# BIV Platform

## Руководство разработчика
## Модуль ИИ агенты

Набор компонент для создания ИИ-агентов

Версия документа 1.0
20 октября 2026 года

Подключение локальных и облачных моделей, построение агентских потоков, диалоговый интерфейс и наблюдение за выполнением.



## *Как пользоваться руководством*

Руководство предназначено для backend- и frontend-разработчиков прикладных проектов на BIV Platform. Оно описывает путь от подключения библиотек до работающего диалога с агентами и диагностики ошибок.

**Статус технических решений.** Базовый состав и возможности заданы требованиями. Недостающие детали дополнены единой проектной реализацией: Quarkus, Angular, PostgreSQL, REST и OpenTelemetry. Названия артефактов, интерфейсов, настроек и таблиц являются предложенным контрактом, а не подтверждением наличия опубликованного SDK.

Для первого подключения используйте разделы 8.1–8.3 и 9. Для разработки сценария — 3 и 8.4–8.7. Для сопровождения — 8.8, 10 и 12. Числовые лимиты приведены как стартовый профиль тестового стенда.

## *Содержание*

1 Общие понятия ..... 3
2 Назначение модуля ..... 3
3 Понятие настройки и её структура ..... 5
4 Типы пользователей ..... 6
5 Виды и доступность настроек ..... 6
6 Права на настройки и данные ..... 6
7 Ограничения выполнения ..... 7
8 Документация для Back End разработчика ..... 8
Провайдер и агентский поток ..... 9
Алгоритм агента и инструменты ..... 11
Диалог и хранение состояния ..... 12
Наблюдение и серверный API ..... 13
9 Документация для Front End разработчика ..... 16
10 Администрирование и запуск ..... 18
11 Примеры использования и проверка ..... 19
12 Возможные ошибки и их решения ..... 20
Приложение А Паспорт подключения ..... 21
Приложение Б Источники и проектные решения ..... **Error! Bookmark not defined.**

## *1 Общие понятия*

**ИИ-агенты** — краткое название модуля «Набор компонент для создания ИИ-агентов». Модуль предназначен для разработчиков прикладных решений на BIV Platform, которым нужны обращения к языковой модели, взаимодействие нескольких агентов и диалог с пользователем.

| Понятие | Значение в руководстве |
|---|---|
| Большая языковая модель или LLM | Модель, которая получает сообщения и формирует ответ. Может работать в локальном контуре или в облаке. |
| Провайдер модели | Компонента обращения к LLM, отделяющая прикладной сценарий от конкретного способа вызова модели. |
| Агент | Участник сценария с заданной задачей, инструкцией, доступным контекстом и правилом завершения. |
| Агентский флоу | Цепочка или граф выполнения агентов с правилами переходов. Далее также называется потоком. |
| Диалог | Последовательность сообщений пользователя и системы, объединённых контекстом общения. |
| Выполнение | Один запуск потока для обработки обращения пользователя. В одном диалоге может быть несколько выполнений. |
| Трасса | Связанные события выполнения, позволяющие восстановить вызовы и переходы между агентами. |

## *2 Назначение модуля*

Модуль объединяет повторяющиеся части агентских решений. Прикладная команда задаёт задачи агентов и бизнес-логику, подключает необходимые компоненты и встраивает диалог в своё приложение.

**Базовый состав и возможности по требованиям**

| Компонента | Назначение |
|---|---|
| Провайдер обращения к LLM | Подключение локальной или облачной модели. |
| Библиотека агентского флоу | Формирование произвольных агентских цепочек. |
| Библиотека слоя наблюдения | Отслеживание обращений и переходов между агентами. |
| Компонента интерфейса чата | Диалоговое взаимодействие пользователя с агентом. |
| Документация разработчика | Описание применения и подключения компонент. |

В проектную реализацию этого руководства дополнительно включены вызовы прикладных инструментов, хранение истории, REST API, события чата и ограничения выполнения. Обучение моделей, встроенный RAG и визуальный редактор потоков остаются за рамками базовой реализации. При необходимости поиск по знаниям подключается как отдельный инструмент.

### 2.1 Принятая архитектура

Руководство описывает следующую проектную реализацию. Backend работает на Java и Quarkus, чат поставляется как Angular-компонента, состояние диалогов хранится в PostgreSQL. Потоки исполняются серверной библиотекой, а обращение к каждой модели выполняется через отдельный адаптер провайдера.

| Часть решения | Принятое решение | Ответственность |
|---|---|---|
| Интерфейс | Angular-компонента чата и REST-клиент | Сообщения, состояния отправки, история и ошибки. |
| Прикладной сервис | Quarkus REST и серверный исполнитель | Авторизация, диалог, запуск потока, связь с бизнес-сервисами. |
| Агентский поток | Версионируемый граф и реестр агентов | Маршрутизация, условия завершения, контекст и лимиты. |
| Провайдер LLM | Единый Java-интерфейс и адаптеры | Локальные и облачные модели, преобразование форматов. |
| Состояние | PostgreSQL и журнал выполнений | История, статусы, дедупликация, ожидание ввода. |
| Наблюдение | События выполнения и OpenTelemetry | Переходы агентов, вызовы моделей и инструментов, длительность и ошибки. |

### 2.2 Порядок обработки сообщения

1. Чат передаёт текст и requestId в прикладной сервис.
2. Сервис проверяет пользователя, доступ к диалогу и ограничения; сохраняет сообщение и задание на выполнение в одной транзакции.
3. Исполнитель получает задание, загружает закреплённую версию потока и подготавливает контекст первого агента.
4. Агент вызывает модель. Исполнитель проверяет результат и выполняет разрешённый переход, вызов инструмента или завершение.
5. Каждый шаг формирует событие наблюдения. Сервис сохраняет итог и обновляет статус выполнения.
6. Чат получает статус и ответ; следующий вопрос связывается с тем же диалогом.

