SQUADUS

# Руководство по работе с BotSDK

**ООО «НОВЫЕ ОБЛАЧНЫЕ ТЕХНОЛОГИИ»**

# ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

# SQUADUS

# РУКОВОДСТВО ПО РАБОТЕ С BOTSDK

**1.4**

**На 18 листах**

**Москва**
**2024**

Все упомянутые в этом документе названия продуктов, логотипы, торговые марки и товарные знаки принадлежат их владельцам.

Товарные знаки «МойОфис» и «MyOffice» принадлежат ООО «НОВЫЕ ОБЛАЧНЫЕ ТЕХНОЛОГИИ».

Ни при каких обстоятельствах нельзя истолковывать любое содержимое настоящего документа как прямое или косвенное предоставление лицензии или права на использование товарных знаков, логотипов или знаков обслуживания, приведенных в нем. Любое несанкционированное использование этих товарных знаков, логотипов или знаков обслуживания без письменного разрешения их правообладателя строго запрещено.

**СОДЕРЖАНИЕ**



 

## ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем документе применяют следующие сокращения с соответствующими расшифровками (см. Таблица 1):

Таблица 1 – Сокращения и расшифровки

| Сокращение | Расшифровка |
|---|---|
| ОС | Операционная система |
| ПО | Программное обеспечение |

# 1 ОБЩИЕ СВЕДЕНИЯ

## 1.1 Назначение

Squadus — безопасный корпоративный мессенджер с поддержкой видеоконференций, глубокой интеграцией с продуктами «МойОфис» и возможностью интеграции с внешними информационными системами.

В состав продукта входят:

- коммуникационная система Squadus — для обмена мгновенными сообщениями, документами и медиафайлами между пользователями и в групповых чатах;
- система видеоконференцсвязи (ВКС) — для организации аудио- и видеозвонков и конференций с возможностью гостевого доступа незарегистрированными пользователями;
- приложения Squadus — для рабочего общения с помощью текстовых, голосовых и видео сообщений, а также участия в конференциях в веб-браузерах и на ОС Windows, Linux, macOS, iOS, Android.

Подробное описание функциональных возможностей ПО Squadus приведено в документе «Squadus. Функциональные возможности».

## 1.2 Назначение botsdk

Пакет @squadus/botsdk содержит HTTP-клиент для осуществления запросов к серверу ПО Squadus. Используется для разработки ботов, работающих с мессенджером Squadus.

## 1.3 Системные требования

Пакет @squadus/botsdk поддерживается Node v14 и выше. Рекомендуется использовать последнюю версию Node.

## 1.4 Установка

Для установки @squadus/botsdk необходимо выполнить команду:

```
$ npm install @squadus/botsdk
```

# 2 ИСПОЛЬЗОВАНИЕ

## 2.1 Инициализация клиента

Пакет @squadus/botsdk экспортирует класс `SquadusClient`. Для использования следует создать экземпляр класса, передав в конструктор: адрес сервера, токен и разрешенный путь до папки с файлами, которые разрешено прикреплять к сообщениям, а затем вызвать connect:

```
import SquadusClient from '@squadus/bot.sdk';

// Создание клиента
const squadusClient = new SquadusClient({
    token: 'a89cDxjI1l'
    server: 'https://im.example.ru/',
});
await squadusClient.connect();
```

Токен генерируется вручную в веб или настольном приложении ПО Squadus. Для этого необходимо выполнить следующие действия (см. Рисунок 1):

1. Войти в учетную запись бота.
2. Перейти на странице **Настройки** в раздел **Токены личного доступа**.
3. На открывшейся странице ввести название токена.
4. Установить флажок **Игнорировать двухфакторную аутентификацию**.
5. Нажать **Добавить**.

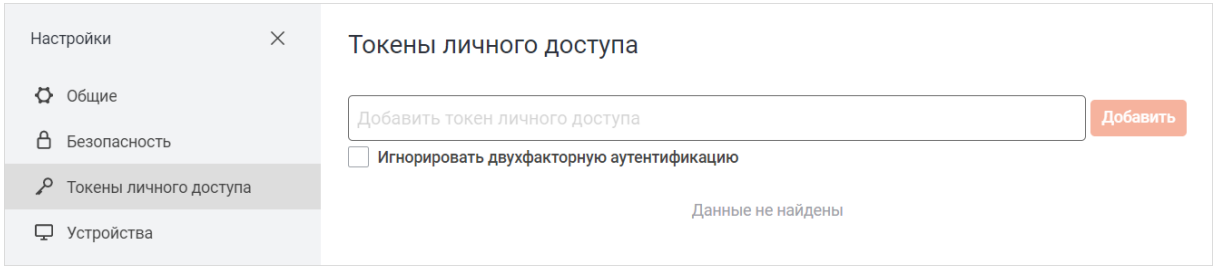


Рисунок 1 — Токены личного доступа

### 2.2 Вызов метода

Каждый из методов `SquadusClient` относится к одной из сущностей:

– room;

– message;

– subscription.

