Общие понятия

Документация по сервису уведомлений - Общие понятия

Содержание

- Назначение
- Требования
  - Общие требования
  - Требования к сервису уведомлений
  - Требования к сервисам-отправителям
  - Требования к интерфейсу пользователя
- Ограничения

Назначение

Сервис уведомлений предназначен для универсальной обработки, хранения и доставки уведомлений в интерфейс пользователя.

Сервис абстрагирован от бизнес-логики систем-отправителей и фокусируется на приёме запросов на отправку, маршрутизации, доставке уведомлений и предоставлении интерфейсов для их отображения и обработки.

Взаимодействие с сервисом осуществляется через библиотеку.

Требования

Общие требования

- Отправителем сообщений может быть любой сервис, которому необходимо отправить уведомление пользователю в интерфейсе.
- Количество сервисов-отправителей не ограничено.
- Получателем сообщений является пользователь, которому адресовано уведомление.
- Должна быть реализована изоляция данных разных сервисов, проектов и клиентов.
- Должна быть реализована библиотека для взаимодействия с сервисом уведомлений.

Требования к сервису уведомлений

Общая функциональность

- Должна быть реализована универсальная обработка и отправка уведомлений с абстракцией от бизнес-логики отправителей.
- Должна быть реализована изоляция данных разных сервисов, проектов и клиентов.
- Должна поддерживаться приоритизация уведомлений для обеспечения немедленной доставки критических событий в обход очередей. Поддерживаемые уровни приоритета:
  - `CRITICAL`
  - `HIGH`
  - `NORMAL`
  - `LOW`
- Должно обеспечиваться хранение истории уведомлений для отображения в пользовательском интерфейсе.
- Должен быть реализован API получения истории уведомлений с пагинацией и параметрами:
  - `limit`
  - `offset`
  - `sort`
- Должно выполняться отслеживание статусов доставки и действий получателя:
  - прочтение
  - клик
  - пользовательское действие
  - ошибка
- Должна поддерживаться категоризация уведомлений:
  - информационное
  - призыв к действию
  - ошибка
- При удалении уведомления пользователем в интерфейсе сообщение должно переводиться в архивный статус без физического удаления.
- Сервис должен быть максимально абстрагирован от бизнес-логики отправителей. Допускается только внутренняя логика маршрутизации, например доставка уведомлений уровня `CRITICAL` в обход очередей.

Аудит

Должно быть реализовано логирование следующих событий:

- получение запросов на отправку
- статусы обработки задач на подготовку и отправку
- факты отправки уведомлений
- действия пользователей с уведомлениями:
  - прочтение

- клик
- выполнение действия пользователя

Администрирование

- Должен быть реализован дашборд статистики через `Admin API` и/или `Grafana` .
- Должны собираться следующие метрики:
  - количество отправок
  - количество доставок
  - количество ошибок
  - статистика по отправителям
- Должна быть доступна ручная отправка уведомлений через `Admin API` для поддержки и тестирования.

Безопасность данных

- Должна быть реализована валидация отправителя.

Интеграция с пользовательским интерфейсом

Должна поддерживаться интеграция по `SSE` для отображения уведомлений в реальном времени, включая:

- всплывающие окна
- всплывающую верхнюю панель (`toasts`) для критических и важных уведомлений
- индикатор количества непрочитанных уведомлений в колокольчике

Требования к сервисам-отправителям

- Взаимодействие с сервисом уведомлений должно выполняться через библиотеку.

Обязательные поля запроса

- идентификатор сервиса-отправителя
- идентификатор пользователя
- `payload` , включающий:
  - заголовок
  - текст сообщения
  - список дополнительных параметров
  - категорию уведомления:
    - информационное
    - призыв к действию
    - ошибка
  - приоритет уведомления:

- `CRITICAL`
- `HIGH`
- `NORMAL`
- `LOW`

Требования к обработке ответа

- Должна выполняться корректная обработка синхронного ответа `202 Accepted` с идентификатором уведомления.
- Идентификатор уведомления должен сохраняться отправителем для последующего отслеживания статуса.

