МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Информационная система кондитерской организации на основе микросервисной схемы взаимодействия

Студент _____ Алеева Султания Рашидовна
(подпись, дата) (ФИО полностью)

Руководитель _____ Такташкин Д.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, 2025

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«20» декабря 2024 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент Алеева Султания Рашидовна гр. 21ВИ1 факультета ВТ

2.Тема работы Информационная система кондитерской организации на основе микросервисной схемы взаимодействия

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

3. Руководитель работы к.т.н. доцент, Такташкин Д.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: автоматизация бизнес-процессов кондитерской организации
Среда выполнения: ОС Windows
Языковые средства разработки: Python, JavaScript
Инструментальные средства: PyCharm
Функции системы:
1. фиксирование изготовления кондитерских изделий;
2. фиксирование использования ингредиентов;
3. создание заказа на поставку ингредиентов;
4. фиксирование приема поставки ингредиентов;
5. списание просроченных кондитерских изделий и ингредиентов;
6. фиксирование продаж кондитерских изделий.

5. Перечень подлежащих разработке вопросов

1. Анализ предметной области и постановка задачи;
2. Планирование разработки оценка бюджета;
3. Разработка требований и проектирование;
4. Реализация;
5. Контроль качества программного обеспечения;

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания "20" декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Л.Р.Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 85 листов, 40 рисунков, 16 использованных источников, 5 таблиц и 3 приложения.

КОНДИТЕРСКАЯ ОРГАНИЗАЦИЯ, ИНФОРМАЦИЯ СИСТЕМА, МИКРОСЕРВИСЫ, WEB-ПРИЛОЖЕНИЕ, PYTHON, FASTAPI, SQLALCHEMY, UML, JINJA2, ASYNCPG.

Объектом исследования является разработка приложения на основе микросервисной схемы взаимодействия.

Целью выпускной квалификационной работы является разработка информационной системы кондитерской организации на основе микросервисной схемы взаимодействия.

Технологии разработки: языки программирования Python, JavaScript, СУБД PostgreSQL, среда разработки PyCharm.

Результат работы: проведены анализ предметной области и требований к программному обеспечению, выполнено проектирование, на основе которого осуществлена реализация приложения. Разработанная информационная система была отлажена и протестирована.

Содержание

Введение 6

1 Постановка задачи и анализ требований 7

1.1 Анализ предметной области 7

1.2 Анализ аналогичных программных продуктов 8

1.3 Анализ требований к разрабатываемой системе 12

1.4 Постановка задачи 15

2 Проектирование информационной системы 17

2.1 Обзор инструментальных средств проектирования и разработки 17

2.2 Проектирование программных средств разрабатываемой системы 19

2.3 Проектирование структур данных 25

2.4 Планирование разработки системы и оценка бюджета 28

3 Разработка программных средств 32

3.1 Разработка серверной части 32

3.2 Разработка клиентской части 39

4 Тестирование программных средств 45

4.1 Функциональное тестирование 45

4.2 Тестирование API 49

Заключение 52

Список использованных источников 53

Приложение А (рекомендуемое) Графическая часть 55

Приложение Б (обязательное) Листинг программы 61

Приложение В (рекомендуемое) Результаты тестирования 82

Введение

Без информационных технологий не обходится ни одна профессиональная область деятельности. Каждой организации от крупного предприятия до малого бизнеса необходима информационная система, автоматизирующая бизнес-процессы и обработку сопутствующих им данных. Внедрение информационной системы позволяет снизить человеческий фактор, увеличить скорость и повысить точность выполнения задач.

Большинство современных информационных систем построены на микросервисной архитектуре. При таком подходе приложение декомпозируется на ряд микросервисов – независимых, слабо связанных подсистем, каждая из которых выполняет определенную бизнес-функцию.

Спрос на кондитерские изделия всегда высок, ведь они являются как атрибутами торжества, так и элементами повседневных чаепитий. В последние годы стало популярным изготовление кондитерских изделий на дому или ведение собственной кондитерской организации. Подобная деятельность не лишена потребности в автоматизации учетных задач.

1 Постановка задачи и анализ требований

1.1 Анализ предметной области

Рассмотрим деятельность небольшой кондитерской организации.

Одним из основных процессов кондитерской организации является изготовление продукции. Кондитерские изделия являются довольно скоропортящимся продуктом, каждое наименование имеет свой срок годности, а для каждого изделия или партии должны быть указаны дата и время производства и истечения срока годности. Необходимо вести учет готовых кондитерских изделий на складе.

Для производства кондитерских изделий необходимы ингредиенты, которые так же имеют срок годности. Ингредиенты закупаются у поставщиков. Каждый поставщик предоставляет определенный набор ингредиентов. Закупка ингредиентов осуществляется посредством формирования заказа на поставку. Для каждой поставки формируется список позиций. При приеме поставки фиксируются полученные ингредиенты. Необходимо вести учет ингредиентов на складе.

Необходимо следить за сроками годности ингредиентов и кондитерских изделий на складе и списывать просрочки, чтобы избежать их использования и продажи.

При реализации готовой продукции необходимо фиксировать информацию о заказах и их позициях.

Таким образом, в данной предметной области можно выделить ряд ключевых информационных объектов:

- кондитерское изделие;
- ингредиент;
- поставка;
- поставщик;
- заказ.

1.2 Анализ аналогичных программных продуктов

Существует множество различных программных продуктов, используемых кондитерскими организациями. На рынке представлены следующие виды систем:

– Customer Relationship Management (CRM) – система управления взаимоотношениями с клиентами;

– Enterprise Resource Planning (ERP) – система планирования ресурсов предприятия;

– Point Of Sale (POS) – комплекс аппаратного и программного обеспечения для обработки продаж, управления запасами и обработки данных клиентов.

Большинство представленных рынке программных продуктов не являются специализированными именно для кондитерского бизнеса.

Далее приведен обзор некоторых программных продуктов, используемых кондитерскими организациями.

«1С:Хлебобулочное и кондитерское производство 2. Модуль для 1С:ERP 2» – отраслевая программа для ведения комплексного учета хозяйственной деятельности предприятия с отражением отраслевой специфики хлебопекарного и кондитерского производства [1].

Данный продукт поставляется в качестве встраиваемого отдельного модуля для «1С:ERP Управление предприятием 2» и расширяет его богатый функционал.

На рисунке 1 представлена архитектура этого комплексного решения.

Данному программному продукту присущи следующие недостатки:

– высокая стоимость самого продукта и сопутствующие расходы;

– высокие технические требования;

– сложность внедрения и интеграции;

– необходимость обучения пользователей;

– потребность привлечения специалистов-консультантов для поддержки.

Рисунок 1 – Архитектура комплексного ERP-решения при использовании модуля «Хлебобулочное и кондитерское производство» совместно с «1С:ERP Управление предприятием 2»

«Restik» – комплексная система для автоматизации предприятий общественного питания [2]. Продукт включает в себя три ключевых компонента:

– автоматизация рабочих процессов и учета – облачный сервис, объединяющий в себе управление заказами, складской учет, финансовый контроль и системы лояльности;

– сайт доставки;

– электронное меню.

Также включает в себя мобильное POS-приложение для официантов и другие компоненты.

Интерфейс сервиса автоматизации «Restik» представлен на рисунке 2.

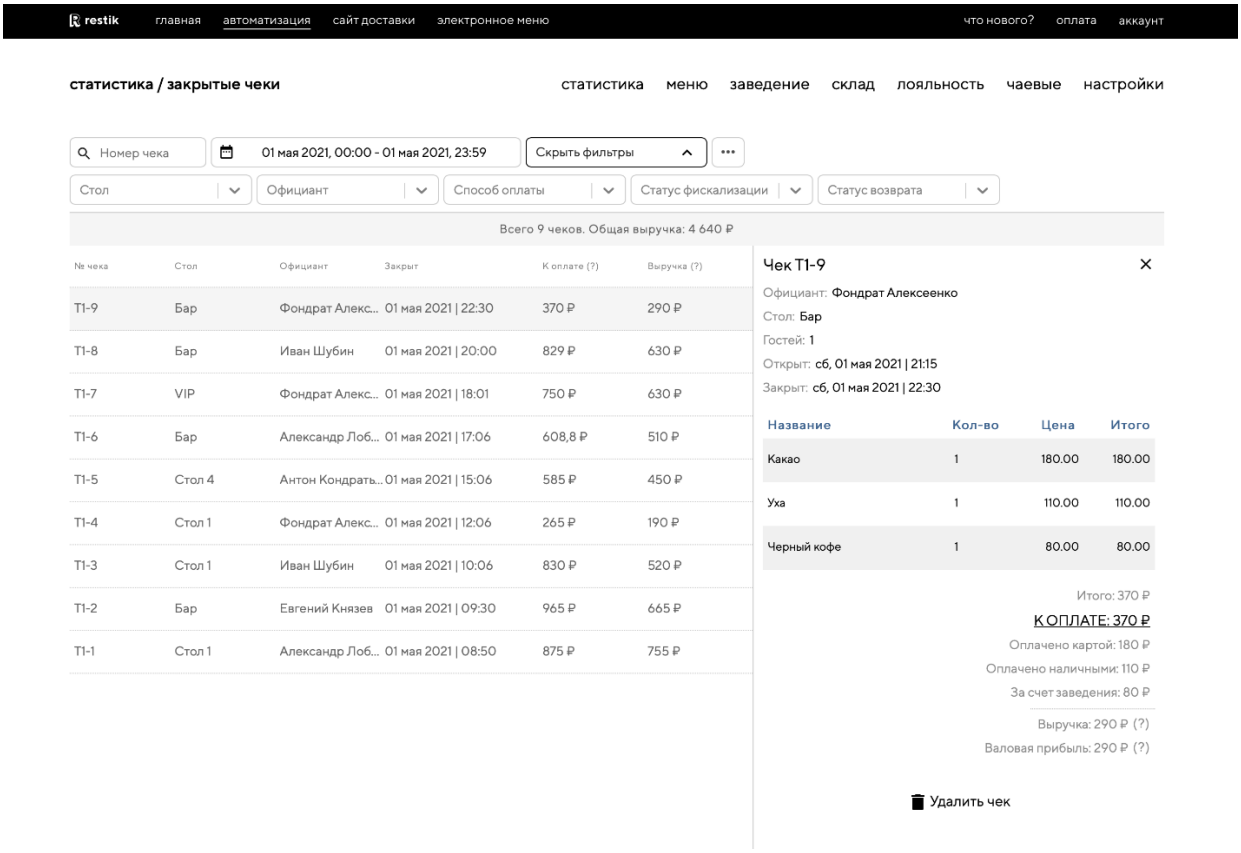

Рисунок 2 – Интерфейс сервиса автоматизации «Restik»

Этот программный продукт имеет следующие ключевые недостатки:

– не специализирован для кондитерского бизнеса;

– поставляется по подписке с периодической оплатой;

– зависимость от интернет-соединения.

«Sweetify Бизнес» – CRM-система для кондитеров, предоставляющая инструменты для оформления заказов, управления каталогом, оформления заказов и маркетинга.

Программное решение представляет собой кондитерский маркетплейс с отдельными приложениями и сайтами для кондитеров и клиентов [3].

Есть возможность бесплатного использования системы с урезанным функционалом. Продукт выпущен недавно и продолжается разработка.

На рисунке 3 представлен интерфейс мобильного приложения для кондитеров.

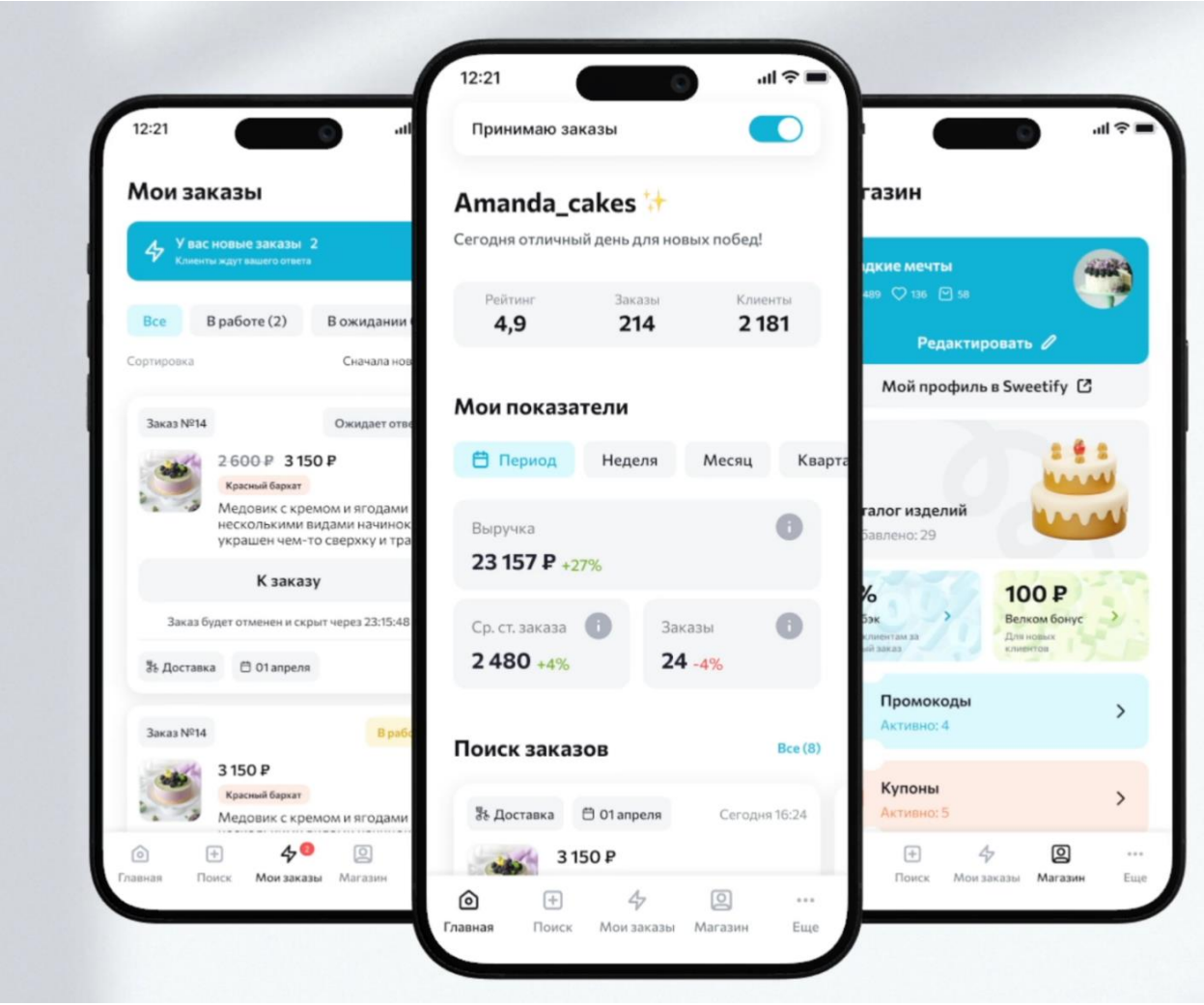

Рисунок 3 – Интерфейс мобильного приложения «Sweetify Бизнес»

Выявлены следующие недостатки:

– отсутствие складского учета;

– зависимость от интернет-соединения;

– ограниченная география – сервис доступен лишь в некоторых городах страны.

Продукт «Sweetify Бизнес» в большей степень направлен на маркетинг кондитерских изделий и не выполняет достаточно учетных функций. Система «Restik» приспособлена для физических заведений общественного питания, но предоставляет возможности учета поставок. Продукты на базе 1С обладают достаточным функционалом, но имеют высокую стоимость и сложны в использовании и обслуживании.

Разрабатываемая информационная система кондитерской организации должна предоставлять достаточно базовых возможностей учета, проста и удобна в использовании.

Также необходимо предусмотреть возможности расширения функционала и интеграции с другими технологиями и сервисами.

1.3 Анализ требований к разрабатываемой системе

Разработка информационной системы кондитерской организации осуществляется с целью автоматизации следующих бизнес-процессов:

- фиксирование использования ингредиентов;
- фиксирование изготовления кондитерского изделия;
- заказ поставки ингредиентов;
- прием поставки ингредиентов;
- формирование заказа;
- списание просроченных ингредиентов и кондитерских изделий.

В рамках анализа требований к разрабатываемой системе была построена диаграмма вариантов использования, представленная на рисунке 4.

Диаграмма вариантов использования позволяет сформулировать общие требования к функциональному поведению системы [4].

Для фиксации использования ингредиентов пользователю необходимо указать наименование ингредиента и его количество. После чего автоматически изменяется информация об остатках ингредиента на складе.

Система не должна позволять выбор несуществующего наименования и указание количества, превышающего действительное количество ингредиента на складе с неистекшим сроком годности.

Так же необходимо уведомить пользователя об остатках после внесения изменений.

При фиксировании изготовления кондитерского изделия пользователю необходимо указать наименование кондитерского изделия и количество его единиц.

Система должна автоматически рассчитать дату истечения срока годности изделий и изменить информацию о количестве кондитерского изделия на складе.

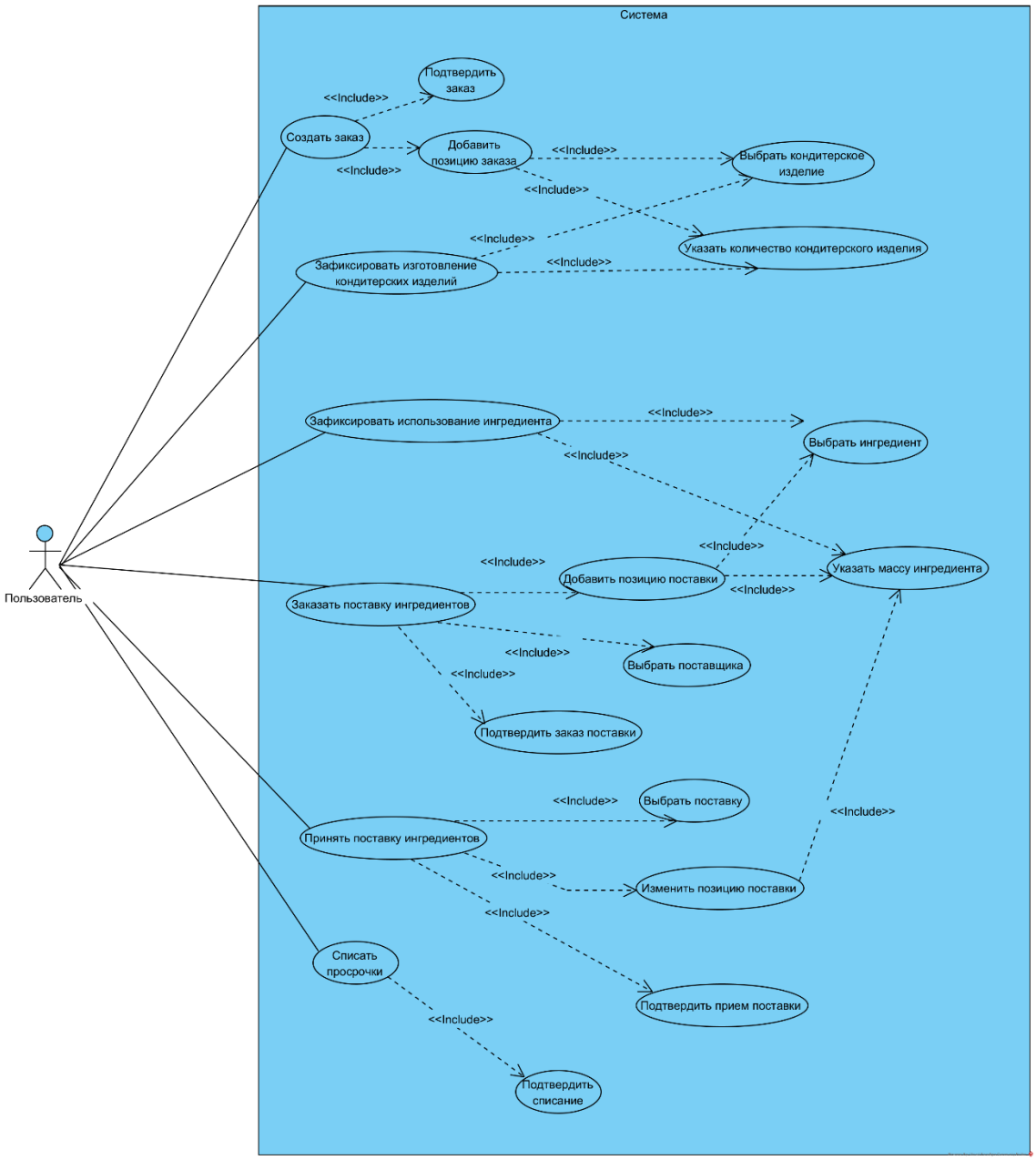

Рисунок 4 – Диаграмма вариантов использования

При заказе поставки ингредиентов пользователю необходимо выбрать поставщика. После чего система должна предоставить список наименований ингредиентов, предоставляемых выбранным поставщиком.

Пользователю необходимо указать наименование ингредиентов и необходимое для поставки количество.

Система должна автоматически рассчитать плановую стоимость поставки.

При приеме поставки пользователю необходимо выбрать необходимый элемент из списка заказов поставок.

Фактические позиции поставки при ее приеме могут не совпадать с плановыми, указанными в заказе, например, если у поставщика не оказалось свежих товаров в необходимом количестве.

Пользователю необходимо указать полученные позиции поставки, после чего система должна рассчитать фактическую стоимость и изменить информацию о количестве ингредиентов на складе.

Для формирования заказа при продаже готовых кондитерских изделий пользователь должен ввести позиции заказа: наименование кондитерского изделия и его количество.

При подтверждении ввода позиции система должна проверить наличие готовых кондитерских изделий.

Если указанное количество превышает число кондитерских изделий на складе с неистекшим сроком годности, система должна уведомить пользователя об этом и создать позицию с максимальным доступным количеством, о чем система должна уведомить пользователя

Если кондитерское изделие с указанным наименованием не найдено или его нет в наличии, позиция не будет создана, о чем также должно быть выведено уведомление.

