

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего  
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

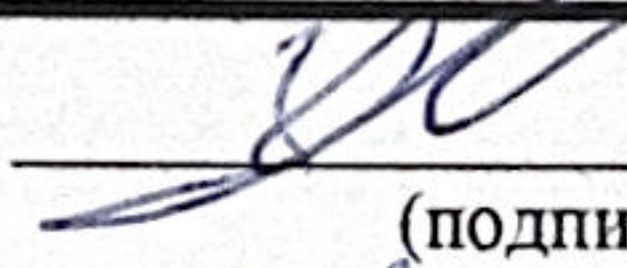
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»  
Профиль подготовки: «Программное обеспечение средств вычислительной  
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

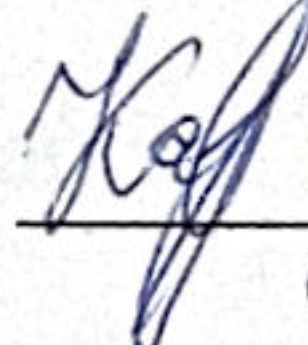
**ВЕБ-ПРИЛОЖЕНИЕ CRM-СИСТЕМА.**  
**ФРОНТЕНД ЧАСТЬ**

Студент

 11.06.2025  
(подпись, дата)

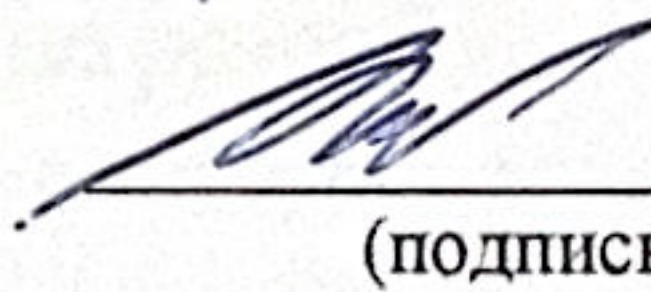
Тюрин Владислав Игоревич  
(ФИО полностью)

Руководитель

 11.06.2025  
(подпись, дата)

Карамышева Н.С.  
(фамилия, инициалы)

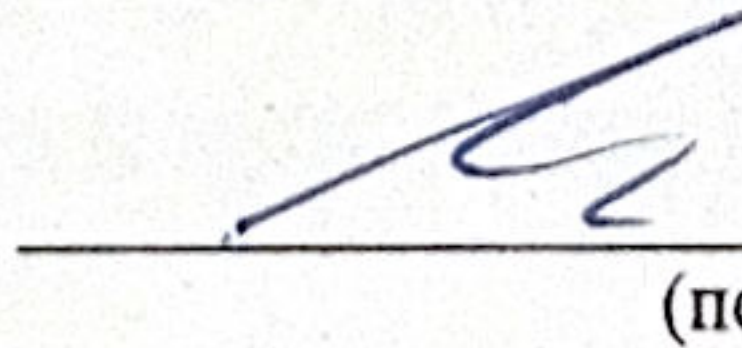
Нормоконтролёр

 11.06.2025  
(подпись, дата)

Бычков А.С.  
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой

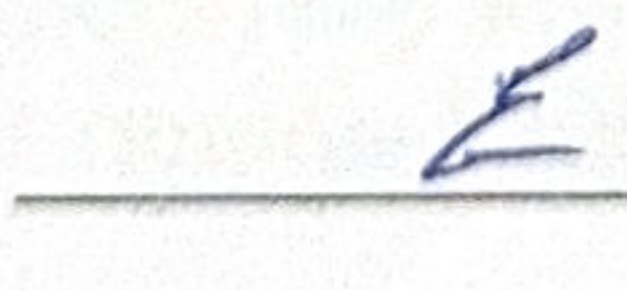
 (подпись)

М.А. Митрохин  
(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

16 июня 2025 № 33 )

Секретарь ГЭК

 (подпись)

Е.И. Гурин  
(фамилия, инициалы)

Пенза 2025

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

Факультет  
вычислительной техники

Кафедра  
«Вычислительная техника»

«Утверждаю»  
Заведующий кафедрой  
М.А. Митрохин  
« 06 » 05 20 25 г.

**ЗАДАНИЕ  
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Тюрин Владислав Игоревич гр. 21ВВП1 факультета вычислительной техники
2. Тема работы Веб-приложение CRM-система. Фронтенд часть

Тема утверждена приказом ПГУ № 601/о от 06.05.2025 г.

3. Руководитель бакалаврской работы доцент кафедры ВТ ПГУ Карамышева Н.С.

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

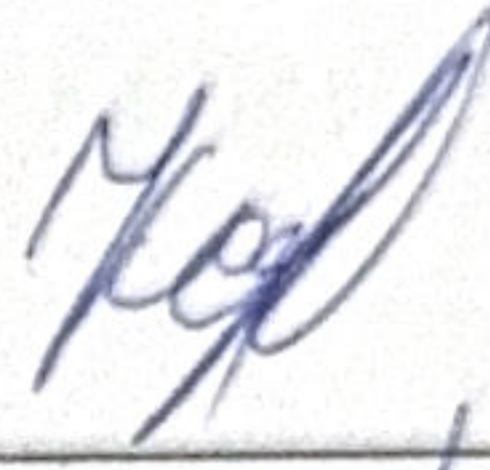
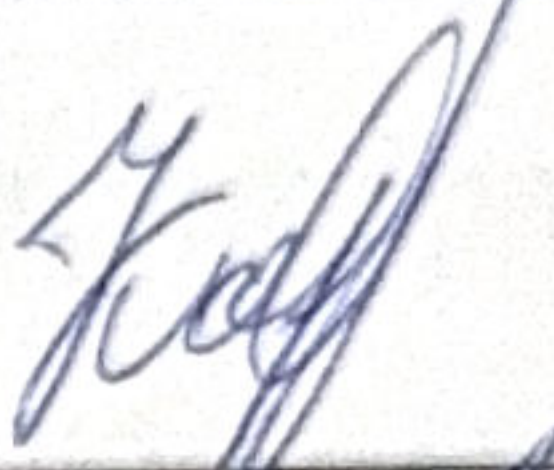
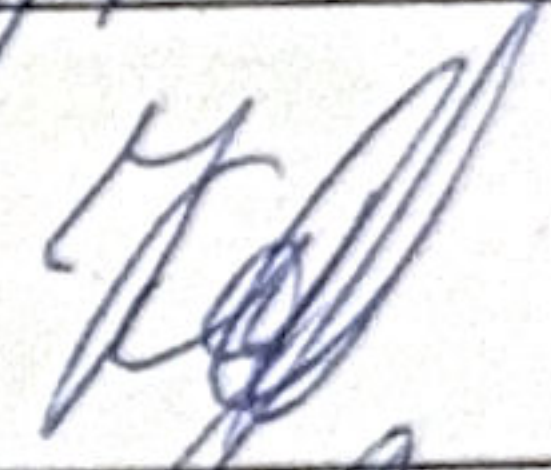
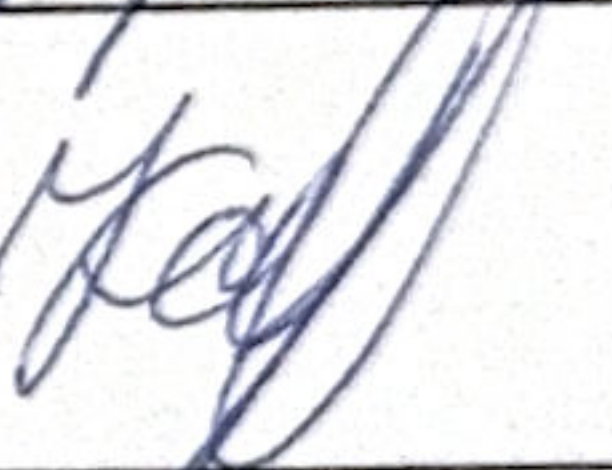
Разработать клиентскую часть web-приложения. Язык программирования — JavaScript, фреймворк — NextJS, среда разработки WebStorm. Клиентская часть должна обеспечивать надёжное взаимодействие с серверной частью через REST API и реализовывать следующие функциональные возможности:

1. Регистрацию и аутентификацию с использованием токенов;
2. Управление воронками и стадиями продаж;
3. Интеграция с телеграм;
4. Общение в реальном времени с клиентом;
5. Интеграцию с серверной частью через REST API.

5. Перечень подлежащих разработке вопросов

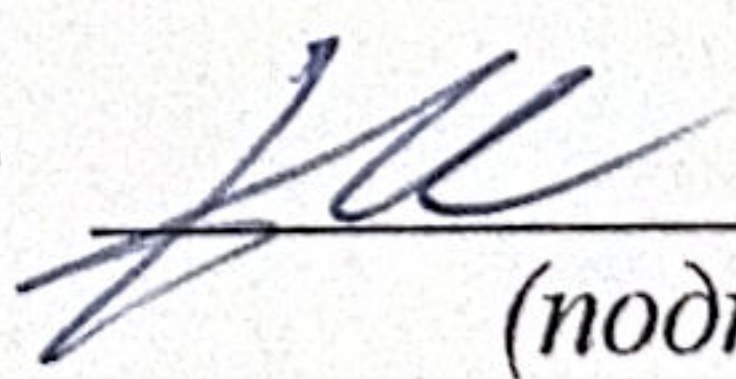
Требуется провести анализ предметной области: изучить существующие аналоги платформ, проанализировать архитектурные решения и подходы к построению грамотной frontend архитектуры. На основе проведенного анализа выбрать фреймворк. Описать архитектуру взаимодействия компонентов системы, реализовать соединение с серверной частью используя принципы REST API. Выгрузить программу на сервер.

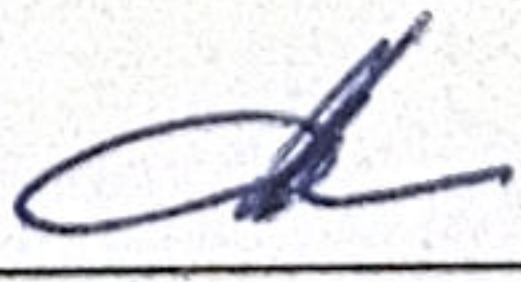
6. Календарный график выполнения работы

| № п/п | Наименование этапов работы                                    | Объем работы | Срок выполнения         | Подпись руководителя                                                                  |
|-------|---------------------------------------------------------------|--------------|-------------------------|---------------------------------------------------------------------------------------|
| 1     | Изучение предметной области                                   | 5%           | 08.05.2025 – 10.05.2025 |    |
| 2     | Анализ требований                                             | 10%          | 11.05.2025 – 14.05.2025 |    |
| 3     | Выбор технологий, применяемых при проектировании и разработке | 10%          | 15.05.2025 – 16.05.2025 |   |
| 4     | Разработка клиентской части                                   | 35%          | 17.05.2025 – 22.05.2025 |  |
| 5     | Выгрузка программы на сервер и его настройка                  | 25%          | 23.05.2025 – 25.05.2025 |  |
| 6     | Оформление пояснительной записки                              | 15%          | 26.05.2025 – 31.05.2025 |  |

Дата выдачи задания "6" 05 2025 г.

Руководитель бакалаврской работы  06.05.25 Карамышева Н.С.  
(подпись, дата)

Задание к исполнению принял студент  06.05.25 Тюрин В.И.  
(подпись, дата)

Работу к защите допустить  
Декан факультета  16.06.2025 Фионова Л.Р.

## Реферат

Бакалаврская работа содержит 144 страницы, 11 рисунков, 2 таблицы, 6 источников, 3 приложения.

WEB-ПРИЛОЖЕНИЕ, CRM-СИСТЕМА, NEXTJS, JAVASCRIPT, TYPESCRIPT, REST API, REDUX

Цель работы – разработать клиентскую часть web-приложения, которое обеспечивает связь между клиентом и менеджером, упрощая взаимодействие и отслеживание этапов сделки.

Разрабатываемая система предоставляет менеджерам удобно управлять воронками продаж, этапами и сделками.

В итоге была разработана клиентская часть web-приложения с использованием NextJS. Обмен информацией с серверной частью реализовано, используя принципы REST API.

|          |      |                 |                                                                                     |          |                                                                            |                            |      |        |
|----------|------|-----------------|-------------------------------------------------------------------------------------|----------|----------------------------------------------------------------------------|----------------------------|------|--------|
|          |      |                 |                                                                                     |          | ПГУ1.090301.22.001 ПЗ                                                      |                            |      |        |
| Изм.     | Лист | № докум.        | Подпись                                                                             | Дата     | Web-приложение CRM-система.<br>Фронтенд часть<br><br>Пояснительная записка | Лит.                       | Лист | Листов |
| Разраб.  |      | Тюрин В.И.      |  | 11.06.25 |                                                                            |                            |      |        |
| Провер.  |      | Карамышева Н.С. |  | 11.06.25 |                                                                            |                            | 2    | 141    |
| Н. Контр |      | Бычков А.С.     |  | 11.06.25 |                                                                            | ФВТ, каф. ВТ<br>гр. 21ВВП1 |      |        |
| Утверд.  |      | Митрохин М.А.   |  | 11.06.25 |                                                                            |                            |      |        |

## Содержание

### Введение

|                                                                                                 |    |
|-------------------------------------------------------------------------------------------------|----|
| 1 CRM-система .....                                                                             | 8  |
| 1.1 Анализ предметной области .....                                                             | 8  |
| 1.1.1 Актуальность разработки.....                                                              | 8  |
| 1.1.2 Анализ существующих аналогов .....                                                        | 10 |
| 1.1.3 Анализ ИИ-платформ для бизнеса.....                                                       | 12 |
| 1.1.4 Постановка задачи .....                                                                   | 16 |
| 1.2 Выбор технологий и средств разработки приложения .....                                      | 16 |
| 1.2.1 Выбор фреймворка .....                                                                    | 17 |
| 1.2.2 Выбор UI-Библиотеки .....                                                                 | 17 |
| 1.2.3 Выбор системы управления состоянием .....                                                 | 18 |
| 1.2.4 Выбор инструментов для работы с API.....                                                  | 18 |
| 1.2.5 Выбор библиотеки для drag-and-drop функциональности.....                                  | 18 |
| 1.2.6 Выбор среды разработки.....                                                               | 19 |
| 1.2.7 Выбор архитектурного стиля API.....                                                       | 20 |
| 1.3 Проектирование CRM-системы .....                                                            | 21 |
| 1.3.1 Паттерн MVC .....                                                                         | 21 |
| 1.3.2 Микрофронтенды в CRM-системе: независимая разработка и кросс-<br>доменная интеграция..... | 24 |
| 1.4 Разработка модуля ID.....                                                                   | 25 |
| 1.4.1 Реализация authGuard .....                                                                | 25 |
| 1.4.2 Реализация основных страниц .....                                                         | 28 |
| 1.4.3 Реализация взаимодействия с API .....                                                     | 35 |
| 1.5 Разработка модуля CRM .....                                                                 | 43 |
| 1.5.1 Реализация authGuard .....                                                                | 43 |
| 1.5.2 Реализация основных страниц и компонентов .....                                           | 45 |
| 1.6 Архитектура развертывания и эксплуатационные требования.....                                | 63 |
| 1.6.1 Конвейер непрерывной интеграции и доставки (CI/CD).....                                   | 63 |

|                                                          |     |
|----------------------------------------------------------|-----|
| 1.6.2 Docker-инфраструктура .....                        | 64  |
| 2 Безопасность при работе с цифровыми устройствами ..... | 65  |
| 2.1 Как безопасно работать за компьютером .....          | 65  |
| 2.2 Безопасность использования смартфонов .....          | 66  |
| Заключение.....                                          | 67  |
| Список использованных источников.....                    | 69  |
| Приложение А – UML-диаграммы .....                       | 70  |
| Приложение Б – Листинг программы.....                    | 76  |
| Приложение В – Презентация .....                         | 140 |

## Перечень сокращений и обозначений

В настоящей выпускной квалификационной работе применяются следующие сокращения и обозначения

ВКР – выпускная квалификационная работа

ИИ – искусственный интеллект

IT – information technology

API – (Application Programming Interface) – интерфейс программирования приложений, набор правил для взаимодействия программных компонентов

CRUD – (Create, Read, Update, Delete) – основные операции управления данными в базе данных

HTTP – (HyperText Transfer Protocol) – протокол передачи гипертекста, используемый для обмена данными между web-браузером и сервером

IDE – (Integrated Development Environment) – интегрированная среда разработки

JWT – (JSON Web Token) – компактный и безопасный способ передачи информации между сторонами как объекта JSON, часто используется для авторизации

REST – (Representational State Transfer) – архитектурный стиль взаимодействия между компонентами в распределённой среде, часто используется для создания API

SEO – (Search Engine Optimization) – это комплекс мер по улучшению сайта для его ранжирования в поисковых системах

SSH – (Secure Shell) – протокол удалённого доступа к серверу через защищённое соединение

UI – (User Interface) – пользовательский интерфейс

UML – (Unified Modeling Language) – унифицированный язык моделирования

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 5    |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

## Введение

Современные условия ведения бизнеса требуют от компаний постоянного повышения эффективности клиентского обслуживания и оптимизации внутренних процессов. Особенно остро этот вопрос стоит перед малым и средним бизнесом, где ограниченные ресурсы не позволяют внедрять сложные и дорогостоящие CRM-решения. При этом потребность в автоматизации процессов работы с клиентами, анализе данных и повышении уровня сервиса остается крайне высокой.

Разрабатываемая в рамках данной работы CRM-система с интегрированными технологиями искусственного интеллекта направлена на решение этих задач. Ее ключевой особенностью является сочетание простоты использования с современными возможностями автоматизации. Система разрабатывается с учетом специфики малого бизнеса, где важны быстрая окупаемость, минимальные затраты на внедрение и простота освоения персоналом. Основное конкурентное преимущество решения заключается в наличии интеллектуального модуля, способного автоматически обрабатывать клиентские запросы в нерабочее время. Это позволяет обеспечивать круглосуточную поддержку без необходимости содержания дополнительного персонала.

Система анализирует входящие обращения, определяет их суть и предоставляет релевантные ответы на основе накопленной базы знаний. При этом сохраняется возможность перенаправления сложных запросов менеджерам при их выходе на работу. Актуальность разработки подтверждается растущим спросом на доступные цифровые инструменты среди малых предприятий. Многие существующие CRM-системы либо слишком сложны и дороги, либо не предлагают современных возможностей автоматизации.

Предлагаемое решение заполняет эту нишу, предоставляя базовый функционал управления клиентской базой в сочетании с передовыми

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 6    |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

технологиями искусственного интеллекта.

Важной особенностью системы является ее адаптивность. Интерфейс интуитивно понятен и не требует длительного обучения, а гибкие настройки позволяют адаптировать платформу под специфику конкретного бизнеса. При этом сохраняется возможность работы в оффлайн-режиме, что особенно важно для предприятий в регионах с нестабильным интернет-соединением.

Разработка такой CRM-системы открывает новые возможности для малого бизнеса в области цифровой трансформации. Внедрение решения позволит компаниям повысить качество обслуживания клиентов, увеличить эффективность работы с базой данных и оптимизировать бизнес-процессы без значительных капиталовложений. Это особенно важно в условиях высокой конкуренции, где уровень клиентского сервиса становится ключевым фактором успеха.

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 7    |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

# 1 CRM-система

## 1.1 Анализ предметной области

### 1.1.1 Актуальность разработки

«ML.CRM» – это CRM-система, оснащенная искусственным интеллектом, которая автоматизирует обработку клиентских запросов в нерабочее время. Основная цель – повышение эффективности работы менеджеров за счет сокращения времени ответа и минимизации потерь потенциальных клиентов. Платформа сочетает в себе функционал классической CRM с передовыми возможностями ИИ-ассистента, создавая комплексное решение для управления клиентскими отношениями.

Современный рынок CRM-решений демонстрирует устойчивый рост, особенно в сегменте интеллектуальных систем. Согласно исследованиям, к 2026 году объем мирового рынка CRM достигнет \$96 миллиардов, при этом системы с ИИ-интеграцией будут занимать около 30% этого рынка. В России спрос на умные CRM-решения увеличился на 45% за последние два года, что свидетельствует о растущей потребности бизнеса в автоматизации клиентского сервиса.

В современных условиях ведения бизнеса, когда конкуренция растет, а клиенты становятся более требовательными, эффективное управление взаимоотношениями с ними становится ключевым фактором успеха. Компании сталкиваются с необходимостью не только привлекать новых клиентов, но и удерживать существующих, повышая их лояльность. В этом контексте CRM-системы приобретают особую значимость, так как они позволяют автоматизировать процессы взаимодействия с клиентами, анализировать их поведение и оптимизировать бизнес-процессы.

Одной из главных проблем, с которой сталкиваются предприятия, является рост объема данных о клиентах. Вручную обрабатывать эту информацию становится все сложнее, что приводит к ошибкам, потере времени и снижению

эффективности работы. CRM-система решает эту проблему, предоставляя единую платформу для хранения и анализа данных. Это позволяет сотрудникам быстрее получать нужную информацию, своевременно реагировать на запросы клиентов и принимать обоснованные решения. Кроме того, автоматизация рутинных задач, таких как напоминания, отчеты и рассылки, высвобождает время для более важных аспектов работы, таких как стратегическое планирование и развитие бизнеса.

