

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программная реализация интернет-магазина
пиццы с доставкой**

Студент

(подпись, дата) Малышев А.С.
(ФИО полностью)

Руководитель

(подпись, дата) Кольчугина Е.А.
(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, 2024

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

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«__» _____ 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Андрей Сергеевич Малышев гр. 20ВП1 факультета ФВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программная реализация интернет-магазина пиццы с доставкой

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

3. Руководитель работы к.т.н., Кольчугина Е.А.

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

1) Назначение: программа предназначена для заказа пиццы в интернет-магазине

2) Средства разработки: язык программирования JavaScript, интегрированная среда разработки (IDE) IntelliJ IDEA

3) Основные функции программы:

а) регистрация/авторизация;

б) возможность добавлять пиццы;

в) возможность просматривать каталог пицц;

г) возможность просматривать детальную информацию о пицце;

д) возможность добавлять пиццы в корзину;

5. Перечень подлежащих разработке вопросов

Анализ предметной области и постановка задачи

Анализ требований на разработку

Проектирование приложения

Реализация приложения

Тестирование и контроль качества приложения

Планирование разработки и оценка бюджета приложения

Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	5%	01.04.2024 08.04.2024	
2	Анализ требований на разработку	5%	08.04.2024 15.04.2024	
3	Проектирование приложения	25%	15.04.2024 30.04.2024	
4	Реализация приложения	35%	30.04.2024 21.05.2024	
5	Тестирование и контроль качества приложения	10%	21.05.2024 28.05.2024	
6	Функционально-стоимостный анализ	10%	28.05.2024 05.06.2024	
7	Оформление пояснительной записки	10%	05.06.2024 13.06.2024	

Дата выдачи задания "___" _____ 20____

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 58 листов, 18 таблиц, 30 рисунков, 10 использованных источников и 2 приложения.

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ИНТЕРНЕТ-МАГАЗИН ПИЦЦЫ, ООП, МИКРОСЕРВИС, INTELLIJ IDEA, UML

Объект разработки – Программное обеспечение интернет-магазина пиццы.

Цель работы – провести анализ предметной области, поставить задачу на разработку, сравнить с конкурирующими программами, выбрать инструменты, спроектировать UML-диаграммы и пользовательский интерфейс, разработать ПО, протестировать его и проанализировать код с помощью метрик, спланировать бюджет разработки.

Средства разработки: язык программирования JavaScript, интегрированная среда разработки IntelliJ IDEA.

Полученные результаты: в ходе выполнения выпускной квалификационной работы была спроектирована программа, которая позволяет организовать интернет-магазин по продаже пиццы.

					ПГУ 09.03.04 – 08ВР191.13 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	«Программная реализация интернет-магазина пиццы с доставкой» Пояснительная записка	Лит.	Лист	Листов
Разраб.		Малышев А.С.						
Провер.		Кольчугина Е.А.					4	71
						Группа 20ВП1		
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку	8
1.1 Постановка задачи	8
1.2 Анализ информационных систем аналогичного назначения.....	9
1.3 Додо Пицца	10
1.4 Papa John's	11
1.5 Domino's Pizza.....	12
1.6 Обоснование для разработки интернет-магазина пиццы	13
2 Проектирование программы	14
2.1 Анализ функциональных требований к системе.....	16
2.2 Проектирование системы.....	16
2.2.1 Анализ прецедентов	16
2.2.2 Проектирование модели предметной области	19
2.2.3 Проектирование структуры БД.....	20
2.2.4 Проектирование обобщенной архитектуры системы.....	21
2.3 Разработка серверного программного обеспечения	22
2.4 Разработка клиентского программного приложения.....	25
2.4.1 Требования к пользовательскому интерфейсу	25
2.4.2 Описание архитектуры пользовательского интерфейса	26
3 Тестирование программного обеспечения	37
3.1 Выбор методологии тестирования.....	37
3.2 Тестирование черного ящика модуля «Регистрация пользователей».....	38
3.2.1 Классы эквивалентности	38
3.2.2 Функциональная диаграмма и таблица решений.....	39
3.3 Модульное тестирование модуля «Регистрация пользователя»	41
3.4 Функциональное тестирование клиентского ПО	43
3.5 Метрики кода	45
3.5.1 Метрика SLOC	45
3.5.2 Метрики Холстеда.....	46
3.5.3 Метрики устойчивости и абстрактности компонент.....	47

3.5.4	Комплексный показатель качества кода.....	48
4	Планирование и оценка бюджета проекта	48
	Заключение	52
	Список использованных источников	53
	Приложение А (рекомендуемое) Глоссарий	55
	Приложение Б (рекомендуемое) Код программы.....	57

Введение

Современный мир – это мир удобства и быстроты. Одной из сфер, где эти качества важны, является сервисная деятельность по доставке еды. Данная услуга «интернет-магазина пиццы» направлена на экономию личного времени, на удовлетворение потребности общества в еде и в отдыхе.

Пицца-национальное блюдо Италии. Итальянская кухня очень популярна во всем мире. Ее готовят дома, кафе, ресторанах и фастфуде. Интернет-магазины пиццы популярны благодаря возможности быстро и легко заказать любимое блюдо домой, в офис, на мероприятие. В условиях постоянной занятости и быстрого ритма жизни, наличие удобного сервиса доставки пиццы позволяет людям экономить время и получать удовольствие от вкусной и горячей пищи и не нужно выходить из дома. Открытие данной услуги будет пользоваться большим спросом и значит будет приносить прибыль.

Доставку пиццы, при растущей популярности, можно улучшить с помощью современных технологий. Один из важных моментов – это автоматизация процесса заказа и доставки. Автоматизированные системы позволяют минимизировать ошибки при оформлении заказов, оптимизировать маршруты доставки, повышать уровень обслуживания клиентов.

Данный проект направлен на создание удобного и функционального интернет-магазина пицц. Основной идеей приложения является предоставление покупателю самому создать заказ на свой вкус, выбрать начинку, исходя из этого будет высчитываться стоимость заказа. И заказывать пиццу станет не сложным процессом.

1 Анализ предметной области и постановка задачи на разработку

1.1 Постановка задачи

Целью данной работы является разработка и создание интернет-магазина пиццы.

Для достижения поставленной задачи были выделены следующие цели:

- разработка базы данных для хранения информации о пиццах, клиентах и сопутствующей информации;
- разработка серверного программного обеспечения, обеспечивающего получение, добавление и изменение информации о пиццах, сотрудниках и их задачах в базе данных;
- разработка клиентского программного обеспечения для комфортной работы пользователей.

Для достижения вышеупомянутых целей были выбраны следующие инструменты:

ostgreSQL для создания базы данных;

- JavaScript и Framework React для создания клиентского программного обеспечения, Nest JS для реализации сервера;

ntelliJ IDEA – среда разработки серверного программного обеспечения;

- Node JS – программная платформа и Visual Studio Code – среда разработки, используемые для реализации клиентского программного обеспечения.

Выполнение данной работы предполагает разработку WEB-приложения. Приложение состоит из двух основных частей: серверная и клиентская часть. Так как серверная часть написана на языке JavaScript, ее можно запустить на любой системе: Windows, Linux, MacOS. Клиентская часть поддерживается в любом современном браузере, соответственно, приложение является универсальным и может работать с любыми современными операционными системами.

Теоретической основой разработки в данной работе является руководство принципами «SOLID» и ООП.

При создании программного продукта использование принципов «SOLID» способствует созданию такой программы, которую будет легко поддерживать и расширять в течении долгого времени [1].

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования

Разрабатываемое приложение имеет архитектуру, основанную на трёхслойной модели, включающей в себя контроллеры, сервисы и репозитории. Контроллеры отвечают за взаимодействие между клиентским и серверным ПО, обрабатывая HTTP-запросы от клиента. Они принимают данные и возвращают результат. Сервисный слой содержит бизнес-логику приложения. Контроллеры вызывают методы этого слоя для преобразования полученных данных в результат. Слой репозитория обеспечивает связь между серверной частью приложения и базой данных. Соответственно, те данные, которые получены сервисным слоем, сохраняются в БД или по ним происходит выборка именно по средствам слоя репозитория. Далее, получив результат обращения к БД, слой репозитория возвращает его сервисному, тот в зависимости от конкретной логики, преобразует его и возвращает контроллерному, который отправляет его уже как ответ на http-запрос туда, откуда он пришел.

1.2 Анализ информационных систем аналогичного назначения

Чтобы понять, стоит ли открывать новую онлайн пиццерию и какие функции могут сделать её конкурентоспособной, нужно изучить успешные примеры таких проектов. В качестве лидеров рынка можно рассмотреть такие компании как «Додо Пицца», Papa John's и Domino's Pizza. При анализе успешных интернет-магазинов пиццы были выделены следующие ключевые аспекты:

- удобный и понятный интерфейс для клиентов;

- функционал, связанный с основной деятельностью пиццерии;
- возможность анализировать данные о работе пользователей системы;
- простота использования программного обеспечения.

1.3 Додо Пицца

Додо Пицца – одна из самых инновационных и популярных сетей пиццерий, которая активно использует технологии для оптимизации процессов и повышения качества обслуживания. Цель создания системы Додо Пиццы – обеспечение эффективной работы пиццерий и предоставление клиентам удобного сервиса для заказа пиццы. ПО ориентировано на автоматизацию процессов и предоставляет следующий функционал:

- наглядное представление списка пицц;
- система создания и обновления заказов;
- ведение проекта от начала до конца;
- отслеживание ошибок и контроль за их своевременным исправлением;
- ведение соответствующей документации.

Перед рассмотрением плюсов и минусов данного приложения, следует отметить, что система Додо Пиццы разрабатывалась с нуля специально для нужд сети, что обеспечивает высокую степень адаптации и интеграции всех процессов.

К плюсам можно отнести, что у Додо Пиццы простой и интуитивно понятный интерфейс, есть возможность отслеживать заказы в реальном времени и в реальном времени просматривать процесс приготовления пиццы по видео камерам. А к минусам, можно отнести необходимость больших затрат на разработку и поддержку.

На рисунке 1 представлен интерфейс данного программного обеспечения.

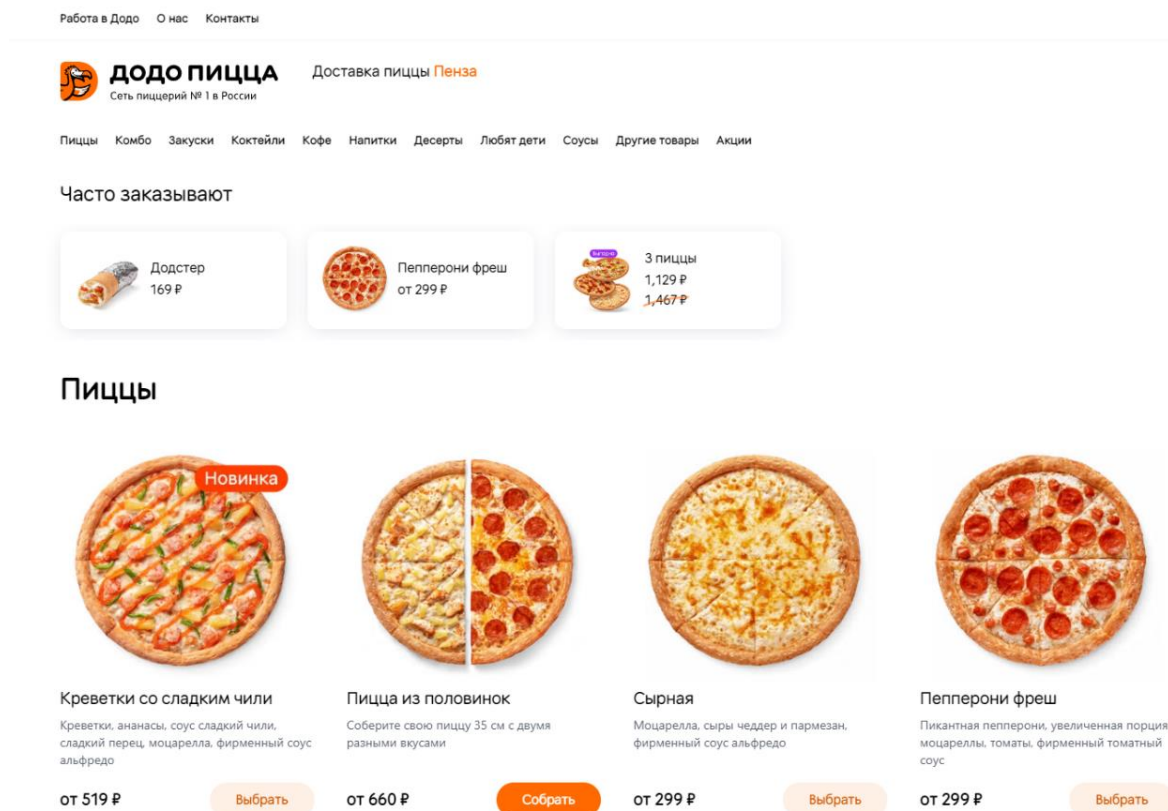


Рисунок 1 – Интерфейс Додо Пиццы

Анализ Додо Пиццы проведен успешно.

1.4 Papa John's

John's – одна из крупнейших международных сетей пиццерий, которая отвечает за качество ингредиентов и высокие стандарты обслуживания. Цель создания системы Papa John's – является обеспечение удобного и быстрый процесс заказа и доставки пиццы для клиентов по всему миру. Программное обеспечение Papa John's ориентировано на поддержку высокого уровня обслуживания и включает следующий функционал:

- понятный интерфейс;
- разделение ролей и задач между сотрудниками пиццерии;
- система бонусов.

На рисунке 2 представлен интерфейс ПО.

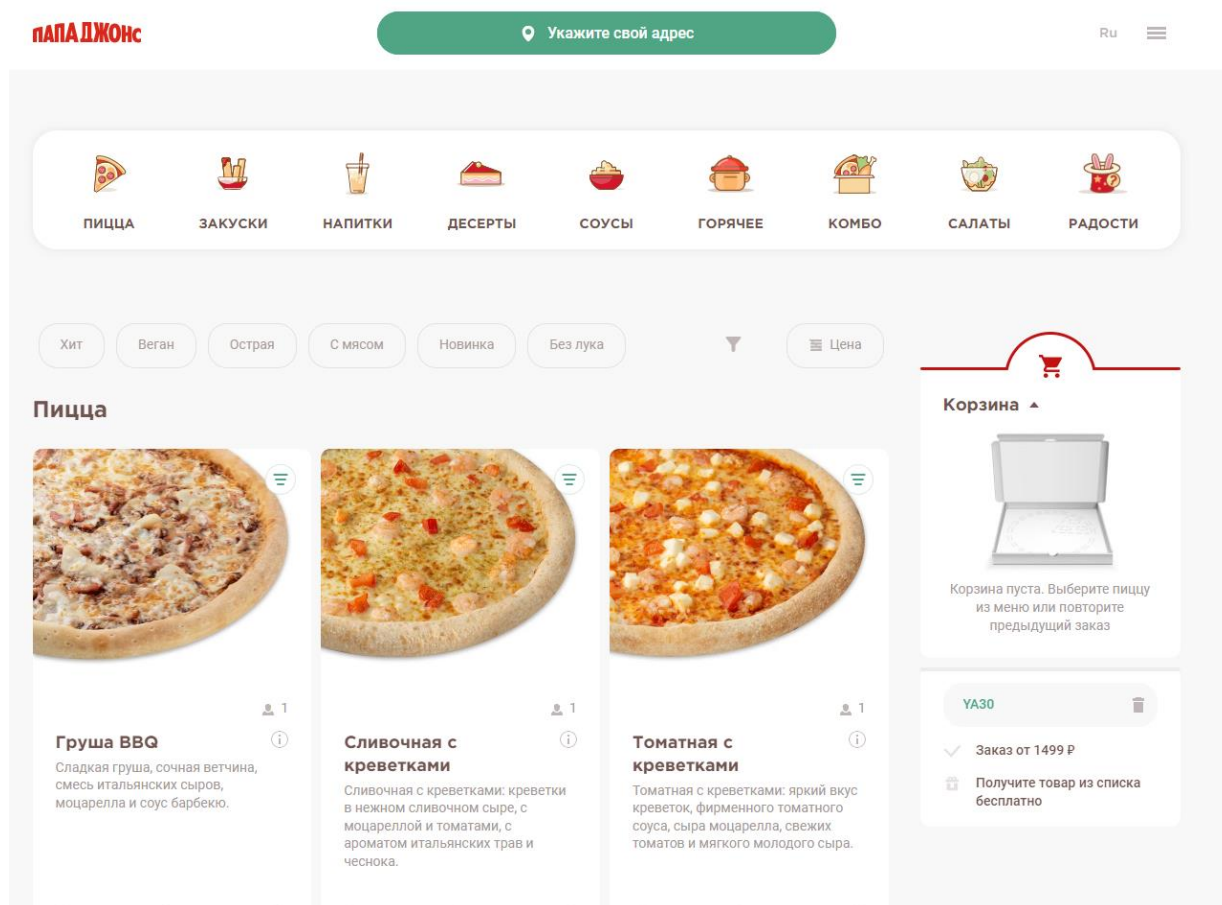


Рисунок 2 – Интерфейс Papa John's

Анализ Papa John's проведен успешно.

1.5 Domino's Pizza

Domino's Pizza – одна из ведущих международных сетей пиццерий, известная своим инновационным подходом к использованию технологий. Цель создания системы Domino's Pizza – обеспечение удобного и быстрого процесса заказа и доставки пиццы, а также автоматизация внутренних процессов. Программное обеспечение Domino's Pizza включает в себя аналогичный функционал, рассмотренный выше. Также можно отметить, что интерфейс удобный и интуитивно понятный.

На рисунке 3 представлен интерфейс приложения.

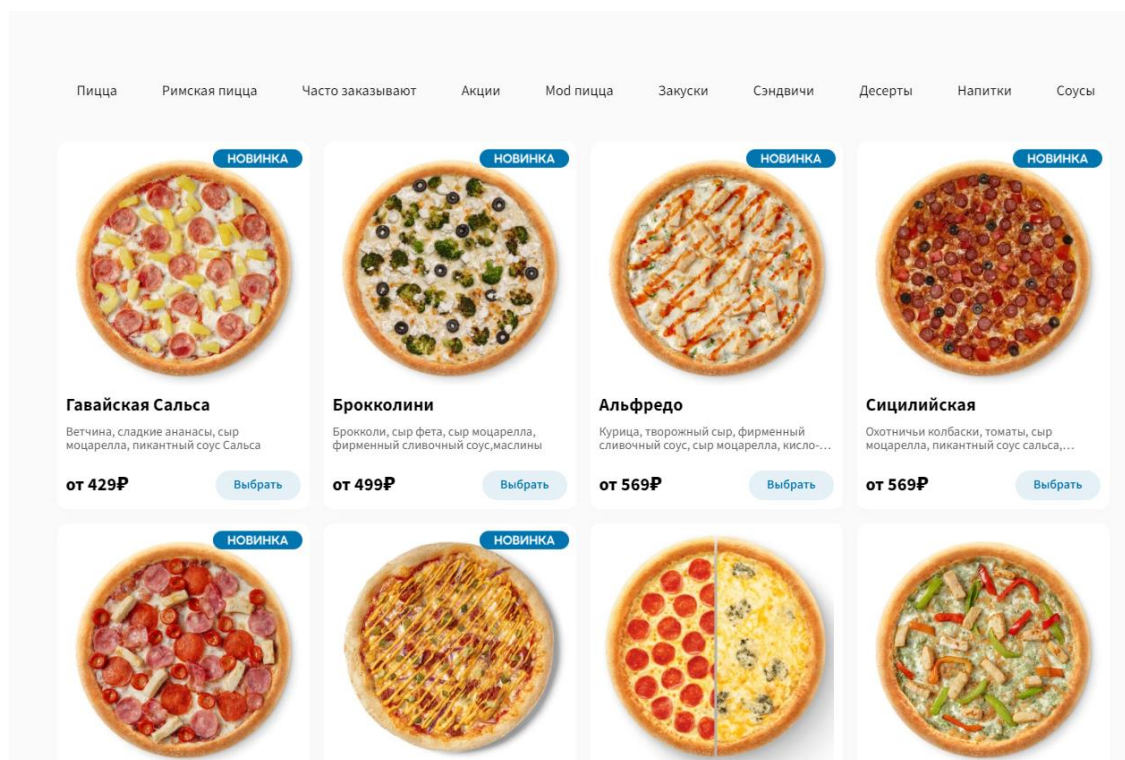


Рисунок 3 – Интерфейс Domino's Pizza

Анализ Domino's Pizza проведен успешно.

1.6 Обоснование для разработки интернет-магазина пиццы

Первым шагом обоснования нужно предоставить сравнительный анализ конкурирующих приложений, представленный в таблице 1.

Таблица 1 – Сравнительный анализ аналогичных приложений

Название пункта	Додо пицца	Papa John's	Domino's Pizza
Простой и интуитивно понятный интерфейс	Да	Да	Да
Разделение по ролям	Да	Да	Да
Бонусная программа	Да	Да	Да
Представление отчетности	Да	Да	Да
Доступность приложения	Нет	Да	Да

Исходя из таблицы 1 можно сделать следующие выводы:

- все приложения просты и удобны в использовании;
- у каждой пиццерии есть бонусные программы;

- все приложения доступны.

Исходя из всего вышесказанного, можно сделать вывод, что интернет-магазин должен быть простым и удобным для пользователей. Дизайн приложения должен быть качественным и привлекательным.

Основная особенность этого приложения – это возможность для каждого пользователя добавлять свои пиццы и получать процент от заказов других пользователей. Приложение будет специализироваться на доставке пиццы, что ограничит его применение другими сферами деятельности, но сделает его более удобным и наглядным для использования по назначению. Ведь обычно инструменты, которые пытаются решить множество задач, либо сложны в использовании, либо не справляются с этими задачами хорошо. Явным плюсом разрабатываемого продукта, является, сужение сферы использования. Применяя данную архитектуру и принципы построения ПО, можно с минимальными усилиями перевести данное приложение из одной предметной области в другую.

2 Проектирование программы

При разработке приложения было принято решение разделить его на три основные части:

- клиентская часть приложения, т. е. интерфейс;
- backend – серверная часть приложения;
- база данных (далее – БД).

Для клиентской части было принято решение использовать язык JavaScript и фреймворк React. Для серверной части – язык JavaScript и фреймворк Nest Js. СУБД – PostgreSQL.

В связи с разделением клиентской и серверной частей, для их взаимодействия используется протокол HTTP. Клиентская часть отправляет запросы к серверу и получает ответы – данные из базы данных, к которой обращается backend. Такая архитектура соответствует классическому WEB-приложению. Для разработки серверной части было решено использовать архитектуру REST. Она накладывает определённые правила на написание

эндпоинтов – методов контроллеров, которые обрабатывают HTTP-запросы. Эти технологии хорошо подходят для создания веб-приложений, поэтому они были выбраны. Кроме того, для работы с базой данных была выбрана концепция ORM (объектно-реляционное отображение). ORM – это своего рода посредник между базой данных и объектно-ориентированным программированием. Он позволяет связать данные с объектами классов-сущностей, которые повторяют структуру таблиц базы данных. Проще говоря, класс-сущность должен полностью соответствовать таблице базы данных, где каждая строка таблицы становится объектом соответствующего класса-сущности. PostgreSQL является одной из самых популярных систем управления базами данных (СУБД). Она предназначена для хранения и получения разнообразной информации. PostgreSQL также является реляционной СУБД, что позволяет создавать таблицы, связывать их внешними ключами и выполнять другие операции. Полная схема представлена далее.

Для разработки данного приложения было принято решение использовать IDE, которая полностью удовлетворяет требованиям современной IDE:

- редактор кода;
- компилятор;
- отладчик;
- средство автоподстановки и генерации кода;
- удобный интерфейс.

Если вернуться к архитектуре приложения в целом, которая описывалась в пункте 1.1, то ее можно дополнить, основываясь на предыдущих суждениях:

- клиентская часть приложения отправляет http-запрос в эндпоинт контроллера;
- эндпоинт контроллера, не преобразуя полученные данные, отправляет их в соответствующий метод сервисного слоя;
- в методах сервисного слоя реализована вся бизнес-логика приложения;

- если по логике того или иного бизнес-процесса необходимо совершить запрос к БД, то вызывается метод слоя репозитория, который формирует и отправляет SQL запрос в СУБД PostgreSQL, возвращая один или список классов-сущностей, в зависимости от самого запроса;

- далее, цепочка вызовов переворачивается, т. е. репозитории возвращают результат сервисам, сервисы – контроллерам, а контроллеры – клиентской части приложения.

Данная архитектура – это классическая архитектура WEB-приложения. При такой архитектуре не важно, в каком виде представлена клиентская часть – будь то WEB-страницы, мобильное приложение, desktop-приложение или другое серверное приложение, которое интегрируется с данным. В этом заключается основной плюс данной архитектуры, из-за чего она и выбрана.

2.1 Анализ функциональных требований к системе

Необходимо разработать интернет магазин для пиццерии. Данная система должна включать в себя следующий функционал:

- просмотр всех пицц и подробной информации о каждой;
- возможность добавления новой пиццы;
- возможность зарегистрироваться в системе;
- возможность добавить пиццу в корзину.

2.2 Проектирование системы

2.2.1 Анализ прецедентов

На основе функциональных требований была разработана диаграмма вариантов использования, показанная на рисунке 4.

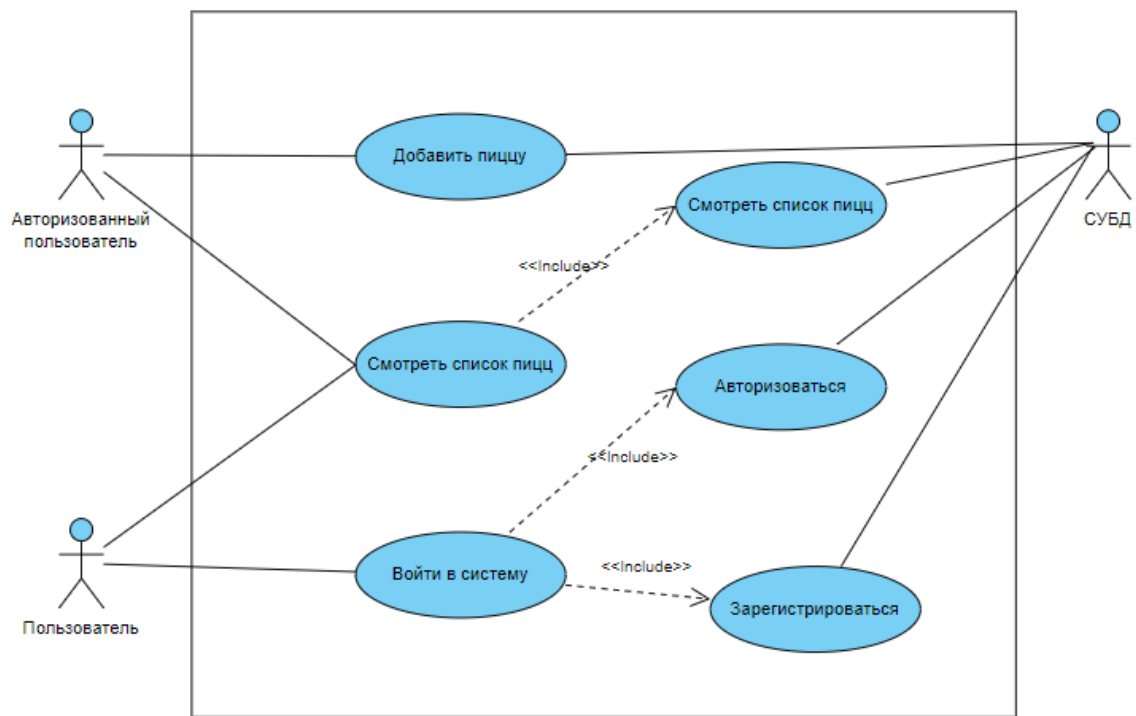


Рисунок 4 – Диаграмма вариантов использования

На рисунке 5 приведена спецификация варианта использования «Регистрация работников».

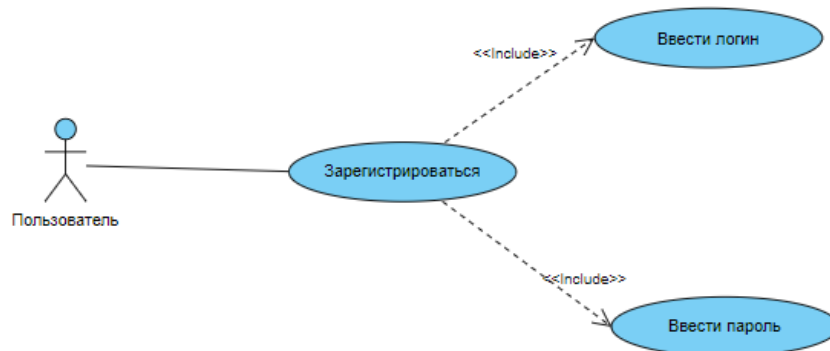


Рисунок 5 – Спецификация прецедента «Регистрация пользователей»

На рисунке 6 приведена спецификация варианта использования «Добавление пиццы».

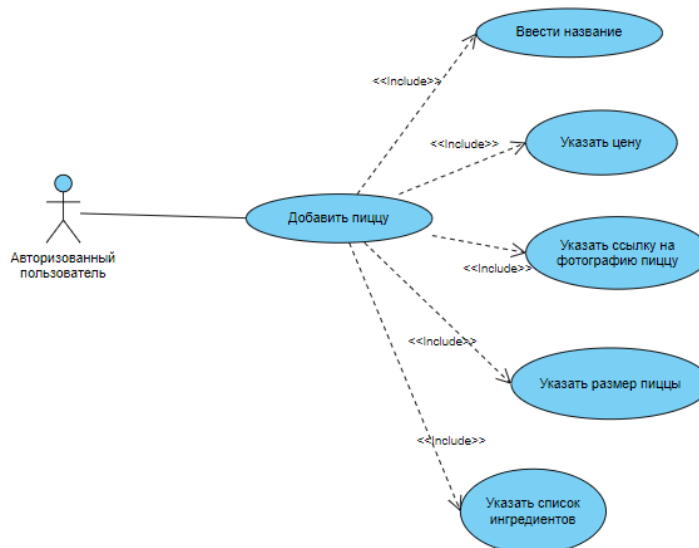


Рисунок 6 – Спецификация прецедента «Добавление пиццы»

Также были составлены диаграммы деятельности, показанные на **рисунках 7 и 8**.

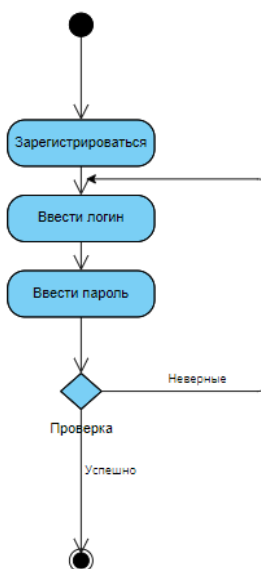


Рисунок 7 – Диаграмма деятельности «Создание аккаунта»

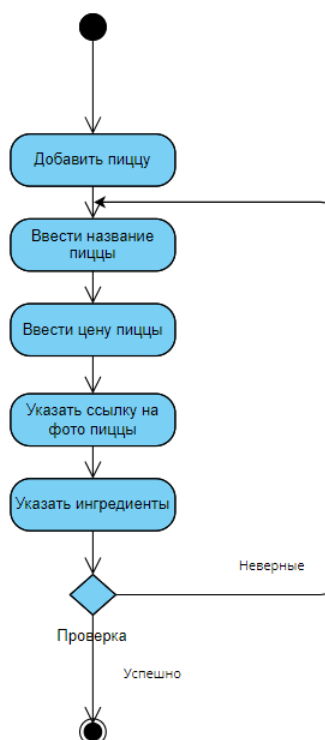


Рисунок 8 – Диаграмма деятельности «Добавление пиццы»

В ходе анализа функциональных требований были выделены и описаны основные варианты использования системы.

2.2.2 Проектирование модели предметной области

В рамках анализа предметной области были выделены следующие сущности предметной области «Интернет-магазин пицц», представленные на рисунке 9.

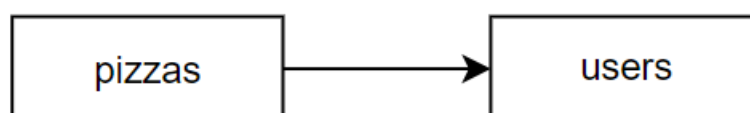


Рисунок 9 – Модель предметной области

Описание сущностей предметной области представлено в таблице 4.

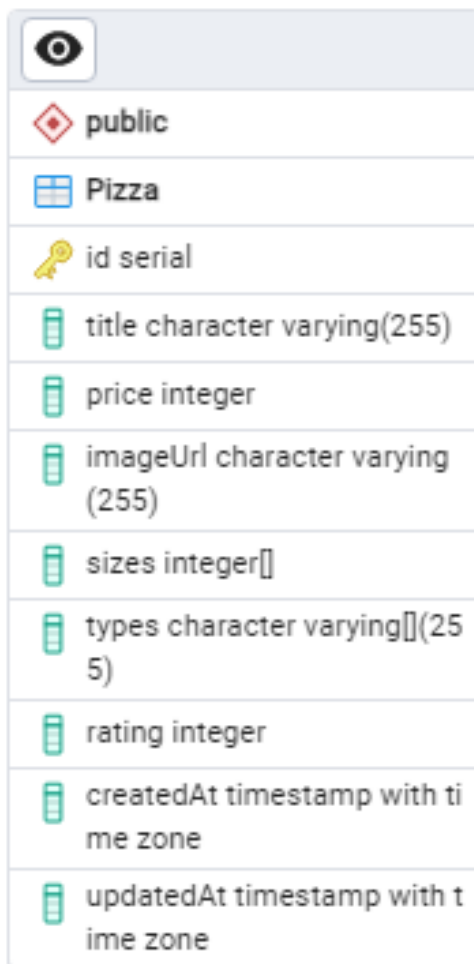
Таблица 4 – Описание сущностей

Название класса	Спецификация класса
User	сущность, описывающая пользователя;
Pizza	сущность, описывающая пиццу

Данная модель предметной области соответствует модели БД.

2.2.3 Проектирование структуры БД

На основе модели сущностей были разработаны схемы БД, представленные на рисунках 10 и 11.



public
Pizza
id serial
title character varying(255)
price integer
imageUrl character varying (255)
sizes integer[]
types character varying[] (255)
rating integer
createdAt timestamp with time zone
updatedAt timestamp with time zone

Рисунок 10 – Схема БД для хранения пицц

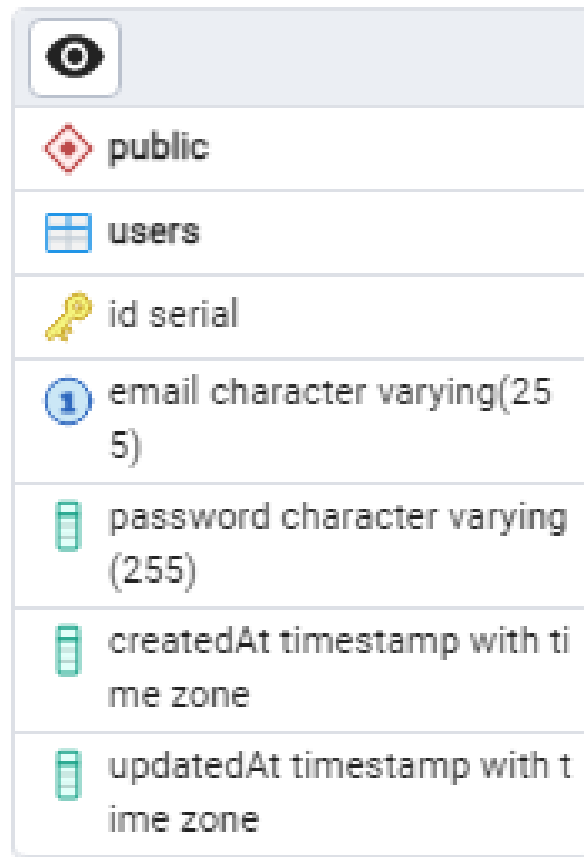


Рисунок 11 – Схема БД для хранения пользователей

Назначение таблиц, представленных на схеме представлено в таблице 5.

Таблица 5 – Описание таблиц БД

Название сущности	Идентификатор сущности	Назначение сущности
Пользователь	users	Содержит Сведения о пользователях
Пицца	pizza	Содержит сведения о пиццах

Подробное описание таблиц базы данных представлено в приложении А.

2.2.4 Проектирование обобщенной архитектуры системы

На основе архитектуры были выделены компоненты, показанные на рисунке 12.

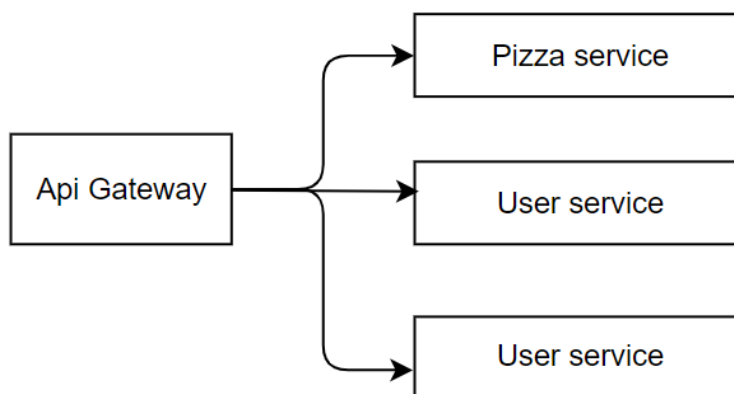


Рисунок 12 – Схема обобщенной архитектуры системы

Один из основных модулей архитектуры – база данных.

Основной элемент системы – это разрабатываемый сервер. Он состоит из трёх компонентов:

- компонент обработки данных, которые поступают от клиента;
- компонент управления данными, или слой подключения к БД;
- компонент генерации данных, который позволяет повторно создавать модели через определенные промежутки времени.

На схеме клиент обозначен как компонент, позволяющий взаимодействовать с сервером через пользовательский интерфейс. Клиент и сервер обмениваются данными по Rest API.

Таким образом, архитектура системы соответствует общим принципам разработки веб-приложений.

2.3 Разработка серверного программного обеспечения

Для создания серверной части приложения была выбрана микросервисная архитектура.

Микросервисный подход в разработке программного обеспечения предполагает разделение программы на небольшие независимые сервисы, каждый из которых отвечает за свою функцию. Этими сервисами управляют

отдельные команды разработчиков, а взаимодействие между ними происходит через чётко определённые API (программные интерфейсы приложения). Микросервисы постепенно вытесняют монолитную архитектуру, где всё приложение рассматривается как единое целое. Преимущества микросервисной архитектуры:

- гибкость масштабирования;
- простота и скорость обновлений;
- независимого развёртывания обновлений.

На рисунке 12 приведена диаграмма компонентов из которых состоит серверная часть приложения.

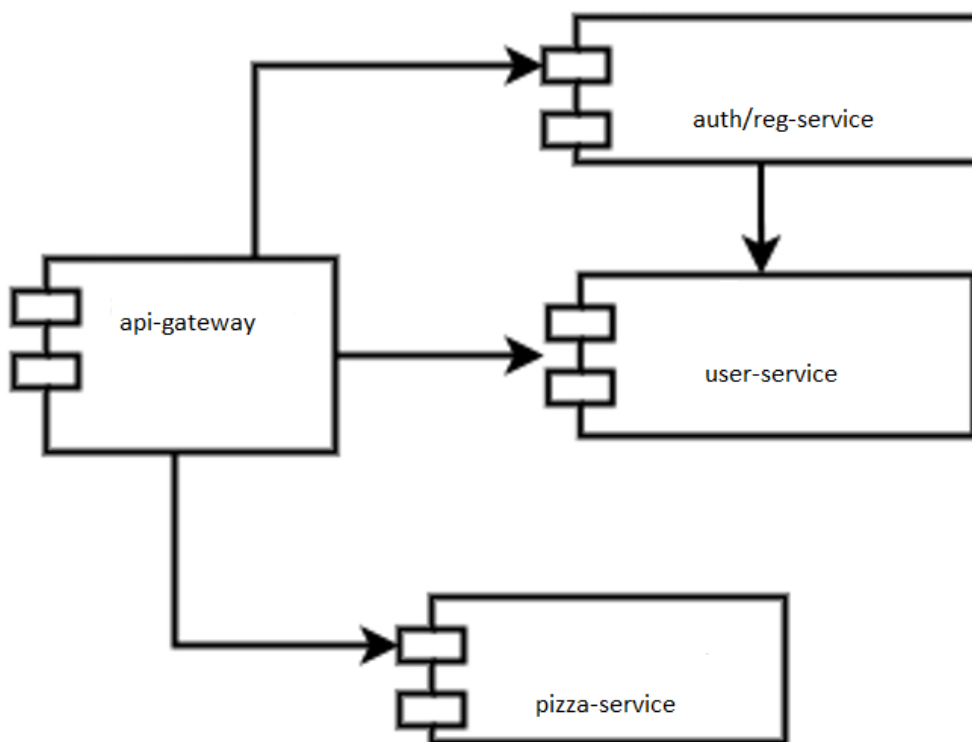


Рисунок 12 – Диаграмма компонентов серверной части приложения

На рисунке 13 представлена диаграмма компонентов клиентской части приложения.

