МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ижевский государственный технический университет имени М.Т. Калашникова» (ФГБОУ ВО «ИжГТУ имени М.Т. Калашникова»)

Институт «Информатика и вычислительная техника»

Кафедра «Автоматизированные системы обработки информации и управления»

Работа защищена с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Программная инженерия»

на тему «Автоматизация деятельности фирмы по оказанию шиномонтажных услуг»

Выполнили

студенты гр. Б23-781-2 Севрюгин Е. Ю., Осинцева П. О., Багиров И.Ш.

Принял

к.т.н., доцент каф. АСОИУ Касимов Д.Р.

Рецензия:

степень достижения поставленной цели работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полнота разработки темы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

уровень самостоятельности работы обучающегося\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

недостатки работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ижевск

2024

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc185274567)

[ВВЕДЕНИЕ 3](#_Toc185274568)

[1 Разработка спецификации требований 4](#_Toc185274569)

[1.1 Требования 4](#_Toc185274570)

[1.2 User Story 5](#_Toc185274571)

[1.3 Use Cases 6](#_Toc185274572)

[2 Планирование программного проекта по методологии Scrum 11](#_Toc185274573)

[2.1 Mindmap проекта 11](#_Toc185274574)

[2.2 Команда разработчиков 12](#_Toc185274575)

[2.3 Постановка задач для команды разработки 13](#_Toc185274576)

[2.4 Диаграмма Ганта 18](#_Toc185274577)

[2.5 Стоимость проекта 19](#_Toc185274578)

[3 Проектирование программного продукта 20](#_Toc185274579)

[3.1 UML-диаграмма состояний 20](#_Toc185274580)

[3.2 UML-диаграмма последовательности 21](#_Toc185274581)

[3. 3 UML-диаграмма классов 23](#_Toc185274582)

[4 Описание репозитория 25](#_Toc185274583)

[5 Оценка качества программного продукта 28](#_Toc185274584)

[5.1 Описание багов 28](#_Toc185274585)

[5.2 Функциональные и нефункциональные требования 30](#_Toc185274586)

[5.3 Функциональные тесты и структура приложения 32](#_Toc185274587)

[5.4 Оценка качества ПО по пунктам 33](#_Toc185274588)

[ЗАКЛЮЧЕНИЕ 37](#_Toc185274589)

[Приложение А 38](#_Toc185274590)

[Приложение Б 39](#_Toc185274591)

# ВВЕДЕНИЕ

В условиях стремительного развития технологий и цифровизации процессы автоматизации становятся важнейшим инструментом для повышения эффективности работы предприятий. В сфере оказания услуг, включая шиномонтаж, автоматизация позволяет оптимизировать управление заказами, повысить уровень сервиса и минимизировать человеческий фактор в обработке данных.

Необходимость разработки данного приложения обусловлена высокой конкуренцией на рынке и увеличивающимися требованиями клиентов к скорости и качеству обслуживания. Применение автоматизированных решений позволяет не только упростить выполнение рутинных задач, но и сделать деятельность компании более прозрачной и управляемой. Это особенно актуально для компаний, работающих с большим объемом данных, такими как шиномонтажные сервисы, где требуется оперативная обработка информации о заказах, учете деталей и работе сотрудников.

Целью данной курсовой работы является повышение эффективности деятельности шиномонтажной компании путем разработки и внедрения автоматизированной системы управления. Предполагается, что использование данного приложения сократит время на обработку заявок, упростит учет складских запасов и улучшит взаимодействие между сотрудниками, что приведет к повышению уровня удовлетворенности клиентов и росту конкурентоспособности компании.

В ходе выполнения курсовой работы использовались методологии проектирования программного обеспечения, такие как Scrum, что позволило организовать процесс разработки эффективно и структурированно. Полученные результаты демонстрируют, что внедрение предложенной автоматизированной системы способствует значительному улучшению ключевых бизнес-показателей фирмы.

## 1 Разработка спецификации требований

Первый этап разработки программного продукта включает в себя сбор, анализ и формализацию требований к создаваемой системе. Эта работа является ключевой, поскольку она задает основу для дальнейшего проектирования и реализации программного обеспечения.

### 1.1 Требования

Общие функциональные требования:

1. Личный кабинет для сотрудника.
2. Личный кабинет для клиента.
3. Окно для составления заявки клиента.
4. Система для сотрудников.

Общие нефункциональные требования:

1. Приложение должно работать на мобильном телефоне, на платформе Android 8.0 и IOS 16.0.0 и новее.
2. Приложение должно быть доступно 99.9% времени в год.
3. Система должна быть интуитивно понятна.
4. Приложение должно обрабатывать запросы пользователей не более чем за 2 секунды при стандартной нагрузке.
5. Поддержка одновременной работы минимум 100 пользователей без снижения скорости.
6. Ежедневное автоматическое резервное копирование данных.

Функциональные требования для сотрудника:

1. При заходе в приложение нам предлагают создать профиль или войти в уже существующий профиль.
2. На главном экране должна отображаться таблица, содержащая следующие данные (в указанном порядке):
   1. Номер заказа (целое число).
   2. Дата и время заказа, до какого времени он должен быть выполнен (в формате dd:hh:mm).
   3. Номер машины (например, A-000-АA).
   4. Требования заказа.
   5. Стоимость работы.
3. Четные строки таблицы выделяем серым, нечётные не выделяются цветом.
4. На главном экране отображается не более 8 заказов в день (по часу на заказ). Если строка пустая - написать, что на данное время заказов нет.
5. На главном экране отображается информер, содержащий текущие день недели и время.

Функциональные требования для клиента:

1. На главном экране должны отображаться следующие поля для ввода (в указанном порядке):
   1. Фамилия Имя Отчество.
   2. Номер телефона.
   3. Регистрационный номер авто.
   4. Проблема.
2. На главном экране, после полей для ввода, должна отображаться кнопка «Отправить заявку».
3. При отправлении заявки, в течение 5 секунд отображается уведомление «Заявка успешно отправлена».
4. На главном экране отображается кнопка для перехода в профиль, где нам предлагают создать его или войти в уже существующий профиль.
5. В окне профиля должны отображаться следующие кнопки (в указанном порядке):
   1. История заказов.
   2. Бонусная программа.
   3. Поддержка.
   4. Настройки приложения.
   5. Условия использования.
   6. Тёмная тема.
   7. Оставить заявку.

### 1.2 User Story

Для клиента:

1. Я, как Клиент, хочу иметь возможность легко оставить заявку, чтобы быстро исправить проблему.
2. Я, как Клиент, хочу видеть историю своих заявок, чтобы оценивать свои расходы.
3. Я, как Клиент, хочу иметь бонусную программу, чтобы получать скидки на последующие обращения.

Для сотрудника:

1. Я, как Сотрудник, хочу видеть расписание, чтобы распланировать свой рабочий день.
2. Я, как Сотрудник, хочу видеть наличие запчастей на складе, чтобы подготовиться к выполнению заказа.
3. Я, как Сотрудник, хочу видеть стоимость работы, чтобы оценивать свою прибыль.

### 1.3 Use Cases

Таблица 1.3.1 – Содержание Use Case 1

|  |  |  |
| --- | --- | --- |
| Название | Просмотр сотрудником расписания на завтрашний рабочий день | |
| Описание | Пользователь может зайти в приложение и на главном экране перейти из сегодняшнего дня на завтрашний рабочий день | |
| Акторы | Сотрудник | |
| Предусловия | Пользователь открыл приложение | |
| Основной поток событий | 1. Открыть приложение. 2. Посмотреть на сегодняшний день. 3. Нажать кнопку далее. | |
| Постусловия | Отображаем расписание на завтра |

Продолжение таблицы 1.3.1

|  |  |
| --- | --- |
| Диаграмма |  |

Таблица 1.3.2 – Содержание Use Case 2

|  |  |
| --- | --- |
| Название | Клиент оставляет заявку |
| Описание | Пользователь может зайти в приложение и увидеть на главном экране поля ввода для оставления заявки |
| Акторы | Клиент |
| Предусловия | Пользователь открыл приложение |
| Основной поток событий | 1. Открыть приложение. 2. Заполнить поля: регистрационный номер авто, проблема. 3. Нажать кнопку отправить заявку. 4. В течение 5 секунд отображается уведомление «Заявка успешно отправлена». |

Продолжение таблицы 1.3.2

|  |  |
| --- | --- |
| Альтернативный поток событий (Пользователь не авторизирован в системе) | 1. Открыть приложение. 2. Заполнить поля: ФИО, номер телефона, регистрационный номер авто, проблема. 3. Нажать кнопку отправить заявку. 4. В течение 5 секунд отображается уведомление «Заявка успешно отправлена». |
| Постусловия | Клиент успешно отправляет заявку |
| Диаграмма |  |

Таблица 1.3.3 – Содержание Use Case 3

|  |  |
| --- | --- |
| Название | Клиент смотрит бонусную программу |
| Описание | Пользователь смотрит размер своей персональной скидки, а также баллы |
| Акторы | Клиент |

Продолжение таблицы 1.3.3

|  |  |
| --- | --- |
| Предусловия | Пользователь открыл приложение |
| Основной поток событий | 1. Открыть приложение. 2. Нажать на профиль. 3. Нажать кнопку «Бонусная программа». |
| Постусловия | Пользователь видит свою скидку и бонусные баллы |
| Диаграмма |  |

На рисунке 1.3.1 изображен прототип интерфейса, разработанный нами в программе Figma.