Еще одним важным аспектом является персонализация взаимодействия с клиентами. Современные потребители ожидают индивидуального подхода, и компании, которые не могут его обеспечить, рискуют потерять свою аудиторию. CRM-системы позволяют анализировать историю покупок, предпочтения и поведение клиентов, что дает возможность предлагать им релевантные товары и услуги. Это не только повышает удовлетворенность клиентов, но и увеличивает средний чек и частоту покупок.

Рост конкуренции на рынке заставляет компании искать новые способы удержания клиентов. CRM-системы помогают снизить процент оттока, выявляя потенциально нелояльных клиентов и позволяя вовремя принимать меры для их удержания. Кроме того, такие системы способствуют повышению конверсии на всех этапах воронки продаж, от первого контакта до завершения сделки. Это особенно важно в условиях, когда стоимость привлечения нового клиента продолжает расти.

Развитие цифровых технологий также влияет на актуальность CRM-систем. Сегодня клиенты взаимодействуют с компаниями через множество каналов: социальные сети, мессенджеры, электронную почту, телефонные звонки. Без единой системы учета этих взаимодействий легко упустить важные детали и потерять клиента. CRM-система интегрирует все каналы коммуникации, обеспечивая целостность данных и удобство работы для сотрудников.

Научная и практическая значимость разработки CRM-системы заключается в том, что она объединяет передовые технологии, такие как искусственный

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 9    |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

интеллект, машинное обучение и big data, с реальными потребностями бизнеса. Использование этих технологий позволяет не только автоматизировать процессы, но и прогнозировать поведение клиентов, выявлять тенденции и оптимизировать маркетинговые стратегии.

Разрабатываемая в данной работе CRM-система с интеграцией искусственного интеллекта призвана обеспечить круглосуточное взаимодействие с клиентами даже в нерабочее время сотрудников. Ключевой особенностью системы является интеллектуальный модуль, способный автоматически обрабатывать входящие запросы, анализировать их содержание и предоставлять клиентам релевантные ответы на основе диалога и заданного контекста. Это решение особенно актуально для малого бизнеса, позволяя поддерживать постоянную коммуникацию с клиентской базой без необходимости расширения штата сотрудников. Система не только обеспечивает базовые ответы на типовые вопросы, но и способна определять срочность и приоритетность обращений, перенаправляя наиболее важные запросы менеджерам при первом их выходе на работу. Такая функциональность значительно повышает качество клиентского сервиса и помогает избежать потери потенциальных заказов в периоды отсутствия персонала.

#### 1.1.2 Анализ существующих аналогов

В рамках исследования рынка CRM-решений для малого и среднего бизнеса был проведен детальный анализ функциональных возможностей ведущих платформ: Битрикс24 и АМОCRM. Эти системы были выбраны как наиболее репрезентативные примеры современных CRM, сочетающие широкий функционал с относительной доступностью.

Для выявления сильных и слабых сторон современных CRM-решений был выполнен детальный сравнительный обзор двух востребованных

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 10   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

платформ российского сегмента — amoCRM и Bitrix24, а также перспективной разработки — ML.CRM.

В таблице 1 представлено сравнение существующих аналогов.

Таблица 1 – Сравнение аналогов

|                                                                                          | amoCRM | Bitrix  | ML.CRM    |
|------------------------------------------------------------------------------------------|--------|---------|-----------|
| Возможность подключения внешнего источника общения                                       | +      | +       | +         |
| Возможность управления сделкой                                                           | +      | +       | +         |
| Возможность управления полями                                                            | +      | +       | +         |
| Возможность настройки воронок и этапов                                                   | +      | +       | +         |
| Маркет решений                                                                           | +      | +       | -         |
| Внешнее API для подключения                                                              | +      | +       | -         |
| Графики аналитики                                                                        | +      | +       | -         |
| Легкость восприятия интерфейса                                                           | Легкий | Сложный | Легкий    |
| Минимальная стоимость за полгода для подключения внешнего сервиса ИИ для чатов и звонков | 6594р  | 14940р  | бесплатно |

Современные CRM-платформы, такие как amoCRM и Bitrix24, представляют собой комплексные решения с богатым функционалом для управления клиентскими отношениями. Эти системы обладают развитыми возможностями интеграции через API, мощными аналитическими инструментами и обширными маркетплейсами с дополнительными модулями.

Однако их главный недостаток заключается в сложности освоения и необходимости значительных временных и финансовых затрат на полноценное внедрение, что делает их не всегда оптимальным выбором для малого бизнеса и стартапов.

В отличие от них, разрабатываемая система ML.CRM создавалась с акцентом на простоту использования и доступность. Её ключевое преимущество – встроенные инструменты искусственного интеллекта, позволяющие автоматизировать основные бизнес-процессы без необходимости подключения дополнительных модулей. Платформа предлагает интуитивно понятный интерфейс, и готовые решения для работы с клиентской базой, что значительно сокращает время на обучение сотрудников.

Особенностью ML.CRM является нативная интеграция с ИИ-ассистентами, обеспечивающая автоматическую обработку обращений с помощью интеллектуальных ботов, анализ клиентских запросов и генерацию персонализированных предложений. При этом система сохраняет гибкость в настройке воронок продаж и полей данных, позволяя адаптировать её под специфику конкретного бизнеса без привлечения технических специалистов.

Такая концепция делает ML.CRM привлекательным решением для компаний, которые стремятся внедрить современные технологии управления клиентскими отношениями, но при этом хотят избежать сложных и дорогостоящих внедренческих процессов. Сочетание простоты использования, доступной стоимости и передовых ИИ-возможностей открывает новые перспективы для малого и среднего бизнеса в области цифровой трансформации.

### 1.1.3 Анализ ИИ-платформ для бизнеса

В контексте разработки CRM-системы с элементами искусственного интеллекта особое внимание уделяется выбору оптимального решения для интеграции интеллектуального ассистента. Современный рынок предлагает несколько специализированных платформ, каждая из которых обладает

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          |      |
| Изм. | Лист | № докум. | Подпись | Дата |                          | 12   |

уникальными характеристиками и возможностями. Среди наиболее перспективных вариантов для интеграции следует рассмотреть Avatarex, Maia, Savvi и AmoAI - технологические решения, различающиеся по функциональному наполнению, степени адаптивности и экономической целесообразности внедрения.

Проведение детального сравнительного анализа указанных платформ обусловлено необходимостью выбора наиболее подходящего технологического партнёра для реализации интеллектуального модуля в разрабатываемой CRM-системе. Критерии оценки включают не только базовые параметры стоимости и простоты интеграции, но и такие важные аспекты как:

- способность платформы к контекстному анализу клиентских запросов;
- возможности обучения и адаптации под специфику конкретного бизнеса;
- качество лингвистической обработки естественной речи;
- гибкость настройки сценариев взаимодействия;
- поддержка многоканальных коммуникаций.

Особое значение при выборе имеет совместимость платформы с существующей архитектурой CRM-системы и возможность её бесшовной интеграции без существенных доработок базового функционала.

Данный анализ позволит определить оптимальное технологическое решение, которое не только обеспечит требуемый уровень автоматизации клиентских коммуникаций, но и сохранит ключевые преимущества разрабатываемой системы.

Таблица 2 – Сравнение ИИ-платформ для бизнеса

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 13   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

| Сервис /<br>Функционал                      | Avatarex | Maia | Savvi | AmoAI | ML.AI |
|---------------------------------------------|----------|------|-------|-------|-------|
| Голосовые боты                              | -        | -    | -     | -     | +     |
| Текстовые боты                              | +        | +    | +     | +     | +     |
| Проведение квалификации                     | +        | -    | +     | -     | +     |
| Работа с базой данных                       | +        | -    | +     | -     | +     |
| Работа с базой знаний                       | -        | +    | +     | -     | +     |
| Работа с поиском по сайту                   | -        | -    | -     | -     | +     |
| Реагирование на ситуации                    | -        | -    | +     | -     | +     |
| Количество доступных провайдеров нейросетей | 1        | 1    | 1     | 1     | 5     |
| Работа с игнорированием                     | -        | -    | -     | -     | +     |

| Сервис /<br>Функционал                         | Avatarex    | Maia        | Savvi       | AmoAI       | ML.AI     |
|------------------------------------------------|-------------|-------------|-------------|-------------|-----------|
| Обязательность<br>использования<br>внешней CRM | +           | +           | -           | -           | -         |
| Автоматическая генерация<br>инструкций         | -           | -           | -           | -           | +         |
| Пауза перед<br>обработкой<br>сообщений         | +           | -           | +           | -           | +         |
| Пауза при<br>вмешательстве<br>менеджера        | +           | -           | +           | -           | +         |
| Скорость<br>ответа бота                        | 28 сек      | 18 сек      | 5 сек       | 5 сек       | 3 сек     |
| Стабильность<br>ответа бота                    | Нестабильно | стабильно   | стабильно   | стабильно   | стабильно |
| Цена<br>подписки                               | бесплатно   | отсутствует | отсутствует | отсутствует | 5000р     |
| Цена диалога<br>из 10<br>сообщений             | 20р         | 10 руб      | 30р         | 0 руб       | 2 руб     |

|                        |          |        |        |       |        |
|------------------------|----------|--------|--------|-------|--------|
| Сервис /<br>Функционал | Avatarex | Maia   | Savvi  | AmoAI | ML.AI  |
| Сложность<br>настройки | Средне   | Сложно | Сложно | Легко | Средне |

#### 1.1.4 Постановка задачи

Необходимо разработать удобный и интуитивно понятный пользовательский интерфейс, обеспечивающий комфортную работу с системой для пользователей с разным уровнем подготовки. Интерфейс должен отличаться продуманной логикой взаимодействия, где каждый элемент расположен в соответствии с частотой использования и важностью функций. Особое внимание следует уделить простоте освоения системы - новые пользователи должны понимать принципы работы без длительного изучения инструкций. Все элементы интерфейса необходимо выполнить в едином стиле, соблюдая принципы визуальной иерархии, когда наиболее важные элементы выделяются размером, цветом или расположением. Важной составляющей является адаптивность интерфейса, который должен корректно отображаться и сохранять удобство использования на устройствах с разными размерами экранов - от десктопных мониторов до мобильных телефонов.

#### 1.2 Выбор технологий и средств разработки приложения

В данном разделе будет проведен детальный анализ технологического стека, использованного при разработке CRM-системы с ИИ-ассистентом. Особое внимание будет уделено специфическим потребностям CRM-систем, таким как работа с большими объемами данных в реальном времени, необходимость обеспечения высокой доступности интерфейса и поддержка сложных бизнес-процессов.

### 1.2.1 Выбор фреймворка

В качестве основного фреймворка для разработки пользовательского интерфейса был выбран Next.js – современный фреймворк на основе React [1]. Этот выбор обусловлен рядом существенных преимуществ, которые особенно важны для CRM-систем. Next.js предоставляет встроенную поддержку серверного рендеринга (SSR) и статической генерации страниц (SSG), что критически важно для обеспечения высокой производительности и SEO-оптимизации. В контексте CRM-приложений это позволяет значительно улучшить время первой загрузки интерфейса, что непосредственно влияет на пользовательский опыт.

Фреймворк предлагает удобную файловую маршрутизацию, что упрощает организацию кодовой базы крупного проекта. Встроенная система API-роутов позволяет легко интегрировать фронтенд с бэкенд-сервисами. Next.js также отличается отличной поддержкой TypeScript, что способствует повышению надежности кода. Сообщество разработчиков активно поддерживает фреймворк, регулярно выпуская обновления и новые функции.

### 1.2.2 Выбор UI-Библиотеки

Для построения пользовательского интерфейса была выбрана библиотека Material UI (MUI) [2], которая представляет собой реализацию Material Design для React. Этот выбор обусловлен несколькими ключевыми факторами. Во-первых, MUI предоставляет обширный набор готовых, профессионально разработанных компонентов, что значительно ускоряет процесс разработки. Во-вторых, библиотека предлагает мощную систему темизации, позволяющую легко адаптировать внешний вид приложения под корпоративный стиль.

Компоненты Material UI отличаются высокой степенью доступности (a11y), что соответствует современным стандартам веб-разработки. Библиотека включает в себя все необходимые элементы для построения сложных CRM-интерфейсов: таблицы, формы, диалоговые окна, элементы

навигации и многое другое. Поддержка адаптивного дизайна в MUI позволяет создавать интерфейсы, которые одинаково хорошо работают на устройствах с различными размерами экранов.

### 1.2.3 Выбор системы управления состоянием

Для управления состоянием приложения был выбран Redux Toolkit – современный подход к работе с глобальным состоянием в React-приложениях. В контексте CRM-систем, где часто требуется обработка больших объемов данных и сложные взаимодействия между компонентами, Redux Toolkit предлагает оптимальное решение. Он предоставляет упрощенный API по сравнению с классическим Redux, уменьшая объем шаблонного кода.

Redux Toolkit включает в себя такие полезные функции как createSlice для создания редьюсеров с автоматической генерацией экшенов, configureStore для настройки хранилища с предустановленными middleware, и createAsyncThunk для работы с асинхронными операциями. В сочетании с React-Redux это обеспечивает предсказуемое управление состоянием и удобные инструменты для отладки.

### 1.2.4 Выбор инструментов для работы с API

Для взаимодействия с бэкенд-API была выбрана библиотека Axios. Этот HTTP-клиент предоставляет удобный интерфейс для выполнения запросов к серверу и обработки ответов. Axios поддерживает все современные возможности работы с HTTP, включая перехватчики запросов и ответов, отмену запросов, автоматическую трансформацию данных JSON и защиту от XSRF [3]. Для типизации запросов и ответов используется TypeScript, что повышает надежность кода и упрощает его поддержку. Библиотека также предоставляет удобные механизмы для обработки ошибок, что особенно важно в корпоративных приложениях.

### 1.2.5 Выбор библиотеки для drag-and-drop функциональности

Для реализации интерактивных возможностей перетаскивания элементов в интерфейсе была выбрана библиотека @dnd-kit – современное

решение для организации drag-and-drop взаимодействий в React-приложениях. Этот выбор обусловлен несколькими ключевыми преимуществами, которые делают @dnd-kit оптимальным вариантом для CRM-систем.

Основным преимуществом @dnd-kit является исключительная производительность даже при работе с большими наборами данных, что критически важно для CRM-систем, где часто требуется манипулировать сотнями элементов. Библиотека использует современные API браузера и оптимизированные алгоритмы рендеринга, минимизируя нагрузку на

Производительность[4].

Для проекта особенно важна была поддержка сложных сценариев:

- перетаскивание карточек между различными статусами в CRM (например, "Новые", "В работе", "Завершенные");
- сортировка элементов внутри списков с возможностью задания сложных правил;
- поддержка touch-устройств для мобильных пользователей. Гибкая система ограничений и условий для перетаскивания;

Библиотека прекрасно интегрируется с Redux, позволяя синхронизировать изменения положения элементов с глобальным состоянием приложения. Реализована поддержка анимаций при перетаскивании, что значительно улучшает пользовательский опыт.

Реализация drag-and-drop с помощью @dnd-kit позволила создать интуитивно понятный интерфейс для управления сделками в CRM-системе, значительно повысив удобство работы пользователей с большими объемами данных.

#### 1.2.6 Выбор среды разработки

В процессе разработки фронтенд-части проекта была выбрана интегрированная среда разработки WebStorm от компании JetBrains. Этот выбор обусловлен рядом важных преимуществ, которые делают WebStorm

оптимальным решением для работы с современным JavaScript/TypeScript-стеком. Среда предоставляет полноценную поддержку всех используемых в проекте технологий - Next.js, React, TypeScript и сопутствующих библиотек. Одним из ключевых преимуществ WebStorm является интеллектуальная система автодополнения кода, которая учитывает TypeScript-типы и значительно ускоряет процесс написания кода. Встроенные инструменты анализа кода помогают выявлять потенциальные проблемы на ранних этапах разработки. WebStorm предлагает мощные возможности рефакторинга, что особенно важно при работе с крупными проектами, где регулярно требуется модифицировать существующую кодобазу. Среда обеспечивает удобную навигацию по проекту с пониманием структуры React-компонентов и их взаимосвязей.

Важной особенностью является глубокая интеграция с системами контроля версий (Git), что упрощает командную работу над проектом. WebStorm также поддерживает ESLint и Prettier, обеспечивая соблюдение единых стандартов кодирования. Встроенный отладчик позволяет эффективно находить и исправлять ошибки в клиентском коде. Все эти функции делают WebStorm мощным и удобным инструментом для разработки сложных веб-приложений, каким является наша CRM-система.

#### 1.2.7 Выбор архитектурного стиля API

Для организации клиент-серверного взаимодействия в проекте был выбран архитектурный стиль REST (Representational State Transfer). Этот выбор обусловлен несколькими ключевыми факторами, делающими REST оптимальным решением для CRM-систем. REST основан на стандартных HTTP-методах (GET, POST, PUT, DELETE, PATCH), что делает API интуитивно понятным и легко используемым. Каждый ресурс в системе имеет уникальный URL, а операции с ресурсами выполняются с помощью соответствующих HTTP-методов.

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 20   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

Для передачи данных используется универсальный JSON-формат, который легко обрабатывается как на клиентской, так и на серверной стороне. В нашем проекте REST API реализован с соблюдением лучших практик: используется единообразная структура ответов, содержащая данные, статус выполнения и при необходимости – сообщения об ошибках.

Для аутентификации применяется схема с JWT-токенами, которые передаются в заголовке Authorization.

API поддерживает версионирование, что позволяет вносить изменения без нарушения работы существующих клиентов. Реализована пагинация для больших наборов данных, а также система фильтрации и сортировки через параметры запроса.

Для документирования API используется Swagger, что упрощает взаимодействие между разработчиками фронтенда и бэкенда. Важным преимуществом REST является его хорошая совместимость с выбранным фронтенд-стеком – библиотека Axios, используемая для HTTP-запросов, идеально подходит для работы с RESTful API. На стороне TypeScript реализована полная типизация всех DTO (Data Transfer Objects), что повышает надежность интеграции. REST также хорошо подходит для кэширования ответов, что улучшает производительность приложения. В целом, выбор REST API позволил создать надежную, масштабируемую и удобную в использовании систему клиент-серверного взаимодействия для нашей CRM-системы.

## 1.3 Проектирование CRM-системы

### 1.3.1 Паттерн MVC

В современном веб-разработке архитектурные паттерны играют ключевую роль в обеспечении масштабируемости, поддерживаемости и удобства сопровождения программных продуктов. В рамках реализации данного проекта был выбран архитектурный подход, сочетающий в себе

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 21   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

элементы паттернов MVC (Model-View-Controller) и Feature-Sliced Design, что позволяет наиболее эффективно организовать структуру клиентской части приложения на базе React и Next.js.

Паттерн MVC (Model-View-Controller) традиционно делит приложение на три взаимосвязанных, но независимых компонента: модель, представление и контроллер. Модель отвечает за бизнес-логику, обработку и хранение данных, а также за взаимодействие с внешними сервисами и API. В нашем проекте роль модели выполняют модули, расположенные в директориях store и api. В частности, использование Redux Toolkit позволяет централизованно управлять состоянием приложения, а отдельные редьюсеры инкапсулируют бизнес-логику различных функциональных областей. Модули API реализуют взаимодействие с сервером, обработку запросов и получение данных, что также относится к слою модели [5].

Представление (View) реализовано с помощью компонентов React, сгруппированных в директориях views и layouts. Эти компоненты отвечают за визуализацию данных, полученных из модели, и формируют пользовательский интерфейс приложения. Благодаря использованию современных библиотек для стилизации и темизации (например, MUI и Emotion), представление становится гибким и легко настраиваемым, что позволяет быстро изменять внешний вид приложения без необходимости затрагивать бизнес-логику.

Контроллер (Controller) в контексте React-приложения реализуется через компоненты-обёртки, хуки и специальные файлы роутинга, такие как файлы в директории pages. Контроллеры принимают действия пользователя, обрабатывают события, осуществляют валидацию данных и инициируют соответствующие изменения в модели. Например, при авторизации пользователя контроллер инициирует запрос к API, обновляет состояние в Redux и передаёт полученные данные в представление для отображения. В

Next.js файлы страниц (pages) также выполняют роль контроллеров, определяя маршруты и связывая между собой остальные компоненты архитектуры.

Важной особенностью архитектуры данного проекта является использование глобальных контекстов (React Context), которые обеспечивают доступ к состоянию авторизации, настройкам приложения и другим глобальным данным из любой части приложения. Это позволяет реализовать единые механизмы управления доступом, темизацией и пользовательскими настройками, не нарушая принципов разделения ответственности. В результате, архитектура приложения обеспечивает чёткое разделение ответственности между слоями, что значительно упрощает процесс разработки, тестирования и поддержки кода. Каждый из слоёв может развиваться независимо, что особенно важно при работе в команде и при масштабировании проекта. Такой подход позволяет быстро внедрять новые функции, не затрагивая существующую логику, а также облегчает поиск и устранение ошибок. Логическая схема архитектуры приложения, реализованного по паттерну MVC продемонстрирована на рисунке 1.

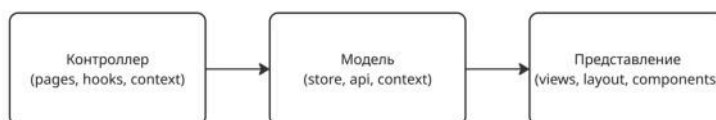


Рисунок 1 – Логическая схема MVC-приложения

В данной схеме контроллеры принимают действия пользователя, взаимодействуют с моделью для получения или изменения данных, а затем передают обновлённую информацию в представление для отображения пользователю. Модель инкапсулирует бизнес-логику и обеспечивает

целостность данных, а представление отвечает за визуализацию и взаимодействие с пользователем.