Требования к интерфейсу пользователя

Отображение уведомлений

- Должно поддерживаться отображение уведомлений в реальном времени через `SSE` .
- Должны поддерживаться:
  - всплывающие окна
  - всплывающая верхняя панель (`toasts`) для критических и важных уведомлений
  - индикатор непрочитанных уведомлений в колокольчике
- Должен отображаться список уведомлений, включая исторические записи.

Работа со статусами и действиями

- Должно поддерживаться управление статусами уведомлений:
  - не прочитано
  - прочитано
  - архивировано
- Должен поддерживаться пользовательский функционал действий по уведомлениям.
- Должна быть возможность проставления статуса прочтения при клике по уведомлению.
- Должна быть возможность отметить все непрочитанные уведомления как прочитанные по кнопке «**Прочитать все**».
- Должна поддерживаться возможность закрепления уведомлений в списке.
- Должно поддерживаться удаление уведомлений пользователем без физического удаления, с переводом на бэке в статус «**архивное**».

Поддерживаемые типы взаимодействия

Уведомления должны поддерживать:

- ссылки на сущности внутри `BIV Platform`
- ссылки на файлы

- ссылки на внешние источники
- кнопки с действиями

Отображение и визуальное оформление

- Должна поддерживаться обрезка текста уведомления для компактного отображения с возможностью раскрытия полного текста.
- Должно быть визуальное выделение важных и критических уведомлений в интерфейсе, например цветом или иконкой.
- Должен отображаться колокольчик с количеством непрочитанных сообщений.

Фильтрация и сортировка

Должна поддерживаться фильтрация уведомлений по следующим параметрам:

- статус уведомления:
  - прочитано
  - не прочитано
- дата и время сообщения
- категория сообщения

Ограничения

- Сервис должен быть максимально абстрагирован от бизнес-логики.

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

Документация для Back-End разработчика

Сервис уведомлений Инструкция для backend

Требования к инфраструктуре

- PostgreSQL 16.6+
- RabbitMQ
- Redis 8.6+ | Valkey 8.1.9+
- Keycloak
- Vault
- Kubernetes

Подготовка инфраструктуры

1. Необходимо в БД, к которой будет осуществляется подключение, настроить схему
   `notifications`
   CREATE SCHEMA notifications;
2. Для redis/valkey завести пользователя notification_app с правами +@all -@admin -@dangerous ~* &*
   ОСОБЕННО ВАЖНО правило &* , оно отвечает за доступ к pub/sub каналам. Без него не будет работать SSE механизм
3. Завести в RabbitMQ пользователя notification_user
4. Завести для Keycloak клиент с client-id svc-notification-orchestrator
5. Заводим в Vault
   секрет в используемом kv-secret-engine-mount-path с путем
   svc/notification-orchestrator

он должен содержать в себе следующие ключи и их значения :

quarkus.datasource.password = *Пароль пользователя базы данных, который будет использоваться для работы *

quarkus.oidc-client.credentials.secret= *секрет Клиента Keycloack из пункта 4"

rabbitmq-password = *Пароль пользователя RabbitMQ из пункта 3"

redis.password = *Пароль пользователя Redis из пукта 2*

6. убедиться в наличии следующих конфигураций в кластере k8s на namespace, где будет развернут сервис

common-database-cfg,
common-keycloak-cfg,
common-logging-cfg,
common-rabbit-cfg,
common-vault-cfg,\

7. Завести конфигурацию в кластере k8s svc-notification-orchestrator-cfg на namespace, где будет развернут сервис
   Конфигурация должна содержать

#keycloak
quarkus.oidc-client.auth-server-url=*АДРЕС СЕРВЕРА KEYCLOAK*:*Порт для подключения к серверу KEYCLOAK*/realms/*НАЗВАНИЕ РЕАЛМА системы(в том же реалме должен быть пользователь svc-notification-orchestrator)*
quarkus.oidc.auth-server-url=*АДРЕС СЕРВЕРА KEYCLOAK*:*Порт для подключения к серверу KEYCLOAK*/realms/*НАЗВАНИЕ РЕАЛМА системы(в том же реалме должен быть пользователь svc-notification-orchestrator)*
quarkus.oidc.client-id=svc-notification-orchestrator
quarkus.oidc.application-type=service
quarkus.oidc-client.client-id=svc-notification-orchestrator
quarkus.oidc-client.grant.type=client
quarkus.oidc.tls.tls-configuration-name=oidc-dev
quarkus.tls.oidc-dev.trust-all=true

