

Документация API-платформа: native.api

Содержание

Описание модуля	5
Обзор модуля	
Особенности версии	7
Возможности	9
Особенности платформы	11
Сравнение возможностей	12
Системные требования	13
Установка и запуск	
Установка	15
Запуск	17
Проверка работы	20
Проверка работы	
Сценарии проверки	24
Дашборд	31
Документация	35
Токены авторизации	38
Входящие запросы	42
Журнал безопасности	47
События модуля	49
Настройки модуля	
Основные	52
Безопасность	55
Роуты	60
Аналитика	63
Документация	65
Логирование	68
Доступ	73
Разработчикам	
Контекст API	76
Получение настроек модуля	78
Разработчикам — Карта роутов	
Подключение карты	79
Ключи карты	81

Адреса роутов	83
Контроллеры	85
Параметры запроса	86
Вложенные параметры	89
Безопасность	92
Ответ и кеш	96
Поля OpenAPI	99
Разработчикам	
Рендер документации	105
Разработчикам — Нативные роуты	
Сервисные роуты	107
Провайдер	110
Токены	112
Разработчикам — События	
Подписка на события	115
Запуск и роутинг	119
Проверки доступа	121
Обработка запроса	123
События токенов	125
Разработчикам — Константы	
Константы модуля	127
CORS	129
HTTP-заголовки	130
HTTP-статусы	131
Разработчикам — Константы — HTTP-статусы	
Названия статусов	135
Разработчикам — Константы	
OpenAPI	139
Безопасность	142
Кеш	144
Контроллеры	145
Настройки	146
Основные	151
Ответ	152
Ошибки	154
Параметры	157

Права и группы	158
Провайдер	159
Результаты	160
Роуты	161
Сегменты роутов	162
События API	163
События аудита	166
Статусы запросов	167
Типы и форматы	168
 Разработчикам — Примеры работы	
Пример карты	169
Контроллеры	178
События	180
 Разработчикам — Примеры работы — Запросы	
cURL	184
JavaScript	187
PHP	191
 Разработчикам — Примеры работы — Ответы	
JSON	195
XML	196
 Используемые ресурсы	 197

Описание модуля

Название	Описание
Наименование	API-платформа
Код модуля	native.api
Дата релиза	02.10.2026
Дата обновления	02.10.2026
Версия	1.0.0
Документация	Скачать

Модуль централизованного управления API-трафиком и интеграциями для 1С-Битрикс. Он объединяет обработку HTTP-запросов, авторизацию, проверку параметров, управление доступом, кеширование, документацию и журналирование.

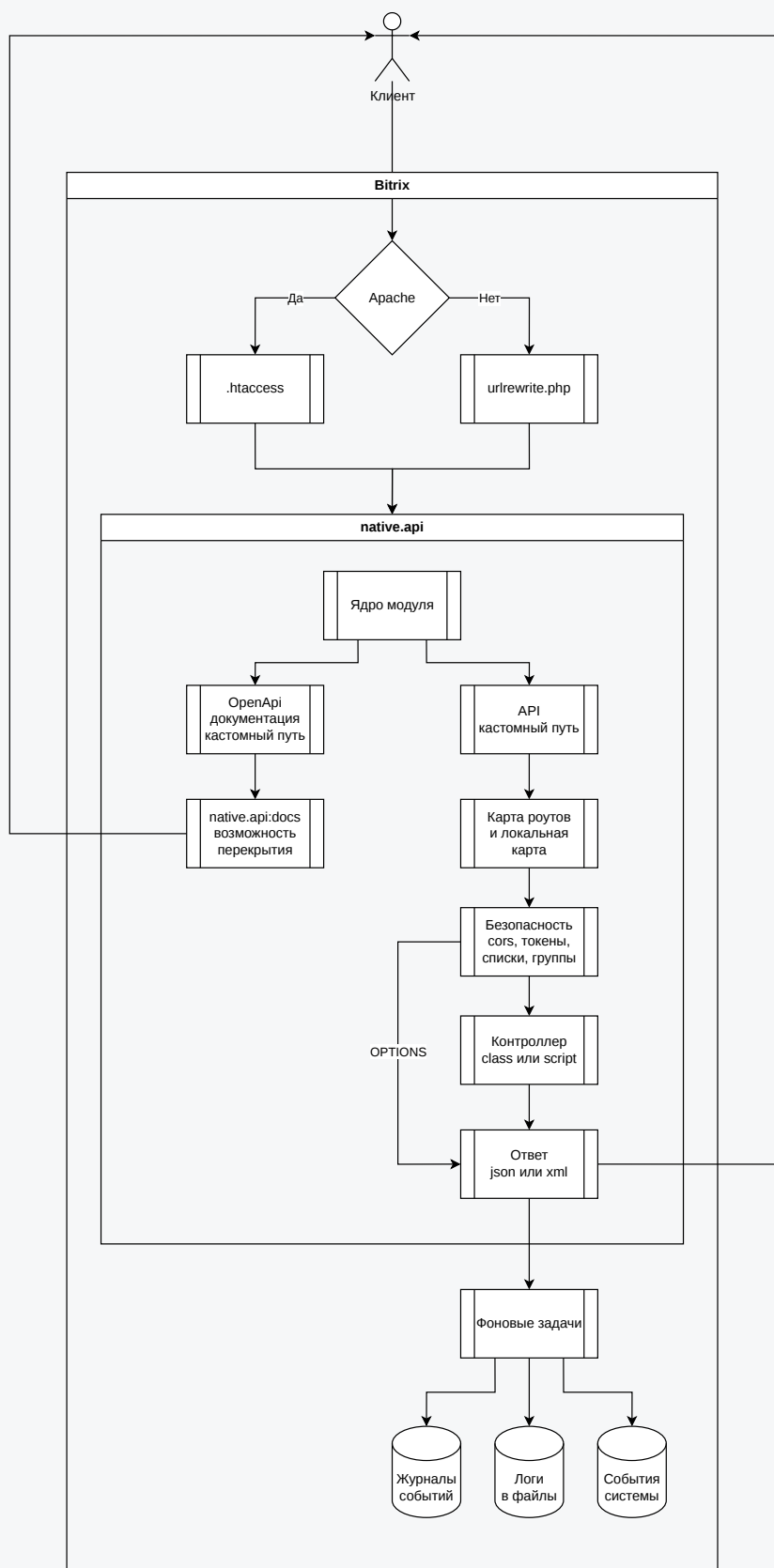
Назначение

API-платформа служит общей основой для обмена данными между сайтом и внешними системами: мобильными приложениями, личными кабинетами, ERP, CRM и сервисами партнёров. Общие правила обработки запросов настраиваются в одном модуле и могут уточняться для отдельных роутов.

Разработчик описывает роуты в карте: адрес, HTTP-метод, параметры, правила доступа и контроллер. Контроллер выполняет действие (экшн) — например, получает каталог или создаёт заказ — и возвращает результат. Бизнес-логика действий и проверки прав на конкретные данные остаются частью интеграции.

Схема работы

При обработке запроса платформа находит роут, проверяет доступ и параметры, затем вызывает контроллер и возвращает результат клиенту в JSON или XML. Если для роута настроено кеширование, результат может быть получен из кеша; проверки доступа при этом сохраняются.



Особенности версии

native.api продолжает развитие artamonov.rest, впервые опубликованного в 2018 году. В таблице — отличия от предыдущего модуля.

Что изменилось

Возможность	artamonov.rest	native.api
Запуск API	Для работы с текущим пользователем требовалось самостоятельно подключать классы и модули	Сервис подключает ядро Битрикса; пользователь доступен через контекст API
Клиенты пользователя	Один токен на пользователя	Несколько клиентских сессий с отдельными парами токенов
Обновление доступа	Управление одним токеном и его сроком действия	Пары access/refresh-токенов с отдельными сроками и ротацией
Отзыв доступа	Удаление общего токена пользователя	Отзыв выбранной клиентской сессии без отключения остальных
Хранение токенов	Исходный токен в поле пользователя	Хеши токенов вместо исходных значений
Подготовка роутов	Карты загружаются при обработке запроса	Скомпилированные карты кешируются и используются повторно
Событийная модель	Расширение логики через собственные контроллеры	Пользовательские обработчики до и после основных этапов запроса и операций с токенами
Пути API и документации	Выбор API по первому сегменту пути	Независимые вложенные пути API и документации с автоматическим обновлением точек входа
Документация	Описание роутов в админке и JSON-описание для клиента	OpenAPI и готовая интерактивная документация с выполнением запросов
Видимость документации	Настройки включения роута в административное и публичное описание	Доступ к документации и набору роутов по группам пользователей

Журналирование	Журналы запросов, ответов и лимитов в базе	История с карточкой запроса; данные запросов и ответов в базе и файлах; аудит событий; независимая настройка записи
Рабочее пространство	Административные страницы модуля	Дашборд с собственными и настраиваемыми гаджетами

Возможности

Роуты и параметры

Статические адреса и адреса с переменными, собственные контроллеры и действия. Многоуровневая валидация входящих параметров по карте роутов: обязательность, типы, допустимые значения и вложенные структуры, включая элементы списков.

Авторизация и доступ

Basic- и Bearer-авторизация, выпуск и обновление токенов доступа, настройка сроков их действия и отзыв клиентских сессий.

IP-ограничения для всего API и отдельных роутов, ограничения по группам пользователей, расписания доступности, лимитирование запросов и настройка CORS для обращений из браузера.

Ответы и кеширование

Форматы JSON и XML. Общее кеширование задаётся в настройках модуля для GET/HEAD. Настройки кеширования отдельного роута имеют приоритет над общими.

Документация

Генерация OpenAPI из карт роутов и рендер документации для просмотра и выполнения запросов. Доступ к странице документации и видимость роутов настраиваются по группам пользователей.

Дашборд и мониторинг

Дашборд, счётчики запросов по HTTP-методам, техническая история входящих запросов со статусами для расчёта лимитов, карточка запроса с параметрами, заголовками и ответом. Входящие и исходящие данные можно независимо сохранять в базу и файлы при включённом сборе запросов. Также доступны журнал событий и файловые логи. Дашборд поддерживает собственные гаджеты.

Событийная модель

Обработчики событий до и после этапов запроса и операций с токенами.

Особенности платформы

Платформа предоставляет общие механизмы обработки запросов. Операции с данными сайта разрабатываются под задачи проекта. Полный перечень механизмов — в разделе «Возможности».

Настройка под проект

Разработчик задаёт адреса, HTTP-методы, параметры и правила доступа в карте роутов. Контроллеры выполняют действия, а события позволяют дополнить обработку запроса.

Формат ответа и описание API можно адаптировать к требованиям подключаемого приложения.

Ответственность интеграции

Модуль проверяет доступ к роуту и параметры согласно карте. Контроллер должен отдельно проверять права пользователя на конкретные записи, например заказ или документ, и выполнять бизнес-правила проекта.

Встроенные роуты предоставляют операции с токенами, диагностические запросы и вызов разрешённых методов через провайдер. Прикладные операции, например получение каталога или создание заказа, добавляет разработчик интеграции.

Собственные доработки

Храните карту и контроллеры вне каталога модуля. Для замены интерфейса документации используйте переопределение через `local`.

После изменения карты очистите кеш модуля. После обновления проверьте используемые сценарии интеграции.

Сравнение возможностей

native.api задаёт API через карты роутов и контроллеры. Штатный rest предоставляет методы, зарегистрированные модулями платформы. Сравнение относится к серверному модулю rest 26.650.0; состав методов и доступность авторизации зависят от установленного продукта и модулей.

Механизм	native.api	rest
Описание операций	Карта с URL, HTTP-методами, параметрами и правилами доступа	Реестр именованных REST-методов модулей
Расширение	Контроллеры и события этапов запроса	Регистрация собственных методов, в том числе через OnRestServiceBuildDescription
Авторизация	Basic и локальные пары access/refresh-токенов для клиентов пользователя	OAuth-приложения и входящие вебхуки
Форматы ответа	JSON и XML; служебная обёртка успешного ответа отключается для роута	JSON и XML в формате ответа REST

Для API с собственной структурой URL, правилами валидации и ответами используйте возможности native.api. Если интеграция уже обращается к методам rest, её контракт и авторизацию нужно учитывать отдельно: совпадение формата JSON/XML не делает API взаимозаменяемыми.

Системные требования

Платформа

PHP >= 8.2; 1С-Битрикс: Управление сайтом или Битрикс24 (коробочная версия).

Модуль работает в окружении сайта. Сервер и хостинг должны соответствовать официальным требованиям 1С-Битрикс. Ниже приведены основные требования к окружению «1С-Битрикс: Управление сайтом».

Сервер и база данных

Для размещения сайта подходит UNIX-окружение с веб-сервером Apache версии 2.0 и выше или актуальной стабильной версией Nginx.

При использовании MySQL требуется версия 8.0 или новее и соответствующее расширение PHP для подключения к базе. Серверные компоненты должны работать с UTF-8, а база MySQL — с кодировкой utf8mb4.

Расширения PHP

В окружении должны быть доступны расширения, необходимые установленным модулям платформы. В основной набор входят средства работы с XML, многобайтовыми строками (Multibyte String), хешированием (Hash), шифрованием (OpenSSL), сжатием данных (Zlib) и базой данных.

Полный состав расширений и параметры PHP приведены в требованиях к серверному программному обеспечению.

Хостинг

Можно использовать хостинг, соответствующий требованиям платформы. Перед установкой проверьте версии серверного ПО и доступность необходимых расширений PHP.

На странице официальных требований доступен скрипт проверки хостинга. Он помогает проверить окружение до установки платформы.

Браузер

Используйте актуальную версию Google Chrome, Mozilla Firefox, Microsoft Edge, Safari или Яндекс Браузера.

Для мастера установки, административных интерфейсов модуля и встроенного рендера документации в браузере должен быть включён JavaScript.

Установка

1. Подготовьте сайт

1. Проверьте системные требования.
2. Войдите в административный раздел сайта под администратором.
3. Убедитесь, что в браузере включён JavaScript: он нужен мастеру установки.

Первую установку и проверку выполняйте на тестовом сайте.

2. Загрузите модуль

1. Откройте карточку модуля в Маркетплейсе.
2. Нажмите «Установить» и укажите адрес тестового сайта.
3. Следуйте указаниям Маркетплейса для загрузки модуля на сайт.

Модуль также можно найти по коду «native.api» в разделе «Marketplace → Каталог решений». Демонстрационный период — 14 дней.

Для загрузки требуется активный срок получения обновлений 1С-Битрикс. Инструкция Маркетплейса.

Если у вас есть купон, сначала активируйте его: «Marketplace → Обновление решений → Активация купона». Затем загрузите доступную версию модуля.

3. Выполните установку

1. Откройте «Marketplace → Установленные решения».
2. Найдите «API-платформа» и выберите «Установить модуль».
3. На вкладке «Компоненты установки» нажмите «Продолжить установку».
4. Дождитесь сообщения «Все компоненты установлены».
5. Нажмите «Перейти к модулю».

Результат: открылся дашборд модуля. Он также доступен через «Сервисы → API-платформа → Дашборд».

При первой установке API ещё нужно настроить. Перейдите к запуску.

Если установка прервалась

1. Прочитайте сообщение мастера и устраните указанную причину.
2. Нажмите «Повторить».
3. Дождитесь завершения установки.

Если версия PHP не подходит, смените её для сайта согласно системным требованиям и снова откройте мастер.

Запуск

Выполните после установки, в административном разделе сайта.

1. Задайте адреса

1. Откройте «Сервисы → API-платформа → Настройки модуля».
2. На вкладке «Основные», в блоке «Настройка путей», заполните поля:

Поле	Значение
Путь интерфейса	api
Путь документации	api/docs
Путь карты роутов	Оставьте пустым

3. Нажмите «Сохранить».

Пути указываются без домена. Модуль сам создаст точки входа и включит API.

2. Включите проверочные запросы

1. Перейдите на вкладку «Роуты».
2. Отметьте «Активировать сервисные роуты».
3. Нажмите «Сохранить».

3. Откройте документацию API

1. Выберите «Сервисы → API-платформа → Документация».
2. Проверьте адрес страницы: при указанных выше настройках это `/api/docs/` на вашем сайте.
3. В поле `Servers` проверьте адрес API: он должен указывать на ваш сайт и путь `/api/`.

В списке операций должен быть `GET /up`. Это встроенный запрос проверки API.

4. Выполните первый запрос

1. Раскройте `GET /up`.
2. Нажмите `Try it out`, затем `Execute`. Кнопка `Authorize` для этого запроса не нужна.
3. В блоке `Server response` найдите фактический код и тело ответа.

Результат: код `200`, в `Response body` — JSON с `result.method = GET` и `result.action = list`. Пример полного ответа:

```

1  {
2    "request_id": "0af1b6c2d3e4f5a6b7c8d9e0f1a2b3c4",
3    "status": {
4      "code": 200,
5      "title": "200 OK",
6      "message": "success"
7    },
8    "result": {
9      "method": "GET",
10     "controller": "Native\\Api\\Controller\\Up\\UpIndex",
11     "action": "list",
12     "description": "Проверка доступности и работоспособности API-сервиса",
13     "parameters": []
14   },
15   "time": {
16     "start": 1790586000.125,
17     "finish": 1790586000.1875,
18     "duration": 0.0625,
19     "date_start": "28.09.2026 12:00:00",
20     "date_finish": "28.09.2026 12:00:00",
21     "timestamp": {
22       "date_start": 1790586000,
23       "date_finish": 1790586000
24     }
25   }
26 }
```

5. Продолжите проверку

Оставьте сервисные роуты включёнными и перейдите к проверке работы. Там проверяются параметры, авторизация, токены и история запросов.

Если результат отличается

Что произошло	Что проверить
Нет пункта «Документация»	Сохранён ли «Путь документации». Открыта ли админка под администратором

В документации нет GET /up	Включены ли сервисные роуты. Сохраните настройки и обновите страницу документации
Вместо JSON — страница сайта или HTML-ошибка 404	Адрес в Servers и сохранённый «Путь интерфейса». Если они верны — настройки обработки URL на сервере
JSON с кодом ROUTE_NOT_FOUND	Выбран ли GET /up, включены ли сервисные роуты, нет ли ошибки в адресе
HTTP 403	Код ошибки в ответе и ограничения доступа. Вход в админку не отменяет ограничения API
HTTP 500 или Failed to fetch	Для 500 проверьте журнал ошибок PHP; для Failed to fetch — доступность адреса из Servers и HTTPS-сертификат. Если документация и API на разных доменах — CORS

Если ошибка остаётся, сохраните адрес запроса, HTTP-код и ответ без паролей и токенов. Передайте их в поддержку.

Проверка работы

Начните после успешного `GET /up` из раздела «Запуск». Все шаги выполняйте на тестовом сайте, в том же браузере и под той же учётной записью администратора. Для проверки токенов используйте HTTPS.

Ниже API расположен по адресу `/api/`. Если вы задали другой путь, замените его в примерах.

1. Проверьте параметры запроса

1. Откройте новую вкладку браузера.
2. На своём сайте перейдите по адресу `/api/up/user/15?name=Ivan`.
3. Найдите в JSON поле `result.parameters`.

Результат: в `parameters` есть `id` со значением `15` и `name` со значением `Ivan`. Это проверочный запрос: он не создаёт и не изменяет пользователя.

2. Проверьте отсутствующий адрес

Откройте `/api/check-not-found` на своём сайте.

Результат: JSON-ошибка `ROUTE_NOT_FOUND`, в `status.code` — `404`.

3. Проверьте авторизацию

1. В настройках модуля откройте вкладку «Роуты».
2. Отметьте «Активировать нативные роуты» и нажмите «Сохранить».
3. Обновите страницу документации `/api/docs/`. В ней появятся операции `/token`.
4. Нажмите `Authorize`. Если в `basicAuth` или `bearerAuth` уже сохранены данные, нажмите для них `Logout`, затем `Close`. Из админки сайта выходить не нужно.
5. Откройте `GET /token` и нажмите `Try it out` → `Execute`.

Без авторизации: в `Server response` код `401`.

6. Нажмите `Authorize`. В блоке `basicAuth` укажите логин и пароль учётной записи, под которой открыта админка. Нажмите `Authorize`, затем `Close`.

7. Повторите `GET /token` кнопкой `Execute` .

С авторизацией: код `200` . Если токены ещё не выпускались, поле `result` может отсутствовать.

Если получен `AUTH_MFA_REQUIRED` , завершите вход с одноразовым кодом на сайте под этим же пользователем и повторите запрос в том же браузере. Проверку MFA отключать не нужно.

4. Выпустите токены

1. Откройте `POST /token` и нажмите `Try it out` → `Execute` . Параметры заполнять не нужно.
2. Проверьте код `201` в `Server response` .
3. В `Response body` , в массиве `result` , найдите два объекта: с `type = access` и `type = refresh` .
4. Сохраните значение `token` из каждого объекта. Не меняйте браузер до конца проверки.
5. Выполните `GET /token` и сохраните `client.id` из записи с датой только что выполненного выпуска. Он понадобится для отзыва.

Повторный `POST /token` в этом браузере заменит выданную пару. Для следующих шагов используйте последние значения.

5. Обновите токены

1. Нажмите `Authorize` . В `basicAuth` нажмите `Logout` .
2. В блок `bearerAuth` вставьте **refresh-токен** без слова `Bearer` . Нажмите `Authorize` , затем `Close` .
3. Выполните `PUT /token` через `Try it out` → `Execute` .
4. Проверьте код `200` и сохраните **оба новых токена** из `result` .
5. Не меняя значение в `bearerAuth` , повторите `PUT /token` .

Результат повторного запроса: код `401` . Старый refresh-токен больше не действует.

6. Отзовите токены

1. Нажмите `Authorize` . В `bearerAuth` нажмите `Logout` и укажите **новый access-токен** из шага 5.
2. Нажмите `Authorize` , затем `Close` .
3. Откройте `DELETE /token` , нажмите `Try it out` и вставьте сохранённый `client.id` в поле `client_id` .
4. Нажмите `Execute` .

Результат: код `200` , в ответе `result.delete = true` . Если получено `false` , проверьте `client_id` : нужная сессия не была отозвана.

5. Повторите тот же запрос с тем же токеном.

После отзыва: код `401` .

7. Проверьте историю и данные запроса

1. В настройках модуля, на вкладке «Аналитика», отметьте «Собирать входящие запросы» и сохраните настройки.
2. На вкладке «Логирование», в блоке «Входящие запросы», отметьте «Записывать входящие данные» и «Записывать исходящие данные». Соседние флажки «записывать в файл» для этой проверки не нужны.
3. Сохраните настройки.
4. В новой вкладке откройте `/api/up?check=history` на своём сайте. Скопируйте `request_id` из ответа.
5. Перейдите в «Сервисы → API-платформа → Входящие запросы». Обновите список и найдите запрос по этому идентификатору.
6. Откройте его карточку нажатием на ID. Проверьте вкладки «Входящие данные» и «Исходящие данные».

Результат: для этого `request_id` есть одна запись. Во входящих данных виден параметр `check = history` , в исходящих — ответ API. Если данных нет, проверьте, что запрос выполнен **после** сохранения обеих настроек логирования и включения сбора.

8. Завершите проверку

1. В документации нажмите **Authorize** и **Logout** для оставшихся сохранённых данных.
2. На вкладке «Роуты» отключите сервисные роуты. Нативные роуты оставьте включёнными только если будете использовать операции с токенами.
3. Сохраните настройки. Верните параметры аналитики и логирования к нужным для сайта значениям.
4. Удалите сохранённые для проверки значения токенов.

Если ожидаемые результаты получены, базовая проверка пройдена. Для проверки прав, ограничений, собственных роутов и удаления модуля используйте подробные сценарии.

Если шаг не пройден, сохраните его номер, HTTP-код и ответ без паролей и токенов. Разбор ошибок запуска.

Сценарии проверки

После базовой проверки выполните сценарии для используемых в проекте механизмов.

Подготовка

Проверки с изменением прав, ограничений, карт роутов и данных выполняйте на тестовой копии сайта. Подготовьте отдельного пользователя, двух API-клиентов и тестовый роут с контролируемым результатом. Административные права проверяйте под пользователем без прав администратора.

Используйте адреса API и документации из настроек своего сайта. Каждое ограничение проверяйте отдельно: остальные условия должны разрешать запрос. Если механизм не используется в проекте, отметьте пункт как неприменимый.

Установка и удаление

Этот сценарий выполняйте на отдельной тестовой копии сайта, которую можно восстановить. Обычная проверка после обновления не требует удаления модуля.

Проверить	Как проверить	Ожидаемый результат
Установка	Пройдите мастер установки и откройте дашборд	Все шаги завершены; административные страницы и настроенные точки входа доступны согласно правам
Удаление с сохранением данных	Создайте тестовые данные, при удалении оставьте соответствующие категории невыбранными, затем установите модуль повторно	Сохранённые данные доступны после повторной установки; при сохранении действующих токенов они продолжают работать
Удаление с очисткой	На восстановленной копии повторите удаление, выбрав категории для очистки, затем установите модуль снова	Данные выбранных категорий удалены. Несвязанные данные сайта и файлы других модулей сохранены

После установки

Выполните эти проверки после первоначального запуска. После обновления повторите их перед проверкой собственных доработок.

Точки входа

Проверить	Как проверить	Ожидаемый результат
Точки входа	Откройте настроенные адреса API и документации. Выполните запрос к существующему и отсутствующему роуту	Существующий роут возвращает ответ модуля, отсутствующий — HTTP 404. Документация открывается при наличии права чтения
Документация	Откройте её под пользователями с разными правами на страницу и группы роутов, затем повторите проверку после заполнения кеша	Пользователь видит только доступные ему роуты. Без права чтения доступ к документации запрещён
Сервисные роуты	Включите сервисные роуты и выполните запросы из документации, затем отключите их	Включённые роуты отвечают согласно своим описаниям и правам доступа; отключённые недоступны

Авторизация и токены

Проверить	Как проверить	Ожидаемый результат
Защищённый роут	Выполните запрос с верными данными авторизации, без них и с неверными данными	Верные данные дают доступ; отсутствие или ошибка обязательной авторизации возвращает HTTP 401
Выпуск и список токенов	Выпустите пары токенов для двух клиентов одного пользователя и запросите список сессий	У клиентов отдельные пары; список содержит их сессии без исходных значений токенов
Обновление токенов	Обновите пару первого клиента через refresh-токен, затем повторите запрос со старым refresh-токеном	Новая пара работает; повторное использование старого refresh-токена отклоняется. Второй клиент продолжает работать
Срок действия и отзыв	Проверьте доступ с истёкшим токеном. Отзовите сессию первого клиента и повторите запросы обоих клиентов	Истёкший и отозванный токены не дают доступ; сессия второго клиента остаётся действующей

Для токенов, выпущенных через API, проверьте работу с теми же заголовками клиента и отказ после их изменения. При отказе обновления из-за смены заголовков исходная пара должна оставаться действующей для прежнего клиента.

Права в админке

Проверить	Как проверить	Ожидаемый результат
Чтение страниц модуля	Разрешите пользователю чтение дашборда, токенов, входящих запросов или настроек, затем отзовите право. Проверьте меню и прямой переход на страницу	Раздел доступен только при наличии права; отсутствие пункта меню нельзя обойти прямой ссылкой
Изменение данных	С правом только чтения попробуйте сохранить настройки или изменить токены, в том числе прямым запросом. Повторите с правом изменения	Изменяющие операции требуют соответствующего права. Очистка кеша модуля отдельно разрешена пользователю с правом чтения настроек

Для журналов отдельно проверьте видимость ссылок в меню модуля и доступ к системному журналу событий. Просмотр самого журнала регулируется правами платформы: скрытая ссылка в меню модуля не запрещает прямой переход пользователю с таким доступом.

После обновления

Дополнительно проверьте используемые в проекте настройки, свои карты роутов, контроллеры и обработчики событий. Сравнивайте ответы с ожидаемым поведением вашей интеграции до обновления.

