

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

КЛИЕНТ-СЕРВЕРНАЯ
ИНФОРМАЦИОННАЯ СИСТЕМА
ВЕДЕНИЯ КЛИЕНТСКОЙ БАЗЫ МАЛОГО
И СРЕДНЕГО БИЗНЕСА

Студент

И.И. Ипполитов
(подпись, дата)

Ипполитов И.И.
(ФИО полностью)

Руководитель

А.С. Бычков
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Нормоконтролёр

А.С. Бычков
(подпись, дата)

А. С. Бычков
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2016 г. № 14)

Заведующий кафедрой

М.А. Митрохин
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

24 июня 2016 г. № 45)

Секретарь ГЭК

Е.И. Гурин
(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2026

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 07 » 04 20 26 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Ипполитов Иван Дмитриевич гр. 22ВВП1 факультета вычислительной техники

2. Тема работы: Клиент-серверная информационная система ведения клиентской базы для среднего и малого бизнеса

Тема утверждена приказом ПГУ № 347/0 от "07" апреля 2026 г.

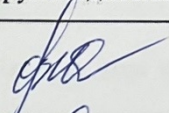
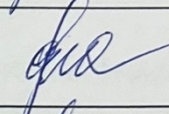
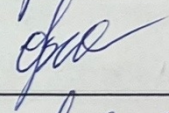
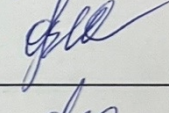
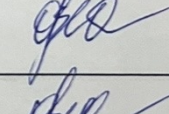
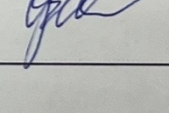
3. Руководитель бакалаврской работы к. т. н., доцент кафедры ВТ ПГУ Юрова О. В.

4. Задание на работу (назначение разработки, исходные данные и т.п.): Разработать веб-приложение с функциями управления клиентами, системой поиска, аутентификацией пользователей и разделением прав на обычного пользователя и администратора. Приложение должно позволять пользователю добавлять, удалять и редактировать клиентов, также искать клиента по ФИО, для администратора дополнительно должно позволять создать новых пользователей или администраторов и удалять их.

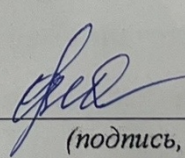
5. Перечень подлежащих разработке вопросов

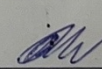
Требуется произвести анализ предметной области: изучить существующие аналоги веб-сервисов (amoCRM, Bitrix24, Megaplan, Excel), выявить их достоинства и недостатки в части управления данными и функциональности. Также необходимо изучить технологии для Разработки веб-приложения, такие как HTML, CSS, NodeJs, React, PostgreSQL, Prisma, а также принципы построения REST API

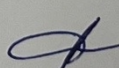
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	7.04.2026-14.04.2026	
2.	Анализ требований	8	15.04.2026-16.04.2026	
3.	Выбор технологий, применяемых при проектировании и разработке	40	17.04.2026-20.04.2026	
4.	Описание алгоритмов	50	21.04.2026-25.04.2026	
5.	Разработка веб-приложения	170	26.04.2026-25.05.2026	
6.	Оформление пояснительной записки	40	26.05.2026-10.06.3036	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы  Юрова О. В.
(подпись, дата)

Задание к исполнению принял студент  Ипполитов И. Д.
(подпись, дата)

Работу к защите допустить
Декан факультета  23.06.2026 Фионова Л. Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 65 листов, 12 рисунков, 8 источников и 3 приложения.

КЛИЕНТ-СЕРВЕРНАЯ ИНФОРМАЦИОННАЯ СИСТЕМА, ВЕБ-ПРИЛОЖЕНИЕ, CRM, УПРАВЛЕНИЕ КЛИЕНТСКОЙ БАЗОЙ, АУТЕНТИФИКАЦИЯ, УПРАВЛЕНИЕ ДОСТУПОМ, РАЗГРАНИЧЕНИЕ РОЛЕЙ, КРЕДИТ/ДЕБЕТ

Цель работы – разработать клиент-серверную информационную систему для ведения клиентской базы малого и среднего бизнеса с разграничением прав доступа и финансовым учётом.

Объектами разработки являются: серверная часть для аутентификации, авторизации и CRUD-операций; клиентская часть для взаимодействия с пользователем; база данных для хранения информации о клиентах и пользователях.

В результате проделанной работы проанализирована предметная область, проведён обзор существующих CRM-решений, выбраны инструменты разработки (JavaScript, React, Node.js, Express, PostgreSQL, Prisma), спроектирована архитектура по паттерну MVVM, реализовано и протестировано веб-приложение, обеспечивающее ведение клиентской базы, аутентификацию с JWT, разграничение ролей (администратор/пользователь), управление пользователями и финансовый учёт с цветовой индикацией.

					ПГУ 1.090301.13.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Клиент-серверная информационная система ведения клиентской базы для малого и среднего бизнеса Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Ипполитов И. Д.		14.06.26					
Провер.		Юрова О.В.		14.06.26			2	65	
Н. Контр.		Бычков А. С.		14.06.26		ФВТ, каф. ВТ			
Утверд.		Митрохин М.А.		14.06.26					

Содержание

Введение.....	5
1 Клиент-серверная информационная система для ведения.....	7
1.1 Анализ предметной области.....	7
1.2 Обзор аналогов.....	8
1.2.2 Платформа Bitrix24.....	9
1.2.3 CRM-система Megaplan.....	9
1.2.4 Таблицы Excel / GoogleSheets.....	10
1.2.5 Анализ аналогов.....	10
1.3 Постановка задачи.....	11
1.3.1 Назначение системы.....	11
1.3.2 Функциональные требования.....	11
1.3.3 Нефункциональные требования.....	12
2 Выбор инструментов разработки.....	16
2.1 Выбор языка программирования и среды выполнения.....	16
2.1.1 Единый язык для клиента и сервера.....	16
2.1.2 Высокая производительность асинхронного ввода-вывода.....	16
2.1.3 Богатая экосистема пакетов.....	16
2.1.4 Кроссплатформенность.....	17
2.1.5 Поддержка современных стандартов JavaScript.....	17
2.1.6 Интеграция с базами данных.....	17
2.1.7 Большое сообщество и документация.....	17
2.2 Выбор среды разработки: Visual Studio Code.....	17
2.2.1 Лёгкость и высокая производительность.....	18
2.2.2 Полноценная поддержка JavaScript «из коробки».....	18
2.2.3 Встроенный терминал и отладка.....	18
2.2.4 Обширная библиотека расширений.....	18
2.2.5 Интеграция с системами контроля версий.....	19
2.2.6 Бесплатность и кроссплатформенность.....	19

3 Архитектура приложения и выбор инструментов.....	20
разработки.....	20
3.1 Архитектура приложения.....	20
3.1.1 Клиентская часть.....	20
3.1.2 Серверная часть.....	21
3.1.3 База данных.....	22
3.1.4 UML диаграмма классов.....	26
3.2 Описание функционала.....	27
3.3 Описание работы приложения.....	28
3.4 Описание интерфейса приложения.....	32
3.5 Тестирование приложения.....	32
3.5.1 Функциональное тестирование.....	32
3.5.2 Интеграционное тестирование.....	33
3.5.3 Стресс-тестирование.....	33
3.5.4 Результаты тестирования.....	34
4 Безопасность жизнедеятельности.....	35
4.1 Введение.....	35
4.2 Требования к помещению и рабочему месту.....	35
4.3 Требования к освещению.....	36
4.4 Требования к режиму работы и отдыха.....	36
4.5 Рекомендации по использованию программного обеспечения.....	37
Заключение.....	38
Список использованных источников.....	40
Приложение А.....	41
Приложение Б.....	46
Приложение В.....	60

Введение

В современном быстро меняющемся мире, характеризующемся высокой конкуренцией и необходимостью оперативного принятия решений, эффективное управление взаимоотношениями с клиентами приобрело особую важность. Информация о покупателях, их предпочтениях, истории заказов и обращений перестала быть вспомогательным элементом деловой активности и превратилась в стратегический ресурс, напрямую влияющий на прибыльность и устойчивость предприятия. Для субъектов малого и среднего предпринимательства этот тезис звучит особенно остро: потеря даже нескольких постоянных клиентов из-за несвоевременного ответа или утери контактных данных способна критично отразиться на финансовых показателях.

Одной из распространённых проблем для малых и средних предприятий остаётся отсутствие структурированного подхода к хранению и обработке клиентских данных. До сих пор многие организации полагаются на бумажные носители, разрозненные таблицы Excel или отдельные заметки сотрудников в блокнотах и мессенджерах. Подобная практика не только замедляет рабочие процессы, но и порождает целый спектр рисков: дублирование записей, невозможность отследить полную историю общения с заказчиком, утрату информации при увольнении менеджера, а также несанкционированный доступ к конфиденциальным сведениям. Такие традиционные способы ведения клиентской базы не обеспечивают централизованного доступа, актуальности информации и защиты от потери данных, что неизбежно сказывается на качестве сервиса и уровне доверия со стороны потребителей.

Стремление преодолеть указанные недостатки стимулировало бурное развитие рынка CRM-систем (. Современные программные продукты предлагают широкий функционал: от простых картотек контактов до многофункциональных платформ с элементами искусственного интеллекта, управлением проектами и встроенной телефонией. Однако готовые решения далеко не всегда соответствуют специфике и финансовым возможностям

небольших компаний. Крупные облачные сервисы нередко обладают избыточной функциональностью, затрудняющей освоение, а их абонентская плата за каждого пользователя с течением времени формирует ощутимую статью расходов. Локальные же коробочные продукты либо требуют значительных вложений на старте, либо, напротив, предлагают слишком примитивный инструментарий, не позволяющий автоматизировать даже базовые бизнес-процессы.

Таким образом, сложившаяся ситуация высвечивает противоречие: малый и средний бизнес остро нуждается в системном учёте клиентов, но существующие программные средства либо не удовлетворяют его по критерию «цена – качество», либо оказываются технически и организационно труднодоступными. В связи с этим цель настоящей работы заключается в анализе предметной области, обзоре существующих аналогов и последующей разработке клиент-серверной информационной системы, ориентированной именно на потребности небольших предприятий. Для её достижения необходимо решить ряд задач: изучить типовые процессы взаимодействия с клиентами, выявить ключевые требования к функционалу, провести сравнительный анализ популярных CRM-решений, а также спроектировать и реализовать собственный программный продукт, сочетающий в себе минимально необходимый набор инструментов, интуитивно понятный интерфейс и невысокую стоимость владения.

Структура работы определена логикой проектирования информационной системы. В первом разделе представлен анализ предметной области, рассмотрены особенности автоматизации клиентского учёта в различных сферах деятельности и выполнен обзор аналогов, на основании которого сформулирована постановка задачи. Второй раздел посвящён проектированию системы, включая архитектурные решения и проектирование базы данных. В третьем разделе описываются этапы реализации и тестирования. Заключение содержит основные выводы и перспективы дальнейшего развития продукта.

1 Клиент-серверная информационная система для ведения клиентской базы малого и среднего бизнеса

1.1 Анализ предметной области

В последние годы автоматизация процессов управления клиентской базой (CRM-системы) постепенно вытеснила ручные методы учёта и обработки информации о клиентах. Всё больше представителей малого и среднего бизнеса предпочитают поручать ведение контактов, истории взаимодействий и контроль выполнения обязательств перед клиентами специализированным информационным системам, а не полагаться на память сотрудников или локальные файлы.

Следует отметить, что потребность в таких системах варьируется в зависимости от сферы деятельности. Например, в розничной торговле CRM помогает отслеживать покупательские предпочтения и формировать персональные предложения. В сфере услуг (медицина, образование, консалтинг) система позволяет записывать клиентов, напоминать о визитах и фиксировать историю обращений. В корпоративном секторе инструменты ведения клиентской базы используются для распределения потенциальных клиентов между менеджерами и координации работы отдела продаж.

Анализ статистики использования CRM-систем предоставляет дополнительные возможности для выявления закономерностей в поведении клиентов и эффективности работы менеджеров. Это позволяет определять сезонные колебания спроса, оценивать загруженность персонала и прогнозировать будущие продажи. Например, система может автоматически предлагать время для повторного контакта с клиентом на основе анализа предыдущих успешных взаимодействий.

1.2 Обзор аналогов

Изучив множество источников сети Интернет, выяснилось, что системы для ведения клиентской базы имеют большую популярность. Однако многие из них либо избыточно сложны и дороги для малого бизнеса, либо, наоборот, предоставляют слишком примитивный функционал (например, просто список контактов). В настоящее время на рынке представлено множество как облачных, так и локальных решений. В приоритете сервисы, которые обеспечивают удобный интерфейс, клиент-серверную архитектуру и возможность масштабирования.

1.2.1 облачная система amoCRM

Первым на очереди можно рассмотреть amoCRM — облачную систему для управления взаимоотношениями с клиентами, ориентированную на малый и средний бизнес. Она позволяет вести базу контактов, строить воронку продаж, ставить задачи и фиксировать историю общения. Доступна через веб-интерфейс и мобильные приложения. Пример интерфейса представлен на рисунке 1.

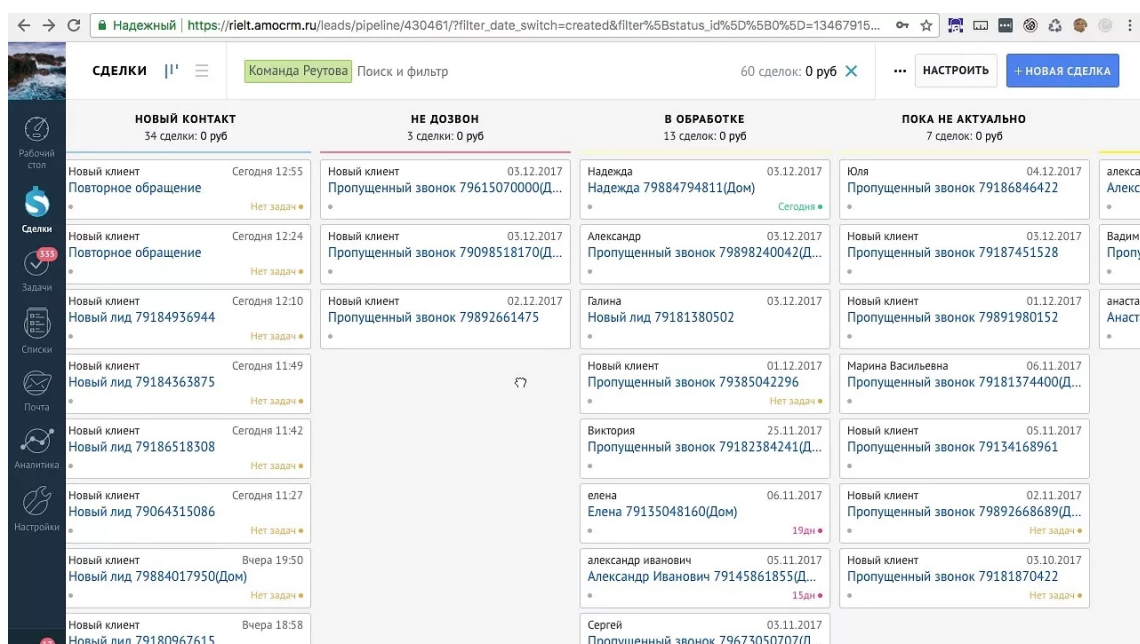


Рисунок 1 – Интерфейс amoCRM

1.2.2 Платформа Bitrix24

Вторым по счёту можно назвать Bitrix24 — многофункциональную платформу, включающую CRM, средства совместной работы, телефонию и документооборот. Она подходит для компаний, которым нужна не только клиентская база, но и внутренние коммуникации. Однако из-за обилия функций Bitrix24 может оказаться избыточным для очень малого бизнеса. Пример интерфейса представлен на рисунке 2.