#rabbitmq
rabbitmq-username=notification_user
#reddis
quarkus.redis.hosts=redis://notification_app:${redis.password}@*АДРЕС СЕРВЕРА REDIS*:*Порт для подключения(по умолчанию- 6379)*

quarkus.datasource.username=*username пользователя для работы с БД*

8. Развернуть сервис

##Подключение библиотеки lib.notifications.backend

1. добавление зависимости

```
<dependency>
    <groupId>com.bivgroup.lib</groupId>
    <artifactId>notifications-rabbit</artifactId>
    <version>*Версия зависимости*</version>
</dependency>
```

2. в application properties добавить
   в quarkus.kubernetes-config.config-maps
   должна быть следующая конфигурация

common-rabbit-notifications-cfg

Создание сервиса- отправителя

Для начала необходимо покдлючить адаптер для необходимого брокера(сейчас доступен RMQ)

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

```
## использовать конфиг k8s
common-rabbit-notifications-cfg,\

# RabbitMQ connector
mp.messaging.outgoing.notifications-sse.connector=smallrye-rabbitmq

## Чтобы при возникновение ошибки отправки не выкидывать ошибки
lib.notifications.producer.bypass-mode=true
```

В модуле, из которого будет производится отправка уведомления сделать инжект

```
@Inject
NotificationService notificationService;
```

На данный момент для отправки уведомления конкретному пользователю, если известен только userLogin необходимо подключать дополнительно

```
<dependency>
     <groupId>io.quarkus</groupId>
     <artifactId>quarkus-keycloak-admin-rest-client</artifactId>
</dependency>
```

добавить application.properties

```
quarkus.keycloak.admin-client.enabled=true
quarkus.keycloak.admin-client.server-url=http://{ip кейклоака}:8080
quarkus.keycloak.admin-client.client-id={client id}
quarkus.keycloak.admin-client.realm={realm}
quarkus.keycloak.admin-client.client-secret={клиент секрет}
quarkus.keycloak.admin-client.username= # remove default username
quarkus.keycloak.admin-client.password= # remove default password
quarkus.keycloak.admin-client.grant-type=client_credentials
```

и сделать инжект

```
@Inject
Keycloak keycloak;
```

Пример рабочего сценария ценарий (реализация https://gitlab.bivgroup.ru/biv-platform-sandbox/svc.notification-sandbox)

```
//получение пользователя из keycloak
UserRepresentation user;
        try {
            user = keycloak.realm(realmName).users().search(notification.getUs
        } catch (NoSuchElementException e) {
            throw new HandledServiceException("user does not exists");
        }

        NotificationPayload payload = notification.getPayload();



//Создание builder уведомления
NotificationService.NotificationBuilder builder = notificationService.notifica
                .userId(UUID.fromString(user.getId())) //UIID пользователя из
                .title(payload.getTitle()) //Заголовок уведомления
                .message(payload.getMessage()) //Текст сообщеения
                .category(NotificationCategory.valueOf(notification.getCategor
                .isPinned(notification.getIsPinned() != null ? notification.ge
                .priority(NotificationPriority.valueOf(notification.getPriorit
                // Приоритет уведомления CRITICAL/HIGH/NORMAL/LOW (на данный м

//Добавление действий через builder
if(payload.getActions() != null) {
   for (NotificationAction bean : notification.getPayload().getActions()) {
     builder.addAction(bean);
    }
}

                //Добавление параметров через builder
if (payload.getParams() != null) {
   for (Map.Entry<String, String> entry : payload.getParams().entrySet()) {
       builder.param(entry.getKey(), entry.getValue());
    }
}
builder.send();
```

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

Описание

Описание

Сервис уведомлений предназначен для централизованного сбора уведомлений от микросервисов и их отображения в веб-интерфейсе.

Данная Front-End библиотека предоставляет Angular-интерфейс для отображения уведомлений, получаемых от Back-End сервиса.

Репозитории и артефакты

- Общий репозиторий сервиса уведомлений
- Репозиторий библиотеки
- Репозиторий web-devkit-обертки для разработки
- Пакеты библиотеки в Nexus

Порядок сборки библиотеки и публикации пакета в Nexus описан в инструкции по release.