В качестве основного транспорта используется асинхронный REST API с чтением статуса. SSE служит дополнительным каналом доставки событий; работа чата сохраняется при его отключении. Параметры конкретной LLM изолированы в адаптере и не передаются из браузера.

Взаимодействие с авторизацией, аудитом и мониторингом строится через соответствующие механизмы BIV Platform. Наблюдение за агентским маршрутом дополняет аудит прикладных операций.

## *3 Понятие настройки и её структура*

Для интеграции рекомендуем разделить настройки на профиль модели, описание агента, описание потока и параметры диалога. Это позволяет менять модель или инструкцию без смешивания с бизнес-логикой маршрутизации.

**Модель конфигурации.** Имена полей ниже используются в предлагаемом контракте модуля и примерах интеграции.

| Объект | Основные атрибуты | Что проверить |
|---|---|---|
| Профиль модели | providerId, размещение, endpoint, modelId, ссылка на секрет, тайм-аут | Доступность модели, разрешённый контур, поддерживаемый формат запроса. |
| Агент | agentId, версия, инструкция, providerId, формат результата | Однозначная задача, разрешённые данные, критерий завершения. |
| Поток | flowId, версия, начальный агент, узлы, переходы, ограничения | Все узлы существуют; переходы определены; есть завершение и обработка ошибок. |
| Диалог | conversationId, владелец, flowId, версия потока, политика истории | Доступ владельца, изоляция истории, срок хранения. |
| Выполнение | executionId, conversationId, requestId, статус, traceId | Связь с сообщением и трассой, защита от повторной обработки. |

### 3.1 Правила изменения настроек

Версионируйте инструкции и потоки вместе с прикладным кодом либо через согласованный реестр конфигураций. Секреты храните отдельно; в конфигурацию включайте ссылку на секрет. Для одного выполнения фиксируйте версии потока, агентов и профиль модели.

Для длительного диалога заранее выберите правило обновления: сохранить старую версию до завершения диалога или выполнить явную миграцию. Не подменяйте инструкцию или граф незаметно для уже начатого выполнения.

### 3.2 Пример настройки агента

YAML хранится в ресурсах приложения и загружается реестром агентов при старте. Загрузчик проверяет уникальность agentId, существование профиля и корректность схемы результата.

```
agentId: request-classifier
version: "1"
providerId: approved-local
instruction: "Определи тему запроса. Не выполняй бизнес-операции."
resultSchema:
  category: [platform, project, unknown]
onUnknown: clarification
```

Значения category проверяйте в прикладном коде. Свободный текст модели нельзя использовать как идентификатор произвольного следующего агента.

## *4 Типы пользователей*

Роли ниже описывают ответственность. Прикладная команда сопоставляет их с ролями и политиками своего приложения.

| Роль | Ответственность |
|---|---|
| Пользователь приложения | Ведёт разрешённые диалоги и просматривает доступную ему историю. |
| Прикладной разработчик | Определяет агентов, переходы, интеграции, права и обработку ошибок. |
| Администратор модуля | Настраивает разрешённые профили моделей, версии потоков и лимиты. |
| Специалист сопровождения | Анализирует разрешённые трассы и ошибки, контролирует состояние интеграции. |

## *5 Виды и доступность настроек*

Разделяйте настройки по окружениям разработки, тестирования и промышленной эксплуатации. Профиль модели выбирайте на сервере по настройке сценария; интерфейс показывает только разрешённые пользователю варианты.

В базовом варианте пользователь выбирает доступный сценарий, а его состав и модель определяет команда проекта. Общие, групповые и личные настройки добавляются при необходимости; правила наследования настроек «Гибкой таблицы» автоматически не применяются.

## *6 Права на настройки и данные*

Для BIV Platform используйте действующий механизм аутентификации и авторизации приложения. Права проверяйте на сервере до чтения диалога, запуска потока и обращения к прикладным данным.

| Действие | Рекомендуемое ограничение |
|---|---|
| Запустить поток | Доступ к сценарию и соответствующей бизнес-функции. |
| Прочитать или продолжить диалог | Владелец диалога либо отдельно разрешённый доступ. |
| Изменить агента или профиль модели | Выделенное административное право. |
| Просмотреть трассу | Право диагностики с ограничением состава данных. |
| Выполнить прикладную операцию | Отдельная проверка прав на операцию и конкретный объект. |

Идентификатор пользователя берите из проверенного серверного контекста. Переданный браузером conversationId не подтверждает право на диалог. Вывод модели не является решением об авторизации.

Для операций изменения данных реализуйте проверку аргументов и повторов, а при необходимости — подтверждение пользователем по разделу 8.5.

# *7 Ограничения выполнения*

Задавайте ограничения до первого запуска сценария. Числа в таблице — пример начального профиля для тестового стенда, а не значения по умолчанию модуля и не норматив производительности. Подберите их по длительности сценариев и результатам нагрузочных проверок.

| Ограничение | Пример | Поведение при достижении |
|---|---|---|
| Длина сообщения | 8 000 символов | Отклонить до обращения к модели и предложить сократить текст. |
| Время одного вызова модели | 30 секунд | Завершить вызов по тайм-ауту; применить согласованную политику повтора. |
| Общее время выполнения | 90 секунд | Не запускать новые шаги; вернуть понятный статус. |
| Число шагов потока | 8 | Прервать цикл и записать причину остановки. |
| Повтор временно неуспешного вызова | Не более 1 | Учесть повтор в общем лимите времени и числе вызовов. |
| Активные обращения в одном диалоге | 1 | Отклонить или поставить следующее сообщение в очередь. |

## 7.1 Контекст модели и история

Лимит истории определяйте по вместимости выбранной модели с учётом инструкции, текущего сообщения и резерва на ответ. Ограничение в символах не заменяет контроль токенов. Используйте счётчик или оценку из адаптера конкретной модели.