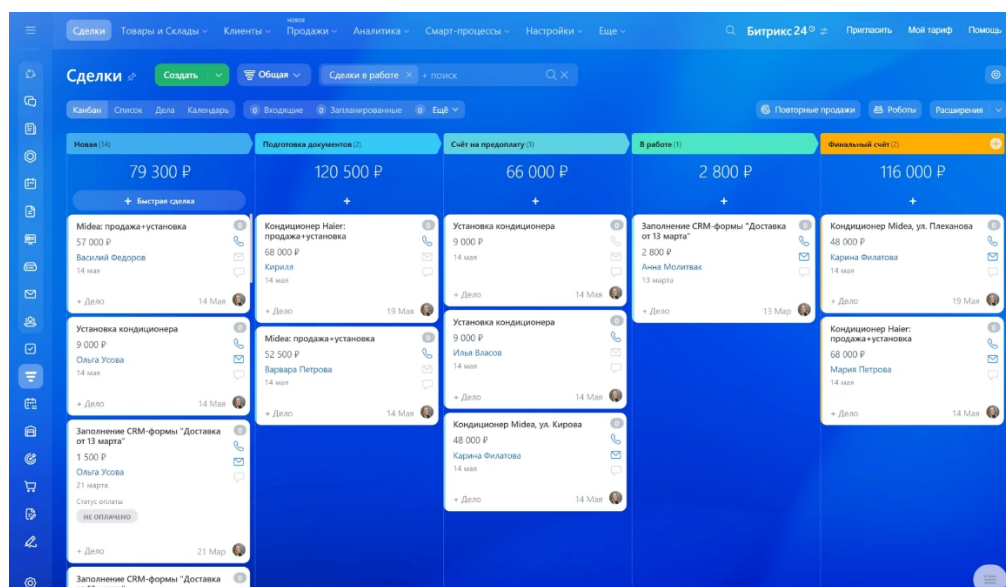


Рисунок 2 – Интерфейс Bitrix24

1.2.3 CRM-система Megaplan

Третьим вариантом выступает Megaplan — CRM-система, делающая упор на управление продажами и проектами. Она предоставляет удобный интерфейс для ведения карточек клиентов, планирования встреч и контроля выполнения задач. Интерфейс представлен на рисунке 3.

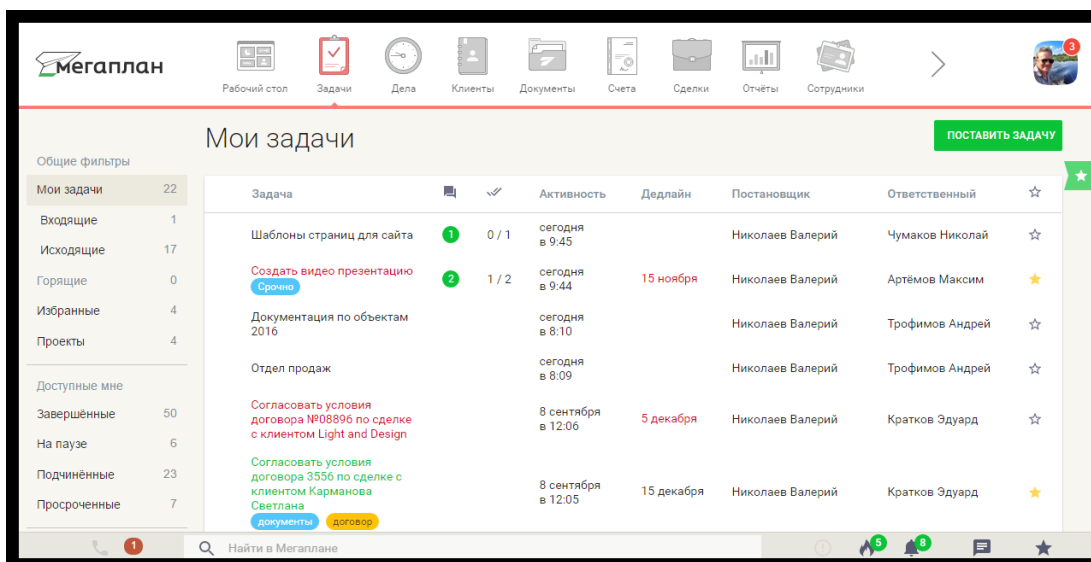


Рисунок 3 – Интерфейс Megaplan

1.2.4 Таблицы Excel / GoogleSheets

Ознакомившись с полнофункциональными CRM, следующий выбор пал на простые и доступные решения, такие как «Простая CRM» или даже Microsoft Excel с шаблонами для учёта клиентов. Эти инструменты не требуют финансовых затрат, но не обеспечивают многопользовательский доступ, разграничение прав, автоматические напоминания и защиту от конфликтов данных.

1.2.5 Анализ аналогов

После анализа и тестирования различных сервисов для управления клиентской базой, включая amoCRM, Bitrix24, Megaplan и табличные процессоры, были сделаны следующие выводы.

amoCRM зарекомендовала себя как удобный инструмент для малого бизнеса благодаря простому интерфейсу и гибкой настройке воронок продаж. Однако её функционала может не хватать для глубокой аналитики и интеграции с некоторыми учётными системами.

Bitrix24, с другой стороны, представляет собой мощную экосистему, но её внедрение требует больше времени и ресурсов, а интерфейс может показаться перегруженным для небольшой компании.

Megaplan отличается хорошей структурой управления задачами и проектами, однако стоимость лицензий для нескольких пользователей может быть существенной для стартапов.

Табличные процессоры — полностью бесплатны и просты в освоении, но не обеспечивают клиент-серверную архитектуру, централизованное хранение с контролем версий и защиту от несанкционированного доступа.

Таким образом, можно сделать вывод, что существующие решения имеют различные преимущества и ограничения. Для многих малых и средних предприятий актуальна разработка собственной клиент-серверной информационной системы, которая сочетала бы в себе необходимый минимум функций (ведение базы, поиск, история взаимодействий, напоминания) с низкой стоимостью владения, простотой использования и возможностью развёртывания на локальном сервере или в облаке.

1.3 Постановка задачи

Постановка задачи для разработки клиент-серверной информационной системы ведения клиентской базы малого и среднего бизнеса была следующей.

1.3.1 Назначение системы

Система предназначена для централизованного хранения, управления и обработки информации о клиентах организации. В отличие от аналогов, упор делается на лёгкость развёртывания, интуитивно понятный интерфейс и базовый набор функций, необходимых для эффективной работы отдела продаж или службы поддержки.

1.3.2 Функциональные требования

Основные функциональные требования:

- добавление, редактирование и удаление карточек клиентов (ФИО, контакты, интересы, баланс);
- поиск и фильтрация клиентов по ФИО в реальном времени (через строку поиска);

- аутентификация и авторизация пользователей (регистрация, вход, выход);
- разграничение ролей: администратор (полный доступ к клиентам и управлению пользователями) и обычный пользователь (только работа с клиентской базой);
- управление пользователями (для администратора): создание и удаление учётных записей, назначение ролей;
- финансовый учёт: отображение баланса клиента (кредит/дебет) с цветовой индикацией строк;
- система должна быть реализована по клиент-серверной модели (REST API + веб-интерфейс), все операции с клиентами и пользователями защищены JWT-токенами.

1.3.3 Нефункциональные требования

Система должна удовлетворять следующим нефункциональным требованиям, касающимся качества эксплуатации, производительности, безопасности и удобства работы.

Производительность и отзывчивость:

- время отклика интерфейса при выполнении операций (поиск, добавление, редактирование, удаление) не должно превышать 2 секунд при штатной нагрузке (до 10 одновременных пользователей и до 10 000 записей в БД);
- время первоначальной загрузки веб-приложения (First Contentful Paint) не должно превышать 3 секунд при скорости интернета от 10 Мбит/с;
- серверная часть должна обрабатывать не менее 50 запросов в секунду без деградации производительности;
- на клиенте используется контекст для поиска, что исключает повторные запросы к серверу при фильтрации; фильтрация списка клиентов по ФИО происходит мгновенно без задержек, связанных с сетевым взаимодействием.

Надёжность и отказоустойчивость:

- система должна функционировать без критических сбоев в течение рабочего дня (8 часов) при штатном использовании;
- при некорректном завершении работы сервера ранее сохранённые данные не должны теряться (обеспечение свойств ACID на уровне СУБД);
- в случае сбоя соединения или ошибки сервера клиентская часть должна отображать понятное сообщение и предлагать повторную попытку;
- система должна поддерживать автоматическое или ручное резервное копирование базы данных.

Безопасность:

- доступ к функциям добавления, редактирования и удаления клиентов должен быть ограничен авторизованными пользователями;
- пароли пользователей хешируются с использованием библиотеки bcrypt, сессии не хранятся — вместо них используется JWT (JSON Web Token) с ограниченным сроком жизни (7 дней);
- доступ к API защищён middleware, проверяющим валидность токена и роль пользователя. Все запросы, кроме /auth/login и /auth/register, требуют наличия валидного JWT в заголовке Authorization: Bearer <token>;
- при передаче данных по открытым сетям используется шифрование (HTTPS);
- все входные данные, поступающие от клиента (поля форм), проходят валидацию и санитацию на сервере для предотвращения XSS-атак и SQL-инъекций (Prisma ORM автоматически параметризует запросы);
- администратор имеет доступ ко всем операциям с клиентами и к разделу «Управление пользователями»; обычный пользователь видит только таблицу клиентов и может выполнять CRUD-операции с ними (добавление, редактирование, удаление, просмотр).

Масштабируемость и расширяемость:

- система поддерживает увеличение объёма клиентской базы до 100 000 записей без изменения архитектуры;
- архитектура позволяет добавление новых полей в карточку клиента без переписывания существующих модулей (благодаря использованию JSON-поля contacts и гибкой модели Prisma).

Удобство использования:

- интерфейс интуитивно понятен и не требует специального обучения;
- основные операции (добавление, редактирование, удаление, поиск) доступны через стандартные элементы управления;
- критически важные действия (удаление клиента, удаление пользователя) запрашивают подтверждение через модальное окно;
- все длительные операции (загрузка данных, сохранение) сопровождаются визуальными индикаторами (спиннеры, индикация загрузки в кнопках).

Технические требования к окружению:

- серверная часть: Node.js версии 18.x или выше, PostgreSQL версии 14.x или выше, не менее 1 ГБ оперативной памяти;
- клиентская часть: современные браузеры (Google Chrome, Mozilla Firefox, Safari, Edge) с поддержкой HTML5, CSS3, JavaScript (ES6+);
- среда разработки: Visual Studio Code с расширениями ESLint, Prettier, Prisma, Thunder Client;
- протокол взаимодействия: HTTP/1.1 (при необходимости может быть заменён на HTTPS с использованием SSL-сертификата).

UML диаграмма вариантов использования представлена на рисунке 4

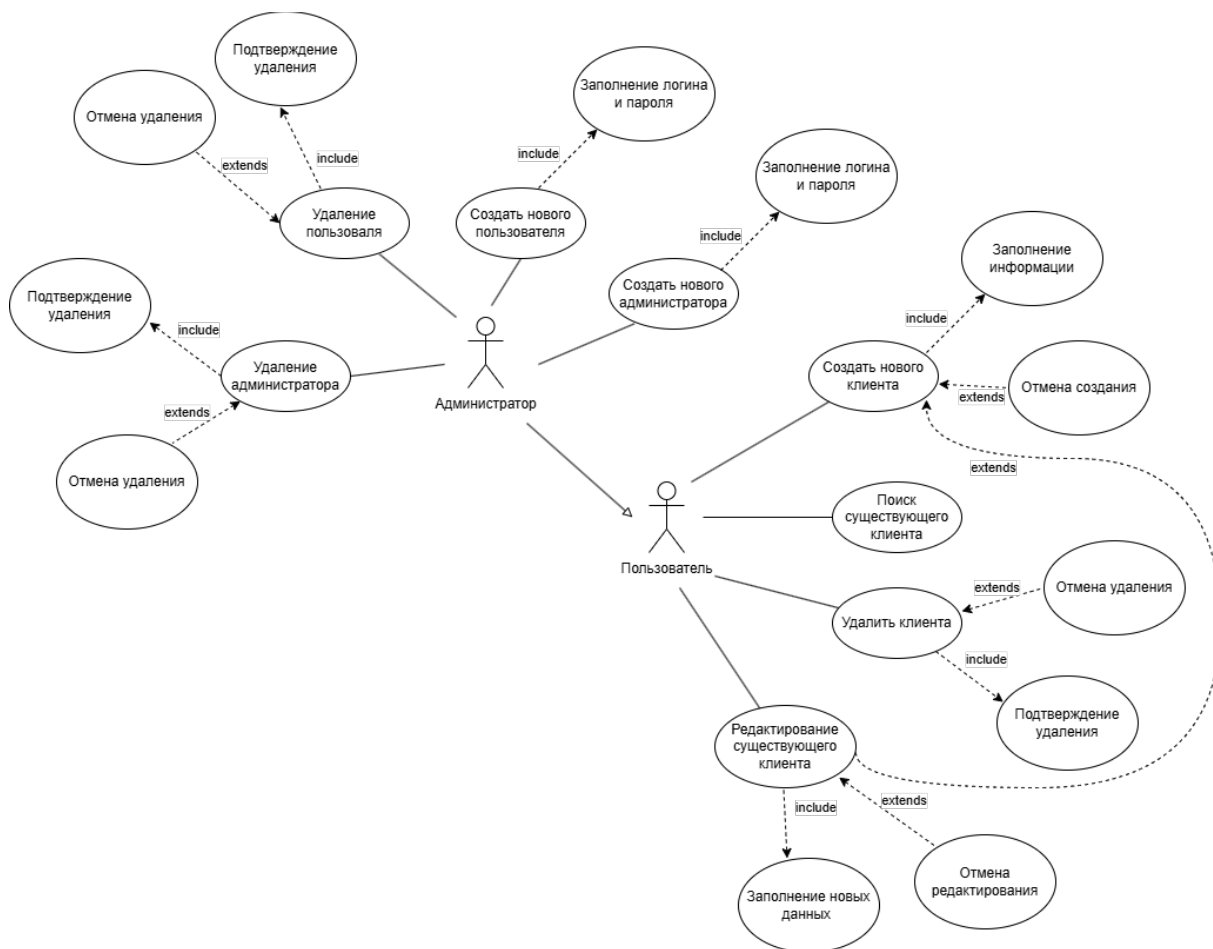


Рисунок 4 – UML диаграмма вариантов использования

2 Выбор инструментов разработки

2.1 Выбор языка программирования и среды выполнения

Для реализации клиент-серверной информационной системы ведения клиентской базы был выбран язык JavaScript и среда выполнения Node.js на сервере. На клиенте используется библиотека React. Такой выбор обусловлен рядом причин.

