[ГХ] 6.2.3 Конфигурации Вконтакте

Оглавление

Фрагмент файла конфигурации

```
...
"integrations": {
...
    "vk": {
        "url": "https://api.vk.com/method",
        "clientId": "12345678",
        "clientSecret": "werdfghjk12jmnbv3456",
        "scope": "wall",
        "redirect_uri": "https://tripwithme.com",
        "timeout": 10000,
        "retryPolicy": {
            "retries": 2,
            "backoffMs": 500
        }
    },
...
```

Описание параметров

Параметр	Описание	Влияние на функциональность
`integrations.vk.url`	Базовый URL API VK	Куда идут запросы для публикации, редактирования, удаления постов
`integrations.vk.clientId`	Идентификатор приложения	Используются для авторизации через OAuth 2.0 во внешнюю систему
`integrations.vk.clientSecret`	Секретный ключ	
`integrations.vk.scope`	Права доступа, которые необходимы приложению TWM	
`integrations.vk.redirect_uri`	Адрес для возврата в приложение TWM	
`integrations.vk.timeout`	Таймаут HTTP-запросов (в миллисекундах)	Обеспечивает стабильную работу при задержках со стороны внешней системы

`integrations.vk.retryPolicy`	Политика повтора запросов	Обеспечивает выполнение повторных запросов при временных ошибках со стороны внешней системы
`integrations.vk.retryPolicy.retries`	Количество повторных запросов	
`integrations.vk.retryPolicy.backoffMs`	Задержка между повторными запросами (в миллисекундах)	

[ГХ] 6.2.4 Аутентификация и авторизация для Web во Вконтакте

Оглавление

Общее описание

Тип авторизации: OAuth 2.0

Access токен:

- Время жизни: 1 час
- Хранение: в кэше Redis

Refresh токен:

- Время жизни: 180 дней
- Хранение: в таблице refresh_tokens (БД PostgreSQL)

Настройка приложения в документации VK ID: Этап 1. Создание и настройка приложения

Первичная аутентификация и авторизация

Получение Access токена в обмен на код авторизации

Предусловия

1. Параметры веб приложения TWM сохранены в файл конфигураций
   a. `integrations.vk.client_id`
   b. `integrations.vk.scope`
   c. `integrations.vk.redirect_uri`
2. Нет сохраненного Access токена в кэше Redis
3. Нет сохраненного Refresh токена в таблице refresh_tokens
4. Пользователь инициирует процесс подключения учетки VK в настройках приложения TWM

Алгоритм работы

1. **Frontend TWM** отправляет запрос подключение учетки VK на **Backend TWM**
2. **Backend TWM** получает параметры из файла конфигураций среды выполнения:

a. `client_id` - идентификатор приложения
b. `scope` - права доступа, которые необходимы приложению TWM
c. `redirect_uri` - адрес для возврата в веб приложение TWM
   - Если параметры отсутствуют или недействительны:
     - Вернуть пользователю сообщение: **“Извините, произошла непредвиденная ошибка. Попробуйте позднее или свяжитесь с технической поддержкой.”**
     - Зафиксировать событие в журнале ошибок и отправить уведомление в систему мониторинга

3. **Backend TWM** генерирует набор параметров PKCE, необходимых для защиты передаваемых данных, а также параметр state:
   a. `code_verifier` - случайно сгенерированная строка, новая на каждый запрос авторизации; может состоять из следующих символов: `a-z`, `A-Z`, `0-9`, `_`, `-`; длина от 43 до 128 символов
   b. `code_challenge` - значение code_verifier, преобразованное с помощью code_challenge_method и закодированное в base64
   c. `code_challenge_method=S256` - метод преобразования code_verifier в code_challenge, константа
   d. `state` - произвольная строка состояния приложения
4. **Backend TWM** формирует ссылку, используя параметры, и отправляет её на **Frontend TWM**:
   a. `response_type=code` - требуемый ответ, константа
   b. `client_id` - идентификатор приложения
   c. `scope` - права доступа, которые необходимы приложению TWM
   d. `redirect_uri` - адрес для возврата в веб приложение TWM
   e. `state` - произвольная строка состояния приложения
   f. `code_challenge` - значение code_verifier, преобразованное с помощью code_challenge_method и закодированное в base64
   g. `code_challenge_method=S256` - метод преобразования code_verifier в code_challenge, константа

