

Санкт-Петербургский государственный университет

КРАЕВ Антон Иванович

Выпускная квалификационная работа

**Система управления доступом к цифровым
товарам в рамках оказания услуги по
анализу почвы для установки свай**

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2021 «Программная инженерия»*

Научный руководитель:
профессор кафедры системного программирования, д. т. н., Кознов Д. В.

Рецензент:
младший инженер-программист, ООО «АТБ Электроника», Захаров Ф. П.

Санкт-Петербург
2025

Saint Petersburg State University

Anton Kraev

Bachelor's Thesis

Digital goods access management system as
part of the provision of soil analysis service
for pile installation

Education level: bachelor

Speciality *09.03.04 "Software Engineering"*

Programme *CB.5080.2021 "Software Engineering"*

Scientific supervisor:
Sc.D, prof. D.V. Koznov

Reviewer:
junior software engineer at "ATB Electronics", P.P. Zakharov

Saint Petersburg
2025

Оглавление

Введение	4
1. Постановка задачи	6
2. Обзор	7
2.1. Обзор предметной области	7
2.2. Готовые решения для продажи цифровых товаров	8
2.3. Выводы	11
3. Архитектура	12
4. Особенности реализации	20
4.1. Хранение данных	20
4.2. Серверная часть	21
4.3. Клиентская часть	23
4.4. Запуск и использование проекта	24
5. Тестирование и внедрение	28
Заключение	30
Список литературы	31

Введение

В настоящее время деятельность компаний все чаще включает в себя электронную коммерцию. И актуальной задачей в этой сфере является продажа цифровых товаров, таких как файлы, программное обеспечение, доступ к страничке сайта. Для решения этой задачи нужно уметь принимать платежи от клиентов и предоставлять доступ к оплаченным товарам.

Существует множество готовых решений, позволяющих так или иначе продавать цифровые товары, от инструментов, которые упрощают эту задачу, такие как сервисы для проведения платежей, до полноценных настраиваемых решений, которые предоставляют прием платежей, организацию доступа к цифровым товарам и их отсылку клиентам, ведение статистики. Каждое решение позволяет решать стандартные задачи определенных групп пользователей, однако не всегда подходит под специфику конкретного бизнеса. И одним из вариантов реализации такой функциональности для удовлетворения всех потребностей бизнеса с нестандартными задачами является разработка собственного сервиса, который работает с платежным провайдером для создания платежей и обработки событий об изменении статусов этих платежей, хранит данные клиентов и их покупки, управляет доступом к товару в зависимости от статуса платежа.

Одной из компаний, продающих цифровые товары, является ООО «Стройматик». Компания предоставляет услугу динамического зондирования грунта [34], то есть анализа почвы на предмет возможности монтажа свайного фундамента. По результатам оказания услуги с использованием сервисов компании формируется файл с отчетом, являющийся цифровым товаром в данном случае. Специфика в том, что товар уникален для каждого пользователя и неизвестен заранее. На данный момент отслеживание статуса оплаты и отправка отчета происходит вручную, клиент переводит деньги по реквизитам и ему отправляют файл в мессенджере, у такого подхода есть недостатки, откуда вытекает запрос на автоматизацию процесса приема платежей и предо-

ставления отчетов, но не ломая при этом пользовательской привычки личного общения с представителем компании. Помимо оказания услуг, «Стройматик» продает оборудование и технологии для установки свай, в том числе программное обеспечение, что является дополнительной мотивацией разработки собственного решения.

Данная работа посвящена разработке системы с открытым исходным кодом для продажи цифровых товаров, в том числе отчетов об анализе почвы как заключительного этапа бизнес-процесса ООО «Стройматик», а также для включения в группу технологий компании для дальнейшей реализации другим компаниям.

1. Постановка задачи

Целью работы является создание системы для доставки клиентам цифровых товаров и управления доступом к ним в зависимости от статуса платежа в рамках оказания услуги по анализу почвы для установки свай. Для её выполнения были поставлены следующие задачи.

1. Сформулировать требования к системе с учетом потребностей заказчика и ограничений предметной области.
2. Спроектировать систему с учетом требований и возможности использования готовых решений.
3. Разработать систему.
4. Выполнить тестирование системы.
5. Внедрить систему в процесс выполнения услуги по анализу почвы.

2. Обзор

2.1. Обзор предметной области

Компания «Стройматик» предоставляет услугу зондирования почвы перед установкой свай, по результатам работы формируется отчет в виде pdf-файла, который сам по себе представляет ценность и является цифровым товаром в данном случае. Этот файл создается другими сервисами компании и сохраняется в объектное хранилище MinIO [25]. Отчет уникален для каждого клиента и неизвестен заранее, так как формируется после оказания услуги.

У компании есть простая веб-страница с формой для запроса обратного звонка для согласования работ по анализу почвы, но нет интерфейса для покупки и предоставления отчета клиенту. Сейчас для оплаты и получения отчета клиент общается с представителем (администратором) компании в мессенджере, получает реквизиты для оплаты, переводит деньги и получает файл с отчетом. Хочется автоматизировать процесс оплаты и доставки товара клиенту, но не ломая при этом пользовательскую привычку личного общения с представителем компании. Также важно иметь с одной стороны полностью независимый сервис с понятным клиентам и администратору интерфейсом (не обязательно графическим) для оплаты и доставки цифровых товаров, но с другой стороны сервис должен предоставлять возможность интеграции в существующую систему и использования по API другими сервисами компании без участия администратора.

Также «Стройматик», помимо оказания услуг по анализу почвы и монтажу свайных фундаментов, продает оборудование и технологии, в том числе программное обеспечение, другим компаниям в сфере строительства, которые тоже могут оказывать услуги клиентам и использовать сервис для продажи файлов или других цифровых товаров. То есть система, которой посвящена данная работа, будет включена в список технологий компании, что является дополнительной мотивацией разработки собственного решения, что, в свою очередь, также позво-

лит создать гибкое решение, которое можно будет модифицировать под нужды новых клиентов.

2.2. Готовые решения для продажи цифровых товаров

Данный раздел посвящен обзору готовых решений для оплаты и доставки цифровых товаров, которые потенциально можно использовать для решения задачи, описанной ранее, а также сервисов, не решающих задачу доставки, но позволяющих проводить онлайн-платежи.

Существуют различные форматы полноценных решений, которые включают в себя и проведение платежей, и доставку оплаченных цифровых товаров: маркетплейсы, SaaS-платформы и другие настраиваемые веб-сервисы, боты в мессенджерах. Ниже приведен обзор некоторых готовых решений.

Ozon [27] — маркетплейс, позволяющий продавать как физические, так и цифровые товары с доставкой по электронной почте или в приложение партнера, хорошо подходит для продажи тиражируемых товаров, таких как электронные книги, сертификаты, лицензии. Для «Стройматик» же товар, то есть файл с отчетом, уникален для каждого клиента, можно было бы создавать отдельную карточку с товаром для каждого пользователя, однако на маркетплейсе есть модерация перед выкладыванием товара, соответственно клиент не сможет купить отчет сразу после его формирования по результатам оказания услуги анализа почвы.

