

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программные средства обработки событий на
основе механизма Веб-хук**

Студент _____ Градсков Алексей Владимирович
(подпись, дата) (ФИО полностью)

Руководитель _____ Гурьянов Л.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»

Заведующий кафедрой

_____ А.Ю. Козлов

«__» _____ 2026 г

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Градсков Алексей Владимирович гр.22ВП1 факультета ВТ
2. Тема работы Программные средства обработки событий на основе механизма Веб-хук
Тема утверждена приказом ПГУ № _____ от «__» _____ 2026 г.
3. Руководитель работы к.т.н доцент кафедры «МО и ПЭВМ» Гурьянов Л.В.
4. Задание на работу (назначение разработки, исходные данные)

1) Назначение: система обработки и отправки веб-хуков для событий, возникающих в чатах для общения менеджеров и клиентов.

2) Среда выполнения: серверная инфраструктура, web.

Языковые средства разработки: Golang, Typescript.

Инструментальные средства: Goland, Webstorm, draw.io.

Технологии разработки: ООП.

3) Основные функции приложения:

а) просмотр ранее созданных веб-хуков

б) создание веб-хука

в) редактирование веб-хука

г) деактивация

д) проверка работы

е) механизм повторной отправки

ж) настройка подписи/секрета для валидации на стороне получателя

5. Перечень подлежащих разработке вопросов:

- 1) Анализ предметной области и постановка задачи;
- 2) Проектирование программного обеспечения;
- 3) Реализация;
- 4) Контроль качества программного продукта;
- 5) Планирование и расчет бюджета;
- 6) Оформление пояснительной записки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	10%	12.04.2026 – 20.04.2026	
2	Проектирование программного обеспечения	25%	20.04.2026 – 12.05.2026	
3	Реализация	30%	12.05.2026 – 30.05.2026	
4	Контроль качества программного продукта	10%	30.05.2026 – 02.06.2026	
5	Планирование и расчет бюджета	5%	02.06.2026 – 04.06.2026	
6	Оформление пояснительной записки	20%	04.06.2026 – 20.06.2026	

Дата выдачи задания: «__» _____ 2026 г.

Руководитель бакалаврской работы _____

(подпись, дата)

Задание принял к исполнению студент _____

(подпись, дата)

Работу к защите допустить

Декан факультета _____

(подпись, дата)

Реферат

Пояснительная записка содержит 80 листов, 25 рисунков, 8 таблиц, 20 использованных источников, 5 приложений.

МИКРОСЕРВИСЫ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, ИНФОРМАЦИОННАЯ СИСТЕМА, МОНИТОРИНГ, РАСПРЕДЕЛЕННАЯ СИСТЕМА, REST API, GO, POSTGRESQL, RABBITMQ, POSTMAN, PROMETHEUS, GRAFANA, UML

Объектом разработки является серверный компонент интеграционной платформы, принимающий события из чат-канала и обеспечивающий их передачу во внешние системы и обработку обратной связи.

Целью работы является разработка программных средств для регистрации, отправки и контроля доставки веб-хуков, возникающих при обработке событий корпоративного чат-виджета.

Средства разработки: язык программирования Go, среды разработки JetBrains GoLand, JetBrains DataGrip, СУБД PostgreSQL, брокер сообщений RabbitMQ.

Среда исполнения – семейство ОС UNIX.

Результаты работы: выполнен анализ предметной области обработки событий в чат-системах, рассмотрены аналогичные решения Intercom, Zendesk Chat и Crisp, сформулированы функциональные и нефункциональные требования к системе обработки веб-хуков, спроектирована архитектура серверной и клиентской частей, описана структура хранения веб-хуков и истории отправок, реализованы основные сценарии создания, редактирования, деактивации, тестовой отправки и повторной доставки веб-хуков, проведено тестирование разработанного программного продукта.

					ПГУ 09.03.04 – 08ВР221.5 ПЗ							
Изм.	Лист	№ докум.	Подп.	Дата	Программные средства обработки событий на основе механизма Веб-хук. Пояснительная записка.			Лит.		Лист	Листов	
Разраб.		Градсков А.В.								4	80	
Пров.		Гурьянов Л.В.										
Реценз.												
Н. контр.		Такташкин Д.В.										
Утвер		Козлов А.Ю.						Группа 22ВП1				

Содержание

Введение.....	7
1 Анализ решений со схожими возможностями.....	9
1.1 Анализ предметной области и постановка задачи	9
1.2 Цели и задачи сравнительного анализа	10
1.3 Анализ аналогичных систем	10
1.3.1 Чат виджет «Intercom».....	12
1.3.2 Чат виджет «Zendesk Chat»	15
1.3.3 Чат виджет «Crisp».....	17
1.4 Сравнительный анализ аналогов	19
1.5 Анализ функциональных требований.....	23
1.6 Анализ нефункциональных требований	27
2 Проектирование системы отправки вебхуков.....	30
2.1 Архитектура системы	30
2.2 Проектирование структуры базы данных	32
2.3 Проектирование программных средств.....	32
2.4 Проектирование пользовательского интерфейса	33
3 Реализация системы.....	37
3.1 Реализация серверной части	37
3.2 Реализация клиентской части	39
4 Контроль качества программного продукта	42
4.1 Тестирование приложения черным ящиком	42
4.1.1 Классы эквивалентности	42

4.1.2 Граничные условия	43
4.1.3 Тест кейсы.....	43
4.2 Функциональное тестирование белым ящиком.....	44
4.3 Планирование и расчет бюджета разрабатываемого приложения	46
Заключение	50
Список использованных источников	51
Приложение А (справочное) Глоссарий.....	53
Приложение Б (обязательное) Код серверной части.....	55
Приложение В (обязательное) Код клиентской части	64
Приложение Г (рекомендуемое) Результаты тестирования	73
Приложение Д (необязательное) Справка о внедрении.....	79

Введение

В сервисах деловых поездок чат используется для оперативных вопросов: изменения маршрута, лимитов, бронирования, статуса заявки.

В корпоративных сервисах деловых поездок чат-виджет служит точкой взаимодействия между пользователем, операторами поддержки и информационными системами компании. Через него могут обрабатываться запросы, связанные с бронированием поездок, согласованием командировок и проверкой корпоративных ограничений. Для выполнения таких сценариев требуется передача событий между чатом и внешними системами, что делает необходимым использование специализированного механизма обмена сообщениями.

Для передачи событий из чат-виджета во внешние системы используется механизм веб-хук. При возникновении события, например создании нового диалога или поступлении первого сообщения пользователя, система формирует HTTP-запрос и направляет его на заранее зарегистрированный адрес обработчика.

В рассматриваемом проекте веб-хуки используются как механизм передачи событий между чат-виджетом и внешними сервисами. Через них осуществляется уведомление интеграционных компонентов о появлении новых сообщений, создании диалогов и изменении состояния обращений.

Целью данной работы – разработка системы для обработки событий и отправки их в виде вебхуков.

Практическая значимость связана с первым сообщением в обращении: в нем пользователь формулирует проблему. Если событие сразу уходит через веб-хук, бот или оператор получают контекст без ожидания. Система должна сохранять событие, фиксировать доставку и повторять отправку при ошибке.

В рассматриваемой системе события передаются непосредственно в момент их возникновения. Это позволяет начать обработку первого

сообщения пользователя сразу после его получения, не дожидаясь очередного обращения внешней системы к API чат-виджета.

Такая организация обмена особенно эффективна в ситуациях, когда важно не пропустить первое сообщение в новом обращении. Именно в этот момент можно автоматически определить тематику запроса, выбрать маршрут обработки, запустить сценарий бота или подготовить информацию для оператора.

Дополнительной сложностью является необходимость различать пользовательские и системные сообщения, корректно обрабатывать повторные доставки, учитывать временную недоступность внешних систем и не создавать циклические вызовы. Поэтому описание программных средств обработки событий должно охватывать не только сетевой обмен, но и вопросы хранения технического состояния, трассировки, идемпотентности и административного управления интеграциями.

В данной работе программные средства обработки событий рассматриваются как набор компонентов, обеспечивающих работу с веб-хук-событием: от приема и проверки запроса до передачи данных внешним системам и появлении результата в чат-виджете.

1 Анализ решений со схожими возможностями

1.1 Анализ предметной области и постановка задачи

В документации популярных платформ клиентской поддержки, таких как Intercom, Zendesk и Crisp, веб-хуки рассматриваются как один из способов передачи событий во внешние системы. Для разрабатываемой системы этот механизм важен потому, что позволяет передавать сведения о новых сообщениях и изменениях обращений без постоянного опроса API.

Вебхук (Webhook) – это механизм, с помощью которого один веб-сервис автоматически отправляет HTTP-запрос вида POST на URL другого сервиса при возникновении определенного события. Термин был введен в 2007 году на основе термина hook, применяемого в программировании [1].

Вебхуки широко используются, в том числе в платежных системах, мессенджерах, аналитике и непрерывной интеграции (CI/CD). Например, GitHub или SourceCraft может отправлять вебхук на ваш сервер при пуше в репозиторий, чтобы автоматически запустить тестирование или развертывание новой версии приложения.

Интеграция систем через события и сообщения позволяет значительно снизить связанность компонентов и повысить гибкость архитектуры распределённых приложений [2].

При обработке менеджерами обращений клиентов важна скорость обработки. Если внешняя система узнает о событии с задержкой, это ухудшит клиентский опыт и увеличит нагрузку на менеджеров. В таком случае менеджеру придется самостоятельно искать данные клиента.

Периодический опроса приведет к нагрузке сети и усложнит масштабирование, поэтому событийная модель на базе веб-хуков оказывается более оправданной.

Для рассматриваемого чат-виджета актуальность задачи связана с тем, что событие нового сообщения должно быть быстро передано внешнему

обработчику, а затем внешняя система сама отправит необходимые данные в чат. Поэтому система должна заниматься не только обработкой и отправкой событий, но и хранить историю доставок, поддерживать повторные попытки и позволять администратору управлять адресами, на которые нужно отправлять.

1.2 Цели и задачи сравнительного анализа

Цель сравнительного анализа систем-аналогов – выявление характерных различий между сервисами, предоставляющими обработку событий на основе механизма вебхук. Это необходимо для разработки наиболее актуального для пользователей и высококачественного программного продукта.

Для достижения данной цели существует определённый круг задач, которые нужно решить:

- составление списка критериев, необходимых для сравнения приложений, направленных на наиболее полный анализ в соответствии с характерными чертами сервисов;
- поиск программных продуктов, подходящих для сравнительного анализа;
- обзор приложений-аналогов, краткое описание их характеристик по утверждённым критериям;
- получение итоговых результатов сравнительного анализа информационных систем в наглядном представлении.

Выполнив сравнительный анализ систем, можно наиболее качественно решить задачи и достичь поставленной цели

1.3 Анализ аналогичных систем

В процессе изучения аналогов для сравнительного анализа были выбраны следующие платформы для обслуживания клиентов: «Intercom», «Zendesk» и «Crisp».

Intercom – комплексная платформа обслуживания клиентов, обеспечивающая бесперебойную коммуникацию между компаниями и их клиентами. Она улучшает взаимодействие с клиентами через множество каналов связи, включая чат в реальном времени, чат-ботов и автоматизированный обмен сообщениями, используя инструмент автоматизации.

Zendesk Chat – это инструмент для онлайн-поддержки клиентов, который позволяет бизнесу общаться с посетителями сайта или приложения в режиме реального времени через чат. Он является частью комплексной платформы Zendesk Suite и предназначен для управления взаимодействием с клиентами, автоматизации поддержки и анализа данных.

Crisp – это плоская организация, стремящаяся предложить более дешевую, быструю и простую в использовании платформу обслуживания клиентов. Он был разработан как недорогая альтернатива ведущим поставщикам услуг по обслуживанию клиентов, включая Intercom.

Можно выделить три базовых подхода к интеграции с внешними системами: периодический опрос программного интерфейса (polling), прямой синхронный вызов внешнего сервиса при происхождении события и обработка событий через вебхуки.

Первый подход самый простой, но не обеспечивает минимальную задержку и создает избыточную нагрузку. Второй дает быстрый ответ, однако жестко систему с внешними системами и немасштабируемый. Третий подход, основанный на вебхуках, обеспечивает наилучшую масштабируемость, поскольку позволяет отделить факт возникновения события от его последующей обработки.

Сравнение реализации вебхуков в популярных чат-платформах - Intercom, Zendesk Chat и Crisp показывает различие в подходах к событийно-ориентированной интеграции. Intercom предоставляет развитую инфраструктуру вебхуков с широкими возможностями кастомизации через

JavaScript SDK и API для создания собственных приложений в формате Canvas, что делает его лидером по гибкости интеграций. Zendesk Chat использует модель Switchboard – программный маршрутизатор, который определяет, какая система (бот, агент или внешняя интеграция) в данный момент управляет диалогом.

Вебхуки в Zendesk тесно интегрированы с триггерами, которые запускаются при наступлении определенных событий и могут выполнять операции над тикетами. Crisp предлагает двухуровневую систему: Website Hooks для простых сценариев (быстрая настройка, но без повторных попыток доставки) и Plugin Hooks для production-интеграций (с подписью запросов, ретрай-механизмами и поддержкой расширенного набора событий).

В современных системах поддержки клиентов механизм вебхуков обычно дополняется очередями сообщений, политиками повторной отправки (как в Crisp Plugin Hooks с автоматическими повторными попытками при неудачной доставке), проверкой подписи запроса (HMAC-SHA256 в Crisp) и журналированием. Это позволяет использовать асинхронную обработку и ограничивать влияние временных отказов внешних систем на работу пользовательского интерфейса.

По результатам проведенного сравнения Crisp получает преимущество по критериям стоимости и доступности базовых интеграционных возможностей. Intercom целесообразен для проектов, где важна развитая кастомизация, а Zendesk – для сценариев со сложной маршрутизацией обращений и развитой системой поддержки.

1.3.1 Чат виджет «Intercom»

В рамках данной работы Intercom рассматривается как пример чат-платформы, поддерживающей событийную интеграцию через веб-хуки.

Для сравнения с разрабатываемым решением здесь имеют значение не все возможности Intercom как продукта, а прежде всего те механизмы, которые

связаны с обработкой событий. Важно рассмотреть, какие события может отправлять платформа, как настраиваются подписки на них, в каком формате передается полезная нагрузка и какие средства предусмотрены для контроля доставки. Именно эти характеристики позволяют оценить, насколько гибко сервис может встраиваться в существующую инфраструктуру и насколько надежно он передает данные между системами.

Сервис активно применяется в SaaS-продуктах, интернет-магазинах, банковских системах и корпоративных сервисах [3].

Интерфейс настройки веб-хуков представлен на рисунке 1.