Пример ссылки:

```
https://id.vk.com/authorize?response_type=code
&client_id=12345678
&scope=wall
&redirect_uri=https%3A%2F%2Ftripwithme.com
&state=<Значение>
&code_challenge=<Значение>
&code_challenge_method=S256
```

5. **Frontend TWM** открывает ссылку в рамках одной вкладки браузера

Код для запуска авторизации в рамках одной вкладки:

```
location.assign('https://id.vk.com/authorize' + query);
```

6. На стороне **VK ID** проходит алгоритм:
   a. Пользователь проходит аутентификацию
   b. Пользователю отображается форма с разрешением на предоставление доступов приложению TWM, и он разрешает доступы
   c. Запрос на предоставление доступов приложению TWM передается в VK ID
   d. VK ID генерирует код авторизации и выполняет редирект по указанному адресу TWM с параметрами:
      - `state` - произвольная строка состояния приложения, сгенерированная в шаге 3
      - `code` - код авторизации
      - `device_id` - уникальный идентификатор устройства
7. **Frontend TWM** передает `state`, `code`, `device_id` на **Backend TWM**

8. **Backend TWM** проверяет значение полученного параметра state со значением, сгенерированным в шаге 3
    - Если значение полученного параметра state не совпадает со значением, сгенерированным в шаге 3:
        - Вернуть пользователю сообщение: **“Извините, произошла непредвиденная ошибка. Попробуйте позднее или свяжитесь с технической поддержкой.”**
        - Зафиксировать событие в журнале ошибок и отправить уведомление в систему мониторинга
9. **Backend TWM** отправляет запрос в **API VK ID**, чтобы обменять авторизационный код на токены, используя параметры:
    a. `client_id` - идентификатор приложения
    b. `grant_type=authorization_code` - параметр, который указывает, какой тип grant используется для получения токена, константа
    c. `code_verifier` - случайно сгенерированная строка, сгенерированная в шаге 3
    d. `device_id` - уникальный идентификатор устройства, полученный от VK ID
    e. `code` - код авторизации, полученный от VK ID
    f. `redirect_uri` - адрес для возврата в веб приложение TWM

Пример запроса:

```
curl "https://id.vk.com/oauth2/auth" -d "client_id=12345678&grant_type=authorization_code&code_verifier=<Значение
```

10. В ответ на код авторизации **VK ID** возвращает токены:
    a. Access token
    b. Refresh token

Пример тела ответа:

```
{
  "access_token": "XXXXX",
  "refresh_token": "XXXXX",
  "id_token": "XXXXX",
  "expires_in": 0,
  "user_id": 1234567890,
  "state": "XXX",
  "scope": "wall"
}
```

11. **Backend TWM** сохраняет Access токен в кэш Redis, Refresh токен - в таблицу refresh_tokens

Получение Access токена в обмен на Refresh токен

Предусловия

1. Пройден алгоритм получения Access токена в обмен на код авторизации
2. Есть сохранённый Refresh токен в таблице refresh_tokens
3. Время жизни Refresh токена не истекло

Алгоритм работы

1. **Backend TWM** отправляет запрос в **API VK ID**, чтобы обменять Refresh токен на Access токен, используя параметры:
    - `grant_type=refresh_token` - параметр, который указывает, какой тип grant используется для получения токена, константа
    - `refresh_token` - Refresh токен, полученный при первичной авторизации
    - `client_id` - идентификатор приложения
    - `device_id` - уникальный идентификатор устройства, полученный при первичной авторизации
    - `state` - произвольная строка состояния приложения

Пример запроса:

```
POST https://id.vk.com/oauth2/auth
Content-Type: application/x-www-form-urlencoded
grant_type=refresh_token&refresh_token=<Значение>&client_id=12345678&device_id=<Значение>&state=<Значение>&
```

2. В ответ на код авторизации **VK ID** возвращает токены:
   a. Access token
   b. Refresh token

Пример тела ответа:

```
{
  "access_token": "<Значение>",
  "refresh_token": "<Значение>",
  "expires_in": 0,
  "user_id": 1234567890,
  "state": "<Значение>",
  "scope": "wall"
}
```