АвтоВебОфис [32] — платформа, нацеленная на продажу онлайн-курсов, однако позволяющая продавать и другие файлы с доставкой по электронной почте или доступом к ним через личный кабинет. Для задачи продажи файла платформа имеет избыточный интерфейс, требует регистрацию, а также все отчеты, загруженные на платформу, доступны для покупки любому пользователю.

Фоху [15] — гибкое решение для электронной коммерции, включающее множество вариантов оплаты и доставки цифровых товаров и

предоставляющее различные варианты интеграции. Недостаток в том, что не поддерживается оплата с российской карты.

Рокуро [36] — сервис для продажи цифровых товаров, позволяет создавать ссылки для оплаты и получения купленного файла, доступна доставка по электронной почте и скачивание файла по ссылке. Минималистичное решение, хорошо подходящее для продажи файлов, однако отсутствует возможность добавления новых файлов по API.

UniotBot [29] — конструктор Телеграм-ботов [30] для продажи цифровых товаров. Созданный через данное решение бот ведет статистику, поддерживает различные платежные провайдеры и форматы продаваемых файлов, позволяет генерировать ссылки на оплату конкретного файла, то есть клиенту будет доступно только то, что ему нужно. В целом интересное и удобное решение, но варианты доставки файла ограничены сообщением в Телеграм и есть сложности с добавлением новых файлов через API.

Stripe-link-file-delivery [10] — бессерверное решение с открытым исходным кодом на Node.js [35] для оплаты файлов через зарубежный платежный провайдер Stripe [37] и отправки по электронной почте. Простое решение для продажи файлов, но весь код содержится в одном файле и для замены платежного провайдера и доработки решения под задачу придется переписывать большую часть кода.

Из обзора рассмотренных решений можно сделать вывод, что каждое из них решает свою проблему, но не подходит по одному или нескольким критериям для рассматриваемой в данной работе задачи, ниже приведен список таких недочетов у различных решений:

- зарубежный — отсутствуют способы оплаты внутри России и есть риски блокировки;
- сложный, неудобный, содержащий много лишнего для клиента интерфейс — требует регистрации, содержит рекламу и т.д.;
- нет возможности интеграции по API, добавление файлов вручную;

- не поддерживает добавление файлов динамически;
- после добавления файл становится доступен всем клиентам, а не только заинтересованному;
- ограниченная логика доставки после оплаты — нет возможности отправить отчет клиенту в мессенджере, на электронную почту или по ссылке;
- не ведет статистику.

Перейдем к рассмотрению сервисов, которые позволяют частично решить задачу, а именно платежных провайдеров — сервисов, которые не предоставляют возможность доставки цифровых товаров, но позволяют проводить онлайн-платежи. Ниже приведен список рассматриваемых платежных провайдеров:

- ЮKassa [23],
- Robokassa [21],
- Т-Касса [22],
- сервис интернет-эквайринга от Альфа-Банка [18],
- Payselection [24].

Все перечисленные решения схожи по функциональности и условиям подключения и предоставляют API со следующими возможностями:

- создание платежной ссылки, по которой можно совершить оплату различными способами с рублевого счета;
- правка HTTP-уведомления (вебхука) при изменении статуса платежа;
- получение статуса платежа по его идентификатору.

Из рассмотренных платежных провайдеров для использования в первой версии разрабатываемой системы выбран ЮKassa за счет более простых и удобных по сравнению с аналогами API и документации для разработчиков, а также простоты открытия тестового магазина, что позволит быстрее внедрить и протестировать систему.

2.3. Выводы

С учетом проведенного анализа предметной области и пожеланий заказчика и недостаточной гибкости готовых решений, принято решение разработать собственную систему, которая будет реализовывать продажу цифровых товаров, то есть прием платежей и доставку товаров клиентам после оплаты. При этом для проведения платежей будет использоваться внешний платежный провайдер.

3. Архитектура

Данный раздел посвящен описанию системы в сборе и включает в себя описание групп пользователей и их варианты использования системы, требования к системе, используемые технологии, структуру и взаимодействие компонентов, интеграции с другими сервисами, описание основных сценариев использования системы.

На рисунке 1 представлены сценарии использования системы в виде диаграммы вариантов использования UML [28]. У реализуемой системы управления доступом, названной “GoPay”, есть три группы пользователей, ниже представлено краткое описание их ролей и целей использования системы.

- Администратор — человек или сервис со стороны компании, который создает платежные ссылки и отправляет их клиенту, работает со статистикой использования сервиса.
- Клиент — клиент компании, оплачивает цифровой товар по ссылке, а затем получает его удобным способом.
- Платежный провайдер — внешний сервис для осуществления электронных платежей, отправляет уведомления об изменении статусов платежей.

Рассмотрим обобщенно требования к системе, сформированные из потребностей заказчика и анализа предметной области и групп пользователей.

Были сформулированы следующие функциональные требования к системе.

- Должна быть возможность для администратора создать внутреннюю ссылку сервиса, которая будет связана с клиентом, платежом в платежном сервисе и продаваемым цифровым товаром.
- Система должна предоставлять данные о платежах для администратора.



Рис. 1: Основные сценарии работы пользователей с системой

- Система должна позволять клиенту оплатить товар по ссылке через Систему Быстрых Платежей (СБП) [38] или введя реквизиты банковской карты, используя карту российского банка с рублевым счетом.
- Система должна реализовывать способ доставки товара по ссылке, то есть после оплаты клиент может перейти по ссылке и посмотреть купленный файл, а при отсутствии оплаты доступ должен быть заблокирован.
- Система должна принимать уведомления об изменении статуса платежа от платежного сервиса и предоставлять клиенту доступ к купленному товару в случае успешной оплаты.

Ниже представлены нефункциональные требования к системе.

- Необходимо использовать надежное энергонезависимое хранилище для данных о пользователях и их платежах.
- Система должна полноценно работать как самостоятельно, так и позволять интегрироваться с другими сервисами через API.
- Архитектура системы должна позволять заменить файловое хранилище, базу данных и платежный сервис на подобные за счет

реализации модуля для этого без изменения существующего кода.

- Архитектура системы должна позволять без изменения существующего кода реализовать новые способы доставки цифрового товара, такие как по электронной почте или через личные сообщения в мессенджере помимо доставки по ссылке.

Перейдем к обоснованию выбранных технологий для реализации решения.

- Язык программирования Go [31] для реализации серверной логики и Telegram-бота [30] для администратора — высокопроизводительный язык с богатым набором стандартных инструментов и большим сообществом разработчиков, наиболее часто используется для реализации микросервисов, также на нем написаны все сервисы компании.
- Веб-фреймворк Echo [19] для Go — высокопроизводительный фреймворк, предоставляющий гибкую маршрутизацию и валидацию запросов, включает в себя набор готовых middleware и поддерживает добавление новых.
- Встроенная СУБД VoltDB [7] с поддержкой требований ACID для хранения данных о клиентах и их платежах — предоставляет базу данных ключ-значение для языка Go, оптимизированную для чтения. Выбрана, основываясь на простоте интеграции для первой версии продукта за счет того, что не нужно поднимать сервер и можно хранить данные разной схемы, а также на потенциальном использовании как кэша перед другой СУБД в дальнейшем.

