Искра Мессенджер – корпоративный мессенджер с ИИ версия 1.0

Руководство разработчика

Содержание

Данное руководство предназначено для разработчиков, которые подключают собственных ИИ-агентов к Искра Мессенджер. Документ объясняет модель агента в Искре, порядок подготовки сервиса агента и создания агента администратором пространства, подключение инструментов и MCP-серверов, API сервиса агента, обработку событий, отправку ответов, потоковые ответы, OAuth-авторизацию пользователя в чате, запросы инструментов и передачу обращения оператору. Цель руководства - дать разработчику достаточный контракт, чтобы реализовать внешний сервис агента без доступа к внутреннему коду Искры.

Термины контракта

Термин	Значение в этом документе
ИИ-агент	Виртуальный участник Искры, который отвечает пользователям через настроенный сервис агента.
Виртуальный пользователь	Учетная запись агента в Gateway Искры. Именно она участвует в диалогах и владеет bot credential.
Сервис агента	Серверный процесс разработчика или встроенный сервис `assistant`, который обрабатывает события агента и отправляет ответы в Искру.
API сервиса агента	Набор точек доступа `/v1/bot-runtime/...`, через которые сервис агента получает lease, читает контекст и отправляет ответы.
Учетные данные агента (credential)	Пара `credential_id.secret`, которая используется сервисом агента как токен доступа агента.
Аренда обработки (lease)	Краткоживущее право одного активного сервиса агента обрабатывать события одного виртуального пользователя. Далее используется точное имя поля API `lease`.
Потоковый ответ (stream)	Ответ агента, который пользователь видит по мере формирования до создания итогового сообщения.
Запрос инструмента (tool request)	Запрос агента на действие, которое требует подтверждения человека или внешнего обработчика.
MCP-сервер	Отдельный сервер, который публикует инструменты по Model Context Protocol. Искра хранит описание подключения и назначение серверов агентам; фактическое выполнение зависит от сервиса агента.
Передача оператору (handoff)	Передача обращения клиента оператору

	поддержки.
Карточка авторизации (auth-card)	Структурированная часть сообщения `tool_card` с кнопкой OAuth-авторизации.

Версия и совместимость API

Документ описывает контракт Искра Мессенджер 1.0 для API сервиса агента версии `/v1`. Внешний сервис агента должен соблюдать правила совместимости:

- игнорировать неизвестные поля в JSON-ответах;
- логировать и безопасно пропускать неизвестные `event.type`;
- не полагаться на порядок полей JSON;
- считать новые значения enum неподдержанными, но не аварийными;
- использовать только документированные в этом руководстве точки доступа API сервиса агента;
- хранить `event_id` последнего успешно обработанного события и быть готовым к повторной доставке.

Нормативный приоритет материалов:

1. Файл `docs/api/iskra-bot-runtime-openapi.yaml` задает машинно-читаемые пути, методы, заголовки и схемы HTTP API.
2. Настоящее руководство задает правила жизненного цикла, восстановления, безопасности и эксплуатации сервиса агента.
3. Примеры из `docs/api/conformance/` используются для проверки сериализации, но не расширяют контракт.

Совместимые добавления могут вводить новые необязательные поля и новые типы событий в пределах `/v1`. Изменение смысла существующего поля, удаление поля или новое обязательное поле требует новой версии пути API. При расхождении примера и нормативной схемы разработчик должен руководствоваться схемой OpenAPI и зарегистрировать расхождение у поставщика.

1. Что такое ИИ-агент в Искре

ИИ-агент в Искре - это виртуальный участник диалогов, который действует от имени настроенного агентского профиля. Для пользователей он выглядит как отдельный участник или помощник в пространстве: у него есть имя, описание, видимость, права доступа и понятная роль в коммуникации.

Агент не является обычным пользовательским клиентом. Он не входит в систему через телефон или SSO, не хранит пользовательскую сессию и не использует клиентские API мобильного или web-приложения. Для дополнительного типа команда заказчика сначала развертывает и регистрирует сервис агента, а затем администратор пространства создает агентский профиль.

Типовые сценарии:

- первая линия поддержки и квалификация обращений;
- корпоративный помощник по базе знаний;
- ассистент в проектном или сервисном пространстве;
- агент, который собирает данные, готовит черновик ответа и передает сложный случай оператору;
- агент, который создает запрос на действие через tool request, но ждет подтверждения человека перед выполнением.

Важное ограничение: агент должен отвечать только в рамках прав, выданных ему в Искре. Настройку видимости, подключение к очереди и другие действия, требующие полномочий администратора пространства, выполняет администратор в интерфейсе Искры. Учетные данные агента Gateway выпускает автоматически при создании агента.

2. Как устроен агент в Искре

Экземпляр агента в Искре образуют три продуктовые сущности:

Сущность	**Где находится**	**Назначение**
Виртуальный пользователь	Gateway Искры	Учетная запись агента в разговорах. Gateway хранит имя, видимость, участие, права, состояние настройки и учетные данные агента.
Тип контроллера	Реестр `ai_agent_controllers` в Gateway	Связывает `controller_key` с внутренним `service_base_url` сервиса агента и объявляет поддерживаемые настройки через `capabilities_json`. Один тип может обслуживать несколько виртуальных пользователей.
Сервис агента	Встроенный сервис Искры или серверный сервис заказчика	Получает конфигурацию конкретного виртуального пользователя, обрабатывает сообщения и отправляет ответы через API сервиса агента. Вызов языковой модели, встроенных обработчиков и MCP-клиента зависит от реализации выбранного сервиса.

Искра использует два независимых направления взаимодействия:

Направление	Кто вызывает	Точка доступа	Что передается
Подготовка и обновление агента	Gateway вызывает сервис агента по его `service_base_url`	`PUT /v1/agents/<virtual_user_id>`	Полный актуальный снимок настроек агента, адрес Gateway, `bot_token` и `credential_id`.
Работа агента в разговорах	Сервис агента вызывает Gateway	`/v1/bot-runtime/...` и `/v1/bot-runtime/ws`	Lease, события, последние доступные сообщения, memo, вложения, потоковые ответы, запросы инструментов и handoff.

Точка подготовки не является API администратора пространства. Администратор сохраняет карточку агента в интерфейсе Искры, после чего Gateway сам формирует и отправляет запрос `PUT`. При последующем изменении карточки Gateway снова передает полный снимок конфигурации; сервис агента должен заменить сохраненные настройки этим снимком, включая удаление снятых назначений инструментов и MCP-серверов.

Для `basic_assistant` сервисом агента является встроенный `assistant`; для `event_organizer` используется встроенный сервис организатора мероприятий. Эти сервисы развертываются вместе с Искрой. Пользовательский тип работает только после того, как on-premise команда заказчика развернула его сервис и зарегистрировала соответствующую запись `ai_agent_controllers`. Администратор пространства не регистрирует тип контроллера и не запускает сервис агента.

Для пользовательского типа последовательность выглядит следующим образом:

1. On-premise команда запускает сервис агента и проверяет его готовность принимать `PUT /v1/agents/<virtual_user_id>`.
2. Команда регистрирует `controller_key`, `service_base_url`, `capabilities_json` и `default_tools_json` в Gateway. Только после этого тип появляется в списке интерфейса.
3. Администратор пространства выбирает зарегистрированный тип и сохраняет карточку нового агента.
4. Gateway создает виртуального пользователя, выпускает `bot_token` и только затем вызывает точку подготовки сервиса агента.
5. Gateway передает `virtual_user_id`, `bot_api_base_url`, `bot_token`, `credential_id` и полный снимок настроек. Сервис атомарно сохраняет конфигурацию и секрет либо отклоняет запрос без частичного применения.
6. При успешном ответе Gateway переводит агента в активное состояние. Если соединение не установлено, сервис вернул HTTP-ошибку или `accepted=false`, виртуальный пользователь не удаляется автоматически: карточка получает состояние **Ошибка настройки**, а `controller_status` - `provisioning_failed`.
7. Подготовленный сервис задает идентификатор своей сессии, получает lease и открывает WebSocket с теми же идентификаторами сессии и lease.

8. На событие сообщения сервис читает последние доступные сообщения и memo, выполняет свою бизнес-логику или вызов модели и отправляет ответ. После разрыва WebSocket сервис сверяет состояние через HTTP API, поскольку доставка событий и доступная история имеют ограничения, описанные в разделах 14 и 18.

Запрос подготовки содержит действующий `bot_token`, поэтому его транспорт относится к контуру секретов. Gateway 1.0 добавляет подписанный заголовок `Authorization` для встроенного `event_organizer`, но не добавляет универсальный заголовок авторизации для произвольного пользовательского типа. В on-premise внедрении доступ к `service_base_url` пользовательского типа должен быть разрешен только Gateway посредством сетевых правил или доверенного внутреннего прокси; сервис не должен публиковать эту точку в пользовательской сети или Интернете.

Поддерживаемые функции определяются одновременно записью контроллера и реализацией сервиса. Например, `basic_assistant` объявляет возможность назначения MCP-серверов, но в версии 1.0 не выполняет произвольные MCP-вызовы; точный статус приведен в разделе 9.1. Handoff применяется только для агентов клиентской поддержки. Для виртуального пользователя в групповом чате передача конкретному сотруднику в версии 1.0 не поддерживается.

3. Границы ответственности

Gateway Искры владеет:

- учетными записями людей и виртуальных пользователей;
- участием в диалогах и правами доступа;
- видимостью агента в пространстве;
- credential агента;
- lease одного активного потребителя;
- историей диалога;
- потоковыми ответами;
- жизненным циклом tool request;
- доставкой событий в реальном времени.

Сервис агента владеет:

- системной инструкцией и ролью агента;
- поставщиком LLM;
- инструментами и MCP-серверами;
- режимом реакции на сообщения;
- бизнес-логикой ответа;
- хранением собственных внешних секретов;
- повторами и backoff своего worker-процесса.

Не храните состояние диалога только в памяти сервиса агента. После переподключения агент должен восстанавливать разговоры, последние сообщения и memo из API сервиса агента и сверять

их со своим checkpoint WebSocket. API сервиса агента 1.0 не предоставляет чтение состояния stream, поэтому незавершенный stream после неопределенного сетевого результата нельзя считать восстановимым.

4. Создание агента администратором пространства

Администратор пространства создает агента только после того, как его сервис готов к приему конфигурации. Для встроенных типов `basic_assistant` и `event_organizer` соответствующие сервисы развертываются в составе Искры. Для дополнительного типа on-premise команда заказчика до этого выполняет процедуру из раздела 8:

1. развертывает и запускает сервис агента;
2. открывает Gateway сетевой доступ к `PUT /v1/agents/<virtual_user_id>`;
3. регистрирует тип контроллера в `ai_agent_controllers`;
4. проверяет, что тип агента появился в списке интерфейса Искры.

После этого администратор пространства выполняет штатную операцию создания:

1. Открыть Искру под учетной записью администратора пространства.
2. Перейти в нужное пространство.
3. Открыть **Настройки пространства**.
4. Открыть раздел **ИИ Агенты**.
5. Нажать **Создать агента**.
6. Выбрать тип ИИ-агента.
7. Указать название агента.
8. Выбрать видимость: `private` - только для явно разрешенных пользователей или сценариев; `space_public` - для пользователей пространства; `global_public` - для более широй видимости, если это разрешено политикой внедрения.
9. Выбрать режим реакции на сообщения и класс языковой модели, если выбранный тип агента поддерживает эти параметры.
10. Заполнить поле **Инструкция** и, при необходимости, приветственное сообщение.
11. Выбрать навыки, инструменты и MCP-серверы, которые должны быть доступны агенту. Интерфейс показывает только параметры, которые поддерживает выбранный тип.
12. Для агента клиентской поддержки при необходимости включить передачу оператору и указать сообщение для пользователя. Для агента группового чата эту функцию не включать.
13. Проверить положение переключателя **Включен** и нажать **Сохранить**.
14. Вернуться в список агентов и убедиться, что агент имеет статус **Включен**, а в карточке нет статуса **Ошибка настройки**.

При сохранении Gateway автоматически создает виртуального пользователя и его учетные данные, а затем вызывает точку подготовки выбранного сервиса агента. Администратор

пространства не создает учетные данные отдельно и не передает их разработчику: `bot_token`, `credential_id`, `virtual_user_id` и базовый URL API передаются в сервис агента в запросе `PUT /v1/agents/<virtual_user_id>`.

Если агент должен работать только в определенной очереди поддержки, администратор добавляет агента в эту очередь или назначает его в правилах обработки обращений. Сервис агента не должен самостоятельно расширять область доступа агента через API.

5. Базовые URL и авторизация

В примерах используются:

```
GATEWAY_BASE_URL=https://im.iskra-messenger.ru
ASSISTANT_BASE_URL=https://assistant.example.internal
```

Management endpoints выполняются от имени авторизованного пользователя/администратора:

```
Authorization: Bearer <user_access_token>
Content-Type: application/json
```

Точки доступа API сервиса агента выполняются от имени virtual user:

```
Authorization: Bearer <credential_id>.<secret>
X-Bot-Session-ID: <agent_service_session_id>
X-Bot-Lease-ID: <lease_id>
Content-Type: application/json
```

Для websocket также поддерживаются query-параметры:

```
token=<credential_id.secret>&session_id=<agent_service_session_id>&lease_id=<lease_id>&last_event_id=<event_id>
```

6. Минимальный сценарий подключения

Шаг 1. Развертывание и регистрация. On-premise команда заказчика запускает сервис агента и регистрирует его тип контроллера по процедуре из раздела 8.

Шаг 2. Создание агента. Администратор пространства выбирает зарегистрированный тип и создает агента по процедуре из раздела 4.

Шаг 3. Автоматическая передача конфигурации. Gateway вызывает `PUT /v1/agents/<virtual_user_id>` и передает сервису агента `bot_api_base_url`, `virtual_user_id`, `credential_id`, `bot_token` и настройки агента.

Шаг 4. Хранение секрета. Сервис агента сохраняет `bot_token` только в серверном хранилище секретов и возвращает успешный статус подготовки.

