

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МОиПЭВМ

Направление подготовки

09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Информационная система формирования и
сопровождения заказов кондитерской организации**

Студент

(подпись, дата)

Коновалова Елизавета Александровна

(ФИО полностью)

Руководитель

(подпись, дата)

Иванчуков А.Г.

(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

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

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

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

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МОиПЭВМ

«Утверждаю»
Заведующий кафедрой

А.Ю. Козлов

«__» _____ 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Коновалова Елизавета Александровна гр. 19ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Информационная система формирования и сопровождения заказов
кондитерской организации

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Иванчуков А.Г.

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

Разработать информационную систему, обеспечивающую возможность пользователям
осуществлять заказы в кондитерской организации

Основные функции системы:

1. Предоставление каталога кондитерских изделий;
2. Предоставление возможности добавления товаров в корзину;
3. Предоставление возможности формирования заказа пользователем;
4. Предоставление возможности сопровождения заказа администратором;
5. Предоставление возможности просмотра совершённых заказов.

Языки программирования: Rust, JavaScript.

Среды программирования: IntelliJ IDEA, WebStorm.

5. Перечень подлежащих разработке вопросов:

1. анализ предметной области и постановка задачи;
2. проектирование программных средств;
3. реализация программных средств;
4. тестирование программных средств;
5. оформление ВКР.

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%	12.04.2023 – 17.04.2023	
2	Проектирование программных средств	20%	18.04.2023 – 26.04.2023	
3	Реализация программных средств	40%	27.04.2023 – 17.05.2023	
4	Тестирование программных средств	10%	18.05.2023 – 23.05.2023	
5	Оформление пояснительной записки	20%	24.05.2023 – 01.06.2023	

Дата выдачи задания 22 декабря 2022г.

Руководитель бакалаврской работы

(подпись, дата)

Задание к исполнению принял студент

(подпись, дата)

Работу к защите допустить

Декан факультета

(подпись, дата)

Реферат

Пояснительная записка содержит 110 листов, 35 рисунков, 23 таблицы, 23 листинга, 15 использованных источников и 3 приложения.

ИНФОРМАЦИОННАЯ СИСТЕМА, WEB ПРИЛОЖЕНИЕ, RUST, JS, ДИАГРАММЫ UML, КОНДИТЕРСКАЯ ОРГАНИЗАЦИЯ

Объект работы: информационная система формирования и сопровождения заказов кондитерской организации.

Цель работы: разработка информационной системы формирования и сопровождения заказов кондитерской организации.

Результаты работы: в результате работы была спроектирована, разработана и протестирована информационная система формирования и сопровождения заказов кондитерской организации.

Область применения: продажи в кондитерской сфере.

Средства разработки: IntelliJ, WebStorm, Rust, React, PostgreSQL.

					ПГУ 09.03.02 – 08БР191.08 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Конвоалова Е.А.			Информационная система формирования и сопровождения заказов кондитерской организации	Лит.	Лист	Листов
Провер.		Иванчуков А.Г.					3	110
Реценз.						группа 19ВИ1		
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	6
1 Постановка задачи и анализ предметной области	8
1.1 Анализ предметной области.....	8
1.2 Анализ существующих решений	9
1.3 Сравнительный анализ существующих решений	11
1.4 Постановка задачи и анализ функциональных требований.....	14
2 Планирование и расчет бюджета разрабатываемой системы.....	19
3 Проектирование информационной системы	22
3.1 Инструментальные средства и методы проектирования.....	22
3.1.1 Модель «сущность-связь»	22
3.1.2 Язык моделирования UML.....	22
3.2 Проектирование базы данных системы.....	23
3.3 Проектирование серверной части системы	26
3.4 Проектирование клиентской части.....	29
3.5 Проектирование графического интерфейса.....	32
4 Реализация информационной системы	35
4.1 Архитектура разрабатываемой информационной системы	35
4.2 Разработка хранилища данных	36
4.3 Реализация серверной части системы	42
4.4 Реализация клиентской части.....	44
5 Тестирование информационной системы.....	52
5.1 Тестирование формы формирования заказа	52
5.2 Тестирование метрикой Холстеда	58
5.3 Автоматизированное тестирование	61
Заключение	64
Список использованных источников	66
Приложение А. Код серверной части.....	68
Приложение Б. Код клиентской части	81
Приложение В. Результаты тестирования	104

Введение

В настоящее время информационные технологии совершенно точно стали неотъемлемой частью нашей жизни. Бизнес-сфера все больше ориентируется на использование новых компьютерных технологий, чтобы улучшить качество предоставляемых услуг. Так как число различных компаний, осуществляющих реализацию услуг, стремительно растёт, то клиенты становятся более требовательными и избирательными в выборе того или иного поставителя услуг. Поэтому, чтобы иметь прибыльный и перспективный бизнес, организации должны выделяться среди конкурентов.

Далеко не все организации имеют свой собственный сервис для предоставления различных услуг, однако эта возможность является очень удобной для клиента. Одной из ключевых потребностей клиентов является возможность совершать заказы онлайн, без необходимости посещать физический офис или магазин. Онлайн-заказы позволяют клиентам экономить время, избежать очередей и сократить неудобства, связанные с посещением офиса или магазина. Кроме того, клиенты могут ознакомиться с полной информацией о предлагаемых услугах и товарах, изучить ассортимент и сделать выбор.

Автоматизация процессов становится необходимостью для сферы кондитерской деятельности. Приобретение кондитерских изделий на заказ является актуальной темой для многих людей. Однако не каждый клиент готов тратить время на посещение кондитерской организации для оформления заказа. Поэтому оптимальным решением этой проблемы является разработка такой системы, которая позволит клиентам делать заказы онлайн.

Таким образом, разработка приложения для клиентов кондитерской организации является актуальной задачей. Это поможет предоставить клиентам удобный и быстрый сервис заказа, что позволит увеличить продажи и общую эффективность работы организации.

Целью данной работы является разработка информационной системы формирования и сопровождения заказов кондитерской организации. В рамках неё необходимо выполнить следующие задачи:

- провести анализ предметной области;
- спроектировать и разработать информационную систему;
- протестировать разработанное приложение.

Для реализации поставленных задач будут использованы следующие инструментальные средства: среда разработки программного обеспечения JetBrains IntelliJ, JetBrains WebStorm, PostgreSQL. Языки программирования: Rust, JavaScript.

1 Постановка задачи и анализ предметной области

1.1 Анализ предметной области

Кондитерская организация представляет собой предприятие, специализирующееся на изготовлении и продаже кондитерских изделий на заказ. Клиенты, посещая веб-страницу кондитерской организации, имеют возможность просмотреть широкий ассортимент продукции, выбрать понравившиеся товары и добавить их в корзину для последующего оформления заказа. Также важным моментом является возможность отслеживать статус сформированного заказа.

Анализ предметной области предполагает изучение основных аспектов задач и требований, связанных с разработкой и использованием информационной системы для управления заказами в кондитерской организации. К таким аспектам следует отнести:

- процесс формирования заказов;
- взаимодействие с клиентами;
- требования к информационной системе.

Процесс формирования заказов происходит следующим образом. Клиент просматривает ассортимент десертов, выбирает десерт исходя из своих предпочтений и добавляет его в корзину. В корзине отображаются добавленные товары с указанием их количества, которое можно изменять. Также в корзине указывается сумма всего заказа. После того, как клиент закончил выбор десертов, он может оформить заказ, который поступит администратору кондитерской организации и будет обработан. Для оформления заказа клиенту необходимо авторизоваться в системе.

Взаимодействие организации с клиентами заключается в приёме заказов. При оформлении заказа клиент указывает необходимую информацию – фамилию, имя, номер телефона. Заказ поступает администратору, после чего он обрабатывается – администратор передаёт его в работу и уведомляет клиента о статусе его готовности. Клиент в своём личном кабинете может наблюдать свои заказы и отслеживать их статус готовности.

Основным требованием к разрабатываемой информационной системе является обеспечение пользователю такого удобного и интуитивно понятного интерфейса, который позволит ему осуществлять заказы – просматривать каталог продукции, добавлять товары в корзину и формировать заказы.

Таким образом, были рассмотрены основные аспекты предметной области.

1.2 Анализ существующих решений

Одним из важных этапов разработки информационной системы формирования и сопровождения заказов кондитерской организации является анализ существующих решений. Такой анализ позволит выявить особенности некоторых систем, в том числе их преимущества и недостатки. Выявление особенностей информационных систем необходимо для определения собственных ключевых аспектов, которые необходимо реализовать в процессе разработки системы.

В результате анализа существующих решений были рассмотрены три веб-сайта кондитерских организаций, предоставляющие возможность клиентам осуществлять заказы.

В первую очередь была рассмотрена система кондитерской организации «SIT.CAKES». Она имеет привлекательный интерфейс с качественными изображениями. На странице каталога представлено множество категорий десертов, каждая из которых содержит большое количество десертов в разном ценовом сегменте. Также на сайте представлена полная информация о доставке и оплате. Перечисленные моменты можно отнести к преимуществам данной системы.

К недостаткам стоит отнести отсутствие личного кабинета покупателя. Его наличие позволило бы клиентам сохранять свои личные данные в системе, а также просматривать текущие и прошедшие заказы.

Особенностью системы кондитерской «SIT.CAKES» можно считать наличие не только десертов, доступных к заказу, но и некоторых рецептов, которые тоже можно приобрести на сайте, добавив их в корзину. Таким образом,

можно приобрести рецепт десерта и приготовить его самостоятельно. Это может быть преимуществом для некоторых клиентов.

Следующая рассмотренная система – веб-сайт кондитерской «Bakerno». Она имеет не менее привлекательный анимированный интерфейс. Можно отметить и наличие таких страниц, как «Контакты» и «Частые вопросы». Их наличие способствует облегчению взаимодействия с заказчиками. Также в системе представлен личный кабинет пользователя. В нём можно добавлять и редактировать личные данные, а также просматривать совершённые покупки. Все эти особенности являются преимуществами данного сайта.

Недостатком системы можно считать отсутствие страницы корзины – покупка десертов осуществляется с помощью звонка по указанному номеру телефона или сообщения. Данный подход не позволяет делать заказ онлайн, без прямого взаимодействия покупателя с администратором.

Особенность веб-сайта кондитерской «Bakerno» заключается в наличии в продаже авторских курсов от профессионального кондитера. Таким образом, кондитерская предоставляет своим клиентам не только каталог доступной к покупке продукции, но и выбор из множества курсов, обучающих кондитерскому делу. Такой подход позволит привлечь больше клиентов в свою кондитерскую организацию, что не может не быть преимуществом данной системы.

Заключительной системой, которая была проанализирована, является «Кондитерская Юлии Ханакиной». В отличие от других рассмотренных систем, данный сайт не отличается своей презентабельностью. Он не соответствует таким основным принципам построения хорошего интерфейса, как простота, доступность, видимость, принцип обратной связи. На сайте отсутствуют изображения товаров, а осуществить заказ онлайн не представляется возможным.

В данной системе можно выделить и преимущества – наличие подробной информации о самой кондитерской организации, возможность индивидуального заказа и персональные рекомендации десертов.

1.3 Сравнительный анализ существующих решений

Учитывая особенности, преимущества и недостатки рассмотренных систем формирования и сопровождения заказов кондитерской организации, можно сформулировать несколько критериев для их сравнения:

- качество пользовательского интерфейса;
- доступность заказа товаров;
- наличие личного кабинета пользователя;
- доступность информации об организации и товарах.

Критерий «качество пользовательского интерфейса» показывает соответствие системы таким основным принципам и правилам построения интерфейса, как простота, доступность, видимость, наличие обратной связи, привлекательность. Системы будут оцениваться по данному критерию соответствующими числами от 0 до 5. Описания значений данных оценок приведены в таблице 1.

Таблица 1 – Расшифровка оценок критерия «Качество пользовательского интерфейса»

Оценка	Описания
0	Интерфейс не соответствует ни одному из основных принципов удобного интерфейса (простота, доступность, видимость, наличие обратной связи, привлекательность)
1	Интерфейс соответствует одному из основных принципов удобного интерфейса (простота, доступность, видимость, наличие обратной связи, привлекательность)
2	Интерфейс соответствует двум основным принципам удобного интерфейса (простота, доступность, видимость, наличие обратной связи, привлекательность)
3	Интерфейс соответствует трём основным принципам удобного интерфейса (простота, доступность, видимость, наличие обратной связи, привлекательность)
4	Интерфейс соответствует четырём основным принципам удобного интерфейса (простота, доступность, видимость, наличие обратной связи, привлекательность)
5	Интерфейс соответствует всем основным принципам удобного интерфейса (простота, доступность, видимость, наличие обратной связи, привлекательность)

Критерий «доступность заказа товаров» позволяет оценить от 0 до 2 возможность сделать заказ десертов с помощью веб-сайта. Описания значений данных оценок приведены в таблице 2.

Таблица 2 – Расшифровка оценок критерия «Доступность заказа товаров»

Оценка	Описания
0	Заказ сделать невозможно: в системе нет возможности заказа онлайн и нет контактной информации для заказа продукции по телефону или иным способом
1	В системе нет возможности сделать заказ онлайн, но есть контактная информация для осуществления заказа по телефону или иным способом
2	В системе есть возможность сделать заказ онлайн

Критерий «наличие личного кабинета пользователя» позволяет оценить от 0 до 1 наличие в системе страницы, содержащей личный кабинет. Описания значений данных оценок приведены в таблице 3.

Таблица 3 – Расшифровка оценок критерия «Наличие личного кабинета пользователя»

Оценка	Описания
0	Личный кабинет в системе отсутствует
1	Личный кабинет содержится в системе

Критерий «доступность информации об организации и товарах» показывает, насколько в системе доступна информация для покупателей. По данному критерию системы будут оцениваться от 0 до 2. Описания значений данных оценок приведены в таблице 4.

Таблица 4 – Расшифровка оценок критерия «Доступность информации об организации и товарах»

Оценка	Описания
0	В системе не представлена информация ни об организации, ни о товарах
1	В системе представлена информация об организации или о товарах
2	В системе представлена информация и об организации, и о товарах

На основе описанных критериев можно оценить системы кондитерских организаций «SIT.CAKES», «Bakerno» и «Кондитерская Юлии Ханакиной». Сравнительный анализ данных систем по критериям приведен в таблице 5.

Таблица 5 – Расшифровка оценок критерия «Доступность информации об организации и товарах»

	Система кондитерской «SIT.CAKES»	Система кондитерской «Bakerno»	Система кондитерской «Кондитерская Юлии Ханакиной»
Качество пользовательского интерфейса	4	5	1
Доступность заказа товаров	2	1	1
Наличие личного кабинета пользователя	0	1	0
Доступность информации организации товарах	1	2	1

В результате сравнительного анализа данных систем было выявлено, что наиболее удобным и привлекательным интерфейсом обладает система кондитерской «Bakerno», которая оценивается в пять баллов из пяти возможных. Система проста в использовании, имеет всю необходимую информацию, иллюстрации, предоставляет пользователю обратную связь. Хуже всего по этому критерию себя показал сайт «Кондитерская Юлии Ханакиной». Он является довольно простым в использовании, так как содержит лишь несколько основных разделов, которые находятся в общей доступности, но не имеет обратной связи на некоторые действия, не содержит необходимой информации о продукции, содержит светлый текст на светлом фоне, что не является удобным для пользователя.

По критерию «доступность заказа товаров» видно, что сделать онлайн-заказ можно только в кондитерской «SIT.CAKES». Остальные же системы предоставляют пользователю возможность заказа иными способами.

Что касается наличия личного кабинета, то его имеет только система кондитерской «Bakerno».

Полная информация об организации и продаваемой продукции представлена лишь в кондитерской «Bakerno». Кондитерская «SIT.CAKES»

предоставляет информацию лишь о товарах, а «Кондитерская Юлии Ханакиной» – лишь об организации.

Таким образом, из трёх рассмотренных систем наиболее всего отвечающей некоторым потребностям покупателей является веб-сайт кондитерской «Bakerno». По описанным критериям он набрал наивысшую оценку, что говорит об удобстве его использования. Но несмотря на это, данная система не предоставляет пользователю возможности заказа товаров онлайн, хотя наличие этой возможности является одним из важных аспектов подобных систем.

Из анализа существующих систем формирования и сопровождения заказов кондитерской организации можно сделать выводы:

- в системе кондитерской «SIT.CAKES» не хватает личного кабинета пользователя;
- в системе кондитерской «Bakerno» не хватает возможности осуществления заказа десертов онлайн;
- система кондитерской Юлии Ханакиной не обладает привлекательным и правильно функционирующим интерфейсом, возможностью осуществления заказа онлайн и не содержит личный кабинет пользователя.

Также системы кондитерских «SIT.CAKES» и Юлии Ханакиной не предоставляют пользователю достаточной информации о своей организации и представленных товарах.

Таким образом, необходимо учесть данные недостатки при разработке собственной информационной системы.

1.4 Постановка задачи и анализ функциональных требований

В результате анализа существующих систем формирования и сопровождения заказов кондитерских организаций были учтены их преимущества и недостатки. Таким образом, исходя из ранее определенных преимуществ и недостатков систем, можно сформулировать собственные требования к разрабатываемой системе, чтобы она в большей мере удовлетворяла некоторым потребностям её пользователей:

- качественный пользовательский интерфейс;
- доступность заказа товаров онлайн;
- наличие личного кабинета пользователя;
- доступность информации о товарах;
- доступность информации о кондитерской организации;
- возможность осуществления индивидуального заказа.

Таким образом, задача на данную работу заключается в разработке такой информационной системы, которая будет удовлетворять данным требованиям. Реализация поставленных требований позволит повысить оценки разрабатываемой системы по некоторым критериям в сравнении с другими существующими аналогичными системами.

Разрабатываемая информационная система формирования и сопровождения заказов кондитерской организации должна предоставлять клиентам удобный интерфейс для заказа кондитерских изделий, а администратору – для сопровождения этих заказов.

Приложение должно включать в себя главную страницу, страницы с каталогом продукции, страницу с информацией об организации, корзину для отображения добавленных товаров, страницу личного кабинета, страницу для просмотра совершённых заказов, страницу авторизации и панель администратора для соответствующей роли. Основные функциональные требования, предъявляемые к информационной системе для роли клиента, следующие:

- возможность просмотра каталога кондитерских изделий;
- просмотр общей и контактной информации об организации;
- запрос индивидуального заказа;
- возможность добавления товаров в корзину;
- возможность оформления заказа;
- возможность регистрации и авторизации;
- возможность редактировать личные данные пользователя.

Клиенту, который авторизовался в информационной системе кондитерской организации, должна быть доступна страница, отображающая совершённые им заказы.

Администратору после авторизации в системе должна отображаться страница, которая содержит заказы, поступившие в кондитерскую от клиентов. Администратор системы должен иметь возможность сообщать статус готовности заказа клиенту. У клиента на странице совершённых заказов, соответственно, должен отображаться статус, указанный администратором. Так он сможет отслеживать готовность своего заказа.

У администратора должна быть и возможность просмотреть обращения от клиентов по поводу индивидуальных заказов на отдельной вкладке. Также администратору следует добавить возможность создавать самостоятельно заказы в системе. Это будет полезно для отображения в системе индивидуальных заказов и заказов, сформированных по личному обращению клиента, а именно при личной встрече или по звонку на контактный номер телефона.

Для отображения функциональных возможностей приложения была спроектирована диаграмма вариантов использования, представленная на рисунке 1.

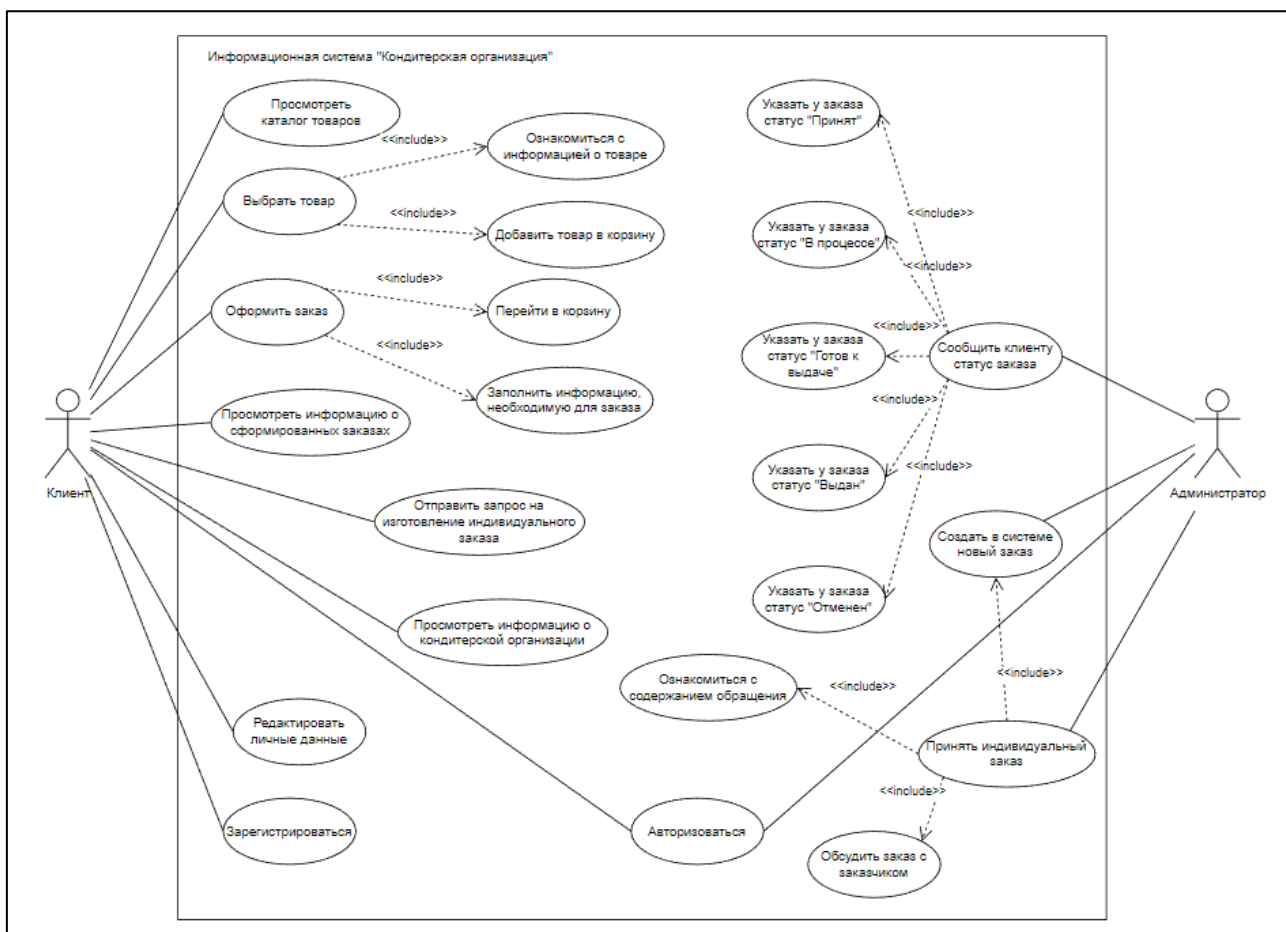


Рисунок 1 – Диаграмма вариантов использования

Диаграмма вариантов использования (use case diagram) – диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне [1].

В предметной области «Кондитерская организация» были выделены два действующих лица – клиент и администратор. Они представлены на диаграмме актёрами. Клиенту доступны следующие действия:

- просмотреть каталог товаров;
- выбрать товар;
- оформить заказ;
- просмотреть информацию о сформированных заказах;
- зарегистрироваться;
- отправить запрос на изготовление индивидуального заказа;

- просмотреть информацию о кондитерской организации;
- редактировать личные данные.

Эти действия являются вариантами использования (ВИ). Они связаны с актером отношениями ассоциации. Вариант использования «Выбрать товар» связан с «Ознакомиться с информацией о товаре» и «Добавить товар в корзину» отношением «включения». ВИ «Оформить заказ» включает в себя варианты использования «Перейти в корзину» и «Заполнить информацию, необходимую для заказа».

Администратор может совершать в системе следующие действия:

- сообщить клиенту статус заказа;
- создать в системе новый заказ;
- принять индивидуальный заказ.

Администратор может установить у заказа пять статусов: «Принят», «В процессе», «Готов к выдаче», «Выдан», «Отменён». Также администратор может выполнять действие «Принять индивидуальный заказ», что включает в себя ознакомление с содержанием обращения, обсуждение заказа с заказчиком и создание в системе нового заказа.

Также клиент и администратор имеют общий вариант использования «Авторизоваться». Это означает, что обоим ролям доступно данное действие.

2 Планирование и расчет бюджета разрабатываемой системы

Этап планирования и расчета бюджета необходим при реализации любой информационной системы. Он позволяет спланировать не только общие денежные траты, необходимые для создания какого-либо программного продукта, но и все основные задачи, подзадачи и ресурсы, требуемые для достижения определенного результата.

Правильно спланировать разработку продукта и необходимые средства на каждый этап позволит диаграмма Ганта.