При переполнении контекста применяйте согласованную стратегию: убрать старые сообщения, сформировать проверяемое резюме или предложить новый диалог. Не удаляйте незаметно текущую задачу, ограничения доступа и данные, без которых ответ меняет смысл.

## 7.2 Ошибки и повторы

Повторяйте только временные ошибки, для которых повтор разрешён: например, кратковременную недоступность. Ошибка прав, неверная конфигурация или неподдерживаемая модель требуют исправления, а не автоматического повтора. Учитывайте, что повтор обращения к облачной модели может тарифицироваться повторно.

Для повторной отправки одного сообщения сохраняйте requestId и проверяйте дубликат на сервере. Ключ дедупликации связывайте с пользователем и диалогом. Повторное выполнение бизнес-операции допускайте только при предусмотренной защите от дублирования.

## 7.3 Границы передачи данных

Для каждого профиля укажите разрешённые классы данных и допустимое размещение модели. Не переключайте локальный профиль на облачный автоматически без отдельного правила. Содержимое сообщений и документов считайте входными данными: инструкции внутри них не должны менять серверные права, список переходов и доступные операции.

# *8 Документация для Back End разработчика*

## 8.1 Подключение библиотеки в проект

Рекомендуем размещать оркестрацию в прикладном backend-сервисе, который проверяет пользователя и права, управляет диалогом и вызывает компоненты модуля. Браузер обращается к этому сервису; учётные данные провайдера остаются на сервере.

Для сервисов BIV Platform на Quarkus используйте совместимые версии Java, Quarkus и платформенных библиотек. Совместимость фиксируйте в паспорте подключения из приложения А. Проектный состав серверных артефактов: ai-agents-provider, ai-agents-flow и ai-agents-observability с groupId com.bivgroup.ai. Компонента чата публикуется как @biv/ai-chat.

**Перед подключением получите:** координаты и версии библиотек провайдера, флоу и наблюдения; доступ к репозиторию; пример конфигурации релиза; параметры тестовой модели; контракт чата; способ хранения истории и экспорта трасс.

Пример Maven-зависимости использует проектное имя библиотеки. Повторите запись для ai-agents-flow и ai-agents-observability; свойство biv.ai.version задайте по версии опубликованной поставки.

```
<dependency>
  <groupId>com.bivgroup.ai</groupId>
  <artifactId>ai-agents-provider</artifactId>
  <version>${biv.ai.version}</version>
</dependency>
```

Библиотеки подключаются в один прикладной сервис; независимое развёртывание каждой библиотеки не требуется. Провайдер модели и хранилище состояния оформляются как заменяемые адаптеры.

## 8.2 Первый запуск

1. Подключите зависимости выбранного релиза и проверьте сборку сервиса.
2. Создайте один профиль локальной либо облачной модели с тестовыми данными.
3. Определите агента с простой задачей: ответить на короткий вопрос без обращения к бизнес-операциям.
4. Создайте поток из одного агента с явным завершением и лимитами.
5. Подключите слой наблюдения и создайте серверную операцию отправки сообщения.
6. Отправьте первое сообщение, затем уточняющий вопрос в том же диалоге.
7. Проверьте ответ, сохранение контекста и связь двух выполнений с одним диалогом.
8. Добавьте второго агента и убедитесь, что в трассе виден переход между ними.

Результат первого запуска: диалог работает, история изолирована по пользователю, модель вызывается через провайдер, а путь обработки доступен для диагностики. Чат подключается к этому же серверному контракту по разделу 9.

### 8.3 Провайдер обращения к модели

Провайдер должен позволять использовать локальную или облачную модель. В прикладной интеграции рекомендуем обращаться к нему через единый адаптер и явно проверять возможности выбранной модели.

| Параметр профиля | Для чего нужен |
|---|---|
| Размещение | Определяет, разрешена ли передача данных в локальный или облачный контур. |
| Адрес и идентификатор модели | Выбирают доступную модель и точку обращения. |
| Ссылка на секрет | Позволяет получить учётные данные на сервере без включения их в код. |
| Тайм-аут и лимит ответа | Ограничивают длительность и объём вызова. |
| Возможности модели | Отмечают поддерживаемые форматы, потоковую выдачу и другие используемые функции. |

**Проектные параметры application.properties.** Модуль читает настройки из пространства biv.ai; секрет разрешается через отдельный серверный SecretResolver.

```
biv.ai.provider.id=approved-local
biv.ai.provider.endpoint=${MODEL_ENDPOINT}
biv.ai.provider.model-id=${MODEL_ID}
biv.ai.provider.secret-ref=${MODEL_SECRET_REF}
biv.ai.call-timeout-ms=30000
biv.ai.flow-timeout-ms=90000
biv.ai.max-steps=8
```

MODEL_ENDPOINT и MODEL_ID задаются для каждого окружения. MODEL_SECRET_REF содержит ссылку на запись в хранилище секретов. При отсутствии обязательного параметра сервис отклоняет профиль до обработки пользовательских сообщений.

**Рекомендуемый логический контракт вызова**

| Направление | Состав данных |
|---|---|
| Вход | Профиль модели, инструкция агента, разрешённая история, сообщение, лимиты и идентификаторы наблюдения. |
| Успешный выход | Текст или проверенная структура ответа, причина завершения, технические сведения о модели и расходе токенов, если доступны. |
| Ошибка | Категория ошибки, признак допустимости повтора и диагностический идентификатор без секретов. |

При смене модели повторите функциональные проверки. Совпадение транспортного API не гарантирует одинаковую поддержку ролей сообщений, формата результата и поведения инструкций. Сведения о расходе, которых провайдер не вернул, обозначайте как неизвестные.