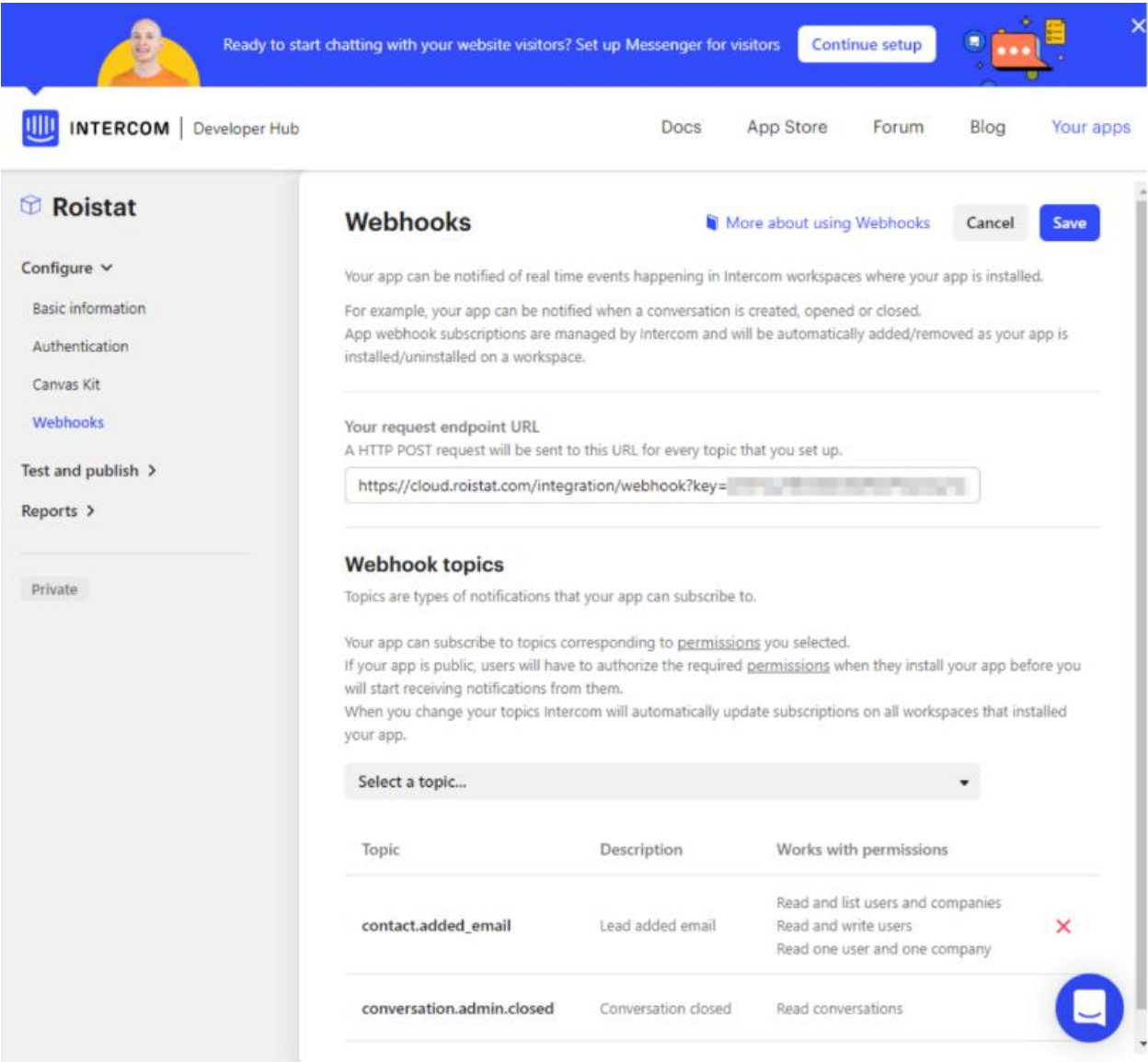


Рисунок 1 – Интерфейс настройки веб-хуков в Intercom

Одним из ключевых компонентов архитектуры Intercom является событийная модель взаимодействия, основанная на использовании веб-хуков. Платформа позволяет внешним системам подписываться на различные типы событий, происходящих внутри чат-системы.

К основным событиям относятся:

- создание нового диалога;
- получение нового сообщения;
- изменение статуса обращения;
- назначение оператора;
- обновление пользовательского профиля;
- завершение диалога;
- срабатывание автоматических сценариев.

При возникновении события сервер Intercom формирует HTTP POST-запрос и отправляет его на заранее настроенный URL обработчика. Данные передаются в формате JSON и содержат:

- идентификатор события;
- тип события;
- временную метку;
- сведения о пользователе;
- содержимое сообщения;
- служебную информацию.

Важной особенностью платформы является развитый механизм надежности доставки:

- повторная отправка webhook-запросов при временных ошибках;
- использование очередей событий;
- логирование ошибок доставки;
- контроль времени ответа обработчика.

Для обеспечения безопасности применяются:

- https-соединения;
- hmac-подписи запросов;
- проверка секретного ключа;
- ограничение источников запросов.

Intercom предоставляет достаточно гибкую систему фильтрации webhook-событий. Администратор может выбирать только интересующие типы событий, что позволяет уменьшить нагрузку на принимающий сервер.

Для проектируемой системы опыт Intercom полезен прежде всего наличием подписок на отдельные типы событий, журналированием и проверкой подлинности запросов. В то же время использование готовой платформы не закрывает задачу разработки собственного контура обработки событий, поскольку требуется управление веб-хуками внутри существующего чат-виджета и учет внутренних правил доставки.

1.3.2 Чат виджет «Zendesk Chat»

Zendesk представляет собой комплексную платформу поддержки пользователей, включающую чат-систему, систему тикетов и инструменты автоматизации клиентского сервиса [4].

Интерфейс настройки веб-хуков в Zendesk Chat представлен на рисунке 2.

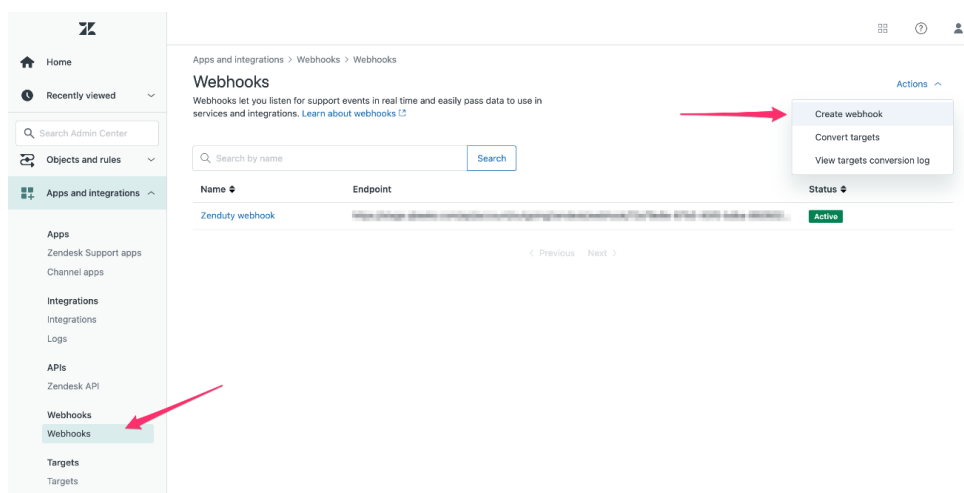


Рисунок 2 – Интерфейс настройки веб-хуков в Zendesk Chat

Zendesk активно использует веб-хуки для организации интеграции с внешними сервисами и синхронизации событий в режиме реального времени.

Платформа поддерживает webhook-события при:

- начале новой чат-сессии;
- отправке сообщений;
- изменении состояния диалога;
- переводе обращения между операторами;
- изменении статуса пользователя;
- закрытии обращения.

Архитектура Zendesk предполагает централизованную обработку событий через API-шлюз. После возникновения события система формирует POST-запрос и отправляет его внешнему обработчику.

Особенностью Zendesk является тесная интеграция webhook-механизма с внутренней системой автоматизации. Например, поступление нового сообщения может автоматически запускать:

- создание тикета;
- обновление CRM;
- уведомление сотрудника;
- запуск бизнес-процесса;
- подключение чат-бота.

Для защиты webhook-запросов используются:

- oauth-аутентификация;
- https;
- токены безопасности;
- контроль IP-адресов.

Преимуществом Zendesk является развитая экосистема интеграций и высокая отказоустойчивость. Недостатком можно считать относительно сложную настройку webhook-сценариев и ограниченную гибкость

пользовательской логики без использования дополнительных middleware-сервисов.

1.3.3 Чат виджет «Crisp»

Crisp представляет собой современную облачную платформу онлайн-коммуникации, предназначенную для организации поддержки пользователей. Ключевой особенностью платформы является развитая событийная архитектура, основанная на использовании веб-хуков, REST API и RTM API (Real-Time Messaging API) [5].

Интерфейс системы Crisp представлен на рисунке 3.

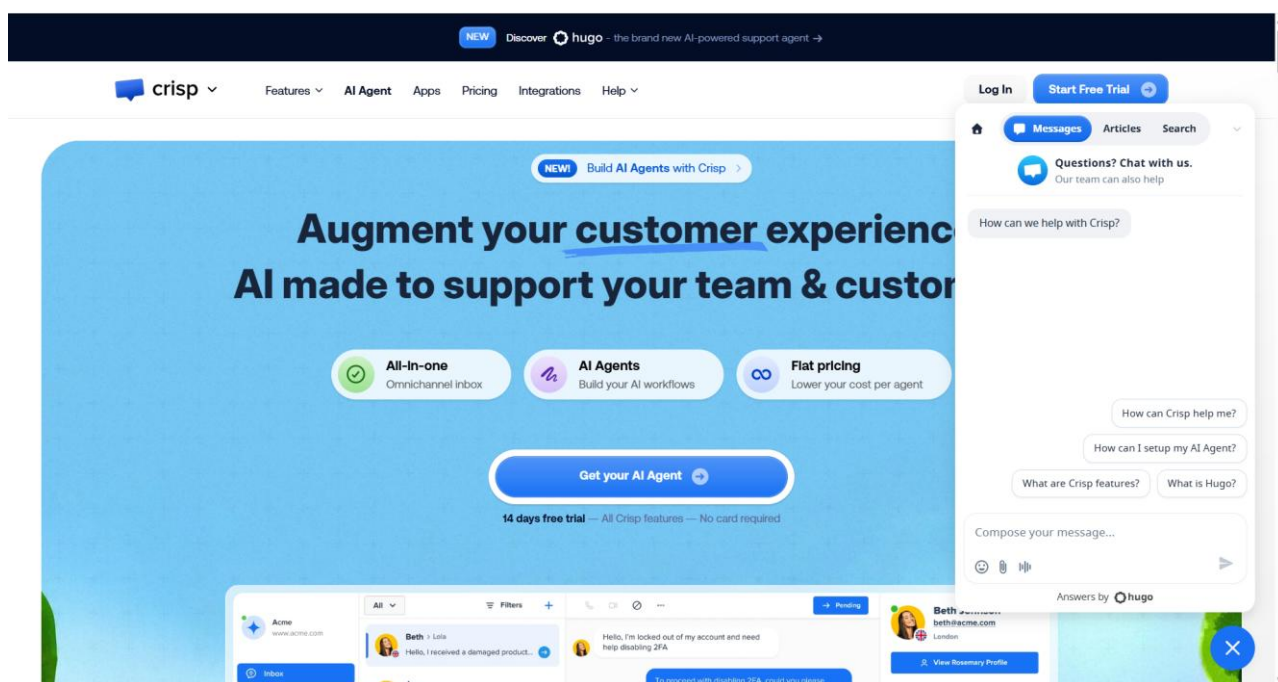


Рисунок 3 – Интерфейс Crisp

В отличие от многих чат-платформ, Crisp изначально ориентирован не только на операторскую поддержку, но и на разработчиков, которым требуется гибкая интеграция с внешними информационными системами, CRM, аналитикой и корпоративными сервисами.

Система поддерживает два типа webhook-интеграций:

- website hooks – веб-хуки для отдельного сайта или рабочего пространства;

- plugin hooks – веб-хуки для интеграционных модулей и плагинов Marketplace.

Plugin Hooks считаются более надежным и масштабируемым решением, поскольку поддерживают:

- повторную отправку событий;
- расширенную аутентификацию;
- большее количество событий;
- интеграцию с несколькими рабочими пространствами одновременно.

Одной из сильных сторон Crisp является комбинированное использование нескольких механизмов интеграции. Помимо webhook-системы платформа предоставляет:

- REST API;
- RTM API на основе WebSocket-соединений.

REST API используется для:

- отправки сообщений;
- обновления пользовательских данных;
- управления чатами;
- изменения состояния диалогов.

RTM API предназначен для:

- получения событий в режиме реального времени;
- обработки асинхронных уведомлений;
- организации постоянного соединения между сервисами.

В типовой архитектуре:

- REST API выполняет действия;
- Web Hooks уведомляют о событиях;
- RTM API обеспечивает потоковую синхронизацию.

Такой подход делает платформу удобной для построения распределенных событийно-ориентированных систем.

Основными преимуществами Crisp являются:

- развитая webhook-инфраструктура;
- поддержка событий в реальном времени;
- сочетание Web Hooks, REST API и RTM API;
- удобство интеграции;
- хорошая документация для разработчиков;
- поддержка автоматизации и пользовательских событий;
- высокая скорость доставки уведомлений.

Система также имеет ряд минусов:

- часть функций только в платном тарифе;
- сложность настройки интеграций;
- неполная гарантия доставки в Website Hooks;
- нужно реализовывать свою систему обработки.

Кроме того, при высокой нагрузке требуется отдельная инфраструктура обработки webhook-событий:

- очереди сообщений;
- балансировка нагрузки;
- механизмы дедупликации.

1.4 Сравнительный анализ аналогов

Подход сравнительного анализа – это метод, применяемый в различных областях, таких как экономика, социология и бизнес, для сопоставления различных объектов или явлений с целью выявления их различий и схожестей. Этот подход позволяет углубить понимание исследуемых предметов, прояснить их особенности и оценить эффективность на основе полученных данных [6].

Для проведения сравнительного анализа существующих на рынке приложений было выявлено несколько критериев для их оценки, а именно: поддерживаемые типы событий, скорость доставки событий, надежность доставки, безопасность, возможности фильтрации событий, средства мониторинга и логирования, высокая скорость и стоимость использования, локализация и адаптация для рынка России.

Поддерживаемые типы событий означает, какие виды событий система умеет обрабатывать и отправлять получателям. Речь про такие события, как новое сообщение в чате, создание, закрытие или откладывание чата, изменение статуса обращения, действия менеджеров или клиентов, обновление статусов менеджеров. Большое количество разных событий позволяет гибко работать с чатами во внешних системах. Например, нужно уведомить внешнюю систему о переоткрытии чата.

Скорость обработки событий показывает, как быстро происходит обработка и доставка до внешнего обработчика. При работе менеджера с обращением клиента это важно, потому что задержки напрямую влияют на скорость реакции ботов и систем. При задержке меньше 1 секунды работа считается приемлемой, так как менеджер только начинает знакомиться с чатом.

Надёжность доставки описывает, насколько система гарантирует доставку вебхуков при возникновении ошибок сети или недоступности принимающего сервиса. Сюда относятся механизмы повторных отправок, очереди сообщений, хранение событий и защита от потерь или дублирования данных.

Безопасность связана с тем, насколько защищена передача вебхуков. Для этого применяются различные механизмы защиты, такие как подпись данных, использование ключа. Так принимающая сторона может убедиться в достоверности данных. Это важный критерий так как через вебхук могут передаваться данные клиента.

Возможности фильтрации событий показывает, можно ли настроить получение только нужных типов событий. Это поможет снизить нагрузку на принимающий сервер, уменьшить трафик и обрабатывать нужные клиенту уведомления.

Средства мониторинга и логирования включает инструменты для отслеживания истории, ошибок отправки. К средствам мониторинга относятся логи, просмотр содержимого запросов, уведомления об ошибках и возможность повторной отправки событий. Это поможет расследовать инциденты при сбоях.

Стоимость использования отражает, насколько платформа доступна по цене. Сюда входят бесплатные тарифы, ограничения по количеству событий, стоимость расширенных функций и лимиты использования вебхуков.

Локализация и адаптация для рынка России означает, насколько удобно использовать систему при работе с сегментом клиентов в России. Важны русскоязычный интерфейс и документация, поддержка российских сервисов и отсутствие ограничений, которые мешают работе платформы в России. В условиях санкций это особенно важно, так как не придется переносить клиентов из одной системы в другую с риском потери данных.

Для каждого критерия определяем методику оценки выполнения критерия. Возможные результаты: 1, если критерий выполняется и 0, если не выполняется.

Двухбалльная система оценивания позволяет однозначно оценить, соответствует ли система заданному критерию. Проведенный анализ поможет выявить все достоинства и недостатки сравниваемых систем для дальнейшего анализа функциональных и нефункциональных требований. Это особенно важный шаг, так как от указанных ранее требований зависит архитектура, востребованность и другие аспекты системы.

Для оценивания систем по критериям будем использовать двухбалльную систему:

- 0 баллов – «не соответствует критерию»;
- 1 балл – «соответствует критерию».

Результаты сравнительного анализа представлены в таблице 1.

Таблица 1 – Сравнительный анализ программных средств

Критерии	Intercom	Zendesk	Crisp
Поддерживаемые типы событий	1	1	1
Скорость доставки событий	1	1	1
Надежность доставки	1	1	1
Безопасность	1	1	1
Возможности фильтрации событий	1	0	1
Средства мониторинга и логирования	1	1	0
Стоимость использования	0	0	1
Локализация и адаптация для рынка России	0	0	1

Проведённый сравнительный анализ показывает, что большинство современных платформ для обработки вебхук событий уже имеют достаточно развитую событийную архитектуру и предоставляют широкий набор инструментов для интеграции с внешними сервисами.

Intercom и Zendesk можно использовать в продуктовых системах. В целом это надёжные решения: вебхуки у них доставляются стабильно, безопасность хорошо продумана, мониторинг тоже реализован удобно. Но есть и минусы. В первую очередь это высокая цена, а также сложности при работе на российском рынке.