Диаграмма Ганта – это популярный тип столбчатых диаграмм (гистограмм), который используется для иллюстрации плана, графика работ по какому-либо проекту. Является одним из методов планирования проектов. Используется в приложениях по управлению проектами [2].

Для построения диаграммы Ганта необходимо определить основные задачи, поставленные для реализации проекта, их подзадачи и сроки их выполнения. Перечень задач проекта и их сроков приведен в таблице 6.

Таблица 6 – Задачи проекта и их сроки выполнения

Наименование задачи	Длительность задачи в днях	Дата начала	Дата окончания
Начало проекта	0	12.04.2023, Ср.	12.04.2023, Ср.
Анализ предметной области и требований к системе	4	12.04.2023, Ср.	17.04.2023, Пн
Анализ предметной области	1	12.04.2023, Ср.	12.04.2023, Ср.
Анализ аналогичных систем	2	13.04.2023, Чт.	14.04.2023, Пт.
Постановка задачи и анализ функциональных требований	1	17.04.2023, Пн.	17.04.2023, Пн.
Проектирование программных средств	7	18.04.2023, Вт.	26.04.2023, Ср.
Выбор технологий проектирования	1	18.04.2023, Вт.	18.04.2023, Вт.
Проектирование базы данных	1	19.04.2023, Ср.	19.04.2023, Ср.
Проектирование серверной части	3	20.04.2023, Чт.	24.04.2023, Пн.
Проектирование клиентской части	2	25.04.2023, Вт.	26.04.2023, Ср
Реализация программных средств	15	27.04.2023, Чт.	17.05.2023, Ср.
Построение архитектуры системы	2	27.04.2023, Чт.	28.04.2023, Пт.
Разработка базы данных	1	01.05.2023, Пн.	01.05.2023, Пн.
Разработка серверной части	8	02.05.2023, Вт.	11.05.2023, Чт.
Разработка клиентской части	4	12.05.2023, Пт.	17.05.2023, Ср.
Тестирование программных средств	4	18.05.2023, Чт.	23.05.2023, Вт.
Тестирование методом черного ящика	1	18.05.2023, Чт.	18.05.2023, Чт.
Тестирование метрикой Холстеда	1	19.05.2023, Пт.	19.05.2023, Пт.

Продолжение таблицы 6

Наименование задачи	Длительность задачи в днях	Дата начала	Дата окончания
Автоматизированное тестирование	2	22.05.2023, Пн.	23.05.2023, Вт.
Оформление документации	7	24.05.2023, Ср.	01.06.2023, Чт.
Написание документации	6	24.05.2023, Ср.	31.05.2023, Ср.
Оформление и печать документации	1	01.06.2023, Чт.	01.06.2023, Чт.
Конец проекта	0	01.06.2023, Чт.	01.06.2023, Чт.

На основе этих данных была построена диаграмма Ганта. Она представлена на рисунке 2.

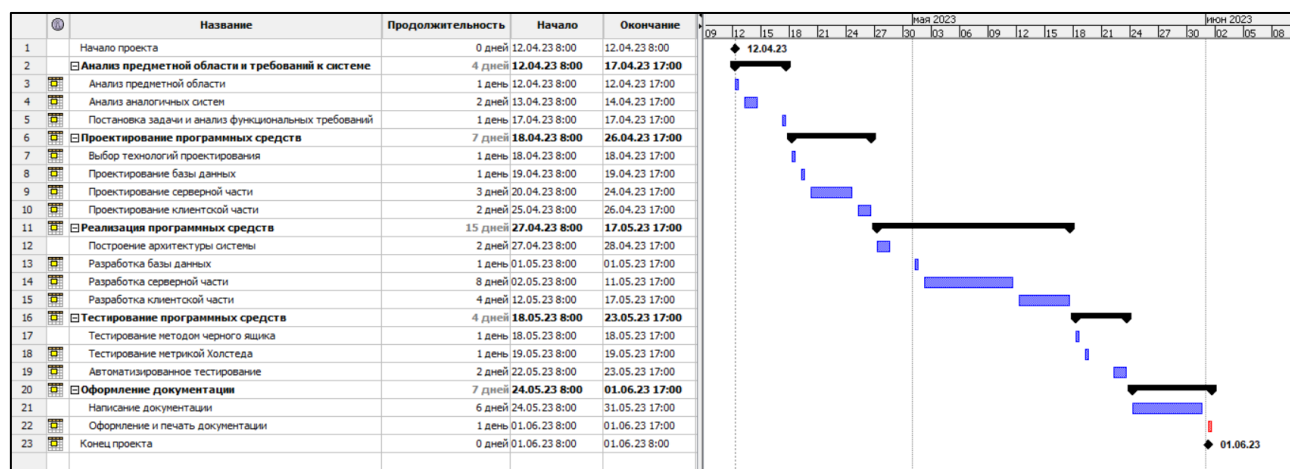


Рисунок 2 – Диаграмма Ганта

Для определения и подсчета затрат на разработку программного продукта были определены ресурсы проекта. Данные ресурсы представлены в таблице 7.

Таблица 7 – Ресурсы проекта

Наименование ресурса	Тип	Стандартная ставка	Начисление
Аналитик	Трудовой	200 руб./час	Пропорциональное
Проектировщик систем	Трудовой	200 руб./час	Пропорциональное
Разработчик back-end части программного обеспечения (ПО)	Трудовой	850 руб./час	Пропорциональное
Разработчик front-end части ПО	Трудовой	300 руб./час	Пропорциональное
Тестировщик ПО	Трудовой	150 руб./час	Пропорциональное
Компьютер	Материальный	50000 руб.	В начале
Принтер	Материальный	10000 руб.	В начале
Бумага для печати	Материальный	.300 руб.	В начале

В результате анализа задач и необходимых для их выполнения ресурсов были произведены расчеты касательно стоимости всех составляющих всего проекта. Затраты на проект приведены в таблице 8.

Таблица 8 – Задачи проекта и затраты на их выполнение

Наименование задачи	Длительность задачи в днях	Название ресурса	Затраты
Начало проекта	0	Компьютер, Принтер, Бумага для печати	60 300 руб.
Анализ предметной области и требований к системе	4		6 400 руб.
Анализ предметной области	1	Аналитик	1 600 руб.
Анализ аналогичных систем	2	Аналитик	3 200 руб.
Постановка задачи и анализ функциональных требований	1	Аналитик	1 600 руб.
Проектирование программных средств	7		11 200 руб.
Выбор технологий проектирования	1	Проектировщик систем	1 600 руб.
Проектирование базы данных	1	Проектировщик систем	1 600 руб.
Проектирование серверной части	3	Проектировщик систем	4 800 руб.
Проектирование клиентской части	2	Проектировщик систем	3 200 руб.
Реализация программных средств	15		84 400 руб.
Построение архитектуры системы	2	Разработчик back-end части ПО	13 600 руб.
Разработка базы данных	1	Разработчик back-end части ПО	6 800 руб.
Разработка серверной части	8	Разработчик back-end части ПО	54 400 руб.
Разработка клиентской части	4	Разработчик front-end части	9 600 руб.
Тестирование программных средств	4	.	4 800 руб.
Тестирование методом черного ящика	1	Тестировщик ПО	1 200 руб.
Тестирование метрикой Холстеда	1	Тестировщик ПО	1 200 руб.
Автоматизированное тестирование	2	Тестировщик ПО	2 400 руб.
Оформление документации	7		11 200 руб.
Написание документации	6	Аналитик	9 600 руб.
Оформление и печать документации	1	Аналитик	1 600 руб.
Конец проекта	0		

Таким образом, в результате определения всех задач проекта и подсчетов затрат на них, было определено, что общие затраты на проект составят 178 300 рублей, а общая продолжительность реализации проекта составит 37 дней.

3 Проектирование информационной системы

3.1 Инструментальные средства и методы проектирования

Для проектирования системы были выбраны такие средства, как модель «сущность-связь» и унифицированный язык моделирования (Unified Modeling Language, UML). Данные технологии позволят описать схему проектируемой базы данных и работу разрабатываемой системы кондитерской организации. Далее будут рассмотрены эти инструментальные средства.

3.1.1 Модель «сущность-связь»

Метод моделирования «сущность-связь» дает абстрактную модель предметной области, используя следующие основные понятия: сущности, взаимосвязи между сущностями и атрибуты для представления свойств сущностей и взаимосвязей.

Модель «сущность-связь» (Entity-Relationship model, ER-модель) – представляется с помощью ER-диаграмм, которые являются графической нотацией для абстрагирования данных в виде сущностей, взаимосвязей и атрибутов [3].

ER-диаграмма позволяет представить все элементы модели согласно простым, интуитивно понятным, но строго определенным правилам. Диаграмма может принимать форму концептуальной модели данных, логической модели данных или физической модели данных.

Использование ER-моделирования в рамках данной работы позволит представить модель базы данных, которая будет спроектирована для предметной области кондитерской организации.

3.1.2 Язык моделирования UML

Унифицированный язык моделирования (Unified Modeling Language, UML) представляет собой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем [4].

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций, получивших название диаграмм.

Диаграмма – наглядное представление некоей совокупности элементов модели системы в виде графа. В своём графическом виде различные виды диаграмм UML применяются для визуализации разных аспектов устройства или поведения моделируемой системы [5].

Язык моделирования UML был выбран для проектирования информационной системы в качестве удобного средства для описания структуры и поведения системы.

3.2 Проектирование базы данных системы

В ходе выполнения задания был выполнен этап проектирования базы данных разрабатываемой информационной системы формирования и сопровождения заказов кондитерской организации. В результате анализа функциональных требований к системе было выявлено, что она должна содержать данные о десертах, пользователях, корзине, заказах и обращениях.

Для проектирования базы данных было использовано инструментальное средство Visual Paradigm 16.3. С его помощью можно выполнять моделирование схем вида «сущность-связь».

Была смоделирована ER-диаграмма в форме логической модели данных. Логическая модель подразумевает представление данных в виде сущностей, связей между ними и атрибутов. Также на диаграмме, описывающей логическую модель данных, определяются первичные и внешние ключи сущностей. Смоделированная в программном средстве Visual Paradigm 16.3 диаграмма «сущность связь» представлена на рисунке 3.

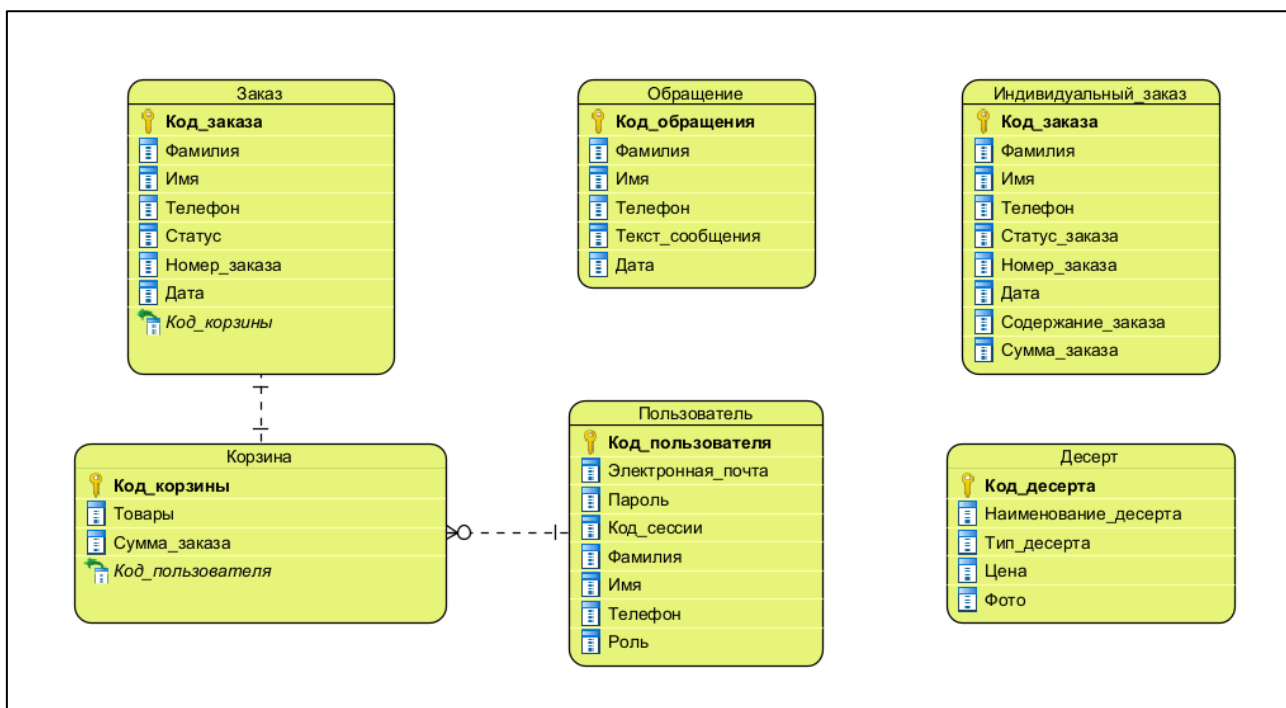


Рисунок 3 – Логическая модель базы данных

Описание сущностей приведено в таблице 9.

Таблица 9 – Описание сущностей логической модели

Название сущности	Описание сущности	Атрибут	Роль атрибута	Описание атрибута
Заказ	Хранение информации о заказах, сформированных пользователями	Код_заказа	РК (первичный ключ)	Уникальный идентификатор заказа
		Фамилия		Фамилия заказчика
		Имя		Имя заказчика
		Телефон		Номер телефона заказчика
		Статус		Статус заказа
		Номер_заказа	АК (альтернативный ключ)	Номер заказа
		Дата		Дата осуществления заказа
		Код_корзины	FK (внешний ключ)	Номер корзины покупателя

Продолжение таблицы 9

Название сущности	Описание сущности	Атрибут	Роль атрибута	Описание атрибута
Корзина	Хранение информации о корзинах, сформированных пользователями	Код_корзины	РК	Уникальный идентификатор корзины покупателя
		Товары		Перечень товаров, добавленных в корзину
		Сумма_заказа		Сумма всех товаров в корзине
		Код_пользователя	FK	Номер покупателя
Пользователь	Хранение информации о пользователях системы	Код_пользователя	РК	Уникальный идентификатор пользователя
		Электронная_почта	AK	Электронная почта пользователя
		Пароль		Пароль пользователя
		Код_сессии	AK	Номер сессии пользователя
		Фамилия		Фамилия пользователя
		Имя		Имя пользователя
		Телефон	AK	Номер телефона пользователя
		Роль		Роль пользователя
Обращение	Хранение информации об обращениях пользователей по поводу осуществления индивидуальных заказах	Код_обращения	РК	Уникальный идентификатор обращения пользователя
		Фамилия		Фамилия пользователя
		Имя		Имя пользователя
		Телефон		Номер телефона пользователя
		Текст_сообщения		Содержание сообщения пользователя
		Дата		Дата обращения

Окончание таблицы 9

Название сущности	Описание сущности	Атрибут	Роль атрибута	Описание атрибута
Индивидуальный заказ	Хранение информации о созданных администратором заказах на основе обращений пользователей	Код_заказа	РК	Уникальный идентификатор заказа
		Фамилия		Фамилия заказчика
		Имя		Имя заказчика
		Телефон		Номер телефона заказчика
		Статус заказа		Статус заказа
		Номер заказа	АК	Номер заказа
		Дата		Дата заказа
		Содержание_заказа		Содержание заказа
		Сумма_заказа		Сумма заказа
Десерт	Хранение информации о перечне десертов кондитерской	Код_десерта	РК	Уникальный идентификатор десерта
		Наименование_десерта		Наименование десерта
		Тип_десерта		Тип десерта
		Цена		Цена десерта
		Фото		Изображение десерта

Таким образом, логическая модель данных представлена шестью сущностями для хранения необходимых данных.

3.3 Проектирование серверной части системы

Важным этапом проектирования информационной системы является проектирование серверной части системы. Для начала необходимо определить её структуру.

Структура серверной части информационной системы формирования и сопровождения заказов кондитерской организации представлена совокупностью трёх слоёв – слоя «Entity», слоя «Services» и слоя «Data Access Object» (DAO).

Слой «Entity» (сущность) ответственен за представление данных, с которыми работает система. Он включает в себя структуры, которые описывают сущности, определенные при проектировании базы данных.

Цель слоя «Entity» состоит в том, чтобы предоставить модель данных, которая отражает сущности предметной области. Он фокусируется на описании структуры данных и их связей без привязки к способу хранения или обработки данных. Этот слой не содержит бизнес-логику, а лишь служит абстракцией, которая описывает модель данных.

Слой «Services» (сервисы) предназначен для реализации бизнес-логики и обработки различных операций над данными. Он служит посредником между слоем «Entity» и слоем «DAO». Этот слой содержит функции для обработки запросов клиента. Он использует объекты и методы из слоя «Entity», чтобы работать с данными.

Data Access Object (DAO) – это шаблон проектирования в объектно-ориентированном программировании, который предоставляет унифицированный интерфейс для доступа к данным в базе данных. Этот слой скрывает детали работы с базой данных от остальной части приложения, что упрощает поддержку и развитие приложения.

Слой «DAO» служит для управления доступа к данным и взаимодействия с базой данных. Он предоставляет методы для создания, чтения, обновления и удаления данных. Слой позволяет выполнять типичные операции с данными, такие как добавление новой записи, получение данных по идентификатору, обновление существующей записи и удаление записей.

В ходе проектирования серверной части была смоделирована диаграмма классов, которая представлена на рисунке 4.

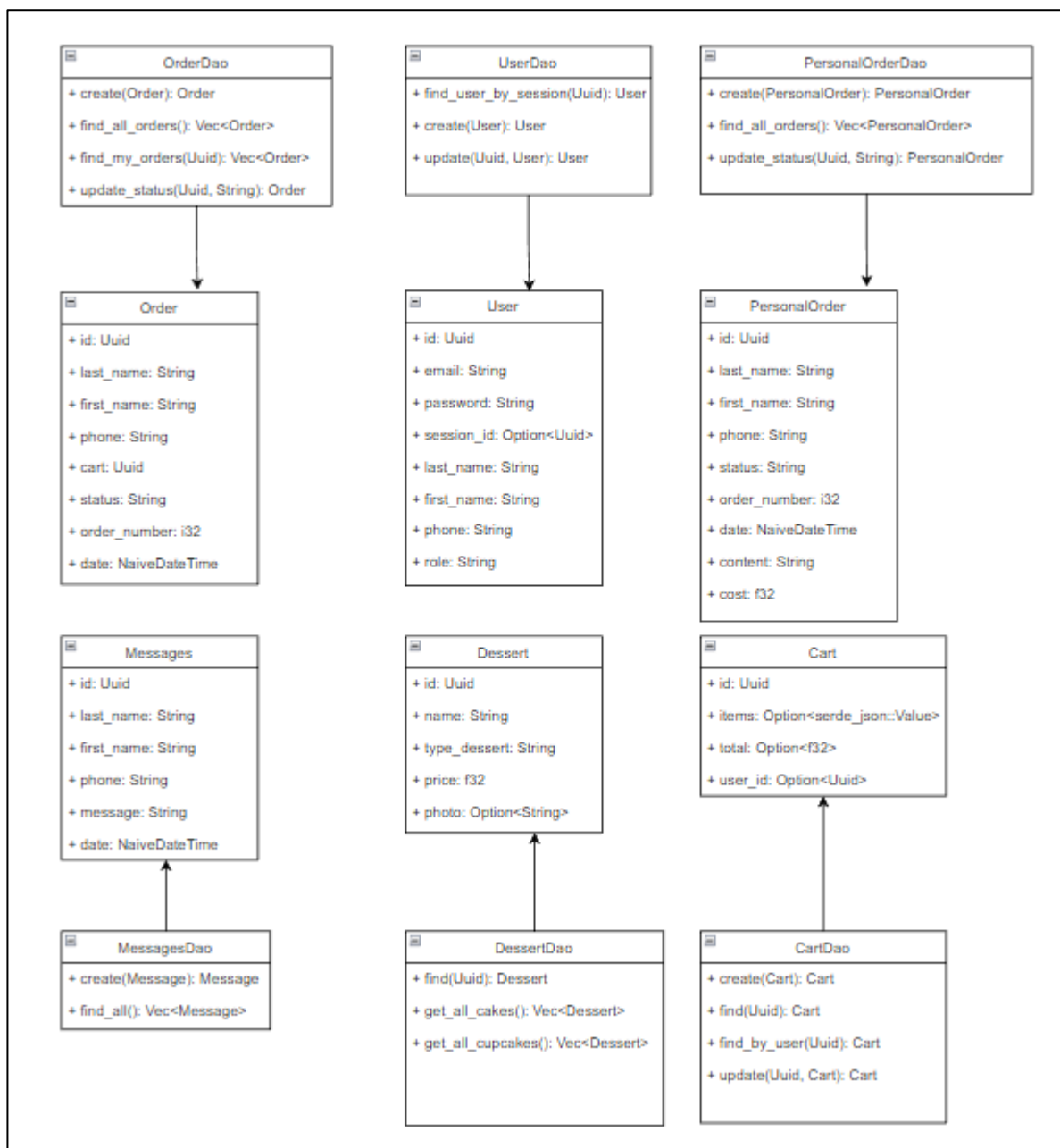


Рисунок 4 – Диаграмма классов

На диаграмме представлены классы слоя «Entity» и классы слоя «DAO».

Классы слоя «Entity» содержат в себе поля, соответствующие атрибутам таблиц базы данных, и предназначены для определения структуры сущностей спроектированной базы данных.

Классы слоя «DAO» содержат в себе методы и предназначены для взаимодействия с базой данных. Назначение методов классов приведено в таблице 10.

Таблица 10 – Описание методов классов слоя DAO

Название класса	Название метода	Назначение метода
OrderDao	create(Order): Order	Предназначен для создания нового заказа
	find_all_orders(): Vec<Order>	Предназначен для поиска всех заказов
	find_my_orders(Uuid): Vec<Order>	Предназначен для поиска заказов пользователя
	update_status(Uuid, String): Order	Предназначен для обновления статуса заказа
UserDao	find_user_by_session(Uuid): User	Предназначен для поиска пользователей по идентификатору сессии
	create(User): User	Предназначен для создания нового пользователя
	update(Uuid, User): User	Предназначен для обновления данных пользователя
PersonalOrderDao	create(PersonalOrder): PersonalOrder	Предназначен для создания нового индивидуального заказа
	find_all_orders(): Vec<PersonalOrder>	Предназначен для поиска всех заказов
	update_status(Uuid, String): PersonalOrder	Предназначен для обновления статуса заказа
MessagesDao	create(Message): Message	Предназначен для создания нового обращения пользователя
	find_all(): Vec<Message>	Предназначен для поиска всех обращений
DessertDao	find(Uuid): Dessert	Предназначен для поиска десерта по идентификатору
	get_all_cakes(): Vec<Dessert>	Предназначен для получения всех тортов
	get_all_cupcakes(): Vec<Dessert>	Предназначен для получения всех капкейков
CartDao	create(Cart): Cart	Предназначен для создания корзины
	find(Uuid): Cart	Предназначен для поиска корзины по идентификатору
	find_by_user(Uuid): Cart	Предназначен для поиска корзины по идентификатору пользователя
	update(Uuid, Cart): Cart	Предназначен для обновления корзины

Объекты диаграммы классов имеют отношения ассоциации, которые указывают на связь между классами слоя «Entity» и слоя «DAO».

3.4 Проектирование клиентской части

На этапе проектирования клиентской части была определена структура клиентской части информационной системы формирования и сопровождения

заказов кондитерской организации. Для её представления целесообразно использовать диаграмму компонентов.

Диаграмма компонентов описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами [6].

Диаграмма компонентов клиентской части представлена на рисунке 5.

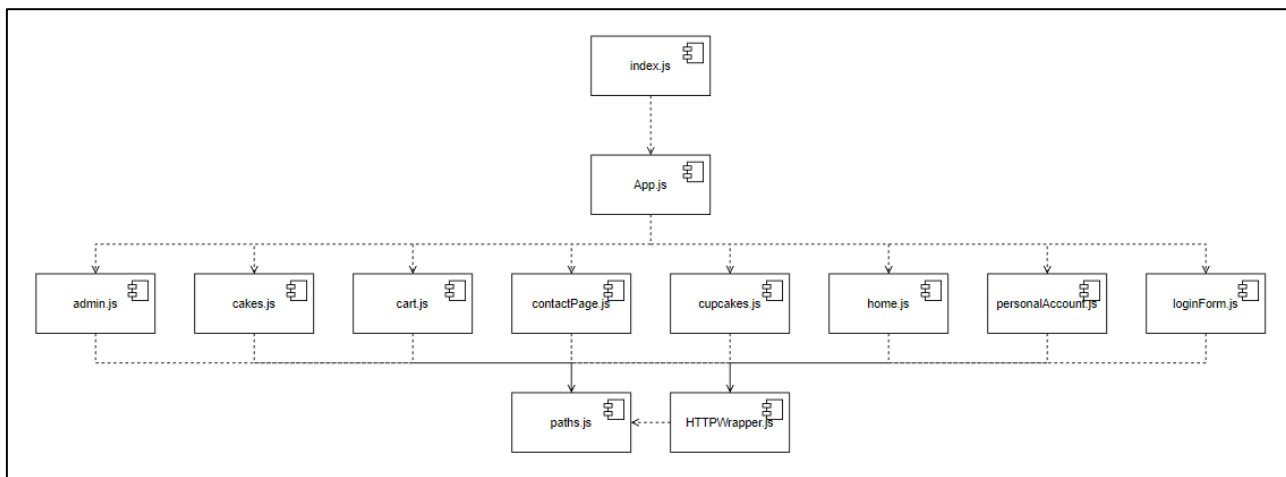


Рисунок 5 – Диаграмма компонентов клиентской части

Представленная диаграмма компонентов содержит следующие компоненты:

- index.js;
- app.js;
- admin.js;
- cakes.js;
- cart.js;
- contactPage.js;
- cupcakes.js;
- home.js;
- personalAccount.js;
- loginForm.js;
- paths.js;

– HTTPWrapper.js.

Компонент «index.js» является точкой входа в программу.

Компонент «App.js» представлен файлом, который определяет маршрутизацию в приложении.

Компонент «paths.js» определяет URL-адреса для запросов на сервер. Используется большинством компонентов.

Компонент «HTTPWrapper.js» определяет класс с набором методов, необходимых для отправки и обработки HTTP-запросов. Этот компонент используется всеми компонентами, которые обрабатывают какие-либо данные.

Компонент «admin.js» необходим для отображения кабинета администратора пользователя, обработки пользовательского ввода и отправки HTTP-запросов на сервер.

Компоненты «cakes.js» и «cupcakes.js» предназначен для отображения пользователю доступных тортов и капкейков соответственно. Также они обрабатывают команды пользователя по добавлению десерта в корзину.

Компонент «cart.js» предназначен для формирования корзины пользователя, а также формирования заказов.

Компонент «contactPage.js» предназначен для отображения контактной информации об организации пользователю и обработки запросов на обратную связь по поводу индивидуальных заказов.

Компонент «home.js» предназначен для отображения главной страницы системы формирования и сопровождения заказов.

Компонент «personalAccount.js» предназначен для предоставления пользователю страницы личного кабинета с возможностью редактирования данных в нём.

Компонент «loginForm.js» предназначен для регистрации и авторизации пользователей.

3.5 Проектирование графического интерфейса

Графический интерфейс пользователя – система средств для взаимодействия пользователя с электронными устройствами, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т. п.) [7].

Проектирование интерфейса – это процесс создания эффективного и удобного для использования графического интерфейса для программного продукта. В этом процессе необходимо учитывать требования пользователя и задачи, которые система должна выполнять, а также общий дизайн продукта и его стиль.

Основными задачами проектирования интерфейса являются:

- создание эффективной структуры интерфейса: необходимо создать удобную структуру интерфейса, которая позволит пользователям легко и быстро находить нужную информацию и выполнять нужные действия;
- выбор правильных элементов интерфейса: для каждой задачи нужно выбрать правильный элемент интерфейса, который позволит пользователю выполнить задачу наиболее эффективным способом;
- создание согласованного дизайна: дизайн интерфейса должен соответствовать общему дизайну продукта и быть согласованным с другими элементами интерфейса;
- учет требований пользователей: при проектировании интерфейса необходимо учитывать требования и ожидания пользователей и создавать интерфейс, который будет удобен и понятен для них.

Кроме того, при проектировании интерфейса необходимо учитывать такие аспекты, как доступность, видимость, простота, привлекательность, наличие обратной связи.

Проектируемый графический интерфейс информационной системы формирования и сопровождения заказов кондитерской организации должен

содержать главную страницу, а также страницы «Торты», «Капкейки», «Корзина», «Личный кабинет», «О нас» и страницу авторизации. Данные страницы должны быть доступны неавторизованному пользователю и авторизованному покупателю. Администратору системы должна быть доступна страница, отображающая панель администратора, которая будет содержать вкладки «Заказы» и «Сообщения».

Рассмотрим интерфейс, который будет реализован для покупателя, авторизованного в системе.

В шапке страницы будут расположены кнопки «Главная», «Личный кабинет», «Войти» и «Корзина».

Главная страница должна предоставлять пользователю возможность выбрать категорию десертов – торты или капкейки. Соответственно, при нажатии на ту или иную иконку будет открыта соответствующая страница – «Торты» или «Капкейки», которые будут содержать перечень десертов и информацию о них. На этих же страницах можно добавить понравившиеся десерты в корзину.

Страница «Корзина» будет отображать десерты, добавленные пользователем в корзину. Здесь же будет формироваться заказ.

Сформированный заказ будет отображаться на странице «Личный кабинет» на вкладке «Заказы». На ней будут отображаться сведения о заказах, а также можно будет отследить статус заказа. На этой же странице будет расположена вкладка «Профиль», которая будет содержать личную информацию клиента.

На странице «О нас» будет располагаться основная информация о кондитерской организации, а также будет содержаться форма обратной связи для осуществления индивидуального заказа, который будет направлен администратору.

Интерфейс, спроектированный для роли администратора системы, будет представлен страницей панели администратора. На данной странице будут размещены две вкладки – «Заказы» и «Обращения». На вкладке «Заказы»

должны отображаться заказы, принятые от клиента системы. Их можно будет обрабатывать, устанавливая каждому заказу соответствующий статус о его готовности, что будет также отображено у клиента. На вкладку «Обращения» будут попадать обращения клиентов по поводу индивидуальных заказов. На основе сообщения покупателя администратор сможет на этой же вкладке создать новый заказ в системе, в котором укажет необходимую информацию о клиенте и состав заказа.

Спроектированная модель интерфейса главной страницы системы кондитерской организации представлена на рисунке 6.

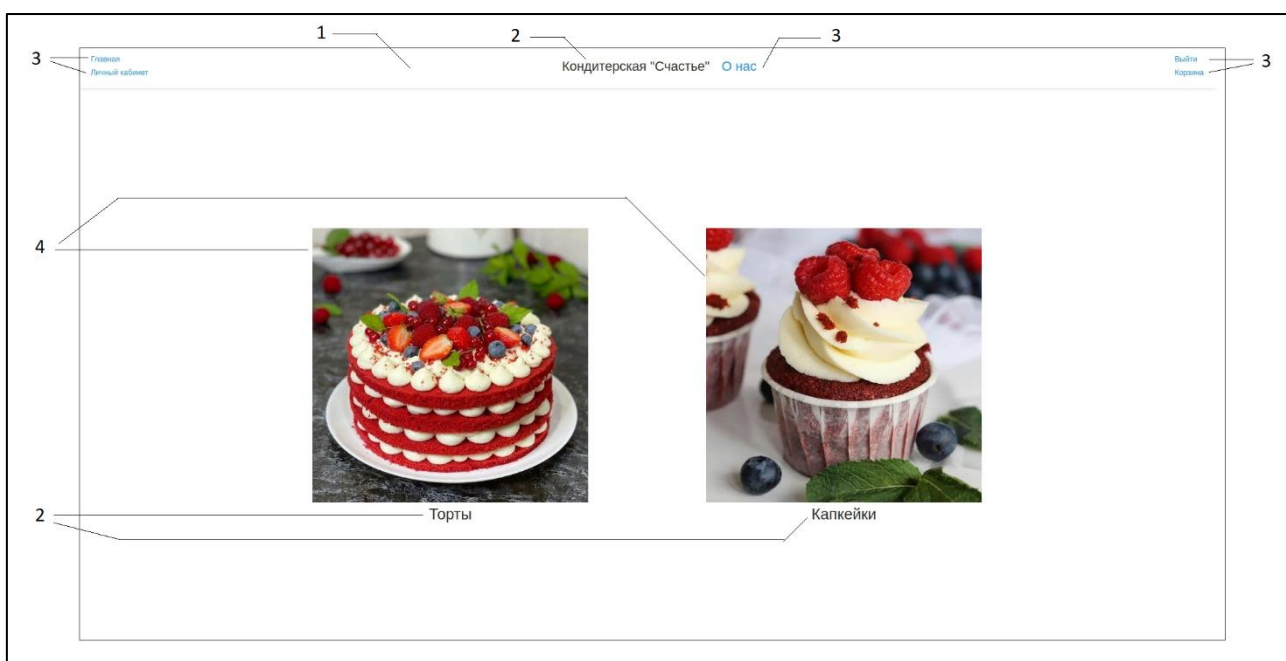


Рисунок 6 – Модель интерфейса главной страницы

Модель содержит следующие элементы:

- 1 – заголовок страницы;
- 2 – текстовые элементы;
- 3 – кнопки;
- 4 – изображения.

Скриншоты реализованного интерфейса пользователя будут представлены в данной работе в п. 4.4.

4 Реализация информационной системы

4.1 Архитектура разрабатываемой информационной системы

Разрабатываемая информационная система имеет трехуровневую архитектуру «Клиент – Сервер – База данных».

Трёхуровневая архитектура – архитектурная модель программного комплекса, предполагающая наличие в нём трёх типов компонентов (уровней, звеньев): клиентских приложений (с которыми работают пользователи), серверов приложений (с которыми работают клиентские приложения) и серверов баз данных (с которыми работают серверы приложений) [8].

Слой клиента предоставляет интерфейс пользователю. Этот слой не имеет непосредственной связи с базой данных (БД). Для этого существует слой сервера приложения, где реализована бизнес-логика, отвечающая в том числе за формирования запросов к базе данных.

На уровне сервера сосредоточена бизнес-логика. Сервер базы данных обеспечивает хранения данных и представлен, как правило, системами управления базами данных.

Важно уточнить взаимодействие клиента и сервера между собой. Это взаимодействие происходит посредством протокола HyperText Transfer Protocol (HTTP), который обеспечивает их взаимосвязь путем отправки клиентом HTTP-запроса. Сервер, в свою очередь, обрабатывает запрос и возвращает клиенту HTTP-ответ.

Что касается взаимодействия сервера с базой данных, то оно осуществляется с использованием SQL-запросов к базе данных. Сервер формирует SQL-запрос к базе данных, а база данных возвращает необходимые данные.

Так происходит взаимодействие всех трёх уровней архитектура системы.

Для представления структуры разрабатываемой информационной системы была спроектирована диаграмма коммуникации, приведенная на рисунке 7.

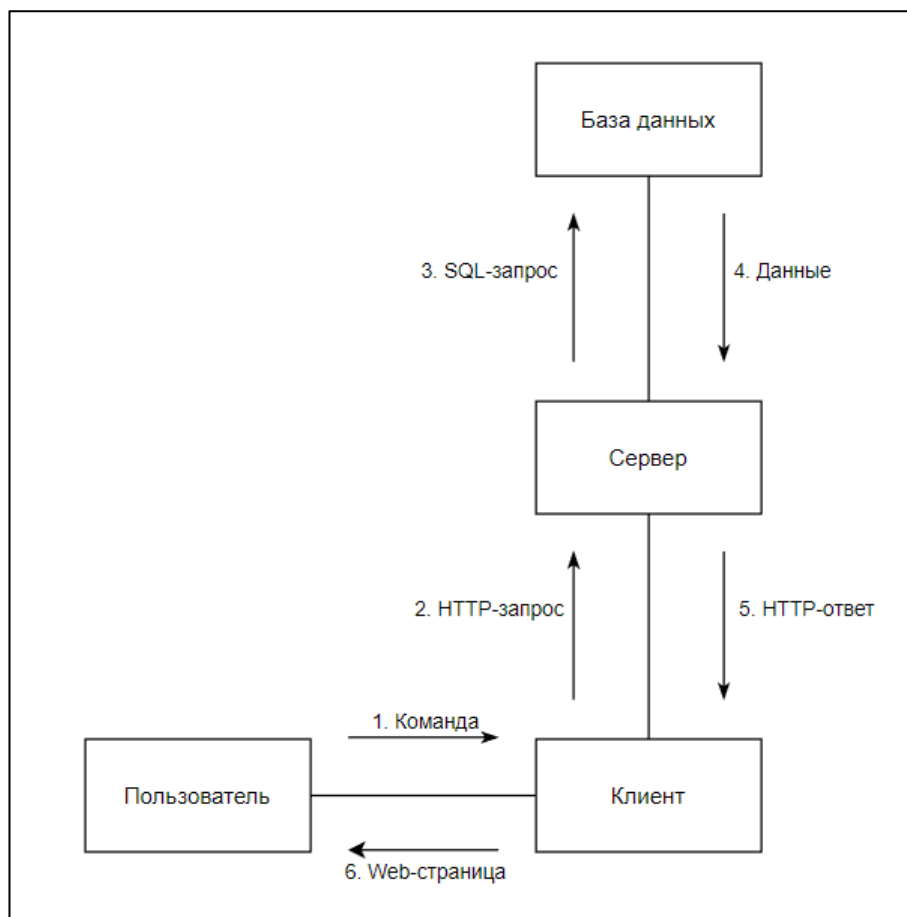


Рисунок 7 – Диаграмма коммуникации

На данном рисунке изображена диаграмма коммуникации относительно трёх уровней архитектуры и пользователя, взаимодействующего с ними. Сам пользователь взаимодействует непосредственно со слоем клиента путем отправки некоторой команды. Клиент распознаёт эту команду и отправляет на сервер HTTP-запрос, после чего сервер обращается к базе данных системы с помощью SQL-запроса. База данных отправляет в ответ необходимые данные, которые сервер возвращает клиенту в виде HTTP-ответа. В конце пользователь получает от клиента результат выполнения запрошенной команды.

4.2 Разработка хранилища данных

Для разработки хранилища данных информационной системы формирования и сопровождения заказов кондитерской организации была выбрана система управления базами данных (СУБД) PostgreSQL.

Эта СУБД является надежной, масштабируемой и производительной, обладает широкими возможностями для работы с данными различных типов и сложных структур. PostgreSQL поддерживает многопоточную обработку запросов и масштабирование горизонтально (через репликацию) и вертикально (через увеличение ресурсов сервера). Она также обладает широкими возможностями для обеспечения безопасности данных, включая авторизацию и аутентификацию пользователей, шифрование данных и другие меры защиты.

Таким образом, использование PostgreSQL в качестве СУБД для выполнения технического задания на проектирование и разработку позволяет обеспечить надежное и эффективное хранение и обработку данных.

При использовании СУБД PostgreSQL была построена физическая модель базы данных системы кондитерской организации, которая представлена на рисунке 8.

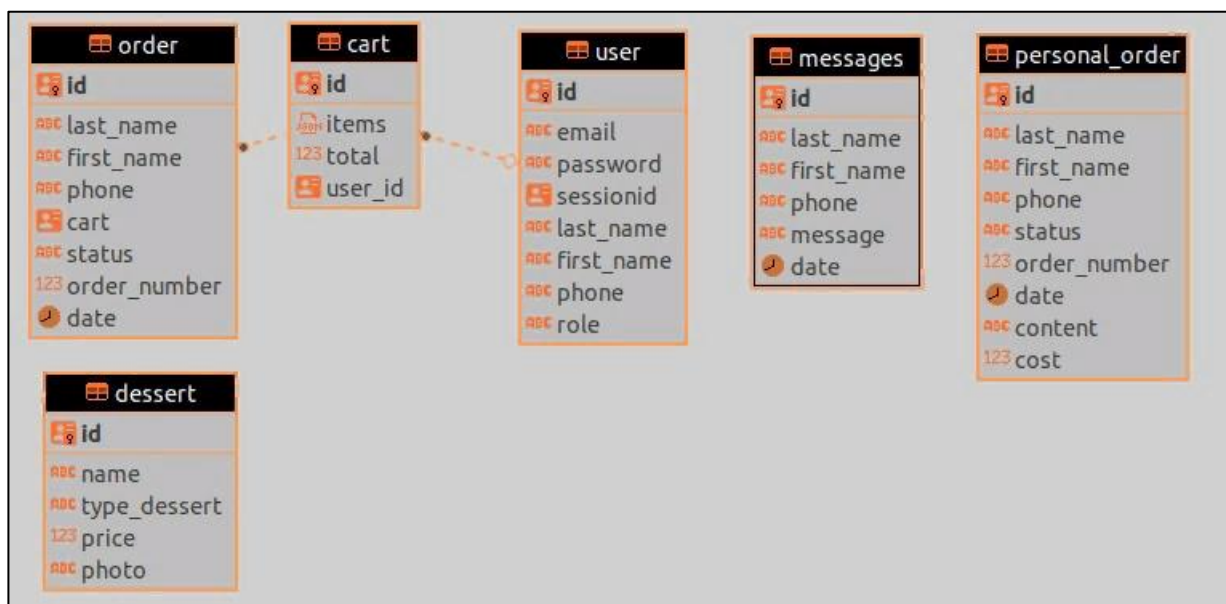


Рисунок 8 – Физическая модель базы данных

Реализованная схема базы данных соответствует логической модели данных, представленной в п. 2.2. База данных системы содержит шесть таблиц: «Order», «Dessert», «Cart», «User», «Messages», «Personal_order».

Описание таблиц физической модели представлено в таблицах 11–16.

Таблица 11 – Описание таблицы «Order»

Наименование	Назначение	Типа данных	Возможность значения NULL	Роль	Пример
id	Идентификатор заказа	Uuid	Нет	PK	b784a16a-0bf1-4865-b325-a19d814ae879
last_name	Фамилия заказчика	Строковый	Нет		Иванов
first_name	Имя заказчика	Строковый	Нет		Иван
phone	Номер телефона заказчика	Строковый	Нет		9005003838
cart	Идентификатор корзины	Uuid	Нет	FK	b784a16a-0bf1-4865-b325-a19d814ae879
status	Статус заказа	Строковый	Нет		В процессе
order_number	Номер заказа	Целочисленный	Нет	AK	0098
date	Дата заказа	Дата	Нет		2022-08-23 16:21:58

Таблица 12 – Описание таблицы «Dessert»

Наименование	Назначение	Типа данных	Возможность значения NULL	Роль	Пример
id	Идентификатор десерта	Uuid	Нет	PK	b784a16a-0bf1-4865-b325-a19d814ae879
name	Наименование десерта	Строковый	Нет		Торт Шоколадный
type_dessert	Тип десерта	Строковый	Нет		cake
price	Стоимость десерта	Вещественный	Нет		3000,00
photo	Путь к изображению десерта	Строковый	Да		/img/cake.png

Таблица 13 – Описание таблицы «Cart»

Наименование	Назначение	Типа данных	Возможность значения NULL	Роль	Пример
id	Идентификатор корзины	Uuid	Нет	PK	b784a16a-0bf1-4865-b325-a19d814ae879
items	Товары	jsonb	Да		[{"id": "b784a16a-0bf1-4865-b325-a19d814ae879", "count": 1}]
total	Сумма заказа	Вещественный	Да		3000,00
user_id	Идентификатор пользователя	Uuid	Нет	FK	b784a16a-0bf1-4865-b325-a19d814ae879

Таблица 14 – Описание таблицы «User»

Наименование	Назначение	Типа данных	Возможность значения NULL	Роль	Пример
id	Идентификатор заказа	Uuid	Нет	PK	b784a16a-0bf1-4865-b325-a19d814ae879
email	Электронная почта пользователя	Строковый	Нет	AK	i.ivanov@mail.ru
password	Пароль пользователя	Строковый	Нет		1-Asdfgh259
session_id	Идентификатор сессии	Uuid	Да	AK	b784a16a-0bf1-4865-b325-a19d814ae879
last_name	Фамилия пользователя	Строковый	Нет		Иванов
first_name	Имя пользователя	Строковый	Нет		Иван
phone	Номер телефона пользователя	Строковый	Нет	AK	9005003838
role	Роль пользователя	Строковый	Нет		Registered

Таблица 15 – Описание таблицы «Messages»

Наименование	Назначение	Типа данных	Возможность значения NULL	Роль	Пример
id	Идентификатор обращения	Uuid	Нет	PK	b784a16a-0bf1-4865-b325-a19d814ae879
last_name	Фамилия пользователя	Строковый	Нет		Иванов
first_name	Имя пользователя	Строковый	Нет		Иван
phone	Номер телефона пользователя	Строковый	Нет		9005003838
message	Текст обращения	Строковый	Нет		Добрый день!
date	Дата обращения	Дата	Нет		2022-08-23 16:21:58

Таблица 16 – Описание таблицы «PersonalOrder»

Наименование	Назначение	Типа данных	Возможность значения NULL	Роль	Пример
id	Идентификатор заказа	Uuid	Нет	PK	b784a16a-0bf1-4865-b325-a19d814ae879
last_name	Фамилия заказчика	Строковый	Нет		Иванов
first_name	Имя заказчика	Строковый	Нет		Иван
phone	Номер телефона заказчика	Строковый	Нет		9005003838
content	Состав заказа	Строковый	Нет		Торт Шоколадный
status	Статус заказа	Строковый	Нет		В процессе
order_number	Номер заказа	Целочисленный	Нет	AK	0098
date	Дата заказа	Дата	Нет		2022-08-23 16:21:58
cost	Стоимость заказа	Вещественный	Нет		3000,00

Таблица «Order» хранит информацию о заказах. Она содержит поля для хранения фамилии клиента, имени, номера телефона, корзины, статуса заказа, номера заказа и даты.

Таблица «Dessert» содержит информацию о десертах, представленных в системе – о тортах, капкейках. Она включает поля для хранения наименования и типа десерта, стоимости и изображения.

Таблица «Cart» содержит данные о корзине пользователя – о добавленных товарах, сумме заказа и идентификаторе пользователя.

Таблица «User» предназначена для хранения данных о пользователях системы – покупателях и администраторах. Она хранит данные об электронной почте пользователя, пароле, фамилии, имени, номере телефона, роли. Поле «role» (роль) может содержать два значения – «зарегистрированный» и «администратор».

Таблица «Messages» содержит информацию об обращениях, оставленных пользователями по поводу индивидуальных заказов. В системе предусмотрена возможность делать не только заказ товаров, представленных в ассортименте, но и заказ десертов в индивидуальном порядке. Пользователь оставляет свои данные (фамилию, имя, номер телефона и само сообщение) и отправляет запрос на создание индивидуального заказа. Администратор, соответственно, получает данное сообщение и может отреагировать на него – связаться с заказчиком и обсудить подробности заказа. При договорённости о деталях заказа администратор может создать новую запись в списке заказов, которая будет храниться в таблице «Personal_order».

Таблица «Personal_order» предназначена для хранения информации об индивидуальных заказах покупателей. Она содержит поля для хранения фамилии заказчика, имени, номера телефона, статуса заказа, номера заказа, даты, стоимости заказа и его содержания.

4.3 Реализация серверной части системы

Для реализации серверной части информационной системы был выбран язык программирования Rust с использованием фреймворка Actix Web и объектно-реляционной системы (ORM) управления базами данных Diesel.

Rust – это многопарадигмальный язык программирования, в котором особое внимание уделяется производительности, безопасности типов и параллелизму [9].

Actix Web – это мощный, прагматичный и чрезвычайно быстрый веб-фреймворк для Rust. Он обладает множеством возможностей, позволяющих создавать мощные и эффективные веб-приложения. К таким возможностям можно отнести следующие:

- маршрутизация запросов;
- обработка HTTP-запросов;
- асинхронность.

Маршрутизация запросов позволяет определить различные URL-шаблоны и связать их с обработчиками запросов, которые будут выполнять нужные действия.

Механизм Actix Web предоставляет механизмы для обработки таких типов HTTP-запросов, как GET, POST, PUT, DELETE и другие.

Возможность асинхронного выполнения запросов также является одним из удобных механизмов, предоставляемых фреймворком Actix Web.

Diesel – это ORM фреймворк для Rust, предназначенный для упрощения работы с базами данных. Он предоставляет безопасный программный интерфейс приложения (Application Programming Interface, API) для взаимодействия с БД и позволяет писать эффективный код на Rust при работе с данными в базах данных. Diesel также предоставляет мощные инструменты для создания запросов к БД, что упрощает написание сложных запросов и избавляет от необходимости писать SQL-запросы вручную. Он также поддерживает множество типов баз данных и имеет хорошую производительность.

В результате была разработана серверная часть информационной системы. Для описания её структуры можно представить диаграмму компонентов.

Смоделированная диаграмма компонентов представлена на рисунке 9.

