Руководство администратора ПО

«Чат-бот для службы поддержки»

1. ВВЕДЕНИЕ

1.1. Область применения

Настоящий документ предназначен для администраторов, эксплуатирующих программное обеспечение «Интеллектуальный чат-сервис поддержки пользователей» (далее — Система).

Руководство содержит порядок развертывания, настройки, администрирования и эксплуатации Системы, а также описание механизмов интеграции с внешними сервисами и операторскими контурами поддержки.

Система предназначена для автоматизации первичной обработки пользовательских обращений в чате с возможностью передачи диалога оператору поддержки и интеграции с внешними backend-сервисами по протоколам HTTP (REST) и RabbitMQ (RPC).

1.2. Перечень выполняемых функций администратора/оператора

В перечень выполняемых функций администратора Системы входят:

- Развертывание Системы в контейнерной среде (Docker, Kubernetes или иной оркестратор, применяемый на стороне Заказчика);
- Настройка переменных окружения и параметров интеграции с внешними сервисами;
- Конфигурация подключения к backend-сервисам (REST или RabbitMQ RPC);
- Настройка ключей доступа и параметров безопасности;
- Организация масштабирования Системы в рамках инфраструктуры Заказчика;
- Мониторинг работоспособности сервиса;
- Анализ логов и восстановление работоспособности при сбоях.

Масштабирование Системы выполняется путем развертывания дополнительных экземпляров (реплик) сервиса за ingress/reverse-proxy либо средствами используемого оркестратора.

Конфигурация оркестратора (Docker Compose, Docker Swarm, Kubernetes, Helm-чарты, манифесты и иные инфраструктурные артефакты) осуществляется на стороне Заказчика и определяется его внутренними стандартами эксплуатации.

1.3. Уровень подготовки администратора/оператора

Администратор Системы должен:

- владеть навыками работы с контейнерной инфраструктурой;
- уметь настраивать reverse-proxy/ingress;
- понимать принципы взаимодействия сервисов по HTTP и RabbitMQ;
- иметь опыт эксплуатации backend-сервисов в среде Linux.

Рекомендуемая численность персонала для эксплуатации Системы — 1 штатная единица.

1.4. Перечень документации

В состав документации, с которой необходимо ознакомиться администратору Системы входят:

- Функциональная спецификация системы
- Техническое задание на разработку системы
- Настоящее руководство администратора

2. УСТАНОВКА СИСТЕМЫ

2.1. Системные требования

2.1.1. Аппаратные требования

Сервис предназначен для запуска в контейнеризованной среде. Ниже приведены минимальные рекомендуемые требования для одного экземпляра приложения (одного pod/контейнера API):

- CPU: не менее 2 логических ядер;
- Оперативная память: не менее 2 ГБ (рекомендуется 4 ГБ при высокой нагрузке);
- Дисковое пространство: от 10 ГБ (без учета резервных копий).

Фактические требования зависят от интенсивности входящего потока данных. При горизонтальном масштабировании требования применяются к каждому экземпляру отдельно.

2.1.2. Программные требования

Сервис разворачивается в контейнерной среде и требует:

- Kubernetes, Docker Swarm или другой оркестратор;
- Reverse-proxy или ingress-контроллер (Nginx, Traefik и др.)

Операционная система хоста должна поддерживать запуск контейнеров (Linux-дистрибутив серверного класса).

2.2. Порядок установки

- Получить исходный код Системы.
- Собрать Docker-образ:
  docker build -t smart-chat .
- Подготовить файл конфигурации переменных окружения.
- Запустить контейнер:
  docker run -d --env-file .env -p 8080:8080 smart-chat
- Настроить ingress/reverse-proxy.
- Проверить доступность сервиса.

3. НАСТРОЙКА СИСТЕМЫ

3.1. Общие сведения

Система поддерживает интеграцию с внешними backend-сервисами:

- по HTTP (REST);
- по RabbitMQ (RPC).

Интеграция по HTTP возможна при наличии внешнего сервиса, реализующего ожидаемый набор REST-эндпоинтов.