Например, метод `sendMessageByRid` используется для отправки сообщения. Для вызова метода необходимо выполнить следующий запрос:

```
squadusClient.message.sendMessageByRid()
```

Где:

– `squadusClient` — instance класса `SquadusClient`;

– `message` — сущность, которая содержит в себе методы для совершения операций над комнатами;

– `sendMessageByRid` — метод для отправки сообщений.

```
// При известном ID комнаты (канала, команды или личной переписки)
const roomId = '...';

(async () => {
    const result = await squadusClient.message.sendMessageByRid({
        msg: 'Hello world!',
        rid: roomId,
    });

    console.log('Message is sent successfully');
})();
```

Некоторые методы, такие как `connect`, `getSettings`, не относятся ни к одной из сущностей. Такие методы можно вызвать напрямую из instance класса `SquadusClient`.

Для использования имени метода в виде строки необходимо выполнить следующий запрос:

```
squadusClient.restClient.post(im.create, [options])
```

Такая конструкция позволяет:

– динамически определять вызываемый метод;

– вызывать методы, которые недоступны в используемой версии клиента.

Пример использования метода:

```
(async () => {
    // Использование restClient позволяет приложению вызывать любой метод
требующий авторизации
    const response = await squadusClient.restClient.post('im.create', {
        username: 'new_user',
    });
})();
```

 

### 2.3 Обработка ошибок

Ошибки могут возникать по нескольким причинам: например, пользователь не имеет прав вызывать метод или был использован неверный аргумент. В этих случаях возвращенный promise будет отклонен с ошибкой. Полученную ошибку следует обработать для восстановления функций приложения, использующего пакет @squadus/botsdk

Для получения списка возможных ошибок необходимо экспортировать ErrorCode:

```
// Import ErrorCode from the package
import { SquadusClient, ErrorCode } from 'squadus/botsdk';

const { USERNAME, PASSWORD, SERVER } = process.env;

const squadusClient = new SquadusClient({
    userName: USERNAME || '',
    password: PASSWORD || '',
    server: SERVER || '',
});

(async () => {
    try {
        await squadusClient.connect();
    } catch (error) {
        if (error === ErrorCode.AuthorizationError) {
            console.log('Authorization error');
        }
    }
})();
```

### 2.4 Типы ошибок

Типы ошибок ErrorCode представлены в таблице 2.

Таблица 2 — Типы ошибок ErrorCode

| ErrorCode | Описание ошибки | Примечание |
|---|---|---|
| AuthorizationError | Запрос не может быть отправлен из-за ошибки в авторизации | Основные причины – не пройденная авторизация или истекший токен |
| ErrorCode.NoSuchFileOrDirectory | Нет такого файла или директории | Ошибка может возникнуть при отправке файлов, если был неверно передан путь к файлу |
| ErrorCode.EmptyResult | Нет результата отправки | Если сообщение было отправлено с ошибочным id комнаты (rid), то в ответе не будет сообщения. Значит, сообщение не было доставлено |

| ErrorCode | Описание ошибки | Примечание |
|---|---|---|
| ErrorCode.InvalidUser | Ошибочный пользователь | Если производится попытка создать комнату с пользователем, которого нет, комната создана не будет |
| ErrorCode.RoomNotFound | Комната не найдена | Ошибка возникает, если при добавлении или удалении пользователя используется неверный rid |
| ErrorCode.CommonError | Общая ошибка | Если не было установлено причин ошибки, будет возвращена CommonError ошибка |
| ErrorCode.ChannelNameExists | Канал с указанным названием уже существует | - |
| ErrorCode.FileToLarge | Отправляемый файл превышает допустимый размер | - |
| ErrorCode.MessageSizeExceeded | Количество символов в отправляемом сообщении превышает установленное ограничение на сервере | - |

### 2.5 Методы API

В таблице 3 приведено описание методов API, их аргументов и возвращаемых ими значений.