2.1.1 Единый язык для клиента и сервера

JavaScript позволяет писать как фронтенд (React) [1], так и бэкенд (Node.js) [2] на одном языке. Это упрощает обмен данными в формате JSON, повторное использование кода и сокращает время разработки. Разработчику не нужно переключаться между разными синтаксисами и парадигмами.

2.1.2 Высокая производительность асинхронного ввода-вывода

Node.js построен на неблокирующей событийно-ориентированной архитектуре. Это позволяет серверу эффективно обрабатывать множество параллельных запросов от клиентов (добавление, поиск, экспорт данных), не создавая отдельный поток на каждый запрос. Такой подход особенно полезен для систем с интенсивным обменом данными.

2.1.3 Богатая экосистема пакетов

Менеджер пакетов npm (или yarn) предоставляет десятки тысяч готовых библиотек для решения любых задач:

- для работы с базой данных – Prisma [3], Sequelize;
- для экспорта/импорта данных – ExcelJS, csv-parse;
- для построения API – Express [4];
- для авторизации – JSON Web Token [5], bcrypt [6];
- для работы с датами – date-fns.

Это значительно ускоряет разработку по сравнению с написанием всех компонентов с нуля.

2.1.4 Кроссплатформенность

Node.js-приложения работают на Windows, macOS и Linux. Клиентская часть на React запускается в любом современном браузере. Таким образом, система не привязана к конкретной операционной системе и может быть развернута практически на любом хостинге.

2.1.5 Поддержка современных стандартов JavaScript

Современный JavaScript (ES2020+) поддерживает асинхронные функции (async/await), модули (import/export), классы, стрелочные функции, что позволяет писать чистый, понятный и легко поддерживаемый код. React использует функциональные компоненты и хуки, что делает интерфейс декларативным и упрощает тестирование.

2.1.6 Интеграция с базами данных

Node.js имеет отличные драйверы для всех популярных СУБД: PostgreSQL [7], MySQL, MongoDB. В сочетании с ORM это обеспечивает удобную и безопасную работу с клиентскими данными, а также защиту от SQL-инъекций.

2.1.7 Большое сообщество и документация

JavaScript и Node.js входят в число самых популярных технологий в мире. Это гарантирует наличие множества обучающих материалов, готовых решений и ответов на вопросы, что особенно важно при выполнении дипломного проекта.

Таким образом, выбор JavaScript + React + Node.js позволяет реализовать современную, быструю и масштабируемую клиент-серверную систему с удобным веб-интерфейсом и гибкой серверной логикой.

2.2 Выбор среды разработки: Visual Studio Code

В качестве основной среды разработки была выбрана Visual Studio Code (VS Code) [8] – бесплатный и легковесный редактор кода от компании Microsoft. Выбор обусловлен следующими факторами:

2.2.1 Лёгкость и высокая производительность

VS Code запускается быстро, потребляет мало оперативной памяти и отлично работает даже на недорогих ноутбуках. Это особенно важно при частых переключениях между клиентской и серверной частями проекта.

2.2.2 Полноценная поддержка JavaScript «из коробки»

Редактор обеспечивает интеллектуальное автодополнение, подсветку синтаксиса, навигацию по коду (переход к определению, поиск ссылок), рефакторинг (переименование, выделение метода) и автоматическое форматирование. Встроенный линтер помогает находить потенциальные ошибки.

2.2.3 Встроенный терминал и отладка

VS Code имеет встроенный терминал, в котором можно запускать как сервер (Node.js), так и клиент (npm run dev). Отладчик позволяет устанавливать точки останова, просматривать значения переменных и стек вызовов как для серверного кода, так и для клиентского (при запуске через браузер).

2.2.4 Обширная библиотека расширений

Для VS Code существуют тысячи плагинов, упрощающих разработку:

- ESLint – статический анализ кода на предмет ошибок и стиля;
- Prettier – автоматическое форматирование;
- JavaScript (ES6) code snippets – полезные сниппеты;
- npm Intellisense – автодополнение для импорта модулей;
- Path Intellisense – автодополнение путей к файлам;
- Prisma – подсветка и автодополнение для схемы базы данных;
- Thunder Client – тестирование API без выхода из редактора;
- GitLens – удобная работа с Git.

2.2.5 Интеграция с системами контроля версий

Поддержка Git встроена прямо в интерфейс: можно видеть изменения, делать коммиты, пушить и пуллить без использования командной строки.

2.2.6 Бесплатность и кроссплатформенность

VS Code распространяется бесплатно и работает на Windows, Linux и macOS, что не требует дополнительных затрат и позволяет каждому разработчику использовать свою операционную систему.

Альтернативой могла бы служить WebStorm от JetBrains – мощная IDE с глубокой интеграцией для JavaScript, но она платная и более тяжёлая. Для целей дипломного проекта возможностей VS Code более чем достаточно.

Таким образом, выбор Visual Studio Code обеспечил комфортную разработку на React и Node.js, удобную отладку, контроль версий и гибкую настройку под специфику проекта.

3 Архитектура приложения и выбор инструментов разработки

3.1 Архитектура приложения

Разрабатываемое программное обеспечение представляет собой клиент-серверное веб-приложение, реализующее функциональность ведения клиентской базы, включая добавление, редактирование, удаление, поиск и фильтрацию записей. Архитектура разделена на три основные логические компонента, следующие паттерну MVVM (Model-View-ViewModel):

3.1.1 Клиентская часть

Представления (View) – компоненты, отвечающие за отображение интерфейса и взаимодействие с пользователем:

- Login – форма входа;
- Header – строка поиска, панель пользователя (имя, кнопка «Выйти»);
- Table – таблица клиентов;
- TableRow – строка таблицы с кнопками «Изменить» и «Удалить», а также условной окраской фона в зависимости от баланса;
- UserManagement – панель администратора для создания и удаления пользователей;
- модальные окна для добавления/редактирования клиента и подтверждения удаления.

Модель представления (ViewModel) – управляет состоянием и логикой интерфейса:

- хуки состояния (useState, useEffect);
- контексты: SearchContext, AuthContext;
- библиотека React Query (хуки useQuery, useMutation) для кеширования данных и синхронизации с сервером.

Модель (Model) – функции для отправки HTTP-запросов к серверу:

- api/hooks.js – обёртки над axios, автоматически добавляющие JWT-токен в заголовок Authorization;

- перехватчик (axios.interceptors) для обработки ошибок 401 и перенаправления на страницу входа.

3.1.2 Серверная часть

Маршруты (Routes):

- /clients – CRUD-операции с клиентами (GET, POST, PATCH, DELETE);

- /auth – регистрация, логин, получение информации о текущем пользователе;

- /admin/users – управление пользователями (только для роли ADMIN).

Контроллеры и бизнес-логика:

- валидация входных данных;
- хеширование паролей (bcrypt) [6];
- генерация и проверка JWT-токенов [5];
- промежуточные слои (authMiddleware, adminMiddleware) для защиты маршрутов.

Сервис базы данных (Prisma Client):

- модели Client и User;
- все операции с PostgreSQL выполняются через Prisma Client, что гарантирует типобезопасность и защиту от SQL-инъекций.

3.1.3 База данных

Таблица clients:

- id (автоинкремент, первичный ключ);
- fio – ФИО клиента;
- interests – сфера интереса (текст, может отсутствовать);
- contacts – JSON-поле для телефона, Telegram и других контактов;
- balance – числовое поле для кредита/дебета (по умолчанию 0);
- dateCreated, dateUpdated – даты создания и последнего изменения.

Таблица users:

- id (автоинкремент, первичный ключ);
- username – уникальный логин;
- passwordHash – хеш пароля (bcrypt);
- role – перечисление (ADMIN или USER);
- createdAt, updatedAt – даты создания и обновления записи.

На рисунке 5 представлена ER диаграмма

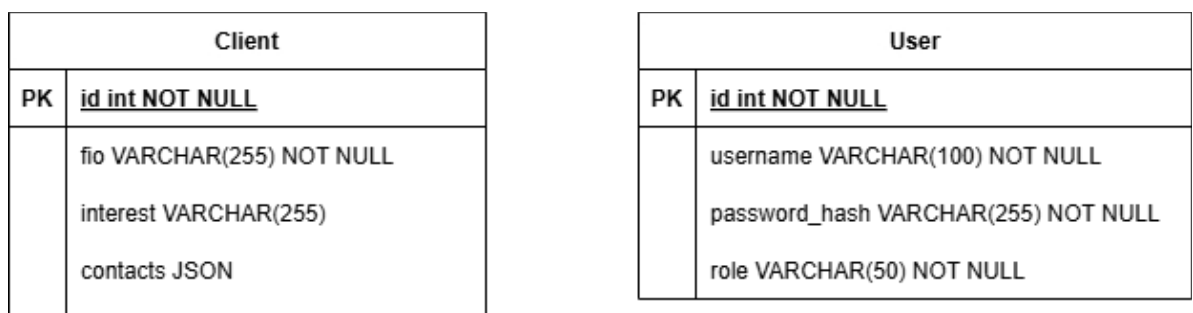


Рисунок 5 – ER-диаграмма

Диаграмма модулей (Рисунок 6) показывает клиентскую и серверную части. Пользователь взаимодействует с React-компонентами (Login, Header, Table). Действия пользователя вызывают обновление состояния (контексты,

React Query), затем через api/hooks отправляются HTTP-запросы к серверу. Сервер обрабатывает запросы, взаимодействует с БД через Prisma и возвращает JSON. Полученные данные обновляют интерфейс. Справа расположена серверная часть. Запросы поступают в маршруты Express, которые делегируют их контроллерам. Контроллеры через Prisma Client (Model) выполняют операции в PostgreSQL. Результат (JSON) возвращается клиенту. Полученные данные обновляют состояние, и React автоматически перерисовывает таблицу.

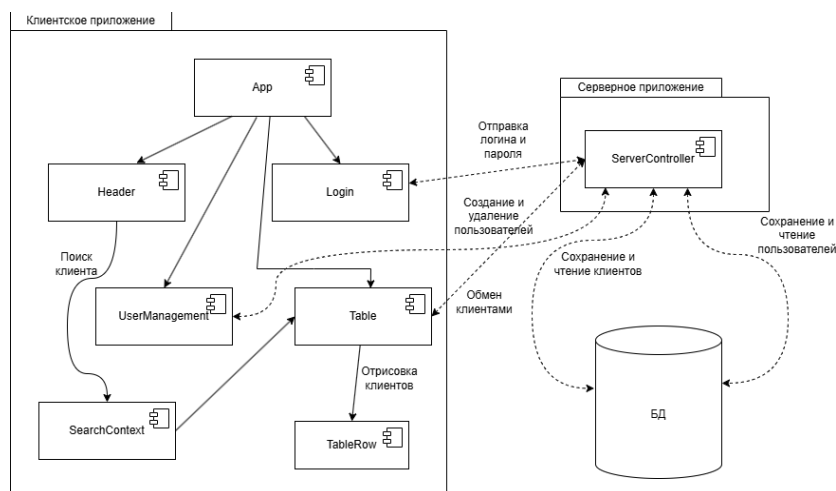


Рисунок 6 – Диаграмма модулей приложения

В центре — протокол HTTP и формат JSON, используемые для обмена данными между клиентом и сервером.

Эта архитектура обеспечивает модульность, масштабируемость и удобство разработки благодаря разделению ответственности и использованию единого языка JavaScript на всех уровнях.

Диаграмма последовательности представлена на рисунке 7

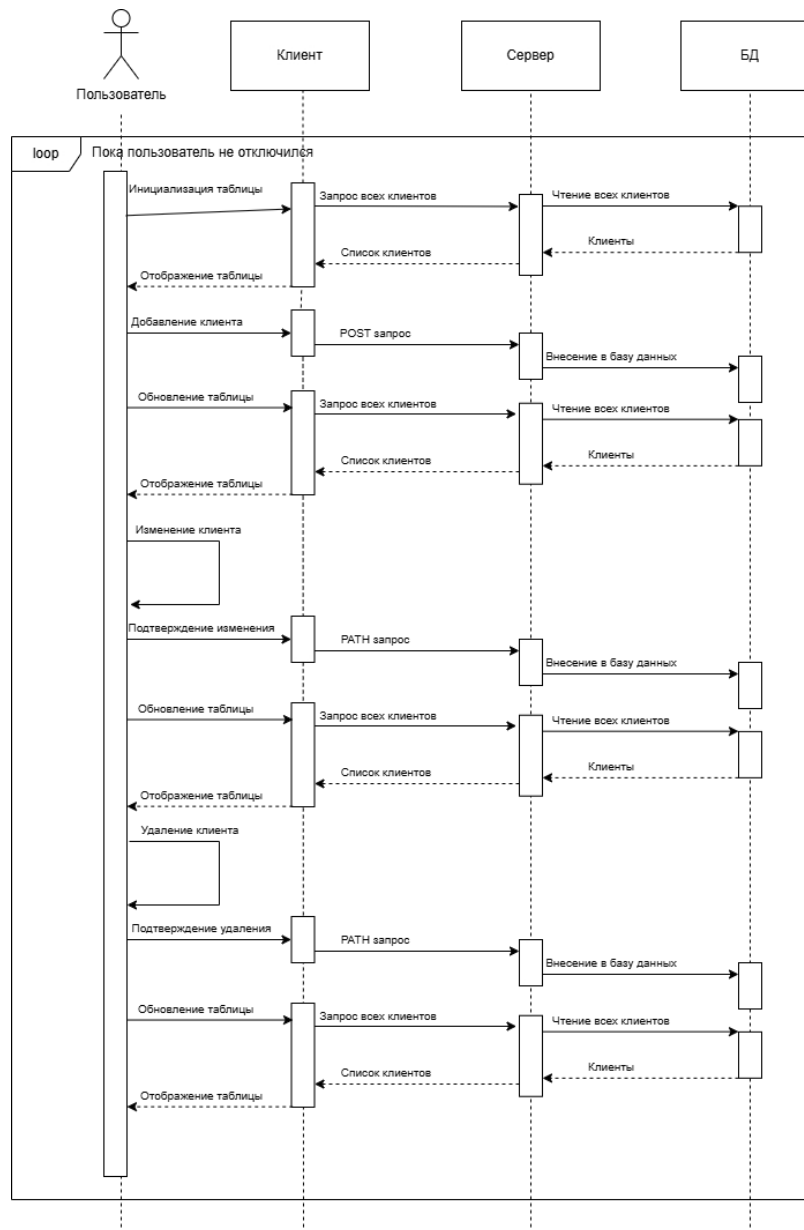


Рисунок 7 - Диаграмма последовательности

Диаграмма последовательности иллюстрирует процесс загрузки и отображения списка клиентов при открытии страницы.