Если сравнивать рассмотренные платформы, то Crisp выглядит наиболее близким аналогом. Он хорошо показывает себя по нескольким важным критериям: стоимости, фильтрации событий и удобству для российских пользователей. Хотя здесь важно отметить, что такая оценка всё равно достаточно общая. Для более точного сравнения нужны были бы конкретные

числа, например по задержкам доставки вебхуков или по проценту успешных отправок.

В итоге можно сказать, что готовые решения уже закрывают большую часть задач, связанных с вебхук-интеграциями. Но не всегда они подходят под конкретные условия. Некоторые платформы слишком дорогие и сложные, другие хуже адаптированы под локальный рынок или отдельные сценарии работы. Поэтому разработка собственной системы обработки вебхуков остаётся обоснованной, особенно если учитывать требования к производительности, безопасности, масштабируемости и удобной интеграции.

1.5 Анализ функциональных требований

Вариант использования фиксирует соглашение между участниками системы о ее поведении. Вариант использования описывает поведение системы при ее ответах на запрос одного из участников, называемого основным действующим лицом, в различных условиях.

Основное действующее лицо инициирует взаимодействие с системой, чтобы добиться некоторой цели. Система отвечает, соблюдая интересы всех участников. Различные модели поведения, или сценарии, развертываются в зависимости от определенных запросов и условий, при которых делались эти запросы. Вариант использования собирает вместе эти сценарии [7].

К числу функциональных требований относятся: разрабатываемая система обеспечивает прием и обработку событий, которые инициируют пользователи, а затем отправки их в виде вебхуков в сторонние системы. Отправка осуществляется по адресам, заданным в базе данных.

В процессе обработки выполняется определение типа события и проверка принадлежности сообщения к новому обращению пользователя. Для последующей обработки и трассировки сохраняются идентификаторы диалога, канала связи, клиента и сообщения.

После получения события его данные преобразуются к унифицированному формату и передаются во внешнюю информационную систему. Ответ, полученный от внешнего сервиса, автоматически публикуется в соответствующем чате от имени бота.

Для обеспечения надежности взаимодействия реализован механизм повторной доставки событий в случаях временной недоступности принимающей стороны. Кроме того, система ведет журналы доставки и аудита, что позволяет контролировать корректность обмена данными и проводить анализ возникающих ошибок.

Дополнительно предусмотрены средства управления конфигурацией интеграций, обеспечивается логическая изоляция данных различных арендаторов, а также поддерживается возможность временного отключения отдельных подписок на веб-хуки.

Для разработчиков предоставляются инструменты мониторинга и диагностики, предназначенные для оперативного выявления и устранения неисправностей.

Администратор сервиса может отслеживать статусы доставки вебхуков путем просмотра их в панели управления. Такими правами обладает только суперадминистратор сервиса. Также у него будут возможность создания, редактирования, деактивации, настройки подписи или секрета и тестовой отправки вебхука.

У сервиса, который инициирует события должна быть возможность передавать события в обработчик.

На рисунке 4 изображена диаграмма вариантов использования, составленная в соответствии с функциональными требованиями к сервису обработки веб-хук-событий.

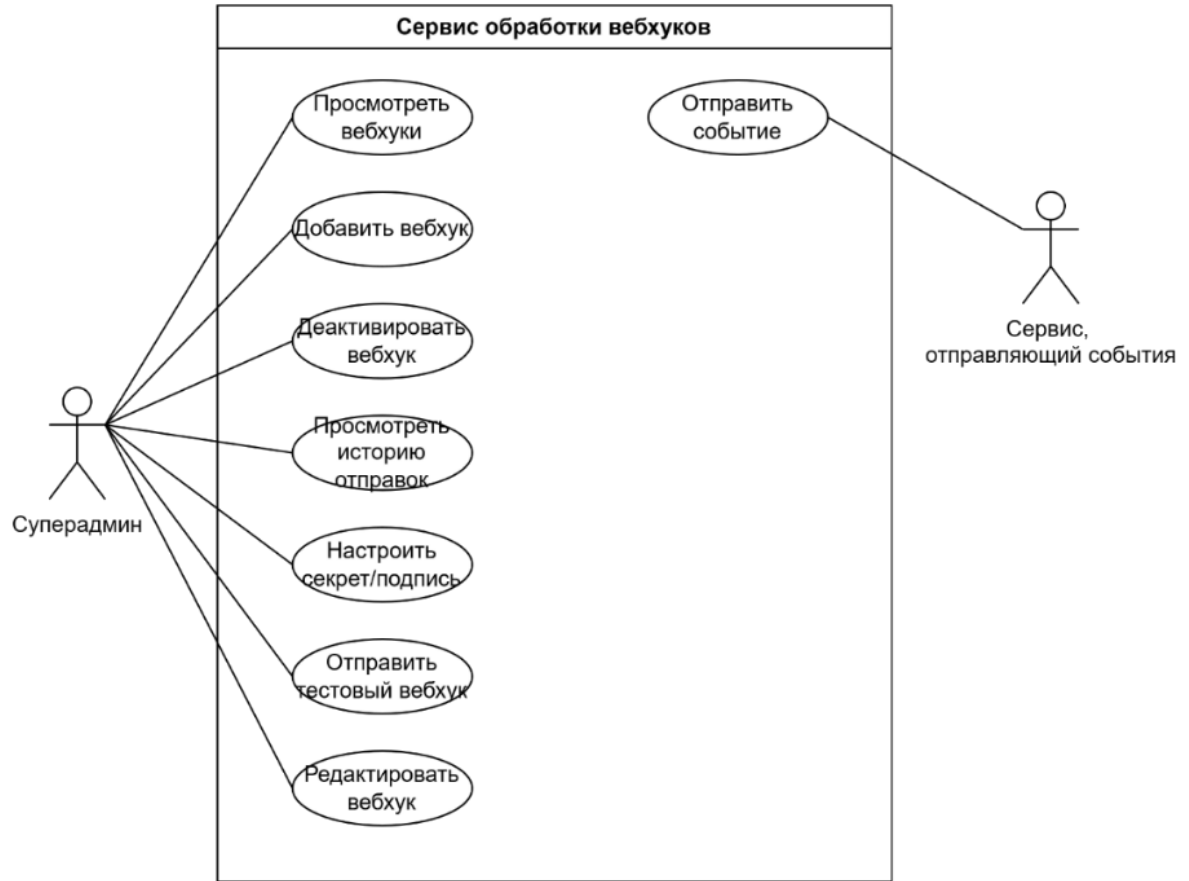


Рисунок 4 – Диаграмма вариантов использования

В таблице 2 представлена спецификация для прецендента «Просмотреть вебхуки».

Таблица 2 – Спецификация прецендента «Просмотреть вебхуки»

Прецендент: Просмотреть вебхуки
ID: 1
Описание: Пользователь просматривает ранее настроенные вебхуки
Главные актеры: Суперадмин
Второстепенные актеры: Отсутствуют
Предусловия: 1. Пользователь открыл web приложение 2. Пользователь является суперадмином 3. Пользователь перешел на вкладку просмотра вебхуков
Основной поток: 1. Пользователь просматривает настроенные вебхуки
Постусловия:
Альтернативные потоки: Отсутствуют

В таблице 3 представлена спецификация для прецендента «Добавить вебхук».

Таблица 3 – Спецификация прецендента «Добавить вебхук»

Прецендент: Добавить вебхук
ID: 2
Описание: Пользователь настраивает новый вебхук
Главные актеры: Суперадмин
Второстепенные актеры: Отсутствуют
Предусловия: <ol style="list-style-type: none"> 1. Пользователь открыл web приложение 2. Пользователь является суперадмином 3. Пользователь перешел на вкладку просмотра вебхуков 4. Пользователь нажал на кнопку «Добавить вебхук»
Основной поток: <ol style="list-style-type: none"> 1. Пользователь вводит url, куда нужно отправлять вебхуки 2. Пользователь выбирает типы событий для отправки 3. Пользователь нажал на клавишу «Сохранить»
Постусловия: <ol style="list-style-type: none"> 1. Система отправляет указанные события на указанный адрес
Альтернативные потоки: Отсутствуют

В таблице 4 представлена спецификация для прецендента «Редактировать вебхук».

Таблица 4 – Спецификация прецендента «Редактировать вебхук»

Прецендент: Редактировать вебхук
ID: 3
Описание: Пользователь редактирует существующий вебхук
Главные актеры: Суперадмин
Второстепенные актеры: Отсутствуют
Предусловия: <ol style="list-style-type: none"> 5. Пользователь открыл web приложение 6. Пользователь является суперадмином 7. Пользователь перешел на вкладку просмотра вебхуков 8. Пользователь нажал на кнопку «Редактировать» у необходимого вебхука.
Основной поток: <ol style="list-style-type: none"> 4. Пользователь редактирует url, куда нужно отправлять вебхуки 5. Пользователь редактирует типы событий для отправки 6. Пользователь нажал на клавишу «Сохранить»
Постусловия: <ol style="list-style-type: none"> 2. Система отправляет вебхуки по обновленным данным

Таким образом были составлены функциональные требования, которым система должна соответствовать.

1.6 Анализ нефункциональных требований

Нефункциональные требования (НФТ) описывают, как должен работать программный продукт и какими свойствами или характеристиками обладать, чтобы доставить ту ценность, которую несёт система, с учетом условий ее существования. Такие требования вносят вклад в инфраструктуру, а не в поведение системы.

Часто к ним относятся с пренебрежением, ведь их влияние на осуществление пользовательских требований неочевидно. Как показывает практика, именно их несоблюдение напрямую сказывается на отказоустойчивости системы, её безопасности, а также на претензиях со стороны регуляторов [7].

При анализе нефункциональных требований нужно опираться на текущее число клиентов, которые пользуются системой. Две основных категории пользователей: менеджеры и клиенты. Клиентов значительно больше, так как один менеджер может вести работу с несколькими чатами клиентов. Однако при работе менеджеров создается намного больше событий, так как действия менеджеров не только связаны напрямую с чатами, но и в целом с системой.

Система считается глобальным продуктом, в будущем планируются интеграции с другими системами. Следовательно число клиентов будет расти.

От работы системы зависит опыт пользования клиентами, а также эффективность работы менеджеров, так как для ускорения обработки запроса от клиента используется вебхук для отправки данных о клиенте в чат.

Исходные данные о системе собирались через средства мониторинга, такие как Prometheus и Grafana. Производился сбор таких данных, как

минимальное и максимальное числа клиентов, количество памяти, которое потребляет система, количество горутин.

Число клиентов будет постоянно расти, так как бизнес заинтересован в расширении клиентской базы. При этом система должна работать с максимальным откликом, так как некоторые запросы клиентов касаются денежных операций, например срочная отмена авиабилета.

Нужно с особым вниманием отнестись к анализу нефункциональных, потому что от этого зависит клиентский опыт, финансовые операции компании и эффективность работы менеджеров.

С точки зрения безопасности обработка вебхуков должна учитывать риск поддельных запросов и утечки служебной информации. Злоумышленники могут подделать запрос и отправить фальшивые данные. Для этого необходимо предусмотреть механизм защиты для отправляемых запросов. Например, подписывать передаваемые данные ключом, который есть только с нашей и на стороне получателя.

Для корректной работы с большим количеством событий необходимо обратить внимание к требованиям по производительности. Менеджер одним действием может инициировать до 30 событий, это означает, что система должна справляться с обработкой большого количества событий в пиковые часы нагрузки.

Масштабируемость системы определяется не только количеством обрабатываемых событий, количеством интегрируемых систем, но и сценариями обработки. Одни события требуют преобразования малых объектов, другие обладают большим набором полей, что сказывается на нагрузке на центральный процессор при обработке.

Надежность системы требует настройки записи ошибок в логи, а также создание оповещений в корпоративный канал для быстрой реакции на возникающие неполадки. Механизм переотправки вебхуков после неудачной

отправки позволит снизить число недоставленных данных. Это особенно критично, так как напрямую влияет на результат работы.

Требования к переносимости определяют способность системы работать в различных средах или на разных платформах, обеспечивая гибкость развёртывания. В данном случае, при переезде на другого поставщика облачных ресурсов, необходимо предусмотреть, чтобы переезд принес минимальное количество работы для развёртывания.

На основе проведенного анализа была поставлена задача по разработке программных средств, обеспечивающих надежность, производительность, безопасность и переносимость. Решение должно быть пригодно для промышленной эксплуатации в среде деловых поездок и сопровождаться средствами контроля качества.

2 Проектирование системы отправки вебхуков

2.1 Архитектура системы

Основной принцип архитектуры программного обеспечения заключается в том, что каждая программная система создается для удовлетворения бизнес-целей организации, и что архитектура системы – это мост между этими (часто абстрактными) бизнес-целями и конечной (конкретной) результирующей системой [8].

Архитектура решения строится по модульному принципу. Входной шлюз принимает веб-хук-запрос, выполняет базовую проверку подписи и схемы данных, после чего передает событие в слой нормализации. Нормализатор приводит полезную нагрузку различных источников к единой внутренней модели, включающей тип события, идентификаторы сущностей, временные метки, полезную нагрузку и технические метаданные.

После формирования событие передается в модуль маршрутизации. На этом этапе система определяет, какие внутренние обработчики или внешние подписчики должны получить информацию о произошедшем действии. Иными словами, событие не отправляется всем компонентам без разбора: оно направляется только тем частям системы, для которых оно действительно имеет значение.

Отдельная логика предусмотрена для ситуации, когда пользователь отправляет первое сообщение. В этом случае система дополнительно анализирует историю обращения. Если диалог был создан впервые или до этого не содержал сообщений от пользователя, событию присваивается специальный признак. Для повторных отправок необходимо реализовать ретрай механизмы.

Схематичное представление многоуровневой архитектуры приведено на рисунке 5.

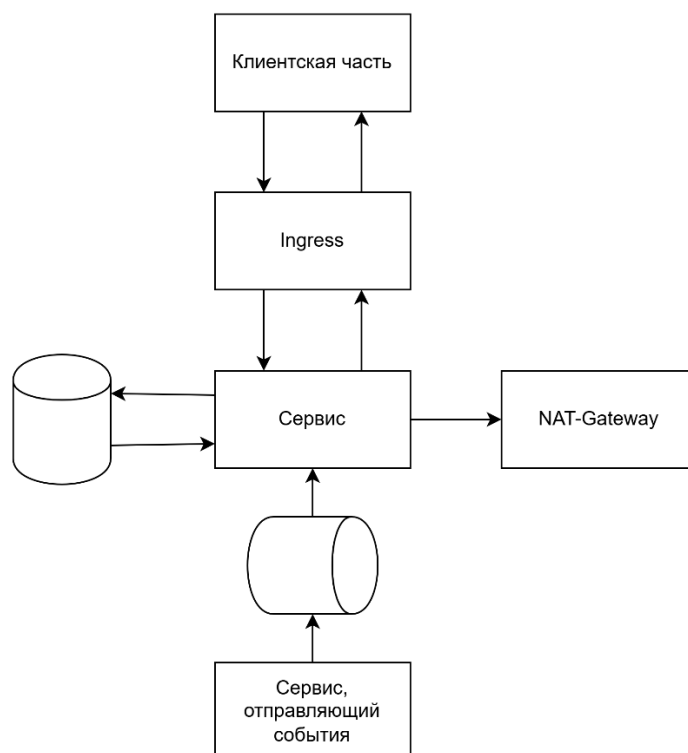


Рисунок 5 – Архитектура сервиса обработки веб-хук-событий

Как представлено на рисунке для реализации системы необходимы не только сервисы, написанные на языке Go и TypeScript, но и внешняя инфраструктура, от настройки которой будет зависеть надежность системы. Для хранения и работы с данными будет использовать СУБД PostgreSQL. В качестве очереди сообщений, куда будут поступать и затем сервис обработки будет получать события, задействуется RabbitMQ. Это позволит реализовать асинхронную работу системы в фоновом режиме.

Для обращений во внешнюю сеть необходим NAT Gateway, который предоставляет Yandex Cloud. NAT-шлюз позволяет предоставлять доступ в интернет облачным ресурсам, не назначая им публичные IP-адреса. Вместо этого они будут выходить в интернет через NAT-шлюз, который получит адрес из отдельного диапазона публичных IP-адресов. Шлюз – региональный ресурс, который присутствует во всех зонах доступности. Управлять шлюзами можно через консоль управления, с помощью CLI, Terraform или API [9].

Для доступа из внешней сети используется ingress, который предоставляет Kubernetes.

Ingress предоставляет доступ к HTTP- и HTTPS-маршрутам за пределами кластера для сервисов внутри кластера. Маршрутизация трафика контролируется правилами, заданными в ресурсе Ingress [10].