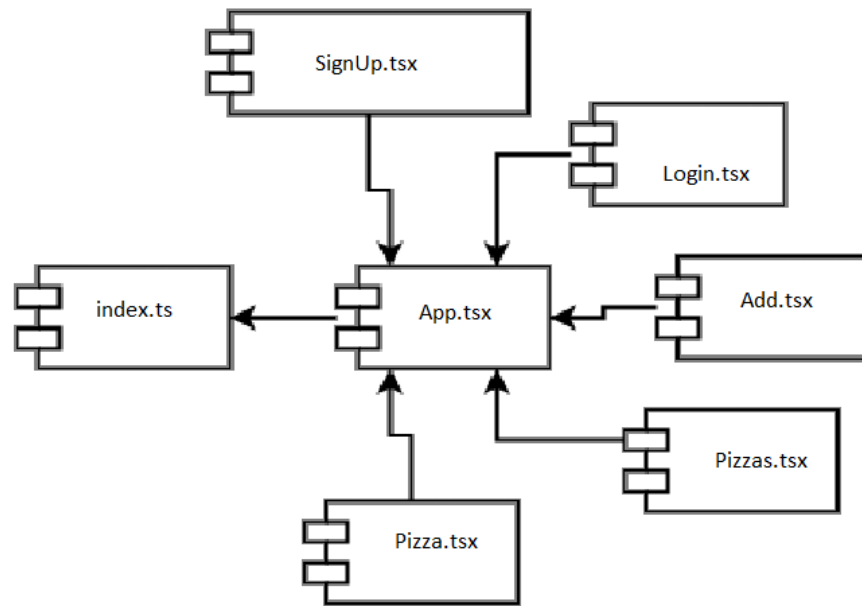


Рисунок 13 – Диаграмма компонентов клиентской части приложения

На рисунке 14 приведен пример компоновки классов для работы с информацией о пиццах.

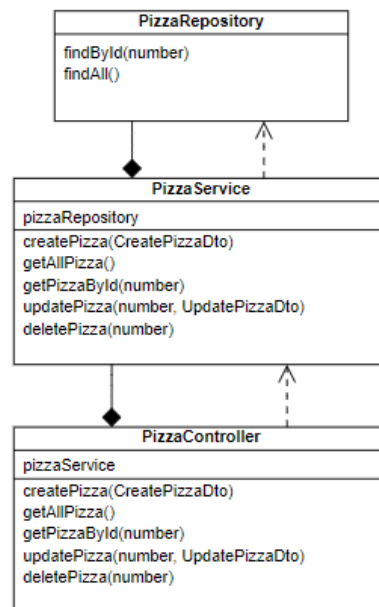


Рисунок 14 – Компоновка классов для обработки информации о пиццах

Показанные диаграммы наглядно демонстрируют серверную часть приложения.

2.4 Разработка клиентского программного приложения

2.4.1 Требования к пользовательскому интерфейсу

Общие требования к пользовательскому интерфейсу были сформированы следующим образом:

- интерфейс системы должен поддерживать русский язык;
- задачи пользователя должны решаться максимально быстро, просто и удобно;
- структура информации должна наглядно представлена, чтобы обеспечить интуитивное понимание и быстрый доступ к разделам системы;
- навигационные элементы должны быть понятными для пользователя и обеспечивать доступ ко всем разделам системы, а также отображать соответствующую информацию;
- дизайн и удобство интерфейса должны соответствовать ожиданиям современного пользователя, имеющего опыт работы с лучшими образцами подобных сервисов, и восприниматься им как комфортная, удобная и приятная рабочая среда;
- интерфейс системы должен быть адаптирован под контекст использования: где, при каких обстоятельствах и с помощью каких устройств пользователь будет взаимодействовать с системой;
- интерфейс должен быть доступен для понимания и использования людьми без специальных технических знаний и навыков в области компьютерной техники;
- для авторизации интерфейс должен предоставлять отдельную страничку.
- в интерфейсе должны быть отдельная страница для отображения пицц.

2.4.2 Описание архитектуры пользовательского интерфейса

При создании клиентской части приложения была использована архитектура, показанная на рисунке 15, которая позволяет создавать страницы из небольших элементов, объединяемых в более крупные блоки.

Например, в приложении есть кнопки – это можно назвать «атомами». Несколько кнопок объединяются в блок – «молекулу». Более крупная часть – это «организм», а затем шаблон страницы. В результате объединения всех этих элементов получается готовая страница со всеми необходимыми компонентами.

На рисунке 16 показано, как молекулы состоят из атомов, организмы – из молекул, а отображаемая страница представляет собой набор перечисленных элементов. Атомы представлены в виде кнопок меню, которое является организмом. При этом маленькие части, такие как атомы, могут сразу объединяться в организм. Молекулами на рисунке являются карточки с описанием пиццы, а организмами – меню, набор пицц и добавление новой пиццы.

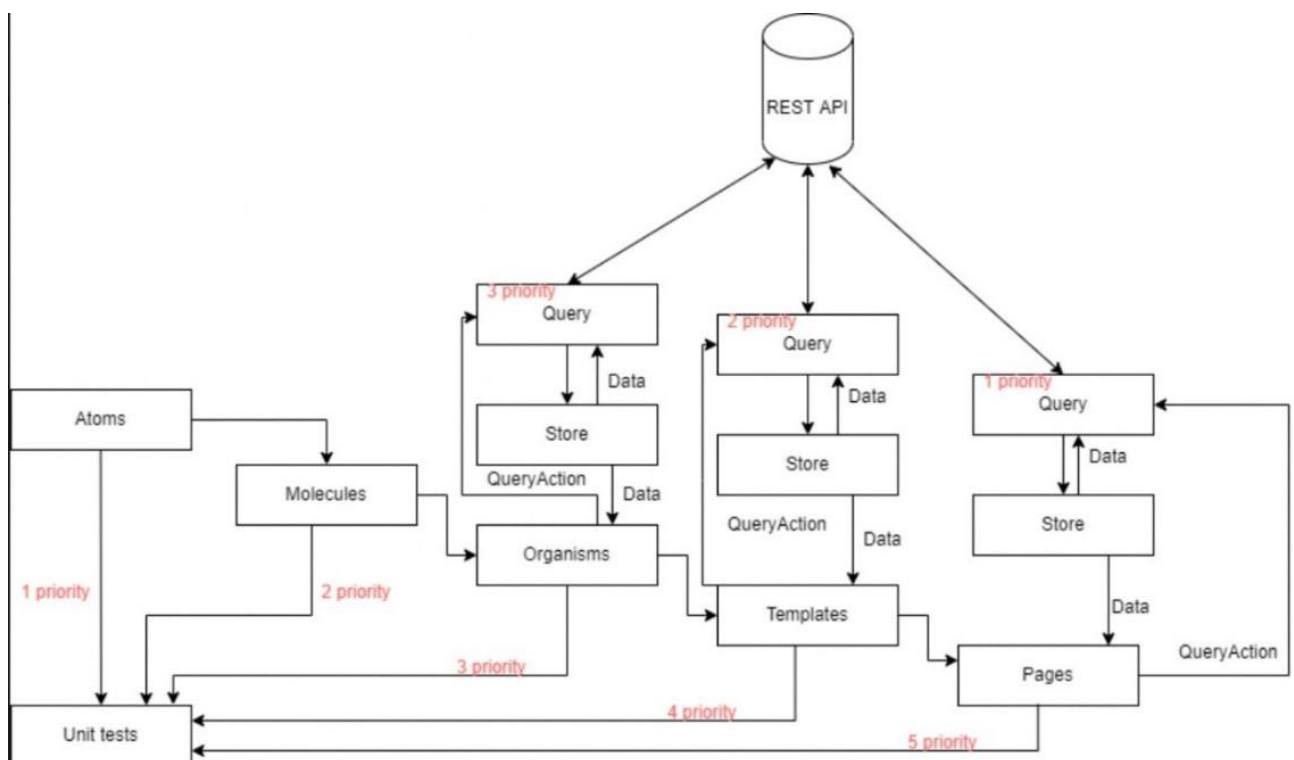


Рисунок 15 – Архитектура клиентского ПО

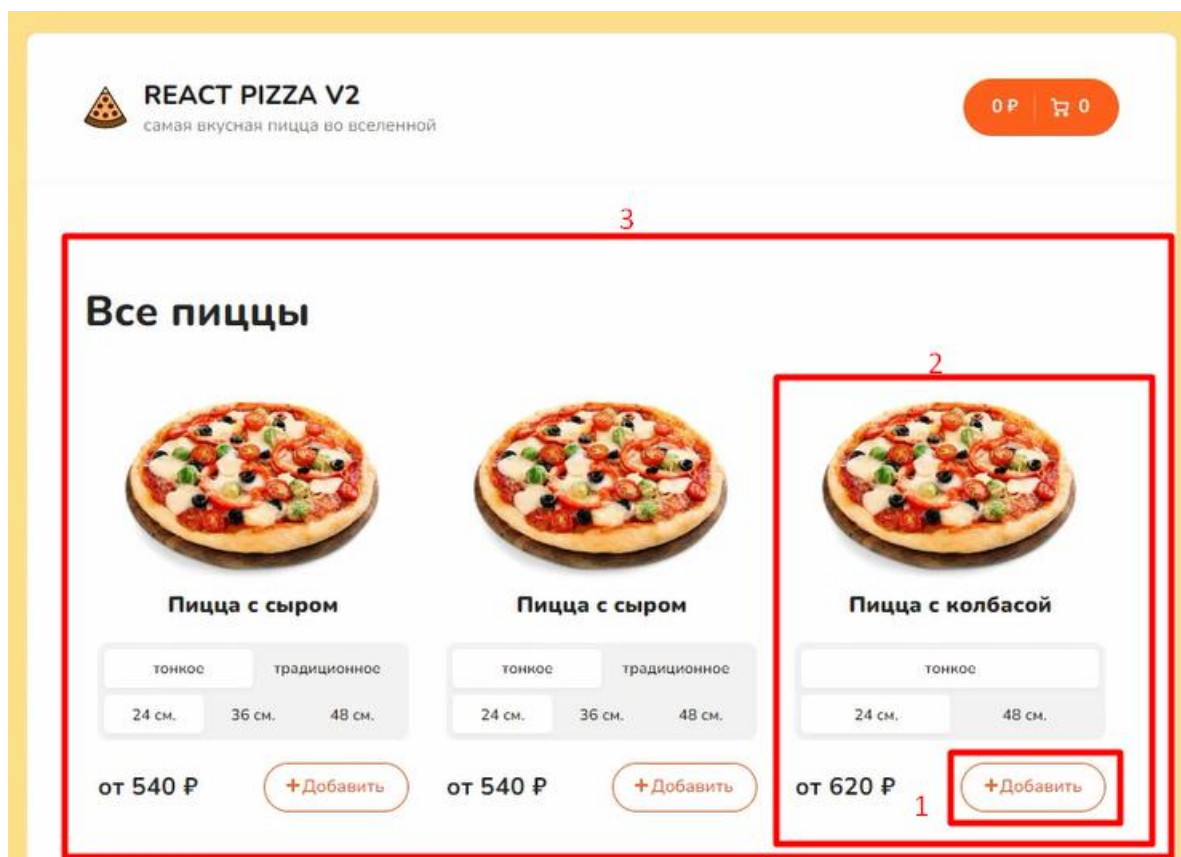


Рисунок 16 – Пример агрегации элементов страницы каталога пицц

На рисунке показаны атом (1), молекула (2) и организм (3). Почти у всех частей имеется собственное хранилище, в которое складывается информация об ответах на запросы с серверной части приложения, после они отображаются пользователю. Запросы могут отправляться разными частями страничек.

Различные страницы могут включать в себя одинаковые под части, например, организмы или молекулы. Переход между страничками достигается с помощью библиотеки React Route. Он помогает настроить пути для различных страничек.

Общение с серверной частью было реализовано с помощью Rest API.

Клиентская часть была разработана на языке JavaScript с использованием Фреймворка React [5].

Далее приведена иерархия страниц.

На рисунке 17 приведена иерархия страниц для роли администратор.

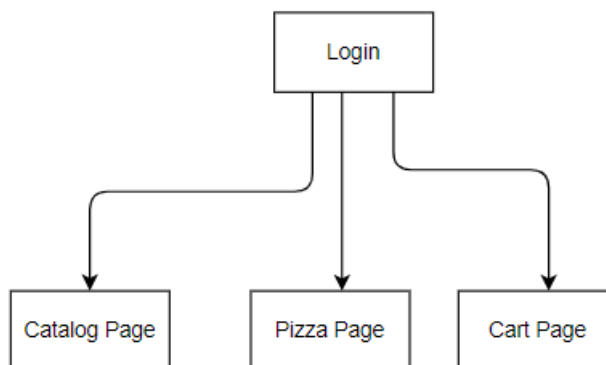


Рисунок 17 – Иерархия страниц

В приложении Б показаны примеры страниц системы.

В листингах 1 – 2 приведен код атома для компонента Button. Данный компонент – это кнопка может переиспользоваться в разных частях приложения. На рисунке 10 компонент используется для добавления пиццы в корзину.

Листинг 1 – Описание атома для компонента Button на языке JavaScript

```

import styles from './Button.module.scss';
import React from "react";
export const Button = (props:
React.ButtonHTMLAttributes<HTMLButtonElement>) => {
return <button className={styles.button} {...props}/>
}

```

Листинг 2 – Описание стилей атома для компонента Button с помощью css

```

@import '../..//scss/variables';
@import '../..//scss/libs/normalize';

.button {
  display: inline-block;
  background-color: $orange;
  border-radius: 30px;
  padding: 10px 20px;
  min-width: 100px;
  text-align: center;
  cursor: pointer;
  transition: background-color $duration ease-in-out, border-color
$duration ease-in-out;
  border: 1px solid transparent;
  @include noselect();
}
0

```

```

&,
span {
  color: #fff;
}

i,
span,
path,
svg {
  transition: all $duration ease-in-out;
}

&:hover {
  background-color: darken($orange, 8%);
}

&:active {
  background-color: darken($orange, 12%);
  transform: translateY(1px);
}

&--circle {
  display: flex;
  align-items: center;
  justify-content: center;
  width: 32px;
  height: 32px;
  min-width: 32px;
  padding: 0;
  border-width: 2px;
}

&--black {
  background-color: $black;

  &:hover,
  &:active {
    background-color: lighten($color: $black, $amount: 10);
  }
}

&--outline {
  background-color: #fff;
  border-color: $orange;
  &,
  span {
    color: $orange;
  }

  svg {
    path {
      fill: $orange;
    }
  }

  &:hover {
    background-color: $orange;
  }
}

```

```

    &,
    span {
        color: #fff;
    }

    svg {
        path {
            fill: #fff;
        }
    }
}

&:active {
    background-color: darken($orange, 8%);
}
}

&__delimiter {
    width: 1px;
    height: 25px;
    background-color: rgba(255, 255, 255, 0.25);
    margin-left: 14px;
    margin-right: 14px;
}

&--add {
    svg {
        margin-right: 2px;
    }

    span {
        font-weight: 600;
        font-size: 16px;
    }

    &:hover {
        i {
            background-color: #fff;
            color: $orange;
        }
    }

    i {
        display: inline-block;
        border-radius: 30px;
        background-color: $orange;
        color: #fff;
        font-weight: 600;
        width: 22px;
        height: 22px;
        font-style: normal;
        font-size: 13px;
        line-height: 22px;
        position: relative;
        top: -1px;
        left: 3px;
    }
}
}

```

```

&--cart {
  display: flex;
  align-items: center;
  line-height: 23px;
  padding: 12px 25px;

  svg {
    margin-right: 8px;
    margin-bottom: 1px;
  }

  span {
    font-weight: 600;
    font-size: 16px;
  }
}

```

Данный код демонстрирует JSX разметку компонента Button и CSS стилизация.

3 Тестирование программного обеспечения

3.1 Выбор методологии тестирования

Существует два основных подхода к тестированию: сценарное (scripted) и исследовательское (exploratory).

Сценарное тестирование предполагает разделение задачи на этапы (подготовка, выполнение, завершение и т. д.) и последующее следование этим этапам. Этот подход можно описать как «я заранее продумываю план и затем его выполняю».

Исследовательское тестирование, напротив, подразумевает более гибкий подход. Процесс тестирования разрабатывается непосредственно во время работы. Тестирование начинается с определённой точки, в процессе выявляются другие важные аспекты для проверки, и специалист решает, какой из них наиболее значим и в каком направлении двигаться дальше. То есть планирование и запуск тестов происходят в постоянном взаимодействии [6].

В этом случае был выбран сценарный подход, так как он наиболее эффективен для проверки работоспособности приложения именно разработчиком, который понимает логику его работы и может правильно

разработать тестовые сценарии. В сценарном подходе выделяют следующие виды:

- тестирование белого ящика – метод тестирования ПО, который предполагает полный доступ к коду проекта;
- тестирование чёрного ящика – метод тестирования ПО, при котором не предполагается доступ (полный или частичный) к системе, основывающийся на работе исключительно с внешним интерфейсом тестируемой системы;
- системное тестирование – процесс тестирования системы, на котором проводится не только функциональное тестирование, но и оценка характеристик качества системы – ее устойчивости, надежности, безопасности и производительности;
- интеграционное тестирование – тестирование, направленное на проверку корректности взаимодействия нескольких модулей, объединенных в единое целое;
- модульное тестирование – проводится для тестирования какого-либо одного логически выделенного и изолированного элемента (модуля) системы в коде;
- функциональное тестирование – направлено на проверку корректности работы функциональности приложения [7].

В данной работе были выбраны следующие виды тестирования: тестирование черного ящика, модульное тестирование, функциональное тестирование.

3.2 Тестирование черного ящика модуля «Регистрация пользователей»

3.2.1 Классы эквивалентности

В таблице 9 представлены классы эквивалентности поля «Роль» для варианта использования «Регистрация пользователей».

Таблица 9 – Классы эквивалентов для поля «Роль»

№	Входные данные	Правильность класса
1	Пользователь	+
2	a..я, A..Я	-
3	a..z, A..Z	-
4	(-9999...0]	-
5	(9999...+∞)	-
6	∅	-
7	Иные символы	-

В таблице 10 представлены тесты для поля «Роль» в соответствии с классами эквивалентности.

Таблица 10 – Тесты для поля «Роль»

Входные данные	Предполагаемые выходные данные	ЭК	Результат
Пользователь	Ввод данных	1	Данные успешно добавлены в базу
123	Невозможно ввести данные в поле	2	Данные не вводятся в поле
абв	Невозможно ввести данные в поле	3	Данные не вводятся в поле
abc	Невозможно ввести данные в поле	4	Данные не вводятся в поле
-1	Сообщение об ошибке	5	Данные не вводятся в поле
∅	Невозможно ввести данные в поле	6	Данные не вводятся в поле
*#	Невозможно ввести данные в поле	7	Данные не вводятся в поле

Тестирование пройдено успешно.

3.2.2 Функциональная диаграмма и таблица решений

Метод причинно-следственных связей показан в таблице 11.

В таблице 11 представлены причины и следствия различных исходов варианта использования «Добавление пиццы».

Таблица 11 – Причины и следствия возможных исходов варианта использования «Добавление пиццы»

Условное обозначение причины	Название причины	Условное обозначение следствия	Название следствия
1	2	3	4
П1	Неавторизованный пользователь	С1	Сообщение: «Нет прав»
1	2	3	4
П2	Авторизованный пользователь	С2	Сообщение: «Пицца успешно добавлена»

На рисунке 18 представлена функциональная диаграмма, составленная по таблице 11.

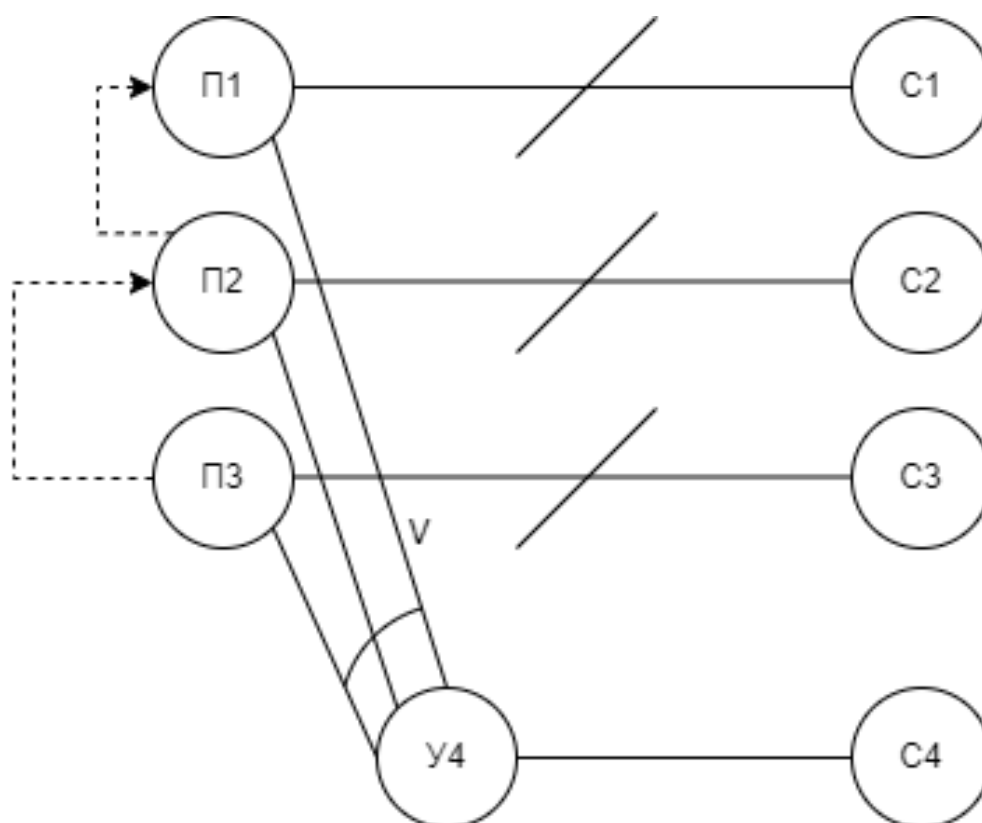


Рисунок 18 – Функциональная диаграмма

В таблице 12 представлены решения по функциональной диаграмме (рисунок 18).

Таблица 12 – Таблица решений по функциональной диаграмме

		1	2	3	4	6	5	7	8
1	П	1	0	1	0	1	0	1	1
2	П	–	1	–	0	–	1	–	1
3	П	–	–	–	0	–	–	–	1
1	С	1	1	0	1	0	1	0	0
2	С	1	1	1	0	1	1	1	1

На рисунке 19 представлена диаграмма s&t для варианта использования «Регистрация работника».

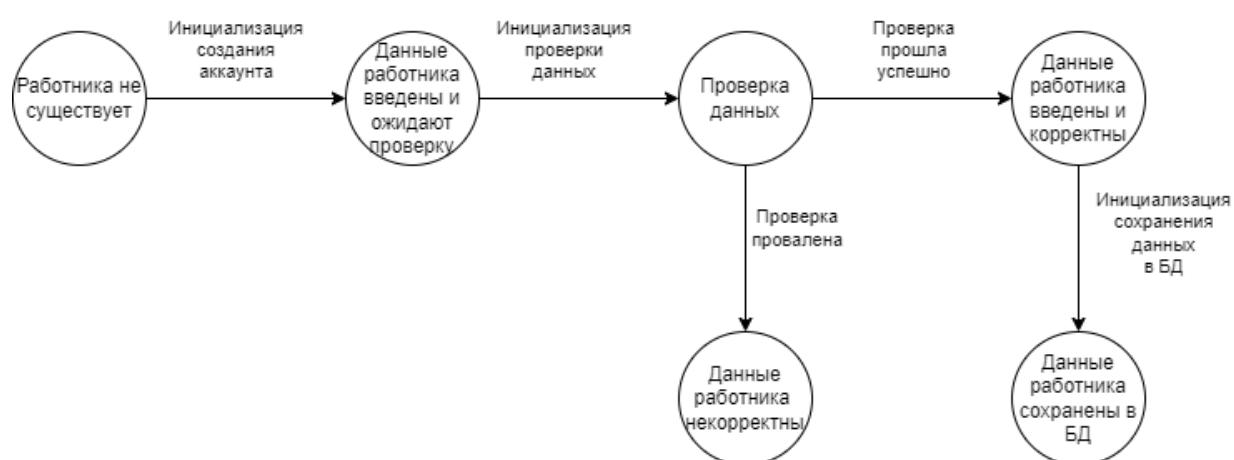


Рисунок 19 – Диаграмма s&t

3.3 Модульное тестирование модуля «Регистрация пользователя»

Для проверки работоспособности приложения было совершено Unit-тестирование всех вариантов использования. Суммарно проведено более 100 тестов. Пример тестирования модуля «Регистрация пользователя» приведен в таблице 13.

Таблица 13 – Unit-тестирование

Вариант использования/название теста	Состав теста	Результат
Регистрация пользователя (добавление пользователя в пустую таблицу)	Логин: string Пароль: string	Тест пройден (Рисунок 18)
Регистрация пользователя (добавление пользователя в таблицу с одной записью)	Логин: string1 Пароль: string1	Тест пройден
Регистрация пользователя (добавление пользователя в таблицу, где N записей)	Логин: string3 Пароль: string3	Тест пройден (Рисунок 19)
Регистрация пользователя (добавление пользователя, пустое тело запроса)		Тест пройден
Регистрация пользователя (добавление пользователя, некорректные данные)	Логин: string3 Пароль: string3	Тест пройден (Рисунок 20)

На рисунках 20, 21 представлены данные в БД, а на рисунке 22 тестирование

	id [PK] integer	email character varying (255)	password character varying (255)	createdAt timestamp with time zone	updatedAt timestamp with time zone
1	1	andrey@mail.ru	\$2a\$05\$wt9VBZNpnf6Q4VOd0dHeZepfbun4uYfLNN8854hRIQqw.LnY36C...	2024-04-22 20:56:47.624+03	2024-04-22 20:56:47.624+03

Рисунок 20 – Добавление работника в пустую таблицу

	id [PK] integer	email character varying (255)	password character varying (255)	createdAt timestamp with time zone	updatedAt timestamp with time zone
1	1	andrey@mail.ru	\$2a\$05\$wt9VBZNpnf6Q4VOd0dHeZepfbun4uYfLNN8854hRIQqw.LnY36C...	2024-04-22 20:56:47.624+03	2024-04-22 20:56:47.624+03
2	2	test@mail.ru	\$2a\$05\$SLgboOdVSj/f1gwzk0VR.ZZ9X2UwG0ebug8MyOABoc7.IG2Q/YS	2024-04-27 09:35:53.913+03	2024-04-27 09:35:53.913+03

Рисунок 21 – Добавление работника в таблицу с одной записью

✓ Test Results	753 ms
✓ auth.controller.spec.ts	753 ms
✓ AuthControllerTest	753 ms
✓ Добавление в пустую таблицу	231 ms
✓ Добавление в таблицу с одной записью	252 ms
✓ Ввод некорректных данных	270 ms

Рисунок 22 – Добавление в пустую таблицу, добавление в таблицу с одной записью и ввод некорректных данных

Тестирование пройдено успешно.

3.4 Функциональное тестирование клиентского ПО

Подготовка для функционального тестирования заключалась в анализе функциональных требований. Все тесты производились вручную. В таблице 15 представлены основные тесты и их описание.

Таблица 15 – Интеграционное тестирование

Состав теста	Ожидаемый результат	Наблюдаемый результат
Тестирование работоспособности страницы для просмотра каталога пицц	Тест пройден	Тест пройден (Рисунок 23)
Тестирование работоспособности страницы для просмотра детальной информации о пицце	Тест пройден	Тест пройден (Рисунок 24)
Тестирование работоспособности страницы для авторизации пользователя.	Тест пройден	Тест пройден (Рисунок 25)

На рисунках 23 – 25 приведены скриншоты отражающие результаты тестирования.

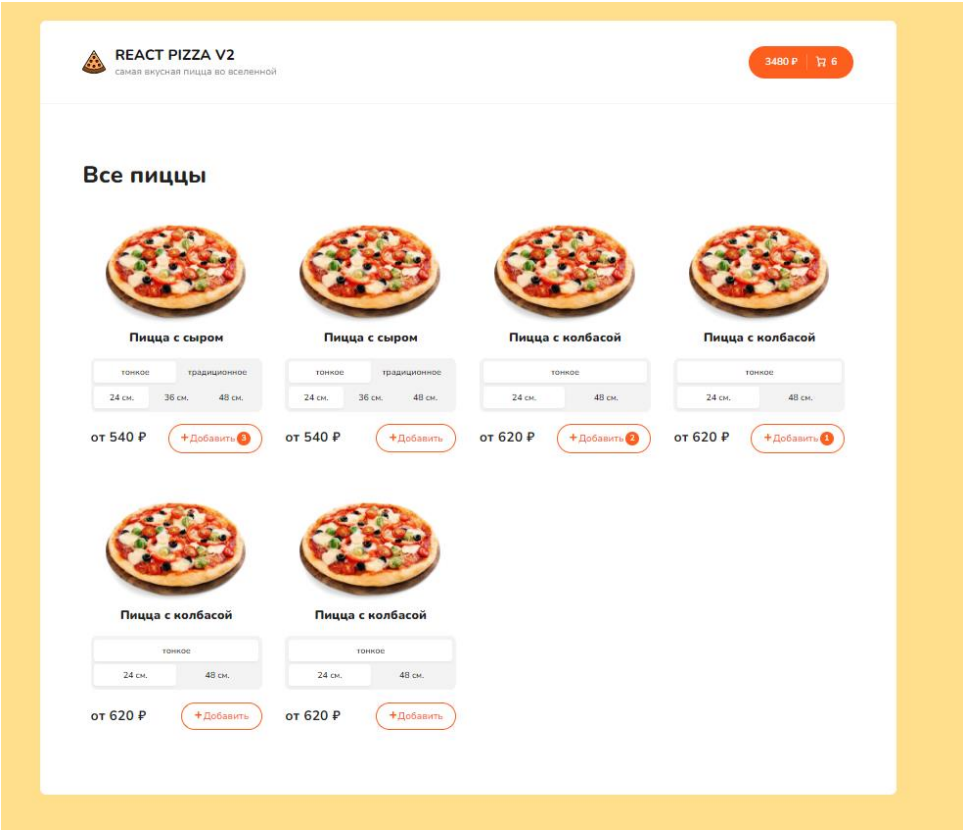


Рисунок 23 – Тестирование страницы для просмотра каталога пицц

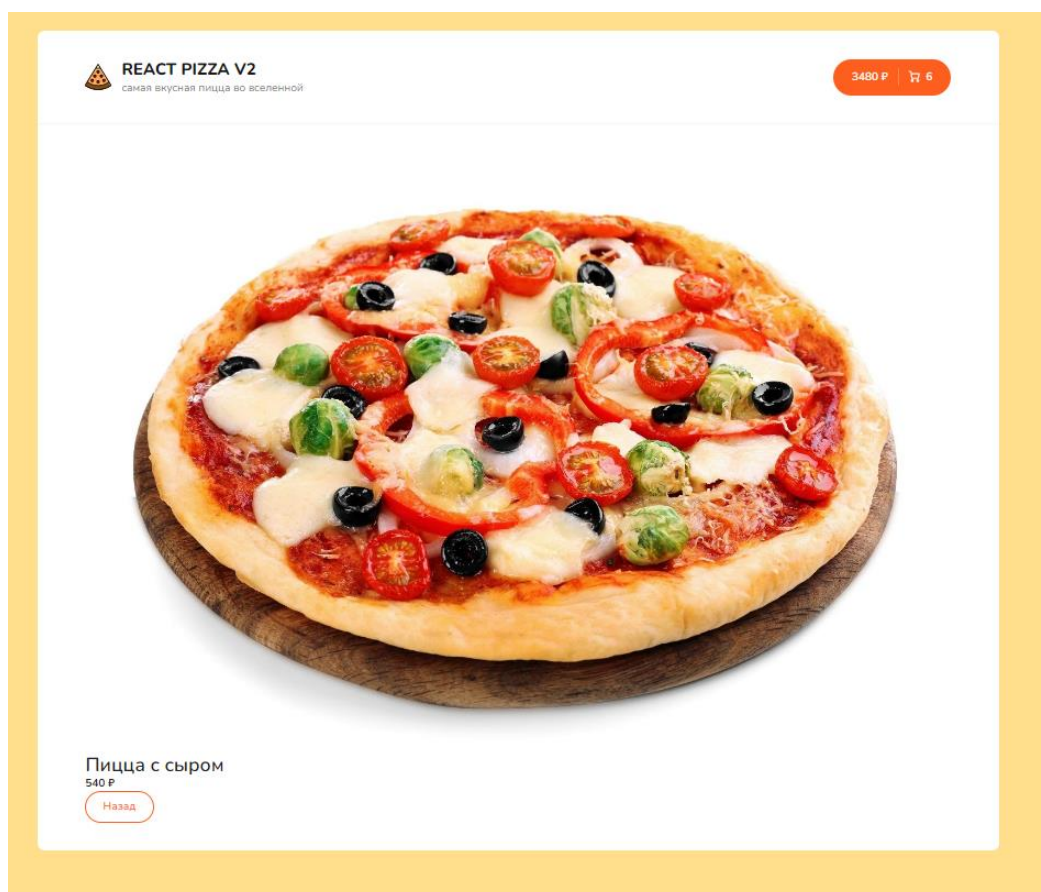


Рисунок 24 – Тестирование страницы для просмотра детальной информации о пицце

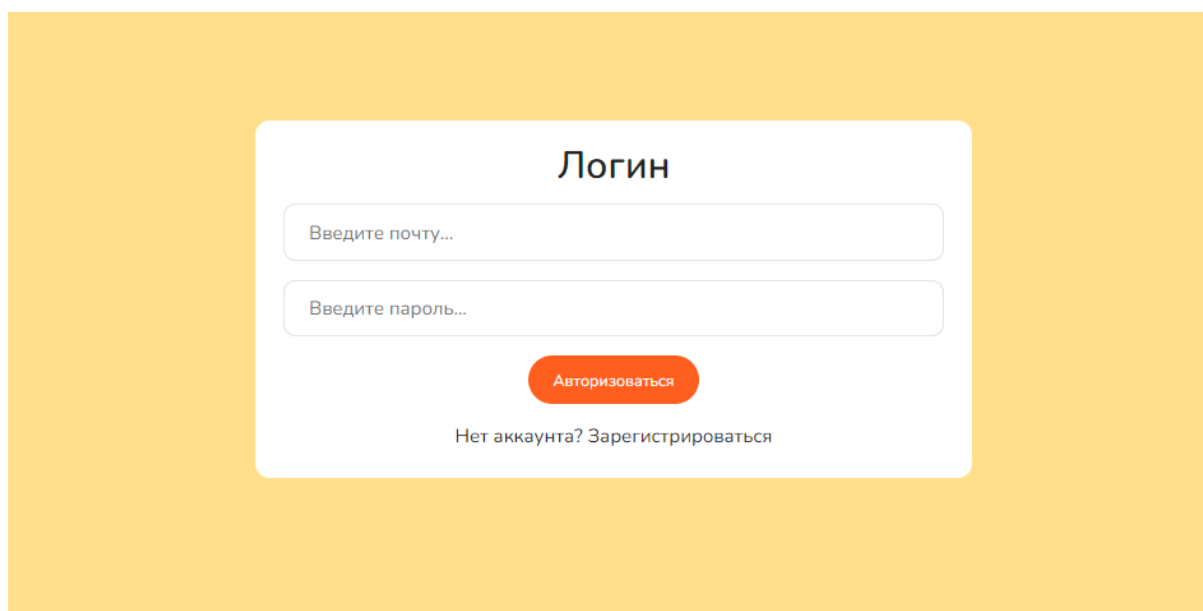


Рисунок 25 – Тестирование страницы для авторизации

Тестирование пройдено успешно.

3.5 Метрики кода

В мире разработки программного обеспечения существуют количественные показатели, которые позволяют оценить качество и эффективность продукта или процесса. Эти показатели называются мерами. Они могут измерять количество ошибок, время выполнения задачи или другие аспекты работы системы.

Одним из видов мер являются метрики программного кода. Они представляют собой количественные характеристики, позволяющие оценить, насколько хорошо система или её компоненты соответствуют определённым требованиям. Примером такой метрики может быть количество обнаруженных ошибок на затраченный человеко-час [8].

Метрики программного кода можно разделить на группы:

- метрики размера программ (Source Lines of Code – SLOC-метрика), которые измеряют объем кода в строках;
- метрики сложности потока управления программ (метрики Холстеда), которые оценивают сложность структуры программы.

Для варианта использования «Добавить пиццу» можно вычислить соответствующие метрики кода, чтобы оценить его качество и сложность. Это поможет понять, насколько эффективно работает система и какие улучшения можно внести для повышения её производительности.

3.5.1 Метрика SLOC

SLOC-метрика позволяет подсчитать физические и логические строки кода. Физические строки – это все непустые строки исходного файла программы. Логические строки кода – это количество команд, используемых в программе;

Определение команды зависит от конкретного языка программирования. Применяв SLOC-метрику к программному коду для добавления пиццы (см. рисунок 26), **мы** получили результаты, которые представлены в таблице 16.

```

async createPizza(dto: CreatePizzaDto) : Promise<Pizza> { Show usages
    return await this.pizzaRepository.create(dto);
}

```

Рисунок 26 – Программный код для добавления пиццы

Таблица 16 – SLOC-метрики

Число физических строк	3
Число логических строк	1
Число пустых строк	0
Число строк-комментариев	0

3.5.2 Метрики Холстеда

Метрики Холстеда – это количественные характеристики, которые позволяют оценить сложность программы. Они включают в себя четыре измеряемые характеристики:

- общее число операторов в программе;
- число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций;
- общее число операндов в программе;
- число уникальных операндов программы.

На основе метрик Холстеда можно получить следующие оценки:

- словарь программы = число уникальных операторов + число уникальных операндов;
- длина программы = общее число операторов + общее число операндов;
- показатель усилий программиста = сложность программы * объем программы;
- объем программы = длина программы $\log_2$ словарь программы;
- сложность программы = (число уникальных операторов/2) * (общее число операндов/число уникальных операндов).

Применим метрики Холстеда к программному коду для расчета обратной матрицы и получаем результаты, представленные в таблице 17.

Таблица 17 – Метрики Холстеда

Число операторов (уникальных)	Число операндов (уникальных)
6 (3)	4 (3)

Для данного метода были рассчитаны следующие параметры:

- показатель усилий программиста $\approx 64,625$.
- сложность программы $= (3/2) * (10/6) \approx 2,5$;
- объем программы $= 10 \log_2 6 \approx 25,85$;
- длина программы $= 6 + 4 = 10$;
- словарь программы $= 3 + 3 = 6$;

3.5.3 Метрики устойчивости и абстрактности компонент

Для компонента сервисного слоя, который отвечает за обработку информации о пиццах и находится в классе `PizzaServiceImpl`, были вычислены метрики $S_a = 3$ и $S_e = 1$.

Значение I составило $1/4$. В коде программы абстракцией для класса, поэтому значение метрики A составило $0,5$.

Соотношение между устойчивостью (I) и абстрактностью (A) компонента обработки информации о пиццах можно представить на графике, изображённом на рисунке 27.

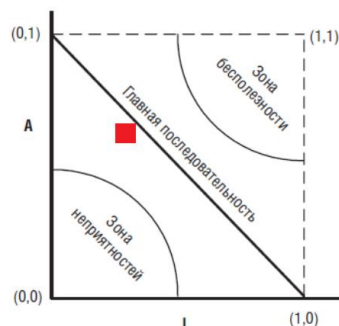


Рисунок 27 – Соотношение между устойчивостью (I) и абстрактностью (A) для компонента `PizzaServiceImpl`

Значение D находится в диапазоне от 0 до примерно 0,707. Для компонента `PizzaServiceImpl` это значение приблизительно равно 0,18.

Для упрощения анализа можно использовать нормированное расстояние D' , которое рассчитывается по формуле: $D' = |A + I - 1|$.