Подключение библиотеки к проекту

Библиотека поставляется через клиентские хранилища пакетов, например Sonatype Nexus Repository , в виде npm-пакета.

1. Добавление пакета в `package.json`

Добавьте зависимость:

```
"@biv/notifications": "^20.0.0"
```

2. Подключение провайдера в конфиге приложения

Импортируйте `provideWebNotificationsConfig` :

```
import { provideWebNotificationsConfig } from '@biv/notifications';
```

Затем зарегистрируйте провайдер в конфигурации приложения, например в `app.config.ts` или `angular.config.ts` :

```
provideWebNotificationsConfig({
  apiBaseUrl: '/api',
  useMocks: true,
}),
```

Параметры конфигурации

- `apiBaseUrl` — URL Back-End сервиса уведомлений
- `useMocks` — включает режим работы на моках для тестирования без Back-End

3. Подключение assets

Библиотека использует собственные элементы дизайна: иконки, изображения и шрифты.

Добавьте assets в `angular.json` по пути:

`projects -> <имя проекта> -> architect -> build -> options -> assets`

```
{
  "glob": "**/*",
  "input": "node_modules/@biv/notifications/assets",
  "output": "/assets/@biv/notifications"
}
```

Использование библиотеки в компонентах проекта

Список экспортируемых сущностей библиотеки доступен в public-api.

Ниже приведен пример подключения библиотеки в standalone-компоненте.

Импорты

```
import { CommonModule } from '@angular/common';
import {
  NotificationBellButtonComponent,
  NotificationItem,
  NotificationListActionEvent,
  NotificationListContainerComponent,
  NotificationRealtimeToastComponent,
  NotificationsRealtimeService,
  NotificationsStoreService,
  NotificationsStreamEvent,
} from '@biv/notifications';
```

Подключение в `imports`

```
standalone: true,
imports: [
  CommonModule,
  NotificationListContainerComponent,
  NotificationBellButtonComponent,
  NotificationRealtimeToastComponent,
],
```

Пример минимального подключения

```
import { CommonModule } from '@angular/common';
import { Component } from '@angular/core';
import {
  NotificationBellButtonComponent,
  NotificationListContainerComponent,
  NotificationRealtimeToastComponent,
} from '@biv/notifications';

@Component({
  selector: 'app-notifications-demo',
  standalone: true,
  imports: [
    CommonModule,
    NotificationListContainerComponent,
    NotificationBellButtonComponent,
    NotificationRealtimeToastComponent,
  ],
  template: `
    <biv-notification-bell-button />
    <biv-notification-list-container />
    <biv-notification-realtime-toast />
  `,
})
export class NotificationsDemoComponent {}
```

Краткая схема подключения

1. Установите пакет `@biv/notifications`
2. Подключите `provideWebNotificationsConfig(...)` в конфигурации приложения
3. Добавьте assets библиотеки в `angular.json`
4. Импортируйте необходимые компоненты и сервисы в нужный standalone-компонент или модуль

Полезные ссылки

- Сервис уведомлений
- Front-End библиотека
- Back-End сервис
- Инструкция по release
- Public API библиотеки

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

Стилизация

Стилизация

Для стилизации компонентов библиотеки используются CSS-переменные. Такой подход позволяет гибко переопределять внешний вид компонентов на стороне клиентского приложения без изменения исходного кода библиотеки.

Пример использования CSS-переменных

```
&__badge {
  position: absolute;
  top: var(--web-notifications-bell-badge-top, -3px);
  right: var(--web-notifications-bell-badge-right, -6px);
  min-width: var(--web-notifications-bell-badge-min-width, 16px);
  height: var(--web-notifications-bell-badge-height, 14px);
  padding: var(--web-notifications-bell-badge-padding, 2px 3px 2px 2px);
  border-radius: var(--web-notifications-bell-badge-radius, 12px);
  background: var(--web-notifications-bell-badge-background, #ff4d4f);
  color: var(--web-notifications-bell-badge-color, #ffffff);
  border: var(--web-notifications-bell-badge-border-width, 1px) solid
    var(--web-notifications-bell-badge-border-color, #ff4d4f);
  box-sizing: border-box;
  display: inline-flex;
  align-items: center;
  justify-content: center;
  white-space: nowrap;
  line-height: 1;
  font-size: var(--web-notifications-bell-badge-font-size, 12px);
  pointer-events: none;
}
```

Темы с CSS-переменными

CSS-переменные сгруппированы по тематическим файлам.