2.2 Проектирование структуры базы данных

Единая модель веб-хук-события должна содержать идентификатор события, тип, источник, идентификатор арендатора, идентификаторы диалога и сообщения, признаки первого сообщения и автоматической обработки, а также вложенный объект полезной нагрузки. Помимо деловых данных в модели сохраняются заголовки запроса, хеш-подпись, дата получения и технический статус обработки.

Для хранения конфигурации интеграций требуется сущность подписки, включающая адрес внешней системы, перечень интересующих событий, секрет подписи, режим повторной доставки, статус активности и ограничения по арендатору. Отдельно хранится журнал попыток доставки, где фиксируются попытки отправки, ответы получателя и причины ошибок.

2.3 Проектирование программных средств

Серверная часть включает слой программного интерфейса, валидатор, нормализатор, маршрутизатор, очередь, обработчики деловой логики и модуль обратной доставки ответов бота. Для обеспечения расширяемости обработчики проектируются как независимые компоненты с единым соглашением взаимодействия. Это позволяет добавлять новые типы событий без изменения базового конвейера.

Краткая спецификация компонентов программного продукта представлена на рисунке 6.

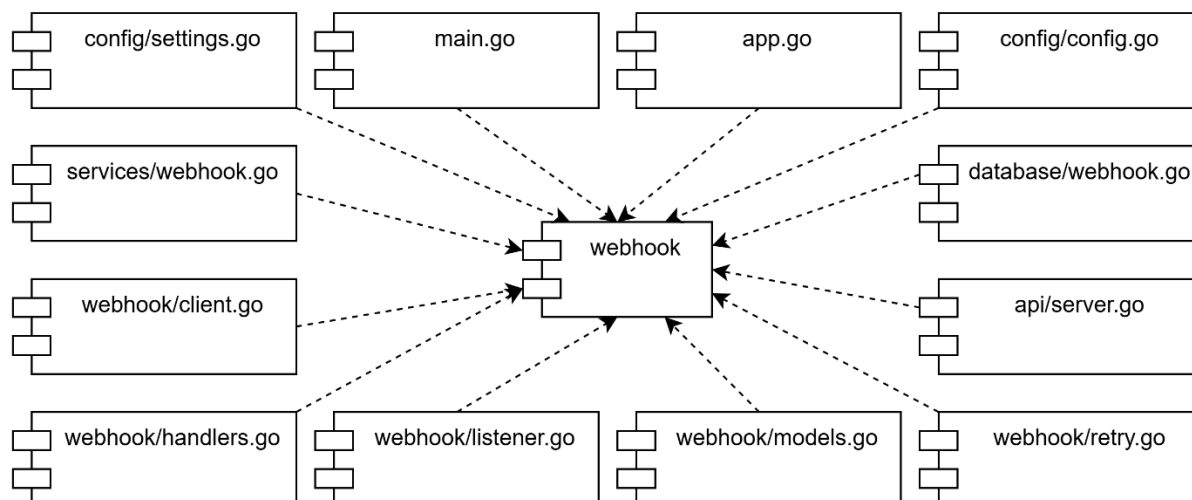


Рисунок 6 – Диаграмма компонентов

Особое внимание уделяется идемпотентности. Идемпотентность операции означает, что повторное выполнение запроса не приводит к изменению результата системы [8]. Один и тот же веб-хук может быть доставлен повторно, поэтому до запуска деловой логики система должна проверить, обрабатывался ли идентификатор события ранее. В случае дубля допускается безопасное повторное подтверждение приема без повторного выполнения критичных действий.

2.4 Проектирование пользовательского интерфейса

Проектирование интерфейсов начинается с формирования требований. Для системного аналитика в задачах проектирования интерфейса важно не только формализовать эти требования, но и связать их с визуальным представлением – прототипами. Этот процесс позволяет согласовать ожидания всех участников проекта и минимизировать риски недопонимания [9].

Исходя из диаграммы вариантов использования, был разработан графический интерфейс приложения. Пользовательский интерфейс должен быть прост и понятен в использовании.

Для проектирования визуальной части сервиса был выбран многофункциональный редактор растровой графики Adobe Photoshop.

Редактор обладает большими возможностями в сфере проектирования и верстки программных продуктов. Широкий выбор инструментов данной программы позволил спроектировать внешний вид клиентской части, еще до ее функциональной реализации, что позволило сократить время разработки.

В данном программном обеспечении работа с вебхуками начинается с экрана просмотра вебхуков, показанным на рисунке 7 с описанием его элементов.

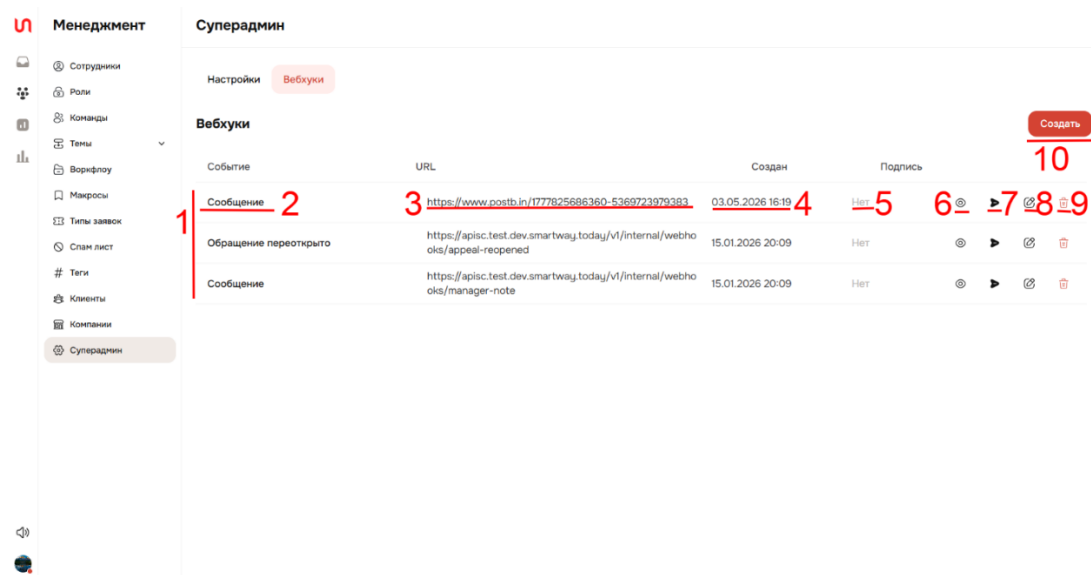


Рисунок 7 – Экран просмотра вебхуков

Основные элементы интерфейса, показанного на рисунке 7:

- 1 – список ранее настроенных вебхуков;
- 2 – поле для отображения типа события;
- 3 – поле для отображения url, куда нужно отправлять вебхуки;
- 4 – поле для отображения даты создания;
- 5 – поле для отображения наличия подписи вебхука;
- 6 – кнопка для просмотра истории отправок вебхука;
- 7 – кнопка для отправки тестового запроса;
- 8 – кнопка для редактирования вебхука;

9 – кнопка для деактивации вебхука;

10 – кнопка для перехода на страницу создания вебхука.

При создании нового вебхука требуется необходимые для отправки данные. Данная информация заносится в соответствующие поля файла типа JSON, что позволяет осуществить реализацию изменений уже созданного вебхука. На рисунке 8 изображен интерфейс создания вебхука.

Новый вебхук

URL

1

События

2 ☒ Сообщение

☐ Обращение создано

☐ Обращение переоткрыто

Секрет подписи

3

Будет использован для HMAC-SHA256 подписи canonical JSON тела

Создать

Рисунок 8 – Окно создания вебхука

Элементы экрана регистрации рисунка 8:

1 – поле url;

2 – селектор типа события;

3 – поле ввода секрета вебхука.

Экран просмотра истории отправок представлен на рисунке 9.

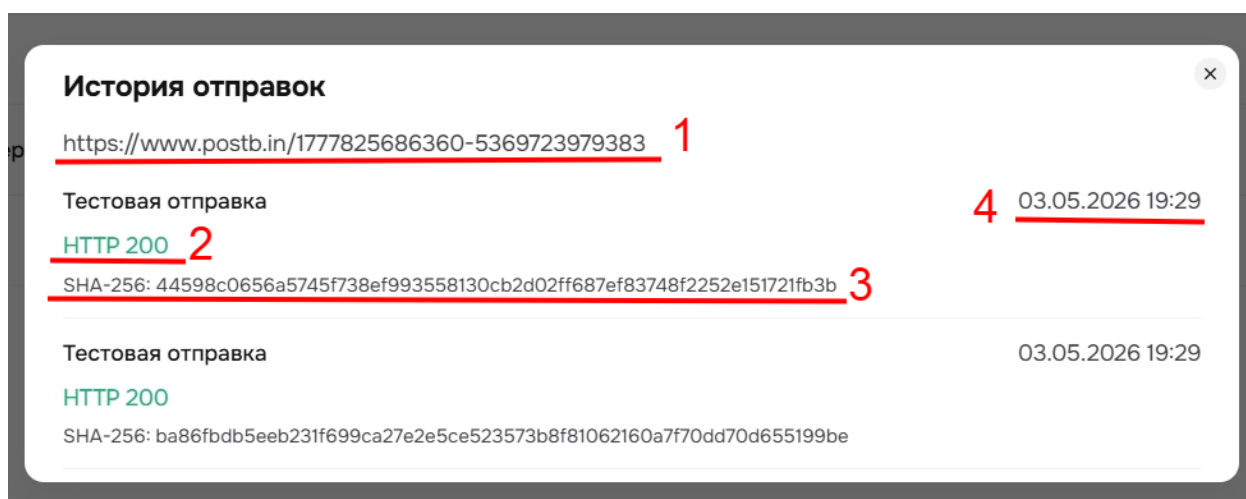


Рисунок 9 – Окно просмотра событий отправки

Основные элементы интерфейса, показанного на рисунке 9:

- 1 – поле для отображения url, куда осуществлялась отправка;
- 2 – поле для отображения кода ответа;
- 3 – поле для отображения хеша полезной нагрузки;
- 4 – поле для отображения даты отправки.

Проектирование графического интерфейса на ранних этапах разработки позволяет продумать цепочки процессов бизнес-логики, используя наглядные примеры. Пользуясь инструментами интегрированной среды разработки, можно быстро составить структуру экранов приложения.

Исходя из вышеописанных моделей проектирования пользовательского интерфейса, можно понять, что графическое представление проекта реализует все функциональные требования.

3 Реализация системы

3.1 Реализация серверной части

Реализация серверной части включает настройку очереди сообщений, подключение отправителей событий в очередь, а также разработку клиента, предназначенного для обработки событий и рассылки вебхуков.

Для работы с обращениями была создана отдельная очередь `appeals`, в которую отправители помещают события, связанные с обращениями, например появление нового сообщения. Очередь функционирует без использования маршрутизирующего ключа и обладает пропускной способностью обработки одного сообщения за единицу времени.

При вызове пользователем функций сервиса посредством REST-запросов и изменении состояния данных, например при создании обращения, сервис формирует и отправляет сообщения в формате JSON. Помимо информации о событии, сообщение содержит метаданные обращения, включая его идентификатор и рабочее пространство, в котором произошло событие. Наличие всех необходимых данных непосредственно в сообщении позволяет избежать дополнительных запросов к базе данных для получения информации об изменённом объекте.

Сервис, отвечающий за обработку и рассылку вебхуков, осуществляет прослушивание очереди сообщений. При поступлении нового сообщения оно принимается в байтовом формате и преобразуется в соответствующую структуру данных. Для каждого типа события реализован отдельный обработчик. После обработки сервис получает список URL-адресов, на которые необходимо отправить уведомление о событии, и выполняет рассылку вебхуков.

В случае неудачной доставки данные о событии помещаются в локальную очередь повторных отправок. Повторная рассылка осуществляется через определённые временные интервалы. Повторная отправка реализована

отдельным фоновым обработчиком на основе горутины. Обработчик периодически выбирает события, для которых наступило время следующей попытки, и повторно отправляет запрос получателю. После пяти неудачных попыток доставки URL-адрес, на который не удалось выполнить отправку, автоматически деактивируется.

Для хранения информации о вебхуках используется база данных PostgreSQL. В рамках реализации были созданы две таблицы: `webhooks` и `webhook_history`. Таблица `webhooks` предназначена для хранения URL-адресов, флага активности, типа события, а также дат создания и обновления записи. Таблица `webhook_history` используется для сохранения истории отправок и содержит идентификатор вебхука, тип события, тело запроса, хеш запроса, подпись запроса, а также тело и статус ответа.

Основной логика по обработке и рассылке находится в файле `listener.go`. Функция `Start` используется для получения событий из RabbitMQ и запускается в отдельной горутине, она обрабатывает ошибки, которые возникают в процессе работы слушателя, и запускает менеджер повторных отправок. При получении события, она передает ее в функцию обработчик `handle`, который занимается дальнейшей обработкой данных.

Логика по обработке очереди на повторную отправку находится в файле `retry.go`. Он включает FIFO-очередь, которая содержит данные для повторной отправки. Очередь обрабатывается в параллельном режиме, обработчик проверяет время, когда должна быть выполнена переотправка.

Обработка заключается в преобразовании слайса байтов в необходимую структуру, определении типа события, получение адресов для отправки, а также передачу в отдельные сервис для отправки по http. После отправки осуществляется запись в базу данных с результатом. Если в процессе отправки возникла ошибка, то функция передает данные в очередь для переотправки.

Основной код функции `Start` отображен на рисунке 10.

```

        if err != nil {
            l.logger.Errorf( format: "rabbit error: %v", err)
            l.consumerMu.RLock()
            if l.consumer != nil {
                l.consumer.Close()
            }
            l.consumerMu.RUnlock()
        }
    }
    case err = <-l.handleErrChan:
        if err != nil { l.logger.Errorf( format: "handle event error: %v", err) }
    case <-ctx.Done():
        l.logger.Debug( message: "webhook listener context cancelled")
        l.consumerMu.RLock()
        if l.consumer != nil {
            l.consumer.Close()
        }
        l.consumerMu.RUnlock()
        return
    case <-done:
        return
    }
}
}()

l.consumerMu.RLock()
runConsumer := l.consumer
l.consumerMu.RUnlock()

if runConsumer == nil { return }

err = runConsumer.Run(l.handler)
if err != nil { l.rabbitErrChan <- err }

```

Рисунок 10 – Код функции Start

Все компоненты, необходимые в работе с очередью и отправкой работают в параллельном режиме, что увеличивает скорость обработки событий. Использование горутинов и других возможностей языка Go позволило при минимальных усилиях реализовать надежную и быструю систему по обработке и отправке вебхуков.

3.2 Реализация клиентской части

Исходя из функциональных требований, в клиентской части были реализованы основные бизнес-процессы, которые позволят пользователю работать с созданием, редактированием и управлением вебхуками.

За логику работы клиента отвечает большое количество классов, наиболее значимым из них является класс SuperAdminWebhooks.tsx. Именно

он выступает как контейнер для отображения списка текущих настроенных вебхуков, редактирования уже существующих. В классе хранятся несколько переменных для отображения состояний: `isLoading`, `isFetching`, `isError`. Именно этот компонент отвечает за объединение всех остальных компонентов. Для отслеживания состояний используется `useToggle()` и `useState()`.

На рисунке 11 представлен исходный код компонента.

```
export const SuperAdminWebhooks: FC = () => {
  const {
    state: isCreateModalOpen,
    setFalse: closeCreateModal,
    setTrue: openCreateModal,
  } = useToggle();
  const {
    state: isHistoryModalOpen,
    setFalse: closeHistoryModal,
    setTrue: openHistoryModal,
  } = useToggle();
  const [editableWebhook, setEditableWebhook] = useState<WebhookDto>();
  const [historyWebhook, setHistoryWebhook] = useState<WebhookDto>();

  const {
    data: webhooks,
    isLoading,
    isFetching,
    isError,
    refetch,
  } = useGetSuperAdminWebhooksQuery();
  const [createWebhook, createWebhookState] = useCreateSuperAdminWebhookMutation();
  const [updateWebhook, updateWebhookState] = useUpdateSuperAdminWebhookMutation();
  const [deleteWebhook] = useDeleteSuperAdminWebhookMutation();
  const [testWebhook] = useTestSuperAdminWebhookMutation();
  const [
    getHistory,
    {
      data: history,
      isLoading: isHistoryLoading,
      isFetching: isHistoryFetching,
      isError: isHistoryError,
    },
  ],
```

Рисунок 11 – Код компонента SuperAdminWebhook

За логику работы с созданием вебхука отвечает `webhook-create-modal.tsx`. Она собирает данные для создания: адрес, типы событий и секрет для подписи. За валидацию входных данных отвечает `zodResolver`. Он проверяет основные ограничения при вводе данных пользователем.