Инициатор – пользователь открывает веб-страницу.

React-компонент Table (View) при монтировании вызывает функцию `getAllClients()` через хук `useQuery (ViewModel)`.

Функция `getAllClients` отправляет асинхронный GET-запрос на серверный маршрут `GET /clients`.

Express-маршрут принимает запрос и передаёт управление в контроллер.

Контроллер вызывает `prisma.client.findMany()` (Model) для получения всех записей из таблицы `clients`.

Prisma Client выполняет SQL-запрос к PostgreSQL и возвращает массив объектов.

Контроллер отправляет массив обратно клиенту в формате JSON.

React Query кеширует полученные данные и передаёт их в компонент Table (View).

Компонент Table отрисовывает строки таблицы, отображая информацию о клиентах.

Аналогичные диаграммы могут быть построены для операций добавления, удаления и редактирования, где шаги будут включать отправку POST/DELETE/PATCH запроса с телом (для добавления/изменения) или параметром ID (для удаления).

Диаграмма классов (Рисунок 8) описывает структуру ключевых компонентов:

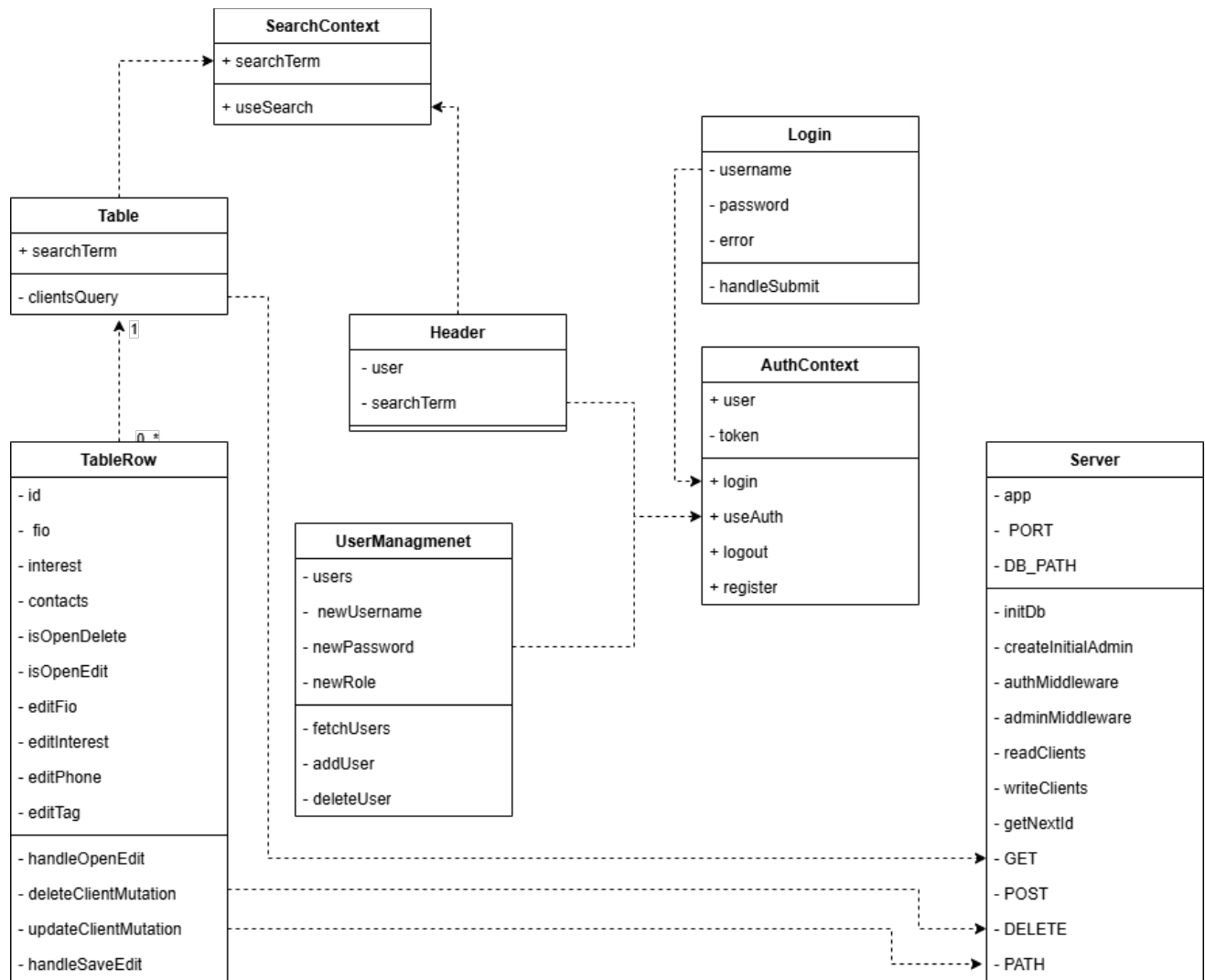


Рисунок 8 – Диаграмма классов

Клиентская часть (React):

- `TableRow` – компонент, отображающий одну строку таблицы. Содержит пропсы: `id`, `fio`, `interests`, `contacts`, `dateUpdated`.
- `Table` – компонент, который использует `useQuery` для получения данных и рендерит список `TableRow`.
- `ModalForm` – компонент модального окна для добавления/редактирования. Содержит локальное состояние для полей ввода.

- useMutation (из @tanstack/react-query) – хук для выполнения мутаций (добавление, удаление, изменение).

Серверная часть (Node.js):

- app – экземпляр Express, определяющий маршруты,
- GET /clients,
- POST /clients,
- DELETE /clients/:id,
- PATCH /clients/:id – обработчики маршрутов.
- PrismaClient – экземпляр ORM для работы с БД.

База данных (модель Prisma)

Client – модель с полями:

- id (Int),
- fio (String),
- interests (String?),
- contacts (Json),
- dateCreated (DateTime),
- dateUpdated (DateTime).

3.2 Описание функционала

Разрабатываемое приложение представляет собой систему для управления клиентской базой малого и среднего бизнеса. Основной функционал:

Просмотр списка клиентов – отображение таблицы с полями:

- ID,
- ФИО,

- сфера интереса,
- контакты (телефон, Telegram).
- Добавление нового клиента – модальное окно с полями:
- имя,
- фамилия,
- отчество,
- сфера интереса,
- телефон,
- Telegram.

Редактирование клиента позволяет изменять:

- ФИО,
- интересов
- контактов
- предоплаты или задолженности

Удаление клиента – подтверждение удаления через модальное окно.

Поиск клиентов – фильтрация списка по ФИО в реальном времени (реализована через контекст React).

3.3 Описание работы приложения

На рисунке 9 представлен вход в веб-приложение

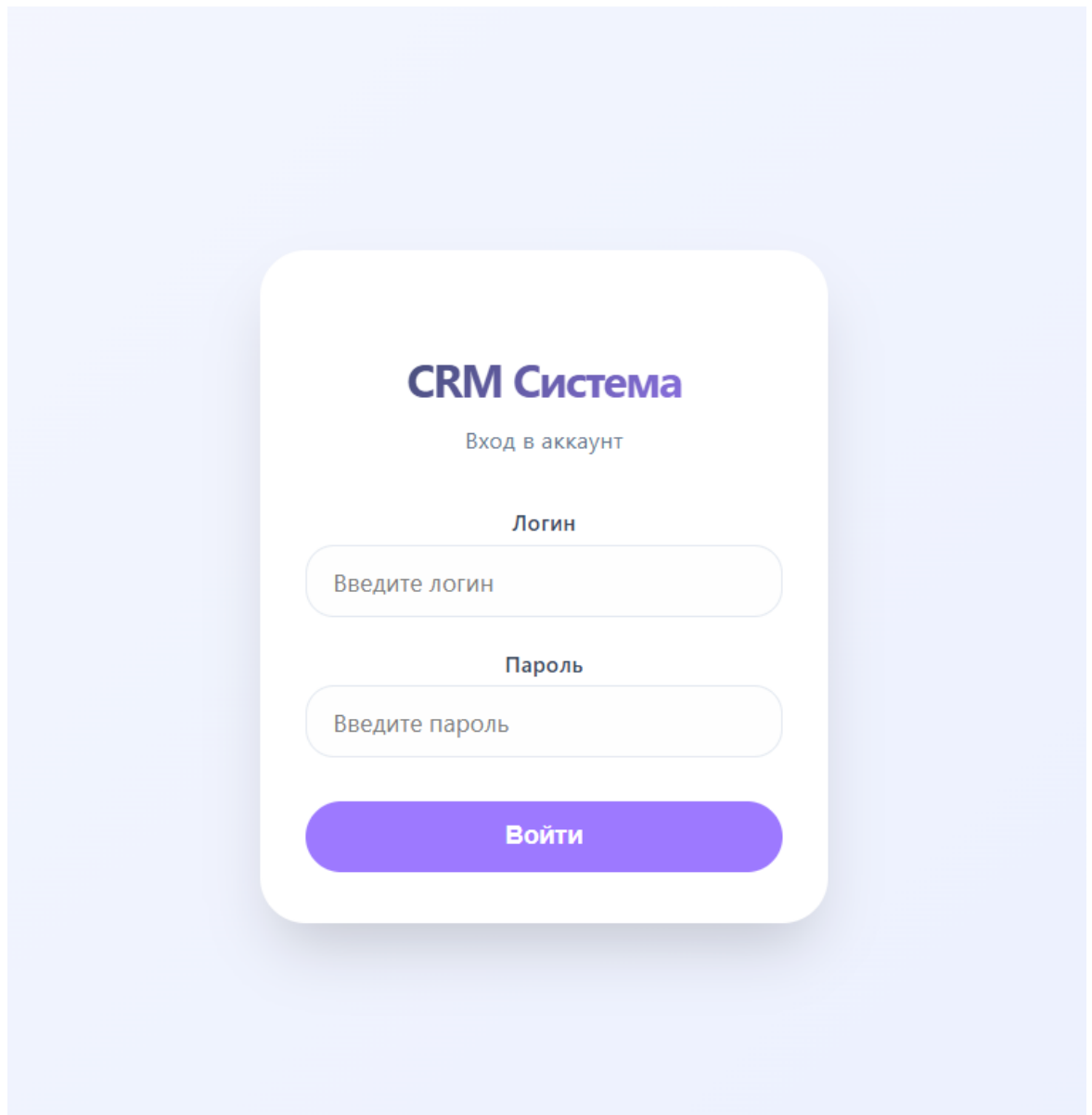
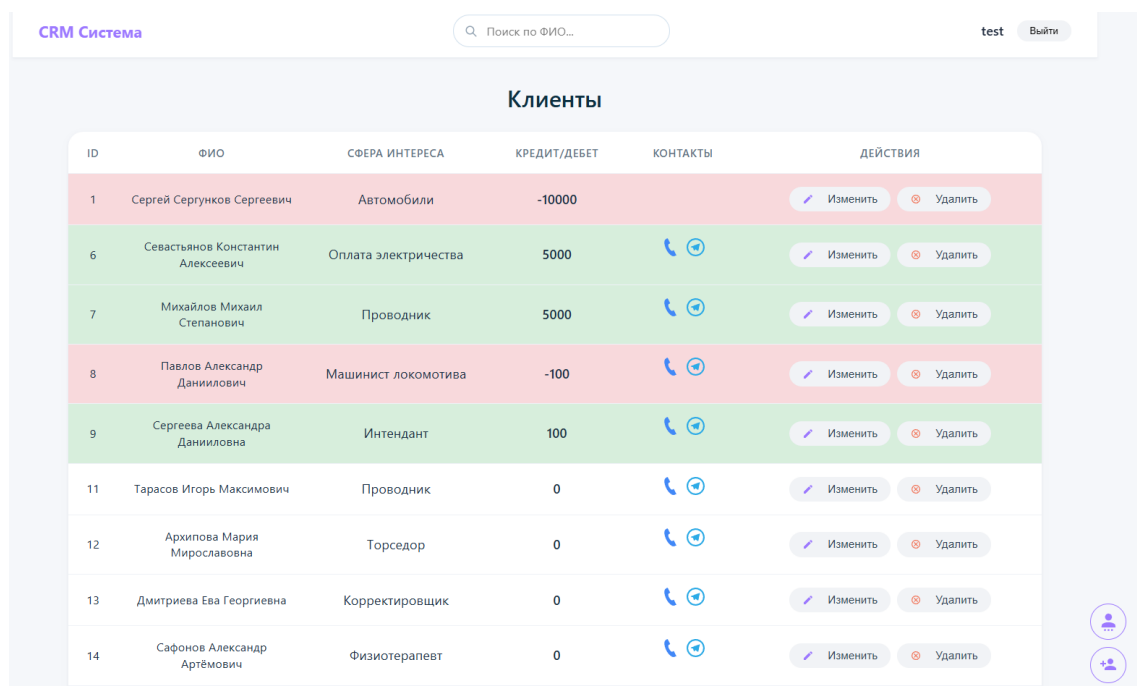


Рисунок 9 – Страница входа в веб-приложение

. Если пользователь не авторизован, он видит страницу входа. После ввода логина и пароля отправляется POST-запрос на `/auth/login`. При успешной проверке сервер возвращает JWT-токен, который сохраняется в `localStorage`. Клиент переходит к главному экрану.