`./notification-list.theme`

Содержит переменные для стилизации компонентов, связанных со списком уведомлений:

- панель уведомлений
- элементы списка

- состояние прочитанных и непрочитанных уведомлений
- закрепленные уведомления
- детализация уведомления

`./notification-bell.theme`

Содержит переменные для стилизации иконки уведомлений:

- кнопка с иконкой колокольчика
- счетчик непрочитанных уведомлений
- позиционирование и размеры badge
- цвета и типографика индикатора

Глобальное подключение тем в библиотеке

Темы с CSS-переменными подключаются в библиотеке глобально:

```
@use './notification-list.theme' as list-theme;
@use './notification-bell.theme' as bell-theme;

:root {
  @include list-theme.web-notifications-css-vars();
  @include bell-theme.web-notification-bell-css-vars();
}
```

Это означает, что базовые значения переменных доступны глобально и применяются ко всем компонентам библиотеки по умолчанию.

Кастомизация стилей в клиентском проекте

Для легкой кастомизации большинства стилей достаточно переопределить нужные CSS-переменные в своем проекте.

Например, это можно сделать глобально в `styles.scss` или в другой общей точке входа стилей приложения.

Пример глобального переопределения CSS-переменных

```
:root {
  --web-notifications-panel-width: 420px;
  --web-notifications-panel-height: 100vh;

```

```
  --web-notifications-color-primary: #1677ff;
  --web-notifications-color-primary-hover: #4096ff;

  --web-notifications-color-unread-bg: #edf7ff;
  --web-notifications-color-pinned-bg: #fff7d6;

  --web-notifications-color-text-primary: rgba(0, 0, 0, 0.92);
}
```

Рекомендации по кастомизации

- Для изменения внешнего вида библиотеки предпочтительно использовать переопределение CSS-переменных.
- Такой способ упрощает поддержку библиотеки и снижает необходимость переопределять внутренние классы компонентов.
- Глобальное переопределение через `:root` подходит для единообразной стилизации во всем приложении.
- При необходимости можно задавать переменные и локально, если требуется кастомизация только в рамках отдельного контейнера.

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

История версий

20.0.0

- Версия Angular: `20.1.0`

Что реализовано

1. Отображение списка уведомлений пользователя

Реализован интерфейс для просмотра и обработки пользовательских уведомлений.

Доступная функциональность

- прочтение уведомления
- закрепление и открепление уведомления
- отметка всех уведомлений как прочитанных
- удаление уведомления (архивирование)
- сортировка уведомлений по статусу прочтения
- открытие полного уведомления с детализацией

2. Сервис получения real-time уведомлений

Реализован сервис для работы с уведомлениями в режиме реального времени.

Доступная функциональность

- подключение к сервису уведомлений
- авторизация для работы с real-time потоком
- показ всплывающих баннеров при получении новых уведомлений

3. Иконка уведомлений для перехода к списку

Реализован bell-компонент для быстрого перехода к списку уведомлений.

Доступная функциональность

- отображение количества непрочитанных уведомлений

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

Релизы библиотеки @biv/notifications

Работа с ветками и оформление changelog

Для начала нужно создать ветку с номером задачи из Jira. Например BIVPL-XXX/new-feature



Рабочая ветка создается от мастер ветки. На 02.04.2026 мастер веткой является v.20.X.X

После реализации задачи необходимо также в файле `VERSIONS.md` описать сделанные изменения.
В `package.json` библиотеки необходимо поменять версию:

```
{
  "name": "@biv/notifications",
  "version": "20.0.0",
    ....
}
```

Все изменения закомитить и запушить в рабочую ветку.