Функциональность операторской панели поддержки поддерживается Системой на уровне протокола взаимодействия, при этом пользовательский интерфейс операторов реализуется и сопровождается Заказчиком либо внешним специализированным модулем.

3.2. Конфигурируемые параметры

Основные переменные окружения:

- HOST — адрес и порт, на котором запускается сервис
- JWT_SECRET — секрет для подписи JWT-токенов
- ACCESS_TOKEN — токен доступа к внешнему REST backend
- REST_BACKEND_URL — адрес внешнего REST-сервиса
- AMQP_URL — строка подключения к RabbitMQ
- AMQP_QUEUE — имя очереди RPC
- INTEGRATION_MODE — режим интеграции (rest, rpc, db)
- AUTO_ESCALATION_THRESHOLD — количество нераспознанных сообщений до эскалации оператору

4. ОПИСАНИЕ API

4.1. Общие сведения

Система предоставляет REST API по HTTP(S).

Все запросы и ответы передаются в формате JSON.

Аутентификация выполняется с использованием JWT.

Заголовок Content-Type должен быть application/json

4.2. Методы API

4.2.1. Регистрация пользователя

Метод предназначен для создания учетной записи пользователя в Системе.

HTTP

POST /api/register

Тело запроса

```
{
  "username": "string",
  "password": "string"
}
```

Параметры

- username (string, обязательный) — уникальное имя пользователя.
- password (string, обязательный) — пароль пользователя.

Успешный ответ

```
{
  "message": "user registered"
}
```

Возможные ошибки

- 400 — пользователь уже существует
- 400 — некорректные входные данные

4.2.2. Авторизация пользователя

Метод предназначен для аутентификации пользователя и получения JWT-токена доступа.

HTTP

POST /api/login

Тело запроса

```
{
 "username": "string",
 "password": "string"
}
```

Параметры

- username (string, обязательный) — имя пользователя
- password (string, обязательный) — пароль пользователя

Успешный ответ

```
{
 "token": "JWT_TOKEN"
}
```

Описание ответа

- token (string) — JWT-токен, используемый в заголовке:
  Authorization: Bearer <token>

Возможные ошибки

- 401 — неверные учетные данные

4.2.3. Отправка сообщения в чат

Метод предназначен для передачи пользовательского сообщения в Систему и получения ответа бота либо статуса эскалации оператору.

HTTP

POST /api/chat

Заголовки

- Authorization: Bearer <JWT>
- Content-Type: application/json

Тело запроса

```
{
 "message": "string"
}
```

Параметры

- message (string, обязательный) — текст сообщения пользователя.

Логика обработки

Система выполняет анализ сообщения.

Определяется базовый интент (по ключевым словам).

В зависимости от режима:

- выполняется HTTP-вызов к внешнему backend;
- выполняется RPC-запрос через RabbitMQ;
- либо используется локальный режим.

При превышении порога нераспознанных сообщений инициируется перевод к оператору.

Успешный ответ

```
{
  "reply": "string",
  "mode": "bot | waiting_operator | operator"
}
```

Параметры ответа

- reply (string) — текст ответа пользователю
- mode (string) — текущий режим диалога:
- bot — автоматический режим
- waiting_operator — ожидание оператора
- operator — диалог с оператором

Возможные ошибки

- 401 — отсутствует или недействительный JWT
- 500 — ошибка интеграционного backend-сервиса

4.2.4. Принудительная эскалация оператору

Метод инициирует перевод пользователя в режим ожидания оператора.

HTTP

POST /api/escalate

Заголовки

- Authorization: Bearer <JWT>

Успешный ответ

```
{
  "status": "waiting_operator"
}
```

Описание

Метод переводит текущую сессию в режим ожидания оператора поддержки.

Фактическая обработка очереди операторов осуществляется внешним операторским контуром.

4.2.5. Возврат к автоматическому режиму

Метод переводит диалог обратно в автоматический режим.

HTTP

POST /api/return-to-bot

Заголовки

- Authorization: Bearer <JWT>

Успешный ответ

```
{
  "status": "bot"
}
```

4.2.6. Проверка состояния сервиса

Метод используется для проверки работоспособности Системы.