3. **Backend TWM** сохраняет Access токен в кэш Redis, Refresh токен - в таблицу refresh_tokens

Выполнение запросов к API VK

Во всех запросах к **API VK** необходимо передавать Access токен в качестве значения заголовка `Authorization` запроса (в Headers):

```
Authorization: Bearer <Access токен>
```

Пример запроса:

```
curl --location 'https://api.vk.com/method/wall.post' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'Authorization: Bearer <Access токен>' \
--data-urlencode 'owner_id=<Значение>' \
--data-urlencode 'friends_only=<Значение>' \
--data-urlencode 'message=<Значение>' \
--data-urlencode 'v=<Значение>'
```

Реавторизация при получении HTTP-401

В API VK не предусмотрена отправка в ответ кода состояния `HTTP 401 Unauthorized`.

Если в запросе отсутствует или недействительный Access токен, то VK возвращает код состояния HTTP 200 OK с кодом и сообщением ошибки.

Код ошибки передается в элементе `error.error_code`, сообщение ошибки - в элементе `error.error_msg`.

Пример тела ответа с ошибкой авторизации:

```
{
    "error": {
        "error_code": 5,
        "error_msg": "User authorization failed: ...",
```

```
        "request_params": [
        ...
        ]
    }
}
```

Когда от **API VK** возвращается ответ:

1. Проверить наличие в сообщении элемента `error`
2. Проверить значение элемента `error.error_code`
3. Если в значении элемента `error.error_code` код 5, получить новый Access токен в обмен на Refresh токен
4. Повторно выполнить исходный запрос пользователя
5. Если ошибка повторяется, то:
   a. Вернуть пользователю сообщение: **“Ваш сеанс авторизации VK завершен. Пожалуйста, войдите в систему повторно, чтобы продолжить работу.”**
   b. Зафиксировать событие в журнале ошибок с деталями запроса и ответа и отправить уведомление в систему мониторинга

Максимальное кол-во попыток реавторизации равно 2.

Обработка ошибки доступа при получении HTTP-403

В API VK не предусмотрена отправка в ответ кода состояния `HTTP 403 Forbidden`.

Если ошибка доступа, то VK возвращает код состояния HTTP 200 OK с кодом, дополнительным кодом и сообщением ошибки.

Код ошибки передается в элементе `error.error_code`, дополнительный код - в элементе `error.error_subcode`, сообщение ошибки - в элементе `error.error_msg`.

Пример тела ответа с ошибкой авторизации:

```
{
    "error": {
        "error_code": 15,
        "error_subcode": 1133,
        "error_msg": "Access denied: no access to call this method. It cannot be called with current scopes.",
        "request_params": [
        ...
        ]
    }
}
```

Когда от **API VK** возвращается ответ:

1. Проверить наличие в сообщении элемента `error`
2. Проверить значение элемента `error.error_code`
3. Если в значении элемента `error.error_code` код 15, то:
   a. Вернуть пользователю сообщение: **“Доступ ограничен. Пожалуйста, проверьте права, которые вы предоставили при авторизации приложению TWM, или свяжитесь с технической поддержкой.”**
   b. Зафиксировать событие в журнале ошибок с деталями запроса и ответа и отправить уведомление в систему мониторинга

Деавторизация

Предусловия:

1. Пройден алгоритм получения Access токена в обмен на код авторизации
2. Есть сохранённый Access токен в кэше Redis
3. Время жизни Access токена не истекло
4. Пользователь инициирует процесс отключения учетки VK в настройках приложения TWM

Алгоритм выполнения:

1. **Frontend TWM** отправляет запрос на **Backend TWM**
2. **Backend TWM** отправляет запрос в **API VK ID**, чтобы проинвалидировать токен и завершить сессию пользователя, используя параметры:
   a. `access_token` - access_token, полученный на четвертом шаге
   b. `client_id` - идентификатор приложения

Пример запроса:

```
curl "https://id.vk.com/oauth2/logout" -d "client_id=7915193&access_token=XXX"
```

4. **VK ID** возвращает ответ

Пример тела ответа:

```
{"response":1}
```