На рисунке 10 представлен главная страница веб-приложения



The screenshot shows the main interface of a CRM system. At the top, there is a header with the text "CRM Система" on the left, a search bar "Поиск по ФИО..." in the center, and "test" and "Выйти" buttons on the right. Below the header, the title "Клиенты" is centered. The main content is a table with the following columns: ID, ФИО, СФЕРА ИНТЕРЕСА, КРЕДИТ/ДЕБЕТ, КОНТАКТЫ, and ДЕЙСТВИЯ. The table contains 14 rows of client data, alternating between light red and light green background colors. Each row has an "Изменить" (Edit) button with a pencil icon and a "Удалить" (Delete) button with a trash icon. On the right side of the table, there are two circular icons: one with a person icon and another with a plus icon.

ID	ФИО	СФЕРА ИНТЕРЕСА	КРЕДИТ/ДЕБЕТ	КОНТАКТЫ	ДЕЙСТВИЯ
1	Сергей Сергунков Сергеевич	Автомобили	-10000		Изменить Удалить
6	Севастьянов Константин Алексеевич	Оплата электричества	5000	📞 ✉️	Изменить Удалить
7	Михайлов Михаил Степанович	Проводник	5000	📞 ✉️	Изменить Удалить
8	Павлов Александр Данилович	Машинист локомотива	-100	📞 ✉️	Изменить Удалить
9	Сергеева Александра Даниловна	Интендант	100	📞 ✉️	Изменить Удалить
11	Тарасов Игорь Максимович	Проводник	0	📞 ✉️	Изменить Удалить
12	Архипова Мария Мирославовна	Торседор	0	📞 ✉️	Изменить Удалить
13	Дмитриева Ева Георгиевна	Корректировщик	0	📞 ✉️	Изменить Удалить
14	Сафонов Александр Артёмович	Физиотерапевт	0	📞 ✉️	Изменить Удалить

Рисунок 10 – Главная страница веб-приложения

На главном экране компонент Table с помощью useQuery отправляет GET-запрос на /clients (с токеном в заголовке). Сервер через Prisma Client выполняет prisma.client.findMany() и возвращает массив клиентов. React Query кеширует данные, и таблица отрисовывается.

При нажатии кнопки «Добавить» открывается модальное окно (Рисунок 11), где пользователь заполняет форму. После отправки формируется объект клиента и отправляется POST-запрос на сервер. Сервер валидирует данные, создаёт запись в БД и возвращает созданный объект. React Query инвалидирует кеш, таблица автоматически обновляется.

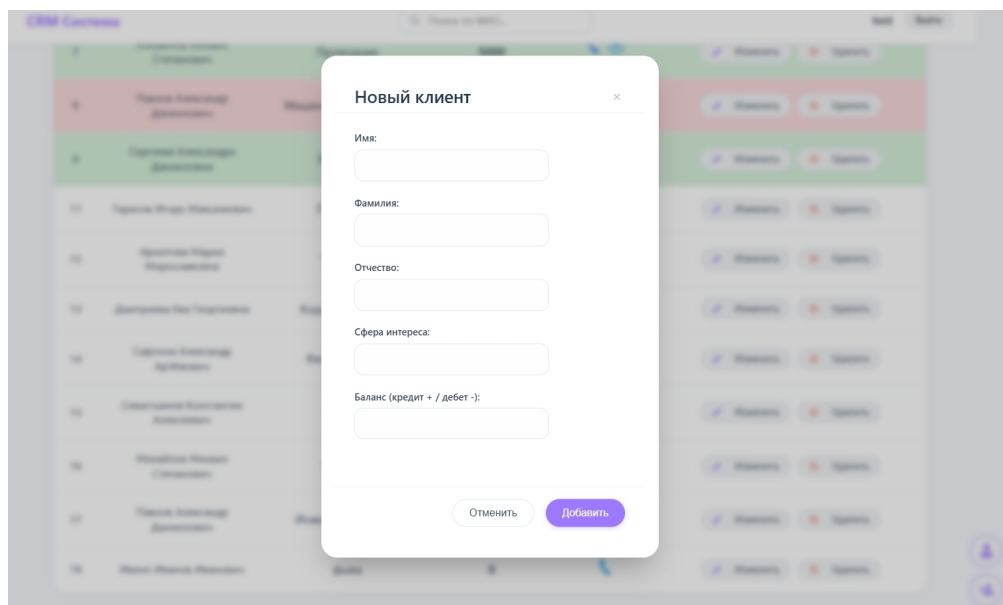


Рисунок 11 – Окно добавления клиента

Аналогично работают операции редактирования (PATCH) и удаления (DELETE). Поиск осуществляется на клиенте: значение из поля ввода передаётся через SearchContext в компонент Table, который фильтрует отображаемые строки без повторных запросов к серверу.

Администратор видит в интерфейсе дополнительную кнопку «Управление пользователями», которая открывает панель UserManagement (Рисунок 12). В этой панели можно просмотреть список пользователей, создать нового (логин, пароль, выбор роли) или удалить существующего.

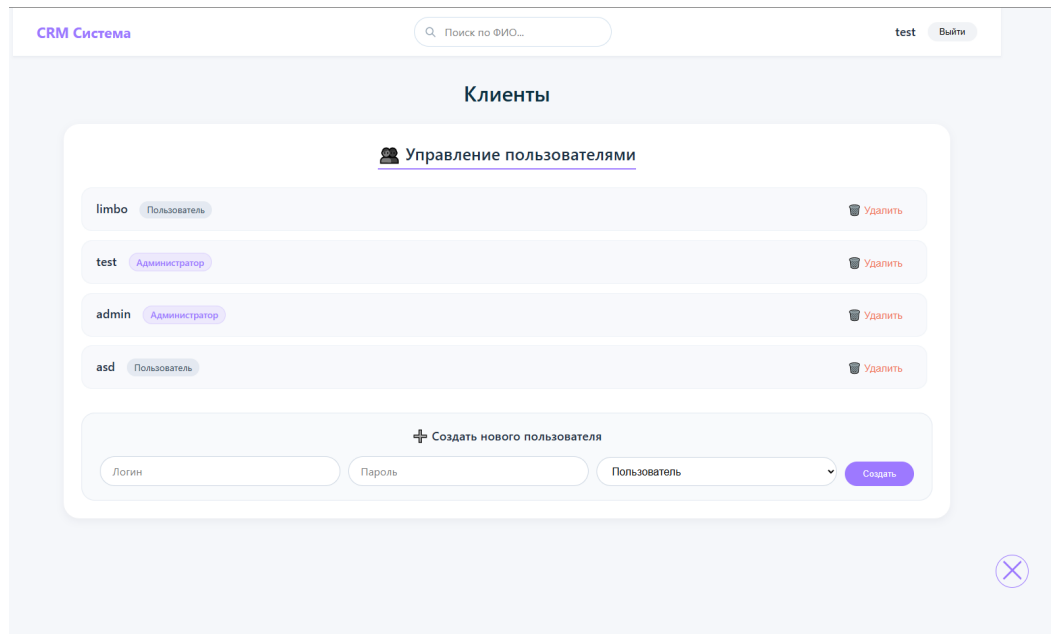


Рисунок 12 – Панель управления пользователями

3.4 Описание интерфейса приложения

Интерфейс разработан с использованием React и CSS, адаптирован под разные размеры экрана.

Страница входа – центрированная карточка с полями «Логин», «Пароль», кнопкой «Войти».

Главная страница (после входа) – шапка (логотип, поле поиска, имя пользователя, кнопка «Выйти»), таблица клиентов и плавающая кнопка добавления клиента (фиолетовый круг с плюсом).

Таблица клиентов – колонки: ID, ФИО, Сфера интереса, Кредит/Дебет, Контакты (иконки телефона и Telegram с всплывающими подсказками), Действия (кнопки «Изменить», «Удалить»). Клиенты с предоплатой имеют зелёный фон, с задолженностями – красный.

Модальные окна – для добавления/редактирования клиента (поля: имя, фамилия, отчество, сфера интереса, телефон, Telegram, баланс) и подтверждения удаления.

Панель управления пользователями (доступна только администратору) – отображает список пользователей в виде карточек, форму создания нового пользователя (логин, пароль, выбор роли) и кнопки удаления.

Интерфейс отображает состояние загрузки (текст «Загрузка...») и ошибки («Произошла ошибка. Повторить запрос»). Обновление данных происходит без перезагрузки страницы (SPA).

3.5 Тестирование приложения

После этапа разработки проведено функциональное, интеграционное и стресс-тестирование.

3.5.1 Функциональное тестирование

Ручное тестирование охватывало сценарии:

- Регистрация и вход с разными учётными записями (администратор, пользователь);
- Доступ к клиентской базе только после входа;
- Отображение списка клиентов, добавление / редактирование / удаление клиента;
- Проверка валидации полей (обязательность ФИО, допустимость пустого баланса);
- Цветовая индикация строк в зависимости от баланса (зелёный – положительный, красный – отрицательный);
- Поиск клиентов по ФИО в реальном времени (фильтрация без задержек);
- Переход на панель управления пользователями (только для администратора);
- Создание и удаление пользователей администратором;
- Выход из системы – очистка токена, перенаправление на логин.

3.5.2 Интеграционное тестирование

Проверено взаимодействие клиента, сервера и БД:

- Корректность REST-запросов (GET, POST, DELETE, PATCH) с разными телами и параметрами;
- Обработка сервером ошибочных данных (неполные поля, несуществующий ID);
- Своевременное обновление таблицы после мутаций (через `invalidateQueries`);
- Защита маршрутов: запросы без токена возвращают 401, запросы к `/admin/users` без роли ADMIN – 403.

3.5.3 Стресс-тестирование

Имитированы одновременные запросы от нескольких клиентов (с помощью Thunder Client и curl). Сервер на Node.js показал стабильную работу; время отклика оставалось в пределах допустимого (менее 2 с). Загрузка базы данных до 1000 записей не привела к заметному падению производительности.

3.5.4 Результаты тестирования

Все проверенные функции работают корректно. Выявленные в процессе отладки ошибки (например, некорректное обновление баланса, потеря контекста при редактировании) были устранены. Приложение соответствует заявленным функциональным и нефункциональным требованиям. Рекомендовано в дальнейшем внедрить автоматическое модульное тестирование с использованием Jest и нагрузочное тестирование с большим количеством параллельных сессий.

4 Безопасность жизнедеятельности

4.1 Введение

Безопасность при работе за компьютером играет важную роль для разработчиков и других специалистов, которые проводят значительное время за монитором. Невыполнение правил безопасности может привести к различным проблемам со здоровьем, таким как напряжение глаз, нарушения осанки, повышенный риск развития карпального туннельного синдрома и других профессиональных заболеваний.

В данном разделе рассматриваются основные аспекты безопасности при работе за компьютером и предоставляются рекомендации для разработчиков клиент-серверных информационных систем.

4.2 Требования к помещению и рабочему месту

При выборе места для рабочего места с компьютером рекомендуется отдавать предпочтение помещениям, ориентированным на север или северо-восток. Площадь одного рабочего места с компьютером и жидкокристаллическим монитором должна составлять не менее 4,5 квадратных метров.

Помещения, предназначенные для работы с компьютером, должны иметь как искусственное, так и естественное освещение, а также быть обеспечены достаточной вентиляцией для поддержания свежего воздуха и снижения концентрации вредных веществ, таких как формальдегид, который может выделяться из работающей техники. Концентрация вредных веществ в воздухе таких помещений не должна превышать установленных гигиенических норм. Рекомендуемый уровень шума в помещении не должен превышать 50 децибел, чтобы избежать отвлекающих звуков и избыточного стресса.

4.3 Требования к освещению

Неправильное освещение рабочего места негативно сказывается на зрении и может привести к серьёзным проблемам со здоровьем. Для предотвращения негативных последствий необходимо использовать качественный монитор, обеспечивать достаточное искусственное освещение и при необходимости применять защитные очки, доступные в специализированных магазинах.

В соответствии с гигиеническими нормами, рабочее место должно быть хорошо освещено, цветовая температура источников света должна находиться в диапазоне от 3900 до 6100 Кельвинов. Монитор желательно иметь с антибликовым покрытием и использовать со средним уровнем яркости, рекомендованным производителем в инструкции по эксплуатации.

4.4 Требования к режиму работы и отдыха

Соблюдение режима труда и отдыха является важнейшим аспектом сохранения здоровья при работе за компьютером. Переработки и постоянный труд без перерывов негативно сказываются на физическом и психологическом состоянии. Требования к режиму работы можно разделить на три категории: физические, психологические и организационные.

Физические требования:

- поддержание прямой осанки и правильного положения тела;
- выполнение разминки и гимнастики для глаз через каждые 45–60 минут работы;
- правильное расположение монитора на расстоянии вытянутой руки, с наклоном экрана около 30 градусов;
- обеспечение правильного освещения рабочего места.

Психологические требования:

- обязательные перерывы между периодами работы;
- смена деятельности в перерывах (прогулки на свежем воздухе, лёгкая физическая активность);

- избегание работы в выходные и праздничные дни.

Организационные требования:

- поддержание нормальной температуры воздуха в помещении (19–23°C);
- соблюдение графика работы и чёткого плана задач;
- организация полноценного отдыха в нерабочее время.

4.5 Рекомендации по использованию программного обеспечения

Для снижения нагрузки на зрение и поддержания работоспособности рекомендуется использовать специализированное программное обеспечение, включающее:

- инструменты управления шрифтом и масштабированием для снижения чувствительности глаз и обеспечения более плавной картинки;
- светофильтры, снижающие напряжение глаз при длительной работе за монитором;
- приложения для планирования рабочего времени и распределения задач, позволяющие оптимизировать график и повысить продуктивность труда.

Данные меры позволяют снизить негативное воздействие длительной работы за компьютером на здоровье разработчика и повысить общую эффективность трудового процесса.

Заключение

В ходе выполнения выпускной квалификационной работы была спроектирована, разработана и протестирована клиент-серверная информационная система для ведения клиентской базы малого и среднего бизнеса.

Был проведён анализ предметной области автоматизации клиентского учёта, выявлены типовые проблемы малых и средних предприятий: отсутствие структурированного хранения данных, дублирование записей, потеря информации при смене сотрудников, невозможность отследить историю взаимодействий. На основе анализа существующих CRM-решений (amoCRM, Bitrix24, Megaplan) были сформулированы требования к собственной системе, ориентированной на небольшой бизнес.

В работе обоснован выбор технологического стека: JavaScript, React, Node.js, Express, PostgreSQL, Prisma, JWT, bcrypt. Спроектирована архитектура по паттерну MVVM, разработана база данных, состоящая из двух независимых таблиц (клиенты и пользователи). Реализованы основные функциональные модули:

- ведение клиентской базы (CRUD-операции);
- аутентификация и авторизация с использованием JWT-токенов;
- разграничение ролей (администратор и пользователь);
- управление пользователями (для администратора);
- финансовый учёт с цветовой индикацией кредита/дебета;
- поиск и фильтрация клиентов по ФИО в реальном времени.