PR и выпуск релиз кандидата (RC)

После того как сделаны комиты по вашей задаче в вашу рабочую ветку BIVPL-XXX/new-feature необходимо создать релизную ветку с названием release/v.X.X.X, где X.X.X - новый номер версии, готовой к выпуску, например, release/v.20.1.3



Релизная ветка создается от мастер ветки. На 02.04.2026 мастер веткой является v.20.X.X

После создания ветки делаем PR ветки BIVPL-XXX/new-feature в release/v.X.X.X.
После полученного апрува должна автоматически запуститься сборка

Если деплой прошел успешно можно зайти в job "publish-rc" и посмотреть номер выпущенного Релиз кандидата(RC). Например для релизной ветки release/v.20.0.0 job с RC версией можно посмотреть тут

В самом низу лога будет запись:

```
@biv/notifications@20.0.0-rc.0
```

Тестирование релиз кандидата (RC)

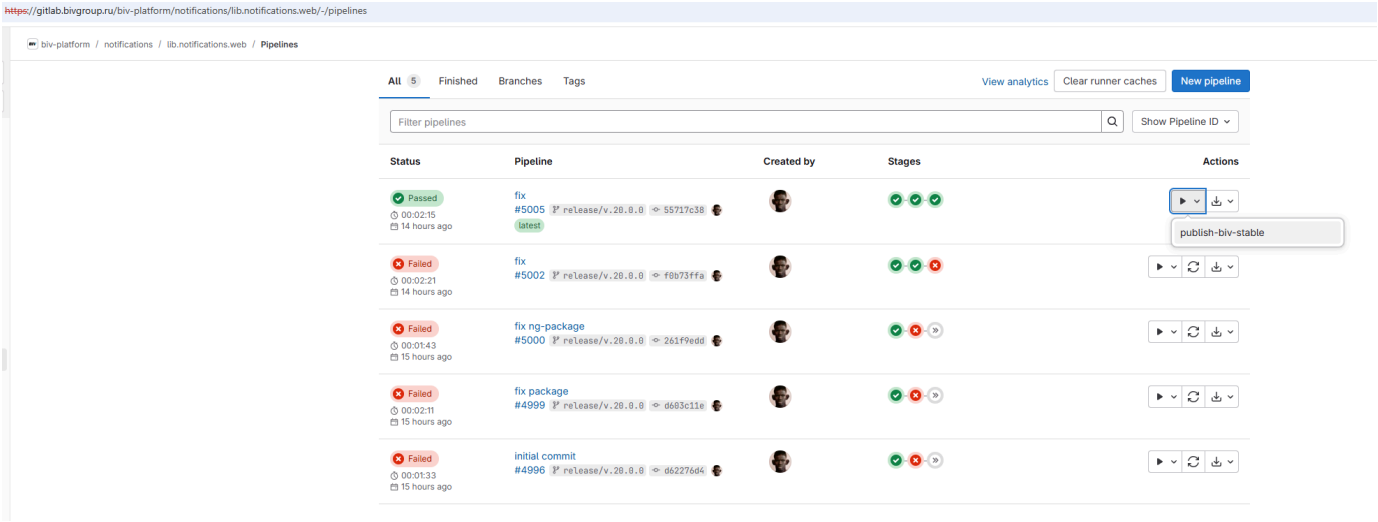
Если вы используете непосредственно библиотеку `@biv/notifications` , то в том проекте, в котором ее используете обновляете версию и смотрите примененные изменения:

```
$ npm i @biv/notifications@20.0.0-rc.0
```

Если вы хотите собрать уведомления `@biv/notifications` без подключения к проекту, то тогда нужно развернуть окружение для разработки уведомлени

Выпуск стабильной версии

После тестирования релиз-кандидата заходим в последнюю сборку релиз кандидата. Сбоку кликаем по кнопке "play" и выбираем сборку для выпуска стабильной версии `publish-biv-stable` .



PR в мастер-ветку

После успешного выпуска версии делаем PR релизной ветки в мастер-ветку (v.20.x.x)

Обновить историю версий в wiki

После релиза добавить в раздел История версий обновления, которые вышли в новой версии.

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

Компоненты билиотеки

Компоненты и сущности библиотеки

Ниже приведен список основных сущностей библиотеки, доступных для импорта через `public-api`.