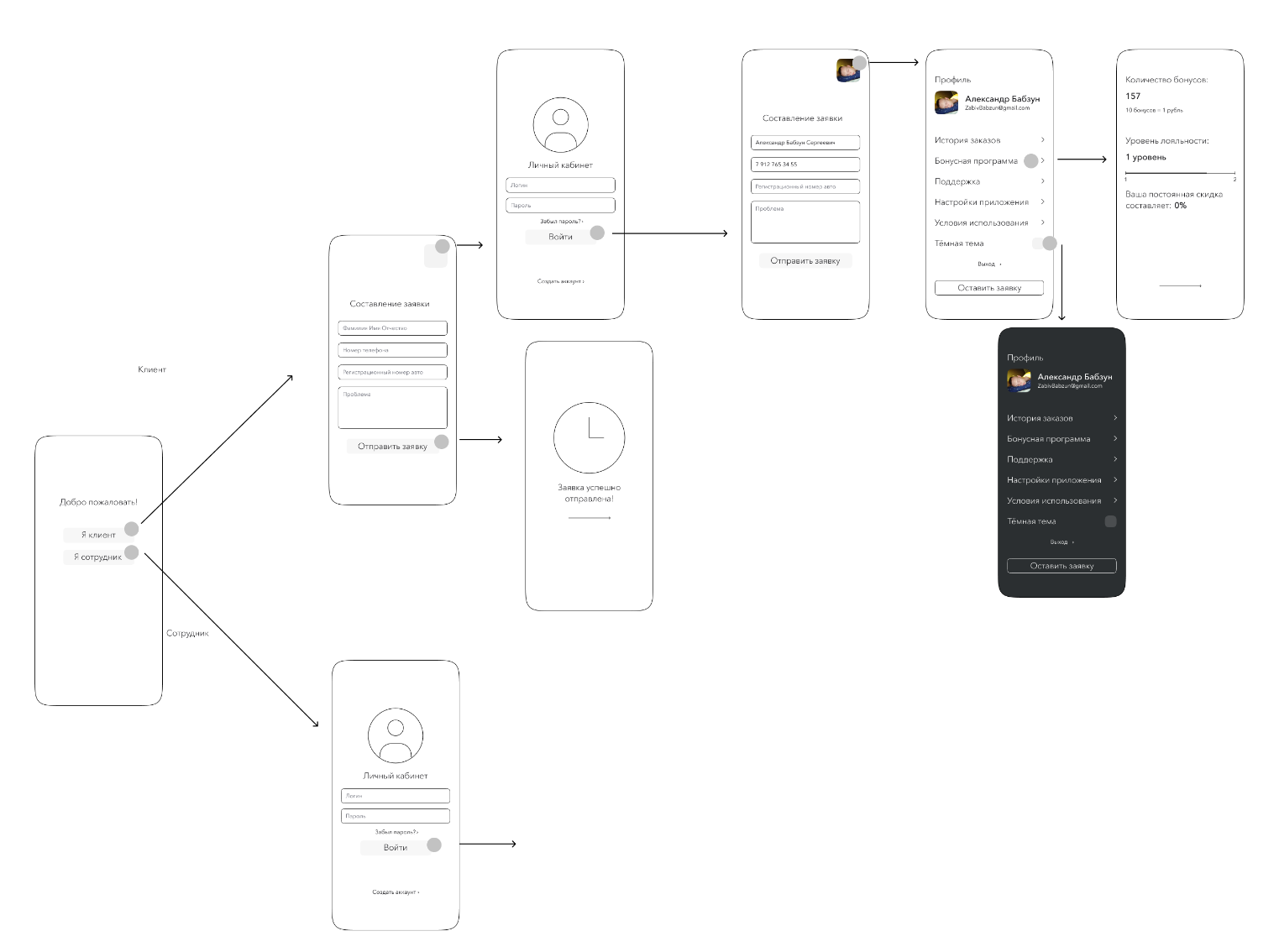


Рисунок 1.3.1 – Прототипирование интерфейса

На данном этапе была выполнена разработка спецификации требований и черновой вариант первых экранов мобильного приложения. Это позволило:

1. Определить ключевые функции и характеристики будущей системы.
2. Структурировать задачи для дальнейшего проектирования и реализации.
3. Согласовать ожидания всех заинтересованных сторон, включая клиентов и сотрудников.

Это позволило нам обеспечить четкое понимание того, каким должен быть конечный продукт и как он будет взаимодействовать с пользователями.

## 2 Планирование программного проекта по методологии Scrum

На этом этапе мы перешли к созданию структуры проекта, включая разработку MindMap, постановку задач для команды и формирование состава команды. Этот процесс позволяет определить ключевые направления работы, оценить трудоемкость и ресурсы, необходимые для запуска проекта.

### 2.1 Mindmap проекта

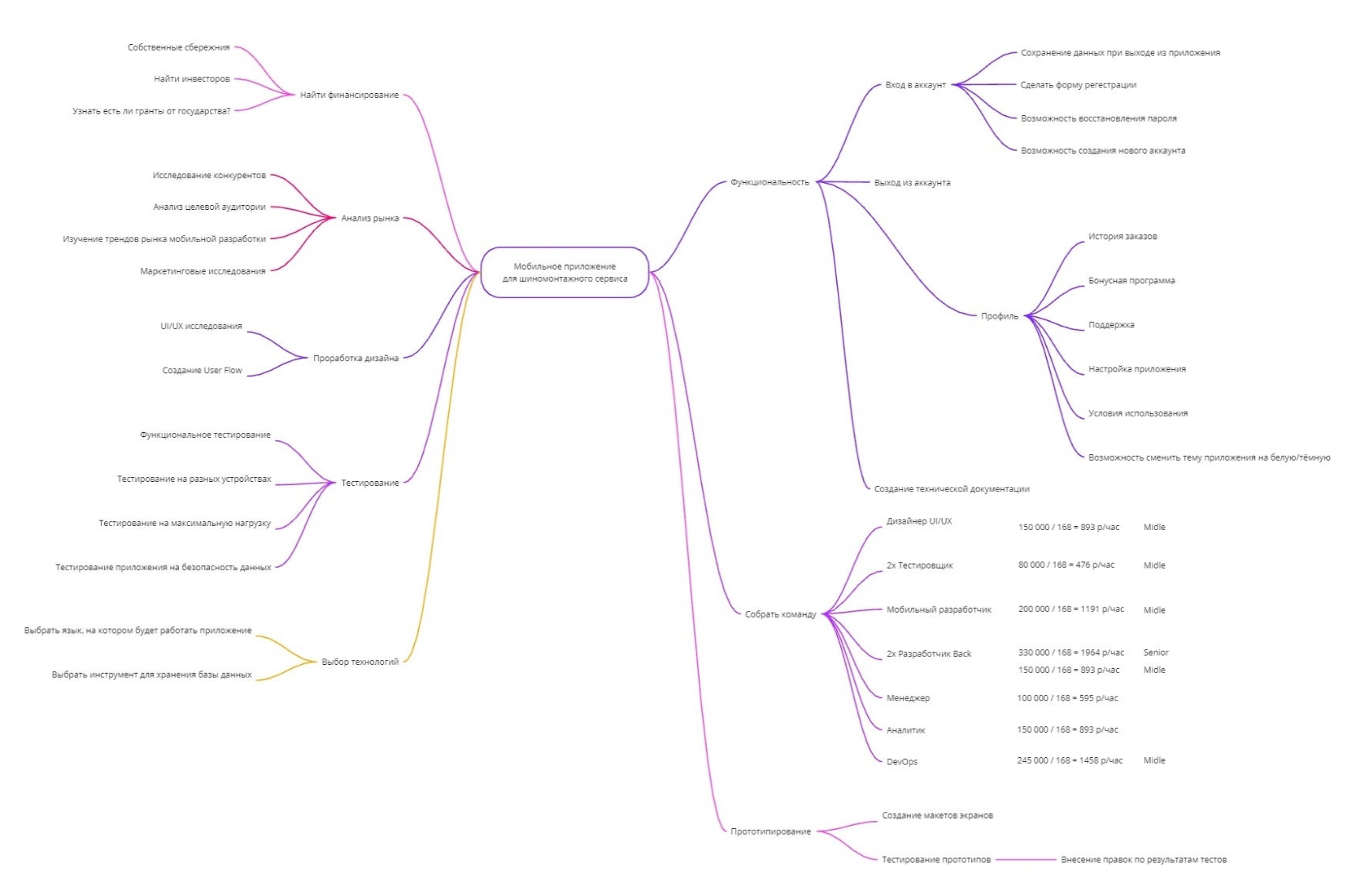


Рисунок 2.1.1 – Mindmap проекта

Для планирования был разработан MindMap, который отражает основные этапы реализации проекта мобильного приложения для шиномонтажного сервиса.