Проведённое тестирование подтвердило корректность работы всех функций, стабильность системы и соответствие заявленным требованиям. Интерфейс приложения интуитивно понятен, адаптивен и не требует специального обучения.

В результате выполнения работы были получены и закреплены практические навыки разработки современных веб-приложений на JavaScript с

использованием React и Node.js, работы с реляционными базами данных через ORM Prisma, проектирования безопасной аутентификации и реализации ролевой модели доступа.

Дальнейшее развитие проекта может включать:

- добавление истории взаимодействий с клиентами (звонки, встречи, письма);
- реализацию экспорта и импорта данных в форматы CSV и XLSX;
- создание системы отчётов и аналитики;
- внедрение модуля автоматических напоминаний и задач;
- развёртывание системы в облачной инфраструктуре.

Опыт, полученный в процессе выполнения работы, является ценным для дальнейшего профессионального развития и трудоустройства по специальности в области разработки программного обеспечения.

Список использованных источников

1. React — Официальная документация React [Электронный ресурс]. – Режим доступа: <https://react.dev/>, свободный.
2. Node.js — Официальная документация Node.js [Электронный ресурс]. – Режим доступа: <https://nodejs.org/en/learn>, свободный.
3. Prisma ORM — Официальная документация Prisma ORM [Электронный ресурс]. – Режим доступа: <https://www.prisma.io/docs>, свободный.
4. Express.js — Официальная документация Express.js [Электронный ресурс]. – Режим доступа: <https://expressjs.com/>, свободный.
5. JWT — Официальный сайт JSON Web Tokens [Электронный ресурс]. – Режим доступа: <https://jwt.io/>, свободный.
6. bcrypt — Пакет bcrypt на npm [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/bcrypt>, свободный.
7. PostgreSQL — Официальная документация PostgreSQL [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/docs/>, свободный.
8. Visual Studio Code — Официальная документация VS Code [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com/docs>, свободный.

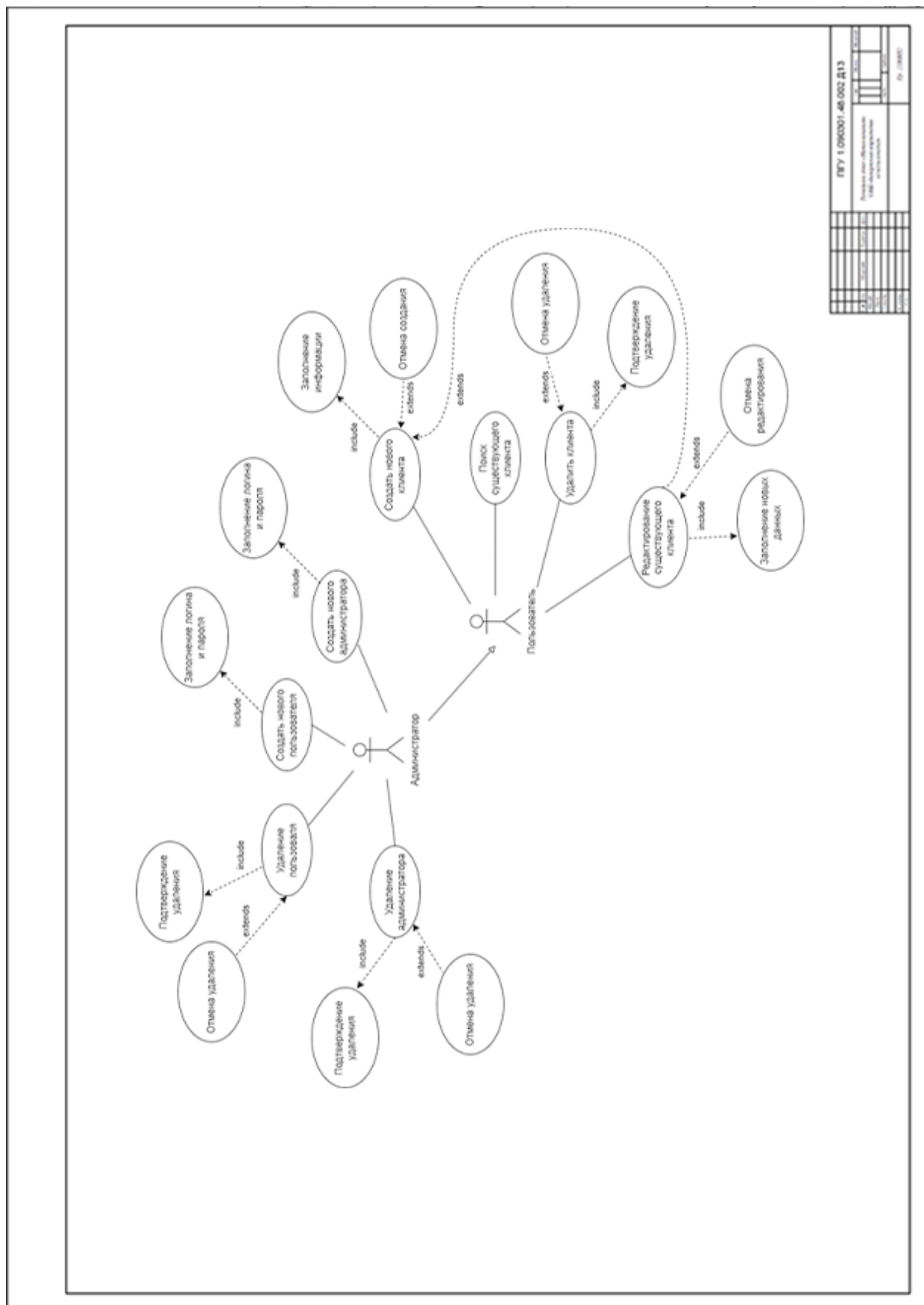
Приложение А

(обязательное)

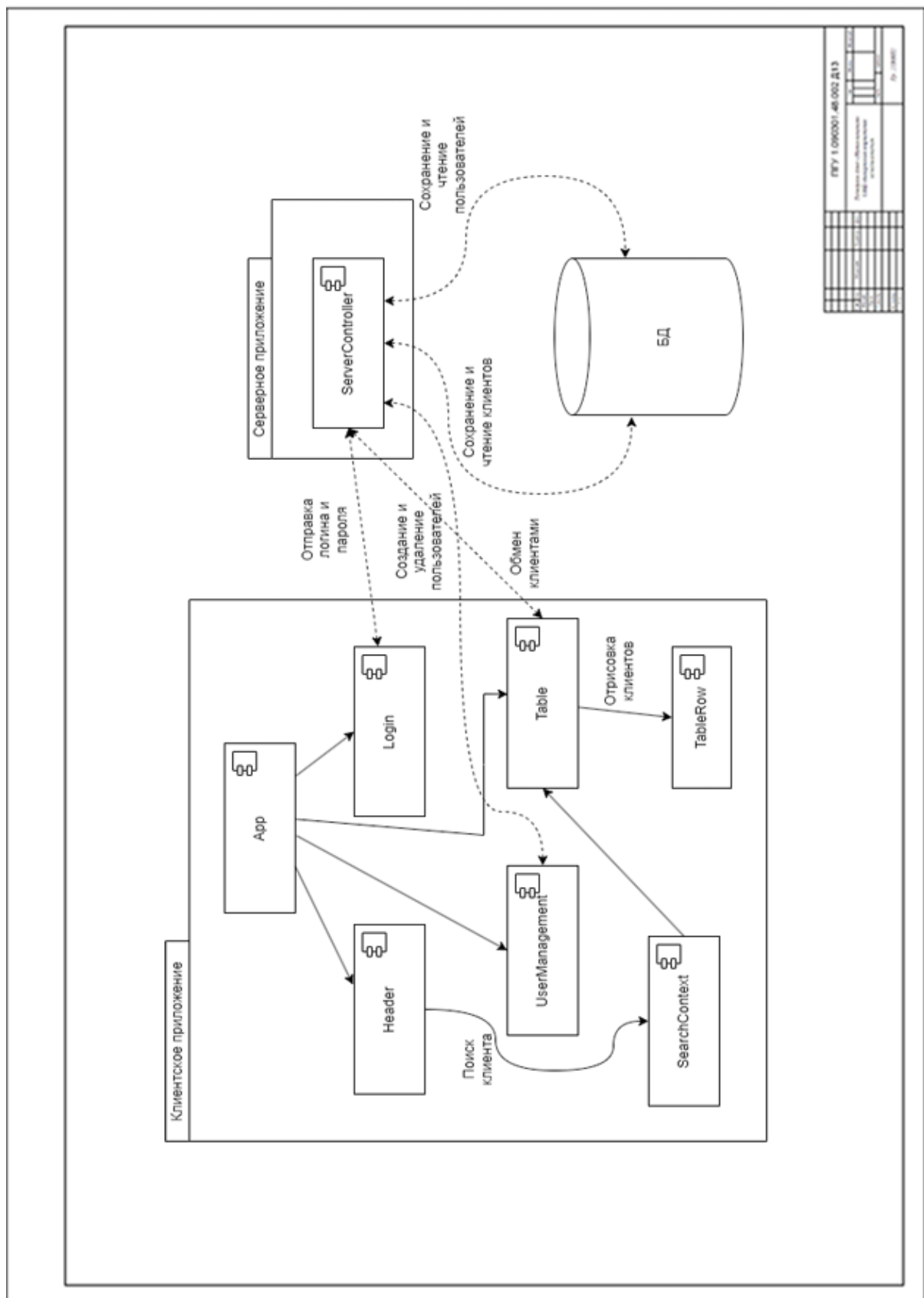
Клиент-серверная информационная система для
ведения клиентской базы малого и среднего бизнеса

UML–диаграммы

UML– диаграмма вариантов использования



UML-диаграмма взаимодействия компонентов



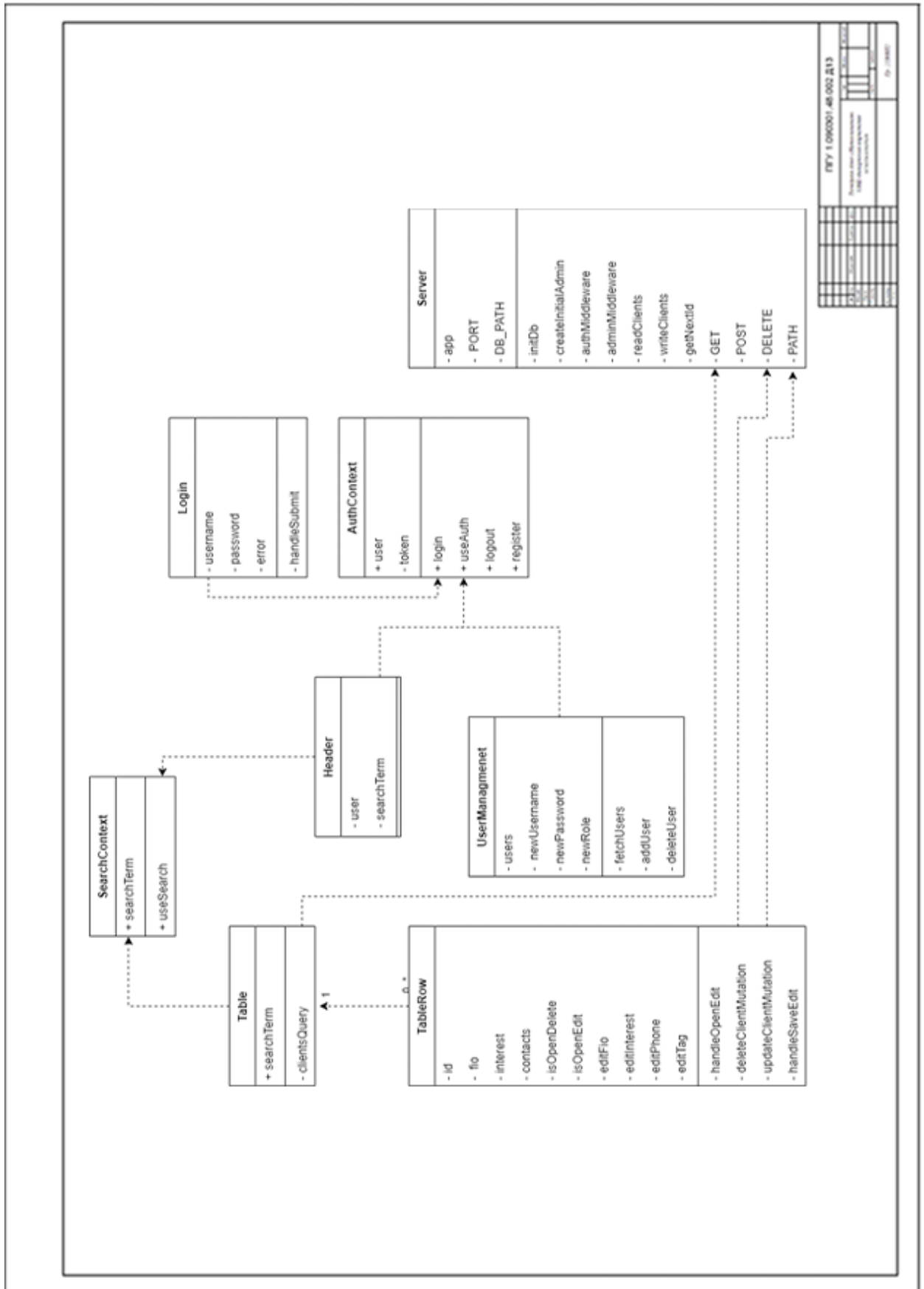
UML-диаграмма объектов

Client	
PK	<u>id</u> int NOT NULL
	fio VARCHAR(255) NOT NULL interest VARCHAR(255) contacts JSON

User	
PK	<u>id</u> int NOT NULL
	username VARCHAR(100) NOT NULL
	password_hash VARCHAR(255) NOT NULL
	role VARCHAR(50) NOT NULL

[illegible]

UML– диаграмма классов



Приложение Б

(обязательное)