Экспортируемые файлы

```
// Модели
export * from './lib/models/notifications.models';
export * from './lib/models/notification-list-ui.models';

export * from './lib/mocks/notifications.mock';

// Сервисы
export * from './lib/services/notifications.service';
export * from './lib/services/notifications-realtime.service';
export * from './lib/store/notifications-store.service';
export * from './lib/tokens/web-notifications-config.token';

// Список уведомлений
export * from './lib/components/notification-list-container/notification-list-
export * from './lib/components/notification-realtime-toast/notification-realt
// Bell
export * from './lib/components/notification-bell-button/notification-bell-but
```

Описание экспортируемых сущностей

Модели

`./lib/models/notifications.models`

Общие модели библиотеки, используемые для работы с уведомлениями, их статусами, действиями и данными, получаемыми от Back-End сервиса.

`./lib/models/notification-list-ui.models`

UI-модели для отображения списка уведомлений и страницы детализации уведомления.
Используются для подготовки и представления данных в компонентах библиотеки.

`./lib/mocks/notifications.mock`

Набор моков для работы библиотеки без Back-End сервиса.
Позволяет тестировать отображение уведомлений, пользовательские действия и realtime-сценарии в изолированном режиме.

Сервисы

`./lib/services/notifications.service`

Основной сервис для работы со списком уведомлений.

Методы

- `getNotifications` — получение списка уведомлений
- `getNotificationById` — получение уведомления по `id`
- `updateNotificationStatus` — обновление статуса уведомления
- `markAllAsRead` — отметить все уведомления как прочитанные
- `updateNotificationPin` — закрепить или открепить уведомление
- `executeNotificationAction` — обработка пользовательского действия по уведомлению

`./lib/services/notifications-realtime.service`

Сервис для получения уведомлений в realtime-режиме через Server-Sent Events (SSE).

Методы

`authorizeStream`

Метод передачи токена авторизации перед открытием SSE-подключения.

Во время вызова токен пользователя передается в сервис уведомлений и сохраняется в `httpOnly cookie` . Это позволяет затем открыть SSE-подключение без передачи токена в заголовках запроса.

Конечная точка: `GET /notifications/stream-auth`
Назначение: подготовка авторизации для дальнейшей работы с SSE.

`connectStream`

Метод установления Server-Sent Events подключения для получения уведомлений в реальном времени.

Конечная точка: `GET /notifications/stream`
Назначение: открытие realtime-канала для получения входящих уведомлений.

`./lib/store/notifications-store.service`

Сервис клиентского состояния библиотеки.

Используется для хранения и управления состоянием уведомлений на стороне приложения: списка уведомлений, количества непрочитанных, закрепленных элементов, выбранного уведомления и других данных, необходимых для работы UI.

`./lib/tokens/web-notifications-config.token`

Токен конфигурации библиотеки.

Используется для передачи пользовательских настроек библиотеки через `provideWebNotificationsConfig(...)`, например:

- `apiBaseUrl`
- `useMocks`

Компоненты

`./lib/components/notification-list-container/notification-list-container.component`

Панель со списком уведомлений.

Компонент отвечает за отображение списка уведомлений, фильтрацию, навигацию по элементам списка, открытие детализации и пользовательские действия над уведомлениями.

`./lib/components/notification-realtime-toast/notification-realtime-toast.component`

Компонент всплывающих уведомлений (`toast`).

Используется для отображения realtime-уведомлений, поступающих через SSE, в виде всплывающих сообщений поверх интерфейса.

`./lib/components/notification-bell-button/notification-bell-button.component`

Компонент кнопки уведомлений в виде иконки колокольчика.

Отображает текущее количество непрочитанных уведомлений и используется как точка входа для открытия панели уведомлений.

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

Окружение для разработки

Клонирование проекта

Окружение для разработки находится тут
В проекте используются `git submodule`

Для клонирования проекта необходимо выполнить команду

```
$ git clone --recurse-submodules -b my-favorite-branch ssh://git@gitlab.bivgro
```

Необходимо заменить `my-favorite-branch` на нужную Вам ветку

Данная команда склонирует проект вместе со всеми зависимостями.

Настройка и запуск

1. Устанавливаем npm-пакеты

```
$ npm i
```

2. Затянуть последний комит библиотеки

```
git submodule update --remote projects/biv/notifications
```

3. Сборка библиотеки

```
npx ng build @biv/notifications
$ npm run start
```

4. Запускаем сам проект в режиме разработки

```
$ npm run start
```

5. Открываем http://localhost:4200

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