На дашборде проверьте гаджет «Логирование»: его флаги должны совпадать с настройками одноимённой вкладки, включая разные состояния записи в базу и файлы. Если гаджета нет в сохранённой раскладке, добавьте его через «Добавить гаджет». Успешность самой записи проверяйте по журналам после нового запроса.

Роуты и расширения

Проверить	Как проверить	Ожидаемый результат

Пользовательская карта	Вызовите собственный роут и роут, переопределённый локальной картой, с указанными в карте HTTP-методами	Вызываются нужные контроллеры и действия; параметры, права и формат ответа соответствуют вашей карте
Роут провайдера	Вызовите разрешённый метод, затем метод вне разрешённого списка и явно запрещённый метод	Разрешённый вызов выполняется; остальные отклоняются до выполнения метода. Явный запрет имеет приоритет
Рекурсивная валидация	Передайте корректные параметры, затем по одному нарушайте обязательность, типы и допустимые значения, включая вложенные объекты и элементы списков	Корректные данные доходят до контроллера; ошибочные отклоняются на проверке параметров, действие контроллера не выполняется
Событийная модель	Подключите свои обработчики событий и проверьте их вызов и влияние на результат. Повторите запрос с кешированием ответа	Обработчики работают на соответствующих этапах. При попадании в кеш ответа контроллер и события до и после него повторно не вызываются

Кеширование

Проверить	Как проверить	Ожидаемый результат
Кеш карты и роутов	Выполните запрос после очистки кеша и повторно. Измените тестовую карту, очистите кеш модуля и снова вызовите роут	Первичное и повторное обращения используют одинаковые правила; после изменения и очистки применяется новая карта
Кеш настроек	Сохраните изменение настройки, влияющей на запрос, и сразу повторите запрос	Новое значение применяется без ожидания истечения прежнего кеша
Общее кеширование ответов	Для тестового GET-роута без собственного срока кеша задайте общий срок. По счётчику вызовов контроллера проверьте первый запрос, повторный и запрос после истечения срока	Повторный запрос получает сохранённый результат без вызова контроллера; после истечения срока контроллер выполняется снова

Кеш отдельного роута	Задайте собственный срок кеша роута при отключённом общем кеше. Затем отключите кеш роута при включённом общем	Настройка роута имеет приоритет: в первом случае результат кешируется, во втором контроллер вызывается при каждом запросе
Разделение ответов и права	Проверьте один роут с разными параметрами, пользователями и клиентами. После заполнения кеша запретите доступ одному из них и повторите запрос	Результаты разных пользователей и клиентов не смешиваются; изменённые параметры учитываются. Запрет доступа действует и при заполненном кеше

Ограничения доступа

Проверить	Как проверить	Ожидаемый результат
Режим работы	Настройте разрешённый интервал и проверьте запрос внутри и вне него, учитывая день недели и время сервера PHP	Внутри разрешённого интервала запрос проходит; вне него отклоняется
Количество запросов	Настройте небольшой лимит на тестовом стенде. Последовательно выполните запросы до его достижения, сверх него и после освобождения лимита	В пределах лимита запросы проходят; превышение возвращает HTTP 429. После освобождения лимита доступ восстанавливается
CORS	Выполните запросы из браузера с разрешённого и запрещённого источника. Проверьте предварительный OPTIONS-запрос с разрешёнными и запрещёнными заголовками	Разрешённый источник получает нужные CORS-заголовки; запрещённый источник или заголовки отклоняются. Разрешённый OPTIONS завершается без вызова контроллера
IP-адреса	Проверьте запросы с разрешённого адреса, адреса вне белого списка и адреса из чёрного списка	Доступ соответствует спискам; чёрный список имеет приоритет при совпадении
Группы пользователей	Выполните запрос под пользователями из разрешённых, неразрешённых и явно запрещённых групп	Роут доступен только разрешённым группам; явный запрет имеет приоритет

Для групповых ограничений используйте авторизацию, предусмотренную роутом: вход в административную часть сайта сам по себе не авторизует API-запрос.

Проверка собственных обработчиков

Если интеграция использует события, проверьте её результат при успешном запросе и отказе в доступе. Повторите запрос с включённым кешированием: обязательные действия не должны зависеть от событий контроллера, которые при попадании в кеш не вызываются.

Для обработчика, возвращающего ответ досрочно, проверьте HTTP-статус, заголовки и данные клиента. Последующие обработчики и контроллер выполняться не должны.

История и данные запросов

Проверьте сбор запросов и четыре независимых направления записи. Используйте тестовые значения секретов.

Условие	Ожидаемый результат
Сбор включён, лимит выключен	Одна запись на запрос, включая отказ, OPTIONS и досрочный ответ события
Сбор выключен, лимит действует	Только основная запись; флаги данных не создают записи в базе или файлы
Сбор и лимит выключены	История и журналы данных не создаются
Сбор включён, флаги данных включаются по одному	Записывается только выбранное направление: входящие или исходящие данные, база или файл
Данные записываются только в файлы	Карточка сообщает об отсутствии данных в базе; файлы доступны по REQUEST_ID
JSON, XML, HEAD и нестандартный ответ	Формат и отсутствие тела соответствуют ответу; непустое неподдерживаемое тело заменено маской
Вложенные пароли, токены, Authorization и Cookie	Распознаваемые секреты замаскированы в базе и файлах
Запрос с включённым лимитом	В подсчёт входит только статус PROCESSED; ошибка контроллера после проверки лимита не меняет его

Удаление запроса

Основная запись и связанные данные удалены из базы; файлы сохранены

После запроса обновите список истории и откройте его карточку.

Завершение проверки

Зафиксируйте версию модуля, окружение, проверенные сценарии и результат каждого пункта. Для ошибки сохраните шаги воспроизведения, HTTP-статус и ответ без паролей и токенов.

Удалите тестовые данные и обработчики, восстановите временно изменённые настройки, права и карты роутов либо восстановите тестовый стенд из снимка.

Дашборд

Дашборд находится в разделе «Сервисы → API-платформа → Дашборд». Он показывает состояние модуля, параметры авторизации и логирования, состав роутов и накопленную статистику запросов. Чтобы получить актуальные значения, обновите страницу.

Для просмотра и персональной настройки требуется право чтения раздела «Рабочий стол» на вкладке «Доступ».

Гаджеты

В поставку входят пять гаджетов. По умолчанию они размещены в двух столбцах: «Запросы» — под «Маршрутами» слева, «Логирование» — под «Авторизацией» справа.

Настройки

Сводка текущих настроек и окружения сайта. Помогает проверить, включён ли API и какие пути используются.

Показатель	Что показывает
API-интерфейс	Активен ли API-интерфейс модуля
Путь интерфейса и Путь документации	Адреса API и встроенной документации
Путь карты маршрутов	Путь пользовательской карты роутов
Путь к журналам	Каталог для файловых журналов
Нативные роуты, Сервисные роуты, Роут-провайдер	Какие наборы роутов включены
Версия PHP	Версия PHP, под которой открыта страница дашборда

Для незаполненного пути выводится прочерк. Изменить пути можно на вкладке «Основные», а наборы роутов — на вкладке «Роуты».

Авторизация

Показывает способы авторизации в загруженной карте роутов, сроки жизни токенов и наличие лимитов запросов.

Показатель	Что показывает
Авторизация по логину и паролю	Есть ли в карте роут с этим способом авторизации
Авторизация по токену	Есть ли в карте роут с авторизацией по токену
Срок жизни access-токена и Срок жизни refresh-токена	Настроенные сроки в секундах
Ограничение запросов	Задан ли положительный лимит количества запросов в общем правиле или хотя бы для одной группы
Выдано токен-пар	Количество сохранённых access-токенов, каждый из которых представляет токен-пару

Наличие способа авторизации определяется и по отключённым в карте роутам. Статус «Активно» не означает, что авторизация обязательна для всех запросов.

«Выдано токен-пар» — текущее количество сохранённых пар, а не число выдач за всё время. Истёкшие пары учитываются, пока их записи не удалены. При наличии права чтения токенов авторизации число становится ссылкой на их список.

Сроки жизни токенов и лимиты задаются на вкладке «Безопасность».

Маршруты

Показывает количество роутов по HTTP-методам и отдельные счётчики их ограничений:

- отключённые роуты;
- роуты с авторизацией по логину или по токену;
- роуты с белым или чёрным списком групп пользователей.

Учитывается объединённая карта: встроенные и пользовательские роуты с учётом включённых наборов на вкладке «Роуты». Отключённые наборы исключаются, но отдельный роут с `active: false` остаётся в подсчёте своего HTTP-метода и дополнительно попадает в число отключённых.

Один роут может учитываться в нескольких показателях — например, требовать токен и иметь белый список групп. Поэтому складывать все строки гаджета для получения общего числа роутов не нужно.

Запросы

Показывает общее количество учтённых обращений и распределение по HTTP-методам: **GET** , **POST** , **PUT** , **PATCH** , **DELETE** , **OPTIONS** и **HEAD** .

Подсчёт включается на вкладке «Аналитика» отдельно для каждого метода. Учитываются обращения к обработчику API, в том числе отказы, ошибки и ответы из кеша. Значения накопительные, без разбивки по дням.

Отключение сбора сохраняет прежние значения. Для начала подсчёта заново используйте «Очистить статистику» на вкладке «Аналитика». Счётчики не зависят от истории входящих запросов, используемой для лимитирования.

Логирование

Показывает параметры вкладки «Логирование» по алфавиту названий. Для каждой строки отдельно указаны два направления:

- «Журнал» — запись событий в системный журнал Битрикса, а входящих и исходящих данных — для просмотра в карточке запроса.
- «Логи» — запись в файлы.

Каждый флаг отображается как «Активно» (зелёный) или «Отключено» (красный). Гаджет показывает настройки, но не проверяет успешность записи. Для сохранения входящих и исходящих данных дополнительно включите сбор входящих запросов.

После обновления сохранённая раскладка рабочего стола не меняется автоматически. Если гаджета нет, добавьте его через «Добавить гаджет» после установки файлов обновления.

Рабочий стол

Состав и расположение

Кнопка «Добавить гаджет» открывает список доступных гаджетов. Перетаскивайте их за заголовок между столбцами и меняйте порядок. Через «Настроить» в заголовке гаджета можно изменить его название, через «Удалить» — убрать с рабочего стола. Удаление гаджета не удаляет данные модуля; его можно добавить снова.

В меню «Настройки → Настроить рабочий стол» задаются число столбцов от 1 до 9 и ширина каждого в пикселях или процентах. Состав, расположение и настройки гаджетов сохраняются для текущего пользователя.

Настройки по умолчанию

«Настройки → Сбросить текущие настройки» возвращает ваш рабочий стол к настройкам по умолчанию. Настройки API и накопленные данные при этом сохраняются.

Администратору также доступно действие «Сохранить как настройки по умолчанию». Оно задаёт начальное оформление для пользователей, которые ещё не настраивали свой рабочий стол. Уже сохранённые персональные настройки других пользователей сохраняются.

Собственные гаджеты

Дашборд поддерживает добавление собственных гаджетов и перекрытие штатных через `local/gadgets/native.api/`. Гаджет с новым именем каталога дополняет список, с именем штатного — заменяет его. После добавления разработчиком гаджет можно выбрать через «Добавить гаджет». Так на рабочий стол можно вывести показатели и сведения, нужные вашему проекту.

Документация

Встроенная документация формируется из карты роутов и отображается через рендер документации. На одной странице можно изучить операции, параметры и форматы ответов, а затем выполнить запрос к API.

Открытие документации

В административном разделе выберите «Сервисы → API-платформа → Документация». Пункт появляется, когда заполнен «Путь документации» и пользователю разрешён просмотр. Эту же страницу можно открыть по прямой ссылке и передать разработчику интеграции.

Для работы встроенного рендера в браузере должен быть включён JavaScript. Название, описание и адрес сервера задаются на вкладке «Документация» в настройках.

Просмотр роутов

Операции и параметры

Роуты сгруппированы по разделам. У каждой операции указаны HTTP-метод и путь относительно адреса API. Раскройте операцию, чтобы посмотреть описание, обязательные параметры, тело запроса и ответы. Состав этих сведений зависит от карты роутов: подробные описания и примеры добавляет разработчик.

Видимость по группам

На странице видны активные роуты из включённых наборов, разрешённые группам текущего пользователя сайта. Доступ к странице и видимость операций — отдельные проверки: схема прав.

Кнопка **Authorize** задаёт авторизацию API-запроса и не меняет видимый список операций.

Выполнение запроса

Подготовка

1. Проверьте адрес API в блоке **Servers** : запрос будет отправлен на этот сервер.

2. Откройте нужную операцию и нажмите **Try it out**.
3. Заполните обязательные параметры и тело запроса, если они предусмотрены.
4. Если роут требует авторизацию, нажмите **Authorize** и укажите данные для соответствующей схемы.
5. Нажмите **Execute** и изучите фактический HTTP-статус, тело и заголовки ответа.

Это реальные обращения к API. Операции создания, изменения и удаления выполняют действия на выбранном сервере.

Авторизация

Для Basic укажите логин и пароль пользователя. Для Bearer вставьте только значение токена: рендер добавит префикс **Bearer** в заголовок **Authorization** самостоятельно.

Тип токена зависит от операции: для прикладных запросов обычно используется access-токен, для обновления пары — refresh-токен. После ротации замените прежнее значение в **Authorize**. Порядок выпуска и обновления описан в разделе «Токены».

Вход в административный раздел не заменяет Basic- или Bearer-авторизацию API. При выполнении запроса продолжают действовать ограничения роута и настройки безопасности.

Первая проверка

После включения сервисных роутов откройте **GET /up**, нажмите **Try it out**, затем **Execute**. В штатной карте этот роут не требует токена. При разрешённом доступе ожидается HTTP 200 и JSON-ответ со сведениями о выполненном запросе: методе, контроллере, действии и параметрах.

Остальные сервисные операции проверяйте таким же способом, учитывая их параметры и ограничения. После проверки отключите сервисные роуты, если они больше не нужны, для дополнительной безопасности.

Обновление и оформление

Изменения карты роутов

После редактирования файла карты очистите кеш кнопкой «Очистить кеш» в настройках модуля и обновите страницу документации. При сохранении изменённых настроек модуль очищает кеш автоматически.

Если операция не появилась, проверьте её активность, включение соответствующего набора роутов и разрешения групп. Отсутствие операций может означать, что текущему пользователю нечего показывать.

Собственный интерфейс

Способ замены интерфейса описан в разделе «Рендер документации».

Access- и refresh-токены, выпущенные через API, проверяют привязку к клиенту при каждом использовании. При несовпадении возвращается `AUTH_INVALID` (HTTP 401). Сохраняйте одинаковые заголовки клиента при выпуске, обновлении и вызове защищённых роутов. Состав заголовков и исключение для ручных токенов.

Работа через API

Для встроенных операций включите нативные роуты. В таблице пути указаны относительно адреса API.

Метод	Путь	Авторизация	Назначение
POST	/token	Basic	Выпустить пару для текущего клиента
GET	/token	Basic	Получить список активных токенов-сессий пользователя
PUT	/token	Bearer с refresh-токеном	Заменить прежнюю пару новой
DELETE	/token	Bearer с access-токеном	Отозвать токены клиента, указанного в параметре <code>client_id</code>

Откройте эти операции во встроенной документации: там приведены параметры, примеры ответов и порядок выполнения запросов. Для удаления берите `client_id` из списка сессий. Можно завершить свою сессию или сессию другого клиента этого же пользователя.

Если при выпуске получена ошибка `AUTH_MFA_REQUIRED`, пройдите многофакторную авторизацию на штатной форме сайта и повторите запрос с логином, паролем и cookie подтверждённой сессии того же пользователя.

Административный список

Поля и поиск

Каждая строка соответствует одному токenu. Одна пара представлена двумя записями. Список содержит и истёкшие токены, пока их записи не удалены.

Поле	Назначение
------	------------

Пользователь	Владелец токена
Дата создания и Дата окончания	Время выпуска и срок действия конкретного токена
Тип токена	<code>access</code> или <code>refresh</code>
ID клиента, Название клиента, Тип клиента	Сведения о клиентской сессии
Агент	Сведения User-Agent
IP-адрес	Адрес, сохранённый при выпуске токена
Хеш токена	Хешированное значение; для авторизации не используется
ID связанного токена	Ссылка на второй токен пары

Используйте фильтры по пользователю, датам и типу токена, а строку поиска — по названию клиента. Дополнительные столбцы можно включить в настройках списка. Нажатие на ID открывает карточку токена.

Ручной выпуск

Предпочтителен выпуск через API: запрос поступает от самого клиента и содержит его сведения. При необходимости создать пару вручную:

1. Нажмите «Добавить токен».
2. Выберите пользователя и укажите будущую дату окончания access-токена.
3. Сохраните форму. Refresh-токен получит срок из настроек безопасности.
4. Сохраните показанные значения access- и refresh-токенов: после обновления или ухода со страницы они больше не отображаются.

Токены, выпущенные вручную, можно использовать в другом приложении. Они не привязаны к заголовкам клиента; срок действия и возможность отзыва сохраняются.

Ручной выпуск для пользователя с ID = 1 запрещён. Действие копирования в списке открывает форму выпуска для того же пользователя; оно создаёт новую пару, а не копирует секреты прежней.

Изменение и удаление

У существующей записи можно изменить «Дату окончания», «Название клиента» и «Агент». Изменение относится только к выбранной записи: срок второго токена пары автоматически не меняется. Это не ротация — значение токена остаётся прежним.

Действие «Удалить» отзывает выбранный токен вместе с парным. Поддерживается удаление выделенных записей и всех записей, соответствующих фильтру. Перед массовым удалением проверьте выборку.

Для отзыва доступа используйте удаление пары: сокращение срока только access-токена не запрещает получение новой пары по ещё действующему refresh-токену.

Журналирование

Запись операций включается на вкладке «Логирование». Просмотр доступен через «События токенов» и «События модуля» при наличии прав системного журнала.

Входящие запросы

Раздел «Сервисы → API-платформа → Входящие запросы» содержит три списка:

- «Список запросов» — история обращений с пользователем, IP-адресом, методом и статусом.
- «Входящие данные» — сохранённые параметры и заголовки входящих запросов.
- «Исходящие данные» — сохранённые ответы API и их заголовки.

Для просмотра всех списков и карточки требуется право чтения входящих запросов, для удаления записей — также право изменения на вкладке «Доступ».

Состав истории

Связь истории и данных:

```

1  REQUEST_ID
2  |─ Основная запись запроса
3  |   |─ Входящие данные в базе → карточка запроса
4  |   |─ Исходящие данные в базе → карточка запроса
5  |─ файлы: native.api/REQUEST_RECEIVED/{REQUEST_ID}/
6  |   |─ request.json
7  |   |─ response.json или response.xml

```

Данные и файлы создаются только при включённых направлениях записи.

Поля записи

Поле	Что хранится
ID	Идентификатор записи в базе
Дата запроса	Время постановки записи в историю после отправки ответа
Идентификатор запроса	Идентификатор для поиска запроса и связанных с ним данных
Пользователь	Пользователь, авторизованный в API; без авторизации — «Неавторизован»
Метод	HTTP-метод запроса
Статус	«Обработан» или «Отклонен» — результат прохождения этапа проверки лимита

IP-адрес	Адрес отправителя
----------	-------------------

Путь, параметры, заголовки и ответ хранятся отдельно, если включена запись входящих и исходящих данных. HTTP-код ответа и длительность обработки отдельными полями истории не сохраняются.

Когда появляется запись

Включите «Собирать входящие запросы» на вкладке «Аналитика». История будет пополняться независимо от лимитов и выбранных для статистики HTTP-методов: в том числе при отказах, **OPTIONS** и досрочных ответах обработчиков событий.

При действующем для клиента лимите основная запись создаётся и с выключенным сбором. Если выключены и сбор, и лимит, запись не создаётся. Один запрос создаёт одну запись истории. До успешной API-авторизации запрос записывается без пользователя и учитывается по IP-адресу.

Запись может появиться после получения ответа клиентом. При аварийном завершении обработки она может не сохраниться. Ошибки, возникшие до запуска API, в историю не попадают.

Статусы запросов

- «Обработан» (**PROCESSED**) — запрос прошёл этап проверки лимита, в том числе при отключённом лимите. Последующая ошибка параметров или контроллера не меняет этот статус.
- «Отклонен» (**REJECTED**) — запрос превысил лимит или завершился до этого этапа. Такой статус получают и успешный **OPTIONS**, и досрочный ответ события до проверки лимита.

Статус показывает участие в расчёте лимитов, а не успешность HTTP-ответа. Для разбора отказов используйте «Журнал безопасности», для ошибок и других событий — «События модуля».

Поиск запросов

В основном списке используйте фильтры по идентификатору запроса, пользователю, IP-адресу, HTTP-методу, статусу и дате. В поле пользователя можно указать ID или данные для поиска; значение «Неавторизован» отбирает обращения без API-авторизации.

Строка поиска основного списка ищет IP-адрес, HTTP-метод или значение «Неавторизован». Для разбора превышения лимита выберите пользователя либо IP-адрес, период и статус «Обработан».

В списках данных доступны дата и `REQUEST_ID` ; для входящих данных — также URI и Content-Type, для исходящих — формат ответа. Строка поиска этих двух списков ищет точный `REQUEST_ID` . Заголовки и содержимое не включаются в колонки, фильтры, сортировку и экспорт списков.

Карточка запроса

Нажмите ID или идентификатор запроса в основном списке. Карточка содержит три вкладки: общие сведения, входящие данные и исходящие данные. Все поля доступны только для чтения. В обоих направлениях сначала показаны заголовки, затем данные; HTML и XML выводятся как текст.

В списках данных ссылка по `REQUEST_ID` открывает карточку основного запроса, а ID строки — соответствующую вкладку входящих или исходящих данных.

Входящие данные

Сохраняются данные, полученные до изменений пользовательскими обработчиками:

- `URI` — путь запроса без GET-параметров.
- `CONTENT_TYPE` — значение входящего Content-Type; если заголовка нет, поле пустое.
- `HEADER` — заголовки в формате JSON.
- `DATA` — JSON с `QUERY` , `POST` , `JSON` и `FILES` : параметры URL, поля формы, разобранные JSON-параметры и метаданные файлов.

Исходное тело запроса отдельно не копируется. Для multipart сохраняются поля и метаданные, без содержимого файлов и временных путей на сервере.

Исходящие данные

Сохраняется итоговый ответ, отправленный клиенту: формат `json` / `xml` , заголовки в JSON и тело ответа после маскирования. Формат определяется по Content-Type ответа; для нестандартного формата поле пустое. Для `HEAD` и статусов, не допускающих тело, содержимое пустое.

Учитываются заголовки ответа модуля и отправленные PHP-заголовки. Заголовки, добавленные внешним прокси или веб-сервером, модулю недоступны. Правила маскирования данных одинаковы для базы и файлов.

Если данных нет

Карточка читает данные только из базы. При записи исключительно в файл вкладка сообщает об отсутствии данных; искать их нужно в файловом журнале запроса. Проверьте включение сбора и нужного направления записи в момент выполнения запроса: старые данные не восстанавливаются задним числом.

Связь с ограничениями

Подсчёт обращений

Лимиты задаются на вкладке «Безопасность». Учитываются только записи со статусом «Обработан»: для авторизованных запросов — по пользователю, для неавторизованных — по IP-адресу. При положительном периоде учитываются записи за этот период; при периоде 0 — вся сохранённая история с этим статусом для выбранного признака.

HTTP-методы не разделяются на отдельные лимиты. Записи «Отклонен», включая **OPTIONS** и ответы до проверки лимита, в подсчёт не входят. Фильтр списка не влияет на расчёт.

Отличие от аналитики

Данные	Назначение	Управление
Список запросов	История обращений и расчёт лимитов по статусу «Обработан»	Флаг сбора или действующий лимит
Входящие и исходящие данные	Разбор параметров, заголовков и ответа	Сбор запросов и отдельные флаги записи в базу или файлы
Аналитика	Счётчики HTTP-методов на дашборде	Выбор методов; «Очистить статистику» обнуляет счётчики
События модуля	Причины ошибок и аудит событий	Отдельные флаги системного журнала и файлов

Очистка счётчиков не удаляет историю и данные запросов. Удаление истории не обнуляет счётчики.

Удаление записей

В основном списке можно удалить одну запись через меню строки, несколько выделенных или все записи по текущему фильтру. Вместе с запросом удаляются связанные входящие и исходящие данные в базе. Файлы запроса сохраняются.

Удаление обработанных запросов влияет на лимит: обращения перестают участвовать в подсчёте, и у пользователя или IP-адреса может снова появиться возможность выполнять запросы. Проверяйте фильтр перед массовым удалением. Счётчики аналитики и системный журнал сохраняются.

При удалении модуля очистка входящих запросов удаляет историю и сохранённые в базе данные запросов и ответов. Файловые журналы очищаются отдельной опцией.

Журнал безопасности

Раздел «Сервисы → API-платформа → Журнал безопасности» открывает системный журнал событий платформы с фильтром по модулю и отказам по ограничениям доступа. Он помогает установить, какое правило отклонило запрос.

Подготовка

Запись событий

На вкладке «Логирование» включите запись нужных отказов в системный журнал. Флажок «Записывать в файл» работает независимо и сам по себе не добавляет записи в этот список.

События сохраняются с момента включения записи. Отсутствие записей за прошлый период не подтверждает отсутствие отказов, если журналирование было выключено.

Права просмотра

Право чтения журналов на вкладке «Доступ» показывает пункт меню. Для открытия самого журнала нужен доступ к системному журналу событий платформы. Это отдельное разрешение: права модуля его не предоставляют.

Причины отказов

Событие	Что проверить
Отказ по Blacklist групп	Не входит ли пользователь в запрещённую группу роута
Отказ по Whitelist групп	Есть ли у пользователя разрешённая группа роута
Отказ по Blacklist IP-адресов	Не включён ли адрес в глобальный или маршрутный чёрный список
Отказ по Whitelist IP-адресов	Проходит ли адрес и глобальный, и маршрутный белый список
Запрет по режиму работы	Попадает ли время запроса в разрешённое расписание
Превышение лимита запросов	Не исчерпано ли число обращений для пользователя или IP-адреса

Отказ CORS	Разрешены ли источник страницы и запрошенные заголовки
------------	--

Группы и IP-списки роута задаются в его карте. Глобальные IP-списки, расписания, лимиты и CORS — на вкладке «Безопасность». При проверке расписания учитывайте время сервера PHP.

Разбор отказа

1. Выберите период, соответствующий проблемному запросу.
2. Уточните фильтр по типу события, пользователю или IP-адресу.
3. Посмотрите описание записи: состав сведений зависит от причины отказа.
4. Сопоставьте событие с ответом API и действующими правилами доступа.
5. После исправления настроек повторите запрос и проверьте результат.

Для превышения лимита дополнительно откройте «Входящие запросы». При отказе CORS сравнивайте настройки с адресом страницы, из которой выполняется запрос.

Журнал не содержит всех возможных ошибок API. Например, ошибку авторизации, отсутствие роута или ошибку контроллера ищите в «Событиях модуля» при включённой записи ошибок.

Хранение записей

Срок хранения и очистка системного журнала управляются платформой. Файловые журналы хранятся отдельно в настроенном каталоге и не отображаются в этом списке.

События модуля

Раздел «Сервисы → API-платформа → События модуля» открывает системный журнал событий платформы с фильтром по модулю. Здесь можно проследить изменения настроек, операции с токенами и причины ошибок при работе API.

В этот список также входят записанные события безопасности. Для просмотра только отказов по ограничениям используйте «Журнал безопасности».

Данные входящих запросов и ответов просматриваются отдельно — в списках и карточке запроса. Их основные флажки на вкладке «Логирование» сохраняют данные в таблицах модуля, а не в этом системном журнале.