При добавлении позиции система автоматически должна рассчитать ее стоимость и предварительную стоимость всего заказа. После того, как все позиции сформированы, пользователь подтверждает заказ, после чего система должна изменить количество готовых кондитерских изделий на складе и уведомить об успешном создании заказа.

Система должна предоставлять пользователю список кондитерских изделий и ингредиентов на складе с датой истечения срока годности близкой к текущей или позднее.

Пользователь должен подтвердить списание просроченных единиц, после чего система изменяет количество кондитерских изделий и ингредиентов на складе.

Разрабатываемая информационная система должна предоставлять пользователю удобный web-интерфейс для навигации между функциями системы, ввода и просмотра данных.

1.4 Постановка задачи

Целью работы является разработка информационной системы кондитерской организации на основе микросервисной схемы взаимодействия.

Необходимо выделить микросервисы и определить, какие бизнес-процессы автоматизирует каждый из них.

Требуется разработать серверное приложение, реализующее выделенные микросервисы и их взаимодействие.

Информационная система должна включать клиентское приложение, реализующее пользовательский интерфейс.

Для достижения поставленной цели выделен ряд задач:

- анализ предметной области;
- анализ аналогичных программных продуктов;
- анализ требований к системе;
- проектирование программных средств;
- проектирование структур данных;
- планирование и расчет бюджета разработки системы;
- реализация серверной части;
- реализация клиентской части;
- тестирование программных средств.

По их выполнении необходимо провести анализ проведенной работы и сформировать вывод о достижении поставленной цели.

2 Проектирование информационной системы

2.1 Обзор инструментальных средств проектирования и разработки

Проектирование программных средств осуществляется посредством Unified Modeling Language (UML) – унифицированного языка моделирования. Диаграммы UML можно разделить на две группы: структурные и поведенческие. Структурные диаграммы, например классов и компонентов, позволяют описать иерархию и взаимоотношения элементов модели. Для описания процессов служат поведенческие диаграммы, такие как диаграммы вариантов использования и состояний. Для создания UML-диаграмм использована среда моделирования Visual Paradigm и инструмент PlantUML.

Visual Paradigm поддерживает большое количество нотаций, предоставляет бесплатную и онлайн версии. Ключевым свойством, по которому был выбран этот инструмент, является возможность детального управления атрибутами элементов [5].

PlantUML – инструмент для создания диаграмм с помощью текстового описания со специальным синтаксисом [6]. В отличие от создания диаграмм посредством расположения графических элементов в рабочей области, для чего используется Visual Paradigm, синтаксис PlantUML позволяет описать элементы и связи диаграммы в текстовом виде. Такой подход обеспечил удобное и более точное определение и расположение линий связей на диаграммах.

Основным языком программирования был выбран Python. Этот язык широко используется для разработки web-приложений, к которым также относится разрабатываемая информация система. Разрабатываемое приложение имеет микросервисную архитектуру, особенности которой учтены при выборе языка. Синтаксис языка Python лаконичен и прост для восприятия, что облегчает командную разработку системы. Доступны возможности интеграции с различными технологиями от web-сервисов до других языков программирования, включая средства для разработки web-интерфейса

клиентского приложения: HTML, JavaScript, CSS. Для разработки на языке программирования Python существует множество библиотек с доступной документацией для решения множества различных задач. Также поддерживается асинхронное программирование [7].

Ключевые библиотеки, использованные для разработки информационной системы:

– FastAPI – web-фреймворк для создания Application Programming Interface (API) [8] – дословно программный интерфейс приложения, описание способов взаимодействия приложений или компонентов системы;

– SQLAlchemy – библиотека для синхронизации объектов Python и записей базы данных, позволяет описывать структуры баз данных и способы взаимодействия с ними на языке Python без использования SQL [9];

– Asyncpg – асинхронный драйвер для взаимодействия с PostgreSQL [10];

– Jinja2 – библиотека для HTML-шаблонизации, позволяющая использовать переменные Python в коде HTML-документа [11].

В качестве среды разработки выбран PyCharm. Это кроссплатформенная интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains на основе IntelliJ IDEA. Рассмотрим ее достоинства. Автоматическое дополнение кода с учетом типов данных, контекста и импортированных модулей значительно ускоряет написание кода. Анализ кода в реальном времени выявляет синтаксические и логические ошибки, неиспользуемые переменные и дублирования кода. Инструменты рефакторинга позволяют переименовывать элементы, извлекать методы, перемещать код изменять сигнатуры функций и так далее. При этом автоматически обновляются все ссылки на изменяемые элементы, что делает рефакторинг более безопасным и удобным. Для отладки в PyCharm предусмотрен удобный графический интерфейс для пошагового выполнения программ. В этот продукт встроена поддержка различных систем контроля версий и инструменты для разработки и

развертывания приложений в контейнерах, такие как Docker и Kubernetes. PyCharm предоставляет пользователю множество всевозможных настроек как косметических, так и функциональных. Также существует множество плагинов, расширяющих функционал среды разработки. PyCharm предоставляет возможность сборки и отладки приложений под разные операционные системы из одного интерфейса вне зависимости от операционной системы в которой работает разработчик. Также в эту среду разработки встроен инструмент для работы с базами данных DataGrip, позволяющий подключаться к различным источникам данных, реляционным и NoSQL СУБД [12]. В данной работе используется PostgreSQL.

2.2 Проектирование программных средств разрабатываемой системы

На основе анализа предметной области и требуемых функций разрабатываемой информационной системы принято решение использовать микросервисную архитектуру для ее построения. При таком подходе приложение разбито на ряд независимых и взаимодействующих сервисов. Каждый сервис самостоятельно управляет только необходимыми ему данными. Микросервисная архитектура позволяет независимо масштабировать и развертывать, тестировать и обсуживать сервисы. Изоляция сервисов обеспечивает отказоустойчивость всей системы и возможность использования различных технологий в разработке каждого из них [13].

Информационной системы кондитерской организации состоит из клиентской части, серверной части и баз данных. На рисунке 5 представлена диаграмма компонентов системы.

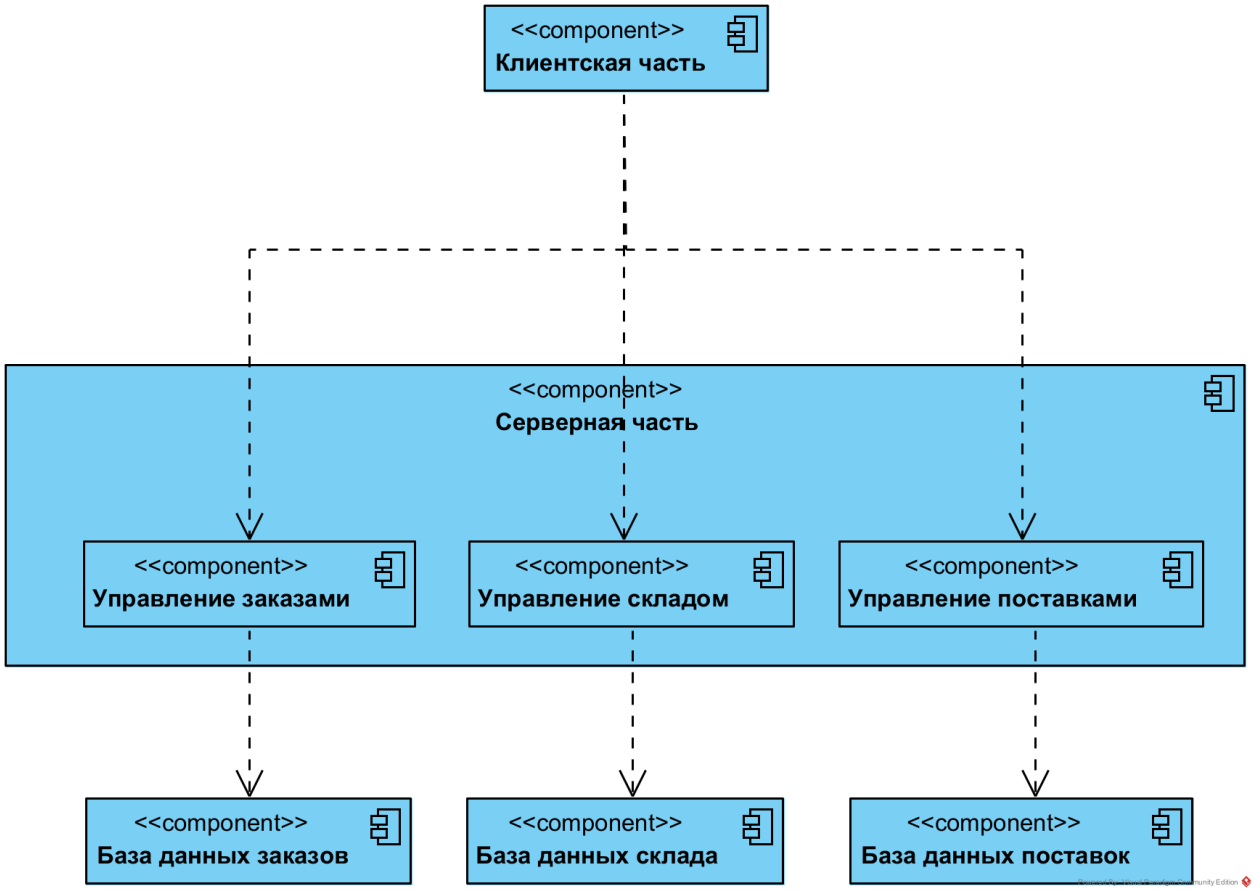

Рисунок 5 – Диаграмма компонентов

В серверной части приложения выделен ряд микросервисов:

– микросервис «Управление заказами» позволяет формировать, сохранять и просматривать информацию о продажах;

– микросервис «Управление складом» обрабатывает информацию о наличии кондитерских изделий и ингредиентов, списание просрочек также выполняется посредством этого сервиса;

– микросервис «Управление поставками» позволяет пользователю оформлять заказы на поставку ингредиентов и фиксировать их прием.

Навигация пользователя между функциями системы, ввод и чтение данных осуществляется посредством клиентской части. Клиентская часть отправляет запросы серверной части системы, где происходит обработка введенных данных и обращения к базе данных. Клиентская часть обрабатывает ответ сервера и предоставляет информацию пользователю.

Смоделируем процессы выполнения ключевых функций системы с помощью диаграмм деятельности.

На рисунке 6 представлена диаграмма деятельности микросервиса «Управление заказами» при фиксировании продажи кондитерских изделий.

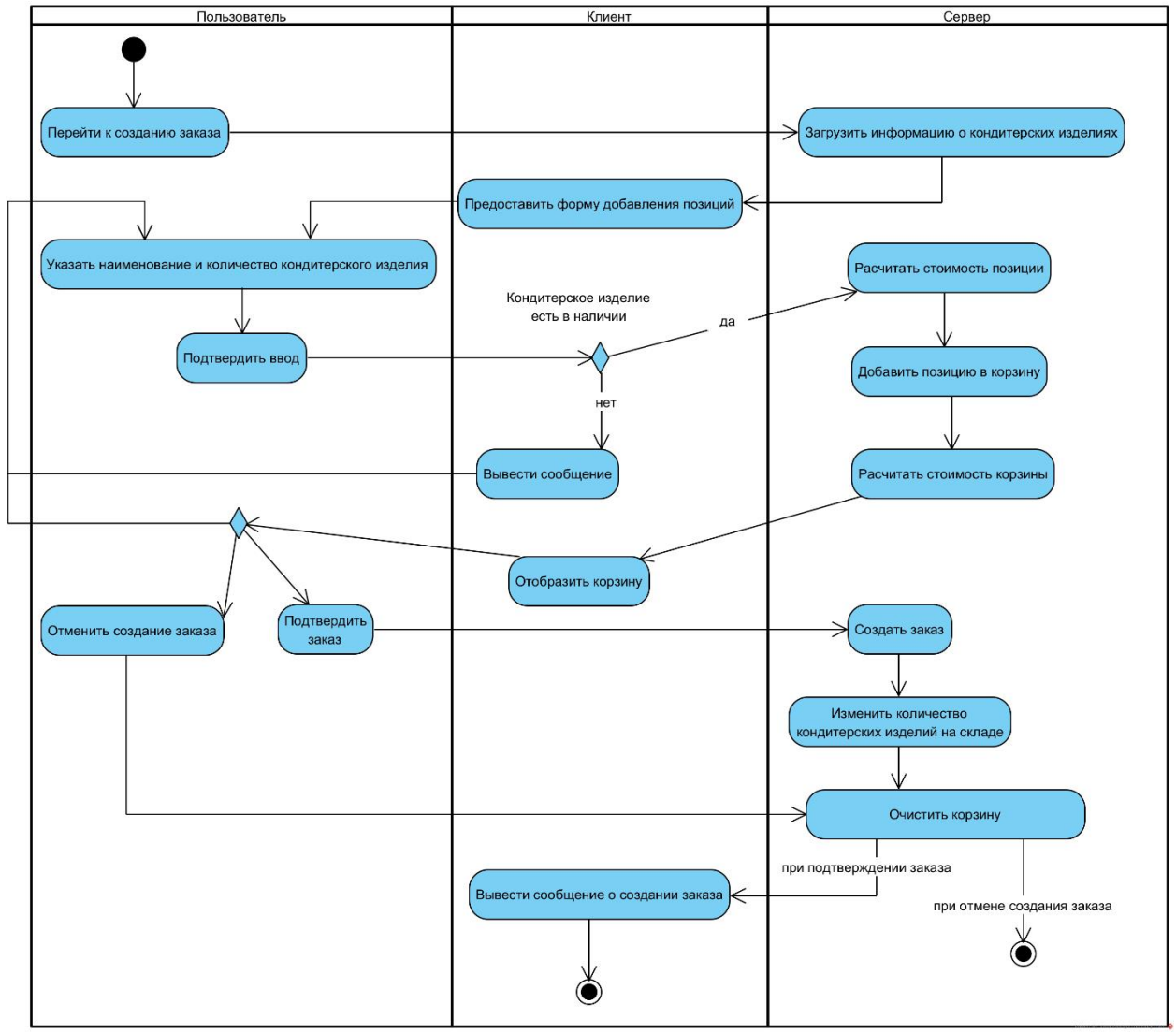

Рисунок 6 – Создание заказа

При фиксировании продажи кондитерских изделий пользователю необходимо ввести позиции заказа, указав наименования и количество кондитерских изделий. При добавлении каждой позиции серверная часть информационной системы проверяет наличие кондитерского изделия на складе и рассчитывает стоимость позиции и предварительную стоимость заказа.

Изменение количества кондитерских изделий на складе осуществляется только после подтверждения заказа.

На рисунке 7 представлена диаграмма деятельности микросервиса «Управление поставками» при создании заказа на поставку ингредиентов.

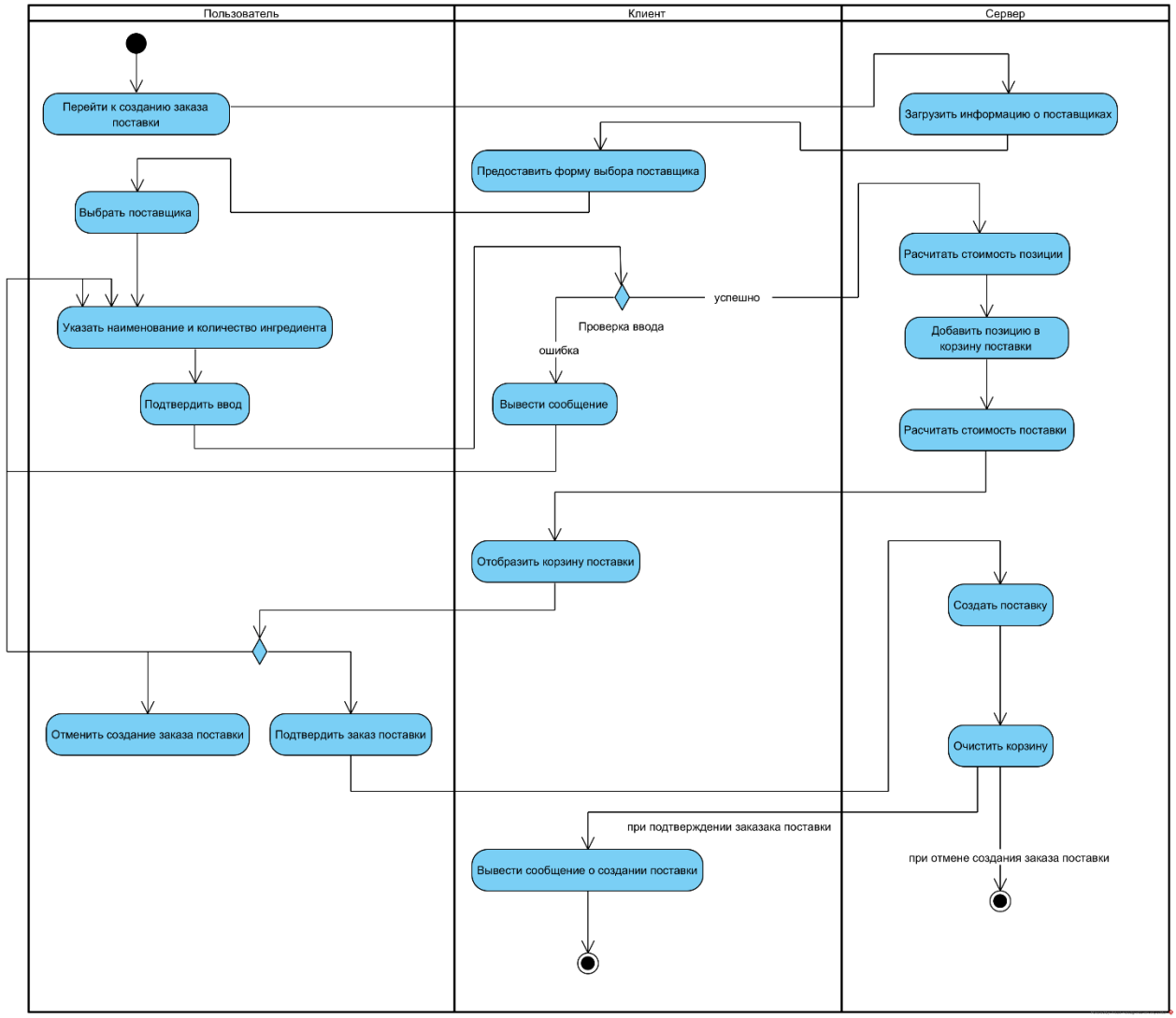

Рисунок 7 – Создание заказа поставки

При создании заказа поставки пользователю сначала необходимо указать поставщика. После чего пользователю доступен выбор ингредиентов, предоставляемых выбранным поставщиком. Заказ поставки также состоит из позиций, для добавления которых пользователю необходимо указать наименования и количество ингредиентов. При добавлении позиций серверная часть системы рассчитывает ее стоимость и предварительную стоимость

поставки по плану, так как фактическое содержимое поставки может отличаться от фактического при приеме.

На рисунке 8 представлена диаграмма деятельности микросервиса «Управление поставками» при приеме поставки.

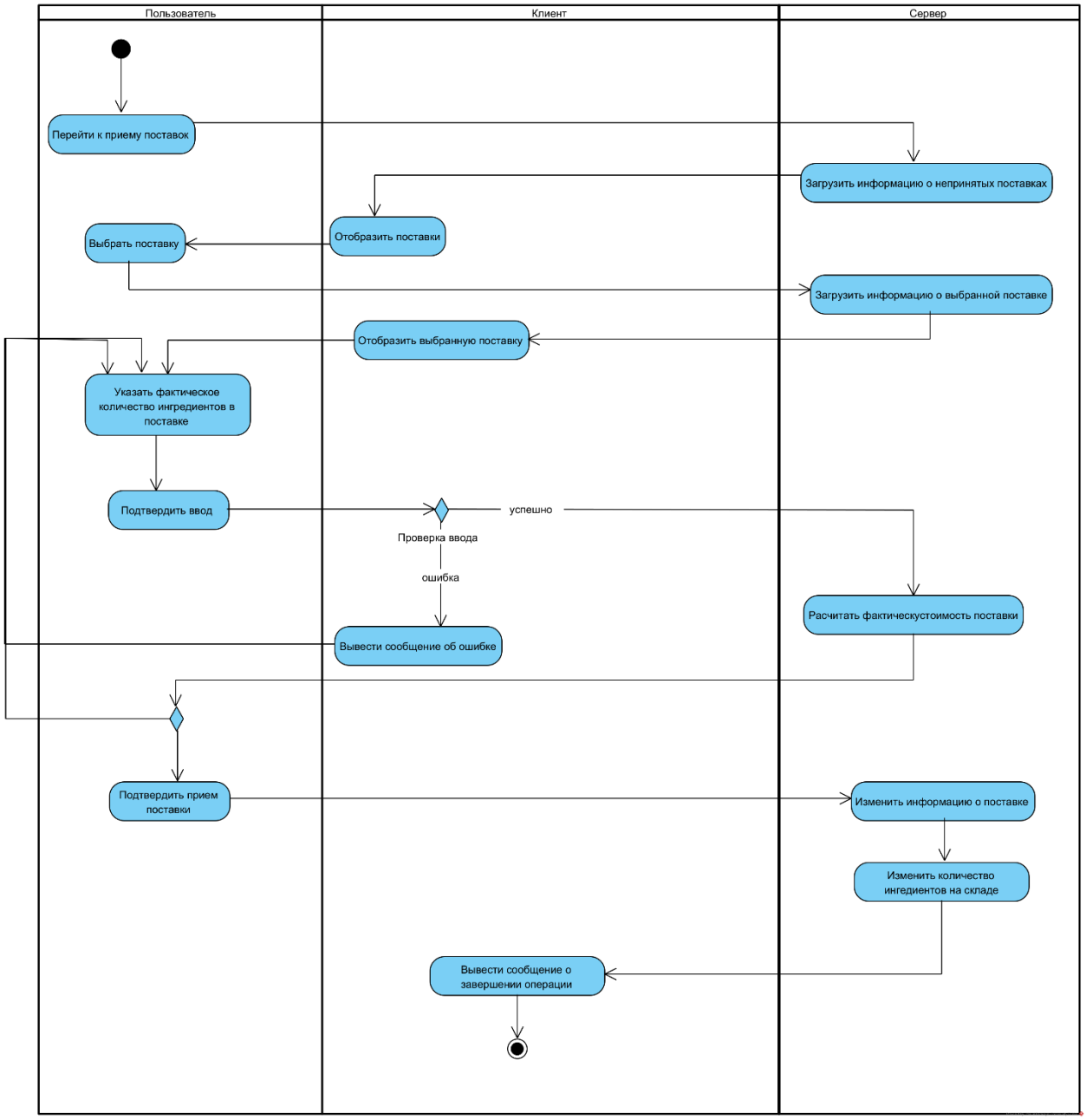

Рисунок 8 – Прием поставки

При приеме поставки пользователю сначала необходимо выбрать элемент из списка ожидаемых поставок. Выбрав поставку, пользователю необходимо указать ее фактическое содержимое.

При этом рассчитывается фактическая стоимость позиций и всей поставки. После подтверждения приема поставки выполняется изменение количества ингредиентов на складе.

На рисунке 9 представлена диаграмма деятельности микросервиса «Управление складом» при списание просроченных кондитерских изделий и ингредиентов.

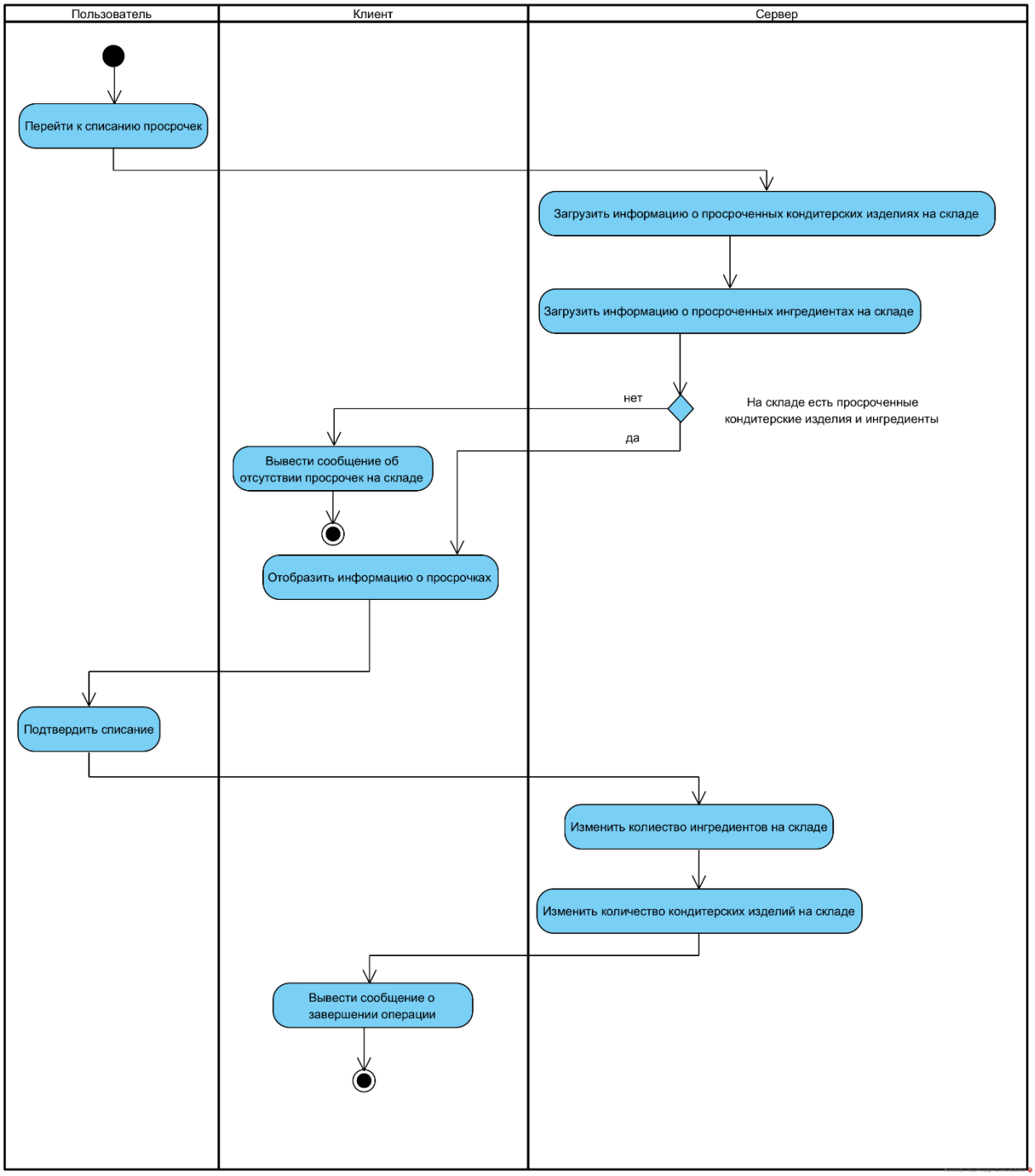

Рисунок 9 – Списание просрочек

При переходе пользователя к данной функции система предоставляет ему информацию о наличии просрочек. Если таковы имеются, пользователю необходимо подтвердить списание, после чего будет изменено количество кондитерских изделий и ингредиентов на складе.

2.3 Проектирование структур данных

Важным этапом создания информационной системы является проектирование структур данных, которыми она будет оперировать. От структуры данных зависят программные интерфейсы системы и их взаимодействие.Проектирование структур данных также важно для рационального хранения информации в базе данных.

Определим структуры информационных объектов каждого микросервиса с помощью диаграмм классов.

На рисунке 10 представлена диаграмма классов микросервиса «Управление складом».

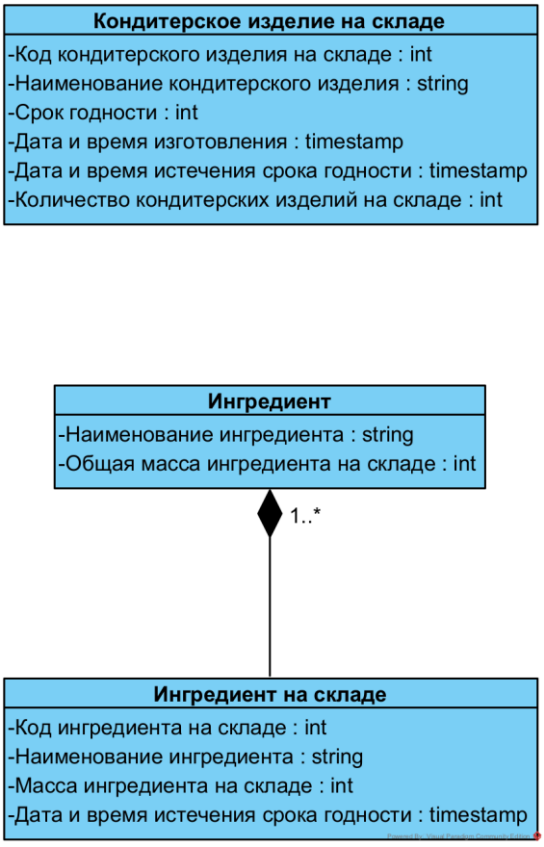

Рисунок 10 – Диаграмма классов микросервиса «Управление складом»

Объект класса «Кондитерское изделие на складе» представляет собой множество единичных кондитерских изделий одного наименования входящих в одну партию, то есть имеющих одинаковые дату и время изготовления и истечения срока годности.

Объект класса «Ингредиент» содержит информацию об общем количестве ингредиента конкретного наименования.

Объект класса «Ингредиент на складе» описывает множество ингредиентов на складе с общими датой и временем истечения срока годности.

Диаграмма классов микросервиса «Управление заказами» приведена на рисунке 11.

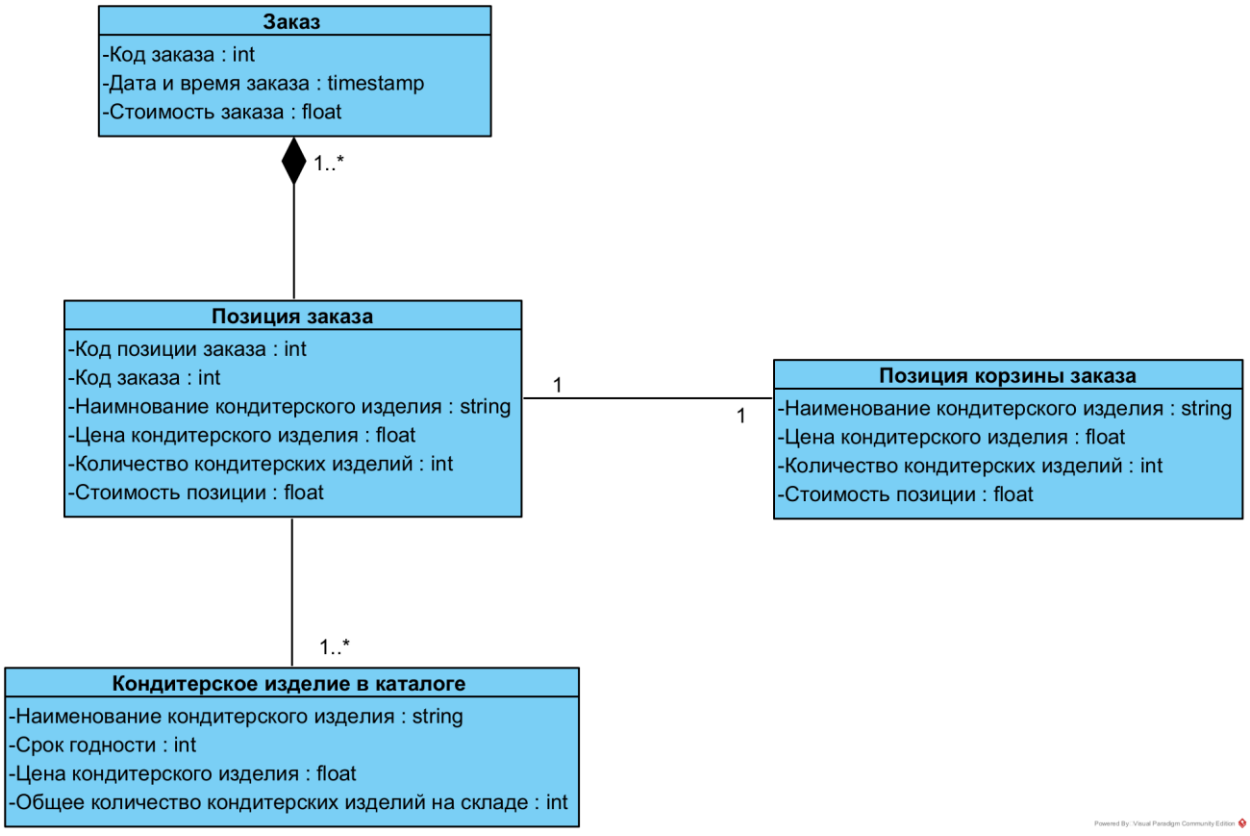

Рисунок 11 – Диаграмма классов микросервиса «Управление заказами»

Множество объектов класса «Кондитерское изделие в каталоге» представляет собой перечень всех предоставляемых кондитерской организацией наименований с информацией о них: срок годности, цена и остатки на складе.

До подтверждения создания заказа позиции формируются в корзине. После подтверждения, формируется заказ и его позиции из корзины.

Позиции корзины описываются наименованием ценой и количеством кондитерского изделия и стоимость.

Заказ содержит дату и время осуществления продажи и общую стоимость.

Для каждого заказа формируется множество позиций из корзины, с указанием принадлежности к конкретному заказу.

Структура данных, оперируемая микросервисом «Управление поставками» приведена на рисунке 12.

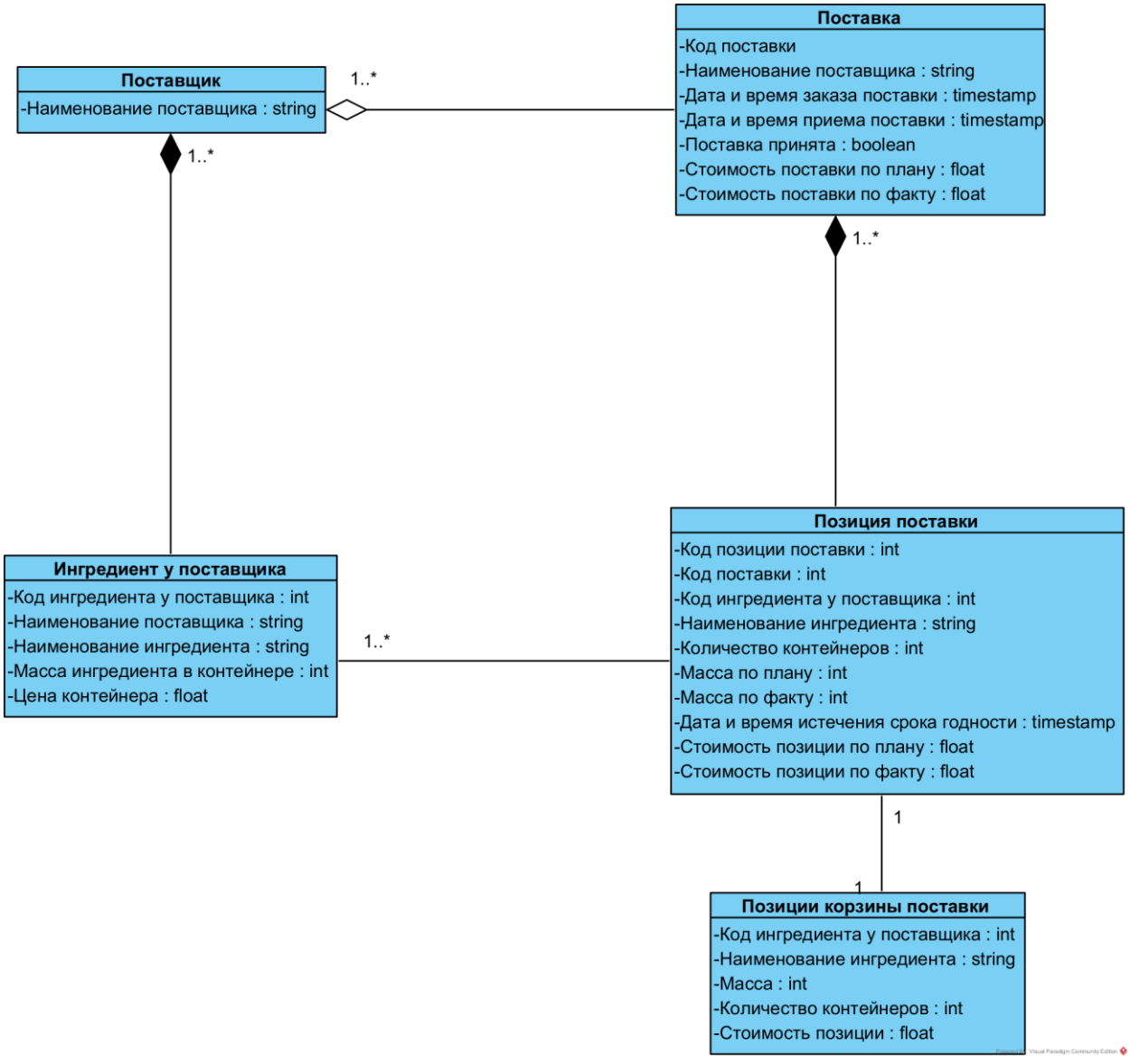

Рисунок 12 – Диаграмма классов микросервиса «Управление поставками»

Информация о каждом поставщике представляется одним объектом «Поставщик» и множеством «Ингредиентов у поставщика». Ингредиент у поставщика описывает одну единицу поставки предоставляемого наименования.

Объект класса «Поставка» содержит информацию о поставщике и общем количестве ингредиентов в поставке, дате и времени заказа и приема поставки, является ли поставка принятой, плановой и фактической стоимости поставки.

Объект «Позиция корзины поставки» содержит информацию об ингредиенте, его количестве и стоимости.

Для каждой поставки существует множество объектов класса «Позиция поставки» с информацией о плановых и фактических количестве и стоимости ингредиента, а также датой и временем истечения срока годности.

Для каждого класса будет сформирована таблица в базе данных. Каждому объекту соответствует одна запись в таблице.

2.4 Планирование разработки системы и оценка бюджета

Разработка программного обеспечения имеет стоимость. Разрабатываемая информационная система кондитерской организации является учебным проектом и не является коммерческим продуктом, однако есть возможность оценить ее бюджет, основываясь на поставленных задачах, их оценочной длительности, трудовых ресурсах, необходимых для их выполнения. Трудовыми ресурсами являются различные IT-специалисты:

- аналитик;
- разработчик;
- тестировщик.

Планирование разработки системы и оценка бюджета проводится с учетом производственного календаря на 2025 год, рабочего графика специалистов – 5 дней в неделю, 8 часов в день и их зарплатной ставки.

В рамках планирования разработки были выделены этапы работы над проектом, включающие несколько задач, определены их исполнители и оценочная длительность выполнения.

Перечень задач приведен в таблице 1.

Таблица 1 – Задачи проекта

Этап разработки	Исполнитель	Задача	Длительность, дни
Постановка задачи и анализ требований	Аналитик	Анализ предметной области	1
		Анализ аналогичных программных продуктов	2
		Анализ требований к разрабатываемой системе	1
		Анализ технологий проектирования и разработки	1
Проектирование информационной системы	Разработчик	Проектирование серверной части	3
		Проектирование структур данных	1
		Проектирование клиентской части	1
Разработка программных средств		Разработка серверной части	9
		Разработка клиентской части	4
Тестирование программных средств	Тестировщик	Функциональное тестирование	2
		Тестирование API	2

На основе оценочной длительности выделенных задач и зарплатных ставок их исполнителей рассчитан бюджет проекта. Результат приведен в таблице 2.

Таблица 2 – Оценка бюджета проекта по стоимости работы специалистов

Исполнитель	Зарплатная ставка, рублей в час	Затраченное время, часов	Всего, рублей
Аналитик	912	40	36480
Разработчик	1150	144	165600
Тестировщик	812	32	25984

На рисунке 13 приведено графическое представление планирования разработки в виде диаграммы Ганта.

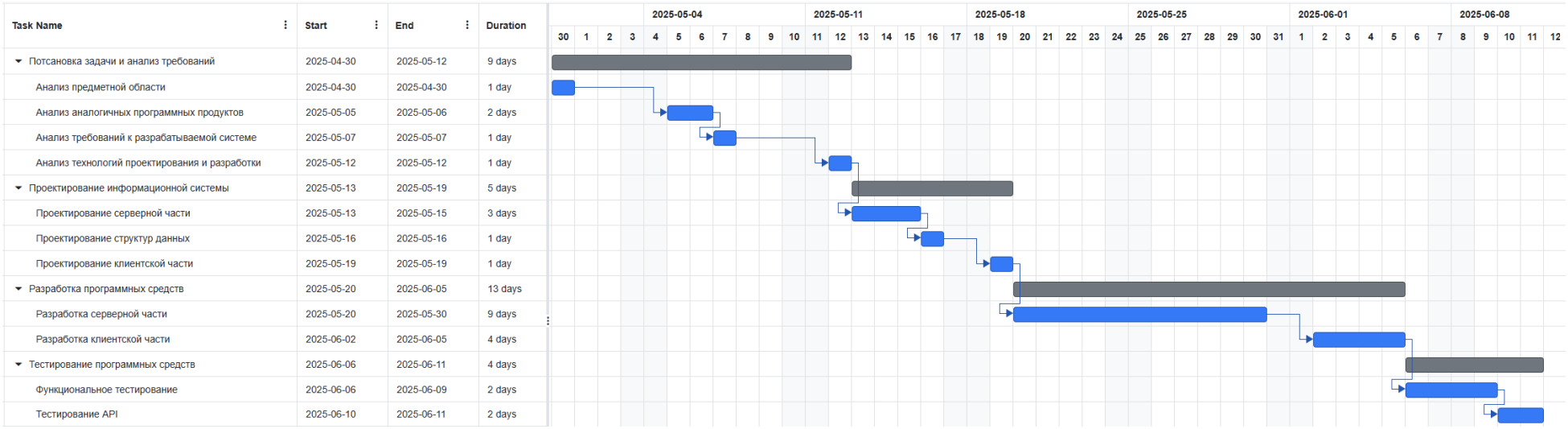

Рисунок 13 – Диаграмма Ганта

Почасовые ставки специалистов рассчитаны на основе зарплатных отчетов сервиса «Хабр Карьера» [14].

По итогам планирования разработки общая длительность работ над проектом составляет 216 часов, что эквивалентно 27 рабочим дням. Бюджет проекта оценен в 228064 рубля.

3 Разработка программных средств

3.1 Разработка серверной части

Ключевым этапом разработки серверной части информационной системы с микросервисной архитектурой является определение HTTP-запросов, посредством которых осуществляется взаимодействие компонентов системы и обработка данных.

Для визуального представления обработчиков запросов в качестве средства автоматической документации использован инструмент OpenAPI, также известный как Swagger.

Далее приведены обработчики запросов на примере микросервиса «Управление заказами».

Необходимо получение полного списка предоставляемых кондитерских изделий. Для этого определен обработчик GET-запроса представленный на рисунке 14.

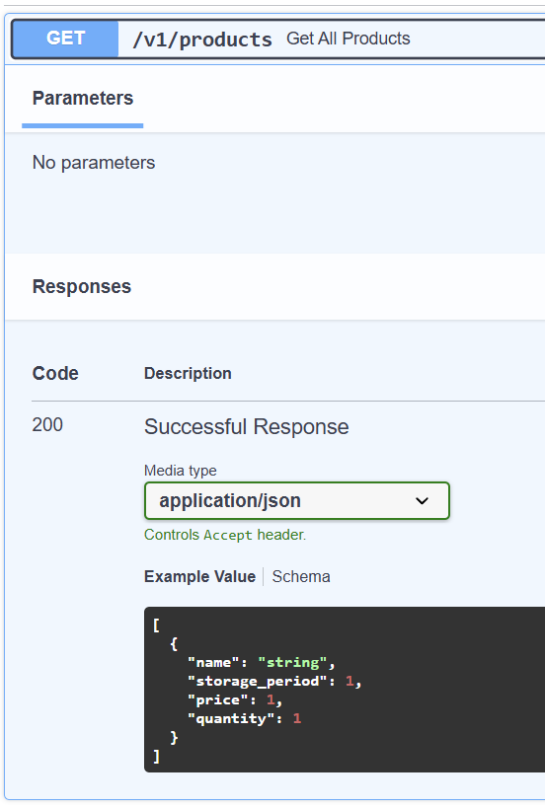

Рисунок 14 – Получение всех кондитерских изделий

Добавление позиции в корзину осуществляется посредством POST-запроса, обработчик которого представлен на рисунке 15.

В теле запроса указываются строка – наименование кондитерского изделия, и целое число – количество кондитерских изделий.

Тело ответа содержит информативное сообщение и полные данные о позиции с ее рассчитанной стоимостью.

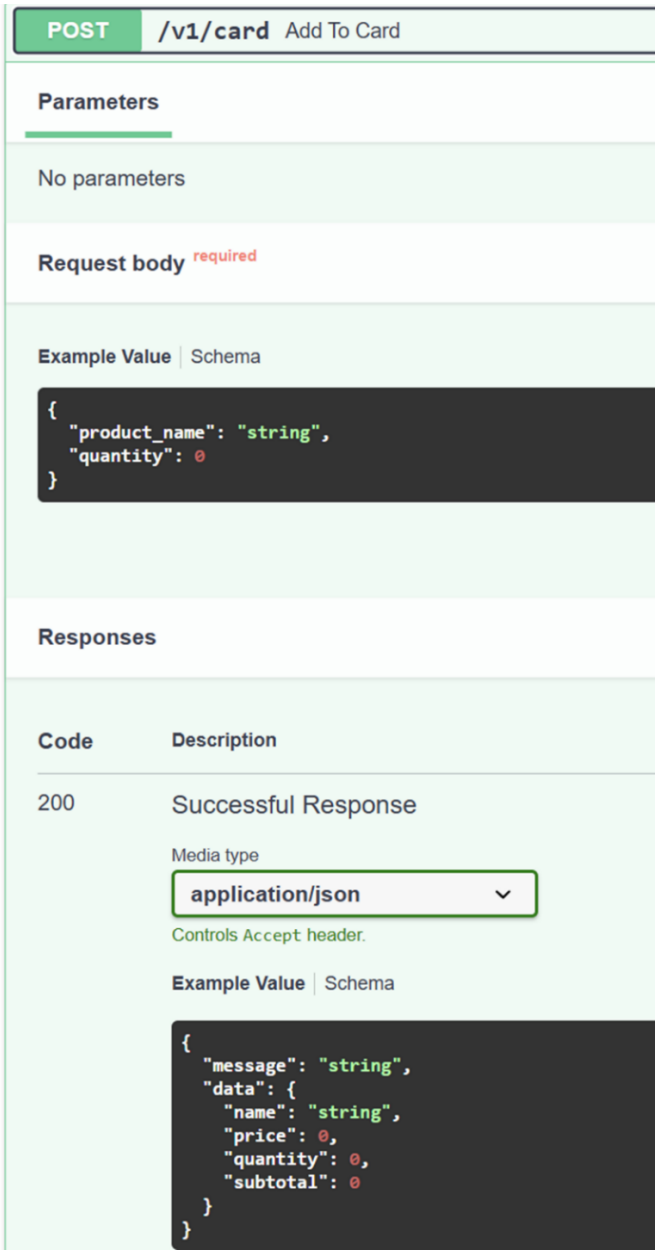

Рисунок 15 – Добавление в корзину

Для очистки корзины используется DELETE-запрос, его обработчик представлен на рисунке 16.

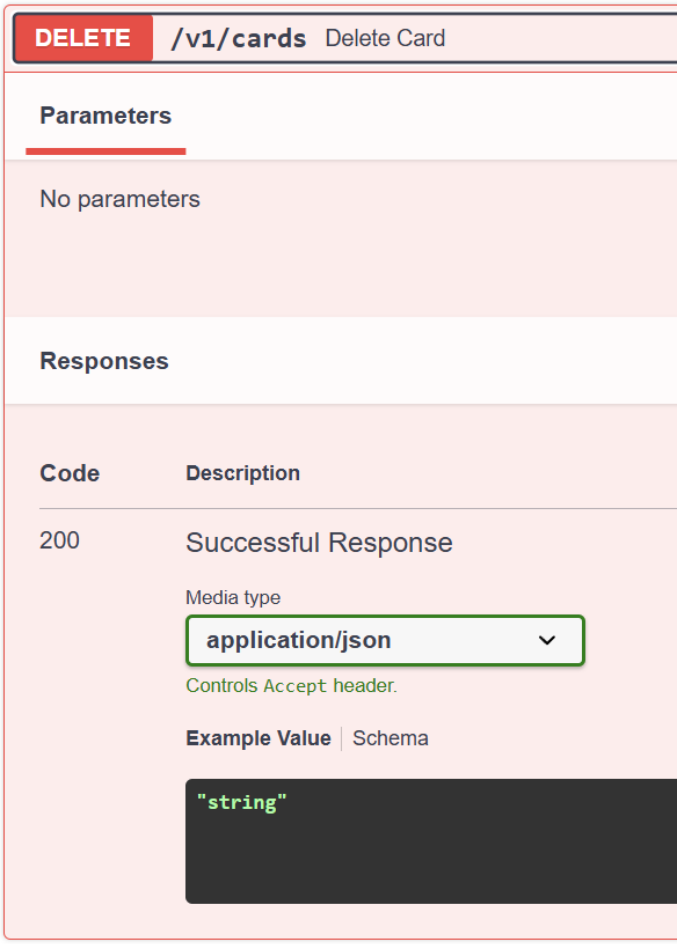

Рисунок 16 – Очистка корзины

На рисунке 17 приведен обработчик GET-запроса для просмотра корзины. Тело ответа имеет следующие поля:

– total – общая стоимость заказа;
– positions – список позиций в заказе.

Каждый элемент этого списка содержит поля:

– строка с наименованием кондитерского изделия;
– цена единицы кондитерского изделия;
– количество кондитерских изделий данного наименования;
– стоимость позиции.

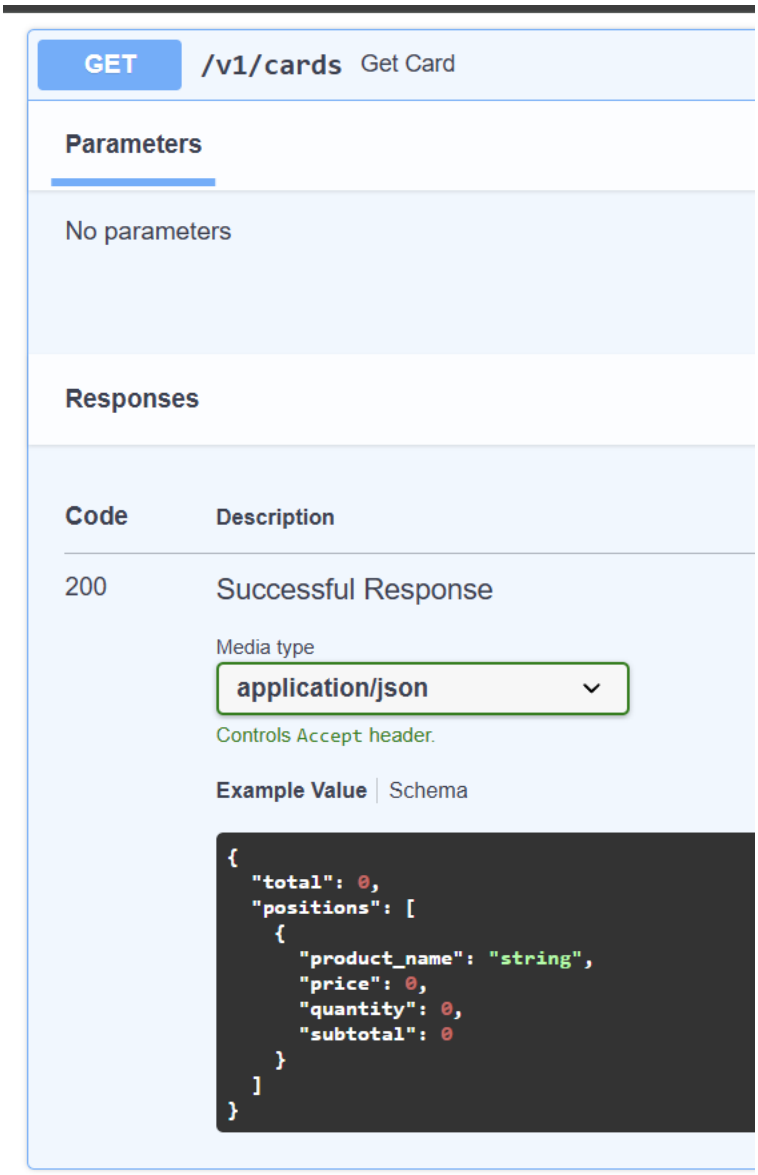

Рисунок 17 – Просмотр корзины

Обработчик POST-запроса для создания заказа представлен на рисунке 18.

В теле данного запроса указываются дата и время создания заказа и общая стоимость заказа.

Эти данные заполняются значениями по умолчанию. Дата и время создания заказа принимает значение текущего времени.

Тело ответа содержит полную информацию о заказе, включая перечень позиций.

Именно при выполнении этого запроса из позиций корзины формируются позиции заказа с указанием принадлежности к конкретному заказу.

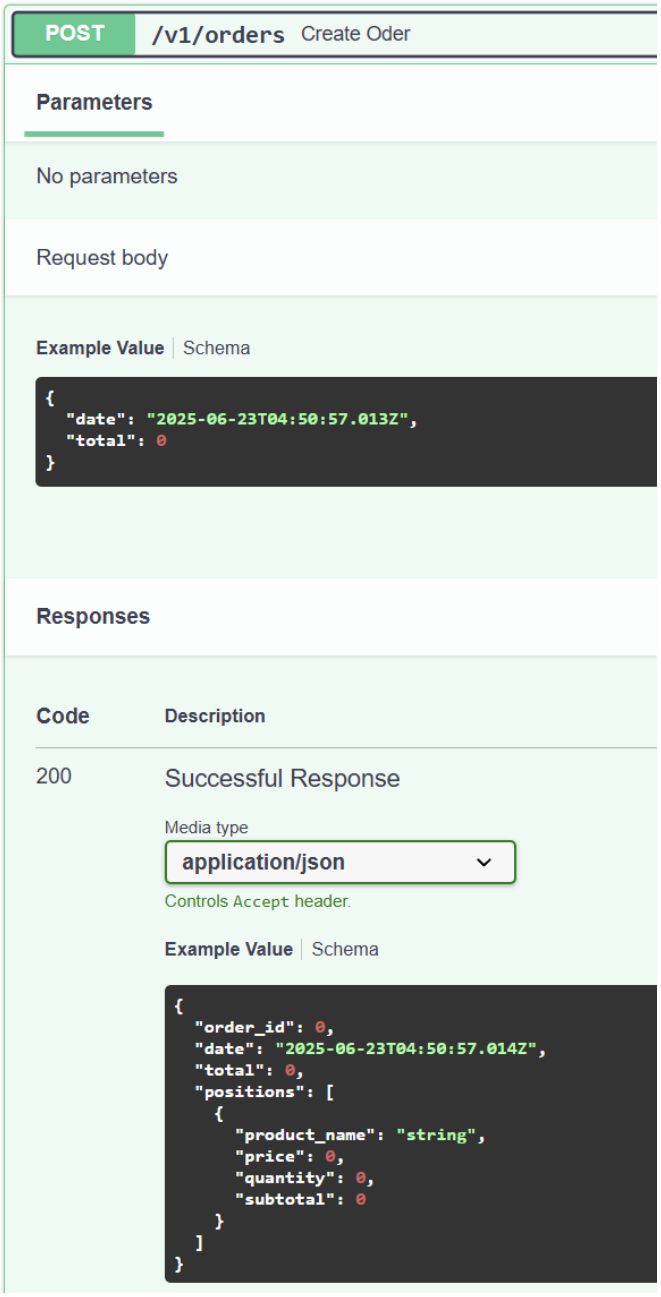

Рисунок 18 – Создание заказа

Для просмотра заказов определен GET-запрос, обработчик которого представлен на рисунке 19.

Этот запрос не имеет параметров.

В теле ответа содержится список заказов. Структура элементов этого списка соответствует телу ответа POST-запроса для создания заказа

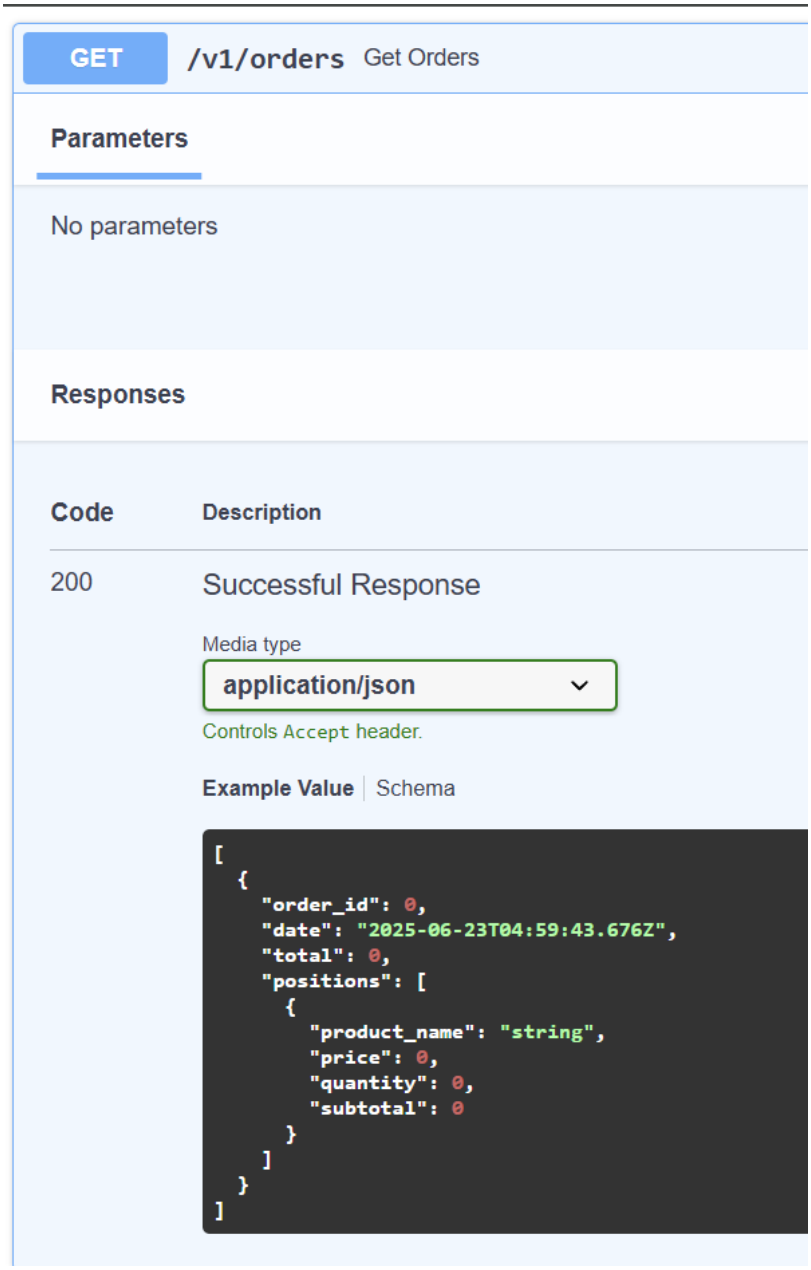

Рисунок 19 – Просмотр заказов

Рассмотрим код обработчика запроса на примере создания заказа. Код обработчика приведено на листинге 1.

Листинг 1 – Код обработчика POST-запроса для создания заказа

```
# Создание заказа
@router.post("/orders", response_model=OrderOutput)
async def create_oder(order: AddOrder | None = None, session:
AsyncSession = Depends(get_session)):

    # Создание пустого заказа для получения его кода
    order = order if order else AddOrder()
    my_order = Order.model_validate(order)
```

```
    session.add(my_order)
    await session.commit()
    await session.refresh(my_order)
    new_order_id = my_order.id

    # Получение позиций из корзины
    card = await get_card(session=session)
    card_items = card["card"]


    # Расчет стоимости заказа
    total = sum([c.subtotal for c in card_items])

    # Формирование позиций заказа из позиций в корзине
    poses = []
    for card_item in card_items:
        pos = Position(
            product_name=card_item.name,
            price=card_item.price,
            quantity=card_item.quantity,
            subtotal=card_item.subtotal,
            order_id=new_order_id)
        poses.append(pos)
        session.add(pos)
        await session.commit()

        # Изменение количества кондитерского изделия на складе
        stmt = update(Product).where(Product.name ==
card_item.name).values(quantity=Product.quantity -
card_item.quantity)
        await session.exec(stmt)
        await session.commit()

    # Обновление пустого заказа
    stmt = update(Order).where(Order.id ==
new_order_id).values(total=total, completed = True)
    await session.exec(stmt)
    await session.commit()
    await session.refresh(my_order)

    # Очистка корзины
    await delete_card(session=session)

    return {"order": my_order, "pos": poses}
```

Аналогичным образом реализованы обработчики запросов остальных микросервисов.

3.2 Разработка клиентской части

Клиентская часть информационной системы реализует пользовательский интерфейс для навигации по функциям системы, ввода и представления данных. Клиентская часть представляет собой набор web-страниц:

– «Домашняя страница» – содержит перечень функций системы, сгруппированных по микросервисам;

– «Каталог» – перечень предоставляемых кондитерских изделий с указанием цены, срока годности и остатках на складе;

– «Создать заказ»;

– «Просмотр заказов»;

– «Изготовление кондитерских изделий»;

– «Использование ингредиентов»;

– «Списание просрочек»;

– «Поставщики» – содержит информацию о поставщиках и предоставляемых ими ингредиентах;

– «Заказать поставку»;

– «Прием поставок».

Загрузка страниц осуществляется с помощью GET-запросов. В листинге 2 приведен код обработчика такого запроса на примере домашней страницы.

Листинг 2 – Код обработчика GET-запроса для загрузки домашней страницы

```
@router.get("/index", name="home")
async def home(request: Request) :
    return templates.TemplateResponse(
        request=request,
        name="index.html")
```

Для использования переменных Python в коде HTML-документа, необходимо передать их в обработчике запроса в поле «context». В качестве примера в листинге 3 представлен код обработчика запроса для загрузки страницы «Создание заказа».

Листинг 3 – Код обработчика GET-запроса для загрузки страницы «Создание заказа»

```
@router.get("/index/create_order", name="create_order")
async def input_form(request: Request, session: AsyncSession =
Depends(get_session)):

    await delete_card(session)
    all_prods = await get_all_products(session)

    return templates.TemplateResponse(
        request=request,
        name="create_order.html",
        context={
            "all_prods": all_prods})
```

Для демонстрации обращения к переменной Python в коде HTML-документа на листинге 4 приведен фрагмент кода страницы «Каталог».

Листинг 4 – Обращение к переменной Python в коде HTML-документа для страницы «Создание заказа»

```
<datalist id="all_prods">
        {% for prod in all_prods %}
            <option value="{{ prod.name }}"></option>
        {% endfor %}
    </datalist>
```

В HTML-документы страниц встроены скрипты на языке JavaScript для обработки действий пользователя, динамического изменения содержимого страницы, отправки запросов серверу и обработки ответов.

На листинге 5 приведен фрагмент скрипта страницы «Создание заказа» с отправкой GET-запроса для просмотра корзины и обработкой ответа.

Листинг 5 – Фрагмент скрипта страницы «Создание заказа» с отправкой GET-запроса для просмотра корзины и обработкой ответа

```
let get_card_response = await fetch('/v1/cards');
        let resp = await get_card_response.json();
        let card_list = resp.card;
```

В качестве примера работы с POST-запросом на листинге 6 приведен фрагмент скрипта страницы «Изготовление кондитерских изделий» для фиксирования изготовления кондитерских изделий.

Листинг 6 – Фрагмент скрипта страницы «Изготовление кондитерских изделий» для работы с POST-запросом при фиксировании изготовления кондитерского изделия

```
let productInput = document.getElementById("prod_input").value;
let quantityInput =
document.getElementById("quantity_input").value;

if (!productInput || !quantityInput) {
    alert("Выберите наименование кондитерского изделия и укажите
количество");
    return;}

let ProdInput = {
    product_name: productInput,
    quantity: parseInt(quantityInput)};

let add_response = await fetch('/v1/storage/product', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'},
    body: JSON.stringify(ProdInput)});

let resp = await add_response.json();
alert(resp.message);
```

Для графического отображения взаимодействия пользователя и микросервисов использованы диаграммы последовательности.

На диаграмме последовательности неявно присутствует ось времени, что позволяет визуализировать временные отношения между передаваемыми сообщениями.

С помощью диаграммы последовательности можно представить взаимодействие элементов модели как своеобразный временной график «жизни» всей совокупности объектов, связанных между собой для реализации варианта использования программной системы или выполнения какой-либо задачи.

На рисунке 20 приведена диаграмма последовательности для выполнения функций микросервиса «Управление заказами».

Для их выполнения требуется взаимодействие с сервисом «Управление складом».

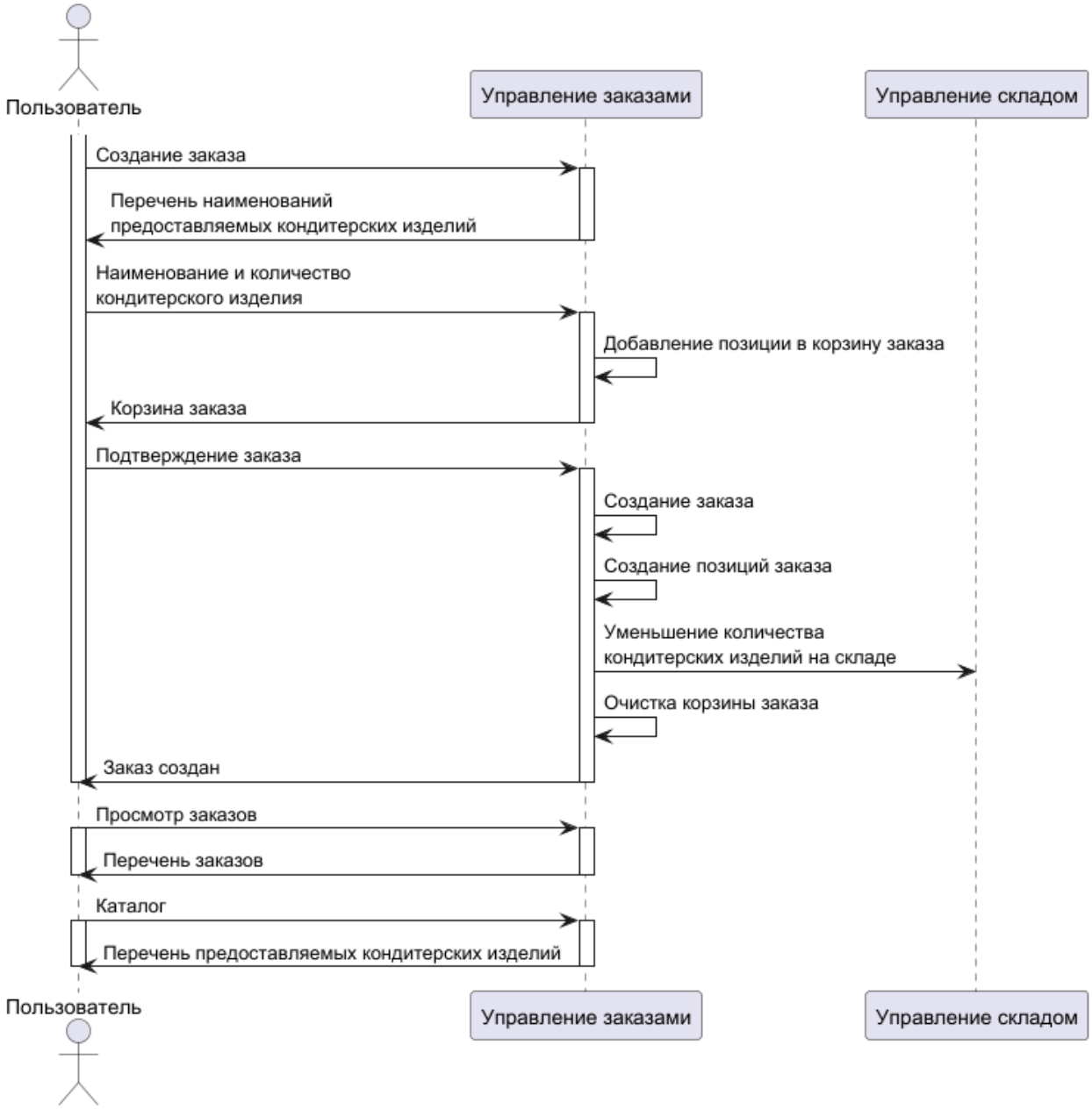

Рисунок 20 – Диаграмма последовательности для микросервиса «Управление заказами»

Диаграмма последовательности для выполнения функций микросервиса «Управление складом» представлена на рисунке 21.

Поскольку перечень наименований всех предоставляемых кондитерских изделий реализован на стороне микросервиса «Управление заказами», необходим обмен данными с компонентом.

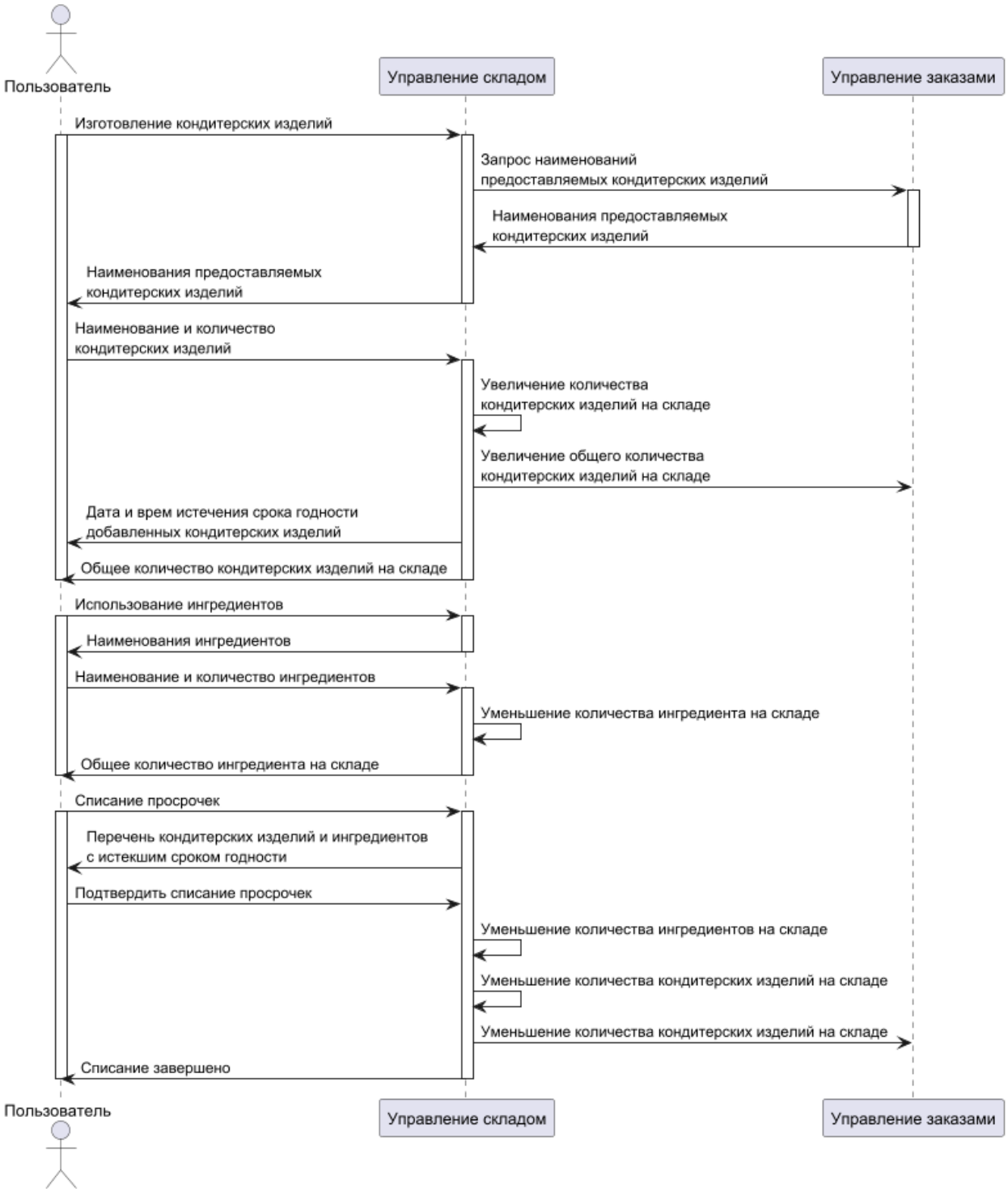

Рисунок 21 – Диаграмма последовательности для микросервиса «Управление складом»

Рисунок 22 содержит диаграмму последовательности для выполнения функций микросервиса «Управление поставками».

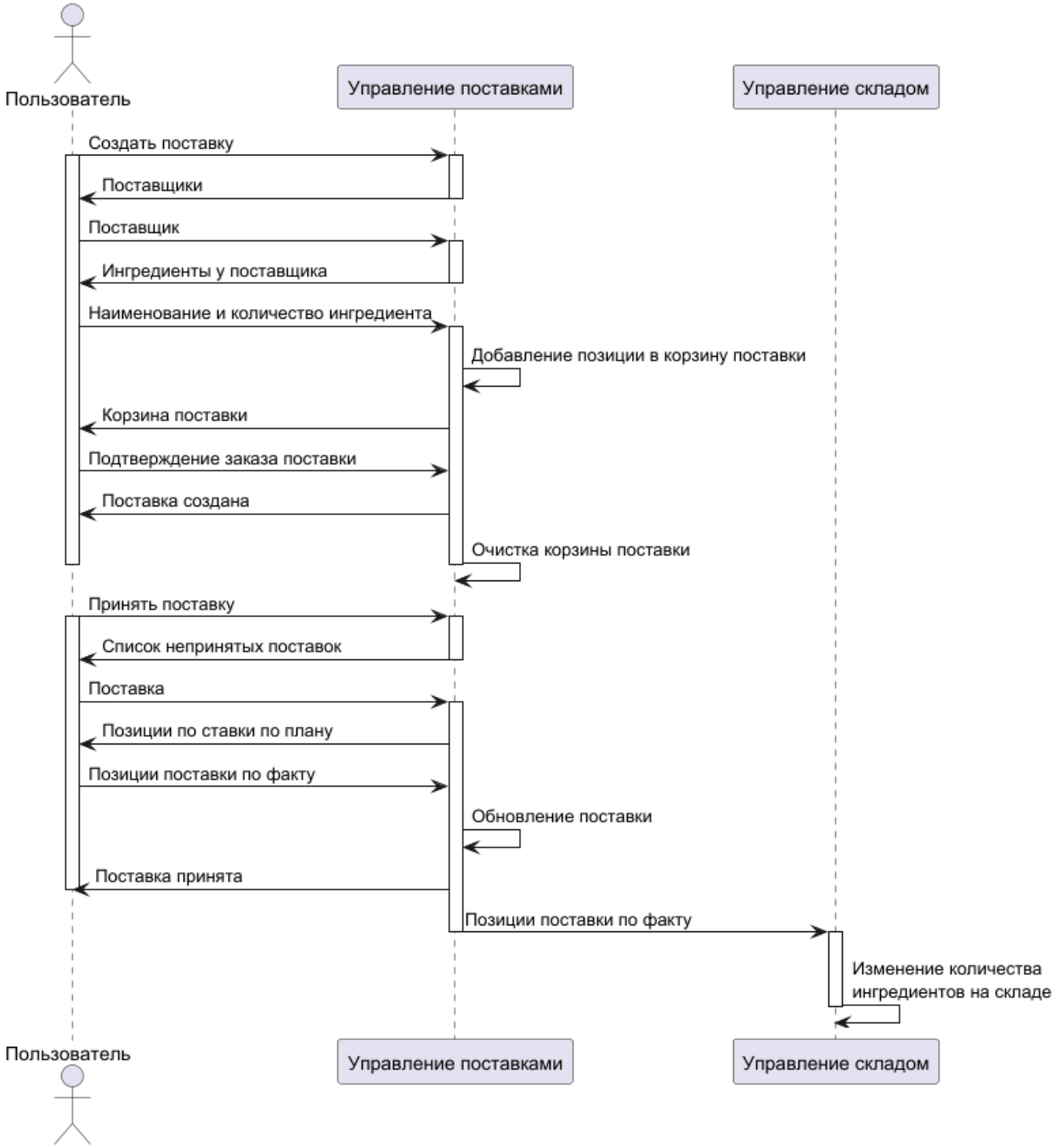

Рисунок 22 – Диаграмма последовательности для микросервиса «Управление поставками»

Внешний вид web-страниц представлен на рисунках в приложении А. Листинг программы приведен в приложении Б.

4 Тестирование программных средств

4.1 Функциональное тестирование

По завершении разработки программных средств была осуществлена проверка соответствия системы выдвинутым функциональным требованиям. Для этого было выполнено функциональное тестирование методом функциональных диаграмм.

Функциональная диаграмма или же диаграмма причинно-следственных связей является способом формального описания спецификаций. Спецификация определяет причины и следствия поведения программы. Причинами являются входные условия, следствия – результат выполнения программы [15].

Для составления функциональной диаграммы выделим причины и следствия поведения системы на примере добавления позиций при создании заказа. Причины и следствия приведены в таблице 3.

Таблица 3 – Причины и следствия

Причины	
П1	Не указано наименование кондитерского изделия
П2	Указано недействительное наименование кондитерского изделия
П3	Количество кондитерских изделий с указанным наименованием в наличии равно нулю
П4	Позиция с указанным наименованием кондитерского изделия уже есть в корзине
П5	Указанное количество кондитерских изделий превышает количество кондитерских изделий с указанным наименованием в наличии не равное нулю
П6	Указанное количество кондитерских изделий не превышает количество кондитерских изделий с указанным наименованием в наличии
Следствия	
С1	Сообщение «Выберите наименование кондитерского изделия и укажите количество»
С2	Сообщение «Кондитерское изделие не найдено»
С3	Сообщение «Кондитерского изделия нет в наличии»
С4	Сообщение «Кондитерское изделие уже добавлено»

Продолжение таблицы 3

Следствия	
С5	Сообщение «Кондитерское изделие в наличии: N», где N – количество кондитерских изделий с указанным наименованием в наличии
С6	Успешное добавление позиции

На основе выделенных причин и следствий составлена функциональная диаграмма, представленная на рисунке 23.

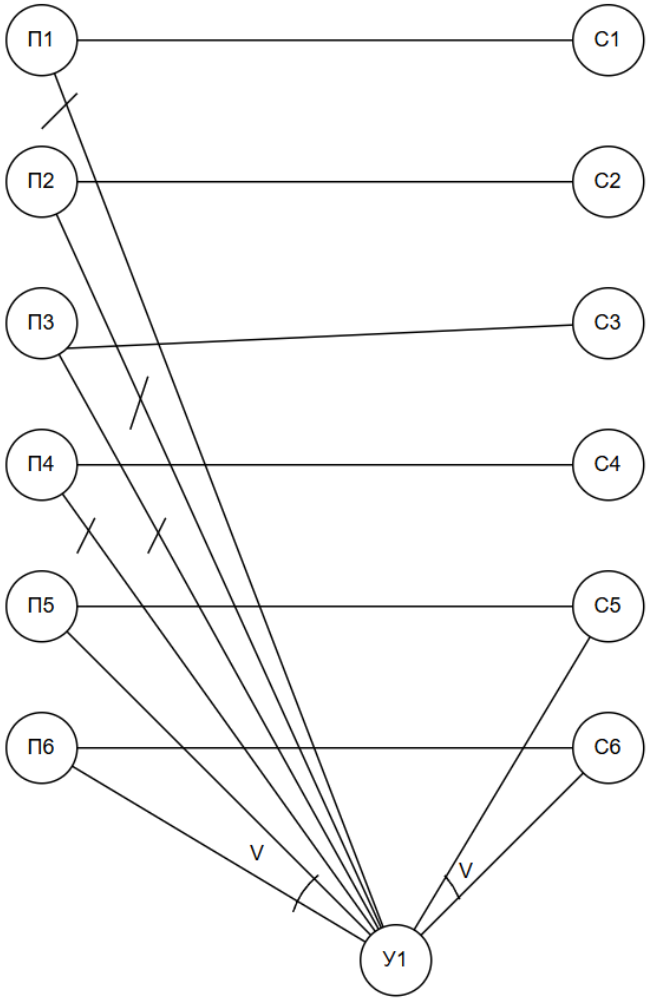

Рисунок 23 – Функциональная диаграмма

Посредством методического прослеживания состояний условий диаграммы, она преобразуется в таблицу решений, каждый столбец которой представляет определенный тест программы.

Если состояние системы удовлетворяет условия проверки, в таблице записывается единица, в противном случае – ноль.

Значения для следствий формируются аналогично.

При построении таблицы решений также необходимо отследить невозможные сочетания причин и следствий.

Таблица решений приведена в таблице 4.

Таблица 4 – Таблица решений

	1	2	3	4	5	6
П1	1	0	0	0	0	0
П2	0	1	0	0	0	0
П3	0	0	1	0	0	0
П4	0	0	0	1	0	0
П5	0	0	0	0	1	0
П6	0	0	0	0	0	1
С1	1	0	0	0	0	0
С2	0	1	0	0	0	0
С3	0	0	1	0	0	0
С4	0	0	0	1	0	0
С5	0	0	0	0	1	0
С6	0	0	0	0	1	1

На основе таблицы решений составлены тестовые случаи.

Тестовые случаи описываются входными и выходными данными и результатом.

Поведение системы при выполнении тестов представлено на рисунках в приложении В.

Описание тестовых случаев приведено в таблице 5.

Таблица 5 – Тестовые случаи

№	Входные данные	Выходные данные	Результат тестирования
1	Наименование: пустая строка Количество: 1	Сообщение «Выберите наименование кондитерского изделия и укажите количество»	Тест пройден (рисунок В.1)
2	Наименование: абв Количество: 1	Сообщение «Кондитерское изделие абв не найдено»	Тест пройден (рисунок В.2)
3	Наименование: Нежный крем-брюле Количество: 1 Количество кондитерских изделий наименования «Нежный крем-брюле» в наличии: 0	Сообщение «Нежный крем-брюле: нет в наличии»	Тест пройден (рисунок В.3)
4	Наименование: Медовый рулет Количество: 2	Сообщение «Кондитерское изделие уже добавлено»	Тест пройден (рисунок В.4)
	Наименование: Медовый рулет Количество: 1		
5	Наименование: Ореховый трюфель Количество: 7 Количество кондитерских изделий наименования «Ореховый трюфель» в наличии: 5	Сообщение «Кондитерское изделие Ореховый трюфель в наличии: 5», успешное добавление позиции	Тест пройден (рисунки В.5-В.6)
6	Наименование: Карамельная трость Количество: 1 Количество кондитерских изделий наименования «Карамельная трость» в наличии: 5	Успешное добавление позиции	Тест пройден (рисунок В.7)

По результатам функционального тестирования сделан вывод о том, что система исправно выполняет добавление позиций при создании заказа.

4.2 Тестирование API

Выполнение основных функций разработанной информационной системы осуществляется посредством HTTP-запросов. Эти запросы формируют API-системы, что делает его важной частью всей системы, от которой зависит работа всего приложения.

Для тестирования API выбран метод тестирования запросов. Данный метод позволяет убедиться в том, что запросы выполняются правильно и возвращают ожидаемый результат [16]. Для тестирования запросов создан сценарий опроса API:

- загрузка домашней страницы;
- загрузка страницы «Каталог»;
- получение всех кондитерских изделий;
- загрузка домашней страницы;
- загрузка страницы «Создание заказа»;
- получение всех кондитерских изделий;
- добавление в корзину позиции с количеством, не превышающим количество кондитерского изделия в наличии;
- просмотр корзины;
- добавление в корзину позиции с количеством, превышающим количество кондитерского изделия в наличии;
- просмотр корзины;
- создание заказа;
- очистка корзины;
- загрузка домашней страницы;
- загрузка страницы «Просмотр заказов»;
- просмотр заказов.

Код сценария приведен в листинге 7.

Листинг 7 – Код сценария тестирования запросов

```
GET http://localhost:8763/v1/index
HTTP/1.1 200

GET http://localhost:8763/v1/index/products
HTTP/1.1 200

GET http://localhost:8763/v1/products
HTTP/1.1 200

GET http://localhost:8763/v1/index
HTTP/1.1 200

GET http://localhost:8763/v1/index/create_order
HTTP/1.1 200

GET http://localhost:8763/v1/products
HTTP/1.1 200

POST http://localhost:8763/v1/card
Content-Type: application/json{
    "product_name": "Медовый рулет",
    "quantity": 2}
HTTP/1.1 200

GET http://localhost:8763/v1/cards
HTTP/1.1 200

POST http://localhost:8763/v1/card
Content-Type: application/json{
    "product_name": "Воздушный зефир",
    "quantity": 8
}
HTTP/1.1 200

GET http://localhost:8763/v1/cards
HTTP/1.1 200

POST http://localhost:8763/v1/orders
HTTP/1.1 200

DELETE http://localhost:8763/v1/cards
HTTP/1.1 200


GET http://localhost:8763/v1/index
HTTP/1.1 200

GET http://localhost:8763/v1/index/orders
```

```
HTTP/1.1 200

GET http://localhost:8763/v1/orders
HTTP/1.1 200
```

Результат выполнения сценария представлен на рисунке 24.

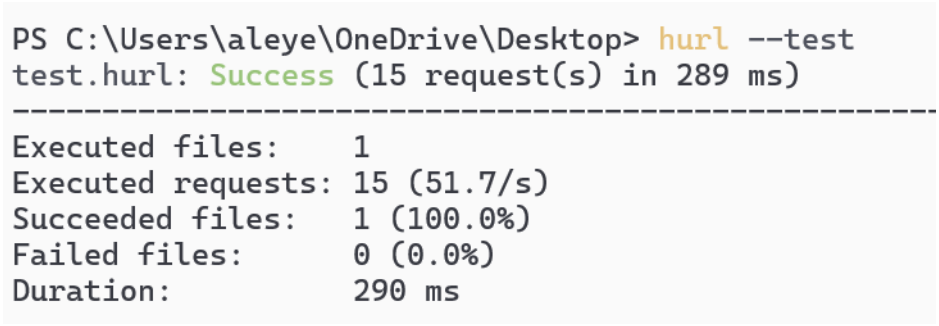

Рисунок 24 – Результат выполнения сценария тестирования запросов

Тестирование показало, что все запросы вернули успешный результат. Выполнение сценария заняло 290 миллисекунд.

Заключение

В процессе выполнения выпускной квалификационной работы проведен анализ предметной области, анализ аналогичных разрабатываемой информационной системе программных продуктов, определены требования к системе.

На основе проведенного анализа было выполнено проектирование программных средств и структур данных системы.

Проведен анализ выбранных инструментальных средств проектирования и разработки.

Произведено планирование разработки и оценка бюджета проекта. При планировании разработки составлена диаграмма Ганта. Оценка бюджета произведена на основе зарплатных отчетов.

В рамках реализации информационной системы разработаны базы данных, серверная часть приложения и пользовательский интерфейс.

Составлено несколько моделей разрабатываемой системы или ее компонентов при помощи UML-диаграмм: диаграмма вариантов использования, диаграмма компонентов, диаграмма деятельности, диаграмма классов, диаграмма последовательности.

Выполнено тестирование разработанных программных средств методом функциональных диаграмм и тестирования запросов. Получены положительные результаты тестирования.

Список использованных источников

1. 1С:Хлебобулочное и кондитерское производство 2. Модуль для 1С:ERP 2 // Группа компаний «1С-Рарус». [Электронный ресурс]. URL: https://rarus.ru/1c-branches/bakery/avtomatizatsiya-konditerskogo-proizvodstva/article-1c-hlebobulochnoe-i-konditerskoe-proizvodstvo-2-modul-dlya-1c-erp-2/ (дата обращения:).

2. Из чего состоит Restik // Общество с ограниченной ответственностью «Рестик». [Электронный ресурс]. URL: https://restik.com/blog/iz-chego-sostoit-restik/ (дата обращения:).

3. Сайт для кондитера без разработчиков – Sweetify Бизнес // Общество с ограниченной ответственностью «Свитифай». [Электронный ресурс]. URL: https://business.sweetify.ru/ (дата обращения:).

4. Балашова, И.Ю. Современные информационные технологии в проектировании программных систем и комплексов : учеб. Пособие / И.Ю. Балашова, Д.В. Такташкин ; под ред. П.П. Макарычева. – Пенза : Изд-во ПГУ, 2019. – 106 с.

5. Visual Paradigm // Visual Paradigm International Ltd. [Электронный ресурс]. URL: https://www.visual-paradigm.com/ (дата обращения: 20.05.2022).

6. PlantUML // Ezoic Inc. [Электронный ресурс]. URL: https://plantuml.com/ru/ (дата обращения:).

7. About Python // Python Software Foundation. [Электронный ресурс]. URL: https://www.python.org/about/ (дата обращения:).

8. FastAPI // tiangolo. [Электронный ресурс]. URL: https://fastapi.tiangolo.com/ (дата обращения:).

9. SQLAlchemy – The Database Toolkit for Python // SQLAlchemy authors and contributors. [Электронный ресурс]. URL: https://www.sqlalchemy.org/ (дата обращения:).

10. Asyncpg – asyncpg Documentation // The asyncpg authors and contributors. [Электронный ресурс]. URL: https://magicstack.github.io/asyncpg/current/# (дата обращения:).

11. Jinja Documentation (3.1.x) // Palets. [Электронный ресурс]. URL: https://jinja.palletsprojects.com/en/stable/ (дат обращения:).

12. PyCharm: The only Python IDE you need // JetBrains s.r.o. [Электронный ресурс]. URL: https://www.jetbrains.com/pycharm/ (дата обращения:).

13. Микросервисная архитектура // Atlassian. [Электронный ресурс]. URL: https://www.atlassian.com/ru/microservices/microservices-architecture (дата обращения:).

14. Зарплаты IT-специалистов во второй половине 2024: +19% к предыдущему полугодию // Habr. [Электронный ресурс]. URL: https://habr.com/ru/specials/883242/ (дата обращения:).

15. Майерс Г. Искусство тестирования программ / Г. Майерс – М.: Финансы и статистика, 1982. – 176 с.

16. Тестирование API: типы, принципы, инструменты – как тестировать API // Общество с ограниченной ответственностью «Яндекс». [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/kak-testirovat-api/ (дата обращения:).

Приложение А

(рекомендуемое)

Графическая часть

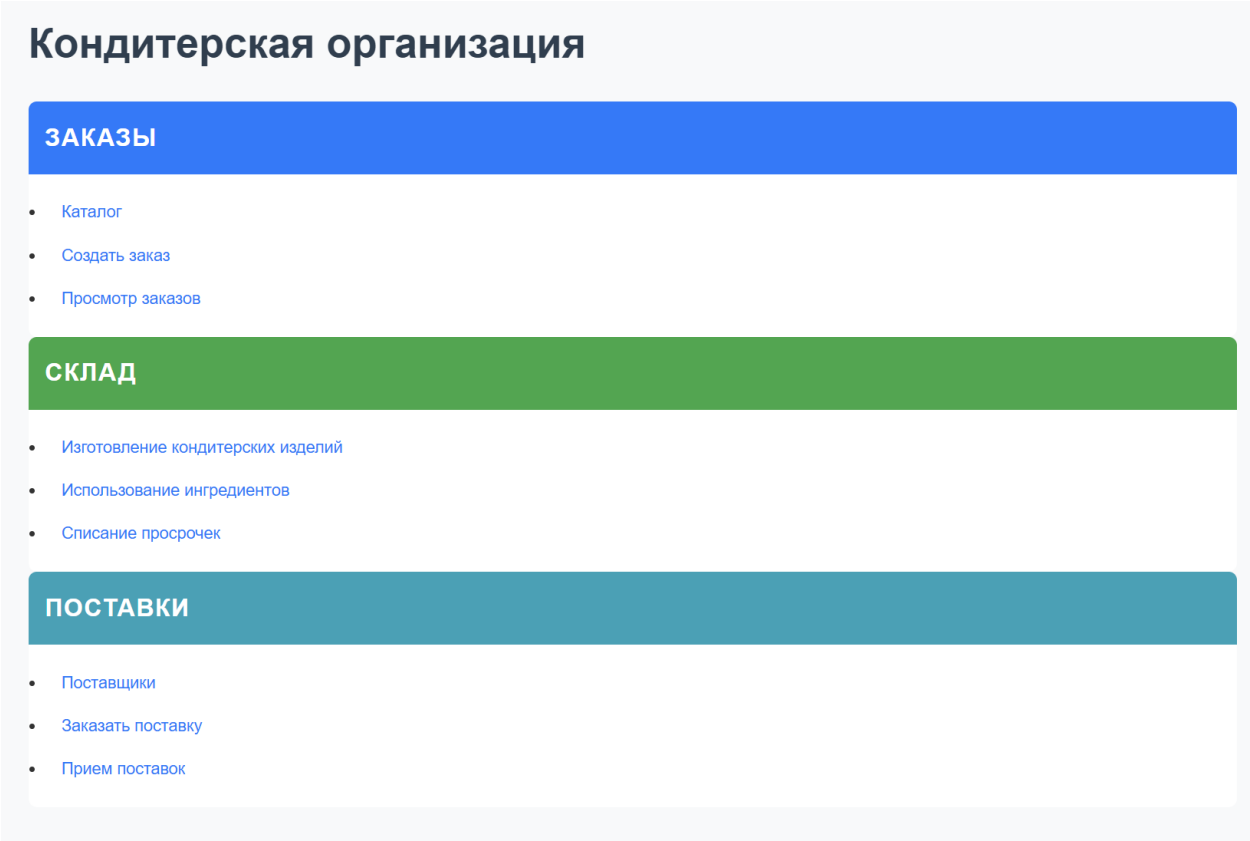

Рисунок А.1 – Внешний вид домашней страницы

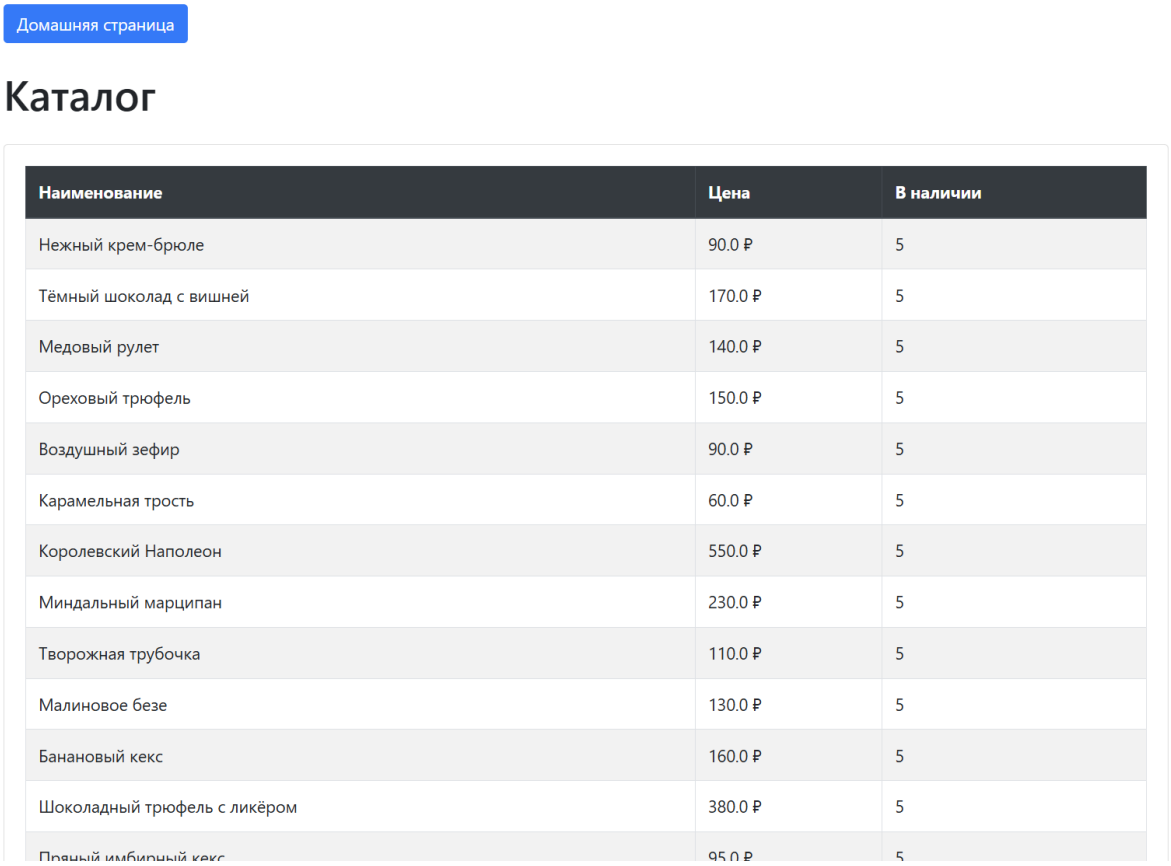

Рисунок А.2 – Внешний вид страницы «Каталог»

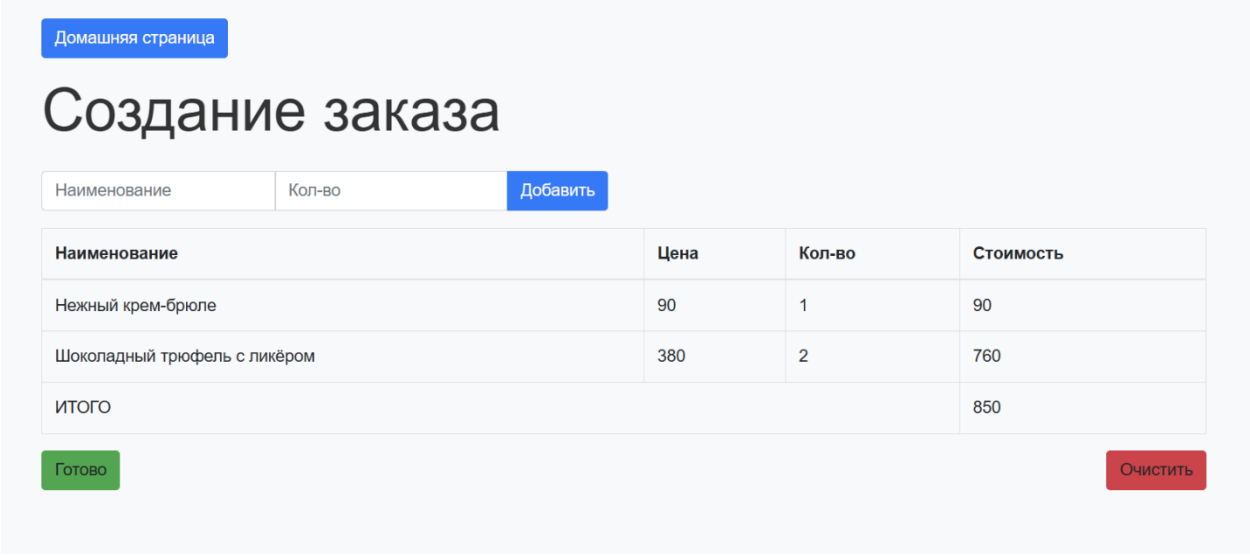

Рисунок А.3 – Внешний вид страницы «Создание заказа»

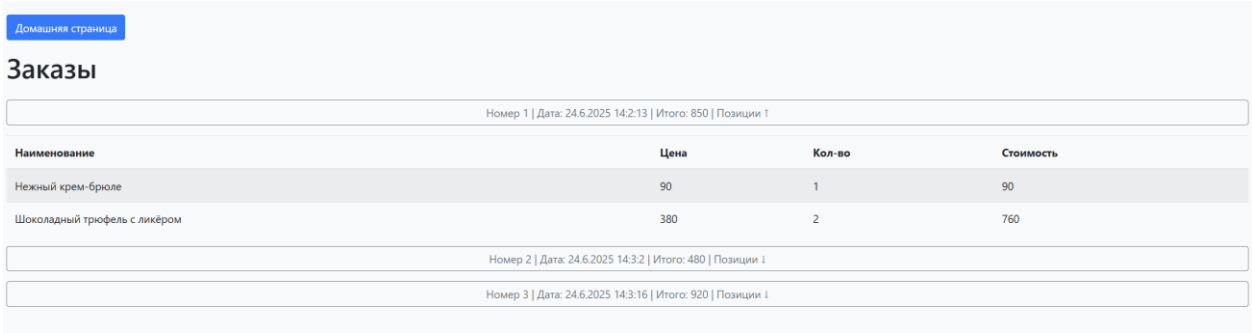

Рисунок А.4 – Внешний вид страницы «Просмотр заказов»

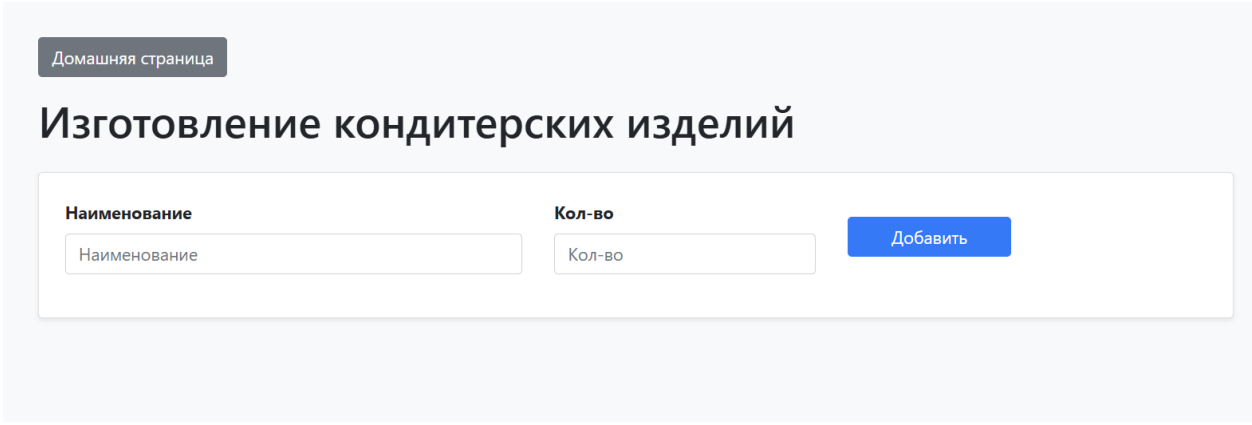

Рисунок А.5 – Внешний вид страницы «Изготовление кондитерских изделий»

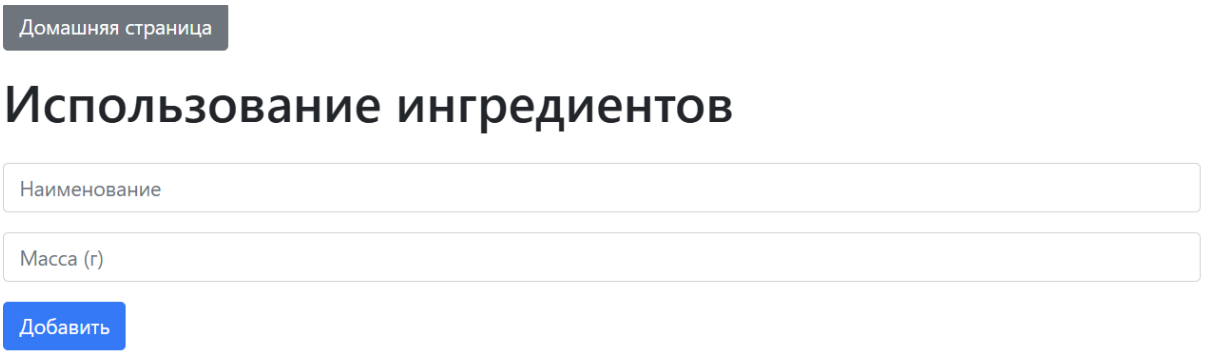

Рисунок А.6 – Внешний вид страницы «Использование ингредиентов»

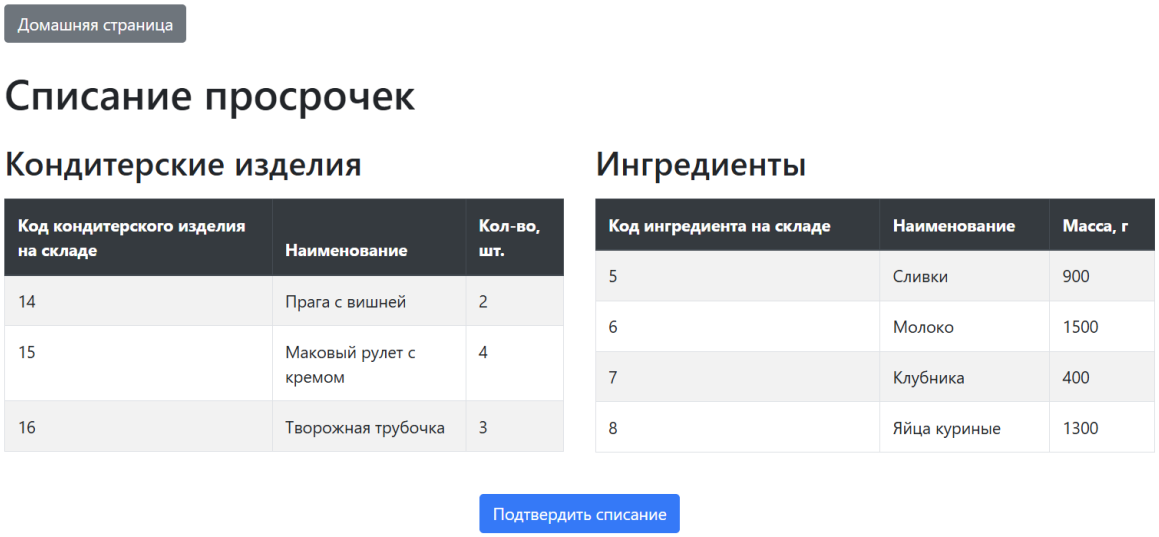

Рисунок А.7 – Внешний вид страницы «Списание просрочек»

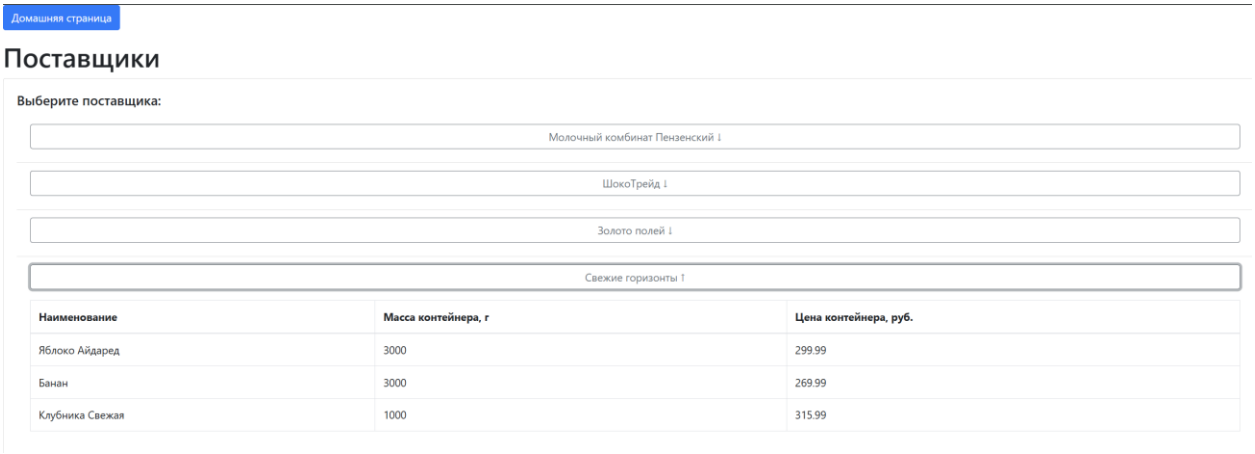

Рисунок А.8 – Внешний вид страницы «Поставщики»

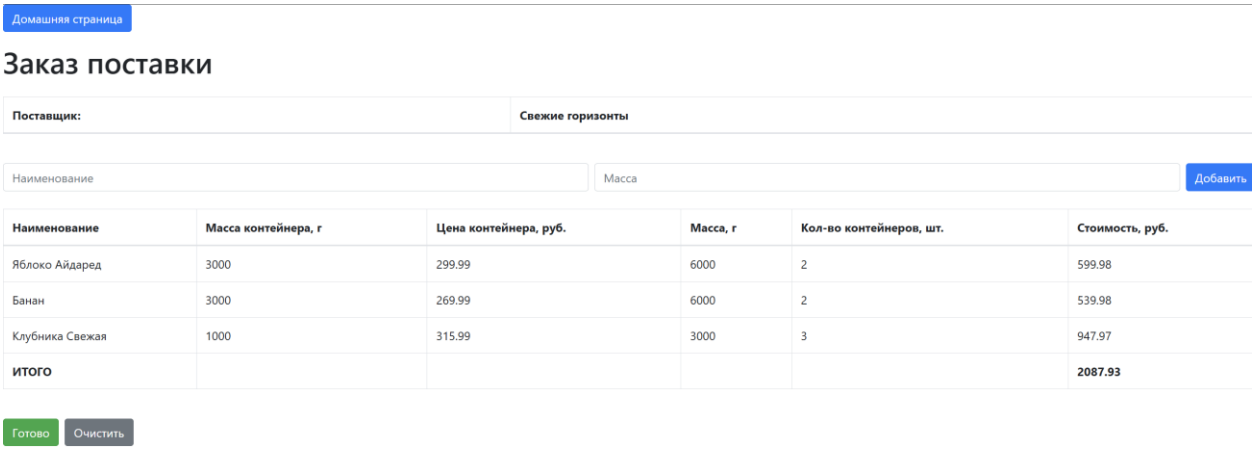

Рисунок А.9 – Внешний вид страницы «Заказ поставки»

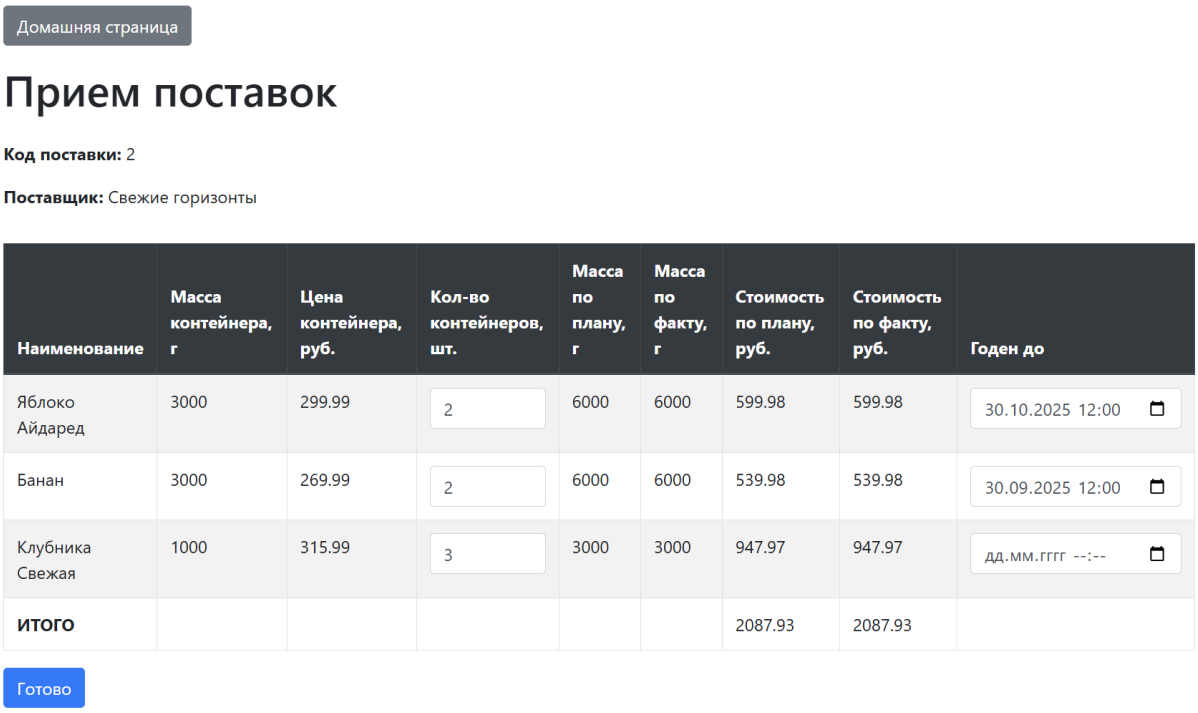

Рисунок А.10 – Внешний вид страницы «Прием поставок»

Приложение Б

(обязательное)

Листинг программы

Листинг Б.1 – Файл main.py

```
if __name__ == '__main__':
    workers = 1
    if workers > 1:
        import platform

        if platform.system() == "Windows":
            # https://github.com/encode/uvicorn/issues/342
            raise SystemError("uvicorn is unable to run multiple
workers with Windows. Please, set workers=1")
        delegated_app = "src.orders.app:app"
    else:
        delegated_app = app

    uvicorn.run(
        delegated_app,
        host="0.0.0.0",
        port=8763,
        workers=workers,)
```

Листинг Б.2 – Файл app.py

```
from src.orders.events import setup_tables
from src.orders.exception_handlers import load_exception_handlers
from src.orders.middlewares import set_session
from src.orders.controllers import router as backend_router
from src.orders.frontend import router as frontend_router

__all__ = ["app"]

def init_app(a: FastAPI) -> FastAPI:
    load_exception_handlers(a)

    a.mount("/static", StaticFiles(directory="../../templates"),
name="templates")

    # noinspection PyUnresolvedReferences
    include_router = a.router.include_router

    include_router(backend_router,
default_response_class=JSONResponse, tags=["Shop"], prefix="/v1")
    include_router(frontend_router,
default_response_class=HTMLResponse, tags=["Frontend"],
prefix="/v1")

    # a.router.include_router(router)

    # noinspection PyTypeChecker
    a.add_middleware(BaseHTTPMiddleware, dispatch=set_session)
```

```
    a.add_event_handler("startup", setup_tables)

    return a
initial_app = FastAPI(title="Shop", debug=True)

app = init_app(initial_app)
```

Листинг Б.3 – Файл controllers.py

```
from src.orders.depends import get_session
from src.orders.dto import AddOrder, PositionOutput,
PositionInput, OrderOutput
from src.orders.models import Order, Product, Position, CardItem

router = APIRouter()

@router.get(
    "/products",
    tags=["Кондитерские изделия"],
    name="Все кондитерские изделия",
    response_model=list[Product],
)
async def get_all_products(session: AsyncSession =
Depends(get_session))->list[Product]:
    rows = await session.scalars(select(Product))
    res = rows.all()
    return res


@router.post("/positions")
async def create_position(position_input: PositionInput, session:
AsyncSession = Depends(get_session)):
    rows = await
session.scalars(select(Product).where(Product.name ==
position_input.product_name))
    product = rows.one_or_none()
    if product is None:
        message = f"Product {position_input.product_name} not
found"
        position_output = None
    elif product.quantity == 0:
        message = f"Product {position_input.product_name} is out
of stock"
        position_output = None
    elif product.quantity < position_input.quantity:
        message = f"Кондитерское изделие
{position_input.product_name} в наличии: {product.quantity}"
        position_output = PositionOutput(
            product_name=product.name,
            price=product.price,
            quantity=product.quantity,
            subtotal=product.price * product.quantity
```

```
        )
    else:
        message = "ok"
        position_output = PositionOutput(
            product_name=product.name,
            price=product.price,
            quantity=position_input.quantity,
            subtotal=product.price * position_input.quantity
        )
    return {"message": message, "data": position_output}


@router.post("/card")
async def add_to_card(position_input: PositionInput, session:
AsyncSession = Depends(get_session)):
    position_output = await create_position(position_input,
session=session)
    msg = position_output["message"]
    data = position_output["data"]
    message = msg
    card_item = None

    if data is not None:

        card_item = CardItem(
            name=data.product_name,
            price=data.price,
            quantity=data.quantity,
            subtotal=data.subtotal
        )
        session.add(card_item)
        await session.commit()

    return {"message": message, "data": card_item}

@router.get("/cards")
async def get_card(session: AsyncSession = Depends(get_session)):
    rows = await session.scalars(select(CardItem))
    res = rows.all()
    return {"card": res, "len": len(res)}


@router.delete("/cards")
async def delete_card(session: AsyncSession =
Depends(get_session)):
    stmt = delete(CardItem)
    await session.exec(stmt)
    await session.commit()
    return {"ok"}

@router.post("/orders")
async def create_oder(order: AddOrder | None = None, session:
```

```
AsyncSession = Depends(get_session)):
    order = order if order else AddOrder()

    my_order = Order.model_validate(order)
    session.add(my_order)
    await session.commit()
    await session.refresh(my_order)
    new_order_id = my_order.id

    card = await get_card(session=session)
    card_items = card["card"]

    total = sum([c.subtotal for c in card_items])
    poses = []
    for card_item in card_items:
        pos = Position(
            product_name=card_item.name,
            price=card_item.price,
            quantity=card_item.quantity,
            subtotal=card_item.subtotal,
            order_id=new_order_id)
        poses.append(pos)
        session.add(pos)
        await session.commit()
        stmt = update(Product).where(Product.name ==
card_item.name).values(quantity=Product.quantity -
card_item.quantity)
        await session.exec(stmt)
        await session.commit()

    stmt = update(Order).where(Order.id ==
new_order_id).values(total=total, completed = True)
    await session.exec(stmt)
    await session.commit()
    await session.refresh(my_order)
    await delete_card(session=session)
    return {"order": my_order, "poses": poses}



@router.get("/orders")
async def get_orders(session: AsyncSession =
Depends(get_session)):
    orders_table_rows = await session.scalars(select(Order))
    orders_table = orders_table_rows.all()
    list_of_orders = []
    for order in orders_table:
        list_of_positions = []
        poses_table_rows = await
session.scalars(select(Position).where(Position.order_id ==
order.id))
        poses_table = poses_table_rows.all()
```

```
        for pos in poses_table:
            list_of_positions.append(PositionOutput(
                product_name=pos.product_name,
                price=pos.price,
                quantity=pos.quantity,
                subtotal=pos.subtotal
            ))
        order_json = OrderOutput(
            order_id=order.id,
            date=order.date,
            total=order.total,
            positions=list_of_positions)
        list_of_orders.append(order_json)
    return list_of_orders
```

Листинг Б.4 – Файл depends.py

```
from sqlalchemy.ext.asyncio import AsyncSession
from starlette.requests import Request


async def get_session(request: Request) -> AsyncSession:  #
pyright: ignore [reportInvalidTypeForm]
    return request.state.session
```

Листинг Б.5 – Файл dto.py

```
from datetime import datetime

from pydantic import BaseModel, Field


class AddOrder(BaseModel):
    date: datetime = Field(default_factory=datetime.now)
    total: float = 0



class PositionInput(BaseModel):
    product_name: str
    quantity: int

class PositionOutput(BaseModel):
    product_name: str
    price: float
    quantity: int
    subtotal: float

class OrderOutput(BaseModel):
    order_id: int
    date: datetime
```

```
    total: float
    positions: list[PositionOutput]

class Sas(BaseModel):
    message: str
    data: PositionOutput
```

Листинг Б.6 – Файл events.py

```
from sqlmodel import SQLModel
from src.orders.globals import AsyncLocalSession

async def setup_tables():
    async with AsyncLocalSession() as session:
        connection = await session.connection()
        await connection.run_sync(SQLModel.metadata.drop_all)
        await connection.commit()
        await connection.run_sync(
            SQLModel.metadata.create_all # pyright: ignore # noqa
        )
        await connection.commit()
```

Листинг Б.7 – Файл exception_handlers.py

```
from asyncpg import ForeignKeyViolationError, UniqueViolationError
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse
from starlette import status
from sqlalchemy.exc import DBAPIError, IntegrityError


__all__ = ["load_exception_handlers"]


def _get_field_name_data_for_integrity_error(exc_text: str):
    end_field_ind = exc_text.find(")=")
    start_field_ind = exc_text.rfind("(", 0, end_field_ind)
    end_date_ind = exc_text.find(")", end_field_ind + 2)
    field_name = exc_text[start_field_ind + 1 : end_field_ind]
    field_data = exc_text[end_field_ind + 3 : end_date_ind]
    return field_name, field_data


def load_exception_handlers(app: FastAPI) -> FastAPI:


    # noinspection PyUnusedLocal,PyUnresolvedReferences
    def sql_exception_handler(request: Request, exc:
IntegrityError) -> JSONResponse:
```

```
        if exc.orig.sqlstate == ForeignKeyViolationError.sqlstate:
# pyright: ignore [reportAttributeAccessIssue,
reportOptionalMemberAccess]
            return JSONResponse(
                status_code=status.HTTP_404_NOT_FOUND,

content=f"'{(_get_field_name_data_for_integrity_error(repr(exc)))[
0]}' не существует"
            )
        elif exc.orig.sqlstate == UniqueViolationError.sqlstate:
# pyright: ignore [reportAttributeAccessIssue,
reportOptionalMemberAccess]
            field_name, field_value =
_get_field_name_data_for_integrity_error(repr(exc))
            return JSONResponse(
                status_code=status.HTTP_409_CONFLICT,
                content=f"'{field_name}' со значением
`{field_value}` уже есть в базе"
            )
        else:
            return JSONResponse(
                status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                content=repr(exc))


    # noinspection PyTypeChecker
    app.add_exception_handler(IntegrityError,
sql_exception_handler)  # pyright: ignore [reportArgumentType]
    # noinspection PyTypeChecker
    app.add_exception_handler(DBAPIError, sql_exception_handler)
# pyright: ignore [reportArgumentType]

    return app
```

Листинг Б.8 – Файл frontend.py

```
router = APIRouter()


@router.get("/index", name="home")
async def home(request: Request) :
    return templates.TemplateResponse(
        request=request,
        name="index.html")


@router.get("/index/products", name="products")
async def get_prods(
    request: Request,
    session: AsyncSession = Depends(get_session),):
    prods = await get_all_products(session)
```

```
    return templates.TemplateResponse(
        request=request,
        name="get_prods.html",
        context={"prods": prods})


@router.get("/index/create_order", name="create_order")
async def input_form(request: Request, session: AsyncSession =
Depends(get_session)):

    await delete_card(session)
    all_prods = await get_all_products(session)

    return templates.TemplateResponse(
        request=request,
        name="create_order.html",
        context={
            "all_prods": all_prods})

@router.get("/index/orders", name="orders")
async def get_orders(request: Request):
    return templates.TemplateResponse(
        request=request,
        name="orders.html")


@router.get("/index/add_product", name="add_product")
async def add_product(request: Request, session: AsyncSession =
Depends(get_session)):
    all_prods = await get_all_products(session)
    return templates.TemplateResponse(
        request=request,
        name="add_product.html",
        context={
            "all_prods": all_prods})


@router.get("/index/use_ingredient", name="use_ingredient")
async def use_ingredient(request: Request):
    return templates.TemplateResponse(
        request=request,
        name="use_ingredient.html")

@router.get("/index/out_of_date", name="out_of_date")
async def out_of_date(request: Request):
    return templates.TemplateResponse(
        request=request,
        name="out_of_date.html")

@router.get("/index/suppliers", name="suppliers")
async def suppliers(request: Request):
    return templates.TemplateResponse(
```

```
        request=request,
        name="suppliers.html")

@router.get("/index/order_supply", name="order_supply")
async def order_supply(request: Request):
    return templates.TemplateResponse(
        request=request,
        name="order_supply.html")

@router.get("/index/complete_supply", name="complete_supply")
async def complete_supply(request: Request):
    return templates.TemplateResponse(
        request=request,
        name="complete_supply.html")
```

Листинг Б.9 – Файл globals.py

```
DATABASE_CONNECTION_URL = URL.create(
    drivername="postgresql+asyncpg",
    host="postgres" if bool(os.environ.get("DOCKER", False)) else
"localhost",
    port=5484,
    database="shop",
    username="employee",
    password="PASSword",
)

async_engine = create_async_engine(
    DATABASE_CONNECTION_URL,
    echo=True,
    pool_size=1,
)

AsyncLocalSession: Callable[[], AsyncSession] = sessionmaker( #
noqa
    bind=async_engine,  # pyright: ignore
    class_=AsyncSession,
    autoflush=True,
    expire_on_commit=False,
)
AsyncSession.exec = AsyncSession.execute  # pyright: ignore
[reportAttributeAccessIssue]

DB = declarative_base()
SQLModel.metadata = DB.metadata
```

Листинг Б.10 – Файл middlewares.py

```
from typing import Callable
from fastapi import Request, Response
from src.orders.globals import AsyncLocalSession
```

```
async def set_session(request: Request, handler: Callable) ->
Response:
    async with AsyncLocalSession() as session:
        request.state.session = session
        response: Response = await handler(request)
    return response
```

Листинг Б.11 – Файл models.py

```
from datetime import datetime
from typing import ClassVar

from pydantic import field_validator
from sqlmodel import SQLModel, Field, Column
from sqlalchemy import  String, Sequence, Integer, ForeignKey


from src.orders.utils import TapMixin
from src.orders.validators import datetime_validator


class Product(SQLModel, TapMixin, table=True):
    __tablename__: ClassVar[str] = "products"  # pyright: ignore
[reportIncompatibleVariableOverride]

    name: str = Field(sa_column=Column(String, primary_key=True))
    price: float
    quantity: int


class Order(SQLModel, TapMixin, table=True):
    __tablename__: ClassVar[str] = "orders"  # pyright: ignore
[reportIncompatibleVariableOverride]

    id: int | None = Field(None, sa_column=Column(Integer,
Sequence("order_seq_id", start=1), primary_key=True))
    date: datetime = datetime.now()
    total: float = 0
    completed: bool = False


    @field_validator("date", mode="before")
    def validate_date(cls, value) -> datetime:
        return datetime_validator(value)

# Товар уже в чеке
class Position(SQLModel, TapMixin, table=True):
    __tablename__: ClassVar[str] = "positions"  # pyright: ignore
[reportIncompatibleVariableOverride]
```

```
    id: int | None = Field(None, sa_column=Column(Integer,
Sequence("position_seq_id", start=1), primary_key=True))

    product_name: str = Field(sa_column=Column(String,
ForeignKey("products.name", ondelete="CASCADE"), nullable=False))
    price: float
    quantity: int = Field(gt=0)
    subtotal: float

    order_id: int = Field(foreign_key="orders.id", nullable=False)


class CardItem(SQLModel, TapMixin, table=True):
    __tablename__: str = "card_items"

    name: str = Field(sa_column=Column(String, primary_key=True))
    price: float
    quantity: int
    subtotal: float
```

Листинг Б.12 – Файл templates.py

```
import pathlib

from fastapi.templating import Jinja2Templates

print(pathlib.Path(__file__).parent.parent.parent)
templates =
Jinja2Templates(directory=(pathlib.Path(__file__).parent.parent.pa
rent / "templates"))
```

Листинг Б.13 – Файл validators.py

```
import datetime
from typing import Any

def datetime_validator(value: str | datetime.datetime | Any,
remove_tz: bool = True) -> datetime.datetime:
    if isinstance(value, str):
        return datetime.datetime.fromisoformat(value)
    elif isinstance(value, datetime.datetime):
        if remove_tz is True and value.tzinfo is not None:
            return value.replace(tzinfo=None)
        return value
    else:
        raise TypeError(f"Wrong type {type(value)} for datetime
field")
```

Листинг Б.14 – Файл base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}

    {% endblock %}</title>
</head>
<body>
{% block body %}

{% endblock %}
</body>
</html>
```

Листинг Б.15 – Файл index.html

```
{% extends 'base.html' %}

{% block title %}
    Homepage
{% endblock %}

{% block body %}
    <h1>Кондитерская организация</h1>
    <h1>Заказы</h1>
    <h1><a href="{{ url_for("products") }}">Каталог</a></h1>
    <h1><a href="{{ url_for("create_order") }}">Создать
заказ</a></h1>
    <h1><a href="{{ url_for("orders") }}">Заказы</a></h1>

<h1>Склад</h1>
    <h1><a href="{{ url_for("add_products") }}">Изготовление
кондитерских изделий</a></h1>
    <h1><a href="{{ url_for("use_ingredient") }}">Использование
ингредиентов</a></h1>
    <h1><a href="{{ url_for("out_of_date") }}">Списать
просрочки</a></h1>

<h1>Поставки</h1>
    <h1><a href="{{ url_for("suppliers") }}">Поставщики</a></h1>
    <h1><a href="{{ url_for("order_supply") }}">Заказать
поставку</a></h1>
    <h1><a href="{{ url_for("complete_supply") }}">Прием
поставок</a></h1>
{% endblock %}
```

Листинг Б.16 – Файл get_prods.html

```
{% extends 'base.html' %}
```

```
{% block title %}
    Catalog
{% endblock %}

{% block body %}
    <a href="{{ url_for("home") }}">
        Домашняя страница
    </a>
    <h1>Каталог</h1>