Для более детального отражения особенностей архитектуры, используемой в современных Next.js приложениях, можно привести следующую схему на рисунке 2, иллюстрирующую взаимодействие между слоями и модулями.



Рисунок 2 – Схема взаимодействие между слоями и модулями

### 1.3.2 Микрофронтенды в CRM-системе: независимая разработка и кросс-доменная интеграция

Современные CRM-системы требуют высокой степени модульности, масштабируемости и независимости компонентов. В данной работе реализована архитектура микрофронтендов, где каждый модуль (авторизация, управление клиентами, аналитика) представляет собой полностью автономное приложение, развернутое на отдельном домене. Ключевая особенность подхода — отсутствие жесткой связи между модулями, в отличие от Module Federation. Взаимодействие между ними осуществляется через query параметры в перенаправлениях.

В предложенной CRM-системе реализованы:

- основное приложение (CRM Core) – развернуто на `crm.magiclab.space`;
- микрофронтенд авторизации (Auth Frontend) – размещен на `id.magiclab.space`;
- бекенд-сервис ID (ID Service) – работает на `api.id.magiclab.space`;
- бекенд-сервис CRM (CRM Service) – работает на `crmapi.magiclab.space`.

На рисунке 3 продемонстрирована схема взаимодействия сервисов.

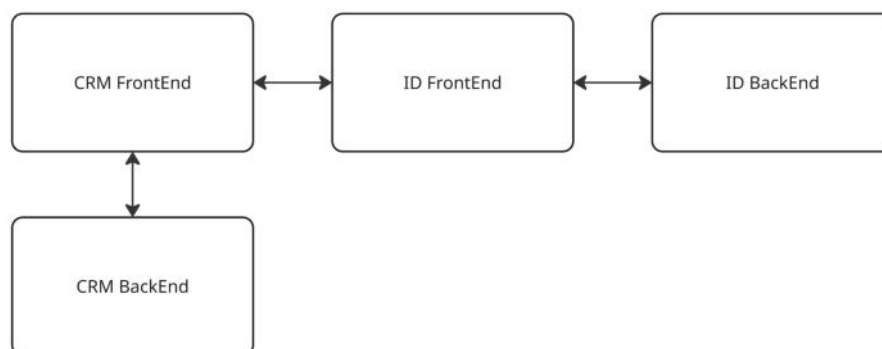


Рисунок 3 – Схема взаимодействие между сервисами приложения

## 1.4 Разработка модуля ID

### 1.4.1 Реализация authGuard

В современных веб-приложениях часто возникает необходимость ограничивать доступ к определённым разделам сайта только для авторизованных пользователей. Для решения этой задачи применяются так называемые "guard"-компоненты (от англ. guard — "страж", "защитник"). В данном проекте для этих целей реализован компонент AuthGuard, который обеспечивает защиту маршрутов (страниц) от неавторизованного доступа.

Основные задачи AuthGuard продемонстрированы далее.

- проверка статуса авторизации пользователя — определяет, вошёл ли пользователь в систему;
- перенаправление неавторизованных пользователей — если пользователь не авторизован, происходит автоматический переход на страницу входа;
- отображение индикатора загрузки — пока идёт проверка статуса авторизации, пользователю показывается специальный компонент;
- отображение защищённого контента — если пользователь авторизован,

ему предоставляется доступ к защищённому содержимому.

Основная логика компонента `AuthGuard` описана ниже.

```
const AuthGuard = (props: AuthGuardProps) => {
  const { children, fallback } = props
  const auth = useAuth()
  const router = useRouter()

  useEffect(() => {
    if (!router.isReady) return

    if (auth.user === null &&
!window.localStorage.getItem('userData')) {
      if (router.asPath !== '/') {
        router.replace({
          pathname: '/login',
          query: { returnUrl: router.asPath }
        })
      } else {
        router.replace('/login')
      }
    }
  }, [router.route])

  if (auth.loading || auth.user === null) {
    return fallback
  }

  return <>{children}</>
}
```

С помощью хука `useAuth` получаем объект `auth`, который содержит элементы описанные ниже.

– `user` — данные текущего пользователя (если авторизован) или `null` (если

не авторизован);

- loading — булево значение, показывающее, идёт ли сейчас процесс проверки авторизации.

С помощью хука useRouter получаем объект router, который позволяет управлять переходами между страницами.

Побочный эффект (useEffect) срабатывает при изменении маршрута (router.route). Внутри эффекта: Проверяется, готов ли маршрутизатор (router.isReady). Если нет — эффект не выполняется. Если пользователь не авторизован (auth.user === null) и в локальном хранилище браузера (localStorage) отсутствуют данные пользователя (userData), происходит перенаправление на страницу входа (/login). Если пользователь пытался попасть не на главную страницу (/), в параметры запроса добавляется путь возврата (returnUrl). Это позволяет после успешной авторизации вернуть пользователя на ту страницу, куда он изначально хотел попасть. Если данные пользователя ещё загружаются (auth.loading) или пользователь не авторизован (auth.user === null), компонент возвращает fallback — обычно это индикатор загрузки. Если пользователь авторизован, компонент возвращает дочерние элементы (children), то есть отображает защищённый контент. Пример использования authGuard: описан ниже.

```
const Guard = ({children, authGuard, guestGuard}: GuardProps)
=> {
  if (guestGuard) {
    return <GuestGuard
fallback={<Spinner/>}>{children}</GuestGuard>
  }
  if (authGuard) {
    return <AuthGuard
fallback={<Spinner/>}>{children}</AuthGuard>
  }
  return <>{children}</>
}
```

Для страниц, требующих авторизации, устанавливается флаг `authGuard: true`, и страница автоматически оборачивается в компонент `AuthGuard`.

Компонент `AuthGuard` реализует типовой паттерн защиты маршрутов в одностраничных приложениях. Его работа основана на следующих принципах:

- централизованная проверка авторизации — все защищённые страницы используют единый механизм проверки;
- прозрачность для пользователя — если пользователь не авторизован, он автоматически перенаправляется на страницу входа, а после успешной авторизации возвращается на исходную страницу;
- гибкость — компонент легко интегрируется в архитектуру приложения и может быть использован для защиты любых частей интерфейса.

Данная реализация обеспечивает безопасность и удобство работы с защищёнными разделами веб-приложения, минимизируя риски несанкционированного доступа.

#### 1.4.2 Реализация основных страниц

Главная страница (`@src/pages/index.tsx`) или "root"-страница — это точка входа в приложение. В текущей реализации она служит как промежуточный экран загрузки. На ней отображается только компонент `Spinner`, который визуализирует процесс загрузки. Такая реализация часто используется, когда требуется дождаться инициализации данных пользователя или перенаправления на другую страницу (например, профиль или дашборд). Главная страница не содержит бизнес-логики и служит технической точкой входа, листинг продемонстрирован ниже.

```
import Spinner from "../../core/components/spinner";

const Home = () => {
  return <><Spinner/></>
}
```

```
export default Home
```

Страница входа (src/pages/login/index.tsx) предназначена для аутентификации пользователя — ввода email и пароля для получения доступа к защищённым разделам приложения.

Ключевые элементы:

- использование библиотеки react-hook-form для управления формой;
- валидация данных с помощью схемы yup;
- визуальные компоненты Material UI для современного интерфейса;
- кастомные стили для адаптивности и эстетики.

Фрагмент кода страницы входа:

```
const schema = yup.object().shape({
  email: yup.string().email('Введите корректный email').required('Email - обязательное поле'),
  password: yup.string().min(5, 'Пароль должен содержать минимум 5 символов').required('Пароль - обязательное поле')
})

const LoginPage = () => {
  const [showPassword, setShowPassword] =
  useState<boolean>(false)
  const auth = useAuth()
  const theme = useTheme()
  const { control, setError, handleSubmit, formState: {errors}
} = useForm({
  defaultValues,
  mode: 'onBlur',
  resolver: yupResolver(schema)
})

  const onSubmit = (data: FormData) => {
    const {email, password} = data
    auth.login({email, password}, () => {
      setError('email', {
```

```

        type: 'manual',
        message: 'Неверная почта или пароль'
    })
  })
}

```

Визуальное представление страницы отображено на рисунке 4.

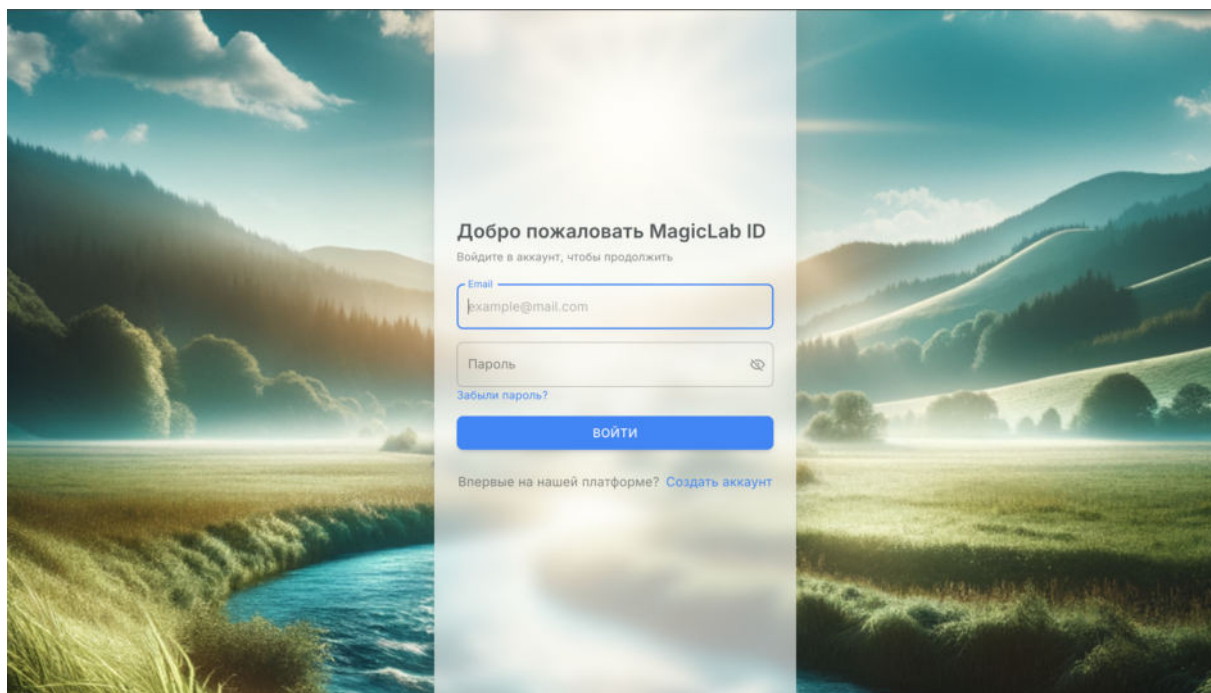


Рисунок 4 – Страница входа

Форма входа реализована с учётом всех современных требований к UX: валидация, обработка ошибок, адаптивность. При успешном вводе данных вызывается метод `auth.login`, который инициирует процесс аутентификации. В случае ошибки (например, неверный email или пароль) пользователю выводится соответствующее сообщение.

Страница регистрации (`src/pages/register/index.tsx`) позволяет новому пользователю создать учётную запись, введя необходимые данные. Ключевые элементы:

- использование `react-hook-form` для управления формой;
- валидация всех полей (имя, фамилия, email, пароль, подтверждение пароля).
- проверка совпадения пароля и его подтверждения.

- современный дизайн с использованием Material UI.

Фрагмент кода страницы регистрации:

```
const Register = () => {
  const [showPassword, setShowPassword] =
  useState<boolean>(false)
  const auth = useAuth()
  const theme = useTheme()
  const { register, handleSubmit, formState: {errors}, watch }
  = useForm<FormData>()

  const onSubmit = (data: FormData) => {
    auth.register(data)
  }
}
```

Визуальное представление страницы отображено на рисунке (5).

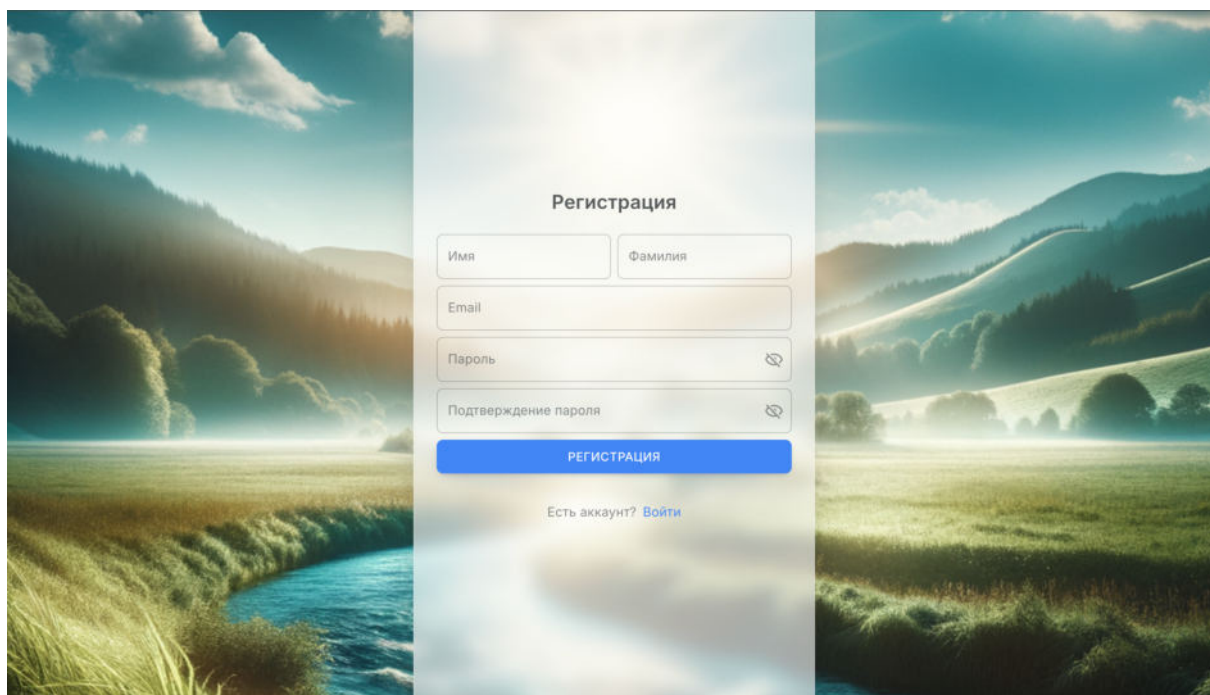


Рисунок 5 – Страница регистрации

Все поля формы обязательны для заполнения, что контролируется через валидацию. Для пароля реализована возможность показать/скрыть вводимые символы. При успешной регистрации вызывается метод `auth.register`, который

отправляет данные на сервер и автоматически авторизует пользователя. В случае ошибок пользователю выводятся подробные сообщения.

Страница профиля пользователя (src/pages/profile/index.tsx) отображает персональную информацию пользователя, его активность, проекты, команды и другие связанные данные. Фрагмент кода страницы профиля продемонстрирован ниже.

```
const Profile = () => {
  const theme = useTheme()
  const {settings} = useSettings()
  const dispatch = useDispatch<AppDispatch>()
  const store = useSelector((state: RootState) => state.user)
  const {skin} = settings

  return (
    <Box
      className='app-chat'
      sx={{
        width: '100%',
        borderRadius: 1,
        overflow: 'hidden',
        display: 'flex',
        justifyContent: 'center',
        position: 'relative',
        backgroundColor: theme.palette.action.hover,
        boxShadow: skin === 'bordered' ? 0 : 6,
        ...(skin === 'bordered' && {border: `1px solid
${theme.palette.divider}`})
      }}
    >
      <ProfileContent
        store={store}
        dispatch={dispatch}
        getInitials={getInitials}
      />
    </Box>
  )
}
```

```

    </Box>
  )
}

```

ProfileContent — это основной компонент, отвечающий за отображение информации о пользователе на странице профиля. Он визуализирует ключевые персональные данные, роль, email, а также (опционально) детали тарифного плана пользователя.

Компонент получает данные пользователя через пропсы. Отображает аватар, имя, фамилию, роль и email пользователя. Имеет задел для отображения информации о тарифном плане и управления им. Использует современные компоненты Material UI для стилизации и адаптивности. Легко расширяется для добавления новых секций профиля. Ключевой фрагмент кода компонента ProfileContent продемонстрирован ниже.

```

<Grid container spacing={6} sx={{padding: {sx: 0, md: 6},
justifyContent: 'center'}}>
  <Grid item xs={12} md={6} sx={{height: '100%'}}>
    <Card sx={{ ... }}>
      <CardContent sx={{pt: 15, display: 'flex', alignItems:
'center', flexDirection: 'column'}}>
        <CustomAvatar ...>{getInitials(`${user?.name} || ''
${user?.surname} || ''`)}</CustomAvatar>
        <Typography variant='h6' sx={{mb: 2}}>{user?.name}
{user?.surname}</Typography>
        <CustomChip ... label={user?.role} ... />
      </CardContent>
      <CardContent>
        <Typography variant='h6'>Детали</Typography>
        <Divider sx={{mt: theme => `${theme.spacing(4)}
!important`}}/>
        <Box sx={{pt: 2, pb: 1}}>
          <Box sx={{display: 'flex', mb: 2.7}}>
            <Typography variant='subtitle2' sx={{mr: 2,
color: 'text.primary'}}>Email:</Typography>

```

```

        <Typography
variant='body2'>{user?.email}</Typography>
      </Box>

    </Box>

    </CardContent>
  </Card>
</Grid>
{ /* Правая колонка с тарифом — скрыта */ }
</Grid>

```

Визуальное представление страницы отображено на рисунке 6.

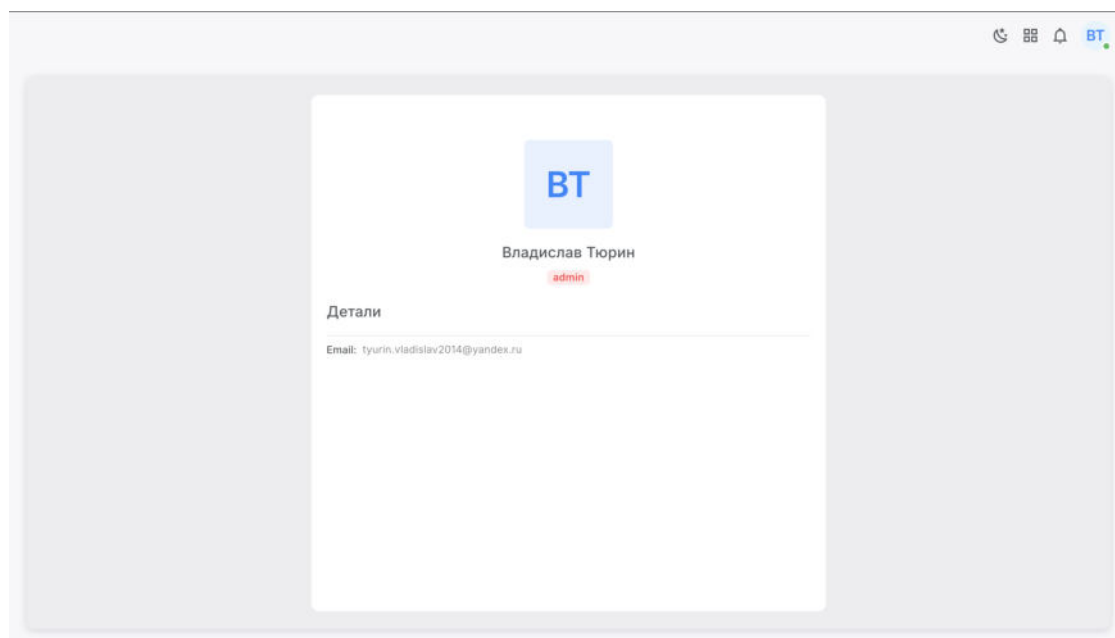


Рисунок 6 – Страница профиля

Реализация основных страниц приложения построена на современных подходах к разработке веб-интерфейсов:

- используются функциональные компоненты React и хуки для управления состоянием и жизненным циклом;
- для форм применяется библиотека react-hook-form с валидацией через уир;
- визуальная часть реализована с помощью Material UI, что обеспечивает адаптивность и единый стиль;

– все страницы интегрированы с системой авторизации и глобальным хранилищем данных, что обеспечивает безопасность и персонализацию пользовательского опыта.

### 1.4.3 Реализация взаимодействия с API

В современных веб-приложениях взаимодействие с сервером (API) — это фундаментальная часть, обеспечивающая обмен данными между клиентской частью (frontend) и серверной (backend). В вашем проекте для работы с API используется библиотека Axios, которая предоставляет удобный и мощный интерфейс для выполнения HTTP-запросов. Вся логика работы с API централизована в отдельном модуле, что позволяет:

- Стандартизировать обработку запросов и ответов;
- Централизованно управлять авторизацией;
- Легко масштабировать и поддерживать проект.

Пример кода базового HTTP-клиента продемонстрирован ниже.

```
import axios from 'axios'
import authConfig from "../configs/auth";

const http = axios.create({
  baseURL: process.env.NEXT_PUBLIC_BACKEND_API,
})
```

Метод `axios.create` — создаёт отдельный экземпляр клиента с базовым URL, который берётся из переменной окружения. Это позволяет легко переключаться между разными серверами (например, продакшн, тестовый, локальный) без изменения кода.