Запись и доступ

Нужные типы событий включаются на вкладке «Логирование». Для отображения в административном журнале включите флажок самого события. Отдельный флажок «Записывать в файл» управляет файловой копией. Записи за время, когда журналирование было выключено, не восстанавливаются задним числом.

Право чтения журналов в модуле показывает пункт меню, а для просмотра записей нужен отдельный доступ к системному журналу событий платформы. Подробнее — в разделе «Доступ».

Типы событий

Настройки и работа API

Событие	Назначение
Изменение настроек модуля	Проверить, какие настройки были изменены
Очистка кеша модуля	Найти выполнение действия «Очистить кеш»
Ошибка запуска модуля	Изучить ошибки, зарегистрированные при обработке API-запроса

Название «Ошибка запуска модуля» используется и для ошибок обработки запросов, например неверной авторизации или отсутствующего роута. Смотрите код и описание конкретной записи, а не только название типа события.

Токены

Событие	Назначение
Создание токена	Найти добавление записи access- или refresh-токена
Изменение токена	Сравнить изменённые значения до и после сохранения
Удаление токенов клиента	Найти отзыв токенов клиента через API
Смена ключа шифрования токенов	Установить момент генерации нового ключа в настройках

Одна пара состоит из двух записей, поэтому её выпуск создаёт отдельные события для каждого токена. Событие «Удаление токенов клиента» относится к отзыву через API; оно не является журналом каждого удаления строки из административного списка.

Действия с токенами описаны в разделе «Токены авторизации».

Обработчики событий

Событие «Вызов пользовательского обработчика» помогает разобрать работу расширений модуля: какое событие обрабатывалось, какие обработчики вызывались и как изменились данные. Запись включается флажком «Логировать пользовательские обработчики».

Этот режим полезен при диагностике пользовательских сценариев. После разбора ошибки его можно отключить, чтобы не накапливать лишние подробности.

Поиск причины ошибки

1. Выберите период и тип события. При необходимости уточните пользователя или IP-адрес.
2. Сопоставьте время записи с запросом клиента или действием администратора.
3. Изучите описание: для ошибки оно может содержать код, HTTP-статус и сообщение, для изменения — значения полей до и после операции.
4. Исправьте причину и повторите действие.

Если записей нет, проверьте выбранный период, фильтры и включение соответствующего события в системный журнал. При записи только в файл искать результат нужно в каталоге файловых журналов.

Срок хранения системного журнала задаётся платформой. Очистка кеша модуля и счётчиков аналитики не удаляет его записи.

Основные

Вкладка «Основные» находится в разделе «Сервисы → API-платформа → Настройки модуля». Здесь задаются пути API, документации, пользовательской карты роутов и файловых журналов.

Настройка путей

Указывайте пути относительно корня сайта, без домена. Серые примеры внутри пустых полей служат подсказками и не означают, что значение уже сохранено.

Путь интерфейса

Базовый путь для запросов к API. Например, значение `api` задаёт адрес `/api/`, а `api/v1` — `/api/v1/`. Адрес конкретного роута добавляется к этому пути.

Заполненное поле включает API. Если очистить его и сохранить настройки, API отключится, а зарегистрированная точка входа будет удалена.

При смене пути обновите адрес API в подключённых клиентах: старый адрес перестанет обслуживаться модулем.

Путь документации

Путь к странице встроенной документации. Например, `api/docs` задаёт адрес `/api/docs/`. Это полный путь относительно корня сайта: он не добавляется автоматически к пути интерфейса.

Пустое значение отключает эту точку входа. Настройки доступа и работы со страницей описаны в разделе «Документация».

Для API и документации выбирайте разные пути. Они не должны совпадать с существующими каталогами сайта; путь `rest` зарезервирован платформой. Не используйте пробелы, обратные косые черты и сегменты `.` или `..`. Точки входа создаются при сохранении — вручную создавать каталоги для них не нужно.

Модуль автоматически выбирает способ обработки адресов. Если после сохранения API недоступен, проверьте имя сервера в настройках сайта и правила обработки URL на веб-сервере.

Путь карты роутов

Путь к существующему PHP-файлу пользовательской карты, например `local/api/routes.php`. Файл должен находиться внутри сайта и возвращать массив роутов.

Пользовательская карта дополняет встроенные роуты. При совпадении адреса и HTTP-метода используется описание из пользовательской карты. Пустое поле оставляет только встроенные роуты, включённые на вкладке «Роуты».

После изменения содержимого файла нажмите «Очистить кеш», чтобы модуль заново загрузил карту.

Путь к логам

Каталог для файловых журналов. Если поле пустое, используется `local/logs`. К выбранному пути модуль добавляет собственный каталог `native.api`. События записываются в `YYYY-MM-DD/EVENT_CODE.log`, а данные запросов и ответов — в `REQUEST_RECEIVED/{REQUEST_ID}/request.json` и `response.json` или `response.xml`.

Можно хранить журналы вне каталога сайта: `../logs` задаёт каталог на уровень выше, а `../../logs` — на два уровня выше корня сайта. Выход выше корня разрешён только для журналов.

Путь указывается без завершающей косой черты. Абсолютные пути, пробелы и символьные ссылки не допускаются. PHP должен иметь право записи в выбранное расположение; ограничения сервера также учитываются при сохранении. При сохранении пути и перед записью логов модуль создаёт в своём каталоге `native.api` файл `.htaccess` с запретом прямого HTTP-доступа, если такого файла ещё нет. Существующий `.htaccess` остаётся без изменений, даже если он пустой или разрешает доступ. Если создать файл не удалось, путь не сохраняется или соответствующая файловая запись не выполняется.

Запрет работает в Apache 2.4 при разрешённой обработке `.htaccess`. Если файлы отдаёт Nginx, доступ нужно закрыть в его конфигурации. После настройки проверьте, что существующий файл журнала нельзя прочитать по прямому URL. Правило не запрещает работу PHP и доступ через SSH/SFTP при наличии файловых прав.

Новый путь применяется к новым штатным записям; ранее созданные файлы и их `.htaccess` не переносятся и не удаляются при смене пути. Резервный журнал ошибки запуска использует `local/logs/native.api` независимо от выбранного пути; для него также создаётся отсутствующий `.htaccess`. Проверьте HTTP-защиту резервного и прежних каталогов. Выбор событий для записи выполняется на вкладке «Логирование».

Разрешить сбор данных

Флажок сохраняет разрешение на передачу технических данных разработчику для улучшения модуля. Сейчас сбор и передача данных не выполняются; изменение флажка не влияет на работу API.

Безопасность

Вкладка «Безопасность» находится в разделе «Сервисы → API-платформа → Настройки модуля». Она задаёт сроки действия токенов и ограничения для входящих API-запросов.

Токены авторизации

Срок access-токена

Время действия токена доступа в секундах. Пустое поле использует значение по умолчанию — **10800** секунд, или 3 часа.

Срок refresh-токена

Время действия токена обновления в секундах. Пустое поле использует значение по умолчанию — **2592000** секунд, или 30 дней. Оно не может быть меньше срока жизни access-токена.

Для обоих полей допустимо целое число от **1** до **3153600000** секунд. Ноль не допускается. Срок отсчитывается от выпуска токена. Изменение настройки применяется к новым парам, в том числе при обновлении; сроки уже выпущенных токенов не пересчитываются.

Выпуск, обновление и отзыв отдельных клиентов описываются в разделе «Токены авторизации».

Ключ шифрования

Так поле называется в интерфейсе. Ключ используется для хеширования токенов перед хранением в базе.

Кнопка «Сгенерировать ключ» после подтверждения немедленно делает недействительными все ранее выпущенные access- и refresh-токены всех клиентов. Для продолжения работы клиентам потребуется получить новые пары токенов. Генерация требует права изменения настроек.

Расписание

Дни и время

В строке «Все пользователи» задайте общее расписание: отметьте разрешённые дни недели и заполните «начало» и «окончание» в формате **ЧЧ:ММ**. Используется время сервера, на котором выполняется PHP.

- По умолчанию доступ разрешён каждый день круглосуточно.
- Для круглосуточной работы в выбранные дни укажите **00:00** в обоих полях.
- Одинаковые начало и окончание означают отсутствие ограничения по времени внутри выбранных дней.
- Окончание не может быть раньше начала; интервал через полночь одной строкой не задаётся.
- Если дни не выбраны, эта строка не разрешает доступ ни в один день.

Расписания групп

Расписания групп дополняют общий режим. Для допуска достаточно одного подходящего правила:

```

1 Текущие день и время сервера
2 └─ Подходят общему расписанию → доступ по времени разрешён
3 └─ Не подходят
4 └─ Подходят расписанию группы → доступ по времени разрешён
5 └─ Не подходят ни одному → HTTP 403

```

Например, общий режим можно ограничить буднями с **09:00** до **18:00**, а отдельной группе разрешить круглосуточный доступ во все дни. При общем круглосуточном доступе более узкое расписание группы его не ограничит.

Лимиты запросов

Количество и период

В поле «количество» задайте число запросов, а в поле «секунды» — период их учёта. Например, **100** и **60** задают ограничение в 100 запросов за последние 60 секунд.

Значение	Поведение
«количество» пустое или равно 0	Правило отключено
«секунды» пустое	Правило отключено
«секунды» равно 0 , количество больше 0	Учитывается вся сохранённая история без ограничения по времени

Запросы авторизованного пользователя учитываются совместно для его клиентов и IP-адресов. Для запросов без авторизации учёт ведётся по IP-адресу. История доступна в разделе «Входящие запросы»; для расчёта учитываются только записи со статусом «Обработан». Отказы лимитера и запросы, завершённые до проверки лимита, включая **OPTIONS**, в подсчёт не входят. Удаление обработанных записей влияет на расчёт лимитов.

При превышении возвращается HTTP 429. Для периода больше нуля заголовок **Retry-After** указывает, через сколько секунд можно повторить запрос. При одновременных запросах возможно превышение настроенного порога.

Лимиты групп

- 1 | Есть активные лимиты групп пользователя?
- 2 | └ Нет → правило «Все пользователи»
- 3 | └ Да → только подходящие групповые правила
- 4 | └ Количество → максимум из правил
- 5 | └ Период → максимум из правил

Количество и период выбираются независимо: итог может объединять значения разных групп.

Например, правила 100 за 120 секунд и 200 за 60 секунд дадут 200 за 120 секунд. Отключённое правило группы не отменяет общий лимит.

Расписания и групповые лимиты используют пользователя, определённого авторизацией API. Открытая административная сессия сама по себе не подменяет эту авторизацию.

CORS

CORS настраивает обращения к API из JavaScript на другом сайте. Эти настройки не заменяют авторизацию: запрос без заголовка **Origin** не проверяется по списку источников.

Кеширование CORS

Срок, на который браузер может запомнить результат предварительного запроса OPTIONS. Пустое поле использует **3600** секунд; **0** отключает кеширование этой проверки. Браузер может ограничить фактический срок. На кеширование ответов роутов поле не влияет.

Разрешённые Origin

Укажите по одному источнику на строку: протокол, домен и при необходимости порт, без пути страницы. Поддерживаются точные адреса и шаблоны поддоменов:

```
1 | https://app.example.com
2 | https://*.example.org
```

Шаблон во второй строке разрешает, например, **https://shop.example.org**, но не сам **https://example.org**. Значение ***** разрешает любой источник и указывается отдельно от других значений. Пустое поле также использует ***** — оно не запрещает обращения с других сайтов. Запрос с запрещённым Origin отклоняется с HTTP 403.

Разрешённые заголовки

Дополнительные заголовки, которые браузер может передавать в запросе, — по одному на строку. Например, **X-Client-Version**.

Origin, **Content-Type**, **Authorization** и **X-Requested-With** разрешены всегда. Пустое поле оставляет этот базовый набор; пользовательские значения дополняют его.

Заголовки ответа

Заголовки ответа, которые JavaScript на другом сайте сможет прочитать, — по одному на строку. Например, **X-Total-Count**. Заголовок **Retry-After** доступен всегда. Поле не создаёт заголовок: он должен присутствовать в ответе API.

Ограничения по IP

Эти списки действуют на весь API и проверяются до выбора роута, в том числе для `OPTIONS`. Дополнительные IP-ограничения конкретного роута задаются в его карте. Запрос должен пройти оба уровня.

Указывайте по одному IPv4-, IPv6-адресу или подсети CIDR на строку. Например:

```
1 192.0.2.10
2 198.51.100.0/24
3 2001:db8::/32
```

Чёрный список IP

Запрещает запросы с перечисленных адресов и подсетей. Проверяется первым; совпадение запрещает доступ даже при наличии адреса в белом списке.

Белый список IP

Если заполнен, разрешает запросы только с перечисленных адресов и подсетей. Пустой список не вводит дополнительных ограничений. При двух пустых глобальных списках остаются только IP-ограничения, заданные в карте выбранного роута.

Проверяется адрес из `REMOTE_ADDR`. Заголовки `X-Forwarded-For` и `Forwarded` не учитываются. Если сайт работает за прокси, учитывайте адрес, который веб-сервер передаёт PHP. Запрет по IP возвращает HTTP 403.

Роуты

Вкладка «Роуты» находится в разделе «Сервисы → API-платформа → Настройки модуля». Здесь включаются встроенные роуты и задаются общие параметры кеширования и ответа.

Активация роутов

Три группы роутов включаются независимо. Отключённые группы исключаются из карты и встроенной документации. Включение сохраняет требования авторизации и ограничения доступа.

Нативные роуты

Включает встроенные операции с токенами: выпуск, обновление, просмотр сессий и отзыв клиента. Используйте этот параметр, если подключённые приложения управляют токенами через API.

Работа с ними описывается в разделе «Токены авторизации». Отключение этих роутов само по себе не отзывает уже выпущенные токены.

Сервисные роуты

Включает роуты для проверки работы интерфейса, в том числе `GET /up`. Они помогают проверить обработку запросов без создания собственных контроллеров.

После включения проверьте доступные операции через «Документацию». Если сервисные роуты больше не нужны, отключите их для дополнительной безопасности: это закроет доступ к диагностическим операциям и сведениям о выполнении запросов.

Роут-провайдер

Включает операции `GET /provider` и `POST /provider`: просмотр описания разрешённых методов и их выполнение через API. Для обращения требуется access-токен.

Доступны только классы и методы, разрешённые провайдером; явные запреты имеют приоритет. Само включение флажка не открывает доступ ко всем методам платформы.

Настройки роутов

Время кеширования

Срок хранения результата контроллера в секундах. Общее значение применяется к GET- и HEAD-роутам, если для них не задано собственное правило кеширования.

Значение	Поведение
Положительное целое число	Включает общее кеширование на указанный срок
0 или пустое поле	Отключает общее кеширование результатов

Например, 300 означает 5 минут, а 86400 — 24 часа. Число 86400 в подсказке пустого поля не применяется автоматически. Кеш карты роутов работает независимо от этого поля.

Приоритет настроек

```

1  В карте роута задан ttl?
2  └─ Да
3     └─ ttl > 0 → кешировать на этот срок
4        └─ ttl = 0 → не кешировать
5
6  └─ Нет
7     └─ GET или HEAD → взять общее значение
8        └─ Другой метод → не кешировать

```

Индивидуальный срок действует для любого HTTP-метода, включая POST. При попадании в кеш контроллер не выполняется, но доступ и параметры проверяются. Пример настройки и порядок обработки. Для действий, которые должны выполняться при каждом запросе, отключите кеширование.

Технические поля

По умолчанию модуль добавляет к ответу служебные поля `request_id`, `status` и `time`. Непустой результат контроллера помещается в `result`.

Включённый флажок убирает эту обёртку из успешных ответов: JSON, XML. В JSON данные контроллера становятся корневой структурой; в XML также убирается внешний элемент `response`. Это позволяет формировать ответ под требования подключаемой системы.

Для XML контроллер должен вернуть один корневой элемент. HTTP-статус и заголовки продолжают применяться. Ошибки, сформированные модулем, сохраняют техническую структуру.

Поведение можно переопределить для отдельного роута в его карте. Индивидуальная настройка имеет приоритет над флажком на этой вкладке.

Аналитика

Вкладка «Аналитика» находится в разделе «Сервисы → API-платформа → Настройки модуля». Здесь включается история входящих запросов, выбираются HTTP-методы для подсчёта и очищается накопленная статистика.

Сбор входящих запросов

Флажок «Собирать входящие запросы» включает историю обращений, в том числе отказы, **OPTIONS** и досрочные ответы событий. По умолчанию он выключен. Сбор не зависит от выбранных ниже HTTP-методов для статистики.

При действующем для клиента лимите основная запись запроса сохраняется и без этого флажка. При отключённом сборе и лимите история не пополняется.

Для сохранения параметров, заголовков и ответов дополнительно включите нужные направления на вкладке «Логирование». Без включённого сбора эти журналы не записываются ни в базу, ни в файлы, даже если действует лимит. Изменение флажка сбора не меняет флажки логирования и наоборот.

Сбор статистики

HTTP-методы

Отметьте флажки «Собирать статистику ...-запросов» для нужных методов: **GET** , **POST** , **PUT** , **PATCH** , **DELETE** , **OPTIONS** и **HEAD** . Для каждого метода ведётся отдельный счётчик. Если ничего не выбрано, сбор отключён.

Учитываются обращения, дошедшие до обработчика API модуля, включая запросы с отказом в доступе, ошибкой или ответом из кеша. Поэтому счётчики показывают количество обращений, а не только успешных выполнений контроллера.

Накопленные данные

После включения учитываются новые обращения. Ранее выполненные запросы не добавляются задним числом. Отключение метода останавливает его подсчёт, но сохраняет накопленное значение.

Общее количество и распределение по HTTP-методам отображаются на дашборде.

Статистика хранится отдельно от входящих запросов, которые используются для лимитирования. Отключение сбора статистики не отключает ограничения по запросам.

Очистка статистики

Кнопка «Очистить статистику» в строке «Накопленная статистика» обнуляет счётчики всех HTTP-методов, включая отключённые. Перед удалением требуется подтверждение. Отменить очистку нельзя.

После очистки подсчёт продолжается для выбранных методов. История входящих запросов и журналы событий сохраняются.

Для очистки требуется право изменения настроек. Кнопка не сохраняет изменения флажков: новый набор методов нужно сохранить отдельно.

Документация

Вкладка «Документация» находится в разделе «Сервисы → API-платформа → Настройки модуля». Здесь задаются общие сведения об API для рендера документации. Описания роутов и их параметров формируются из карты роутов.

Все поля необязательны. Для незаполненных полей используются предусмотренные значения по умолчанию; дополнительные сведения без значений, например контакты, не отображаются.

После изменения полей сохраните настройки. Кеш модуля, включая встроенную документацию, очищается автоматически; при следующем открытии она сформируется с новыми значениями.

Адрес страницы задаётся на вкладке «Основные», а разрешение на просмотр — на вкладке «Доступ».

Настройки документации

Заголовки и описание

Поле	Назначение	Если не заполнено
Заголовок страницы	Название страницы в шаблоне сайта	Документация
Заголовок	Название API в документации	Документация
Описание	Краткое описание назначения API	Описание API-методов интерфейса
Версия	Версия вашего API, отображаемая в документации	Текущая версия модуля

Например, для интеграции с мобильным приложением можно указать заголовок «API личного кабинета», а в описании перечислить основные доступные операции. Поле «Версия» меняет только сведения в документации: оно не переключает версию роутов или модуля.

Контакты

Укажите, к кому обращаться с вопросами по вашему API.

Поле	Что указать
Контакты: Заголовок	Имя ответственного или название службы поддержки
Контакты: URL	Ссылку на страницу поддержки
Контакты: E-mail	Электронную почту для обращений

Незаполненные контактные поля не добавляются в описание API.

Условия и лицензия

Поле	Что указать
Условия: URL	Ссылку на условия использования API
Лицензия: Заголовок	Название лицензии
Лицензия: URL	Ссылку на текст лицензии

В полях URL для контактов, условий и лицензии можно указать полный адрес с `https://` или путь относительно сервера, например `support/`. Относительные ссылки дополняются адресом из поля «Сервер: URL». Пустые поля не создают ссылки автоматически.

Сервер

Сервер: Заголовок

Название сервера в документации, например «Основной сервер» или «Тестовый стенд». Если поле пустое, используется название сайта из настроек главного модуля 1С-Битрикс.

Сервер: URL

Базовый адрес сервера, к которому рендер документации отправляет запросы. Укажите адрес с протоколом, например `https://example.ru`. Если поле пустое, модуль использует имя сервера из настроек сайта и протокол текущего запроса.

Путь интерфейса добавляется автоматически. Например, при адресе `https://example.ru` и пути `api` запросы отправляются на `https://example.ru/api`. Повторно добавлять `api` в поле «Сервер: URL» не нужно.

Этот параметр задаёт адрес запросов в документации. Саму точку входа API он не переносит. Если документация и API находятся на разных источниках, настройте CORS для адреса страницы документации.

После настройки откройте документацию и проверьте заголовки, ссылки и сервер, выбранный для выполнения запросов.

Логирование

Вкладка «Логирование» находится в разделе «Сервисы → API-платформа → Настройки модуля». Здесь включаются журналы входящих и исходящих данных, выбираются события для аудита и места хранения записей.

Способы записи

Для событий модуля, токенов и безопасности используются два независимых флажка:

```

1 | Событие
2 | └─ Флажок события → системный журнал Битрикса
3 |   └─ «События модуля» → все записанные события модуля
4 |     └─ «Журнал безопасности» → только отказы по ограничениям
5 |
6 | └─ «Записывать в файл» → отдельный файл события

```

Можно включить один способ или оба. Если оба выключены, событие не записывается. Отключение записи не отменяет проверок доступа.

Для входящих и исходящих данных основная галочка сохраняет данные для просмотра в карточке запроса, а «Записывать в файл» — в файлах запроса. Эти данные не направляются в системный журнал событий.

Журналы в админке

«События модуля» и «Журнал безопасности» открывают один системный журнал с разными фильтрами. Для просмотра нужны права системного журнала Битрикса. Право чтения журналов в модуле управляет только ссылками в меню.

Данные запросов доступны в разделе «Входящие запросы» и его карточке. Для них действует право чтения входящих запросов.

Файловые журналы

Файлы сохраняются в каталоге, заданном в поле «Путь к логам». Там же описаны создание защитного `.htaccess`, сохранение существующих правил и настройка Nginx. События сохраняются в `native.api/YYYY-MM-DD/EVENT_CODE.log`, где `EVENT_CODE` — код события. Данные запроса хранятся в отдельном каталоге:

```

1 | native.api/REQUEST_RECEIVED/{REQUEST_ID}/
2 | └─ request.json

```

3 | response.json или response.xml

`request.json` содержит дату, `REQUEST_ID`, путь `URI`, `CONTENT_TYPE`, заголовки `HEADER` и параметры `DATA`. Файл ответа содержит только итоговое тело после маскирования, без заголовков, служебных полей и дополнительной обёртки. Заголовки ответа сохраняются в базе.

Для XML-ответа создаётся `response.xml`, для JSON — `response.json`. При неизвестном формате используется `response.json` с замаскированным непустым содержимым; расширение в этом случае не означает, что внутри корректный JSON. Для `HEAD` и ответов без тела файл ответа пустой. Каждый файл создаётся только при включённом направлении записи.

В файловых записях модуль маскирует распознанные пароли, токены и другие секреты. К записям в системном журнале событий такая дополнительная обработка не применяется. Учитывайте это при выборе способа записи, особенно для данных пользовательских обработчиков.

Входящие запросы

1. На вкладке «Аналитика» включите «Собирать входящие запросы» и сохраните настройки.
2. В группе «Входящие запросы» вкладки «Логирование» выберите данные и места хранения.
3. Выполните новый API-запрос и откройте его карточку либо каталог файлов по `REQUEST_ID`.

Параметр	Основная галочка	«Записывать в файл»
Записывать входящие данные	Параметры и заголовки в базе	<code>request.json</code> с параметрами и заголовками
Записывать исходящие данные	Тело и заголовки ответа в базе	Только тело в <code>response.json</code> или <code>response.xml</code>

Все четыре флажка независимы и по умолчанию выключены. Сбор запросов включается отдельно.

- 1 | Собирать входящие запросы?
- 2 | — Да → сохранять все обращения к обработчику API
- 3 | | — Входящие данные → включённые направления: база / файл
- 4 | | — Исходящие данные → включённые направления: база / файл

- 5 | — Нет
- 6 | — Действует лимит → только основная запись запроса
- 7 | — Лимита нет → запись истории не создаётся

Журналы содержат полученные данные запроса и итоговый ответ клиенту. Если данные не появились в карточке, проверьте настройки и выбранное место хранения: поиск данных запроса.

Маскирование данных

Перед записью входящих и исходящих данных в базу и файлы маскируются распознаваемые логины, пароли, токены, ключи, Authorization и Cookie, включая вложенные JSON-значения и XML-элементы и атрибуты. Маска — *********. Если содержимое ответа невозможно безопасно обработать, оно целиком заменяется маской.

URI хранит путь без GET-параметров и не проходит строковое маскирование. Параметры URL сохраняются отдельно в **QUERY**, где чувствительные значения маскируются. Не передавайте секреты в сегментах пути. Маскирование не гарантирует распознавание секрета под произвольным именем и не заменяет ограничение доступа к журналам.

Это правило относится к журналам данных запросов и файловому аудиту. Системный журнал событий получает данные без дополнительного маскирования модулем.

События модуля

Параметр	Что фиксируется
Записывать обновление настроек модуля	Изменения сохранённых настроек
Записывать ошибки запуска модуля	Ошибки при запуске API
Записывать очистку кеша модуля	Очистка через кнопку «Очистить кеш»

Аудит обработчиков

Записывает вызовы пользовательских обработчиков событий модуля: название события, обработчики, входные данные и результат их работы.

Включайте запись при разработке и разборе ошибок. Она может содержать большой объём данных запроса; после диагностики отключите её, если постоянный сбор не нужен.

События токенов

Параметр	Что фиксируется
Записывать создание токена	Добавление записи токена
Записывать изменение токена	Изменённые значения записи до и после обновления
Записывать удаление токенов клиента	Удаление токенов при отзыве клиента через API
Записывать смену ключа шифрования	Генерация нового ключа в настройках безопасности

Операции с токенами описываются в разделе «Токены авторизации», а срок действия и смена ключа — на вкладке «Безопасность».

События безопасности

Для каждого ограничения можно включить отдельную запись отказов.

Параметр	Причина отказа
Записывать отказы по Blacklist групп	Группа пользователя входит в запрещённый список роута
Записывать отказы по Whitelist групп	У пользователя нет группы из разрешённого списка роута
Записывать отказы по Blacklist IP-адресов	IP-адрес входит в запрещённый список
Записывать отказы по Whitelist IP-адресов	IP-адрес отсутствует в разрешённом списке
Записывать отказы по режиму работы запросов	Запрос поступил вне разрешённого времени
Записывать превышение лимита запросов	Исчерпано допустимое количество запросов
Записывать отказы CORS	Источник или запрошенные заголовки не разрешены настройками CORS

Режим работы, лимиты, глобальные IP-списки и CORS задаются на вкладке «Безопасность». Ограничения по группам и IP-списки роута задаются в его карте. Параметры журналирования отказов по IP действуют на оба уровня проверки.

Передача данных вендору

Параметр предназначен для записи событий передачи технических данных разработчику. Сейчас сбор и отправка этих данных не выполняются, поэтому такие записи не создаются. Включение журналирования само по себе не разрешает передачу данных.

Разрешение задаётся отдельно на вкладке «Основные».

Доступ

Вкладка «Доступ» находится в разделе «Сервисы → API-платформа → Настройки модуля». Здесь назначаются права групп пользователей на разделы модуля и встроенную документацию. Вкладка доступна пользователям с правом изменения настроек.