Шаг 5. Получение lease. Сервис агента получает lease:

```
POST /v1/bot-runtime/leases
Authorization: Bearer <credential_id>.<secret>
X-Bot-Session-ID: agent-worker-1
Content-Type: application/json
{
  "lease_id": "agent-worker-1-lease"
}
```

Шаг 6. Подключение к событиям. Сервис агента подключается к событиям:

```
wss://im.iskra-messenger.ru/v1/bot-runtime/ws?
token=<credential_id.secret>&session_id=agent-worker-1&lease_id=agent-worker-1-lease
```

Шаг 7. Обработка сообщения. На событие `message.created` сервис агента читает последние сообщения и memo.

Во всех запросах чтения на этом шаге сервис агента передает `X-Bot-Lease-ID`. Без действующего lease Gateway отвечает `409 bot_lease_unavailable`.

Шаг 8. Генерация ответа. Сервис агента вызывает LLM или свою бизнес-логику.

Шаг 9. Отправка ответа. Сервис агента отправляет ответ через обычное сообщение или точки доступа потоковых ответов.

Шаг 10. Передача оператору. Для агента клиентской поддержки: если вопрос требует человека, сервис агента обновляет memo и вызывает handoff. Для виртуального пользователя в групповом чате этот шаг не применяется.

7. Простейший пример ИИ-агента на Node.js

Ниже минимальный сервис агента, который получает lease, продлевает lease, последовательно обрабатывает события WebSocket, берет последние сообщения, отправляет текст пользователя в совместимую с OpenAI точку доступа LLM и пишет ответ в Искру обычным сообщением. Пример демонстрирует рабочий протокол, но не является промышленной реализацией. Для промышленного сервиса агента обязательны устойчивое хранение `last_event_id`, повторное подключение с увеличивающейся задержкой и случайным разбросом, сверка контекста через REST после каждого разрыва, метрики, ограничение параллелизма, фильтрация вложений и политика передачи оператору.

Установка зависимостей:

```
npm install ws openai
```

Переменные окружения:

```
export BOT_API_BASE_URL="https://im.iskra-messenger.ru"
export BOT_TOKEN="<credential_id>.<secret>"
export VIRTUAL_USER_ID="<virtual_user_id>"
export OPENAI_API_KEY="<llm_api_key>"
export OPENAI_BASE_URL="https://api.openai.com/v1"
export OPENAI_MODEL="gpt-4.1-mini"
```

Пример:

```
import WebSocket from "ws";
import OpenAI from "openai";
import { randomUUID } from "node:crypto";

const botApiBaseUrl = process.env.BOT_API_BASE_URL;
const botToken = process.env.BOT_TOKEN;
const virtualUserId = process.env.VIRTUAL_USER_ID;
const sessionId = "simple-agent-" + Date.now();
let activeLeaseId = "";
```

```
let lastEventId = "";

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY,
  baseURL: process.env.OPENAI_BASE_URL,
});

async function botFetch(path, options = {}) {
  const response = await fetch(botApiBaseUrl + path, {
    ...options,
    headers: {
      Authorization: "Bearer " + botToken,
      "X-Bot-Session-ID": sessionId,
      ...(activeLeaseId ? { "X-Bot-Lease-ID": activeLeaseId } : {}),
      "Content-Type": "application/json",
      ...(options.headers || {}),
    },
  });
  if (!response.ok) {
    throw new Error((options.method || "GET") + " " + path + " failed: " +
response.status);
  }
  return response.headers.get("content-type")?.includes("application/json")
    ? response.json()
    : undefined;
}

async function acquireLease() {
  return botFetch("/v1/bot-runtime/leases", {
    method: "POST",
    body: JSON.stringify({ lease_id: sessionId + "-lease" }),
  });
}

async function renewLease(leaseId) {
  await botFetch("/v1/bot-runtime/leases/" + encodeURIComponent(leaseId), {
    method: "PATCH",
  });
}

async function answer(conversationId, eventId) {
  const messages = await botFetch(
    "/v1/bot-runtime/conversations/" + conversationId + "/messages?limit=10"
  );
  const userText = [...messages.messages]
```

```
    .reverse()
    .find((message) => message.sender_id !== virtualUserId)?.body;

  if (!userText) return;

  const completion = await openai.chat.completions.create({
    model: process.env.OPENAI_MODEL || "gpt-4.1-mini",
    messages: [
      {
        role: "system",
        content:
          "Ты корпоративный ИИ-агент Искры. Отвечай кратко, уточняй детали и не
выполняй действия без подтверждения человека.",
      },
      { role: "user", content: userText },
    ],
  });

  await botFetch("/v1/bot-runtime/conversations/" + conversationId + "/messages", {
    method: "POST",
    body: JSON.stringify({
      body: completion.choices[0]?.message?.content || "Не удалось подготовить
ответ.",
      client_request_id: eventId ? "reply-" + eventId : randomUUID(),
    }),
  });
}

const lease = await acquireLease();
activeLeaseId = lease.lease_id;
const renewTimer = setInterval(() => {
  renewLease(lease.lease_id).catch((error) => {
    console.error("lease renewal failed", error);
    process.exit(1);
  });
}, 10_000);

process.on("SIGTERM", async () => {
  clearInterval(renewTimer);
  await botFetch("/v1/bot-runtime/leases/" + encodeURIComponent(lease.lease_id), {
    method: "DELETE",
  }).catch(() => {});
  process.exit(0);
});
```

```
const wsUrl = new URL("/v1/bot-runtime/ws", botApiBaseUrl.replace(/^http/, "ws"));
wsUrl.searchParams.set("token", botToken);
wsUrl.searchParams.set("session_id", sessionId);
wsUrl.searchParams.set("lease_id", lease.lease_id);

const ws = new WebSocket(wsUrl);
let processing = Promise.resolve();
ws.on("message", (raw) => {
  processing = processing.then(async () => {
    const event = JSON.parse(raw.toString());
    if (event.type === "message.created" &&
        !event.payload?.sender_is_virtual &&
        event.payload?.sender_id !== virtualUserId) {
      await answer(event.conversation_id, event.event_id);
    }
    // В промышленной реализации записать checkpoint в устойчивое хранилище.
    // Запись выполняется только после успешной обработки или надежной постановки в
очередь.
    lastEventId = event.event_id || lastEventId;
  }).catch((error) => {
    console.error("agent failed; event checkpoint not advanced", error);
  });
});
```

При повторном подключении передайте сохраненный `lastEventId` как `last_event_id`, а затем обязательно перечитайте последние сообщения доступных разговоров. Пример не реализует цикл повторного подключения, чтобы не скрывать основной протокол за инфраструктурным кодом.

8. Регистрация нового типа агента в on-premise контуре

Искра поставляется со стандартными типами контроллеров агента:

- `basic_assistant` - стандартный ИИ-агент Искры;
- `event_organizer` - контроллер организатора мероприятий.

Дополнительный тип агента с собственной реализацией сейчас регистрируется только в локальном on-premise контуре заказчика. В текущей реализации это не самостоятельная операция администратора пространства и не функция публичного UI. Регистрацию выполняет команда, которая администрирует серверный контур Искры и имеет доступ к PostgreSQL, конфигурации Gateway и пользовательскому сервису агента.

Раздел 8 разделяет две зоны ответственности:

Участник	**Что выполняет**
Разработчик сервиса агента	Реализует серверный сервис агента, точку подготовки `PUT /v1/agents/<virtual_user_id>`, обработку API сервиса агента, хранение секретов,

	метрики и бизнес-логику.
On-premise команда заказчика	Развертывает сервис агента, регистрирует тип контроллера в Искре, настраивает сетевой доступ Gateway к точке подготовки и проверяет готовность сервиса до создания агента.
Администратор пространства	После регистрации типа создает конкретного агента, выбирает видимость, prompt, tools, права и включение. Gateway автоматически выпускает учетные данные и передает их в сервис агента во время подготовки.

Если разработчик поставляет сервис агента как продуктовый компонент, он передает on-premise команде: `controller_key`, внутренний URL сервиса, список поддерживаемых capabilities, список tools по умолчанию, требования к сети, healthcheck, переменные окружения и правила хранения секретов.

8.1 Что должен предоставить пользовательский сервис агента

Пользовательский сервис агента должен быть серверным сервисом, доступным из прикладной зоны Gateway по внутреннему адресу. Сервис должен реализовать точку подготовки агента:

```
PUT /v1/agents/<virtual_user_id>
Content-Type: application/json
```

Gateway вызывает эту точку при создании агента с соответствующим `controller_key`. В теле запроса передается конфигурация агента:

```
{
  "virtual_user_id": "0195f7f2-bot-user",
  "space_id": "0195f7f2-space",
  "display_title_snapshot": "Внешний агент",
  "prompt": "Инструкция агента из Искры.",
  "skills": [],
  "tools": [],
  "mcp_servers": [],
  "trigger_mode": "every_message",
  "model_tier": "default",
  "send_memo_with_messages": true,
  "human_in_loop_enabled": true,
  "human_in_loop_message": "Передаю обращение оператору.",
  "show_work_progress": true,
  "show_thinking": false,
  "enabled": true,
  "bot_api_base_url": "https://im.company.example",
  "bot_token": "<credential_id>.<secret>",
  "credential_id": "<credential_id>",
  "controller_key": "custom_support_agent",
```

```
  "config_version": "v1",
  "platform_version": "messenger.v1"
}
```

Сервис агента должен сохранить `bot_api_base_url`, `bot_token`, `virtual_user_id`, `credential_id` и собственную конфигурацию в защищенном серверном хранилище. После успешной подготовки он отвечает:

```
{
  "accepted": true,
  "controller_instance_id": "custom-support-agent-0195f7f2",
  "runtime_status": "active",
  "status_reason": "active",
  "controller_version": "v1"
}
```

Если сервис агента не принимает конфигурацию, он возвращает:

```
{
  "accepted": false,
  "runtime_status": "rejected",
  "status_reason": "missing required external setting"
}
```

8.2 Регистрация типа контроллера в PostgreSQL

Регистрация нового типа является контролируемым изменением базы данных локального внедрения. Ее выполняет администратор PostgreSQL или платформенный инженер заказчика в согласованное окно изменений; администратор пространства эту операцию не выполняет. До изменения команда заказчика создает резервную копию базы данных либо проверенную точку восстановления, фиксирует номер изменения, владельца сервиса агента, `controller_key`, URL сервиса и план отката. После каждого обновления Искры запись и вызов подготовки агента проверяются повторно.

На сервере PostgreSQL основного контура Искры добавьте запись в таблицу `ai_agent_controllers` внутри транзакции. Значение `key` затем используется как `controller_key` при создании агента.

Пример для пользовательского агента поддержки:

```
BEGIN;

INSERT INTO ai_agent_controllers (
  key,
  title,
  description,
  service_base_url,
  status,
  auth_mode,
  capabilities_json,
  default_prompt,
  default_tools_json,
```

```
  controller_version,
  created_at,
  updated_at
) VALUES (
  'custom_support_agent',
  'Пользовательский агент поддержки',
  'Пользовательский on-premise сервис агента поддержки',
  'http://custom-support-agent.internal:8080',
  'active',
  'none',
  '{
    "prompt": true,
    "model_tier": true,
    "greeting_message": true,
    "human_in_loop": true,
    "show_work_progress": true,
    "send_memo_with_messages": true
  }'::jsonb,
  '',
  '[]'::jsonb,
  'v1',
  NOW(),
  NOW()
)
ON CONFLICT (key) DO UPDATE
SET title = EXCLUDED.title,
    description = EXCLUDED.description,
    service_base_url = EXCLUDED.service_base_url,
    status = EXCLUDED.status,
    auth_mode = EXCLUDED.auth_mode,
    capabilities_json = EXCLUDED.capabilities_json,
    default_prompt = EXCLUDED.default_prompt,
    default_tools_json = EXCLUDED.default_tools_json,
    controller_version = EXCLUDED.controller_version,
    updated_at = NOW();

SELECT key, title, service_base_url, status, auth_mode, controller_version
FROM ai_agent_controllers
WHERE key = 'custom_support_agent';

COMMIT;
```

Если контрольный `SELECT` возвращает неверные значения, выполните `ROLLBACK` вместо `COMMIT`. Откат уже примененной регистрации выполняется отдельным согласованным изменением:

```
BEGIN;
```

```
UPDATE ai_agent_controllers
SET status = 'inactive', updated_at = NOW()
WHERE key = 'custom_support_agent';

-- Удаление допускается только после проверки, что агенты этого типа отсутствуют.
SELECT user_id, display_title, controller_key
FROM virtual_users
WHERE controller_key = 'custom_support_agent';

COMMIT;
```

Сначала переводите контроллер в `inactive`. Не удаляйте запись, пока существуют виртуальные пользователи с этим `controller_key`. После переноса или удаления таких агентов администратор PostgreSQL может выполнить `DELETE FROM ai_agent_controllers WHERE key = 'custom_support_agent'` в отдельной транзакции.

Поля:

- `key` - стабильный технический идентификатор типа агента; менять после запуска нельзя без миграции существующих агентов.
- `title` - название типа агента для административных списков.
- `service_base_url` - внутренний URL сервиса агента, на который Gateway отправит `PUT /v1/agents/<virtual_user_id>`.
- `status` - должен быть `active`, иначе создание агента с этим типом контроллера будет отклонено.
- `auth_mode` - справочное поле для пользовательского контроллера. Gateway 1.0 не добавляет универсальный заголовок авторизации при вызове произвольного контроллера, даже если в записи указано `service_token`. Подписанный служебный токен реализован только для встроенного `event_organizer`. Для пользовательского типа указывайте `none` и разрешайте `PUT /v1/agents/<virtual_user_id>` только с адресов Gateway посредством внутреннего сегмента сети, сетевой политики и mTLS или проверяющего обратного прокси. Публиковать эту точку в пользовательскую или внешнюю сеть запрещено.
- `capabilities_json` - признаки, какие настройки сервиса агента поддерживает.
- `default_tools_json` - массив ключей инструментов, которые должны быть включены для агента этого типа по умолчанию. У встроенных `basic_assistant` и `event_organizer` сейчас используется `[]`.
- `controller_version` - версия контракта сервиса агента, используйте `v1`.