Такой подход гарантирует, что все запросы будут отправляться на нужный сервер. Пример кода интерцептора запросов показан ниже.

```
http.interceptors.request.use(
  (config) => {
    const token =
      localStorage.getItem(authConfig.storageTokenKeyName) || ''
```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                          | 35   |

```

        if (token) {
            config.headers['Authorization'] = 'Bearer ' +
token
        }
        return config
    },
    (error) => Promise.reject(error),
)

```

Интерцептор — это функция, которая автоматически вызывается перед каждым HTTP-запросом. Здесь происходит автоматическое добавление заголовка Authorization с токеном пользователя, если он есть в localStorage. Это позволяет не дублировать логику добавления токена в каждом запросе вручную. Если токена нет, запрос отправляется без заголовка авторизации (например, для публичных эндпоинтов). Пример кода интерцептора ответов ниже.

```

http.interceptors.response.use(
    (res) => res,
    (err) => {
        if (err.response && err.response.status === 401) {
            // Очистка токенов и данных пользователя при ошибке
авторизации
            localStorage.removeItem('access_token')

localStorage.removeItem(authConfig.storageTokenKeyName)
            localStorage.removeItem('crmType')
            localStorage.removeItem('userData')
            localStorage.removeItem('isCrmConnected')
            localStorage.removeItem('subscription')
            if (window.location.pathname !== '/login') {
                window.location.assign('/login')
            }
        }
        return Promise.reject(err)
    },
)

```

)

Интерцептор ответов позволяет централизованно обрабатывать ошибки, возникающие при выполнении запросов.

Если сервер возвращает ошибку 401 (Unauthorized), это означает, что токен пользователя истёк или невалиден.

В этом случае происходит автоматический "выход" пользователя: все связанные сессии и токены удаляются из localStorage, и пользователь перенаправляется на страницу входа.

Такой подход повышает безопасность приложения и предотвращает несанкционированный доступ. Универсальные функции для запросов продемонстрированы ниже.

```
const get = (url: string, headers: any, params = {}) => {  
  return http.get(url, {  
    params: {...params},  
    headers: {  
      ...headers,  
    },  
  })  
}
```

```
const post = (url: string, data: any, headers = {}, params =  
{}) => {  
  return http.post(url, data, {  
    ...params,  
    headers: {  
      ...headers,  
    },  
  })  
}
```

```
const put = (url: string, data: any, headers = {}) => {  
  return http.put(url, data, {  
    headers: {
```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 37   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        ...headers,
    },
})
}

const remove = (url: string, data: any, headers = {}, params
= {}) => {
    return http.delete(url, {
        params: {...params},
        headers: {
            ...headers,
        },
        data,
    })
}

const patch = (url: string, data: any, headers = {}) => {
    return http.patch(url, data, {
        headers: {
            ...headers,
        },
    })
}

```

Для каждого типа HTTP-запроса реализованы отдельные функции-обёртки.

Это позволяет вызывать API в других частях приложения с минимальным количеством кода, не заботясь о деталях реализации. Все функции возвращают промисы, что позволяет использовать их с `async/await` и обрабатывать результаты асинхронно.

Реализация конкретных запросов к API. В этом модуле реализованы функции для работы с аутентификацией и профилем пользователя:

```

import axios from 'axios'
import http from '../http'

```

```

const domain = process.env.NEXT_PUBLIC_BACKEND_API
export const registerUser = async (userData: any) => {
  const res = await
  axios.post(`${domain}/v1/identity/register`, userData, {})
  return res.data
}

export const resetPassword = async (data: any) => {
  const res = await axios.post(`${domain}/v1/identity/reset`,
data, {});

  return res.data;
};

export const sendResetCode = async (email: string) => {
  const res = await
  axios.post(`${domain}/v1/identity/reset/send-code`, {email:
email}, {});

  return res.data;
};

export const loginUser = async (data: any) => {
  const res = await axios.post(`${domain}/v1/identity/login`,
data, {})

  return res.data
}

export const logoutUser = async () => {
  const res = await http.post('/api/v1/auth/logout', {})

  return res.data
}

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 39   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```
export const fetchUser = async () => {
  const res = await http.get('/v1/identity/profile', {})

  return res.data
}
```

Контекст авторизации инкапсулирует всю логику работы с API, связанную с пользователем.

Пример кода использования API:

```
const initAuth = async (): Promise<void> => {
  const storedToken =
window.localStorage.getItem(authConfig.storageTokenKeyName)
  if (storedToken) {
    setLoading(true)
    try {
      const res = await fetchUser()
      localStorage.setItem('userData', JSON.stringify(res))
      setUser(res)
      dispatch(setUserStore(res))
    } finally {
      setLoading(false)
    }
  } else {
    setLoading(false)
  }
}
```

При запуске приложения или после успешного входа/регистрации вызывается функция `initAuth`. Если в `localStorage` есть токен, отправляется запрос на получение профиля пользователя. Если запрос успешен, данные пользователя сохраняются в состоянии и локальном хранилище. Пример кода функции регистрации пользователя показан ниже.

```
const handleRegister = async (params: RegisterParams) => {
  try {
```

```

const data = {
  email: params.email,
  name: params.firstName,
  surname: params.lastName,
  password: params.password,
  phone: params.phone
}

const res = await registerUser(data)
localStorage.setItem(authConfig.storageTokenKeyName,
res.token)

await initAuth()
await router.replace('/profile')
} catch (e) {
  console.log(e)
}
}

```

После успешной регистрации токен сохраняется, инициализируется сессия, пользователь перенаправляется на страницу профиля. Пример кода входа пользователя показан ниже.

```

const handleLogin = async (params: LoginParams,
errorCallback?: ErrCallbackType) => {
  try {
    const data = {
      email: params.email,
      password: params.password,
    }

    const res = await loginUser(data)
    localStorage.setItem(authConfig.storageTokenKeyName,
res.token)

    await initAuth()
    const returnUrl = router.query.returnUrl
    if (returnUrl) {

```

```

        await
router.replace(`${returnUrl}?token=${res.token}`)

    } else {
        await router.replace('/profile')
    }
} catch (err) {
    if (errorCallback) errorCallback(err as { [key: string]:
string });
}
}

```

После успешного входа токен сохраняется, инициализируется сессия, пользователь перенаправляется на нужную страницу. Пример кода выхода пользователя показан ниже.

```

const handleLogout = async () => {
    setUser(null)
    window.localStorage.removeItem('userData')

window.localStorage.removeItem(authConfig.storageTokenKeyName)
    await router.push('/login')
    setLoading(false)
}

```

При выходе данные пользователя и токен удаляются, пользователь перенаправляется на страницу входа.

Преимуществами данной архитектуры являются централизация всей логики работы с API, она вынесена в отдельные модули, что облегчает поддержку и развитие проекта, безопасность так , как токен автоматически добавляется ко всем защищённым запросам, а при его истечении пользователь выходит из системы, масштабируемость, потому что для добавления новых API-методов достаточно реализовать новую функцию в соответствующем модуле и переиспользуемость функций для работы с API, они могут быть

использованы в любом компоненте приложения. На рисунке 6 продемонстрирована схема полного цикла взаимодействия.

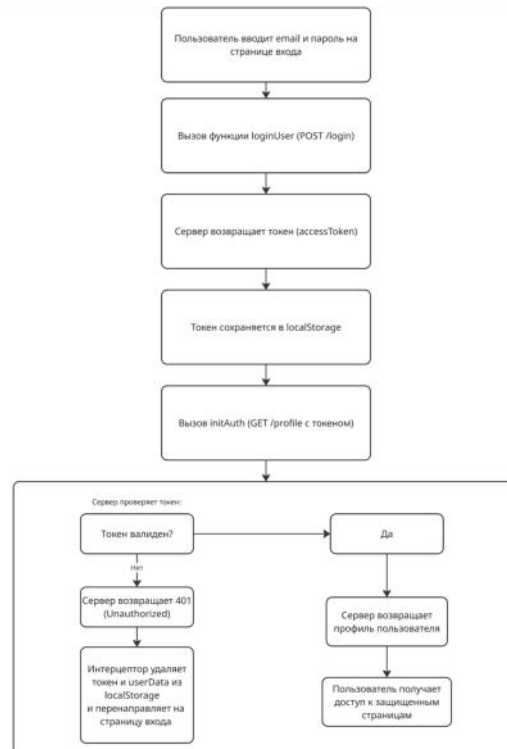


Рисунок 6 – Пример полного цикла взаимодействия

## 1.5 Разработка модуля CRM

### 1.5.1 Реализация authGuard

В модуле CRM тоже присутствует authGuard, но с некоторыми особенностями. Поскольку это полноценный отдельный модуль, у него отсутствуют страницы входа, регистрации и сброса пароля, этим занимается модуль ID. Поэтому для аутентификации пользователя он будет перенаправляться в модуль ID для дальнейшей авторизации, будешь возвращен на адрес с которого был перенаправлен с query параметром token, который нужен для идентификации пользователя и входа в систему. Пример кода authGuard продемонстрирован ниже.

```

import {ReactElement, ReactNode, useEffect, useState} from
'react'

import {useRouter} from 'next/router'

import {useAuth} from 'src/hooks/useAuth'
  
```

```

import {RootState} from "../../../../../store";
import {useSelector} from "react-redux";

interface AuthGuardProps {
  children: ReactNode
  fallback: ReactElement | null
}

const AuthGuard = (props: AuthGuardProps) => {
  const {children, fallback} = props
  const auth = useAuth()
  const router = useRouter()
  const store = useSelector((state: RootState) => state.user)
  const user = store?.user
  const [minFallbackTimePassed, setMinFallbackTimePassed] =
    useState(false)

  useEffect(() => {
    const timer = setTimeout(() => {
      setMinFallbackTimePassed(true)
    }, 1800)

    return () => clearTimeout(timer)
  }, [])

  useEffect(() => {
    if (!router.isReady) {
      return
    }

    if (auth.user === null && !user && !auth.loading) {
      window.location.href =
        `https://id.magiclab.space/login?returnUrl=${encodeURIComponent(window.location.href)}`;
    }

    }, [router.route, auth.loading])

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 44   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

    if (auth.loading || auth.user === null ||
!minFallbackTimePassed) {

        return fallback
    }

    return <>{children}</>
}

export default AuthGuard

```

### 1.5.2 Реализация основных страниц и компонентов

Рабочее пространство является центральным узлом CRM-системы, обеспечивающим взаимодействие пользователя с основными функциями управления продажами, клиентами и задачами. Его разработка велась с использованием библиотеки React и компонентного фреймворка Material-UI, что позволило создать интуитивно понятный и отзывчивый интерфейс.

Воронки продаж — один из ключевых инструментов CRM, позволяющий визуализировать этапы сделок и контролировать их выполнение. В системе реализованы следующие функции:

- пользователь может добавлять новые воронки, задавая их название, этапы и цветовую маркировку. Для выбора цвета использован компонент HexColorPicker, обеспечивающий удобный интерфейс настройки визуального отображения;
- перед удалением система запрашивает подтверждение через модальное окно, что предотвращает случайную потерю данных;
- каждая воронка может включать произвольное количество этапов, которые можно переименовывать, менять местами и настраивать под конкретные бизнес-процессы.

Пример кода реализующий воронки продаж продемонстрирован ниже.

<Box

```

sx={{
  display: "flex",
  maxHeight: "90vh",
  minHeight: "90vh",

  alignItems: "start",
  borderRadius: '16px',
  borderTopLeftRadius: '0',
  borderTopRightRadius: '0',
  overflowY: "hidden",
  flexDirection: "column",
  gap: 5,
  backgroundColor: theme.palette.background.paper
}}
>
<Dialog
  open={deleteDialogOpen}
  onClose={() => setDeleteDialogOpen(false)}
  aria-labelledby="alert-dialog-title"
  aria-describedby="alert-dialog-description"
>
  <DialogTitle id="alert-dialog-title">
    Подтверждение удаления
  </DialogTitle>
  <DialogContent>
    <DialogContentText id="alert-dialog-description">
      Вы уверены, что хотите удалить воронку
      "{pipelines.find(p => p.id === pipelineToDelete)?.name}"?
      Это действие нельзя отменить.
    </DialogContentText>
  </DialogContent>
  <DialogActions>
    <Button
      onClick={handleDeletePipeline}
      color="error"
    >

```

```

        variant="text"
        autoFocus
    >
        Удалить
    </Button>

<Button variant='outlined' onClick={() =>

setDeleteDialogOpen(false)}>Отмена</Button>
    </DialogActions>
</Dialog>

<Box sx={{
    borderBottom: `1px solid ${theme.palette.divider}`,
    borderTop: `1px solid ${theme.palette.divider}`,
    backgroundColor: theme.palette.background.paper,
    width: "100%",
    display: "flex",
    justifyContent: "space-between",
    pr: 8,
    alignItems: "center"
}}>
    <Box sx={{display: "flex", justifyContent: "space-
between"}}>
        <Box
            sx={{
                padding: 2,
                cursor: "pointer",
                display: "flex",
                gap: 1,
                alignItems: "center",
                justifyContent: "space-between",
                minWidth: '300px',
                maxWidth: '300px',
                height: '62px',

```

```

        borderRight:                                `1px                solid
    ${theme.palette.divider}` ,
    }}
    onClick={handleClick}
  >
    <Box sx={{display: "flex", alignItems: "center",

    justifyContent: "space-between", gap: 1,}}>
      {isLoading ? (
        <Skeleton          variant="circular"          width={18}
height={18}/>
      ) : localSelectedPipeline ? (
        <Box
          sx={{
            width: 18,
            height: 18,
            borderRadius: '3px',
            backgroundColor:
localSelectedPipeline.color,
            cursor:      editPipeline      ?      'pointer'      :
'default',

          }}
        />
      ) : null}

      {isLoading ? (
        <Skeleton width={150} height={24}/>
      ) : (
        <Typography
          variant="body1"
          sx={{
            maxWidth: '300px',
            whiteSpace: 'nowrap',
            textOverflow: 'ellipsis',
            overflow: 'hidden',

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 48   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        display: 'flex',
        alignItems: 'center',
        gap: 1
      }}
    >
    {localSelectedPipeline ? (
      <>

```

Воронка:

```

      <Typography
        component="span"
        variant="body2"
        sx={{
          whiteSpace: 'nowrap',
          textOverflow: 'ellipsis',
          overflow: 'hidden',
          maxWidth: '150px',
        }}
      >
        {localSelectedPipeline.name}
      </Typography>
    </>
  ) : (
    "Выберите воронку"
  )}
</Typography>
)}
</Box>
<Icon icon="mdi:chevron-down" fontSize="1.2rem"/>

</Box>

{(localSelectedPipeline && editPipeline) && (
  <Box

```

```

        sx={{
          padding: 2,
          display: "flex",
          alignItems: "center",
          gap: 1,
          height: '62px',
          cursor: editPipeline ? 'pointer' : 'default',
        }}

onClick={() => {
  setTempColor(localSelectedPipeline.color);
  setColorPickerOpen(true);
}}

>
<Typography variant="body2" sx={{mr: 1}}>
  Цвет:
</Typography>
<Box
  sx={{
    width: 24,
    height: 24,
    borderRadius: '4px',
    backgroundColor:
localSelectedPipeline.color,
    border: `1px solid ${theme.palette.divider}`,
  }}
/>
</Box>

)}

</Box>

<Popover
  open={colorPickerOpen}

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 50   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        anchorEl={colorPickerAnchorEl}
        onClose={handleColorPickerClose}
        anchorOrigin={{
            vertical: 'bottom',
            horizontal: 'left',
        }}
    >
    <Box sx={{p: 2}}>
        <HexColorPicker
            color={tempColor}
            onChange={setTempColor}
            style={{width: '200px', height: '200px'}}
        />
        <Box sx={{display: 'flex', justifyContent: 'flex-
end', mt: 2}}>
            <Button onClick={handleColorPickerClose}>
                Отмена
            </Button>
            <Button
                variant="contained"
                onClick={handleColorSave}
                sx={{ml: 1}}
            >
                Сохранить
            </Button>
        </Box>
    </Box>
</Popover>

<Popover
    id={id}
    open={open}
    anchorEl={anchorEl}
    onClose={handleClose}

```

```

        anchorOrigin={{
          vertical: "bottom",
          horizontal: "left",
        }}
        transformOrigin={{
          vertical: "top",
          horizontal: "left",
        }}
        sx={{
          top: '1px'
        }}
      }}

      PaperProps={{
        sx: {
          borderRadius: '5px',

minWidth: '300px',
        }
      }}
    >
    <Box sx={{p: 2, minWidth: 250}}>
      <Typography variant="body1" sx={{mb: 2}}>
        Воронки
      </Typography>

      {isLoading ? (
        <Box sx={{display: 'flex', flexDirection:
'column', gap: 1}}>
          [...Array(3)].map((_, index) => (
            <Box key={index} sx={{display: 'flex',
alignItems: 'center', gap: 1, py: 1}}>
              <Skeleton variant="rectangular" width={18}
height={18} sx={{borderRadius: '3px'}}/>
              <Skeleton width="70%" height={20}/>
            </Box>
          )
        )
      )}
    </Box>
  )}

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 52   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

    ))}
  </Box>
) : (
  <>
    <Box    sx={{maxHeight:    "300px",    overflowX:
"auto"}}>

      {pipelines.map((pipeline: Pipeline) => (
        <MenuItem
          key={pipeline.id}
          onClick={() =>
handleSelectPipeline(pipeline)}

          sx={{

            display: 'flex',
            alignItems: 'center',
            gap: 1,
            backgroundColor:    pipeline.id    ===
localSelectedPipeline?.id    ?    theme.palette.action.selected    :
'transparent',

            '&:hover': {
              backgroundColor:
theme.palette.action.hover,
            },
          }}
        >
          <Box
            sx={{
              width: 18,
              height: 18,
              borderRadius: '3px',
              backgroundColor: pipeline.color,
            }}
          />
          <Typography variant="body2">
            {pipeline.name}

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          |      |
| Изм. | Лист | № докум. | Подпись | Дата |                          | 53   |

```

        </Typography>
      </MenuItem>
    )))
  </Box>
  <Divider sx={{border: theme => `0.5px solid
    ${theme.palette.divider}`, my: 1}}/>

    {isAddingPipeline ? (
      <Box sx={{display: "flex", alignItems:
        "center", gap: 1, mt: 1}}>
        <TextField
          size="small"

          placeholder="Название воронки"
          value={newPipelineName}
          onChange={ (e) =>
            setNewPipelineName (e.target.value) }

          InputProps={{
            endAdornment: (
              <InputAdornment position="end">
                <IconButton
                  onClick={handleConfirmNewPipeline}>
                    <Icon icon="mdi:check"
                      fontSize="1.2rem"/>
                </IconButton>
              </InputAdornment>
            ),
          }}
        />
      </Box>
    ) : (
      <MenuItem
        onClick={handleCreateNewPipeline}

```

```

                sx={{display: "flex", alignItems: "center",
gap: 1}}
            >
            <Typography variant="body2">
                Создать воронку
            </Typography>
            <ListItemIcon>
                <Icon
                    icon="mdi:plus"
fontWeight='bold' />
            </ListItemIcon>
        </MenuItem>
    ) }
</>
) }

</Box>

</Popover>
<Box sx={{display: 'flex', gap: 1}}>
    {editPipeline ? (
        <>

        <Button
            variant="text"
            color="error"
            size="small"
            sx={{
                borderRadius: '5px',
                textTransform: 'none',
                marginY: 3,
            }}
            onClick={() => localSelectedPipeline    &&
openDeleteDialog(localSelectedPipeline.id) }
            startIcon={<Icon    icon="mdi:delete-outline"
fontSize="1.2rem"/>}
        >

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 55   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        Удалить воронку
    </Button>
    <Button
        variant="text"
        sx={{
            color: theme.palette.text.primary,
            borderRadius: '5px',
            textTransform: 'none',
            marginY: 3,
            '&:hover': {
                backgroundColor:
theme.palette.action.hover
            }
        }}
        onClick={() => setEditPipeline(false)}

        endIcon={
            <svg width="20" height="20" viewBox="0 0 24
24" fill="none" xmlns="http://www.w3.org/2000/svg">
                <path d="M4 12.0002H20M20 12.0002L14
6.00024M20 12.0002L14 18.0002" stroke="#3c4043de"
                stroke-width="1.5"
                stroke-linecap="round" stroke-
linejoin="round"/>
            </svg>
        }
    >
        Назад
    </Button>
</>

```

Для удобного управления сделками внедрена канбан-доска, позволяющая перемещать карточки между этапами с помощью drag-and-drop.

Основные особенности:

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 56   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

- при перемещении карточки автоматически синхронизируются с сервером, обеспечивая актуальность информации;
- пользователь может фильтровать сделки по статусу, дате, ответственному менеджеру или другим параметрам;
- доска корректно отображается как на широкоформатных мониторах, так и на мобильных устройствах.