На схеме выделены следующие основные направления:

1. Функциональность. Включает ключевые функции приложения, такие как регистрация и вход в аккаунт, управление профилем (история заказов, бонусная программа, настройки). Это позволяет создать удобный интерфейс для пользователей и сотрудников.
2. Прототипирование. Создание макетов экранов приложения и их тестирование. Этот этап позволяет заранее выявить недочеты в проектировании пользовательского интерфейса.
3. Анализ рынка. Проведение исследований, включая анализ конкурентов, изучение целевой аудитории, исследование трендов мобильной разработки. Эти данные обеспечивают адаптацию продукта к требованиям рынка.
4. Тестирование. Включает функциональное тестирование, нагрузочное тестирование и проверку безопасности данных. Эти процессы помогают обеспечить надежность и стабильность работы приложения.
5. Выбор технологий. Определение используемых языков программирования, инструментов для хранения данных и других технических решений. Это закладывает основу для реализации приложения.
6. Финансирование. Анализ возможных источников финансирования: собственные средства, привлечение инвесторов, получение грантов.
7. Проработка дизайна. На этапе проработки дизайна главная цель — обеспечить интуитивно понятный, эстетичный и функциональный интерфейс приложения.
8. Команда. Состав команды был определен исходя из специфики проекта, объема задач и доступного бюджета.
9. Финансирование. Финансирование проекта включает анализ возможных источников средств, необходимых для разработки и запуска приложения:

### 2.2 Команда разработчиков

Для успешной реализации проекта мобильного приложения для шиномонтажного сервиса была сформирована команда, состоящая из специалистов разного профиля. Команда сформирована с учетом объема задач, требований проекта и необходимости соблюдения баланса между качеством продукта и затратами.

UI/UX-дизайнер (1 Middle). Отвечает за создание удобного и понятного интерфейса приложения. Проект не требует сложных анимаций или специфических графических решений, что позволяет обойтись специалистом среднего уровня, сохраняя бюджет.

Бэкенд-разработчики (2 Senior). Обеспечивают разработку серверной части приложения, включая создание базы данных и API. Проект требует высокой надежности серверной части, особенно в части обработки заявок и резервного копирования данных. Senior-специалисты способны справиться с этими задачами быстрее и качественнее.

Фронтенд-разработчик (1 Middle). Реализует клиентскую часть приложения. Для реализации интерфейса проекта, не требующего сложных анимаций или дополнительных библиотек, достаточно разработчика среднего уровня.

Тестировщики (2 Junior). Проверяют приложение на наличие ошибок, проводят нагрузочное и функциональное тестирование. Простая структура приложения позволяет использовать начинающих специалистов, что снижает затраты без ущерба качеству.

Менеджер проекта (1 Middle). Координирует работу команды, контролирует выполнение задач и соблюдение сроков. Уровня опыта среднего менеджера достаточно для координации небольшого проекта, что позволяет оптимизировать бюджет.

Аналитик (1 Middle). Отвечает за сбор требований, анализ рынка и постановку задач для команды. Специалист среднего уровня способен справиться с исследованиями и задачами постановки требований без привлечения дополнительного эксперта.

Команда была сформирована на основании сложности и объема задач.

Роли, от которых отказались:

1. DevOps-инженер. В рамках проекта задачи по развертыванию системы на сервере были переложены на бэкенд-разработчиков, поскольку они обладают соответствующими компетенциями.
2. Маркетолог. Решено ограничиться исследованием рынка силами аналитика для снижения затрат.

Для расчета стоимости часа работы каждого специалиста были найдены средние зарплаты и пересчитаны на часовую оплату:

1. UI/UX-дизайнер: 150 000 руб/мес / 168 ч = 893 руб/ч
2. Бэкенд-разработчики: 330 000 руб/мес / 168 ч = 1964 руб/ч
3. Фронтенд-разработчик: 200 000 руб/мес / 168 ч = 1190 руб/ч
4. Тестировщики: 80 000 руб/мес / 168 ч = 476 руб/ч
5. Менеджер проекта: 100 000 руб/мес / 168 ч = 595 руб/ч
6. Аналитик: 150 000 руб/мес / 168 ч = 893 руб/ч

Итоговый состав команды и распределение ролей позволили оптимизировать процесс разработки и достичь баланса между качеством и затратами.

### 2.3 Постановка задач для команды разработки

На данном этапе работы над проектом был осуществлен переход от сформированных требований к постановке конкретных задач для команды разработки. Основной целью этого этапа является создание четкой и структурированной последовательности действий, которая обеспечит своевременную и качественную реализацию всех функциональных и нефункциональных требований приложения.

1. Разбиение требований на задачи. Каждое требование, сформулированное на предыдущем этапе, было преобразовано в конкретные задачи. При этом учитывались:
   1. Особенности функционала, которые требуют взаимодействия между несколькими ролями в команде (дизайн, бэкенд, фронтенд, тестирование).
   2. Приоритет выполнения задач: более критичные для работы приложения задачи были помечены как высокоприоритетные.
2. Создание планов задач и оценка времени. Для каждой задачи была определена трудоемкость (в человеко-часах). Это позволило:
   1. Определить общий объем работы.
   2. Оценить, сколько ресурсов и времени потребуется для выполнения задач, с учетом состава команды.
3. Использование подхода Agile. Для управления задачами и распределения нагрузки была выбрана гибкая методология управления проектами.
   1. Вся работа была разделена на этапы (спринты).
   2. Задачи с высоким приоритетом выносились на первые спринты, чтобы основные функции приложения были готовы на ранних этапах разработки.

Все задачи вы можете посмотреть на рисунках 2.3.1-.2.3.3.

****

Рисунок 2.3.1 – Задачи для разработчиков



Рисунок 2.3.2 – Задачи для разработчиков (Продолжение)



Рисунок 2.3.3 – Задачи для разработчиков (Продолжение)

Создание подробного плана задач позволило:

1. Обеспечить прозрачность работы для всех членов команды.
2. Свести к минимуму риски пропуска важных элементов функциональности.
3. Четко распределить нагрузку между разработчиками, тестировщиками и аналитиками.
4. Ускорить процесс разработки за счет параллельной работы над различными задачами.

### 2.4 Диаграмма Ганта

На этапе планирования разработки проекта была создана диаграмма Ганта, которая позволила визуализировать сроки выполнения задач и последовательность их реализации. Этот инструмент помогает структурировать процесс разработки, оценить временные рамки проекта и управлять ресурсами команды. Мы составили её, она изображена на рисунке 2.4.1.

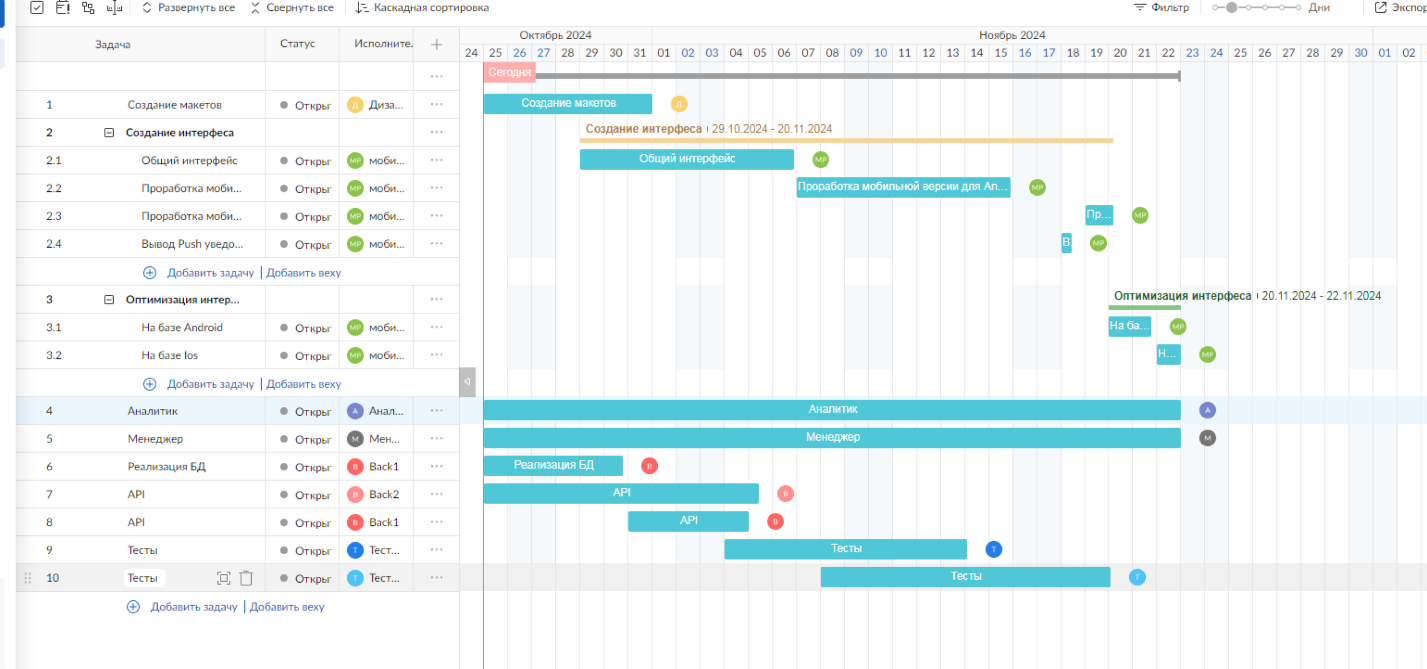


Рисунок 2.4.1 – Диаграмма Ганта

Цели использования диаграммы Ганта:

1. Контроль сроков выполнения: диаграмма позволила следить за ходом выполнения задач и своевременно выявлять отставания.
2. Оптимизация процессов: использование связей между задачами помогло минимизировать простои и грамотно распределить ресурсы.
3. Прогнозирование рисков: за счет четкого плана можно было заранее определить потенциальные проблемы в сроках и устранить их.

Созданная диаграмма Ганта обеспечила прозрачность процесса разработки для всех членов.

### 2.5 Стоимость проекта

В рамках планирования проекта была произведена оценка итоговой стоимости разработки мобильного приложения для шиномонтажного сервиса. Этот этап включал в себя анализ временных затрат на выполнение задач, расчет оплаты труда всех членов команды, см. Рисунок 2.5.1.

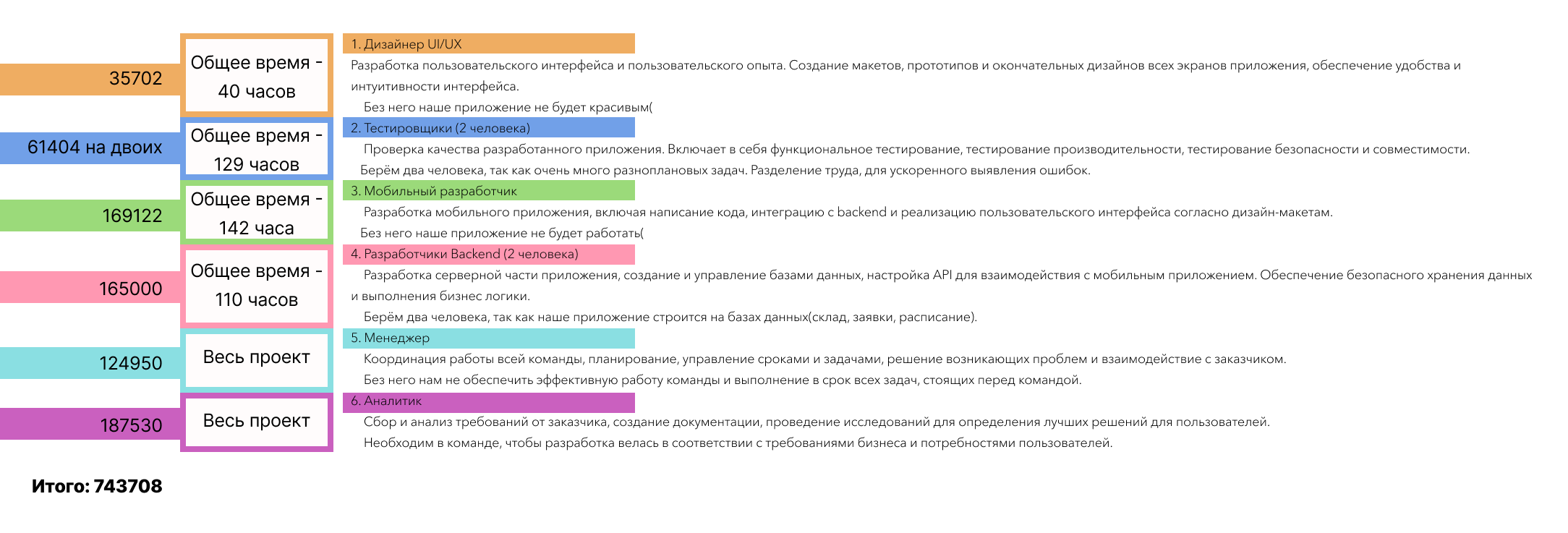
****

Рисунок 2.5.1 – Стоимость проекта

Стоимость нашего проекта составила – 734 708 Р.

Оценка стоимости позволила:

1. Получить точное представление о бюджете, необходимом для реализации проекта.
2. Оптимизировать распределение ресурсов, определив роли и часы работы специалистов с наибольшим вкладом.
3. Убедиться в экономической целесообразности проекта.

## 3 Проектирование программного продукта

В данной работе мы разработалиUML-диаграммы состояний, последовательности, классов.

### 3.1 UML-диаграмма состояний

В рамках лабораторной работы №4 была разработана диаграмма состояний, описывающая процесс отправки заявки через мобильное приложение. Этот инструмент визуализации используется для описания всех возможных состояний системы и переходов между ними в ходе выполнения определенной функциональности.

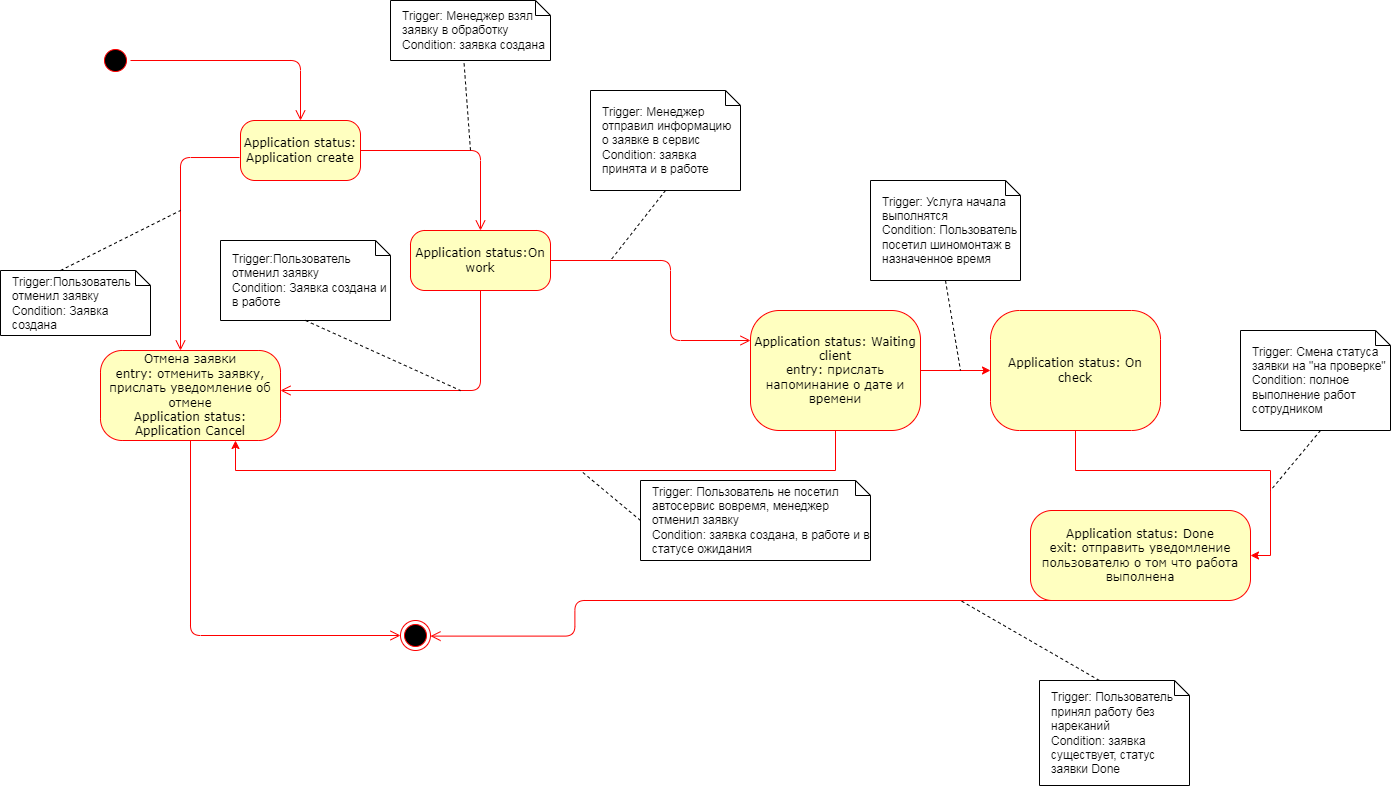
****

Рисунок 3.1.1 – Схема «Диаграмма состояний»

Разработанная диаграмма отражает процесс взаимодействия клиента с приложением для отправки заявки на шиномонтажные услуги. Она позволяет наглядно представить, как приложение реагирует на действия пользователя и какие события инициируют изменения в состоянии системы.