HTTP

GET /health

Успешный ответ

```
{
  "status": "ok"
}
```

Используется для мониторинга и health-check в оркестраторах.

5. РЕЖИМЫ ИНТЕГРАЦИИ

5.1. Поддерживаемые режимы

Система поддерживает следующие режимы интеграции (выбор режима выполняется переменной окружения AUTH_BACKEND):

- DB-режим (AUTH_BACKEND=db) — автономный режим на локальном хранилище (SQLite). Предназначен для тестирования и демо-эксплуатации.
- REST-режим (AUTH_BACKEND=rest) — интеграция с внешним backend-сервисом по HTTP (REST).
- RabbitMQ RPC-режим (AUTH_BACKEND=rabbitmq) — интеграция с внешним backend-сервисом через RabbitMQ в формате RPC.

Дополнительно контур взаимодействия с операторской поддержкой может настраиваться отдельными параметрами OPERATOR_*. Если OPERATOR_* не заданы, используются значения, заданные для основного backend-контура (REST или RabbitMQ).

5.2. REST-режим интеграции (AUTH_BACKEND=rest)

5.2.1. Назначение

В REST-режиме Система обращается к внешнему backend-сервису Заказчика по HTTP(S) для выполнения операций:

- регистрация/поиск пользователей,
- получение и обновление баланса,
- получение истории ставок,
- получение истории выводов средств,
- работа с сессией оператора (очередь/статусы/обмен сообщениями).

5.2.2. Параметры подключения

- AUTH_REST_ENDPOINT (обязательно) — базовый URL внешнего REST backend (например, https://backend.example/api).
- AUTH_REST_BEARER_TOKEN (необязательно) — Bearer-токен, добавляется в заголовок Authorization.
- AUTH_REST_TIMEOUT_SECONDS (необязательно, по умолчанию 5) — таймаут HTTP-запросов.

Отдельно для операторского контура (если требуется иной endpoint/токен/таймаут):

- OPERATOR_REST_ENDPOINT (необязательно) — базовый URL операторского backend (по умолчанию равен AUTH_REST_ENDPOINT).
- OPERATOR_REST_BEARER_TOKEN (необязательно) — Bearer-токен (по умолчанию равен AUTH_REST_BEARER_TOKEN).

- OPERATOR_REST_TIMEOUT_SECONDS (необязательно) — таймаут (по умолчанию равен AUTH_REST_TIMEOUT_SECONDS).

5.2.3. Общие требования к ответам внешнего backend

Формат данных: JSON

Заголовок запроса: Content-Type: application/json

Заголовок авторизации (если задан токен): Authorization: Bearer <token>

Коды ответов:

- 2xx — успешное выполнение
- 404 — объект не найден (для части операций трактуется как “данных нет”)
- 4xx/5xx — ошибка (Система считает вызов неуспешным)

Также предусмотрена повторная попытка выполнения запроса при сетевой ошибке: выполняется один повтор с задержкой 10 секунд (повтор выполняется только при ошибке транспорта/соединения).

5.2.4. Вызываемые REST-эндпоинты и параметры

Операции с пользователями (аутентификация/регистрация):

POST /users

Тело запроса:

```
{
  "username": "string",
  "email": "string",
  "password": "string"
}
```

Ожидаемый ответ:

- 2xx и JSON с данными пользователя или пустое тело (оба варианта допускаются).
- при ошибке — 4xx/5xx.

