDataRestriction(EntityQueryBuilderContext ct
        DataRestrictionDTO result = null;
        if (Boolean.TRUE.equals(appSettingService.getAppSetting("isHideBranch"
            result = new DataRestrictionDTO();
            result.setWhereHQL("MainAlias.code != ('SPB')");
        }
        return result;
    }
}
```

Авторизация REST API

Все REST-эндпоинты требуют аутентификации (`@Authenticated`). Токен передаётся через заголовок:

```
Authorization: Bearer <token>
```

Требования по эндпоинтам

Эндпоинт	Минимальные права
`POST /as/v3/` (get)	Аутентификация + права чтения `CmnAppSetting`
`PATCH /as/v3/` (update by key)	Аутентификация + права обновления `CmnAppSetting`

Эндпоинт	Минимальные права
`DELETE /as/v3/` (delete by key)	Аутентификация + права удаления `CmnAppSetting`
`GET /as/v3/admin/{id}`	Аутентификация + роль `GridViewAdmin`
`POST /as/v3/admin/list`	Аутентификация + роль `GridViewAdmin`
`PUT /as/v3/admin/`	Аутентификация + роль `GridViewAdmin`
`PATCH /as/v3/admin/{id}`	Аутентификация + роль `GridViewAdmin`
`DELETE /as/v3/admin/{id}`	Аутентификация + роль `GridViewAdmin`

Различия между User API и Admin API

- **User API** (`/as/v3`) — работает по ключу + срезы. Требует только аутентификации и прав CRUD на сущность `CmnAppSetting` . Подходит для повседневного использования.
- **Admin API** (`/as/v3/admin`) — работает по зашифрованному ID. Требует роль `GridViewAdmin` . Предназначен для администрирования и UI управления параметрами.

> Подробное описание REST API и примеры — см. раздел REST API.

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Параметры приложения - Классы

Классы (DTO)

Оглавление

- Архитектура
- Требования
- Конфигурация
- Модель данных
  - Создать параметр
  - Обновить параметр
  - Получить параметр
  - Получить все параметры
  - Удалить параметр
  - Описание AppSetting
  - Описание AppSettingType
  - Описание AppSettingScope

POJO-классы (DTO), используемые для передачи данных между слоями приложения и в REST API.

Архитектура

Методы разделяются по логике. Каждому методу выделен свой пакет, в котором располагаются классы запроса/ответа. Общие классы находятся в корневом пакете:

```
com.bivgroup.lib.appsetting.pojo
├── AppSetting.java              — основной DTO ответа
├── AppSettingScope.java         — базовый класс срезов (scope)
├── AppSettingType.java          — enum типов значений
├── constants/
│   ├── Constants.java           — константы (GLOBAL = "GLOBAL")
│   └── Message.java             — сообщения валидации и OpenAPI-описания
├── create/
│   └── CreateAppSettingRequest.java
├── update/
│   └── UpdateAppSettingRequest.java

```

```
├── get/
│   ├── GetAppSettingRequest.java
│   └── GetAppSettingsByScopeRequest.java
├── delete/
│   ├── DeleteAppSettingRequest.java
│   └── DeleteAppSettingResponse.java
├── utils/
│   └── CastUtils.java           – конвертация значений
└── validation/
    ├── CheckAppSettingScope1.java   – аннотация валидации среза 1
    ├── CheckAppSettingScope2.java   – аннотация валидации среза 2
    ├── CheckAppSettingScope3.java   – аннотация валидации среза 3
    ├── CheckAppSettingScopeValidator1.java – валидатор среза 1
    ├── CheckAppSettingScopeValidator2.java – валидатор среза 2
    └── CheckAppSettingScopeValidator3.java – валидатор среза 3
```

Требования

- Java 21+
- Quarkus 3.33.1+
- Maven 3.8+

Конфигурация

Добавить зависимость `application-setting-pojo` в `pom.xml` (транзитивно подтягивается через `application-setting-service` или `application-setting-gate`):

```xml
<dependency>
    <groupId>com.bivgroup.lib</groupId>
    <artifactId>application-setting-pojo</artifactId>
    <version>${biv-platform.version}</version>
</dependency>
```

Модель данных

Создать параметр

Запрос: `CreateAppSettingRequest`

▶ Таблица параметров

▶ Формат OpenAPI

Ответ: `AppSetting` — Описание AppSetting

▶ Таблица HTTP-кодов статуса ответа

Обновить параметр

Запрос: `UpdateAppSettingRequest`

▶ Таблица параметров

▶ Формат OpenAPI

Ответ: `AppSetting` — Описание AppSetting

▶ Таблица HTTP-кодов статуса ответа

Получить параметр

Запрос: `GetAppSettingRequest`

▶ Таблица параметров

▶ Формат OpenAPI

Ответ: `AppSetting` — Описание AppSetting

▶ Таблица HTTP-кодов статуса ответа

Получить все параметры

> Используется для отображения через журнал (журнал гибких таблиц).

Запрос: `GetAppSettingsByScopeRequest` — расширение класса `PagedListGridViewRequest` из зависимости entity-journal

▶ Таблица параметров

Ответ: `DataResponseObjectList` из зависимости entity-data

▶ Таблица HTTP-кодов статуса ответа

Удалить параметр

Запрос: `DeleteAppSettingRequest`

▶ Таблица параметров

▶ Формат OpenAPI

Ответ: `DeleteAppSettingResponse`

▶ Таблица параметров ответа

▶ Формат OpenAPI

▶ Таблица HTTP-кодов статуса ответа

Описание AppSetting

Класс, используемый для возврата ответа с описанием параметра приложения. Основной DTO, возвращаемый всеми операциями чтения.

▶ Таблица параметров

▶ Формат OpenAPI

> **Примечание**: Метод `getConvertValue()` (отмечен `@JsonIgnore`) позволяет получить типизированное значение, конвертируя `Object value` в нужный Java-тип на основе `AppSettingType`.

Описание AppSettingType

Enum, описывающий типы значений параметров приложения.

▶ Таблица значений

> Тип значения определяется автоматически при создании/обновлении параметра. Также может быть указан явно в запросе.

Описание AppSettingScope

Базовый класс для всех DTO запросов, содержащий данные о срезах.

▶ Таблица параметров

Правила валидации срезов (проверяются аннотациями `@CheckAppSettingScope1/2/3`):

Для каждого уровня среза выполняется одно из:

- Оба поля (`scopeLevel` и `scopeId`) пустые — срез не используется
- `scopeLevel = "GLOBAL"`, `scopeId` пустой — глобальный срез
- Оба поля заполнены — конкретный срез

Конструкторы: 4 перегрузки (для 0, 1, 2 или 3 уровней срезов) с опциональным параметром `fillDefaultScope`, который заполняет пустые срезы значением `"GLOBAL"`.

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js