Клиент-серверная информационная система ведения клиентской базы для

среднего и малого бизнеса

Листинг программы

Файл «App.jsx»

```
import { useAuth } from './context/AuthContext';
import { Login } from './components/Login/Login';
import { Header } from './components/Header/Header';
import { Table } from './components/Table/Table';
import { UserManagement } from
'./components/UserManagement/UserManagement';
import { useState } from 'react';
import { SearchProvider } from './context/SearchContext';
import { addNewClient } from './api/hooks';
import { useMutation, useQueryClient } from '@tanstack/react-query';
import './App.css';

function App() {
  const { user } = useAuth();
  const [showUsers, setShowUsers] = useState(false);
  const [isModalOpen, setModalOpen] = useState(false);
  const queryClient = useQueryClient();

  // Состояния для полей формы нового клиента
  const [clientName, setClientName] = useState('');
  const [clientSurname, setClientSurname] = useState('');
  const [clientPatronymic, setClientPatronymic] = useState('');
  const [clientInterests, setClientInterests] = useState('');
  const [clientBalance, setClientBalance] = useState('');

  // Мутация добавления клиента
  const addNewClientMutation = useMutation({
    mutationFn: addNewClient,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['clients'] });
      closeModal();
    },
  });

  const closeModal = () => {
    setClientName('');
    setClientSurname('');
    setClientPatronymic('');
    setClientInterests('');
    setClientBalance('');
    setModalOpen(false);
  };

  const handleAddButton = () => {
    const fio = `${clientName} ${clientSurname} ${
clientPatronymic}`.trim();
    if (!fio) {
      alert('Введите хотя бы имя');
      return;
    }
    addNewClientMutation.mutate({
      fio,
```

```

        interests: clientInterests,
        contacts: {},
        balance: parseFloat(clientBalance) || 0,
    });
};

if (!user) return <Login />;

return (
    <SearchProvider>
        <Header />
        <main id="center">
            <h2>Клиенты</h2>
            {user.role === 'ADMIN' && (
                <button className='userSettingsBtn' onClick={() =>
setShowUsers(!showUsers)}>
                    {showUsers ?
                        <svg xmlns="http://www.w3.org/2000/svg" with="30"
height="30" viewBox="0 0 2048 2048"><title>Chrome Close SVG
Icon</title><path fill="currentColor" d="m1169 1024l879 879l-145 145l-879-
879l-879 879l0 1903l879-879l0 145l145 0l879 879l1903 0l145 145z"/></svg>
                        :
                        <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24
24"><title>Account Settings SVG Icon</title><path fill="currentColor"
d="M12 4a4 4 0 0 1 4 4a4 4 0 0 1-4 4a4 4 0 0 1-4-4a4 4 0 0 1 4-4m0 10c4.42
0 8 1.79 8 4v2H4v-2c0-2.21 3.58-4 8-4m-5 8h2v2H7zm4 0h2v2h-2zm4 0h2v2h-
2z"/></svg>
                    }
                </button>
            )}
            {showUsers ? <UserManagement /> : <Table />}
        </main>

        {/* Кнопка добавления клиента */}
        {!showUsers && (
            <button className='addNewClientBtn' onClick={() =>
setModalOpen(true)}>
                <svg width="22" height="16" viewBox="0 0 22 16" fill="none"
xmlns="http://www.w3.org/2000/svg">
                    <path d="M14 8C16.21 8 18 6.21 18 4C18 1.79 16.21 0 14 0C11.79
0 10 1.79 10 4C10 6.21 11.79 8 14 8ZM5 6V3H3V6H0V8H3V11H5V8H8V6H5ZM14
10C11.33 10 6 11.34 6 14V16H22V14C22 11.34 16.67 10 14 10Z"
fill="#9873FF"/>
                </svg>
            </button>
        )}

        {/* Модальное окно добавления клиента */}
        {isModalOpen && (
            <>
                <div className="modalOverlay" onClick={closeModal}></div>
                <div className="modalWindow">
                    <div className="modal_header">
                        <h2 className="modal-header__heading">Новый клиент</h2>
                        <button className="modal-header__closeBtn"
onClick={closeModal}>

```

```

        X
      </button>
    </div>
    <div className="modal_main">
      <p>Имя:</p>
      <input className="modal_input" type="text"
value={clientName} onChange={e => setClientName(e.target.value)} />
      <p>Фамилия:</p>
      <input className="modal_input" type="text"
value={clientSurname} onChange={e => setClientSurname(e.target.value)} />
      <p>Отчество:</p>
      <input className="modal_input" type="text"
value={clientPatronymic} onChange={e =>
setClientPatronymic(e.target.value)} />
      <p>Сфера интереса:</p>
      <input className="modal_input" type="text"
value={clientInterests} onChange={e => setClientInterests(e.target.value)}
/>
      <p>Баланс (кредит + / дебет -):</p>
      <input className="modal_input" type="number" step="any"
value={clientBalance} onChange={e => setClientBalance(e.target.value)} />
    </div>
    <div className="modal_footer">
      <button onClick={closeModal}>Отменить</button>
      <button onClick={handleAddButton}
disabled={addNewClientMutation.isPending}>
        {addNewClientMutation.isPending ? 'Добавление...' :
'Добавить'}
      </button>
    </div>
  </div>
</>
  )}
</SearchProvider>
);
}

export default App;

```

Файл «main.jsx»

```

import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
import { QueryClientProvider } from '@tanstack/react-query'
import { queryClient } from './api/queryClient'
import { AuthProvider } from './context/AuthContext.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <QueryClientProvider client={queryClient}>
      <AuthProvider>
        <App />
      </AuthProvider>
    </StrictMode>
  )

```

```

    </QueryClientProvider>
  </StrictMode>,
)

```

Файл «Header.jsx»

```

import { useAuth } from '../..context/AuthContext';
import { useSearch } from '../..context/SearchContext';
import './Header.css';

export const Header = () => {
  const { user, logout } = useAuth();
  const { searchTerm, setSearchTerm } = useSearch();

  return (
    <header className="header">
      <div className="logo">CRM Система</div>
      <input
        type="text"
        className="search-input"
        placeholder="Поиск по ФИО..."
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
      />
      <div className="user-panel">
        <span className="username">{user?.username}</span>
        <button onClick={logout} className="logout-btn">Выйти</button>
      </div>
    </header>
  );
};

```

Файл «Login.jsx»

```

import { useState } from 'react';
import { useAuth } from '../..context/AuthContext';
import './Login.css';

export const Login = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const { login } = useAuth();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    try {
      await login(username, password);
    } catch (err) {
      setError(err.response?.data?.error || 'Ошибка входа');
    }
  };

  return (
    <div className="login-container">
      <div className="login-card">

```

```

    <div className="login-header">
      <h1>CRM Система</h1>
      <p>Вход в аккаунт</p>
    </div>
    <form onSubmit={handleSubmit} className="login-form">
      {error && <div className="login-error">{error}</div>}
      <div className="input-group">
        <label>Логин</label>
        <input
          type="text"
          placeholder="Введите логин"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          required
        />
      </div>
      <div className="input-group">
        <label>Пароль</label>
        <input
          type="password"
          placeholder="Введите пароль"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          required
        />
      </div>
      <button type="submit" className="login-btn">Войти</button>
    </form>
  </div>
</div>
);
};

```

Файл «Table.jsx»

```

import { TableRow } from "../TableRow/TableRow";
import { getAllClients } from '../../api/hooks';
import { useQuery } from "@tanstack/react-query";
import { useSearch } from '../../context/SearchContext';
import "../Table.css";

export const Table = () => {
  const { searchTerm } = useSearch();

  const clientsQuery = useQuery({
    queryKey: ['clients'],
    queryFn: getAllClients,
  });

  const filteredClients = clientsQuery.data?.filter(client =>
    client.fio.toLowerCase().includes(searchTerm.toLowerCase())
  ) ?? [];

  switch (clientsQuery.status) {
    case "success":
      return (

```

```

        <table className="main">
          <thead>
            <tr>
              <th>ID</th>
              <th>ФИО</th>
              <th>Сфера интереса</th>
              <th>Кредит/Дебет</th>
              <th>Контакты</th>
              <th>Действия</th>
            </tr>
          </thead>
          <tbody>
            {filteredClients.map(el => (
              <TableRow
                key={el.id}
                id={el.id}
                fio={el.fio}
                interest={el.interests}
                contacts={el.contacts}
                balance={el.balance}
              />
            ))}
          </tbody>
        </table>
      );
      case "pending":
        return <span>Загрузка ...</span>;
      case "error":
        return (
          <>
            <span>Произошла ошибка :(</span>
            <button onClick={() =>
clientsQuery.refetch()}>Повторить запрос</button>
          </>
        );
    }
  };
};

```

Файл «TableRow.jsx»

```

import Tippy from '@tippyjs/react';
import { useState, useEffect } from 'react';
import { createPortal } from 'react-dom';
import 'tippy.js/dist/tippy.css';
import { updateClient, deleteClient } from "../../api/hooks";
import { useMutation, useQueryClient } from "@tanstack/react-query";
import { queryClient } from '../../api/queryClient';
import './TableRow.css';

export const TableRow = ({ id, fio: initialFio, interest, contacts,
balance }) => {
  const [isOpenDelete, setOpenDeleteState] = useState(false);
  const [isOpenEdit, setOpenEditState] = useState(false);

  // Черновики для редактирования
  const [editFio, setEditFio] = useState(initialFio);
  const [editInterest, setEditInterest] = useState(interest);

```

```

const [editPhone, setEditPhone] = useState(contacts?.phone || '');
const [editTg, setEditTg] = useState(contacts?.tg || '');
const [editBalance, setEditBalance] = useState(balance);

// При открытии модальки копируем текущие данные
const handleOpenEdit = () => {
  setEditFio(initialFio);
  setEditInterest(interest);
  setEditPhone(contacts?.phone || '');
  setEditTg(contacts?.tg || '');
  setEditBalance(balance);
  setOpenEditState(true);
};

// Мутация удаления
const deleteClientMutation = useMutation({
  mutationFn: (id) => deleteClient(id),
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ["clients"] });
    setOpenDeleteState(false);
  }
});

// Мутация обновления
const updateClientMutation = useMutation({
  mutationFn: ({ id, updates }) => updateClient(id, updates),
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ["clients"] });
    setOpenEditState(false);
  }
});

// Сбор изменений и вызов мутации
const handleSaveEdit = () => {
  const updates = {};
  if (editFio !== initialFio) updates.fio = editFio;
  if (editInterest !== interest) updates.interests = editInterest;
  if (editPhone !== (contacts?.phone || '') || editTg !==
(contacts?.tg || '')) {
    updates.contacts = { phone: editPhone, tg: editTg };
  }
  if (editBalance !== balance) updates.balance =
parseFloat(editBalance) || 0;
  if (Object.keys(updates).length === 0) {
    setOpenEditState(false);
    return;
  }
  updateClientMutation.mutate({ id, updates });
};

// Блокировка скролла
useEffect(() => {
  if (isOpenDelete || isOpenEdit) {
    document.body.classList.add('active');
    window.scrollTo(0, 0);
  } else {

```

```

        document.body.classList.remove('active');
    }
    return () => document.body.classList.remove('active');
}, [isOpenDelete, isOpenEdit]));

const rowClass = balance > 0 ? 'positive-row' : (balance < 0 ?
'negative-row' : '');

return (
    <>
        <tr className={rowClass}>
            <td className='row__id'>{id}</td>
            <td className='row__fio'>{initialFio}</td>
            <td className='row__interest'>{interest}</td>
            <td className='row__balance'>{balance}</td>
            <td className='row__contacts'>
                {contacts.phone && (
                    <Tippy content={contacts.phone}>
                        <button className='contacts__btn'>
                            <svg className='contacts__icon'
xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24
24">
                                <path d="M20 22.6211-3.521-
6.795c-.008.004-1.974.97-2.064 1.011-2.24 1.086-6.799-7.82-4.609-
8.99412.083-1.026-3.493-6.817-2.106 1.039c-7.202 3.755 4.233 25.982 11.6
22.615.121-.055 2.102-1.029 2.11-1.033z"/>
                                </svg>
                            </button>
                        </Tippy>
                    )}
                {contacts.tg && (
                    <Tippy content={contacts.tg}>
                        <button className='contacts__btn'>
                            <svg className='contacts__icon'
version="1.1" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
                                <path d="M12,0c-6.627,0 -12,5.373
-12,12c0,6.627 5.373,12 12,12c6.627,0 12,-5.373 12,-12c0,-6.627 -5.373,-12
-12,-12Zm0,2c5.514,0 10,4.486 10,10c0,5.514 -4.486,10 -10,10c-5.514,0
-10,-4.486 -10,-10c0,-5.514 4.486,-10 10,-10Zm2.692,14.889c0.161,0.115
0.368,0.143 0.553,0.073c0.185,-0.07 0.322,-0.228 0.362,-0.42c0.435,-2.042
1.489,-7.211 1.884,-9.068c0.03,-0.14 -0.019,-0.285 -0.129,-0.379c-0.11,-
0.093 -0.263,-0.12 -0.399,-0.07c-2.096,0.776 -8.553,3.198 -11.192,4.175c-
0.168,0.062 -0.277,0.223 -0.271,0.4c0.006,0.177 0.125,0.33
0.296,0.381c1.184,0.354 2.738,0.847 2.738,0.847c0,0 0.725,2.193
1.104,3.308c0.047,0.139 0.157,0.25 0.301,0.287c0.145,0.038 0.298,-0.001
0.406,-0.103c0.608,-0.574 1.548,-1.461 1.548,-1.461c0,0 1.786,1.309
2.799,2.03Zm-5.505,-4.33810.84,2.76910.186,-1.754c0,0 3.243,-2.925 5.092,-
4.593c0.055,-0.048 0.062,-0.13 0.017,-0.188c-0.045,-0.057 -0.126,-0.071
-0.188,-0.032c-2.143,1.368 -5.947,3.798 -5.947,3.798Z"/>
                                </svg>
                            </button>
                        </Tippy>
                    )}
                </td>
            <td className='row__actions'>

```

```

        <button onClick={handleOpenEdit}
className='actions__btn'>
            <svg width="16" height="16" viewBox="0 0 16 16"
fill="none" xmlns="http://www.w3.org/2000/svg">
                <path d="M2 11.5002V14.0002H4.5L11.8733
6.62687L9.37333 4.12687L2 11.5002ZM13.8067 4.69354C14.0667 4.43354 14.0667
4.01354 13.8067 3.75354L12.2467 2.19354C11.9867 1.93354 11.5667 1.93354
11.3067 2.19354L10.0867 3.41354L12.5867 5.91354L13.8067 4.69354Z"
fill="#9873FF"/>
            </svg>
            <span>Изменить</span>
        </button>
        <button onClick={() => setOpenDeleteState(true)}
className='actions__btn'>
            <svg width="16" height="16" viewBox="0 0 16 16"
fill="none" xmlns="http://www.w3.org/2000/svg">
                <path d="M8 2C4.682 2 2 4.682 2 8C2 11.318
4.682 14 8 14C11.318 14 14 11.318 14 8C14 4.682 11.318 2 8 2ZM8 12.8C5.354
12.8 3.2 10.646 3.2 8C3.2 5.354 5.354 3.2 8 3.2C10.646 3.2 12.8 5.354 12.8
8C12.8 10.646 10.646 12.8 8 12.8ZM10.154 5L8 7.154L5.846 5L5 5.846L7.154
8L5 10.154L5.846 11L8 8.846L10.154 11L11 10.154L8.846 8L11 5.846L10.154
5Z" fill="#F06A4D"/>
            </svg>
            <span>Удалить</span>
        </button>
    </td>
</tr>

    { /* Модалка удаления */ }
    { isOpenDelete && createPortal(
        <div className='modalOverlay'>
            <div className='modalWindow'>
                <div className='modal_header'>
                    <h2>Вы действительно хотите удалить этого
клиента?</h2>
                    <button className='modal-header__closeBtn'
onClick={() => setOpenDeleteState(false)}>X</button>
                </div>
                <div className='modal_footer'>
                    <button onClick={() =>
setOpenDeleteState(false)}>Отменить</button>
                    <button onClick={() =>
deleteClientMutation.mutate(id)}>Удалить</button>
                </div>
            </div>
            <div className="blackout" onClick={() =>
setOpenDeleteState(false)}></div>
        </div>,
        document.body
    ) }

    { /* Модалка редактирования */ }
    { isOpenEdit && createPortal(
        <div className='modalOverlay'>
            <div className='modalWindow'>
                <div className='modal_header'>