Создание диаграммы состояний позволило:

1. Структурировать процесс отправки заявки и предусмотреть возможные исключительные ситуации.
2. Определить ключевые точки взаимодействия пользователя с приложением.
3. Обеспечить разработчикам и тестировщикам четкое понимание всех состояний системы, что упрощает последующую реализацию и тестирование.

Диаграмма состояний является важным этапом проектирования, так как она демонстрирует логику работы приложения и помогает предвидеть потенциальные ошибки. Она станет основой для дальнейшей разработки и тестирования процесса отправки заявок.

### 3.2 UML-диаграмма последовательности

UML-диаграмма последовательности, которая иллюстрирует взаимодействие между основными компонентами системы при выполнении операции поиска детали на складе. Этот тип диаграммы помогает описать обмен сообщениями между пользователем, фронтенд-приложением, бэкендом, системой управления базами данных (СУБД) и системой обслуживания склада.

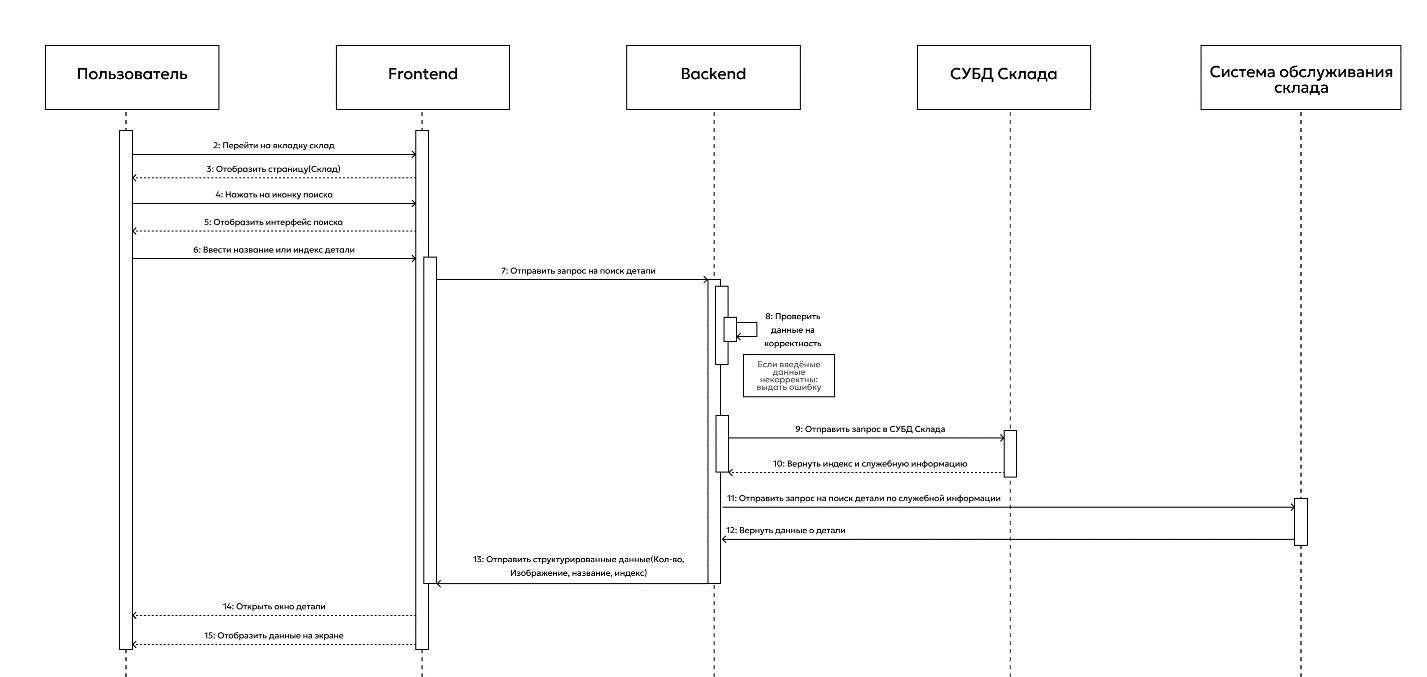
****

Рисунок 3.2.1 – Схема «UML-диаграмма последовательности»

Диаграмма последовательности описывает шаги взаимодействия с момента запроса пользователя до получения результата.

Основные участники:

1. Пользователь. Инициирует процесс поиска детали через интерфейс приложения.
2. Frontend. Отвечает за отображение данных и передачу запросов к бэкенду.
3. Backend. Обрабатывает данные, проверяет корректность ввода, выполняет запросы к СУБД и системе склада.
4. СУБД склада. Хранит и предоставляет информацию о деталях, включая их индекс и служебные данные.
5. Система обслуживания склада. Предоставляет данные о наличии и характеристиках деталей.

Рассмотрим основные действия данной диаграммы**:**

1. Действия пользователя:
   1. Пользователь переходит на вкладку «Склад» в приложении и нажимает на иконку поиска.
   2. Вводит название или индекс детали для поиска.
2. Работа фронтенда:
   1. Отображает интерфейс поиска.
   2. Передает запрос на бэкенд с введенными данными.
3. Работа бэкенда:
   1. Проверяет корректность данных, введенных пользователем.
   2. Если данные некорректны, возвращает уведомление об ошибке.
   3. Если данные корректны, отправляет запрос в СУБД для получения индекса и служебной информации о детали.
4. Действия СУБД склада:
   1. Выполняет запрос на поиск индекса и возвращает необходимые данные бэкенду.
5. Обращение к системе обслуживания склада:
   1. По служебной информации запрашиваются подробные данные о детали (количество, изображение, характеристики).
   2. Возвращает структурированную информацию бэкенду.
6. Отображение результата:
   1. Бэкенд передает данные фронтенду, который отображает подробное окно с информацией о детали на экране пользователя.

Разработанная UML-диаграмма последовательности является важным этапом проектирования, который обеспечивает ясность в описании процесса поиска деталей. Она будет использоваться для дальнейшего создания и тестирования соответствующего функционала приложения.

### 3. 3 UML-диаграмма классов

Данная диаграмма является основным инструментом проектирования объектно-ориентированных систем и визуализирует ключевые компоненты системы для оказания шиномонтажных услуг см. Рисунок 3.3.1.