Назначение прав

Чтение и изменение

«Чтение» разрешает просмотр раздела. «Изменение» разрешает действия с его данными. Эти права независимы: для работы с изменениями в административном разделе отметьте оба флажка.

Если пользователь состоит в нескольких группах, достаточно разрешения хотя бы у одной из них. Снятый флажок у другой группы не отменяет уже предоставленное право. Администраторам доступны все разделы независимо от выбранных групп.

Разделы модуля

Раздел	Чтение	Изменение
Рабочий стол	Просмотр дашборда и его персональная настройка	Отдельное право не предусмотрено
Документация	Просмотр встроенной документации API	Отдельное право не предусмотрено
Токены авторизации	Просмотр списка токенов	Действия с токенами в административном разделе
Входящие запросы	Просмотр трёх списков и карточки запроса	Удаление записей истории
Журнал безопасности и События модуля	Показ ссылок на оба журнала в меню модуля	Отдельное право не предусмотрено
Настройки	Просмотр настроек и очистка кеша модуля	Сохранение настроек, очистка статистики и генерация ключа шифрования

Право чтения входящих запросов распространяется на «Список запросов», «Входящие данные», «Исходящие данные» и карточку. Отдельных прав для параметров, заголовков и ответов нет. Удаление доступно в основном списке с

правом изменения; карточка открывается только для чтения.

Право чтения журналов управляет ссылками «Журнал безопасности» и «События модуля» в меню. Сам просмотр разрешается правами системного журнала событий платформы. Пользователь с таким доступом может открыть журнал напрямую, даже если ссылки в меню модуля скрыты.

Персональная настройка дашборда доступна с правом чтения «Рабочего стола» и не требует права изменения настроек модуля.

Доступ к документации

Все пользователи

По умолчанию флажок «Все пользователи» включён: документацию можно открыть без авторизации на сайте. Чтобы ограничить доступ, снимите его и отметьте группы, которым разрешён просмотр.

Если «Все пользователи» выключен и ни одна группа не выбрана, документация остаётся доступна только администраторам.

Выбранные группы

Посетитель должен войти на сайт под учётной записью из разрешённой группы. Разрешение читать документацию и видимость её роутов проверяются отдельно: в списке отображаются только операции, доступные группам пользователя. Если таких роутов нет, список операций будет пустым.

- 1 | Посетитель страницы документации
- 2 | └─ Право чтения → можно ли открыть страницу
- 3 | └─ Группы посетителя → какие операции видны
- 4 |
- 5 | API-запрос из документации
- 6 | └─ Authorize → данные Basic или Bearer
- 7 | └─ Проверки API → можно ли выполнить запрос

Authorize задаёт данные отправляемого запроса и не меняет группы посетителя страницы.

Права разделов и API

Разрешения на этой вкладке управляют интерфейсами модуля. Они не заменяют авторизацию API и ограничения, заданные в карте роутов и на вкладке «Безопасность».

Например, право просмотра документации не освобождает запрос от токена, если роут требует авторизацию. А право чтения списка токенов не разрешает их изменение.

После назначения прав проверьте доступ под учётной записью нужной группы. Учётная запись администратора для такой проверки не подходит: её права не ограничены этими флажками.

Контекст API

Контекст API хранит данные текущего запроса: параметры, роут и сведения об авторизованном пользователе. Контроллеры и обработчики событий используют один объект контекста.

Получение контекста

```
1 use Native\Api\Context;
2
3 $context = Context::getCurrent();
```

Данные запроса

Метод	Результат
getRequestId()	Идентификатор запроса
getRequestMethod()	HTTP-метод
isCli()	<code>true</code> , если модуль выполняется через PHP CLI
getParams()	Объединённые параметры адреса, строки запроса и тела
getRoute()	Текущий роут; до его выбора — <code>null</code>
getAuth()	Данные авторизации, включая User ID; без авторизации — <code>null</code>

Используйте `getRequestId()`, чтобы найти запрос в истории. Этот идентификатор связывает основную запись, входящие и исходящие данные и каталог файловых журналов. Метод `getRequestStatus()` возвращает `PROCESSED` или `REJECTED`: статус отражает прохождение проверки лимита.

Выполнение в CLI

`$context->isCli()` позволяет определить запуск через командную строку. В этом режиме обработчики `onAfterResponse` выполняются сразу после отправки ответа; их результат не меняет уже отправленный ответ.

Пользователь

После авторизации API контекст содержит ID пользователя Битрикса, от имени которого выполняется запрос:

```
1 | $userId = $context->getAuth()?->getUserId();
```

Без авторизации `$userId` равен `null`. Права пользователя на запрашиваемые данные проверяет контроллер.

Настройки роута

В контроллере полная конфигурация роута, включая пользовательские настройки, доступна через:

```
1 | $config = $context->getRoute()  
2 |     ->getConfig()  
3 |     ->toArray();
```

IP-списки выбранного роута можно получить без чтения полного конфига:

```
1 | $config = $context->getRoute()?->getConfig();  
2 | $whitelist = $config?->getSecurityIpWhitelist() ?? [];  
3 | $blacklist = $config?->getSecurityIpBlacklist() ?? [];
```

Назначение списков описано в безопасности роута.

События

В обработчики событий контекст передаётся в параметре `apiContext`. Метод `setParams()` позволяет изменить параметры запроса.

Получение настроек модуля

После подключения модуля получите объект настроек:

```
1 use Native\Api\Application\Providers\ModuleOptionsProvider;  
2 use Native\Api\Config\Internals\OptionConfig;  
3  
4 $options = ModuleOptionsProvider::getInstance();
```

Метод	Значение
<code>\$options->getPathModule()</code>	Путь API
<code>\$options->getPathRouteSettings()</code>	Путь карты роутов
<code>\$options->getRouteCacheTtl()</code>	Срок кеширования результата
<code>\$options->getTokenAccessTtl()</code>	Срок access-токена
<code>\$options->getTokenRefreshTtl()</code>	Срок refresh-токена

Строковая настройка:

```
1 $title = $options->getString(OptionConfig::DOCS_TITLE);
```

Значения TTL возвращаются в секундах. Ключи настроек перечислены в справочнике OptionConfig.

Подключение карты

Карта роутов — PHP-файл, возвращающий через `return` массив роутов, сгруппированных по HTTP-методам.

Подключение

1. Разместите файл карты внутри сайта, вне каталога модуля.
2. Откройте «Сервисы → API-платформа → Настройки модуля», вкладку «Основные».
3. В поле «Путь карты роутов» укажите путь к файлу относительно корня сайта.
4. Нажмите «Сохранить».

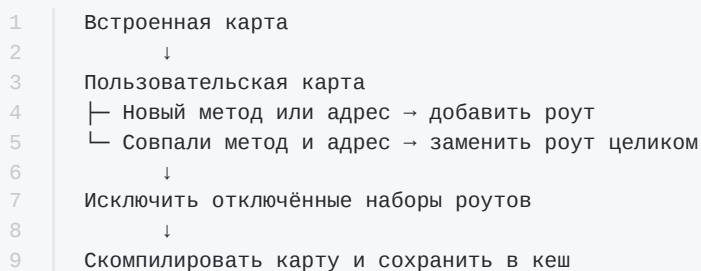
Файл должен существовать и быть доступен PHP для чтения. В пути не допускаются пробелы, обратные косые черты и сегменты `.` или `..`.

Для отключения карты очистите поле «Путь карты роутов» и сохраните настройки.

Настройка роутов

Переопределение роутов

Пользовательская карта дополняет встроенные роуты. При совпадении HTTP-метода и адреса описание из пользовательской карты полностью заменяет встроенное, включая настройки безопасности.



Наборы `native`, `provider` и `service` включаются в настройках роутов. Эта схема описывает сборку карты; порядок поиска конкретного роута — в разделе «Адреса роутов».

Свои настройки

Помимо ключей модуля, в описание каждого роута можно добавлять любые собственные настройки, включая вложенные массивы.

В контроллере получите настройки текущего роута:

```
1  <?php
2
3  $config = \Native\Api\Context::getCurrent()
4          ->getRoute()
5          ->getConfig()
6          ->toArray();
```

Обновление карты

После обновления карты роутов требуется очистка кеша модуля. При сохранении изменённого пути кеш очищается автоматически.

Ключи карты

Карта содержит HTTP-методы, внутри них — адреса роутов и массивы настроек. Ключи модуля задаются константами из `Native\Api\Config\Internals`.

Настройки роута

Константа	Значение
ControllerConfig::CONTROLLER	controller
RouteConfig::ACTIVE	active
RouteConfig::ROUTE_TYPE	type
RouteConfig::DESCRIPTION	description
RouteConfig::PARAMETERS	parameters
RouteConfig::SECURITY	security
RouteConfig::FORMAT	format
RouteConfig::CACHE	cache
RouteConfig::RESPONSE	response

`active` по умолчанию равен `true`, формат ответа — `json`. В `format` можно выбрать JSON или XML. Контроллер задаётся как класс и метод, вызываемый класс или PHP-файл.

`type` необязателен. Значения `native`, `provider`, `service` связывают роут с настройкой активности соответствующей категории. При `active => false` роут не включается в документацию. Проверка активности возвращает `ROUTE_DISABLED` (HTTP 403).

Вложенные ключи

Блок	Ключи	Описание раздела
parameters[имя]	type, required, possibleValue, parameters	Параметры запроса
security	auth, token_type, group, ip	Безопасность
security.group	whitelist, blacklist	Доступ по группам

security.ip	whitelist, blacklist	Доступ по IP
cache	ttl	Ответ и кеш
response	technical_keys_disabled	Ответ и кеш

Поля описания документации приведены в отдельном разделе. `response` управляет ответом модуля, а `responses` описывает ответы в документации.

Пользовательские поля

Получение собственных полей:

```
1 $config = \Native\Api\Context::getCurrent()  
2     ->getRoute()  
3     ->getConfig()  
4     ->toArray();
```

Адреса роутов

Адрес роута указывается относительно «Пути интерфейса», без домена и префикса API.

HTTP-методы

Первый уровень карты — HTTP-метод в верхнем регистре. Используйте константы `Bitrix\Main\Web\HttpClient`: `HTTP_GET`, `HTTP_POST`, `HTTP_PUT`, `HTTP_PATCH`, `HTTP_DELETE`, `HTTP_HEAD`.

Для каждого HTTP-метода нужна отдельная запись в карте.

Формат адреса

Вид	Запись	Сопоставление
Корень API	Пустая строка	Сам «Путь интерфейса»
Статический	<code>orders/list</code>	Точное совпадение сегментов
Динамический	<code>orders/{id}</code>	Значение сегмента попадает в параметр <code>id</code>

Имя параметра внутри фигурных скобок может содержать латинские буквы, цифры и `_`. Параметр занимает целый сегмент. Необязательные сегменты, маски и регулярные выражения не поддерживаются. Статические сегменты сравниваются с учётом регистра.

Параметры адреса имеют приоритет над строкой запроса и телом. Проверки задаются в карте параметров.

Порядок сопоставления

Поиск выполняется среди роутов HTTP-метода запроса:

```
1 | Адрес относительно пути API
2 |
3 | └─ Есть точный статический адрес?
4 |   └─ Да → выбрать роут
5 |
6 | └─ Нет → проверить динамические шаблоны
7 |       с тем же числом сегментов
```

8

└ Есть совпадение → выбрать роут

9

└ Нет совпадений → HTTP 404

Среди динамических шаблонов приоритет у тех, где фиксированные сегменты расположены ближе к началу пути. Например, `orders/{id}` имеет приоритет над `{section}/{id}` при запросе `orders/501`.

Контроллеры

Контроллер задаётся ключом `ControllerConfig::CONTROLLER`.

Форматы

Класс

Значение	Вызов
[OrderController::class, 'get']	<code>(new OrderController())->get()</code>
OrderController::class . '::get'	<code>(new OrderController())->get()</code>
OrderAction::class	<code>(new OrderAction())->__invoke()</code>

Класс должен быть доступен автозагрузке. Конструктор и публичный метод вызываются без аргументов.

PHP-файл

В `controller` укажите полный путь к файлу. Пример PHP-контроллера:

```
1 | $_SERVER['DOCUMENT_ROOT'] . '/local/api/controllers/order.php'
```

Данные запроса

Параметры, текущий роут и ID авторизованного пользователя доступны через единый контекст API.

Доступ к данным проверяет контроллер.

Результат

Контроллер возвращает массив через `return`. Модуль преобразует его в JSON или XML согласно настройкам роута.

Формат, HTTP-статус и служебные поля — в разделе «Ответ и кеш».

Параметры запроса

`RouteConfig::PARAMETERS` содержит массив: имя параметра → настройки проверки.

Формат запроса

Модуль поддерживает JSON-тело запроса, POST-поля и параметры URL. XML и SOAP не поддерживаются как форматы входящих запросов.

Ключи параметра

Константа	Значение
<code>ParameterConfig::REQUIRED</code>	<code>required</code>
<code>ParameterConfig::TYPE</code>	<code>type</code>
<code>ParameterConfig::POSSIBLE_VALUE</code>	<code>possibleValue</code>
<code>RouteConfig::PARAMETERS</code>	<code>parameters</code>
<code>RouteConfig::DESCRIPTION</code>	<code>description</code>
<code>ParameterConfig::EXAMPLE</code>	<code>example</code>

Вложенные поля задаются в `parameters`: см. вложенные параметры.

`description` и `example` используются для рендера документации.

Обязательность и тип

При `required => true` отклоняются отсутствующее значение, `null`, пустая строка и пустой массив. `0` и `false` сами по себе не считаются пустыми.

Тип	Допустимое значение
<code>string</code>	Строка
<code>integer</code>	Целое число или строка из цифр с необязательным начальным -
<code>array</code>	PHP-массив, включая объект или список из JSON

Константы: `TypeConfig::STRING`, `TypeConfig::INTEGER`, `TypeConfig::ARRAY`. Без `type` тип не проверяется. Значения не преобразуются.

Допустимые значения

`possibleValue` принимает непустой массив значений либо диапазон из трёх элементов: нижняя граница, `'...'`, верхняя граница. Границы включаются в диапазон.

```
1 // Одно из перечисленных значений
2 ParameterConfig::POSSIBLE_VALUE => ['new', 'completed', 'cancelled'],
```

```
1 // От 1 до 100 включительно
2 ParameterConfig::POSSIBLE_VALUE => [1, '...', 100],
```

Список сравнивается после приведения значений к строке. Диапазон допускает целые числа, строки целых чисел и числа PHP типа `float`.

Непустой `possibleValue` заменяет проверки `type` и вложенных `parameters`. Проверка `required` сохраняется.

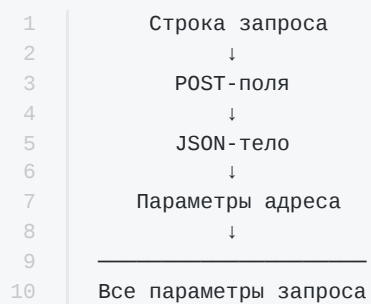
Порядок проверки

Для каждого параметра, описанного в карте:

```
1  Параметр передан?
2  | Нет → required: true → ошибка обязательности
3  |     иначе → пропустить параметр
4  |
5  |  Да → проверить required
6  |
7  |     Проверка пройдена
8  |     | possibleValue задан и непустой
9  |     | | Проверить список или диапазон
10 |     | | На этом проверка параметра завершена
11 |     |
12 |     | possibleValue не задан или пустой
13 |     | | Проверить type, если задан
14 |     | | Тип допустим → проверить вложенность
```

Ошибка обязательности или типа останавливает проверку этого параметра. Остальные параметры продолжают проверяться; ошибки собираются в один ответ.

Источники данных



При совпадении имени значение из источника ниже заменяет предыдущее.

Поля, не описанные в карте, не удаляются и не проверяются. Ошибка проверки возвращает **PARAMETERS_INVALID** (HTTP 400) с указанием проблемных полей.

Вложенные параметры

Вложенная структура задаётся ключом `RouteConfig::PARAMETERS` внутри описания параметра. Для родителя используйте `ParameterConfig::TYPE` со значением `TypeConfig::ARRAY`.

```

1  Объект customer
2  └─ parameters
3     └─ name → правило поля customer.name
4     └─ lastName → правило поля customer.lastName
5
6  Список items
7  └─ parameters
8     └─ [0] → общая карта полей каждого элемента
9         └─ productId → правило поля items[i].productId
10        └─ name → правило поля items[i].name

```

`[0]` в карте задаёт правила для всех элементов списка. Это не ограничение проверки первым элементом.

Объект

В `parameters` перечислите поля с настройками `type`, `required`, `possibleValue` и вложенными `parameters`.

```

1  // {"customer": {
2  //   "name": "Иван",
3  //   "lastName": "Иванов",
4  //   "birthDate": "1990-05-21"
5  // }}
6  'customer' => [
7    ParameterConfig::TYPE => TypeConfig::ARRAY,
8    ParameterConfig::REQUIRED => true,
9    RouteConfig::PARAMETERS => [
10      'name' => [
11        ParameterConfig::TYPE => TypeConfig::STRING,
12        ParameterConfig::REQUIRED => true,
13      ],
14      'lastName' => [
15        ParameterConfig::TYPE => TypeConfig::STRING,
16        ParameterConfig::REQUIRED => true,
17      ],
18      'birthDate' => [
19        ParameterConfig::TYPE => TypeConfig::STRING,
20        ParameterConfig::REQUIRED => true,
21      ],
22    ],
23  ],

```

Список объектов

Для списка объектов обязательна дополнительная пара квадратных скобок `[]` внутри `parameters`. Она помещает карту полей в `parameters[0]`, чтобы проверять каждый объект списка. Без этих скобок описывается один объект.

```

1  // {
2  //   "items": [
3  //     {
4  //       "productId": 101,
5  //       "name": "Термокружка",
6  //       "price": 149900,
7  //       "url": "https://example.ru/products/101/",
8  //       "properties": {
9  //         "color": "Чёрный",
10 //         "warrantyMonths": 12
11 //       }
12 //     },
13 //     {
14 //       "productId": 102,
15 //       "name": "Рюкзак",
16 //       "price": 299900,
17 //       "url": "https://example.ru/products/102/"
18 //     }
19 //   ]
20 // }
21 'items' => [
22   ParameterConfig::TYPE => TypeConfig::ARRAY,
23   ParameterConfig::REQUIRED => true,
24   RouteConfig::PARAMETERS => [
25     [ // Обязательные скобки: карта полей одного элемента списка
26       'productId' => [
27         ParameterConfig::TYPE => TypeConfig::INTEGER,
28         ParameterConfig::REQUIRED => true,
29       ],
30       'name' => [
31         ParameterConfig::TYPE => TypeConfig::STRING,
32         ParameterConfig::REQUIRED => true,
33       ],
34       // Цена в копейках
35       'price' => [
36         ParameterConfig::TYPE => TypeConfig::INTEGER,
37         ParameterConfig::REQUIRED => true,
38       ],
39       'url' => [
40         ParameterConfig::TYPE => TypeConfig::STRING,
41         ParameterConfig::REQUIRED => false,
42       ],
43       // Необязательные характеристики товара
44       'properties' => [
45         ParameterConfig::TYPE => TypeConfig::ARRAY,
46         ParameterConfig::REQUIRED => false,
47         RouteConfig::PARAMETERS => [
48           'color' => [
49             ParameterConfig::TYPE => TypeConfig::STRING,
50             ParameterConfig::REQUIRED => false,
51           ],
52           'warrantyMonths' => [
53             ParameterConfig::TYPE => TypeConfig::INTEGER,
54             ParameterConfig::REQUIRED => false,
55           ],

```

```
56         ],  
57     ],  
58 ],  
59 ],  
60 ],
```

Обязательность

`required` вложенного поля проверяется, только если передан родитель. Для обязательной структуры задайте `required` на каждом уровне.

Непустой `possibleValue` у родителя отключает проверку вложенных полей.

Ошибки содержат вложенные поля и индексы. Оба формата показаны в примере карты.

Безопасность

Настройки безопасности роута задаются в `RouteConfig::SECURITY`. Ключи перечислены в `SecurityConfig`.

Авторизация

Константа	Значение
<code>SecurityConfig::AUTH</code>	<code>auth</code>
<code>SecurityConfig::TOKEN_TYPE</code>	<code>token_type</code>

Используйте константы `SecurityConfig::LOGIN`, `SecurityConfig::TOKEN`, `SecurityConfig::TOKEN_ACCESS`, `SecurityConfig::TOKEN_REFRESH`. `auth` принимает `login` для Basic или `token` для Bearer. Тип токена применяется при `auth => token`: `access` или `refresh`, по умолчанию — `access`.

Без `auth` запрос выполняется без авторизации API, независимо от сессии сайта. Неизвестное строковое значение `auth` возвращает `AUTH_INVALID` (HTTP 401).

Логин и пароль

```
1 // Basic-авторизация пользователя Битрикса
2 RouteConfig::SECURITY => [
3     SecurityConfig::AUTH => SecurityConfig::LOGIN,
4 ],
```

Access-токен

```
1 // Bearer-авторизация по токenu доступа
2 RouteConfig::SECURITY => [
3     SecurityConfig::AUTH => SecurityConfig::TOKEN,
4     SecurityConfig::TOKEN_TYPE => SecurityConfig::TOKEN_ACCESS,
5 ],
```

Refresh-токен

```
1 // Bearer-авторизация по токenu обновления
2 RouteConfig::SECURITY => [
3     SecurityConfig::AUTH => SecurityConfig::TOKEN,
```

```

4     SecurityConfig::TOKEN_TYPE => SecurityConfig::TOKEN_REFRESH,
5 ],

```

Для обоих типов Bearer проверяется привязка к клиенту. При несовпадении возвращается `AUTH_INVALID` (HTTP 401). Токены, выпущенные вручную, можно использовать в другом приложении.

Группы

В `SecurityConfig::GROUP` задаются списки символьных кодов групп, а не числовых ID:

Константа	Значение
<code>SecurityConfig::WHITELIST</code>	whitelist
<code>SecurityConfig::BLACKLIST</code>	blacklist

В `whitelist` достаточно совпадения хотя бы с одной группой пользователя. Совпадение хотя бы с одной группой из `blacklist` запрещает доступ. Для обычных групп используется `STRING_ID`; регистр кодов не учитывается. Пустой список не вводит ограничение. Запрет `blacklist` имеет приоритет над `whitelist`.

```

1     RouteConfig::SECURITY => [
2         SecurityConfig::AUTH => SecurityConfig::TOKEN,
3         SecurityConfig::GROUP => [
4             // Доступ для участников хотя бы одной из групп
5             SecurityConfig::WHITELIST => ['api_clients', 'partners'],
6             // Запрет имеет приоритет над разрешением
7             SecurityConfig::BLACKLIST => ['api_blocked'],
8         ],
9     ],

```

Системные группы

`anonymous` обозначает запрос без авторизации API; `administrators` — администраторов; `everyone` — группу «Все пользователи» авторизованного пользователя.

Константа	Значение
<code>AccessConfig::GROUP_CODE_ANONYMOUS</code>	anonymous
<code>AccessConfig::GROUP_CODE_ADMINISTRATOR</code>	administrators

AccessConfig::GROUP_CODE_EVERYONE

everyone

```

1 // Доступ по токену для группы «Все пользователи»
2 RouteConfig::SECURITY => [
3     SecurityConfig::AUTH => SecurityConfig::TOKEN,
4     SecurityConfig::GROUP => [
5         SecurityConfig::WHITELIST => [
6             AccessConfig::GROUP_CODE_EVERYONE,
7         ],
8     ],
9 ],

```

Администраторы проходят проверку групп независимо от списков. Без авторизации API проверяются права группы `AccessConfig::GROUP_CODE_ANONYMOUS`.

Эти же списки фильтруют операции документации по группам пользователя, открывшего документацию. Кнопка `Authorize` задаёт данные API-запроса и не меняет видимый список операций.

IP-адреса

Для конкретного роута задайте `SecurityConfig::IP` внутри `RouteConfig::SECURITY`. Оба списка принимают массивы IPv4-, IPv6-адресов или подсетей CIDR:

```

1 use Native\Api\Config\Internals\RouteConfig;
2 use Native\Api\Config\Internals\SecurityConfig;
3
4 // Фрагмент конфигурации роута
5 RouteConfig::SECURITY => [
6     SecurityConfig::IP => [
7         SecurityConfig::WHITELIST => ['192.0.2.0/24', '2001:db8::/32'],
8         SecurityConfig::BLACKLIST => ['192.0.2.50', '2001:db8::50'],
9     ],
10 ],

```

В этом примере разрешены две подсети, кроме двух явно запрещённых адресов. Blacklist проверяется первым. Непустой whitelist допускает только совпавшие адреса; отсутствующие или пустые списки не добавляют ограничений роута.

Запрос должен пройти и глобальные IP-списки, и списки роута. Разрешение на одном уровне не отменяет запрет на другом. Проверяется `REMOTE_ADDR`; заголовки `X-Forwarded-For` и `Forwarded` не используются. Исключения для администраторов по IP нет.

Проверка роута выполняется после `onAfterRouting`, между событиями `onBeforeIpAccess` и `onAfterIpAccess`, до авторизации и чтения кеша ответа, поэтому срабатывает и при готовом кеше. Штатный `OPTIONS` завершается до выбора роута и проверяет только глобальные IP-списки. После изменения карты очистите кеш модуля.

Before может изменить списки текущей проверки, After вызывается только при её успешном завершении. Примеры — в регистрации событий.

При отказе возвращается HTTP 403 с кодом `IP_BLACKLIST` или `IP_WHITELIST`. Эти же коды используются для глобальных ограничений. IP-списки ограничивают выполнение запроса; список операций Swagger по ним не фильтруется.

Остальные ограничения

Глобальные IP-ограничения, расписание и лимиты задаются в настройках безопасности. Доступ к данным проверяет контроллер.

Ответ и кеш

Кеширование

Срок задаётся в `RouteConfig::CACHE` ключом `CacheConfig::TTL`, в секундах.

Значение <code>ttl</code>	Поведение
Больше <code>0</code>	Кеширование результата, в том числе для POST и других методов
<code>0</code>	Кеширование роута отключено
Не задано	Общая настройка модуля; применяется только к GET и HEAD

```
1 // Кешировать результат на 5 минут; 0 — отключить кеш
2 RouteConfig::CACHE => [
3     CacheConfig::TTL => 300,
4 ],
```

После проверок доступа и параметров модуль выбирает источник результата:

```
1 Доступ и параметры проверены
2     ↓
3 Кеширование включено?
4   ├── Нет → выполнить контроллер
5   |       └─ Сформировать ответ
6   |
7   └─ Да → результат есть в кеше?
8         ├── Да → сформировать ответ из кеша
9         └─ Нет → выполнить контроллер
10                └─ Сохранить результат в кеш
11                   └─ Сформировать ответ
```

При попадании в кеш контроллер и его события не вызываются. Схема показывает штатный ход запроса; обработчики событий могут изменить результат или остановить выполнение.

Кеш учитывает метод, адрес, контроллер, действие, TTL, формат, пользователя, клиента и параметры. Если результат зависит от других данных, отключите кеш.

После обновления карты требуется очистка кеша модуля.

Формат ответа

Формат задаётся в `RouteConfig::FORMAT`. По умолчанию используется JSON. Параметр `format` в строке запроса имеет приоритет над картой.

JSON

```
1 // Ответ в JSON
2 RouteConfig::FORMAT => TypeConfig::FORMAT_JSON,
```

Примеры: с техническими полями и без них.

XML

```
1 // Ответ в XML
2 RouteConfig::FORMAT => TypeConfig::FORMAT_XML,
```

Для XML без технической обёртки контроллер должен вернуть один корневой элемент. Примеры: с техническими полями и без них.