Также была выполнена интеграция с двумя внешними системами:

- платежным провайдером ЮKassa [23], позволяющим создавать ссылки для оплаты и принимать по ним оплату в рублях по Системе Быстрых Платежей (СБП) и реквизитам банковских карт,

сервис отправляет уведомления об изменении статусов платежей по HTTP;

- файловым хранилищем MinIO [25] — объектным хранилищем, совместимым с Amazon S3 [16], которое используется в компании для хранения файлов с отчетами о зондировании почвы.

Провайдер ЮKassa выбран за счет более простых и удобных, чем у других аналогичных сервисов, API и документации для разработчиков, простоты в открытии тестового магазина, что позволит быстрее внедрить и протестировать первую версию системы.

Перейдем к структуре системы. На рисунке 2 изображена компонентная структура системы в виде диаграммы компонентов UML. Серверная часть реализована по принципам «чистой» [13] архитектуры и предоставляет REST [3] API для пользователей, описанных выше.

Payment Handler отвечает за обработку основных HTTP-запросов к API системы. Платежный провайдер ЮKassa отправляет запросы напрямую. Клиент отправляет запросы через браузер Clients Web Browser, API спроектирован таким образом, что нет необходимости в графическом интерфейсе. Администратор работает с системой через Telegram-бота Admins Telegram Bot, для которого реализована библиотека Admins GoPay Client Library — HTTP-клиент для GoPay API на Go. Библиотека реализована с применением паттернов проектирования [2] «фасад» для доступа ко всем запросам из одного объекта и «строитель» для пошагового создания запроса, что является распространенным подходом при реализации HTTP-клиентов и особенно удобно для использования в Telegram-боте, так как данные, необходимые для запроса, в него поступают поэтапно. Обработчики из Payment Handler и их функциональность описаны ниже.

- POST /api/payments — создание платежа.
- GET /api/payments — получение статусов всех платежей.
- GET /api/payments/:id — получения статуса платежа по id.

- GET /api/:id — перенаправление на оплату (ЮKassa) или на получение файла (Delivery Handler) в зависимости от статуса платежа.
- POST /api/checkout — обработка уведомления об изменении статуса платежа.

Delivery Handler отвечает за доставку цифрового товара по запросу, данный запрос не отправляются напрямую, а через редирект из Payment Handler. На данный момент реализован обработчик GET /api/files/:id — получение файла из файлового хранилища MinIO по id. Для работы с MinIO реализован клиент, удовлетворяющий интерфейсу File storage.

Payment Manager на диаграмме представляет из себя слой бизнес-логики и взаимодействует с СУБД BoltDB через объект репозитория, реализующего интерфейс Payment storage, с платежным провайдером ЮKassa через объекта клиента, реализующего интерфейс Payment provider, и использует LinkGenerator для создания уникального идентификатора платежа и внутренней ссылки вида /api/:id, для перехода по которой реализован обработчик, описанный выше.

Система спроектирована таким образом, что можно без проблем заменить компоненты MinIO, LinkGenerator, BoltDB и ЮKassa на аналогичные, реализовав дополнительный модуль, удовлетворяющий соответствующему интерфейсу. Помимо этого, Delivery Handler может быть заменен на любой другой обработчик GET HTTP-запроса без тела и дополнительных заголовков, что позволяет реализовать гибкую логику доставки товара.

Далее рассмотрим подробнее основные успешные сценарии работы с системой со стороны клиента и администратора.

На рисунке 3 изображена диаграмма последовательности UML для основного успешного сценария работы администратора с системой, а именно создания платежа. Сначала администратор использует соответствующую команду в Telegram-боте, далее бот запрашивает необходимые данные для создания платежа, администратор вводит данные, бот валидирует их и отправляет HTTP-запрос к серверу GoPay. В рамках

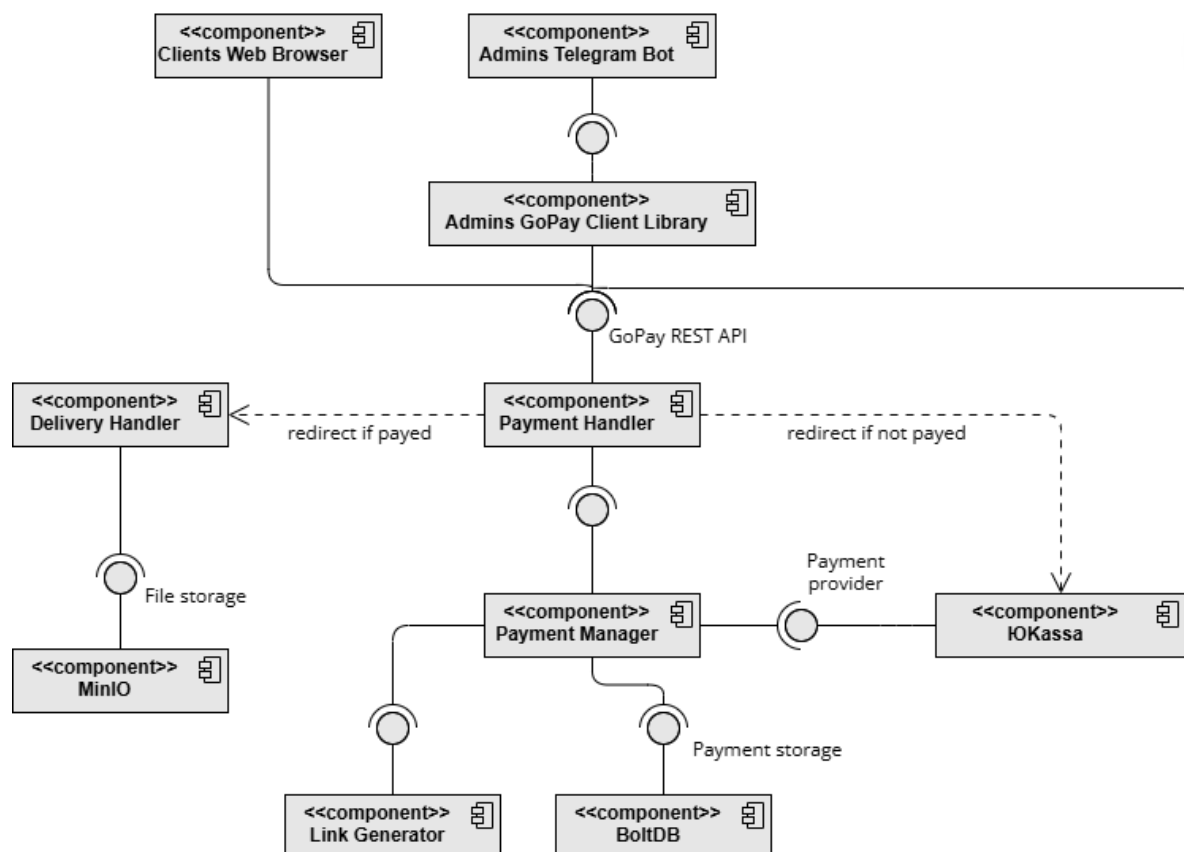


Рис. 2: Структура системы

выполнения запроса на сервере генерируется внутренний идентификатор платежа и ссылка, по которой затем можно будет оплатить и получить цифровой товар. Затем создается ссылка для оплаты в сервисе ЮKassa и все данные сохраняются в BoltDB, то есть создаются записи, связывающие данные клиента, внешнюю ссылку для оплаты и данные о предоставляемом цифровом товаре с уникальным идентификатором платежа в системе. После этого система отправляет в ответе внутреннюю платежную ссылку.

На рисунке 4 изображена диаграмма последовательности UML, представляющая основной сценарий работы клиента с системой, а именно, совершение оплаты и получения доступа к оплаченному цифровому товару, что возможно сделать только по ссылке за счет перенаправления на другую ссылку в зависимости от статуса платежа. Изначально клиент получает внутреннюю ссылку системы от администратора и переходит по ней, то есть делает запрос к серверу GoPay, так как товар

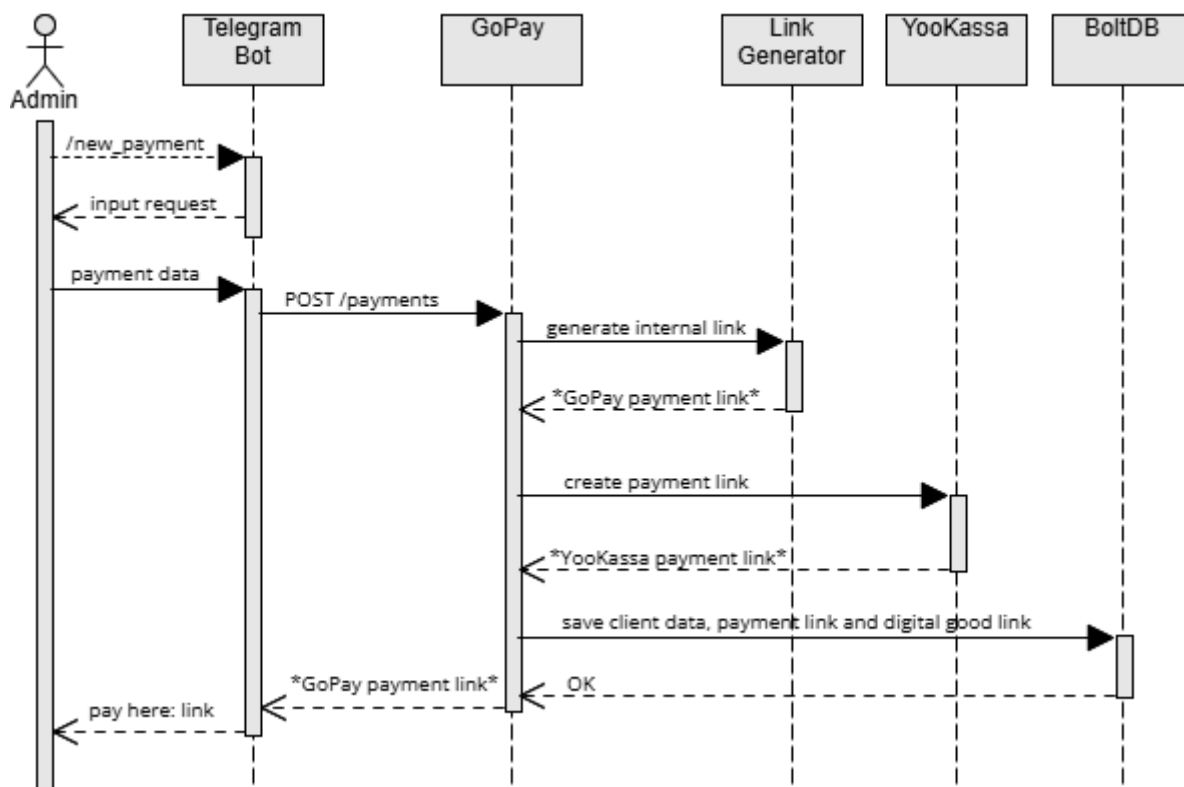


Рис. 3: Главный сценарий работы администратора с системой

еще не оплачен, то из BoltDB сервер получает ссылку на страницу для оплаты в ЮKassa и делает туда редирект. Затем клиент на этой странице совершает платеж и ЮKassa посредством HTTP-запроса (вебхука) уведомляет об оплате сервер GoPay, в системе обновляется статус платежа и подменяется ссылка для редиректа. Теперь, повторно перейдя по ссылке, полученной от администратора, клиент перенаправляется уже не на оплату, а на внутренний обработчик GoPay, который получает файл с отчетом из файлового хранилища MinIO и отображает его для клиента.

В связи с тем, что переход по ссылке для оплаты и получения цифрового товара является самым частым и важным запросом к системе, схема хранения данных спроектирована таким образом, что ссылка для редиректа хранится отдельно. Это позволяет ускорить запрос за счет того, что не нужно проверять статус платежа перед редиректом, достаточно получить ссылку из базы данных и сразу перенаправить на нее, так как технически редирект на оплату и получение товара ничем не отличается.