```

```

        <h2>Изменение клиента</h2>
        <button className='modal-header__closeBtn'
onClick={() => setOpenEditState(false)}>X</button>
    </div>
    <div className='modal_main'>
        <p>ФИО:</p>
        <input type="text" className="modal_input"
value={editFio} onChange={(e) => setEditFio(e.target.value)} />

        <p>Сфера интереса:</p>
        <input type="text" className="modal_input"
value={editInterest} onChange={(e) => setEditInterest(e.target.value)} />

        <p>Телефон:</p>
        <input type="tel" className="modal_input"
value={editPhone} onChange={(e) => setEditPhone(e.target.value)} />

        <p>Telegram:</p>
        <input type="text" className="modal_input"
value={editTg} onChange={(e) => setEditTg(e.target.value)} />

        <p>Баланс (кредит + / дебет -):</p>
        <input type="number" step="any"
className="modal_input" value={editBalance} onChange={(e) =>
setEditBalance(e.target.value)} />
    </div>
    <div className='modal_footer'>
        <button onClick={() =>
setOpenEditState(false)}>Отменить</button>
        <button onClick={handleSaveEdit}
disabled={updateClientMutation.isPending}>
            {updateClientMutation.isPending ?
'Sохранение...' : 'Сохранить'}
        </button>
    </div>
</div>
<div className="blackout" onClick={() =>
setOpenEditState(false)}></div>
</div>,
document.body
    )}
</>
);
};

```

Файл «UserManagement.jsx»

```

import { useState, useEffect } from 'react';
import api from '../api/axiosConfig';
import './UserManagement.css';

export const UserManagement = () => {
    const [users, setUsers] = useState([]);
    const [newUsername, setNewUsername] = useState('');
    const [newPassword, setNewPassword] = useState('');
    const [newRole, setNewRole] = useState('USER');

```

```

const fetchUsers = async () => {
  try {
    const res = await api.get('/admin/users');
    setUsers(res.data);
  } catch (err) {
    console.error('Ошибка загрузки пользователей', err);
  }
};

useEffect(() => {
  fetchUsers();
}, []);

const addUser = async () => {
  if (!newUsername.trim() || !newPassword.trim()) {
    alert('Введите логин и пароль');
    return;
  }
  try {
    await api.post('/admin/users', {
      username: newUsername,
      password: newPassword,
      role: newRole,
    });
    fetchUsers();
    setNewUsername('');
    setNewPassword('');
  } catch (err) {
    alert('Ошибка создания: ' + (err.response?.data?.error || err.message));
  }
};

const deleteUser = async (id) => {
  if (!window.confirm('Удалить пользователя?')) return;
  try {
    await api.delete(`/admin/users/${id}`);
    fetchUsers();
  } catch (err) {
    alert('Ошибка удаления: ' + (err.response?.data?.error || err.message));
  }
};

return (
  <div className="user-management">
    <h3 className="um-title">👤 Управление пользователями</h3>

    <div className="um-users-list">
      {users.map((u) => (
        <div key={u.id} className="um-user-card">
          <div className="um-user-info">
            <span className="um-username">{u.username}</span>
            <span className={`um-role ${u.role === 'ADMIN' ? 'role-admin' : 'role-user'}`}>

```

```

        {u.role === 'ADMIN' ? 'Администратор' : 'Пользователь'}
      </span>
    </div>
    <button className="um-delete-btn" onClick={() =>
deleteUser(u.id)}>&img alt="trash icon" data-bbox="321 118 341 133"/> Удалить</button>
    </div>
  )))
  {users.length === 0 && <div className="um-empty">Пользователей
пока нет</div>}
</div>

<div className="um-add-form">
  <h4>+ Создать нового пользователя</h4>
  <div className="um-form-fields">
    <input
      type="text"
      placeholder="Логин"
      value={newUsername}
      onChange={(e) => setNewUsername(e.target.value)}
      className="um-input"
    />
    <input
      type="password"
      placeholder="Пароль"
      value={newPassword}
      onChange={(e) => setNewPassword(e.target.value)}
      className="um-input"
    />
    <select value={newRole} onChange={(e) =>
setNewRole(e.target.value)} className="um-select">
      <option value="USER">Пользователь</option>
      <option value="ADMIN">Администратор</option>
    </select>
    <button onClick={addUser} className="um-create-
btn">Создать</button>
  </div>
</div>
</div>
);
};

```

Файл «AuthContext.jsx»

```

import { createContext, useContext, useState, useEffect } from 'react';
import axios from 'axios';

const API = 'http://localhost:5000';
const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [token, setToken] = useState(localStorage.getItem('token'));

  useEffect(() => {
    if (token) {
      try {

```

[illegible]

Файл « SearchContext.jsЭЖ,х»

```
const SearchContext = createContext();
```

```
export const useSearch = () => {
  const context = useContext(SearchContext);
  if (!context) {
    throw new Error('useSearch must be used within SearchProvider');
  }
  return context;
};
```

Приложение В

(обязательное)

Клиент-серверная информационная система ведения клиентской базы для
среднего и малого бизнеса

Презентация

Клиент-серверная информационная система для ведения клиентской базы малого и среднего бизнеса

Автор работы:

студент гр. 21ВВП1 Ипполитов Иван Дмитриевич

Научный руководитель:

к.т.н., доцент кафедры ВТ ПГУ Юрова Ольга Викторовна

Цели и задачи работы

Цель работы

Разработать клиент-серверную информационную систему для ведения клиентской базы малого и среднего бизнеса с разграничением прав доступа и финансовым учётом.

Задачи

01

Изучить актуальность темы и существующие аналоги CRM-систем

02

Определить функциональные и нефункциональные требования

03

Выбрать инструменты и технологии для разработки

04

Спроектировать архитектуру и базу данных

05

Реализовать и протестировать веб-приложение

Актуальность проблемы

Проблемы малого и среднего бизнеса

Отсутствие структурированного хранения клиентских данных

Использование бумажных носителей, Excel-таблиц и заметок

Потеря данных при смене менеджеров, дублирование записей

Невозможность отследить историю взаимодействий

Почему готовые CRM не подходят

amoCRM, Bitrix24, Megaplan — избыточно сложны и дороги для малого бизнеса

Excel / Google Sheets — не обеспечивают многопользовательский доступ и безопасность

Обзор аналогов

Решение	Преимущества	Недостатки
amoCRM	Простой интерфейс, воронка продаж	Ограниченная аналитика
Bitrix24	Мощная экосистема, телефония	Перегружен, дорог <u>для малого</u> бизнеса
Megaplan	Управление проектами	Дорогие лицензии
Excel / Google Sheets	Бесплатно, просто	Нет многопользовательского доступа, защиты, истории

Функционал разработанной системы

CRM Система

Поиск по ФИО...

test Вылн

ID	ФИО	СФЕРА ИНТЕРЕСА	КРЕДИТ/ДЕБЕТ	КОНТАКТЫ	ДЕЙСТВИЯ
1	Сергей Сергеевич Сергеев	Автомобили	-10000		Изменить Удалить
6	Себастьян Константин Алексеевич	Оплата электричества	5000		Изменить Удалить
7	Михайлов Михаил Степанович	Проводник	5000		Изменить Удалить
8	Павел Александр Данилович	Машинист локомотива	-100		Изменить Удалить
9	Сергеев Александр Данилович	Иппедант	100		Изменить Удалить
11	Тарасов Игорь Максимович	Проводник	0		Изменить Удалить
12	Арипова Мария Мирославовна	Торговар	0		Изменить Удалить
13	Дмитриев Ва Георгиевна	Корректировщик	0		Изменить Удалить
14	Сефонов Александр Аргенович	Физикотерапевт	0		Изменить Удалить
15	Себастьян Константин Алексеевич	Модель	0		Изменить Удалить
16	Михайлов Михаил Степанович	Механик	0		Изменить Удалить

1 Ведение клиентской базы

CRUD-операции: добавление, просмотр, редактирование, удаление клиентов

2 Поиск в реальном времени

Фильтрация по ФИО без перезагрузки страницы

3 Аутентификация и роли

JWT, хеширование паролей, разграничение: администратор / пользователь

4 Финансовый учёт

Баланс клиента (кредит/дебет), цветовая индикация: зелёный — предоплата, красный — задолженность

Используемые технологии

	Клиентская часть React - компонентный подход, виртуальный DOM React Context - глобальное состояние React Query - кеширование и
	Серверная часть Node.js + Express - REST API JWT + bcrypt - безопасная аутентификация
	База данных PostgreSQL - надёжность, JSON Prisma ORM - <u>типобезопасность</u> , миграции



React
Клиентский интерфейс



Express
REST API и маршрутизация



Prisma ORM
Типобезопасные запросы



bcrypt
Хеширование паролей



Node.js
Среда выполнения сервера

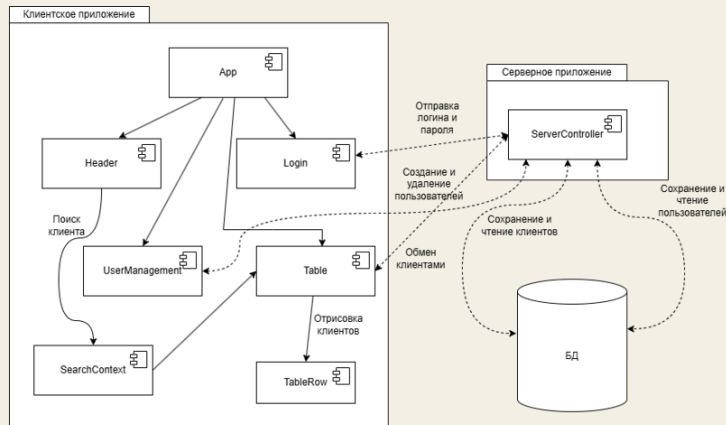


PostgreSQL
Надёжная база данных



JWT
Аутентификация и токены

Архитектура приложения



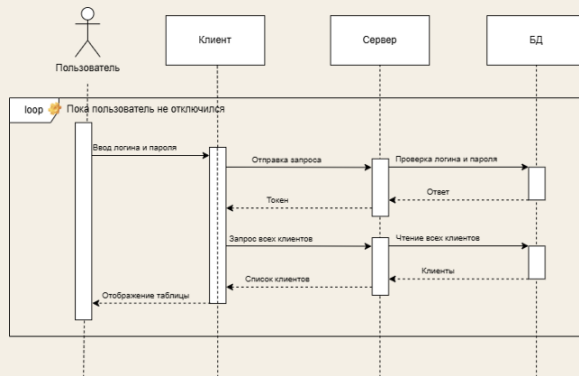
Клиент (React)

- **View:** компоненты Login, Header, Table, TableRow, UserManagement
- **ViewModel:** AuthContext, SearchContext, React Query
- **Model:** API-вызовы через axios с перехватчиком токена

Сервер и БД

- Маршруты: /clients, /auth, /admin/users
- Middleware: authMiddleware, adminMiddleware
- Таблица clients: id, fio, interests, contacts (JSON), balance
- Таблица users: id, username, passwordHash, role (ADMIN/USER)

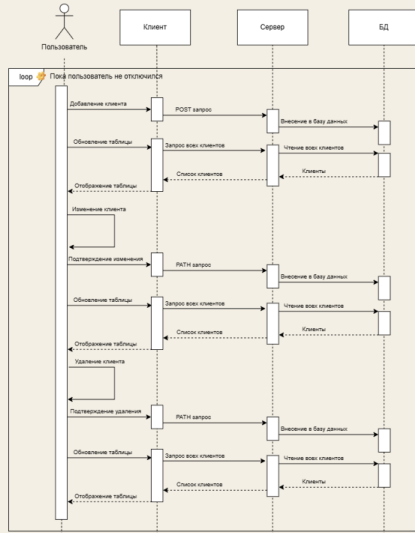
Диаграмма последовательности



Сценарий входа в систему

Пользователь вводит логин и пароль, система отправляет запрос на вход, проверяет данные и возвращает JWT-токен для перехода на главную страницу.

Диаграмма последовательности



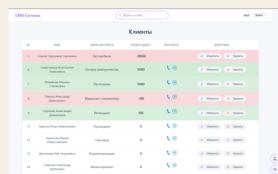
Загрузка списка клиентов

Компонент таблицы запрашивает список клиентов через API, сервер возвращает данные из базы, а интерфейс отображает их и кеширует результат.

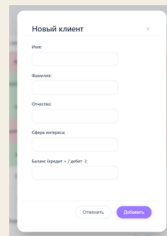
Интерфейс приложения



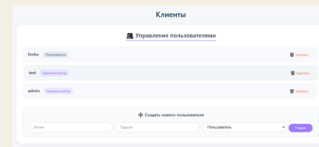
Страница входа
Центрированная
карточка с полями
«Логин» и «Пароль»



Главная страница
Шапка с поиском,
таблица клиентов (ID,
ФИО, интересы, баланс,
контакты, действия),
плавающая кнопка
добавления



Модальные окна
Добавление и
изменение клиента,
подтверждение
удаления



Панель
администратора
Список пользователей
в виде карточек, форма
создания нового
пользователя с
назначением роли

Заключение

Результаты работы

- Разработано клиент-серверное веб-приложение для ведения клиентской базы
- Реализованы CRUD-операции, аутентификация, ролевая модель, финансовый учёт
- Система протестирована, работает стабильно, ошибок не обнаружено

Ключевые навыки

- Современная веб-разработка на JavaScript (React, Node.js, Express)
- Работа с PostgreSQL через Prisma ORM
- Безопасная аутентификация (JWT, bcrypt)
- Проектирование ролевой модели и REST API

Перспективы развития

- История взаимодействий с клиентами (звонки, встречи)
- Экспорт и импорт данных (CSV, XLSX)
- Система отчётов и аналитики
- Автоматические напоминания и задачи