Отображением истории отправок занимается `webhook-history-modal.tsx`. Он выводит историю отправок выбранного пользователем вебхука, полученную с серверной части.

Элемент списка описан в `webhooks-list.tsx`. В нем описаны основные элементы управления и данные конкретного вебхука. Среди элементов управления можно выделить кнопку для отправки тестового запроса на указанный адрес, а также кнопку для редактирования.

Благодаря своевременному проектированию графического интерфейса, удалось снизить трудозатраты на разработку, что значительно упростило процесс работы над клиентской частью. Использование встроенных возможностей TypeScript в виде готовых компонентов, а также стилей, которые используются в других местах проекта позволило значительно облегчить и ускорить разработку клиентской части.

4 Контроль качества программного продукта

4.1 Тестирование приложения черным ящиком

Тестирование – это процесс выполнения программы с целью обнаружения ошибок, при этом невозможно доказать отсутствие ошибок, можно лишь показать их наличие [9].

Особенность тестирования черным ящиком заключается в том, что вместо свойств составных частей системы, изучается ее реакция на изменение условий эксплуатации. Под стратегией данного подхода к тестированию понимается создание тестов для тестовых наборов.

4.1.1 Классы эквивалентности

Классы эквивалентности – это группы входных данных, которые обрабатываются программой одинаково. Суть метода в том, чтобы вместо тестирования всех возможных значений проверить по одному представителю из каждого класса. Это сокращает количество тест-кейсов, сохраняя эффективность проверки [10]. Для разрабатываемого приложения классы эквивалентности представлены в таблице 5.

Таблица 5 – Классы эквивалентности

	Верные классы эквивалентности	Неверные классы эквивалентности
Длина пути	1-2048 (1)	0 (2) 2049-∞ (3)
Структура url	Начинается с https (4)	Начинается не с https (5)
Содержание url	Буквы (6) Цифры (7) Символы (8)	Остальные символы (9)

Данные в таблице 5 необходимы для корректной работы метода проверки url.

4.1.2 Граничные условия

Граничное тестирование использует принцип, при котором один класс эквивалентности переходит в другой. Именно на границе перехода чаще всего возникают ошибки. К граничным значениям относится крайнее значение, начальное-конечное, нижнее-верхнее и максимальное-минимальное. Тип и описание граничные значения для разрабатываемого продукта представлены в таблице 6.

Таблица 6 – Граничные значения

Границы	Описание
Физические	Поле не может быть пустым
Технологические	Ограничения оперативной памяти устройства
Произвольные	url не более 2048 символов

Эти граничные условия помогут избежать сбоев системы на переходах функционала одного класса к другому.

4.1.3 Тест кейсы

Тест-кейс – это набор шагов или действий, которые выполняются для проверки того, соответствует ли система требованиям и работает ли она должным образом. Он помогает убедиться, что ПО правильно функционирует в разных условиях и отвечает ожиданиям. Тест-кейсы играют важную роль в выявлении ошибок и обеспечении качества продукта [17].

По созданным классам эквивалентности и граничным условиям был создан набор тест-кейсов. Такой подход к тестированию представляет собой алгоритм действий, по которому проводится тестирование уже написанной программы.

Набор тест-кейсов для тестирования системы представлен в таблице 7.

Таблица 7 – Набор тест-кейсов

№	Класс эквивалентности	Входные данные	Выходные данные
1	1,4,6,7,8	https://apisc.test.dev.smartway.today/v1/internal/webhooks/appeal-reopened	Успешно
2	5,9	http://apisc.test.dev.smartway.today/v1/internal/webhooks/appeal-reopened@	Недопустимый url
3	2		Пустое значение

Написанные тест-кейсы помогут выявить недостатки работы системы и предотвратят сбои в ее работе.

4.2 Функциональное тестирование белым ящиком

Тестирование программного обеспечения белым ящиком – это метод тестирования, который предполагает, что структура кода и особенности системы заранее известны тестирующему. При создании тест-кейсов важно избегать любых зависимостей, потому что один из них может быть удален из-за ненужности, тогда сломается логика проверки зависимых от него тест-кейсов [18]. Результаты функционального тестирования приведены в таблице 8.

Таблица 8 – Результаты функционального тестирования

Вариант использования	Тест и его входные параметры	Результат
Создать вебхук	Ввод верных данных: URL: https://apisc.test.dev.smartway.today/v1/internal/webhooks/appeal-reopened; Событие: appeal_reopened; Секрет подписи: SecretKey_2026.	Тест пройден (рисунок Г.1)
	Ввод пустого URL: URL: `` (пустое значение); Событие: appeal_reopened; Секрет подписи: SecretKey_2026.	Тест пройден (рисунок Г.2): отображается сообщение об ошибке «Укажите URL», сохранение недоступно.

Продолжение таблицы 8

Вариант использования	Тест и его входные параметры	Результат
Создать вебхук	Ввод URL с неверным протоколом: URL: http://apisc.test.dev.smartway.today/v1/internal/webhooks/appeal-reopened; Событие: appeal_reopened.	Тест пройден (рисунок Г.3): отображается ошибка «URL должен начинаться с https://».
	Ввод URL с недопустимыми символами: URL: http://apisc.test.dev.smartway.today/v1/internal/webhooks/appeal-reopened@; Событие: appeal_reopened.	Тест пройден (рисунок Г.4): отображается сообщение «URL содержит недопустимые символы» / «Недопустимый URL».
	Ввод URL длиной 2048 символов.	Тест пройден (рисунок Г.5): запись успешно сохраняется.
	Ввод URL длиной 2049 символов.	Тест пройден (рисунок Г.6): отображается ошибка «URL не должен превышать 2048 символов».
	Тип события не выбран.	Без выбора события: URL валидный, событие не выбрано.
Редактировать вебхук.	Изменить URL на валидный https://.../appeal-reopened-v2, событие оставить прежним.	Тест пройден (рисунок Г.8): изменения сохранены, в списке отображается новый URL.
Просмотреть историю отправок.	Открыть «История» у созданного вебхука после отправки тестового события.	Тест пройден (рисунок Г.9): в истории отображаются дата, статус отправки, HTTP-код и тело ответа.

Окончание таблицы 8

Вариант использования	Тест и его входные параметры	Результат
Проверить обработку ошибок	Целевой URL возвращает 404.	Тест пройден (рисунок Г.10): попытка

Функциональное тестирование показало корректность работы основных сценариев управления веб-хуками: создания подписки, проверки URL, редактирования параметров, просмотра истории отправок и фиксации ошибок доставки. Выявленные сценарии подтверждают соответствие реализованного интерфейса ключевым функциональным требованиям.

4.3 Планирование и расчет бюджета разрабатываемого приложения

Оценка стоимости программного обеспечения является одной из ключевых задач управления проектом. Она включает определение трудозатрат, необходимых ресурсов и календарных сроков выполнения работ. Точность оценки напрямую влияет на успешность проекта, поскольку позволяет планировать бюджет, распределять ресурсы и контролировать отклонения в процессе разработки. При этом оценка стоимости должна учитывать неопределённость требований, сложность системы и возможные риски, возникающие в процессе жизненного цикла программного продукта [19].

Для планирования и расчета бюджета разрабатываемого приложения была использована программа Microsoft Project. Данный программный продукт обеспечивает разработку планов, анализ объемов работ, отслеживание прогресса деятельности [20]. Реализация проекта была начата 8 апреля 2026 года.

Во время планирования разработки было выявлено 37 задач. Наименования и сроки их выполнения представлены на рисунке 12.

	Разработка системы обработки вебхуков	47 дней?	08.04.2026, 09:00	09.06.2026, 18:00
	Начало проекта	0 дней	08.04.2026, 09:00	08.04.2026, 09:00
	Анализ программных средств	12 дней?	08.04.2026, 09:00	22.04.2026, 18:00
	Анализ предметной области разрабатываемых программных средств	2 дней	08.04.2026, 09:00	09.04.2026, 18:00
	Анализ и сравнение программного обеспечения аналогичного назначения	2 дней	10.04.2026, 09:00	13.04.2026, 18:00
	Анализ функциональных требований к разрабатываемой системе	3 дней	14.04.2026, 09:00	16.04.2026, 18:00
	Анализ нефункциональных требований	2 дней	17.04.2026, 09:00	20.04.2026, 18:00
	Осуществление планирования разработки и оценки бюджета	2 дней	21.04.2026, 09:00	22.04.2026, 18:00
	Конец анализа программных средств	0 дней	18.04.2026, 09:00	20.04.2026, 18:00
	Проектирование системы	13 дней?	20.04.2026, 09:00	06.05.2026, 18:00
	Проектирование серверной части	11 дней?	20.04.2026, 09:00	04.05.2026, 18:00
	Проектирование архитектуры системы	4 дней	23.04.2026, 09:00	28.04.2026, 18:00
	Проектирование структур данных	2 дней	29.04.2026, 09:00	30.04.2026, 18:00
	Проектирование программных средств	2 дней	01.05.2026, 09:00	04.05.2026, 18:00
	Конец проектирования серверной части	0 дней	20.04.2026, 09:00	20.04.2026, 09:00
	Проектирование клиентской части	2 дней	05.05.2026, 09:00	06.05.2026, 18:00
	Конец проектирования приложения	0 дней	21.04.2026, 09:00	21.04.2026, 09:00
	Разработка системы	20 дней?	29.04.2026, 09:00	25.05.2026, 18:00
	Разработка серверного приложения	15 дней?	29.04.2026, 09:00	19.05.2026, 18:00
	Создание и настройка очереди сообщений	3 дней	07.05.2026, 09:00	11.05.2026, 18:00
	Реализация производителя событий	4 дней	12.05.2026, 09:00	15.05.2026, 18:00
	Реализация системы обработки	2 дней	18.05.2026, 09:00	19.05.2026, 18:00
	Конец разработки серверного приложения	0 дней	29.04.2026, 09:00	29.04.2026, 09:00
	Разработка клиентского приложения	16 дней?	05.05.2026, 09:00	25.05.2026, 18:00
	Реализация дизайна	1 день	20.05.2026, 09:00	20.05.2026, 18:00
	Реализация клиентского приложения	3 дней	21.05.2026, 09:00	25.05.2026, 18:00
	Конец разработки клиентского приложения	0 дней	05.05.2026, 09:00	05.05.2026, 09:00
	Конец разработки приложения	0 дней	05.05.2026, 09:00	05.05.2026, 09:00
	Контроль качества программного обеспечения	19 дней?	07.05.2026, 09:00	01.06.2026, 18:00
	Выбор метрик программного кода	1 день	26.05.2026, 09:00	26.05.2026, 18:00
	Проведение функционального тестирования	2 дней	27.05.2026, 09:00	28.05.2026, 18:00
	Проведение модульного тестирования	1 день	29.05.2026, 09:00	29.05.2026, 18:00
	Вычисление метрик программного кода	1 день	01.06.2026, 09:00	01.06.2026, 18:00
	Конец контроля качества программного обеспечения	0 дней	07.05.2026, 09:00	07.05.2026, 09:00
	Составление отчета	21 дней?	13.05.2026, 09:00	09.06.2026, 18:00
	Оформление пояснительной записки	3 дней	02.06.2026, 09:00	04.06.2026, 18:00
	Составление глоссария и приложений	3 дней	05.06.2026, 09:00	09.06.2026, 18:00

Рисунок 12 – Список и сроки задач

Основным ресурсом разработки является разработчик. Его стандартная ставка составляет 500 р/час.

На рисунке 13 изображена диаграмма Ганта. Этот инструмент представляет собой горизонтальную гистограмму, которая отражает сроки задач и зависимости между ними.

Для реализации системы требуются ресурсы в виде разработчика. Ресурсы, задействованные в проекте, показаны на рисунке 14.

[illegible]

Обзорный отчет проекта представлен на рисунке 15.

Dates			
Start	08.04.2026, 09:00	Finish	09.06.2026, 18:00
Baseline Start		Baseline Finish	
Actual Start		Actual Finish	

Duration			
Scheduled	47 дней	Remaining	47 дней
Baseline	0 дней	Actual	0 дней
		Percent Complete	0 %

Work			
Scheduled	368 часов	Remaining	368 часов

Рисунок 15 – Обзорный отчет проекта

Отчет дает понять, что общий срок разработки при этом равен 47 дней.

Заключение

В результате выполнения выпускной квалификационной работы были выполнены основные этапы разработки системы обработки событий на основе механизма веб-хук.

На стадии анализа программных продуктов-аналогов были сформулированы функциональные и нефункциональные требования к разрабатываемой системе.

На стадии проектирования сервиса была спроектирована архитектура системы, в том числе с применением очереди сообщений, базы данных и маршрутизаторов запросов. На основе спроектированных данных была разработана его архитектура, организована структура базы данных, спроектирован пользовательский интерфейс.

Серверная часть приложения была выполнена в интегрированной среде разработки Goland с использованием языка программирования Go. Для работы с СУБД была использована среда Datagrip, а для работы с RabbitMQ использовался RabbitMQ Management.

Было выполнено планирование и расчет бюджета, в ходе которого удалось вывести ресурсы, задачи и затраты для разрабатываемого продукта.

Исходя из вышесказанного, можно сделать вывод, что за время выполнения выпускной квалификационной работы были выполнены все поставленные задачи.

Список использованных источников

1. Вебхук // Yandex Cloud URL: <https://yandex.cloud/ru/docs/glossary/webhook> (дата обращения: 10.04.2026).
2. К. Ларман Применение uml и шаблонов проектирования. - 2004. - 624 с.
3. Как мы учимся понимать наших пользователей // Хабр URL: <https://habr.com/ru/companies/trackduck/articles/230555/> (дата обращения: 03.05.2026).
4. Обзор чата Zendesk: удивительная цена, интегрированная с авторитетной системой билетов // E-commerce URL: <https://ecommerce-platforms.com/ru/ecommerce-reviews/zendesk-chat-review> (дата обращения: 03.05.2026).
5. Интерком против Crisp: какое приложение для живого чата лучше? // chaty URL: <https://chaty.app/ru/blog/intercom-vs-crisp-live-chat-app/> (дата обращения: 03.05.2026).
6. Сравнительный анализ: определение, принципы, применение и ограничения метода // InvestFuture URL: <https://investfuture.ru/articles/sravnitelnyy-analiz-opredelenie-printsipy-primenenie-i-ogranicheniya-metoda> (дата обращения: 01.05.2026).
7. К. Алистер Современные методы описания функциональных требований к системам. - 2013. - 285 с.
8. Systems and software engineering Systems and software Quality Requirements and Evaluation (SQuaRE) // System and software quality models [Электронный ресурс]. // URL: <https://www.iso.org/standard/35733.html> (дата обращения: 03.05.2026).
9. Шлюзы // Yandex Cloud URL: <https://yandex.cloud/ru/docs/vpc/concepts/gateways> (дата обращения: 30.04.2026).

10. Ingress // Kubernetes URL: <https://kubernetes.io/docs/concepts/services-networking/ingress/> (дата обращения: 30.04.2026).
11. М. Цукалис Golang для профи: создаем профессиональные утилиты, параллельные серверы и сервисы. - 3-е изд. - 2025. - 624 с.
12. Козыра Н. Go на практике // Н. Козыра, М. Батчер, М. Фарина - 2-е изд. - 2025. - 400 с.
13. Р. Мартин Чистый код: создание, анализ и рефакторинг. - 2022. - 610 с.
14. Хороший ретрай, плохой ретрай, или История одного падения // Хабр URL: <https://habr.com/ru/companies/yandex/articles/762678/> (дата обращения: 29.04.2026).
15. Что такое Архитектура ПО? // Хабр URL: <https://habr.com/ru/articles/988560/> (дата обращения: 03.05.2026).
16. С. Кори Искусство тестирования программ. - 3-е изд. - 2020. – 272 с.
17. Классы эквивалентности в тестировании // Точка качества URL: <https://tquality.ru/blog/klassy-ekvivalentnosti-v-testirovanii/> (дата обращения: 01.05.2026).
18. Всё о тест-кейсах: от основ до примеров // QaRocks URL: <https://qarocks.ru/all-about-test-cases/> (дата обращения: 01.05.2026).
19. Основы проектирования интерфейсов для системных аналитиков: ключевые принципы и подходы // Хабр URL: <https://habr.com/ru/articles/872168/> (дата обращения: 03.05.2026).
20. Блэк, Р. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование / Р. Блэк, 2025. 526 с. Текст: непосредственный.