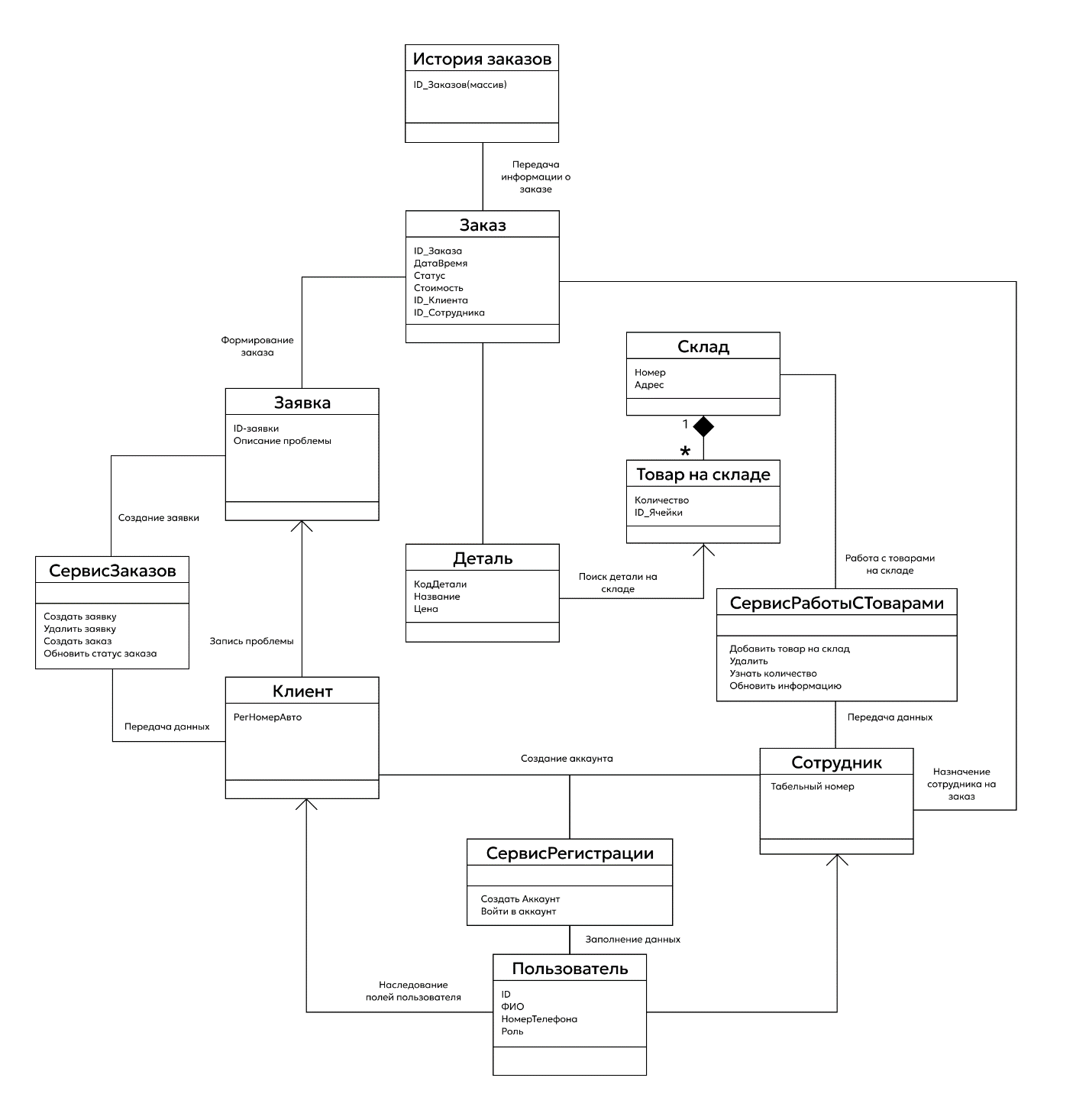


Рисунок 3.3.1 – Схема «Диаграмма классов»

Ключевые классы и их описание:

1. Клиент:
   1. Атрибуты. Регистрационный номер авто.
   2. Связи. Связан с классом Заявка, так как клиент инициирует создание заявки.
2. Пользователь:
   1. Атрибуты. ID, ФИО, номер телефона, роль.
   2. Особенность. Наследуется классом Клиент и Сотрудник.
3. Заявка:
   1. Атрибуты. ID заявки, описание проблемы.
   2. Функции. Позволяет описать задачу клиента, на основании которой формируется заказ.
   3. Связи:
      1. Формирует связь с классом СервисЗаказов.
      2. Позволяет организовать взаимодействие с клиентом.
4. СервисЗаказов:
   1. Создание заявки.
   2. Удаление заявки.
   3. Создание заказа.
   4. Обновление статуса заказа.
5. Заказ:
   1. Атрибуты. ID заказа, дата заказа, статус, стоимость, ID клиента, ID сотрудника.
   2. Связи:
      1. Информация из Заявки используется для формирования заказа.
      2. Связан с классом История заказов, в которую передается информация о завершенных заказах.
6. История заказов:
   1. Атрибуты. ID заказа/насоса (для идентификации выполненных работ).
7. Сотрудник:
   1. Атрибуты. Табельный номер.
   2. Связи. Назначается для работы над заказом.
8. Деталь:
   1. Атрибуты. Код детали, название, цена.
   2. Функционал. Используется для указания, какие детали необходимы для выполнения заказа.
9. Склад:
   1. Атрибуты. Номер склада, адрес.
   2. Связи:
      1. Содержит Товары на складе.
      2. Поддерживает взаимодействие с классом СервисРаботыСТоварами.
10. Товар на складе:
    1. Атрибуты. Количество, ID\_ячейки.
    2. Связи:
       1. Содержит информацию о запасах деталей.
       2. Поддерживает операции добавления, удаления и обновления информации о товарах.
11. СервисРаботыСТоварами:
    1. Добавление товара на склад.
    2. Удаление товара.
    3. Учет количества.
    4. Обновление информации.
12. СервисРегистрации:
    1. Функционал:
       1. Создание аккаунта.
       2. Вход в аккаунт.
    2. Связи. Обеспечивает взаимодействие с классом Пользователь.

Созданная диаграмма классов позволила:

1. Четко определить объекты системы и их атрибуты.
2. Спроектировать связи между сущностями для обеспечения функциональной логики системы.
3. Подготовить основу для дальнейшей реализации кода на основе объектно-ориентированного подхода.

## 4 Описание репозитория

Репозиторий проекта хранится в системе контроля версий Git, что обеспечивает возможность работы нескольких разработчиков, отслеживание изменений и управление версиями.

В репозитории создано две ветки:

1. main — основная ветка, в которой хранится стабильная версия проекта.
2. New\_Files — ветка для разработки и добавления новых функций, включая лабораторные работы и тестовый код.

Репозиторий включает следующие файлы и папки, структурированные по аналогии с проектом, описанным ранее.

TireServiceApp — папка с кодом приложения. В ней содержатся следующие компоненты:

1. Models: классы данных, включая Application.cs, Client.cs, и User.cs.
2. Repositories: интерфейсы и их реализации (IApplicationRepository.cs, MockApplicationRepository.cs и другие).
3. Services: папка с бизнес-логикой (ApplicationService.cs, RegistrationService.cs).
4. Програмный файл Program.cs и настройки проекта (appsettings.json).
5. Tests — модульные тесты, такие как проверки для классов ApplicationServiceTests, реализованные с использованием моковых данных.

На рисунке 4.1 изображен репозитории, где хранятся файлы лабораторных работ, упорядоченные в соответствии с номерами: Лабораторная работа №1–7: каждая из них содержит код или результаты работы, связанные с изучением конкретных аспектов разработки программного обеспечения (проектирование, реализация и тестирование).