GET /users/by-username/{username}

Параметры:

- {username} — имя пользователя (в URL)

Ожидаемый ответ:

- 200 и JSON пользователя
- 404 — пользователь не найден

Операции по балансу:

GET /balances/by-user?user_id={id}

Query-параметры:

- user_id (int64)

Ожидаемый ответ:

- 200 и JSON баланса:


```
{
 "id": 1,
 "user_id": 100,
 "amount": 123.45,
 "frozen": 0,
 "updated_at": "2026-02-23T12:34:56Z"
}
```


- 404 — баланс отсутствует

POST /balances

Тело запроса:


```
{ "user_id": 100 }
```


Ожидаемый ответ:

- 200/201 и JSON созданного баланса (структура как выше)

PUT /balances/{user_id}

Тело запроса: JSON объекта баланса (как в ответе GET /balances/by-user)

Ожидаемый ответ:

- 2xx без дополнительных требований к телу ответа

Операции по ставкам:

GET /bets?user_id={id}&limit={n}

Query-параметры:

- user_id (int64)
- limit (int)

Ожидаемый ответ:

- 200 и JSON-массив ставок (может быть пустым)
- 404 — трактуется как “данных нет” (пустой список)

GET /bets/by-date-range?user_id={id}&start_date={RFC3339}&end_date={RFC3339}

Query-параметры:

- user_id (int64)
- start_date (string, RFC3339)
- end_date (string, RFC3339)

Ожидаемый ответ:

- 200 и JSON-массив ставок (может быть пустым)
- 404 — трактуется как “данных нет” (пустой список)

POST /bets

Тело запроса: JSON объекта ставки:

```
{
  "id": 1,
  "user_id": 100,
  "amount": 10,
  "event_type": "string",
  "event_name": "string",
  "status": "pending|win|lose",
  "win_amount": 0,
  "created_at": "2026-02-23T12:34:56Z"
}
```

Ожидаемый ответ:

- 2xx (тело ответа не критично)

Операции по заявкам на вывод:

GET /withdrawals?user_id={id}

Query-параметры:

- user_id (int64)

Ожидаемый ответ:

- 200 и JSON-массив заявок (может быть пустым)
- 404 — трактуется как “данных нет” (пустой список)

GET /withdrawals/{id}

Параметры:

- {id} — идентификатор заявки

Ожидаемый ответ:

- 200 и JSON заявки
- 404 — заявка не найдена

POST /withdrawals

Тело запроса: JSON объекта заявки:

```
{
 "id": 1,
 "user_id": 100,
 "amount": 50,
 "method": "card|wallet",
 "status": "pending|processing|completed|rejected",
 "created_at": "2026-02-23T12:34:56Z",
 "processed_at": "2026-02-23T13:00:00Z"
}
```

Ожидаемый ответ:

- 2xx (тело ответа не критично)

Операторская поддержка (контур Заказчика):

GET /operator/session?user_id={id}

Query-параметры:

- user_id (int64)

Ожидаемый ответ:

- 200 и JSON сессии:

```
{
 "user_id": 100,
 "mode": "bot|waiting_operator|operator",
 "queue_position": 3,
 "operator_id": 501,
 "status_message": "string",
 "last_operator_text": "string"
}
```

- 404 — трактуется как отсутствие операторской сессии (диалог в режиме бота)

POST /operator/request

Тело запроса:

```
{
 "user_id": 100,
 "message": "string"
}
```

Ожидаемый ответ:

- 200 и JSON сессии (как выше). Если mode не задан, Система трактует состояние как ожидание оператора.

POST /operator/user-message

Тело запроса:

```
{
 "user_id": 100,
 "message": "string"
}
```

Ожидаемый ответ:

- 200 и JSON сессии (как выше)

5.3. RabbitMQ RPC-режим интеграции (AUTH_BACKEND=rabbitmq)

5.3.1. Назначение

В RabbitMQ RPC-режиме Система публикует RPC-запросы в RabbitMQ и ожидает RPC-ответ от внешнего backend-сервиса Заказчика (обработчик RPC находится на стороне Заказчика).

5.3.2. Параметры подключения

- AUTH_RABBITMQ_URL (обязательно) — строка подключения RabbitMQ (например, amqp://user:pass@host:5672/).
- AUTH_RABBITMQ_EXCHANGE (необязательно) — exchange для публикации (может быть пустым, в этом случае используется обменник по умолчанию).
- AUTH_RABBITMQ_ROUTING_KEY (необязательно, по умолчанию smart-chat.rpc) — routing key для RPC-запросов.
- AUTH_RABBITMQ_TIMEOUT_SECONDS (необязательно, по умолчанию 5) — таймаут ожидания RPC-ответа.