Приложение А
(справочное)
Глоссарий

Веб-хук – механизм уведомления внешней системы о наступлении события посредством сетевого запроса.

Полезная нагрузка – содержательная часть события, передаваемая в теле запроса.

Идемпотентность – свойство безопасной повторной обработки одного и того же события.

Корреляционный идентификатор – обозначение для сквозной трассировки запроса между компонентами системы.

Событие первого сообщения – событие первого пользовательского сообщения в новом обращении.

Подписка на веб-хук – настроенная связь между источником события и внешней системой-получателем, включающая перечень событий и параметры доставки.

Маршрутизация события – выбор обработчика или внешнего получателя на основе типа события, признаков диалога и конфигурации интеграции.

Повторная доставка – повторная попытка отправки события после временного отказа внешней системы или сетевого сбоя.

Журнал доставки – совокупность записей о приеме, передаче, ошибках и результатах обработки событий.

Приложение Б
(обязательное)
Код серверной части

Листинг Б.1 – handlers.go

```

package api

import (
    "encoding/json"
    "net/http"

    "github.com/example/supportdesk-webhook-service/internal/service"
    "github.com/gorilla/mux"
)

type WebhookHandler struct {
    service *service.WebhookService
}

func NewWebhookHandler(svc *service.WebhookService) *WebhookHandler {
    return &WebhookHandler{service: svc}
}

func (h *WebhookHandler) RegisterRoutes(r *mux.Router) {
    r.HandleFunc("/api/webhooks", h.List).Methods(http.MethodGet)
    r.HandleFunc("/api/webhooks", h.Create).Methods(http.MethodPost)
    r.HandleFunc("/api/webhooks/{id}",
h.Update).Methods(http.MethodPut)
    r.HandleFunc("/api/webhooks/{id}",
h.Delete).Methods(http.MethodDelete)
    r.HandleFunc("/api/webhooks/history",
h.History).Methods(http.MethodGet)
    r.HandleFunc("/api/webhooks/{id}/test",
h.Test).Methods(http.MethodPost)
}

func (h *WebhookHandler) List(w http.ResponseWriter, r *http.Request) {
    items, err := h.service.GetAll(r.Context())
    if err != nil {
        writeError(w, http.StatusInternalServerError, err)
        return
    }
    writeJSON(w, http.StatusOK, items)
}

```

```

func (h *WebhookHandler) Create(w http.ResponseWriter, r *http.Request)
{
    var req service.CreateWebhookRequest
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        writeError(w, http.StatusBadRequest, err)
        return
    }
    if err := h.service.Create(r.Context(), req); err != nil {
        writeError(w, http.StatusInternalServerError, err)
        return
    }
    w.WriteHeader(http.StatusCreated)
}

func (h *WebhookHandler) Update(w http.ResponseWriter, r *http.Request)
{
    id := mux.Vars(r)["id"]
    var req service.UpdateWebhookRequest
    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        writeError(w, http.StatusBadRequest, err)
        return
    }
    item, err := h.service.Update(r.Context(), id, req)
    if err != nil {
        writeError(w, http.StatusInternalServerError, err)
        return
    }
    writeJSON(w, http.StatusOK, item)
}

func (h *WebhookHandler) Delete(w http.ResponseWriter, r *http.Request)
{
    id := mux.Vars(r)["id"]
    if err := h.service.Delete(r.Context(), id); err != nil {
        writeError(w, http.StatusInternalServerError, err)
        return
    }
    w.WriteHeader(http.StatusNoContent)
}

func (h *WebhookHandler) History(w http.ResponseWriter, r *http.Request)
{

```

```

        url := r.URL.Query().Get("url")
        if url == "" {
            http.Error(w, "url query param required",
http.StatusBadRequest)
            return
        }
        items, err := h.service.DeliveryHistoryByURL(r.Context(), url)
        if err != nil {
            writeError(w, http.StatusInternalServerError, err)
            return
        }
        writeJSON(w, http.StatusOK, items)
    }

func (h *WebhookHandler) Test(w http.ResponseWriter, r *http.Request) {
    id := mux.Vars(r)["id"]
    result, err := h.service.SendTest(r.Context(), id)
    if err != nil {
        writeError(w, http.StatusInternalServerError, err)
        return
    }
    writeJSON(w, http.StatusOK, result)
}

func writeJSON(w http.ResponseWriter, status int, payload any) {
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(status)
    _ = json.NewEncoder(w).Encode(payload)
}

func writeError(w http.ResponseWriter, status int, err error) {
    writeJSON(w, status, map[string]string{"error": err.Error()})
}

```

Листинг Б.2 – service.go

```

package service

import (
    "context"
    "fmt"
    "strings"
    "time"

```

```

        "github.com/example/supportdesk-webhook-service/internal/models"
        "github.com/example/supportdesk-webhook-
service/internal/repository"
        "github.com/example/supportdesk-webhook-service/internal/webhook"
        "github.com/google/uuid"
    )

```

```

type CreateWebhookRequest struct {
    Events          []string `json:"events"`
    URL             string   `json:"url"`
    SigningSecret   *string  `json:"signingSecret,omitempty"`
}

```

```

type UpdateWebhookRequest struct {
    EventName       string   `json:"eventName"`
    URL             string   `json:"url"`
    SigningSecret   *string  `json:"signingSecret,omitempty"`
}

```

```

type WebhookInfo struct {
    ID              string   `json:"id"`
    EventName       string   `json:"eventName"`
    URL             string   `json:"url"`
    HasSigningSecret bool      `json:"hasSigningSecret"`
    Active          bool      `json:"active"`
    CreatedAt       time.Time `json:"createdAt"`
    UpdatedAt       time.Time `json:"updatedAt"`
}

```

```

type DeliveryHistoryInfo struct {
    ID              string   `json:"id"`
    WebhookID       string   `json:"webhookId"`
    EventName       string   `json:"eventName"`
    URL             string   `json:"url"`
    RequestBody     string   `json:"requestBody"`
    RequestHash     string   `json:"requestHash"`
    RequestSignature *string  `json:"requestSignature,omitempty"`
    ResponseStatus  *int     `json:"responseStatus,omitempty"`
    ResponseBody    *string  `json:"responseBody,omitempty"`
    Error           *string  `json:"error,omitempty"`
    IsTest          bool      `json:"isTest"`
    CreatedAt       time.Time `json:"createdAt"`
}

```

```

    }

    type WebhookService struct {
        repo    *repository.WebhookRepository
        client  *webhook.Client
    }

    func NewWebhookService(repo    *repository.WebhookRepository,    client
    *webhook.Client) *WebhookService {
        return &WebhookService{repo: repo, client: client}
    }

    func (s    *WebhookService) Create(ctx    context.Context,    req
    CreateWebhookRequest) error {
        secret := normalizeSecret(req.SigningSecret)
        rows := make([]models.Webhook, len(req.Events))

        for i, event := range req.Events {
            id, err := uuid.NewUUID()
            if err != nil {
                return fmt.Errorf("generate id: %w", err)
            }
            rows[i] = models.Webhook{
                ID:            id.String(),
                EventName:     event,
                URL:          req.URL,
                SigningSecret: secret,
            }
        }

        return s.repo.SaveAll(ctx, rows)
    }

    func (s    *WebhookService) GetAll(ctx    context.Context) ([]WebhookInfo,
    error) {
        rows, err := s.repo.GetAll(ctx)
        if err != nil {
            return nil, err
        }

        result := make([]WebhookInfo, 0, len(rows))
        for _, row := range rows {

```

```

        result = append(result, mapWebhookInfo(row))
    }
    return result, nil
}

func (s *WebhookService) Update(ctx context.Context, id string, req
UpdateWebhookRequest) (WebhookInfo, error) {
    wh, err := s.repo.ByID(ctx, id)
    if err != nil {
        return WebhookInfo{}, err
    }

    wh.EventName = req.EventName
    wh.URL = req.URL
    if req.SigningSecret != nil {
        wh.SigningSecret = normalizeSecret(req.SigningSecret)
    }

    if err = s.repo.Update(ctx, wh); err != nil {
        return WebhookInfo{}, err
    }

    updated, err := s.repo.ByID(ctx, id)
    if err != nil {
        return WebhookInfo{}, err
    }
    return mapWebhookInfo(updated), nil
}

func (s *WebhookService) Delete(ctx context.Context, id string) error {
    return s.repo.Delete(ctx, id)
}

func (s *WebhookService) DeliveryHistoryByURL(ctx context.Context, url
string) ([]DeliveryHistoryInfo, error) {
    rows, err := s.repo.DeliveryHistoryByURL(ctx, url)
    if err != nil {
        return nil, err
    }

    result := make([]DeliveryHistoryInfo, 0, len(rows))
    for _, row := range rows {

```

```

        result = append(result, mapHistoryInfo(row))
    }
    return result, nil
}

func (s *WebhookService) SendTest(ctx context.Context, id string)
(DeliveryHistoryInfo, error) {
    wh, err := s.repo.ByID(ctx, id)
    if err != nil {
        return DeliveryHistoryInfo{}, err
    }

    payload := map[string]any{
        "type": "test",
        "name": wh.EventName,
        "payload": map[string]any{
            "webhookId": wh.ID,
            "url":      wh.URL,
            "sentAt":    time.Now().UTC().Format(time.RFC3339),
        },
    }

    result := s.client.Send(ctx, webhook.DeliveryRequest{
        URL:      wh.URL,
        SigningSecret: wh.SigningSecret,
        Payload:    payload,
    })

    if err = webhook.SaveDeliveryHistory(ctx, s.repo, wh, result,
true); err != nil {
        return DeliveryHistoryInfo{}, err
    }

    history, err := s.repo.DeliveryHistoryByURL(ctx, wh.URL)
    if err != nil || len(history) == 0 {
        return DeliveryHistoryInfo{}, fmt.Errorf("load test history")
    }
    return mapHistoryInfo(history[0]), nil
}

func mapWebhookInfo(wh models.Webhook) WebhookInfo {
    return WebhookInfo{

```

```

        ID:                wh.ID,
        EventName:         wh.EventName,
        URL:               wh.URL,
        HasSigningSecret:  wh.SigningSecret != nil &&
strings.TrimSpace(*wh.SigningSecret) != "",
        Active:           wh.Active,
        CreatedAt:        wh.CreatedAt,
        UpdatedAt:        wh.UpdatedAt,
    }
}

```

```

func mapHistoryInfo(h models.WebhookDeliveryHistory) DeliveryHistoryInfo
{
    return DeliveryHistoryInfo{
        ID:                h.ID,
        WebhookID:         h.WebhookID,
        EventName:         h.EventName,
        URL:               h.URL,
        RequestBody:       h.RequestBody,
        RequestHash:       h.RequestHash,
        RequestSignature:  h.RequestSignature,
        ResponseStatus:    h.ResponseStatus,
        ResponseBody:      h.ResponseBody,
        Error:             h.Error,
        IsTest:            h.IsTest,
        CreatedAt:         h.CreatedAt,
    }
}

```

```

func normalizeSecret(secret *string) *string {
    if secret == nil {
        return nil
    }
    trimmed := strings.TrimSpace(*secret)
    if trimmed == "" {
        return nil
    }
    return &trimmed
}

```

Приложение В
(обязательное)
Код клиентской части

Листинг В.1 – webhooks.ts

```
/**
 * RTK Query API-клиент для управления вебхуками.
 * Аналог super-admin-webhooks/api.ts из web-app.
 */

export type WebhookDto = {
  id: string;
  eventName: string;
  url: string;
  hasSigningSecret: boolean;
  active: boolean;
  createdAt: string;
  updatedAt: string;
};

export type CreateWebhookRequest = {
  url: string;
  events: string[];
  signingSecret?: string;
};

export type UpdateWebhookRequest = {
  eventName: string;
  url: string;
  signingSecret?: string;
};

export type WebhookDeliveryHistoryDto = {
  id: string;
  webhookId: string;
  eventName: string;
  url: string;
  requestBody: string;
  requestHash: string;
  requestSignature?: string;
  responseStatus?: number;
  responseBody?: string;
  error?: string;
  isTest: boolean;
  createdAt: string;
};
```

```

};

const API_BASE = import.meta.env.VITE_WEBHOOK_API_URL ??
'http://localhost:8080';

async function request<T>(path: string, init?: RequestInit): Promise<T>
{
  const response = await fetch(`${API_BASE}${path}`, {
    ...init,
    headers: {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${localStorage.getItem('token')} ?? ''`,
      ...init?.headers,
    },
  });

  if (!response.ok) {
    throw new Error(`HTTP ${response.status}`);
  }

  if (response.status === 204) {
    return undefined as T;
  }

  return response.json();
}

export const webhooksApi = {
  list: () => request<WebhookDto[]>('/api/webhooks'),
  create: (body: CreateWebhookRequest) =>
    request<void>('/api/webhooks', { method: 'POST', body:
JSON.stringify(body) }),
  update: (id: string, body: UpdateWebhookRequest) =>
    request<WebhookDto>(`/api/webhooks/${id}`, { method: 'PUT', body:
JSON.stringify(body) }),
  remove: (id: string) => request<void>(`/api/webhooks/${id}`, { method:
'DELETE' }),
  test: (id: string) =>
request<WebhookDeliveryHistoryDto>(`/api/webhooks/${id}/test`, { method:
'POST' }),
  history: (url: string) =>

```

```

request<WebhookDeliveryHistoryDto[]>(`/api/webhooks/history?url=${encodeURIComponent(url)}`),
    };
WebhookCreateModal.ts
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';

import { webhooksApi } from '../api/webhooks';
import { WEBHOOK_EVENT_OPTIONS } from '../constants';
import { webhookFormSchema, type WebhookFormValues } from
'../schemas/webhook-form';

type Props = {
    onSuccess: () => void;
    onClose: () => void;
};

/** Модальное окно создания вебхука. */
export function WebhookCreateModal({ onSuccess, onClose }: Props) {
    const {
        register,
        handleSubmit,
        formState: { errors, isSubmitting },
    } = useForm<WebhookFormValues>({
        resolver: zodResolver(webhookFormSchema),
        defaultValues: { url: '', events: [], signingSecret: '' },
    });

    const onSubmit = handleSubmit(async (values) => {
        await webhooksApi.create({
            url: values.url,
            events: values.events,
            signingSecret: values.signingSecret || undefined,
        });
        onSuccess();
        onClose();
    });

    return (
        <dialog open>
            <form onSubmit={onSubmit}>

```

```

    <h2>Создать вебхук</h2>

    <label>
      URL (HTTPS)
      <input type="url" {...register('url')}
placeholder="https://example.com/hooks" />
      {errors.url && <span>{errors.url.message}</span>}
    </label>

    <fieldset>
      <legend>События</legend>
      {WEBHOOK_EVENT_OPTIONS.map((event) => (
        <label key={event.value}>
          <input type="checkbox" value={event.value}
{...register('events')} />
          {event.label}
        </label>
      ))}
      {errors.events && <span>{errors.events.message}</span>}
    </fieldset>

    <label>
      Секрет подписи (необязательно)
      <input type="password" {...register('signingSecret')}
autoComplete="off" />
    </label>

    <footer>
      <button type="button" onClick={onClose}>
        Отмена
      </button>
      <button type="submit" disabled={isSubmitting}>
        Сохранить
      </button>
    </footer>
  </form>
</dialog>
);
}