Пример кода реализации канбан-доски продемонстрирован ниже.

```
<DndContext
  sensors={sensors}
  onDragStart={onDragStart}
  onDragEnd={onDragEnd}
  onDragOver={onDragOver}
>
  <Box sx={{
    margin: '0 auto',
    display: 'flex',
    gap: 4,
    height: '100%',
    paddingX: '16px',
    backgroundColor: theme.palette.background.paper
  }}>
    {isLoading ? (
      <Box sx={{display: 'flex', justifyContent: 'center',
width: '100%', py: 4}}>
        {renderSkeletonColumns()}
      </Box>
    ) : (
      <>
        <Box sx={{display: 'flex', gap: 4, height:
'100%'}}>
          <SortableContext items={columnsId}>
            {columns.map((col) => (
              <ColumnContainer
```

```

        key={col.id}
        column={col}
        deleteColumn={deleteColumn}
        updateColumn={updateColumn}
        createTask={createTask}
        deleteTask={deleteTask}
        updateTask={updateTask}
        tasks={tasks.filter((task) => task.stageId
=== col.id)}

        crmId={selectedCRM?.id.toString()}

        pipelineId={selectedPipeline?.id.toString()}
        editPipeline={editPipeline}
    />
    )})
</SortableContext>
</Box>

<Button
    sx={{minWidth: '300px', maxHeight: '41px', mt:
'6px', borderRadius: '5px'}}
    onClick={createNewColumn}
    disabled={isLoading}
    startIcon={<Icon icon='mdi:plus' />}

>
    Добавить статус
</Button>
</>
)}
</Box>

{createPortal(
    <DragOverlay>
        {activeColumn && (

```

```

        <ColumnContainer
            key={activeColumn.id}
            column={activeColumn}
            deleteColumn={deleteColumn}
            updateColumn={updateColumn}
            createTask={createTask}
            deleteTask={deleteTask}
            updateTask={updateTask}
            tasks={tasks.filter((task) => task.stageId ===
activeColumn.id)}
            crmId={selectedCRM.id.toString()}
            pipelineId={selectedPipeline.id.toString()}
            editPipeline={editPipeline}
        />
    )}
    {activeTask && (
        <TaskCard
            task={activeTask}

deleteTask={deleteTask}
        />
    )}
</DragOverlay>,
document.body
    )}
</DndContext>

```

CRM-система поддерживает подключение внешних сервисов, что расширяет ее функциональность. Реализованы следующие механизмы:

- Telegram-бот: позволяет получать уведомления о новых сделках и изменениях статусов напрямую в мессенджер;
- для каждой интеграции можно сгенерировать токен доступа, а также отозвать его при необходимости;
- все подключенные сервисы отображаются в отдельной панели, где

можно настраивать их параметры;

На рисунке 7 продемонстрирован интерфейс страницы /workspace:



Рисунок 7 – Интерфейс страницы /workspace

Страница карточки сделки представляет собой сложный компонент, сочетающий в себе функциональность управления сделкой и коммуникации между участниками. Рассмотрим ключевые аспекты реализации этого модуля. Компонент DealCard реализован с использованием современных подходов

React-разработки:

- типизированный код с использованием TypeScript;
- клиентский рендеринг (директива 'use client');
- состояние компонента управляется через React-хуки;
- использование Material-UI для UI-компонентов;
- интеграция с WebSocket для общения с клиентами в реальном[6] времени;
- разделение на логические блоки.

Компонент реализует сложную логику управления состоянием через несколько хуков, код которых продемонстрирован ниже.

```
const [deal, setDeal] = useState<Deal | null>(null) // Данные сделки
```

```

const [loading, setLoading] = useState(true) // Статус загрузки
const [stages, setStages] = useState<Stage[]>([]) // Список этапов
const [messages, setMessages] = useState<Message[]>([]) // Сообщения чата
const [connectionStatus, setConnectionStatus] =
  useState('disconnected')

```

Реализация WebSocket включает несколько ключевых аспектов:

- создается при монтировании компонента;
- использует токен аутентификации из localStorage;
- передает идентификаторы сделки, CRM и воронки;
- реализует механизм повторного подключения.

Обработчики событий продемонстрированы ниже.

```

socket.onopen = () => {
  setConnectionStatus('connected')
  // Инициализационные действия
}

socket.onmessage = event => {
  // Обработка входящих сообщений
  setMessages(prev => [...prev, newMessage])
}

socket.onerror = error => {
  setConnectionStatus('error')
}

socket.onclose = () => {
  setConnectionStatus('disconnected')
}

```

На рисунке 8 представлена UML-диаграмма последовательности, отражающая пошаговое взаимодействие компонентов системы при выполнении типовых операций. Диаграмма наглядно демонстрирует процесс обработки запросов от пользовательского интерфейса до базы данных и обратно, включая этапы авторизации, валидации и обработки данных. Особое внимание на диаграмме уделено временной последовательности операций и обмену сообщениями между компонентами системы.



Рисунок 8 – UML- диаграмма последовательности

Рисунок 9 содержит UML-диаграмму компонентов, которая визуализирует структурную организацию системы. На диаграмме четко прослеживается разделение на фронтенд и бэкенд компоненты, а также их взаимосвязи через определенные интерфейсы. Особый интерес представляют компоненты интеграции с внешними сервисами и модули работы с данными, которые обеспечивают основные бизнес-функции системы.

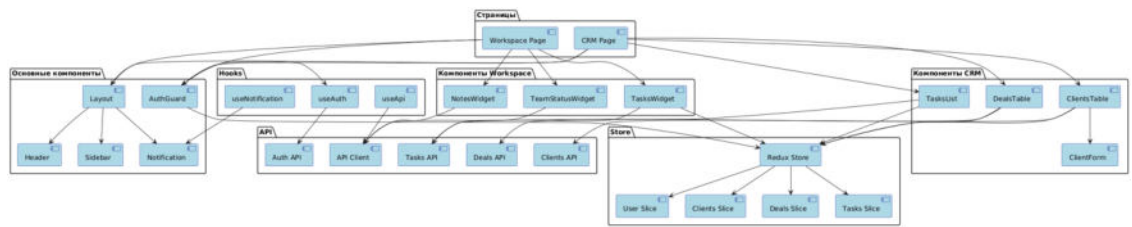


Рисунок 9 – UML- диаграмма компонентов

UML-диаграмма вариантов использования, представленная на рисунке 10, охватывает все ключевые сценарии взаимодействия пользователей с системой.

Диаграмма включает основные акторы и варианты использования, такие как управление сделками, работа с воронками продаж и интеграционные возможности. Визуальное представление позволяет быстро оценить функциональные возможности системы и взаимодействие между различными ролями пользователей.

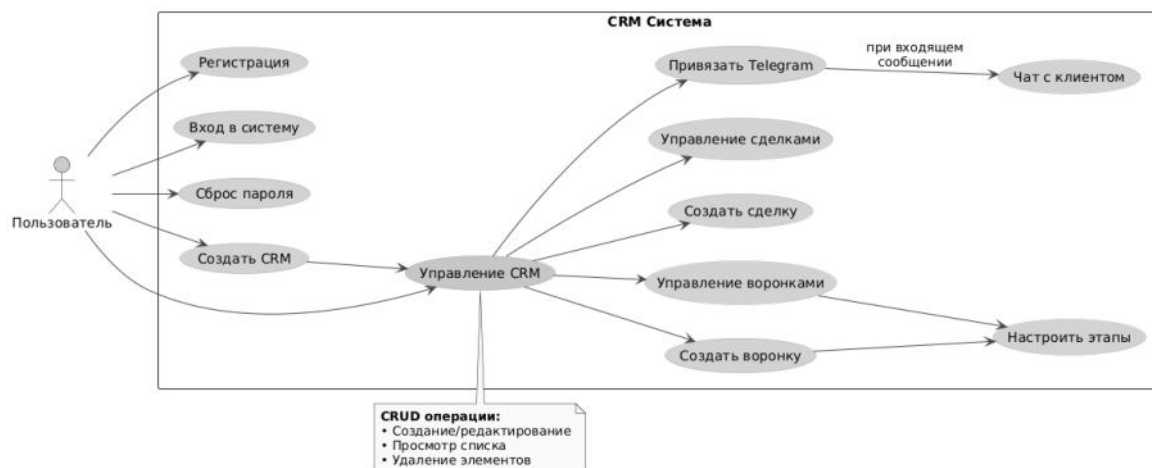


Рисунок 10 – UML- диаграмма вариантов использования

## 1.6 Архитектура развертывания и эксплуатационные требования

### 1.6.1 Конвейер непрерывной интеграции и доставки (CI/CD)

Реализованный процесс автоматизированного развертывания включает следующие этапы:

- подготовка среды:
  - 1) установка необходимых утилит (rsync, openssh-client);
  - 2) настройка SSH-агента для безопасного подключения
  - 3) добавление приватного ключа в SSH-агент (с обработкой формата)
  - 4) конфигурация known\_hosts для верификации сервера
- процедура деплоя:
  - 1) установка корректного PATH для Node.js;
  - 2) переход в рабочую директорию проекта

(~/magiclab/crm/crm\_frontend);

- 3) обновление кода (переключение на ветку main и pull изменений);
- 4) рестарт Docker-контейнеров с пересборкой образов;
- триггеры выполнения:

- 1) автоматический запуск только при изменениях в ветке main.

## 1.6.2 Docker-инфраструктура

Docker-инфраструктура построена на базе официального образа Node.js 20.15.1 LTS-версии с использованием многоэтапной сборки для оптимизации конечного размера образа. Процесс сборки начинается с копирования файлов зависимостей `package.json` и `package-lock.json`, после чего выполняется установка пакетов с флагом `--force` для обработки потенциальных конфликтов версий. Затем копируются все исходные файлы проекта, включая конфигурационные файлы окружения. На этапе сборки запускаются скрипты `build` и `lint` для создания `production`-сборки и проверки качества кода. Готовое приложение запускается на порту 3131 в `production`-режиме, что явно указано через переменную окружения `NODE_ENV`.

Конфигурация `docker-compose` использует версию 3.8. Подробная UML-диаграмма разворачивания продемонстрирована на рисунке 11.

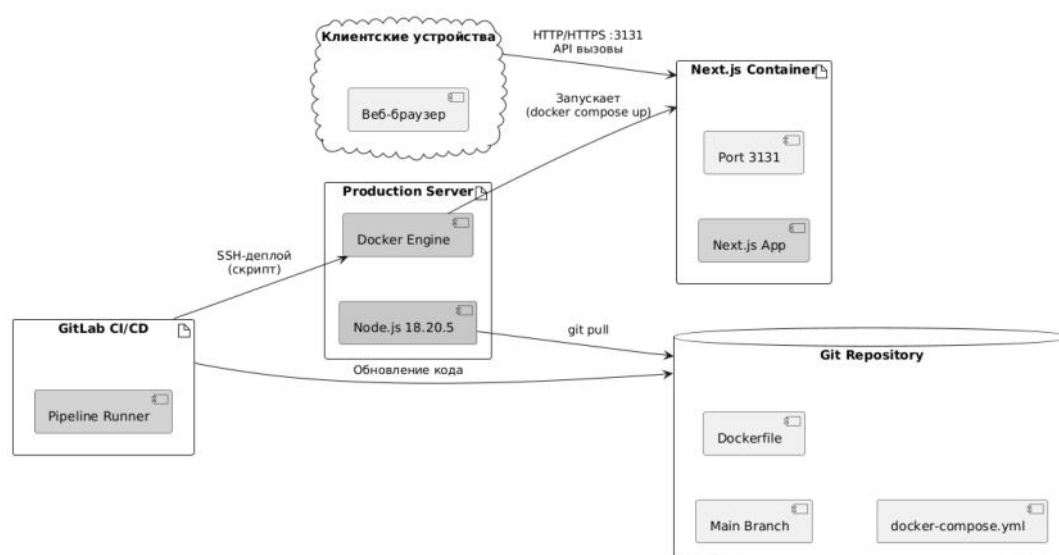


Рисунок 11 – UML- диаграмма разворачивания

## 2 Безопасность при работе с цифровыми устройствами

Многие недооценивают риски, связанные с ежедневным использованием компьютеров и смартфонов. На практике длительная работа с электронными устройствами действительно может негативно влиять на здоровье, если не соблюдать базовые правила безопасности.

### 2.1 Как безопасно работать за компьютером

Современные исследования и нормативные документы (включая ГОСТ 12.0.003-2015) выделяют несколько основных факторов риска:

- перегрев оборудования и рабочей зоны;
- статическое электричество при контакте с техникой;
- электромагнитное излучение от работающих устройств;
- усталость глаз из-за постоянного напряжения;
- проблемы с осанкой при неправильном положении тела;
- монотонность операций, ведущая к переутомлению.

Практические советы:

- организуйте рабочее место так, чтобы монитор находился на расстоянии вытянутой руки, с небольшим наклоном;
- используйте кресло с регулируемой высотой и поддержкой поясницы;
- следите за освещением - свет должен быть достаточно ярким, но не создавать бликов на экране;
- делайте перерывы каждые 45-50 минут для разминки;
- размещайте системный блок так, чтобы задняя панель не была направлена на рабочее место.

Важно помнить: при появлении любых неисправностей оборудования (перегрев, странные звуки, запах гари) следует немедленно отключить устройство от сети.

## 2.2 Безопасность использования смартфонов

Разрабатываемое веб-приложение будет доступно и на мобильных устройствах, поэтому стоит учитывать особенности их эксплуатации:

Основные рекомендации:

- используйте только оригинальные зарядные устройства;
- избегайте перегрева - не оставляйте телефон под прямыми солнечными лучами или под подушкой во время зарядки;
- не подвергайте устройство механическим повреждениям;
- с осторожностью относитесь к использованию телефона во время грозы.

Для сохранения здоровья:

- ограничивайте время непрерывного использования;
- при длительных разговорах используйте гарнитуру;
- не носите включенный телефон в карманах одежды рядом с телом;
- соблюдайте "цифровую гигиену" - делайте перерывы в работе с устройством.

Хотя современные исследования пока не подтвердили однозначного вреда от электромагнитного излучения смартфонов, разумные меры предосторожности не будут лишними.

Следование этим простым правилам поможет сохранить не только работоспособность техники, но и ваше собственное здоровье при ежедневной работе с цифровыми устройствами.

## Заключение

В процессе разработки дипломного проекта была создана CRM-система для управления продажами и клиентскими отношениями. В работе применялись современные технологии веб-разработки:

Для реализации фронтенд-части использовался JavaScript-фреймворк React в сочетании с библиотекой Material-UI, что позволило создать интуитивно понятный и отзывчивый пользовательский интерфейс. Бэкенд-часть разрабатывалась на платформе Node.js с использованием Express.js, обеспечивающей высокую производительность обработки запросов.

Система хранения данных реализована на PostgreSQL - надежной реляционной СУБД, оптимально подходящей для хранения структурированной информации о клиентах и сделках. Для развертывания приложения применялась технология Docker, позволяющая обеспечить согласованную работу всех компонентов системы в различных окружениях.

Ключевые функциональные возможности разработанного решения включают:

- управление воронками продаж с настраиваемыми этапами;
- создание и отслеживание сделок;
- интеграцию с мессенджерами (Telegram);
- ведение переписки с клиентами непосредственно в системе;
- гибкую систему отчетности и аналитики.

Особенностью системы является продуманный UX/UI-дизайн, обеспечивающий комфортную работу пользователей. Реализованный механизм перетаскивания карточек сделок между этапами воронки значительно упрощает процесс управления продажами.

Перспективы развития проекта включают:

- Расширение списка интеграций с популярными сервисами;
- Внедрение системы уведомлений о важных событиях;
- Разработку мобильной версии приложения;

- Добавление инструментов для командной работы;
- Внедрение системы аналитики и прогнозирования.

Разработанная CRM-система полностью соответствует поставленным задачам и может быть успешно внедрена в работу малого и среднего бизнеса для автоматизации процессов продаж и повышения эффективности работы с клиентами.

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          |      |
| Изм. | Лист | № докум. | Подпись | Дата |                          | 68   |

### Список использованных источников

1. Официальная документация React: [Электронный ресурс]. URL: <https://react.dev/learn> (дата обращения: 15.05.2025).
2. Material-UI документация [Электронный ресурс]. URL: <https://mui.com/material-ui/getting-started/> (дата обращения: 15.05.2025).
3. Работа с API в React Habr: [Электронный ресурс]. URL: <https://habr.com/ru/articles/725530/> (дата обращения: 18.05.2025)
4. Лучшие практики фронтенд-разработки MDN Web Docs: [Электронный ресурс]. URL: [https://developer.mozilla.org/ru/docs/Learn/Tools\\_and\\_testing/Client-side\\_JavaScript\\_frameworks](https://developer.mozilla.org/ru/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks)(дата обращения: 21.05.2025)
5. State-менеджмент с Redux Toolkit [Электронный ресурс]. URL: <https://redux-toolkit.js.org/> (дата обращения: 20.05.2025)
6. Интеграция WebSocket в React: [Электронный ресурс]. URL: <https://socket.io/how-to/use-with-react> (дата обращения: 23.05.2025)

## Приложение А

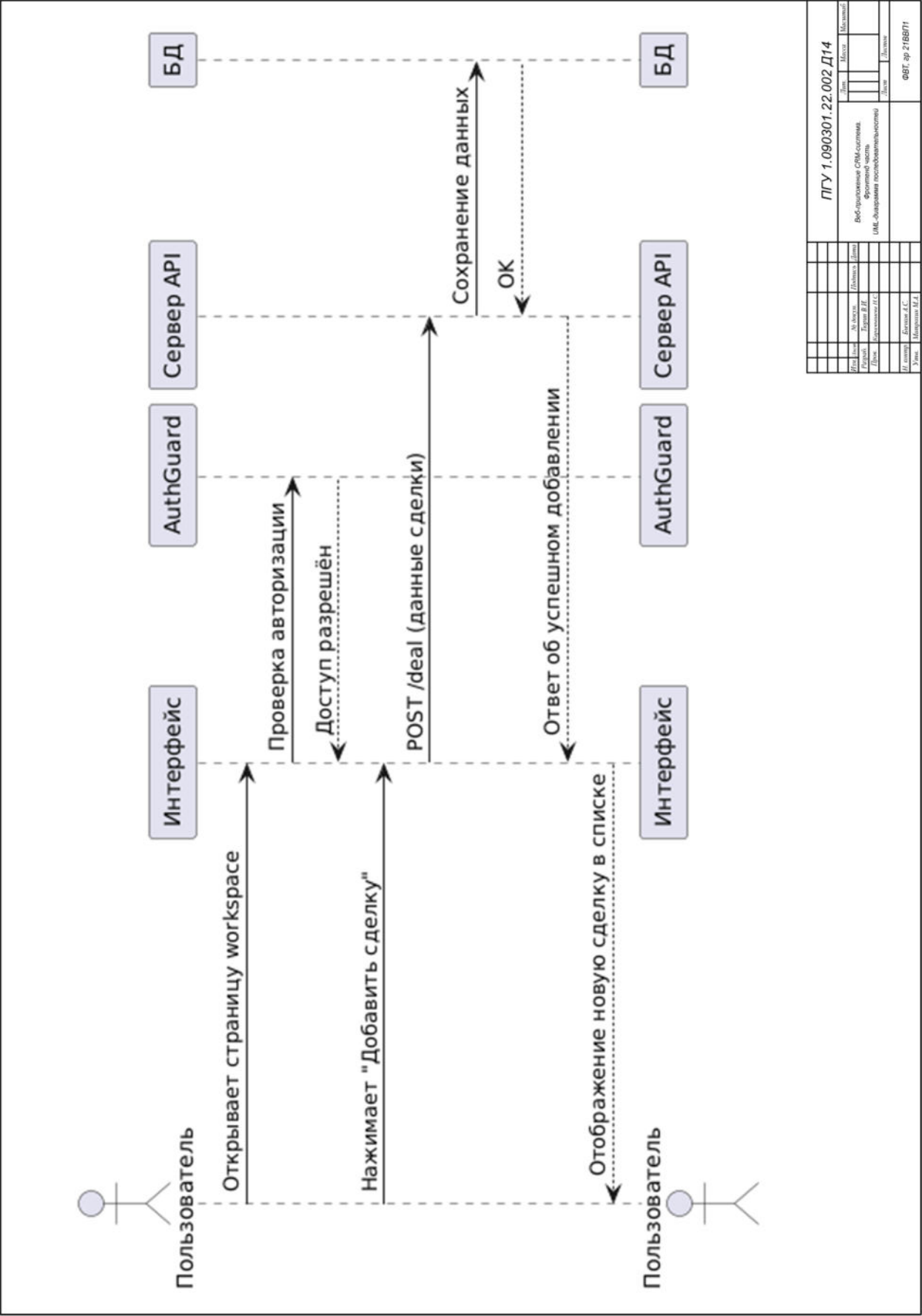
(обязательное)

Веб-приложение CRM-система. Фронтенд часть

UML-диаграммы

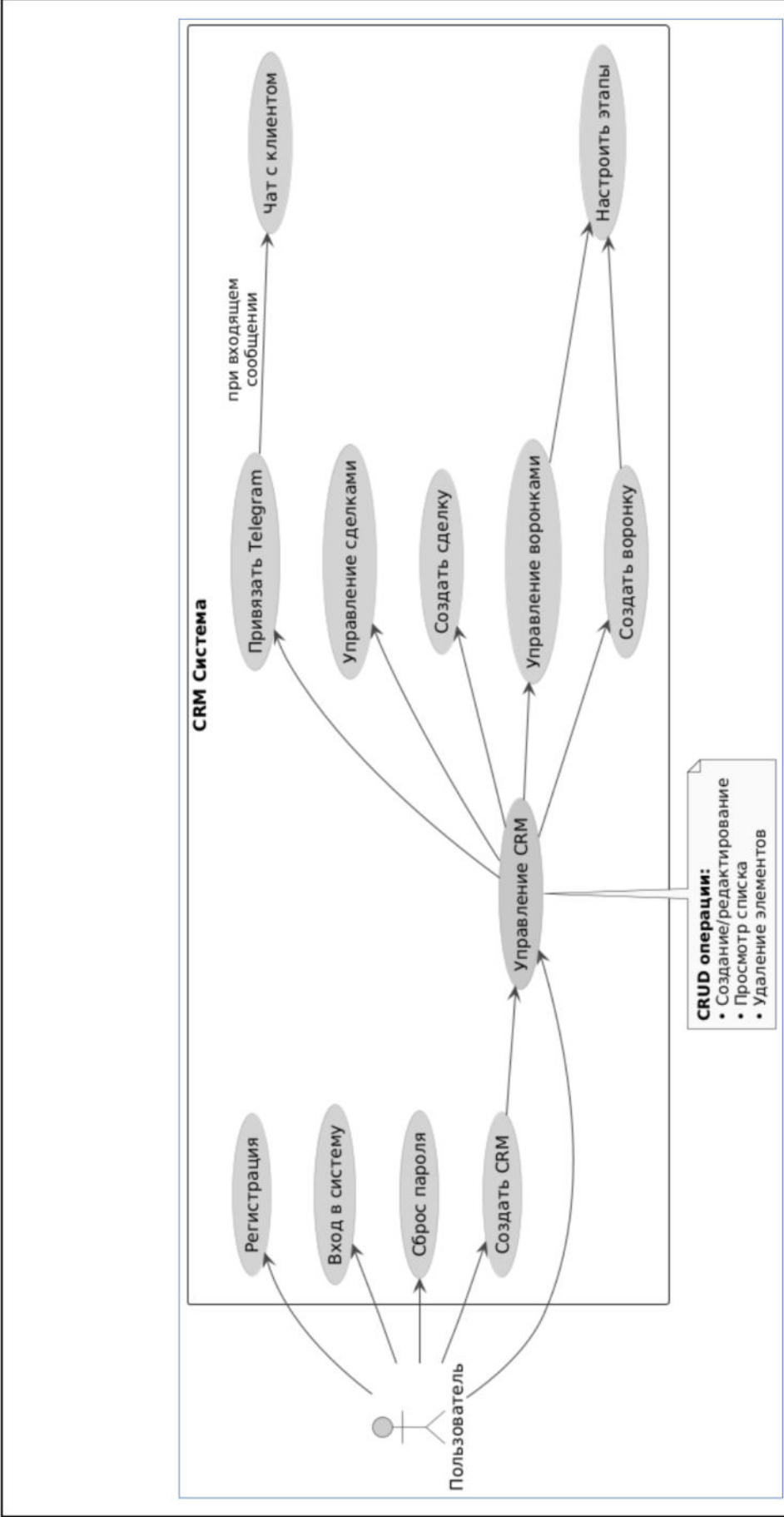
|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 70   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

Приложение А.1-Диаграмма последовательности



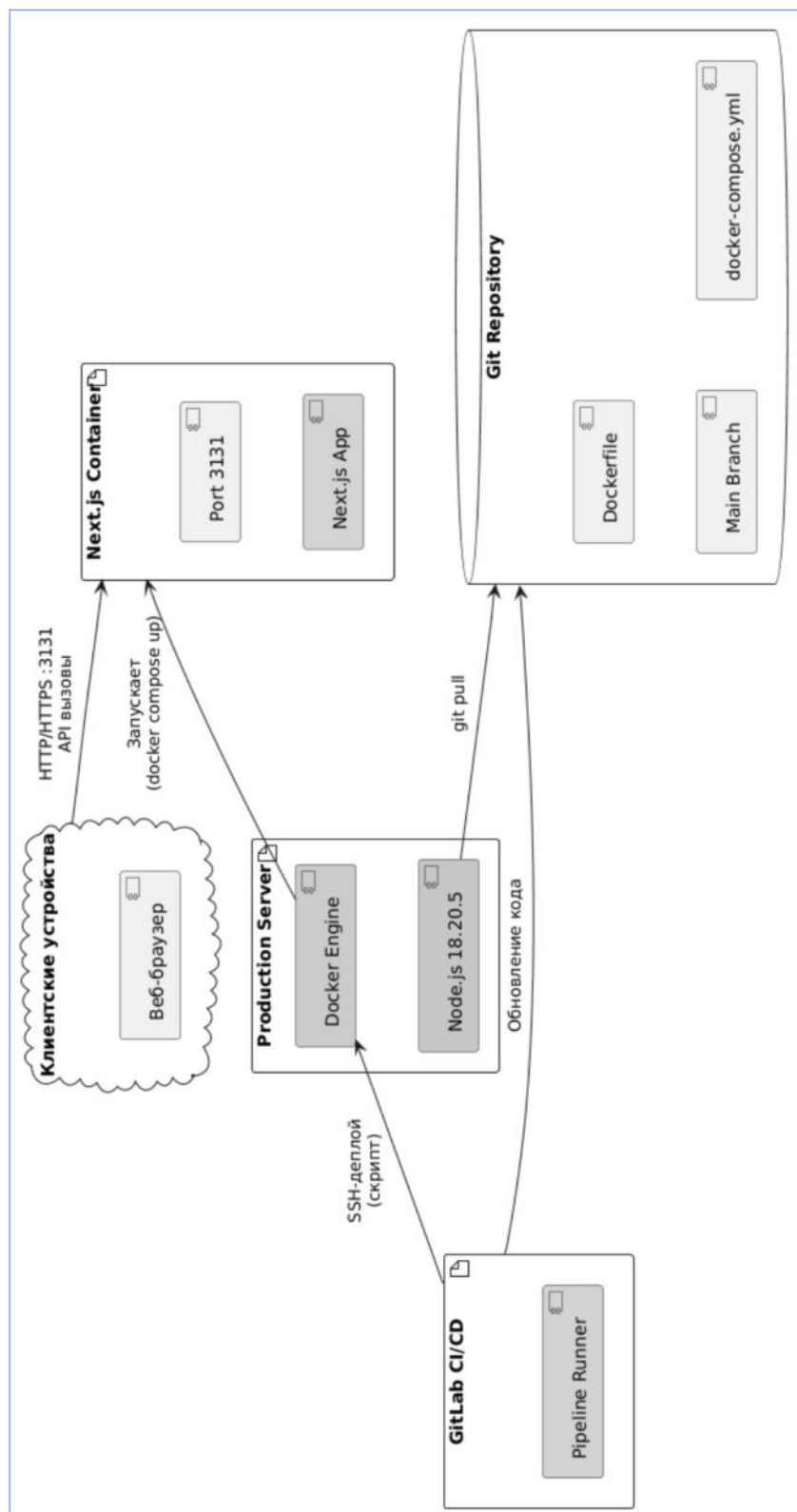
|                         |            |            |            |            |            |            |            |            |            |
|-------------------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| ПУУ 1.090301.22.002 Д14 |            |            |            |            |            |            |            |            |            |
| Имя                     | Фамилия    | Имя        | Фамилия    | Имя        | Фамилия    | Имя        | Фамилия    | Имя        | Фамилия    |
| Город                   | Город      | Город      | Город      | Город      | Город      | Город      | Город      | Город      | Город      |
| Должность               | Должность  | Должность  | Должность  | Должность  | Должность  | Должность  | Должность  | Должность  | Должность  |
| Подпись                 | Подпись    | Подпись    | Подпись    | Подпись    | Подпись    | Подпись    | Подпись    | Подпись    | Подпись    |
| И. о. лица              | И. о. лица | И. о. лица | И. о. лица | И. о. лица | И. о. лица | И. о. лица | И. о. лица | И. о. лица | И. о. лица |
| Уполн.                  | Уполн.     | Уполн.     | Уполн.     | Уполн.     | Уполн.     | Уполн.     | Уполн.     | Уполн.     | Уполн.     |
| ФОРМ. № 2168971         |            |            |            |            |            |            |            |            |            |

Приложение А.2-Диаграмма вариантов использования



|                        |                   |         |      |         |                   |  |  |  |  |
|------------------------|-------------------|---------|------|---------|-------------------|--|--|--|--|
| ПУ 1.090301.22.005 Д13 |                   |         |      |         |                   |  |  |  |  |
| Исполн.                | М. Ветух          | Получен | Дата | Масштаб |                   |  |  |  |  |
| Разработ.              | Григорьев В.Д.    | Принят  | Дата | Исполн. |                   |  |  |  |  |
| Провер.                | Борисовичева И.С. | Принят  | Дата | Исполн. |                   |  |  |  |  |
| Испытат.               | Алексеев А.С.     | Принят  | Дата | Исполн. |                   |  |  |  |  |
| Умк.                   | Морозова И.А.     | Принят  | Дата | Исполн. |                   |  |  |  |  |
|                        |                   |         |      |         | ФОРМ. пр. 218ВТ71 |  |  |  |  |

### Приложение А.3-Диаграмма развёртывания

[illegible]

|      |      |          |         |      |
|------|------|----------|---------|------|
|      |      |          |         |      |
|      |      |          |         |      |
| Изм. | Лист | № докум. | Подпись | Дата |

**Приложение А.4-Диаграмма компонентов**



## Приложение Б

(обязательное)

Веб-приложение CRM-система. Фронтенд часть

Листинг программы

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 76   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

## Файл «workspace.tsx»

```
import React, {useState, useEffect} from "react";

import KanbanBoard from "../../views/workspace/kanbanBoard";

import Box from "@mui/material/Box";

import Typography from "@mui/material/Typography";

import Popover from "@mui/material/Popover";

import MenuItem from "@mui/material/MenuItem";

import ListItemIcon from "@mui/material/ListItemIcon";

import {Icon} from "@iconify/react";

import Divider from "@mui/material/Divider";

import TextField from "@mui/material/TextField";

import InputAdornment from "@mui/material/InputAdornment";

import IconButton from "@mui/material/IconButton";

import {useTheme} from "@mui/material/styles";

import {useDispatch, useSelector} from "react-redux";

import {setSelectedPipeline} from "../../store/pipeline";

import {createPipeline, getPipelines, putPipelineById,
removePipelineById} from "../../api/requests/pipelines";

import {getIntegrations, createIntegration} from
"../../api/requests/integrations";

import toast from "react-hot-toast";

import Button from "@mui/material/Button";

import Skeleton from "@mui/material/Skeleton";

import {HexColorPicker} from "react-colorful";

import FallbackSpinner from "../../@core/components/spinner";

import Dialog from "@mui/material/Dialog";

import DialogTitle from "@mui/material/DialogTitle";

import DialogContent from "@mui/material/DialogContent";
```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 77   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

import DialogContentText from "@mui/material/DialogContentText";
import DialogActions from "@mui/material/DialogActions";
import Drawer from "@mui/material/Drawer";
import List from "@mui/material/List";
import ListItem from "@mui/material/ListItem";
import ListItemText from "@mui/material/ListItemText";
import ListItemButton from "@mui/material/ListItemButton";
import Collapse from "@mui/material/Collapse";
import {getStages} from "../../api/requests/stages";
import {Deal, Stage} from "../../types/Kanban";
import {getDeals} from "../../api/requests/deals";
import {Select} from "@mui/material";

```

```

interface Pipeline {
  id: string;
  name: string;
  color: string;
  crmId?: number;
  createdAt?: string;
  updatedAt?: string | null;
}

```

```

interface Integration {
  stageId: number | string;
  telegramIntegration?: {
    token: string;
  };
}

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 78   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

function Workspace() {

    const [anchorEl, setAnchorEl] = useState<HTMLInputElement | null>(null);

    const [isAddingPipeline, setIsAddingPipeline] =
    useState<boolean>(false);

    const [newPipelineName, setNewPipelineName] =
    useState<string>("");

    const [pipelines, setPipelines] = useState<Pipeline[]>([]);

    const [isLoading, setIsLoading] = useState<boolean>(true);

    const [editPipeline, setEditPipeline] = useState(false);

    const [colorPickerAnchorEl, setColorPickerAnchorEl] =
    useState<HTMLInputElement | null>(null);

    const [deleteDialogOpen, setDeleteDialogOpen] = useState(false);

    const [pipelineToDelete, setPipelineToDelete] = useState<string
    | null>(null)

    const [colorPickerOpen, setColorPickerOpen] = useState(false);

    const [tempColor, setTempColor] = useState("#FF5733")

    const [localSelectedPipeline, setLocalSelectedPipeline] =
    useState<Pipeline | null>(null);

    const [integrations, setIntegrations] =
    useState<Integration[]>([]);

    const [integrationsDrawerOpen, setIntegrationsDrawerOpen] =
    useState(false);

    const [newIntegration, setNewIntegration] =
    useState<Partial<Integration>>({

        stageId: 0,

        telegramIntegration: {

            token: ""

        }

    });

    const [columns, setColumns] = useState<Stage[]>([]);

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 79   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

const [openStages, setOpenStages] = useState<number[]>([]);

const [tasks, setTasks] = useState<Deal[]>([]);

const theme = useTheme();

// Redux state

const dispatch = useDispatch();

const {selectedPipeline} = useSelector((store: any) =>
store.pipeline);

const {selectedCRM} = useSelector((store: any) => store.crms);

const fetchPipelines = async () => {
  try {
    dispatch(setSelectedPipeline(null));
    setLocalSelectedPipeline(null);
    const res = await getPipelines(selectedCRM.id);
    setPipelines(res || []);
    const firstPipeline = res?.[0] || null;
    dispatch(setSelectedPipeline(firstPipeline));
    setLocalSelectedPipeline(firstPipeline);
  } catch (err) {
    toast.error('Не удалось загрузить воронки');
    console.error("Error fetching pipelines:", err);
  } finally {
  }
};

```

```

const fetchIntegrations = async () => {

  if (!selectedCRM?.id || !localSelectedPipeline?.id) return;

  try {

    const res = await getIntegrations(selectedCRM.id,
localSelectedPipeline.id);

    setIntegrations(res || []);

  } catch (err) {

    toast.error('Не удалось загрузить интеграции');

    console.error("Error fetching integrations:", err);

  }

};

useEffect(() => {

  if (localSelectedPipeline?.id && selectedCRM?.id) {

    fetchIntegrations();

  }

}, [localSelectedPipeline?.id, selectedCRM?.id]);

const updatePipelineColor = async (pipelineId: string, newColor:
string) => {

  try {

    await putPipelineById(selectedCRM.id, pipelineId, {

      color: newColor,

      name: selectedPipeline.name

    });

    const updatedPipelines = pipelines.map(p =>

      p.id === pipelineId ? {...p, color: newColor} : p

    );

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          |      |
| Изм. | Лист | № докум. | Подпись | Дата |                          | 81   |

```

    setPipelines(updatedPipelines);

    if (selectedPipeline?.id === pipelineId) {
        dispatch(setSelectedPipeline({...selectedPipeline, color:
newColor}));
    }

    toast.success('Цвет воронки обновлен');
} catch (error) {
    console.error("Error updating pipeline color:", error);
    toast.error('Не удалось обновить цвет воронки');
}
};

useEffect(() => {
    if (selectedCRM?.id) {
        fetchPipelines();
    }
}, [selectedCRM?.id]);

const handleColorSave = async () => {
    if (selectedPipeline) {
        await updatePipelineColor(selectedPipeline.id, tempColor);
        setColorPickerOpen(false);
    }
};

const handleColorPickerClose = () => {

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 82   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        setColorPickerAnchorEl(null);
    };

    useEffect(() => {
        if (pipelines.length > 0 && !selectedPipeline) {
            dispatch(setSelectedPipeline(pipelines[0]));
        }

        const timer = setTimeout(() => {
            setIsLoading(false)
        }, 500)

        return () => clearTimeout(timer)
    }, [pipelines, selectedPipeline, dispatch]);

    const handleClick = (event: React.MouseEvent<HTMLDivElement>) => {
        setAnchorEl(event.currentTarget);
    };

    const handleClose = () => {
        setAnchorEl(null);
        setIsAddingPipeline(false);
    };

    const handleSelectPipeline = (pipeline: Pipeline) => {
        dispatch(setSelectedPipeline(pipeline));
        setLocalSelectedPipeline(pipeline);
        setEditPipeline(false)
        handleClose();
    };

```

```

};

const handleCreateNewPipeline = () => {
    setIsAddingPipeline(true);
};

const handleConfirmNewPipeline = async () => {
    if (newPipelineName.trim()) {
        try {
            const color = "#" + Math.floor(Math.random() *
16777215).toString(16);

            const newPipeline = await createPipeline(selectedCRM.id, {
                name: newPipelineName,
                color: color
            });

            setPipelines(prev => [...prev, newPipeline]);
            dispatch(setSelectedPipeline(newPipeline));

            setNewPipelineName("");
            setIsAddingPipeline(false);
        } catch (error) {
            console.error("Error creating Pipeline:", error);
            toast.error('Не удалось создать воронку');
        }
    }
};

```

```

const open = Boolean(anchorEl);
const id = open ? "pipeline-popover" : undefined;

const handleDeletePipeline = async () => {
  if (!pipelineToDelete) return;

  try {
    await removePipelineById(selectedCRM.id, pipelineToDelete);
    toast.success('Воронка успешно удалена');

    const updatedPipelines = pipelines.filter(p => p.id !==
pipelineToDelete);
    setPipelines(updatedPipelines);

    if (updatedPipelines.length > 0) {
      dispatch(setSelectedPipeline(updatedPipelines[0]));
    } else {
      dispatch(setSelectedPipeline(null));
    }

    setDeleteDialogOpen(false);
    setPipelineToDelete(null);
    setEditPipeline(false);
  } catch (error) {
    console.error("Error deleting pipeline:", error);
    toast.error('Не удалось удалить воронку');
  }
};

```

```

const openDeleteDialog = (pipelineId: string) => {
    setPipelineToDelete(pipelineId);
    setDeleteDialogOpen(true);
};

const handleAddIntegration = async () => {
    if (!newIntegration.stageId ||
!newIntegration.telegramIntegration?.token) {
        toast.error('Заполните все поля');

        return;
    }

    try {
        const numericStageId = typeof newIntegration.stageId ===
'string'
            ? parseInt(newIntegration.stageId.replace(/\D/g, ''), 10)
            : newIntegration.stageId;

        await createIntegration(
            selectedCRM.id,
            localSelectedPipeline?.id || '',
            {
                ...newIntegration,
                stageId: numericStageId
            } as Integration
        );
    }

```

```

toast.success('Интеграция успешно добавлена');

fetchIntegrations();

setNewIntegration({
  stageId: 0,
  telegramIntegration: {
    token: ""
  }
});

} catch (error) {
  console.error("Error creating integration:", error);
  toast.error('Не удалось добавить интеграцию');
}

};

const toggleStage = (stageId: number) => {
  setOpenStages(prev =>
    prev.includes(stageId)
      ? prev.filter(id => id !== stageId)
      : [...prev, stageId]
  );
};

const fetchStages = async () => {
  if (selectedPipeline?.id && selectedPipeline?.crmId) {
    try {
      setIsLoading(true);
    }
  }
}

```

```

// Загружаем этапы

const stages = await getStages(
  selectedCRM?.id.toString(),
  selectedPipeline?.id.toString()
);

const loadedColumns = stages
  .sort((a: { order: number; }, b: { order: number; }) =>
a.order - b.order) // Сортируем по order
  .map((stage: Stage) => ({
    id: `col-${stage.id.toString()}`,
    name: stage.name,
    color: stage.color,
    order: stage.order
  }));

setColumns(loadedColumns);

if (stages) {
  const deals = await getDeals(
    selectedPipeline.crmId.toString(),
    selectedPipeline.id.toString()
  );

  setTasks(deals.map((deal: { id: { toString: () => any; }; stageId: { toString: () => any; }; }) => ({
    ...deal,
    id: deal.id.toString(),
    stageId: `col-${deal.stageId.toString()}`,
  })));
}

```

```

    }

    } catch (error) {
        toast.error('Ошибка загрузки данных');
        console.error("Error loading data:", error);
    } finally {
        setIsLoading(false);
    }
} else {
    setColumns([]);
    setTasks([]);
}
};

useEffect(() => {
    fetchStages();
}, [selectedPipeline])

const getStageNameById = (stageId: number) => {
    const stage = columns.find(s => parseInt(s.id) === stageId);

    return stage ? stage.name : `Этап ${stageId}`;
};

return (
    <Box
        sx={{

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          |      |
| Изм. | Лист | № докум. | Подпись | Дата |                          | 89   |

```

        display: "flex",
        maxHeight: "90vh",
        minHeight: "90vh",
        alignItems: "start",
        borderRadius: '16px',
        borderTopLeftRadius: '0',
        borderTopRightRadius: '0',
        overflowY: "hidden",
        flexDirection: "column",
        gap: 5,
        backgroundColor: theme.palette.background.paper
    }}
>
<Dialog
    open={deleteDialogOpen}
    onClose={() => setDeleteDialogOpen(false)}
    aria-labelledby="alert-dialog-title"
    aria-describedby="alert-dialog-description"
>
    <DialogTitle id="alert-dialog-title">
        Подтверждение удаления
    </DialogTitle>
    <DialogContent>
        <DialogContentText id="alert-dialog-description">
            Вы уверены, что хотите удалить воронку
            "{pipelines.find(p => p.id === pipelineToDelete)?.name}"?
            Это действие нельзя отменить.
        </DialogContentText>

```

```

</DialogContent>

<DialogActions>

  <Button

    onClick={handleDeletePipeline}

    color="error"

    variant="text"

    autoFocus

  >

    Удалить

  </Button>

  <Button variant='outlined' onClick={() =>
setDeleteDialogOpen(false)}>Отмена</Button>

</DialogActions>

</Dialog>

<Box sx={{

  borderBottom: `1px solid ${theme.palette.divider}`,

  borderTop: `1px solid ${theme.palette.divider}`,

  backgroundColor: theme.palette.background.paper,

  width: "100%",

  display: "flex",

  justifyContent: "space-between",

  pr: 8,

  alignItems: "center"

}}>

  <Box sx={{display: "flex", justifyContent: "space-
between"}}>

    <Box

      sx={{

```

```

padding: 2,
cursor: "pointer",
display: "flex",
gap: 1,
alignItems: "center",
justifyContent: "space-between",
minWidth: '300px',
maxWidth: '300px',
height: '62px',
borderRight: `1px solid ${theme.palette.divider}`,
}}
onClick={handleClick}
>
<Box sx={{display: "flex", alignItems: "center",
justifyContent: "space-between", gap: 1,}}>
  {isLoading ? (
    <Skeleton variant="circular" width={18}
height={18}/>
  ) : localSelectedPipeline ? (
    <Box
      sx={{
        width: 18,
        height: 18,
        borderRadius: '3px',
        backgroundColor: localSelectedPipeline.color,
        cursor: editPipeline ? 'pointer' : 'default',
      }}
    />
  ) : null}

```

```

{isLoading ? (
  <Skeleton width={150} height={24}/>
) : (
  <Typography
    variant="body1"
    sx={{
      maxWidth: '300px',
      whiteSpace: 'nowrap',
      textOverflow: 'ellipsis',
      overflow: 'hidden',
      display: 'flex',
      alignItems: 'center',
      gap: 1
    }}
  >
    {localSelectedPipeline ? (
      <>
        Воронка:
        <Typography
          component="span"
          variant="body2"
          sx={{
            whiteSpace: 'nowrap',
            textOverflow: 'ellipsis',
            overflow: 'hidden',
            maxWidth: '150px',
          }}

```

```

        >
        {localSelectedPipeline.name}
    </Typography>
</>
) : (
    "Выберите воронку"
)}
</Typography>
)}
</Box>

<Icon icon="mdi:chevron-down" fontSize="1.2rem"/>

</Box>

{ (localSelectedPipeline && editPipeline) && (
    <Box
        sx={{
            padding: 2,
            display: "flex",
            alignItems: "center",
            gap: 1,
            height: '62px',
            cursor: editPipeline ? 'pointer' : 'default',
        }}
        onClick={() => {
            setTempColor(localSelectedPipeline.color);
            setColorPickerOpen(true);
        }}
    </Box>
)}

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 94   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

    >

    <Typography variant="body2" sx={{mr: 1}}>

        Цвет:

    </Typography>

    <Box

        sx={{

            width: 24,

            height: 24,

            borderRadius: '4px',

            backgroundColor: localSelectedPipeline.color,

            border: `1px solid ${theme.palette.divider}`,

        }}

    />

</Box>

)}

</Box>

<Popover

    open={colorPickerOpen}

    anchorEl={colorPickerAnchorEl}

    onClose={handleColorPickerClose}

    anchorOrigin={{

        vertical: 'bottom',

        horizontal: 'left',

    }}

>

    <Box sx={{p: 2}}>

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 95   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        <HexColorPicker
            color={tempColor}
            onChange={setTempColor}
            style={{width: '200px', height: '200px'}}
        />

        <Box sx={{display: 'flex', justifyContent: 'flex-end',
mt: 2}}>

            <Button onClick={handleColorPickerClose}>

                Отмена

            </Button>

            <Button

                variant="contained"

                onClick={handleColorSave}

                sx={{ml: 1}}

            >

                Сохранить

            </Button>

        </Box>

    </Box>

</Popover>

<Popover
    id={id}
    open={open}
    anchorEl={anchorEl}
    onClose={handleClose}
    anchorOrigin={{
        vertical: "bottom",

```

```

        horizontal: "left",
      }}
      transformOrigin={{
        vertical: "top",
        horizontal: "left",
      }}
      sx={{
        top: '1px'
      }}
      PaperProps={{
        sx: {
          borderRadius: '5px',
          minWidth: '300px',
        }
      }}
    >

    <Box sx={{p: 2, minWidth: 250}}>

      <Typography variant="body1" sx={{mb: 2}}>

        Воронки

      </Typography>

      {isLoading ? (

        <Box sx={{display: 'flex', flexDirection: 'column',
gap: 1}}>

          {[...Array(3)].map((_, index) => (

            <Box key={index} sx={{display: 'flex',
alignItems: 'center', gap: 1, py: 1}}>

              <Skeleton variant="rectangular" width={18}
height={18} sx={{borderRadius: '3px'}}/>

```

```

        <Skeleton width="70%" height={20}/>

      </Box>

    )})

  </Box>

) : (

  <>

    <Box sx={{maxHeight: "300px", overflowX: "auto"}}>

      {pipelines.map((pipeline: Pipeline) => (

        <MenuItem

          key={pipeline.id}

          onClick={() =>
handleSelectPipeline(pipeline)}

          sx={{

            display: 'flex',

            alignItems: 'center',

            gap: 1,

            backgroundColor: pipeline.id ===
localSelectedPipeline?.id ? theme.palette.action.selected :
'transparent',

            '&:hover': {

              backgroundColor:
theme.palette.action.hover,

            },

          }}

        >

          <Box

            sx={{

              width: 18,

              height: 18,

              borderRadius: '3px',

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 98   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        backgroundColor: pipeline.color,
      }}
    />
    <Typography variant="body2">
      {pipeline.name}
    </Typography>
  </MenuItem>
  )})
</Box>

  <Divider sx={{border: theme => `0.5px solid
${theme.palette.divider}`, my: 1}}/>

  {isAddingPipeline ? (
    <Box sx={{display: "flex", alignItems: "center",
gap: 1, mt: 1}}>
      <TextField
        size="small"
        placeholder="Название воронки"
        value={newPipelineName}
        onChange={ (e) =>
setNewPipelineName (e.target.value) }
        InputProps={{
          endAdornment: (
            <InputAdornment position="end">
              <IconButton
onClick={handleConfirmNewPipeline}>
                <Icon icon="mdi:check"
fontSize="1.2rem"/>
              </IconButton>
            </InputAdornment>

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 99   |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        ),
      },
    ]),
  ],
  gap: 1}}
</Box>
) : (
  <MenuItem
    onClick={handleCreateNewPipeline}
    sx={{display: "flex", alignItems: "center",
gap: 1}}
  >
    <Typography variant="body2">
      Создать воронку
    </Typography>
    <ListItemIcon>
      <Icon icon="mdi:plus" fontWeight='bold' />
    </ListItemIcon>
  </MenuItem>
) }
</>
) }
</Box>
</Popover>
<Box sx={{display: 'flex', gap: 1}}>
  {editPipeline ? (
    <>
      <Button
        variant="text"
        color="error"

```

```

        size="small"

        sx={{
            borderRadius: '5px',
            textTransform: 'none',
            marginY: 3,
        }}

        onClick={() => localSelectedPipeline &&
openDeleteDialog(localSelectedPipeline.id)}

        startIcon={<Icon icon="mdi:delete-outline"
fontSize="1.2rem"/>}

    >

        Удалить воронку

    </Button>

    <Button

        variant="text"

        sx={{
            color: theme.palette.text.primary,
            borderRadius: '5px',
            textTransform: 'none',
            marginY: 3,
            '&:hover': {
                backgroundColor: theme.palette.action.hover
            }
        }}

        onClick={() => setEditPipeline(false)}

        endIcon={

            <svg width="20" height="20" viewBox="0 0 24 24"
fill="none" xmlns="http://www.w3.org/2000/svg">

                <path d="M4 12.0002H20M20 12.0002L14
6.00024M20 12.0002L14 18.0002" stroke="#3c4043de"

```

```

stroke-width="1.5"
stroke-linecap="round" stroke-
linejoin="round"/>
    </svg>
  }
>
  Назад
</Button>
</>
) : (
<>
<Button
  variant="outlined"
  sx={{
    color: theme.palette.text.primary,
    borderRadius: '5px',
    textTransform: 'none',
    marginY: 3,
    '&:hover': {
      backgroundColor: theme.palette.action.hover
    }
  }}
  onClick={() => setIntegrationsDrawerOpen(true)}
  endIcon={<Icon icon="mdi:connection"
fontSize="1.2rem"/>}
>
  Интеграции
</Button>
<Button

```

```

variant="outlined"

sx={{
  color: theme.palette.text.primary,
  borderColor: theme.palette.text.primary,
  borderRadius: '5px',
  textTransform: 'none',
  marginY: 3,
  '&:hover': {
    backgroundColor: theme.palette.action.hover
  }
}}

onClick={() => setEditPipeline(true)}

endIcon={
  <svg width="20" height="20" viewBox="0 0 24 24"
fill="none" xmlns="http://www.w3.org/2000/svg">

    <path
      d="M3 8.00024L15 8.00025M15 8.00025C15
9.6571 16.3431 11.0002 18 11.0002C19.6569 11.0002 21 9.6571 21
8.00024C21 6.34339 19.6569 5.00024 18 5.00024C16.3431 5.00024 15
6.34339 15 8.00025ZM9 16.0002L21 16.0002M9 16.0002C9 17.6571 7.65685
19.0002 6 19.0002C4.34315 19.0002 3 17.6571 3 16.0002C3 14.3434
4.34315 13.0002 6 13.0002C7.65685 13.0002 9 14.3434 9 16.0002Z"

      stroke="#3c4043de" stroke-width="2" stroke-
linecap="round" stroke-linejoin="round"/>

    </svg>
  >

  Настроить
</Button>

</>

)}

</Box>

```

</Box>

```
<Box sx={{overflowX: "auto", width: "100%", scrollBehavior:
'smooth', overscrollBehavior: 'smooth'}}>
```

```
{isLoading ? (
```

```
<Box sx={{p: 3}}>
```

```
<FallbackSpinner/>
```

```
</Box>
```

```
) : (
```

```
<KanbanBoard selectedPipeline={localSelectedPipeline}
selectedCRM={selectedCRM} editPipeline={editPipeline}
```

```
columns={columns || []}
setColumns={setColumns} tasks={tasks || []} setTasks={setTasks}
```

```
isLoading={isLoading}/>
```

```
)}
```

</Box>

```
{/* Integrations Drawer */}
```

```
<Drawer
```

```
anchor="right"
```

```
open={integrationsDrawerOpen}
```

```
onClose={() => setIntegrationsDrawerOpen(false)}
```

```
elevation={0}
```

```
sx={{
```

```
'& .MuiDrawer-paper': {
```

```
width: 400,
```

```
height: 'calc(100vh - 1.05rem)',
```

```
margin: '7.8px 0',
```

```
boxSizing: 'border-box',
```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 104  |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        p: 3,
      },
    }}
  >
  <Box sx={{display: 'flex', justifyContent: 'space-
between', alignItems: 'center', mb: 3}}>

    <Typography variant="h6">Управление
интеграциями</Typography>

    <IconButton onClick={() =>
setIntegrationsDrawerOpen(false)}>

      <Icon icon="mdi:close" fontSize="1.5rem"/>

    </IconButton>

  </Box>

  <Typography variant="subtitle1" sx={{mb: 2}}>
    Текущие интеграции
  </Typography>

  {integrations.length === 0 ? (
    <Typography variant="body2" sx={{color:
theme.palette.text.secondary, mb: 3}}>
      Нет активных интеграций
    </Typography>
  ) : (
    <List sx={{mb: 3}}>
      {integrations.map((integration, index) => (
        <React.Fragment key={index}>
          <ListItemButton onClick={() =>
toggleStage(parseInt(integration.stageId as string))}>

```

```

        <ListItemText
primary={getStageNameById(parseInt(integration.stageId as
string)) }/>

{openStages.includes(parseInt(integration.stageId as string)) ?
<Icon icon='mdi:chevron-down' /> :

        <Icon icon='mdi:chevron-up' /> }

    </ListItemButton>

    <Collapse
in={openStages.includes(parseInt(integration.stageId as string))}
timeout="auto"

        unmountOnExit>

        <List component="div" disablePadding>

            {integration.telegramIntegration && (

                <ListItem sx={{pl: 4}}>

                    <ListItemText

                        primary="Telegram"

                        secondary={`Token:
${integration.telegramIntegration.token.substring(0, 10)}...`}

                    />

                </ListItem>

            ) }

            { /* Add other integration types here */ }

        </List>

    </Collapse>

</React.Fragment>

)}}

</List>

)}

<Divider sx={{my: 2}} />

```

```

<Typography variant="subtitle1" sx={{mb: 2}}>
    Добавить новую интеграцию
</Typography>

<Select
    labelId="stage-select-label"
    id="stage-select"
    size="small"
    sx={{mb: 3}}
    value={newIntegration.stageId || ''}
    onChange={(e) => setNewIntegration({
        ...newIntegration,
        stageId: e.target.value
    })}
>

    {columns.map((stage) => (
        <MenuItem key={stage.id} value={stage.id}>
            <Box sx={{display: 'flex', alignItems: 'center',
gap: 1}}>

                <Box
                    sx={{
                        width: 12,
                        height: 12,
                        borderRadius: '2px',
                        backgroundColor: stage.color ||
theme.palette.primary.main,
                    }}
                />

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 107  |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        {stage.name}

    </Box>

</MenuItem>

    ) ) }

</Select>

<Typography variant="subtitle2" sx={{mb: 1}}>
    Telegram
</Typography>

<Box sx={{mb: 3}}>

    <TextField

        size='small'

        fullWidth

        label="Токен бота"

        value={newIntegration.telegramIntegration?.token ||

''}

        onChange={ (e) => setNewIntegration({

            ...newIntegration,

            telegramIntegration: {

                token: e.target.value

            }

        }) }

    />

</Box>

<Button

    fullWidth

    variant="contained"

```

```

        onClick={handleAddIntegration}

        disabled={!newIntegration.stageId ||
!newIntegration.telegramIntegration?.token}

        >

        Добавить интеграцию

    </Button>

</Drawer>

</Box>

);
}

```

```
export default Workspace;
```

файл «detail/[id]/index.tsx»

```

'use client'

import {useEffect, useState, useRef} from 'react'

import {

    Box,

    Typography,

    Avatar,

    Button,

    TextField,

    MenuItem,

    Select,

    SelectChangeEvent,

    IconButton,

    Tooltip,

    Tabs,

    Tab

```

```

    } from '@mui/material'

    import {Icon} from '@iconify/react'

    import {useTheme} from '@mui/material/styles'

    import {Deal} from '../../../types/Kanban'

    import {getDealById} from '../../../api/requests/deals'

    import {useSearchParams} from 'next/navigation'

    import {useRouter} from 'next/router'

    import {getStages} from '../../../api/requests/stages'

    import InputAdornment from '@mui/material/InputAdornment'

    import FallbackSpinner from
    "../../../core/components/spinner";

```

```

type Message = {
    id: number
    text: string
    date: string
    isMe: boolean
    replyToId?: number
    reaction?: Reaction
    sender?: string
}

```

```

type Reaction = '👍' | '❤️' | '😂' | '😮' | '👎' | '🔥'

```

```

type Stage = {
    id: string
    name: string
    color: string
}

```

```

    order: number
  }

export default function DealCard() {
  const theme = useTheme()
  const searchParams = useSearchParams()
  const router = useRouter()

  const {id} = router.query
  const crmId = searchParams.get('crmId') as string
  const pipelineId = searchParams.get('pipelineId') as string

  const [deal, setDeal] = useState<Deal | null>(null)
  const [loading, setLoading] = useState(true)
  const [stages, setStages] = useState<Stage[]>([])
  const [message, setMessage] = useState('')
  const [messages, setMessages] = useState<Message[]>([])
  const [editedValues, setEditedValues] = useState({
    name: '',
    amount: '',
    clientId: ''
  })

  const [selectedStage, setSelectedStage] = useState<string>('')
  const [replyTo, setReplyTo] = useState<Message | null>(null)

  const [connectionStatus, setConnectionStatus] =
    useState('disconnected')

  const wsRef = useRef<WebSocket | null>(null)

```

```

const reconnectAttemptRef = useRef(0)

const isMountedRef = useRef(true)

const [activeTab, setActiveTab] = useState(0)

// Initialize WebSocket connection
useEffect(() => {

  isMountedRef.current = true

  const connectWebSocket = () => {

    if (!id || !crmId || !pipelineId) return

    const token = localStorage.getItem('accessToken') || ''

    if (!token) {

      console.error('No token found for WebSocket connection')

      return

    }

    if (wsRef.current?.readyState === WebSocket.OPEN) {

      return

    }

    const socket = new
WebSocket(`wss://chat.crmapi.magiclab.space/chats/ws?token=${token}&
dealId=${id}`)

    wsRef.current = socket

    socket.onopen = () => {

      if (!isMountedRef.current) return

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 112  |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

console.log('WebSocket connected')

setConnectionStatus('connected')

reconnectAttemptRef.current = 0


socket.send(
  JSON.stringify({
    type: 'init',
    dealId: id,
    crmId,
    pipelineId
  })
)
}

socket.onmessage = event => {
  if (!isMountedRef.current) return
  try {
    const data = JSON.parse(event.data)
    console.log('Received message:', data.new_message)

    setMessages(prev => [
      ...prev,
      {
        id: Date.now(),
        text: data.new_message,
        date: new Date().toISOString(),
        isMe: false
      }
    ]
  )
}

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 113  |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        ])
    } catch (error) {
        console.error('Error parsing WebSocket message:', error)
    }
}

socket.onerror = error => {
    if (!isMountedRef.current) return
    console.error('WebSocket error:', error)
    setConnectionStatus('error')
}

socket.onclose = event => {
    if (!isMountedRef.current) return
    console.log('WebSocket disconnected', event.code,
event.reason)
    setConnectionStatus('disconnected')
}
}

connectWebSocket()

// Обработчик для отключения при уходе пользователя со
страницы

const handleBeforeUnload = () => {
    if (wsRef.current) {
        wsRef.current.close()
    }
}

```

```

const handleVisibilityChange = () => {

  if (document.visibilityState === 'hidden') {

    // Страница скрыта - отключаем сокет для экономии ресурсов

    if (wsRef.current) {

      wsRef.current.close()

      setConnectionStatus('disconnected')

    }

  } else if (document.visibilityState === 'visible') {

    connectWebSocket()

  }

}

// Добавляем слушатели событий

window.addEventListener('beforeunload', handleBeforeUnload)

document.addEventListener('visibilitychange',
handleVisibilityChange)

return () => {

  isMountedRef.current = false

  if (wsRef.current) {

    wsRef.current.close()

    wsRef.current = null

  }

  window.removeEventListener('beforeunload',
handleBeforeUnload)

  document.removeEventListener('visibilitychange',
handleVisibilityChange)

}

```

```

    }, [id, crmId, pipelineId])

    // Load deal data
    useEffect(() => {
        const fetchData = async () => {
            if (crmId && pipelineId && id) {
                try {
                    const loadedStages = await getStages(crmId, pipelineId)
                    setStages(loadedStages)

                    const res = await getDealById(crmId, pipelineId, id)
                    setDeal(res)
                    setSelectedStage(res.stage || loadedStages[0]?.name ||
'')

                    setEditedValues({
                        name: res.name,
                        amount: res.amount?.toString() || '',
                        clientId: res.clientId?.toString() || ''
                    })
                } catch (e) {
                    console.error('Ошибка при загрузке данных:', e)
                } finally {
                    setLoading(false)
                }
            }
        }

        fetchData()
    })

```

```

    }, [crmId, pipelineId, id])

    const toggleWebSocket = () => {

        if (wsRef.current && wsRef.current.readyState ===
        WebSocket.OPEN) {

            wsRef.current.close()

            setConnectionStatus('disconnected')

        } else {

            // Подключаем веб-сокеты

            const token = localStorage.getItem('accessToken') || ''

            if (!token || !id || !crmId || !pipelineId) return

            const socket = new
            WebSocket(`wss://chat.crmapi.magiclab.space/chats/ws?token=${token}&
            dealId=${id}`)

            wsRef.current = socket

            socket.onopen = () => {

                console.log('WebSocket connected')

                setConnectionStatus('connected')

                socket.send(

                    JSON.stringify({

                        type: 'init',

                        dealId: id,

                        crmId,

                        pipelineId

                    })

                )

            }

        }

    }

```

```

    }

    socket.onmessage = event => {
        try {
            const data = JSON.parse(event.data)
            console.log('Received message:', data.new_message)

            setMessages(prev => [
                ...prev,
                {
                    id: Date.now(),
                    text: data.new_message,
                    date: new Date().toISOString(),
                    isMe: false
                }
            ])
        } catch (error) {
            console.error('Error parsing WebSocket message:', error)
        }
    }

    socket.onerror = error => {
        console.error('WebSocket error:', error)
        setConnectionStatus('error')
    }

    socket.onclose = () => {
        console.log('WebSocket disconnected')
    }

```

```

        setConnectionStatus('disconnected')
    }
}

const handleSendMessage = () => {
    if (message.trim() && wsRef.current?.readyState ===
WebSocket.OPEN) {
        const messageData = {
            message: message,
            dealId: id
        }

        wsRef.current.send(JSON.stringify(messageData))

        setMessages(prev => [
            ...prev,
            {
                id: Date.now(),
                text: message,
                date: new Date().toISOString(),
                isMe: true,
                replyToId: replyTo?.id
            }
        ])

        setMessage('')

        setReplyTo(null)
    }
}