Значение метрики D' может изменяться в диапазоне от 0 (если компонент находится на главной последовательности) до 1 (если он максимально удалён от неё). Для компонента `PizzaServiceImpl` значение D' составляет 0,25.

3.5.4 Комплексный показатель качества кода

Для оценки сложности поддержки кода рассмотрим метод, который получает все пиццы в классе `PizzaService`.

HV (Halstead Volume) = 8.

CC (Cyclomatic Complexity) = 4.

LoC (Lines of Code) = 7.

MI (Maintainability Index) = 93,14.

Полученные значения показывают, что код легко поддерживать. Исходя из этих метрик, можно сделать вывод, что приложение имеет простой и понятный код, который легко модифицировать и улучшать.

4 Планирование и оценка бюджета проекта

Управление проектами играет важную роль в работе, направленной на достижение целей проекта.

Существует несколько основных групп при планировании проектов:

- планирование проектных задач;
- планирование расписания проекта;
- планирование команды проекта;
- управление рисками.

При планировании разработки было выделено 30 задач. Перечень задач и их виды, предшественники и длительность каждой задачи представлены в таблице 18.

Таблица 18 – Перечень задач проекта

№	Название	Вид задачи	Предшественник	Длительность, дни
1	2	3	4	5
1	Начало реализации проекта	Веха	–	–
2	Программирование	Фаза	–	31
3	Разработка базы данных	Фаза	–	9
4	Постановка задачи	Задача	3	1
5	Разработка структуры базы данных	Задача	4	3
6	Заполнение базы данных	Задача	5	2
7	Разработка триггеров	Задача	6	3
8	Разработка базы данных завершена	Задача	7	–
9	Разработка серверной части приложения	Фаза	3	9
10	Постановка задачи	Задача	9	1
1	2	3	4	5
11	Разработка запросов к базе данных	Задача	10	4
12	Разработка endpoint'ов	Задача	11	4
13	Разработка завершена	Задача	12	–
14	Разработка клиентской части приложения	Фаза	9	5
15	Постановка задачи	Задача	14	1
16	Разработка интерфейса	Задача	15	4
17	Разработка клиентской части завершена	Задача	16	–
18	Программирование завершено	Задача	17	–
19	Отладка	Фаза	14	5
20	Тестирование и исправление ошибок	Задача	19	
21	Составление программной документации	Задача	20	
22	Отладка завершена	Задача	21	–
23	Рефакторинг	Фаза	19	3
24	Оптимизация кода	Задача	23	3
25	Проверка соответствия SOLID	Задача	24	2
26	Документирование	Фаза	23	2
27	Создание установочного файла	Задача	26	1
28	Написание руководства пользования	Задача	27	1
29	Документация завершена	Задача	28	–
30	Конец проекта	Веха	29	–

Планирование разработки и оценка бюджета были выполнены с помощью программы для управления проектами Microsoft Project 2016. Проект был начат 3 мая 2024 года. Результат внесения информации о проекте представлен на рисунке 28.

Ид.	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало реализации проекта	0 дней	Ср 03.05.23	Ср 03.05.23		
2	Программирование	31 дней	Ср 03.05.23	Ср 14.06.23		Интернет[600,00 Р]
3	Разработка базы данных	9 дней	Ср 03.05.23	Пн 15.05.23		
4	Постановка задачи	1 день	Ср 03.05.23	Ср 03.05.23	1	Ведущий программист;Пр
5	Разработка структуры базы данных	3 дней	Чт 04.05.23	Пн 08.05.23	4	Ведущий программист
6	Заполнение базы данных	2 дней	Вт 09.05.23	Ср 10.05.23	5	Ведущий программист
7	Разработка триггеров	3 дней	Чт 11.05.23	Пн 15.05.23	6	Ведущий программист
8	Разработка базы данных завершена	0 дней	Пн 15.05.23	Пн 15.05.23	7	
9	Разработка серверной части приложения	9 дней	Вт 16.05.23	Пт 26.05.23	3	
10	Постановка задачи	1 день	Вт 16.05.23	Вт 16.05.23		
11	Разработка запросов к базе данных	4 дней	Ср 17.05.23	Пн 22.05.23	10	
12	Разработка endpoint'ов	4 дней	Вт 23.05.23	Пт 26.05.23	11	
13	Разработка завершена	0 дней	Пт 26.05.23	Пт 26.05.23	12	
14	Разработка клиентской части приложения	5 дней	Пн 29.05.23	Пт 02.06.23	13	
15	Постановка задачи	1 день	Пн 29.05.23	Пн 29.05.23	1	Ведущий программист;Пр
16	Разработка интерфейса	4 дней	Вт 30.05.23	Пт 02.06.23	15	Программист;Ведущий пр
17	Разработка клиентской части завершена	0 дней	Пт 02.06.23	Пт 02.06.23	16	
18	Программирование завершено	0 дней	Пт 02.06.23	Пт 02.06.23	8;17	
19	Отладка	5 дней	Пн 05.06.23	Пт 09.06.23	18	Интернет[600,00 Р]
20	Тестирование и исправление ошибок	3 дней	Пн 05.06.23	Ср 07.06.23	18	Тестировщик
21	Составление программной документации	2 дней	Чт 08.06.23	Пт 09.06.23	20	Программист;Тестировщ
22	Отладка завершена	0 дней	Пт 09.06.23	Пт 09.06.23	21	
23	Рефакторинг	3 дней	Пн 12.06.23	Ср 14.06.23	22	
24	Оптимизация кода	3 дней	Пн 12.06.23	Ср 14.06.23		
25	Проверка соответствия SOLID	2 дней	Пн 12.06.23	Вт 13.06.23		
26	Документирование	2 дней	Чт 15.06.23	Пт 16.06.23		Интернет[600,00 Р]
27	Создание установочного файла	1 день	Чт 15.06.23	Чт 15.06.23	23	Тестировщик;Лицензия п
28	Написание руководства пользователя	1 день	Пт 16.06.23	Пт 16.06.23	27	Тестировщик
29	Документация завершена	0 дней	Пт 16.06.23	Пт 16.06.23	28	
30	Конец проекта	0 дней	Пт 16.06.23	Пт 16.06.23	29	

Рисунок 28 – Задачи проекта

В соответствии с занесенными задачами проекта, проанализированными ресурсами и соответствующими им затратами, Microsoft Office Project 2016 автоматически построит диаграмму, которая иллюстрирует календарный график выполнения работ по проекту. Диаграмма Ганта представлена на рисунке 29.

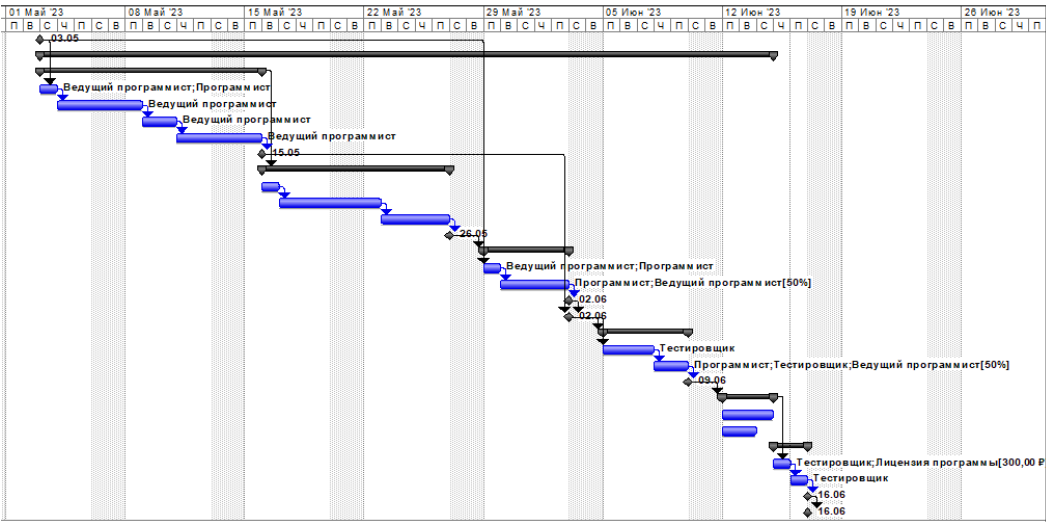


Рисунок 29 – Диаграмма Ганта

На рисунке 30 представлено распределение ресурсов, где синим цветом отмечены нормы рабочего времени работников, а красным – переработки.

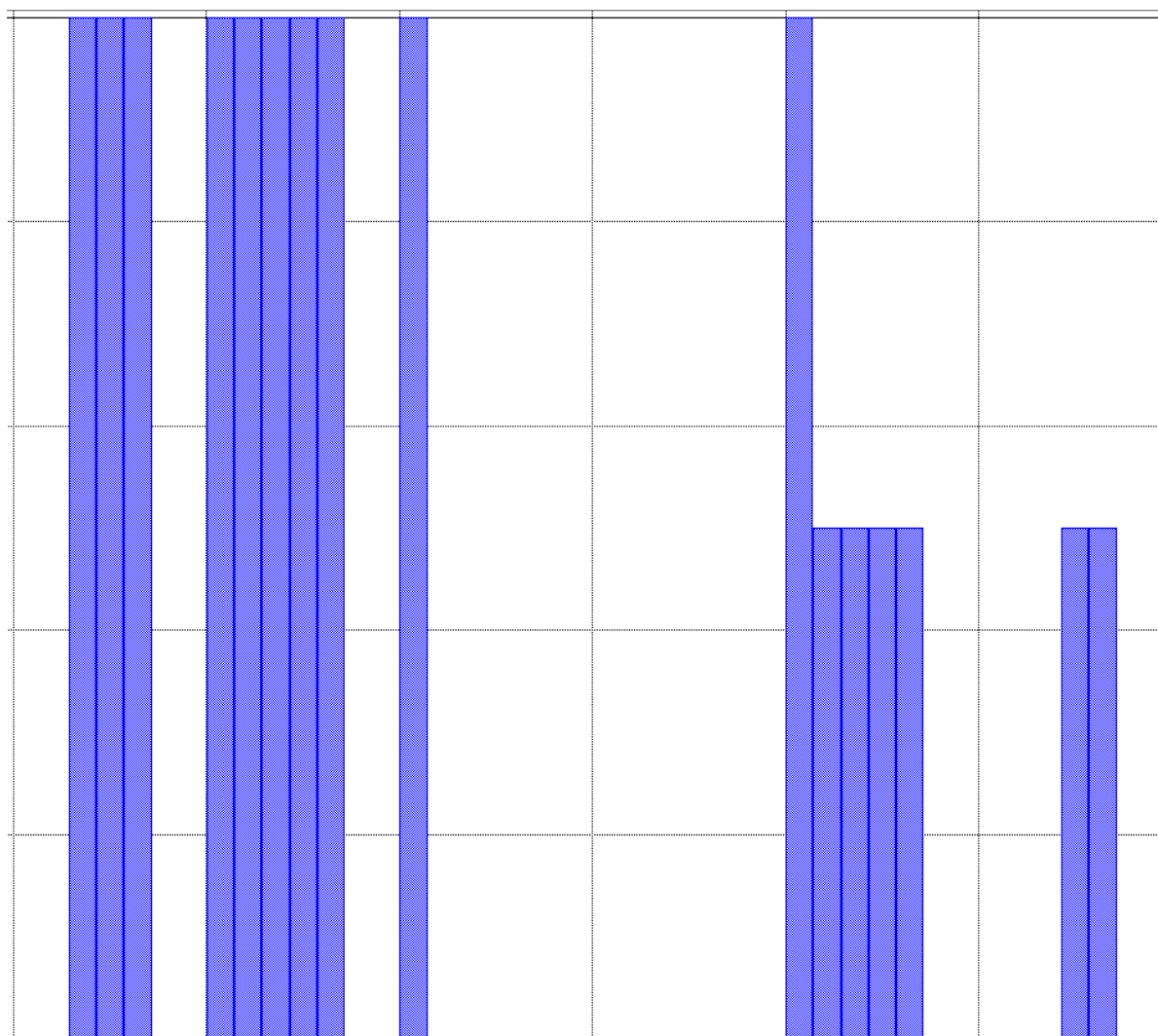


Рисунок 30 – Распределение ресурсов

Как видно по рисунку, переработок в плане нет, более того, в некоторые дни работник занят на 50% времени, соответственно данный план можно считать корректным.

Также приложение Microsoft Office Project 2016 может рассчитать стоимость проекта, она составила: 38925 рублей. Она состоит из затрат на заработную плату разработчиков, а также на электричество и интернет.

Заключение

В результате выполнения данной работы был разработан интернет-магазин онлайн пиццерии и были реализованные поставленные задачи.

В ходе выполнения работы были выполнены следующие этапы:

- постановка задачи и анализ предметной области;
- проектирование элементов системы;
- тестирование программы;
- расчет метрик кода.

В результате работы было разработано Web-приложение. После некоторой доработки на основе жалоб и пожеланий пользователей, оно может быть использовано и внедрено, как основное средство для продажи пиццы онлайн.

Список использованных источников

1. SOLID (объектно-ориентирование программирование) // Википедия [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/SOLID_объектно-ориентированное_программирование (дата обращения: 01.05.2023);
2. Прошкина, Е. Н. Язык моделирования информационных процессов UML 2 / Прошкина Е. Н., Балашова И. Ю. – П. : Изд-во ПГУ, 2018 – 112 с.
3. Объектно-ориентированное программирование // Википедия [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование (дата обращения: 03.05.2023);
4. Юлиана Козмина, Роб Харроп, Крис Шефер, Кларенс Хо. Spring 5 для профессионалов / Диалектика, 2019 – 521 с.;
5. Оригинал Брэда Фроста (Brad Frost), перевод Naissur. Атомарный веб-дизайн // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/249223/> (дата обращения: 10.05.2023);
6. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. – СПб.: Питер, 2021. – 352 с.
7. Оригинал Илья Suzdanitski, перевод ru_vds. React.js: понятное руководство для начинающих // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/company/ruvds/blog/428077/> (дата обращения: 10.05.2023);
8. В поисках самого лучшего способа тестирования системы // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/jugru/articles/341852/> (дата обращения: 11.05.2023);
9. Фундаментальная теория тестирования // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/549054/> (дата обращения: 11.05.2023);
10. Гурьянов, Л. В. Выпускная квалификационная работа: стандарты и программные технологии. Учебно-методическое пособие для бакалавров по направлениям 09.03.04 «Программная инженерия», 09.03.02 «Информационные системы и технологии» / Гурьянов Л. В., Князев В. Н., Балашова И. Ю., Дзюба Е.

А., Попова Н. А., Такташкин Д. В., Самуйлов С. В., Макарычев П. П. – П. : Изд-во ПГУ, 2020 – 182с. с.

Приложение А
(рекомендуемое)
Глоссарий

Таблица А.1 – Подробное описание таблицы «Пользователь»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
	Код_пользователя	целый		– 9999	нет		
Email	Почта	символьный		‘А-я’	нет		andrey@mail.ru
password	Пароль	символьный		‘А – z’, ‘0 – 9’, знаки препинания	нет		432rfdsaf43y6FSA32!

Таблица А.2 – Подробное описание таблицы «Пицца»

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Возможность значения Null	Роль	Пример
id	Код_пиццы	целый	4	0001 – 9999	нет	PK	2
title	Название пиццы	символьный	50	‘А-я’	нет		Моцарелла
price	Цена	вещественный	8	000 – 999	нет		0
imageUrl	Ссылка на изображение пиццы	символьный	50	‘А-я’	нет		https://w.forfun.com/fetch/03/031db8e05db69b30c6bc563ab5def73c.jpeg
rating	Рейтинг пиццы	вещественный	4	-0 – 5.0	нет		

Приложение Б
(рекомендуемое)
Код программы

```

import React from 'react';
import { Link } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';

import { CartItem, CartEmpty } from '../components';

import { selectCart } from '../redux/cart/selectors';
import { clearItems } from '../redux/cart/slice';
import MainLayout from '../layouts/MainLayout';

const Cart: React.FC = () => {
  const dispatch = useDispatch();
  const { totalPrice, items } = useSelector(selectCart);

  const totalCount = items.reduce((sum: number, item: any) => sum +
item.count, 0);

  const onClickClear = () => {
    if (window.confirm('Очистить корзину?')) {
      dispatch(clearItems());
    }
  };

  if (!totalPrice) {
    return <CartEmpty />;
  }

  return (
    <MainLayout>
      <div className="cart-container">
        <div className="cart-wrapper">
          <div className="cart-header">
            <h2 className="cart-title">
              <svg
                width="18"
                height="18"
                viewBox="0 0 18 18"
                fill="none"
                xmlns="http://www.w3.org/2000/svg"
              >
                <path
                  d="M6.33333 16.3333C7.06971 16.3333 7.66667 15.7364
7.66667 15C7.66667 14.2636 7.06971 13.6667 6.33333 13.6667C5.59695 13.6667
5 14.2636 5 15C5 15.7364 5.59695 16.3333 6.33333 16.3333Z"
                  stroke="white"
                  strokeWidth="1.8"
                  strokeLinecap="round"
                  strokeLinejoin="round"
                ></path>
                <path
                  d="M14.3333 16.3333C15.0697 16.3333 15.6667 15.7364
15.6667 15C15.6667 14.2636 15.0697 13.6667 14.3333 13.6667C13.597 13.6667
13 14.2636 13 15C13 15.7364 13.597 16.3333 14.3333 16.3333Z"
                  stroke="white"
                  strokeWidth="1.8"
                  strokeLinecap="round"
                  strokeLinejoin="round"
                ></path>
              </svg>
            </h2>
          </div>
        </div>
      </div>
    </MainLayout>
  );
};

```

```

        <path
            d="M4.78002 4.99999H16.3334L15.2134 10.5933C15.1524
10.9003 14.9854 11.176 14.7417 11.3722C14.4979 11.5684 14.1929 11.6727
13.88 11.6667H6.83335C6.50781 11.6694 6.1925 11.553 5.94689
11.3393C5.70128 11.1256 5.54233 10.8295 5.50002 10.5067L4.48669
2.82666C4.44466 2.50615 4.28764 2.21182 4.04482 1.99844C3.80201 1.78505
3.48994 1.66715 3.16669 1.66666H1.66669"
            stroke="white"
            strokeWidth="1.8"
            strokeLinecap="round"
            strokeLinejoin="round"
        ></path>
    </svg>
    Корзина
</h2>
<div onClick={onClickClear} className="cart-clear">
    <svg
        width="20"
        height="20"
        viewBox="0 0 20 20"
        fill="none"
        xmlns="http://www.w3.org/2000/svg"
    >
        <path
            d="M2.5 5H4.16667H17.5"
            stroke="#B6B6B6"
            strokeWidth="1.2"
            strokeLinecap="round"
            strokeLinejoin="round"
        ></path>
        <path
            d="M6.66663 5.00001V3.33334C6.66663 2.89131 6.84222
2.46739 7.15478 2.15483C7.46734 1.84227 7.89127 1.66667 8.33329
1.66667H11.6666C12.1087 1.66667 12.5326 1.84227 12.8451 2.15483C13.1577
2.46739 13.3333 2.89131 13.3333 3.33334V5.00001M15.8333
5.00001V16.6667C15.8333 17.1087 15.6577 17.5326 15.3451 17.8452C15.0326
18.1577 14.6087 18.3333 14.1666 18.3333H5.83329C5.39127 18.3333 4.96734
18.1577 4.65478 17.8452C4.34222 17.5326 4.16663 17.1087 4.16663
16.6667V5.00001H15.8333Z"
            stroke="#B6B6B6"
            strokeWidth="1.2"
            strokeLinecap="round"
            strokeLinejoin="round"
        ></path>
        <path
            d="M8.33337 9.16667V14.1667"
            stroke="#B6B6B6"
            strokeWidth="1.2"
            strokeLinecap="round"
            strokeLinejoin="round"
        ></path>
        <path
            d="M11.6666 9.16667V14.1667"
            stroke="#B6B6B6"
            strokeWidth="1.2"
            strokeLinecap="round"
            strokeLinejoin="round"
        ></path>
    </div>

```

```

        </svg>
        <span>Очистить корзину</span>
      </div>
    </div>
    <div className="cart-items">
      {items.map((item: any) => (
        <CartItem key={item.id} {...item} />
      ))}
    </div>
    <div className="cart-footer">
      <div className="cart-details">
        <span>
          Всего пицц: <b>{totalCount}</b> шт.</b>
        </span>
        <span>
          Сумма заказа: <b>{totalPrice}</b> ₺</b>
        </span>
      </div>
      <div className="cart-buttons">
        <Link to="/" className="button button--outline button--add
go-back-btn">
          <svg
            width="8"
            height="14"
            viewBox="0 0 8 14"
            fill="none"
            xmlns="http://www.w3.org/2000/svg"
          >
            <path
              d="M7 13L1 6.93015L6.86175 1"
              stroke="#D3D3D3"
              strokeWidth="1.5"
              strokeLinecap="round"
              strokeLinejoin="round"
            ></path>
          </svg>
          <span>Вернуться назад</span>
        </Link>
        <div className="button pay-btn">
          <span>Оплатить сейчас</span>
        </div>
      </div>
    </div>
  </div>
</div>
</MainLayout>
);
};

export default Cart;

import React from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';
import { useParams, useNavigate } from 'react-router-dom';
import { API_URL } from "../consts";
import { Pizza } from "../redux/pizza/types";
import MainLayout from '../layouts/MainLayout';

```

```

const FullPizza: React.FC = () => {
  const [pizza, setPizza] = React.useState<Pizza>();

  const { id } = useParams();
  const navigate = useNavigate();

  React.useEffect(() => {
    async function fetchPizza() {
      try {
        const { data } = await axios.get(`${API_URL}/pizza/${id}`);
        setPizza(data);
      } catch (error) {
        alert('Ошибка при получении пиццы!');
        navigate('/');
      }
    }

    fetchPizza();
  }, []);

  if (!pizza) {
    return <>Загрузка...</>;
  }

  return (
    <MainLayout>
      <div className="pizza-container">
        <img src={pizza.imageUrl} className="pizza-image"/>
        <h2>{pizza.title}</h2>
        <h4>{pizza.price} ₺</h4>
        <Link to="/">
          <button className="button button--outline button--add">
            <span>Назад</span>
          </button>
        </Link>
      </div>
    </MainLayout>
  );
};

export default FullPizza;

import React from 'react';
import { PizzaBlock, Skeleton } from '../components';

import { useSelector } from "react-redux";
import { selectPizzaData } from "../redux/pizza/selectors";
import { fetchPizzas } from "../redux/pizza/asyncActions";
import { useAppDispatch } from "../redux/store";
import MainLayout from "../layouts/MainLayout";

const Home: React.FC = () => {
  const dispatch = useAppDispatch();

  const { items, status } = useSelector(selectPizzaData);

  const getPizzas = async () => {

```

```

    dispatch(fetchPizzas());
    window.scrollTo(0, 0);
  };

  React.useEffect(() => {
    getPizzas();
  }, []);

  const pizzas = items.map((obj: any) => <PizzaBlock key={obj.id} {...obj} />);
  const skeletons = [...new Array(6)].map( (_, index) => <Skeleton key={index} />);

  return (
    <MainLayout>
      <div className="pizza-container">
        <h2 className="pizza-title">Все пиццы</h2>
        {status === 'error' ? (
          <div className="pizza-error-info">
            <h2>Произошла ошибка 😞</h2>
            <p>К сожалению, не удалось получить пиццы. Попробуйте повторить попытку позже.</p>
          </div>
        ) : (
          <div className="pizza-items">{status === 'loading' ? skeletons : pizzas}</div>
        )}
      </div>
    </MainLayout>
  );
};

export default Home;

import React from 'react';
import { NotFoundBlock } from '../components';
import MainLayout from "../layouts/MainLayout";

const NotFound: React.FC = () => (
  <MainLayout>
    <NotFoundBlock />
  </MainLayout>
);

export default NotFound;

import axios from "axios";
import styles from "../SignUp.module.scss";
import React from "react";
import { Input } from "../../components/Input/Input";
import { API_URL } from "../../consts";
import { Button } from "../../components/Button/Button";
import { Link, useNavigate } from "react-router-dom";
import { setIsAuth } from "../../redux/auth/slice";
import { useAppDispatch } from "../../redux/store";

export const SignUp = () => {

```

```

const navigate = useNavigate();
const dispatch = useAppDispatch();

const [email, setEmail] = React.useState('');
const [password, setPassword] = React.useState('');

const signup = async () => {
  const { data } = await axios.post(`${API_URL}/auth/signup`, {
    email,
    password
  });

  if (data.token) {
    localStorage.setItem('pizza-access-token', data.token);
    dispatch(setIsAuth(true));
    navigate('/');
  }
}

return (
  <div className={styles.signupContainer}>
    <form className={styles.signupForm} onSubmit={(e) => {
      e.preventDefault()
      signup();
    }}>
      <h2 className={styles.signupTitle}>Регистрация</h2>
      <div>
        <Input placeholder='Введите почту...' value={email}
onChange={(e) => setEmail(e.target.value)} />
      </div>
      <div>
        <Input type='password' placeholder='Введите пароль...'
value={password} onChange={(e) => setPassword(e.target.value)} />
      </div>
      <div>
        <Button type="submit">Зарегистрироваться</Button>
      </div>
      <div>
        <Link to={'/login'}>Уже есть аккаунт? Авторизоваться</Link>
      </div>
    </form>
  </div>
);
}

import axios from "axios";
import styles from "../Login.module.scss";
import React from "react";
import { Input } from "../../components/Input/Input";
import { API_URL } from "../../consts";
import { Button } from "../../components/Button/Button";
import { Link, useNavigate } from "react-router-dom";
import { setIsAuth } from "../../redux/auth/slice";
import { useAppDispatch } from "../../redux/store";

export const Login = () => {
  const navigate = useNavigate();
  const dispatch = useAppDispatch();

```

```

const [email, setEmail] = React.useState('');
const [password, setPassword] = React.useState('');

const fn = () => {
  axios.post(`${API_URL}/pizza`, {
    title: 'Пицца с сыром',
    price: 540,
    imageUrl: 'https://gas-kvas.com/grafic/uploads/posts/2023-09/1695870651_gas-kvas-com-p-kartinki-pitstva-1.jpg',
    sizes: [24, 36, 48],
    types: ['тонкое', 'традиционное'],
    rating: 4
  });
}

const login = async () => {
  const { data } = await axios.post(`${API_URL}/auth/login`, {
    email,
    password
  });
  if (data.token) {
    localStorage.setItem('pizza-access-token', data.token);
    dispatch(setIsAuth(true));
    navigate('/');
  }
}

return (
  <div className={styles.loginContainer}>
    <form className={styles.loginForm} onSubmit={(e) => {
      e.preventDefault()
      login();
    }}>
      <h2 className={styles.loginTitle}>Логин</h2>
      <div>
        <Input placeholder='Введите почту...' value={email}
onChange={(e) => setEmail(e.target.value)} />
      </div>
      <div>
        <Input type='password' placeholder='Введите пароль...'
value={password} onChange={(e) => setPassword(e.target.value)} />
      </div>
      <div>
        <Button type="submit">Авторизоваться</Button>
      </div>
      <div>
        <Link to={'/signup'}>Нет аккаунта? Зарегистрироваться</Link>
      </div>
    </form>
  </div>
);
}

```