### 8.4 Формирование агентского флоу

Произвольная агентская цепочка определяется составом узлов и правилами переходов [1]. В принятой реализации переходы хранятся как разрешённый граф, а результат агента проверяется до выбора следующего узла. Базовый исполнитель поддерживает последовательность, ветвление и ограниченные циклы. Паттерны последовательной обработки и маршрутизации рассмотрены также. Параллельные ветви подключаются отдельным расширением с правилами объединения результатов.

**Сквозной пример — консультация по BIV Platform.** Агент-классификатор определяет тему вопроса. Агент-консультант готовит ответ по переданным ему разрешённым материалам. Если тема неясна, приложение возвращает уточняющий вопрос. Пример не предполагает встроенный поиск по базе знаний.

| Текущий узел | Условие | Следующее действие |
|---|---|---|
| Классификатор | category = platform | Передать вопрос консультанту по платформе. |
| Классификатор | category = project | Передать вопрос консультанту прикладного проекта. |
| Классификатор | category = unknown | Вернуть уточнение пользователю и завершить текущее выполнение. |
| Любой консультант | Ответ успешно проверен | Вернуть финальное сообщение. |
| Любой узел | Ошибка, неверный формат или лимит | Завершить обработку согласованной ошибкой. |

**Псевдокод оркестрации.** Имена операций условные и не относятся к API библиотеки.

```
проверить права на диалог и поток
зафиксировать requestId и версии настроек
начать трассу выполнения
узел = начальный узел потока
пока узел существует:
    проверить оставшееся время и лимит шагов
    начать событие выполнения узла
    результат = вызвать агента с разрешённым контекстом
    проверить результат по ожидаемой схеме
    завершить событие узла с результатом или ошибкой
    если нужен ответ пользователя: вернуть уточнение
    если получен финальный ответ: завершить выполнение
    следующий = выбрать переход из разрешённого графа
    записать переход от узла к следующему
    узел = следующий
при любой ошибке: завершить трассу и вернуть ошибку
```

Если сценарий допускает цикл, задайте условие выхода и общий лимит итераций. При параллельных ветвях дополнительно определите правило объединения результатов, порядок сообщений и обработку частичного отказа. Не включайте параллельность только ради ускорения без проверки этих правил.

### 8.5 Алгоритм работы агента и инструменты

Внутри одного узла агент может выполнить несколько обращений к модели. На каждой итерации сервер формирует контекст из инструкции, допустимой истории, результатов предыдущих шагов и описаний разрешённых инструментов. Цикл заканчивается ответом, передачей управления, запросом ввода или ошибкой.

| Результат модели | Действие исполнителя |
|---|---|
| FINAL | Проверить формат и вернуть ответ текущего агента в поток. |
| CALL_TOOL | Проверить имя инструмента, аргументы и права; выполнить обработчик и добавить результат в контекст. |
| HANDOFF | Проверить целевого агента по разрешённым переходам и передать ему минимально необходимый контекст. |
| ASK_USER | Сохранить состояние ожидания и вернуть уточняющий вопрос. |

Это внутренние типы результата агента. Адаптер преобразует в них ответ конкретной модели: нативный вызов инструмента либо структурированный ответ. Неподдерживаемый или неверно сформированный результат становится ошибкой MODEL_OUTPUT_INVALID и не выполняется.

**Предлагаемый Java-контракт**

```
interface LlmProvider {
    ModelResponse generate(ModelRequest request);
    ModelCapabilities capabilities();
}
interface Agent {
    AgentResult execute(AgentContext context);
}
interface AgentTool {
    ToolResult execute(ToolContext context, JsonObject arguments);
}
```

Фрагмент показывает сигнатуры; DTO, импорты и реализации входят в проектный SDK. ModelRequest содержит сообщения, описание разрешённых инструментов и лимиты. AgentContext включает состояние потока и ссылки на провайдер, реестр инструментов и наблюдение. ToolContext несёт проверенный сервером контекст пользователя.

Реестр инструментов хранит имя, описание, JSON-схему аргументов, обработчик, тайм-аут и признак изменения данных. Для каждого агента задаётся allowlist инструментов. Модель не может добавлять инструмент, менять его адрес или подменять серверный контекст пользователя.

Для операций изменения данных сервер формирует предложение действия и при необходимости запрашивает подтверждение. Подтверждение привязывается к пользователю, точным аргументам, версии операции и сроку действия; любое изменение аргументов требует нового подтверждения. На выполнение повторно проверяются права и ключ идемпотентности.

Ограничивайте отдельно шаги графа, итерации агента и общее число вызовов LLM. Для стартового профиля примем 4 итерации на агента и 12 вызовов LLM на выполнение, включая повторы, при общем лимите времени 90 секунд.

### 8.6 Диалог и структура хранения

В принятой реализации история и состояние хранятся в PostgreSQL. Идентификаторы создаёт сервер. Для мультитенантного проекта tenantId включается во все проверки доступа и ключи изоляции; в проекте с одним контуром используется постоянное значение.

| Проектная таблица | Ключевые поля и назначение |
|---|---|
| AI_CONVERSATION | id, tenant_id, owner_id, flow_id, flow_version, status, checkpoint_json, created_at, updated_at. Диалог и состояние ожидания. |
| AI_MESSAGE | id, conversation_id, execution_id, sequence_no, role, content, created_at. История сообщений в определённом порядке. |
| AI_EXECUTION | id, conversation_id, request_id, payload_hash, status, trace_id, config_version, started_at, finished_at, error_code. Задание и результат обработки. |
| AI_STEP | id, execution_id, agent_id, attempt, parent_step_id, status, started_at, finished_at. Фактические шаги и повторы. |
| AI_EVENT | execution_id, sequence_no, type, payload_json, created_at. Журнал переходов и событий для чата и диагностики. |

На AI_EXECUTION задайте уникальность пары (conversation_id, request_id). Повтор того же ключа и тела возвращает существующее выполнение; другой payload_hash с тем же ключом возвращает конфликт. Для AI_MESSAGE и AI_EVENT обеспечьте уникальность sequence_no внутри соответствующего диалога или выполнения.