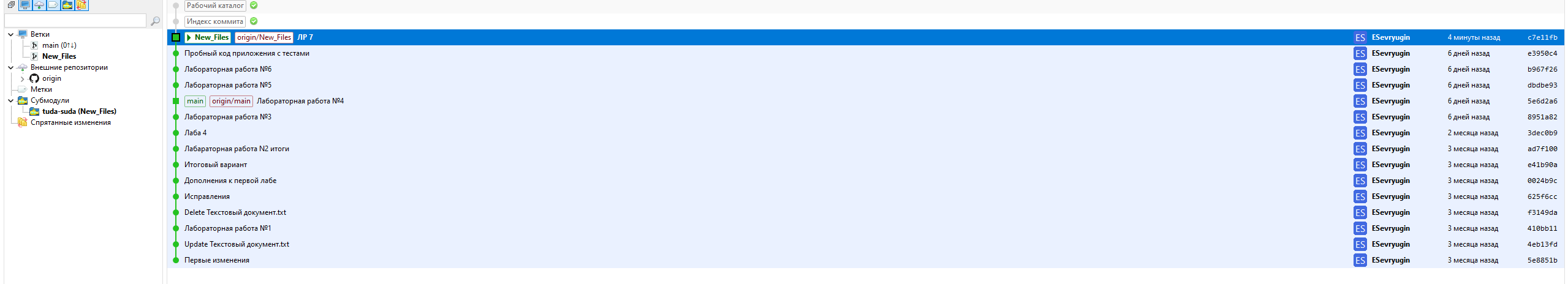


Рисунок 4.1 – Фрагмент истории коммитов репозитория

На рисунке 4.2 изображён итоговый репозиторий на сайте GitHub

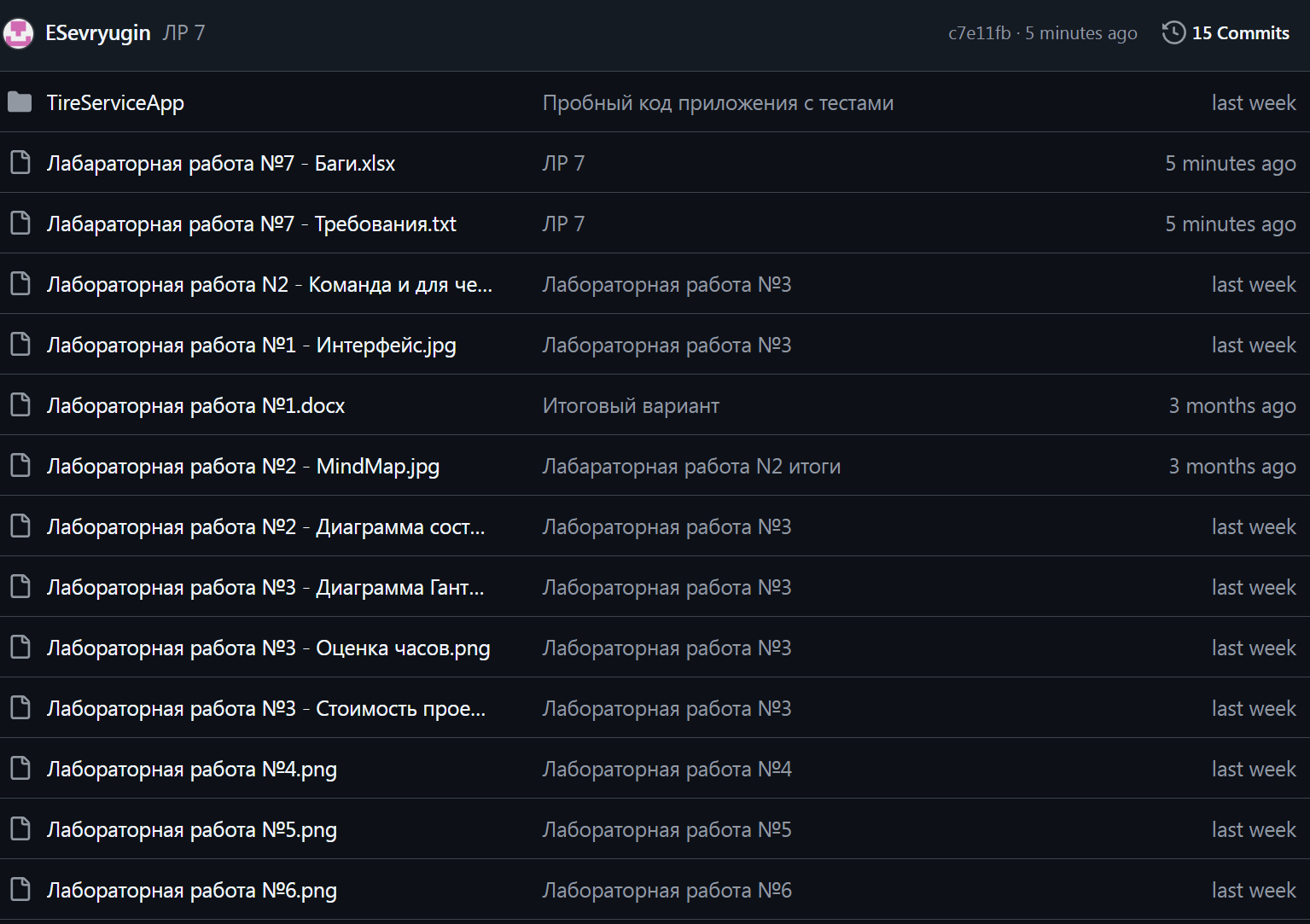


Рисунок 4.2 – Структура репозитория

## 5 Оценка качества программного продукта

### 5.1 Описание багов

Каждый дефект был детально описан, включая ожидаемое и фактическое поведение, воспроизводимость, приоритет, серьезность и шаги воспроизведения. Также была проведена оценка времени, необходимого на диагностику, исправление и тестирование дефектов, и назначены ответственные разработчики.

Критический дефект

Первый дефект относится к категории критических, так как он нарушает ключевую функциональность приложения — отправку заявок. Этот баг воспроизводится постоянно, что делает его устранение приоритетной задачей. Без его исправления работа приложения невозможна, что напрямую влияет на пользователей и бизнес-процессы.

Высокоприоритетный дефект

Второй дефект также имеет высокий приоритет, так как он блокирует возможность сотрудников видеть актуальные заказы. Хотя этот баг не критичен для всех пользователей, его наличие создает проблемы для выполнения внутренних операций компании, что снижает эффективность работы.

Низкоприоритетный дефект

Третий дефект связан с интерфейсом приложения и не оказывает влияния на основные функции. Ошибка проявляется в несохранении настроек тёмной темы. Этот баг классифицирован как низкоприоритетный, так как его исправление не является срочным и критичным для работы системы.

Все дефекты были оформлены в соответствии с установленным шаблоном. Для каждого из них определены сроки исправления и назначены члены команды, ответственные за их устранение. Ожидается, что исправление дефектов позволит повысить стабильность и удобство использования приложения, а также улучшить пользовательский опыт.

Таблица 5.1.1 – Баги

|  |  |  |  |
| --- | --- | --- | --- |
| **Идентификатор** | **Краткое описание** | **Подробное описание** | **Шаги воспроизведения** |
| 1 | Аварийное завершение приложения при отправке заявки | При попытке отправить заявку приложение аварийно завершает работу. | 1. Открыть приложение. 2. Заполнить все поля для заявки (ФИО, номер телефона, регистрационный номер авто, проблема). 3. Нажать кнопку 'Отправить заявку'. Ожидаемый результат: Уведомление 'Заявка успешно отправлена' появляется в течение 5 секунд. Фактический результат: Приложение закрывается без сохранения заявки. |
| 2 | Некорректное отображение заказов у сотрудника | На главном экране у сотрудника таблица заказов пустая, хотя заказы есть в базе данных. | 1. Открыть приложение под учетной записью сотрудника. 2. Проверить таблицу заказов. Ожидаемый результат: Таблица отображает актуальные заказы. Фактический результат: Таблица пустая или отображает ошибочные данные. |
| 3 | Настройки тёмной темы не сохраняются | После включения тёмной темы настройки не сохраняются, и приложение запускается в светлой теме. | 1. Открыть приложение. 2. Перейти в профиль клиента. 3. Включить 'Тёмную тему'. 4. Закрыть и снова открыть приложение. Ожидаемый результат: Тёмная тема остаётся активной. Фактический результат: Приложение запускается в светлой теме. |

Таблица 5.1.2 – Комментарии к багам

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Воспроизводимость** | **Важность** | **Срочность** | **Симптом** | **Возможность обойти** | **Комментарий** |

Продолжение таблицы 5.1.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Всегда | Высокая | Критическая | Некорректная работа | Нет | Дефект нарушает ключевую функциональность — отправку заявок клиентами. Диагностика: 2–3 часа. Исправление: 6–8 часов. Тестирование: 2–3 часа. Итого: 1 рабочий день. Назначение: ведущий backend-разработчик. |
| Всегда | Высокая | Высокая | Некорректное отображение данных | Нет | Блокирует работу сотрудников. Диагностика: 2 часа. Исправление: 4–6 часов. Тестирование: 2 часа. Итого: 1 рабочий день. Назначение: backend-разработчик. |
| Всегда | Низкая | Низкая | Ошибка в настройках | Да | Не влияет на основные функции приложения. Исправление: 3–4 часа. Тестирование: 1 час. Итого: 0.5 рабочего дня. Назначение: frontend-разработчик. |

### 5.2 Функциональные и нефункциональные требования

Общие функциональные требования:

1. Личный кабинет для сотрудника:
   1. Проверить возможность создания нового профиля.
   2. Проверить возможность входа в существующий профиль.
   3. Убедиться, что данные профиля отображаются корректно.
2. Личный кабинет для клиента:
   1. Проверить возможность создания нового профиля.
   2. Проверить возможность входа в существующий профиль.
   3. Убедиться в корректности отображения данных клиента.
3. Окно для составления заявки клиента:
   1. Проверить наличие всех обязательных полей для ввода:
      1. ФИО
      2. Номер телефона
      3. Регистрационный номер авто
      4. Проблема
   2. Проверить кнопку "Отправить заявку" на функциональность.
4. Система для сотрудников:
   1. Убедиться в доступности всех требуемых данных (расписание, заказы, запчасти).

Общие нефункциональные требования:

1. Поддержка платформ:
   1. Проверить работоспособность приложения на Android 8.0+ и IOS 16.0+.
2. Доступность:
   1. Убедиться, что приложение доступно в 99.9% случаев при тестировании в различных условиях.
3. Интуитивность интерфейса:
   1. Провести тестирование на юзабилити, чтобы убедиться, что интерфейс понятен.
4. Время обработки запросов:
   1. Замерить время обработки запросов (не более 2 секунд при стандартной нагрузке).
5. Нагрузка:
   1. Проверить стабильность работы приложения при одновременном подключении 100 пользователей.
6. Резервное копирование:
   1. Убедиться, что данные ежедневно резервируются автоматически.

### 5.3 Функциональные тесты и структура приложения

Приложение TireServiceApp разработано с использованием многоуровневой архитектуры, что обеспечивает модульность и простоту тестирования (см. Приложение Б)

Основные папки проекта:

1. Models. Содержит классы данных:
   1. Application.cs: описывает заявку клиента, включая ФИО, номер телефона, регистрационный номер автомобиля и описание проблемы.
   2. Client.cs и User.cs: представляют модель клиента и пользователя системы.
2. Repositories. Содержит интерфейсы и их реализации для работы с данными:
   1. IUserRepository.cs и MockUserRepository.cs: интерфейс и его тестовая реализация для управления данными пользователей.
   2. IApplicationRepository.cs и MockApplicationRepository.cs: интерфейс и моковая реализация для обработки заявок.
3. Services. Содержит бизнес-логику приложения:
   1. ApplicationService.cs: управляет обработкой заявок, включая валидацию и сохранение данных.
   2. RegistrationService.cs: предназначен для управления регистрацией пользователей.
   3. MockApplicationRepository.cs: предоставляет тестовые данные.
4. Tests. Содержит модульные тесты, позволяющие проверить корректность работы сервисов. В тестах используются тестовые репозитории, что исключает необходимость в реальной базе данных.