Поддерживаемые идентификаторы `capabilities_json` в текущей реализации:

Идентификатор	**Назначение**
`prompt`	UI и API могут передавать системную инструкцию агента в поле `prompt`.
`model_tier`	Агент поддерживает выбор уровня LLM-

	модели: `default`, `medium`, `large`.
`skills`	UI может назначать агенту навыки из `/v1/agent-capabilities`.
`tools`	UI может назначать агенту инструменты из `/v1/agent-capabilities`.
`mcp_servers`	UI может назначать агенту MCP-серверы пространства.
`greeting_message`	Агент поддерживает приветственное сообщение.
`human_in_loop`	Агент поддерживает передачу обращения оператору и соответствующее сообщение.
`show_work_progress`	Агент поддерживает отображение пользователю статусов работы.
`show_thinking`	Агент поддерживает отображение блока размышлений.
`send_memo_with_messages`	Агент поддерживает передачу memo разговора в сервисе агента.
`event_workflows`	Специальная возможность контроллера `event_organizer`: сценарии мероприятия.
`workflow_graph`	Специальная возможность контроллера `event_organizer`: граф workflow мероприятия.

Встроенный `basic_assistant` объявляет `prompt`, `model_tier`, `skills`, `tools`, `mcp_servers`, `greeting_message`, `human_in_loop`, `show_work_progress`, `show_thinking`, `send_memo_with_messages`. Встроенный `event_organizer` объявляет `prompt`, `model_tier`, `greeting_message`, `human_in_loop`, `show_work_progress`, `send_memo_with_messages`, `event_workflows`, `workflow_graph`.

Поддерживаемые ключи инструментов для `default_tools_json` и `tool_ids`:

Ключ инструмента	**Назначение**
`web-search`	Поиск в интернете через настроенный сервер поиска.
`yandex-calendar`	Работа с Yandex Calendar пользователя.
`google-calendar`	Работа с Google Calendar пользователя.
`apple-calendar`	Работа с Apple Calendar пользователя.
`bitrix24-calendar`	Чтение календаря Bitrix24 текущего пользователя для этого агента.
`bitrix24-task`	Чтение задач Bitrix24 текущего пользователя

	для этого агента.
`bitrix24-funnel`	Чтение сделок и лидов Bitrix24 текущего пользователя для этого агента.
`youtrack`	Работа с задачами YouTrack через URL и постоянный токен агента.
`headhunter-resume-search`	Поиск резюме в HeadHunter без открытия контактов и платных действий.
`healthy-lifestyle-consultation`	Специализированная консультация по здоровому образу жизни.
`ida-knowledge-base`	Поиск в настроенной базе знаний Ида.

Если controller type не поддерживает `tools`, `default_tools_json` оставляйте пустым. Если инструмент требует пользовательской авторизации или секретов, одного значения в `default_tools_json` недостаточно: администратор или пользователь должен выполнить соответствующее подключение в UI.

8.3 Проверка регистрации

После добавления записи проверьте, что Gateway видит новый тип контроллера:

```
GET /v1/agent-controllers
Authorization: Bearer <admin_access_token>
```

В ответе должен быть элемент:

```
{
  "key": "custom_support_agent",
  "title": "Пользовательский агент поддержки",
  "service_base_url": "http://custom-support-agent.internal:8080",
  "status": "active",
  "auth_mode": "none",
  "controller_version": "v1"
}
```

Затем выполните пробное создание агента в тестовом пространстве и проверьте журнал сервиса агента: запрос `PUT /v1/agents/<virtual_user_id>` должен прийти от Gateway по разрешенному сетевому пути, без универсального `Authorization` для пользовательского контроллера. Убедитесь, что посторонний узел не может вызвать эту точку. Результаты проверки приложите к записи об изменении.

8.4 Создание агента нового типа

В web UI создания агента поле "Тип ИИ-Агента" показывает зарегистрированные в Gateway типы контроллеров из `/v1/agent-controllers`, например `basic_assistant` и `event_organizer`. Новый пользовательский тип появится в этом списке только после регистрации записи в `ai_agent_controllers` и должен создаваться с соответствующим `controller_key`. Сам UI не регистрирует новый тип контроллера; регистрацию выполняет on-premise административная команда по процедуре из раздела 8.2.

```
POST /v1/virtual-users?space_id=<space_id>
Authorization: Bearer <admin_access_token>
Content-Type: application/json
{
  "bot_definition_key": "ai-agent",
  "controller_key": "custom_support_agent",
  "display_title": "Внешний агент поддержки",
  "visibility_mode": "space_public",
  "trigger_mode": "every_message",
  "model_tier": "default",
  "prompt": "Отвечай как агент первой линии поддержки. Если вопрос требует человека, передай обращение оператору.",
  "greeting_message": "Здравствуйте! Я помогу с первичной консультацией.",
  "send_memo_with_messages": true,
  "show_work_progress": true,
  "show_thinking": false
}
```

При успешном создании Gateway:

1. создает виртуального пользователя агента;
2. выпускает учетные данные агента;
3. вызывает `PUT <service_base_url>/v1/agents/<virtual_user_id>`;
4. передает сервису агента `bot_api_base_url`, `bot_token`, `credential_id` и конфигурацию;
5. сохраняет статус подготовки в карточке виртуального пользователя.

Если сервис агента недоступен или возвращает `accepted=false`, агент будет создан со статусом настройки `setup_failed` или `provisioning_failed`. После исправления сервиса агента повторите создание агента или обновите существующего агента через внутренний административный сценарий.

8.5 Работа пользовательского сервиса агента

Сервис агента запущен до создания агента. После успешного ответа на запрос подготовки он переходит к рабочему циклу:

1. получает lease через `/v1/bot-runtime/leases`;
2. подключается к `/v1/bot-runtime/ws`;
3. обрабатывает `message.created`;
4. читает сообщения и memo;
5. отправляет обычные или потоковые ответы;
6. создает tool requests для действий с подтверждением;
7. для агента клиентской поддержки вызывает handoff, если нужен оператор.

8.6 Вызов выбранных инструментов пользовательским сервисом агента

Выбор инструмента в карточке агента разрешает его использование, но не вызывает встроенную реализацию Искры. В API сервиса агента 1.0 нет точки доступа вида `/tools/<key>/execute`. Пользовательский сервис агента выполняет инструмент самостоятельно: получает описание при подготовке агента, строит схему вызова для своей LLM, проверяет аргументы и обращается к целевой системе напрямую.

Чтобы в карточке пользовательского агента можно было выбирать инструменты, его запись `ai_agent_controllers.capabilities_json` должна содержать `"tools": true`. Администратор пространства выбирает инструменты в поле **Инструменты**. После создания или обновления агента Gateway передает выбранные инструменты в массиве `tools` запроса:

```
PUT <service_base_url>/v1/agents/<virtual_user_id>
Content-Type: application/json
{
  "virtual_user_id": "0195f7f2-bot-user",
  "tools": [
    {
      "id": "9f609d92-6e5d-4f10-a4dd-d54ad58763e6",
      "key": "youtrack",
      "title": "YouTrack",
      "description": "Работа с задачами YouTrack.",
      "execution_type": "server_youtrack",
      "config": {
        "base_url": "https://youtrack.company.example",
        "token": "<permanent_token>"
      }
    }
  ]
}
```

Состав элемента `tools`:

Поле	Как использует пользовательский сервис агента
`id`	Стабильный идентификатор записи инструмента в Искре. Сохраняется вместе с конфигурацией агента.
`key`	Идентификатор инструмента, например `youtrack` или `google-calendar`. По нему сервис агента выбирает собственный обработчик.
`title`, `description`	Метаданные для журналов, административного интерфейса и описания функции для LLM.
`execution_type`	Подсказка о классе встроенного обработчика,

	например `server_youtrack`. Это не исполняемый протокол и не заменяет спецификацию API поставщика.
`config`	Общая конфигурация инструмента, объединенная с настройками и учетными данными, прикрепленными к этому агенту. Содержимое зависит от инструмента.

Если к инструменту прикреплены серверные учетные данные, Gateway раскрывает их при подготовке агента и добавляет в `config`. Поэтому точка `PUT /v1/agents/<virtual_user_id>`, база данных сервиса агента, резервные копии и журналы должны обрабатываться как контур секретов. Не записывайте полный `config` в журнал. Назначайте такие инструменты только доверенному пользовательскому контроллеру.

Gateway не передает JSON Schema аргументов функции, реализацию вызова или нормализованную схему результата. Разработчик пользовательского сервиса агента определяет их в своем коде и версионирует вместе с сервисом агента. Рекомендуемый цикл:

1. При `PUT /v1/agents/<virtual_user_id>` сохранить только инструменты, присутствующие в текущем массиве `tools`; удаленные из массива инструменты отключить.
2. Для каждого поддерживаемого `key` зарегистрировать собственную JSON Schema аргументов LLM и обработчик исполнения.
3. При получении `message.created` передать LLM только схемы инструментов, выбранных для этого агента.
4. Проверить имя инструмента и аргументы независимо от LLM. Запретить неизвестные поля, ограничить длину строк, диапазоны чисел и допустимые операции.
5. Определить пользователя, от имени которого выполняется операция, по `sender_id` исходного сообщения. Не использовать `virtual_user_id` вместо пользователя.
6. Проверить наличие серверной или пользовательской авторизации. Если требуется OAuth, выполнить процедуру из раздела 17.5.
7. Если действие требует подтверждения человека, до обращения к внешней системе выполнить процедуру tool request из раздела 19 и дождаться `completed` посредством polling.
8. Вызвать API поставщика напрямую из сервиса агента с ограничением времени, контролируемым числом повторов и собственным ключом идемпотентности для изменяющих операций.
9. Нормализовать ответ поставщика, удалить секреты и лишние персональные данные, передать результат LLM как результат инструмента.
10. Сформировать пользовательский ответ и отправить его через API сервиса агента. Не утверждать успех, если внешний API вернул ошибку или результат подтверждения не получен.

Минимальная структура диспетчера инструментов:

```
const handlers = {
  youtrack: executeYouTrack,
  "ida-knowledge-base": executeIdaKnowledgeBase,
  "google-calendar": executeGoogleCalendar,
};

async function executeSelectedTool(agent, sourceMessage, call) {
  const selected = agent.tools.find((tool) => tool.key === call.name);
  if (!selected) throw new Error("tool_not_selected");

  const handler = handlers[selected.key];
  if (!handler) throw new Error("tool_not_supported_by_agent_service");

  const args = validateToolArguments(selected.key, call.arguments);
  const context = {
    virtualUserId: agent.virtual_user_id,
    conversationId: sourceMessage.conversation_id,
    requesterUserId: sourceMessage.sender_id,
  };

  return handler({ context, config: selected.config, args });
}
```

Доступность данных для стандартных инструментов:

Инструменты	**Что получает пользовательский сервис агента**	**Что требуется реализовать дополнительно**
`youtrack`	Определение и прикрепленные к агенту URL/постоянный токен, если администратор их настроил.	Валидацию операций, HTTP-клиент YouTrack и нормализацию результата.
`ida-knowledge-base`, `healthy-lifestyle-consultation`	Определение и доступную конфигурацию endpoint/учетных данных.	Протокол вызова соответствующего сервиса и обработку его ошибок.
`web-search`	Определение инструмента; его `config` по умолчанию пуст.	Адрес и учетные данные собственного поискового сервиса, JSON Schema и выполнение поиска. Конфигурация встроенного `assistant` автоматически не передается.
`google-calendar`, `yandex-calendar`, `apple-calendar`	Определение выбранного инструмента.	Собственное OAuth-приложение, OAuth-сессии пользователей,

		хранение/обновление токенов и клиент API календаря.
`bitrix24-calendar`, `bitrix24-task`, `bitrix24-funnel`	Определение выбранного инструмента.	Собственное приложение Bitrix24, OAuth и вызовы REST API. Подключения встроенного `assistant` пользовательскому сервису агента не передаются.
`headhunter-resume-search`	Определение выбранного инструмента.	Собственное OAuth-приложение HeadHunter, callback, хранение токенов и реализацию поиска резюме.
Назначенные MCP-серверы	Массив `mcp_servers` с адресом, конфигурацией и прикрепленными учетными данными.	MCP-клиент, проверку разрешенных инструментов сервера и ограничения времени. Это предпочтительный способ повторно использовать независимую реализацию инструмента.

`tool request` и выбранный инструмент решают разные задачи. Выбранный инструмент сообщает сервису агента, что он может предложить и выполнить функцию. `tool request` фиксирует ожидание подтверждения человека; он не вызывает YouTrack, календарь, Bitrix24 или другой встроенный обработчик Искры.

9. MCP-инструменты для базового и пользовательского агента

MCP используется, когда агенту нужно предоставить инструменты отдельного сервера: поиск во внутренней базе, чтение CRM, создание заявок или другое обращение к корпоративной системе. В Искре MCP-сервер регистрируется на уровне пространства и затем назначается конкретным агентам. Искра 1.0 не импортирует инструменты MCP-сервера в каталог `agent_tools`: единицей назначения является весь MCP-сервер.

9.1 Поддержка в Искре 1.0

Наличие поля **MCP-серверы** в карточке агента означает, что выбранный тип контроллера принимает конфигурацию `mcp_servers`. Оно не гарантирует, что соответствующий сервис агента реализует MCP-клиент.