При приёме сообщения проверка отсутствия активного выполнения, запись сообщения и создание задания выполняются атомарно. Исполнитель захватывает задание с арендой и heartbeat, чтобы несколько экземпляров сервиса не обработали его одновременно. Не держите транзакцию БД открытой во время вызова LLM или внешнего инструмента.

### 8.7 Состояния и продолжение диалога

Выполнение проходит QUEUED → RUNNING → один из конечных статусов: COMPLETED, WAITING_INPUT, FAILED, CANCELLED или TIMED_OUT. WAITING_INPUT завершает текущую обработку сообщения, но оставляет диалог открытым для продолжения. Диалог имеет собственные состояния ACTIVE, WAITING_INPUT и CLOSED.

При уточнении сохраняйте в checkpoint_json ожидаемый агент, вопрос, версию потока и необходимый промежуточный результат. Следующее сообщение создаёт новое executionId, связывается с предыдущим выполнением и продолжает сценарий из сохранённой точки. Не восстанавливайте весь маршрут только из текста последней реплики.

После аварийного завершения процесса выполнение с истёкшей арендой получает FAILED с кодом EXECUTOR_INTERRUPTED. Автоматическое воспроизведение шагов изменения данных запрещено без проверки фактического результата и идемпотентности. Историю и события удаляйте по согласованной политике хранения; общий стартовый срок для тестового стенда — 30 дней.

### 8.8 Слой наблюдения

Требование наблюдения покрывает обращения и переходы между агентами. В предлагаемой реализации каждый запуск получает traceId; агент, вызов модели и инструмент формируют связанные span. OpenTelemetry поддерживает связь span в единую трассу и передачу контекста между сервисами. В Quarkus для этого используется расширение OpenTelemetry.

| Событие | Обязательная диагностическая информация |
|---|---|
| execution.started / finished | executionId, conversationId, flowId, версия, статус и длительность. |
| agent.started / finished | agentId, stepId, номер попытки, результат и длительность. |
| agent.transition | fromAgentId, toAgentId, код выбранного маршрута. |
| model.started / finished | providerId, modelId, длительность, категория ошибки, токены при наличии. |
| tool.started / finished | Имя инструмента, stepId, результат проверки прав, статус. |
| input.requested / limit.reached | Вид ожидания или сработавшего ограничения. |

**Пример записи перехода**

```
{
  "type": "agent.transition",
  "traceId": "9f2c3d4e5a6b7c8d9e0f112233445566",
  "executionId": "exec-42",
  "sequenceNo": 4,
  "fromAgentId": "request-classifier",
  "toAgentId": "platform-consultant",
  "routeCode": "category_platform"
}
```

Для восстановления маршрута сохраняйте AI_EVENT независимо от выборки трасс OpenTelemetry. Экспорт трасс может использовать sampling, но журнал маршрутов выполнений сохраняется полностью в пределах срока хранения. При сбое внешнего экспортера очередь ограничивается, ошибка видна в метриках, работа диалога может продолжаться.

Если записать обязательный журнал выполнения в БД невозможно, не начинайте новый шаг с внешним эффектом; завершите обработку технической ошибкой. События изменения настроек и выполнения бизнес-операций дополнительно передавайте в аудит платформы.

В трассах храните технические события, выбранные маршруты и разрешённые результаты. Не сохраняйте скрытые рассуждения модели. Тексты сообщений, документы, токены доступа и секреты исключайте из общей телеметрии; диагностическую запись содержимого включайте отдельно с ограниченным доступом.

Основные метрики: число и доля ошибочных выполнений, p95 длительности, длина очереди, частота тайм-аутов, число шагов и вызовов LLM, расход токенов при наличии. Не используйте идентификаторы диалогов и пользователей как метки метрик.

### 8.9 Серверный API диалога

Базовый путь API — /api/ai/v1. Все операции требуют аутентификации и проверки доступа. Сервер определяет пользователя, профиль модели и версии агентов.

| Метод и относительный путь | Назначение | Результат |
|---|---|---|
| POST /conversations | Создать диалог по flowId | 201 и conversationId. |
| GET /conversations/{id}/messages | Получить историю с cursor и limit | 200, сообщения и nextCursor. |
| POST /conversations/{id}/messages | Принять сообщение с requestId | 202, executionId и statusUrl. |
| POST /conversations/{id} /confirmations/{confirmationId} | Принять решение approve или reject с новым requestId | 202 и executionId продолжения. |
| GET /executions/{id} | Получить статус и итог | 200, статус, ответ или ошибка. |
| GET /executions/{id}/events | Получить события SSE | Поток разрешённых UI-событий. |
| POST /executions/{id}/cancel | Запросить остановку | 202 либо текущий конечный статус. |
| POST /conversations/{id}/close | Закрыть диалог | 200, статус CLOSED. |

История возвращается по sequenceNo, порциями по 50 сообщений, максимум — 200. Курсор не даёт доступ к чужому диалогу.

Повтор принятого requestId возвращает прежнее executionId: 202 для активного выполнения, 200 для завершённого. Другое сообщение при активном выполнении получает 409 DIALOG_BUSY. После WAITING_INPUT новый вопрос использует новый requestId.