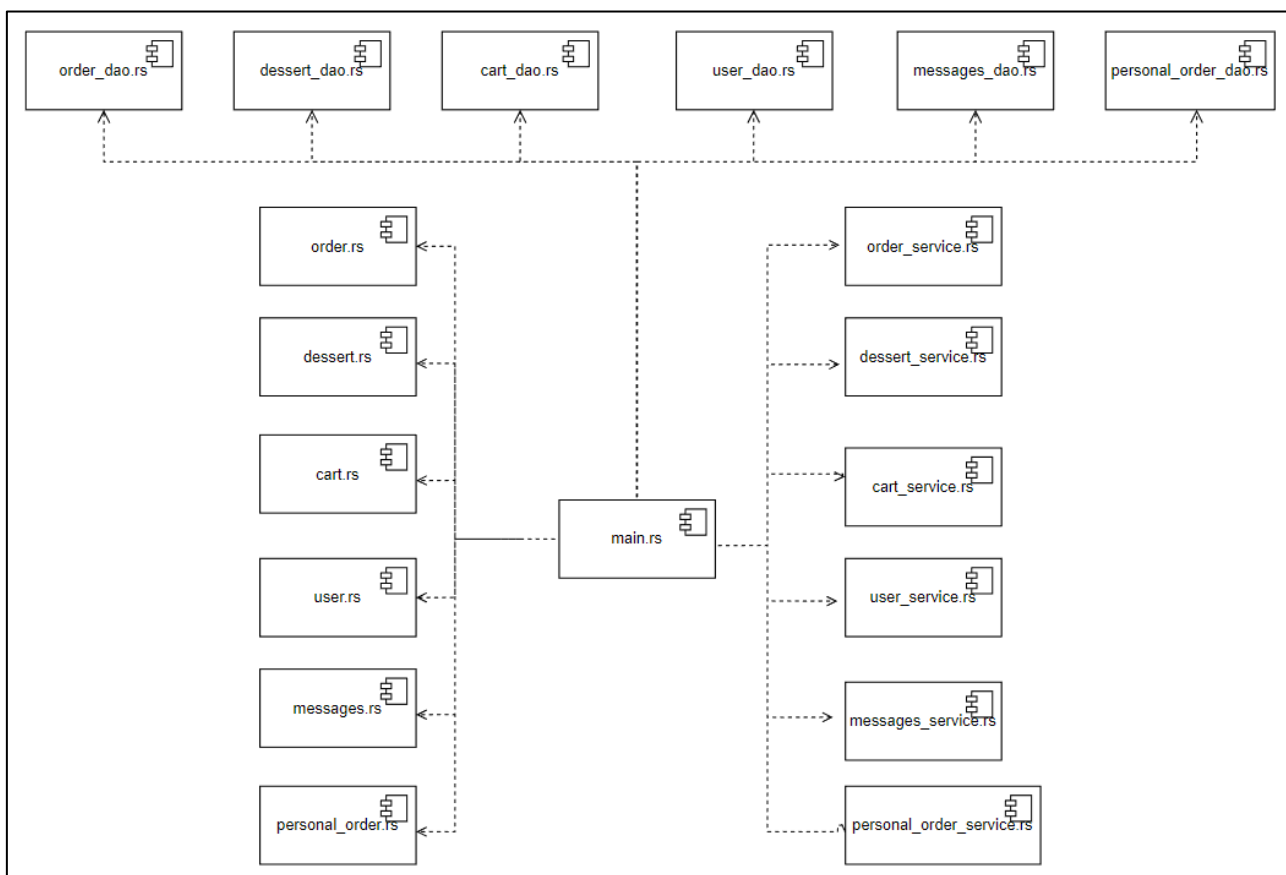


Рисунок 9 – Диаграмма компонентов серверной части

Данная диаграмма описывает архитектуру серверной части системы кондитерской организации. Главным компонентом серверной части является файл `main.rs`, который осуществляет запуск веб-сервера.

Файлы «`order.rs`», «`dessert.rs`», «`cart.rs`», «`user.rs`», «`messages.rs`» и «`personal_order.rs`» представляют модель данных, которая отражает сущности предметной области.

Файлы «`order_dao.rs`», «`dessert_dao.rs`», «`cart_dao.rs`», «`user_dao.rs`», «`messages_dao.rs`» и «`personal_order_dao.rs`» осуществляют взаимодействие с базой данных и позволяют получать, добавлять, изменять и удалять данные в ней.

Файлы «order_service.rs», «dessert_service.rs», «cart_service.rs», «user_service.rs», «messages_service.rs» и «personal_order_service.rs» предназначены для обработки запросов клиента.

Код серверной части представлен в приложении А.

4.4 Реализация клиентской части

Для разработки клиентской части информационной системы была использована JavaScript-библиотека React в сочетании с Ant Design.

React – JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. React может использоваться для разработки одностраничных и мобильных приложений. Его цель – предоставить высокую скорость разработки, простоту и масштабируемость [10].

В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с Ant Design.

Ant Design является популярной библиотекой компонентов пользовательского интерфейса для разработки веб-приложений. Библиотека предлагает широкий набор компонентов, таких как кнопки, формы, таблицы и другие, которые можно использовать для создания интерфейса.

В результате реализации клиентской части системы был разработан пользовательский интерфейс, соответствующий требованиям, описанным в данной работе в п. 2.5.

На рисунке 10 представлена главная страница интерфейса информационной системы. На ней представлены иконки, по нажатию на которые осуществляется переход на соответствующие страницы – на страницу с каталогом тортов или страницу с перечнем капкейков.

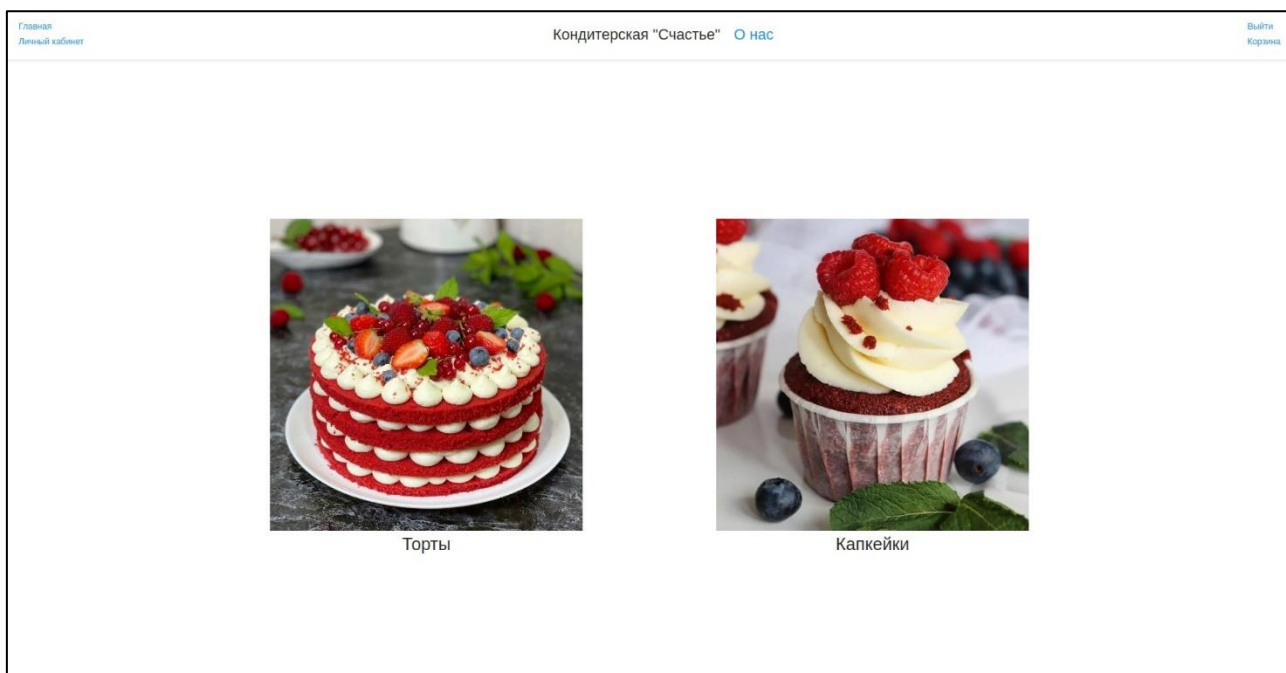


Рисунок 10 – Главная страница

При нажатии на иконку торта осуществится переход на страницу «Торты». На ней представлен список десертов, которые доступны к заказу в кондитерской. Напротив каждого торта есть кнопка «Добавить в корзину», чтобы приобрести кондитерское изделие (рисунок 11).

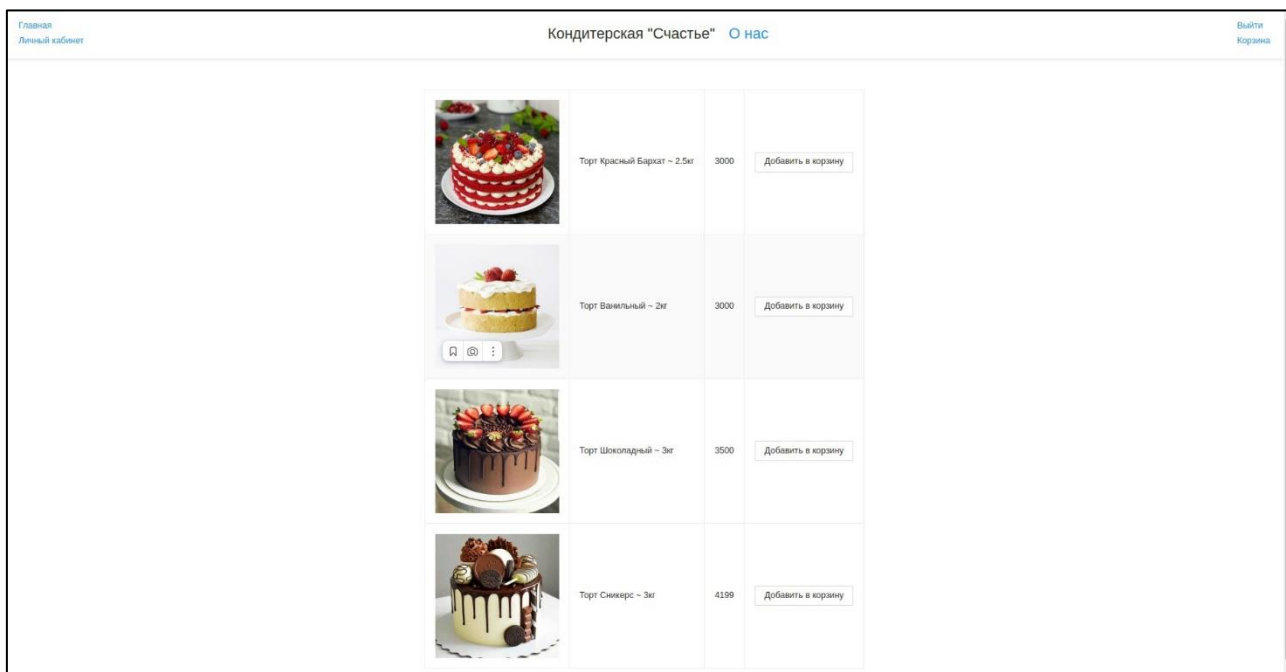


Рисунок 11 – Страница «Торты»

При нажатии на иконку капкейка осуществится переход на страницу «Капкейки». На ней представлен список доступных к заказу десертов. Напротив каждого десерта есть кнопка «Добавить в корзину», по нажатии на которую товар добавится в корзину, откуда можно будет сделать заказ (рисунок 12).

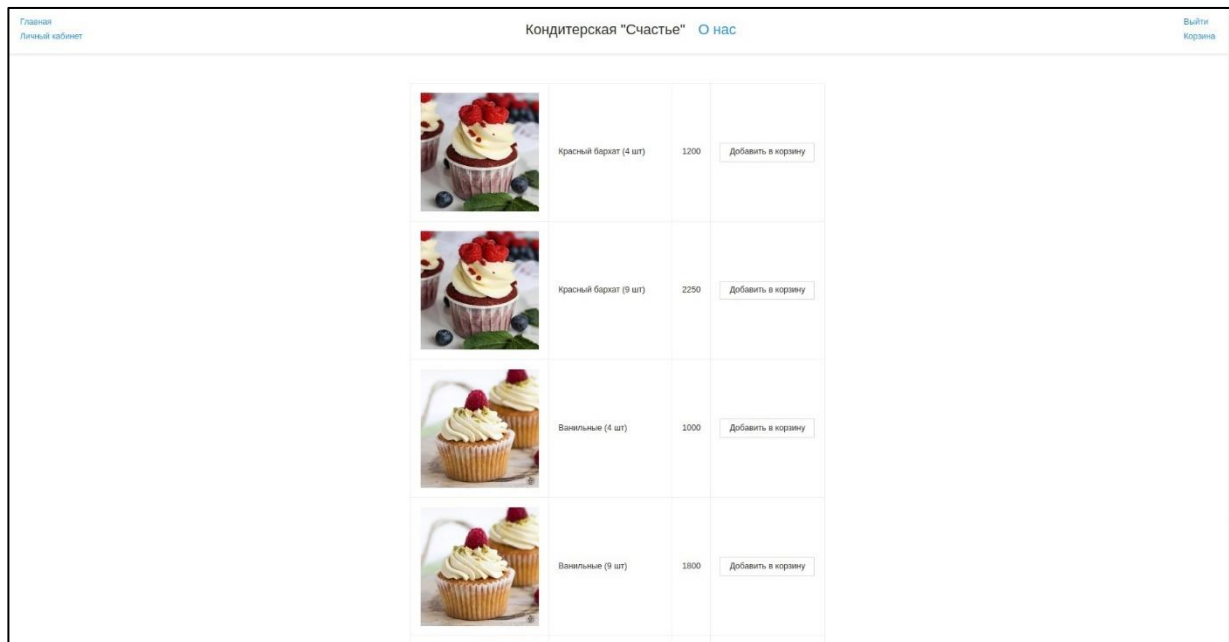


Рисунок 12 – Страница «Капкейки»

Добавленные товары в корзину отобразятся на соответствующей странице «Корзина». Здесь можно изменить их количество и оформить заказ (рисунок 13).

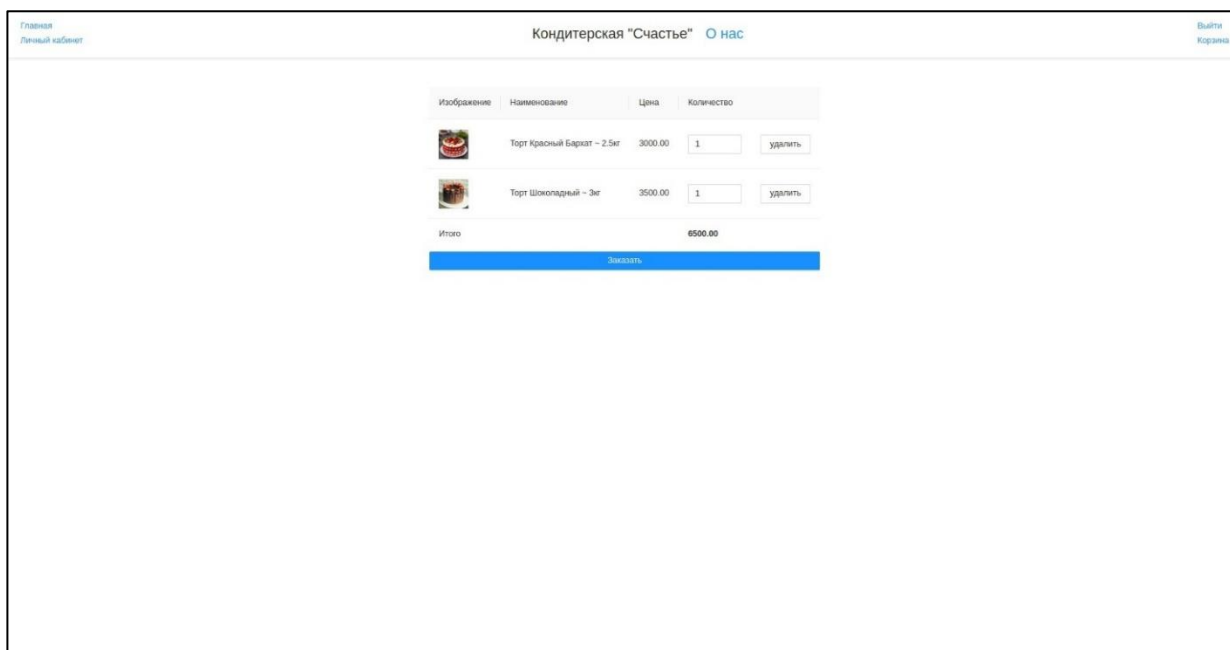


Рисунок 13 – Страница «Корзина»

Для оформления заказа необходимо нажать на кнопку «Заказать», после чего откроется модальное окно «Подтвердите или измените данные», где требуется подтвердить фамилию, имя и номер телефона, указанные при регистрации, или при необходимости изменить их, и нажать на кнопку «Подтвердить» (рисунок 14).

Рисунок 14 – Модельное окно заказа

После совершения заказа он отобразится на странице личного кабинета на вкладке «Заказы». Здесь предоставлена возможность отслеживать статус заказа. Обработкой заказа занимается администратор системы. Он же и устанавливает статусы, а именно «Принят», «В процессе», «Готов к выдаче», «Выдан», «Отменён». Скриншот вкладки «Заказы» приведен на рисунке 15.

Главная

Личный кабинет

Кондитерская "Счастье"

О нас

Выход

Корзина

Профиль

Заказы

Мои заказы

Номер заказа	Дата	Время	Сумма заказа	Состав заказа	Статус
0002	20.05.2023	15:30	6500	<div><div></div>Торт Красный Бархат (Количество: 1)</div> <div><div></div>Торт Шоколадный (Количество: 1)</div>	В процессе
0001	18.05.2023	12:15	5000	<div><div></div>Торт Ванильный (Количество: 1)</div> <div><div></div>Набор Праздничный (5 шт) (Количество: 1)</div>	Готов к выдаче

1

Рисунок 15 – Вкладка «Заказы» у покупателя

На этой же странице, на вкладке «Профиль», можно смотреть и редактировать личные данные, введенные при регистрации в системе (рисунок 16).

Рисунок 16 – Профиль пользователя

Также была разработана страница «О нас», которая отображает основную информацию о кондитерской организации, в том числе контактную. Также на этой же странице можно оставить обращение по поводу индивидуального заказа. Оставленное сообщение отобразится у администратора, и он сможет отреагировать на него. Скриншот страницы представлен на рисунке 17.

Рисунок 17 – Страница «О нас»

Была разработана страница авторизации, на которой пользователю предлагается ввести email и пароль для входа в систему. Если аккаунт ещё не создан, то пользователю будет предложено зарегистрироваться, указав фамилию, имя и номер телефона. Страница представлена на рисунке 18.

Рисунок 18 – Страница авторизации

Также была разработана панель администратора, включающая в себя две вкладки – «Заказы» и «Сообщения». На вкладке «Заказы» отображаются заказы, сделанные покупателем. Здесь администратор может устанавливать статус заказа, который будет виден заказчику (рисунок 19).

Номер заказа	Дата	Номер телефона	Цена	Состав	Статус
0002	20-05-2023	+79826310441	8500	Торт Красный Бархат, Торт Шоколадный	В процессе
0001	18.05.2023	+79861321921	5000	Торт Ванильный, Набор Праздничный (6 шт)	Готов к выдаче

Добавить новый заказ

Рисунок 19 – Вкладка «Заказы» у администратора

На вкладке «Сообщения» отображаются сообщения об индивидуальных заказах, отправленные пользователями. Администратор получает сообщение, после чего может создать в системе заказ на основе потребностей заказчика, указав в нём всю необходимую информацию – фамилию и имя заказчика, номер телефона, состав и сумму заказа (рисунки 20, 21).

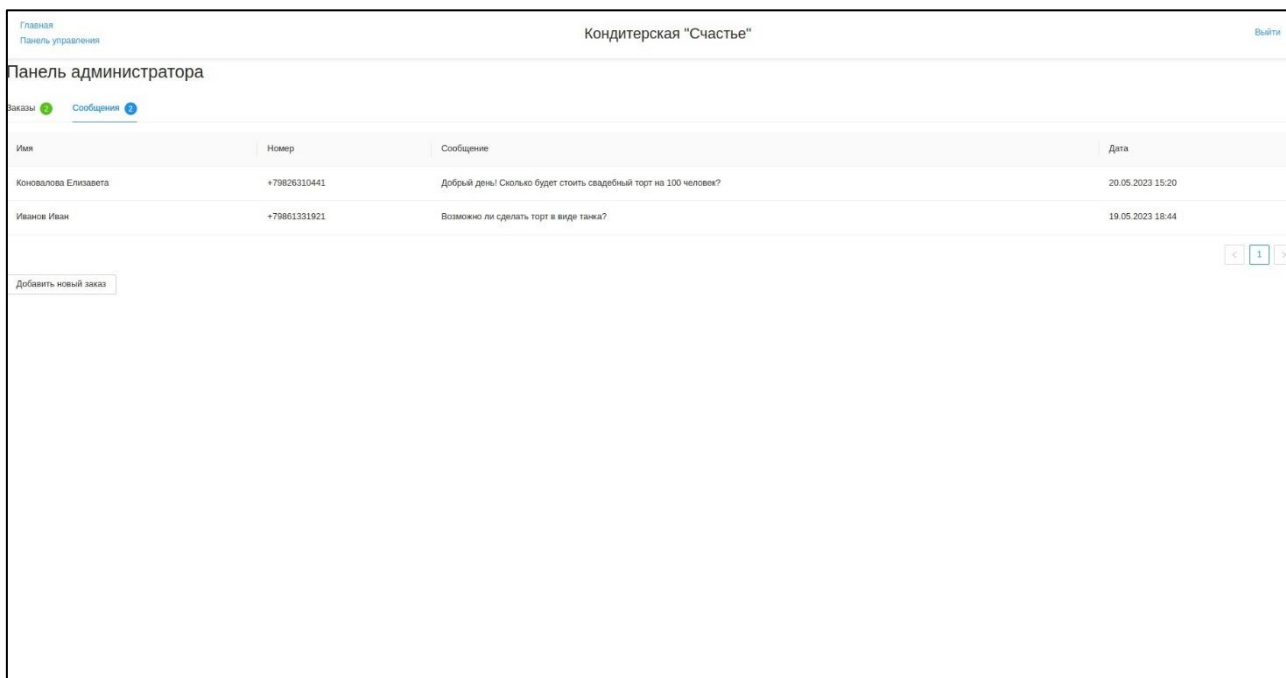


Рисунок 20 – Вкладка «Сообщения»

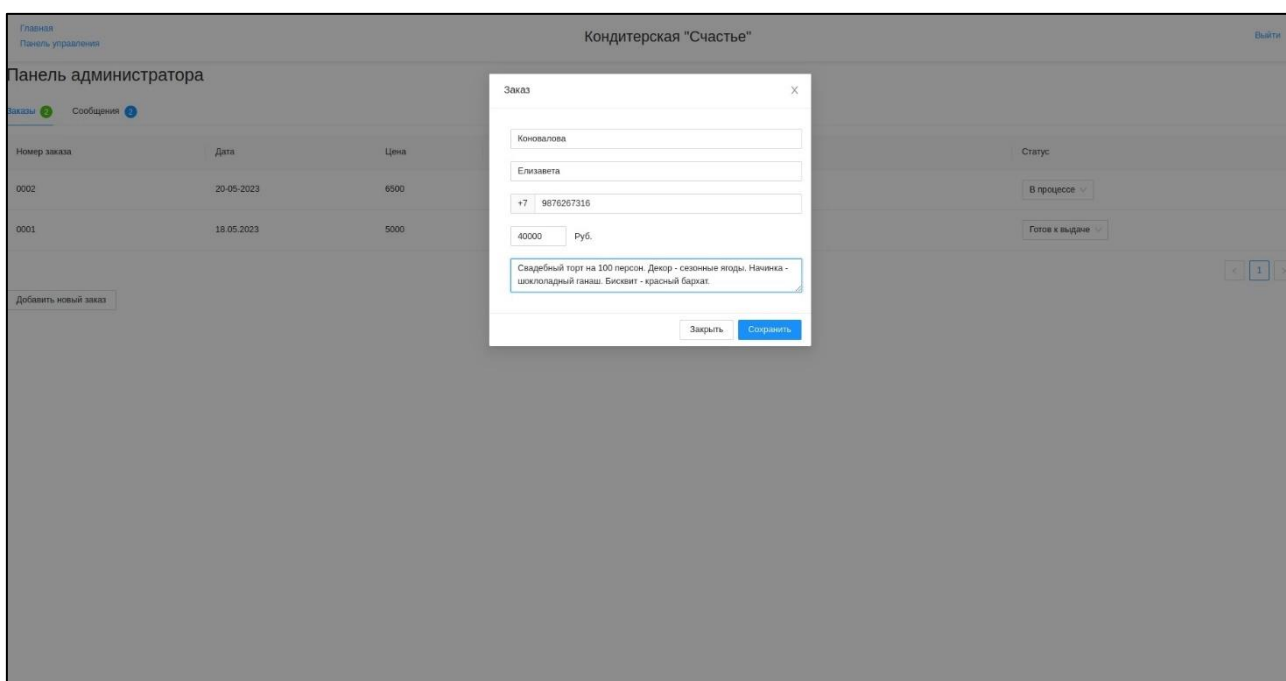


Рисунок 21 – Модальное окно создания нового заказа в системе

Таким образом, был реализован пользовательский интерфейс, удовлетворяющий поставленным требованиям.

Код клиентской части представлен в приложении Б.

5 Тестирование информационной системы

Тестирование программного обеспечения – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта [11].