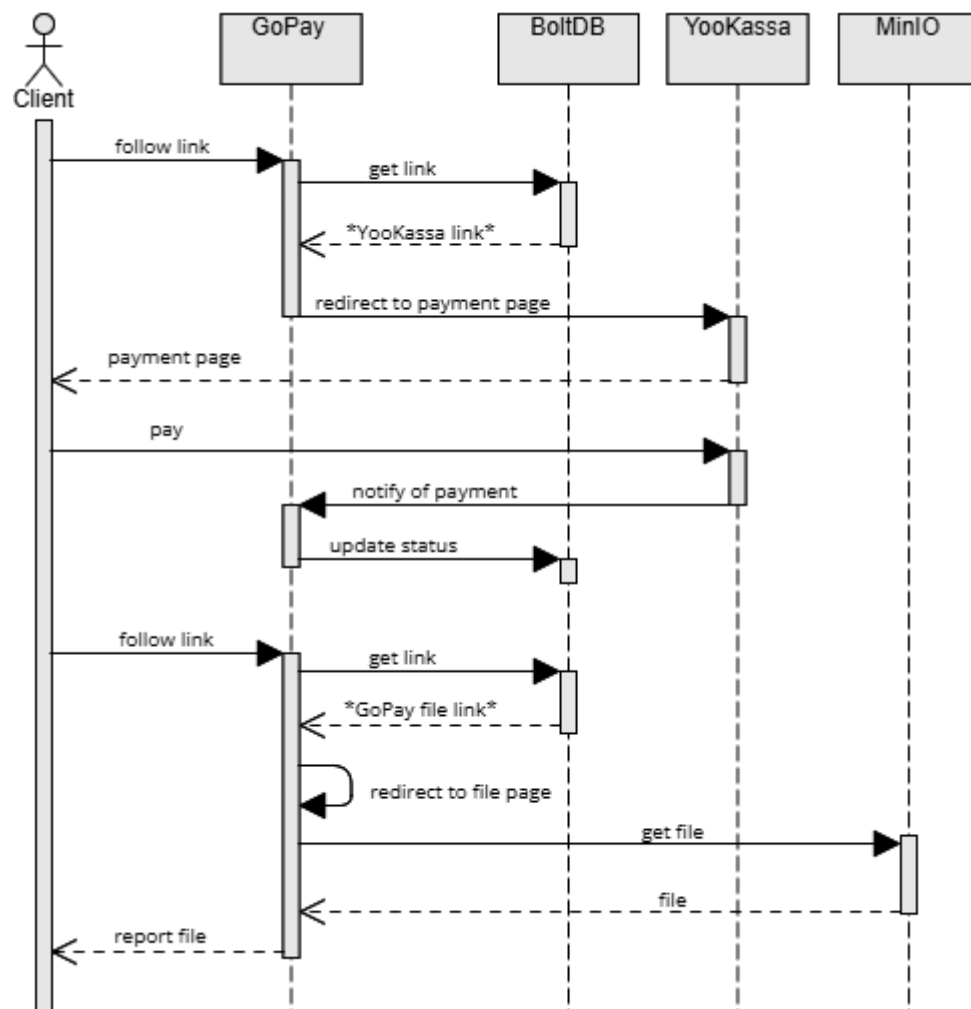


Рис. 4: Главный сценарий работы клиента с системой

4. Особенности реализации

Данный раздел включает в себя описание некоторых деталей реализации хранения данных, обработки запросов на сервере, пользовательского интерфейса и описание вариантов конфигурации и использования представленного решения.

4.1. Хранение данных

В базе данных, реализованной с использованием встроенной СУБД BoltDB (библиотека для языка Go), данные хранятся в коллекциях (bucket) формата ключ-значение, оптимизированных для чтения. В реализованной системе данные разбиты на три коллекции.

- Payments содержит основные данные о заказах в системе: данные о клиенте и платеже, ссылки для оплаты и доставки цифрового товара.
- Links хранит ссылки для редиректа в зависимости от статусов платежей: на страницу доставки цифрового товара, если он оплачен, или на страницу оплаты внешнего платежного сервиса, если оплата еще не была совершена.
- Templates содержит шаблоны платежей, включающие валюту, сумму, описание платежа и вариант доставки, которые можно использовать для минимизации ошибок и упрощения создания похожих платежей.

Получение ссылки из коллекции Links — это наиболее частый и важный для клиента запрос, так как создание платежа и изменение его статуса выполняется по одному разу при создании платежа администратором и при обработке уведомления от платежного провайдера соответственно, а получение ссылки для редиректа происходит каждый раз, когда клиент переходит по внутренней ссылке сервиса. В связи с этим, для оптимизации самого частого запроса ссылка хранится отдельно и подменяется при изменении статуса платежа.

Ключом в коллекциях Payments и Links является идентификатор заказа в системе в формате Universally Unique Identifier версии 4 (UUIDv4) [40], что гарантирует практически нулевую вероятность появления дубликатов. Данные идентификаторы генерируются при создании платежа и также используются при создании внутренней платежной ссылки для клиента.

4.2. Серверная часть

Серверная часть системы представляет из себя сервис на языке Go с использованием веб-фреймворка Echo. API сервиса включает в себя ряд обработчиков HTTP-запросов, описанных в разделе 3. Также для реализованного API из аннотаций в коде генерируется Swagger-документация в формате OpenAPI [39] с помощью инструмента Swag [9] для Go.

Каждый запрос к серверу выполняется в несколько этапов:

- маршрутизация,
- выполнение логики промежуточных обработчиков (middleware),
- валидация входных данных,
- выполнение бизнес-логики,
- логирование.

Маршрутизация выполняется за счет функциональности стандартного роутера из фреймворка Echo. Обработчики для основных запросов к API объединены в группу /api, а для доступа к интерактивной документации через Swagger UI зарегистрирован отдельный обработчик, который также генерируется инструментом Swag. При попытке отправки запроса по маршруту, на который не зарегистрирован ни один обработчик, будет возвращаться ответ со статус-кодом 404 (Not Found).

Промежуточные обработчики (middleware), выполняемые при каждом запросе, перечислены ниже.

- Recover — стандартный middleware из Echo, отлавливающий panic (встроенная функция в Go, сигнализирующая о невозможности продолжить выполнение программы из-за возникшей ошибки) для того, чтобы не завершать работу всего приложения из-за ошибки, а логировать ее.
- RequestID — стандартный middleware, создающий уникальный идентификатор для каждого запроса, который используется для определения, какие записи в логах связаны с одним запросом.
- Slogecho — совместимый с Echo middleware для логирования запросов к серверу с помощью логгера Slog из стандартной библиотеки Go.

Входные данные валидируются с помощью совместимого с Echo инструмента Validator [6], включающего набор готовых функций для валидации (таких, как email и uuid). Также библиотека позволяет реализовать собственные реализации для кастомных типов, были реализованы функции, валидирующие внутренний идентификатор и статус платежа. В случае, когда валидация завершилась с ошибкой, отправляется ответ со статусом 400 (Bad Request).

После валидации выполняется бизнес-логика приложения. Если во время выполнения возникла ошибка, то сервер отправляет ответ со статусом 500 (Internal Server Error). В случае успеха отправляется ответ с кодом 200 (OK) или 307 (Temporary Redirect) при переходе клиента по ссылке для оплаты и получения цифрового товара, полученной от администратора.

Помимо логирования входящих запросов и результатов их выполнения в middleware, также дополнительно логируется ход выполнения запросов в системе, то есть промежуточные результаты и возникающие ошибки. Все записи производятся с использованием логгера Slog для структурированного логирования, что удобно для дальнейшего анализа логов.

4.3. Клиентская часть

Так как API спроектирован таким образом, что клиенты работают с сервером напрямую из браузера, то есть браузер отправляет GET-запрос при переходе по ссылке, то в графическом интерфейсе нет необходимости. Однако API для администратора сложнее, поэтому для управления сервисом реализован интерфейс в виде Telegram-бота.

Начнем с технических деталей реализации. Для взаимодействия бота с API системы реализована библиотека HTTP-клиент для языка Go, бот также реализован на Go с использованием библиотеки Telego [5] для работы с Telegram API.

Бот получает новые сообщения пользователей от сервера Telegram с помощью Long Polling, то есть метода получения новых данных за счет периодической отправки HTTP-запросов к API, которые удерживаются сервером до момента появления обновлений или истечения таймаута.