```

Листинг В.2 – WebhooksPage.tsx

```
import { useCallback, useEffect, useState } from 'react';

import { webhooksApi, type WebhookDto } from '../api/webhooks';
import { getWebhookEventLabel } from '../constants';

/**
 * Страница управления вебхуками (упрощённый аналог super-admin-
webhooks).
 */
export function WebhooksPage() {
  const [webhooks, setWebhooks] = useState<WebhookDto[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  const load = useCallback(async () => {
    setLoading(true);
    setError(null);
    try {
      setWebhooks(await webhooksApi.list());
    } catch {
      setError('Не удалось загрузить вебхуки');
    } finally {
      setLoading(false);
    }
  }, []);

  useEffect(() => {
    void load();
  }, [load]);

  const handleTest = async (webhook: WebhookDto) => {
    await webhooksApi.test(webhook.id);
    alert(`Тестовый вебхук отправлен на ${webhook.url}`);
  };

  const handleDelete = async (webhook: WebhookDto) => {
    if (!confirm(`Удалить вебхук ${webhook.url}?`)) return;
    await webhooksApi.remove(webhook.id);
  };
}
```

```

        await load();
    };

    if (loading) return <p>Загрузка...</p>;
    if (error) return <p>{error}</p>;

    if (webhooks.length === 0) {
        return (
            <section>
                <h1>Вебхуки</h1>
                <p>Вебхуки не настроены. Добавьте URL и выберите события.</p>
                <button type="button">Создать вебхук</button>
            </section>
        );
    }

    return (
        <section>
            <header>
                <h1>Вебхуки</h1>
                <button type="button">Создать вебхук</button>
            </header>

            <table>
                <thead>
                    <tr>
                        <th>Событие</th>
                        <th>URL</th>
                        <th>Создан</th>
                        <th>Подпись</th>
                        <th>Действия</th>
                    </tr>
                </thead>
                <tbody>
                    {webhooks.map((webhook) => (
                        <tr key={webhook.id}>
                            <td>{getWebhookEventLabel(webhook.eventName)}</td>
                            <td>{webhook.url}</td>
                            <td>{new Date(webhook.createdAt).toLocaleString('ru-
RU')}</td>
                            <td>{webhook.hasSigningSecret ? 'Да' : 'Нет'}</td>

```

```

        <td>
            <button          type="button"          onClick={ ()          =>
handleTest (webhook) }>
                Тест
            </button>
            <button          type="button"          onClick={ ()          =>
handleDelete (webhook) }>
                Удалить
            </button>
        </td>
    </tr>
    </tbody>
</table>
</section>
);
}

```

Листинг В.3 – webhook-form.ts

```

import { z } from 'zod';

const MAX_URL_LENGTH = 2048;
const ALLOWED_URL_CHARS = /^[A-Za-z0-9\-\._~:/?#\[\]!$&'()*+,\;=%]+$/;

/** Схема валидации формы создания/редактирования вебхука. */
export const webhookFormSchema = z.object({
  url: z
    .string()
    .trim()
    .min(1, 'Укажите URL')
    .max(MAX_URL_LENGTH, 'URL не должен превышать 2048 символов')
    .refine((v) => v.startsWith('https://'), 'URL должен начинаться с
https://')
    .refine((v) => !/\s/u.test(v), 'URL не должен содержать пробелы')
    .refine((v) => ALLOWED_URL_CHARS.test(v), 'URL содержит недопустимые
символы')
    .refine((v) => {
      try {
        new URL(v);
        return true;
      } catch {

```

```
        return false;
    }
    }, 'Укажите корректный URL'),
    events: z.array(z.string()).min(1, 'Выберите хотя бы одно событие'),
    signingSecret: z.string(),
  });

export type WebhookFormValues = z.infer<typeof webhookFormSchema>;
```