Отдельно для операторского контура (если используется иной URL/exchange/routing key):

- OPERATOR_RABBITMQ_URL
- OPERATOR_RABBITMQ_EXCHANGE
- OPERATOR_RABBITMQ_ROUTING_KEY
- OPERATOR_RABBITMQ_TIMEOUT_SECONDS

5.3.3. Формат RPC-запроса

Система публикует сообщение (Content-Type: application/json) со структурой:

```
{
 "method": "string",
 "payload": { }
}
```

Где:

- method — имя RPC-метода,
- payload — параметры вызова (объект JSON).

5.3.4. Формат RPC-ответа

Ожидаемый формат ответа:

```
{
 "data": {},
 "error": "string",
 "code": 0,
 "status": 0,
 "not_found": false
}
```

Где:

- data — полезная нагрузка (объект/массив/примитив) для успешного вызова,
- error — строка ошибки (если непустая, вызов считается неуспешным),
- not_found / code=404 / status=404 — признак “не найдено” (в ряде операций трактуется как отсутствие данных).

5.3.5. Используемые RPC-методы, параметры и ожидаемые данные

Операции с пользователями:

auth.user.create

Payload:

```
{ "username": "string", "email": "string", "password": "string" }
```

Ожидаемый data: объект пользователя (минимально — id, опционально created_at).

auth.user.get_by_username

Payload:

```
{ "username": "string" }
```

Ожидаемый data: объект пользователя, либо ответ “not_found”.

Операции по балансу:

data.balance.get_by_user_id

Payload:

```
{ "user_id": 100 }
```

Ожидаемый data: объект баланса, либо “not_found”.

data.balance.create

Payload:

```
{ "user_id": 100 }
```

Ожидаемый data: созданный баланс.

data.balance.update

Payload: объект баланса:

```
{ "id": 1, "user_id": 100, "amount": 123.45, "frozen": 0, "updated_at": "..." }
```

Ожидаемый data: допускается пустым (успешность определяется отсутствием error).

Операции по ставкам:

data.bet.get_by_user_id

Payload:

```
{ "user_id": 100, "limit": 20 }
```

Ожидаемый data: массив ставок (может быть пустым), либо “not_found” (трактуется как пустой список).

data.bet.get_by_user_id_and_date_range

Payload:

```
{ "user_id": 100, "start_date": "RFC3339", "end_date": "RFC3339" }
```

Ожидаемый data: массив ставок (может быть пустым).

data.bet.create

Payload: объект ставки (см. структуру в REST-режиме).

Ожидаемый data: допускается пустым.

Операции по заявкам на вывод:

data.withdrawal.get_by_user_id

Payload:

```
{ "user_id": 100 }
```

Ожидаемый data: массив заявок (может быть пустым).

data.withdrawal.get_by_id

Payload:

```
{ "id": 1 }
```

Ожидаемый data: объект заявки, либо “not_found”.

data.withdrawal.create

Payload: объект заявки (см. структуру в REST-режиме).

Ожидаемый data: допускается пустым.

Операторская поддержка (контур Заказчика):

operator.session.get

Payload:

```
{ "user_id": 100 }
```

Ожидаемый data: объект операторской сессии либо “not_found” (в этом случае Система трактует режим как bot).

operator.session.request

Payload:

```
{ "user_id": 100, "message": "string" }
```

Ожидаемый data: объект операторской сессии.

operator.session.user_message

Payload:

```
{ "user_id": 100, "message": "string" }
```

Ожидаемый data: объект операторской сессии.

5.4. DB-режим (AUTH_BACKEND=db)

DB-режим использует локальное файловое хранилище SQLite, путь к файлу задается параметром DATABASE_PATH (если не задан — используется путь по умолчанию).

Режим предназначен для:

- тестирования,
- демонстрационных стендов,
- отладки интеграций и сценариев.

Для промышленной эксплуатации, предполагающей взаимодействие с внешними системами Заказчика и операторским контуром поддержки, рекомендуется использовать REST-режим или RabbitMQ RPC-режим.