Подтверждение проходит ту же дедупликацию. Истёкшее, чужое или не соответствующее действию confirmationId отклоняется. Решение reject завершает ожидание без вызова инструмента. Закрытие диалога допускается после завершения активного выполнения.

### 8.10 Доставка событий и отмена

Чат опрашивает statusUrl до конечного статуса. SSE передаёт статусы, итог и, при поддержке модели, фрагменты ответа. События содержат executionId и возрастающий sequenceNo. При восстановлении клиент передаёт последний event id; если события удалены, перечитывает состояние через REST.

Клиент сопоставляет сообщения и события по messageId и executionId для защиты от дублей. Для SSE с bearer-аутентификацией используйте клиент с поддержкой заголовков; токен в URL не передавайте.

Отмена устанавливает cancelRequested; статус остаётся RUNNING до подтверждения остановки исполнителем, затем становится CANCELLED. Новые шаги не запускаются, текущий вызов по возможности прерывается. Уже выполненная бизнес-операция автоматически не откатывается. Если инструмент завершился одновременно с отменой, сервер сохраняет фактический результат и сообщает его пользователю.

## 8.11 Примеры обмена и обработки ошибок

### Создание диалога

```
POST /api/ai/v1/conversations
Content-Type: application/json

{"flowId":"platform-help"}
```

### Отправка сообщения

```
POST /api/ai/v1/conversations/conv-17/messages
Content-Type: application/json

{"requestId":"req-001","text":"Как подключить модуль?"}
```

```
{
  "conversationId": "conv-17",
  "executionId": "exec-42",
  "status": "QUEUED",
  "statusUrl": "/api/ai/v1/executions/exec-42"
}
```

При GET статуса завершённого выполнения сервер возвращает status = COMPLETED и message с id, role = assistant и text. Для WAITING_INPUT дополнительно возвращаются inputType и вопрос; для подтверждения операции — безопасное описание действия и confirmationId. Идентификаторы conv-17 и exec-42 здесь сокращены для наглядности.

| Ситуация | HTTP при приёме запроса | Проектный код |
|---|---|---|
| Неверное тело или пустой текст | 400 | INVALID_REQUEST |
| Нет входа / нет права | 401 / 403 | UNAUTHENTICATED / ACCESS_DENIED |
| Диалог отсутствует или скрыт политикой | 404 | DIALOG_NOT_FOUND |
| Диалог занят или ключ повторён с другим телом | 409 | DIALOG_BUSY / REQUEST_CONFLICT |
| Превышен лимит запросов | 429 | RATE_LIMITED |
| Сервис не может принять задание | 503 | SERVICE_UNAVAILABLE |

Ошибка после принятия задания не меняет уже отправленный HTTP 202. GET статуса возвращает HTTP 200 с status = FAILED или TIMED_OUT и объектом error: code, userMessage, retryable, traceId. Для модели используйте MODEL_UNAVAILABLE, MODEL_TIMEOUT или MODEL_OUTPUT_INVALID; для ограничений — STEP_LIMIT или TOKEN_LIMIT.

Пользователю показывайте краткое описание и идентификатор обращения. Stack trace, полный prompt, ключи и служебные ответы провайдера остаются в защищённой диагностике.

# *9 Документация для Front End разработчика*

## 9.1 Подключение компоненты чата

В проектном варианте чат поставляется как Angular-библиотека @biv/ai-chat. Она отвечает за отображение диалога и состояния взаимодействия. Политики доступа, выполнение агентов и вызовы LLM остаются в backend.

Установите опубликованную версию библиотеки из внутреннего npm-репозитория и зарегистрируйте экспорт BivAiChatComponent в приложении. Версии Angular и библиотеки фиксируйте в lock-файле. Ниже приведён предлагаемый шаблон компоненты; это проектный API из данного руководства.

```
<biv-ai-chat
  [flowId]="'platform-help'"
  [conversationId]="activeConversationId"
  [transport]="chatTransport"
  [readOnly]="false"
  (conversationChanged)="onConversationChanged($event)"
  (executionChanged)="onExecutionChanged($event)"
  (chatError)="showChatError($event)">
</biv-ai-chat>
```

chatTransport — адаптер к API раздела 8.9. Он использует общий HTTP-клиент приложения для аутентификации, обрабатывает ошибки и при необходимости подписывается на SSE. Его методы: createConversation, loadMessages, sendMessage, submitConfirmation, getExecution, cancelExecution и closeConversation.

## 9.2 Входные параметры

| Параметр | Тип | Назначение |
|---|---|---|
| flowId | string | Разрешённый сценарий для нового диалога. |
| conversationId | string или null | Открыть существующий диалог; null означает новый. |
| transport | ChatTransport | Адаптер серверного API и доставки событий. |
| readOnly | boolean | Просмотр истории без отправки; по умолчанию false. |
| placeholder | string | Подсказка в поле ввода. |
| showProgress | boolean | Показывать безопасные статусы обработки; по умолчанию true. |

При открытии существующего диалога flowId берётся из серверной записи; несовпадающий входной flowId не изменяет сценарий. Компонента не принимает endpoint модели, системный prompt или секрет провайдера.

### 9.3 События и методы компоненты

| Событие | Данные | Когда возникает |
|---|---|---|
| conversationChanged | conversationId | Создан или открыт другой диалог. |
| messageSent | requestId, messageId | Сервер принял сообщение. |
| executionChanged | executionId, status | Изменилось состояние обработки. |
| responseReceived | messageId, text | Получен завершённый ответ либо уточнение. |
| chatError | code, userMessage, traceId | Требуется показать ошибку. |

| Метод | Назначение |
|---|---|
| openConversation(id) | Загрузить историю доступного диалога. |
| startNewConversation() | Переключиться на новый диалог, сохранив старую историю на сервере. |
| send(text) | Отправить сообщение и начать отслеживание выполнения. |
| cancelCurrentExecution() | Запросить остановку активного выполнения. |
| reloadHistory() | Перечитать историю после восстановления связи. |

### 9.4 Состояния интерфейса

После отправки показывайте сообщение пользователя и индикатор обработки. До принятия сервером помечайте сообщение как отправляемое; после ответа 202 — как принятое. Блокируйте повторное нажатие отправки. Сетевой повтор использует прежний requestId.