Управление состоянием пользователей организовано за счет использования простой реализации конечного автомата на основе хэш-таблицы. Для каждого пользователя хранятся введенные им данные и этап, на котором он находится, что важно для пошагового выполнения команд.

Для предоставления доступа к функциям бота используется подход Whitelist-аутентификации по идентификатору пользователя (User ID) в Telegram, то есть авторизация в боте происходит, если в списке пользователей, которые наделены доступом к функциям бота, содержится User ID текущего пользователя. Для этого при запуске бота передается данный список, состоящий из User ID администраторов.

Перейдем к рассмотрению функциональности бота и взаимодействию пользователя с ним. После введение команды /start и успешной авторизации администратором, бот отправляет сообщение со списком доступных команд. Данное сообщение изображено на рисунке 5. Бот умеет выполнять следующие команды.

- /new_payment — команда для создания платежа.
- /all_payment — команда для получения статусов всех платежей.

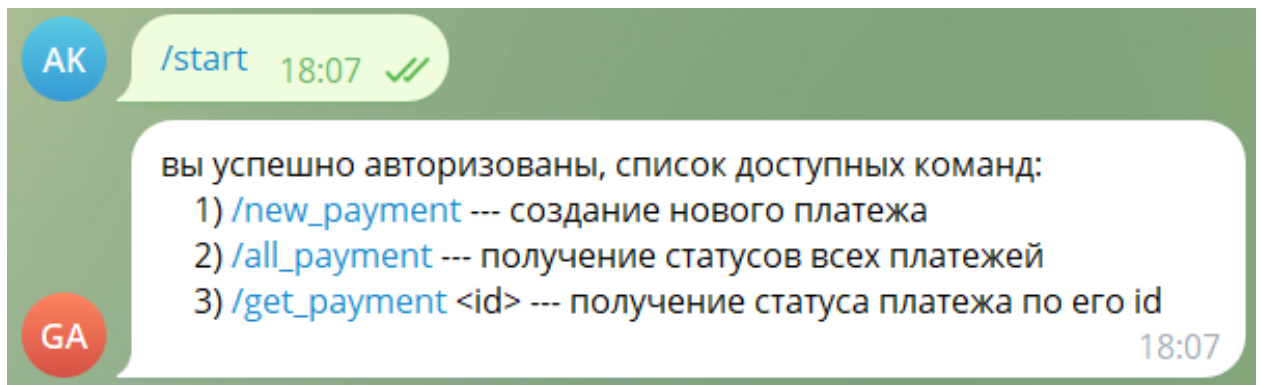


Рис. 5: Список доступных команд Telegram-бота

- `/get_payment <id>` — команда для получения статуса платежа по его идентификатору `id`.

Рассмотрим использование Telegram-бота для управления системой на примере создания платежа. На рисунке 6 представлен процесс создания платежа в системе. После введения администратором команды `/new_payment`, бот поэтапно запрашивает необходимые для создания платежа данные: сумму, валюту и описание. После введения всех данных пользователь бота отправляет подтверждение, нажав на всплывающую кнопку или написав «да». После подтверждения бот совершает соответствующий запрос к API и отправляет пользователю результат в виде созданной платежной ссылки.

Также в боте реализована валидация входных данных. На рисунке 7 представлена попытка ввести данные в некорректном формате на примере валюты платежа. Бот отправляет пользователю описание ошибки и ожидает повторное введение соответствующих данных.

4.4. Запуск и использование проекта

Для целей упрощения сборки, тестирования, статического анализа кода и генерирования документации, а также установки необходимых инструментов для этого, используется утилита GNU Make [14].

Конфигурация запуска API и Telegram-бота реализована с использованием библиотеки CLI [4]. Данная библиотека позволяет использовать для конфигурации приложения несколько источников. В реализован-

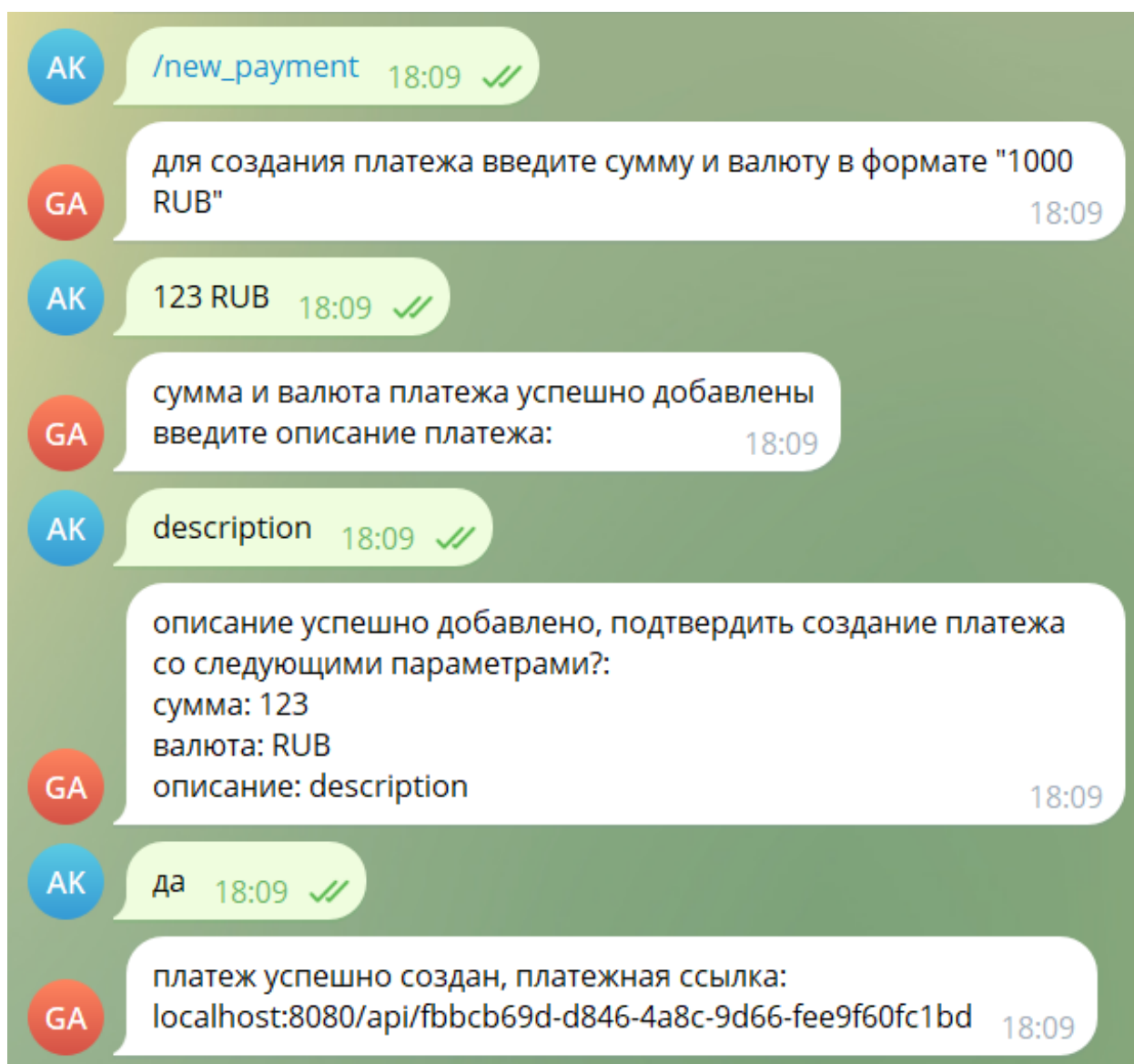


Рис. 6: Создание платежа в системе с помощью Telegram-бота

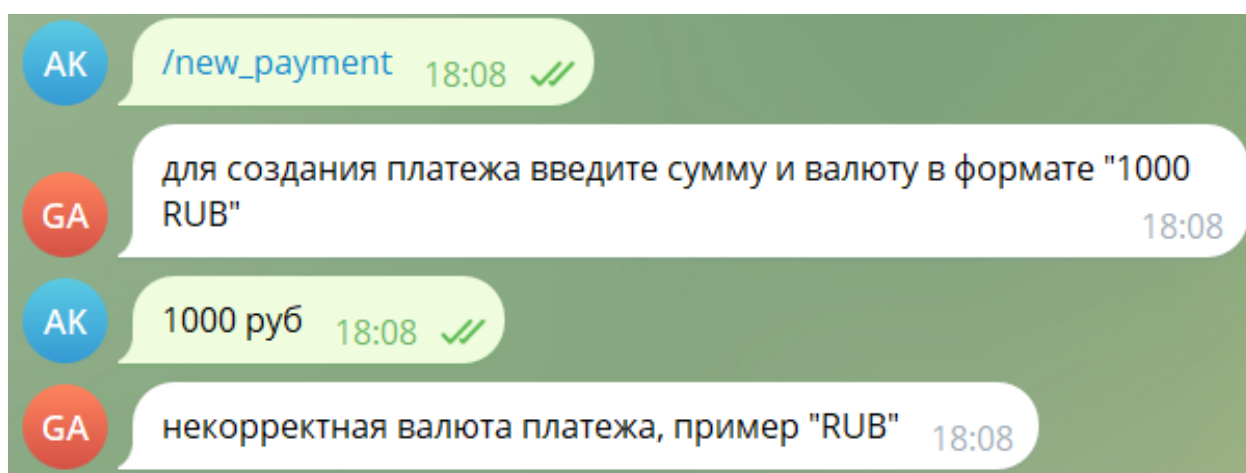


Рис. 7: Валидация входных данных в Telegram-боте

ном приложении используются флаги Go и переменные окружения.

Рассмотрим доступные параметры конфигурации API:

- окружение (dev/prod);
- хост для запуска HTTP-сервера;
- порт для HTTP-сервера;
- путь к файлу базы данных (БД);
- таймаут подключения к БД;
- URL обработчика уведомлений от ЮKassa;
- идентификатор магазина в ЮKassa;
- секретный токен API ЮKassa;

Для конфигурации бота доступны следующие параметры:

- базовый URL сервера;
- секретный токен API бота;
- список идентификаторов администраторов в Telegram;

Реализация системы доступна в открытом репозитории на GitHub [33] и распространяется под открытой лицензией, что позволяет любому разработчику доработать решение под свои задачи или использовать в качестве источника идей для своего решения.

Помимо этого, реализована библиотека, которая опубликована в официальном репозитории пакетов для Go [20] и доступна для установки с помощью стандартного пакетного менеджера Go. Ниже представлен API библиотеки.

- Объект `PaymentManager`, реализующий основные сценарии бизнес-логики приложения. Используется в реализации API системы.

- Объект `AdminClient`, предоставляющий HTTP-клиент для API системы. Используется в реализации Telegram-бота для администратора.

5. Тестирование и внедрение

Для основных сценариев бизнес-логики было выполнено table-driven [1] модульное тестирование¹ с использованием библиотеки Testify [8], совместимой со стандартными инструментами тестирования Go кода, для стандартных проверок в тестовых сценариях.

Для того, чтобы протестировать компонент отдельно от зависимостей, использовался фреймворк GoMock [11], позволяющий с помощью стандартного инструмента go generate [12] создавать объекты-заглушки, реализующие интерфейсы зависимостей тестируемого модуля, для которых можно настраивать поведение, ожидаемое от конкретных реализаций.

Для обеспечения актуальности и стабильности кодовой базы внедрена практика непрерывной интеграции (Continuous Integration, CI) на основе GitHub Actions [17]. На каждый pull request или push в основную ветку main активируется процесс, в рамках которого производится сборка серверной и клиентской частей приложения, а также запускаются модульные тесты и линтеры с помощью инструмента Golangci-lint [26].

На данный момент система частично внедрена для продажи отчетов о зондировании почвы в рамках геодезического проекта компании «Стройматик». Командой менеджеров по продажам, которые являются администраторами в терминах, введенных ранее, было выполнено «ручное» тестирование системы. Была также проведена полная апробация функциональности системы как со стороны администратора, так и со стороны клиента.

В целом, получена положительная обратная связь, реализованный продукт справляется со своей основной задачей — ускорение процесса продажи отчетов. И это важно как для компании, так и для клиентов. Сейчас система для оплаты и доставки цифровых товаров тестируется реальными пользователями. После успешного тестирования и исправле-

¹Table-driven тестирование — подход к тестированию, при котором тестовые сценарии оформляются в виде коллекции и для каждого из сценариев выполняется одинаковая логика тестирования, что уменьшает дублирование кода и упрощает добавление новых тестовых сценариев.

ния недочетов планируется расширение функциональности и внедрение системы в другие проекты и команды компании, которые тоже продают цифровые товары (такие, как, например, план застройки).

По первым результатам использования системы был намечен следующий план развития.

- Реализовать динамические шаблоны платежей, которые будут автоматически по данным клиента определять, какой именно отчет ему продавать, это позволит упростить процесс создания платежей, так как не нужно будет передавать ссылку, по которой можно получить файл.
- Вести больше статистики об использовании сервиса, как администраторами, так и клиентами. Собирать данные о том, когда была произведена оплата и доставка, когда и кем был создан платеж.

Заключение

В рамках работы были получены следующие результаты.

1. Сформулированы функциональные и нефункциональные требования к системе с учетом потребностей заказчика и ограничений, выявленных при анализе предметной области.
2. Спроектирована система с серверной частью в «чистом» архитектурном стиле и API в архитектурном стиле REST с учетом требований, а также необходимости интеграции с внешним платежным провайдером и файловым хранилищем.
3. Разработана система, которая интегрирована с платежным провайдером ЮKassa и файловым хранилищем MinIO; серверная часть написана на языке Go с использованием фреймворка Echo, база данных реализована с использованием СУБД BoltDB; пользовательский интерфейс реализован в виде Telegram-бота.
4. Реализовано модульное тестирование основных сценариев бизнес-логики системы.
5. Система внедрена в процесс выполнения услуги по анализу почвы в рамках геодезического проекта компании «Стройматик».