Приложение Г
(рекомендуемое)
Результаты тестирования

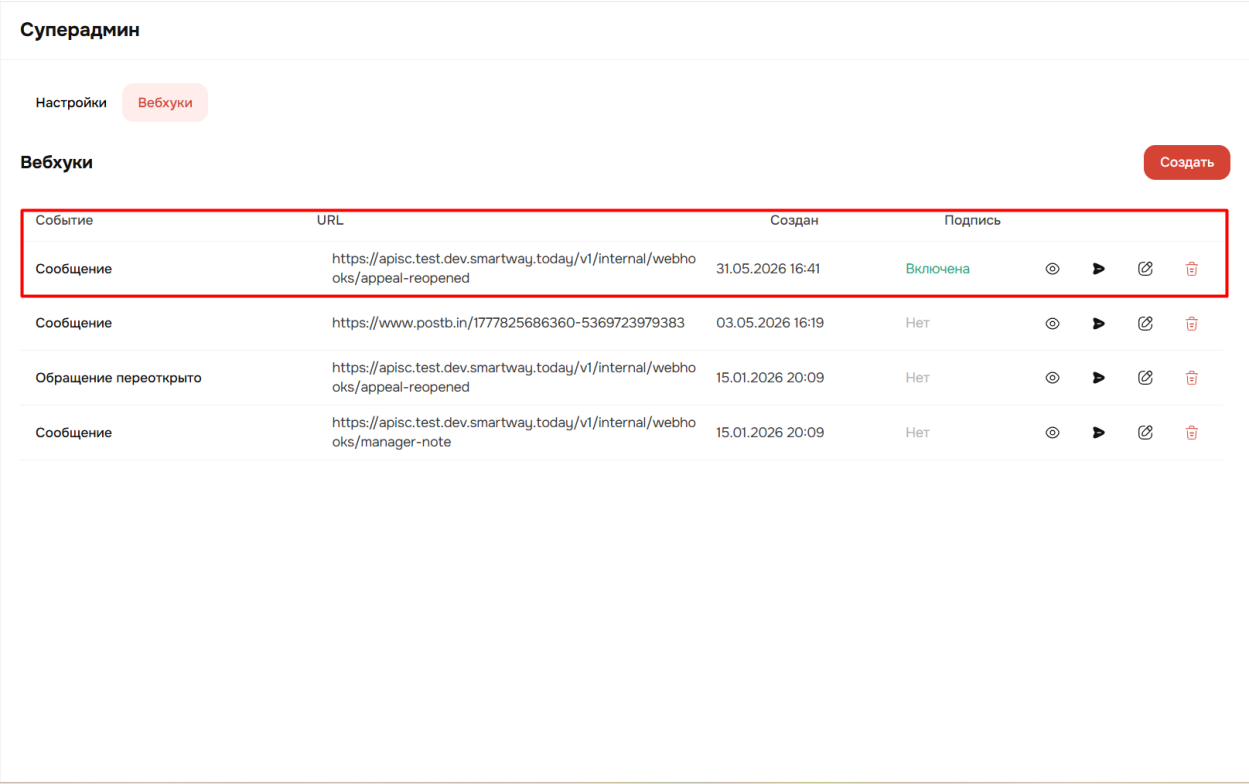


Рисунок Г.1 – Ввод верных данных

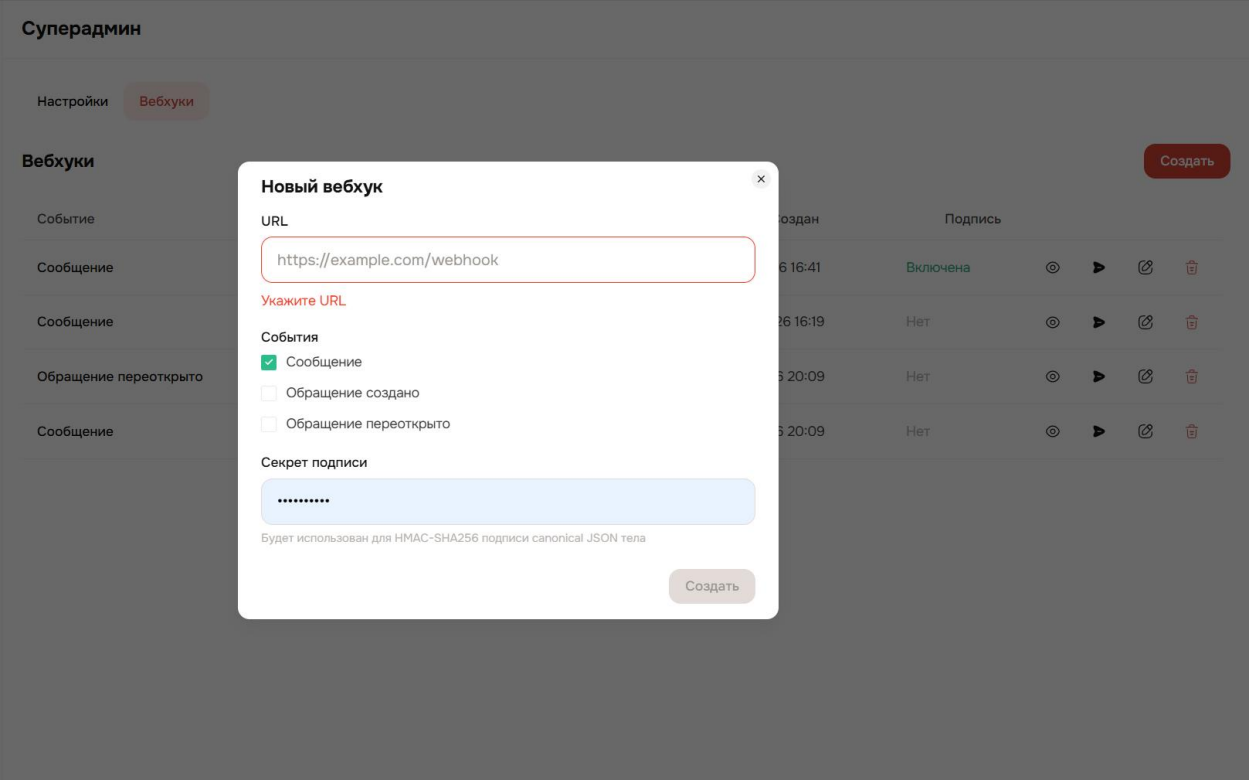


Рисунок Г.2 – Ввод пустого URL

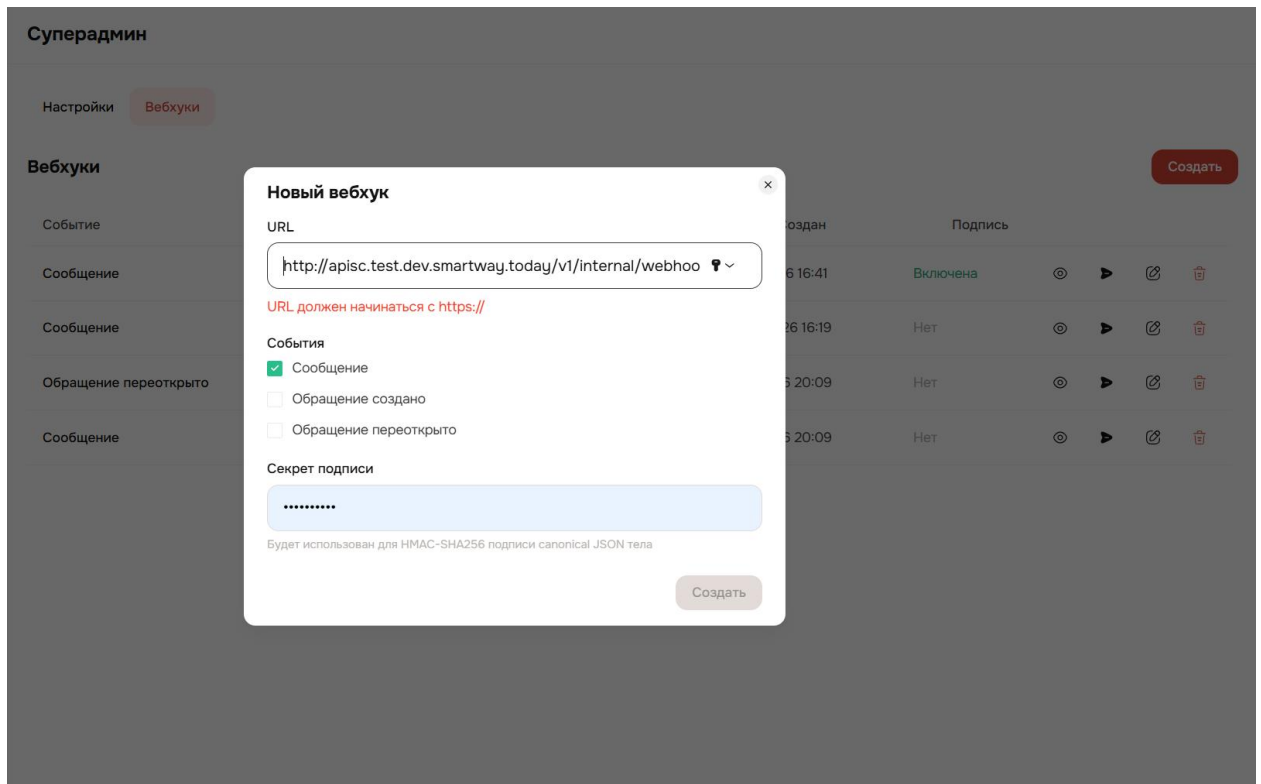


Рисунок Г.3 – Ввод URL с неверным протоколом

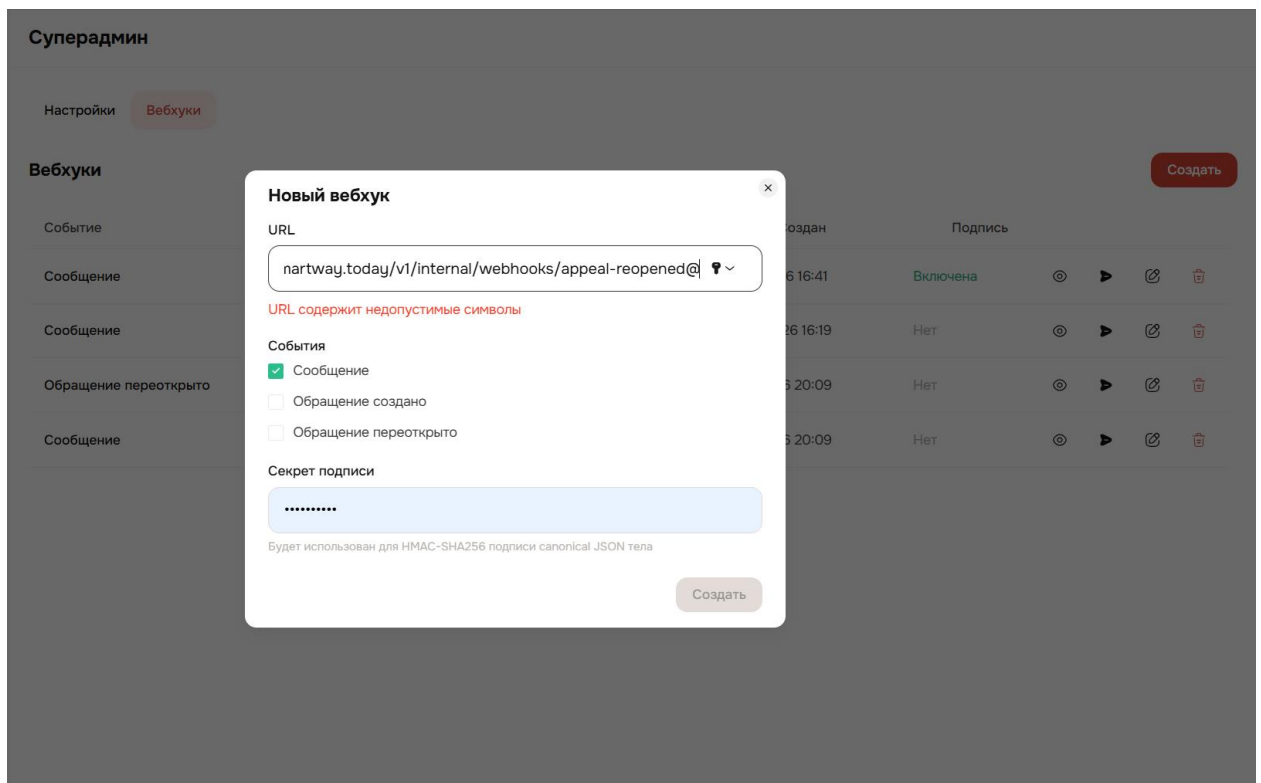


Рисунок Г.4 – Ввод URL с недопустимыми символами

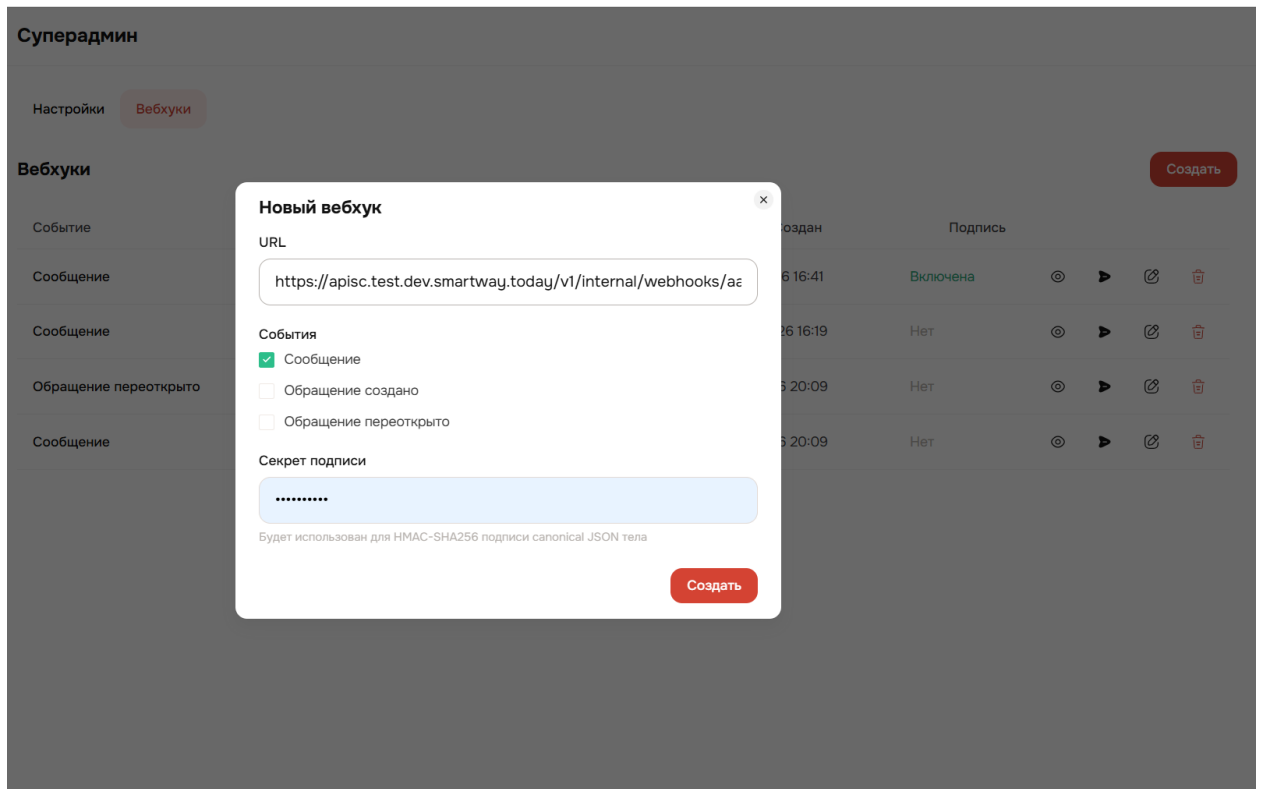


Рисунок Г.5 – Ввод URL длиной 2048 символов

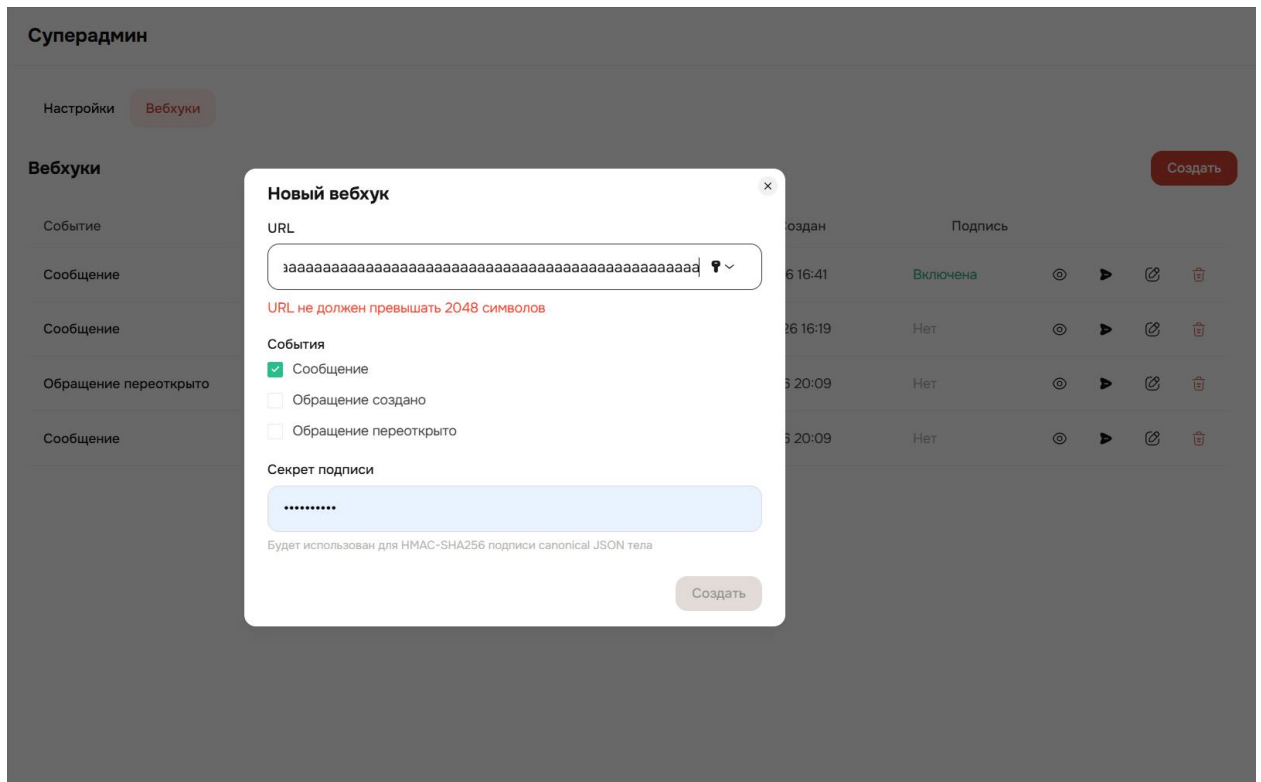


Рисунок Г.6 – Ввод URL длиной 2049 символов

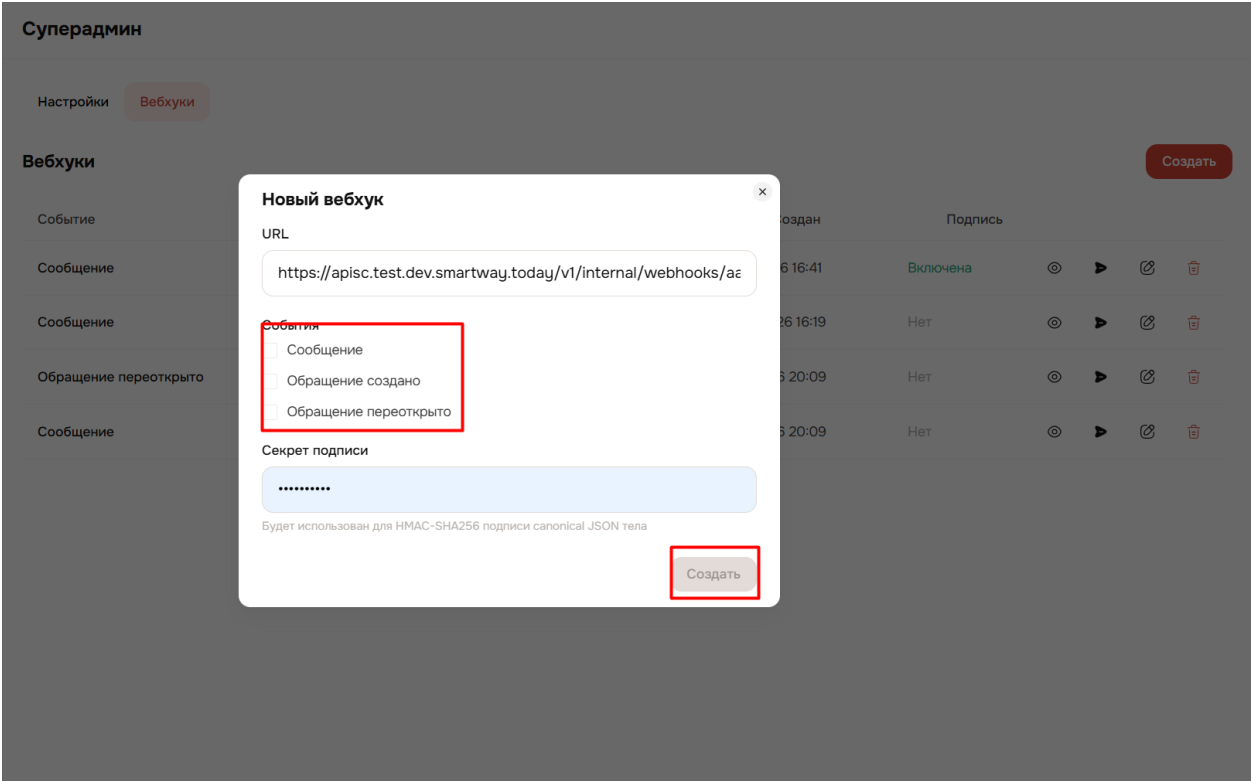


Рисунок Г.7 – Без выбора события

Суперадмин					
Настройки Вебхуки					
Вебхуки		Создать			
Событие	URL	Создан	Подпись		
Сообщение	https://apisc.test.dev.smartway.today/v1/internal/webhooks/apel-reopened-v2	31.05.2026 16:41	Включена	👁	▶
Сообщение	https://www.postb.in/1777825686360-5369723979383	03.05.2026 16:19	Нет	👁	▶
Обращение переоткрыто	https://apisc.test.dev.smartway.today/v1/internal/webhooks/apel-reopened	15.01.2026 20:09	Нет	👁	▶
Сообщение	https://apisc.test.dev.smartway.today/v1/internal/webhooks/manager-note	15.01.2026 20:09	Нет	👁	▶

Рисунок Г.8 – Изменить URL

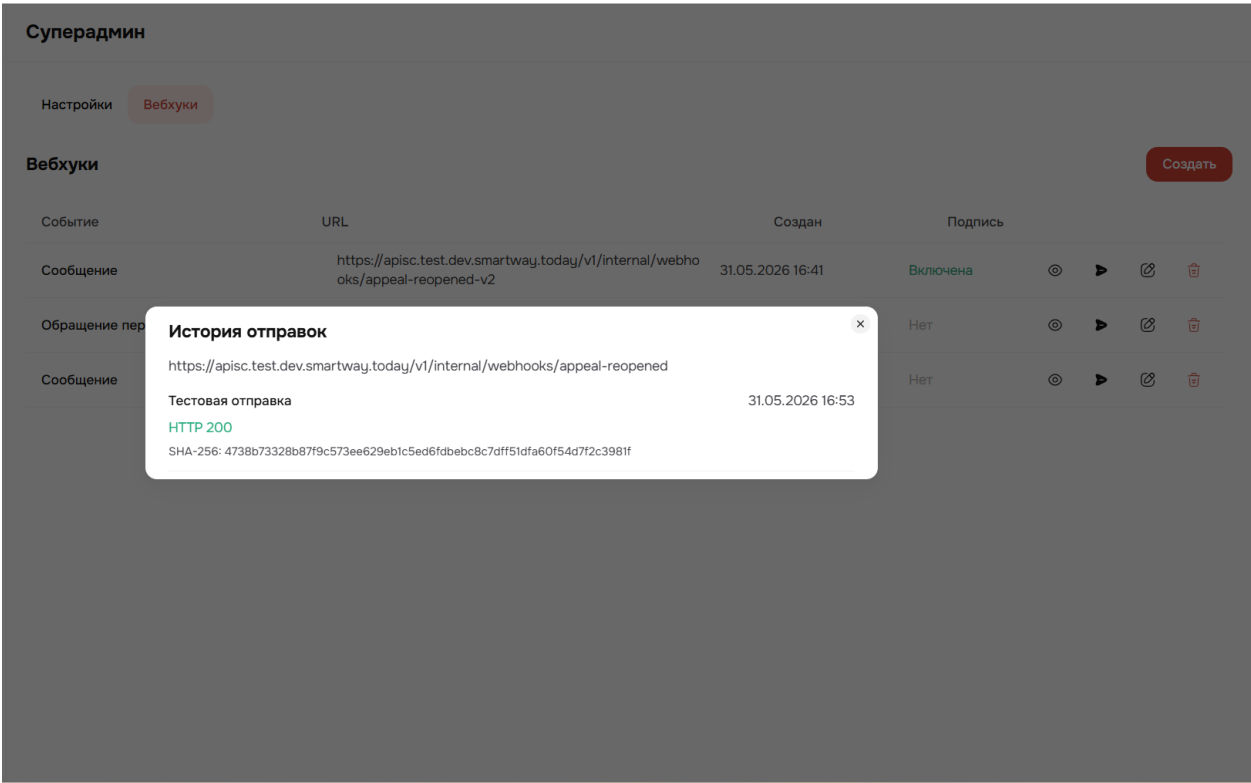


Рисунок Г.9 – Просмотреть историю отправок

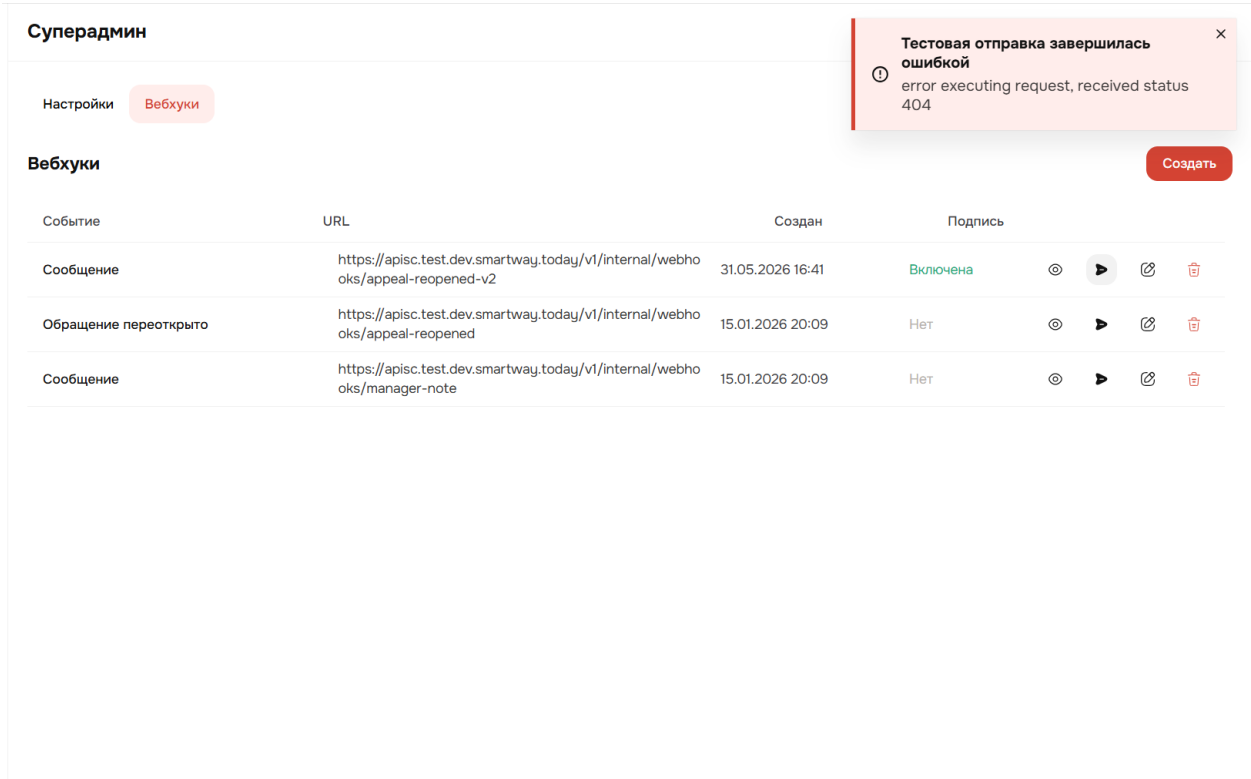


Рисунок Г.10 – Проверка обработки ошибки доставки

Приложение Д
(необязательное)
Справка о внедрении

SAWADY

Общество с ограниченной ответственностью «Савади»
(ООО «Савади»)
ИНН: 7725809338 КПП: 7725010011
115093, г. Москва, муниципальный округ Даниловский,
пер. Партийный д. 1, к. 57, стр. 3

02.06.2026 № 11
На № от

СПРАВКА

о внедрении результатов выпускной квалификационной работы

Результаты выпускной квалификационной работы студента ФГБОУ ВО «Пензенский государственный университет» группы 22ВП1 очной формы обучения Градского Алексея Владимировича на тему «Программные средства обработки событий на основе механизма Веб-хук» рассмотрены и одобрены, как имеющие практическую значимость. Разработанный автором выпускной квалификационной работы механизм распределения обращений используется в работе продукта Smartchat.

Специалист по кадровому делопроизводству



А.А. Краснова