При WAITING_INPUT покажите вопрос и снова разрешите ввод. Подтверждение операции оформляйте отдельными кнопками: запрос содержит confirmationId и решение пользователя; обычный текст «да» не считается серверным подтверждением. После обновления страницы восстановите историю и статус активного выполнения через API.

При разрыве SSE перейдите на опрос статуса. Не показывайте технические трассы и промежуточные результаты инструментов обычному пользователю. Полученный Markdown очищайте перед отображением; исполняемый HTML и опасные URL запрещайте.

### 9.5 Стилизация и запуск

Используйте дизайн-систему BIV Platform. Проектный корневой CSS-класс — biv-ai-chat; внутренние классы: biv-ai-chat__messages, __message, __composer, __progress и __error. Цвета, отступы и типографику задавайте через дизайн-токены, сохраняя фокус клавиатуры, подписи кнопок и читаемость сообщений.

Клонируйте репозиторий UI-примера из паспорта поставки, установите зависимости по lock-файлу, настройте proxy на тестовый backend и запустите dev-скрипт проекта. Проверьте вход, создание диалога, уточняющий вопрос, ошибку модели и восстановление после перезагрузки страницы.

# *10 Администрирование и запуск*

## 10.1 Управление настройками

В базовом варианте профили моделей и определения агентов хранятся в конфигурации приложения и ресурсах поставки. Изменения проходят review и выпускаются вместе с версией. Отдельный экран администратора не является обязательным условием работы модуля.

Администратор управляет разрешёнными моделями, потоками, лимитами и сроками хранения. Секреты меняются через штатное хранилище секретов. Для каждого изменения фиксируются автор, время, предыдущая и новая версия, результат применения.

Новую конфигурацию сначала проверяйте на тестовом сценарии. При активации выполняйте проверку графа, доступности профилей и совместимости схем. Текущие выполнения продолжают использовать закреплённую версию; новые получают активную версию. Для диалогов со старой версией сохраняйте её доступной до завершения или выполните управляемую миграцию.

## 10.2 Развёртывание

1. Выполните миграции таблиц AI_CONVERSATION, AI_MESSAGE, AI_EXECUTION, AI_STEP и AI_EVENT.
2. Разверните сервис с ограниченным пулом исполнителей, параметрами БД, секретов и моделей.
3. Подключите авторизацию приложения, аудит и экспорт телеметрии.
4. Загрузите определения агентов и потоков; проверьте их версии и доступность.
5. Подключите чат к серверному API и выполните проверки из раздела 11.
6. Откройте сценарий согласованной группе пользователей и проверьте метрики под ожидаемой нагрузкой.

Проверка liveness отражает состояние процесса. Readiness проверяет возможность принять и сохранить задание; кратковременную недоступность облачной модели показывайте как отдельный признак деградации. Это позволяет не провоцировать перезапуск всех экземпляров из-за внешнего отказа.

## 10.3 Масштабирование и обновление

Несколько экземпляров backend используют общее хранилище и атомарный захват заданий. Число параллельных вызовов ограничивайте отдельно для каждого провайдера. При переполнении очереди отклоняйте новые обращения управляемо, не накапливая неограниченное число заданий.

Перед обновлением прекратите приём новых заданий на выводимом экземпляре и дождитесь завершения активных в пределах заданного времени. Для отката сохраняйте предыдущие версии библиотек, конфигурации и совместимые миграции. Не откатывайте схему с потерей истории; разрушительные изменения выполняйте отдельно после переноса данных.

# *11 Примеры использования и проверка*

## 11.1 Консультация с передачей между агентами

Вход: «Какие компоненты нужны для интерфейса приложения?» Классификатор возвращает category = platform. Поток передаёт вопрос консультанту по платформе, которому приложение предоставило доступный текст о дизайн-системе и конструкторах. Консультант формирует ответ; сервер завершает выполнение.

Затем пользователь спрашивает: «А что подключать на backend?» Новый запуск использует тот же conversationId и разрешённую историю, чтобы сохранить тему. Проверка успешна, если в журнале виден переход между агентами, ответы связаны с одним диалогом, а пользователь другого аккаунта не может прочитать историю.

## 11.2 Проверка статуса заявки через инструмент

Вход: «Какой статус у заявки 12345?» Агент выбирает разрешённый инструмент getRequestStatus с аргументом requestId = 12345. Сервер проверяет схему аргументов и доступ пользователя к заявке, затем вызывает прикладной сервис. Агент получает только разрешённые поля результата и формулирует ответ.

Если права отсутствуют, инструмент возвращает контролируемую ошибку, а агент сообщает, что данные недоступны. Если номер заявки не указан, агент задаёт уточнение и сохраняет ожидание. Пример демонстрирует цикл модель → инструмент → модель и не предполагает наличия отдельного модуля заявок в BIV Platform.