Таблица 3 – Методы API

| Метод | Группа | Аргумент | Ответ | Описание |
|---|---|---|---|---|
| connect() | – | – | Promise<void> | Выполняет авторизацию |
| getSettings (settings) | _ | settings?: Array<SettingsName> | RestResponse <SettingsResponseData> | Возвращает значения настроек, имена которых были переданы в метод. Если не передавать имена, будут возвращены все настройки |
| sendMessageByRid (params) | message | params: SendMessageByRidParams | Promise <MessageResponseData> | Выполняет отправку сообщения по идентификатору комнаты |
| sendAttachment (params) | message | params: SendAttachmentSDKParams | Promise <MessageResponseData> | Выполняет отправку вложения по идентификатору комнаты |
| createDirectRoom (username) | room | username: string | Promise <DirectRoomResponseData \| undefined> | Создает личную переписку с пользователем, имя которого было передано в качестве аргумента |
| addUsersToRoom (params) | room | params: AddUsersToRoomParams | RestResponse <boolean> | Добавляет пользователя в существующую комнату |
| removeUserFromChannel (params) | room | params: RemoveUserFromChannelParams | RestResponse <RemoveUserFromChannelResponseData> | Удаляет пользователя из комнаты, в которой этот пользователь состоит |

| Метод | Группа | Аргумент | Ответ | Описание |
|---|---|---|---|---|
| createChannel (params) | room | params: CreateChannelSDKParams | RestResponse <ChannelData> | Создает комнату с переданными параметрами |
| setUserRole (params) | room | params: CreatePublicChannelSDKParams \| CreatePrivateChannelSDKParams | RestResponse <CreatePublicChannelResponseData \| CreatePrivateChannelResponseData> | Изменяет роль пользователя в комнате |
| saveRoomSettings (params) | room | params: SaveRoomSettingsSDKParams | RestResponse <SaveRoomSettingsData> | Изменяет настройки комнаты по ее идентификатору |
| onMessage (callback[, rid]) | subscription | callback: (msg: LastMessage) => void, rid?: string | Promise <OnMessagesResponse> | Вызывает переданную функцию с новым полученным сообщением в определенном чате rid или во всех |
| onRoomChange (rid, callback) | subscription | rid: string, callback: (data: RoomEvent) => void | Promise <OnRoomChangeResponse> | Вызывает переданную функцию с каждым событием в чате |
| onReaction (rid, msgId, callback) | subscription | rid: string, msgId: string, callback: (message: Message) => void | Promise <onReactionResponse> | Вызывает переданную функцию с сообщением при изменении его реакций |
| onEvent (callback) | subscription | callback: (data: WsData) => void | Subscription | Вызывает переданную функцию со всеми событиями, полученными по протоколу WebSocket |
| subscribe (collection, event) | subscription | collection: Collections, event: string | Promise<WsSubscription> | Позволяет произвольно подписаться на события, |

| Метод | Группа | Аргумент | Ответ | Описание |
|---|---|---|---|---|
| | | | | которые получены через onEvent |
| close() | subscription | – | Promise<void> | Закрывает соединение |
| reopen() | subscription | – | Promise<void> | Переоткрывает соединение |
| subscribeNotifyUser() | subscription | – | Promise <(WsSubscription)[]> | Осуществляет подписку на WebSocket-события, связанные с текущим пользователем |
| subscribeLoggedNotify() | subscription | – | Promise <(WsSubscription)[]> | Осуществляет подписку на WebSocket-события об изменениях авторизованных пользователей |
| subscribeNotifyAll() | subscription | – | Promise <(WsSubscription)[]> | Осуществляет подписку на общие для всех пользователей WebSocket-события |
| unsubscribe(id) | subscription | id: string | Promise <UnsubscribeResponseData> | Метод позволяет отписываться от WebSocket-событий по их id, который можно получить при подписке |
| getUserInfoByUsername (username) | – | username: string | RestResponse <UserResponseData> | Метод запрашивает информацию о пользователе по его имени |
| sendMessageToThread (params) | message | params: SendMessageToThreadSDKParams | Promise <MessageResponseData> | Отправляет сообщение в цепочку ответов |

| Метод | Группа | Аргумент | Ответ | Описание |
|---|---|---|---|---|
| readThread (parentMessageId) | room | parentMessageId: string | Promise<void> | Помечает все сообщения внутри цепочки ответов с переданным parentMessageId как прочитанные |

### 2.6 Примеры

В корне пакета `@squadus/botsdk` есть папка `examples` – это проект с несколькими примерами использования методов из таблицы 3. С помощью приведенных примеров можно ознакомиться с использованием данного пакета на практике.

Для выполнения кода из папки `examples`, необходимо:

1. Установить зависимости из папки examples: cd examples && yarn.
2. Внести токен пользователя (бота), от чьего имени будут осуществляться действия, адрес сервера и имя собеседника в файл `./examples/.env`.

```
SERVER="https://******"
USERNAME="******"
PASSWORD="******"
```

3. Запустите пример yarn create-direct. Если токен и адрес сервера указаны верно, а также пользователь, с которым создается диалог, существует, в консоль будет выведено сообщение Direct dialog with ${partner_name} was created successfully.