Технические поля

Технические ключи ответа можно включить или выключить для каждого роута. В `RouteConfig::RESPONSE` это задаёт `ResponseConfig::TECHNICAL_KEYS_DISABLED`. Если ключ не задан, используется общая настройка модуля.

```
1 // Включить технические ключи
2 RouteConfig::RESPONSE => [
3     ResponseConfig::TECHNICAL_KEYS_DISABLED => false,
4 ],
```

```
1 // Выключить технические ключи
2 RouteConfig::RESPONSE => [
3     ResponseConfig::TECHNICAL_KEYS_DISABLED => true,
4 ],
```

Структура ответов: JSON и XML. Ошибки модуля сохраняют служебную структуру.

HTTP-статус и заголовки

Задаются в массиве результата контроллера:

Константа	Значение
ResponseConfig::STATUS	status
ResponseConfig::HEADERS	headers

```
1 // Создан заказ: HTTP 201 и ссылка на него в заголовке Location
2 return [
3     ResponseConfig::STATUS => HttpStatusConfig::HTTP_STATUS_201,
4     ResponseConfig::HEADERS => [
5         'Location' => '/api/orders/501',
6     ],
7     'order' => [
8         'id' => 501,
9     ],
10 ];
```

`status` и `headers` управляют HTTP-ответом и не попадают в данные результата.

Статус по умолчанию

Если контроллер не задал статус:

Метод	HTTP-статус
POST	201
DELETE	204
Остальные методы	200

Ответ без тела

Тело ответа не отправляется:

- Для метода `HEAD`
- Для статусов `1xx`, `204`, `205` и `304`

Поля OpenAPI

Ключи описания задаются константами `OpenApiConfig` — см. полный справочник.

Поля OpenAPI в карте задают описание операции: название, параметры, тело запроса и ответы. Здесь показано, как дополнить или переопределить автоматически сформированное описание.

Поля операции

Поля задаются на одном уровне с `controller`, `parameters` и `security`.

Название и оформление

Ключ	Назначение
tags	Разделы документации
summary	Краткое название операции
description	Подробное описание
operationId	Идентификатор операции
externalDocs	Внешняя документация: <code>url</code> , <code>description</code>
deprecated	Признак устаревшей операции
servers	Серверы конкретной операции

```
1 use Native\Api\Config\Internals\OpenApiConfig;
2
3 // Название и раздел операции
4 OpenApiConfig::SUMMARY => 'Получить заказ',
5 OpenApiConfig::DESCRIPTION => 'Возвращает заказ по ID',
6 OpenApiConfig::OPERATION_ID => 'getOrder',
7 OpenApiConfig::TAGS => ['Заказы'],
```

Запрос и ответ

Ключ	Назначение
parameters	Параметры пути, строки запроса и заголовков
requestBody	Тело запроса

responses	Ответы по HTTP-статусам
security	Требования авторизации в документации
callbacks	Описания обратных вызовов
components.schemas	Именованные схемы для <code>\$ref</code>

Автозаполнение

Эти сведения уже формируются из карты — повторять их в полях OpenAPI не нужно.

Что формируется	Источник
Название	<code>summary</code> → <code>description</code> → адрес роута
Раздел	<code>tags</code> , иначе первый сегмент адреса
Параметры пути	Фигурные скобки адреса; всегда обязательны
Остальные параметры	<code>requestBody</code> для POST, PUT и PATCH; <code>query</code> для остальных методов
Тип параметра	<code>type</code> из карты параметров. Если не задан, в документации указывается <code>string</code>
Авторизация	<code>login</code> → <code>basicAuth</code> , <code>token</code> → <code>bearerAuth</code>
Успешный ответ	POST → <code>201</code> , DELETE → <code>204</code> , остальные → <code>200</code>
Допустимые значения	Список <code>possibleValue</code> → <code>enum</code> ; диапазон → <code>minimum</code> и <code>maximum</code>

Совпадающие ключи

parameters

Формат	Назначение
Карта по именам параметров	Проверки модуля и автоматическое описание
Числовой список OpenAPI	Только описание; проверки параметров не создаются

```

1  use Native\Api\Config\Internals\OpenApiConfig;
2
3  // Только описание заголовка в документации
4  OpenApiConfig::PARAMETERS => [
5      [
6          OpenApiConfig::NAME => 'X-Client-Version',
7          OpenApiConfig::IN => 'header',
8          OpenApiConfig::REQUIRED => false,
9          OpenApiConfig::SCHEMA => [OpenApiConfig::TYPE => 'string'],
10         OpenApiConfig::EXAMPLE => '1.0',
11     ],
12 ],

```

security

Формат	Назначение
Ключи <code>auth</code> , <code>token_type</code> , <code>group</code> , <code>ip</code>	Проверки доступа и автоматическое описание
Числовой список OpenAPI	Только описание авторизации; проверку не включает

`group` и `ip` задают проверки доступа, но не создают схемы авторизации OpenAPI.

```

1  use Native\Api\Config\Internals\OpenApiConfig;
2
3  // Только описание Bearer-авторизации в документации
4  OpenApiConfig::SECURITY => [
5      [SecurityConfig::SECURITY_SCHEME_BEARER => []],
6  ],

```

Переопределение

Тело запроса

Явный `requestBody` заменяет автоматический. Для GET и HEAD тело исключается. Для вложенного объекта задайте описание явно: генератор описывает `type => array` как массив объектов.

```

1  use Native\Api\Config\Internals\OpenApiConfig;
2
3  // JSON-объект с обязательным полем name
4  OpenApiConfig::REQUEST_BODY => [
5      OpenApiConfig::REQUIRED => true,
6      OpenApiConfig::CONTENT => [
7          TypeConfig::CONTENT_TYPE_JSON => [
8              OpenApiConfig::SCHEMA => [
9                  OpenApiConfig::TYPE => 'object',

```

```

10         OpenApiConfig::REQUIRED => [OpenApiConfig::NAME],
11         OpenApiConfig::PROPERTIES => [
12             'name' => [OpenApiConfig::TYPE => 'string', OpenApiConfig::EXAMPLE =>
'Иван'],
13         ],
14     ],
15 ],
16 ],
17 ],

```

Параметры и ответы

Что задано явно	Как применяется
Параметр с теми же <code>in</code> и <code>name</code>	Заменяет совпавшее описание
Ответ с тем же HTTP-статусом	Дополняет или переопределяет автоматическое описание
Собственный успешный ответ <code>2xx</code>	Убирает автоматический успешный статус, если его нет в заданном списке

```

1 use Native\Api\Config\Internals\OpenApiConfig;
2
3 // Описание ответов; фактический HTTP-статус задаёт контроллер
4 OpenApiConfig::RESPONSES => [
5     '201' => [OpenApiConfig::DESCRIPTION => 'Заказ создан'],
6     '400' => [OpenApiConfig::DESCRIPTION => 'Ошибка параметров запроса'],
7 ],

```

Именованные схемы

Из `components` поддерживается `schemas`. При совпадении имени применяется последнее описание.

```

1 use Native\Api\Config\Internals\OpenApiConfig;
2
3 // Общая схема заказа
4 OpenApiConfig::COMPONENTS => [
5     OpenApiConfig::SCHEMAS => [
6         'Order' => [
7             OpenApiConfig::TYPE => 'object',
8             OpenApiConfig::PROPERTIES => [
9                 'id' => [OpenApiConfig::TYPE => 'integer'],
10            ],
11        ],
12    ],
13 ],

```

```

1 use Native\Api\Config\Internals\OpenApiConfig;
2
3 // Ссылка на схему в описании тела запроса или ответа
4 OpenApiConfig::SCHEMA => [OpenApiConfig::REF => '#/components/schemas/Order'],

```

Общие схемы ответа

`OpenApiSchemas` формирует описания ответов с техническими полями модуля. Имена схем перечислены в справочнике `OpenApiConfig`.

successResponse

Описание JSON-ответа с техническим конвертом

```

1 use Native\Api\Config\Internals\OpenApiConfig;
2 use Native\Api\Http\Docs\OpenApiSchemas;
3
4 OpenApiSchemas::successResponse(
5     description: 'Заказ получен',
6     resultSchema: [OpenApiConfig::TYPE => 'object'],
7 );

```

Необязательные параметры: `resultSchema` — схема результата, `example` — пример полного ответа

errorResponse

Описание JSON-ошибки с кодами и примером

```

1 use Native\Api\Config\Internals\HttpStatusConfig;
2 use Native\Api\Config\Internals\ExceptionConfig;
3 use Native\Api\Http\Docs\OpenApiSchemas;
4
5 OpenApiSchemas::errorResponse(
6     statusCode: HttpStatusConfig::HTTP_STATUS_404,
7     description: 'Роут не найден',
8     errorCodes: [ExceptionConfig::ERROR_ROUTE_NOT_FOUND],
9 );

```

Для ответа без технических полей задайте схему явно вместо

```
OpenApiSchemas::successResponse()
```

Ссылка на схему

```
1 use Native\Api\Config\Internals\OpenApiConfig;  
2 use Native\Api\Http\Docs\OpenApiSchemas;  
3  
4 $schema = OpenApiSchemas::ref(OpenApiConfig::SCHEMA_RESPONSE_ERROR);
```

Метод возвращает массив с `$ref` на именованную схему.

`requestBody` , `responses` , `callbacks` и схемы влияют только на документацию.
Полная карта — в разделе «Пример карты».

Рендер документации

Рендер документации — интерфейс, который отображает описание API: роуты, параметры и ответы.

По умолчанию

Компонент `native.api:docs` использует Swagger UI для просмотра документации, авторизации и выполнения запросов.

```

1 // Подключение стандартного рендера в сервисе документации
2 $APPLICATION->IncludeComponent(
3     'native.api:docs',
4     '.default',
5     [],
6     false
7 );

```

Порядок работы с интерфейсом — в разделе «Документация».

Описание API

Компонент формирует описание OpenAPI 3.0.3 из карты роутов. В него попадают активные роуты, доступные группам пользователя, открывшего документацию.

```

1  Проверка права чтения документации
2      ↓
3  Карта включённых наборов роутов
4      ↓
5  Отбор активных роутов по группам посетителя
6      ↓
7  Описание OpenAPI в $arResult['SCHEMA']
8      ↓
9  Шаблон компонента
10  └─ Стандартный → Swagger UI
11  └─ Собственный → ваш интерфейс

```

Сформированное описание кешируется с учётом групп посетителя. Право открыть страницу проверяется и при чтении из кеша.

Swagger UI отображает готовое описание. Состав документации и проверки API определяет модуль.

Собственный рендер

Скопируйте компонент в `/local/components/native.api/docs/` и измените его стандартный шаблон, чтобы использовать свой интерфейс вместо Swagger UI.

Описание передаётся в шаблон как JSON в `$arResult['SCHEMA']`.

Пример шаблона

Минимальный вывод списка роутов в файле `templates/.default/template.php` скопированного компонента:

```

1  <?php
2
3  use Bitrix\Main\Web\Json;
4
5  if (!defined('B_PROLOG_INCLUDED') || B_PROLOG_INCLUDED !== true) {
6      die();
7  }
8
9  $schema = Json::decode($arResult['SCHEMA']);
10 ?>
11
12 <ul>
13     <?php foreach ($schema['paths'] as $path => $operations): ?>
14         <?php foreach ($operations as $method => $operation): ?>
15             <li><?= htmlspecialcharsbx(strtoupper($method) . ' ' . $path) ?></li>
16         <?php endforeach; ?>
17     <?php endforeach; ?>
18 </ul>
```

При замене шаблона сохраните класс компонента: он проверяет доступ и отбирает роуты по группам.

Сервисные роуты

Сервисные роуты проверяют обработку запросов и разные типы контроллеров. Все операции диагностические: данные пользователей, групп и других сущностей не изменяются.

Включите их в настройках роутов. Адреса ниже указаны относительно пути интерфейса API.

GET up

Базовая проверка доступности API и вызова метода контроллера. Можно передать произвольные параметры в строке запроса.

Параметр	Значение
Контроллер	Класс <code>UpIndex</code> , метод <code>list()</code>
Обязательные параметры	Нет
Успешный ответ	HTTP <code>200</code> , JSON

Переданные значения возвращаются в поле `parameters` . Пример запроса

GET up/user/{id}

Проверяет передачу параметра в адресе роута. Значение `{id}` попадает в `parameters.id` вместе с остальными параметрами запроса.

Параметр	Значение
Контроллер	Класс <code>UpIndex</code> , метод <code>list()</code>
Параметр адреса	<code>id</code> ; отдельная проверка типа в карте не задана
Успешный ответ	HTTP <code>200</code> , JSON

Пример запроса с ID и строкой запроса

GET up/group/{id}

Показывает настройку доступа по системной группе администраторов.

Параметр	Значение
Контроллер	Класс <code>UpIndex</code> , метод <code>list()</code>
Параметр адреса	<code>id</code>
Группа в whitelist	<code>AccessConfig::GROUP_CODE_ADMINISTRATOR</code>
Ответ при текущей настройке	HTTP <code>403</code> , <code>GROUP_WHITELIST</code>

Роут виден администраторам в документации. В его карте не задана авторизация API, поэтому запрос проверяется как анонимный и не проходит whitelist.

Пример проверки ограничения

POST up

Проверяет передачу тела запроса и вызов метода `create()` . Принятые параметры возвращаются в `parameters` .

Параметр	Значение
Контроллер	Класс <code>UpIndex</code> , метод <code>create()</code>
Обязательные параметры	Нет
Успешный ответ	HTTP <code>201</code> , JSON

Пример передачи JSON

PUT up

Проверяет выполнение PHP-файла как контроллера и формирование XML-ответа.

Параметр	Значение
Контроллер	PHP-файл <code>UpScript.php</code>
Обязательные параметры	Нет
Успешный ответ	HTTP <code>200</code> , XML
Технические ключи	Отключены в карте
Корневой элемент	<code>response</code> , задан самим контроллером

Пример запроса и получения XML

DELETE up

Проверяет выполнение вызываемого класса через `__invoke()`.

Параметр	Значение
Контроллер	Класс <code>UpInvoke</code> , метод <code>__invoke()</code>
Обязательные параметры	Нет
Успешный ответ	HTTP <code>200</code> , JSON

Контроллер явно задаёт статус `200`, поэтому диагностические данные передаются в теле ответа. Пример запроса

Данные ответа

Успешный вызов возвращает:

Поле	Значение
method	HTTP-метод запроса
controller	Класс или путь PHP-контроллера
action	Действие контроллера
description	Описание роута из карты
parameters	Полученные параметры запроса

Расположение данных зависит от технической обёртки. В PUT данные находятся внутри корневого элемента `response`.

Формат ответа можно изменить параметром `format=json` или `format=xml` в строке запроса.

Провайдер

Провайдер вызывает разрешённые публичные методы PHP-классов. Требуется включённых роутов провайдера и access-токена в `Authorization: Bearer <token>`.

Адрес `provider` указан относительно пути интерфейса API.

Методы

Запрос	Назначение	Параметры
GET provider	Описание класса и разрешённых методов	class — обязательно; method — по необходимости
POST provider	Выполнение метода	class, method — обязательно; parameters, callback — по необходимости

`class` — полное имя класса с пространством имён. `method` — имя публичного метода. `parameters` — массив позиционных аргументов или объект именованных аргументов метода. Нестатический метод требует конструктора без обязательных аргументов.

GET возвращает `result.class` и `result.methods` либо `result.method`. POST — `result.data` и HTTP 201. Без технического конверта поля находятся в корне ответа.

Разрешённые методы

Класс	Методы по умолчанию
<code>Bitrix\Main\Localization\Loc</code>	<code>getCurrentLang</code>
<code>Bitrix\Currency\CurrencyManager</code>	<code>getBaseCurrency</code> , <code>getCurrencyList</code>
<code>CCurrencyRates</code>	<code>ConvertCurrency</code> , <code>GetConvertFactorEx</code>

Классы валют доступны при наличии соответствующего модуля Битрикса.

Для расширения наследуйте `Native\Api\Controller\Provider\Provider`, переопределите `getMethodsAllowed()` или `getClassesAllowed()` и подключите `describe` и `execute` в карте.

`getMethodsAllowed()` возвращает карту «класс → методы» с точными именами классов и методов. `getClassesAllowed()` разрешает класс или пространство имён: `Vendor\PublicApi` охватывает `Vendor\PublicApi\Items`, но не `Vendor\PublicApiExtra`. Сравнение не учитывает регистр и начальную обратную косую черту имени класса.

Запреты из `getClassesDenied()` и `getMethodsDenied()` имеют приоритет над разрешениями. Если оба разрешающих списка пусты, все вызовы запрещены. Для точечного доступа используйте список методов: разрешение пространства имён распространяется и на классы, добавленные в него позднее. Права на данные проверяет вызываемый код.

Результаты выборок

`callback` — публичный метод результата БД без обязательных аргументов. По умолчанию: `fetch` для старого API Битрикса, `fetchRaw` для D7.

При нескольких строках возвращается список, при одной — строка выборки, при пустой выборке возможен `false`. Другие callback-методы вызываются один раз.

Массивы и скалярные значения возвращаются без преобразования.

Примеры запросов: cURL, JavaScript, PHP.

Токены

Роут `token` добавляется к пути интерфейса API и работает при включённых нативных роутах.

Запрос	Авторизация	Результат
GET token	Basic: логин и пароль пользователя	Активные токен-сессии пользователя, HTTP 200
POST token	Basic: логин и пароль пользователя	Новая пара access/refresh, HTTP 201
PUT token	Bearer: refresh-токен	Замена пары токенов, HTTP 200
DELETE token?client_id=...	Bearer: access-токен	Отзыв токенов указанного клиента, HTTP 200

Используйте HTTPS. При `AUTH_MFA_REQUIRED` завершите вход с MFA на сайте и повторите запрос с cookie этой сессии.

Выпуск и обновление

POST и PUT возвращают в `result` массив из двух объектов:

Поле	Значение
<code>type</code>	<code>access</code> или <code>refresh</code>
<code>token</code>	Значение токена для заголовка <code>Authorization</code>
<code>active_to</code>	Дата окончания действия
<code>timestamp.active_to</code>	Срок действия в Unix timestamp
<code>timestamp.expires_in</code>	Срок действия в секундах
<code>expires_in</code>	Срок действия в текстовом виде

POST для того же клиента и PUT заменяют прежнюю пару. Сохраните оба новых токена. Схема жизненного цикла, сроки действия.

Привязка к клиенту

Токены, полученные при выпуске через API, привязаны к клиенту. Используйте одинаковые `User-Agent`, `Accept-Language`, `Sec-CH-UA`, `Sec-CH-UA-Mobile` и `Sec-CH-UA-Platform` при выпуске, обновлении и обращении к защищённым роутам. Для серверной интеграции задайте постоянный `User-Agent`.

При изменении этих заголовков access- и refresh-токены могут быть отклонены с `AUTH_INVALID` (HTTP 401). Например, после смены браузера или языка может потребоваться получить новую пару. Если все перечисленные заголовки пусты, возвращается `AUTH_REQUIRED` (HTTP 401).

Для использования в другом приложении можно выпустить пару вручную. Такие токены не привязаны к заголовкам клиента; передавайте `User-Agent` и соблюдайте их срок действия.

Список и отзыв

GET возвращает даты `date_create`, `active_to`, их Unix timestamp в `timestamp` и объект `client`: `id`, `name`, `ip`, `type`, `agent`. Значения токенов не выдаются. Если сессий нет, `result` может отсутствовать.

В DELETE передайте `client.id` из списка в параметре `client_id`. Ответ: `result.delete` — удалены ли токены, `result.client_id` — переданный ID. Если у пользователя нет токенов этого клиента, `delete` равен `false`, а `result.code` содержит `CLIENT_NOT_FOUND`. HTTP-статус остаётся 200. Для чужого и отсутствующего клиента ответ одинаков. При `delete = true` поле `code` отсутствует. Отмена удаления обработчиком события и ошибка записи возвращаются отдельными ошибками.

При отключённом техническом конверте результат возвращается без обёртки `result`.

Пример данных `result`, когда токены клиента не найдены:

```

1  {
2    "delete": false,
3    "client_id": "client-id",
4    "code": "CLIENT_NOT_FOUND"
5  }
```

В XML те же данные представлены элементами (фрагмент ответа):

```
1  <result>
2    <delete></delete>
3    <client_id>client-id</client_id>
4    <code>CLIENT_NOT_FOUND</code>
5  </result>
```

Подписка на события

Регистрируйте обработчики через `Bitrix\Main\EventManager` для модуля `native.api` в `local/php_interface/init.php`.

Схема событий

Обработка запроса

Этапы идут сверху вниз; у каждого указаны события до и после его выполнения.

```

1  native.api
2  |
3  |—— onBeforeStart
4  |  Запуск API
5  |—— onAfterStart
6  |
7  |—— onBeforeIpAccessGlobal
8  |  Глобальный доступ по IP
9  |—— onAfterIpAccessGlobal
10 |
11 |—— onBeforeCors
12 |  Проверка CORS
13 |—— onAfterCors
14 |  // OPTIONS переходит к формированию ответа
15 |
16 |—— onBeforeRouting
17 |  Поиск роута
18 |—— onAfterRouting
19 |
20 |—— onBeforeIpAccess
21 |  Доступ по IP выбранного роута
22 |—— onAfterIpAccess
23 |
24 |—— onBeforeAuthorization
25 |  Авторизация
26 |—— onAfterAuthorization
27 |
28 |—— onBeforeGroupAccess
29 |  Доступ по группам
30 |—— onAfterGroupAccess
31 |
32 |—— onBeforeSchedule
33 |  Расписание
34 |—— onAfterSchedule
35 |
36 |—— onBeforeRequestLimit
37 |  Лимиты запросов
38 |—— onAfterRequestLimit
39 |
40 |—— onBeforeParameters
41 |  Проверка параметров
42 |—— onAfterParameters
43 |
44 |  Проверка кеша
45 |  // Если результат есть в кеше, пропускаем контроллер и его события
46 |  // и сразу переходим к формированию ответа
47 |

```

```

48 | |—— onBeforeController
49 | |— Контроллер
50 | |—— onAfterController
51 | |
52 | |— Формирование ответа
53 | |
54 | |—— onBeforeResponse
55 | |— Отправка ответа
56 | |
57 | |— Фоновая задача
58 | |—— onAfterResponse

```

При отказе CORS `onAfterCors` и последующие этапы не вызываются.

Операции с токенами

События токенов вызываются при соответствующей операции, независимо от основной цепочки запроса.

```

1 | |—— onBeforeTokenAdd
2 | |— Добавление токена
3 | |—— onAfterTokenAdd
4 | |
5 | |—— onBeforeTokenUpdate
6 | |— Изменение токена
7 | |—— onAfterTokenUpdate
8 | |
9 | |—— onBeforeTokenDelete
10 | |— Удаление токена
11 | |—— onAfterTokenDelete

```

Регистрация

Обработчики подключаются через `EventManager::addEventHandler()` после `Loader::includeModule('native.api')`. Используйте константы `ModuleEventConfig` для имён событий и параметров.

В разделе «Примеры работы → События» приведены регистрации Before/After: изменение IP-списков роута, передача проверенного IP, подготовка параметров, изменение результата контроллера и добавление HTTP-заголовка.

Контекст и результат

`Bitrix\Main\Event` передаёт `apiContext` — текущий контекст API. Доступны `getParams()`, `setParams()`, `getRoute()` и `getAuth()`. Сам объект контекста заменять нельзя.

```

1 | Результат обработчика события запроса
2 | └─ null / return; / false → продолжить выполнение
3 | └─ EventResult::SUCCESS → заменить указанные параметры события
4 | └─ EventResult::ERROR
5 |     └─ exception → остановить сценарий исключением
6 |     └─ response → завершить сценарий готовым ответом
7 | └─ Массив, включая [] → немедленно отправить JSON/XML
8 | └─ HttpResponse → немедленно отправить готовый HTTP-ответ

```

При повторной записи ключа применяется значение последнего обработчика. Неверный тип результата вызывает `EVENT_HANDLER_RESULT_INVALID`.

`EventResult` разбирается после вызова обработчиков события. Массив и `HttpResponse` обрабатываются немедленно: следующие обработчики, дальнейшие этапы и события `onBeforeResponse` / `onAfterResponse` не вызываются. Уже выполненные действия и запланированные фоновые задачи автоматически не отменяются. Досрочный ответ из события контроллера не записывается в кеш роута. При попадании в уже готовый кеш события контроллера не вызываются. Проверки, обязательные на каждом запросе, размещайте на более раннем этапе, например в `onBeforeParameters`.

Для событий токенов действует отдельный контракт: `false` отменяет Before-операцию; результат After-обработчика игнорируется.

Досрочный ответ

Массив данных

В теле обработчика верните массив:

```

1 | return ['test' => 123];

```

Модуль вернёт HTTP 200. Формат и технический конверт берутся из выбранного роута; до выбора роута используются JSON и глобальная настройка конверта. Параметр запроса `format` сохраняет приоритет. При включённом конверте данные окажутся в `result`, при выключенном — непосредственно в теле ответа.

HTTP-статус и данные

Статус задаётся существующим ключом `ResponseConfig::STATUS`. Например, обработчик перед контроллером может завершить запрос с HTTP 403:

```

1  use Bitrix\Main\Event;
2  use Bitrix\Main\EventManager;
3  use Native\Api\Config\Internals\BaseConfig;
4  use Native\Api\Config\Internals\ModuleEventConfig;
5  use Native\Api\Config\Internals\ResponseConfig;
6
7  EventManager::getInstance()->addEventHandler(
8      BaseConfig::MODULE_ID,
9      ModuleEventConfig::BEFORE_CONTROLLER,
10     static function (Event $event): array {
11         return [
12             ResponseConfig::STATUS => 403,
13             ResponseConfig::HEADERS => ['X-Reason' => 'access-denied'],
14             'test' => 123,
15         ];
16     },
17 );

```

Регистрируйте обработчик после подключения модуля — см. общий блок подключения. `status` задаёт HTTP-код и не попадает в пользовательские данные. Ключ `headers` позволяет передать заголовки ответа. Для HEAD и статусов без тела, например 204 и 304, данные не отправляются.

Готовый HTTP-ответ

Для ответа обычным текстом или HTML можно вернуть `Bitrix\Main\HttpResponse` : модуль отправит готовый объект без преобразования в JSON/XML. Для JSON/XML, включая собственный HTTP-статус и заголовки, достаточно массива из примера выше.

Для `onAfterResponse` ответ уже отправлен: массив или `HttpResponse` прекращает только следующих обработчиков этого события. Новое тело и статус клиенту не отправляются.

Список событий

Параметры и поведение событий по этапам:

- Запуск и роутинг
- Проверки доступа
- Обработка запроса
- Токены

Имена и ключи параметров — в справочнике ModuleEventConfig.

Запуск и роутинг

Во всех событиях доступен `apiContext` — контекст API.

Событие	Момент вызова	Остальные параметры
<code>onBeforeStart</code>	До проверки активности модуля	<code>httpRequest</code>
<code>onAfterStart</code>	После проверки активности модуля	<code>httpRequest</code>
<code>onBeforeCors</code>	После глобальной проверки IP, до проверки CORS	<code>httpRequest</code>
<code>onAfterCors</code>	После успешного завершения проверки CORS	<code>httpRequest</code>
<code>onBeforeRouting</code>	После CORS, непосредственно перед поиском роута	<code>httpRequest</code>
<code>onAfterRouting</code>	После выбора роута, до его IP-проверки и авторизации	<code>httpRequest</code> , <code>route</code>

Параметры

`httpRequest` — `Bitrix\Main\HttpRequest`. `route` — `Native\Api\Routing\Route\CompiledRoute`; его конфигурация доступна через `getConfig()`.

Через `EventResult::SUCCESS` можно заменить `httpRequest`, а после выбора роута — `route`. Типы объектов должны сохраняться.

CORS и выбор роута

CORS проверяется между `onBeforeCors` и `onAfterCors`, до событий маршрутизации. Настройки CORS не передаются в параметры событий. Заголовок Origin:

```
1 | $origin = $httpRequest->getHeader('Origin');
```

`onAfterCors` вызывается после успешного завершения валидатора, включая запросы без Origin. При отказе CORS это событие и события маршрутизации не вызываются. Обычный массив или `HttpResponse` из CORS-обработчика немедленно

завершает конвейер по общему контракту. Изменение запроса в After не запускает повторную проверку CORS.

После `onAfterRouting` вызывается `onBeforeIpAddress`, проверяются IP-списки выбранного роута. При успешной проверке вызывается `onAfterIpAddress`. Если обработчик `onAfterRouting` заменил `route`, используются списки нового роута.

Штатный `OPTIONS` завершается после `onAfterCors`: ни `onBeforeRouting`, ни `onAfterRouting` не вызываются. Для остальных запросов `onBeforeRouting` выполняется перед поиском роута; если роут не найден, `onAfterRouting` не вызывается.

Константы — в справочнике `ModuleEventConfig`.

Проверки доступа

Проверки выполняются в порядке таблицы. Событие After вызывается после успешной проверки. Во всех событиях доступен `apiContext` — контекст API.

Проверка	До проверки	После проверки	Остальные параметры
Глобальные IP	<code>onBeforeIpAccessGlobal</code>	<code>onAfterIpAccessGlobal</code>	<code>httpRequest</code> , <code>ipBlacklist</code> , <code>ipWhitelist</code>
IP роута	<code>onBeforeIpAccess</code>	<code>onAfterIpAccess</code>	<code>httpRequest</code> , <code>route</code> , <code>ipBlacklist</code> , <code>ipWhitelist</code>
Авторизация	<code>onBeforeAuthorization</code>	<code>onAfterAuthorization</code>	<code>httpRequest</code> , <code>route</code>
Группы	<code>onBeforeGroupAccess</code>	<code>onAfterGroupAccess</code>	<code>httpRequest</code> , <code>route</code>
Расписание	<code>onBeforeSchedule</code>	<code>onAfterSchedule</code>	<code>httpRequest</code> , <code>route</code>
Лимит запросов	<code>onBeforeRequestLimit</code>	<code>onAfterRequestLimit</code>	<code>httpRequest</code> , <code>route</code>

Глобальные IP-списки проверяются до маршрутизации. IP-списки роута — после `onAfterRouting`, до авторизации. Для роута вызываются `onBeforeIpAccess` и `onAfterIpAccess`; `onBeforeIpAccessGlobal` и `onAfterIpAccessGlobal` относятся только к глобальной проверке и повторно не запускаются. События IP-проверки роута вызываются и при пустых списках, в том числе при попадании в кеш ответа. Для штатного OPTIONS и ненайденного роута они не вызываются.

Данные проверки

В `onBeforeIpAccessGlobal` и `onBeforeIpAccess` можно заменить массивы `ipBlacklist` и `ipWhitelist` через `EventResult::SUCCESS`. Списки должны содержать допустимые IP-адреса или CIDR. After получает списки после применения результатов Before; их изменение в After не запускает повторную проверку. Примеры регистрации — в разделе «События».

Исходные IP-списки карты доступны через конфиг роута; изменения Before-обработчика передаются в параметрах события и не записываются в карту:

```
1 $config = $route->getConfig();
2 $blacklist = $config->getSecurityIpBlacklist();
```

```
3 | $whitelist = $config->getSecurityIpWhitelist();
```

Списки групп:

```
1 | $config = $route->getConfig();  
2 | $blacklist = $config->getSecurityGroupBlacklist();  
3 | $whitelist = $config->getSecurityGroupWhitelist();
```

ID пользователя API после авторизации:

```
1 | $userId = $apiContext->getAuth()?->getUserId();
```

Остановка запроса

Для немедленного ответа верните обычный массив или `HttpResponse`. Для штатного результата ошибки используйте `EventResult::ERROR` с `exception` или `response` — см. результат обработчика. Обычный `return false` проверку не отменяет.

Константы — в справочнике `ModuleEventConfig`.

Обработка запроса

Во всех событиях доступен `apiContext` — контекст API.

Событие	Остальные параметры	Что можно изменить
<code>onBeforeParameters</code>	<code>route</code>	Роут; параметры запроса через <code>apiContext->setParams()</code>
<code>onAfterParameters</code>	<code>route</code> , <code>exception</code>	Роут и ошибку проверки
<code>onBeforeController</code>	<code>route</code>	Роут перед вызовом контроллера
<code>onAfterController</code>	<code>route</code> , <code>response</code>	Массив, возвращённый контроллером
<code>onBeforeResponse</code>	<code>httpRequest</code> , <code>response</code>	Готовый <code>Bitrix\Main\HttpResponse</code>
<code>onAfterResponse</code>	<code>httpRequest</code> , <code>response</code>	Ответ уже отправлен

Параметры

В `onAfterParameters` параметр `exception` содержит ошибку проверки или `null`. Чтобы отменить ошибку и продолжить запрос, верните `EventResult::SUCCESS` с `exception => null`.

Контроллер

В `onAfterController` можно заменить массив `response` через `EventResult::SUCCESS`. При попадании в кеш события контроллера не вызываются; проверки доступа и параметров сохраняются.

Ответ

В `onBeforeResponse` можно заменить готовый `Bitrix\Main\HttpResponse`. Возврат массива или `HttpResponse` досрочно завершает обработку, включая следующих обработчиков и `onAfterResponse`.

`onAfterResponse` в HTTP выполняется фоновой задачей после отправки ответа; выполнение при аварийном завершении процесса не гарантируется. В CLI обработчики вызываются сразу в этой точке кода. В обоих режимах результат

обработчика не меняет отправленный ответ. Возврат массива или `HttpResponse` прекращает только следующих обработчиков этого события; второй ответ не отправляется.

Константы — в справочнике `ModuleEventConfig`.

События токенов

События вызываются при добавлении, изменении и удалении токенов через API или админку. Для access и refresh – отдельно.

Событие	Аргумент обработчика	Результат
onBeforeTokenAdd	array &\$fields	Изменение полей; <code>false</code> отменяет добавление
onAfterTokenAdd	Bitrix\Main\Event	Уведомление после добавления
onBeforeTokenUpdate	array &\$fields	Изменение полей; <code>false</code> отменяет обновление
onAfterTokenUpdate	Bitrix\Main\Event	Уведомление после изменения
onBeforeTokenDelete	Bitrix\Main\Event	<code>false</code> отменяет удаление
onAfterTokenDelete	Bitrix\Main\Event	Уведомление после удаления

Перед записью

В `onBeforeTokenAdd` и `onBeforeTokenUpdate` меняйте поля через `array &$fields`. Изменения проверяются перед сохранением.

Параметры Event

`onBeforeTokenDelete` и события `onAfterToken*` передают `apiContext` и доступные данные записи: `token_id`, `user_id`, `client_id`, `token_type`, `token_hash`.
`onAfterTokenAdd` и `onAfterTokenUpdate` также передают `tokenFields` с полями токена. Ключи перечислены в справочнике параметров событий.

Отмена

Возврат `false` из любого события `onBeforeToken*` отменяет операцию и прекращает вызов следующих обработчиков этого события.

Коды ошибок отмены: `TOKEN_ADD_CANCELLED`, `TOKEN_UPDATE_CANCELLED`, `TOKEN_DELETE_CANCELLED`. При выпуске, замене или отзыве пары отмена синхронного добавления или удаления откатывает операцию; API отвечает HTTP 403.

При выпуске пары событие изменения токена может вызываться дополнительно для связывания токенов. В HTTP отмена этого обновления не отменяет уже выпущенную пару. При программном выпуске в CLI она может отменить выпуск. Учитывайте это в обработчиках, запрещающих изменения токенов.

Событие `onAfterToken*` ещё не гарантирует успешное завершение всей операции: она может быть отменена. Возвращаемое значение After-обработчика игнорируется. Возврат массива для досрочного HTTP-ответа к событиям токенов не применяется.

Константы — в справочнике `ModuleEventConfig`.

Константы модуля

```
1 use Native\Api\Config\ModuleConfig;  
2  
3 $value = ModuleConfig::CONTROLLER;
```

ModuleConfig объединяет все интерфейсы констант модуля. Значения констант приведены в справочниках ниже.

Справочники

Раздел	Интерфейс
CORS	CorsConfig
HTTP-заголовки	HeaderConfig
HTTP-статусы	HttpStatusConfig
OpenAPI	OpenApiConfig
Безопасность	SecurityConfig
Кеш	CacheConfig
Контроллеры	ControllerConfig
Настройки	OptionConfig
Основные	BaseConfig
Ответ	ResponseConfig
Ошибки	ExceptionConfig
Параметры	ParameterConfig
Права и группы	AccessConfig
Провайдер	ProviderConfig
Результаты	StatusConfig
Роуты	RouteConfig
Сегменты роутов	SegmentConfig
События API	ModuleEventConfig
События аудита	EventConfig

Статусы запросов	RequestStatusConfig
Типы и форматы	TypeConfig

CORS

```
1 use Native\Api\Config\Internals\CorsConfig;
2
3 $value = CorsConfig::ORIGIN_ALLOWED;
```

Константа	Значение
ORIGIN_ALLOWED	*
ORIGINS_ALLOWED	['*']
METHODS_ALLOWED	['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS', 'HEAD']
HEADERS_ALLOWED	['Origin', 'Content-Type', 'Authorization', 'X-Requested-With']
HEADERS_EXPOSED	['Retry-After']
MAX_AGE	3600

HTTP-заголовки

```
1 use Native\Api\Config\Internals\HeaderConfig;  
2  
3 $value = HeaderConfig::HEADER_AUTHORIZATION;
```

Константа	Значение
HEADER_ORIGIN	Origin
HEADER_CONTENT_TYPE	Content-Type
HEADER_USER_AGENT	User-Agent
HEADER_ACCEPT_LANGUAGE	Accept-Language
HEADER_SEC_CH_UA	Sec-Ch-Ua
HEADER_SEC_CH_UA_MOBILE	Sec-Ch-Ua-Mobile
HEADER_SEC_CH_UA_PLATFORM	Sec-Ch-Ua-Platform
HEADER_ACCESS_CONTROL_ORIGIN_ALLOWED	Access-Control-Allow-Origin
HEADER_ACCESS_CONTROL_METHODS_ALLOWED	Access-Control-Allow-Methods
HEADER_ACCESS_CONTROL_HEADERS_ALLOWED	Access-Control-Allow-Headers
HEADER_ACCESS_CONTROL_HEADERS_EXPOSED	Access-Control-Expose-Headers
HEADER_ACCESS_CONTROL_HEADERS_REQUESTED	Access-Control-Request-Headers
HEADER_ACCESS_CONTROL_MAX_AGE	Access-Control-Max-Age
HEADER_AUTHORIZATION	Authorization
HEADER_X_REQUESTED_WITH	X-Requested-With
HEADER_X_POWERED_CMS_PACKAGE	X-Powered-CMS-Package
HEADER_X_REQUEST_ID	X-Request-Id
HEADER_RETRY_AFTER	Retry-After
HEADER_VARY	Vary

HTTP-статусы

```
1 use Native\Api\Config\Internals\HttpStatusConfig;  
2  
3 $value = HttpStatusConfig::HTTP_STATUS_200;
```

1xx — информация

Константа	Значение
HTTP_STATUS_100	100
HTTP_STATUS_101	101
HTTP_STATUS_102	102
HTTP_STATUS_103	103

2xx — успех

Константа	Значение
HTTP_STATUS_200	200
HTTP_STATUS_201	201
HTTP_STATUS_202	202
HTTP_STATUS_203	203
HTTP_STATUS_204	204
HTTP_STATUS_205	205
HTTP_STATUS_206	206
HTTP_STATUS_207	207
HTTP_STATUS_208	208
HTTP_STATUS_226	226

3xx — перенаправления

Константа	Значение

HTTP_STATUS_300	300
HTTP_STATUS_301	301
HTTP_STATUS_302	302
HTTP_STATUS_303	303
HTTP_STATUS_304	304
HTTP_STATUS_307	307
HTTP_STATUS_308	308

4xx — ошибки клиента

Константа	Значение
HTTP_STATUS_400	400
HTTP_STATUS_401	401
HTTP_STATUS_402	402
HTTP_STATUS_403	403
HTTP_STATUS_404	404
HTTP_STATUS_405	405
HTTP_STATUS_406	406
HTTP_STATUS_407	407
HTTP_STATUS_408	408
HTTP_STATUS_409	409
HTTP_STATUS_410	410
HTTP_STATUS_411	411
HTTP_STATUS_412	412
HTTP_STATUS_413	413
HTTP_STATUS_414	414
HTTP_STATUS_415	415
HTTP_STATUS_416	416

HTTP_STATUS_417	417
HTTP_STATUS_418	418
HTTP_STATUS_421	421
HTTP_STATUS_422	422
HTTP_STATUS_423	423
HTTP_STATUS_424	424
HTTP_STATUS_425	425
HTTP_STATUS_426	426
HTTP_STATUS_428	428
HTTP_STATUS_429	429
HTTP_STATUS_431	431
HTTP_STATUS_451	451

5xx — ошибки сервера

Константа	Значение
HTTP_STATUS_500	500
HTTP_STATUS_501	501
HTTP_STATUS_502	502
HTTP_STATUS_503	503
HTTP_STATUS_504	504
HTTP_STATUS_505	505
HTTP_STATUS_506	506
HTTP_STATUS_507	507
HTTP_STATUS_508	508
HTTP_STATUS_510	510
HTTP_STATUS_511	511

Массив статусов

```
1 public const HTTP_STATUS = [  
2     self::HTTP_STATUS_100 => self::HTTP_STATUS_100_TITLE,  
3     // ... 1xx  
4     self::HTTP_STATUS_200 => self::HTTP_STATUS_200_TITLE,  
5     // ... 2xx  
6     self::HTTP_STATUS_300 => self::HTTP_STATUS_300_TITLE,  
7     // ... 3xx  
8     self::HTTP_STATUS_400 => self::HTTP_STATUS_400_TITLE,  
9     // ... 4xx  
10    self::HTTP_STATUS_500 => self::HTTP_STATUS_500_TITLE,  
11    // ... 5xx  
12 ];
```

Названия статусов

```
1 use Native\Api\Config\Internals\HttpStatusConfig;  
2  
3 $value = HttpStatusConfig::HTTP_STATUS_200_TITLE;
```

Название по умолчанию

Константа	Значение
HTTP_STATUS_DEFAULT_TITLE	Unknown Status

1xx — информация

Константа	Значение
HTTP_STATUS_100_TITLE	100 Continue
HTTP_STATUS_101_TITLE	101 Switching Protocols
HTTP_STATUS_102_TITLE	102 Processing
HTTP_STATUS_103_TITLE	103 Early Hints

2xx — успех

Константа	Значение
HTTP_STATUS_200_TITLE	200 OK
HTTP_STATUS_201_TITLE	201 Created
HTTP_STATUS_202_TITLE	202 Accepted
HTTP_STATUS_203_TITLE	203 Non-Authoritative Information
HTTP_STATUS_204_TITLE	204 No Content
HTTP_STATUS_205_TITLE	205 Reset Content
HTTP_STATUS_206_TITLE	206 Partial Content
HTTP_STATUS_207_TITLE	207 Multi-Status
HTTP_STATUS_208_TITLE	208 Already Reported

HTTP_STATUS_226_TITLE	226 IM Used
-----------------------	-------------

3xx — перенаправления

Константа	Значение
HTTP_STATUS_300_TITLE	300 Multiple Choices
HTTP_STATUS_301_TITLE	301 Moved Permanently
HTTP_STATUS_302_TITLE	302 Found
HTTP_STATUS_303_TITLE	303 See Other
HTTP_STATUS_304_TITLE	304 Not Modified
HTTP_STATUS_307_TITLE	307 Temporary Redirect
HTTP_STATUS_308_TITLE	308 Permanent Redirect

4xx — ошибки клиента

Константа	Значение
HTTP_STATUS_400_TITLE	400 Bad Request
HTTP_STATUS_401_TITLE	401 Unauthorized
HTTP_STATUS_402_TITLE	402 Payment Required
HTTP_STATUS_403_TITLE	403 Forbidden
HTTP_STATUS_404_TITLE	404 Not Found
HTTP_STATUS_405_TITLE	405 Method Not Allowed
HTTP_STATUS_406_TITLE	406 Not Acceptable
HTTP_STATUS_407_TITLE	407 Proxy Authentication Required
HTTP_STATUS_408_TITLE	408 Request Timeout
HTTP_STATUS_409_TITLE	409 Conflict
HTTP_STATUS_410_TITLE	410 Gone
HTTP_STATUS_411_TITLE	411 Length Required
HTTP_STATUS_412_TITLE	412 Precondition Failed

HTTP_STATUS_413_TITLE	413 Content Too Large
HTTP_STATUS_414_TITLE	414 URI Too Long
HTTP_STATUS_415_TITLE	415 Unsupported Media Type
HTTP_STATUS_416_TITLE	416 Range Not Satisfiable
HTTP_STATUS_417_TITLE	417 Expectation Failed
HTTP_STATUS_418_TITLE	418 I'm a teapot
HTTP_STATUS_421_TITLE	421 Misdirected Request
HTTP_STATUS_422_TITLE	422 Unprocessable Content
HTTP_STATUS_423_TITLE	423 Locked
HTTP_STATUS_424_TITLE	424 Failed Dependency
HTTP_STATUS_425_TITLE	425 Too Early
HTTP_STATUS_426_TITLE	426 Upgrade Required
HTTP_STATUS_428_TITLE	428 Precondition Required
HTTP_STATUS_429_TITLE	429 Too Many Requests
HTTP_STATUS_431_TITLE	431 Request Header Fields Too Large
HTTP_STATUS_451_TITLE	451 Unavailable For Legal Reasons

5xx — ошибки сервера

Константа	Значение
HTTP_STATUS_500_TITLE	500 Internal Server Error
HTTP_STATUS_501_TITLE	501 Not Implemented
HTTP_STATUS_502_TITLE	502 Bad Gateway
HTTP_STATUS_503_TITLE	503 Service Unavailable
HTTP_STATUS_504_TITLE	504 Gateway Timeout
HTTP_STATUS_505_TITLE	505 HTTP Version Not Supported
HTTP_STATUS_506_TITLE	506 Variant Also Negotiates
HTTP_STATUS_507_TITLE	507 Insufficient Storage

HTTP_STATUS_508_TITLE	508 Loop Detected
HTTP_STATUS_510_TITLE	510 Not Extended
HTTP_STATUS_511_TITLE	511 Network Authentication Required

OpenAPI

```
1 use Native\Api\Config\Internals\OpenApiConfig;  
2  
3 $value = OpenApiConfig::OPENAPI;
```

Документ

Константа	Значение
OPENAPI	openapi
INFO	info
TITLE	title
VERSION	version
TERMS_OF_SERVICE	termsOfService
CONTACT	contact
LICENSE	license
EMAIL	email
URL	url
PATHS	paths
COMPONENTS	components
SCHEMAS	schemas
SECURITY_SCHEMES	securitySchemes
SCHEME	scheme

Операции

Константа	Значение
TAGS	tags
SUMMARY	summary
DESCRIPTION	description

EXTERNAL_DOCS	externalDocs
OPERATION_ID	operationId
PARAMETERS	parameters
REQUEST_BODY	requestBody
RESPONSES	responses
CALLBACKS	callbacks
DEPRECATED	deprecated
SECURITY	security
SERVERS	servers

Параметры и схемы

Константа	Значение
NAME	name
IN	in
REQUIRED	required
SCHEMA	schema
CONTENT	content
TYPE	type
FORMAT	format
PROPERTIES	properties
ITEMS	items
ENUM	enum
MINIMUM	minimum
MAXIMUM	maximum
MIN_ITEMS	minItems
MAX_ITEMS	maxItems
NULLABLE	nullable

ADDITIONAL_PROPERTIES	additionalProperties
ONE_OF	oneOf
ALL_OF	allOf
REF	\$ref
EXAMPLE	example
EXAMPLES	examples
VALUE	value

Общие схемы ответа

Константа	Значение
SCHEMA_RESPONSE_TIME	ResponseTime
SCHEMA_RESPONSE_STATUS	ResponseStatus
SCHEMA_RESPONSE_ERROR	ResponseError

Безопасность

```
1 use Native\Api\Config\Internals\SecurityConfig;  
2  
3 $value = SecurityConfig::AUTH;
```

Авторизация

Константа	Значение
AUTH	auth
LOGIN	login
TOKEN	token
TOKEN_TYPE	token_type
TOKEN_ACCESS	access
TOKEN_REFRESH	refresh

Группы

Константа	Значение
GROUP	group
WHITELIST	whitelist
BLACKLIST	blacklist

IP-адреса

Константа	Значение
IP	ip

Схемы OpenAPI

Константа	Значение
SECURITY_SCHEME_BASIC	basicAuth

SECURITY_SCHEME_BEARER	bearerAuth
------------------------	------------

Результат отзыва токенов

Константа	Значение
TOKEN_CLIENT_NOT_FOUND	CLIENT_NOT_FOUND

Сроки действия задаются в настройках безопасности.

Кеш

```
1 use Native\Api\Config\Internals\CacheConfig;  
2  
3 $value = CacheConfig::TTL;
```

Константа	Значение
TTL	ttl

Контроллеры

```
1 use Native\Api\Config\Internals\ControllerConfig;
2
3 $value = ControllerConfig::CONTROLLER;
```

Константа	Значение
CONTROLLER	controller
ACTION	action

Настройки

```
1 use Native\Api\Config\Internals\OptionConfig;
2
3 $value = OptionConfig::PATH_MODULE;
```

Пути

Константа	Значение
PATH_MODULE	path_module
PATH_DOCS	path_docs
PATH_ROUTE_SETTINGS	path_route_settings
PATH_LOG	path_log

Роуты

Константа	Значение
ROUTE_NATIVE_ACTIVE	route_native_active
ROUTE_PROVIDER_ACTIVE	route_provider_active
ROUTE_SERVICE_ACTIVE	route_service_active
ROUTE_TECHNICAL_KEYS_DISABLED	route_technical_keys_disabled
ROUTE_CACHE_TTL	route_cache_ttl

Токены

Константа	Значение
TOKEN_ACCESS_TTL	token_access_ttl
TOKEN_REFRESH_TTL	token_refresh_ttl

IP-адреса

Константа	Значение
-----------	----------

REQUEST_IP_BLACKLIST	request_ip_blacklist
REQUEST_IP_WHITELIST	request_ip_whitelist

Расписания и лимиты

Константа	Значение
REQUEST_SCHEDULE_DEFAULT	request_schedule_default
REQUEST_SCHEDULE_GROUPS	request_schedule_groups
REQUEST_LIMIT_DEFAULT	request_limit_default
REQUEST_LIMIT_GROUPS	request_limit_groups

CORS

Константа	Значение
REQUEST_CORS_ORIGINS_ALLOWED	request_cors_origins_allowed
REQUEST_CORS_HEADERS_ALLOWED	request_cors_headers_allowed
REQUEST_CORS_HEADERS_EXPOSED	request_cors_headers_exposed
REQUEST_CORS_MAX_AGE	request_cors_max_age

Аналитика

Константа	Значение
REQUEST_COLLECT_ACTIVE	request_collect_active
ANALYTICS_REQUEST_METHODS	analytics_request_methods

Передача данных

Константа	Значение
SEND_TO_VENDOR	send_to_vendor

Документация

Константа	Значение
DOCS_PAGE_TITLE	docs_page_title
DOCS_TITLE	docs_title
DOCS_DESCRIPTION	docs_description
DOCS_VERSION	docs_version
DOCS_CONTACT_NAME	docs_contact_name
DOCS_CONTACT_URL	docs_contact_url
DOCS_CONTACT_EMAIL	docs_contact_email
DOCS_TERMS_URL	docs_terms_url
DOCS_LICENSE_NAME	docs_license_name
DOCS_LICENSE_URL	docs_license_url
DOCS_SERVER_NAME	docs_server_name
DOCS_SERVER_URL	docs_server_url

Логирование

Данные запросов и ответов

Константа	Значение
EVENT_LOG_REQUEST_DATA	event_log_request_data
EVENT_LOG_RESPONSE_DATA	event_log_response_data
FILE_LOG_REQUEST_DATA	file_log_request_data
FILE_LOG_RESPONSE_DATA	file_log_response_data

Журнал событий

Константа	Значение
EVENT_LOG_SETTINGS_UPDATE	event_log_settings_update

EVENT_LOG_LAUNCH_ERROR	event_log_launch_error
EVENT_LOG_CACHE_CLEAR	event_log_cache_clear
EVENT_LOG_USER_EVENT_HANDLER	event_log_user_event_handler
EVENT_LOG_TOKEN_CREATE	event_log_token_create
EVENT_LOG_TOKEN_UPDATE	event_log_token_update
EVENT_LOG_TOKEN_DELETE	event_log_token_delete
EVENT_LOG_TOKEN_HASH_UPDATE	event_log_token_hash_update
EVENT_LOG_REQUEST_DENIED_GROUP_BLACKLIST	event_log_request_denied_group_blacklist
EVENT_LOG_REQUEST_DENIED_GROUP_WHITELIST	event_log_request_denied_group_whitelist
EVENT_LOG_REQUEST_DENIED_IP_BLACKLIST	event_log_request_denied_ip_blacklist
EVENT_LOG_REQUEST_DENIED_IP_WHITELIST	event_log_request_denied_ip_whitelist
EVENT_LOG_REQUEST_DENIED_SCHEDULE	event_log_request_denied_schedule
EVENT_LOG_REQUEST_DENIED_RATE_LIMIT	event_log_request_denied_rate_limit
EVENT_LOG_REQUEST_DENIED_CORS	event_log_request_denied_cors
EVENT_LOG_SEND_TO_VENDOR	event_log_send_to_vendor

Файловый журнал

Константа	Значение
FILE_LOG_SETTINGS_UPDATE	file_log_settings_update
FILE_LOG_LAUNCH_ERROR	file_log_launch_error
FILE_LOG_CACHE_CLEAR	file_log_cache_clear
FILE_LOG_USER_EVENT_HANDLER	file_log_user_event_handler
FILE_LOG_TOKEN_CREATE	file_log_token_create
FILE_LOG_TOKEN_UPDATE	file_log_token_update
FILE_LOG_TOKEN_DELETE	file_log_token_delete
FILE_LOG_TOKEN_HASH_UPDATE	file_log_token_hash_update

FILE_LOG_REQUEST_DENIED_GROUP_BLACKLIST	file_log_request_denied_group_blacklist
FILE_LOG_REQUEST_DENIED_GROUP_WHITELIST	file_log_request_denied_group_whitelist
FILE_LOG_REQUEST_DENIED_IP_BLACKLIST	file_log_request_denied_ip_blacklist
FILE_LOG_REQUEST_DENIED_IP_WHITELIST	file_log_request_denied_ip_whitelist
FILE_LOG_REQUEST_DENIED_SCHEDULE	file_log_request_denied_schedule
FILE_LOG_REQUEST_DENIED_RATE_LIMIT	file_log_request_denied_rate_limit
FILE_LOG_REQUEST_DENIED_CORS	file_log_request_denied_cors
FILE_LOG_SEND_TO_VENDOR	file_log_send_to_vendor

Права интерфейса

Константа	Значение
ACCESS_RIGHTS_READ_DASHBOARD	access_rights_read_dashboard
ACCESS_RIGHTS_READ_DOCS	access_rights_read_docs
ACCESS_RIGHTS_READ_LOG	access_rights_read_log
ACCESS_RIGHTS_READ_REQUEST_LIST	access_rights_read_request_list
ACCESS_RIGHTS_READ_SETTINGS	access_rights_read_settings
ACCESS_RIGHTS_READ_TOKEN_LIST	access_rights_read_token_list
ACCESS_RIGHTS_WRITE_REQUEST	access_rights_write_request
ACCESS_RIGHTS_WRITE_SETTINGS	access_rights_write_settings
ACCESS_RIGHTS_WRITE_TOKEN	access_rights_write_token

Основные

```
1 use Native\Api\Config\Internals\BaseConfig;
2
3 $value = BaseConfig::MODULE_NAME;
```

Константа	Значение
MODULE_NAME	API-платформа
MODULE_ID	native.api

Ответ

```
1 use Native\Api\Config\Internals\ResponseConfig;  
2  
3 $value = ResponseConfig::STATUS;
```

Управление ответом

Константа	Значение
STATUS	status
HEADERS	headers
STATUS_MESSAGE	statusMessage
TECHNICAL_KEYS_DISABLED	technical_keys_disabled

Данные ответа

Константа	Значение
RESPONSE	response
REQUEST_ID	request_id
RESULT	result
ERROR	error
DATA	data
CODE	code
TITLE	title
MESSAGE	message
METHOD	method

Время обработки

Константа	Значение
TIME	time

TIME_TIMESTAMP	timestamp
TIME_START	start
TIME_FINISH	finish
TIME_DURATION	duration
TIME_DATE_START	date_start
TIME_DATE_FINISH	date_finish

Ошибки

```
1 use Native\Api\Config\Internals\ExceptionConfig;
2
3 $value = ExceptionConfig::INTERNAL_SERVER_ERROR;
```

Модуль и события

Константа	Значение
INTERNAL_SERVER_ERROR	INTERNAL_SERVER_ERROR
ERROR_MODULE_BOOT_FAILED	MODULE_BOOT_FAILED
ERROR_MODULE_NOT_FOUND	MODULE_NOT_FOUND
ERROR_MODULE_INACTIVE	MODULE_INACTIVE
ERROR_EVENT_HANDLER_ABORTED	EVENT_HANDLER_ABORTED
ERROR_EVENT_HANDLER_RESULT_INVALID	EVENT_HANDLER_RESULT_INVALID

Роуты и контроллеры

Константа	Значение
ERROR_ROUTE_MAP_NOT_FOUND	ROUTE_MAP_NOT_FOUND
ERROR_ROUTE_MAP_INVALID	ROUTE_MAP_INVALID
ERROR_ROUTE_MAP_LOAD_FAILED	ROUTE_MAP_LOAD_FAILED
ERROR_ROUTE_CONFIG_NOT_FOUND	ROUTE_CONFIG_NOT_FOUND
ERROR_ROUTE_NOT_FOUND	ROUTE_NOT_FOUND
ERROR_ROUTE_DISABLED	ROUTE_DISABLED
ERROR_CONTROLLER_NOT_FOUND	CONTROLLER_NOT_FOUND
ERROR_CONTROLLER_WRONG_METHOD	CONTROLLER_WRONG_METHOD
ERROR_CONTROLLER_WRONG_RESPONSE	CONTROLLER_WRONG_RESPONSE
ERROR_CONTROLLER_CREATE_FAILED	CONTROLLER_CREATE_FAILED
ERROR_CONTROLLER_EXECUTION_FAILED	CONTROLLER_EXECUTION_FAILED

ERROR_CONTROLLER_SCRIPT_EXECUTION_FAILED

CONTROLLER_SCRIPT_EXECUTION_FAILED

Параметры и провайдер

Константа	Значение
ERROR_PARAMETERS_INVALID	PARAMETERS_INVALID
ERROR_PROVIDER_CLASS_NOT_FOUND	PROVIDER_CLASS_NOT_FOUND
ERROR_PROVIDER_METHOD_NOT_FOUND	PROVIDER_METHOD_NOT_FOUND
ERROR_PROVIDER_ACCESS_DENIED	PROVIDER_ACCESS_DENIED
ERROR_PROVIDER_RESULT_CALLBACK_NOT_FOUND	PROVIDER_RESULT_CALLBACK_NOT_FOUND
ERROR_PROVIDER_EXECUTION_FAILED	PROVIDER_EXECUTION_FAILED

Доступ к API

Константа	Значение
ERROR_AUTH_REQUIRED	AUTH_REQUIRED
ERROR_AUTH_INVALID	AUTH_INVALID
ERROR_AUTH_MFA_REQUIRED	AUTH_MFA_REQUIRED
ERROR_GROUP_BLACKLIST	GROUP_BLACKLIST
ERROR_GROUP_WHITELIST	GROUP_WHITELIST
ERROR_IP_BLACKLIST	IP_BLACKLIST
ERROR_IP_WHITELIST	IP_WHITELIST
ERROR_REQUEST_DENIED_RATE_LIMIT	REQUEST_DENIED_RATE_LIMIT
ERROR_REQUEST_DENIED_SCHEDULE	REQUEST_DENIED_SCHEDULE
ERROR_CORS_DENIED	CORS_DENIED

Токены

Константа	Значение
-----------	----------

ERROR_TOKEN_HASH_KEY_NOT_CONFIGURED	TOKEN_HASH_KEY_NOT_CONFIGURED
ERROR_TOKEN_GENERATION_FAILED	TOKEN_GENERATION_FAILED
ERROR_TOKEN_CREATE_FAILED	TOKEN_CREATE_FAILED
ERROR_TOKEN_ISSUE_FORBIDDEN	TOKEN_ISSUE_FORBIDDEN
ERROR_TOKEN_ROTATION_FAILED	TOKEN_ROTATION_FAILED
ERROR_TOKEN_DELETE_FAILED	TOKEN_DELETE_FAILED
ERROR_TOKEN_DELETE_CANCELLED	TOKEN_DELETE_CANCELLED
ERROR_TOKEN_ADD_CANCELLED	TOKEN_ADD_CANCELLED
ERROR_TOKEN_UPDATE_CANCELLED	TOKEN_UPDATE_CANCELLED

Параметры

```
1 use Native\Api\Config\Internals\ParameterConfig;
2
3 $value = ParameterConfig::TYPE;
```

Константа	Значение
TYPE	type
REQUIRED	required
POSSIBLE_VALUE	possibleValue
EXAMPLE	example

Права и группы

```
1 use Native\Api\Config\Internals\AccessConfig;  
2  
3 $value = AccessConfig::GROUP_CODE_ANONYMOUS;
```

Пользователи и группы

Константа	Значение
GROUP_CODE_ANONYMOUS	anonymous
GROUP_CODE_ADMINISTRATOR	administrators
GROUP_CODE_EVERYONE	everyone

Права

Константа	Значение
PERMISSION_READ	R
PERMISSION_WRITE	W
PERMISSION_ADMIN	X

Провайдер

```
1 use Native\Api\Config\Internals\ProviderConfig;
2
3 $value = ProviderConfig::CLASS_NAME;
```

Константа	Значение
CLASS_NAME	class
METHOD_NAME	method
METHOD_PARAMETERS	parameters
RESULT_CALLBACK	callback

Результаты

```
1 use Native\Api\Config\Internals>StatusConfig;
2
3 $value = StatusConfig::SUCCESS;
```

Константа	Значение
SUCCESS	success
ERROR	error

Роуты

```
1 use Native\Api\Config\Internals\RouteConfig;
2
3 $value = RouteConfig::ROUTE_TYPE;
```

Константа	Значение
ROUTE_TYPE	type
ROUTE_TYPE_NATIVE	native
ROUTE_TYPE_PROVIDER	provider
ROUTE_TYPE_SERVICE	service
ACTIVE	active
DESCRIPTION	description
FORMAT	format
SECURITY	security
PARAMETERS	parameters
CACHE	cache
RESPONSE	response

Сегменты роутов

```
1 use Native\Api\Config\Internals\SegmentConfig;  
2  
3 $value = SegmentConfig::DYNAMIC_MARKER;
```

Константа	Значение
DYNAMIC_MARKER	*

События API

```
1 use Native\Api\Config\Internals\ModuleEventConfig;  
2  
3 $value = ModuleEventConfig::BEFORE_START;
```

Имена событий

Запуск и роутинг

Константа	Значение
BEFORE_START	onBeforeStart
AFTER_START	onAfterStart
BEFORE_CORS	onBeforeCors
AFTER_CORS	onAfterCors
BEFORE_ROUTING	onBeforeRouting
AFTER_ROUTING	onAfterRouting

Проверки доступа

Константа	Значение
BEFORE_IP_ACCESS_GLOBAL	onBeforeIpAccessGlobal
AFTER_IP_ACCESS_GLOBAL	onAfterIpAccessGlobal
BEFORE_IP_ACCESS	onBeforeIpAccess
AFTER_IP_ACCESS	onAfterIpAccess
BEFORE_AUTHORIZATION	onBeforeAuthorization
AFTER_AUTHORIZATION	onAfterAuthorization
BEFORE_GROUP_ACCESS	onBeforeGroupAccess
AFTER_GROUP_ACCESS	onAfterGroupAccess
BEFORE_SCHEDULE	onBeforeSchedule
AFTER_SCHEDULE	onAfterSchedule

BEFORE_REQUEST_LIMIT	onBeforeRequestLimit
AFTER_REQUEST_LIMIT	onAfterRequestLimit

Обработка запроса

Константа	Значение
BEFORE_PARAMETERS	onBeforeParameters
AFTER_PARAMETERS	onAfterParameters
BEFORE_CONTROLLER	onBeforeController
AFTER_CONTROLLER	onAfterController
BEFORE_RESPONSE	onBeforeResponse
AFTER_RESPONSE	onAfterResponse

Токены

Константа	Значение
BEFORE_TOKEN_ADD	onBeforeTokenAdd
AFTER_TOKEN_ADD	onAfterTokenAdd
BEFORE_TOKEN_UPDATE	onBeforeTokenUpdate
AFTER_TOKEN_UPDATE	onAfterTokenUpdate
BEFORE_TOKEN_DELETE	onBeforeTokenDelete
AFTER_TOKEN_DELETE	onAfterTokenDelete

Параметры событий

Константа	Значение
PARAMETER_API_CONTEXT	apiContext
PARAMETER_HTTP_REQUEST	httpRequest
PARAMETER_ROUTE	route
PARAMETER_RESPONSE	response

PARAMETER_EXCEPTION	exception
PARAMETER_IP_BLACKLIST	ipBlacklist
PARAMETER_IP_WHITELIST	ipWhitelist
PARAMETER_USER_ID	user_id
PARAMETER_CLIENT_ID	client_id
PARAMETER_TOKEN_ID	token_id
PARAMETER_TOKEN_TYPE	token_type
PARAMETER_TOKEN_HASH	token_hash
PARAMETER_TOKEN_FIELDS	tokenFields

События аудита

```
1 use Native\Api\Config\Internals\EventConfig;
2
3 $value = EventConfig::EVENT_TYPE_SETTINGS_UPDATE;
```

Константа	Значение
EVENT_TYPE_SETTINGS_UPDATE	SETTINGS_UPDATE
EVENT_TYPE_LAUNCH_ERROR	LAUNCH_ERROR
EVENT_TYPE_CACHE_CLEAR	CACHE_CLEAR
EVENT_TYPE_USER_EVENT_HANDLER	USER_EVENT_HANDLER
EVENT_TYPE_TOKEN_CREATE	TOKEN_CREATE
EVENT_TYPE_TOKEN_UPDATE	TOKEN_UPDATE
EVENT_TYPE_TOKEN_DELETE	TOKEN_DELETE
EVENT_TYPE_TOKEN_HASH_UPDATE	TOKEN_HASH_UPDATE
EVENT_TYPE_REQUEST_DENIED_GROUP_BLACKLIST	REQUEST_DENIED_GROUP_BLACKLIST
EVENT_TYPE_REQUEST_DENIED_GROUP_WHITELIST	REQUEST_DENIED_GROUP_WHITELIST
EVENT_TYPE_REQUEST_DENIED_IP_BLACKLIST	REQUEST_DENIED_IP_BLACKLIST
EVENT_TYPE_REQUEST_DENIED_IP_WHITELIST	REQUEST_DENIED_IP_WHITELIST
EVENT_TYPE_REQUEST_DENIED_RATE_LIMIT	REQUEST_DENIED_RATE_LIMIT
EVENT_TYPE_REQUEST_DENIED_SCHEDULE	REQUEST_DENIED_SCHEDULE
EVENT_TYPE_REQUEST_DENIED_CORS	REQUEST_DENIED_CORS
EVENT_TYPE_SEND_TO_VENDOR	SEND_TO_VENDOR

История запросов

Константа	Значение
EVENT_TYPE_REQUEST_RECEIVED	REQUEST_RECEIVED

Статусы запросов

```
1 use Native\Api\Config\Internals\RequestStatusConfig;  
2  
3 $value = RequestStatusConfig::PROCESSED;
```

Константа	Значение
PROCESSED	PROCESSED
REJECTED	REJECTED

Типы и форматы

```
1 use Native\Api\Config\Internals\TypeConfig;  
2  
3 $value = TypeConfig::STRING;
```

Типы параметров

Константа	Значение
STRING	string
INTEGER	integer
ARRAY	array

Форматы ответа

Константа	Значение
FORMAT_JSON	json
FORMAT_XML	xml
CONTENT_TYPE_JSON	application/json
CONTENT_TYPE_XML	text/xml

Пример карты

Карта разделена по HTTP-методам, как встроенные роуты модуля.

Разместите их в `local/php_interface/api/routes/`. Классы `Local\Api`, контроллер отчёта, коды групп и адреса замените данными своего проекта.

Схема сборки карты

```

1  local/php_interface/api/routes/
2  |
3  ├── GET/orders.php ───┐
4  ├── POST/orders.php ─┐
5  ├── PUT/orders.php ──┐
6  ├── DELETE/orders.php ─┐
7  |                       ┘
8  └── index.php ─────────┘
9  |
10 └── «Путь карты роутов» в настройках модуля

```

Контроллеры в карте

Импорты для примеров:

```

1  use Local\Api\OrderController;
2  use Local\Api\OrderAction;
3  use Native\Api\Config\Internals\ControllerConfig;

```

Запись массивом

```

1  // Рекомендуемая запись класса и метода
2  ControllerConfig::CONTROLLER => [OrderController::class, 'get'],

```

Запись строкой

```

1  // Альтернативная запись того же класса и метода
2  ControllerConfig::CONTROLLER => OrderController::class . '::get',

```

Вызываемый класс

```
1 // Класс с методом __invoke()
2 ControllerConfig::CONTROLLER => OrderAction::class,
```

PHP-файл

```
1 // Контроллер в PHP-файле
2 ControllerConfig::CONTROLLER => $_SERVER['DOCUMENT_ROOT']
3 . '/local/api/controllers/order.php',
```

GET — получение

local/php_interface/api/routes/GET/orders.php

```
1 <?php
2
3 use Native\Api\Config\Internals\OpenApiConfig;
4 use Native\Api\Config\Internals\CacheConfig;
5 use Native\Api\Config\Internals\ControllerConfig;
6 use Native\Api\Config\Internals\ParameterConfig;
7 use Native\Api\Config\Internals\ResponseConfig;
8 use Native\Api\Config\Internals\RouteConfig;
9 use Native\Api\Config\Internals\SecurityConfig;
10 use Native\Api\Config\Internals\TypeConfig;
11 use Local\Api\OrderController;
12 use Local\Api>StatusController;
13
14 return [
15     'orders/{id}' => [
16         ControllerConfig::CONTROLLER => [OrderController::class, 'get'],
17         RouteConfig::DESCRIPTION => 'Получение заказа',
18         RouteConfig::ACTIVE => true,
19         RouteConfig::FORMAT => TypeConfig::FORMAT_XML,
20         RouteConfig::SECURITY => [
21             SecurityConfig::AUTH => SecurityConfig::LOGIN,
22         ],
23         RouteConfig::PARAMETERS => [
24             'id' => [
25                 ParameterConfig::TYPE => TypeConfig::INTEGER,
26                 ParameterConfig::REQUIRED => true,
27                 RouteConfig::DESCRIPTION => 'ID заказа',
28                 ParameterConfig::EXAMPLE => 501,
29             ],
30         ],
31         RouteConfig::CACHE => [
32             CacheConfig::TTL => 300,
33         ],
34         RouteConfig::RESPONSE => [
35             ResponseConfig::TECHNICAL_KEYS_DISABLED => false,
36         ],
37         OpenApiConfig::TAGS => ['Заказы'],
38         OpenApiConfig::OPERATION_ID => 'getOrder',
```

```

39     ],
40     'status' => [
41         ControllerConfig::CONTROLLER => StatusController::class,
42         RouteConfig::ROUTE_TYPE => RouteConfig::ROUTE_TYPE_SERVICE,
43         RouteConfig::DESCRIPTION => 'Состояние интеграции',
44         RouteConfig::CACHE => [
45             CacheConfig::TTL => 0,
46         ],
47     ],
48     'reports/{id}' => [
49         ControllerConfig::CONTROLLER => $_SERVER['DOCUMENT_ROOT']
50             . '/local/api/controllers/report.php',
51         RouteConfig::ACTIVE => false,
52         RouteConfig::DESCRIPTION => 'Отчёт: описание параметров средствами OpenAPI',
53         OpenApiConfig::PARAMETERS => [
54             [
55                 OpenApiConfig::NAME => 'id',
56                 OpenApiConfig::IN => 'path',
57                 OpenApiConfig::REQUIRED => true,
58                 OpenApiConfig::DESCRIPTION => 'ID отчёта',
59                 OpenApiConfig::SCHEMA => [OpenApiConfig::TYPE => 'integer'],
60                 ParameterConfig::EXAMPLE => 15,
61             ],
62             [
63                 OpenApiConfig::NAME => 'page',
64                 OpenApiConfig::IN => 'query',
65                 OpenApiConfig::REQUIRED => false,
66                 OpenApiConfig::DESCRIPTION => 'Номер страницы',
67                 OpenApiConfig::SCHEMA => [
68                     OpenApiConfig::TYPE => 'integer',
69                     OpenApiConfig::MINIMUM => 1,
70                     'default' => 1,
71                 ],
72             ],
73             [
74                 OpenApiConfig::NAME => 'X-Client-Version',
75                 OpenApiConfig::IN => 'header',
76                 OpenApiConfig::REQUIRED => false,
77                 OpenApiConfig::SCHEMA => [OpenApiConfig::TYPE => 'string'],
78             ],
79         ],
80         OpenApiConfig::SECURITY => [
81             [SecurityConfig::SECURITY_SCHEME_BEARER => []],
82         ],
83         OpenApiConfig::TAGS => ['Отчёты'],
84         OpenApiConfig::OPERATION_ID => 'getReport',
85         OpenApiConfig::DEPRECATED => true,
86     ],
87 ];

```

`status` зависит от активации сервисных роутов. Роут `reports/{id}` отключён: его списки `parameters` и `security` описывают документацию, но не включают проверки параметров и авторизацию.

POST — создание

`local/php_interface/api/routes/POST/orders.php`

```

1  <?php
2
3  use Native\Api\Config\Internals\OpenApiConfig;
4  use Native\Api\Config\Internals\CacheConfig;
5  use Native\Api\Config\Internals\ControllerConfig;
6  use Native\Api\Config\Internals\ParameterConfig;
7  use Native\Api\Config\Internals\ResponseConfig;
8  use Native\Api\Config\Internals\RouteConfig;
9  use Native\Api\Config\Internals\SecurityConfig;
10 use Native\Api\Config\Internals\TypeConfig;
11 use Local\Api\OrderController;
12
13 return [
14     'orders' => [
15         ControllerConfig::CONTROLLER => [OrderController::class, 'create'],
16         RouteConfig::ACTIVE => true,
17         RouteConfig::DESCRIPTION => 'Создание заказа',
18         RouteConfig::FORMAT => TypeConfig::FORMAT_JSON,
19         RouteConfig::SECURITY => [
20             SecurityConfig::AUTH => SecurityConfig::TOKEN,
21             SecurityConfig::TOKEN_TYPE => SecurityConfig::TOKEN_ACCESS,
22             SecurityConfig::GROUP => [
23                 SecurityConfig::WHITELIST => ['api_clients'],
24                 SecurityConfig::BLACKLIST => ['api_blocked'],
25             ],
26         ],
27         RouteConfig::PARAMETERS => [
28             'customer' => [
29                 ParameterConfig::TYPE => TypeConfig::ARRAY,
30                 ParameterConfig::REQUIRED => true,
31                 RouteConfig::DESCRIPTION => 'Покупатель',
32                 RouteConfig::PARAMETERS => [
33                     'name' => [
34                         ParameterConfig::TYPE => TypeConfig::STRING,
35                         ParameterConfig::REQUIRED => true,
36                         RouteConfig::DESCRIPTION => 'Имя',
37                         ParameterConfig::EXAMPLE => 'Иван',
38                     ],
39                     'email' => [
40                         ParameterConfig::TYPE => TypeConfig::STRING,
41                         ParameterConfig::REQUIRED => true,
42                         RouteConfig::DESCRIPTION => 'Email',
43                         ParameterConfig::EXAMPLE => 'customer@example.com',
44                     ],
45                 ],
46             ],
47             OpenApiConfig::ITEMS => [
48                 ParameterConfig::TYPE => TypeConfig::ARRAY,
49                 ParameterConfig::REQUIRED => true,
50                 RouteConfig::DESCRIPTION => 'Позиции заказа',
51                 RouteConfig::PARAMETERS => [
52                     [
53                         'productId' => [
54                             ParameterConfig::TYPE => TypeConfig::INTEGER,
55                             ParameterConfig::REQUIRED => true,
56                             RouteConfig::DESCRIPTION => 'ID товара',
57                             ParameterConfig::EXAMPLE => 101,
58                         ],
59                         'quantity' => [
60                             ParameterConfig::TYPE => TypeConfig::INTEGER,
61                             ParameterConfig::REQUIRED => true,
62                             RouteConfig::DESCRIPTION => 'Количество',
63                             ParameterConfig::EXAMPLE => 2,
64                         ],
65                     ],

```

```

66         ],
67     ],
68     'delivery' => [
69         ParameterConfig::TYPE => TypeConfig::STRING,
70         ParameterConfig::REQUIRED => true,
71         ParameterConfig::POSSIBLE_VALUE => ['pickup', 'courier'],
72         RouteConfig::DESCRIPTION => 'Способ получения',
73         ParameterConfig::EXAMPLE => 'pickup',
74     ],
75     'callbackUrl' => [
76         ParameterConfig::TYPE => TypeConfig::STRING,
77         ParameterConfig::REQUIRED => false,
78         RouteConfig::DESCRIPTION => 'Адрес уведомления о смене статуса',
79         ParameterConfig::EXAMPLE => 'https://client.example.com/order-status',
80     ],
81 ],
82 RouteConfig::CACHE => [
83     CacheConfig::TTL => 0,
84 ],
85 RouteConfig::RESPONSE => [
86     ResponseConfig::TECHNICAL_KEYS_DISABLED => true,
87 ],
88 OpenApiConfig::TAGS => ['Заказы'],
89 OpenApiConfig::SUMMARY => 'Создать заказ',
90 OpenApiConfig::OPERATION_ID => 'createOrder',
91 OpenApiConfig::DEPRECATED => false,
92 OpenApiConfig::EXTERNAL_DOCS => [
93     OpenApiConfig::DESCRIPTION => 'Правила оформления заказа',
94     OpenApiConfig::URL => 'https://example.com/docs/orders',
95 ],
96 OpenApiConfig::SERVERS => [
97     [
98         OpenApiConfig::URL => 'https://{environment}.example.com/api',
99         OpenApiConfig::DESCRIPTION => 'Сервер интеграции',
100         'variables' => [
101             'environment' => [
102                 'default' => 'test',
103                 OpenApiConfig::ENUM => ['test', 'production'],
104             ],
105         ],
106     ],
107 ],
108 OpenApiConfig::REQUEST_BODY => [
109     OpenApiConfig::DESCRIPTION => 'Данные заказа',
110     OpenApiConfig::REQUIRED => true,
111     OpenApiConfig::CONTENT => [
112         TypeConfig::CONTENT_TYPE_JSON => [
113             OpenApiConfig::SCHEMA => [
114                 OpenApiConfig::REF => '#/components/schemas/CreateOrder',
115             ],
116             OpenApiConfig::EXAMPLE => [
117                 'customer' => [
118                     OpenApiConfig::NAME => 'Иван',
119                     OpenApiConfig::EMAIL => 'customer@example.com',
120                 ],
121                 OpenApiConfig::ITEMS => [
122                     ['productId' => 101, 'quantity' => 2],
123                 ],
124                 'delivery' => 'pickup',
125             ],
126         ],
127     ],
128 ],
129 OpenApiConfig::RESPONSES => [
130     '201' => [
131         OpenApiConfig::DESCRIPTION => 'Заказ создан',

```

```

132         'headers' => [
133             'Location' => [
134                 OpenApiConfig::DESCRIPTION => 'Адрес созданного заказа',
135                 OpenApiConfig::SCHEMA => [OpenApiConfig::TYPE => 'string'],
136             ],
137         ],
138         OpenApiConfig::CONTENT => [
139             TypeConfig::CONTENT_TYPE_JSON => [
140                 OpenApiConfig::SCHEMA => [OpenApiConfig::REF =>
'#/components/schemas/Order'],
141                 OpenApiConfig::EXAMPLE => ['id' => 501, 'orderStatus' => 'new'],
142             ],
143         ],
144     ],
145     '400' => [
146         OpenApiConfig::DESCRIPTION => 'Ошибка параметров запроса',
147     ],
148     '401' => [
149         OpenApiConfig::DESCRIPTION => 'Требуется авторизация',
150     ],
151     '403' => [
152         OpenApiConfig::DESCRIPTION => 'Доступ запрещён',
153     ],
154 ],
155 OpenApiConfig::CALLBACKS => [
156     'orderStatusChanged' => [
157         '{$request.body#/callbackUrl}' => [
158             'post' => [
159                 OpenApiConfig::SUMMARY => 'Уведомление о смене статуса',
160                 OpenApiConfig::REQUEST_BODY => [
161                     OpenApiConfig::REQUIRED => true,
162                     OpenApiConfig::CONTENT => [
163                         TypeConfig::CONTENT_TYPE_JSON => [
164                             OpenApiConfig::SCHEMA => [OpenApiConfig::REF =>
'#/components/schemas/Order'],
165                         ],
166                     ],
167                 ],
168                 OpenApiConfig::RESPONSES => [
169                     '204' => [OpenApiConfig::DESCRIPTION => 'Уведомление принято'],
170                 ],
171             ],
172         ],
173     ],
174 ],
175 OpenApiConfig::COMPONENTS => [
176     OpenApiConfig::SCHEMAS => [
177         'CreateOrder' => [
178             OpenApiConfig::TYPE => 'object',
179             OpenApiConfig::REQUIRED => ['customer', 'items', 'delivery'],
180             OpenApiConfig::PROPERTIES => [
181                 'customer' => [
182                     OpenApiConfig::TYPE => 'object',
183                     OpenApiConfig::REQUIRED => ['name', OpenApiConfig::EMAIL],
184                     OpenApiConfig::PROPERTIES => [
185                         'name' => [OpenApiConfig::TYPE => 'string'],
186                         'email' => [OpenApiConfig::TYPE => 'string'],
187                     ],
188                 ],
189                 OpenApiConfig::ITEMS => [
190                     OpenApiConfig::TYPE => 'array',
191                     OpenApiConfig::MIN_ITEMS => 1,
192                     OpenApiConfig::ITEMS => [
193                         OpenApiConfig::TYPE => 'object',
194                         OpenApiConfig::REQUIRED => ['productId', 'quantity'],

```

```

195         OpenApiConfig::PROPERTIES => [
196             'productId' => [OpenApiConfig::TYPE => 'integer'],
197             'quantity' => [OpenApiConfig::TYPE => 'integer'],
198         ],
199     ],
200 ],
201 'delivery' => [
202     OpenApiConfig::TYPE => 'string',
203     OpenApiConfig::ENUM => ['pickup', 'courier'],
204 ],
205 'callbackUrl' => [OpenApiConfig::TYPE => 'string',
OpenApiConfig::FORMAT => 'uri'],
206 ],
207 ],
208 'Order' => [
209     OpenApiConfig::TYPE => 'object',
210     OpenApiConfig::REQUIRED => ['id', 'orderStatus'],
211     OpenApiConfig::PROPERTIES => [
212         'id' => [OpenApiConfig::TYPE => 'integer', 'readOnly' => true],
213         'orderStatus' => [
214             OpenApiConfig::TYPE => 'string',
215             OpenApiConfig::ENUM => ['new', 'completed', 'cancelled'],
216         ],
217     ],
218 ],
219 ],
220 ],
221 'project' => [
222     'operation' => 'order.create',
223 ],
224 ],
225 ];

```

project — пользовательское поле. **callbacks**, **email** и **uri** описывают документацию; обработку и проверки реализует контроллер.

PUT — изменение

local/php_interface/api/routes/PUT/orders.php

```

1  <?php
2
3  use Native\Api\Config\Internals\OpenApiConfig;
4  use Native\Api\Config\Internals\CacheConfig;
5  use Native\Api\Config\Internals\ControllerConfig;
6  use Native\Api\Config\Internals\ParameterConfig;
7  use Native\Api\Config\Internals\RouteConfig;
8  use Native\Api\Config\Internals\SecurityConfig;
9  use Native\Api\Config\Internals\TypeConfig;
10 use Local\Api\OrderController;
11
12 return [
13     'orders/{id}' => [
14         ControllerConfig::CONTROLLER => [OrderController::class, 'update'],
15         RouteConfig::DESCRIPTION => 'Изменение заказа',
16         RouteConfig::SECURITY => [
17             SecurityConfig::AUTH => SecurityConfig::TOKEN,
18             SecurityConfig::TOKEN_TYPE => SecurityConfig::TOKEN_ACCESS,
19         ],

```

```

20     RouteConfig::PARAMETERS => [
21         'id' => [
22             ParameterConfig::TYPE => TypeConfig::INTEGER,
23             ParameterConfig::REQUIRED => true,
24         ],
25         'orderStatus' => [
26             ParameterConfig::TYPE => TypeConfig::STRING,
27             ParameterConfig::REQUIRED => true,
28             ParameterConfig::POSSIBLE_VALUE => ['new', 'completed', 'cancelled'],
29         ],
30     ],
31     RouteConfig::CACHE => [
32         CacheConfig::TTL => 0,
33     ],
34     OpenApiConfig::TAGS => ['Заказы'],
35 ],
36 ];

```

DELETE — удаление

local/php_interface/api/routes/DELETE/orders.php

```

1  <?php
2
3  use Native\Api\Config\Internals\OpenApiConfig;
4  use Native\Api\Config\Internals\CacheConfig;
5  use Native\Api\Config\Internals\ControllerConfig;
6  use Native\Api\Config\Internals\ParameterConfig;
7  use Native\Api\Config\Internals\RouteConfig;
8  use Native\Api\Config\Internals\SecurityConfig;
9  use Native\Api\Config\Internals\TypeConfig;
10 use Local\Api\OrderController;
11
12 return [
13     'orders/{id}' => [
14         ControllerConfig::CONTROLLER => [OrderController::class, 'delete'],
15         RouteConfig::DESCRIPTION => 'Удаление заказа',
16         RouteConfig::SECURITY => [
17             SecurityConfig::AUTH => SecurityConfig::TOKEN,
18             SecurityConfig::TOKEN_TYPE => SecurityConfig::TOKEN_ACCESS,
19         ],
20         RouteConfig::PARAMETERS => [
21             'id' => [
22                 ParameterConfig::TYPE => TypeConfig::INTEGER,
23                 ParameterConfig::REQUIRED => true,
24             ],
25         ],
26         RouteConfig::CACHE => [
27             CacheConfig::TTL => 0,
28         ],
29         OpenApiConfig::TAGS => ['Заказы'],
30     ],
31 ];

```

Общий файл карты

В поле «Путь карты роутов» укажите `local/php_interface/api/routes/index.php` :

При повторении адреса внутри одного метода используется первое описание.
После обновления карты роутов требуется очистка кеша модуля.

```
1  <?php
2
3  use Bitrix\Main\Web\HttpClient;
4
5  $routeFiles = [
6      HttpClient::HTTP_GET => ['orders.php'],
7      HttpClient::HTTP_POST => ['orders.php'],
8      HttpClient::HTTP_PUT => ['orders.php'],
9      HttpClient::HTTP_DELETE => ['orders.php'],
10 ];
11
12 $routes = [];
13
14 foreach ($routeFiles as $method => $files) {
15     $routes[$method] = [];
16
17     foreach ($files as $file) {
18         $config = require __DIR__ . '/' . $method . '/' . $file;
19
20         if (!is_array($config)) {
21             continue;
22         }
23
24         $routes[$method] += $config;
25     }
26 }
27
28 return $routes;
```

Контроллеры

Класс и метод

/local/api/controllers/OrderController.php

```
1  <?php
2
3  namespace Local\Api;
4
5  final class OrderController
6  {
7      public function get(): array
8      {
9          return [
10             'order' => [
11                 'id' => 501,
12                 'orderStatus' => 'new',
13             ],
14         ];
15     }
16 }
```

Вызываемый класс

/local/api/controllers/OrderAction.php

```
1  <?php
2
3  namespace Local\Api;
4
5  final class OrderAction
6  {
7      public function __invoke(): array
8      {
9          return [
10             'order' => [
11                 'id' => 501,
12                 'orderStatus' => 'new',
13             ],
14         ];
15     }
16 }
```

PHP-файл

/local/api/controllers/order.php

```
1  <?php
2
3  return [
```

```
4     'order' => [  
5         'id' => 501,  
6         'orderStatus' => 'new',  
7     ],  
8 ];
```

Автозагрузка классов

В `/local/php_interface/init.php` зарегистрируйте namespace `Local\Api` для каталога контроллеров:

```
1 use Bitrix\Main\Loader;  
2  
3 Loader::registerNamespace(  
4     'Local\Api',  
5     $_SERVER['DOCUMENT_ROOT'] . '/local/api/controllers'  
6 );
```

События

Before-обработчик выполняется перед проверкой или действием, After — после соответствующего этапа. Условия вызова зависят от события: например, `onAfterIpAccess` вызывается только при успешной IP-проверке. After не обязательно означает «после отправки ответа»: для этого предусмотрен только `onAfterResponse`.

Порядок событий и контракт результата описаны отдельно.

Подключение

В `local/php_interface/init.php` добавьте общий блок. Нужные обработчики из следующих примеров размещайте внутри `if`, после создания `$eventManager`:

```

1  use Bitrix\Main\Event;
2  use Bitrix\Main\EventManager;
3  use Bitrix\Main\EventResult;
4  use Bitrix\Main\Loader;
5  use Native\Api\Config\Internals\BaseConfig;
6  use Native\Api\Config\Internals\ModuleEventConfig;
7
8  if (Loader::includeModule('native.api')) {
9      $eventManager = EventManager::getInstance();
10
11     // Здесь зарегистрируйте нужные обработчики из примеров ниже.
12 }
```

Примеры используют события конвейера API. В обработчике `$event` доступны параметры конкретного события и `apiContext`. Путь `orders` ниже — путь выбранного роута без префикса точки входа и крайних слешей. Замените его своим путём.

До IP-проверки роута

Добавим запрещённый адрес для `orders`, сохранив остальные ограничения роута:

```

1  $eventManager->addEventHandler(
2      BaseConfig::MODULE_ID,
3      ModuleEventConfig::BEFORE_IP_ACCESS,
4      static function (Event $event): ?EventResult {
5          $route = $event->getParameter(ModuleEventConfig::PARAMETER_ROUTE);
6          if ($route->getPath() !== 'orders') {
7              return null;
8          }
9
10         $blacklist = $event->getParameter(ModuleEventConfig::PARAMETER_IP_BLACKLIST);
11         $blacklist[] = '192.0.2.50';
```

```

12
13         return new EventResult(EventResult::SUCCESS, [
14             ModuleEventConfig::PARAMETER_IP_BLACKLIST => $blacklist,
15         ]);
16     },
17 );

```

Изменение действует на текущую проверку, не сохраняется в карту и не отменяет глобальные ограничения. Обработчик вызывается и при пустых IP-списках роута. Чтобы изменить списки, возвращайте `EventResult::SUCCESS` : обычный массив означает досрочный ответ клиенту.

После IP-проверки роута

Передадим контроллеру `orders` проверенный адрес клиента в параметре `clientId` . Входящее значение этого параметра заменяется адресом из `REMOTE_ADDR` :

```

1  $eventManager->addEventHandler(
2      BaseConfig::MODULE_ID,
3      ModuleEventConfig::AFTER_IP_ACCESS,
4      static function (Event $event): void {
5          $route = $event->getParameter(ModuleEventConfig::PARAMETER_ROUTE);
6          if ($route->getPath() !== 'orders') {
7              return;
8          }
9
10         $request = $event->getParameter(ModuleEventConfig::PARAMETER_HTTP_REQUEST);
11         $context = $event->getParameter(ModuleEventConfig::PARAMETER_API_CONTEXT);
12         $params = $context->getParams();
13         $params['clientId'] = $request->getRemoteAddress();
14         $context->setParams($params);
15     },
16 );

```

After вызывается только при успешной IP-проверке. В нём доступны фактически проверенные `ipBlacklist` и `ipWhitelist` , включая изменения Before-обработчика. Оба события выполняются до авторизации и чтения кеша ответа. Добавленный `clientId` входит в параметры запроса, а значит, учитывается в ключе кеша результата.

До проверки параметров

Уберём пробелы по краям `article` перед штатной валидацией:

```

1  $eventManager->addEventHandler(
2      BaseConfig::MODULE_ID,
3      ModuleEventConfig::BEFORE_PARAMETERS,

```

```

4      static function (Event $event): void {
5          $context = $event->getParameter(ModuleEventConfig::PARAMETER_API_CONTEXT);
6          $params = $context->getParams();
7
8          if (isset($params['article']) && is_string($params['article'])) {
9              $params['article'] = trim($params['article']);
10             $context->setParams($params);
11         }
12     },
13 );

```

Возвращать данные не нужно: параметры изменены через общий контекст запроса.

После контроллера

Удалим служебное поле `internalComment` из результата контроллера `orders`, прежде чем модуль сохранит результат в кеш и сформирует JSON/XML:

```

1      $eventManager->addEventHandler(
2          BaseConfig::MODULE_ID,
3          ModuleEventConfig::AFTER_CONTROLLER,
4          static function (Event $event): ?EventResult {
5              $route = $event->getParameter(ModuleEventConfig::PARAMETER_ROUTE);
6              if ($route->getPath() !== 'orders') {
7                  return null;
8              }
9
10             $response = $event->getParameter(ModuleEventConfig::PARAMETER_RESPONSE);
11             unset($response['internalComment']);
12
13             return new EventResult(EventResult::SUCCESS, [
14                 ModuleEventConfig::PARAMETER_RESPONSE => $response,
15             ]);
16         },
17 );

```

При попадании в кеш контроллер и его события не выполняются. После добавления такого обработчика очистите ранее сохранённый кеш роута, чтобы в нём не осталось старого поля.

До отправки ответа

Добавим заголовок к уже сформированному HTTP-ответу:

```

1      $eventManager->addEventHandler(
2          BaseConfig::MODULE_ID,
3          ModuleEventConfig::BEFORE_RESPONSE,
4          static function (Event $event): void {
5              $response = $event->getParameter(ModuleEventConfig::PARAMETER_RESPONSE);
6              $response->addHeader('X-API-Service', 'native.api');

```

```
7     },  
8     );
```

Здесь `response` — объект `Bitrix\Main\HttpResponse`, а в `onAfterController` — массив данных контроллера. При досрочном ответе из более раннего события `onBeforeResponse` не вызывается.

Не используйте `echo`, `die` или `exit` в обработчиках для формирования ответа. Возврат обычного массива или `HttpResponse` завершает конвейер штатно; `null` и `return;` продолжают выполнение. Сигнатура `: void` подходит только обработчикам без возвращаемого значения.

cURL

Замените `https://example.ru/api/` адресом своего API. Нужные наборы роутов должны быть включены.

Сервисные роуты

Параметры адреса

```
1 # Проверить работу API
2 curl 'https://example.ru/api/up'
3
4 # Передать id в адресе и name в строке запроса
5 curl 'https://example.ru/api/up/user/15?name=Ivan'
```

Передача JSON

```
1 # Передать параметры в теле запроса
2 curl 'https://example.ru/api/up' \
3     -H 'Content-Type: application/json' \
4     --data '{"name": "Иван", "quantity": 2}'
```

XML-ответ

```
1 # Отправить JSON и получить XML-ответ встроенного PUT-роута
2 curl -X PUT 'https://example.ru/api/up' \
3     -H 'Content-Type: application/json' \
4     --data '{"name": "Иван"}'
```

```
1 # Выбрать XML для GET-запроса
2 curl 'https://example.ru/api/up?format=xml'
```

Доступ по группам

```
1 # HTTP 403: в штатной карте задан whitelist без авторизации API
2 curl 'https://example.ru/api/up/group/1'
```

Вызываемый класс

```
1 # Проверить вызываемый класс; данные не удаляются
2 curl -X DELETE 'https://example.ru/api/up'
```

Токены

При `AUTH_MFA_REQUIRED` для выпуска и списка токенов через Basic нужны cookie сессии того же пользователя после входа с MFA. Эти примеры cookie не передают. Для серверной интеграции можно выпустить токен в админке и использовать Bearer.

Вместо `login` укажите логин пользователя — cURL запросит пароль. Подставьте свои `<ACCESS_TOKEN>`, `<REFRESH_TOKEN>` и `<CLIENT_ID>`.

Для приведённых ниже примеров используйте пару, выпущенную с `User-Agent: MyIntegration/1.0`, либо ручную пару без привязки. При переносе токена из браузера клиент будет другим — проверка привязки вернёт HTTP 401.

Выпуск

```
1 # Создать пару токенов для постоянного серверного клиента
2 curl -X POST 'https://example.ru/api/token' \
3     --user 'login' \
4     --user-agent 'MyIntegration/1.0'
```

Обновление

```
1 # Заменить пару по refresh-токену
2 curl -X PUT 'https://example.ru/api/token' \
3     --user-agent 'MyIntegration/1.0' \
4     -H 'Authorization: Bearer <REFRESH_TOKEN>'
```

Список

```
1 # Получить активные токен-сессии пользователя
2 curl 'https://example.ru/api/token' \
3     --user 'login' \
4     --user-agent 'MyIntegration/1.0'
```

Отзыв

```
1 # client.id берётся из списка сессий
2 curl -X DELETE 'https://example.ru/api/token?client_id=<CLIENT_ID>' \
3     --user-agent 'MyIntegration/1.0' \
4     -H 'Authorization: Bearer <ACCESS_TOKEN>'
```

После выпуска или обновления сохраните оба токена из ответа: access и refresh. Поля ответа и правила работы с токенами.

Провайдер

Описание метода

```
1 # Узнать параметры разрешённого метода
2 curl -G 'https://example.ru/api/provider' \
3     --user-agent 'MyIntegration/1.0' \
4     -H 'Authorization: Bearer <ACCESS_TOKEN>' \
5     --data-urlencode 'class=Bitrix\Main\Localization\Loc' \
6     --data-urlencode 'method=getCurrentLang'
```

Выполнение метода

```
1 # Получить текущий язык Битрикса
2 curl 'https://example.ru/api/provider' \
3     --user-agent 'MyIntegration/1.0' \
4     -H 'Authorization: Bearer <ACCESS_TOKEN>' \
5     -H 'Content-Type: application/json' \
6     --data '{
7         "class": "Bitrix\\Main\\Localization\\Loc",
8         "method": "getCurrentLang"
9     }'
```

JavaScript

Замените `https://example.ru/api/` адресом своего API. Нужные наборы роутов должны быть включены.

Отправка запроса

Примеры для браузера. Функция возвращает тело ответа, включая ответы с HTTP-ошибкой

```
1  const apiUrl = 'https://example.ru/api/'
2
3  async function apiRequest(path, {method = 'GET', authorization, body} = {}) {
4    const headers = new Headers()
5
6    if (authorization) headers.set('Authorization', authorization)
7    if (body !== undefined) headers.set('Content-Type', 'application/json')
8
9    const response = await fetch(new URL(path, apiUrl), {
10      method,
11      headers,
12      credentials: 'same-origin',
13      body: body === undefined ? undefined : JSON.stringify(body),
14    })
15
16    return response.text()
17  }
```

Сервисные роуты

Параметры адреса

```
1  // Проверить работу API
2  console.log(await apiRequest('up'))
3
4  // Передать id в адресе и name в строке запроса
5  console.log(await apiRequest('up/user/15?name=Ivan'))
```

Передача JSON

```
1  // Передать параметры в теле запроса
2  console.log(await apiRequest('up', {
3    method: 'POST',
4    body: {name: 'Иван', quantity: 2},
5  }))
```

XML-ответ

```
1 // Отправить JSON и получить XML-ответ встроенного PUT-роута
2 console.log(await apiRequest('up', {
3   method: 'PUT',
4   body: {name: 'Иван'},
5 })))
```

```
1 // Выбрать XML для GET-запроса
2 console.log(await apiRequest('up?format=xml'))
```

Доступ по группам

```
1 // HTTP 403: в штатной карте задан whitelist без авторизации API
2 console.log(await apiRequest('up/group/1'))
```

Вызываемый класс

```
1 // Проверить вызываемый класс; данные не удаляются
2 console.log(await apiRequest('up', {method: 'DELETE'}))
```

Токены

При **AUTH_MFA_REQUIRED** для выпуска и списка токенов через Basic нужны cookie сессии того же пользователя после входа с MFA. При запросе к тому же сайту cookie передаются автоматически — сначала завершите вход с MFA.

Подставьте логин, пароль, **<ACCESS_TOKEN>**, **<REFRESH_TOKEN>** и **<CLIENT_ID>**. Клиента определяют заголовки браузера. Выпускайте и используйте пару в одном клиенте: перенос привязанного токена из другого браузера или сURL приведёт к HTTP 401. Изменение заголовков после обновления браузера или смены языка также может потребовать повторного выпуска пары.

```
1 const login = 'login'
2 const password = '<PASSWORD>'
3 const accessToken = '<ACCESS_TOKEN>'
4 const refreshToken = '<REFRESH_TOKEN>'
5 const clientId = '<CLIENT_ID>'
6 const bytes = new TextEncoder().encode(`${login}:${password}`)
7 const credentials = btoa(Array.from(bytes, byte => String.fromCharCode(byte)).join(''))
8 const basicAuthorization = `Basic ${credentials}`
```

Выпуск

```

1 // Создать пару токенов
2 console.log(await apiRequest('token', {
3   method: 'POST',
4   authorization: basicAuthorization,
5 })))

```

Обновление

```

1 // Заменить пару по refresh-токену
2 console.log(await apiRequest('token', {
3   method: 'PUT',
4   authorization: `Bearer ${refreshToken}`,
5 })))

```

Список

```

1 // Получить активные токен-сессии пользователя
2 console.log(await apiRequest('token', {authorization: basicAuthorization}))

```

Отзыв

```

1 // client.id берётся из списка сессий
2 console.log(await apiRequest(`token?client_id=${encodeURIComponent(clientId)}`, {
3   method: 'DELETE',
4   authorization: `Bearer ${accessToken}`,
5 })))

```

После выпуска или обновления сохраните оба токена из ответа: access и refresh. Поля ответа и правила работы с токенами.

Провайдер

Описание метода

```

1 // Узнать параметры разрешённого метода
2 const query = new URLSearchParams({
3   class: 'Bitrix\\Main\\Localization\\Loc',
4   method: 'getCurrentLang',
5 })
6
7 console.log(await apiRequest(`provider?${query}`, {

```

```
8     authorization: `Bearer ${accessToken}`,  
9   })))
```

Выполнение метода

```
1  // Получить текущий язык Битрикса  
2  console.log(await apiRequest('provider', {  
3    method: 'POST',  
4    authorization: `Bearer ${accessToken}`,  
5    body: {  
6      class: 'Bitrix\\Main\\Localization\\Loc',  
7      method: 'getCurrentLang',  
8    },  
9  })))
```

Для другого домена разрешите Origin, **Authorization** и **Content-Type** в CORS.

PHP

Замените `https://example.ru/api/` адресом своего API. Нужные наборы роутов должны быть включены.

Отправка запроса

Примеры для серверного PHP с расширением cURL. Функция возвращает тело ответа, включая ответы с HTTP-ошибкой

```

1  <?php
2
3  const API_URL = 'https://example.ru/api/';
4
5  function apiRequest(
6      string $path,
7      string $method = 'GET',
8      ?string $authorization = null,
9      ?array $body = null,
10 ): string {
11     $headers = [];
12     $options = [
13         CURLOPT_RETURNTRANSFER => true,
14         CURLOPT_CUSTOMREQUEST => $method,
15         CURLOPT_USERAGENT => 'MyIntegration/1.0',
16         CURLOPT_CONNECTTIMEOUT => 10,
17         CURLOPT_TIMEOUT => 30,
18     ];
19
20     if ($authorization !== null) {
21         $headers[] = 'Authorization: ' . $authorization;
22     }
23
24     if ($body !== null) {
25         $headers[] = 'Content-Type: application/json';
26         $options[CURLOPT_POSTFIELDS] = json_encode($body, JSON_THROW_ON_ERROR);
27     }
28
29     $options[CURLOPT_HTTPHEADER] = $headers;
30     $curl = curl_init(rtrim(API_URL, '/') . '/' . ltrim($path, '/'));
31
32     if ($curl === false) {
33         throw new RuntimeException('Не удалось создать HTTP-запрос');
34     }
35
36     curl_setopt_array($curl, $options);
37     $response = curl_exec($curl);
38     $error = curl_error($curl);
39     curl_close($curl);
40
41     if ($response === false) {
42         throw new RuntimeException($error);
43     }
44
45     return $response;
46 }

```

Сервисные роуты

Параметры адреса

```
1 // Проверить работу API
2 echo apiRequest('up');
3
4 // Передать id в адресе и name в строке запроса
5 echo apiRequest('up/user/15?name=Ivan');
```

Передача JSON

```
1 // Передать параметры в теле запроса
2 echo apiRequest('up', method: 'POST', body: [
3     'name' => 'Иван',
4     'quantity' => 2,
5 ]);
```

XML-ответ

```
1 // Отправить JSON и получить XML-ответ встроенного PUT-роута
2 echo apiRequest('up', method: 'PUT', body: ['name' => 'Иван']);
```

```
1 // Выбрать XML для GET-запроса
2 echo apiRequest('up?format=xml');
```

Доступ по группам

```
1 // HTTP 403: в штатной карте задан whitelist без авторизации API
2 echo apiRequest('up/group/1');
```

Вызываемый класс

```
1 // Проверить вызываемый класс; данные не удаляются
2 echo apiRequest('up', method: 'DELETE');
```

Токены

При `AUTH_MFA_REQUIRED` для выпуска и списка токенов через Basic нужны cookie сессии того же пользователя после входа с MFA. Эти примеры cookie не передают. Для серверной интеграции можно выпустить токен в админке и использовать Bearer.

Подставьте логин, пароль, `<ACCESS_TOKEN>`, `<REFRESH_TOKEN>` и `<CLIENT_ID>`. В функции задан постоянный `User-Agent: MyIntegration/1.0` для выпуска, обновления и использования токенов. Используйте пару, выпущенную этим клиентом, либо пару, созданную вручную: привязанный токен другого клиента вернёт HTTP 401.

```
1 $login = 'login';
2 $password = '<PASSWORD>';
3 $accessToken = '<ACCESS_TOKEN>';
4 $refreshToken = '<REFRESH_TOKEN>';
5 $clientId = '<CLIENT_ID>';
6 $basicAuthorization = 'Basic ' . base64_encode($login . ':' . $password);
```

Выпуск

```
1 // Создать пару токенов для постоянного серверного клиента
2 echo apiRequest('token', method: 'POST', authorization: $basicAuthorization);
```

Обновление

```
1 // Заменить пару по refresh-токену
2 echo apiRequest('token', method: 'PUT', authorization: 'Bearer ' . $refreshToken);
```

Список

```
1 // Получить активные токен-сессии пользователя
2 echo apiRequest('token', authorization: $basicAuthorization);
```

Отзыв

```
1 // client.id берётся из списка сессий
2 echo apiRequest(
3     'token?client_id=' . rawurlencode($clientId),
4     method: 'DELETE',
```

```

5     authorization: 'Bearer ' . $accessToken,
6 );

```

После выпуска или обновления сохраните оба токена из ответа: access и refresh. Поля ответа и правила работы с токенами.

Провайдер

Описание метода

```

1 // Узнать параметры разрешённого метода
2 $query = http_build_query([
3     'class' => 'Bitrix\\Main\\Localization\\Loc',
4     'method' => 'getCurrentLang',
5 ], '', '&', PHP_QUERY_RFC3986);
6
7 echo apiRequest('provider?' . $query, authorization: 'Bearer ' . $accessToken);

```

Выполнение метода

```

1 // Получить текущий язык Битрикса
2 echo apiRequest('provider', method: 'POST', authorization: 'Bearer ' . $accessToken, body: [
3     'class' => 'Bitrix\\Main\\Localization\\Loc',
4     'method' => 'getCurrentLang',
5 ]);

```

JSON

Параметр	Значение
HTTP-статус	200 OK
Content-Type	application/json
Формат ответа	json

Технические поля

В настройках ответа: `technical_keys_disabled => false`

```
1  {
2    "request_id": "0af1b6c2d3e4f5a6b7c8d9e0f1a2b3c4",
3    "status": {
4      "code": 200,
5      "title": "200 OK"
6    },
7    "result": {
8      "order": {
9        "id": 501,
10       "orderStatus": "new"
11      }
12    },
13    "time": {
14      "start": 1790067600.125,
15      "finish": 1790067600.1875,
16      "duration": 0.0625,
17      "date_start": "22.09.2026 12:00:00",
18      "date_finish": "22.09.2026 12:00:00",
19      "timestamp": {
20        "date_start": 1790067600,
21        "date_finish": 1790067600
22      }
23    }
24  }
```

Без технических полей

`technical_keys_disabled => true`

```
1  {
2    "order": {
3      "id": 501,
4      "orderStatus": "new"
5    }
6  }
```

XML

Параметр	Значение
HTTP-статус	200 OK
Content-Type	text/xml
Формат ответа	xml

Технические поля

В настройках ответа: `technical_keys_disabled => false`

```
1  <response>
2    <request_id>0af1b6c2d3e4f5a6b7c8d9e0f1a2b3c4</request_id>
3    <status>
4      <code>200</code>
5      <title>200 OK</title>
6    </status>
7    <result>
8      <order>
9        <id>501</id>
10       <orderStatus>new</orderStatus>
11     </order>
12   </result>
13   <time>
14     <start>1790067600.125</start>
15     <finish>1790067600.1875</finish>
16     <duration>0.0625</duration>
17     <date_start>22.09.2026 12:00:00</date_start>
18     <date_finish>22.09.2026 12:00:00</date_finish>
19     <timestamp>
20       <date_start>1790067600</date_start>
21       <date_finish>1790067600</date_finish>
22     </timestamp>
23   </time>
24 </response>
```

Без технических полей

`technical_keys_disabled => true`

```
1  <order>
2    <id>501</id>
3    <orderStatus>new</orderStatus>
4  </order>
```

Используемые ресурсы

Источник документации

<https://webco.one/docs/bitrix/modules/native.api/>

Дополнительные материалы

<https://www.1c-bitrix.ru/products/cms/requirements.php>

https://dev.1c-bitrix.ru/learning/course/index.php?COURSE_ID=135&LESSON_ID=2593

<https://marketplace.1c-bitrix.ru/solutions/native.api/>

<https://marketplace.1c-bitrix.ru/about/help.php>