Тип агента	**Регистрация и назначение MCP-сервера**	**Фактическое выполнение MCP-инструментов**
Базовый ИИ-агент, `basic_assistant`	Поддерживается. Администратор может назначить сервер в карточке агента; Gateway передает описание встроенному	В версии 1.0 не поддерживается. Встроенный сервис сохраняет описание сервера и добавляет его название в контекст модели,

	сервису `assistant`.	но не устанавливает MCP-сессию, не выполняет `tools/list` и не вызывает `tools/call`. Назначение сервера нельзя считать подтверждением доступности его инструментов.
Пользовательский тип агента	Поддерживается, если в `capabilities_json` типа указано `"mcp_servers": true`. Gateway передает выбранные серверы в запросе подготовки.	Реализуется пользовательским сервисом агента. Он подключается к серверу, получает список инструментов, передает разрешенные схемы модели и выполняет вызовы.
Организатор мероприятия, `event_organizer`	Не поддерживается: тип не объявляет возможность `mcp_servers`, поэтому поле выбора недоступно.	Не поддерживается контрактом этого типа.

Следствие для промышленного сценария: произвольный MCP-инструмент в Искре 1.0 используется через зарегистрированный пользовательский тип агента. Для базового агента назначение MCP-сервера является только сохраненной конфигурацией и не должно включаться в приемочные сценарии как работающая интеграция.

9.2 Что должен предоставить MCP-сервер

MCP-сервер разрабатывает и эксплуатирует команда интеграции заказчика или поставщик внешней системы. Для работы с пользовательским сервисом агента он должен предоставить:

- сетевой URL, доступный из контура пользовательского сервиса агента;
- транспорт и версию MCP, согласованные с клиентской библиотекой сервиса агента;
- операцию инициализации MCP-сессии;
- `tools/list` с устойчивыми именами инструментов, описаниями и схемами входных аргументов;
- `tools/call` с машиночитаемым результатом и признаком ошибки инструмента;
- серверную аутентификацию, если сервер не является общедоступным внутри доверенного контура;
- ограничения времени, размера запроса и результата, а также однозначные ошибки для временного и постоянного отказа;
- идемпотентность или собственный ключ идемпотентности для операций, изменяющих данные.

Для удаленного сервера используйте потоковый HTTP-транспорт MCP через HTTPS. `stdio` не подходит для регистрации по полю **Server URL**, поскольку Искра передает URL, а не команду

запуска локального процесса. Конкретную версию протокола выбирают разработчики MCP-сервера и пользовательского сервиса агента; Gateway Искры ее не согласовывает и не проверяет.

Описание инструмента и его схема являются недоверенными входными данными. Пользовательский сервис агента должен применять собственный список разрешенных имен, ограничивать аргументы и не предоставлять модели инструмент только потому, что он появился в `tools/list`.

Минимальный MCP-сервер на Python может публиковать один безопасный инструмент чтения следующим образом:

Установка зависимости:

```
python -m pip install "mcp[cli]>=1.4,<2"
```

Файл `server.py`:

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("Iskra Company Directory")



@mcp.tool(
    name="find_employee",
    description="Найти сотрудника по точному корпоративному адресу почты.",
)
async def find_employee(email: str) -> dict:
    normalized = email.strip().lower()
    if not normalized.endswith("@company.example"):
        raise ValueError("email_not_allowed")
    return await directory_api_find_employee(normalized)



if __name__ == "__main__":
    mcp.run(transport="streamable-http")
```

Версию библиотеки и транспорта фиксируют в зависимостях MCP-сервера. Внешний обратный прокси завершает TLS, ограничивает размер запросов и публикует фактический URL MCP, который затем вводится в Искре. Функция инструмента повторно проверяет права и аргументы: описание и JSON Schema помогают модели сформировать вызов, но не являются механизмом безопасности.

9.3 Регистрация MCP-сервера администратором пространства

Регистрацию выполняет владелец или администратор пространства после того, как команда интеграции развернула MCP-сервер и проверила его из сетевого контура сервиса агента.

1. Открыть нужное пространство в web-интерфейсе Искры.
2. Перейти на вкладку **ИИ-Агенты**.
3. В блоке **MCP-серверы пространства** нажать **Добавить MCP**.
4. Заполнить **Название** - понятное администраторам имя подключения.

5. Заполнить **Ключ** - устойчивый технический идентификатор. Если поле оставить пустым, Gateway сформирует ключ из названия. После начала эксплуатации ключ менять не следует.
6. Заполнить **Описание** - назначение сервера, допустимые данные и владельца интеграции.
7. В поле **Server URL** указать полный HTTPS URL точки MCP, например `https://mcp.crm.company.example/mcp`.
8. Оставить статус **Активен** и нажать **Создать**.

Текущий интерфейс не проверяет соединение, сертификат, MCP-инициализацию или список инструментов при сохранении. Создание записи подтверждает только сохранение конфигурации в Gateway. Проверку доступности выполняет команда интеграции отдельно по процедуре 9.8.

Интерфейс регистрации не содержит произвольных полей конфигурации и учетных данных. Не помещайте токен, пароль или ключ API в URL. Для общей серверной аутентификации передайте секрет пользовательскому сервису агента через хранилище секретов его среды и сопоставьте его с `key` или `id` MCP-сервера. Настройка Bitrix24 является отдельным специализированным сценарием интерфейса и не определяет общий контракт произвольного MCP-сервера.

9.4 Назначение MCP-сервера агенту

До назначения пользовательский тип агента должен быть зарегистрирован с возможностью:

```
{
  "mcp_servers": true
}
```

Администратор пространства выполняет следующие действия:

1. На вкладке **ИИ-Агенты** открыть существующего агента или начать создание нового.
2. Выбрать пользовательский тип агента, сервис которого реализует MCP-клиент.
3. Раскрыть блок **MCP-серверы**.
4. Отметить только серверы, необходимые этому агенту.
5. Сохранить агента.
6. Убедиться, что карточка агента не показывает **Ошибка настройки**.
7. Передать команде интеграции идентификатор тестового разговора и выполнить функциональную проверку инструмента.

При создании или изменении агента Gateway отправляет актуальный массив `mcp_servers` в `PUT /v1/agents/<virtual_user_id>`. Сервис агента должен считать этот массив полным снимком: удалить отключенные подключения, обновить измененные и не сохранять ранее назначенный сервер, которого больше нет в массиве.

Для планового отключения сначала снимите назначение MCP-сервера со всех агентов и сохраните каждую карточку. Gateway передаст сервисам агентов новый снимок без этого сервера. После подтверждения применения конфигурации переведите запись MCP-сервера в статус **Выключен**. Простое изменение статуса записи не рассылает новую конфигурацию уже подготовленным сервисам агентов, поэтому менять порядок этих действий нельзя.

9.5 Контракт передачи MCP-конфигурации

Пример фрагмента запроса подготовки пользовательского сервиса агента:

```
{
  "virtual_user_id": "0195f7f2-bot-user",
  "controller_key": "company_support_agent",
  "mcp_servers": [
    {
      "id": "0195f7f2-mcp-server",
      "key": "company-crm",
      "title": "Корпоративная CRM",
      "description": "Чтение карточки клиента и создание обращения.",
      "server_url": "https://mcp.crm.company.example/mcp",
      "config": {}
    }
  ]
}
```

Поле	Обязательное использование сервисом агента
`id`	Стабильный идентификатор записи в Искре. Используется как основной ключ сохраненной конфигурации.
`key`	Технический идентификатор интеграции. Может использоваться для выбора локальной политики и секрета, но не заменяет `id`.
`title`, `description`	Административные метаданные. Не используются как основание для выдачи прав.
`server_url`	Адрес подключения. Перед соединением сервис проверяет схему HTTPS и разрешенный домен или сетевой диапазон.
`config`	Непрозрачная для Gateway конфигурация сервера и, при наличии прикрепленных учетных данных, раскрытые значения секрета. Полный объект нельзя записывать в журнал.

Gateway не передает результат `tools/list`, выбранные имена MCP-инструментов, схему подтверждения действий или готовый клиент MCP. Эти элементы принадлежат пользовательскому сервису агента.

Точки Management API для регистрации сервера доступны владельцу или администратору пространства:

Точка доступа	Метод	Назначение

`/v1/agent-capabilities?space_id=<space_id>`	`GET`	получить видимые MCP-серверы в массиве `mcp_servers`
`/v1/spaces/<space_id>/agent-mcp-servers`	`POST`	создать MCP-сервер пространства
`/v1/spaces/<space_id>/agent-mcp-servers/<server_id>`	`PATCH`	изменить название, ключ, описание, URL, `config` или статус

Минимальное тело создания:

```
{
  "key": "company-crm",
  "title": "Корпоративная CRM",
  "description": "Инструменты CRM для агента поддержки.",
  "server_url": "https://mcp.crm.company.example/mcp",
  "status": "active"
}
```

API принимает необязательный объект `config`, хотя текущий административный интерфейс его не редактирует. Значения `config` хранятся в Gateway и передаются сервису агента. Постоянные секреты следует хранить в защищенном хранилище пользовательского сервиса, а не в `config`. Программную настройку через Management API применяйте только в согласованной автоматизации on-premise внедрения; обычную регистрацию выполняет администратор по процедуре 9.3.

9.6 Выполнение MCP-инструмента пользовательским сервисом агента

Для каждого назначенного сервера сервис агента выполняет следующий цикл:

1. Получить новую конфигурацию через `PUT /v1/agents/<virtual_user_id>` и атомарно сохранить снимок.
2. Проверить `server_url` по списку разрешенных адресов и получить серверный секрет из собственного хранилища по `id` или `key`.
3. Установить MCP-сессию и завершить согласование версии протокола.
4. Вызвать `tools/list`, обработать постраничную выдачу и оставить только инструменты из локального списка разрешений этого агента.
5. Передать языковой модели имена, описания и схемы только разрешенных инструментов.
6. После выбора моделью проверить имя инструмента и аргументы по схеме независимо от модели.
7. Для чтения определить пользователя и область данных по `sender_id`, `conversation_id` и политике интеграции. Для изменения данных до вызова MCP выполнить подтверждение по процедуре tool request из раздела 19.

8. Вызвать `tools/call` с ограничением времени и размером результата. Автоматически повторять только безопасные операции чтения либо изменяющие операции с подтвержденной идемпотентностью.
9. Проверить ошибку инструмента, удалить секреты и лишние персональные данные из результата, затем передать нормализованный результат модели.
10. Отправить ответ через API сервиса агента и записать технический журнал без содержимого секретов.

Иллюстративный фрагмент клиента на Node.js с официальной библиотекой MCP:

```
import { Client, StreamableHTTPClientTransport } from
"@modelcontextprotocol/client";

async function connectMcp(server, bearerToken) {
  const authProvider = bearerToken
    ? { token: async () => bearerToken }
    : undefined;
  const transport = new StreamableHTTPClientTransport(
    new URL(server.server_url),
    { authProvider },
  );
  const client = new Client({ name: "iskra-agent-service", version: "1.0.0" });
  await client.connect(transport);
  return client;
}

async function callAllowedMcpTool(client, allowedNames, call) {
  const tools = [];
  let cursor;
  do {
    const page = await client.listTools({ cursor });
    tools.push(...page.tools);
    cursor = page.nextCursor;
  } while (cursor);

  const tool = tools.find((item) => item.name === call.name);
  if (!tool || !allowedNames.has(tool.name)) throw new
Error("mcp_tool_not_allowed");

  validateAgainstJsonSchema(tool.inputSchema, call.arguments);
  const result = await client.callTool({ name: tool.name, arguments:
call.arguments });
  if (result.isError) throw new Error("mcp_tool_failed");
  return sanitizeToolResult(result.content);
}
```

Версию библиотеки MCP фиксируйте в файле зависимостей сервиса агента и проверяйте пример против этой версии: API библиотеки не является частью контракта Искры. В промышленной реализации соединение следует переиспользовать, корректно закрывать при обновлении конфигурации и ограничивать число одновременных вызовов каждого сервера.

9.7 Аутентификация пользователя и сервера

Нужно различать два уровня авторизации:

- **Сервис агента - MCP-сервер.** Это серверная аутентификация подключения: mTLS, постоянный токен или OAuth клиента. Ее настраивает команда интеграции. Текущий общий интерфейс MCP в Искре не запрашивает такой секрет.
- **Пользователь - внешняя система.** Она нужна, если инструмент выполняет действие от имени автора сообщения. Пользовательский сервис агента связывает OAuth-сессию с `sender_id` и выполняет поток из раздела 17.5. Токен пользователя хранится сервисом агента или доверенным MCP-сервером, но не передается модели и не размещается в сообщении.

Если OAuth реализован самим MCP-сервером, сервис агента должен обработать требование авторизации как незавершенный вызов: создать одноразовую сессию, отправить пользователю карточку авторизации, дождаться callback, повторно проверить автора и разговор, затем повторить `tools/call`. Назначение MCP-сервера агенту само по себе не авторизует пользователей.

Для общей интеграции нескольких агентов используйте отдельные серверные учетные данные или область прав на каждого агента, когда это поддерживает внешняя система. Не применяйте один неограниченный токен для всех пространств и агентов.

9.8 Проверка и диагностика

Перед передачей агента пользователям команда интеграции выполняет проверку из того же сетевого контура, где работает пользовательский сервис агента:

1. DNS-имя разрешается, сертификат доверен, HTTPS-соединение устанавливается без обхода проверки TLS.
2. MCP-инициализация завершается с согласованной версией протокола.
3. `tools/list` возвращает ожидаемые имена и схемы; неожиданные инструменты отфильтрованы локальным списком разрешений.
4. Безопасный тестовый `tools/call` возвращает ожидаемый результат.
5. Вызов неизвестного инструмента и неверные аргументы отклоняются.
6. Вызов без учетных данных или с недостаточными правами отклоняется MCP-сервером.
7. Изменяющая операция не выполняется до подтверждения пользователя и не дублируется при сетевом повторе.
8. Тайм-аут, недоступность сервера и ошибка инструмента приводят к честному сообщению пользователю, а не к утверждению об успехе.
9. После снятия назначения и получения нового снимка конфигурации сервис агента прекращает предлагать и вызывать инструменты этого сервера; затем администратор переводит запись сервера в статус **Выключен**.

10. Журналы содержат `virtual_user_id`, `conversation_id`, `mcp_server_id`, имя инструмента, длительность и итоговый статус, но не токены, пароли, полные аргументы с персональными данными и необработанный результат.

Gateway Искры не предоставляет отдельную точку проверки MCP-сервера и не журналирует фактические `tools/call`, выполняемые пользовательским сервисом. Контроль доступности и технический журнал вызовов являются обязанностью команды, эксплуатирующей пользовательский сервис агента и MCP-сервер.

10. Management API: справочно для платформенной интеграции

Основной путь создания агента для заказчика - ручная настройка в административном интерфейсе Искры, описанная в разделе 4. Точки доступа ниже нужны для платформенной интеграции, автоматизированной подготовки или внутренних административных сценариев, если они разрешены политикой внедрения.

Точка доступа	Метод	Назначение
`/v1/bots`	`GET`	список зарегистрированных описаний ботов
`/v1/agent-controllers`	`GET`	доступные типы контроллеров сервиса агента
`/v1/agent-capabilities?space_id=<space_id>`	`GET`	возможности агента в пространстве
`/v1/virtual-users?space_id=<space_id>`	`GET`	список virtual users
`/v1/virtual-users?space_id=<space_id>`	`POST`	создать virtual user
`/v1/virtual-users/<user_id>`	`PATCH`	обновить настройки virtual user
`/v1/virtual-users/<user_id>/credentials/rotate`	`POST`	выдать новую версию bot credential
`/v1/virtual-users/suggestions?space_id=<space_id>`	`POST`	предложить значения по умолчанию для virtual user
`/v1/agent-skills/suggestions?space_id=<space_id>`	`POST`	предложить значения по умолчанию для skills/tools

Режимы видимости

Используйте одно из значений:

- `private` - агент виден только явно разрешенным пользователям и политикам платформы.
- `space_public` - агент виден внутри пространства.

- `global_public` - агент виден платформенно, если это разрешено политикой Искры.

Для сервисных агентов обычно достаточно `space_public`. Для экспериментальных или внутренних агентов используйте `private`.

Ротация credential

Credential имеет версионированный жизненный цикл:

- `issued`;
- `delivered`;
- `acknowledged`;
- `active`;
- `revoked`;
- `failed`.

После ротации старый секрет должен быть удален из сервиса агента. Не логируйте `credential_id.secret` целиком. В логах допустимы только последние 4-6 символов credential id или отдельный correlation id.

11. Assistant Control API

Assistant control plane используется встроенным ИИ-агентом Искры. Внешний сервис агента может не использовать этот API, если хранит системную инструкцию, инструменты и конфигурацию у себя. Но формат полезен как справочная модель.

Точка доступа	Метод	Назначение
`/v1/agents/<virtual_user_id>`	`GET`	получить конфигурацию агента
`/v1/agents/<virtual_user_id>`	`PUT`	создать или обновить конфигурацию
`/v1/agents/<virtual_user_id>/voice-turns...`	разные	голосовые сценарии агента

Пример конфигурации:

```
{
  "virtual_user_id": "0195f7f2-bot-user",
  "space_id": "0195f7f2-space",
  "display_title_snapshot": "AI-консультант",
  "prompt": "Ты консультант первой линии.",
  "tools": [
    {
      "id": "0195f7f2-tool",
      "key": "youtrack",
```

```
      "title": "YouTrack",
      "execution_type": "server_youtrack",
      "config": {}
    }
  ],
  "mcp_servers": [
    {
      "id": "0195f7f2-mcp",
      "key": "company-crm",
      "title": "Корпоративная CRM",
      "server_url": "https://mcp.crm.company.example/mcp",
      "config": {}
    }
  ],
  "trigger_mode": "every_message",
  "enabled": true,
  "bot_api_base_url": "https://im.iskra-messenger.ru",
  "bot_token": "<credential_id>.<secret>",
  "credential_id": "<credential_id>",
  "last_controller_status": "active"
}
```

Правила:

- `display_title_snapshot` - снимок имени из Искры, не источник истины.
- `prompt`, `tools`, `mcp_servers`, `trigger_mode`, `enabled` - поведение сервиса агента.
- `bot_token` хранится как секрет и не должен уходить в клиентское приложение.
- Если `enabled=false`, сервис агента должен остановить websocket и отпустить lease.

12. Общий контракт API сервиса агента

Все точки доступа API сервиса агента находятся под `/v1/bot-runtime`. Сервис агента вызывает их только с серверными учетными данными виртуального пользователя.

Обязательные заголовки:

Заголовок	Когда нужен	Значение
`Authorization`	Всегда	`Bearer <credential_id>.<secret>`
`X-Bot-Session-ID`	Всегда для HTTP-запросов сервиса агента	Стабильный идентификатор текущего процесса сервиса агента. При перезапуске создавайте новый.
`X-Bot-Lease-ID`	Для `GET /conversations`, чтения и записи messages/memo, загрузки	Lease, полученный через `/v1/bot-runtime/leases`. Заголовок не требуется для

	attachments, всех stream-операций, handoff и tool requests	получения, продления и освобождения lease. Текущий `POST /conversations/<id>/events` также не проверяет lease.
`Content-Type`	Для запросов с телом	`application/json`

Форма ошибки:

```
{
  "code": "bot_lease_unavailable",
  "message": "bot lease unavailable"
}
```

Коды ошибок, которые сервис агента должен обрабатывать как часть нормального контракта:

HTTP	**code**	**Что означает для сервиса агента**
`400`	`invalid_virtual_user`, `invalid_agent_tool_request`, `invalid_service_request`	Payload или сценарий некорректен; исправить запрос, не повторять без изменения.
`401`	`invalid_bot_credential`	Credential недействителен или отозван; остановить сервис агента и запросить ротацию credential.
`403`	`forbidden` или доменный запрет	У virtual user нет доступа к объекту; перечитать конфигурацию и права агента.
`404`	`not_found`, `message_stream_not_found`, `agent_tool_request_not_found`	Объект недоступен или уже удален; не считать успешным выполнением.
`409`	`bot_lease_unavailable`	Lease потерян, занят другим session или истек; остановить текущую обработку и reacquire после backoff.
`429`	rate limit	Уменьшить частоту запросов и применить backoff.
`5xx`	`internal_error`	Повторять только идемпотентные операции или операции с собственной дедупликацией сервиса агента.

Правила идемпотентности:

- `GET` можно повторять после сетевых ошибок.
- `PATCH /leases/<lease_id>` можно повторять, пока session и lease совпадают.
- `POST /conversations/<id>/messages` поддерживает `client_request_id`. Для каждой логической отправки создавайте UUID, сохраняйте его до запроса и повторяйте запрос после сетевой ошибки с тем же значением. Gateway возвращает уже созданное сообщение, если запрос с таким идентификатором ранее был принят.
- `POST /streams/<id>` создает пользовательски видимый результат; после сетевого таймаута сервис агента должен сначала перечитать сообщения, чтобы не отправить дубль. Отдельной точки чтения состояния stream в API сервиса агента 1.0 нет.
- `PATCH /streams/<id>` добавляет фрагмент; сервис агента должен хранить уже отправленный текст или строить stream так, чтобы повтор не создавал неверный ответ.
- `POST /tool-requests` создает новый request; не повторяйте без собственной дедупликации по бизнес-ключу в `request_payload`.

Точные ограничения, установленные API сервиса агента 1.0:

Объект	**Ограничение**
Lease	Срок действия `30` секунд; рекомендуемый интервал продления `10` секунд.
История сообщений	`limit` от `1` до `100`, по умолчанию `50`; постраничное чтение отсутствует.
Повторная доставка WebSocket	Не более `500` сохраненных событий после `last_event_id`; признак неполноты отсутствует.
Запрос инструмента	Ожидание результата `2` минуты; после завершения объект хранится еще `10` минут.
Завершенный stream	Состояние хранится `24` часа. Ошибочный или отмененный stream хранится `7` суток. Endpoint чтения stream отсутствует.
Ответ пользовательского контроллера при подготовке агента	Gateway читает не более `1` МиБ ответа.

API сервиса агента 1.0 не устанавливает отдельный публичный лимит длины `body`, `memo.markdown`, `delta` или `request_payload` на уровне обработчика. Фактический предел HTTP-тела может задаваться обратным прокси локального внедрения. Разработчик согласует этот предел с платформенной командой заказчика, ограничивает собственные запросы и обрабатывает `413`, даже если такой ответ формирует не Gateway.

13. API сервиса агента: модель lease

Перед чтением событий и записью сообщений сервис агента получает lease. Lease нужен, чтобы один virtual user не отвечал из нескольких worker-процессов одновременно.

Точка доступа	Метод	Назначение
`/v1/bot-runtime/leases`	`POST`	получить lease
`/v1/bot-runtime/leases/<lease_id>`	`PATCH`	продлить lease
`/v1/bot-runtime/leases/<lease_id>`	`DELETE`	отпустить lease

Получение lease:

```
POST /v1/bot-runtime/leases
Authorization: Bearer <bot_token>
X-Bot-Session-ID: agent-worker-1
Content-Type: application/json
{
  "lease_id": "agent-worker-1-lease"
}
```

Ответ:

```
{
  "user_id": "0195f7f2-bot-user",
  "lease_id": "agent-worker-1-lease",
  "session_id": "agent-worker-1",
  "expires_at": "2026-04-20T12:00:30Z",
  "heartbeat_at": "2026-04-20T12:00:00Z",
  "created_at": "2026-04-20T12:00:00Z"
}
```

Схема `BotConsumerLease`:

Поле	Тип	Назначение
`user_id`	string	Virtual user агента.
`lease_id`	string	Идентификатор lease, заданный сервисом агента при получении lease.
`session_id`	string	Значение `X-Bot-Session-ID`, которому принадлежит lease.
`expires_at`	RFC3339 datetime	Момент истечения lease. В текущей реализации новый lease выдается на 30 секунд.
`heartbeat_at`	RFC3339 datetime	Последнее успешное

		получение или продление lease.
`created_at`	RFC3339 datetime	Момент создания текущего lease.

Продление lease:

```
PATCH /v1/bot-runtime/leases/<lease_id>
Authorization: Bearer <bot_token>
X-Bot-Session-ID: agent-worker-1
```

Успешный ответ: `204 No Content`.

Освобождение lease:

```
DELETE /v1/bot-runtime/leases/<lease_id>
Authorization: Bearer <bot_token>
X-Bot-Session-ID: agent-worker-1
```

Успешный ответ: `204 No Content`.

Практика:

- продлевайте lease каждые 10 секунд или чаще, чем наступает половина интервала до `expires_at`;
- при `409` остановите обработку текущих событий: другой session владеет lease или текущий lease истек;
- после потери lease не завершайте старый stream и не отправляйте сообщение, пока не получите новый lease;
- при shutdown вызывайте `DELETE`;
- если сервис агента перезапустился, генерируйте новый `X-Bot-Session-ID` и новый `lease_id`.

14. API сервиса агента: разговоры и контекст

После lease сервис агента может читать доступные conversation-и и историю.

Точка доступа	**Метод**	**Назначение**
`/v1/bot-runtime/conversations`	`GET`	список разговоров, где участвует агент
`/v1/bot-runtime/conversations/<conversation_id>/messages?limit=20`	`GET`	последние сообщения
`/v1/bot-runtime/conversations/<conversation_id>/memo`	`GET`	markdown-мемо разговора

`/v1/bot-runtime/conversations/<conversation_id>/memo`	`PUT`	обновить markdown-мемо
`/v1/bot-runtime/attachments/<attachment_id>`	`GET`	получить вложение, доступное агенту

Все пять операций чтения в этой таблице требуют `X-Bot-Lease-ID` и совпадающего `X-Bot-Session-ID`.

Ответ списка сообщений:

```
{
  "messages": [
    {
      "id": "0195f7f2-message-example",
      "conversation_id": "0195f7f2-conversation-example",
      "sender_id": "0195f7f2-user-example",
      "sender_is_virtual": false,
      "body": "Здравствуйте, нужен расчет стоимости.",
      "parts": [],
      "attachments": [],
      "mentions": [],
      "reply_message": null,
      "created_at": "2026-04-20T12:00:00Z"
    }
  ],
  "next_cursor": {
    "before_id": "0195f7f2-message-example",
    "before_created_at": "2026-04-20T12:00:00Z"
  },
  "oldest_unread_message_id": ""
}
```

Ключевые поля сообщения:

Поле	**Тип**	**Назначение**
`id`	string	Идентификатор сообщения.
`conversation_id`	string	Разговор, в котором находится сообщение.
`sender_id`	string	Пользователь или virtual user, который отправил сообщение.
`sender_is_virtual`	boolean	`true`, если отправитель является virtual user. Сервис агента обычно игнорирует

		такие сообщения.
`body`	string	Видимый текст сообщения.
`parts`	array	Structured content parts: `text`, `tool_card`, `status`, `thinking` и другие будущие типы.
`attachments`	array	Метаданные вложений. Содержимое скачивается отдельно через `/v1/bot-runtime/attachments/<attachment_id>`.
`mentions`	array	Упоминания пользователей и virtual users.
`reply_message`	object/null	Сообщение, на которое был reply, если оно доступно агенту.
`created_at`	RFC3339 datetime	Время создания сообщения.

Полная схема `Message`, включая `content_kind`, данные отправителя, ответ и пересылку, реакции, состояние расшифровки, данные импорта и доставки, приведена в `docs/api/iskra-bot-runtime-openapi.yaml`. Сервис агента должен считать поля с `omitempty` необязательными и не предполагать, что массивы всегда присутствуют.

Схемы вложенных объектов:

Объект	**Поля текущей версии**
`parts[]`	`type` (обязательно), `text`, `data`. Известные клиентам типы включают `text`, `tool_card`, `status`, `thinking`; неизвестный тип нужно безопасно пропустить или обработать как данные.
`attachments[]` во входящем сообщении	`id`, `kind`, `filename`, `mime_type`, `size_bytes`, `url`, необязательный `thumbnail_url`. Содержимое следует получать через Точка чтения сообщений API сервиса агента по `id`, а не по клиентскому `url`.
`mentions[]`	`user_id`, `label`, `start`, `end`; в ответах также могут присутствовать `message_id`, `conversation_id`, `created_at`, `read_at`. `start` и `end` задают границы упоминания в тексте сообщения.
`reply_message`	Полный объект `Message`, если исходное сообщение доступно агенту; иначе поле отсутствует.

`GET /v1/bot-runtime/conversations` возвращает объект `{ "conversations": [...] }`. Полная схема элемента `ConversationSummary` приведена в OpenAPI. Для маршрутизации сервис агента должен использовать как минимум `id`, `type`, `status`, `space_id`, `role`, `peer_is_virtual`, `has_external_members`, `external_provider` и `service_state`; отображаемые `title`, `subtitle`, аватары и превью не являются устойчивыми идентификаторами.

Параметр `limit` принимает целое число от `1` до `100`; значение по умолчанию - `50`. Если передано нечисловое значение, `0`, отрицательное число или число больше `100`, Gateway 1.0 не возвращает ошибку и использует `50`.

Ограничение версии 1.0: ответ может содержать `next_cursor`, но Точка чтения сообщений API сервиса агента не принимает `before_id` или `before_created_at`. Получить следующую, более старую страницу через эту точку доступа нельзя. Для контекста используйте последние сообщения (рекомендуется `limit=20`) и memo. После длительного простоя сервис агента может сверить только последние `100` сообщений каждого доступного разговора; сценарии, которым требуется гарантированная полная история, должны хранить собственное состояние или получать историю через отдельно согласованный интеграционный интерфейс.

Memo удобно использовать как компактную память разговора. Если memo еще не создано, Искра использует базовую markdown-структуру:

```
{
  "markdown": "# Факты\n\n# Принятые решения\n\n# План\n\n# Полезное\n"
}
```

Стандартные разделы memo:

- `# Факты` - устойчивые факты из разговора: ограничения, предпочтения, исходные данные, важные параметры клиента или задачи.
- `# Принятые решения` - явно согласованные решения, выбранные варианты и зафиксированные договоренности.
- `# План` - список следующих действий. Этот раздел должен быть оформлен как markdown-checklist.
- `# Полезное` - справочные заметки, выдержки, ссылки, кодовые фрагменты, таблицы или другие материалы, которые полезно сохранить для дальнейшей работы.

Раздел `# План` должен содержать только задачи в формате checklist:

```
# План

- [ ] Уточнить регион и объем пользователей
- [ ] Подготовить расчет стоимости
- [x] Отправить первичное предложение
```

Правила для `# План`:

- открытая задача пишется как `- [ ] <действие>`;
- выполненная задача пишется как `- [x] <действие>`;
- формулировка должна начинаться с действия: "Уточнить", "Проверить", "Подготовить", "Исправить", "Согласовать";

- не храните в `# План` факты, рассуждения или открытые вопросы без действия;
- если сервис агента получает обычный список или нумерованный список в разделе `# План`, Искра нормализует его в checklist при серверной обработке memo.

Если стандартных разделов недостаточно, сервис агента может добавлять пользовательские top-level разделы markdown:

```
# Факты

- Клиент использует локальное внедрение.

# Интеграция с CRM

## Bitrix24

Портал подключается через серверное OAuth-приложение.

# Принятые решения

- Использовать отдельного агента первой линии.

# План

- [ ] Подготовить тестовый сценарий

# Полезное
```

Пользовательские разделы должны иметь заголовок первого уровня `# <Название раздела>`. Искра сохраняет такие разделы при нормализации memo. Не переименовывайте стандартные разделы, если хотите сохранить совместимость с UI и автоматическим добавлением фактов, решений, плана и полезных заметок.

Поддерживаемое форматирование memo:

Формат	**Как писать**	**Как отображается**
Заголовки	`#`, `##`, до `######`	Разделы и подразделы memo. Стандартные top-level разделы должны быть `# Факты`, `# Принятые решения`, `# План`, `# Полезное`.
Абзацы	Обычный текст с пустой строкой между блоками	Текстовые блоки memo.
Выделение	`**жирный**`, `__жирный__`, `*курсив*`, `_курсив_`, `` `код` ``	Inline-форматирование в web и мобильном просмотре memo.
Цитаты	`> текст`	Цитатный блок.

Списки	`- пункт`, `* пункт`, `1. пункт`	Маркированные или нумерованные списки.
Checklist	`- [ ] задача`, `- [x] задача`	Интерактивные задачи в разделе `# План`; пользователь может закрывать и переоткрывать пункты.
Таблицы	Markdown-таблица с разделителем `---`	Таблица с горизонтальной прокруткой при необходимости.
Кодовые блоки	Блок из трех обратных апострофов с языком, например `json`	Моноширинный блок кода.
Mermaid-диаграммы	Блок из трех обратных апострофов с языком `mermaid`	Диаграмма в просмотре memo, если тип диаграммы поддерживается.
Ссылки	`[текст](https://example.com)`	Кликабельная ссылка.

Для раздела `# Полезное` удобно использовать вложенные блоки `## <Название блока>`: UI показывает их как отдельные полезные заметки внутри раздела.

Рекомендации:

- не отправляйте весь длинный чат в LLM, если достаточно последних сообщений и memo;
- обновляйте memo после важных решений;
- добавляйте новые пункты в начало соответствующего стандартного раздела, чтобы свежий контекст был выше старого;
- не сохраняйте персональные данные во внешнем сервисе агента без отдельного основания;
- вложения скачивайте только когда они действительно нужны для ответа.

15. API сервиса агента: отправка сообщений

Простой ответ:

```
POST /v1/bot-runtime/conversations/<conversation_id>/messages
Authorization: Bearer <bot_token>
X-Bot-Session-ID: agent-worker-1
X-Bot-Lease-ID: agent-worker-1-lease
Content-Type: application/json
{
  "body": "Принято. Проверю информацию и вернусь с ответом.",
  "client_request_id": "8d24bcec-0b11-4df4-b510-dca3e7598104"
}
```

Тело запроса:

Поле	Тип	Обязательность	Назначение
`body`	string	Да, если `parts` пустой	Текст сообщения в Markdown.
`parts`	array	Нет	Structured content parts. Для обычных ответов используйте `body`; для auth-card и расширенного UI используйте stream completion с `parts`.
`client_request_id`	string	Настоятельно рекомендуется	Устойчивый идентификатор одной логической отправки. При повторе после сетевой ошибки передавайте то же значение. Для нового ответа создавайте новое значение.
`mentions`	array	Нет	Упоминания пользователей/агентов, если сообщение должно явно упомянуть участника.

API сервиса агента 1.0 не предоставляет процедуру загрузки исходящих вложений и не выдает сервису агента идентификатор подготовленного вложения. Поэтому внешнему сервису агента нельзя формировать `attachments` самостоятельно. Поле присутствует во внутренней модели отправки, но не входит в поддерживаемый внешний сценарий. Для передачи файла требуется отдельно согласованный серверный интеграционный механизм; до его появления отправляйте ссылку с проверенными правами доступа или текстовое сообщение.

Для `mentions` передавайте `user_id`, `label`, `start`, `end`. Смещения должны соответствовать позиции текста упоминания в `body`; не создавайте упоминание только визуальным `@именем` без объекта `mentions`.

Успешный ответ возвращает созданное сообщение Искры. Сервис агента должен сохранить `id` сообщения, если использует собственную дедупликацию или связывает ответ с внешним действием.

Форматирование сообщений

Поле `body` содержит текст сообщения в Markdown. Web-клиент Искры отображает GitHub Flavored Markdown, мобильный клиент поддерживает базовое inline-форматирование сообщений и отдельный просмотр memo с расширенным Markdown.

Поддерживаемые варианты для сообщений агента:

Формат	Синтаксис	Примечание
Абзацы и переносы строк	Пустая строка между абзацами	Для обычного текста ответа.
Жирный	`**текст**` или `__текст__`	Inline-выделение.
Курсив	`*текст*` или `_текст_`	Не используйте `_` внутри технических идентификаторов как форматирование.
Зачеркнутый текст	`~~текст~~`	Inline-выделение.
Inline-код	` `value` `	Для ключей, URL-фрагментов, идентификаторов и коротких команд.
Ссылки	`[текст](https://example.com)` или bare `https://example.com`	Поддерживаются внешние `http/https` ссылки.
Заголовки	`#`, `##`, до `######`	Используйте умеренно; для коротких ответов лучше обычный текст.
Списки	`- пункт`, `* пункт`, `1. пункт`	Поддерживаются маркированные и нумерованные списки.
Цитаты	`> текст`	Используются также для ссылок на исходные сообщения.
Таблицы	Markdown-таблица	Web-клиент показывает таблицу в прокручиваемом контейнере.
Кодовые блоки	Блок из трех обратных апострофов с языком, например `json`	Для JSON, SQL, YAML, логов и примеров кода.
Mermaid-диаграммы	Блок из трех обратных апострофов с языком `mermaid`	Web-клиент рендерит поддерживаемые Mermaid-диаграммы.

Для таблиц используйте обычную Markdown-таблицу:

```
| Параметр | Значение |
| --- | --- |
| Статус | active |
| Модель | default |
```

Для диаграмм используйте только fenced block с языком `mermaid`. Пример содержимого блока:

```
flowchart TD
  A[Сообщение пользователя] --> B[Сервис агента]
  B --> C[Ответ в Искру]
```

Поддерживаемые типы Mermaid-диаграмм: `flowchart`, `graph`, `sequenceDiagram`, `classDiagram`, `stateDiagram`, `stateDiagram-v2`, `erDiagram`, `journey`, `gantt` `pie`, `quadrantChart`, `requirementDiagram`, `gitGraph`, `mindmap`, `timeline`, `sankey-beta`, `xychart-beta`, `xychart`, `C4Context`, `C4Container`, `C4Component`, `C4Dynamic`, `C4Deployment`, `block-beta`, `architecture-beta`, `kanban`, `packet-beta`, `radar-beta`, `treemap-beta`, `venn`, `info`, `zenuml`, `wardley`, `ishikawa`.

Не используйте PlantUML, ASCII-art или неподдерживаемые псевдотипы вроде `componentDiagram`. Если нужна таблица, используйте Markdown-таблицу, а не Mermaid.

Для LLM-ответов лучше использовать потоковый ответ: пользователь сразу видит, что агент работает, а финальный результат становится обычным сообщением.

16. API сервиса агента: потоковые ответы

Точка доступа	Метод	Назначение
`/v1/bot-runtime/conversations/<conversation_id>/streams`	`POST`	начать потоковый ответ
`/v1/bot-runtime/streams/<stream_id>`	`PATCH`	добавить фрагмент текста
`/v1/bot-runtime/streams/<stream_id>`	`POST`	завершить потоковый ответ и создать сообщение
`/v1/bot-runtime/streams/<stream_id>`	`DELETE`	пометить потоковый ответ как ошибочный

Начало:

```
{
  "body": ""
}
```

Успешный ответ возвращает `MessageStream`:

```
{
  "stream_id": "0195f7f2-stream-example",
  "conversation_id": "0195f7f2-conversation-example",
  "sender_user_id": "0195f7f2-bot-user",
  "state": "started",
  "accumulated_content": "",
  "parts": [],
  "created_at": "2026-04-20T12:00:00Z",
  "updated_at": "2026-04-20T12:00:00Z"
}
```

Добавление фрагмента:

```
{
  "delta": "Собираю контекст...",
  "part_type": "text"
}
```

Завершение:

```
{
  "body": "Я проверил информацию. Для вашего случая подойдет тариф Business, но нужно уточнить количество операторов."
}
```

Если сервис агента использует structured parts, завершение должно передать полный финальный массив `parts`, потому что финальное сообщение создается из complete payload:

```
{
  "parts": [
    {
      "type": "text",
      "text": "Я проверил информацию. Для вашего случая подойдет тариф Business."
    }
  ]
}
```

Ошибка:

```
DELETE /v1/bot-runtime/streams/<stream_id>
```

Жизненный цикл stream:

Состояние	Как появляется	Что видит пользователь
`started`	`POST /conversations/<id>/streams`	Начался потоковый ответ агента.
`delta`	`PATCH /streams/<stream_id>`	Пользователь видит добавленный текст или карточку.
`completed`	`POST /streams/<stream_id>`	Stream превращается в обычное сообщение Искры.
`failed`	`DELETE /streams/<stream_id>` или внутренняя ошибка	Потоковый ответ помечается как не завершенный.
`cancelled`	Handoff или внутренний сценарий отмены	Stream скрывается или отменяется без финального текста.

Правила:

- содержимое потокового ответа только добавляется;
- упоминания проверяются при завершении, а не на каждом фрагменте;

- завершенный потоковый ответ становится обычным сообщением Искры;
- после `completed`, `failed` или `cancelled` сервис агента не должен продолжать `PATCH` того же `stream_id`;
- поле `attachments` в запросе завершения stream технически присутствует в JSON-модели, но Gateway 1.0 отклоняет непустой массив; исходящие вложения через API сервиса агента не поддерживаются;
- при ошибке LLM/provider вызывайте `DELETE`, а затем при необходимости отправьте короткое обычное сообщение с извинением; для агента клиентской поддержки можно передать обращение оператору.

17. OAuth-авторизация пользователя в чате

Если агенту нужен доступ к внешней системе от имени пользователя, не просите пользователя вставлять token, code или cookie в чат. Правильный сценарий - показать в чате кнопку авторизации, открыть OAuth-страницу поставщика и сохранить полученные token на серверной стороне.

В текущей реализации Искры встроенный assistant использует этот механизм для:

Провайдер	Где используется	Что видит пользователь
Bitrix24	`bitrix24-calendar`, `bitrix24-task`, `bitrix24-funnel`	Карточка "Нужно подключить Bitrix24" с кнопкой подключения.
Google Calendar	`google-calendar`	Карточка подключения календаря.
Yandex Calendar	`yandex-calendar`	Карточка подключения календаря.
Apple Calendar	`apple-calendar` через CalDAV-авторизацию	Форма подключения календаря в окне авторизации.
HeadHunter	`headhunter-resume-search`	Карточка "Нужно подключить HeadHunter" с кнопкой подключения.

17.1 Поток авторизации

1. Пользователь просит агента выполнить действие, которое требует внешней учетной записи.
2. Сервис агента проверяет, есть ли у пользователя активное подключение для нужного провайдера.
3. Если подключения нет или token устарел, сервис агента создает короткоживущую OAuth-сессию на своем сервере.
4. Сервис агента отправляет в потоковый ответ часть `tool_card` с `kind=auth_required` и `auth_url`.

5. Клиент Искры показывает карточку с кнопкой.
6. Пользователь нажимает кнопку и проходит OAuth у внешнего провайдера.
7. Callback внешнего провайдера приходит на сервер сервиса агента или assistant.
8. Сервер сервиса агента обменивает authorization code на token, сохраняет token в серверном хранилище секретов и связывает подключение с пользователем и агентом.
9. После успешной авторизации агент продолжает сценарий: повторяет проверку подключения и выполняет исходный tool call.

17.2 Формат auth-card в сообщении агента

Auth-card передается как часть потокового ответа. Web-клиент Искры распознает `part_type=tool_card` и показывает карточку действия.

Пример добавления карточки в stream:

```
PATCH /v1/bot-runtime/streams/<stream_id>
Authorization: Bearer <bot_token>
X-Bot-Session-ID: agent-worker-1
X-Bot-Lease-ID: agent-worker-1-lease
Content-Type: application/json
{
  "delta": "",
  "part_type": "tool_card",
  "part_data": {
    "kind": "auth_required",
    "provider": "bitrix24",
    "title": "Нужно подключить Bitrix24",
    "message": "Чтобы агент мог работать с вашим Bitrix24, авторизуйте Bitrix24 для этого агента.",
    "auth_url": "https://im.iskra-messenger.ru/v1/bitrix-auth?session=<session_id>&token=<session_token>",
    "action_label": "Подключить Bitrix24"
  }
}
```

После отправки карточки сервис агента обычно добавляет короткий текстовый фрагмент и завершает stream:

```
{
  "body": "Чтобы продолжить, подключите Bitrix24.",
  "parts": [
    {
      "type": "tool_card",
      "data": {
        "kind": "auth_required",
        "provider": "bitrix24",
        "title": "Нужно подключить Bitrix24",
```

```
        "message": "Чтобы агент мог работать с вашим Bitrix24, авторизуйте Bitrix24 для этого агента.",
        "auth_url": "https://im.iskra-messenger.ru/v1/bitrix-auth?session=<session_id>&token=<session_token>",
        "action_label": "Подключить Bitrix24"
      }
    },
    {
      "type": "text",
      "text": "Чтобы продолжить, подключите Bitrix24."
    }
  ]
}
```

Поля карточки:

Поле	**Назначение**
`kind`	Для OAuth используйте `auth_required`.
`provider`	Технический идентификатор провайдера: например `bitrix24`, `google-calendar`, `yandex-calendar`, `apple-calendar`, `headhunter-resume-search`.
`title`	Заголовок карточки в чате.
`message`	Короткое объяснение, зачем нужна авторизация.
`auth_url`	Серверная ссылка на начало OAuth-сессии. Ссылка должна быть одноразовой или короткоживущей.
`action_label`	Текст кнопки.

Матрица поддержки auth-card:

Возможность	**Текущий статус**
Отображение `tool_card` с `kind=auth_required` в web-клиенте	Поддерживается для сообщений и stream parts.
Открытие `auth_url` из карточки	Поддерживается: web-клиент открывает popup или iframe fallback.
Автоматическое закрытие окна после callback	Поддерживается только для callback-событий, известных клиенту.
Автоматический повтор последнего сообщения пользователя	Поддерживается только для `calendar-auth-complete`, `bitrix-auth-complete`, `headhunter-auth-complete`.

Произвольный OAuth-провайдер внешнего сервиса агента без доработки Искры	Карточка и кнопка поддерживаются, автоматическое возобновление диалога не гарантируется.

17.3 Callback и продолжение диалога

Для встроенных интеграций assistant callback уже реализован в Искре:

Интеграция	Начало авторизации	Callback
Bitrix24	`/v1/bitrix-auth`	`/v1/bitrix-auth/callback`
Google/Yandex/Apple Calendar	`/v1/calendar-auth/<provider>`	`/v1/calendar-auth/<provider>/callback` для OAuth-провайдеров; CalDAV-провайдеры отправляют форму на тот же путь авторизации.
HeadHunter	`/v1/headhunter-auth`	`/v1/headhunter-auth/callback`

После успешного callback встроенный web-клиент закрывает окно авторизации и повторно отправляет последнее сообщение пользователя. Это позволяет агенту снова обработать тот же запрос, увидеть уже активное подключение и выполнить tool call.

Для пользовательского сервиса агента в on-premise контуре есть два поддерживаемых варианта:

1. Сервис агента показывает auth-card со своей `auth_url`, выполняет OAuth callback на своей стороне, сохраняет token у себя и после callback просит пользователя вернуться в чат и повторить запрос.
2. Сервис агента интегрируется с серверной частью Искры как кастомный агентский контроллер и реализует тот же продуктовый сценарий: серверная OAuth-сессия, callback, хранение подключения, затем продолжение исходного запроса.

Не рассчитывайте, что произвольный внешний OAuth callback автоматически возобновит диалог в web-клиенте без доработки серверной интеграции. Автоматическое закрытие окна и повтор последнего сообщения сейчас реализованы для встроенных типов завершения авторизации: `calendar-auth-complete`, `bitrix-auth-complete`, `headhunter-auth-complete`.

17.4 Правила безопасности

- Не отправляйте access token, refresh token, authorization code, client secret или cookie в сообщениях Искры.
- `auth_url` должен указывать на сервер сервиса агента или assistant, а не прямо на финальный token endpoint.
- OAuth session должна иметь случайные `session_id` и `session_token`, срок жизни не более 15 минут и проверку соответствия пользователя.
- Token хранится только на сервере сервиса агента или assistant в зашифрованном виде.

- Подключение должно быть привязано к пользователю Искры, провайдеру, внешнему account id и конкретному агенту или набору tools.
- Пользователь должен иметь возможность посмотреть и отозвать OAuth-подключение в профиле, если подключение хранится в Искре.

17.5 OAuth для инструмента пользовательского сервиса агента

Пользовательский сервис агента может авторизовать пользователя и вызвать OAuth-инструмент самостоятельно. Он не получает и не использует OAuth-подключения, созданные встроенным сервисом `assistant`. Выбор инструмента `google-calendar`, `bitrix24-task` или `headhunter-resume-search` также не передает сервису агента `client_id`, `client_secret`, redirect URI или код выполнения API поставщика.

Команда, эксплуатирующая пользовательский сервис агента, до включения инструмента настраивает:

Параметр	Где хранится
`client_id`, `client_secret`	В серверном хранилище секретов сервисе агента; не в карточке агента и не в сообщении.
Authorization endpoint, token endpoint	В конфигурации адаптера конкретного провайдера.
Redirect URI	HTTPS-адрес callback пользовательского сервиса агента, зарегистрированный у провайдера.
Scopes	Минимальный набор прав для поддерживаемых операций инструмента.
Ключ шифрования токенов	В хранилище секретов сервисе агента, отдельно от базы подключений.

При запросе инструмента сервис агента выполняет следующую процедуру:

1. Из исходного `message.created` получить `sender_id`, `conversation_id`, `event_id` и идентификатор агента `virtual_user_id`.
2. В собственной базе найти активное подключение по составному ключу `virtual_user_id + sender_id + provider`. Проверить scopes и срок действия access token; при необходимости обновить token серверным refresh-запросом.
3. Если подключения нет, создать одноразовую OAuth-сессию. Сохранить случайный `state`, хеш одноразового session token, `virtual_user_id`, `sender_id`, `conversation_id`, `tool_key`, проверенные аргументы ожидающего вызова и срок действия не более 15 минут. Для провайдеров с PKCE сохранить `code_verifier` и передать `code_challenge`.
4. Сформировать `auth_url` на сервере сервиса агента. Ссылка из чата сначала открывает сервис агента, который проверяет одноразовую сессию и перенаправляет пользователя на authorization endpoint поставщика.
5. Отправить `tool_card` с `kind=auth_required` по формату раздела 17.2 и завершить текущий stream без заявления об успешном выполнении инструмента.

6. В callback проверить `state`, срок жизни сессии, одноразовость, redirect URI и PKCE. Обменять authorization code на токены только с сервера сервиса агента.
7. Зашифровать access/refresh token и сохранить подключение с привязкой к `virtual_user_id`, `sender_id`, provider, внешнему account id и scopes. Пометить OAuth-сессию использованной.
8. Поставить продолжение ожидающего вызова во внутреннюю устойчивую очередь сервиса агента. Callback-процесс не должен писать в чат напрямую, если он не владеет действующим lease агента.
9. Активный worker с lease повторно проверяет подключение, выполняет сохраненный вызов инструмента и отправляет результат в исходный `conversation_id`. Если автоматическое продолжение в сервисе агента не реализовано, карточка должна попросить пользователя вернуться в чат и повторить запрос.

Пример записи ожидающей OAuth-сессии во внутреннем хранилище сервиса агента:

```
{
  "state_hash": "<sha256(state)>",
  "virtual_user_id": "0195f7f2-bot-user",
  "requester_user_id": "0195f7f2-user-example",
  "conversation_id": "0195f7f2-conversation-example",
  "tool_key": "google-calendar",
  "pending_arguments": {
    "operation": "list_events",
    "time_min": "2026-07-15T00:00:00+03:00",
    "time_max": "2026-07-16T00:00:00+03:00"
  },
  "expires_at": "2026-07-15T12:15:00Z",
  "used": false
}
```

После callback сервис агента вызывает API провайдера напрямую. Он не вызывает endpoint встроенного инструмента Искры, потому что общего endpoint выполнения стандартных инструментов в версии 1.0 нет. Если заказчику требуется единая реализация инструмента для нескольких пользовательских типов агентов, поставьте ее как отдельный MCP-сервер и назначьте этот MCP-сервер соответствующим агентам. Встроенный `basic_assistant` в версии 1.0 произвольные MCP-инструменты не выполняет; ограничение описано в разделе 9.1.

18. WebSocket API сервиса агента

Точка доступа:

```
GET /v1/bot-runtime/ws
```

Пример:

```
wss://im.iskra-messenger.ru/v1/bot-runtime/ws?token=<bot_token>&session_id=agent-worker-1&lease_id=agent-worker-1-lease&last_event_id=<event_id>
```

Важные типы событий:

- `message.created`;

- `message.updated`;
- `message.stream.started`;
- `message.stream.delta`;
- `message.stream.completed`;
- `message.stream.failed`;
- `conversation.created`;
- `conversation.updated`;
- `conversation.read_updated`;
- `typing.updated`;
- события звонков и конференций, если агент участвует в таких сценариях.

Гарантии доставки:

Свойство	**Контракт**
Порядок	Сервис агента должен обрабатывать события в порядке получения внутри одного WebSocket-соединения. Абсолютный глобальный порядок между reconnect не гарантируется; используйте `occurred_at` и перечитывание REST-контекста при разрывах.
Повторная доставка	Событие может прийти повторно после reconnect или сетевой ошибки. Сервис агента обязан дедуплицировать по `event_id`.
Пропуски	Повторная доставка является best effort. Gateway читает не более `500` событий после `last_event_id`. Если курсор отсутствует или событий больше лимита, протокол 1.0 не сообщает сервису агента о неполноте. После каждого reconnect сервис агента обязан сверить состояние через REST.
Фильтрация	Gateway отправляет только события, доступные virtual user агента, но сервис агента все равно должен фильтровать собственные сообщения и нежелательные virtual-user сообщения.
Lease	WebSocket должен открываться с действующим `lease_id`. После потери lease сервис агента закрывает соединение и получает новый lease.

`last_event_id` передается как последний успешно обработанный `event_id`. Сохраняйте его только после успешного завершения обработчика события или после надежной записи события в

собственную устойчивую очередь. После reconnect Gateway пытается доставить до `500` событий, которые произошли после этого события. Gateway 1.0 не отправляет признак `cursor_missing`, `truncated` или иной статус полноты. Поэтому сервис агента не должен считать `last_event_id` заменой чтения истории: после каждого разрыва перечитайте список разговоров, memo и последние сообщения. Из-за отсутствия постраничного чтения такая сверка через API ограничена последними `100` сообщениями разговора.

Схема конверта WebSocket:

Поле	Тип	Обязательность	Назначение
`type`	string	Да	Тип события. Неизвестное значение нужно записать в журнал и пропустить.
`event_id`	string	Для сохраненных событий	Идентификатор дедупликации и курсор best-effort replay. Некоторые оперативные события могут не иметь устойчивого идентификатора.
`conversation_id`	string	Для событий разговора	Разговор, к которому относится событие.
`occurred_at`	RFC3339 datetime	Да	Время формирования события Gateway.
`payload`	object	Зависит от типа	Для `message.*` используется объект `Message` или `MessageStream`; для `conversation.*` - объект разговора/изменения; сервис агента должен игнорировать неизвестные поля.

Машинные схемы `RealtimeEvent`, `Message` и `MessageStream` находятся в OpenAPI-файле. WebSocket использует те же JSON-схемы, хотя сам переход протокола описывается этим разделом, а не HTTP-операцией OpenAPI.

Пример `message.created`:

```
{
  "event_id": "0195f7f2-event-example",
  "type": "message.created",
```

```
  "conversation_id": "0195f7f2-conversation-example",
  "occurred_at": "2026-04-20T12:00:00Z",
  "payload": {
    "id": "0195f7f2-message-example",
    "conversation_id": "0195f7f2-conversation-example",
    "sender_id": "0195f7f2-user-example",
    "sender_is_virtual": false,
    "body": "Здравствуйте, нужен расчет стоимости.",
    "created_at": "2026-04-20T12:00:00Z"
  }
}
```

Предотвращение циклов:

- игнорируйте события, где `sender_id` равен `virtual_user_id` агента;
- по умолчанию игнорируйте `sender_is_virtual=true`;
- не запускайте bot-to-bot автодиалоги без отдельной политики.

Повторная доставка:

- сохраняйте `event_id` только после успешной обработки или надежной постановки события в очередь;
- при переподключении передавайте `last_event_id`;
- после каждого переподключения перечитывайте разговоры, memo и последние сообщения через REST независимо от количества повторно полученных событий.

19. Tool requests и действия с подтверждением

Tool request нужен, когда агент хочет выполнить действие, которое требует подтверждения человека или внешнего обработчика.

Точка доступа	Метод	Назначение
`/v1/bot-runtime/conversations/<conversation_id>/tool-requests`	`POST`	создать tool request
`/v1/bot-runtime/tool-requests/<request_id>`	`GET`	проверить состояние tool request
`/v1/agent-tool-requests/<request_id>`	`POST`	завершить request пользовательским или административным API

Создать request:

```
{
  "tool_key": "create_invoice",
  "request_payload": {
```

```
    "customer_id": "cust-123",
    "amount": 15000,
    "currency": "RUB"
  }
}
```

Завершить request:

```
POST /v1/agent-tool-requests/<request_id>
Authorization: Bearer <user_access_token>
Content-Type: application/json
{
  "result_payload": {
    "invoice_id": "inv-456"
  }
}
```

Если действие отклонено или завершилось ошибкой, передайте `error_message`:

```
{
  "error_message": "Пользователь отклонил создание счета."
}
```

Ответ создания или проверки tool request:

```
{
  "request_id": "0195f7f2-request-example",
  "conversation_id": "0195f7f2-conversation-example",
  "virtual_user_id": "0195f7f2-bot-user",
  "tool_key": "create_invoice",
  "status": "pending",
  "request_payload": {
    "customer_id": "cust-123",
    "amount": 15000,
    "currency": "RUB"
  },
  "expires_at": "2026-04-20T12:02:00Z",
  "created_at": "2026-04-20T12:00:00Z",
  "updated_at": "2026-04-20T12:00:00Z"
}
```

Жизненный цикл tool request:

Статус	**Как появляется**	**Что должен делать сервис агента**
`pending`	После `POST /tool-requests`	Периодически читать `/v1/bot-runtime/tool-requests/<request_id>` до итогового состояния или истечения срока.

`completed`	Пользователь или обработчик завершил request с `result_payload` без `error_message`	Прочитать `result_payload` и продолжить разговор.
`failed`	Завершение с `error_message`	Сообщить пользователю, что действие не выполнено, и не утверждать успех.

В текущей реализации новый tool request имеет TTL 2 минуты. После завершения request хранится еще 10 минут, чтобы сервис агента успел прочитать результат.

Gateway 1.0 отправляет событие `agent.tool.requested` участникам разговора, кроме самого агента. Завершение request обновляет хранилище, но событие `agent.tool.completed` агенту не отправляется. Универсальная карточка подтверждения в клиенте для произвольного `tool_key` контрактом не гарантируется. Следовательно, сервис агента обязан получать результат polling-запросами; ожидание события завершения приведет к зависанию сценария.

Права завершения: `POST /v1/agent-tool-requests/<request_id>` вызывается от имени пользователя, который является участником conversation. Сервис агента не должен завершать request за пользователя через bot credential.

Рекомендации:

- используйте tool requests для платежей, изменения CRM, записи в календарь, отправки внешних писем и других действий с последствиями;
- показывайте пользователю понятное описание действия до подтверждения;
- сервис агента должен polling-запросами проверить итог request и продолжить разговор;
- не выполняйте irreversible action только на основании текста LLM.

20. Handoff оператору

Handoff сейчас предназначен для агентов клиентской поддержки (customer support agents), которые работают с клиентскими обращениями и передают обращение сотруднику или оператору поддержки. Используйте handoff, если support-агент не уверен, не имеет права выполнить действие или клиент просит человека.

Для виртуальных пользователей в групповых чатах handoff сейчас не используется. В текущей реализации передача группового чата от virtual user конкретному сотруднику не поддерживается, поэтому сервис агента такого агента должен отвечать в рамках своих прав, попросить участника явно привлечь нужного сотрудника или остановить автоматический ответ. Поддержка handoff для групповых сценариев планируется в будущих версиях Искры.

```
POST /v1/bot-runtime/conversations/<conversation_id>/handoff
Authorization: Bearer <bot_token>
X-Bot-Session-ID: agent-worker-1
X-Bot-Lease-ID: agent-worker-1-lease
```

Перед handoff в customer support сценарии полезно:

- обновить memo с краткой сводкой;

- отправить клиенту короткое сообщение, что обращение передано оператору;
- не продолжать отвечать в этом conversation, пока сервис агента не получит новую явную политику.

21. Рекомендуемый цикл работы сервиса агента

Псевдокод:

```
load agent config
create session_id
acquire lease
start lease renewal loop
connect websocket with last_event_id

for each event:
  if event.type != message.created:
    commit event_id after successful skip
    continue
  if event.sender_is_virtual or event.sender_id == virtual_user_id:
    commit event_id after successful skip
    continue
  messages = GET /conversations/{id}/messages?limit=20
  memo = GET /conversations/{id}/memo
  stream = POST /conversations/{id}/streams
  try:
    for delta in llm(prompt, memo, messages):
      PATCH /streams/{stream_id}
    POST /streams/{stream_id}
    update memo if needed
  except recoverable_error:
    DELETE /streams/{stream_id}
    maybe send short failure message
  except needs_human_for_support_agent:
    DELETE /streams/{stream_id}
    update memo
    POST /conversations/{id}/handoff
  after successful processing or durable enqueue:
    commit event_id
```

22. Ошибки, retry и совместимость

Рекомендуемая обработка:

Код	Что делать сервис агента
`400`	исправить payload, не повторять без

	изменения
`401`	bot token неверный или отозван; остановить сервис агента и запросить rotation
`403`	нет доступа или lease не подходит; перечитать config/lease
`404`	conversation/request/stream недоступен агенту
`409`	конфликт lease или состояния; backoff и reacquire
`429`	уважать rate limit/cooldown
`5xx`	retry с exponential backoff и jitter

Минимальный набор машинных кодов ошибок, которые сервис агента должен различать:

`code`	**Обработка**
`invalid_bot_credential`	Остановить сервис агента, не пытаться бесконечно переподключаться, запросить ротацию credential.
`bot_lease_unavailable`	Остановить текущую обработку, закрыть WebSocket, получить новый lease после backoff.
`message_stream_not_found`	Не продолжать stream; перечитать conversation и при необходимости начать новый ответ.
`agent_tool_request_not_found`	Считать request недоступным или истекшим; не выполнять действие повторно без нового request.
`invalid_agent_tool_request`	Исправить payload или остановить сценарий с понятным сообщением пользователю.

Ошибки по операциям API сервиса агента 1.0:

Операция	**Успех**	**Ожидаемые ошибки контракта**
`POST /leases`	`201`	`400 invalid_json`, `400 invalid_lease_id`, `401 invalid_bot_credential`, `409 bot_lease_unavailable`
`PATCH`, `DELETE /leases/<id>`	`204`	`401 invalid_bot_credential`, `404 not_found` для пустого маршрута, `409`

		`bot_lease_unavailable`, `405 method_not_allowed`
`GET /conversations`	`200`	`401 invalid_bot_credential`, `409 bot_lease_unavailable`, `405 method_not_allowed`
`GET /conversations/<id>/messages`	`200`	`401 invalid_bot_credential`, `403 forbidden`, `404 not_found`, `409 bot_lease_unavailable`
`POST /conversations/<id>/messages`	`201`	`400 invalid_json`, доменная ошибка `400`, `403 forbidden`, `404 not_found`, `409 bot_lease_unavailable`
`GET`, `PUT /conversations/<id>/memo`	`200`	`400 invalid_json` для PUT, `403 forbidden`, `404 not_found`, `409 bot_lease_unavailable`
`GET /attachments/<id>`	двоичный ответ	`401 invalid_bot_credential`, `404 not_found`, `409 bot_lease_unavailable`
`POST /conversations/<id>/streams`	`201`	`400 invalid_json`, `403 forbidden`, `404 not_found`, `409 bot_lease_unavailable`
`PATCH`, `POST /streams/<id>`	`200`	`400 invalid_json`, `400` при исходящем вложении, `403 forbidden`, `404 message_stream_not_found`, `409 bot_lease_unavailable`
`DELETE /streams/<id>`	`204`	`403 forbidden`, `404 message_stream_not_found`, `409 bot_lease_unavailable`
`POST /conversations/<id>/tool-requests`	`201`	`400 invalid_json`, `400 invalid_agent_tool_request`, `403 forbidden`, `404 not_found`, `409 bot_lease_unavailable`
`GET /tool-requests/<id>`	`200`	`403 forbidden`, `404 agent_tool_request_not_f`

		`ound`, `409 bot_lease_unavailable`
`POST /conversations/<id>/handoff`	`200`	`400 invalid_service_request`, `403 forbidden`, `404 not_found`, `409 bot_lease_unavailable`

Во всех операциях возможны `405 method_not_allowed`, инфраструктурный `500 internal_error` и ответ ограничителя частоты `429`. Поле `message` предназначено для диагностики и может меняться; программная логика должна опираться на HTTP-статус и `code`.

Сохраняйте forward compatibility:

- неизвестные event types логируйте и пропускайте;
- неизвестные поля в payload не должны ломать сервис агента;
- новые visibility/trigger values обрабатывайте как unsupported, но безопасно;
- не делайте бесконечные retry loops по одному сообщению.

23. Безопасность

- Bot token храните только на backend-стороне.
- Не передавайте bot token в web/mobile клиент.
- Не записывайте в журналы полный `bot_token`, prompt с секретами, OAuth tokens и payload внешних инструментов.
- Для tool requests проверяйте права человека, который подтверждает действие.
- Для OAuth не принимайте token через текст чата; используйте только серверную authorization-ссылку и callback.
- Внешние MCP/tool endpoints должны иметь собственную аутентификацию.
- Ограничивайте размер контекста, вложений и tool payload.
- Для private agents используйте список разрешенных пользователей и настройки видимости, а не только скрытие в интерфейсе.

24. Чек-лист готовности агента

- Агент создан администратором в правильном пространстве.
- Видимость соответствует назначению агента.
- Bot credential сохранен в серверном хранилище секретов.
- Credential rotation протестирован.
- Сервис агента получает и продлевает lease.

- WebSocket reconnect передает `last_event_id`, а после каждого разрыва сервис агента сверяет разговоры, memo и последние сообщения через REST.
- Сервис агента игнорирует свои сообщения и другие virtual-user сообщения.
- Ответы отправляются через stream endpoints.
- OAuth-сценарии используют auth-card, серверный callback и серверное хранение token; token не попадают в сообщения и журналы.
- Сервис агента различает `invalid_bot_credential`, `bot_lease_unavailable`, `message_stream_not_found`, `agent_tool_request_not_found`.
- Сервис агента имеет дедупликацию по `event_id` и не отправляет дубль после сетевого таймаута.
- `event_id` фиксируется только после успешной обработки или надежной постановки события в очередь.
- Отправка сообщения использует устойчивый `client_request_id`, который сохраняется до получения ответа.
- Ошибки LLM/provider закрывают stream как failed.
- Memo обновляется после важных решений.
- Tool requests используются для действий с последствиями.
- Результат tool request читается polling-запросами; сервис агента не ожидает событие завершения.
- Пользовательский сервис агента разрешает только инструменты из актуального массива `tools` и имеет собственные схемы аргументов и обработчики для каждого поддерживаемого `key`.
- Пользовательский сервис агента применяет массив `mcp_servers` как полный снимок, проверяет URL по списку разрешений и не сохраняет снятые назначения.
- MCP-клиент обрабатывает инициализацию, постраничный `tools/list`, `tools/call`, ошибки инструмента и закрытие сессии; модели доступны только явно разрешенные имена.
- Приемочная проверка MCP проводится на пользовательском типе агента; назначение MCP-сервера базовому `basic_assistant` не считается проверкой выполнения инструмента в версии 1.0.
- Для OAuth-инструментов пользовательский сервис агента имеет собственное OAuth-приложение, связывает подключение с `sender_id` и не рассчитывает на токены встроенного `assistant`.
- Callback OAuth не пишет в API сервиса агента без действующего lease; продолжение передается активному worker через устойчивую очередь или пользователь повторяет запрос.
- Для customer support агента handoff оператору работает; для group-chat virtual users handoff не используется.

- Журналы сервиса агента не содержат token, персональные данные без необходимости и raw tool secrets.
- Есть alert на потерю lease, частые `5xx`, частые failed streams и рост handoff rate для customer support агентов.

25. Пакет контракта и приемочная проверка

В поставку документации для разработчика входят:

Артефакт	Назначение
`docs/api/iskra-bot-runtime-openapi.yaml`	Машинно-читаемое описание HTTP-путей, заголовков, тел запросов, ответов и основных JSON-схем API сервиса агента 1.0.
`docs/api/conformance/README.md`	Порядок локальной проверки схемы и обязательные проверки совместимости сервиса агента.
`docs/api/conformance/examples/`	Эталонные примеры lease, списка сообщений, события WebSocket, сообщения с `client_request_id` и tool request.

До подключения к промышленному пространству разработчик выполняет приемочную проверку на тестовом агенте: получает и продлевает lease, читает контекст с lease-заголовком, обрабатывает повторное событие без дубля, повторяет отправку с тем же `client_request_id`, восстанавливается после разрыва WebSocket, завершает и аварийно закрывает stream, опрашивает tool request до итогового состояния и подтверждает, что токены не попадают в журналы. Результаты проверки передаются владельцу локального внедрения Искры вместе с версией сервиса агента и OpenAPI-контракта.