    <table>
        <thead>
            <tr>
                <th>Наименование</th>
                <th>Цена</th>
                <th>В наличии</th>
            </tr>
        </thead>
        <tbody>
            {% for prod in prods %}
                <tr>
                    <td>{{ prod.name }}</td>
                    <td>{{ prod.price }}</td>
                    <td>{{ prod.quantity }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
{% endblock %}
```

Листинг Б.17 – Файл create_order.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Create order</title>

</head>
<body>
<a href="{{ url_for("home") }}">Домашняя страница</a>
<h1>Создание заказа заказ</h1>


<div id="input_div">
    <input id="prod_input" list="all_prods"
placeholder="Наименование">
    <datalist id="all_prods">
        {% for prod in all_prods %}
            <option value="{{ prod.name }}"></option>
        {% endfor %}
    </datalist>
    <input id="quantity_input" , type="number" , value="1" ,
min="1" , placeholder="Кол-во">
```

```
    <button id="add_button"
onclick="add_button_function()">Добавить</button>
    <br>
    <br>
</div>

<script>

    var table;
    let body = document.querySelector("body");
    let all_prods = [];
    sas();
    console.log(all_prods);
    let quantity_input =
document.getElementById("quantity_input");


    let prod_input = document.getElementById("prod_input");
    prod_input.addEventListener('input', function () {
        console.log(this.value);
        let prod_name = this.value;
        let prod = all_prods.find(function (prod) {
            return prod.name == prod_name;
        });
        quantity_input.setAttribute("max", prod.quantity);
        console.log(prod.quantity);
        if (prod.quantity == 0) {
            alert(prod_name + ": нет в наличии");
            prod_input.value = "";
            quantity_input.value = 1;}});
    function sas() {
        fetch("/v1/products")
            .then(response => response.json())
            .then(data => {
                data.forEach(item => {
                    all_prods.push(item);});});}
    if (!document.getElementById("table")) {
        table = document.createElement("table");
        table.id = "table";
        body.appendChild(table);
        let thead = document.createElement("thead");
        table.appendChild(thead)
        let trh = document.createElement("tr");
        thead.appendChild(trh);
        let th1 = document.createElement("th");
        th1.innerHTML = "Наименование";
        trh.appendChild(th1);

        let th2 = document.createElement("th");
        th2.innerHTML = "Цена";
        trh.appendChild(th2);
```

```
        let th3 = document.createElement("th");
        th3.innerHTML = "Кол-во";
        trh.appendChild(th3);

        let th4 = document.createElement("th");
        th4.innerHTML = "Стоимость";
        trh.appendChild(th4);


        let tbody = document.createElement("tbody");
        table.appendChild(tbody)

        let tfoot = document.createElement("tfoot");
        let trf = document.createElement("tr");
        let tdf1 = document.createElement("td");
        tdf1.innerHTML = "ИТОГО"
        let tdf2 = document.createElement("td");
        tdf2.id = "total";
        tdf2.innerHTML = 0;
        trf.appendChild(tdf1);
        trf.appendChild(tdf2);
        tfoot.appendChild(trf);
        table.appendChild(tfoot);
        table.style.display = "none";
    }

    async function add_button_function() {  // Use async/await for
cleaner code

        let productInput =
document.getElementById("prod_input").value;
        let quantityInput =
document.getElementById("quantity_input").value;

        if (!productInput || !quantityInput) {
            alert("Выберите наименование кондитерского изделия и
укажите количество");
            return;
        }

        let PositionInput = {
            product_name: productInput,
            quantity: parseInt(quantityInput)};

        try {
            let add_response = await fetch('/v1/card', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'},
                body: JSON.stringify(PositionInput)});

            if (!add_response.ok) {
```

```
                throw new Error(`HTTP error! Status:
${add_response.status}`);
            }

            let resp = await add_response.json();
            if (resp.message != "ok") {
                alert(resp.message)
            }
        } catch (error) {

            alert("Кондитерское изделие уже добавлено");
        }

        let get_card_response = await fetch('/v1/cards');
        let resp = await get_card_response.json();
        console.log(resp);
        console.log(resp.card);
        let card_list = resp.card;

        if (card_list != []) {
            table = document.getElementById("table");
            table.style.display = "unset";

            let tbody = document.querySelector("tbody");
            tbody.innerHTML = "";
            let total = 0;

            for (let i = 0; i < card_list.length; i++) {
                let item = card_list[i];
                let trd = document.createElement("tr");
                let tdn = document.createElement("td");
                tdn.innerHTML = item.name;
                let tdp = document.createElement("td");
                tdp.innerHTML = item.price;
                let tdq = document.createElement("td");
                tdq.innerHTML = item.quantity;
                let tds = document.createElement("td");
                tds.innerHTML = item.subtotal;
                total += item.subtotal;


                trd.appendChild(tdn);
                trd.appendChild(tdp);
                trd.appendChild(tdq);
                trd.appendChild(tds);
                tbody.appendChild(trd);}

            tdf2 = document.getElementById("total");
            tdf2.innerHTML = total;

            let prod_input =
document.getElementById("prod_input");
```

```
            prod_input.value = "";
            let quantity_input =
document.getElementById("quantity_input");
            quantity_input.value = 1;

            async function cancel_button_function() {
                window.location.reload();}

            async function save_button_function() {
                console.log("save")

                let save_response = await fetch('/v1/orders', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json'},
                    body: null});
                let resp = await save_response.json();
                console.log(resp)

                alert("Заказ создан");
                window.location.reload();}


            body.appendChild(document.createElement("br"));
            body.appendChild(document.createElement("br"));
            let save_button = document.createElement("button");
            save_button.id = "save_button";
            save_button.innerHTML = "Готово";
            save_button.onclick = save_button_function;
            if (!document.getElementById("save_button")) {
                body.appendChild(save_button);}


            body.appendChild(document.createElement("br"));
            body.appendChild(document.createElement("br"));
            let cancel_button = document.createElement("button");
            cancel_button.id = "cancel_button";
            cancel_button.innerHTML = "Очистить";
            cancel_button.onclick = cancel_button_function;

            if (!document.getElementById("cancel_button")) {
                body.appendChild(cancel_button);}}}


</script>
</body>
</html>
```

Листинг Б.18 – Файл orders.html

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Orders</title>
</head>
<body>
<a href="{{ url_for("home") }}">
    Домашняя страница
</a>
<h1>Заказы</h1>

<br>
<br>


<script>
    let all_orders = [];
    getOrders();
    console.log(all_orders);
    let body = document.querySelector("body");


    async function getOrders() {
        let response = await fetch("/v1/orders");
        let result = await response.json();

        result.forEach(item => {
            all_orders.push(item);
        });

        all_orders.sort((a,b) => {
            let timeA = new Date(a.timestamp);
            let timeB = new Date( b.timestamp);
            return timeA.getTime() - timeB.getTime();})

        createButtons();}

    function createButtons() {

        if (all_orders.length < 1) {
            let p = document.createElement("p");
            p.textContent = "Заказов пока нет";
            body.appendChild(p);}


        for (let i = 0; i < all_orders.length; i++) {

            let my_date_timestamp = new Date(all_orders[i].date);
            let my_date =
`${my_date_timestamp.getDate()}.${my_date_timestamp.getMonth()}.${
my_date_timestamp.getFullYear()}
${my_date_timestamp.getHours()}:${my_date_timestamp.getMinutes()}:
${my_date_timestamp.getSeconds()} `;

            let div = document.createElement("div");
```

```
            let button = document.createElement("button");
            button.setAttribute("class", "collapse_button");
            button.textContent = "Номер " + all_orders[i].order_id
+ " | Дата: " + my_date + " | Итого: " + all_orders[i].total + " |
Позиции ↓";
            div.appendChild(button);
            body.appendChild(div);
            body.appendChild(document.createElement("br"));
             body.appendChild(document.createElement("br"));


            let pos_div = document.createElement("div");
             pos_div.appendChild(document.createElement("br"));
            pos_div.setAttribute("id", all_orders[i].order_id);
            div.appendChild(pos_div);


            table = document.createElement("table");
            table.setAttribute("class", "table");
            pos_div.appendChild(table);

            let thead = document.createElement("thead");
            table.appendChild(thead);

            let trh = document.createElement("tr");
            thead.appendChild(trh);


            let th1 = document.createElement("th");
            th1.innerHTML = "Наименование";
            trh.appendChild(th1);

            let th2 = document.createElement("th");
            th2.innerHTML = "Цена";
            trh.appendChild(th2);

            let th3 = document.createElement("th");
            th3.innerHTML = "Кол-во";
            trh.appendChild(th3);

            let th4 = document.createElement("th");
            th4.innerHTML = "Стоимость";
            trh.appendChild(th4);

            let tbody = document.createElement("tbody");
            table.appendChild(tbody);

            let positions = all_orders[i].positions;
            for (let j = 0; j < positions.length; j++) {
                let item = positions[j];
```

```
                let trd = document.createElement("tr");
                let tdn = document.createElement("td");
                tdn.innerHTML = item.product_name;
                let tdp = document.createElement("td");
                tdp.innerHTML = item.price;
                let tdq = document.createElement("td");
                tdq.innerHTML = item.quantity;
                let tds = document.createElement("td");
                tds.innerHTML = item.subtotal;

                trd.appendChild(tdn);
                trd.appendChild(tdp);
                trd.appendChild(tdq);
                trd.appendChild(tds);
                tbody.appendChild(trd);}
            pos_div.style.display = "none"

            button.addEventListener("click", function () {
            if (pos_div.style.display === "none") {
                pos_div.style.display = "block";
                button.textContent = "Номер " +
all_orders[i].order_id + " | Дата: " + my_date + " | Итого: " +
all_orders[i].total + " | Позиции ↑";
            } else {
                pos_div.style.display = "none";
                button.textContent = "Номер " +
all_orders[i].order_id + " | Дата: " + my_date + " | Итого: " +
all_orders[i].total + " | Позиции ↓";}});}}

</script>
</body>
</html>
</html>
```

Приложение В

(рекомендуемое)

Результаты тестирования

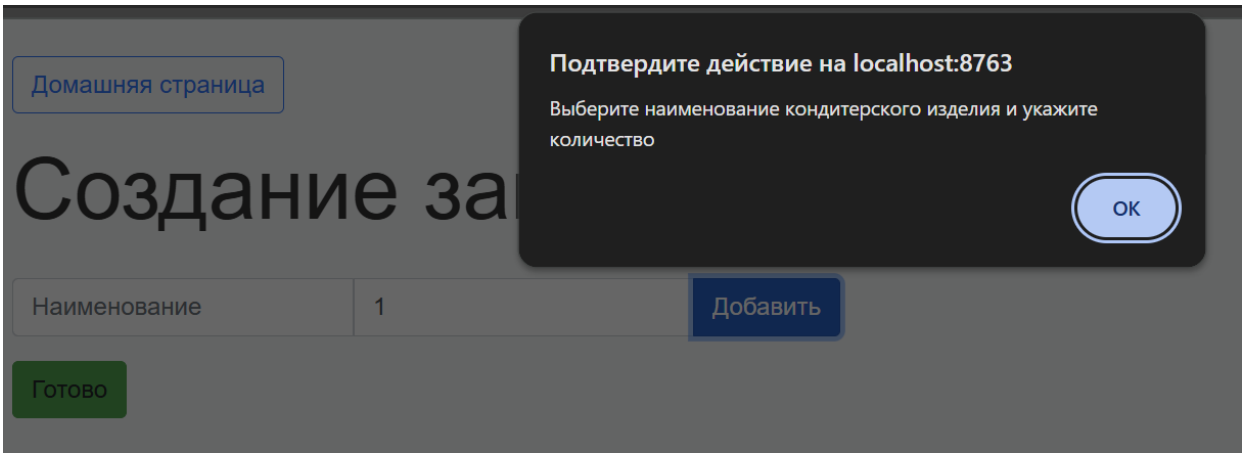

Рисунок В.1 – Результат теста 1

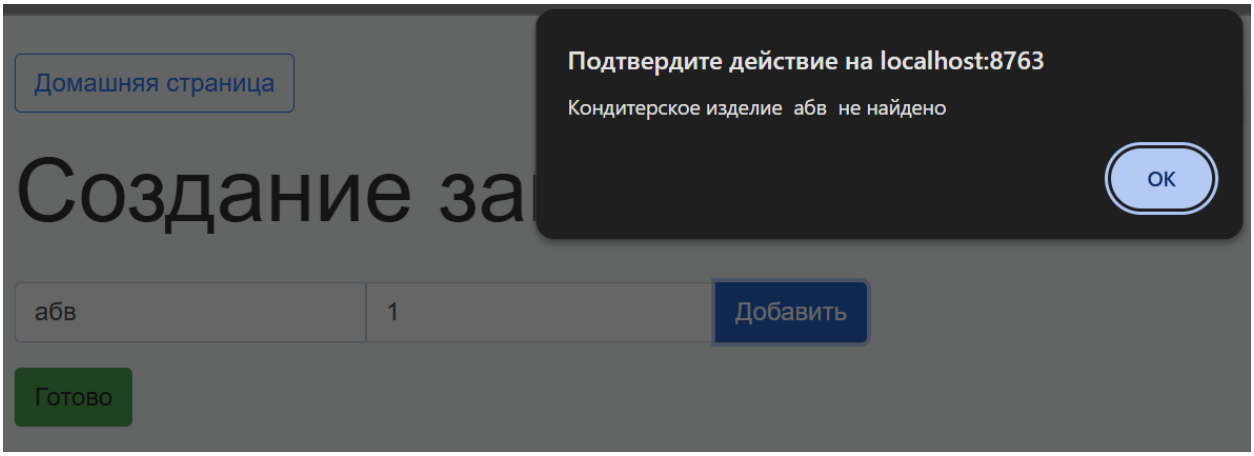

Рисунок В.2 – Результат теста 2

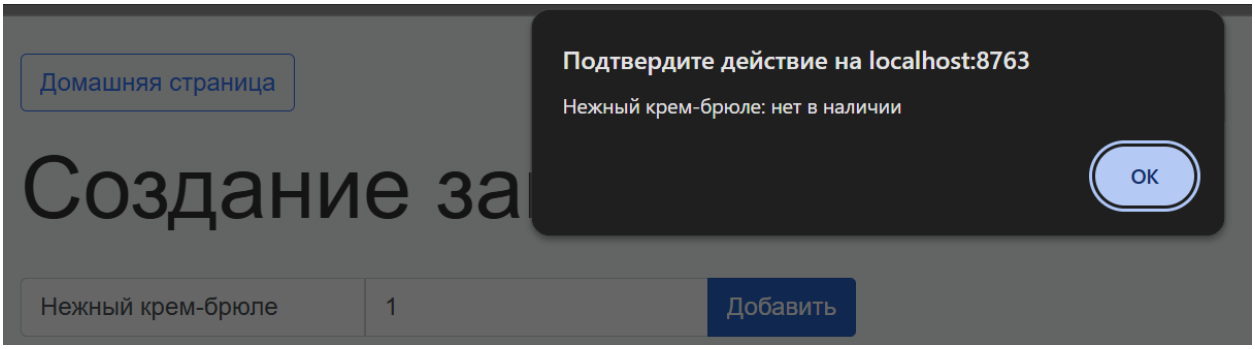

Рисунок В.3 – Результат теста 3

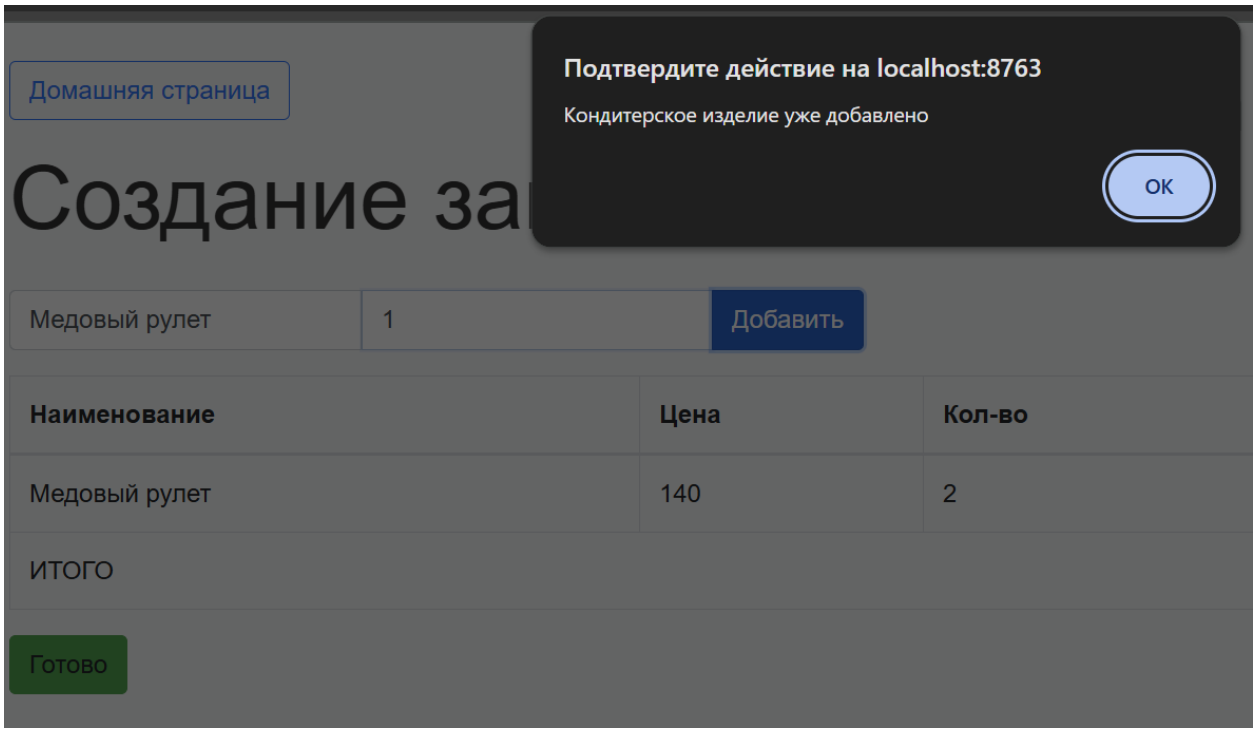

Рисунок В.4 – Результат теста 4

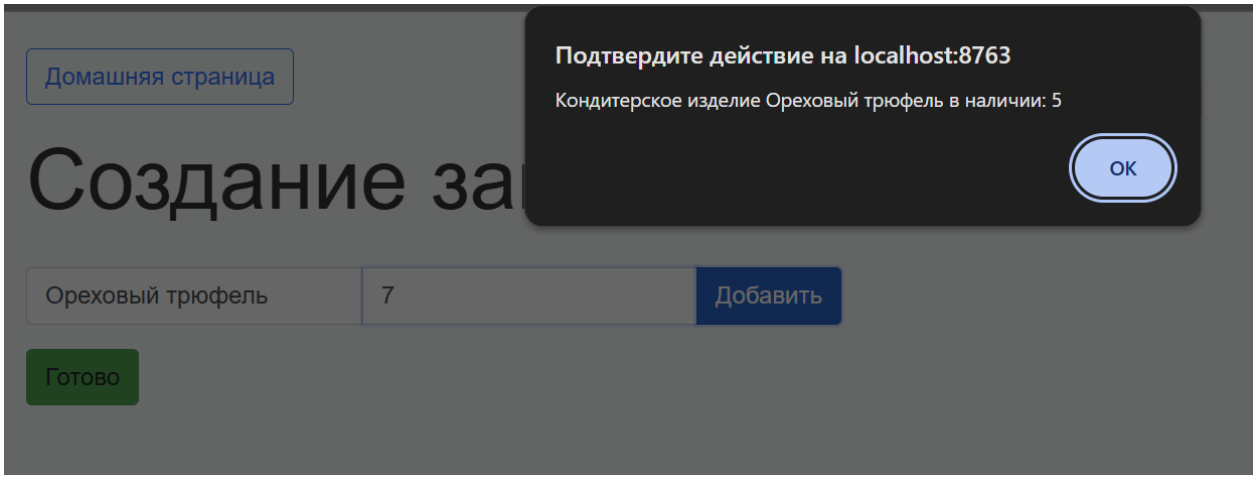

Рисунок В.5 – Результат теста 5 (сообщение)

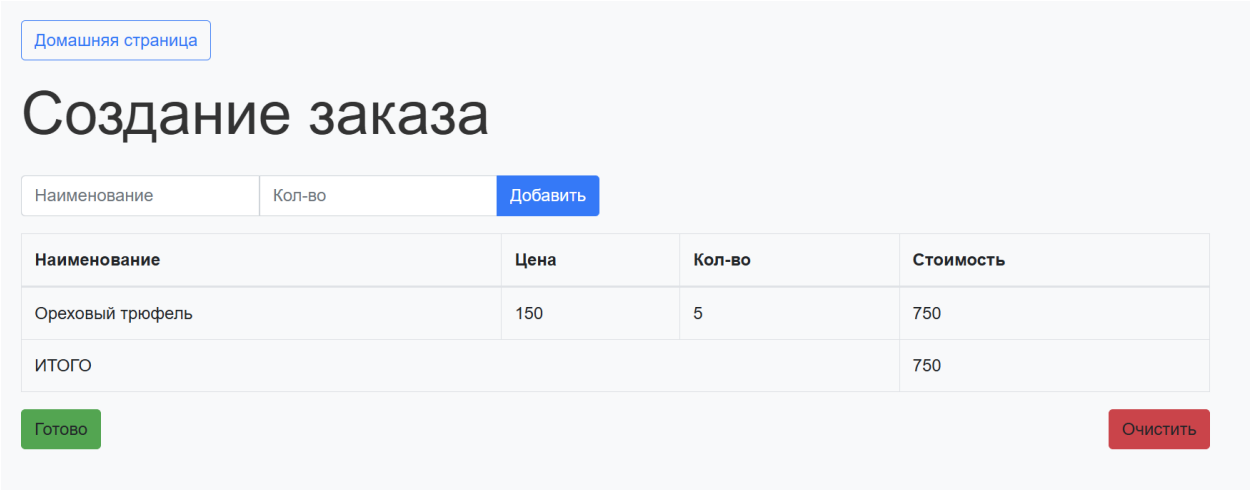

Рисунок В.6 – Результат теста 5 (успешное добавление позиции)

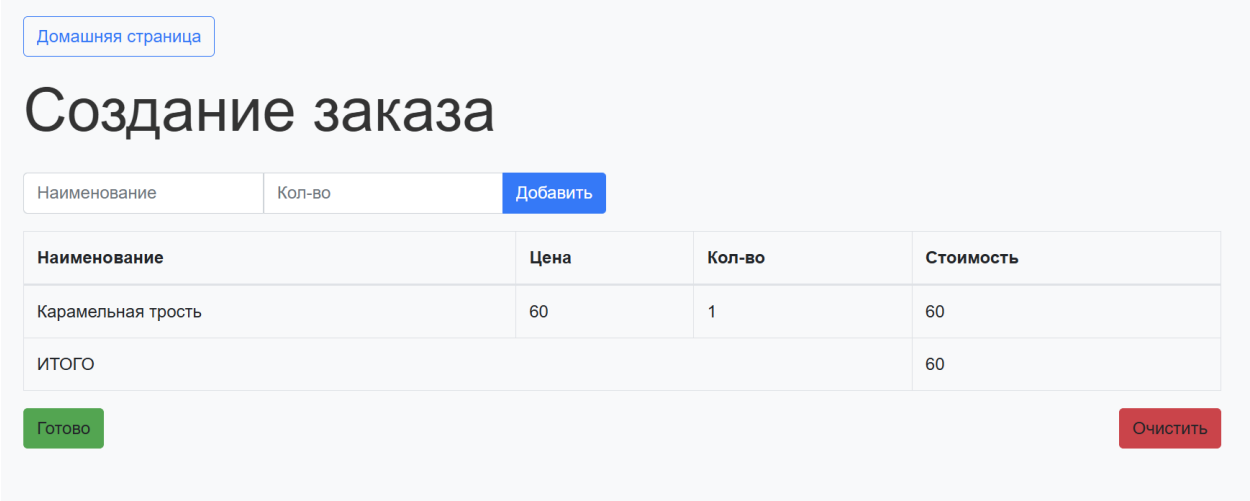

Рисунок В.7 – Результат теста 6