```

```

    }

    const handleReaction = (messageId: number, reaction: Reaction)
=> {
    if (wsRef.current?.readyState === WebSocket.OPEN) {
        wsRef.current.send(
            JSON.stringify({
                type: 'reaction',
                messageId,
                reaction
            })
        )
    }

    setMessages(prev => prev.map(msg => (msg.id === messageId ?
{...msg, reaction} : msg)))
}

const handleStageChange = async (e: SelectChangeEvent) => {
    const newStage = e.target.value
    setSelectedStage(newStage)

    if (deal && crmId && pipelineId && wsRef.current?.readyState
=== WebSocket.OPEN) {
        wsRef.current.send(
            JSON.stringify({
                type: 'stage_change',
                dealId: id,
                stage: newStage
            })
        )
    }
}

```

```

        })

    )

}

}

const handleFieldChange = async (field: keyof typeof
editedValues, value: string) => {

    if (!deal || !crmId || !pipelineId) return

    const updatedValues = {...editedValues, [field]: value}
    setEditedValues(updatedValues)

    const updatedDeal = {

        ...deal,

        [field]: field === 'amount' ? parseFloat(value) || 0 : field
=== 'clientId' ? parseInt(value) || 0 : value

    }

    setDeal(updatedDeal)

    if (wsRef.current?.readyState === WebSocket.OPEN) {

        wsRef.current.send(

            JSON.stringify({

                type: 'deal_update',

                field,

                value,

                dealId: id

            })

        )

    }

}

```

```

    }

}

if (loading) {
    return <FallbackSpinner/>
}

if (!deal) {
    return <Typography sx={{p: 3}}>Сделка не найдена</Typography>
}

return (
    <Box
        sx={{
            display: 'flex',
            height: '100%',
            backgroundColor: theme.palette.background.paper
        }}
    >
        <Box
            sx={{
                width: '25%',
                borderRight: `1px solid ${theme.palette.divider}`,
                display: 'flex',
                flexDirection: 'column',
                backgroundColor: theme.palette.background.paper
            }}
        >

```

```

        {/* Заголовок сделки */}

        <Box sx={{p: 0, borderBottom: `1px solid
${theme.palette.divider}`}}>

            <TextField

                value={editedValues.name}

                onChange={e => setEditedValues({...editedValues, name:
e.target.value})}

                onBlur={e => handleFieldChange('name',
e.target.value)}

                fullWidth

                variant='outlined'

                size='small'

                InputProps={{

                    sx: {

                        fontSize: '1.1rem',

                        fontWeight: 500,

                        '&:before': {borderBottom: 'none'},

                        '&:hover:not(.Mui-disabled):before':
{borderBottom: 'none'}

                    }

                }}

                sx={{

                    '& .MuiOutlinedInput-root': {

                        backgroundColor: 'transparent',

                        '& fieldset': {

                            borderColor: 'transparent !important'

                        }

                    }

                }}

            />

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 123  |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

<Box
  sx={{
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'space-between',
    mt: 1,
    padding: '0 20px'
  }}
>
<Typography variant='caption' color='text.secondary'>
  ID: {deal.id}
</Typography>
<Box sx={{display: 'flex', alignItems: 'center'}}>
  <Box
    sx={{
      width: 8,
      height: 8,
      borderRadius: '50%',
      bgcolor: connectionStatus === 'connected' ?
'success.main' : 'error.main',
      mr: 1
    }}
  />
  <Button
    size='small'
    onClick={toggleWebSocket}
    sx={{
      textTransform: 'none',

```

```

        fontSize: '0.75rem',

        p: 0,

        minWidth: 0,

        color: 'text.secondary'

      }}

    >

      {connectionStatus === 'connected' ? 'Отключить' :
'Подключить'}

    </Button>

  </Box>

</Box>

</Box>

<Tabs

  value={activeTab}

  onChange={(_, newValue) => setActiveTab(newValue)}

  variant='fullWidth'

  TabIndicatorProps={{

    style: {

      height: 2

    }

  }}

  sx={{

    borderBottom: `1px solid ${theme.palette.divider}`,

    minHeight: 12,

    padding: '0 20px'

  }}

>

```

```

<Tab
  sx={{
    minHeight: 32,
    padding: '4px 8px',
    fontSize: 13
  }}
  disableRipple
  disableTouchRipple
  label='Основное'
/>

<Tab
  sx={{
    minHeight: 32,
    padding: '4px 8px',
    fontSize: 13
  }}
  disableRipple
  disableTouchRipple
  label='Дополнительное'
/>

</Tabs>

{activeTab === 0 && (
  <Box sx={{p: 2, flex: 1, padding: '10px 20px 0'}}>
    <Box sx={{mb: 3, display: 'flex', alignItems:
'center', gap: 2}}>
      <Typography variant='caption' color='text.secondary'
sx={{mb: 0.5, display: 'block', width: '30%'}}>
        Сумма

```

```

</Typography>

<TextField

  value={editedValues.amount}

  onChange={e => setEditedValues({...editedValues,
amount: e.target.value})}

  onBlur={e => handleFieldChange('amount',
e.target.value)}

  fullWidth

  size='small'

  variant='outlined'

  InputProps={{

    endAdornment: (

      <InputAdornment position='end'>

        <Typography variant='body2'>P</Typography>

      </InputAdornment>

    ),

    sx: {

      fontSize: '0.875rem',

      fontWeight: 500

    }

  }}

  sx={{

    width: '70%',

    '& .MuiOutlinedInput-root': {

      backgroundColor: 'transparent',

      '& fieldset': {

        borderColor: 'transparent !important'

      },

      '&:hover fieldset': {

```

```

        borderColor: 'transparent'
      },
      '&:focus fieldset': {
        borderColor: 'transparent'
      },
      '&:focus': {
        borderColor: 'transparent !important'
      }
    }
  }}
/>
</Box>

```

```

<Box sx={{mb: 3, display: 'flex', alignItems:
'center', gap: 2}}>

```

```

  <Typography variant='caption' color='text.secondary'
sx={{mb: 0.5, display: 'block', width: '30%'}}>

```

Этап

```

</Typography>

```

```

<Select

```

```

  fullWidth

```

```

  size='small'

```

```

  value={selectedStage}

```

```

  onChange={handleStageChange}

```

```

  variant='outlined'

```

```

  sx={{

```

```

    width: '70%',

```

```

    fontSize: '0.875rem',

```

```

    '& .MuiOutlinedInput-root': {

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 128  |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        backgroundColor: theme.palette.grey[100],
        '& fieldset': {
            borderColor: 'transparent !important'
        }
    }
}}
>
    {stages.map(stage => (
        <MenuItem key={stage.id} value={stage.name}
sx={{fontSize: '0.875rem'}}>
        <Box
            sx={{
                display: 'flex',
                alignItems: 'center',
                gap: 1
            }}
        >
            <Box
                sx={{
                    width: 12,
                    height: 12,
                    borderRadius: '2px',
                    backgroundColor: stage.color
                }}
            />
            {stage.name}
        </Box>
    </MenuItem>

```

```

    ))}

</Select>

</Box>

<Box sx={{mb: 3, display: 'flex', alignItems:
'center', gap: 2}}>

    <Typography variant='caption' color='text.secondary'
sx={{mb: 0.5, display: 'block', width: '30%'}}>

        Клиент

    </Typography>

    <Box sx={{display: 'flex', alignItems: 'center',
gap: 1, width: '70%'}}>

        <Avatar

            sx={{

                width: 28,

                height: 28,

                fontSize: '0.75rem',

                bgcolor: theme.palette.grey[300]

            }}

        >

            {deal.clientId ? `C${deal.clientId}` : '?'}

        </Avatar>

        <TextField

            value={editedValues.clientId}

            onChange={e => setEditedValues({...editedValues,
clientId: e.target.value})}

            onBlur={e => handleFieldChange('clientId',
e.target.value)}

            fullWidth

            size='small'

```

```

        variant='outlined'

        type='number'

        placeholder='ID клиента'

        sx={{
            fontSize: '0.875rem',

            '& .MuiOutlinedInput-root': {
                backgroundColor: 'transparent',

                '& fieldset': {
                    borderColor: 'transparent !important'
                }
            }
        }}
    />

</Box>

</Box>

<Box>

    <Typography variant='caption' color='text.secondary'
sx={{mb: 0.5, display: 'block'}}>

        Создана

    </Typography>

    <Typography variant='body2' sx={{fontSize:
'0.875rem'}}></Typography>

</Box>

</Box>

)}

</Box>

{/* Chat */}

```

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 131  |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

```

        <Box sx={{flex: 1, display: 'flex', flexDirection:
'column'}}>

            <Box sx={{p: 2, borderBottom: `1px solid
${theme.palette.divider}`}}>

                <Typography variant='h6'>Переписка</Typography>

                <Typography variant='caption' color={connectionStatus
=== 'connected' ? 'success.main' : 'error.main'}>

                    {connectionStatus === 'connected' ? 'Online' :
'Offline'}

                </Typography>

            </Box>

            <Box sx={{flex: 1, overflowY: 'auto', p: 2, maxHeight:
'74vh'}}>

                {messages.map(msg => {

                    const replyMessage = msg.replyToId ? messages.find(m
=> m.id === msg.replyToId) : null

                    return (

                        <Box

                            key={msg.id}

                            sx={{

                                display: 'flex',

                                flexDirection: msg.isMe ? 'row-reverse' : 'row',

                                mb: 2

                            }}

                        >

                            <Avatar

                                sx={{

                                    width: 32,

                                    height: 32,

```

```

        mr: msg.isMe ? 0 : 1,
        ml: msg.isMe ? 1 : 0
    }}
    >
    {msg.isMe ? 'Я' : msg.sender ?
msg.sender.charAt(0) : 'К'}
</Avatar>
<Box sx={{position: 'relative'}}>
    {replyMessage && (
        <Box
            sx={{
                p: 1,
                backgroundColor: theme.palette.grey[100],
                borderLeft: '3px solid',
                borderColor: theme.palette.primary.light,
                fontSize: '0.8rem',
                mb: 0.5
            }}
        >
            {replyMessage.text}
        </Box>
    )}
    <Box
        sx={{
            p: '12px 16px',
            backgroundColor: msg.isMe ?
theme.palette.primary.light : theme.palette.grey[200],
            borderRadius: '8px',
            wordBreak: 'break-word',

```

```

        overflowWrap: 'break-word',
        boxSizing: 'border-box',
        position: 'relative',
    }}
>
    {msg.sender && !msg.isMe && (
        <Typography variant='caption' sx={{display:
'block', fontWeight: 'bold', mb: 0.5}}>
            {msg.sender}
        </Typography>
    )}
<Typography
    variant='body2'
    sx={{
        whiteSpace: 'pre-wrap',
        lineHeight: 1.4,
        color: msg.isMe ? 'white' : 'black',
        paddingRight: msg.isMe ? '40px' : 0,
        paddingLeft: msg.isMe ? 0 : '40px',
    }}
>
    {msg.text}
</Typography>

{/* Время - позиционируется в зависимости от
того, кто написал */}

<Typography
    variant='caption'
    sx={{

```

```

        color: msg.isMe ? 'rgba(255, 255, 255,
0.6)' : 'rgba(0, 0, 0, 0.4)',

        fontSize: '11px',

        position: 'absolute',

        right: msg.isMe ? '8px' : 'auto',

        left: msg.isMe ? 'auto' : '8px',

        bottom: '8px',

    }}

    >

    {new Date(msg.date).toLocaleTimeString([],
{hour: '2-digit', minute: '2-digit'})}

</Typography>

{msg.reaction && (

    <Typography

        sx={{

            mt: 1,

            display: 'inline-block'

        }}

    >

        {msg.reaction}

    </Typography>

    )}

</Box>

<Box

    sx={{

        mt: 0.5,

        display: 'flex',

        gap: 1,

```

```

        justifyContent: msg.isMe ? 'flex-end' :
'flex-start'

    }}
  >
    {!msg.isMe && (
      <div style={{display: 'none'}}>
        <Tooltip title='Ответить'>
          <IconButton size='small' onClick={() =>
setReplyTo(msg)}>
            <Icon icon='mdi:reply' />
          </IconButton>
        </Tooltip>
        <Tooltip title='Добавить реакцию'>
          <IconButton size='small' onClick={() =>
handleReaction(msg.id, '❤️')>
            <Icon icon='mdi:emoticon-happy-
outline' />
          </IconButton>
        </Tooltip>
      </div>
    ) }
  </Box>
</Box>
</Box>
)
}}}
</Box>

{replyTo && (
  <Box

```

```

        sx={{
          p: 1,
          px: 2,
          backgroundColor: theme.palette.grey[100],
          display: 'flex',
          alignItems: 'center',
          position: 'absolute',
          left: 0,
          width: '100%',
          top: '-35px'
        }}
      >
      <Typography variant='body2' sx={{flex: 1}}>
        Ответ на: <i>{replyTo.text}</i>
      </Typography>
      <IconButton size='small' onClick={() =>
setReplyTo(null)}>
        <Icon icon='mdi:close' />
      </IconButton>
    </Box>
  )}
<Box
  sx={{
    p: 4,
    borderTop: `1px solid ${theme.palette.divider}`,
    display: 'flex',
    alignItems: 'center',
    position: 'relative'
  }}

```

```

    }}
  >
  <TextField
    fullWidth
    size='small'
    multiline
    minRows={3}
    maxRows={3}
    placeholder='Написать сообщение...'
    value={message}
    onChange={e => setMessage(e.target.value)}
    onKeyPress={e => {
      if (e.key === 'Enter' && !e.shiftKey) {
        e.preventDefault()
        handleSendMessage()
      }
    }}
    sx={{mr: 1}}
    variant='standard'
    disabled={connectionStatus !== 'connected'}
  />
  <Button
    variant='contained'
    onClick={handleSendMessage}
    disabled={!message.trim() || connectionStatus !==
'connected'}
  >
    Отправить

```

```
        </Button>

    </Box>

</Box>

</Box>

)

}
```

## Приложение В

(обязательное)

Веб-приложение CRM-система. Фронтенд часть

Презентация

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 140  |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |



# ВЕБ-ПРИЛОЖЕНИЕ CRM-СИСТЕМА. ФРОНТЕНД ЧАСТЬ



Автор работы: Студент группы 21ВВП1 Тюрин В.И.  
Научный руководитель: к.т.н. доцент кафедры ВТ Карамышева Н.С.

@AyatoIT



# ВЕБ-ПРИЛОЖЕНИЕ CRM-СИСТЕМА. ФРОНТЕНД ЧАСТЬ

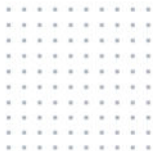
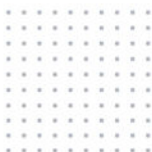


Автор работы: Студент группы 21ВВП1 Тюрин В.И.  
Научный руководитель: к.т.н. доцент кафедры ВТ Карамышева Н.С.

@AyatoIT

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                          | 141  |

# Актуальность проекта



04



## Постановка задачи

### Рост потребности в CRM-системах

Современный бизнес требует эффективных инструментов для управления клиентскими отношениями.

### Конкуренция на рынке

В условиях высокой конкуренции уровень клиентского сервиса становится ключевым фактором успеха.

### Цифровая трансформация

Проект соответствует тренду на цифровизацию бизнес-процессов, предоставляя доступные инструменты для автоматизации работы с клиентами.

### Интеграция искусственного интеллекта

Использование ИИ для автоматической обработки запросов в нерабочее время позволяет обеспечивать круглосуточную поддержку клиентов.

**Разработать клиентскую часть CRM-системы с интуитивно понятным интерфейсом, обеспечивающим:**

- Управление воронками и стадиями продаж.
- Интеграцию с Telegram и REST API.
- Общение с клиентами в реальном времени.
- Аутентификацию через JWT-токены.

|      |      |          |         |      |
|------|------|----------|---------|------|
|      |      |          |         |      |
| Изм. | Лист | № докум. | Подпись | Дата |

1 ПГУ 1.090301.22.001 ПЗ

Лист

142

05

# Преимущества

01

Бесплатный доступ

02

Простота интерфейса

03

Встроенный ИИ-ассистент

06

# Функционал

## Kanban

Управление воронками продаж (Drag-and-Drop).

## Реалтайм чат

Чат с клиентами в реальном времени.

## Источники

Интеграция с Telegram для создания сделок с клиентами

## Настройка

Настройка этапов и воронок сделок

|      |      |          |         |      |                          |      |
|------|------|----------|---------|------|--------------------------|------|
|      |      |          |         |      | 1 ПГУ 1.090301.22.001 ПЗ | Лист |
|      |      |          |         |      |                          | 143  |
| Изм. | Лист | № докум. | Подпись | Дата |                          |      |

# Технологии

## NextJs

Фреймворк на основе React

## REST

Архитектурный стиль для разработки сетевых приложений

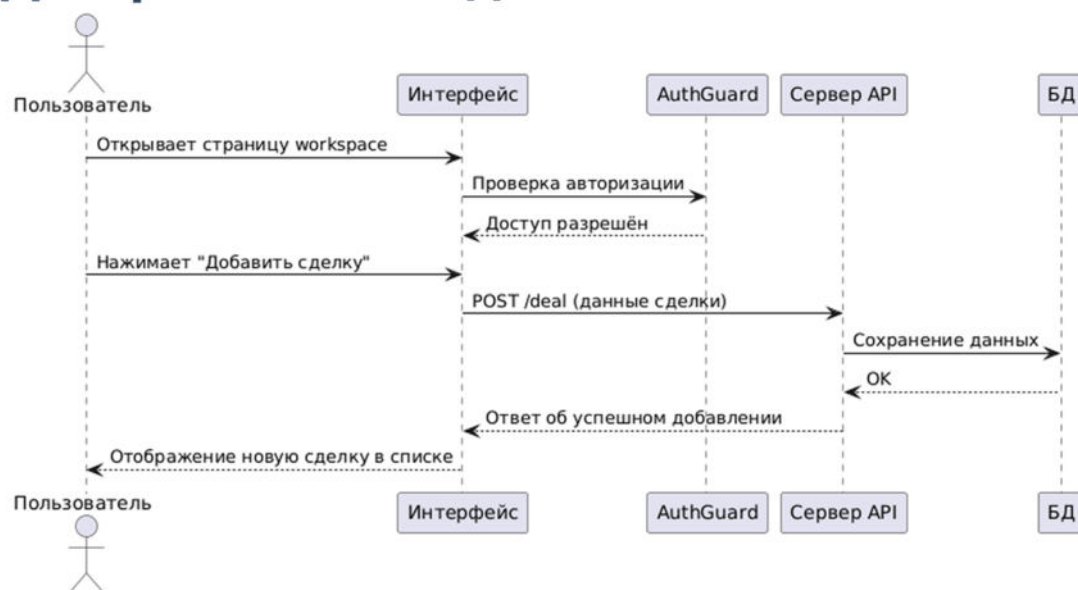
## Docker

ПО для доставки и запуска веб-приложений в средах с поддержкой контейнеризации

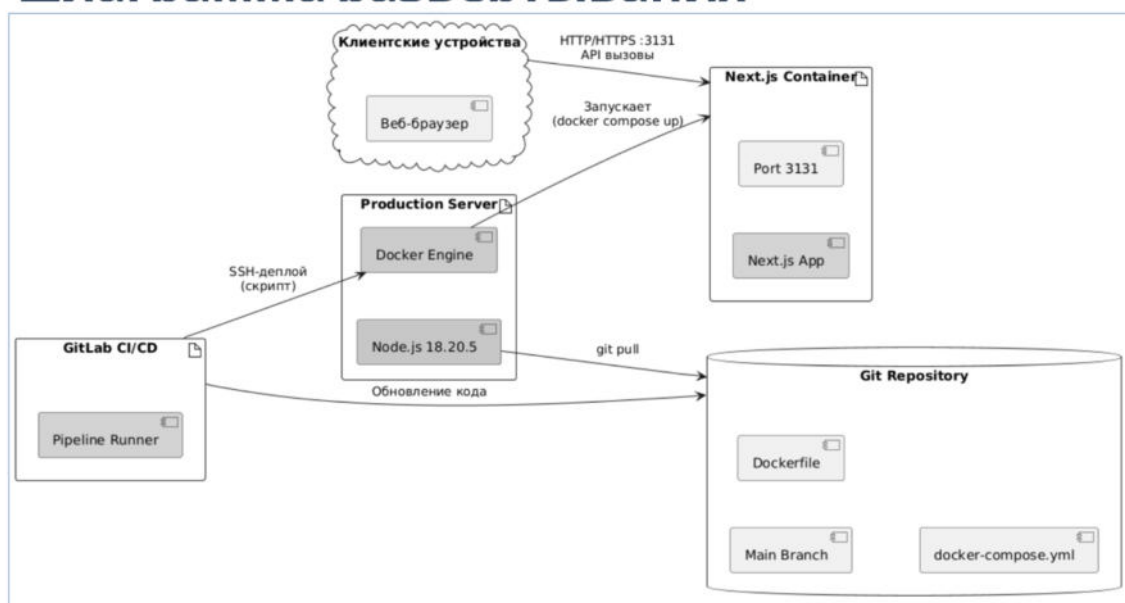
## Material-UI

Удобный и современный интерфейс

## Диаграмма последовательности



## Диаграмма развертывания



## Заключение и перспективы развития

Успешно реализована CRM-система, оптимизировала управление клиентами и сделками.

Перспективы:

- Интеграция с платёжными системами для бесшовных транзакций.
- Развитие чат-бота для мгновенной поддержки клиентов.
- Внедрение аналитики на основе ИИ для прогнозирования спроса.