# Документация Just AI Agent Platform. Руководство пользователя

# О Just AI Agent Platform

Just AI Agent Platform от Just AI — это единая платформа для создания AI-агентов. С ее помощью вы сможете:

- **Создавать умных помощников**, которые общаются с клиентами в чате и голосом.
- **Автоматизировать бизнес-процессы**, доверяя рутинные задачи AI-агентам.
- **Строить мультиагентные системы** для решения масштабных задач.

### Быстрый старт

Начните работу с Just AI Agent Platform и создайте свой первый проект

# Быстрый старт

В этой статье вы узнаете, как использовать триггеры, AI-агента и функции в Just AI Agent Platform.

> **БЕТА**
>
> Платформа Just AI Agent Platform выпущена в бета-версии. Интерфейс и функции могут меняться, мы будем регулярно обновлять эту статью.

В качестве примера мы создадим процесс для работы с новостями про космос. Процесс позволит:

- Запрашивать статьи на нужную тему.
- Регулярно получать последние новости.
- Просить агента составить краткий обзор сообщений в чате.

Чтобы начать работу:

1. Создайте проект.
2. Настройте поиск статей через AI-агента.
3. Настройте регулярную отправку последних новостей в чат.

# Шаг 1. Создание проекта

1. Зарегистрируйтесь в Just AI Agent Platform.

   > **К СВЕДЕНИЮ**
   >
   > Just AI Agent Platform и Conversational Cloud используют общую базу аккаунтов, поэтому если вы зарегистрированы в Conversational Cloud, дополнительно регистрироваться в Just AI Agent Platform не нужно.

2. На странице *Проекты* нажмите *Создать AI-приложение*.
3. Выберите шаблон *Пустой проект* и укажите название.

   > **ПОДСКАЗКА**
   >
   > Вы также можете создать проект на основе шаблона *Быстрый старт*. Он уже содержит полностью готовый процесс из этой статьи.

4. Нажмите *Создать*. Вы сразу перейдете внутрь проекта.

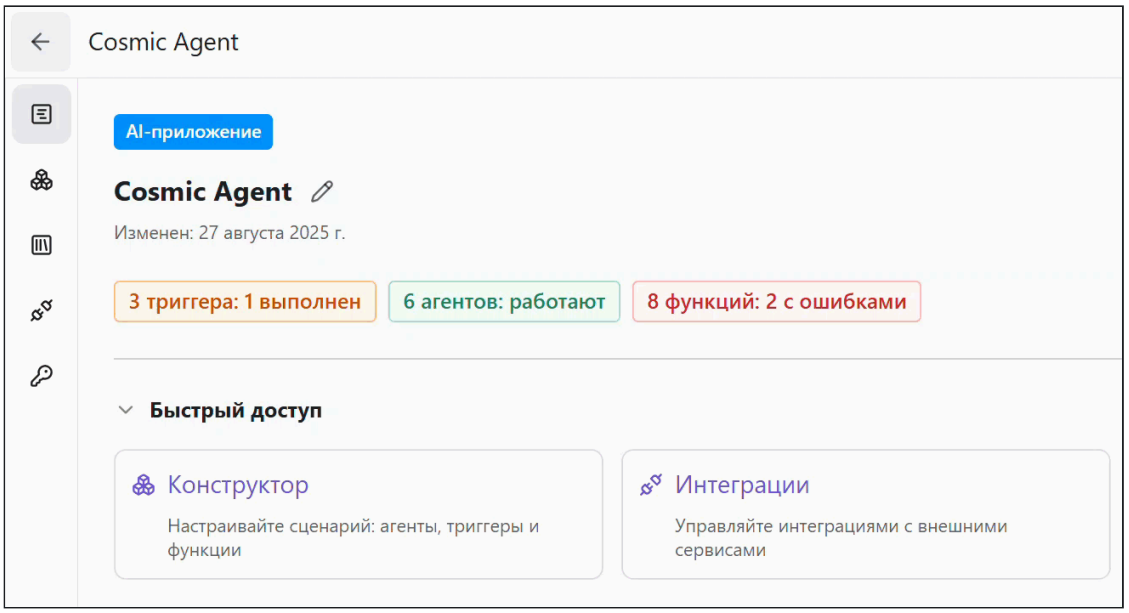
# Шаг 2. Поиск статей через AI-агента

## Триггер по новому сообщению

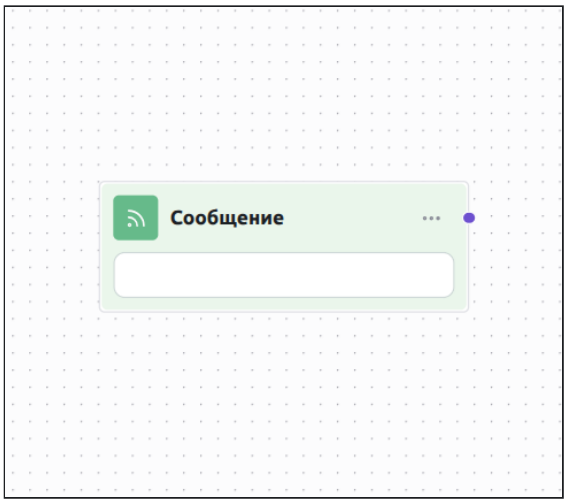
Триггер

1. На левой панели нажмите , чтобы открыть конструктор процессов.
2. В верхнем левом углу нажмите , чтобы открыть список блоков.

   Далее раскройте раздел *Триггеры:*




3. Перетащите блок *Сообщение* на холст. Триггер будет срабатывать каждый раз, когда пользователь отправляет сообщение в чат.




## Функция для агента

Пользовательская функция | Инструмент

Добавьте новую функцию:

1. В списке блоков откройте раздел *Функции*.

2. Нажмите + справа от раздела *Пользовательские*.
3. В открывшемся окне выберите, куда сохранить новую функцию — *Новая коллекция* и нажмите *Далее*.
4. Укажите информацию о коллекции:
   - *Название:* `Новости про космос`.
   - *Идентификатор* `SpaceArticles`.
   - *Описание:* `Коллекция функций для работы с новостями про космос`.
5. Нажмите *Добавить*.
6. В открывшемся окне укажите информацию о новой функции:
   - *Название:* `Найти статьи`.
   - *Идентификатор:* `searchArticles`.
   - *Описание:* `Найти последние статьи по нужной теме`. По описанию агент будет понимать, что делает функция.
7. Нажмите *Сохранить*.

Настройте функцию:

1. Перетащите новую функцию *Найти статьи* из раздела *Функции* → *Пользовательские* → *Новости про космос* на холст.
2. На холсте нажмите на блок этой функции, чтобы открыть ее настройки.
3. Нажмите *Посмотреть код*. В открывшемся окне вы сможете настроить параметры функции и написать ее код.
4. Нажмите *Добавить* в секции *Параметры* и выберите *string*. Далее:
   - В поле *Название* укажите название параметра, например `query`.
   - В поле *Описание* введите `Термин на английском языке, по которому нужно найти статьи`. По этому описанию агент будет понимать, что нужно передать в параметр.
5. В секции *Поля ответа* в поле *Описание* укажите `Данные статей`.
6. Перейдите на вкладку *Код* и введите тело функции на языке JavaScript:

```
// Отправляем запрос, чтобы получить статьи по термину
return Http.get({
  url: "https://api.spaceflightnewsapi.net/v4/articles/",
  params: {
    // Передаем в API-запрос параметр query
    search: query,
    ordering: "-published_at"
  },
  headers: {}
})
  .then(function (response) {
    var body = typeof response.body === "string" ? JSON.parse(response.body) :
response.body;
```

```
  // Сохраняем нужные данные о статьях
  var articles = (body.results || []).map(function (a) {
    return {
      title: a.title,
      summary: a.summary,
      url: a.url,
      publishedAt: a.published_at
    };
  });

  // Если статей нет
  if (articles.length === 0) {
    return "No data";
  }

  return JSON.stringify(articles);
})
.catch(function (err) {
  return "Error fetching articles: " + err;
});
```

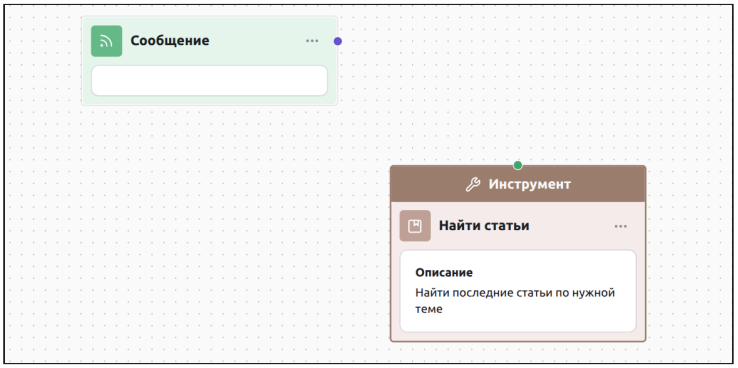
Функция получает релевантные статьи из Spaceflight News API и возвращает их данные.

7. Нажмите *Сохранить*.

Далее переведите функцию в режим инструмента, чтобы AI-агент мог ее использовать:

1. В окне настроек включите *Режим инструмента* и у параметра `query` включите опцию *Заполнит агент*.
2. Нажмите *Сохранить*.

Теперь на холсте есть два блока:




## Создание AI-агента

Агент

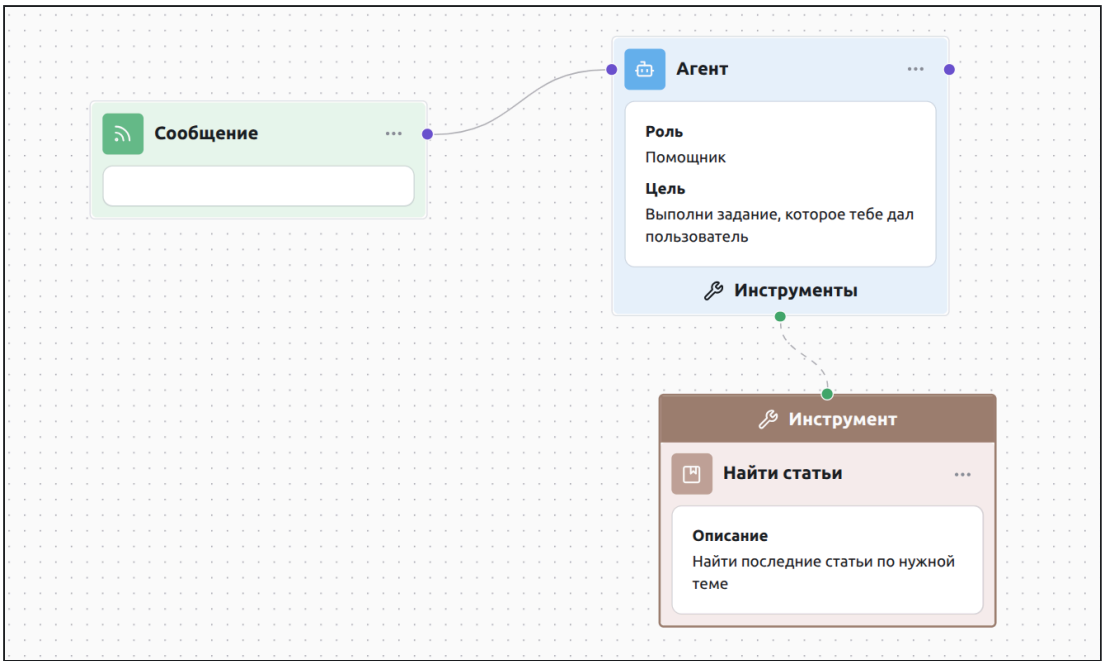
Добавьте агента:

1. Перетащите блок *Агенты → Агент* на холст.
2. На холсте нажмите на блок агента.
3. Раскройте панель *Основные:*
   i. В поле *LLM* выберите *Добавить новую интеграцию*.
   ii. В поле *Доступ к LLM* выберите *Через Just AI*.
   iii. В поле *Провайдер* выберите *YANDEX*, а в поле *Модель — yandex-yandexgpt*.
4. Раскройте панель *Промты:*
   - *Роль:* `Помощник`.
   - *Цель:* `Выполни задание, которое тебе дал пользователь`.
   - *Инструкции:*

```
Если пользователь просит найти новости:
1. Вызови функцию searchArticles и передай термин на английском языке в качестве параметра.
2. Когда получишь новости, составь один абзац текста со всей информацией.
3. Переведи на русский язык, подпиши, за какой период новости, и приложи список ссылок.
4. Отправь пользователю.

Если пользователь просит краткий обзор новостей за последнее время:
1. Проанализируй последние сообщения в чате.
2. Составь один абзац текста с основными событиями, переведи на русский язык. Ссылки и даты не включай.
3. Отправь пользователю.
```

5. Нажмите *Сохранить* вверху окна настроек.

Далее соедините триггер *Сообщение* с агентом, а функцию *Найти статьи* присоедините к агенту как инструмент.



> **ПОДСКАЗКА**
>
> Чтобы удалить связь между блоками, нажмите на связь и затем нажмите `Delete` или `Backspace`.

## Тестирование

Вы можете протестировать процесс в тестовом виджете. Нажмите *Тестировать* в правом верхнем углу редактора.

При открытии виджета отправляется системное сообщение — агент сразу среагирует на него и напишет приветствие в чат.

Пример диалога:

1. **Ассистент**:
   > Здравствуйте! Я ваш помощник. Чем могу помочь вам сегодня?
2. **Пользователь**:
   > Может есть что интересное про Марс?
3. **Ассистент**:
   > Вот последние интересные новости о Марсе за август 2025 года:
   >
   > В начале августа марсоход Curiosity продолжил исследование уникальных структур на Марсе, известных как «boxwork» — это сложные рельефы, которые могут дать новые данные о геологической истории планеты. Команда марсохода подробно изучила так называемые «peace sign» гребни, чтобы узнать больше о процессах формирования этих структур. Кроме того, NASA активно использует марсоход Perseverance для создания панорам и изучения атмосферы Марса, а Европейское космическое агентство разрабатывает новые технологии для будущих марсианских миссий. Важными остаются также международные планы по отправке новых аппаратных комплексов для поиска признаков жизни и исследования климата на Марсе.
   >
   > Ссылки на источники:
   >
   > - https://science.nasa.gov/blog/curiosity-blog-sols-4624-4626-a-busy-weekend-at-the-boxwork/
   > - https://www.nasa.gov/missions/mars-2020-perseverance/perseverance-rover/nasas-perseverance-rover-captures-mars-vista-as-clear-as-day/
   > - https://europeanspaceflight.com/esa-to-develop-hummingbird-inspired-flying-mars-explorer/

- https://europeanspaceflight.com/italy-signs-agreement-with-spacex-for-starship-mars-mission/

# Шаг 3. Регулярная отправка новостей в чат

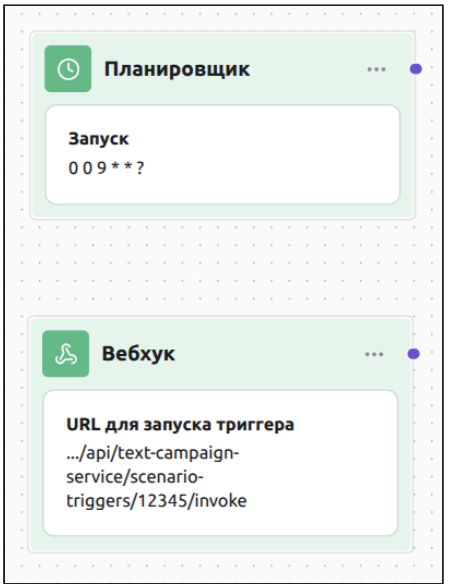
## Расписание и вебхук

Триггеры

Добавьте блок *Планировщик:*

1. Перейдите к блоку *Триггеры → Планировщик* и перетащите его на холст.
2. На холсте нажмите на блок.
3. В поле *Cron-выражение в формате Quartz* введите `0 0 9 * * ?`, чтобы запускать процесс каждый день в 9:00 UTC.
4. Нажмите *Сохранить*.

Далее добавьте блок *Вебхук* — перейдите в раздел *Триггеры* и перетащите этот блок на холст. Когда GET-запрос приходит на адрес вебхука, триггер срабатывает и запускает процесс. Это позволяет внешним системам запускать процесс, например, при наступлении какого-то события.




## Получение новостей

Пользовательская функция

Добавьте новую функцию:

1. В списке блоков выберите *Функции*.
2. Нажмите + справа от раздела *Пользовательские*.

3. Выберите коллекцию `Новости про космос` и нажмите *Далее*.
4. В открывшемся окне укажите информацию о новой функции:
   - *Название:* `Сделать пост`.
   - *Идентификатор:* `sendNews`.
   - *Описание:* `Получить последние новости и составить пост`. По описанию агент будет понимать, что делает функция.
5. Нажмите *Сохранить*.

Настройте функцию:

1. Перетащите новую функцию *Сделать пост* из раздела *Функции* → *Пользовательские* → *Новости про космос* на холст.
2. На холсте нажмите на блок этой функции, чтобы открыть ее настройки.
3. Нажмите *Посмотреть код*. В открывшемся окне вы сможете настроить параметры функции и написать ее код.
4. Функция не имеет параметров, поэтому оставьте их пустыми.
5. В секции *Поля ответа* в поле *Описание* укажите `Текст поста`.
6. Перейдите на вкладку *Код* и введите тело функции:

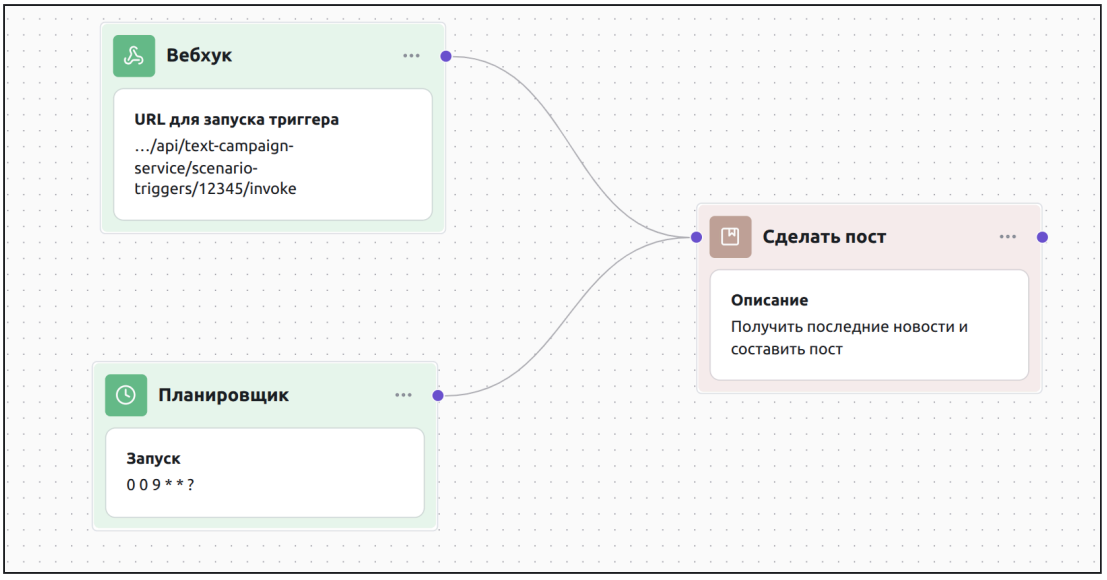
```
// Получаем дату и время, которые были день назад в формате ISO
var oneDayAgoISO = new Date(Date.now() - 24 * 60 * 60 * 1000).toISOString();

// Отправляем запрос, чтобы получить статьи, опубликованные за последний день
return Http.get({
  url: "https://api.spaceflightnewsapi.net/v4/articles/",
  params: {
    published_at_gte: oneDayAgoISO,
    ordering: "-published_at"
  },
  headers: {}
})
  .then(function (response) {
    var body = typeof response.body === "string" ? JSON.parse(response.body) :
response.body;

    var articles = (body.results || []).map(function (a) {
      return {
        title: a.title,
        url: a.url,
        publishedAt: a.published_at
      };
    });

    // Если статей нет
    if (articles.length === 0) {
      return "☁ Сегодня ничего нового";
```

```
  }

  // Формируем текст поста
  var postText = "🚀 Статьи про космос за день:\n\n" +
    articles.map(function (a) {
      return "**" + a.title + "**\n" +
        "🔗 " + a.url + "\n" +
        "🗓 " + new Date(a.publishedAt).toUTCString();
    }).join("\n\n");

  return postText;
})
.catch(function (err) {
  return "Произошла ошибка: " + err;
});
```

Функция получает новости про космос за день из Spaceflight News API и возвращает текст поста.

7. Нажмите *Сохранить* в окне редактирования и далее еще раз *Сохранить* в окне настроек функции.

Далее перетащите новую функцию из списка блоков на холст и соедините ее с триггерами.




## Перевод новостей и отправка сообщения

**Встроенные функции**

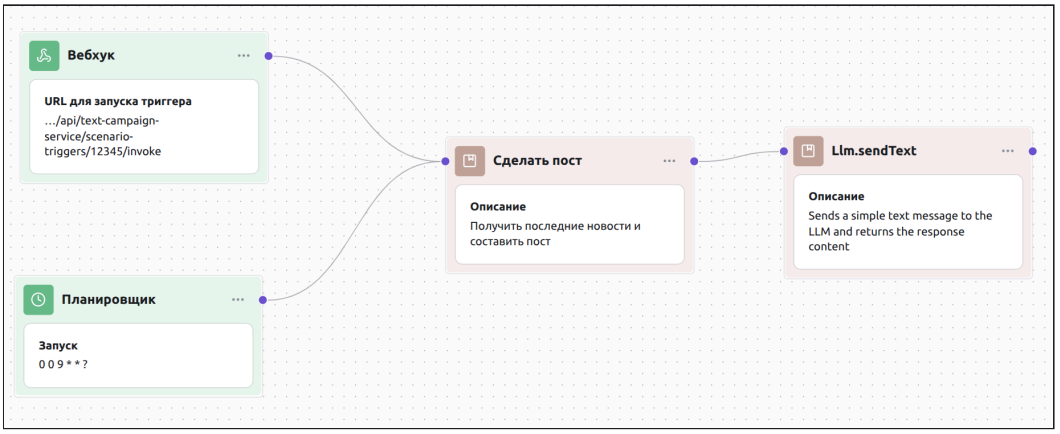
Функция `sendNews` получает заголовки статей на английском языке. Добавьте запрос к LLM, чтобы перевести текст на русский язык:

1. В списке блоков откройте раздел *Функции* → *Встроенные* → *Llm* и перетащите функцию *sendText* на холст.

2. На холсте нажмите на блок этой функции.
3. В параметре *llmModelKey* выберите *YANDEX — yandex-yandexgpt*.
4. В параметре *text* введите `{{"Переведи пост полностью на русский язык, верни только перевод в ответе: " + Context.getLastFunctionResult()}}`.

> **ПОДСКАЗКА**
>
> Встроенная функция `Context.getLastFunctionResult()` возвращает результат последней выполненной функции. В данном случае это текст сообщения с новостями, который формируется в `sendNews`.
>
> `{{}}` используется для подстановки JavaScript-выражений в параметры функций.

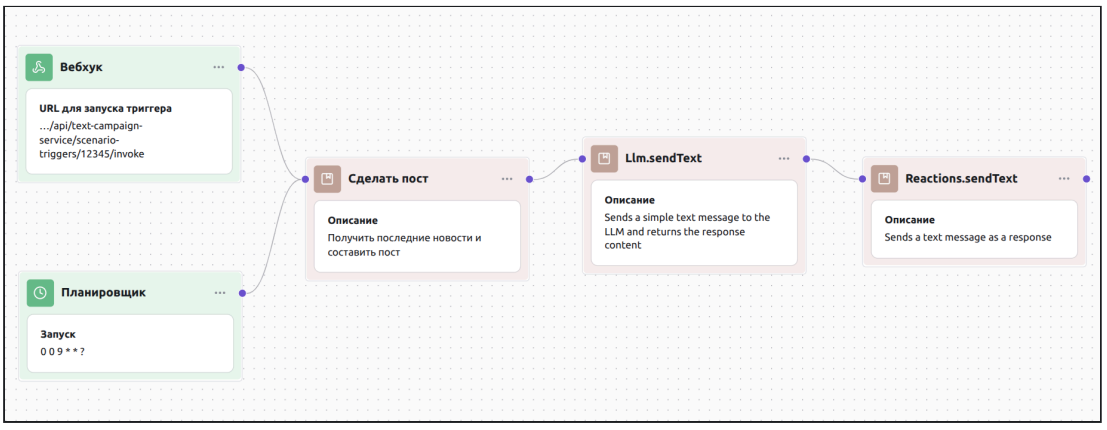
5. Нажмите *Сохранить* и соедините *Сделать пост* с *Llm.sendText*, чтобы эти функции выполнялись последовательно:




Добавьте функцию для отправки сообщения в чат:

1. В списке блоков откройте раздел *Функции* → *Встроенные* → *Reactions* и перетащите функцию *sendText* на холст.
2. На холсте нажмите на блок этой функции.
3. В параметре *text* введите `{{Context.getLastFunctionResult()}}` и нажмите *Сохранить*.
4. Соедините эту функцию с *Llm.sendText*.

В итоге три функции будут выполняться последовательно:



## Тестирование

1. Нажмите *Тестировать* в правом верхнем углу редактора.
2. При открытии виджета отправляется системное сообщение — агент сразу среагирует на него и напишет приветствие в чат:

   > Здравствуйте! Я ваш помощник. Чем могу помочь вам сегодня?

3. Чтобы не ждать запуска по расписанию, вызовите вручную вебхук:
   i. На холсте нажмите на блок *Вебхук*.
   ii. Скопируйте URL из поля *URL для запуска триггера*.
   iii. Откройте в браузере скопированный URL. Это отправит GET-запрос на адрес вебхука и запустит триггер.
4. В чате появится сообщение с последними новостями про космос:

   > 🛰️ Статьи про космос за день:
   >
   > **Взгляд на рождение звёзд с телескопа «Джеймс Уэбб» от NASA** 📎
   > https://science.nasa.gov/missions/webb/glittering-glimpse-of-star-birth-from-nasas-webb-telescope/ 🗓️ Чт, 04 сен 2025 14:00:00 GMT
   >
   > **Офицер NASA помогает осуществить возвращение на Луну** 📎
   > https://www.nasa.gov/image-article/co-patricia-white-helps-moon-journey/ 🗓️ Чт, 04 сен 2025 14:00:00 GMT
   >
   > ...

5. Попросите агента составить краткий обзор последних новостей:

   > Напиши краткий обзор новостей за последний день

6. Агент ответит:

   > Вот краткий обзор главных новостей за последний день:
   >
   > За последний день в космической отрасли произошло несколько значимых событий. Телескоп Джеймс Уэбб предоставил впечатляющие изображения рождения звёзд, а NASA активно готовится к возвращению на Луну — среди прочего, контрактный офицер агентства помогает реализовать лунные миссии. Также стало известно название новой дочерней миссии Galileo — Celeste. В области технологий NASA Glenn тестирует мини-рентген для медицины будущих полётов.

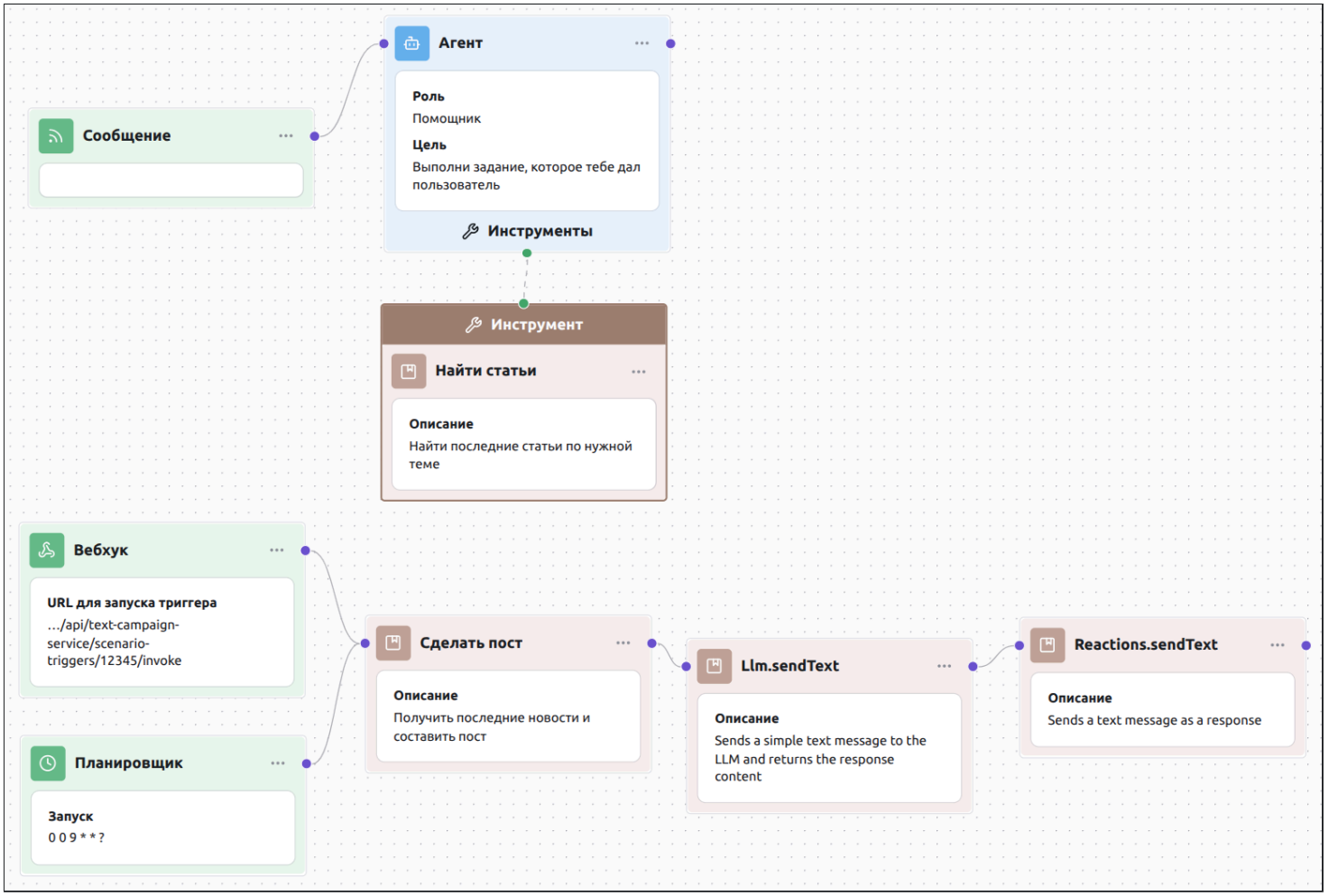
7. Агент по умолчанию видит последние 50 сообщений в чате. Попросите агента составить краткий обзор по всем доступным сообщениям в чате:

   > Что самое интересное ты писал в чат за всё время?

8. Агент ответит:

> За всё время в чате я писал о ключевых событиях в космической сфере: соперничестве США и Китая за первенство в возвращении человека на Луну, дискуссиях вокруг отмены лунных программ SLS и Lunar Gateway, а также кадровых перестановках в NASA. Я делал обзоры о новых технологиях — от испытаний возвращаемых капсул и спутников до медицинских инноваций для космоса и новых материалов для марсианских миссий. Также рассказывал о научных открытиях: телескоп Джеймс Уэбб предоставил уникальные изображения рождения звёзд, а учёные наблюдали редкие типы чёрных дыр и запускали новые миссии по их изучению.

# Полный процесс




# Дальнейшие шаги

Подключите канал к процессу. Например, вы можете опубликовать процесс в Telegram-боте.

# Обзор конструктора процессов

Конструктор процессов в Just AI Agent Platform — это визуальный low-code инструмент, разработанный с фокусом на простоту использования. Благодаря интуитивному drag-and-drop интерфейсу, он не требует глубоких технических знаний. Это делает конструктор доступным даже для менее подготовленных пользователей.

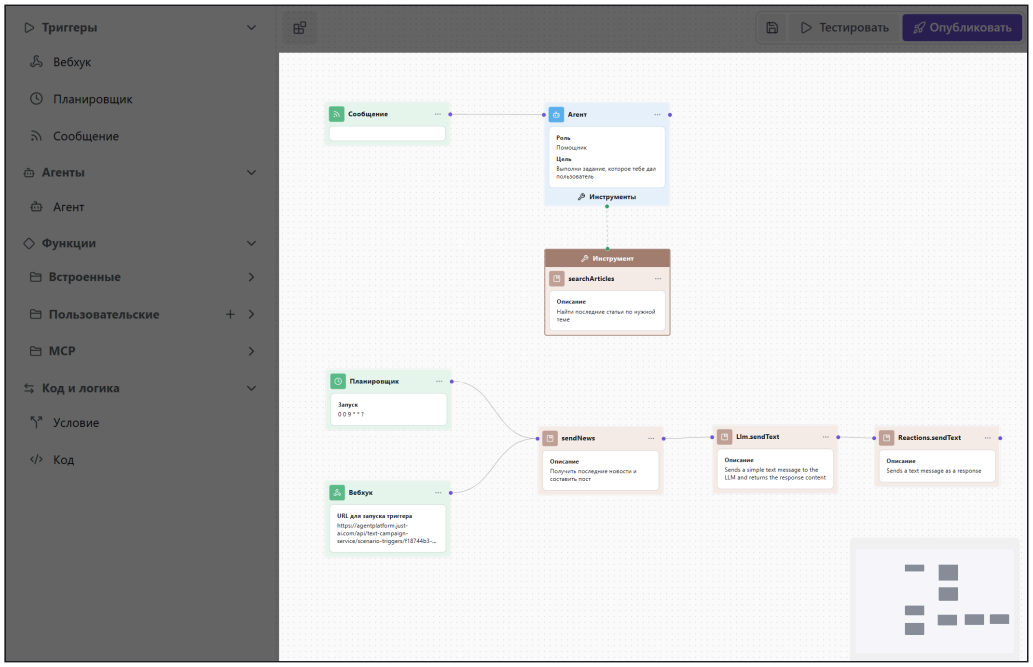
Технические специалисты могут расширить возможности конструктора с помощью встроенных и пользовательских функций, интеграций с MCP-серверами и блоков кода.

После создания проекта AI-приложения вы увидите интерфейс с основными компонентами: холст, список блоков и панель управления.

## Компоненты интерфейса

### Холст

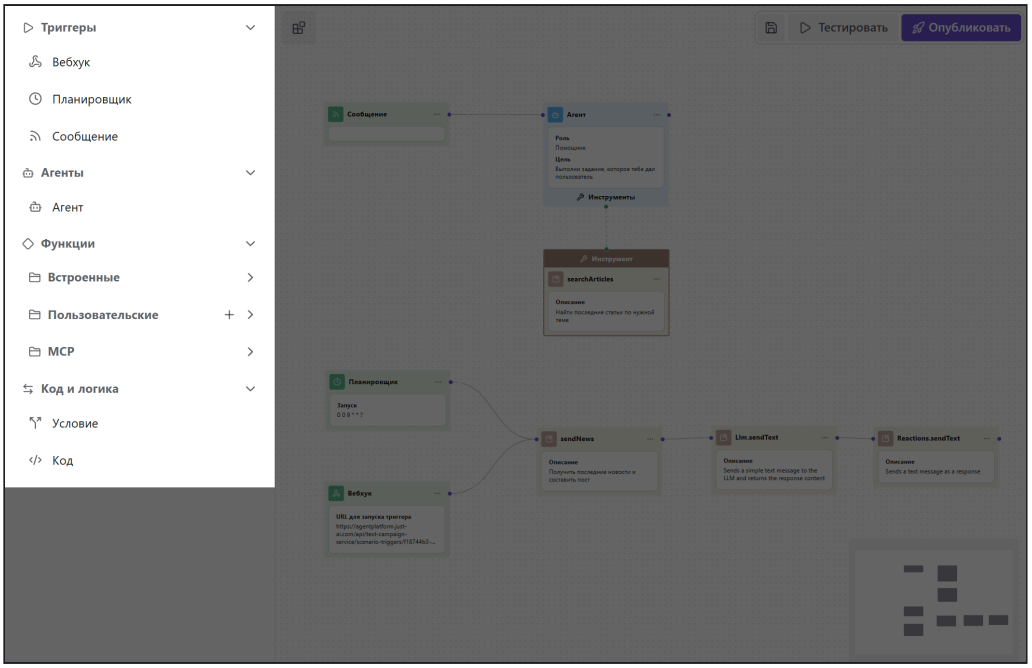
Холст — это основное рабочее пространство, на котором вы выстраиваете процесс, размещая и соединяя функциональные блоки.




### Список блоков

На панели слева находится список функциональных блоков. Блоки — это компоненты вашего процесса, которые принимают, обрабатывают и передают данные между различными частями вашего AI-приложения.

Чтобы развернуть список блоков, нажмите ⊞ в левом верхнем углу холста.

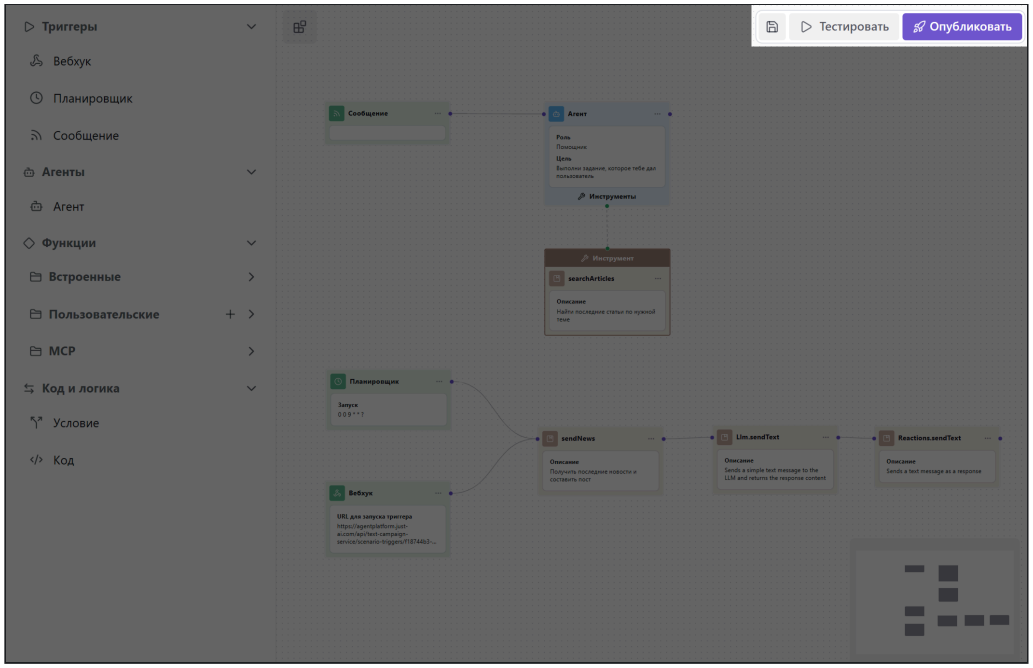



Основные категории блоков:

- *Триггеры:* запускают процесс в ответ на определенные события. Вы можете настроить запуск по вебхуку, при получении сообщения ассистентом или с определенной периодичностью (планировщик). Подробнее в разделе Триггеры.
- *Агенты:* позволяют интегрировать и использовать возможности различных AI-моделей от множества провайдеров. Подробнее в разделе Агент.
- *Функции:* предоставляют функции для обработки данных. Вы можете выбрать из доступных встроенных функций, создать свои собственные или подключить MCP-серверы, которые предоставляют функции нужных сервисов.
- *Код и условия:* позволяют добавлять пользовательский JavaScript-код и логические условия для сложной обработки данных.

## Панель управления

На панели управления в правом верхнем углу находятся основные команды: сохранить, тестировать, опубликовать.




Если вы хотите протестировать текущий процесс, нажмите *Тестировать*. Чтобы сделать процесс доступным для внешнего использования, например в каналах, нажмите *Опубликовать*.

# Основные сценарии

## Работа с блоками

С блоками можно выполнять следующие действия:

- Добавлять на холст: просто перетащите блок из боковой панели.
- Связывать между собой: наведите курсор на маленький кружок блока-источника и перетащите линию к кружку блока-приёмника.

  Связи устанавливают порядок выполнения блоков. Для блока *Агент* также доступны связи с другими блоками в качестве инструментов. Они позволяют агенту использовать дополнительные функции для обработки запросов. Последовательность соединения инструментов с агентом не имеет значения.
- Редактировать: нажмите на блок, чтобы открыть его настройки.
- Удалять: выберите блок и нажмите •••, а затем *Удалить*.

Чтобы удалить связь между блоками, нажмите на связь, а затем `Delete` или `Backspace`. Перетаскивать связи между блоками нельзя.

## Тестирование процесса

Чтобы протестировать процесс, нажмите *Тестировать* на панели управления. Откроется окно тестирования, где вы можете ввести сообщение для ассистента и увидеть, как ваш процесс

обрабатывает запрос.

Если в качестве триггера вы использовали вебхук или планировщик:

- Для вебхука: скопируйте URL для запуска триггера из блока *Вебхук* и вставьте его в адресную строку браузера.
- Для планировщика: укажите ближайшее время в настройках блока *Планировщик* и дождитесь наступления этого времени.

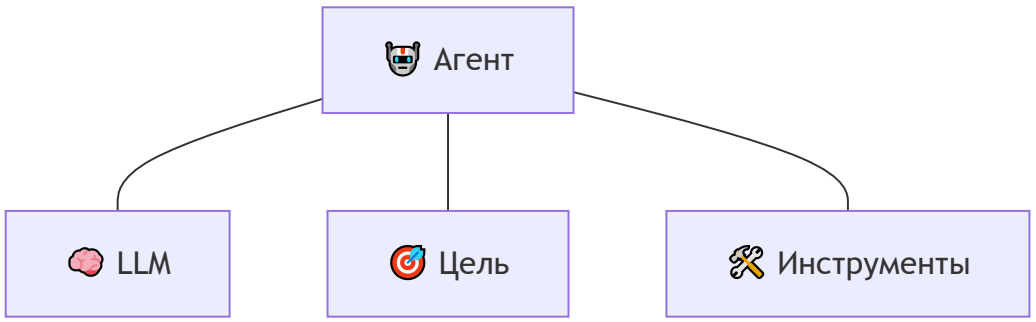
Как только триггер сработает, рабочий процесс начнет выполняться, и вы увидите результаты в окне тестирования.

## Публикация процесса

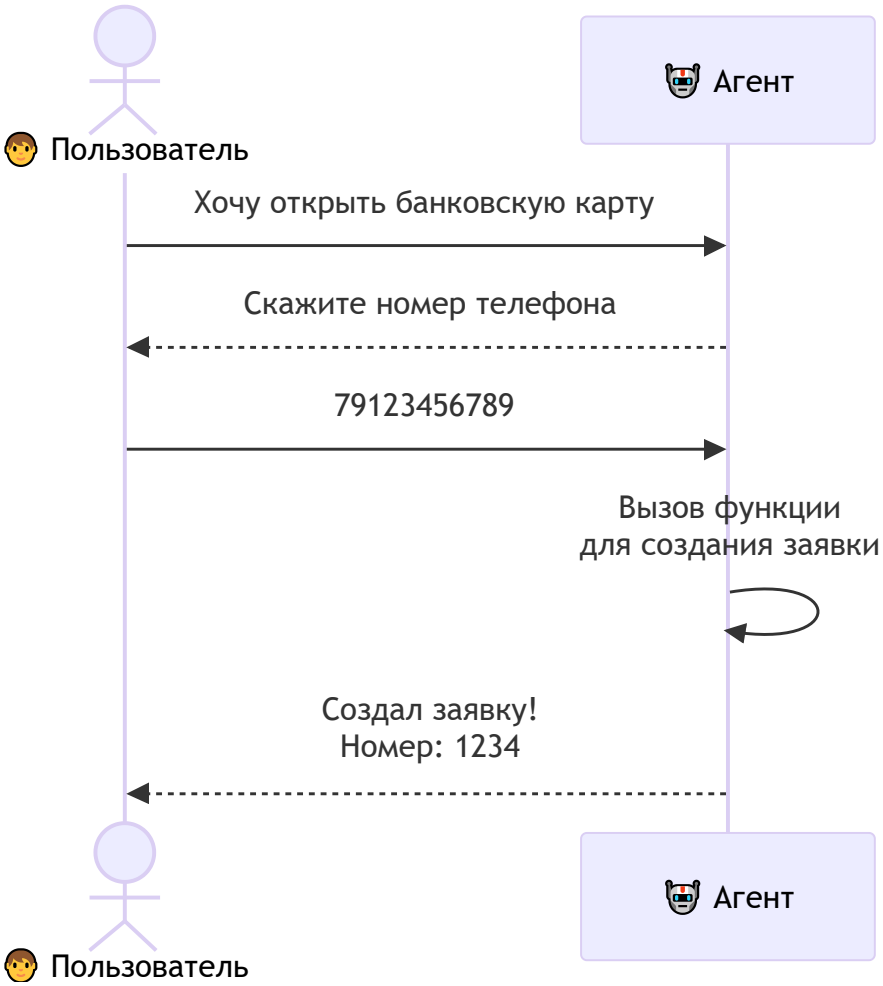
После того как вы протестировали и убедились, что процесс работает правильно, вы можете опубликовать его. Для этого нажмите *Опубликовать* на панели управления.

# Блок «Агент»

С помощью блока «Агент» вы можете добавить в процесс **AI-агента** — умного помощника на основе LLM, который сам принимает решения для достижения цели. Он способен гибко отвечать на вопросы, уточнять информацию и вызывать нужные инструменты, как это сделал бы живой оператор.

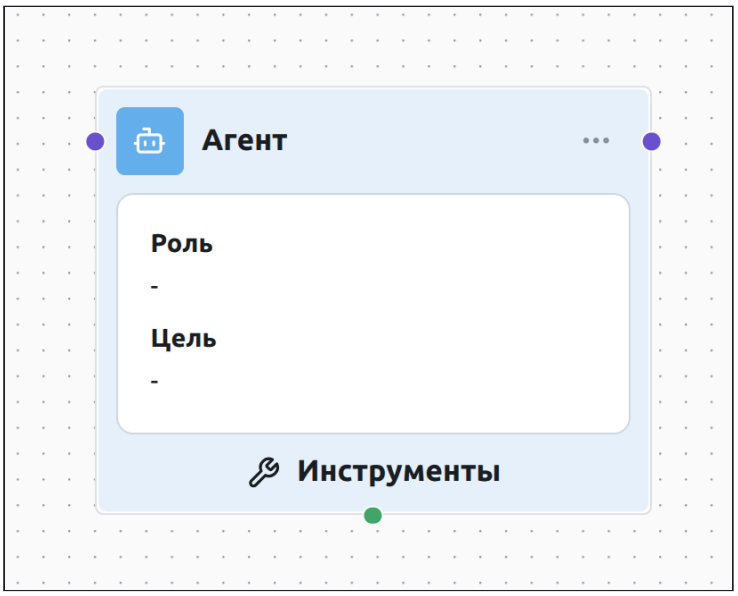



Например, если пользователь хочет оформить банковскую карту, агент может уточнить его данные, самостоятельно создать заявку на выпуск карты и сообщить номер заявки пользователю:




## Как добавить агента на холст

1. Перейдите в проект и на левой панели нажмите , чтобы открыть конструктор процессов.
2. В верхнем левом углу холста нажмите — откроется список блоков.
3. Перетащите блок *Агенты* → *Агент* на холст.




# Настройки агента

Чтобы изменить настройки агента:

1. Нажмите на блок *Агент* на холсте.
2. В открывшемся окне укажите настройки.
3. Нажмите *Сохранить*.

## Основные настройки

В поле *Настройки* → *Основные* → *LLM* выберите существующую интеграцию с LLM или добавьте новую. Если вы создадите новую интеграцию, она станет доступна в разделе *Интеграции*.

Агент будет использовать выбранную модель для генерации ответов, принятия решений и вызова инструментов.

> **ПРЕДУПРЕЖДЕНИЕ**
>
> Для работы агента выбранная модель LLM должна поддерживать function calling. Вы можете найти информацию о поддержке function calling в официальной документации самой модели.

## Промты

В разделе *Настройки → Промты* заполните следующие поля:

| Поле | Описание | Пример |
|---|---|---|
| *Роль* | Роль или персона агента в диалоге. Влияет на тон общения и на то, как агент отвечает на вопросы о себе. | `Сотрудник банка` |
| *Цель* | В диалоге агент будет пытаться выполнить указанную цель. | `Создай заявку на выдачу банковской карты` |
| *Инструкции* | Представьте, что вы даете задание новому сотруднику. Здесь укажите, как именно он должен выполнить свою работу: с чего начать, какие шаги предпринять и какие инструменты использовать. | `Сначала уточни номер телефона. После этого создай заявку с помощью функции Application.create. Отвечай коротко и всегда на русском языке` |

**ПОДСКАЗКА**

Мы рекомендуем подробно указывать роль, цель и инструкции для агента. Так вы сможете получать более предсказуемые и стабильные результаты.

## Генерация и поведение

В разделе *Настройки → Генерация и поведение* вы можете управлять параметрами генерации ответов и действиями агента после генерации.

| Поле | Описание |
|---|---|
| *Температура* | Регулирует креативность ответов. При высоких значениях результаты будут более творческими и менее предсказуемыми.<br><br>Не рекомендуем менять параметры *Температура* и *Top P* одновременно. |
| *Top P* | Регулирует разнообразность ответов. При низких значениях нейросеть выбирает из меньшего количества вероятных слов, а при высоких — ответ может получиться разнообразнее. |

| Поле | Описание |
| --- | --- |
| | Не рекомендуем менять параметры *Температура* и *Top P* одновременно. |
| *Presence penalty* | Снижает вероятность повторного использования слов в тексте. Помогает сделать ответ разнообразнее по лексике. |
| *Frequency penalty* | Снижает вероятность слишком частого повторения одних и тех же слов. Уменьшает риск того, что агент будет зацикливаться на одинаковых фразах. |
| *Максимальный размер ответа, в токенах* | Максимальное количество токенов, которое модель может сгенерировать за одну итерацию. |
| *Отправлять ответ пользователю* | По умолчанию агент сразу отвечает пользователю в чат. Вы можете отключить эту опцию, чтобы использовать агента как функцию. Подробнее смотрите в статье Работа агента в процессе. |
| *Режим диалога* | Включите, чтобы агент мог продолжать диалог с пользователем, не переходя к другим блокам процесса. Подробнее смотрите в статье Работа агента в процессе. |

## Поведение в процессе

На вкладке *Процесс* вы можете настроить логику переходов между блоками в процессе.

| Поле | Описание |
| --- | --- |
| *Следующий шаг процесса* | Указывает, какой блок будет выполнен после агента. Если при этом включена опция *Режим диалога*, то переход будет выполнен, только когда агент достигнет цели или пользователь не захочет продолжать диалог с агентом. Подробнее смотрите в статье Работа агента в процессе. |
| *Передача управления другим агентам* | Выберите других агентов, которым текущий агент сможет передавать контекст и историю диалога. Текущий агент анализирует цели других агентов и сам решает, кто лучше сможет обработать запрос пользователя. Подробнее смотрите в статье Работа агента в процессе. |

| Поле | Описание |
| --- | --- |
| | Вы также можете использовать опцию *Включая новых созданных агентов*, чтобы любые новые агенты также становились доступны для передачи диалога. |

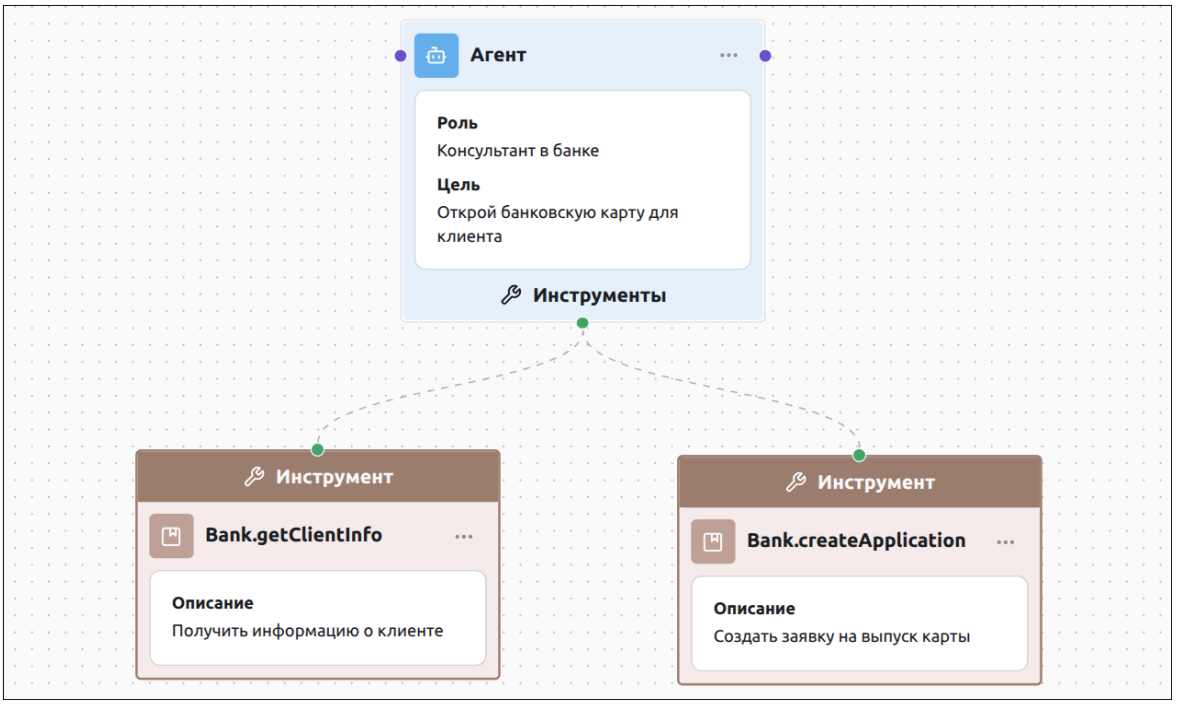
# Подключение инструментов к агенту

**Инструмент** — функция, которую может вызвать агент для выполнения цели.

Чтобы подключить инструмент:

1. Перетащите любую функцию на холст.
2. Нажмите на блок функции:
    i. В открывшемся окне включите *Режим инструмента* и выберите параметры, которые должен заполнить агент при вызове функции.
    ii. Нажмите *Сохранить*.
3. Соедините блок функции с блоком *Агент*.

К одному агенту может быть подключено несколько функций. Агент сам решает, когда и какую функцию вызвать.



# Работа агента в процессе

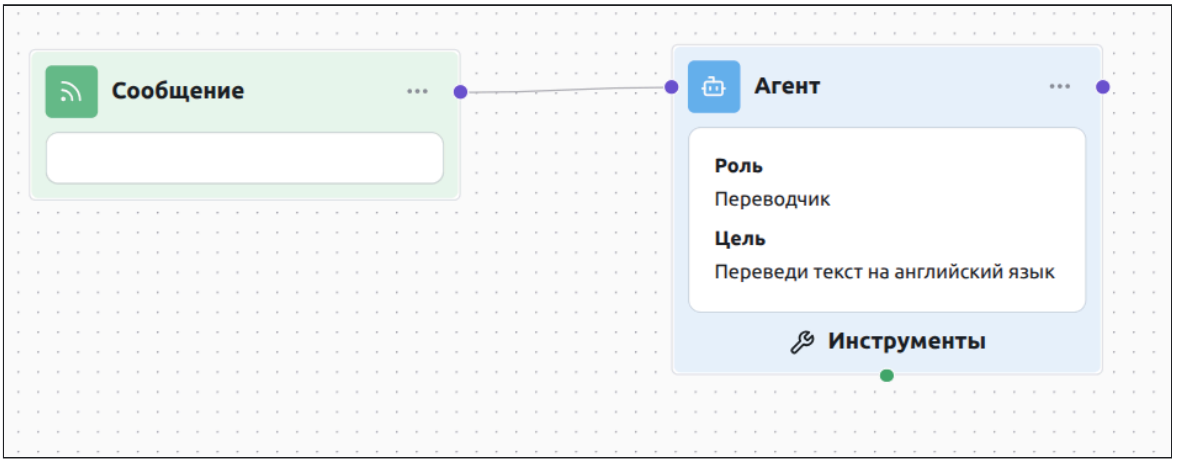
В зависимости от вашей задачи, агент может работать в процессе по-разному: быть частью сложной автоматизации, напрямую отвечать пользователю в чате или вести полноценный диалог.

Из этой статьи вы узнаете, как настроить агента для разных задач:

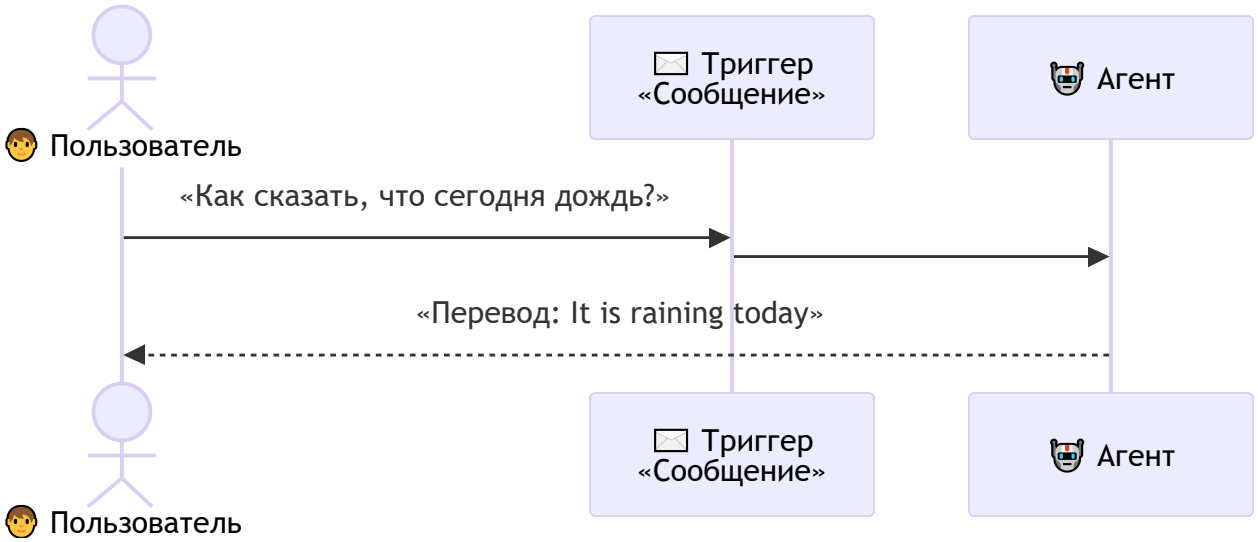
- Напрямую отвечать пользователю: режим по умолчанию, в котором агент сразу отправляет свой ответ в чат.
- Вести полноценный диалог: как научить агента поддерживать беседу для решения многошаговых задач.
- Использовать как функцию для генерации контента: когда нужно, чтобы агент создал текст и передал его в другой блок.
- Создавать мультиагентные процессы: как объединить несколько агентов для построения сложных и надежных систем.

## Ответ пользователю

По умолчанию у агента включена опция *Отправлять ответ пользователю*, поэтому агент автоматически отправляет сообщение в чат сразу после генерации ответа.




Процесс будет работать так:

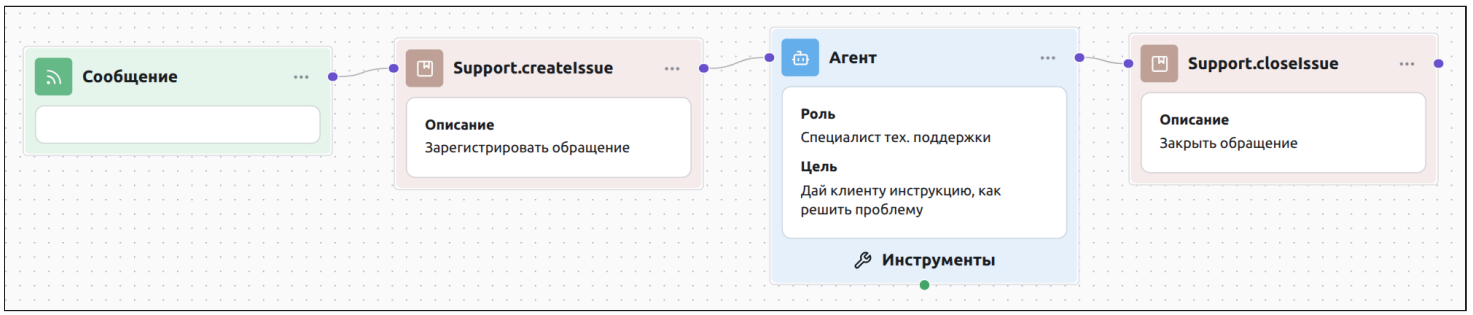



> **ПОДСКАЗКА**
>
> При этом вы можете получить ответ агента и далее использовать его в процессе с помощью функции `Context.getLastFunctionResult()`.

## Диалог с агентом

Часто для решения задачи агенту нужно обменяться с пользователем несколькими сообщениями, например, чтобы уточнить детали или задать вопросы. Однако по умолчанию агент обрабатывает только одно сообщение и процесс сразу переходит к следующему блоку.

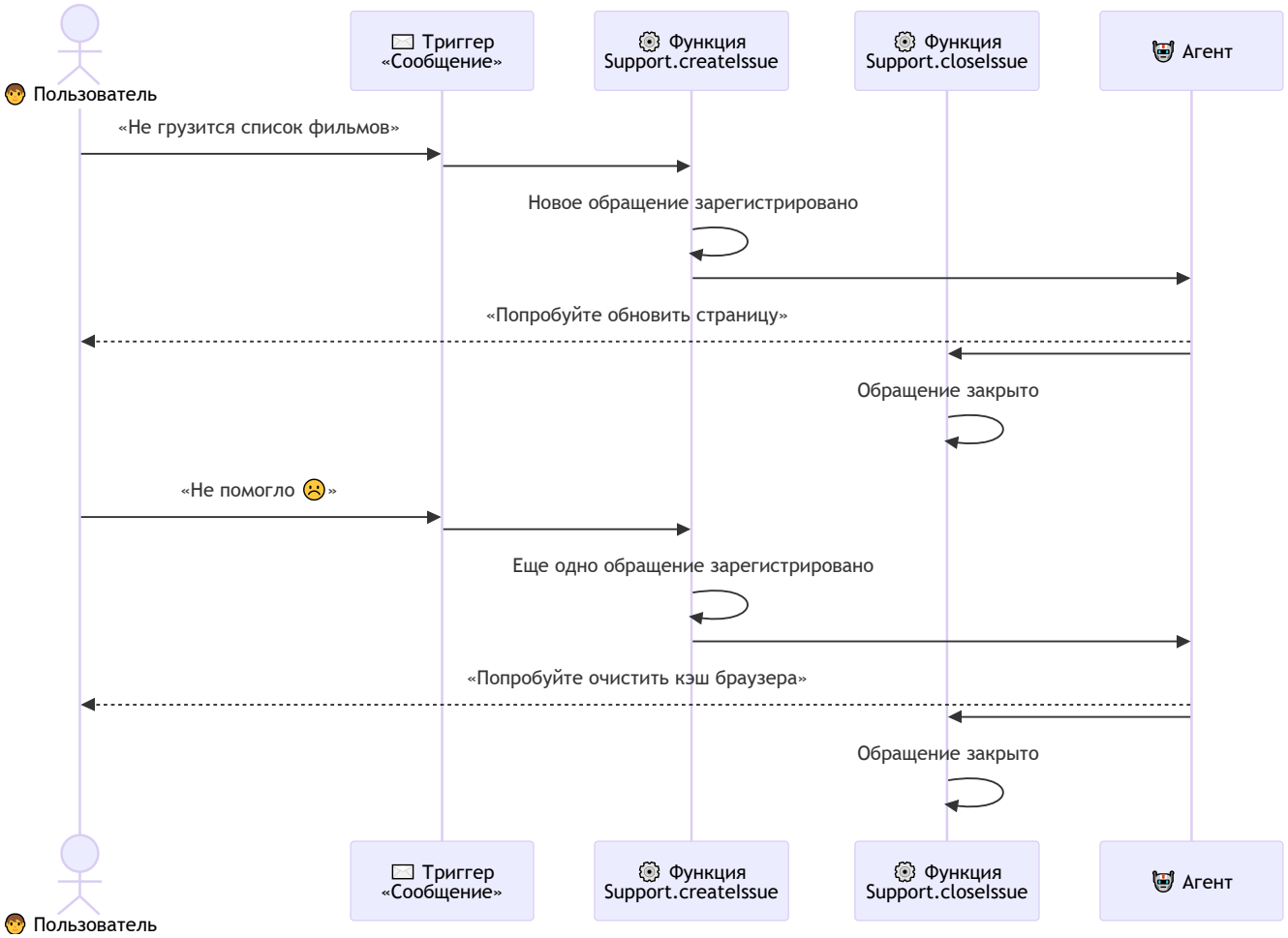
Предположим, у нас есть процесс агента поддержки:




- У агента включена опция *Отправлять ответ пользователю*.
- Перед блоком агента выполняется функция `Support.createIssue`, которая регистрирует новое обращение в поддержку. После блока агента выполняется функция `Support.closeIssue`, которая закрывает обращение.

  > **ПОДСКАЗКА**
  >
  > `Support.createIssue` и `Support.closeIssue` — это пользовательские функции, которые мы создали заранее для этого примера.

В таком процессе на **каждое** сообщение пользователя будет создаваться новое обращение. Обращение также будет закрываться после **каждого** ответа агента.

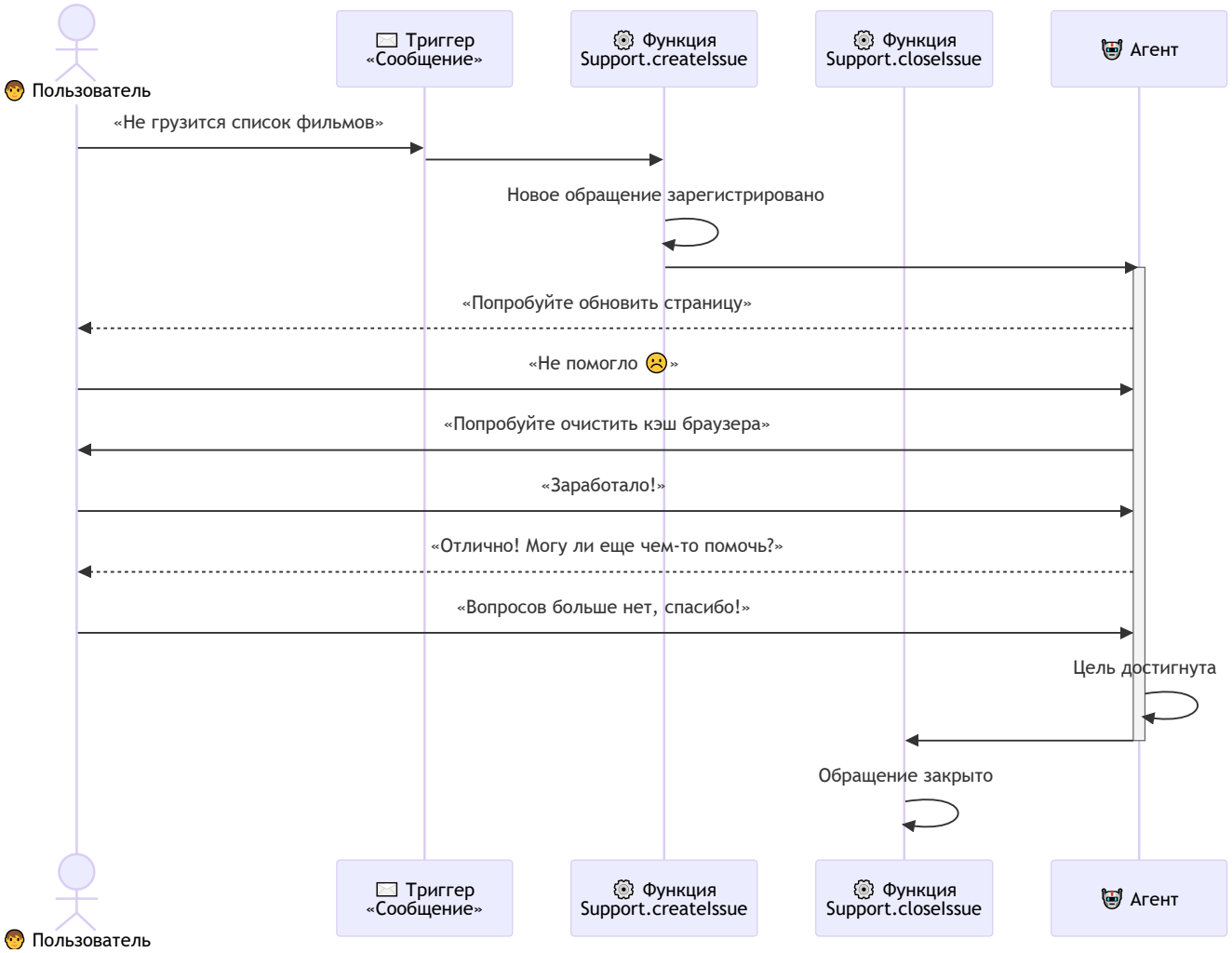



Чтобы агент мог вести полноценный диалог, нужно «остановить» процесс на его блоке до тех пор, пока задача пользователя не будет решена. Для этого:

1. Перейдите в настройки блока *Агент*.
2. На вкладке *Настройки → Генерация и поведение* включите опцию *Режим диалога*. При этом опция *Отправлять ответ пользователю* включится автоматически.
3. Выберите блок, в который должен происходить переход после достижения цели. Это можно сделать двумя способами:
   - В настройках агента на вкладке *Процесс* выберите блок в поле *Следующий шаг процесса*.
   - Просто присоедините блок к выходу агента на холсте.

> **ПОДСКАЗКА**
>
> В нашем примере это уже блок с функцией `Support.closeIssue`.

Теперь процесс будет работать правильно: обращение откроется один раз и закроется только тогда, когда агент выполнит свою цель.



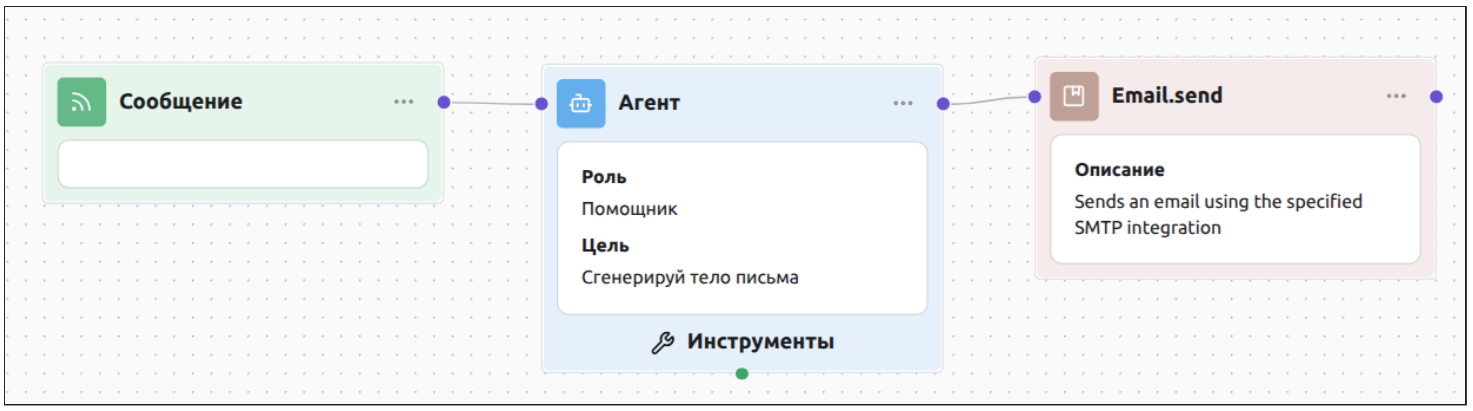

# Агент как функция

Если вы хотите использовать ответ агента в других блоках или как-то дополнительно обрабатывать его, то используйте агента в качестве функции. Для этого:

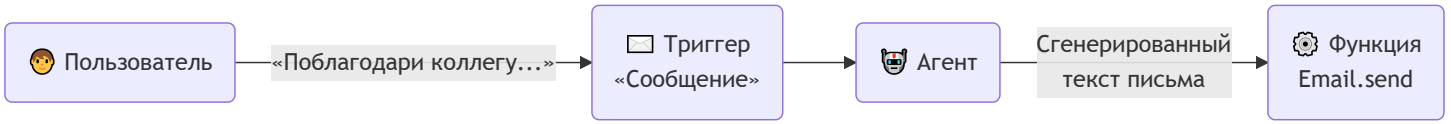
1. Отключите настройку *Отправлять ответ пользователю*. Агент будет генерировать ответ, но не будет отправлять его в чат.
2. В следующем блоке получите ответ агента с помощью системной функции `Context.getLastFunctionResult()`.

Например, вы можете генерировать тело письма через агента и использовать этот текст в следующем блоке:

1. Добавьте системную функцию `Email.send` и соедините с агентом.
2. У функции в параметре `content` укажите `{{Context.getLastFunctionResult()}}`. Тогда в параметр будет подставляться результат последней выполненной функции, в данном случае это будет ответ агента.




В итоге процесс будет работать так:




# Мультиагентные процессы

## Преимущества

Преимущества использования нескольких агентов в одном процессе:

- **Разделение ответственности**

  Каждый агент выполняет узкую роль — LLM лучше справляется с одной задачей.
- **Контроль данных**

  Вы можете контролировать, какие данные получает каждый агент, и не передавать лишнюю информацию.
- **Гибкость и масштабируемость**

  Легко добавлять новых агентов для дополнительных задач.

Такой подход облегчает разработку сложных процессов и повышает надежность.

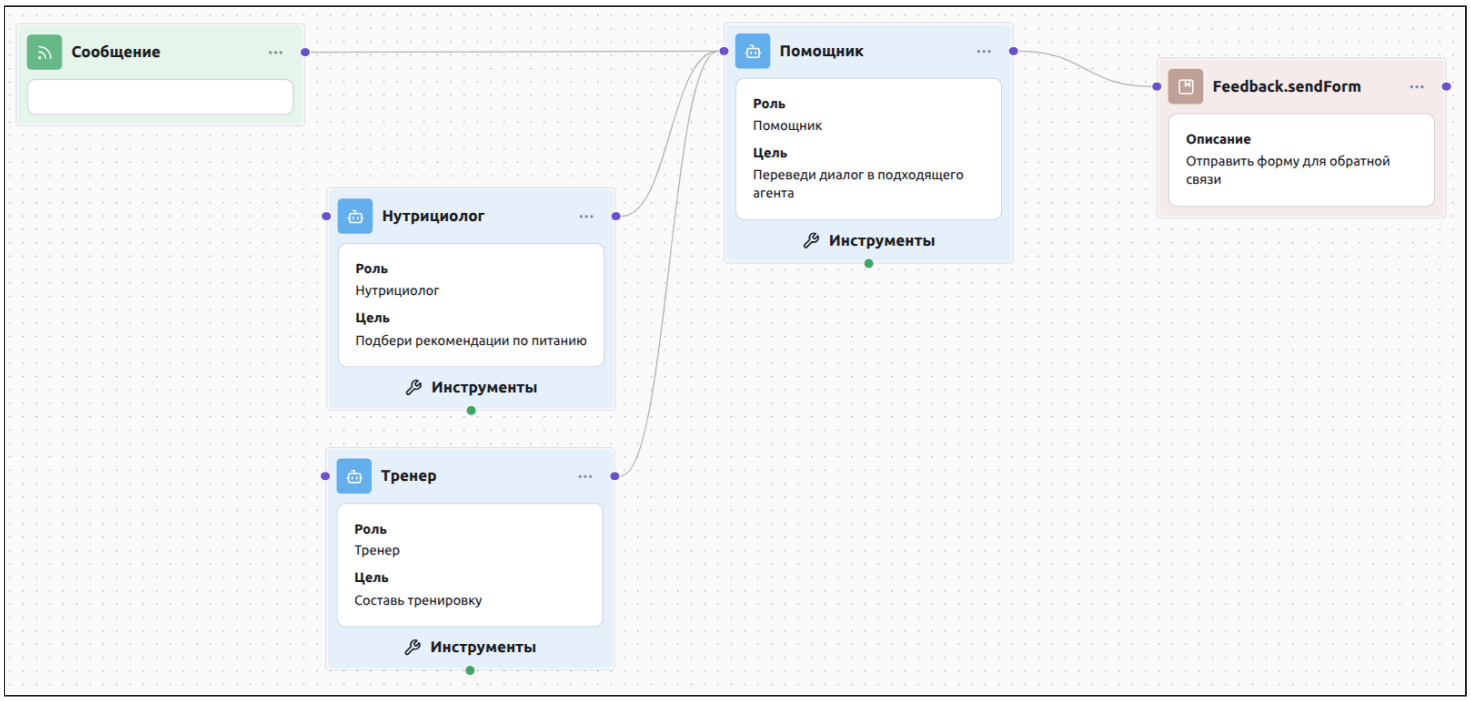
## Как настроить процесс

В примере ниже показан процесс с тремя агентами. В этом процессе пользователь может получить консультацию по питанию или попросить составить план тренировок.

Первое сообщение пользователя попадет к агенту-оркестратору, который определит, к какому из двух специализированных агентов (по питанию или по тренировкам) нужно перенаправить запрос. Пользователь сможет продолжить диалог с выбранным агентом или переключиться

на другого. Если цель достигнута или пользователь откажется от помощи, процесс завершится — будет выполнена функция `Feedback.sendForm`.

> **ПОДСКАЗКА**
>
> `Feedback.sendForm` — это пользовательская функция, которую мы создали заранее для этого примера.




> **ОТОБРАЖЕНИЕ СВЯЗЕЙ МЕЖДУ АГЕНТАМИ**
>
> Передача диалога другим агентам настраивается только в окне агента-оркестратора, при этом на холсте **не отображаются связи между агентами**.
>
> Стрелки на холсте показывают, куда перейдет процесс после того, как агент-специалист (например, «Нутрициолог») полностью завершит свою работу. В нашем примере он вернет управление обратно «Помощнику».

Рассмотрим настройки каждого из агентов.

- **Агент «Помощник»** — это оркестратор, который перенаправит запрос к нужному агенту.
  - У агента включена опция *Режим диалога*, чтобы он мог поддерживать диалог с пользователем и процесс не завершался после первого ответа.
  - На вкладке *Процесс* в секции *Передача управления другим агентам* выбраны *Нутрициолог* и *Тренер*. Это позволяет агенту-оркестратору классифицировать запросы и передавать их нужному агенту.

    Оркестратор учитывает цели агентов, чтобы правильно классифицировать запросы.

> **ПОДСКАЗКА**
>
> Вы также можете включить опцию *Включая новых созданных агентов*. Так любые новые агенты в процессе будут автоматически доступны для оркестратора.

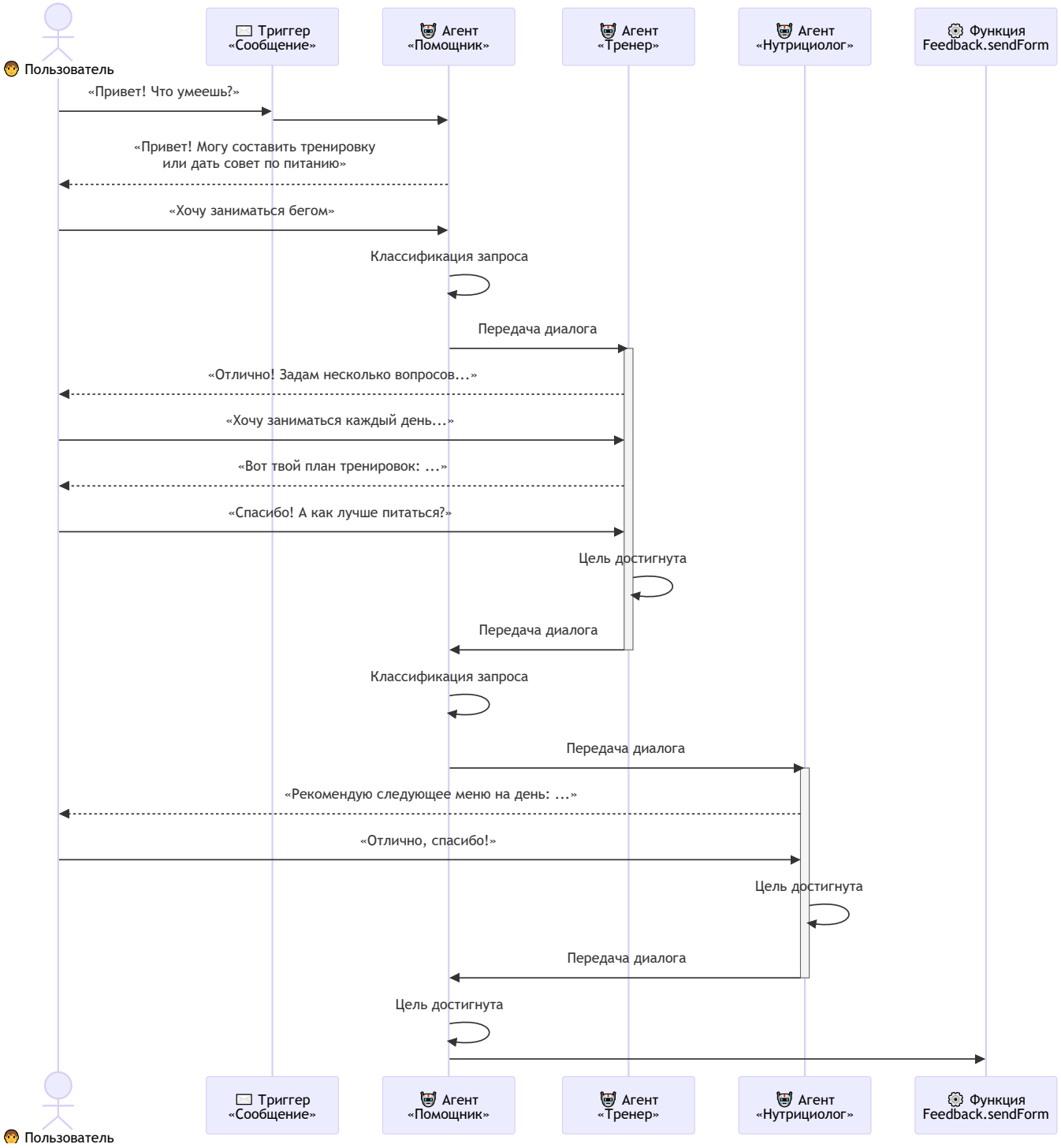
- **Агенты «Тренер» и «Нутрициолог»** — это специализированные агенты, которые обрабатывают запросы по своим направлениям.
  - У каждого агента включена опция *Режим диалога*, чтобы они могли поддерживать диалог с пользователем и процесс не завершался после первого ответа.
  - *Следующий шаг процесса* у каждого агента — это агент оркестратор.

    Если у пользователя нет больше вопросов или вопрос не по теме агента:

    a. Диалог перейдет к агенту-оркестратору.

    b. Оркестратор может передать запросу другому агенту или посчитать, что можно завершить диалог.

Процесс работает так:


Пользователь
Триггер «Сообщение»
Агент «Помощник»
Агент «Тренер»
Агент «Нутрициолог»
Функция Feedback.sendForm
«Привет! Что умеешь?»
«Привет! Могу составить тренировку или дать совет по питанию»
«Хочу заниматься бегом»
Классификация запроса
Передача диалога
«Отлично! Задам несколько вопросов...»
«Хочу заниматься каждый день...»
«Вот твой план тренировок: ...»
«Спасибо! А как лучше питаться?»
Цель достигнута
Передача диалога
Классификация запроса
Передача диалога
«Рекомендую следующее меню на день: ...»
«Отлично, спасибо!»
Цель достигнута
Передача диалога
Цель достигнута
Пользователь
Триггер «Сообщение»
Агент «Помощник»
Агент «Тренер»
Агент «Нутрициолог»
Функция Feedback.sendForm

# Встроенные функции

Just AI Agent Platform предоставляет встроенные JavaScript-функции, которые позволяют управлять диалогом, взаимодействовать с базами данных и интеграциями, отправлять HTTP-запросы и многое другое.

Подробнее о работе с функциями читайте в разделе Использование функций в процессе.

## Встроенные коллекции

| Коллекция | Описание |
|---|---|
| `Asr` и `Tts` | Распознавать и синтезировать речь |
| `Context` | Получать метаданные, подробную информацию о запросе пользователя, отслеживать состояние процесса |
| `Credentials` | Получать учетные данные для интеграций |
| `Db` и `SessionDb` | Работать с базами данных |
| `Email` | Отправлять email-сообщения |
| `Http` | Выполнять HTTP-запросы |
| `Llm` | Взаимодействовать с языковыми моделями |
| `Rag` | Работать с базами знаний RAG |
| `Reactions` | Отправлять ответы клиенту в подключенном канале |
| `Telegram` | Отправлять сообщения в Telegram без подключения канала |

### Asr и Tts

Для работы этих функций нужно подключить интеграции: *Модель ASR* и *Голос TTS*.

- `Asr.recognize` — получает файл по URL и распознает речь в нем.
- `Tts.synthesize` — синтезирует речь из текста и возвращает URL, по которому можно скачать аудиофайл.

Подробнее о работе с этими функциями читайте в статье ASR и TTS в текстовых каналах.

## Context

- **Получайте различные метаданные процесса**:
  - `getAccountId` — ID вашего аккаунта Just AI Agent Platform.
  - `getProjectShortName` — короткое имя проекта.
  - `getBotId`, `getChannelType`, `isAsyncChannel`, `isTestChannel` — ID и тип канала.
  - `getChatID`, `getSessionID` — идентификаторы диалога и текущей сессии.
  - `getClientInfo` — информация о пользователе, который взаимодействует с процессом.
- **Получайте подробную информацию о запросе пользователя**:
  - `getRequestId` — ID запроса.
  - `getMessageContent`, `getRawRequest` — контент сообщения и полные данные запроса.
- **Отслеживайте состояние диалога**:
  - `getChatHistory` — история сообщений в диалоге.
  - `getLastFunctionResult` — результат последнего вызова функции в процессе.

## Credentials

Получайте учетные данные для интеграций из хранилища с помощью функции `get` вместо непосредственной вставки в код.

## Db и SessionDb

С помощью `Db` вы можете работать с базами данных, которые вы настроили в разделе *Интеграции*.

Функции из коллекции `SessionDb` работают аналогично, но не требуют настройки интеграции с БД и обрабатывают данные только в рамках текущей сессии.

- **Получайте данные:**
  - `get` — получить запись по ключу.
  - `findByFilters` — найти одну или несколько записей по заданным фильтрам.
  - `countByFilters` — получить количество записей, которые соответствуют фильтрам.
- **Добавляйте и изменяйте данные:**

    - `put` — добавить новую запись в базу.
    - `updateByFilters` — обновить поля в записях, которые соответствуют фильтрам.
- **Удаляйте данные:**
    - `delete` — удалить запись по ключу.
    - `deleteByFilters` — удалить все записи, которые соответствуют фильтрам.

## Email

Отправляйте email-сообщения с помощью функции `send`.

## Http

- Функции для вызова HTTP-методов:
    - `get`
    - `post`
    - `put`
    - `delete`
    - `patch`
- `execute` — функция, в которой вы можете выбрать HTTP-метод сами.

## Llm

- `sendText` — отправляйте простые текстовые запросы к LLM.
- `sendRequest` — настраивайте все параметры запроса вручную.
- `getChatHistory` — получайте историю сообщений в специальном формате для LLM.

## Rag

Используйте интеграцию с базой знаний RAG:

- `retrieveChunks` — получить фрагменты документов, релевантные запросу пользователя.
- `generateAnswer` — сгенерировать ответ на вопрос пользователя с помощью RAG.

## Reactions

Отправляйте сообщения клиентам в подключенных каналах:

- `sendText` — текстовое сообщение.
- `sendHtml` — сообщение в разметке HTML.

- `sendAudio`, `sendImage`, `sendVideo`, `sendFile` — аудио, изображение, видео или файл.
- `sendLocation` — геолокация.
- `sendRawRequest` — данные для ответа в формате, который поддерживает канал.

### Telegram

Отправляйте сообщения в Telegram без подключения канала. В каждой функции укажите API-токен вашего Telegram-бота. Эти функции можно использовать вместо `Reactions`, например, в следующих случаях:

- Вы в процессе динамически определяете, какой Telegram-бот должен написать пользователю.
- Вы получили запрос от пользователя в одном Telegram-боте, но хотите отправить ответ из другого бота.

Функции:

- `sendText` — отправить текстовое сообщение.
- `sendImage`, `sendAudio`, `sendVoice` — отправить изображение, аудио или голосовое сообщение.
- `sendButtons` — отправить сообщение с кнопками.

## Асинхронность функций

Большинство встроенных функций являются асинхронными. Вы можете вызывать их с помощью `await`.

Синхронными являются только следующие функции:

- `Context`:
  - `getAccountId`
  - `getProjectShortName`
  - `getBotId`
  - `getChannelType`
  - `isAsyncChannel`
  - `isTestChannel`
  - `getChatID`
  - `getSessionID`
  - `getClientInfo`
  - `getRequestId`

    - `getMessageContent`
    - `getRawRequest`
    - `getLastFunctionResult`
- `Credentials.get`

# Пользовательские функции

Пользовательские функции — это мощный инструмент для расширения стандартных возможностей платформы. Они позволяют вам писать собственный JavaScript-код для решения уникальных задач: от сложных вычислений до интеграции со сторонними API.

Каждая созданная вами функция становится многоразовым компонентом, который можно использовать в качестве отдельного шага в процессе, в виде инструмента для AI-агента, а также вызывать из кода. Подробнее об этом в разделе Использование функций в процессе.

## Коллекции

Функции группируются в коллекции. При вызове функции в коде необходимо указывать коллекцию:

```
// Вызов функции myFunction из коллекции MyCollection
MyCollection.myFunction();
```

> **К СВЕДЕНИЮ**
>
> Пользовательские функции можно добавить только в пользовательские коллекции.
> Если в проекте еще нет таких коллекций, потребуется создать коллекцию при добавлении функции.

## Создание и настройка функции

Создание пользовательской функции включает несколько шагов:

1. Добавление функции.
2. Настройка параметров.
3. Написание кода.
4. Добавление зависимостей.

### Добавление функции

Добавить функцию можно в левом меню или непосредственно на холсте — эти способы эквивалентны.

**В меню**

1. В левом меню выберите , затем нажмите *Добавить → функцию*.
2. Выберите или создайте коллекцию.

   Для новой коллекции заполните поля:
   - *Название* коллекции для отображения в интерфейсе. Должно быть уникальным среди пользовательских коллекций.
   - *Идентификатор* коллекции для использования в коде. Должен быть уникальным в рамках проекта и не может совпадать со встроенными коллекциями.
   - *Описание* для отображения на странице *Функции*.
3. Для новой функции заполните поля:
   - *Название* функции для отображения в интерфейсе.
   - *Идентификатор* функции для использования в коде.
   - *Описание* — особенно важно, если функция будет использоваться в качестве инструмента для AI-агента. Помогает агенту понять, для чего следует вызывать эту функцию.
4. Нажмите *Сохранить*.

**На холсте**

1. Чтобы развернуть список блоков, нажмите в левом верхнем углу холста.
2. В списке блоков выберите *Функции*, затем в разделе *Пользовательские* нажмите +.
3. Выберите или создайте коллекцию.

   Для новой коллекции заполните поля:
   - *Название* коллекции для отображения в интерфейсе. Должно быть уникальным среди пользовательских коллекций.
   - *Идентификатор* коллекции для использования в коде. Должен быть уникальным в рамках проекта и не может совпадать со встроенными коллекциями.
   - *Описание* для отображения на странице *Функции*.
4. Для новой функции заполните поля:
   - *Название* функции для отображения в интерфейсе.
   - *Идентификатор* функции для использования в коде.
   - *Описание* — особенно важно, если функция будет использоваться в качестве инструмента для AI-агента, и помогает агенту понять, для чего следует вызывать эту функцию.
5. Нажмите *Сохранить*.

## Настройка параметров

Перейдите к редактированию параметров и кода функции.

**В меню**

1. В левом меню выберите
2. На странице *Функции* нажмите .

**На холсте**

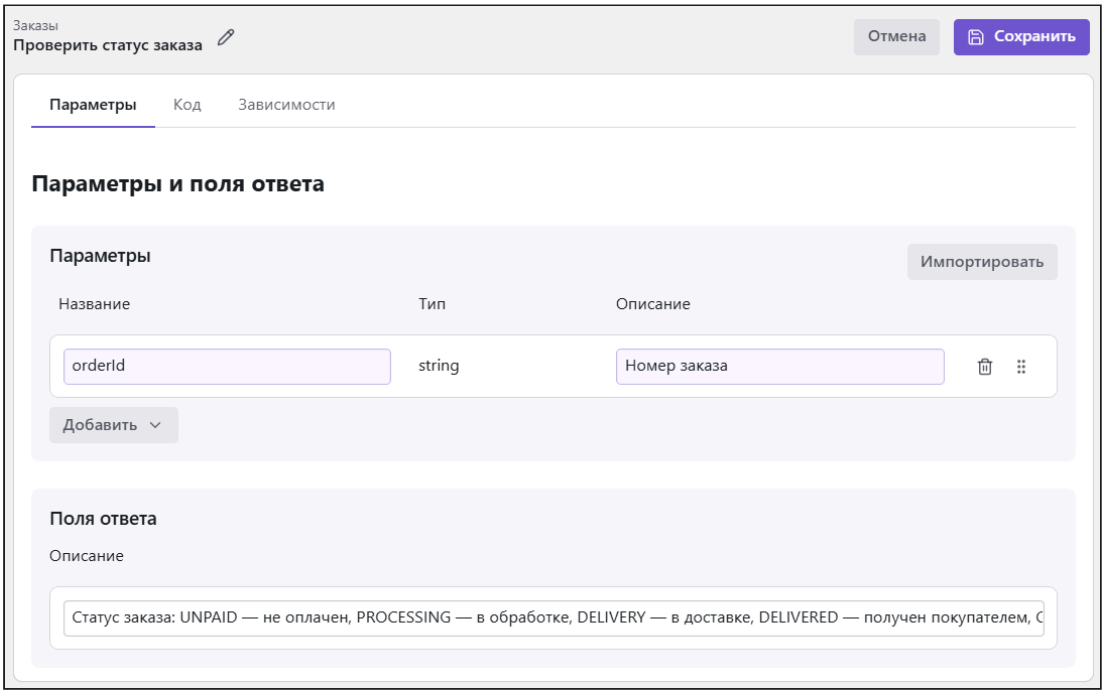
1. Перетащите функцию на холст.
2. На холсте нажмите на блок этой функции, чтобы открыть ее настройки.
3. Нажмите *Посмотреть код*.

На вкладке *Параметры* укажите:

- **Параметры функции**:
  - *Тип* параметра. Поддерживаются как простые типы (`string`, `number`, `boolean`), так и структуры (`object`, `array`) до 5 уровней вложенности.
  - *Название* для использования в коде.
  - *Описание* — помогает AI-агенту понять, что именно следует передавать в параметре.

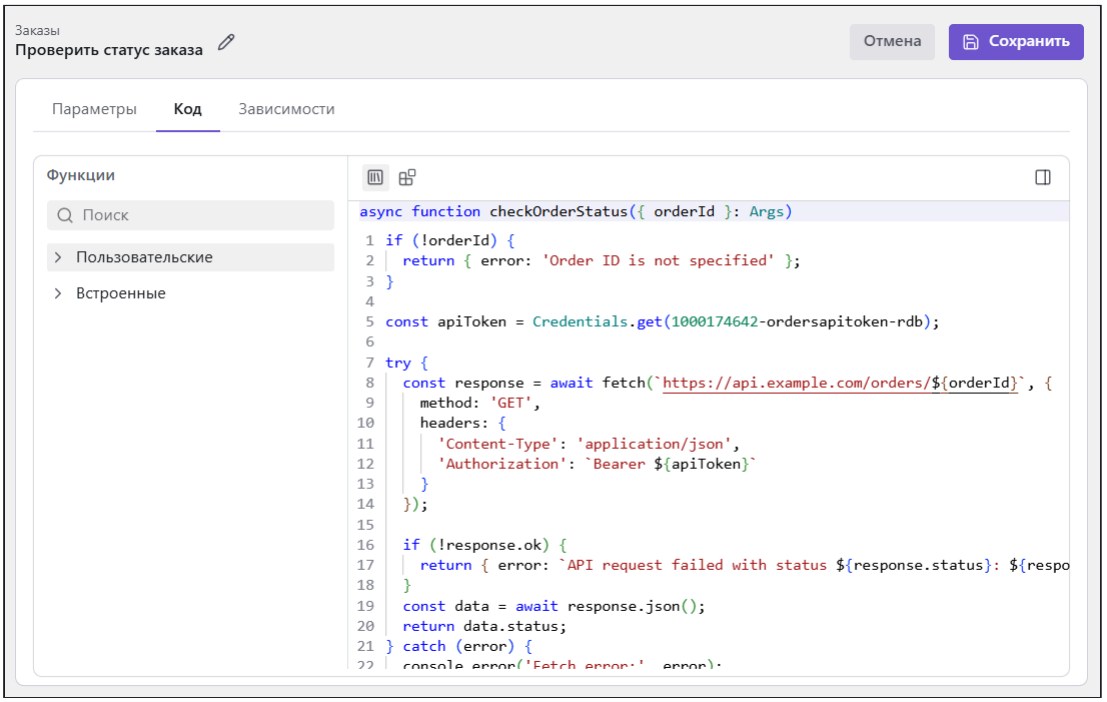
> **ПОДСКАЗКА**
>
> Вы можете загрузить параметры в виде JSON-схемы: нажмите *Импортировать* и скачайте образец.

- **Описание ответа**: помогает AI-агенту правильно интерпретировать результат работы функции.




# Написание кода

На вкладке *Код* напишите тело функции на JavaScript. Встроенный редактор поддерживает подсветку синтаксиса, автодополнение, подсказки при наведении на функцию и отображение ошибок.




Особенности и рекомендации:

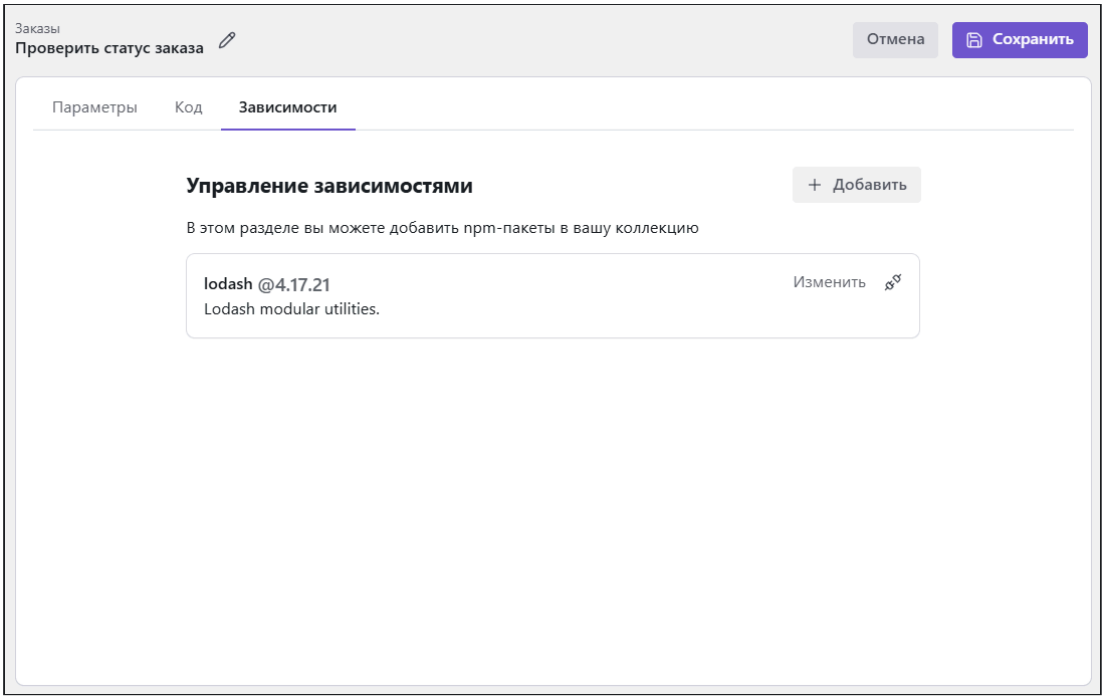
- Все пользовательские функции являются асинхронными (`async`). Это позволяет использовать в их коде `await` для асинхронных операций, например для выполнения HTTP-запросов с помощью `fetch()`.
- **Не вставляйте учетные данные и другие секреты в код функции.** Добавьте их в хранилище и используйте в коде с помощью встроенной функции `Credentials.get()`. Подробнее смотрите в разделе Учетные данные для интеграций.

Кнопки:

- — открыть панель *Функции*, где можно посмотреть список функций, найти нужную и вставить ее вызов в код двойным нажатием.
- — посмотреть подключенные пакеты.
- — посмотреть описание и парамеры функции, выбранной в панели *Функции*.

## Зависимости

Если для работы вашего кода требуются внешние npm-пакеты (например, `axios` или `lodash`), добавьте их на вкладке *Зависимости*.




Добавленные пакеты доступны для использования во всех функциях из той же коллекции. Вместе с тем, управление зависимостями выполняется на уровне проекта в целом. В частности, изменение версии пакета автоматически применяется ко всем коллекциям.

# Использование функций в процессе

Как встроенные, так и пользовательские функции можно использовать тремя способами:

- в качестве отдельного шага в процессе;
- в виде инструмента для AI-агента;
- в коде других блоков.

## Режим «Функция»

Базовый режим, чтобы использовать функцию как отдельный шаг в процессе.

- **Как работает:** система вызывает выбранную функцию с заданными параметрами. Для получения результата в следующем блоке используйте метод `Context.getLastFunctionResult()`.
- **Когда использовать:** для задач, где вызов внешней системы — это четкий шаг в процессе. Например: «получить курс валют», «рассчитать доставку», «проверить статус заказа».

#### Настройка

1. В левом меню перейдите в *Конструктор* → ⊞.
2. Выберите *Функции* → *Встроенные* или *Пользовательские* → *<Нужная коллекция>*.
3. Перетащите функцию на холст и укажите параметры. Вы можете задать их вручную или использовать JavaScript-выражения для динамической подстановки данных, например `{{'ID этого канала: ' + Context.getBotId()}}`.

## Режим «Инструмент»

Используется совместно с блоком *Агент*, позволяя AI самостоятельно решать, когда вызывать функцию.

- **Как работает:** вы не вызываете функцию напрямую — она становится «инструментом» для агента. Агент сам решает, когда и как ее использовать, исходя из диалога с пользователем.
- **Когда использовать:** для гибких систем, где агент взаимодействует с внешним миром: например, «забронируй переговорную» или «какой у меня баланс?».

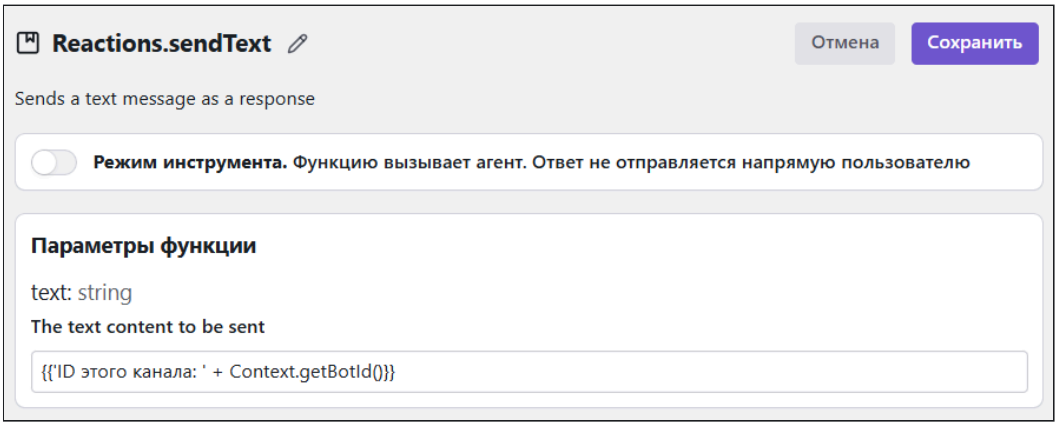
Подробнее о настройке функций в режиме инструмента смотрите в разделе про блок «Агент».

# Режим вызова

Функцию можно вызвать:

- Из JavaScript-выражений в блоках *Код* и *Условие*. Пример смотрите в разделах Код и Условие.
- Из кода других (пользовательских) функций.
- Из параметров других функций.

  Пример:




Добавлять функцию на холст при этом не требуется.

При вызове функций указывайте коллекцию, например:

```
// Вызов функции myFunction из коллекции MyCollection
MyCollection.myFunction();
```

# Триггеры

Триггеры — это стартовые блоки сценария. Триггер определяет, какое событие и при каких условиях запустит ваш процесс. Когда происходит это событие, триггер активируется и передает управление следующему блоку.

На холст можно добавить несколько триггеров разных типов. Таким образом, один сценарий может обрабатывать различные события.

На платформе доступны следующие типы триггеров:

- Вебхук — запускается по HTTP-запросу из внешней системы.
- Планировщик — активируется по заданному расписанию.
- Сообщение — реагирует на сообщения и события в текстовых каналах.

> ⚠ **ПРЕДУПРЕЖДЕНИЕ**
>
> - Триггеры *Вебхук* и *Планировщик* запускают процесс сразу для всех пользователей и каналов.

## Как добавить триггер на холст

1. Перейдите в проект и на левой панели нажмите , чтобы открыть конструктор.
2. В верхнем левом углу холста нажмите  — откроется список блоков.
3. Раскройте список *Триггеры* и перетащите нужный блок на холст.
4. Соедините триггер со следующим блоком в сценарии.

## Триггер «Вебхук»

Этот триггер запускает процесс по HTTP-запросу из внешней системы, например из CRM или ERP.

Особенности:

- Для каждого блока *Вебхук* на холсте автоматически генерируется уникальный URL.
- Вебхук не требует дополнительной аутентификации. Любой, у кого есть ссылка, может запустить связанный с ней процесс.
- Вебхук запускает процесс сразу для всех пользователей во всех каналах.

- Вебхук не передает в процесс никакую дополнительную информацию.

## Настройка

Нажмите на блок *Вебхук* и скопируйте уникальный URL.

Чтобы запустить процесс, внешняя система должна отправить GET-запрос на этот URL.
При успешном вызове сервер вернет `200 OK`, и сценарий будет запущен.

> **ПОДСКАЗКА**
>
> Чтобы протестировать вебхук, скопируйте его URL и откройте в браузере.

# Триггер «Планировщик»

Этот триггер запускает процесс по заданному расписанию. Он идеально подходит
для автоматизации рутинных задач, таких как формирование отчетов или рассылка сообщений.

Особенности:

- Планировщик запускает процесс сразу для всех пользователей во всех каналах.
- Планировщик не передает в процесс никакую дополнительную информацию.

## Настройка

1. Нажмите на блок *Планировщик*, чтобы открыть настройки.
2. Выберите вариант расписания:
   - *Простое* — для быстрой настройки с помощью готовых интервалов. Укажите дату и время первого запуска, выберите периодичность и условие окончания.

     > **ПРЕДУПРЕЖДЕНИЕ**
     >
     > Если время первого запуска уже прошло и выбран вариант *Не повторять*, то триггер запустится один раз сразу после сохранения.

   - *Cron-выражение* — для более гибкой настройки. Укажите Cron-выражение в формате Quartz и условие окончания. Например, `0 30 9 ? * MON` — каждый понедельник в 9:30.

   После настройки расписания платформа отобразит информацию о ближайших запланированных запусках (не более 5).
3. Нажмите *Запустить*.

**К СВЕДЕНИЮ**

Новое расписание начинает работать сразу, как только вы нажали *Запустить*, независимо от публикации процесса. Публикация нужна, только если вы меняли логику самого процесса: например, добавляли блоки или меняли текст сообщений.

Чтобы случайно не отправить сообщение во все каналы при редактировании триггера в уже опубликованном процессе, используйте для отладки блок Условие с функцией `Context.isTestChannel()`. Добавьте условие после планировщика, опубликуйте процесс, затем внесите изменения и протестируйте их в тестовом виджете.

# Триггер «Сообщение»

Этот триггер срабатывает, когда в подключенный канал поступает сообщение, файл, геолокация, контакт или стикер, а также при открытии окна тестового виджета.

Каналы работают независимо друг от друга. Ответ будет доставлен в тот канал, откуда пришло исходное событие.

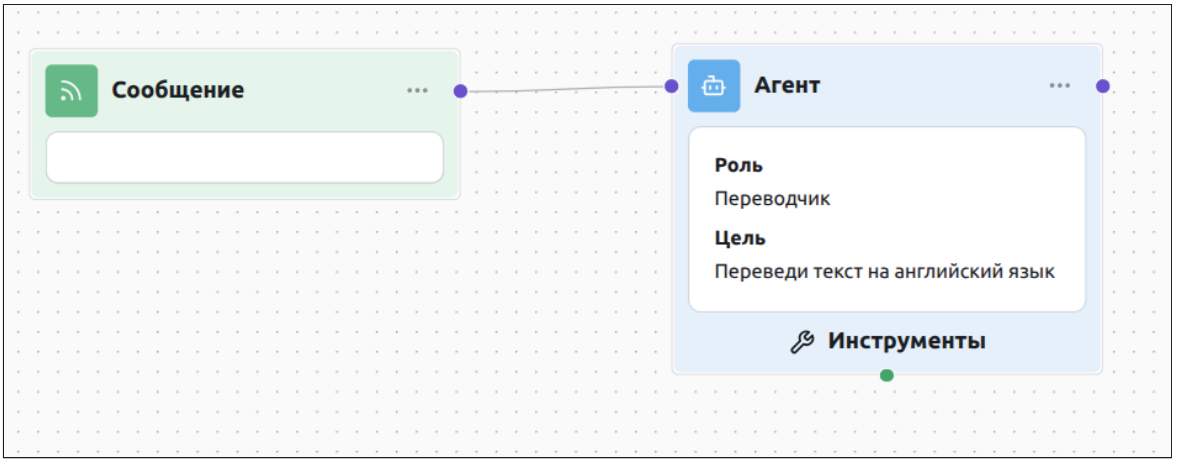
## Настройка

В триггере *Сообщение* автоматически подключены все настроенные в проекте каналы. Добавить или редактировать каналы можно как в разделе *Интеграции*, так и в самом триггере — эти способы эквиваленты. Если в проекте не подключен ни один канал, триггер срабатывает только в тестовом виджете.

## Использование

### С агентом

Соедините триггер *Сообщение* с блоком *Агент*. Агент обработает сообщение пользователя в соответствии со своим промтом и другими настройками. Примеры смотрите в разделах Быстрый старт: Шаг 2 и Работа агента в процессе.



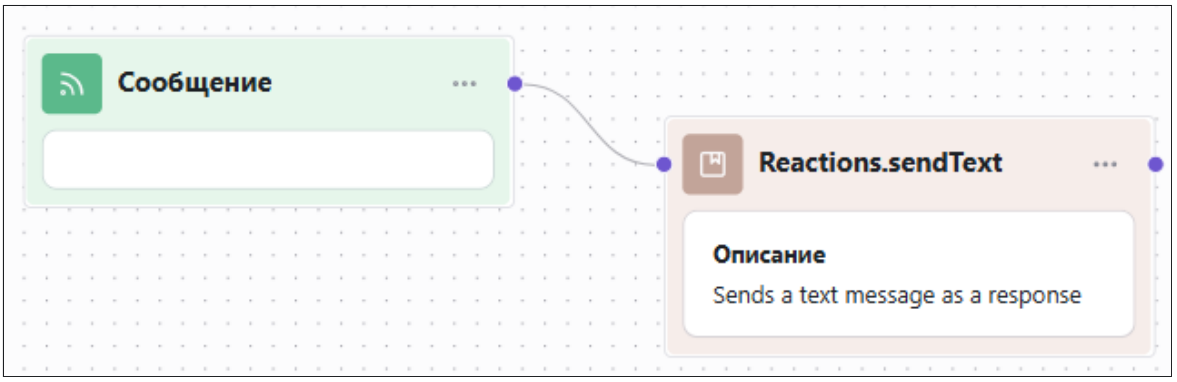

### С функцией

Триггер *Сообщение* можно соединить с функцией. Для извлечения различных данных о сообщении используйте встроенные функции из раздела *Context*. Например, функция `Context.getMessageContent()` возвращает содержание сообщения, `Context.getChannelType()` — тип канала.

Рассмотрим тривиальный пример эхо-бота:

1. Добавьте на холст триггер *Сообщение* и функцию `Reactions.sendText()`, соедините их последовательно.
2. В настройках функции `Reactions.sendText` укажите значение `{{"Вы сказали: " + Context.getMessageContent().text}}`.

Такой бот на сообщение «Привет» ответит «Вы сказали: Привет».



# Блок «Код»

С помощью блока «Код» вы можете добавить в процесс фрагмент JavaScript-кода. Это универсальный блок для вычислений, логики и работы с данными, контекстом и переменными.

> **ОТЛИЧИЕ ОТ ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИЙ**
>
> Используйте блок «Код» для кода, который будете выполнять один раз, а пользовательские функции — для кода, который будете переиспользовать.

## Как добавить блок «Код» на холст

1. Перейдите в проект и на левой панели нажмите , чтобы открыть конструктор процессов.
2. В верхнем левом углу холста нажмите  — откроется список блоков.
3. Из раздела *Код и логика* перетащите блок *Код* на холст.

## Интерфейс редактора кода

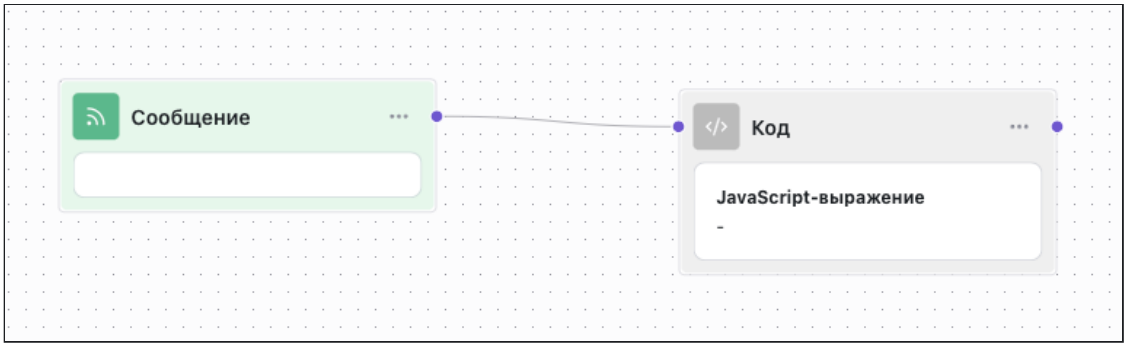
Чтобы настроить блок, нажмите на него на холсте. В этом блоке доступны следующие элементы:

- Редактор кода с подсветкой синтаксиса и автокомплитом.
- Библиотека функций . Панель быстрого поиска встроенных и пользовательских функций проекта. Двойной клик — вставка вызова функции в код.
- Панель документации. Отображает подсказки и описание выбранной функции: имя, тип, описание. Автоматически появляется в правой части окна при нажатии на любую функцию из библиотеки.

## Пример использования

Для примера создадим блок, который считает количество слов в сообщении.

1. Перейдите в проект и на левой панели нажмите , чтобы открыть конструктор процессов.
2. В верхнем левом углу холста нажмите  — откроется список блоков.
3. Перетащите блок *Триггеры → Сообщение* на холст.
4. Перетащите блок *Код и логика → Код* на холст и соедините его с блоком *Сообщение*.




5. Нажмите на блок *Код*.
6. Вставьте в него следующий код, в котором используются встроенные функции Context и Reactions:

```
// Получаем объект с сообщением
const content = await Context.getMessageContent();

// Проверяем, есть ли вообще текст
if (!content || typeof content.text !== 'string') {
  Reactions.sendText("Я не вижу сообщения");
  return;
}

// Извлекаем текст и убираем лишние пробелы
const message = content.text.trim();

// Разбиваем сообщение по пробелам и фильтруем пустые элементы
const words = message.split(/\s+/).filter(Boolean);

// Считаем количество слов
const count = words.length;

// Отправляем простой ответ
Reactions.sendText(`В твоём сообщении ${count} ${count === 1 ? "слово" : "слов(а)"}`);
```

7. Нажмите *Сохранить*.
8. В конструкторе нажмите *Тестировать*. Откроется окно чата, и вы сможете проверить работу вашего кода.

# Рекомендации

- Используйте явные имена переменных, чтобы вам сразу было понятно, что она делает. Например, если вы сохраняете в нее текст сообщения, то назовите ее `message`.
- Старайтесь не делать слишком большой и сложный код — лучше разбить на несколько функций и блоков.

# Блок «Условие»

Блок «Условие» позволяет создавать ветвления внутри процесса. От результата проверки условия зависит, по какому пути продолжится выполнение процесса.

Этот блок подходит для:

- Направления сценария в зависимости от полученных данных.
- Обработки результатов внешних API.
- Реализации гибкой пошаговой логики.

## Как добавить блок «Условие» на холст

1. Перейдите в проект и на левой панели нажмите , чтобы открыть конструктор процессов.
2. В верхнем левом углу холста нажмите  — откроется список блоков.
3. Перетащите блок *Код и логика → Условие* на холст.

## Интерфейс настройки условия

После добавления блока на холст вы можете задать логику и указать, какая ветка будет выполнена при выполнении или невыполнении условия.

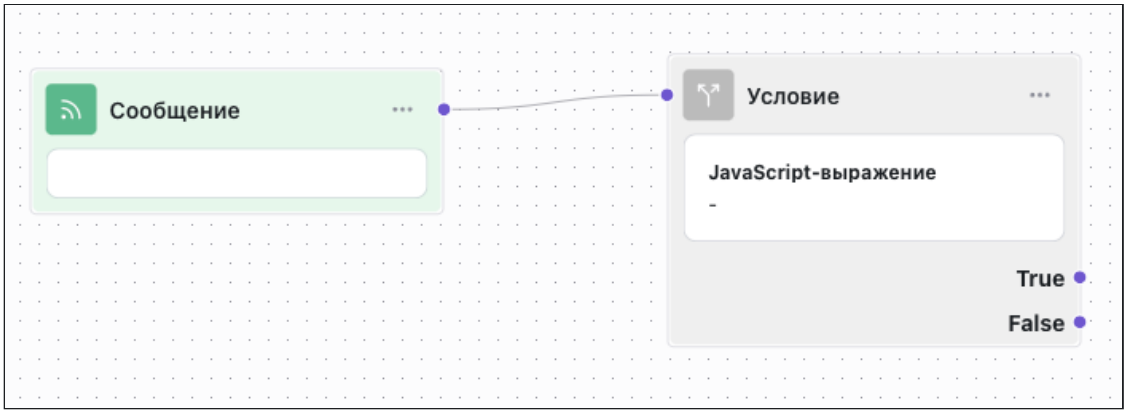
1. *JavaScript-выражение:* логическое выражение в JavaScript.
2. *Если True — перейти в* и *Если False — перейти в:* из выпадающего списка выберите следующий блок.

Вы также можете перетащить соединение от *True* или *False* к нужному блоку в конструкторе.

## Примеры использования

Для примера проверим, будет ли значение числа больше 100 или нет.

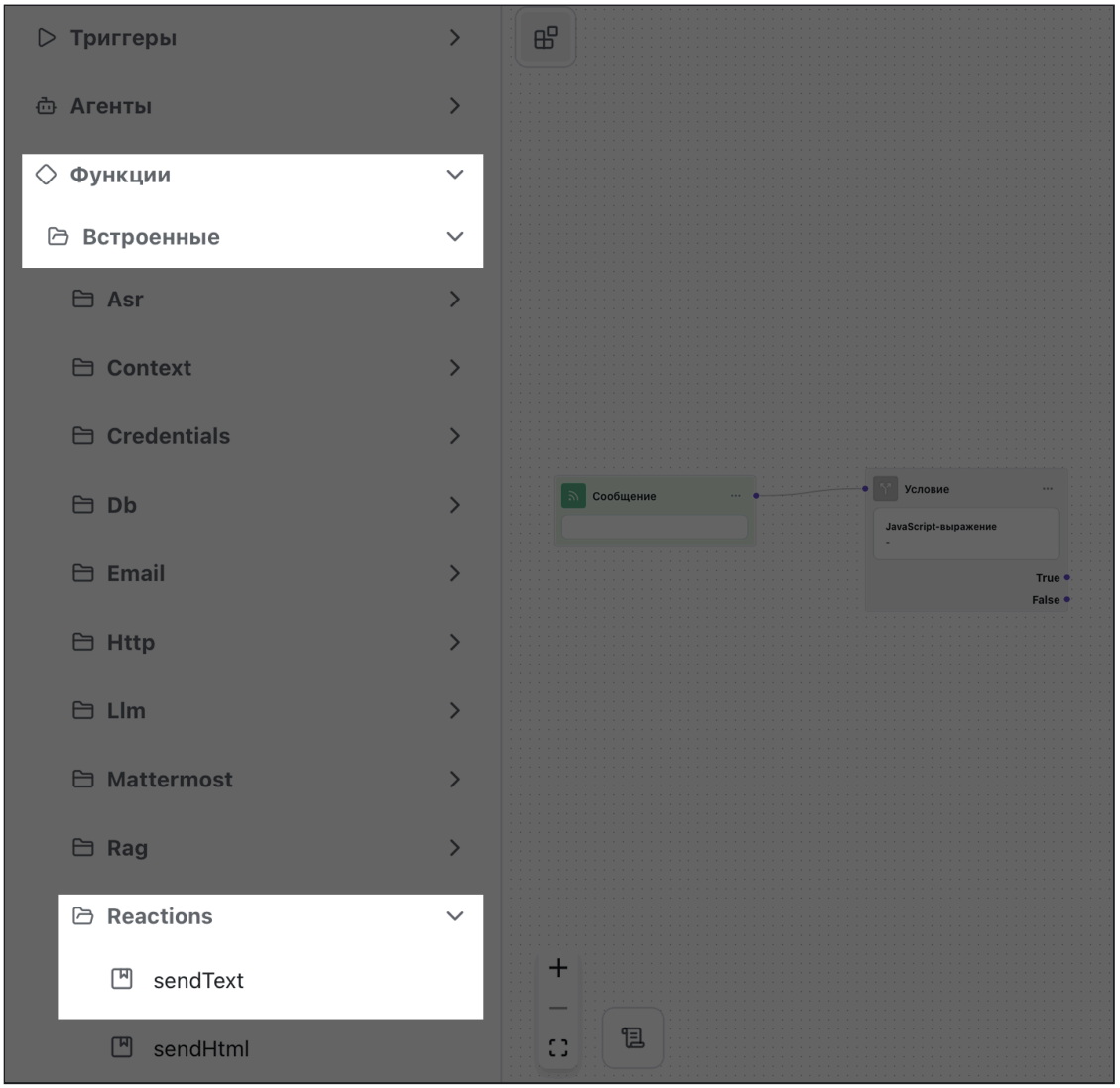
1. Перейдите в проект и на левой панели нажмите , чтобы открыть конструктор процессов.
2. В верхнем левом углу холста нажмите  — откроется список блоков.
3. Перетащите блок *Триггеры → Сообщение* на холст.
4. Перетащите блок *Код и логика → Условие* на холст и соедините его с блоком *Сообщение*.



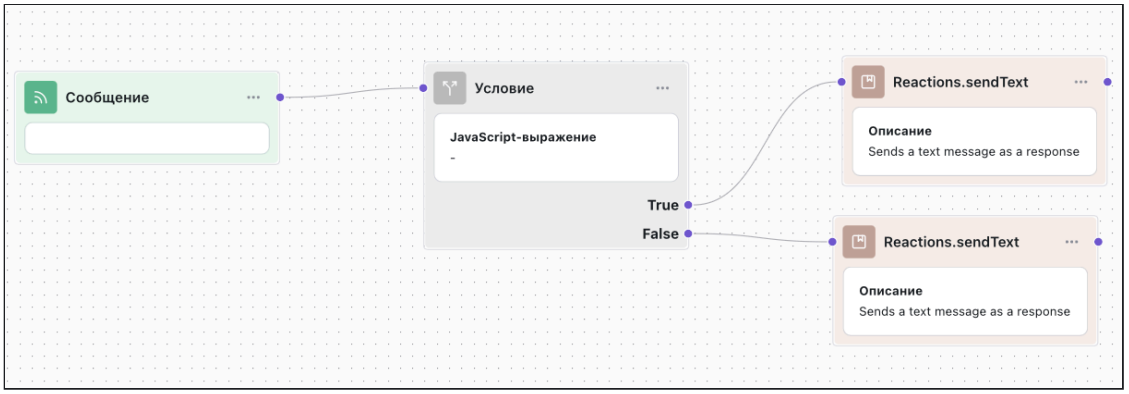

5. Нажмите на блок *Условие* на холсте, чтобы открыть его.
6. В поле *JavaScript-выражение* вставьте:

```
parseInt(Context.getMessageContent().text, 10) > 100
```

7. Перетащите на холст два блока *Функции* → *Встроенные* → *Reactions* → *SendText*.




8. Соедините один блок *SendText* со значением *True*, а другой со значением *False* на блоке *Условие*.




9. Откройте блок, который соединен с *True*, и в параметр *text* вставьте `Больше 100`.
10. Откройте блок, который соединен с *False*, и в параметр *text* вставьте `Меньше 100`.
11. В конструкторе нажмите *Тестировать*. Откроется окно чата, и вы сможете проверить работу вашего процесса. Отправьте в чат число, а вам придет ответ с результатом сравнения.

# Рекомендации

- Используйте блок *Код* до блока *Условие* для подготовки данных.
- Проверяйте ошибки в логах, если условие ведет не по тому сценарию.

# Как настроить звонки

Just AI Agent Platform позволяет обрабатывать входящие звонки клиентов на ваши телефонные номера.

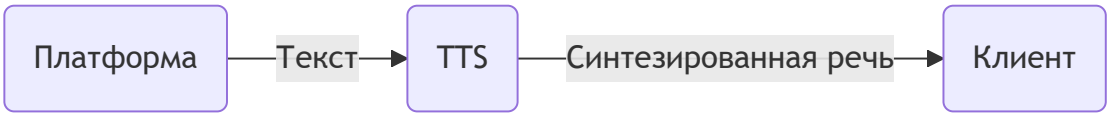
Для обработки звонков потребуются:

- **SIP-транк** — услуга, которую вы можете получить у сторонних провайдеров SIP-телефонии. Транк позволяет принимать множество звонков одновременно на один номер.
- **Аккаунты у провайдеров ASR и TTS** для распознавания и синтеза речи.

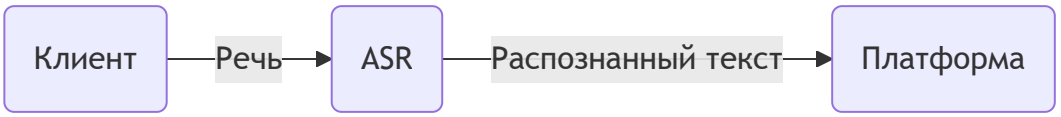
## ASR и TTS

Чтобы разговаривать с клиентом, Just AI Agent Platform использует синтез (TTS) и распознавание речи (ASR):

- Когда Just AI Agent Platform отправляет сообщение клиенту, TTS преобразует этот текст в речь.




- Когда клиент говорит в телефонном канале, ASR преобразует речь в текст.




### Поддерживаемые провайдеры

| Провайдер | ASR | TTS |
|---|---|---|
| 3i VoxKit | ✓ | ✓ |
| Audiogram | ✓ | ✓ |
| SaluteSpeech (Сбер) | ✓ | ✓ |
| T-Bank VoiceKit | ✓ | — |
| Yandex SpeechKit | ✓ | ✓ |

# Как настроить входящие звонки

## Шаг 1. Настройте SIP-подключение

Чтобы принимать входящие звонки, настройте подключение к SIP-серверу вашего провайдера.

Добавьте учетные данные:

1. В проекте перейдите в раздел *Учетные данные*.
2. Добавьте новые учетные данные с типом *SIP-аккаунт*. Используйте данные, предоставленные вашим провайдером SIP-телефонии.

Далее создайте интеграцию:

1. Перейдите в раздел *Интеграции*.
2. Создайте новую интеграцию с типом *SIP-подключение*.

   Подробнее о настройках читайте в статье SIP-подключение.

## Шаг 2. Настройте ASR и TTS

Добавьте учетные данные для провайдера ASR/TTS:

1. В проекте перейдите в раздел *Учетные данные*.
2. Добавьте новые учетные данные с типом *Провайдер ASR/TTS*.

   Эти данные можно использовать как для ASR, так и для TTS. Если у вас провайдеры ASR и TTS разные, создайте по одному набору учетных данных для каждого провайдера.

   Подробнее о нужных данных читайте в статье Учетные данные для ASR/TTS.

Далее создайте интеграции для ASR и TTS:

1. Перейдите в раздел *Интеграции*.
2. Создайте две интеграции: одну с типом *Модель ASR*, другую — с типом *Голос TTS*.

   > **ПРЕДУПРЕЖДЕНИЕ**
   >
   > Даже если вы используете одного провайдера для обеих функций, вам все равно нужно создать две отдельные интеграции.

   Подробнее о настройках для интеграций читайте в статье Модели ASR и голоса TTS.

## Шаг 3. Подключите телефонный канал

1. Перейдите в раздел *Конструктор*.
2. Перетащите триггер *Сообщение* на холст и нажмите на этот блок, чтобы открыть настройки.

> **ПОДСКАЗКА**
>
> *Сообщение* — универсальный триггер, который может срабатывать как на текстовые сообщения, так и на звонки.

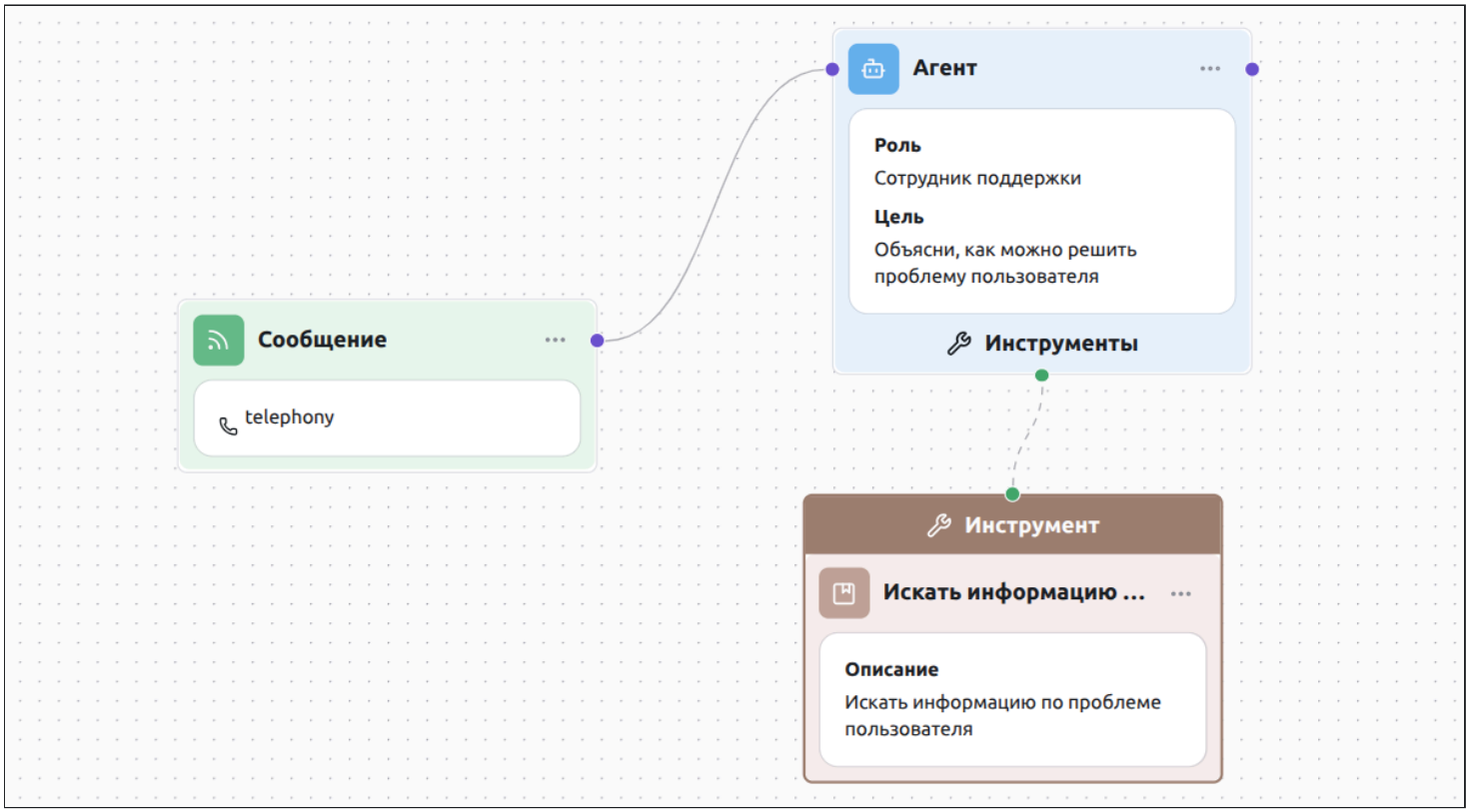
3. Нажмите *Подключить канал* и выберите *Телефонный канал*.
4. В настройках канала:
   - Выберите уже созданные интеграции SIP, ASR и TTS.
   - Включите опцию *Принимать входящие звонки*.
5. Нажмите *Сохранить*.

Далее соедините этот триггер с той частью процесса, которая будет обрабатывать звонки.

# Пример процесса для обработки звонков

Предположим, у вас есть простой сценарий с одним агентом:

- Агент выполняет роль сотрудника поддержки, он должен помочь клиенту с его вопросами.
- У агента есть инструмент для поиска по базе знаний, чтобы он мог находить ответы на похожие вопросы.
- В триггере *Сообщение* подключен телефонный канал.



Вот как будет работать этот процесс:

1. Клиент звонит на номер, связанный с вашим SIP-транком.
2. Just AI Agent Platform принимает звонок. Триггер *Сообщение* срабатывает и отправляет в агента системное сообщение `/start`.
3. Агент генерирует приветственный текст. TTS озвучивает его, и клиент слышит приветствие.
4. Клиент задает вопрос. ASR распознает речь, текст вопроса поступает в агента.
5. Агент ищет ответ в базе знаний и генерирует текст ответа. TTS синтезирует речь из ответа агента, и клиент слышит ответ по телефону.

# ASR и TTS в текстовых каналах

Вы можете использовать ASR и TTS не только в звонках, но и в текстовых каналах. В Just AI Agent Platform доступны функции:

- `Asr.recognize` — получает файл по URL и распознает речь в нем.

  С помощью этой функции вы можете, например, распознать речь в записи совещания. Полученный текст можно передать в агента и попросить составить краткое содержание встречи.
- `Tts.synthesize` — синтезирует речь из текста и возвращает URL, по которому можно скачать аудиофайл.

  Например, вы можете попросить агента подготовить текст для подкаста и далее озвучить его с помощью этой функции.

# ASR

## Как настроить интеграцию

Функция `Asr.recognize` использует интеграцию типа *Модель ASR*.

Чтобы создать такую интеграцию:

1. Добавьте учетные данные для провайдера ASR/TTS:
   i. В проекте перейдите в раздел *Учетные данные*.
   ii. Добавьте новые учетные данные с типом *Провайдер ASR/TTS*.

   Подробнее о нужных данных читайте в статье Учетные данные для ASR/TTS.
2. Перейдите в раздел *Интеграции*.
3. Создайте интеграцию с типом *Модель ASR*.

   Подробнее о настройках для интеграции читайте в статье Модели ASR и голоса TTS.

## Вызов функции

Функция `Asr.recognize` принимает следующие параметры:

- `asrIntegrationKey` — идентификатор интеграции *Модель ASR*. В разделе *Интеграции* вы можете скопировать этот идентификатор на карточке интеграции. Пример: `1000001-name-abC`.

- `audioUrl` — URL аудиофайла, в котором нужно распознать речь. Файл должен быть доступен из интернета без авторизации.

Функция возвращает объект с результатами распознавания. Пример:


```
{
   "metadata":{
      "initiatedAt":1763637249747,
      "requestId":"1abc3def-1a23-4567-8901-164aaa3d6662"
   },
   "settings":{
      "type":"YANDEX",
      "model":"general",
      "languageCode":"ru-RU",
      "numbersAsWords":true,
      "sensitivityReduction":true
   },
   "result":{
      "text":"хочу отменить заказ"
   }
}
```


Здесь:

- `metadata` — метаданные запроса.
- `settings` — настройки интеграции ASR, которые использовались для распознавания.
- `result.text` — результат распознавания.

# TTS

## Как настроить интеграцию

Функция `Tts.synthesize` использует интеграцию типа *Голос TTS*.

Чтобы создать такую интеграцию:

1. Добавьте учетные данные для провайдера ASR/TTS:
   i. В проекте перейдите в раздел *Учетные данные*.
   ii. Добавьте новые учетные данные с типом *Провайдер ASR/TTS*.
   Подробнее о нужных данных читайте в статье Учетные данные для ASR/TTS.
2. Перейдите в раздел *Интеграции*.
3. Создайте интеграцию с типом *Голос TTS*.

Подробнее о настройках для интеграции читайте в статье Модели ASR и голоса TTS.

## Вызов функции

Функция `Tts.synthesize` принимает следующие параметры:

- `ttsIntegrationKey` — идентификатор интеграции *Голос TTS*. В разделе *Интеграции* вы можете скопировать этот идентификатор на карточке интеграции. Пример: `1000001-name-abC`.
- `text` — текст для синтеза речи.
- `languageCode` — код языка, на котором будет синтезироваться речь. Возможные значения: `DE`, `KK` или `KZ`, `RU`, `US`, `UZ`. Ваш TTS-провайдер должен поддерживать выбранный язык.
- `audioFormat` — формат аудиофайла: `WAV`.

Функция возвращает объект с результатами синтеза речи. Пример:

```
{
   "settings":{
      "type":"YANDEX",
      "languageCode":"ru-RU",
      "voice":"alena",
      "speakingRate":1,
      "volume":-19,
      "role":"neutral",
      "useVariables":false
   },
   "result":{
      "url":"https://a12345b6-1234-41ec-ab00-790c5c412a3b.example.com/tts-cache/a123b45c-a123-45bc-d678-eef86d26abc1.wav"
   }
}
```

Здесь:

- `settings` — настройки интеграции TTS, которые использовались для синтеза речи.
- `result.url` — публичный URL, по которому можно скачать сгенерированный аудиофайл. Файл хранится на сервере бессрочно.

# Подключение баз знаний (RAG)

Использование баз знаний (RAG, от англ. Retrieval-Augmented Generation) — один из способов наделить AI-агента экспертизой в конкретной области.

Базы знаний позволяют агенту находить ответы в документах, статьях и других источниках. Благодаря этому он генерирует точные и релевантные ответы, основанные на ваших данных, а не на общих знаниях модели, которых может быть недостаточно.

Платформа Just AI Agent Platform имеет встроенную интеграцию с базами знаний, которые размещаются в Jay Knowledge Hub — платформе для создания, хранения и управления знаниями от компании Just AI. Если у вас есть аккаунт в Just AI Agent Platform, вы можете использовать его для доступа к Jay Knowledge Hub без дополнительной регистрации.

## Создание подключения

Подготовьте в Jay Knowledge Hub базу знаний с документами, которые агент будет использовать для поиска ответов. Задайте необходимые настройки поиска релевантных фрагментов и генерации ответов — они будут использоваться при вызове функций из Just AI Agent Platform. Подробнее о создании базы знаний смотрите в документации Jay Knowledge Hub.

> **К СВЕДЕНИЮ**
>
> Just AI Agent Platform и Jay Knowledge Hub поддерживают работу в нескольких аккаунтах — группах проектов. Вы можете использовать одинаковые или разные аккаунты в Just AI Agent Platform и Jay Knowledge Hub. Это будет влиять на способ подключения.
>
> Выбор аккаунта в Just AI Agent Platform

1. В левом меню перейдите в раздел *Интеграции*.
2. Нажмите *Добавить* и выберите *База знаний RAG*.
3. В поле *Доступ к базе знаний* выберите, как вы будете подключаться к базе знаний:
   - *Текущий аккаунт* — если база знаний находится в том же аккаунте Jay Knowledge Hub, что и проект в Just AI Agent Platform.
   - *Другие учетные данные* — если база знаний находится в другом аккаунте Jay Knowledge Hub.

4. Заполните параметры в зависимости от выбранного способа подключения:
   - **Текущий аккаунт**
     - Выберите нужный проект с базой знаний из списка.
     - Укажите название интеграции — оно будет отображаться на странице с интеграциями.
   - **Другие учетные данные**
     - Выберите учетные данные для доступа к нужному аккаунту Jay Knowledge Hub из списка или создайте новые.

       API-ключ для доступа к базе знаний можно получить в личном кабинете Jay Knowledge Hub. Вам необходимо создать ключ типа «Проектный».
     - Укажите название интеграции — оно будет отображаться на странице с интеграциями.
5. Нажмите *Сохранить*.

Интеграция появится в списке и получит уникальный идентификатор вида `123456789-kb_name-id`. Он указан на карточке интеграции. Этот идентификатор необходим для вызова функций `Rag` в процессах — передайте его в параметре `ragIntegrationKey`.

# Использование в процессе

После создания интеграции вы сможете использовать встроенные функции `Rag` в конструкторе процессов:

- `retrieveChunks` — ищет в базе знаний фрагменты документов, релевантные запросу. Полученные фрагменты можно передать агенту для генерации ответа или дополнительной обработки. В этом случае функцию нужно использовать как инструмент.
- `generateAnswer` — генерирует ответ на вопрос на стороне Jay Knowledge Hub, используя базу знаний. Эту функцию лучше использовать как отдельный шаг в процессе.

## Режим «Функция»

Базовый режим, чтобы использовать функцию как отдельный шаг в процессе.

- **Как работает:** система вызывает выбранную функцию с заданными параметрами. Для получения результата в следующем блоке используйте метод `Context.getLastFunctionResult()`.
- **Когда использовать:** для задач, где вызов внешней системы — это четкий шаг в процессе. Например: «получить курс валют», «рассчитать доставку», «проверить статус заказа».

**Настройка**

1. В левом меню перейдите в *Конструктор* → ⊞.

2. Выберите *Функции → Встроенные → Rag*.
3. Перетащите функцию на холст и укажите параметры. Вы можете задать их вручную или использовать JavaScript-выражения для динамической подстановки данных, например `{{'ID этого канала: ' + Context.getBotId()}}`.

## Режим «Инструмент»

Используется совместно с блоком *Агент*, позволяя AI самостоятельно решать, когда вызывать функцию.

- **Как работает:** вы не вызываете функцию напрямую — она становится «инструментом» для агента. Агент сам решает, когда и как ее использовать, исходя из диалога с пользователем.
- **Когда использовать:** для гибких систем, где агент взаимодействует с внешним миром: например, «забронируй переговорную» или «какой у меня баланс?».

Подробнее о настройке функций в режиме инструмента смотрите в разделе про блок «Агент».

# Подключение MCP-серверов

Интеграция с MCP-серверами (от англ. Model Context Protocol) — ключевой способ расширить возможности вашей системы. MCP-серверы позволяют AI-агентам и процессам безопасно взаимодействовать с внешними системами, базами данных и сервисами. После подключения MCP-сервера вы сможете **использовать его функции** как шаги в бизнес-процессах или как инструменты для AI-агента.

Платформа Just AI Agent Platform поддерживает подключение к разным типам MCP-серверов:

- Публичные реестры и маркетплейсы (например, Smithery).
- Приватные self-hosted инсталляции.
- Сервисы с прямым доступом по URL.

> **ОСОБЕННОСТИ РАБОТЫ С MCP-СЕРВЕРАМИ**
>
> - Стабильность и доступность: MCP-серверы — это внешние сервисы. Платформа Just AI Agent Platform не отвечает за их работу и доступность.
> - Потребление контекста: вызов функций и их описание занимают значительную часть контекстного окна, что может ухудшить качество работы моделей с небольшим контекстом. Рекомендуем использовать модели с большим контекстным окном.
> - Стоимость: использование функций MCP-серверов может быть платным. Уточняйте тарифы у провайдера сервиса.

## Создание подключения

> **К СВЕДЕНИЮ**
>
> Платформа Just AI Agent Platform находится в активной разработке и тексты интерфейса могут меняться.

Чтобы добавить MCP-сервер в проект, выполните следующие шаги:

1. В левом меню перейдите в раздел *Интеграции*.
2. Нажмите *Добавить* и выберите *MCP*.
3. Заполните параметры:
    - *Название* — укажите понятное имя, которое будет отображаться в списке интеграций. Например, `Калькулятор тарифов` или `API базы знаний`.

- *URL MCP-сервера* — введите адрес, по которому доступен сервер.
- *Имя коллекции* — укажите имя для группы функций, полученных с сервера. Оно будет использоваться как префикс и поможет избежать конфликтов при нескольких MCP-интеграциях.

  По нему вы также сможете найти функции в конструкторе.
- *Учетные данные* — если требуется аутентификация (например, API-ключ), выберите учетные данные из списка или создайте новые. Если доступ открыт, оставьте поле пустым.

4. Нажмите *Сохранить*.

Платформа автоматически подключится к серверу, получит список доступных функций и их параметры. Теперь вы можете использовать их в процессах.

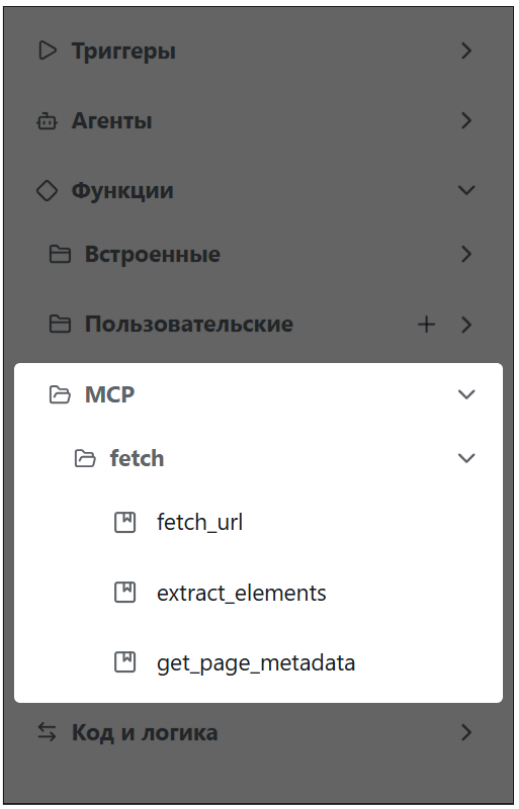
## Пример: подключение через Smithery

Рассмотрим подключение на примере MCP-сервера из публичного реестра Smithery. Этот сервер предоставляет функции для получения содержимого веб-страниц и извлечения данных.

При добавлении интеграции укажем следующие параметры:

- *Название:* `Fetch`
- *URL MCP-сервера:* `https://server.smithery.ai/@smithery-ai/fetch/mcp?api_key=<server_api_key>` — это адрес MCP-сервера с переданным в URL API-ключом для доступа к функциям. Чтобы его получить на сайте Smithery, нажмите *Get URL with keys instead*.
- *Имя коллекции:* `fetch`
- *Учетные данные:* оставим пустым, так как API-ключ уже передан в URL.

После сохранения интеграции она появится в списке, а в конструкторе процессов вы сможете найти функции `fetch.fetch_url`, `fetch.extract_elements` и `fetch.get_page_metadata`:




# Использование в процессе

После подключения вы можете использовать функции MCP на холсте.

## Режим «Функция»

Базовый режим, чтобы использовать функцию как отдельный шаг в процессе.

- **Как работает:** система вызывает выбранную функцию с заданными параметрами. Для получения результата в следующем блоке используйте метод `Context.getLastFunctionResult()`.
- **Когда использовать:** для задач, где вызов внешней системы — это четкий шаг в процессе. Например: «получить курс валют», «рассчитать доставку», «проверить статус заказа».

### Настройка

1. В левом меню перейдите в *Конструктор* → ⊞.
2. Выберите *Функции* → *MCP*.
3. Перетащите функцию на холст и укажите параметры. Вы можете задать их вручную или использовать JavaScript-выражения для динамической подстановки данных, например `{{'ID этого канала: ' + Context.getBotId()}}`.

## Режим «Инструмент»

Используется совместно с блоком *Агент*, позволяя AI самостоятельно решать, когда вызывать функцию.

- **Как работает:** вы не вызываете функцию напрямую — она становится «инструментом» для агента. Агент сам решает, когда и как ее использовать, исходя из диалога с пользователем.
- **Когда использовать:** для гибких систем, где агент взаимодействует с внешним миром: например, «забронируй переговорную» или «какой у меня баланс?».

Подробнее о настройке функций в режиме инструмента смотрите в разделе про блок «Агент».

# Каналы общения

Канал — интерфейс для диалога клиентов с агентами. Вы можете управлять каналами в разделе *Интеграции*, все подключенные каналы также отображаются в конструкторе, в триггере Сообщение.

Когда вы будете получать запрос через триггер «Сообщение», он пройдет через весь процесс, и затем ответ будет возвращен в тот канал, откуда пришел запрос.

Для некоторых каналов также доступны специальные встроенные функции. Так, например, функции для Telegram можно использовать, чтобы отправлять сообщения в каналы, когда сценарий начинается не по сообщению из канала или нужно отправить сообщение не в тот канал, из которого пришёл исходный запрос пользователя.

> **ПРИМЕЧАНИЕ**
>
> Just AI Agent Platform поддерживает мультиканальность: любой процесс может быть опубликован одновременно в нескольких каналах. Например, если вы разрабатываете агента, который должен помогать клиентам на сайте и в мессенджере, вы можете подключить к проекту каналы чат-виджет и Telegram. Диалоги в каждом канале ведутся независимо друг от друга — контекст и история сообщений не передаются между каналами.

# Доступные каналы

Just AI Agent Platform поддерживает следующие каналы:

- Chat API — REST API для интеграции в сторонние приложения. Например, для реализации чата в мобильном приложении, на сайте или в игре.
- edna Chat Center — платформа для организации продаж и поддержки клиентов в цифровых каналах.
- Mattermost — мессенджер, который используется в качестве внутреннего чата в организациях и компаниях.
- Prompter API — позволяет подключить суфлеров в Aimychat или интегрировать их в сторонний сервис.
- Telegram — один из самых используемых мессенджеров в России.
- Чат-виджет — скрипт, который вы сможете вставить на свой сайт, через него можно будет общаться с агентом.

# Подключение канала

Добавить новый канал в проекте можно несколькими способами:

- В проекте перейдите на вкладку *Интеграции*, выберите *Подключить → Канал*.
- На холсте выберите *Сообщение → Подключить канал*.

Дальше вам нужно выбрать тип канала и настроить его.

## Настройка нового канала

1. Выберите нужный тип канала.
2. Задайте название. Оно будет отображаться в списке интеграций.
3. Задайте настройки канала. Подробную информацию с настройками каждого канала можно найти на странице документации о нем.
4. Выберите, публиковать ли процесс в этом канале автоматически.
5. Нажмите *Подключить*.

Чтобы отредактировать параметры подключения или удалить созданный канал, нажмите ⋮.

# Как подключить edna Chat Center

edna Chat Center — платформа для ведения продаж и клиентской поддержки через цифровые каналы. Позволяет подключать чат-ботов, настраивать маршрутизацию обращений и использовать их в веб-интерфейсе и мессенджерах.

Чтобы опубликовать процесс в edna Chat Center:

1. Подключите канал edna Chat Center в проект.
2. Настройте чат-бота в аккаунте edna Chat Center.
3. Завершите настройку подключения в Just AI Agent Platform.
4. Протестируйте процесс.

## Подключение канала

Добавить новый канал в проекте можно несколькими способами:

- В проекте перейдите в раздел *Интеграции*, выберите *Подключить* → *Канал* → *edna Chat Center*.
- На холсте выберите *Сообщение* → *Подключить канал* → *edna Chat Center*.

1. Задайте название. Оно будет отображаться в списке интеграций.
2. Укажите *Адрес аккаунта* — это URL вида `https://YourAddress.edna.io`.
3. Выберите способ публикации изменений:
   - *Автоматически* — изменения публикуются, если вы нажали *Опубликовать* на холсте.
   - *Вручную* — чтобы опубликовать изменения:
     a. В конструкторе нажмите на триггер *Сообщение*.
     b. Нажмите *Опубликовать* у нужного канала.
4. Оставьте поле *Токен* пустым. Его вы заполните после настройки чат-бота в edna Chat Center.
5. Нажмите *Подключить*.
6. После создания канала перейдите в раздел *Интеграции*. Наведите курсор на карточку интеграции и нажмите ⋮ → *Редактировать*.
7. Скопируйте адрес вебхука.

## Настройка чат-бота в edna Chat Center

1. Войдите в ваш аккаунт edna Chat Center с правами администратора.
2. Перейдите в меню *Интеграции* → *Чат-бот*.
3. Нажмите ＋, чтобы создать нового бота.
4. Укажите:
   - *Имя чат-бота* — так бот будет называться в edna.
   - Вставьте скопированный адрес вебхука во все три поля:
     - *URL бекенда чат-бота для обработки сообщений клиента*.
     - *URL бекенда чат-бота для инициации диалога с клиентом*.
     - *URL бекенда чат-бота для обработки закрытия веб-формы в чате* (необязательно).
   - *Таймаут ожидания ответа* — укажите при необходимости. Если таймаут не указан, применятся настройки по умолчанию.
5. Нажмите *Сохранить*.
6. Скопируйте токен, нажав *Token* напротив бота. Он будет показан только один раз — сохраните его.
7. Перейдите в *Маршрутизация* → *Маршруты* и нажмите ＋:
   - Укажите название маршрута.
   - Выберите нужный сегмент (или создайте новый).
   - В типе маршрута выберите *Чат-бот*, затем укажите только что созданного бота.
   - Нажмите *Сохранить*.

# Завершение настройки в Just AI Agent Platform

1. Вернитесь в раздел *Интеграции* в Just AI Agent Platform.
2. Наведите курсор на карточку интеграции с edna Chat Center. Нажмите ⋮ → *Редактировать*.
3. Вставьте ранее скопированный токен в соответствующее поле. Если в этом поле уже что-то есть, замените это тем токеном, который скопировали.
4. Сохраните изменения.

# Тестирование процесса

1. Опубликуйте процесс в канал edna.
2. Перейдите в окно диалога с ботом в edna Chat Center и напишите ему.

После этого должен начаться ваш диалог. Бот должен ответить так, как вы настроили в процессе Just AI Agent Platform.

# Mattermost

Mattermost — мессенджер, который используется в качестве внутреннего чата в организациях и компаниях. Он позволяет создавать группы и каналы, обмениваться файлами и настраивать различные интеграции.

Чтобы запустить ваш процесс в Mattermost:

1. Создайте бота в Mattermost.
2. Подключите канал.
3. Протестируйте процесс.

## Создание бота в Mattermost

1. Создайте аккаунт бота в Mattermost.
2. Во время создания аккаунта вы получите токен бота. Сохраните токен, он нужен для подключения бота к Just AI Agent Platform.
3. Добавьте бота в вашу команду.

## Подключение канала

Добавить новый канал в проекте можно несколькими способами:

- В проекте перейдите в раздел *Интеграции*, выберите *Подключить* → *Канал* → *Mattermost*.
- На холсте выберите *Сообщение* → *Подключить канал* → *Mattermost*.

1. Задайте название. Оно будет отображаться в списке интеграций.
2. Укажите токен, который вы получили при создании бота.
3. Укажите адрес сервера, на котором установлен Mattermost. Например: `https://mattermost.example.com`.
4. Выберите способ публикации изменений:
    - *Автоматически* — изменения публикуются в канале сразу по кнопке *Опубликовать* на холсте.
    - *Вручную* — чтобы опубликовать изменения:
        a. В конструкторе нажмите на триггер *Сообщение*.
        b. Нажмите *Опубликовать* у нужного канала.

5. Нажмите *Подключить*.

Теперь вы можете получать и отправлять сообщения в этом канале.

# Тестирование процесса

1. Добавьте бота Mattermost в канал или группу. Также вы можете общаться с агентом в личных сообщениях.
2. Напишите боту. После этого должен начаться ваш диалог с агентом.

# Prompter API

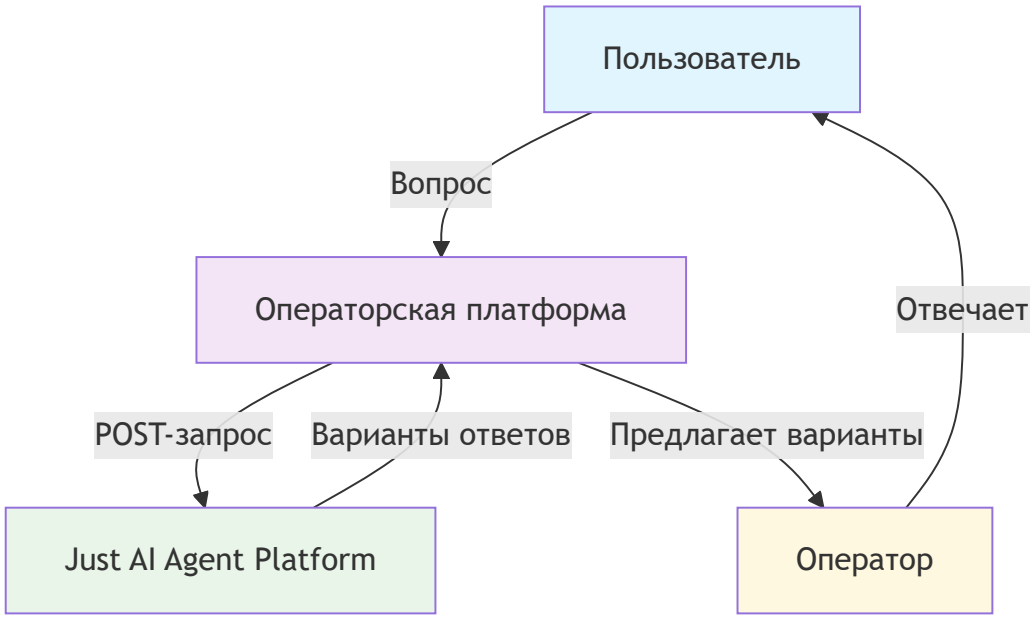
**Суфлер** — помощник, который анализирует вопросы клиентов и подсказывает операторам наиболее подходящие варианты ответов. Он не общается с клиентом напрямую, а помогает оператору быстрее реагировать на запросы.

Prompter API позволяет:

- Подключить суфлеров в Aimychat для автоматической поддержки операторов.
- Интегрировать суфлеров в сторонние сервисы через API.

## Как работает интеграция

Когда пользователь отправляет вопрос в операторскую платформу, система передает этот запрос через вебхук Prompter API в Just AI Agent Platform. Процесс в Just AI Agent Platform анализирует вопрос, формирует подходящие варианты ответов и возвращает их обратно в операторскую платформу. Оператор получает готовые варианты ответов и может выбрать наиболее подходящий для отправки пользователю.




## Подключение Prompter API

Добавить новый канал в проекте можно несколькими способами:

- В проекте перейдите в раздел *Интеграции*, выберите *Подключить → Канал → Prompter API*.

- На холсте выберите *Сообщение* → *Подключить канал* → *Prompter API*.

1. Задайте название. Оно будет отображаться в списке интеграций.
2. Выберите способ публикации изменений:
   - *Автоматически* — изменения публикуются, если вы нажали *Опубликовать* на холсте.
   - *Вручную* — чтобы опубликовать изменения:
     a. В конструкторе нажмите на триггер *Сообщение*.
     b. Нажмите *Опубликовать* у нужного канала.
3. Нажмите *Подключить*.

После создания канала получить токен и адрес вебхука можно двумя способами:

- Нажмите ⋮ → *Редактировать* у канала на вкладке *Интеграции*.
- Нажмите *Получить вебхук* в триггере *Сообщение* на холсте. Токен — это часть URL вебхука после `https://bot.jaicp.com/chatapi/webhook/prompter_api/`

# Интеграция в сторонние сервисы

Вы можете подключить любой сторонний сервис. Он должен использовать метод `POST /chatapi/webhook/prompter_api/{token}`. Этот метод отправляет в Just AI Agent Platform вопрос клиента и получает подходящие варианты ответа.

## Параметры запроса

Параметр пути:

- `token` — токен канала Prompter API, который можно получить после подключения канала.

Параметры тела запроса:

- `clientInfo` — объект с информацией о клиенте.
  - `id` — уникальный идентификатор клиента в вашей системе.
  - `firstName` и `lastName` — имя и фамилия клиента (необязательные параметры).
- `chatId` — уникальный идентификатор диалога в вашей системе.
- `text` — текст вопроса клиента.

## Ответ на запрос

В ответ на запрос платформа возвращает JSON-объект с вариантами ответов:

```
{
    "clientInfo": {
        "id": "testClientId"
    },
    "chatId": "testChatId",
    "questionId": "questionId",
    "answerOptions": [
        {
            "messages": [
                {
                    "type": "TEXT",
                    "text": "Ответ на вопрос клиента",
                    "markup": "plain"
                }
            ]
        }
    ]
}
```

> **ПОДСКАЗКА**
>
> Подробнее о методе, параметрах запроса и формате ответа смотрите в спецификации Prompter API.

## Aimychat

Для подключения суфлера в Aimychat можно воспользоваться вебхуком:

1. Перейдите в раздел *Настройки* → вкладка *Суфлеры*.
2. Добавьте нового суфлера и укажите адрес вебхука Prompter API.
3. На вкладке *Группы* выберите нужную группу или создайте новую. В окне редактирования группы перейдите на вкладку *Суфлеры*, выберите ваш суфлер и нажмите *Добавить*.

Вебхук представляет собой эндпоинт на стороне Just AI Agent Platform, который принимает POST-запросы от Aimychat с вопросами клиентов.

# Telegram

Через Just AI Agent Platform вы можете опубликовать процесс в виде:

- обычного Telegram-бота;
- Telegram-бота для бизнес-аккаунтов.

Чтобы использовать ваш процесс в Telegram:

1. Создайте бота.
2. Подключите канал.
3. Протестируйте процесс.

## Создание бота

Прежде чем опубликовать процесс в Telegram, нужно создать бота на стороне Telegram.

1. Откройте Telegram и в поиске контактов введите `BotFather`.
2. Отправьте команду `/newbot` в чат с BotFather.
3. Придумайте имя бота и отправьте его в чат. Имя будет отображаться в контактах и чатах.
4. Придумайте `username` для бота и отправьте его в чат.
   - `username` — короткое имя, которое используется для упоминаний бота и в ссылках на профиль.
   - Может состоять из букв латинского алфавита, подчеркиваний и цифр.
   - Должно заканчиваться на `bot`, например: `test_bot` или `TestBot`.
   - Может быть от 5 до 32 символов.
5. BotFather отправит ссылку на вашего бота и токен. Скопируйте токен, он потребуется при подключении канала. Токен представляет собой набор символов вида `123456789:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw`.
6. Если вы хотите подключить процесс к бизнес-аккаунту:
   i. Отправьте команду `/mybots` в чат с BotFather.
   ii. Выберите бота в списке.
   iii. Перейдите в *Bot Settings* → *Business Mode*.
   iv. Нажмите *Turn on*.
   v. Добавьте бота в бизнес-аккаунт.

> **ПОДСКАЗКА**
>
> Если вы потеряли токен, отправьте команду `/token` в чат с BotFather. BotFather создаст новый токен.

# Подключение канала

Добавить новый канал в проекте можно несколькими способами:

- В проекте перейдите в раздел *Интеграции*, выберите *Подключить* → *Канал* → *Telegram*.
- На холсте выберите *Сообщение* → *Подключить канал* → *Telegram*.

1. Задайте название. Оно будет отображаться в списке интеграций.
2. Укажите токен, который вы получили при создании бота.
3. Выберите способ публикации изменений:
   - *Автоматически* — изменения публикуются в канале по кнопке *Опубликовать* в холсте.
   - *Вручную* — чтобы опубликовать изменения:
     a. В конструкторе нажмите на триггер *Сообщение*.
     b. Нажмите *Опубликовать* у нужного канала.
4. Нажмите *Подключить*.

Теперь вы можете получать и отправлять сообщения в этом канале в рамках своего процесса.

# Тестирование процесса

Перейдите в окно диалога с ботом в Telegram и напишите ему. После этого должен начаться ваш диалог с ботом согласно вашему процессу.

# SIP-подключение

Используйте SIP-подключение, чтобы принимать в Just AI Agent Platform звонки, которые поступают на ваш SIP-транк. Подробнее про телефонию читайте в статье Как настроить звонки.

## Основные настройки

- *Учетные данные* — создайте или выберите учетные данные типа *SIP-аккаунт*.
- *Имя хоста или IP-адрес* — укажите адрес SIP-сервера вашего провайдера.
- *Порт* — укажите порт SIP-сервера.
- *Принимать входящие звонки* — включите, чтобы по этому подключению могли поступать входящие вызовы.

## Расширенные настройки

- *Авторизация по username* — заполните, если username для авторизации отличается от логина в учетных данных типа *SIP-аккаунт*.
- *Включить регистрацию* — включите, если ваш провайдер требует регистрацию для приема входящих звонков.
- *Сопоставление по IP и порту* — включите опцию, если в одном подключении нужно обрабатывать входящие вызовы на несколько DNID (dialed number identifier, идентификатор набранного номера).
- *Игнорировать порт* — включите для авторизации только по IP, без учета порта.
- *Отправлять локальный домен в заголовке FROM* — включите опцию, чтобы в SIP-заголовке `From` передавался адрес локального домена, а не SIP URI.
- *Проверять соединение* — включите опцию и заполните *Период проверки в секундах*, если нужна проверка подключения.
- *Таймаут регистрации* — укажите время в секундах, через которое Just AI Agent Platform должна повторно регистрироваться на SIP-сервере. Значение используется в заголовке `Expires` у SIP-запроса `REGISTER`.

  > **ПРИМЕЧАНИЕ**
  >
  > Параметры *Таймаут регистрации* и *Проверять соединение* не зависят друг от друга.

- *Кодеки* — выберите кодеки, которые будут использоваться для передачи аудио во время звонков.

# Учетные данные для провайдеров ASR/TTS

Эти учетные данные нужны для подключения моделей ASR и голосов TTS.

ASR и TTS используются:

- Во время звонков. Подробнее о телефонии читайте в статье Как настроить звонки.
- В текстовых каналах для распознавания или генерации аудиофайлов.

## 3i VoxKit

*Токен* — вставьте токен доступа к API 3i VoxKit.

## Audiogram

Получите доступ к сервису на сайте Audiogram. В Just AI Agent Platform нужны следующие данные:

- *IAM account*
- *IAM workspace*
- *Client ID*
- *Client secret*

## SaluteSpeech

- *Client ID*.
- *Ключ авторизации*.
- *Scope* — выберите версию API:
  - *SALUTE_SPEECH_PERS* — для физических лиц.
  - *SALUTE_SPEECH_CORP* — для юридических лиц, у которых проект SaluteSpeech создан на условиях постоплаты.
  - *SALUTE_SPEECH_B2B* — для юридических лиц, у которых проект SaluteSpeech создан на условиях предоплаты.
  - *SBER_SPEECH* — устаревшее значение для юридических лиц.

Все эти данные вы можете получить в личном кабинете SaluteSpeech.

# T-Bank VoiceKit

*Ключ API* и *Секретный ключ* — вставьте значения `API_KEY` и `SECRET_KEY` из личного кабинета T-Bank VoiceKit.

# Yandex SpeechKit

Чтобы подключить Yandex SpeechKit, создайте сервисный аккаунт и авторизованный ключ.

- *Закрытый ключ сервисного аккаунта* — вставьте закрытую часть авторизованного ключа в формате:

  ```
  -----BEGIN PRIVATE KEY-----EXAMPLE1234567G9w0BAQEFAASCBKgwggS
  cmQxJjAkBgNVBAoTHVByb2dyZXNzIFNvZnR3YXJlIENvcnBvcmF0aW9uMSAwHgYD
  VQQDDBcqLmF3cy10ZXN0LnByb2dyZXNzLmNvbTCCASIwDQYJKoZIhvcNAQEBBQAD
  ...
  EXAMPLE1234567YWxzaGEyZzIuY3JsMIGgBggrBgEFBQcBAQSBkzCBkDBNBggrBgEF
  BQcwAoZBaHR0cDovL3NlY3VyZS5nbG9iYWxzaWduLmNvbS9jYWNlcnQvZ3Nvcmdh
  z3P668YfhUbKdRF6S42Cg6zn-----END PRIVATE KEY-----
  ```

- *ID сервисного аккаунта* — вставьте ID сервисного аккаунта. Его можно получить в списке сервисных аккаунтов.
- *ID открытого ключа* — вставьте ID открытого ключа. Он отображается напротив ранее созданного авторизованного ключа.
- *ID каталога* — вставьте ID каталога, в котором создан сервисный аккаунт.

# Модели ASR и голоса́ TTS

Эти интеграции нужны для распознавания и синтеза речи:

- Во время звонков. Подробнее о телефонии читайте в статье Как настроить звонки.
- В текстовых каналах для распознавания или генерации аудиофайлов.

Учетные данные для этих интеграций описаны в статье Учетные данные для ASR/TTS.

## Модели ASR

### 3i VoxKit

*Модель* — идентификатор модели для распознавания речи.

### Audiogram

Сервис распознает речь только на русском языке. У интеграции нет дополнительных настроек.

### SaluteSpeech

Сервис распознает речь только на русском языке. У интеграции нет дополнительных настроек.

### T-Bank VoiceKit

Сервис распознает речь только на русском языке. У интеграции нет дополнительных настроек.

### Yandex SpeechKit

- *Язык* — код языка, на котором говорят ваши клиенты.
- *Модель* — версия модели для распознавания речи.
- *Числа прописью* — если опция включена, то в распознанном тексте числа записываются словами. Например, `тринадцать` вместо `13`.
- *Подавление шума* — опция уменьшает чувствительность модели к фоновым шумам.

## Голоса TTS

## 3i VoxKit

*Голос* — выберите голос для синтеза речи.

## Audiogram

- *Модель* — модель для синтеза речи. Сейчас доступна только модель *high_quality*.
- *Голос* — голос для синтеза речи:
  - Женские голоса:
    - *borisova*
    - *kishchik*
  - Мужские голоса:
    - *gandzhaev*
    - *gavrilov*
- *Эмоция* — эмоциональная окраска голоса:
  - *neutral* — нейтральная
  - *happy* — радостная

## SaluteSpeech

*Голос* — введите код голоса для синтеза речи, например `Nec_8000`. Just AI Agent Platform поддерживает только голоса с частотой 8 кГц — коды таких голосов заканчиваются на `_8000`.

Смотрите список голосов в документации SaluteSpeech.

## Yandex SpeechKit

- *Язык* — код языка, на котором будет синтезироваться речь.
- *Манера речи* — соответствует **амплуа** из документации Yandex SpeechKit. Это характеристика звучания голоса — например, диктор может говорить более дружелюбно или шепотом.
- *Скорость* — чем выше значение, тем быстрее будет речь. `1` — нормальная скорость голоса.
- *Голос* — название голоса для синтеза речи. Just AI Agent Platform поддерживает голоса v3.
- *Громкость* — громкость относительно цифровой полной шкалы LKFS/LUFS. Рекомендуемый диапазон значений: от `-20` до `-16`.

> **К СВЕДЕНИЮ**
>
> Список доступных голосов v3, языков и амплуа смотрите в документации Yandex SpeechKit.

# Учетные данные для интеграций

Когда вы создаете интеграцию с LLM, каналом или другим внешним сервисом, то вы можете использовать свои учетные данные для аутентификации.

В Just AI Agent Platform есть специальное хранилище для ваших паролей, токенов и логинов — раздел *Учетные данные*. Чтобы перейти в него, откройте проект и выберите раздел на левой панели.

Раздел позволяет:

- **Безопасно хранить данные**

  Учетные данные не будут видны в явном виде в блоках или коде.
- **Легко использовать данные**

  Если вы добавили данные в хранилище, то сможете выбирать их в настройках интеграций или получать в процессах с помощью встроенной функции `Credentials.get`.

# Управление учетными данными

## Добавление

1. Нажмите *Добавить*.
2. Выберите тип учетных данных. Набор данных отличается в зависимости от типа.

> **ПРИМЕЧАНИЕ**
>
> Каждая интеграция поддерживает один определенный тип учетных данных:
>
> - Например, если вы хотите настроить интеграцию с LLM, то вам нужно создать учетные данные типа *LLM*.
> - Если подходящей встроенной интеграции нет, выберите тип *Другое*. Далее напишите свою функцию для обращения к внешнему сервису и используйте в ней учетные данные с помощью `Credentials.get`.
> - Данные типа *Провайдер ASR/TTS* можно использовать как для ASR, так и для TTS. Если у вас провайдеры ASR и TTS разные, создайте по одному набору учетных данных для каждого провайдера.

3. Укажите нужные данные и настройте срок действия.
4. Нажмите *Добавить*.

### Редактирование

1. Нажмите ✎ в таблице с вашими учетными данными.
2. Внесите изменения.
3. Нажмите *Сохранить*.

Во всех связанных интеграциях сразу будут использоваться новые данные. Также обновленные данные будет возвращать `Credentials.get`.

### Удаление

1. Нажмите 🗑 в таблице с вашими учетными данными.
2. Подтвердите удаление.

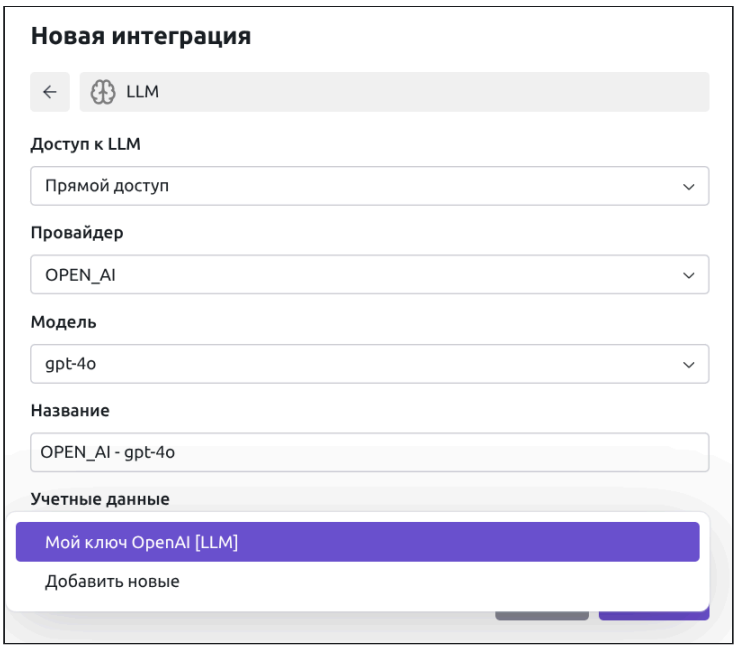
> ⚠ **ПРЕДУПРЕЖДЕНИЕ**
>
> Связанные интеграции сразу перестанут работать — это может привести к ошибкам в процессах.

## Как использовать

### Выбор данных во встроенных интеграциях

Сохраненные учетные данные можно выбрать в двух местах:

- В разделе *Интеграции* при подключении нового сервиса.
- Прямо в процессе — например, при настройке LLM в блоке *Агент*.




В обоих случаях вам не придется вводить токен или ключ API вручную — просто выберите нужные данные из выпадающего списка.

## Подключение к любому другому сервису

Например, вы хотите, чтобы ваш бот мог сообщать погоду, используя сторонний сервис WeatherAPI.com. Однако встроенной интеграции с этим сервисом в Just AI Agent Platform нет и вам нужно обращаться к нему напрямую по API.

Для таких задач используйте функцию `Credentials.get`. Она позволяет безопасно получить сохраненный API-ключ прямо в коде и использовать его для отправки запросов.

### Шаг 1: Сохраните API-ключ

1. Зарегистрируйтесь на WeatherAPI.com и получите бесплатный API-ключ.
2. В Just AI Agent Platform перейдите в раздел *Учетные данные*.
3. Добавьте новые учетные данные с типом *Другое* и авторизацией по токену. В поле *Токен* вставьте ваш API-ключ.
4. Новые учетные данные появятся в таблице в разделе *Учетные данные*. Скопируйте ID из таблицы. Предположим, что ID выглядит так: `1000111111-weather-abc`.

### Шаг 2: Напишите функцию для запроса погоды

Теперь вы можете написать простую функцию, которая будет обращаться к API погоды. В ней используйте `Credentials.get`, чтобы безопасно получить ключ. Вот как будет выглядеть код:

```
async function getWeather({ city }) {
    // Получаем объект с учетными данными по ID из таблицы
```

```
    const credentials = Credentials.get('1000111111-weather-abc');

    // Формируем запрос
    const requestConfig = {
        url: 'https://api.weatherapi.com/v1/current.json',
        params: {
            // Подставляем токен из наших учетных данных
            key: credentials.token,
            q: city
        }
    };

    // Выполняем запрос с помощью встроенной функции Http.get
    const response = await Http.get(requestConfig);

    // Извлекаем температуру из ответа и возвращаем результат
    return `Температура: ${response.body.current.temp_c}°C`;
}
```

Таким образом, ваш API-ключ надежно хранится в Just AI Agent Platform и не виден в коде функции.

# Подходы к построению ИИ-решений

При создании ИИ-систем — цифровых ассистентов, инструментов автоматизации или умных интерфейсов — применяют несколько архитектурных подходов. Каждый из них подходит для определенного типа задач:

- Workflow (рабочий процесс) — линейная, заранее определенная последовательность действий, как в классических чат-ботах.
- Агентский подход — логика на базе LLM, которая позволяет адаптироваться к контексту и вызывать внешние инструменты.
- MAS (Multi-Agent System) — система из нескольких агентов, объединенных в общую структуру для решения сложных задач.

**Пример задачи:** Пользователь хочет проанализировать договор и получить краткое резюме с ключевыми условиями. Система должна извлечь из документа важную информацию (стороны, сроки, суммы), проверить ее на корректность и сформировать структурированный отчет.

Рассмотрим, как эта задача решается разными подходами.

# Workflow подход

**Решение через workflow:**

1. Пользователь загружает файл → проверка формата (только PDF или DOCX).
2. Если формат правильный → извлечение текста.
3. Поиск ключевых слов: «стороны», «сумма», «срок».
4. Заполнение фиксированного шаблона отчета.
5. Отправка результата пользователю.

Workflow работает по жесткому алгоритму: каждый шаг выполняется последовательно, без возможности адаптации к содержимому документа.

# Агентский подход

## Что такое агент

Агент — это полноценный автономный исполнитель на базе языковой модели (LLM). У агента есть своя цель, логика и набор инструментов. Он не требует жестко заданной цепочки действий, а сам решает, какие шаги предпринять для достижения цели.

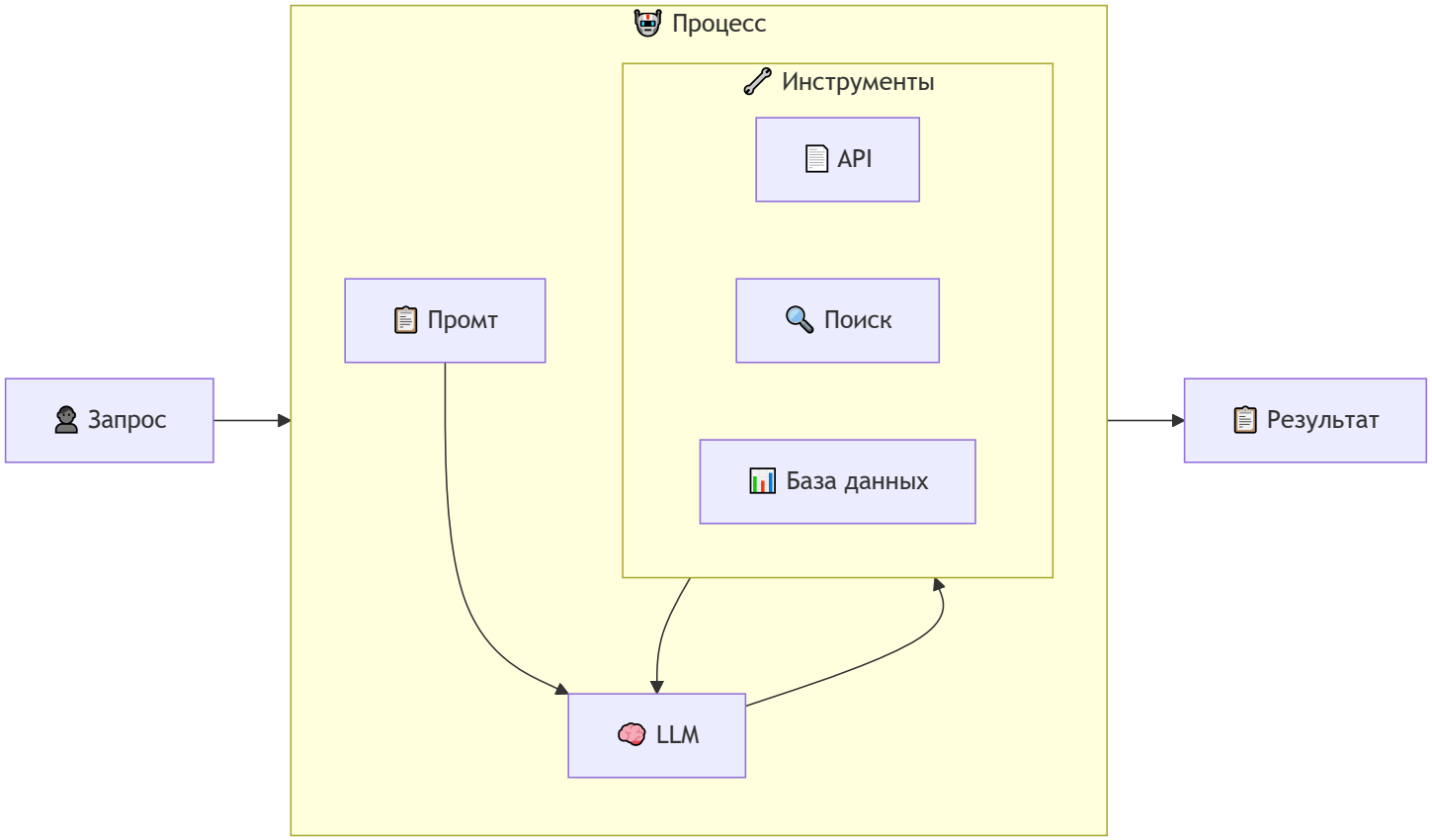
Агент может:

- самостоятельно понимать запрос пользователя;
- определять, какие действия необходимо выполнить;
- вызывать внешние инструменты (функции, API, парсеры);
- уточнять у пользователя детали;
- адаптироваться к различным ситуациям и контексту;
- возвращать осмысленный результат.

Пример реализации агента вы можете увидеть в Быстром старте, а подробную настройку — в документации по блоку «Агент».

## Компоненты агента

| Компонент | Назначение |
|---|---|
| **LLM (модель)** | «Мозг» агента: интерпретирует запросы и принимает решения. |
| **Промт** | Определяет роль агента, стиль ответов и правила работы. |
| **Инструменты (tools)** | Функции, которые агент может использовать для взаимодействия с системами, которые делают то, на что LLM не способна самостоятельно: API-запросы, поиск в базах данных, обработка файлов. |




## Отличия от workflow

Workflow (рабочий процесс) — это заранее определенная последовательность действий и условий, где каждый шаг четко регламентирован. В workflow-системах логика работы описывается через схемы типа «если условие X, то выполнить действие Y», что обеспечивает предсказуемость, но ограничивает гибкость.

| **Характеристика** | **Workflow** | **Агент** |
|---|---|---|
| **Определение действий** | Заранее заданная последовательность: «если А, то Б». | Самостоятельно определяет необходимые действия. |
| **Предсказуемость** | Высокая — при одинаковых условиях результат всегда одинаковый. | Адаптивная — результат зависит от контекста. |
| **Отладка** | Простая — четкие правила и логика. | Сложнее — требует анализа решений модели. |
| **Гибкость** | Низкая — плохо справляется с неопределенностью. | Высокая — адаптируется к различным ситуациям. |

| Характеристика | Workflow | Агент |
| --- | --- | --- |
| **Понимание запросов** | Заранее определенные категории (интенты или друга логика). | Контекстуальная интерпретация. |
| **Последовательность действий** | Фиксированная. | Динамическая, в зависимости от задачи. |

Агентский подход особенно полезен, когда:

- запросы пользователя разнообразны;
- логика зависит от входных данных;
- существуют разные пути решения задачи;
- невозможно заранее описать все сценарии.

> **ПРИМЕЧАНИЕ**
>
> На практике агент часто встраивается в workflow. Workflow управляет общим процессом, а агент используется на интеллектуальных этапах: анализ, генерация, поиск.

## Рекомендации по настройке агента

При создании агента определите три ключевых аспекта:

| Поле | Назначение | Принцип | Пример |
| --- | --- | --- | --- |
| **Роль** | Профессия или специализация агента. | Конкретность, не общие термины. | `Аналитик договоров`. |
| **Цель** | Конкретная задача, которую выполняет агент. | Измеримость, четкие границы. | `Извлечь из договора: стороны, сроки, суммы, риски. Вернуть структурированный JSON`. |
| **Инструкции** | Правила работы, стиль общения, ограничения. | Структурированность, примеры. | Процесс работы, формат ответа, ограничения (пример хорошей |

| Поле | Назначение | Принцип | Пример |
|---|---|---|---|
| | | | инструкции смотрите ниже). |

**Структура эффективных инструкций:**

1. Процесс работы — пошаговый алгоритм действий.
2. Стиль общения — тон и манера ответов.
3. Формат ответа — как структурировать результат.
4. Ограничения — что агент не должен делать.

**Пример:**

```
Роль: Аналитик договоров
Цель: Проверь, соответствует ли документ стандартам

Инструкции:
ПРОЦЕСС:
- Изучи документ полностью
- Извлеки ключевую информацию (стороны, сроки, суммы)
- Проверь соответствие стандартам компании

ОТВЕТ:
- Краткое резюме
- Список ключевых условий
- Выявленные риски

ОГРАНИЧЕНИЯ:
- Не давай юридических советов
- При сомнениях рекомендуй специалиста
```

# Создание инструментов для агента

**Принципы хороших инструментов:**

| Принцип | Описание | Пример |
|---|---|---|
| **Атомарность** | Один инструмент = одна цель | `Поиск документов` вместо `Поиск документа и рисование картинок в нем` |

| Принцип | Описание | Пример |
| --- | --- | --- |
| **Понятное название** | Название отражает действие | `Извлечь данные из PDF` |
| **Четкое описание** | Агент понимает, что делает инструмент | `Ищет документы по ключевым словам в базе данных` |

**Рекомендации по описанию инструментов:**

- Начинайте с глагола: `Извлекает`, `Анализирует`, `Отправляет`.
- Указывайте источник данных: `из базы`, `из файла`, `через API`.
- Поясняйте формат результата: `возвращает список документов`, `создает отчет в PDF`.
- Отмечайте ограничения: `работает только с текстовыми файлами`.

**Пример хорошего описания:**

```
Название: Поиск договоров
Описание: Ищет договоры в корпоративной базе по названию компании,
номеру договора или дате. Возвращает список найденных документов
с основной информацией. Поддерживает только активные договоры.
```

**Частые ошибки:**

- Неясные описания инструментов: `обрабатывает данные`, `работает с файлами`.
- Отсутствие ограничений: не указано, с какими форматами работает инструмент.
- Противоречивые инструкции: `будь кратким` + `объясняй подробно`.

# Пример агента

**Решение через агента:**

Агент анализирует тот же договор, но адаптируется к его содержимому:

1. Получает документ любого формата (PDF, DOCX, изображение).
2. Самостоятельно определяет тип договора (поставка, аренда, услуги).
3. Извлекает релевантную информацию в зависимости от типа.
4. Проверяет логическую согласованность данных.
5. Формирует отчет в подходящем формате.

Агент самостоятельно:

- выбирает подходящий инструмент для извлечения текста.
- определяет, какая информация важна для данного типа договора.
- решает, нужна ли дополнительная проверка данных.
- адаптирует формат отчета под сложность договора.

# Мультиагентная система (MAS)

## Что такое MAS

Мультиагентная система (MAS) — это архитектура, в которой несколько агентов работают совместно, каждый выполняя свою задачу. Подробнее о создании мультиагентного процесса вы можете прочитать в документации.

Вместо одного универсального агента создается несколько специализированных. Например:

- один анализирует документы;
- другой извлекает информацию;
- третий формирует документ по шаблону;
- четвертый отправляет результат.

**Преимущества:**

- Упрощение разработки — каждый агент отвечает за свою задачу.
- Гибкость — изменения можно вносить точечно в конкретного агента.
- Масштабируемость — систему легко расширять, добавляя новых агентов.
- Повторное использование — агентов можно применять в других сценариях.

## Почему не один универсальный агент

Может показаться, что проще создать одного агента с объемной инструкцией. Этот подход работает для простых задач, но в сложных сценариях он неэффективен. Если логика разветвленная, инструкции противоречат друг другу или нужно подключать новых исполнителей, универсальный агент начинает работать нестабильно:

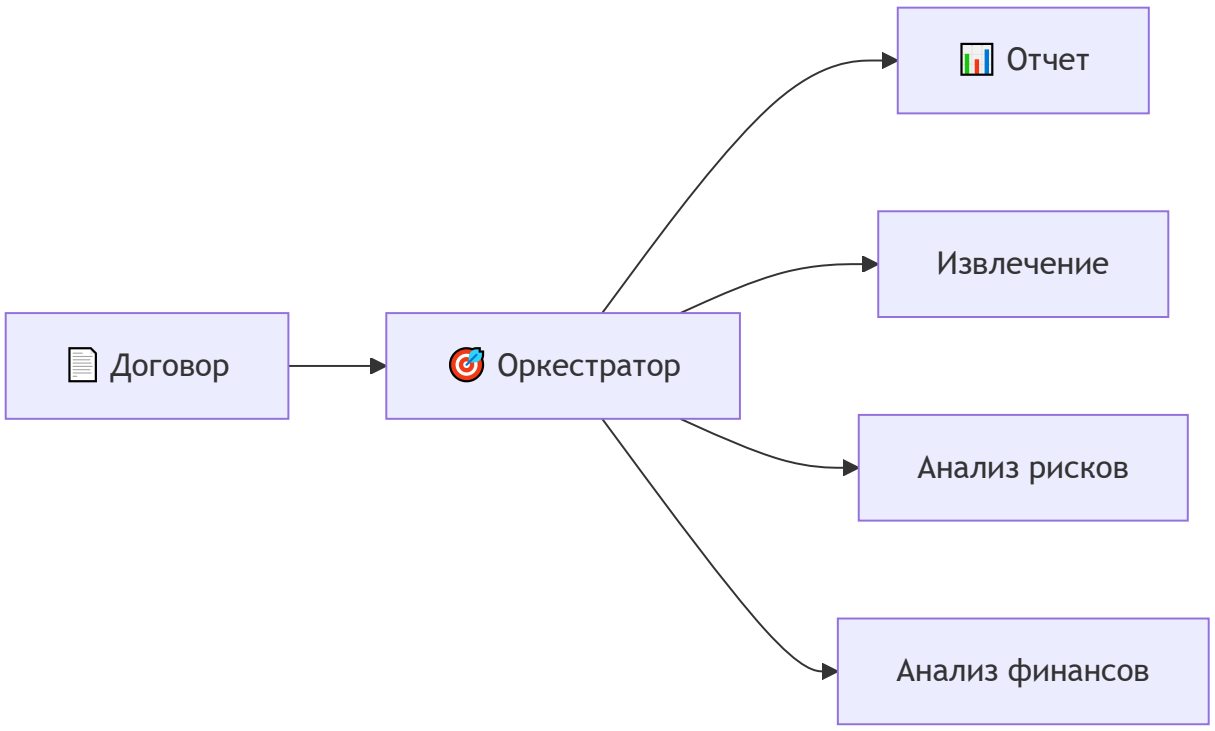
- появляются «галлюцинации»;
- ошибки сложно диагностировать;
- инструкция становится слишком объемной и противоречивой.

## Сравнение: один агент vs. MAS

| | **Один агент** | **MAS** |
|---|---|---|
| **Логика** | Общая, в рамках одного промта. | Разделена между отдельными агентами. |
| **Поддержка** | Усложняется при росте требований. | Легче изменять отдельных агентов. |
| **Масштабируемость** | Ограничена размером промта. | Легко добавить нового агента. |
| **Переиспользуемость** | Низкая. | Высокая. |
| **Надежность** | Зависит от одной модели. | Ошибка локализуется в рамках одного агента. |

## Как устроена MAS

Координация агентов обычно происходит через workflow или специального агента-оркестратора, который управляет остальными.




## Когда использовать MAS

**Использовать MAS, если:**

- У задачи много этапов с разной логикой.
- Требуются разные навыки (поиск, генерация, разметка).
- Сценарий может меняться или расширяться.
- Разные специалисты могут разрабатывать отдельных агентов.

**Не использовать MAS, если:**

- Один агент хорошо справляется с задачей.
- Нет жестких требований к масштабированию.
- Процесс достаточно однообразный и не требует экспертизы в разных областях.

> **ПРИМЕЧАНИЕ**
>
> Важно выбирать MAS только когда это действительно нужно. Если задача простая, один агент будет эффективнее и проще в поддержке.

## Пример MAS

**Решение через MAS:**

Для анализа договора создается команда из трех специализированных агентов:

1. **Агент извлечения** — обрабатывает файл и извлекает текст.
2. **Агент анализа рисков** — ищет проблемные пункты и юридические риски.
3. **Агент анализа финансов** — проверяет суммы, валюты и условия оплаты.

**Роль оркестратора:**

- Координирует работу агентов (сначала извлечение, затем параллельный анализ).
- Решает, какие агенты нужны в зависимости от типа договора.
- Объединяет результаты всех агентов в единый отчет.
- При обнаружении противоречий запрашивает повторный анализ.

Например, если агент рисков находит проблему с суммой, оркестратор может попросить агента финансов дополнительно проверить именно этот пункт.

# Сравнение подходов на примере

Сравним все подходы на примере задачи по обработке документов.

| Аспект | Workflow | Агент | MAS |
|---|---|---|---|
| **Гибкость к формату** | Только PDF/DOCX | Любой формат, поддерживаемый платформой | Любой формат, поддерживаемый платформой |
| **Адаптация к типу договора** | Одинаково для всех | Универсальный подход | Специализированные агенты под тип |
| **Параллельная обработка** | Последовательная | Нет | Да (аналитики работают одновременно) |
| **Обработка неясностей в тексте** | Пропускает или выдает ошибку | Пытается интерпретировать самостоятельно | Привлекает нужного эксперта |
| **Разрешение противоречий** | Не предусмотрено | Ограниченное | Оркестратор координирует решение |
| **Сложность внедрения** | Простая | Средняя | Высокая |
| **Качество анализа** | Базовое | Хорошее | Экспертное (каждый агент — специалист) |
| **Масштабируемость** | Ограниченная | Хорошая | Отличная (легко добавить экспертов) |

**Вывод:** Выбор подхода зависит от требований к качеству, разнообразия задач и ресурсов на разработку.

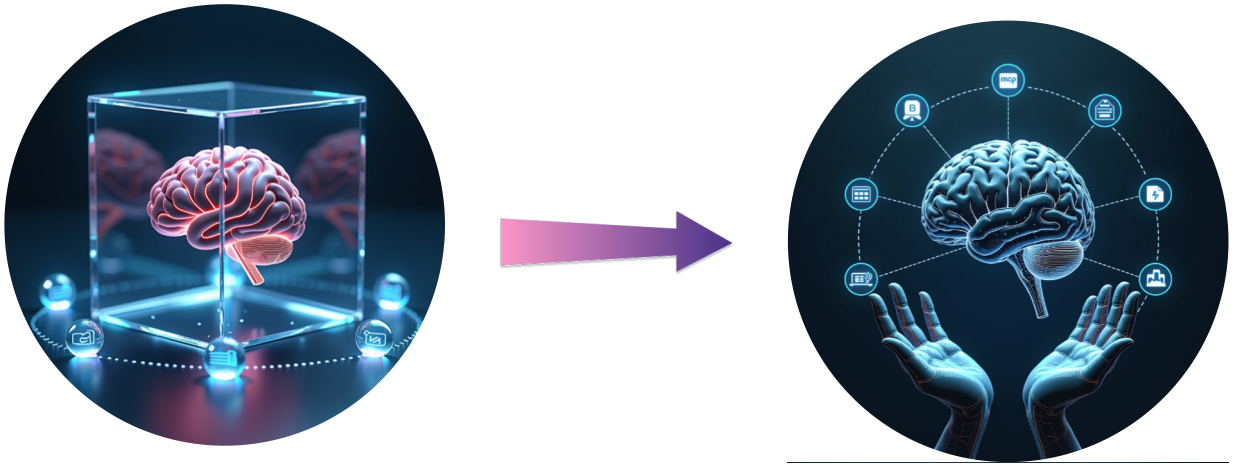
# MCP-серверы в агентных системах

Современные модели искусственного интеллекта обладают огромным потенциалом, но часто остаются изолированными от данных и внешних систем. **MCP-серверы** (от англ. Model Context Protocol) решают эту проблему, выступая в роли «органов чувств» и «рук» для AI, открывая путь к созданию по-настоящему автономных систем.

## Зачем нужны MCP-серверы

Главная задача MCP-сервера — подключать искусственный интеллект к внешним системам, базам данных и сервисам. Он работает по открытому протоколу Model Context Protocol (MCP), который позволяет AI-моделям получать актуальные данные и выполнять действия, снижая необходимость в индивидуальных интеграциях.

Это помогает преодолеть изоляцию AI. Вместо того чтобы опираться только на обучающие данные, модель получает доступ к информации в реальном времени, что повышает ее контекстную осведомленность и эффективность. MCP-серверы становятся ключевым инструментом для интеграции AI с корпоративными системами, базами данных и сервисами.



Без MCP-сервера: Изолированная модель

С MCP-серверами: Расширенная модель

## Чем MCP-сервер отличается от обычного API

На первый взгляд MCP-сервер напоминает API, но между ними есть принципиальные различия. MCP — это не просто набор эндпоинтов, а интеллектуальный посредник, работающий по единому протоколу, специально разработанному для взаимодействия AI-моделей с внешним миром.

Ключевые отличия:

- **Стандартизация.** MCP описывает функции как именованные действия с единым JSON-форматом, тогда как обычные API используют разные HTTP-методы и структуры запросов.
- **Динамическое обнаружение.** Клиент (оркестратор) может программно получить список доступных инструментов у MCP-сервера, без жестко заданных эндпоинтов.
- **Безопасность.** MCP-сервер хранит чувствительные данные (например, ключи доступа) локально и не передает их модели, повышая уровень защиты.
- **Двустороннее взаимодействие.** AI может не только получать данные, но и инициировать действия через сервер под контролем пользователя.
- **Локальная работа.** MCP может функционировать без доступа к сети, что упрощает интеграцию и повышает безопасность.
- **Простота разработки.** Единый протокол облегчает подключение новых источников и инструментов, снижая объем кастомных интеграций.

Иными словами, MCP — это унифицированный слой абстракции над API, адаптированный под нужды моделей и автономных агентов с упором на безопасность и масштабируемость.

# Преимущества MCP для автономных агентов

Для автономных агентов MCP-серверы дают ряд преимуществ, повышающих их функциональность и адаптивность:

- **Универсальное подключение.** Агентам не нужно знать детали каждого API — MCP предоставляет единый способ доступа к различным источникам данных и инструментам.
- **Доступ к данным в реальном времени.** Агенты получают актуальную информацию, необходимую для принятия решений и адаптации к меняющимся условиям.
- **Возможность действовать.** MCP позволяет агентам выполнять действия: обновлять документы, запускать транзакции, управлять смарт-контрактами и др.
- **Повышенная безопасность.** Доступы и ключи хранятся на стороне MCP-сервера, снижая риски. Уровень защиты зависит от конкретной реализации.
- **Координация агентов.** MCP может быть общим слоем доступа к инструментам и данным для нескольких агентов при наличии координации.
- **Приватность.** MCP поддерживает создание персональных AI-ассистентов, интегрированных с локальными приложениями и данными, без передачи информации внешним серверам в локальных конфигурациях.

# Кейсы использования MCP

На практике MCP уже показывает свою ценность в разных сферах:

- **Корпоративные ассистенты.** AI-агенты подключаются к Jira и Confluence через MCP, чтобы отвечать на вопросы о задачах и спецификациях, ускоряя работу команд.
- **Анализ данных.** Имея доступ к БД, агенты переводят запросы на естественном языке в SQL, выполняют их и возвращают готовые результаты, упрощая аналитику.
- **Интеграция с CRM.** AI-ассистенты получают данные о клиентах и сделках, помогая персонализировать общение и автоматизировать рутину.
- **Персональные помощники.** MCP используется для безопасного доступа к локальным файлам и приложениям, минимизируя передачу данных при корректной настройке.
- **Изолированные среды.** MCP позволяет создавать автономные AI-системы без доступа к интернету, что особенно важно для организаций с высокими требованиями к безопасности.

Таким образом, MCP-серверы — это ключевая технология, позволяющая AI-системам выйти за рамки изоляции и эффективно взаимодействовать с реальным миром.

# Техническая поддержка

Если вы не нашли ответа на свой вопрос в документации, обратитесь в техническую поддержку: напишите письмо на адрес support@just-ai.com или заведите заявку на портале. Наши специалисты проконсультируют вас и помогут решить проблему.

## Сообщество в Telegram

Также вступайте в сообщество Just AI Agent Platform в Telegram: https://t.me/Just_AI_Agent_Platform.

Здесь вы можете оставить обратную связь, предложить улучшения и поделиться опытом с другими пользователями.