Код реализации доступен в репозитории на GitHub [33].

Список литературы

- [1] Cheney Dave. Prefer table driven tests. — 2019. — URL: <https://dave.cheney.net/2019/05/07/prefer-table-driven-tests> (дата обращения: 26 мая 2025 г.).
- [2] Contreras Mario Castro. Go Design Patterns. — Packt Publishing, 2017. — ISBN: 9781786466204. — URL: <https://www.packtpub.com/product/go-design-patterns/9781786466204>.
- [3] Fielding Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures : Ph.D. thesis / Roy Thomas Fielding ; University of California, Irvine. — Irvine, CA, USA, 2000. — URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
- [4] GitHub-репозиторий библиотеки CLI для Go. — URL: <https://github.com/urfave/cli> (дата обращения: 26 мая 2025 г.).
- [5] GitHub-репозиторий библиотеки Telego для работы с Telegram API. — URL: <https://github.com/mymmrac/telego> (дата обращения: 26 мая 2025 г.).
- [6] GitHub-репозиторий библиотеки Validator. — URL: <https://github.com/go-playground/validator> (дата обращения: 26 мая 2025 г.).
- [7] GitHub-репозиторий встроенной СУБД BoltDB для языка Go. — URL: <https://github.com/etcd-io/bbolt> (дата обращения: 30 апреля 2025 г.).
- [8] GitHub-репозиторий инструмента Testify для тестирования Go-кода. — URL: <https://github.com/stretchr/testify> (дата обращения: 26 мая 2025 г.).

- [9] GitHub-репозиторий инструмента для генерации Swagger-документации. — URL: <https://github.com/swagger/swag> (дата обращения: 26 мая 2025 г.).
- [10] GitHub-репозиторий с исходным кодом решения для оплаты и доставки файлов по электронной почте. — URL: <https://github.com/mneveroff/stripe-link-file-delivery> (дата обращения: 16 мая 2025 г.).
- [11] GitHub-репозиторий фреймворка GoMock для тестирования Go-кода. — URL: <https://github.com/uber-go/mock> (дата обращения: 26 мая 2025 г.).
- [12] The Go Blog, Generating code. — URL: <https://go.dev/blog/generate> (дата обращения: 26 мая 2025 г.).
- [13] Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Robert C. Martin Series. — Pearson Education, 2017. — ISBN: 9780134494326. — URL: <https://books.google.ru/books?id=uGE1DwAAQBAJ>.
- [14] Prakash Danish. Using Makefile(s) for Go. — 2019. — URL: <https://danishpraka.sh/posts/using-makefiles-for-go/>. 26 мая 2025 г..
- [15] Гибкое решение для электронной коммерции Foxy. — URL: <https://www.foxy.io/> (дата обращения: 16 мая 2025 г.).
- [16] Документация Amazon Simple Storage Service API. — URL: <https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html> (дата обращения: 30 апреля 2025 г.).
- [17] Документация GitHub Actions. — URL: <https://docs.github.com/actions> (дата обращения: 26 мая 2025 г.).
- [18] Документация сервиса Альфа-Банка для интернет-эквайринга. — URL: <https://alfabank.ru/sme/payservice/internet-acquiring/docs/connection-options/api/> (дата обращения: 16 мая 2025 г.).

- [19] Документация веб-фреймворка Echo для языка Go. — URL: <https://echo.labstack.com/> (дата обращения: 30 апреля 2025 г.).
- [20] Документация пакета реализованной системы GoPay в официальном репозитории пакетов для Go. — URL: <https://pkg.go.dev/github.com/Anton-Kraev/gopay> (дата обращения: 26 мая 2025 г.).
- [21] Документация платежного провайдера Robokassa. — URL: <https://docs.robokassa.ru/> (дата обращения: 16 мая 2025 г.).
- [22] Документация платежного провайдера Т-Касса. — URL: <https://www.tbank.ru/kassa/dev/payments/> (дата обращения: 16 мая 2025 г.).
- [23] Документация платежного провайдера ЮKassa. — URL: <https://yookassa.ru/developers> (дата обращения: 16 мая 2025 г.).
- [24] Документация платежного сервиса Payselection. — URL: <https://api.payselection.com/> (дата обращения: 16 мая 2025 г.).
- [25] Документация файлового хранилища MinIO. — URL: <https://min.io/docs> (дата обращения: 30 апреля 2025 г.).
- [26] Инструмент для запуска линтеров Golangci-lint. — URL: <https://golangci-lint.run/> (дата обращения: 26 мая 2025 г.).
- [27] Категория «цифровые товары» на маркетплейсе Ozon. — URL: <https://www.ozon.ru/category/tsifrovye-tovary-32056/> (дата обращения: 16 мая 2025 г.).
- [28] Кознов Д. В. Основы визуального моделирования. Основы информационных технологий. — Москва : Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. — С. 248.

- [29] Конструктор Телеграм-ботов для продажи цифровых товаров. — URL: <https://unionbot.ru/bot-tovarov.html> (дата обращения: 16 мая 2025 г.).
- [30] Официальная документация Telegram по созданию ботов. — URL: <https://core.telegram.org/bots> (дата обращения: 30 апреля 2025 г.).
- [31] Официальная документация языка программирования Go. — URL: <https://golang.org/doc/> (дата обращения: 30 апреля 2025 г.).
- [32] Платформа для продажи онлайн-курсов АвтоВебОфис. — URL: <https://autoweboffice.com/> (дата обращения: 16 мая 2025 г.).
- [33] Репозиторий с кодом реализации системы для управления доступом к цифровым товарам. — URL: <https://github.com/Anton-Kraev/gopay> (дата обращения: 13 апреля 2025 г.).
- [34] Сайт компании «Стройматик», описание услуги динамического зондирования грунта. — URL: <https://stroimatic.ru/uslugi/zondirovanie-grunta/> (дата обращения: 13 апреля 2025 г.).
- [35] Сайт программной платформы Node.js. — URL: <https://nodejs.org/> (дата обращения: 16 мая 2025 г.).
- [36] Сервис для оплаты и доставки цифровых товаров Рокуро. — URL: <https://pokupo.ru/digital-goods> (дата обращения: 16 мая 2025 г.).
- [37] Сервис для приема онлайн-платежей Stripe. — URL: <https://stripe.com/> (дата обращения: 16 мая 2025 г.).
- [38] Система быстрых платежей ЦБ РФ. — URL: <https://sbp.nspk.ru/> (дата обращения: 30 апреля 2025 г.).
- [39] Спецификация OpenAPI. — URL: <https://swagger.io/specification/> (дата обращения: 26 мая 2025 г.).

- [40] Спецификация UUID, документ RFC9562. — URL: <https://datatracker.ietf.org/doc/html/rfc9562> (дата обращения: 26 мая 2025 г.).