Примеры, которые помогут начать пользоваться @squadus/botsdk представлены в таблице 4.

Таблица 4 — Примеры методов

| Файл с примером скрипта | Команда | Описание работы команды |
|---|---|---|
| connect.ts | yarn connect | Будет выполнена авторизация |
| getSettings.ts | yarn get-settings | Будут получены значения настроек с сервера для `Accounts_AllowRealNameChange` и `Accounts_AddGuestsToChats` |
| createDirectRoom.ts | yarn create-direct | Будет создан диалог с собеседником, указанным в файле .env, если такой пользователь существует |
| sendMessage.ts | yarn send-message | С собеседником, указанным в файле .env (если такой пользователь существует), будет создан диалог. Затем ему будет отправлено сообщение с текстом «Hi» |
| sendAttachment.ts | yarn send-attachment | С собеседником, указанным в файле .env, если такой пользователь существует, будет создан диалог. Затем ему будет отправлено изображение example.png из проекта с сообщением «Image with logo». Если изображение будет заменено на файл с размером, превышающим допустимый, в консоли появится сообщение об ошибке: «File is too large». Также файл должен находиться по пути, который соответствует параметру `allowedAttachmentsPath`, указанному при создании `squadusClient` |

| Файл с примером скрипта | Команда | Описание работы команды |
|---|---|---|
| addRemoveChannelUser.ts | yarn add-remove-channel-user | Собеседник, указанный в файле .env, если такой пользователь существует, будет добавлен в указанный канал. Затем он будет удален из него |
| createPrivateChannel.ts | yarn create-channel:private | Будет создан приватный канал с именем `PRIVATE_CHANNEL_NAME` |
| createPrivateChannel.ts | yarn create-channel:public | Будет создан открытый канал с именем, указанным в константе `PUBLIC_CHANNEL_NAME` |
| changeUserRole.ts | yarn change-role | Будет создан открытый канал с пользователем. Затем пользователю будет назначена роль Модератор |
| editChannel.ts | yarn edit-channel | Будет создан канал, затем его имя будет изменено |
| subscribeMessages.ts | yarn subscribe-messages | Ботом будет выполнена подписка на все новые сообщения, а также отдельная подписка на новые личные сообщения от пользователя, имя которого указано в переменной окружения PARTNER_NAME. При получении ботом сообщения «Hi» от пользователя PARTNER_NAME, будет отправлен ответ «Hello». При получении ботом сообщения «Bye» из любого чата, будет отправлен ответ «Bye». Через 20 секунд произойдет отписка от сообщений «Hi», через 40 секунд произойдет отписка от сообщений «Bye» |
| subscribeReaction.ts | yarn subscribe-reaction | Ботом будет выполнена подписка на все новые сообщения. После получения ботом сообщения «start», бот отправит ответное сообщение, на реакции которого произойдет подписка ботом. Если поставить реакцию с эмодзи 👏 на это сообщение любым пользователем, в тот же чат бот отправит сообщение, содержащее эмоджи 🏔, если будет реакция с 😀, то сообщение с эмоджи ⚽. После отправки ботом сообщения на реакцию, бот отпишется от реакций на это сообщение |
| subscribeRoom.ts | yarn subscribe-room | Ботом будет выполнена подписка на события в личных сообщениях с пользователем, имя которого указано в |

| Файл с примером скрипта | Команда | Описание работы команды |
|---|---|---|
| | | переменной окружения `PARTNER_NAME`. Если пользователь `PARTNER_NAME` или сам бот начал или прекратил печатать сообщение в этом чате, в консоль будет выведено сообщение об этом |
| sendMessageToThread.ts | yarn send-message:thread | Будет отправлено сообщение, на основании идентификатора которого будет создана цепочка ответов |
| readThread.ts | yarn read-thread | Будет отправлено сообщение, на основании идентификатора которого будет создана цепочка ответов. Цепочка ответов будет отмечена как прочитанная пользователем, имя и токен которого указаны в переменных окружения `PARTNER_NAME` и `PARTNER_TOKEN` соответственно |

## 3 ТЕХНИЧЕСКАЯ ПОДДЕРЖКА

Контактная информация службы технической поддержки ООО ««Новые облачные технологии» в случае возникновения вопросов, не описанных в данном руководстве:

Адрес электронной почты: support@service.myoffice.ru. Телефон: 8-800-222-1-888.