## 11.3 Минимальный набор проверок

| Проверка | Ожидаемый результат |
|---|---|
| Локальная и облачная модели | Один сценарий работает с обоими профилями без изменения бизнес-маршрута. |
| Ветвление и неизвестная категория | Выбирается допустимый агент; неизвестная категория приводит к уточнению. |
| Продолжение диалога | История и сохранённая точка ожидания восстанавливают контекст. |
| Чужой conversationId | Доступ отклонён до чтения истории и вызова LLM. |
| Повтор requestId | Создано только одно выполнение; другое тело даёт конфликт. |
| Тайм-аут и цикл | Выполнение завершается по лимиту, в журнале есть причина. |
| Подмена инструмента или аргументов | Неразрешённый вызов не выполнен. |
| Сбой SSE и перезапуск сервиса | Статус восстанавливается; нет дублей сообщений и бизнес-действий. |
| Наблюдение и секреты | Маршрут виден целиком; секреты отсутствуют в логах и событиях UI. |

Маршрутизацию проверяйте на заглушке провайдера с фиксированными ответами. На реальных моделях проверяйте смысл, допустимые действия и формат результата, а не дословное совпадение текста. Перед выпуском зафиксируйте версии модели, инструкций и набор контрольных запросов.

## *12 Возможные ошибки и их решения*

Начинайте диагностику с executionId и traceId. Найдите первый неуспешный шаг, его провайдер, попытку и категорию ошибки. Не отправляйте пользовательские сообщения и секреты в общедоступные каналы сопровождения.

| Симптом | Что проверить | Действие |
|---|---|---|
| Нет ответа модели | Endpoint, доступ из backend, секрет, modelId, тайм-аут | Проверить профиль отдельным тестовым вызовом; исправить конфигурацию. |
| Поток зацикливается | Повторяющиеся переходы, условие завершения, счётчики | Исправить граф; сохранить лимит шагов как защиту. |
| Агент забывает предыдущий вопрос | conversationId, загрузку истории, обрезку контекста | Передавать тот же диалог; проверить сборку сообщений и checkpoint. |
| Ответ не соответствует схеме | Инструкцию, возможности модели, валидатор результата | Вернуть контролируемую ошибку; скорректировать схему и промпт. |
| Переходы отсутствуют в трассе | traceId, передачу контекста, запись AI_EVENT | Восстановить инструментирование и экспорт; сверить журнал выполнения. |
| Сообщения дублируются | requestId, уникальные ключи, повторную доставку SSE | Исправить дедупликацию на сервере и сопоставление messageId в UI. |
| В диалоге смешались ответы | Параллельные запуски, атомарность проверки занятости | Разрешать один активный запуск; повторить тест на двух экземплярах. |
| После обновления пропало продолжение | Доступность версии потока и формат checkpoint | Вернуть закреплённую версию или выполнить явную миграцию состояния. |
| Пользователь видит чужие данные | Проверки владельца, tenantId, авторизацию инструмента | Закрыть доступ к сценарию до исправления и проверить затронутые диалоги. |

### 12.1 Что приложить к обращению в поддержку

Укажите окружение, версию модуля и приложения, flowId и версию потока, профиль модели, время ошибки, executionId, traceId, код ошибки и шаги воспроизведения. Приложите обезличенный пример входа и ожидаемый результат. Ссылку на защищённую трассу передавайте только получателям с правом просмотра.

### 12.2 Проверка исправления

Повторите минимальный сценарий сбоя, затем проверьте соседнюю ветку потока и обычный диалог. Если менялась модель или системная инструкция, выполните контрольный набор запросов из раздела 11. Проверку производительности повторяйте при изменении числа вызовов модели, размера контекста или параллельности.

## *Приложение A Паспорт подключения*

Паспорт заполняется для конкретной поставки и хранится рядом с руководством. Он связывает проектные контракты с опубликованными артефактами, окружением и ответственными за сопровождение.

| Поле | Что зафиксировать |
|---|---|
| Версия поставки | Версии модуля, backend и frontend; дата релиза. |
| Репозитории | URL исходного кода, Maven- и npm-репозиториев, UI-примера. |
| Серверные библиотеки | Фактические groupId, artifactId и версии провайдера, флоу и наблюдения. |
| Компонента чата | Пакет, версия, экспортируемые компоненты и совместимый Angular. |
| Совместимость | Java, Quarkus, PostgreSQL и компоненты BIV Platform. |
| Провайдеры | Адаптеры, адреса по окружениям, modelId, ограничения и ссылки на секреты. |
| Контракты | OpenAPI, JSON-схемы, перечень событий и параметры конфигурации. |
| Состояние | Миграции БД, сроки хранения, резервирование и восстановление. |
| Наблюдение | Адреса защищённых журналов, трасс и панелей мониторинга. |
| Эксплуатация | Лимиты, политика повтора, порядок обновления, отката и контакты поддержки. |

### Соответствие исходным требованиям

| Требование | Где описано |
|---|---|
| Поддержка локальных или облачных моделей | 2.1, 8.3, 11.3 |
| Произвольные агентские цепочки | 3, 8.4, 8.5 |
| Диалоговое взаимодействие | 8.6–8.11, 9 |
| Отслеживание обращений и переходов | 8.8, 11, 12 |
| Провайдер LLM | 8.1, 8.3, 8.5 |
| Библиотека агентского флоу | 8.4–8.7 |
| Библиотека слоя наблюдения | 8.8 |
| Компонента интерфейса чата | 9 |
| Документация для разработчиков | Настоящее руководство |

### Границы проектной реализации

Определённые в руководстве названия пакетов com.bivgroup.ai и @biv/ai-chat, Java-интерфейсы, параметры biv.ai, REST-маршруты, таблицы AI_*, состояния, лимиты и механизм подтверждений являются предложенным техническим решением. Они дополняют исходные требования и должны быть реализованы либо сопоставлены с фактическим SDK при подготовке поставки.

Версии опубликованных артефактов, адреса внутренних репозиториев и секреты задаются в паспорте поставки. Документ не присваивает им вымышленные рабочие значения. Примеры кода описывают контракт и алгоритмы, а не самостоятельное приложение для запуска.