Выделяют множество классификаций тестирования: по запуску кода на исполнение, по доступу к коду и архитектуре приложения, по степени автоматизации, по уровню детализации приложения, по степени важности тестируемых функций, по принципам работы с приложением, по природе приложения, по целям и задачам, по моменту выполнения и др. В каждой классификации представлено по несколько видов тестирования. Из множества существующих видов тестирования было выбрано применить тестирование методом черного ящика, тестирование метрикой кода, а также автоматизированное тестирование.

5.1 Тестирование формы формирования заказа

Тестирование «черного ящика» подразумевает неиспользование сведений о структуре исходного кода. Все тестовые примеры составляются только на основе требований, т. е. мы не видим, что находится внутри и как оно работает [12].

Тестирование методом черного ящика было проведено для поля номера телефона заказчика. Для этого были определены классы эквивалентности, функциональная диаграмма и таблица решений.

Классом эквивалентности (КЭ) называется набор данных, который запускает одни и те же модули и должен приводить к одним и тем же результатам [13].

В окне формирования заказа представлено поле для ввода номера телефона заказчика. Номер телефона должен быть введен в формате «9XXXXXXXXXX». Для отображения требования к вводимым данным были определены классы эквивалентности, которые представлены в таблице 17.

Таблица 17 – Классы эквивалентности

	Правильные КЭ	Неправильные КЭ
Длина поля	10 (1)	<10 (2) >10 (3) Null (4)
Структура	Начинается с цифры «9» (5)	Начинается не с цифры «9» (6)
Содержание	Цифры (7)	Остальное (8)

Графическое изображение классов представлено на рисунке 19.

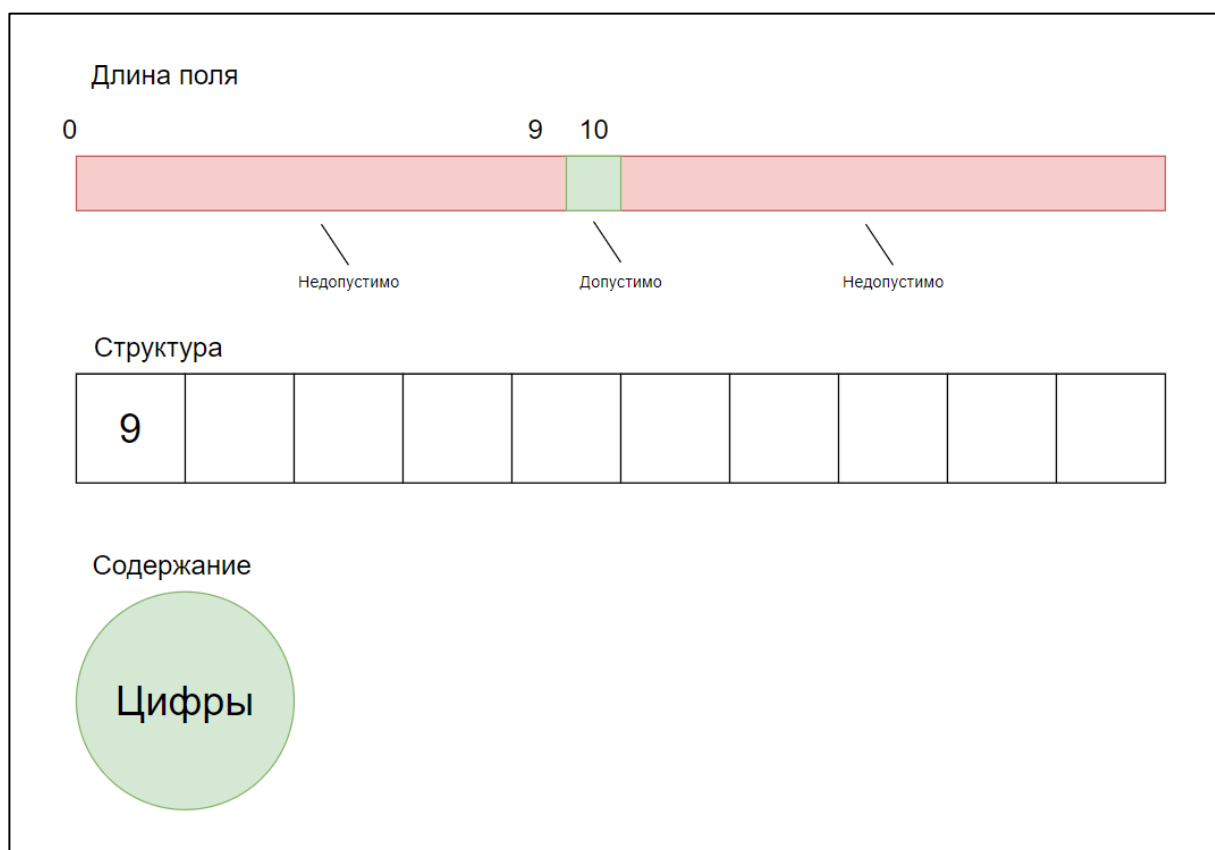


Рисунок 22 – Графическое изображение КЭ

На данном рисунке приведено графическое изображение классов эквивалентности – длина поля, структура и содержание.

Для классов эквивалентности были составлены тест-кейсы, которые приведены в таблице 18.

Таблица 18 – Тест-кейсы

№	КЭ	Входные данные	Выходные данные
0	Все правильные (1, 5, 7)	9001239099	Успешно
1	2, 5, 7	9001239	Ошибка: меньше 10 символов
2	3, 5, 7	900123909999	Ошибка: больше 10 символов
3	4	null	Ошибка: поле должно содержать номер телефона
4	1, 6, 7	8900123909	Ошибка: начинается не с «9»
5	1, 5, 8	900123909A	Ошибка: содержит букву

Для тестирования формы формирования заказа были выделены причины и следствия событий, которые представлены в таблице 19.

Таблица 19 – Причины и следствия

Причины	
П1	Имя введено
П2	Фамилия введена
П3	Номер телефона введен
П4	Номер телефона корректен
Следствия	
С1	Сообщение: «Имя не введено»
С2	Сообщение: «Фамилия не введена»
С3	Сообщение: «Номер телефона не введен»
С4	Сообщение: «Номер телефона введен некорректно»
С5	Сообщение: «Заказ успешно сформирован»

Исходя из этой таблицы была построена функциональная диаграмма, которая представлена на рисунке 20.

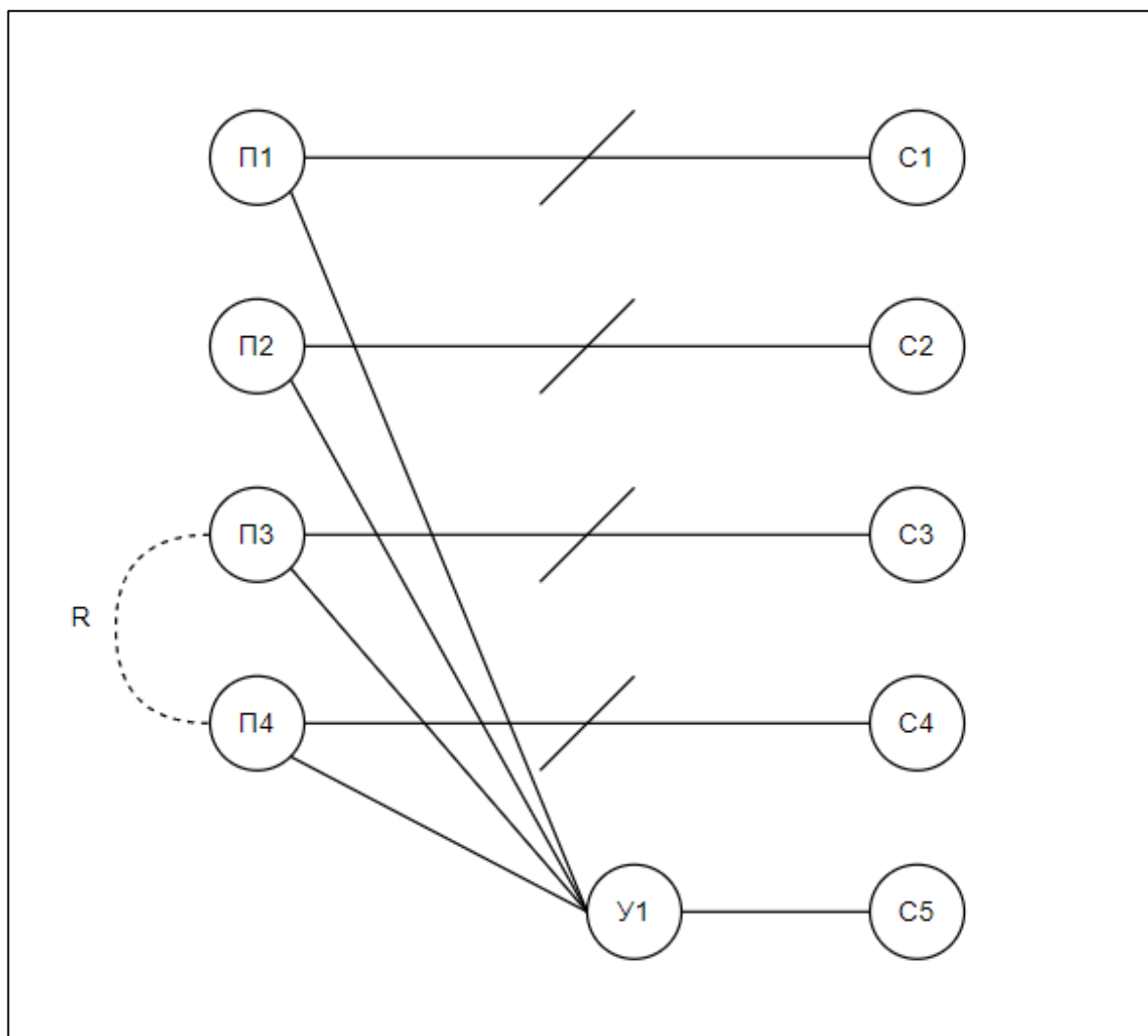


Рисунок 23 – Функциональная диаграмма

На диаграмме представлены причины, следствия и связи между ними. Между причинами 1–4 и следствиями 1–4 есть связи, что означает, что в случае невыполнения причины будет выполняться соответствующее следствие.

Ограничение R означает, что для того, чтобы четвертая причина была истинной, третья причина должна также быть истинной. Это значит, что корректность номера телефона не удастся определить, если он не введен в принципе.

В случае, если все причины выполняются, то выполняется пятое следствие. Оно выводит сообщение о том, что заказ был успешно сформирован.

На основе таблицы причин и следствий была построена таблица решений, которая отражает зависимость причин и следствий друг от друга (таблица 20).

Таблица 20 – Таблица решений

	1	2	3	4	5	6	7	8	9	10	11
П1	0	0	0	0	0	0	1	1	1	1	1
П2	0	0	0	1	1	1	0	0	1	1	1
П3	0	1	1	0	1	1	0	1	0	1	1
П4	0	0	1	0	0	1	0	0	0	0	1
С1	1	1	1	1	1	1	0	0	0	0	0
С2	1	1	1	0	0	0	1	1	0	0	0
С3	1	0	0	1	0	0	1	0	1	0	0
С4	1	1	0	1	1	0	1	1	1	1	0
С5	0	0	0	0	0	0	0	0	0	0	1

В результате были построены наборы тестов по таблице решений. Тест-кейсы представлены в таблице 21.

Таблица 21 – Тест-кейсы по таблице решений

№	Входные данные	Выходные данные	Результаты тестирования
1	Имя: Фамилия: Номер телефона:	Сообщение: «Имя не введено» Сообщение: «Фамилия не введена» Сообщение: «Номер телефона не введен» Сообщение: «Номер телефона введен некорректно»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.1
2	Имя: Фамилия: Номер телефона: 90012390	Сообщение: «Имя не введено» Сообщение: «Фамилия не введена» Сообщение: «Номер телефона введен некорректно»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.2
3	Имя: Фамилия: Номер телефона: 9001239099	Сообщение: «Имя не введено» Сообщение: «Фамилия не введена»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.3
4	Имя: Фамилия: Иванов Номер телефона:	Сообщение: «Имя не введено» Сообщение: «Номер телефона не введен» Сообщение: «Номер телефона введен некорректно»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.4

Продолжение таблицы 21

№	Входные данные	Выходные данные	Результаты тестирования
5	Имя: Фамилия: Иванов Номер телефона: 90012390	Сообщение: «Имя не введено» Сообщение: «Номер телефона введен некорректно»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.5
6	Имя: Фамилия: Иванов Номер телефона: 9001239099	Сообщение: «Имя не введено»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.6
7	Имя: Иван Фамилия: Номер телефона:	Сообщение: «Фамилия не введена» Сообщение: «Номер телефона не введен» Сообщение: «Номер телефона введен некорректно»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.7
8	Имя: Иван Фамилия: Номер телефона: 90012390	Сообщение: «Фамилия не введена» Сообщение: «Номер телефона введен некорректно»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.8
9	Имя: Иван Фамилия: Иванов Номер телефона:	Сообщение: «Номер телефона не введен» Сообщение: «Номер телефона введен некорректно»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.9
10	Имя: Иван Фамилия: Иванов Номер телефона: 90012390	Сообщение: «Номер телефона введен некорректно»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.10
11	Имя: Иван Фамилия: Иванов Номер телефона: 9001239099	Сообщение: «Заказ успешно сформирован»	Результат: тест пройден успешно. Результат приведен в приложении В на рисунке В.11

В результате тестирования методом черного ящика было выявлено, что форма формирования заказа работает корректно.

5.2 Тестирование метрикой Холстеда

Для оценки сложности функции получения корзины по идентификатору сессии была использована метрика Холстеда.

Основу метрики Холстеда составляют четыре измеряемых характеристики программы:

- число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
- число уникальных операндов программы (словарь операндов);
- общее число операторов в программе;
- общее число операндов в программе [14].

Код тестируемой функции представлен в листинге 1.

Листинг 1 – Функция получения корзины по идентификатору сессии

```
pub async fn find_by_session(
    app_data: web::Data<AppData>,
    req: HttpRequest,
) -> Result<HttpResponse, actix_web::Error> {
    let id: Uuid = match
    req.match_info().get("id").unwrap().parse() {
        Ok(uuid) => uuid,
        _ => return Ok(HttpResponse::BadRequest().json("invalid
id")),
    };
    let conn = app_data.conn();
    let res = web::block(move || CartDao::find_by_session(&conn,
id))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::NotFound().finish()
        })?;
    Ok(HttpResponse::Ok().json(res))
}
```

Для оценки сложности функции необходимо посчитать количество операторов и операндов. Число вхождений операторов представлено в таблице 22, операндов – в таблице 23.

Таблица 22 – Операторы функции

Операторы	Число вхождений
:	4
->	1
{	4
}	4
,	7
;	4
=>	2
=	3
.	10
&	1
?	2
return	1
ok	4
n1 = 13	N1 = 47

Таблица 23 – Операнды функции

Операнды	Число вхождений
app_data	2
req	2
id	3
uuid	3
conn	3
res	2
e	2
n2 = 7	N2 = 17

Исходя из этих данных, можно рассчитать сложность функции программы. Для этого необходимо произвести некоторые промежуточные подсчеты.

Длина функции рассчитывается по формуле:

$$N = N1 + N2 = 47 + 17 = 64,$$

где $N1$ – число вхождений операторов;

$N2$ – число вхождений операндов.

Словарь функции рассчитывается по формуле:

$$n = n1 + n2 = 13 + 7 = 20,$$

где $n1$ – количество операторов;

$n2$ – количество операндов.

Объем функции рассчитывается по формуле:

$$NV = N * \log_2(n) = 64 * 4.3219 = 277.1076,$$

где N – длина функции;

n – словарь функции.

Потенциальный объём функции:

$$V^* = (2 + n') * \log_2(2 + n') = 2 * \log_2 2 = 2,$$

где n' – количество уникальных операндов.

Теоретический уровень программы рассчитывается по формуле:

$$L' = \frac{V^*}{NV} = \frac{2}{277.1076} = 0.007215,$$

где V^* – потенциальный объем функции;

NV – объем функции.

Сложность функции рассчитывается по формуле:

$$HD = \frac{n1}{2} * \frac{N2}{n2} = \frac{13}{2} * \frac{17}{7} = 15.786,$$

где $n1$ – количество операторов;

$N2$ – число вхождений операндов;

$n2$ – количество операндов.

Исходя из произведенных расчетов, было определено, что сложность функции получения корзины по идентификатору сессии равна 15.786. Это соответствует показателю простой функции.

5.3 Автоматизированное тестирование

Автоматизированное тестирование ПО – процесс тестирования программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, производятся автоматически с помощью инструментов для автоматизированного тестирования [15].

Было решено разработать автоматизированные тесты для проверки получения корзины по идентификатору сессии и для проверки обновления корзины.

Листинг 2 – Автоматизированные тесты

```
use crate::{find_by_session, Cart};
use diesel::pg::PgConnection;
use diesel::prelude::*;
use diesel::r2d2::{ConnectionManager, Pool, PooledConnection};
use uuid::Uuid;

// Структура для хранения ресурсов базы данных
struct TestDatabase {
    conn: PooledConnection<ConnectionManager<PgConnection>,
}

// Инициализация тестовой базы данных
fn initialize_database() -> TestDatabase {
    // Создание пула соединений к базе данных в памяти
    let database_url = ":memory:";
    let connection_manager =
        ConnectionManager::<PgConnection>::new(database_url);
    let pool = Pool::builder()
        .build(connection_manager)
        .expect("Failed to create database pool");
    let conn = pool.get().expect("Failed to get database connection");

    // Создание схемы и заполнение тестовыми данными
    embedded_migrations::run(&conn).expect("Failed to run database
migrations");

    TestDatabase { conn }
}

#[test]
fn test_find_by_session() {
    // // Инициализация тестовой базы данных
    let test_db = initialize_database();
```

```

let conn = &test_db.conn;
//
// // Вставка тестовых данных в базу данных
let id = uuid::Uuid::default();
let status = "Active".to_string();
let new_cart = NewCart {
    session: id,
    status: status.clone(),
};
diesel::insert_into(schema::cart::table)
    .values(&new_cart)
    .execute(conn)
    .expect("Failed to insert test data");
//
// // Вызов тестируемой функции
let result = find_by_session(conn, id);
//
// // Проверка результата
assert!(result.is_ok());
let cart = result.unwrap();
assert_eq!(cart.session, id);
assert_eq!(cart.status, status);
}

#[test]
fn test_update_cart() {
    // Инициализация тестовой базы данных
    let test_db = initialize_database();
    let conn = &test_db.conn;

    // Создание обновленной корзины
    let updated_cart = Cart {
        id: uuid::Uuid::default(),
        items: Some(vec!["item1".to_string(), "item2".to_string()]),
        total: Some(100.0),
        session: cart.session,
        status: cart.status,
    };

    // Вызов тестируемой функции
    let result = update(conn, cart, updated_cart.clone());

    // Проверка результата
    assert!(result.is_ok());
    let updated_cart_result = result.unwrap();
    assert_eq!(updated_cart_result.items, updated_cart.items);
    assert_eq!(updated_cart_result.total, updated_cart.total);
}

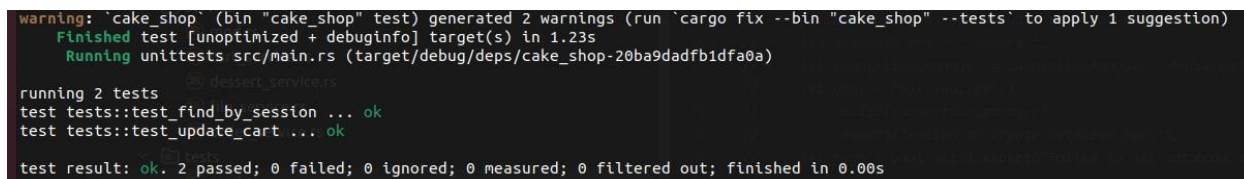
```

Тест «test_find_by_session» проверяет функцию «find_by_session», которая принимает идентификатор сеанса и выполняет запрос к базе данных, чтобы найти корзину с соответствующим идентификатором сеанса. Тест создает

временную базу данных в памяти, заполняет ее тестовыми данными (в данном случае вставляется одна корзина с допустимым идентификатором сеанса), а затем вызывает функцию `find_by_session` с этим идентификатором сеанса. Тест проверяет, что функция возвращает ожидаемый результат (корзину с указанным идентификатором сеанса) и утверждает, что результат успешен.

Тест `test_update_cart` проверяет функцию `update`, которая принимает текущую корзину и обновленную корзину, а затем выполняет обновление данных в базе данных. Тест создает временную базу данных в памяти, вставляет тестовую корзину в базу данных, и затем вызывает функцию `update` с текущей корзиной и обновленной корзиной. Тест проверяет, что функция возвращает ожидаемый результат (обновленную корзину) и утверждает, что результат успешен.

Результат тестирования приведен на рисунке 21.



```

warning: `cake_shop` (bin "cake_shop" test) generated 2 warnings (run `cargo fix --bin "cake_shop" --tests` to apply 1 suggestion)
Finished test [unoptimized + debuginfo] target(s) in 1.23s
Running unittests src/main.rs (target/debug/deps/cake_shop-20ba9dadfb1dfa0a)

running 2 tests
test tests::test_find_by_session ... ok
test tests::test_update_cart ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
  
```

Рисунок 24 – Результат автоматизированного тестирования

Результаты автоматизированного тестирования показали, что функции работают корректно.

Заключение

В результате выполнения задания была разработана информационная система формирования и сопровождения заказов кондитерской организации. Для достижения поставленной задачи были выполнены основные этапы разработки.

На этапе анализа предметной области было выполнено изучение предметной области кондитерской организации, в ходе которого были рассмотрены ключевые моменты, знание которых необходимо для разработки системы кондитерской организации. Также были рассмотрены и проанализированы аналогичные системы, предоставляющие пользователям возможность осуществления заказов. В результате данного анализа были определены некоторые функциональные требования к собственной разрабатываемой системе. Функциональные требования для удобства были отображены на диаграмме вариантов использования.

Было выполнено проектирование схемы базы данных с помощью методологии моделирования схем вида «сущность-связь». В результате данного этапа была построена ER-диаграмма, представляющая модель базы данных. Также были спроектированы серверная и клиентская части.

Далее был осуществлен этап планирования бюджета. Для этого были определены основные задачи и подзадачи проекта. После этого были установлены сроки выполнения данных задач. Также были определены трудовые и материальные ресурсы, требуемые для реализации проекта. В результате подсчетов всех затрат на каждый этап создания продукта было установлено, что стоимость разработки обойдется в 178 300 рублей, а весь процесс займет 37 дней.

На этапе разработки была определена архитектура системы, которая представлена тремя уровнями – клиентом, сервером и базой данных. Также была разработана серверная часть с использованием языка программирования Rust. С помощью языка JavaScript был разработан пользовательский интерфейс, предоставляющий пользователям системы весь функционал для осуществления заказов онлайн.

После разработки был выполнен этап тестирования готовой информационной системы. Тестирование показало, что разработанная система формирования и сопровождения заказов кондитерской организации отвечает установленным функциональным требованиям и в целом работает корректно.

Список использованных источников

1. Диаграмма прецедентов // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_прецедентов (дата обращения: 30.05.2023).
2. Диаграмма Ганта // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_Ганта (дата обращения: 30.05.2023).
3. Проектирование хранилищ данных для приложения систем деловой осведомленности // НОУ ИНТУИТ [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/599/455/lecture/10163> (дата обращения: 30.05.2023).
4. Нотация и семантика языка UML. Лекция 2: Основные элементы языка UML // НОУ ИНТУИТ [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/32/32/lecture/1002> (дата обращения: 30.05.2023).
5. Диаграмма (UML) // Википедия [Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/Диаграмма_\(UML\)](https://ru.wikipedia.org/wiki/Диаграмма_(UML)) (дата обращения: 30.05.2023).
6. Нотация и семантика языка UML. Лекция 12: Элементы графической нотации диаграммы компонентов // НОУ ИНТУИТ [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/32/32/lecture/1022> (дата обращения: 30.05.2023).
7. Графический интерфейс пользователя // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Графический_интерфейс_пользователя (дата обращения: 30.05.2023).
8. Трёхуровневая архитектура // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Трёхуровневая_архитектура (дата обращения: 30.05.2023).
9. Rust (язык программирования) // Википедия [Электронный ресурс]. URL: [https://en.wikipedia.org/wiki/Rust_\(programming_language\)](https://en.wikipedia.org/wiki/Rust_(programming_language)) (дата обращения: 30.05.2023).
10. React // Википедия [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/React> (дата обращения: 30.05.2023).

11. Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. – Минск: Четыре четверти, 2020. – 312 с.
12. Тестирование программного обеспечения // НОУ ИНТУИТ [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/11876/1156/lecture/18264?page=2> (дата обращения: 06.06.2023).
13. Тестирование методом черного ящика // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/462837/> (дата обращения: 06.06.2023).
14. Общие сведения о программном обеспечении // НОУ ИНТУИТ [Электронный ресурс]. URL: https://intuit.ru/studies/professional_skill_improvements/17031/courses/874/lecture/14289?page=2 (дата обращения: 06.06.2023).
15. Автоматизация тестирования программных средств // Habr [Электронный ресурс]. URL: <https://habr.com/ru/articles/160257/> (дата обращения: 06.06.2023).

ПРИЛОЖЕНИЕ А
КОД СЕРВЕРНОЙ ЧАСТИ
(обязательное)

Листинг A.1 – Файл «main.rs»

```
#!/# cake_shop
//!
#[macro_use]
extern crate diesel;
extern crate r2d2;
extern crate r2d2_diesel;

use actix_cors::Cors;
use actix_web::{middleware::Logger, App, HttpServer};
use common::config::app_data::AppData;
use diesel::pg::PgConnection;
use dotenv::dotenv;
use log::{info, LevelFilter};
use r2d2_diesel::ConnectionManager;
use simplelog::{ColorChoice, Config, TermLogger, TerminalMode};

use crate::service::app_config;

mod dao;
mod entity;
mod schema;
mod service;
mod tests;

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    dotenv().ok();
    let config = common::config::Config::from_env().unwrap();
    let main_config = common::config::Config::from_env().unwrap();

    TermLogger::init(
        LevelFilter::Debug,
        Config::default(),
        TerminalMode::Stdout,
        ColorChoice::Always,
    )
    .unwrap();

    let manager =
    ConnectionManager::<PgConnection>::new(main_config.database.url);
    let pool = r2d2::Pool::builder()
        .build(manager)
        .expect("Failed to create postgres pool");

    info!(
        "Starting server at http://{host}:{port}",
        config.server.host, config.server.port
    );
    std::env::set_var("RUST_LOG", "actix_web=info");
```

```

HttpServer::new(move || {
    let cors = Cors::permissive();

    App::new()
        .data(AppData::new(
            Option::from(pool.clone()),
            config.clone(),
            env!("CARGO_PKG_NAME").to_string(),
            env!("CARGO_PKG_VERSION").to_string(),
        ))
        .data(actix_web::web::PayloadConfig::new(1024 * 1024 *
1024))
        .app_data(actix_web::web::JsonConfig::default().limit(
1024 * 1024 * 1024))
        .wrap(Logger::new("%{X-Real-
IP}i %r %s %b %{Referer}i %D"))
        .wrap(cors)
        .configure(app_config)
    })
    .bind(format!(
        "{}:{}",
        main_config.server.host, main_config.server.port
    ))?
    .run()
    .await?;

    Ok(())
}

```

Листинг А.2 – Файл «cart_service.rs»

```

use actix_web::web::Json;
use actix_web::{get, post, put, web, HttpRequest, HttpResponse};
use common::config::app_data::AppData;
use uuid::Uuid;

use crate::dao::cart_dao::CartDao;
use crate::entity::cart::{Cart, CartBuilder};
use crate::entity::item::Item;

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(
        web::scope("cart")
            .service(update)
            .service(find_by_session)
            .service(add_to_cart),
    );
}

#[put("")]
pub async fn update(
    app_data: web::Data<AppData>,

```

```

        updated_cart: Json<Cart>,
    ) -> Result<HttpResponse, actix_web::Error> {
        let conn = app_data.conn();
        let id = updated_cart.id;
        let current_cart = web::block(move || CartDao::find(&conn,
id))
            .await
            .map_err(|e| {
                eprintln!("{:?}", e);
                HttpResponse::NotFound().finish()
            })?;
        let cart_to_update = updated_cart.0;
        let conn = app_data.conn();
        let res = web::block(move || CartDao::update(&conn,
current_cart, cart_to_update))
            .await
            .map_err(|e| {
                eprintln!("{:?}", e);
                HttpResponse::InternalServerError().finish()
            })?;
        Ok(HttpResponse::Ok().json(res))
    }

#[get("{id}")]
pub async fn find_by_session(
    app_data: web::Data<AppData>,
    req: HttpRequest,
) -> Result<HttpResponse, actix_web::Error> {
    let id: Uuid = match
req.match_info().get("id").unwrap().parse() {
        Ok(uuid) => uuid,
        _ => return Ok(HttpResponse::BadRequest().json("invalid
id")),
    };
    let conn = app_data.conn();
    let res = web::block(move || CartDao::find_by_session(&conn,
id))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::NotFound().finish()
        })?;
    Ok(HttpResponse::Ok().json(res))
}

#[post("{session}")]
pub async fn add_to_cart(
    app_data: web::Data<AppData>,
    item: Json<Item>,
    req: HttpRequest,
) -> Result<HttpResponse, actix_web::Error> {
    let session: Uuid = match
req.match_info().get("session").unwrap().parse() {

```

```

        Ok(uuid) => uuid,
        _ => return Ok(HttpResponse::BadRequest().json("invalid
session")),
    };
    let new_item = item.0;
    let mut cart_builder;
    let conn = app_data.conn();
    let cart = match web::block(move ||
CartDao::find_by_session(&conn, session)).await {
        Ok(cart) => {
            cart_builder = CartBuilder::from_cart(&cart);
            cart
        }
        Err(_) => {
            let cart = Cart {
                id: Uuid::new_v4(),
                items: None,
                total: Some(0f32),
                session: Some(session),
                status: "Active".to_string(),
            };
            cart_builder = CartBuilder::from_cart(&cart.clone());
            let conn = app_data.conn();
            let new_cart = cart.clone();
            web::block(move || CartDao::create(&conn, new_cart))
                .await
                .map_err(|e| {
                    eprintln!("{:?}", e);
                    HttpResponse::InternalServerError().finish()
                })?;
            cart
        }
    };
    let new_cart = cart_builder.add_item(new_item).build();
    let conn = app_data.conn();
    let res = web::block(move || CartDao::update(&conn, cart,
new_cart))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::InternalServerError().finish()
        })?;
    Ok(HttpResponse::Ok().json(res))
}

```

Листинг А.3 – Файл «user.rs»

```

use diesel_json::Json;
use serde::{Deserialize, Serialize};
use uuid::Uuid;

use crate::entity::item::Item;

```

```

use super::super::schema::user;

#[derive(
    Clone,
    Serialize,
    Deserialize,
    Queryable,
    Insertable,
    AsChangeset,
    Identifiable,
    QueryableByName,
    Debug,
)]
#[primary_key(id)]
#[table_name = "cart"]
pub struct User {
    pub id: Uuid,
    pub email: String,
    pub passwrod: String,
    pub session: Option<Uuid>,
    pub last_name: String,
    pub first_name: String,
    pub phone: String,
    pub role: String,
}

```

Листинг А.4 – Файл «dessert_service.rs»

```

use actix_web::{get, web, HttpRequest, HttpResponse};
use common::config::app_data::AppData;
use uuid::Uuid;

use crate::dao::dessert_dao::DessertDao;

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(
        web::scope("dessert")
            .service(find_all_cake)
            .service(find_all_cupcake)
            .service(find),
    );
}

#[get("{id}")]
pub async fn find(
    app_data: web::Data<AppData>,
    req: HttpRequest,
) -> Result<HttpResponse, actix_web::Error> {
    let id: Uuid = match
    req.match_info().get("id").unwrap().parse() {
        Ok(uuid) => uuid,
    }
}

```

```

        _ => return Ok(HttpResponse::BadRequest().json("invalid
id")),
    };
    let conn = app_data.conn();
    let res = web::block(move || DessertDao::find(&conn, id))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::NotFound().finish()
        })?;
    Ok(HttpResponse::Ok().json(res))
}

#[get("cakes")]
pub async fn find_all_cake(app_data: web::Data<AppData>) ->
Result<HttpResponse, actix_web::Error> {
    let conn = app_data.conn();
    let res = web::block(move || DessertDao::get_all_cakes(&conn))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::InternalServerError().finish()
        })?;
    Ok(HttpResponse::Ok().json(res))
}

#[get("cupcakes")]
pub async fn find_all_cupcake(
    app_data: web::Data<AppData>,
) -> Result<HttpResponse, actix_web::Error> {
    let conn = app_data.conn();
    let res = web::block(move ||
DessertDao::get_all_cupcakes(&conn))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::InternalServerError().finish()
        })?;
    Ok(HttpResponse::Ok().json(res))
}

```

Листинг A.5 – Файл «order_dao.rs»

```

use diesel::{ExpressionMethods, PgConnection, QueryDsl,
RunQueryDsl};
use uuid::Uuid;

use crate::entity::order::Order;
use crate::schema;
use crate::schema::order::dsl::order;

pub struct OrderDao {}

```

```
impl OrderDao {
    pub fn create(conn: &PgConnection, new_order: Order) ->
Result<usize, diesel::result::Error> {
        diesel::insert_into(order).values(new_order).execute(conn)
    }

    pub fn find(conn: &PgConnection, id: Uuid) -> Result<Order,
diesel::result::Error> {
        order
            .filter(schema::order::id.eq(id))
            .get_result::<Order>(conn)
    }
}
```

Листинг A.6 – Файл «item.rs»

```
use serde::{Deserialize, Serialize};
use uuid::Uuid;

#[derive(Clone, Debug, Serialize, Deserialize)]
#[serde(rename_all = "camelCase")]
pub struct Item {
    pub id: Uuid,
    pub dessert_id: Uuid,
    pub count: u32,
}
```

Листинг A.7 – Файл «file_service.rs»

```
use actix_web::{get, web, HttpRequest, HttpResponse};
use std::fs::File;
use std::io::Read;

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(web::scope("file").service(get_photo_by_name));
}

#[get("{name}")]
pub async fn get_photo_by_name(req: HttpRequest) ->
Result<HttpResponse, actix_web::Error> {
    let name: String = match
req.match_info().get("name").unwrap().parse() {
        Ok(name) => name,
        _ => return Ok(HttpResponse::BadRequest().json("invalid
name")),
    };
    let photo_path = format!("/data/{}", name);
    let photo_file = match File::open(&photo_path) {
        Ok(file) => file,
        Err(_) => return Ok(HttpResponse::NotFound().finish()),
    };
};
```

```

        let content_type = mime_guess::from_path(&photo_path)
            .first_or_octet_stream()
            .to_string();
        Ok(HttpResponse::Ok()
            .content_type(content_type)
            .body(actix_web::web::Bytes::from(
                std::io::BufReader::new(photo_file)
                    .bytes()
                    .map(|b| b.unwrap())
                    .collect::<Vec<u8>>(),
            )))
    }
}

```

Листинг A.8 – Файл «dessert_service.rs»

```

use actix_web::{get, web, HttpRequest, HttpResponse};
use common::config::app_data::AppData;
use uuid::Uuid;

use crate::dao::dessert_dao::DessertDao;

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(
        web::scope("dessert")
            .service(find_all_cake)
            .service(find_all_cupcake)
            .service(find),
    );
}

#[get("{id}")]
pub async fn find(
    app_data: web::Data<AppData>,
    req: HttpRequest,
) -> Result<HttpResponse, actix_web::Error> {
    let id: Uuid = match
    req.match_info().get("id").unwrap().parse() {
        Ok(uuid) => uuid,
        _ => return Ok(HttpResponse::BadRequest().json("invalid
id")),
    };
    let conn = app_data.conn();
    let res = web::block(move || DessertDao::find(&conn, id))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::NotFound().finish()
        })?;
    Ok(HttpResponse::Ok().json(res))
}

#[get("cakes")]

```

```

pub async fn find_all_cake(app_data: web::Data<AppData>) ->
Result<HttpResponse, actix_web::Error> {
    let conn = app_data.conn();
    let res = web::block(move || DessertDao::get_all_cakes(&conn))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::InternalServerError().finish()
        })?;
    Ok(HttpResponse::Ok().json(res))
}

#[get("cupcakes")]
pub async fn find_all_cupcake(
    app_data: web::Data<AppData>,
) -> Result<HttpResponse, actix_web::Error> {
    let conn = app_data.conn();
    let res = web::block(move ||
DessertDao::get_all_cupcakes(&conn))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::InternalServerError().finish()
        })?;
    Ok(HttpResponse::Ok().json(res))
}

```

Листинг А.9 – Файл «cart.rs»

```

use diesel_json::Json;
use serde::{Deserialize, Serialize};
use uuid::Uuid;

use crate::entity::item::Item;

use super::super::schema::cart;

#[derive(
    Clone,
    Serialize,
    Deserialize,
    Queryable,
    Insertable,
    AsChangeset,
    Identifiable,
    QueryableByName,
    Debug,
)]
#[primary_key(id)]
#[table_name = "cart"]
pub struct Cart {
    pub id: Uuid,

```

```

    pub items: Option<Json<Vec<Item>>>,
    pub total: Option<f32>,
    pub session: Option<Uuid>,
    pub status: String,
}

#[derive(Clone, Serialize, Deserialize, Debug)]
pub struct CartBuilder {
    pub id: Uuid,
    pub items: Option<Json<Vec<Item>>>,
    pub total: Option<f32>,
    pub session: Option<Uuid>,
    pub status: String,
}

impl CartBuilder {
    pub fn new() -> CartBuilder {
        CartBuilder {
            id: Uuid::new_v4(),
            items: None,
            total: None,
            session: None,
            status: "Active".to_string(),
        }
    }

    pub fn from_cart(cart: &Cart) -> CartBuilder {
        CartBuilder {
            id: cart.id,
            items: cart.items.clone(),
            total: cart.total,
            session: cart.session,
            status: cart.status.clone(),
        }
    }

    pub fn add_item(mut self, item: Item) -> CartBuilder {
        match self.items {
            None => {
                let mut items = vec![];
                items.push(item);
                self.items = Some(Json::new(items))
            }
            Some(items) => {
                let mut items = items.0;
                items.push(item);
                self.items = Some(Json::new(items))
            }
        }
        self
    }

    pub fn build(self) -> Cart {

```

```

        Cart {
            id: self.id,
            items: self.items,
            total: self.total,
            session: self.session,
            status: self.status,
        }
    }
}

```

Листинг A.10 – Файл «dessert_dao.rs»

```

use diesel::{sql_query, ExpressionMethods, PgConnection, QueryDsl,
RunQueryDsl};
use uuid::Uuid;

use crate::entity::dessert::Dessert;
use crate::schema;
use crate::schema::dessert::dsl::dessert;

pub struct DessertDao {}

impl DessertDao {
    pub fn find(conn: &PgConnection, id: Uuid) -> Result<Dessert,
diesel::result::Error> {
        dessert
            .filter(schema::dessert::id.eq(id))
            .get_result::<Dessert>(conn)
    }

    pub fn get_all_cakes(conn: &PgConnection) ->
Result<Vec<Dessert>, diesel::result::Error> {
        sql_query("select * from dessert d where d.type_dessert =
'cake'").load(conn)
    }

    pub fn get_all_cupcakes(conn: &PgConnection) ->
Result<Vec<Dessert>, diesel::result::Error> {
        sql_query("select * from dessert d where d.type_dessert =
'cupcake'").load(conn)
    }
}

```

Листинг A.11 – Файл «order_service.rs»

```

use actix_web::web::Json;
use actix_web::{post, web, HttpResponse};
use common::config::app_data::AppData;

use crate::dao::cart_dao::CartDao;
use crate::dao::order_dao::OrderDao;
use crate::entity::order::Order;

```

```

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(web::scope("order").service(create));
}

#[post("")]
pub async fn create(
    app_data: web::Data<AppData>,
    order: Json<Order>,
) -> Result<HttpResponse, actix_web::Error> {
    let new_order = order.0;
    let conn = app_data.conn();
    let id = new_order.id;
    let cart_id = new_order.cart;
    web::block(move || OrderDao::create(&conn, new_order))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::InternalServerError().finish()
        })?;
    let conn = app_data.conn();
    let cart = web::block(move || CartDao::find(&conn, cart_id))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::NotFound().finish()
        })?;
    let mut new_cart = cart.clone();
    new_cart.status = "Ordered".to_string();
    let conn = app_data.conn();
    web::block(move || CartDao::update(&conn, cart, new_cart))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::NotFound().finish()
        })?;
    let conn = app_data.conn();
    let res = web::block(move || OrderDao::find(&conn, id))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::InternalServerError().finish()
        })?;
    Ok(HttpResponse::Created().json(res))
}

```

ПРИЛОЖЕНИЕ Б
КОД КЛИЕНТСКОЙ ЧАСТИ
(обязательное)

Листинг Б.1 – Файл «admin.js»

```
import React, {useState} from 'react';
import {Tabs, Badge, Table, Select, Button, Modal, Input,
InputNumber} from 'antd';
import TextArea from "antd/es/input/TextArea";

const {TabPane} = Tabs;
const {Option} = Select;

const AdminPanel = () => {

  const [isModalOpen, setIsModalOpen] = useState(false)
  const showModal = () => {
    setIsModalOpen(true);
  };
  const handleCancel = () => {
    setIsModalOpen(false);
  };

  const activeOrdersCount = orders.filter((order) =>
order.status !== 'Доставлен').length;
  const processedOrdersCount = orders.filter((order) =>
order.status === 'Доставлен').length;

  const handleOrderStatusChange = (orderId, newStatus) => {
    const updatedOrders = orders.map((order) => {
      if (order.id === orderId) {
        return {...order, status: newStatus};
      }
      return order;
    });
    setOrders(updatedOrders);
  };

  const columns = [
    {title: 'Номер заказа', dataIndex: 'number', key:
'number'},
    {title: 'Дата', dataIndex: 'date', key: 'date'},
    {title: 'Номер телефона', dataIndex: 'phone', key:
'phone'},
    {title: 'Цена', dataIndex: 'price', key: 'price'},
    {title: 'Состав', dataIndex: 'items', key: 'items',
render: (items) => items.join(', ')},
    {
      title: 'Статус',
      dataIndex: 'status',
      key: 'status',
      render: (status, record) => (
        <Select value={status} onChange={(newStatus) =>
handleOrderStatusChange(record.id, newStatus)}>
```

```

        <Option value="В обработке">В
обработке</Option>
        <Option value="Принят">Принят</Option>
        <Option value="В процессе">В процессе</Option>
        <Option value="Готов к выдаче">Готов к
выдаче</Option>
        <Option value="Выдан">Выдан</Option>
        <Option value="Отменен">Отменен</Option>
    </Select>
    ),
  },
];

function handleOk() {

  const activeOrdersCount = orders.filter((order) =>
order.status !== 'Доставлен').length;
  const processedOrdersCount = orders.filter((order) =>
order.status === 'Доставлен').length;

  const handleOrderStatusChange = (orderId, newStatus) => {
    const updatedOrders = orders.map((order) => {
      if (order.id === orderId) {
        return {...order, status: newStatus};
      }
      return order;
    });
    setOrders(updatedOrders);
  };
  return (
    <div>
      <Modal
        title="Заказ"
        visible={isModalOpen}
        onCancel={handleCancel}
        footer={[
          <Button key="back" onClick={handleCancel}>
            Закрыть
          </Button>,
          <Button key="submit" type="primary"
onClick={handleOk}>
            Сохранить
          </Button>,
        ]}
      >

      <Input style={{margin:"10px"}}
        placeholder="Фамилия"

      />
      <Input style={{margin:"10px"}}

```

```

        placeholder="Имя"

    />
    <Input style={{margin:"10px"}} addOnBefore="+7"
        placeholder="Номер телефона"

    />
    <InputNumber style={{margin:"10px"}}
        placeholder="Цена"

    /> Руб.
    <TextArea style={{margin:"10px"}}
        placeholder="Состав заказа"

    />

</Modal>
<h1>Панель администратора</h1>
<Tabs defaultActiveKey="orders">
    <TabPane
        tab={
            <span>
                Заказы
                <Badge count={activeOrdersCount} style={{marginLeft:
'8px', backgroundColor: '#52c41a'}}/>
            </span>
        }
        key="orders"
    >
        <Table columns={columns} dataSource={orders}
rowKey="id"/>
    </TabPane>
    <TabPane
        tab={
            <span>
                Сообщения
                <Badge count={messages.length} style={{marginLeft:
'8px', backgroundColor: '#1890ff'}}/>
            </span>
        }
        key="messages"
    >
        <Table columns={messageColumns}
dataSource={messages} rowKey="id"/>
    </TabPane>
</Tabs>
<Button onClick={() =>
    setIsModalOpen(true)}
    > Добавить новый заказ</Button>
</div>

);
};

```

```
const messageColumns = [
  {title: 'Имя', dataIndex: 'username', key: 'username'},
  {title: 'Номер', dataIndex: 'phone', key: 'phone'},
  {title: 'Сообщение', dataIndex: 'text', key: 'text'},
  {title: 'Дата', dataIndex: 'date', key: 'date'},
];

export default AdminPanel;
```

Листинг Б.2 – Файл «cake.js»

```
import React, {useEffect, useState} from "react";
import {Button, Table} from "antd";
import {paths} from "../utils/paths";
import {getCartSingleton} from "../utils/cartSingleton";
import {v4, v4 as uuidv4} from 'uuid';
import 'antd/dist/antd.min.css';
import HTTPWrapper from "../utils/HTTPWrapper";

const Cakes = () => {
  const http = new HTTPWrapper(paths.local)
  const [cakes, setCakes] = useState([])

  useEffect(() => {
    async function getAllCakes() {
      const res = await
http.get(paths.requests.desserts.getAllCakes)
      setCakes(res)
    }

    getAllCakes()
  }, [])

  const [cartItems, setCartItems] = useState([]);

  function addToCart(product) {
    const mySingleton = getCartSingleton();
    const item = {id: v4(), dessertId: product.id, count: 1}
    http.post(paths.requests.cart.addToCart +
mySingleton.getSession(), item)
    mySingleton.addToCart(item)
    setCartItems([...cartItems, product]);
  };

  const columns = [
    {
      title: 'Photo',
      dataIndex: 'photo',
      key: 'photo',
      render: (photoUrl, record) => (
```

```

        <img src={paths.local +
paths.requests.file.getImage + photoUrl} alt={record.name}
width={200}/>
      ),
    },
    {
      title: 'Name',
      dataIndex: 'name',
      key: 'name',
    },
    {
      title: 'Price',
      dataIndex: 'price',
      key: 'price',
    },
    {
      title: '',
      key: 'action',
      render: (_, record) => (
        <Button onClick={() => addToCart(record)}>Добавить
в корзину</Button>
      ),
    },
  ],
);

return (
  <div style={{
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
  }}>
    <div style={{margin: 50}}>
      <Table
        rowKey={record => record.id}
        dataSource={cakes.json}
        columns={columns}
        pagination={false}
        showHeader={false}
        bordered={true}
      />
    </div>
  </div>
);
}
export default Cakes

```

Листинг Б.3 – Файл «cart.js»

```

import React, {useEffect, useState} from 'react';
import {Table, InputNumber, Button, Modal, Input, Alert, message}
from 'antd';
import 'antd/dist/antd.min.css';

```

```

import HTTPWrapper from "../utils/HTTPWrapper";
import {paths} from "../utils/paths";
import {getCartSingleton} from "../utils/cartSingleton";
import {v4} from "uuid";

const columns = [
  {
    title: 'Изображение',
    dataIndex: 'photo',
    render: (value) => <img src={value} alt="Product"
width="50"/>,
  },
  {
    title: 'Наименование',
    dataIndex: 'name',
  },
  {
    title: 'Цена',
    dataIndex: 'price',
    render: (value) => ` ${value.toFixed(2)} `,
  },
  {
    title: 'Количество',
    dataIndex: 'count',
    render: (value, record) => (
      <InputNumber
        min={1}
        max={10}
        value={value}
        onChange={(quantity) =>
          record.onQuantityChange(record.id, quantity)
        }
      />
    ),
  },
  {
    title: '',
    dataIndex: 'delete',
    render: (value, record) => (
      <Button>удалить</Button>
    ),
  },
];

const Cart = ({products}) => {
  const http = new HTTPWrapper(paths.local)
  const [cart, setCart] = useState([])
  const [items, setItems] = useState([])
  const [isModalVisible, setIsModalVisible] = useState(false);

  useEffect(() => {
    async function getCart() {
      setItems([])
    }
  })
}

```

```

        const mySingleton = getCartSingleton();
        const res = await
http.get(paths.requests.cart.getCartBySession +
mySingleton.getSession())
        setCart(res.json)
        const items = res.json.items;
        for (let i = 0; i < items.length; i++) {
            const dessert = await
http.get(paths.requests.desserts.getById + items[i].dessertId)
            items[i].name = dessert.json.name;
            items[i].photo = paths.local +
paths.requests.file.getImage + dessert.json.photo;
            items[i].price = dessert.json.price;
        }
        setItems(items)
    }

    getCart()
}, [isModalVisible])

```

```

const handleQuantityChange = async (productId, count) => {
    setItems((prevProducts) =>
        prevProducts.map((product) =>
            product.id === productId
                ? {
                    ...product,
                    count,
                }
                : product
        )
    );
    for (let i = 0; i < cart.items.length; i++) {
        if (cart.items[i].id === productId) {
            cart.items[i].count = count
        }
    }
    await http.put(paths.requests.cart.update, cart);
};

```

```

const getTotalCost = () => {
    return items.reduce(
        (total, {price, count}) => total + price * count,
        0
    );
};

```

```

const showModal = () => {

    setIsModalVisible(true);
};
const handleCancel = () => {
    setIsModalVisible(false);
};

```

```

    };
    const [formData, setFormData] = useState({id: null,
first_name: null, last_name: null, phone: null, cart: null});

    const handleOk = async () => {
        formData.id = v4()
        formData.cart = cart.id
        const res = await http.post(paths.requests.order.addOrder,
formData)
        setFormData({first_name: null, last_name: null, phone:
null, id: null});
        setIsModalVisible(false);
    };
    return (
        <>
            <Modal
                title="Подтвердите или измените данные"
                visible={isModalVisible}
                onCancel={handleCancel}
                footer={
                    <Button key="back" onClick={handleCancel}>
                        Заккрыть
                    </Button>,
                    <Button key="submit" type="primary"
onClick={handleOk}>
                        Подтвердить
                    </Button>,
                ]}
            >

                <Input

                    placeholder="Фамилия"
                    value={formData.last_name}
                    onChange={(e) => setFormData({...formData,
last_name: e.target.value})}
                />

                <Input
                    placeholder="Имя"
                    value={formData.first_name}
                    onChange={(e) => setFormData({...formData,
first_name: e.target.value})}
                />

                <Input addonBefore="+7"
                    placeholder="Номер телефона"
                    value={formData.phone}
                    onChange={(e) => setFormData({...formData,
phone: e.target.value})}

```

```

        />
      </Modal>
      <div style={{
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
      }}>
        <div style={{margin: 50}}>
          <Table
            columns={columns}
            dataSource={items.map((product) => ({
              ...product,
              key: product.id,
              onQuantityChange:
handleQuantityChange,
            })))}
            pagination={false}
            summary={() => (
              <Table.Summary.Row>

<Table.Summary.Cell>Итого</Table.Summary.Cell>
              <Table.Summary.Cell/>
              <Table.Summary.Cell/>
              <Table.Summary.Cell/>

<b>{getTotalCost().toFixed(2)}</b>
              </Table.Summary.Cell>
              <Table.Summary.Cell/>
            </Table.Summary.Row>
          )}
        </div>
        <Button type="primary" block
onClick={showModal}>
          Заказать
        </Button>
      </div>
    </div>
  </>

  );
};

export default Cart;

```

Листинг Б.4 – Файл «contactPage.js»

```

import { Form, Input, Button, Row, Col, Divider } from 'antd';
import styled from 'styled-components';
import HTTP from "../utils/HTTPWrapper";
import React from "react";

const StyledRow = styled(Row) `

```

```

padding: 20px;
`;

const ContactPage = () => {
  const [form] = Form.useForm();
  const http = new HTTP()
  const onFinish = async (values) => {
    const res = await http.post("feedback", values);
    if (res.status === 201) {
      form.resetFields()
      alert("Успешно!")
    }
  };

  return (
    <StyledRow justify="center">
      <Col xs={24} md={18} lg={12}>
        <h1>О нас</h1>
        <Divider />
        <p>Кондитерская организация "Счастье" - место, где
создаются неповторимые десерты, чтобы приносить радость и восторг
каждому нашему клиенту.</p>
        <p>Мы гордимся тем, что используем только лучшие
ингредиенты и традиционные рецепты, чтобы каждый наш продукт был
настоящим произведением искусства.</p>
        <p>Вы можете связаться с нами по телефону или
отправить нам письмо на электронную почту. Ниже приведены наши
контактные данные для связи с нами.</p>
        <Divider />
        <h2>Адрес:</h2>
        <p>г. Пенза, ул. Лучезарная, д. 123</p>
        <h2>Телефон:</h2>
        <p>+7 900 000 00 01</p>
        <h2>Электронная почта:</h2>
        <p>info@happiness.com</p>
        <Divider />
        <h2>Вы хотите сделать индивидуальный заказ?
Напишите нам!</h2>
        <Form onFinish={onFinish} form={form}>
          <Form.Item
            label="Имя"
            name="last_name"
            rules={[
              {
                required: true,
                message: 'Пожалуйста, введите ваше
имя',
              },
            ]},
          </Form.Item>
          <Form.Item
            label="Телефон"
            name="phone"
            rules={[
              {
                required: true,
                message: 'Пожалуйста, введите ваш телефон',
              },
            ]},
          </Form.Item>
          <Form.Item
            label="Email"
            name="email"
            rules={[
              {
                required: true,
                message: 'Пожалуйста, введите ваш email',
              },
            ]},
          </Form.Item>
          <Form.Item
            label="Сообщение"
            name="message"
            rules={[
              {
                required: true,
                message: 'Пожалуйста, введите ваше сообщение',
              },
            ]},
          </Form.Item>
          <Form.Item>
            <Input type="submit" value="Отправить" />
          </Form.Item>
        </Form>
      </Col>
    </StyledRow>
  );
};

```

```

<Form.Item
  label="Фамилия"
  name="name"
  rules={[
    {
      required: true,
      message: 'Пожалуйста, введите ваше
имя',
    },
  ]}
>
  <Input />
</Form.Item>

<Form.Item
  label="Номер телефона"
  name="phone"
  rules={[
    {
      required: true,
      message: 'Пожалуйста, введите
номер',
    },
    {
      type: 'phone',
      message: 'Неправильный формат ',
    },
  ]}
>
  <Input addonBefore="+7" />
</Form.Item>
<Form.Item
  label="Сообщение"
  name="text"
  rules={[
    {
      required: true,
      message: 'Пожалуйста, введите ваше
сообщение',
    },
  ]}
>
  <Input.TextArea />
</Form.Item>

<Form.Item>
  <Button type="primary" htmlType="submit">
    Отправить
  </Button>
</Form.Item>
</Form>
</Col>
</StyledRow>

```

```

    );
};

export default ContactPage;

```

Листинг Б.5 – Файл «cupcakes.js»

```

import {Button, Space, Table, Tag} from 'antd';
import React, {useEffect, useState} from 'react';
import {paths} from "../utils/paths";
import 'antd/dist/antd.min.css';
import HTTPWrapper from "../utils/HTTPWrapper";
import {getCartSingleton} from "../utils/cartSingleton";
import {v4} from "uuid";
const { Column, ColumnGroup } = Table;

const Cupcakes = () => {
  const http = new HTTPWrapper(paths.local)
  const [cupcakes, setCupcakes] = useState([])

  useEffect(() => {
    async function getAllCupcakes() {
      const res = await
http.get(paths.requests.desserts.getAllCupcakes)
      setCupcakes(res)
    }
    getAllCupcakes()
  }, [])

  const [cartItems, setCartItems] = useState([]);

  const addToCart = (product) => {
    const mySingleton = getCartSingleton();
    const item = {id: v4(), dessertId: product.id, count: 1}
    http.post(paths.requests.cart.addToCart +
mySingleton.getSession(), item)
    mySingleton.addToCart(item)
    setCartItems([...cartItems, product]);
  };

  const columns = [
    {
      title: 'Photo',
      dataIndex: 'photo',
      key: 'photo',
      render: (photoUrl, record) => (
        <img src={paths.local +
paths.requests.file.getImage + photoUrl} alt={record.name}
width={200}/>
      ),
    },
    {

```

```

        title: 'Name',
        dataIndex: 'name',
        key: 'name',
      },
      {
        title: 'Price',
        dataIndex: 'price',
        key: 'price',
      },
      {
        title: '',
        key: 'action',
        render: (_, record) => (
          <Button onClick={() => addToCart(record)}>Добавить
в корзину</Button>
        ),
      },
    ],
  );

  return (
    <div style={{
      display: 'flex',
      justifyContent: 'center',
      alignItems: 'center',
    }}>
      <div style={{margin: 50}}>
        <Table
          rowKey={record => record.id}
          dataSource={cupcakes.json}
          columns={columns}
          pagination={false}
          showHeader={false}
          bordered={true}
        />
      </div>
    </div>
  );
}
export default Cupcakes

```

Листинг Б.6 – Файл «home.js»

```

import React, {useEffect, useState} from "react";
import cake from '../img/cake.png'
import cupcake from '../img/cupcake.png'
import {paths} from "../utils/paths";
import {NavLink} from "react-router-dom";
import jsCookie from "js-cookie";
import {v4} from "uuid";

const Home = () => {
  const sessionId = jsCookie.get("SessionId")

```

```

    if (!sessionId){
      jsCookie.set("SessionId", v4())
    }

    useEffect(() => {

      document.body.style.overflow = 'hidden';
      return () => {
        document.body.style.overflow = 'auto';
      };
    }, []);

    return (<>
      <div style={{
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
        height: '100vh',
        marginTop: '50vh',
        transform: 'translateY(-50%)'
      }}>
        <NavLink to={paths.cakes}>
          <div style={{display: 'flex', flexDirection:
'column', alignItems: 'center'}}>
            <img src={cake} alt="First Photo"
style={{width: '70%'}}/>
            <p style={{textAlign:
'center'}}><h1>Торты</h1></p>
          </div>
        </NavLink>
        <NavLink to={paths.cupcakes}>
          <div style={{display: 'flex', flexDirection:
'column', alignItems: 'center'}}>
            <img src={cupcake} alt="Second Photo"
style={{width: '70%'}}/>
            <p style={{textAlign:
'center'}}><h1>Купкейки</h1></p>
          </div>
        </NavLink>
      </div>
    </>
  )
}
export default Home

```

Листинг Б.7 – Файл «lk.js»

```

import React, {useState} from 'react';
import {Tabs, Badge, Table, Select, Button, Modal, Input,
InputNumber, Form} from 'antd';
const {TabPane} = Tabs;

```

```

const AccountPage = () => {
  const ordersData = [
    {
      orderId: '0002',
      date: '20.05.2023',
      time: '15:30',
      amount: 6500,
      status: 'В процессе',
      items: [
        { name: 'Торт Красный Бархат ', quantity: 1 },
        { name: 'Торт Шоколадный', quantity: 1 },
      ],
    },
    {
      orderId: '0001',
      date: '18.05.2023',
      time: '12:15',
      amount: 5000,
      status: 'Готов к выдаче',
      items: [
        { name: 'Торт Ванильный', quantity: 1 },
        { name: 'Набор Праздничный (6 шт)', quantity: 1 },
      ],
    },
  ],

  const columns = [
    { title: 'Номер заказа', dataIndex: 'orderId', key:
'orderId' },
    { title: 'Дата', dataIndex: 'date', key: 'date' },
    { title: 'Время', dataIndex: 'time', key: 'time' },
    { title: 'Сумма заказа', dataIndex: 'amount', key:
'amount' },
    {
      title: 'Состав заказа',
      dataIndex: 'items',
      key: 'items',
      render: (items) => (
        <ul>
          {items.map((item, index) => (
            <li key={index}>{`${item.name}
(Количество: ${item.quantity})`}</li>
          ))}
        </ul>
      ),
    },
    { title: 'Статус', dataIndex: 'status', key: 'status' },
  ],

  const [form] = Form.useForm();
  const [isEditing, setIsEditing] = useState(false);

```

```

const handleEdit = () => {
  setIsEditing(true);
};

const handleSubmit = (values) => {
  console.log(values); // Обработка отправки данных формы
  setIsEditing(false);
};

return (
  <>
    <Tabs style={{marginLeft: '8px'}}
defaultActiveKey="orders">
      <TabPane
        tab={
          <span>
            Профиль
            <Badge style={{marginLeft: '8px', backgroundColor:
'#52c41a'}}/>
          </span>
        }
        key="orders"
      >
        <div style={{ display: 'flex', justifyContent:
'center' }}>
          <Form form={form} onFinish={handleSubmit}
layout="vertical" style={{ width: '300px' }}>
            <Form.Item
              label="Фамилия"
              name="lastName"
              rules={[{ required: true, message:
'Пожалуйста, введите фамилию' }]}
            >
              <Input disabled={!isEditing} />
            </Form.Item>

            <Form.Item
              label="Имя"
              name="firstName"
              rules={[{ required: true, message:
'Пожалуйста, введите имя' }]}
            >
              <Input disabled={!isEditing} />
            </Form.Item>

            <Form.Item
              label="Email"
              name="email"
              rules={[
                { required: true, message:
'Пожалуйста, введите email' },
                { type: 'email', message:
'Неверный формат email' },

```

```

    ]]
  >
    <Input disabled={!isEditing} />
  </Form.Item>

  <Form.Item
    label="Пароль"
    name="password"
    rules={[{ required: true, message:
'Пожалуйста, введите пароль' }]}
  >
    <Input.Password
disabled={!isEditing} />
  </Form.Item>

  <Form.Item
    label="Номер телефона"
    name="phone"
    rules={[{ required: true, message:
'Пожалуйста, введите номер телефона' }]}
  >
    <Input addonBefore="+7"
disabled={!isEditing} />
  </Form.Item>

  {!!isEditing && (
    <Form.Item>
      <Button type="primary"
onClick={handleEdit}>Редактировать</Button>
    </Form.Item>
  )}

  {isEditing && (
    <Form.Item>
      <Button type="primary"
htmlType="submit">Сохранить</Button>
    </Form.Item>
  )}
</Form>
</div>

</TabPane>
<TabPane
  tab={
    <span>
      Заказы
      <Badge style={{marginLeft: '8px', backgroundColor:
'#1890ff'}}/>
    </span>
  }
  key="messages"
>
  <div>

```

```

                                <br/>
                                <h1 style={{ textAlign: 'center' }}>Мои
заказы</h1>
                                <br/>
                                <Table dataSource={ordersData}
columns={columns} />
                                </div>
                                </TabPane>
                                </Tabs>
                                </>

    );
};

export default AccountPage;

```

Листинг Б.8 – Файл «loginForm.js»

```

import React, { useState } from 'react';
import { Form, Input, Button } from 'antd';

const LoginForm = () => {
    const [form] = Form.useForm();
    const [isRegistration, setIsRegistration] = useState(false);

    const handleFormSubmit = (values) => {
        console.log('Form values:', values);
        // Здесь можно выполнить соответствующие действия,
        например, отправить запрос на сервер для авторизации или
        регистрации.
    };

    const toggleRegistration = () => {
        setIsRegistration(!isRegistration);
        form.resetFields();
    };

    return (
        <div style={{ display: 'flex', justifyContent: 'center',
alignItems: 'center', height: '100vh' }}>
            <Form form={form} onFinish={handleFormSubmit}
layout="vertical">
                <Form.Item
                    label="Email"
                    name="email"
                    rules={[{ required: true, message: 'Введите
email' }]}
                    style={{ width: '300px' }}
                >
                    <Input type="email" size="small" />
                </Form.Item>
                <Form.Item

```

```

        label="Пароль"
        name="password"
        rules=[{ required: true, message: 'Введите
пароль' }]]
        style={{ width: '300px' }}
    >
        <Input.Password size="small" />
    </Form.Item>

    {isRegistration && (<
        <Form.Item
            label="Номер телефона"
            name="phoneNumber"
            rules=[{ required: true, message:
'Введите номер телефона' }]]
            style={{ width: '300px' }}
        >
            <Input addonBefore="+7" size="small"
/>

        </Form.Item>
        <Form.Item
            label="Фамилия"
            name="last_name"
            rules=[{ required: true, message:
'Введите фамилию' }]]
            style={{ width: '300px' }}
        >
            <Input size="small" />
        </Form.Item>
        <Form.Item
            label="Имя"
            name="first_name"
            rules=[{ required: true, message:
'Введите имя' }]]
            style={{ width: '300px' }}
        >
            <Input size="small" />
        </Form.Item>
    </>

    )}

    <Form.Item>
        <Button type="primary" htmlType="submit">
            {isRegistration ? 'Зарегистрироваться' :
'Войти'}
        </Button>
        <Button type="link"
onClick={toggleRegistration}>
            {isRegistration ? 'Уже есть аккаунт?
Войти' : 'Создать аккаунт'}
    </Form.Item>

```

```

        </Button>
      </Form.Item>
    </Form>
  </div>
);
};

```

```
export default LoginForm;
```

Листинг Б.9 – Файл «HTTPWrapper.js»

```

class HTTPWrapper {
  constructor(baseUrl) {
    this.baseUrl = baseUrl;
  }

  async get(endpoint, params) {
    const url = new URL(`${this.baseUrl}${endpoint}`);
    if (params) {
      Object.entries(params).forEach(([key, value]) => {
        url.searchParams.append(key, value);
      });
    }
    const response = await fetch(url);
    const data = {}
    if (response.status === 200) {
      data.json = await response.json();
    }
    data.status = response.status;
    return data;
  }

  async post(endpoint, body) {
    const headers = new Headers();
    headers.set("Content-Type", "application/json");
    const response = await fetch(`${this.baseUrl}${endpoint}`,
{
    method: 'POST',
    headers: headers,
    body: JSON.stringify(body)
  });
    const data = {}
    if (response.status === 200) {
      data.json = await response.json();
    }
    data.status = response.status;
    return data;
  }

  async put(endpoint, body) {
    const headers = new Headers();
    headers.set("Content-Type", "application/json");

```

```

        const response = await fetch(`${this.baseUrl}${endpoint}`,
{
            method: 'PUT',
            headers: headers,
            body: JSON.stringify(body)
        });
        const data = {}
        if (response.status === 200) {
            data.json = await response.json();
        }
        data.status = response.status;
        return data;
    }

    async delete(endpoint) {
        const response = await fetch(`${this.baseUrl}${endpoint}`,
{
            method: 'DELETE'
        });
        const data = {}
        if (response.status === 200) {
            data.json = await response.json();
        }
        data.status = response.status;
        return data;
    }

    async postImage(endpoint, file) {
        const formData = new FormData();
        formData.append('file', file);
        const headers = new Headers();
        headers.set("data", file.name);
        const response = await fetch(`${this.baseUrl}${endpoint}`,
{
            method: 'POST',
            body: formData,
            headers: headers
        });
        const data = {}
        if (response.status === 200) {
            data.json = await response.json();
        }
        data.status = response.status;
        return data;
    }

    async getImage(endpoint) {
        const response = await fetch(`${this.baseUrl}${endpoint}`,
{
            method: 'GET',
            headers: {
                'Content-Type': 'image/jpeg'
            }
        }
    }

```

```

    });
    const blob = await response.blob();
    return URL.createObjectURL(blob);
  }
}

```

```
export default HTTPWrapper;
```

Листинг Б.10 – Файл «paths.js»

```

export const paths = {
  local : "http://172.17.0.1:22222/",
  home: "/home",
  lk: "/account",
  cart : "home/cart",
  about : "about",
  login : "login",
  cakes : "cakes",
  cupcakes : "cupcakes",

  requests: {
    file: {
      getImage: "file/"
    },
    desserts: {
      getById: "dessert/",
      getAllCakes: "dessert/cakes",
      getAllCupcakes: "dessert/cupcakes"
    },
    cart: {
      addToCart: "cart/",
      update: "cart",
      getCartBySession: "cart/"
    },
    order: {
      addOrder: "order"
    }
  },
}

```

ПРИЛОЖЕНИЕ В
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(обязательное)

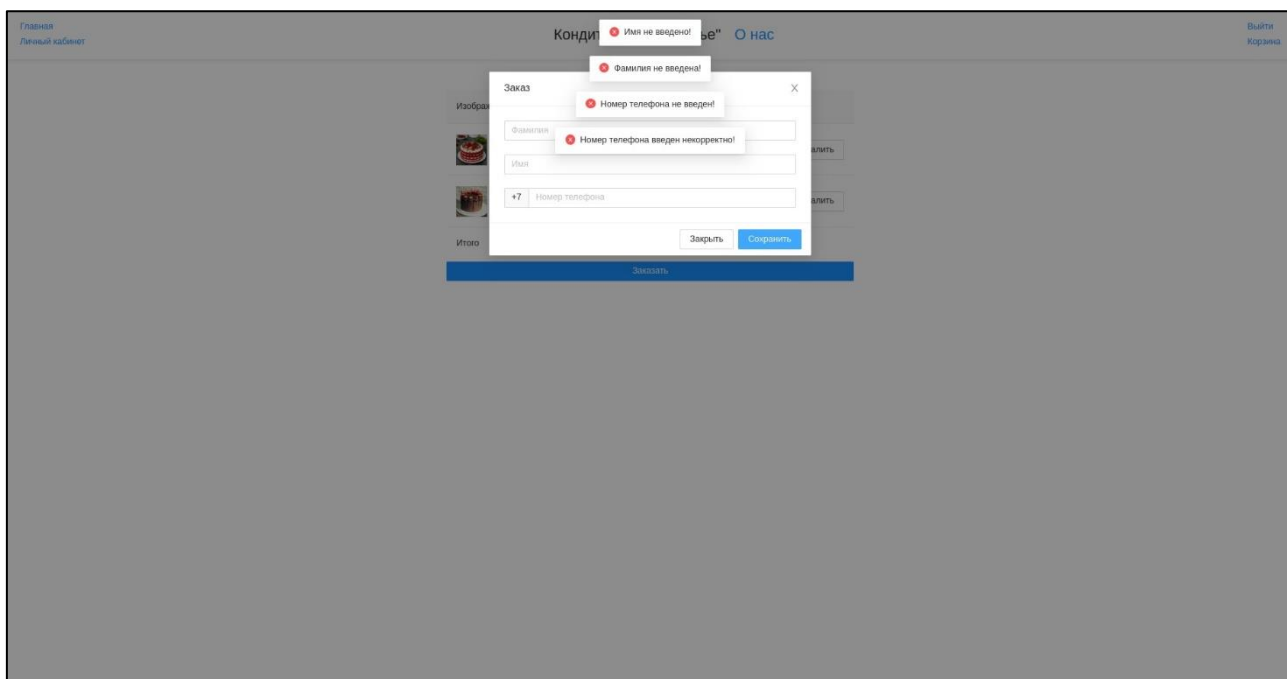


Рисунок В.1 – Результат теста №1

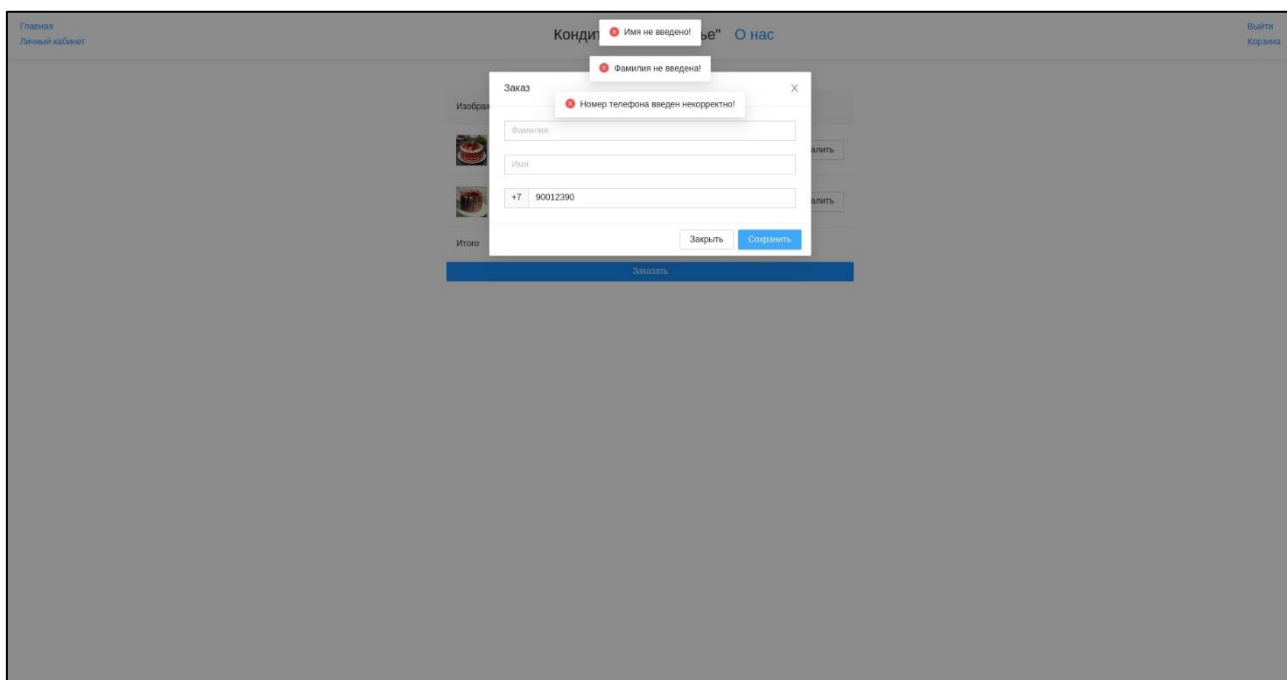


Рисунок В.2 – Результат теста №2

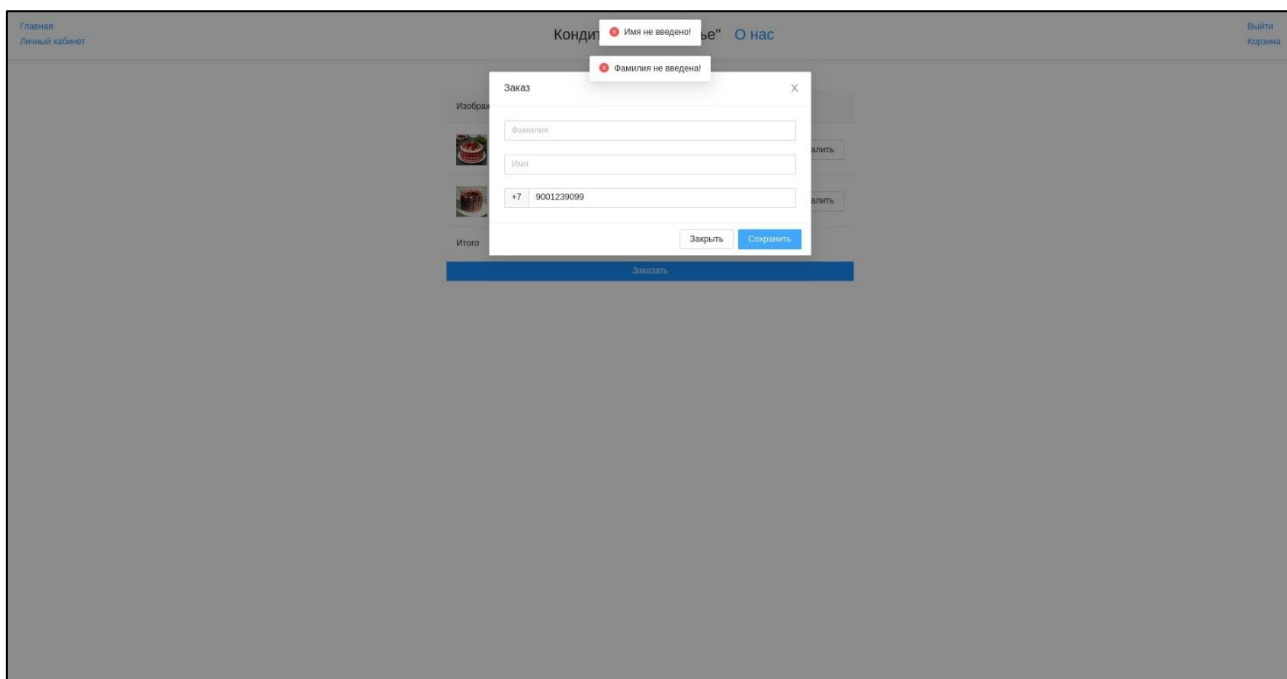


Рисунок В.3 – Результат теста №3

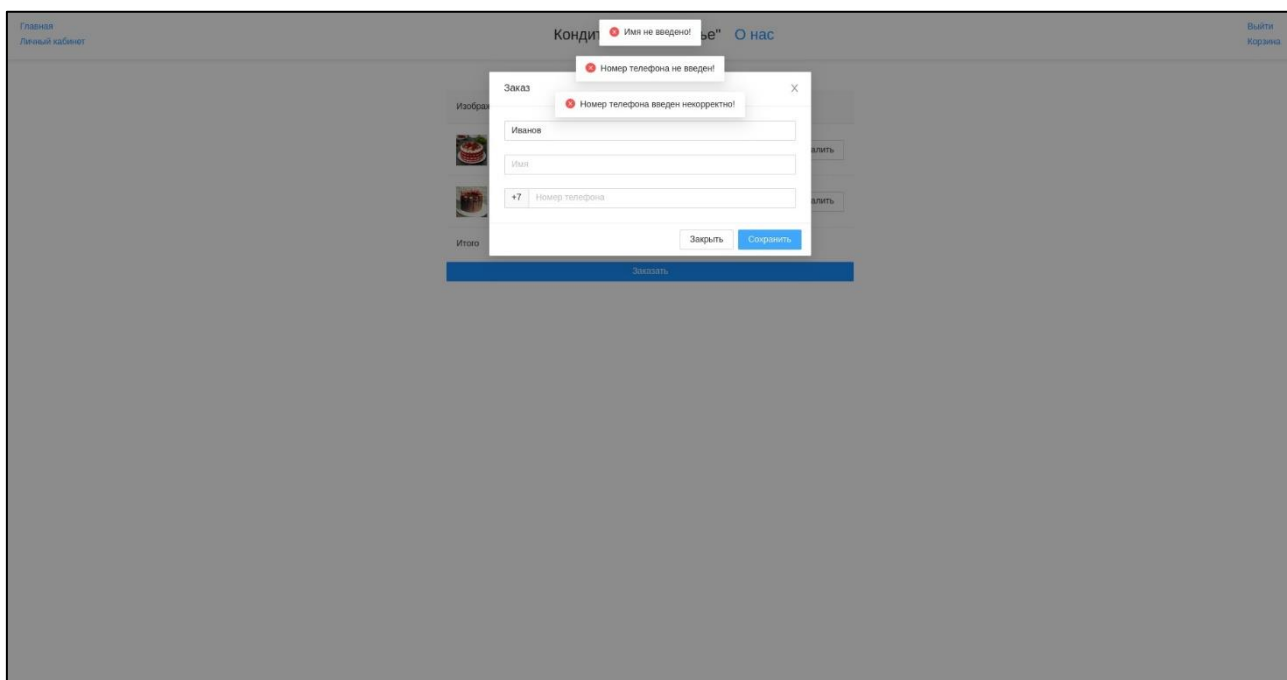


Рисунок В.4 – Результат теста №4

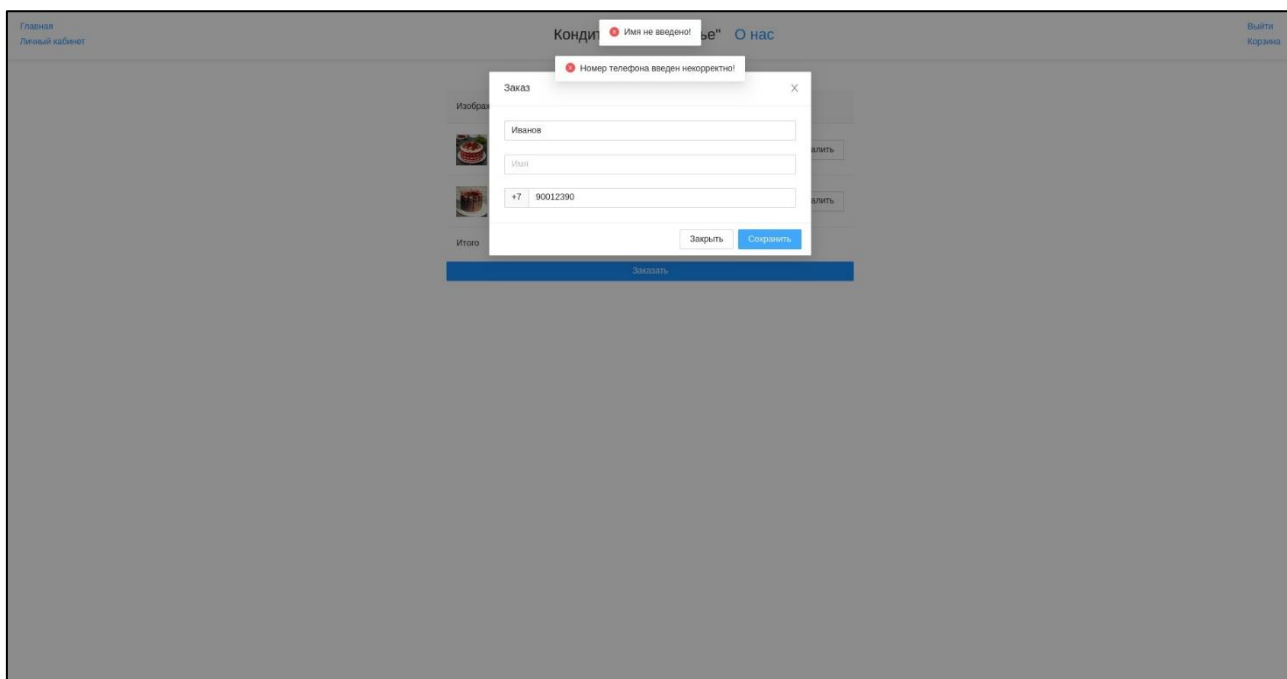


Рисунок В.5 – Результат теста №5

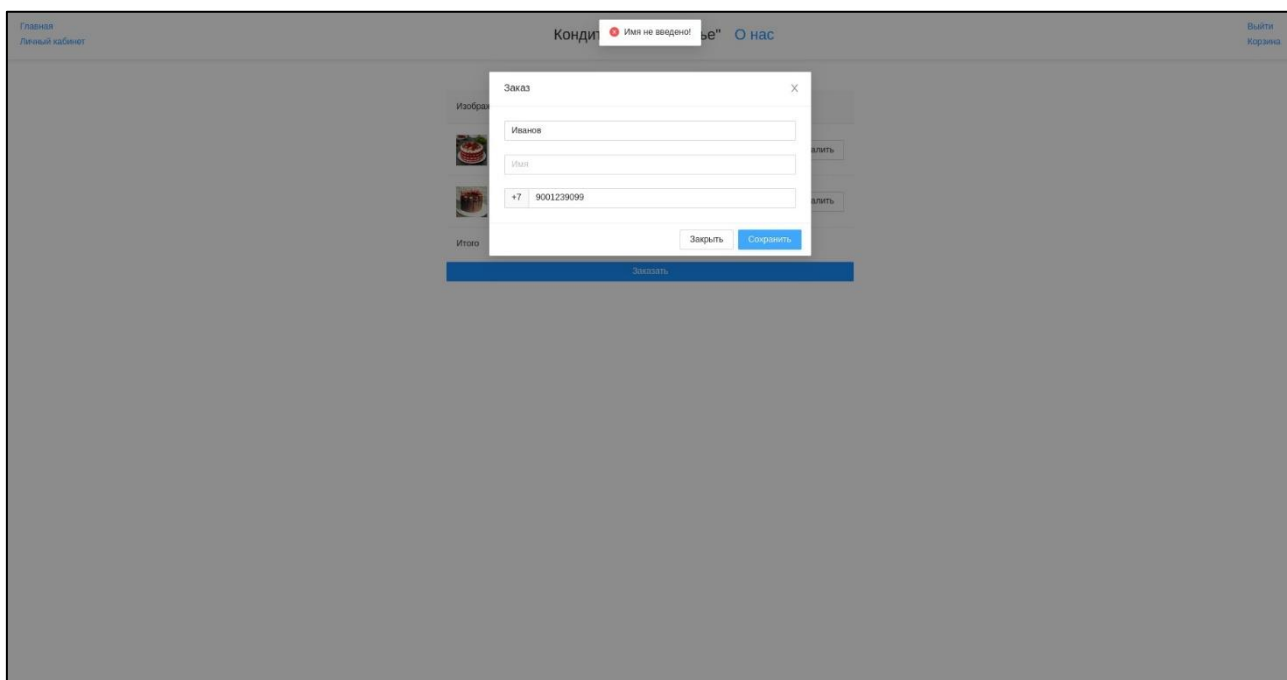


Рисунок В.6 – Результат теста №6

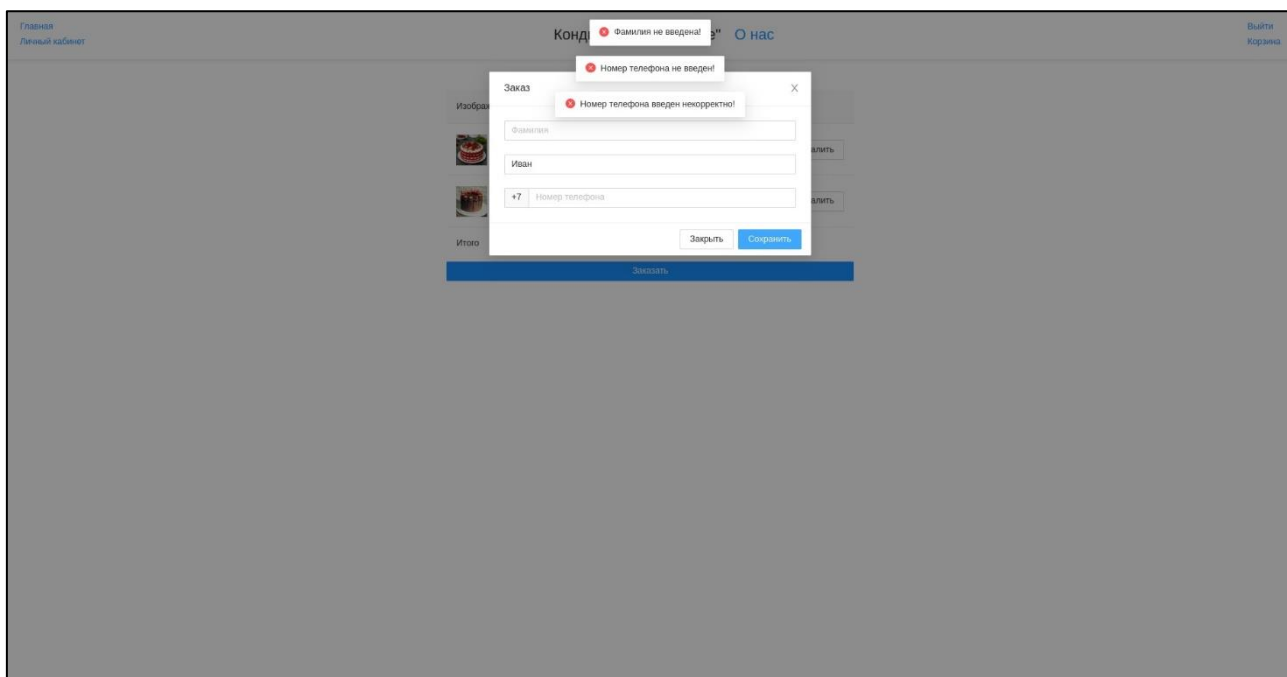


Рисунок В.7 – Результат теста №7

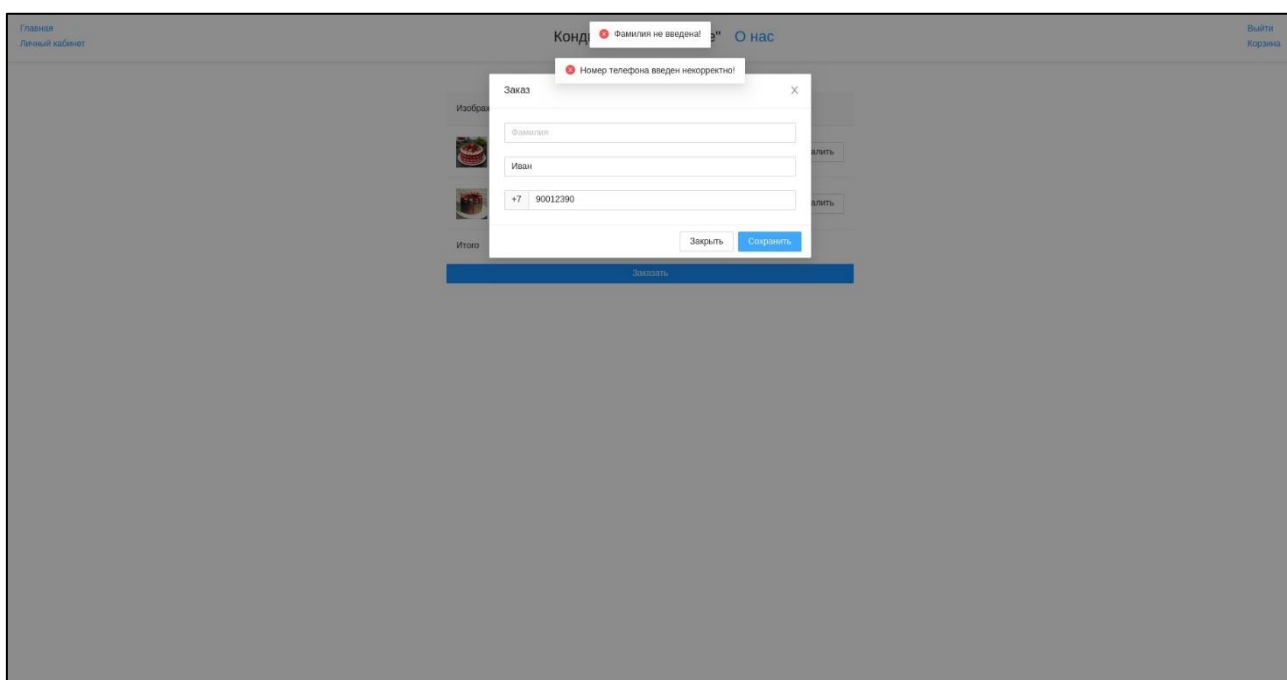


Рисунок В.8 – Результат теста №8

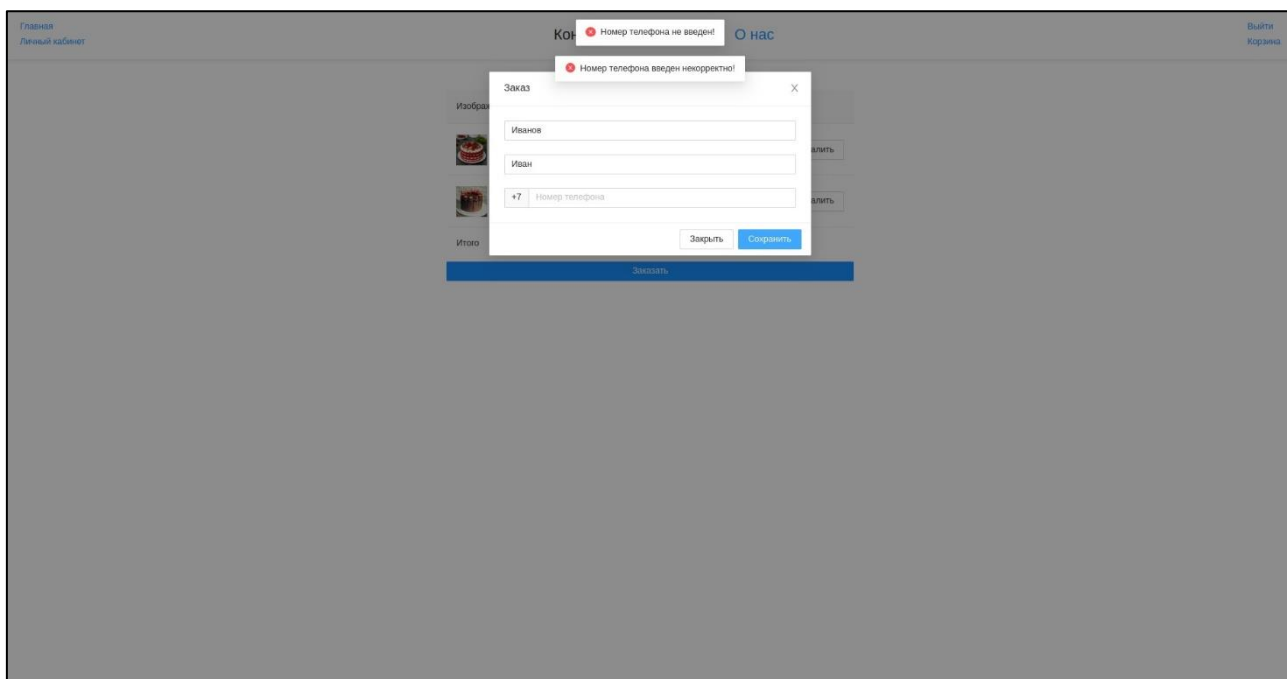


Рисунок В.9 – Результат теста №9

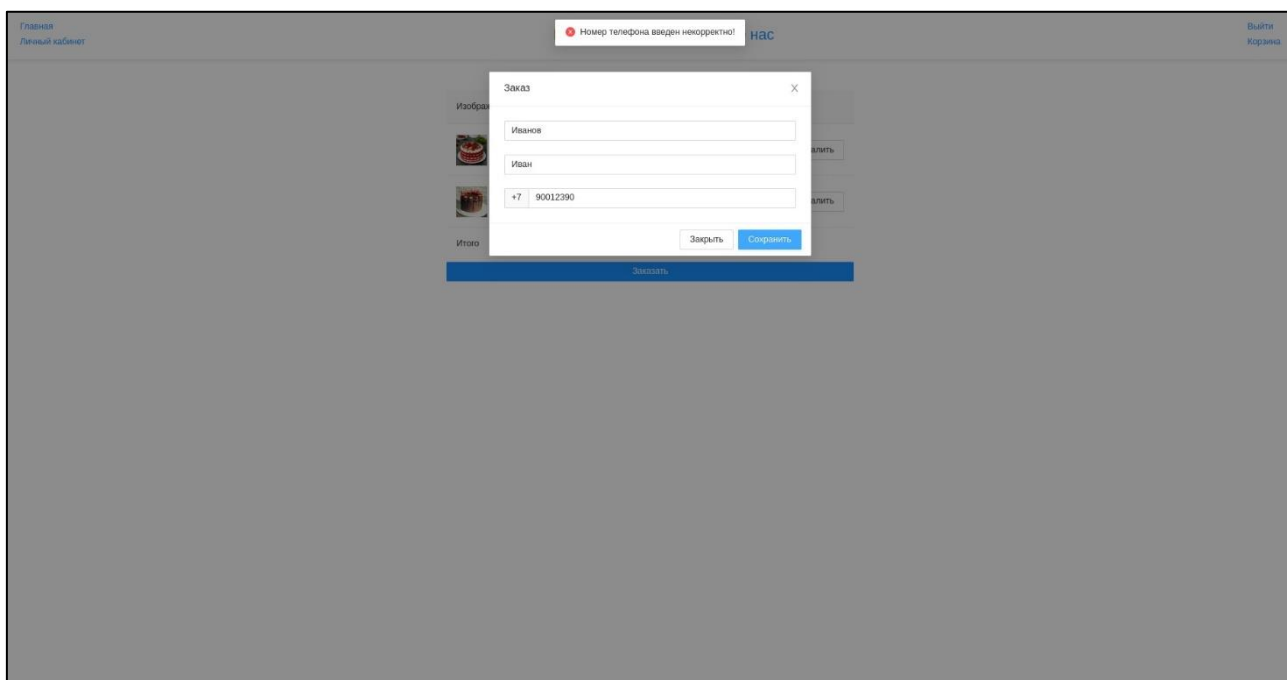


Рисунок В.10 – Результат теста №10

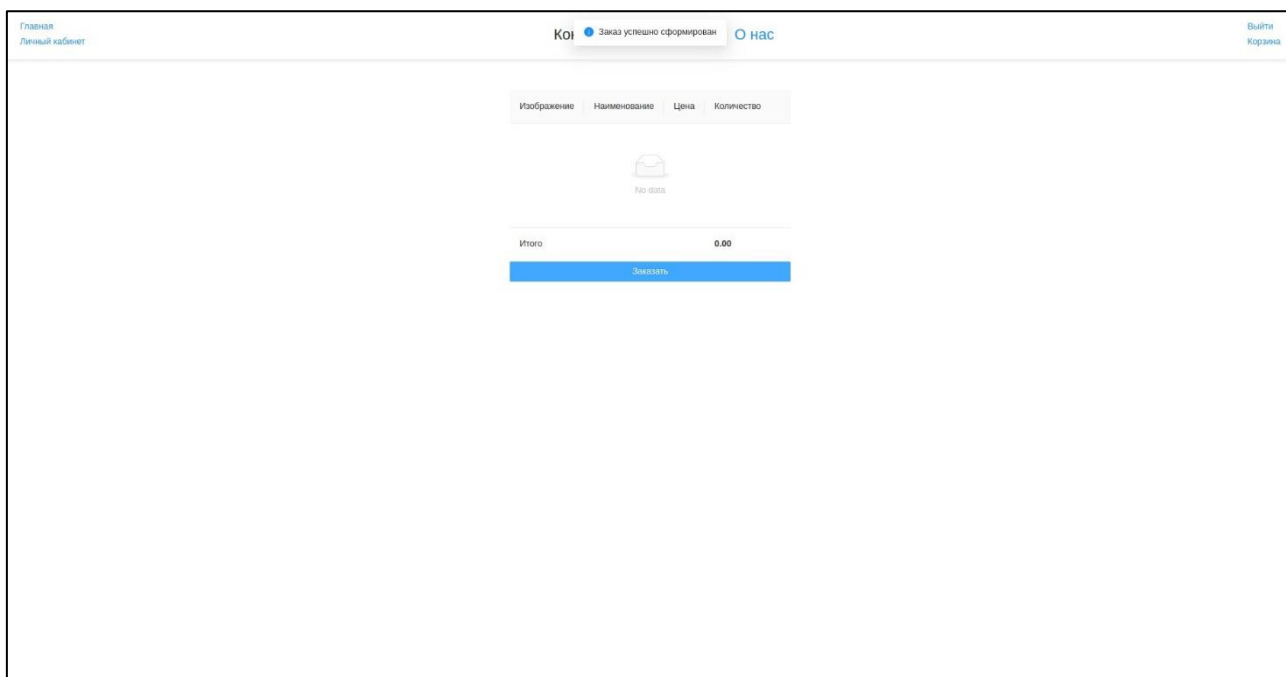


Рисунок В.11 – Результат теста №11