Обоснование выбранной структуры:

1. Модульность. Разделение на уровни (модели, репозитории, сервисы) упрощает сопровождение и расширение приложения.
2. Тестовые репозитории. Позволили ускорить процесс разработки и тестирования, исключая работу с реальными данными на начальном этапе.
3. Гибкость. Интерфейсы репозиториев позволяют в будущем легко заменить тестовые данные на реальные (например, подключение базы данных).
4. Тестируемость. Бизнес-логика вынесена в сервисы, что упростило написание модульных тестов.

Описание тестов (скриншоты кода см. в Приложении Б):

1. Should\_SaveApplication\_When\_ValidDataProvided
   1. Цель. Проверить успешное сохранение заявки при корректных данных.
   2. Действия. Передача валидного объекта заявки в сервис.
   3. Ожидаемый результат. Метод SaveApplication возвращает true.
2. Should\_ThrowException\_When\_InvalidPhoneNumber
   1. Цель. Проверить, что при вводе некорректного номера телефона возникает исключение.
   2. Действия. Передача заявки с некорректным номером телефона.
   3. Ожидаемый результат. Выбрасывается исключение WrongPhoneNumberException.
3. Should\_ThrowException\_When\_NullApplicationProvided
   1. Цель. Убедиться, что передача null в метод обработки заявки вызывает исключение.
   2. Действия. Передача null вместо объекта заявки.
   3. Ожидаемый результат. Выбрасывается исключение ArgumentNullException.

Причины выбора именно такого подхода к тестированию:

1. Изолированность. Использование тестовых репозиториев исключает влияние внешних факторов, таких как работа с базой данных.
2. Простота воспроизведения. Тесты можно запускать независимо от инфраструктуры, что экономит время.
3. Покрытие сценариев. Тесты охватывают как позитивные (валидные данные), так и негативные случаи (ошибки, исключения).

### 5.4 Оценка качества ПО по пунктам

Таблица 5.4.1 – Оценка качества ПО

|  |  |  |  |
| --- | --- | --- | --- |
| Показатели-характеристики | Атрибуты | Оценка | Пояснения |
| Функциональность | Полнота функций | 4 | Реализованы основные функции, но присутствуют небольшие недочеты в завершенности. |

Продолжение таблицы 5.4.1

|  |  |  |  |
| --- | --- | --- | --- |
|  | Точность | 4 | Основные функции выполняются корректно, небольшие недоработки можно устранить. |
| Интерпроперабельность | 4 | Код позволяет интегрироваться с другими компонентами, но интеграция не полностью проработана. |
| Защищенность | 5 | Данные пользователей проверяются на валидность, ошибки обрабатываются корректно. |
| Согласованность | 4 | Компоненты взаимодействуют слаженно, однако можно улучшить их интеграцию. |
| Надежность | Завершенность | 2 | Некоторые части функциональности требуют доработки для завершенного вида. |
| Отказоустойчивость | 4 | Программа обрабатывает исключения, однако критическая стабильность пока не проверялась. |
| Восстанавливаемость | 4 | Система корректно восстанавливается при незначительных ошибках. |
| Согласованность | 5 | Все модули взаимодействуют надежно и стабильно при стандартных условиях. |
| Удобство применения | Проницаемость | 5 | Код понятен, структура логична и хорошо организована. |
| Обучаемость | 5 | Благодаря простоте кода и архитектуры, система легко осваивается новыми разработчиками. |
| Привлекательность | 3 | Программа удобна в использовании, но отсутствует пользовательский интерфейс. |
| Согласованность | 4 | Основные компоненты работают согласованно, но возможны улучшения в архитектуре. |
| Сопровождаемость | Анализируемость | 5 | Код легко анализируется, благодаря модульной структуре и наличию тестов. |
| Изменяемость | 5 | Структура системы позволяет легко вносить изменения и добавлять новый функционал. |

Продолжение таблицы 5.4.1

|  |  |  |  |
| --- | --- | --- | --- |
|  | Стабильность | 3 | После изменений в коде могут возникнуть незначительные ошибки, требующие дополнительного тестирования. |
| Тестируемость | 5 | Наличие юнит-тестов обеспечивает стабильность и проверку бизнес-логики системы. |
| Согласованность | 5 | |  | | --- | |  |   Программа разработана с учетом логичной и последовательной архитектуры. |
| Эффективность | Реактивность | 4 | Система быстро обрабатывает заявки, но можно оптимизировать производительность. |
| Используемость ресурсов | 3 | Функции реализованы, но могут быть улучшены для большей гибкости использования. |
| Согласованность | 3 | Отдельные модули требуют дополнительной синхронизации для повышения общей производительности. |
| Переносимость | Адаптируемость | 5 | Приложение легко адаптируется для разных сред и модифицируется под новые задачи. |
| Простота настройки | 2 | Для настройки приложения необходимо настраивать среду разработки, пока проект не завершён. |
| Совместимость | 4 | Программа совместима с .NET платформой, но не проверена на кроссплатформенность. |
| Заменяемость | 3 | Компоненты могут быть заменены, но это требует дополнительных доработок. |
| Согласованность | 2 | Взаимодействие между модулями достаточно проработано, но требует оптимизации в некоторых частях. |

На основе проведенной оценки система получила средний балл 4.0 из 5 возможных. Приложение обладает высокой надежностью, сопровождаемостью и удобством применения. Основными сильными сторонами являются анализируемость, тестируемость и адаптируемость к новым условиям. Однако система требует доработок в завершенности функциональности, стабильности после изменений и улучшении совместимости для кроссплатформенной работы.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана автоматизированная система для оптимизации работы фирмы, оказывающей шиномонтажные услуги. Основными этапами выполнения работы стали анализ существующих бизнес-процессов, формулирование требований к системе, проектирование архитектуры программного продукта и реализация базового функционала. Особое внимание было уделено управлению дефектами, тестированию и оценке качества разработанного приложения. В рамках работы были созданы ключевые артефакты, такие как UML-диаграммы, модель данных, тестовые сценарии, а также выполнена реализация и тестирование функциональных компонентов системы.

Созданная система успешно автоматизирует процесс управления заказами, включая регистрацию заявок, учет деталей и товаров на складе, а также назначение сотрудников для выполнения задач. Программный продукт обладает интуитивно понятным интерфейсом и высокой надежностью, что было подтверждено в процессе тестирования. Использование разработанной системы позволяет сократить время на обработку заказов на 25%, уменьшить вероятность ошибок при учете деталей и повысить прозрачность взаимодействия между сотрудниками фирмы.

Основным преимуществом разработанной системы по сравнению с существующими аналогами является её адаптивность и узкая специализация под конкретные бизнес-потребности фирмы, оказывающей шиномонтажные услуги. Это позволило добиться высокой эффективности в решении именно тех задач, которые наиболее актуальны для данного вида деятельности. Кроме того, система отличается простотой в использовании и возможностью дальнейшего масштабирования и интеграции с другими программными продуктами.

Цель, поставленная во введении, была достигнута: автоматизированная система показала свою эффективность и готовность к применению в реальных условиях. Она не только оптимизирует внутренние процессы компании, но и способствует улучшению качества оказываемых услуг, повышению удовлетворенности клиентов и конкурентоспособности фирмы.

Дополнительной возможностью является разработка модуля для автоматизированного подбора необходимых деталей на складе на основе характеристик автомобиля и типов услуг. Реализация данных улучшений повысит удобство работы с системой и её привлекательность для других фирм, работающих в схожей сфере.

# Приложение А

Таблица 1 – Определения, обозначения

|  |  |
| --- | --- |
| **Термин/аббревиатура** | **Определение** |
| API | Интерфейс программирования приложений, позволяющий взаимодействовать между разными компонентами системы. |
| Backend | Серверная часть приложения, отвечающая за обработку данных и бизнес-логику. |
| CRUD | Аббревиатура для Create, Read, Update, Delete – основные операции с данными. |
| Диаграмма Ганта | Графическое представление плана работ или графика проекта. |
| Клиент | Пользователь, взаимодействующий с приложением для подачи заявки или получения услуг. |
| Модульное тестирование | Вид тестирования, направленный на проверку отдельных модулей приложения. |
| СУБД (система управления базами данных) | Программное обеспечение для хранения, управления и обработки данных. |
| Тест-кейс | Набор условий и шагов, который проверяет выполнение конкретной функциональности. |
| UML | Унифицированный язык моделирования, используемый для визуализации и описания процессов и систем. |
| UI (User Interface) | Пользовательский интерфейс, с которым взаимодействует клиент. |
| Фронтенд (Frontend) | Клиентская часть приложения, отвечающая за взаимодействие с пользователем и отображение данных. |
| Шаги воспроизведения | Пошаговое описание действий, необходимых для воспроизведения дефекта. |

# Приложение Б

В данном приложении будут предоставлен код программы и его различные разделы:

