

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

<i>Факультет</i>	<i>Кафедра</i>
вычислительной техники	МОиПЭВМ
Направление подготовки	09.03.04 «Программная инженерия»
Направленность	Программная инженерия

БАКАЛАВРСКАЯ РАБОТА
на тему
Автоматизированное рабочее место
администратора отдела информирования
клиентов организации

Студент	_____	Исаева Инесса Романовна
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)

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

Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)

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

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2025

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов «20»
декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Исаева Инесса Романовна гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы: Автоматизированное рабочее место администратора отдела информирования клиентов организации

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

3. Руководитель работы к.т.н., доцент, доцент кафедры «МО и ПЭВМ» Такташкин Д.В.

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

Назначение: разработка приложения «Автоматизированное рабочее место администратора отдела информирования клиентов организации»

Основные функции системы:

- 1) вход в систему;
- 2) регистрация в системе;
- 3) создание и редактирование проекта;
- 4) управление учетными записями операторов;
- 5) предоставление доступа к информации о записанном звонке;
- 6) редактирование данных о записанном звонке;
- 7) сбор и предоставление статистики о работе операторов.

Технология разработки: REST-API.

Язык программирования: C#.

Среда разработки: JetBrains Rider.

5. Перечень подлежащих разработке вопросов:

- 1) анализ предметной области;
- 2) проектирование программных средств;
- 3) реализация программных средств;
- 4) тестирование программных средств;
- 5) оформление ВКР.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 105 листов, 50 рисунков, 10 таблиц, 20 использованных источников и 2 приложения.

ПРОГРАММИРОВАНИЕ, ИНФОРМАЦИОННЫЕ СИСТЕМЫ, ПРОЕКТИРОВАНИЕ, UML, WEB-ПРИЛОЖЕНИЕ, БАЗЫ ДАННЫХ, BLAZOR, POSTGRESQL, REST API, C#, HTTP.

Предмет разработки: разработка веб-приложение для автоматизации рабочего места администратора отдела информирования клиентов организации.

Объект разработки: автоматизированное рабочее место администратора отдела информирования клиентов организации.

Цель разработки: изучить предметную область и провести её анализ, выделить основные функциональные требования, произвести проектирование, реализовать и протестировать веб-приложение для автоматизации рабочего места администратора отдела информирования клиентов организации.

Средства разработки: СУБД PostgreSQL 16, язык программирования C#, среда разработки JetBrains Rider 2024.1.6.

Область применения: приложение может быть использовано для организации рабочей среды сотрудников колл-центров.

					ПГУ 09.03.04 – 07КП211.11 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Автоматизированное рабочее место администратора отдела информирования клиентов организации	Лит.	Лист	Листов
Разраб.		Исаева И.Р.					4	105
Провер.		Такташкин Д.В.				ГРУППА 21ВП1		
Реценз.								
Н. Контр.		Такташкин Д.В.						
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Анализ предметной области	9
1.1 Анализ существующих систем аналогичного назначения.....	10
1.2 Анализ требований.....	15
1.2.1 Анализ функциональных требований	15
1.2.2 Анализ нефункциональных требований	17
1.3 Постановка задачи.....	19
2 Проектирование web-приложения.....	21
2.1 Выбор технологий разработки.....	21
2.2 Проектирование архитектуры приложения.....	22
2.2.1 Проектирование базы данных	23
2.2.2 Проектирование сервисов.....	25
2.3 Проектирование графического интерфейса	29
2.4 Оценка бюджета разработки web-приложения.....	35
3 Разработка и реализация приложения.....	39
3.1 Реализация базы данных приложения.....	39
3.2 Реализация сервисов	43
3.3 Реализация клиентского приложения и интерфейс пользователя	47
4 Функциональное тестирование приложения.....	63
4.1 Ручное тестирование	63
4.2 Автоматизированное тестирование.....	66
4.3 Анализ сложности кода модуля регистрации	71
Заключение	76

Список использованных источников	77
Приложение А (обязательное) Исходный код приложения	80
Приложение Б (рекомендуемое) Графическая часть.....	95
Приложение В (рекомендуемое) Результаты ручного тестирования	101

Введение

Современные бизнес-процессы требуют высокой степени автоматизации и эффективного управления взаимодействием с клиентами. Одной из ключевых составляющих успешной работы компании является эффективное продвижение своих товаров и услуг среди целевой аудитории, а также своевременное информирование постоянных клиентов о новейших предложениях. Многие современные компании прибегают к использованию услуг колл-центров для профессионального продвижения своих продуктов.

В условиях постоянного роста объема данных, с которыми работают организации, автоматизация рабочих процессов становится неотъемлемой частью эффективного управления. Для колл-центров, отделов продаж и информирования клиентов, где от качества обслуживания напрямую зависит репутация компании и ее конкурентоспособность, это становится необходимостью. Таким образом, разработка системы для автоматизации рабочего места администраторов колл-центров представляет собой важную задачу, способствующую повышению эффективности и производительности труда.

Актуальность данной темы обусловлена следующими факторами:

- во-первых, наличие удобных программных средств может как существенно улучшить клиентский опыт общения с оператором, так и снизить уровень стресса, с которым сталкиваются сотрудники;
- во-вторых, автоматизация рутинных процессов позволяет снизить вероятность ошибок и увеличить скорость обзвона клиентской базы, что положительно скажется на удовлетворенности руководства работой отдела информирования клиентов, или же компании-заказчика эффективностью услуг колл-центра.

Целью данной работы является разработка веб-приложения, которое будет служить инструментом для автоматизации рабочего места сотрудников колл-центра или внутреннего отдела информирования клиентов компании. В

результате реализации проекта планируется создать удобный и функциональный инструмент, который будет способствовать улучшению качества и скорости обзвона клиентской базы.

Для достижения поставленных целей требуется выполнить следующие задачи:

- провести анализ предметной области для наилучшего понимания специфики разрабатываемого программного обеспечения и потребностей потенциальных пользователей;
- провести анализ уже существующих приложений и систем аналогичного назначения, чтобы понимать, на какие функциональные особенности стоит ориентироваться и каких ошибок стоит избегать в собственном проекте;
- определить требования к разрабатываемой системе, которые станут ориентиром при проектировании системы;
- разработать UML диаграмму вариантов использования;
- выбрать технологию и средства разработки;
- выполнить проектирование архитектуры приложения;
- разработать диаграмму компонентов системы и диаграммы классов;
- выполнить проектирование графического интерфейса пользователя;
- выполнить оценку бюджета на разработку системы;
- составить график работ с помощью диаграммы Ганта;
- реализовать спроектированную систему;
- выполнить тестирование и убедиться в корректности работы программы.

В результате решения всех вышеперечисленных задач должен быть выполнен анализ проведенной работы и сформулирован вывод о достижении поставленных целей.

1 Анализ предметной области

В данном разделе будет представлен анализ предметной области. Так как проектируемая система может быть применена и как программное обеспечение для организации, специализирующейся на предоставлении услуг продвижения другим компаниям, так и в рамках отдела продвижения услуг конкретной фирмы, то, во избежание противоречий, в тексте данной работы термины «отдел информирования клиентов» и «колл-центр» будут использованы как контекстные синонимы.

Эффективный колл-центр, как правило, преследует в своей деятельности следующие цели:

- информирование клиентов компании о новых продуктах, изменениях в политике предоставляемых услуг;
- наращивание клиентской базы путем холодного обзвона;
- прием входящих вызовов и консультация клиентов по вопросам, связанным с предоставляемыми продуктами компании заказчика услуг колл-центра;
- поддержание имиджа компании и обеспечение связи между клиентом и фирмой;
- предметная область включает в себя следующие ключевые компоненты;
- операторы колл-центра – сотрудники, осуществляющие обзвон клиентов по базе мобильных номеров;
- клиенты – пользователи услуг компании, которых необходимо оповестить об актуальных предложениях;
- администраторы – сотрудники, ответственные за управление работой операторов;
- звонки – информационные объекты системы, хранящие информацию о результате взаимодействия с оператором с клиентом;
- рейтинги – позволяют администраторам отслеживать эффективность

работы операторов колл-центра;

- проекты – основные сущности системы, включающие базу клиентов, операторов, работающих на данном проекте (например, «Консультация по кредитным картам», «Информирование о тарифах оператора связи» и т. д.), скрипт диалога с клиентом, звонки, принадлежащие проекту и другие сущности системы.

1.1 Анализ существующих систем аналогичного назначения

При разработке собственной системы важной частью подготовки к проектированию является анализ уже существующих на рынке продуктов по смежной тематике. Это позволяет разработчику не совершить ошибок, которые уже были совершены авторами аналогов, а также выявить функции, которые можно, помимо прочего, привнести в реализацию собственной системы.

Кроме того, изучение популярных приложений помогает лучше понять потребности пользователей, чтобы в будущем ориентироваться на уже привычный для целевой аудитории интерфейс, дополнив его своими идеями и улучшениями.

В данном подразделе будет приведен обзор и анализ некоторых уже существующих на рынке программных средств, призванных автоматизировать работу колл-центра – «Скорозвон», «Mango Office» и «amoCRM».

Система «Скорозвон» имеет следующие функциональные преимущества:

- клиентская база удобно загружается из документа Excel (рисунок 1, 2);
- повторный звонок одному и тому же клиенту помечается системой, что улучшает опыт взаимодействия клиента с сотрудником и помогает оператору ориентироваться в ходе диалога;
- структурированно и интуитивно понятно расположены разделы в интерфейсе приложения;
- удобная выгрузка отчетов о работе операторов в Excel.

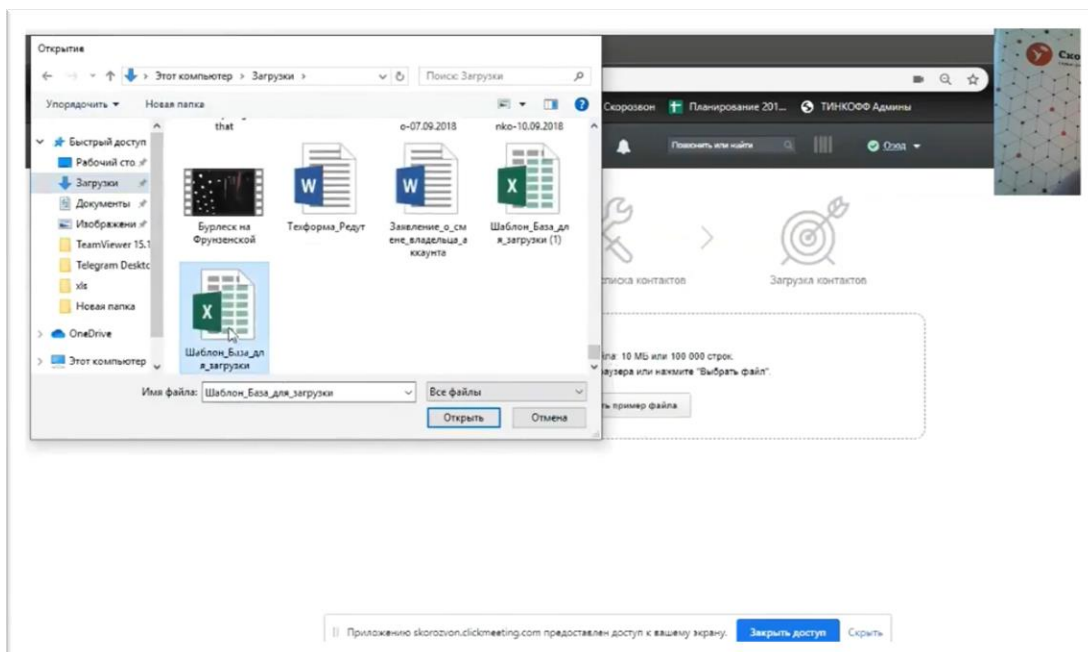


Рисунок 1 – Загрузка клиентской базы в приложении «Скорозвон»
(выбор файла для загрузки)

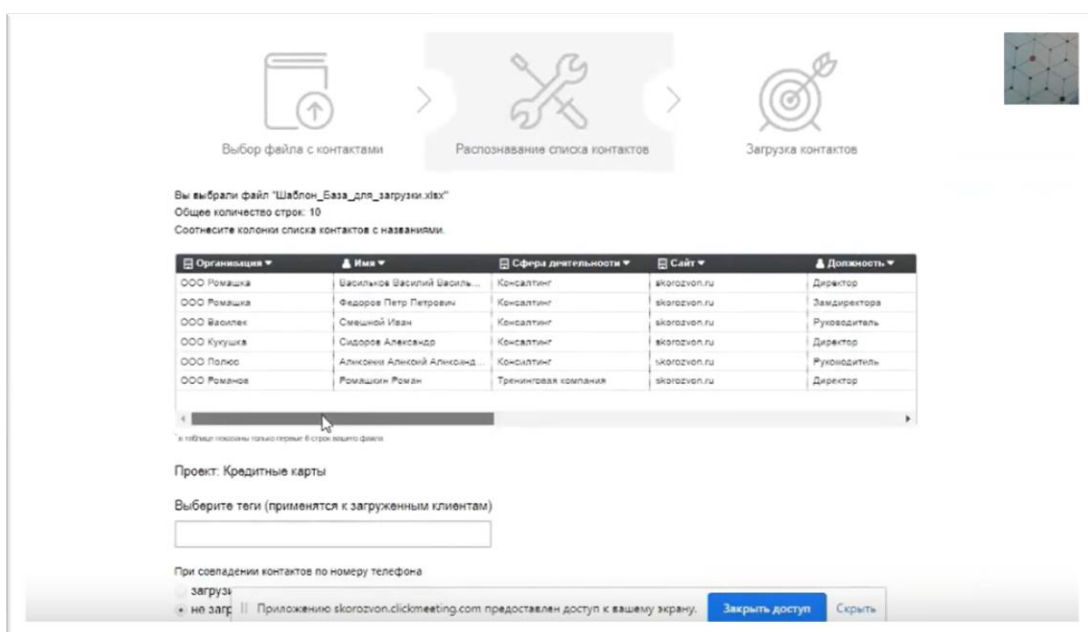


Рисунок 2 – Загрузка клиентской базы в приложении «Скорозвон»
(результат загрузки)

Недостатки системы «Скорозвон»:

- при переключении разделов интерфейса в процессе диалога с клиентом вкладка скрипта диалога закрывается, что может вызвать трудности в работе новых сотрудников с малым опытом;

- отсутствует явная сортировка совершенных звонков по конкретному сотруднику, что может усложнить отслеживание работы отдельного оператора;
- система предоставляется только на коммерческой основе.

Пример интерфейса во время совершения звонка представлен на рисунке 3.

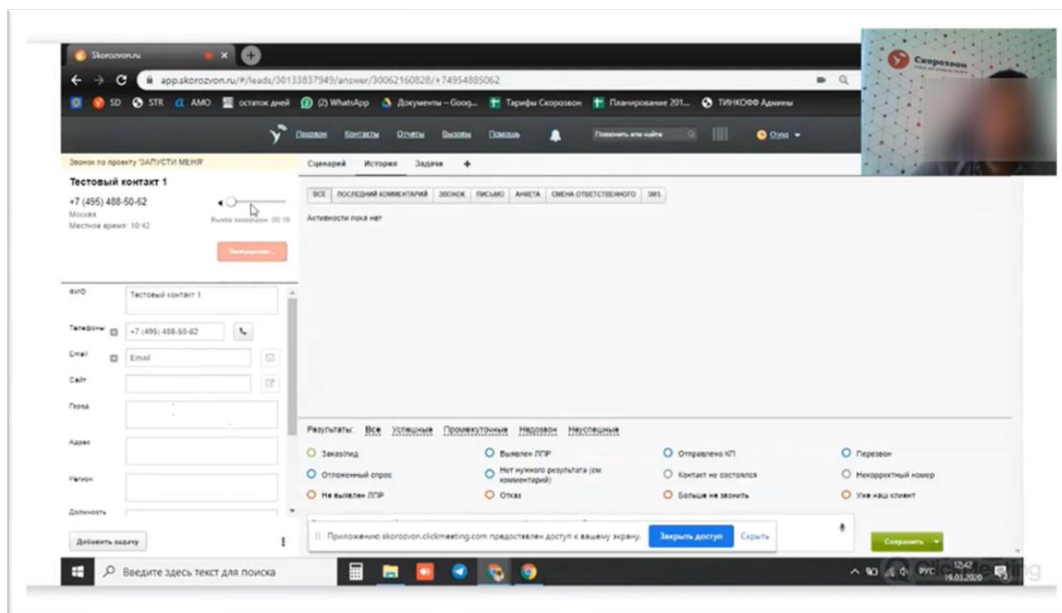


Рисунок 3 – Интерфейс приложения «Скорозвон» во время совершения звонка

На основе перечисленных пунктов можно сделать вывод, что для разрабатываемого проекта будет полезно реализовать функцию сортировки звонков по конкретному сотруднику, а также функцию загрузки клиентской базы из Excel-таблицы. Кроме того, следует обратить внимание на удобство интерфейса – скрипт диалога по возможности должен быть перед глазами оператора на протяжении всего сеанса звонка.

Система «Mango Office» имеет следующие функциональные преимущества:

- удобный пользовательский интерфейс для прослушивания и оценки совершенного звонка (рисунок 4);
- есть настройка интерактивного скрипта разговора (рисунок 5).

Недостатки системы «Mango Office»:

- интерактивный скрипт формализирует диалог с клиентом, что может

негативно сказаться на опыте сотрудника, так как в реальных условиях ход диалога может быть непредсказуем;

- система предоставляется только на коммерческой основе.

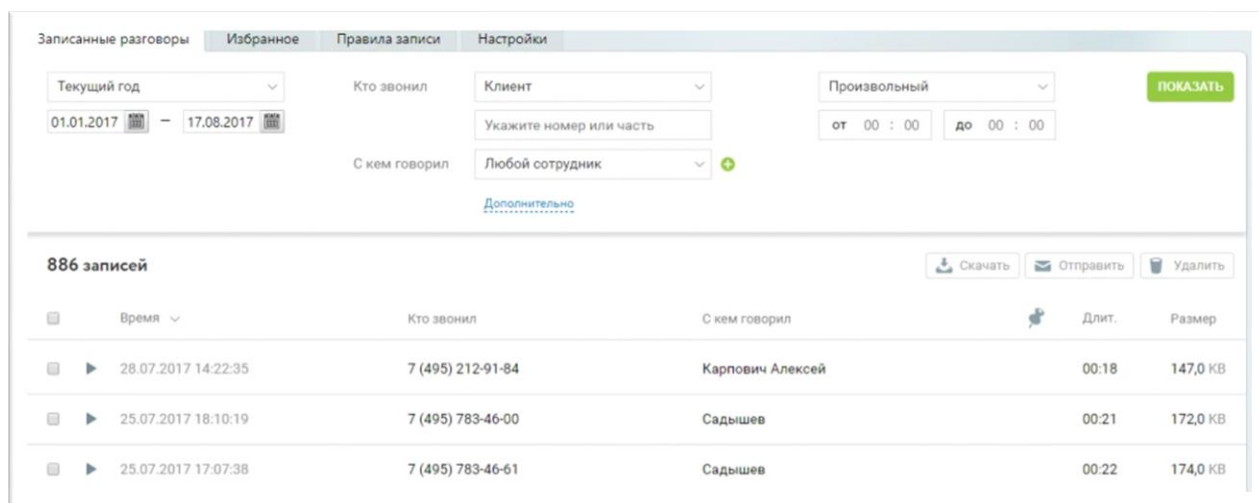


Рисунок 4 – Прослушивание звонка в приложении «Mango Office»

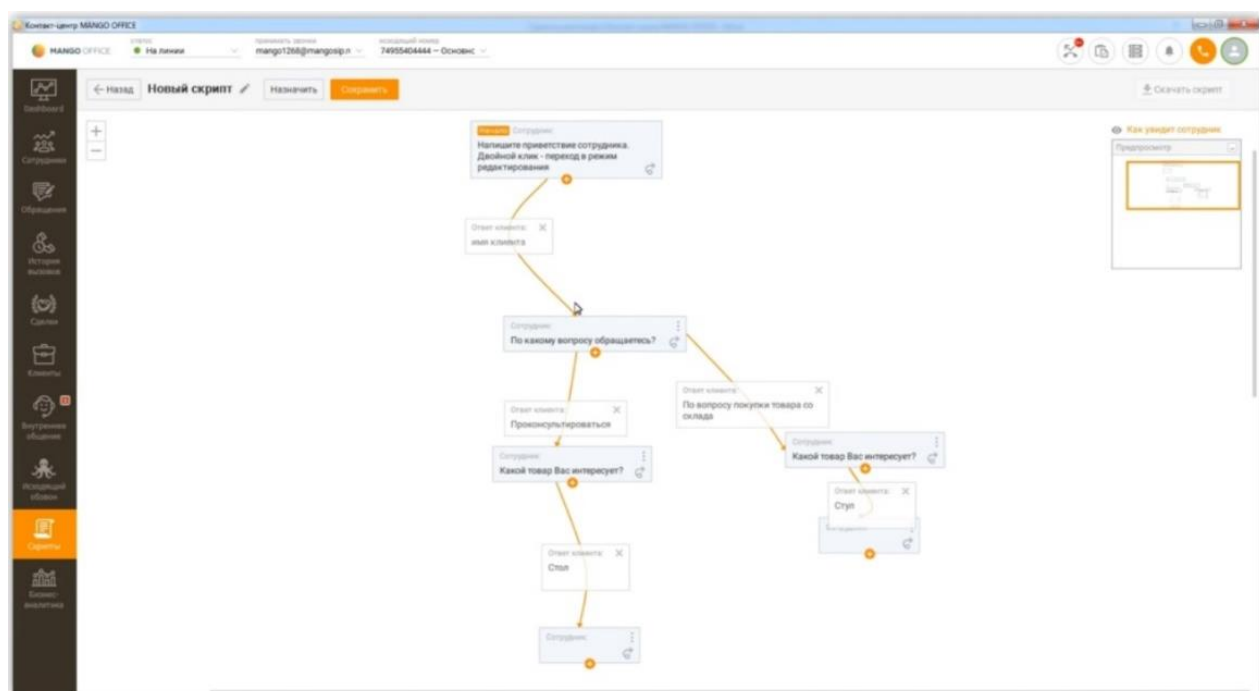


Рисунок 5 – Конструктор интерактивного скрипта «Mango Office»

Для собственного проекта из приложения «Mango Office» можно взять за образец удобный интерфейс для прослушивания звонка администратором. Интерактивный скрипт, несмотря на внешнюю инновационность, в реальных условиях не будет удобен оператору. Такой вывод сделан на основе личного опыта работы оператором в приложении с интерактивным скриптом.

Система «amoCRM» имеет следующие функциональные преимущества:

- интуитивно понятный интерфейс;
- эстетичный вид графических отчетов об эффективности операторов (рисунок 6).

Недостатки «amoCRM»:

- из-за широконаправленности сервиса усложняется настройка холодного обзвона;
- нет возможности явно настроить перечень статусов завершеного звонка.

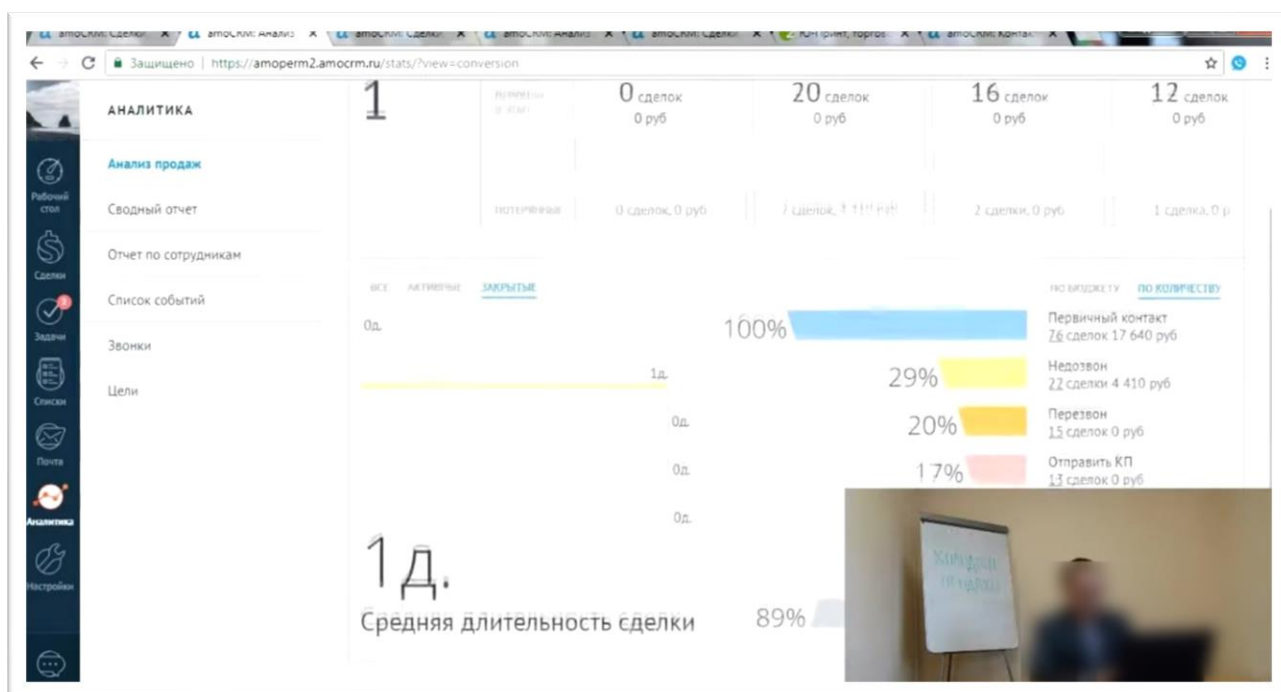


Рисунок 6 – Вид графического отчета в «amoCRM»

Для реализации собственного приложения из системы amoCRM можно взять за образец наглядность предоставляемых графических отчетов об эффективности работы сотрудников.

Таким образом, анализ существующих на рынке систем выявил, на реализацию каких функциональных особенностей следует обратить внимание в разрабатываемом приложении, а также каких недостатков стоит избегать в реализации собственной системы.

1.2 Анализ требований

В данной главе будут проанализированы требования к разрабатываемой системе и определены цели на ее разработку. Анализ требований является важным этапом в разработке приложений. В коммерческих условиях это позволяет минимизировать недопонимание между заказчиком и разработчиком, в рамках дипломного проекта это поможет предварительно сформировать образ будущего проекта и в дальнейшем определиться с технологиями разработки.

1.2.1 Анализ функциональных требований

Функциональные требования определяют, что именно должна делать система, какие функции и возможности должны быть реализованы для обеспечения эффективной работы сотрудников колл-центра. Для разрабатываемой информационной системы можно выделить ключевые функциональные требования, описанные ниже.

Авторизация:

- система должна предоставлять возможность создания и редактирования учетных записей операторов и администраторов;
- система должна обеспечивать процесс аутентификации пользователей с использованием имени пользователя и пароля.

Учетные записи:

- система должна предоставлять сотруднику доступ к персональным данным его учетной записи;
- система должна предоставлять администратору возможность просматривать список всех закрепленных за ним операторов с информацией о статусе их учетных записей.

Настройка проекта:

- система должна предоставлять администратору возможность создания проекта;
- система должна предоставлять администратору возможность

настройки скрипта диалога и статусов совершаемых звонков;

- система должна предоставлять администратору возможность загружать клиентскую базу из документа Excel;
- система должна предоставлять администратору возможность назначать на проект подчиненных операторов;
- система должна предоставлять администратору возможность редактировать проект.

Сбор данных:

- система должна предоставлять сотруднику возможность редактировать незавершенные звонки в рамках текущей сессии;
- система должна предоставлять администратору возможность просмотра истории завершенных звонков операторов, включая дату, время, длительность и информацию о клиенте;
- система должна обеспечивать доступ к записям звонков для прослушивания и анализа;
- система должна предоставлять администратору возможность редактировать некорректные и устанавливать неустановленные статусы звонков;
- система должна предоставлять администратору инструменты для оценки записанных диалогов операторов с клиентами по заранее установленным критериям (следование скрипту, грамотность речи, достоверность информации, корректность отмеченных результатов взаимодействия);
- система должна автоматически рассчитывать общий рейтинг операторов и рейтинг по отдельным критериям на основе выставленных оценок;
- система должна предоставлять администратору возможность выгрузки отчетов о работе операторов в формате Excel.

Звонки:

- система должна предоставлять сотруднику возможность отметить статус (или перечень статусов) звонка;
- система должна предоставлять сотруднику возможность сохранить

ЗВОНОК.

Диаграмма вариантов использования системы представлена на рисунке 7.

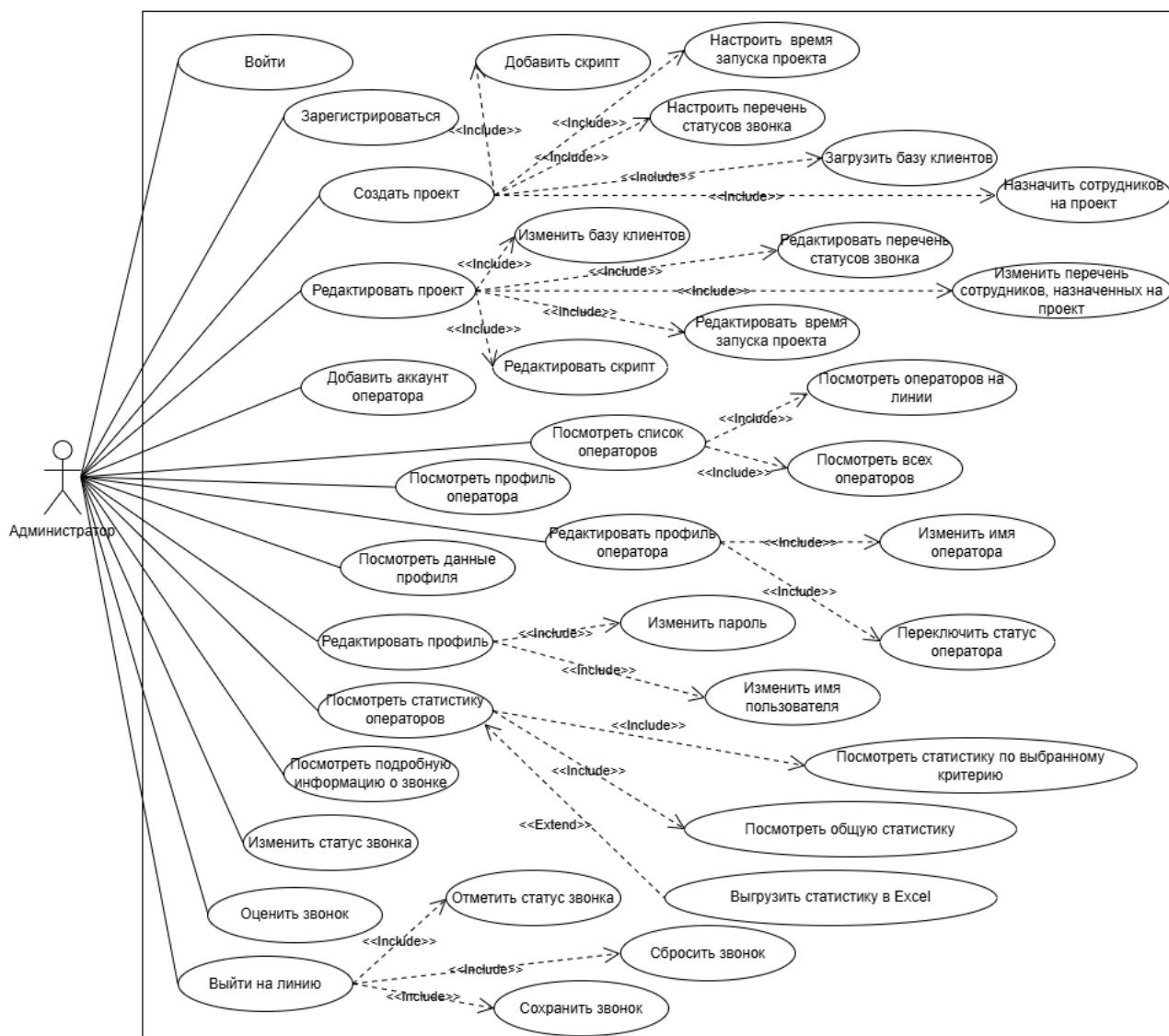


Рисунок 7 – UML диаграмма вариантов использования

Спецификации вариантов использования системы приведены в приложении Б. Выделенные варианты использования будут реализованы при разработке приложения.

1.2.2 Анализ нефункциональных требований

Нефункциональные требования описывают характеристики системы, которые не относятся непосредственно к ее функциональности, но имеют важное значение для ее успешной работы. К ним относятся в том числе ограничения,

налагаемые на систему возможностями оборудования, под управлением которого она должна функционировать. Нефункциональные требования важны, так как позволяют гарантировать соответствие системы потребностям пользователей [1].

В ходе анализа нефункциональных требований были выделены ключевые требования к программному и аппаратному обеспечению. В отношении сервера:

- на сервере должна быть установлена СУБД PostgreSQL, так как приложение использует ее для хранения данных;
- сервер должен обеспечивать достаточную производительность для поддержания работы большого числа пользователей в системе одновременно и без задержек обрабатывать запросы, так как это критически важно из-за специфики предметной области.

Минимальные требования к серверу представлены в таблице 1.

Таблица 1 – Минимальные требования программному и аппаратному обеспечению сервера

Компонент	Минимальные требования
Операционная система	Windows Server 2019/2022, Linux (Ubuntu 20.04+ / Debian 10+ / CentOS 7+)
Процессор	4 ядра
ОЗУ	8 ГБ (рекомендуется 16 ГБ при большой нагрузке)
Жесткий диск	100 ГБ (рекомендуется 200+ ГБ)
СУБД	PostgreSQL
.NET Runtime	.NET 6+

Приложение клиента должно запускаться в браузере, поддерживающем Web Assembly (под данный критерий подходят большинство современных браузеров, исключения составляют старые версии Internet Explorer, а также, в некоторых случаях, Microsoft Edge при включении режима повышенной безопасности [2]), кроме того, для нормальной работы необходимо стабильное

подключение к Интернету. Минимальные требования к программному и аппаратному обеспечению клиента представлены в таблице 2.

Таблица 2 – Минимальные требования программному и аппаратному обеспечению клиента

Компонент	Минимальные требования
Браузер	Google Chrome 57+, Mozilla Firefox 52+, Safari 11+, Microsoft Edge 16+, Opera 44+.
Операционная система	Windows 10 +, macOS 10.15+, Linux (современные дистрибутивы)
Процессор	2 ядра
ОЗУ	4 ГБ (рекомендуется 8 ГБ для комфортной работы)
Интернет-соединение	Стабильное подключение (минимум 5 Мбит/с)

Выделенные нефункциональные требования должны быть учтены при разработке, а также при внедрении системы и ее дальнейшей эксплуатации пользователями.

1.3 Постановка задачи

В рамках дипломного проекта необходимо разработать информационную систему, которая должна предоставить администратору колл-центра удобный и эффективный способ управления проектами и учетными записями сотрудников, а также производить мониторинг их эффективности.

В информационной системе «Автоматизированное рабочее место администратора отдела информирования клиентов организации» должны решаться следующие задачи:

- вход в систему;
- регистрация в системе;
- создание проекта;

- редактирование созданного проекта;
- приостановка и запуск проекта;
- загрузка базы номеров в проект;
- закрытие проекта;
- назначение сотрудников на созданный проект;
- редактирование профиля администратора;
- предоставление доступа к данным учетных записей операторов;
- создание и редактирование учетной записи оператора;
- предоставление доступа к подробной информации о записанном звонке;
- редактирование статуса звонка;
- оценка звонка в соответствии с критериями;
- сбор и предоставление статистики операторов.

В результате выполнения этих задач должна быть получена система, которая предоставляет администратору колл-центра удобный и эффективный способ управления проектами и учетными записями сотрудников, мониторинга их эффективности.

Приложение должно иметь интуитивно понятный пользовательский интерфейс, который будет легок в навигации и обеспечит быстрый доступ к необходимым функциям.

2 Проектирование web-приложения

В данном разделе пояснительной записки будет представлено проектирование информационной системы «Автоматизированное рабочее место администратора отдела информирования клиентов организации», определены технологии для реализации системы, определена структура проекта и проработан графический интерфейс пользователя.

2.1 Выбор технологий разработки

В качестве основного языка для разработки приложения был выбран язык C# и платформа ASP.NET Core. Для интеграции базы данных PostgreSQL был выбран Entity Framework – фреймворк от Microsoft, который позволяет работать с базой данных, используя объекты C# вместо прямых запросов на языке SQL [3]. Кроме того, он представляет такие удобные опции, как:

- автоматическое сопоставление классов таблиц с базой данных;
- автоматическое определение изменений в объектах и их синхронизация с базой данных.

Для разработки графического интерфейса приложения было принято решение использовать Blazor – фреймворк для создания веб-приложений от компании Microsoft, который позволяет разработчикам использовать C# и .NET вместо JavaScript для разработки интерактивных пользовательских интерфейсов [4]. Была выбрана версия Blazor WebAssembly, где HTML рендерится в браузере пользователя, а не обрабатывается сервером. И, хотя эта версия имеет недостаток – медленную загрузку при первом запуске, ее использование позволяет разгрузить сервер. Поскольку система разрабатывается для использования большим количеством пользователей одновременно, то снижение нагрузки на сервер является важным аспектом.

Таким образом, были определены основные технологии разработки приложения «Автоматизированное рабочее место администратора отдела информирования клиентов организации».

2.2 Проектирование архитектуры приложения

В рамках разработки приложения было принято решение придерживаться архитектуры на основе сервисов.

Типовая топология архитектуры на основе сервисов – это распределенная макроуровневая структура, состоящая из отдельно развернутого пользовательского интерфейса, отдельно развернутых удаленных крупномодульных сервисов и монолитной базы данных (рисунок 8) [5].

Характерные черты архитектуры на основе сервисов:

- сервисы, как правило, крупномодульны и разделены по предметной области;
- обычно используется единая монолитная база данных;
- чаще всего количество сервисов в системе варьируется от 4 до 12 единиц;
- иногда присутствует API-уровень, но чаще интерфейс обращается к сервисам напрямую.

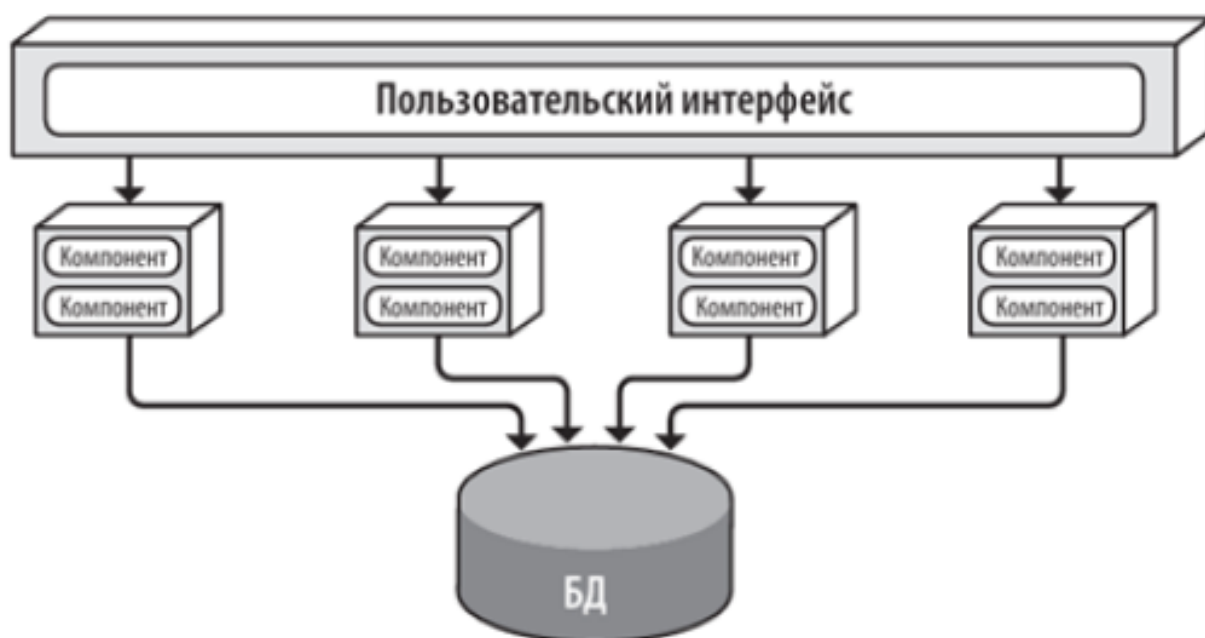


Рисунок 8 – Типовая топология архитектуры на основе сервисов

У использования данного подхода есть ряд преимуществ, являющихся критически важными при разработке описываемого веб-приложения:

- каждый сервис можно разрабатывать и тестировать независимо, что упрощает процесс отладки кода;
- для каждого сервиса можно использовать различные библиотеки и модули, что делает структуру проекта более упорядоченной;
- сервисная архитектура устойчива к сбоям – если один сервис выходит из строя, это не приводит к сбою всей системы;
- сервисы могут независимо дорабатываться, что упрощает модификацию всей системы;
- из-за специфики предметной области возникает необходимость использовать единую базу данных, что является характерной чертой сервисной архитектуры;
- в рамках создания приложения для автоматизации работы сотрудника колл-центра целесообразно распределить функционал системы между небольшим количеством крупных модулей.

Таким образом, использование архитектуры на основе сервисов позволит разбить предметную область на логические составляющие («Статистика», «Управление звонками», «Управление учетными записями», «Управление проектами») и разделить функционал системы между отдельными приложениями в составе одного проекта. Подробно о проектировании сервисов системы будет отмечено в разделе 2.2.2 – «Проектирование сервисов».

2.2.1 Проектирование базы данных

В данном подразделе будет представлено проектирование базы данных разрабатываемого приложения. Базы данных являются неотъемлемой частью большинства современных систем. Ее проектирование можно отнести к одной из самых важных задач в рамках создания проекта, так как база данных является «фундаментом» проекта. Кардинально изменять ее топологию в процессе разработки – это трудозатратное действие, которое, кроме того, может повлечь за собой нарушение целостности данных, поэтому важно ответственно и тщательно подходить к проработке структуры.

К основным принципам построения базы данных можно отнести:

- нормализацию (данные не должны слишком часто дублироваться);
- денормализацию (данные могут осознанно дублироваться, если это значительно ускоряет к ним доступ);
- оптимизацию запросов;
- масштабируемость структуры;
- безопасность [6].

В рамках данного проекта целесообразно использовать единую базу данных, так как все сущности системы неразрывно связаны между собой и должны реагировать на изменения в других сущностях. На основе анализа предметной области была спроектирована схема базы данных, приведенная на рисунке 9.

Ключевым элементом системы является сущность «Проект», вокруг которой организованы все остальные. Как было отмечено в разделе 1 – «Анализ предметной области», проект – это тематика обзвона клиентов. У проекта есть его создатель – администратор, участники – операторы (одним из которых администратор также может являться), а также клиенты. Совершенные звонки также относятся к определенному проекту, кроме того, у проекта есть перечень статусов, которые оператор может выставить по завершению диалога с клиентом, чтобы отметить его результаты. Статусы объединены в группы по тематике. Статусы связаны со звонками по принципу «многие ко многим». Со звонками также связаны сущности «Оценка» и «Заметка». Они нужны для отслеживания работы оператора.

На основе оценок, которые администратор проставляет некоторой случайной выборке звонков оператора, составляется рейтинг оператора. Администратор опционально может оставить заметку, которая поможет оператору понять, какие ошибки были совершены и над чем ему следует поработать.

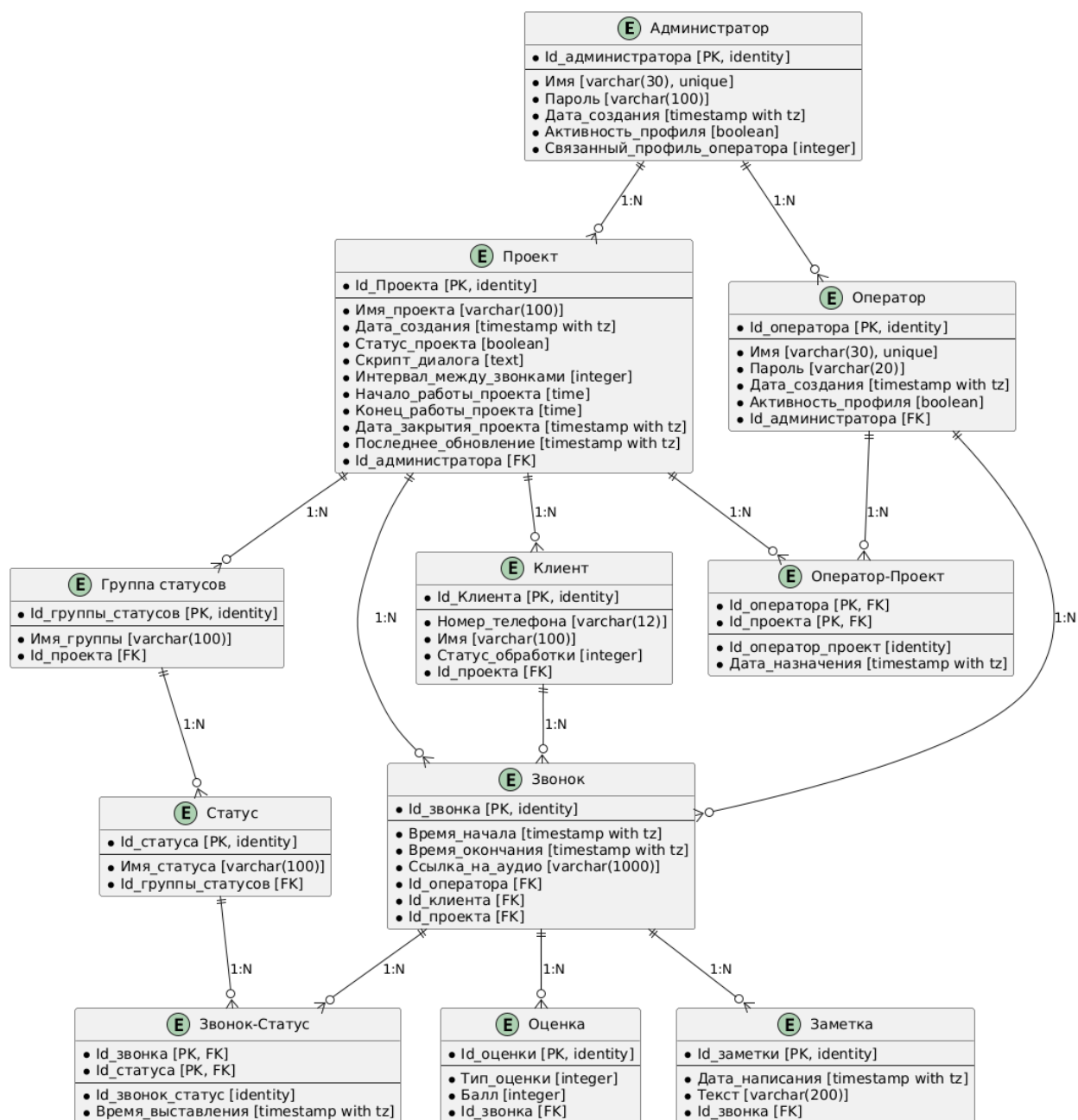


Рисунок 9 – Схема базы данных проекта

Спроектированный концепт базы данных будет использован при разработке реального проекта.

2.2.2 Проектирование сервисов

Согласно выбранной архитектуре, серверная логика приложения будет разбита на несколько независимых сервисов, разделенных согласно предметным областям. В процессе проектирования было принято решение разбить предметную область на следующие логические составляющие:

- сервис авторизации;
- сервис данных об учетных записях пользователей;
- сервис данных о проектах;
- сервис данных звонков;
- сервис расчета и предоставления статистики.

Также будет предусмотрен слой API – приложение в рамках проекта, которое будет перенаправлять запросы клиентского приложения к соответствующим сервисам. Это позволит значительно упростить маршрутизацию и сделать систему более гибкой и масштабируемой.

Для визуализации компонентов, входящих в состав системы, была использована UML-диаграмма компонентов (рисунок 10). Она позволяет отобразить особенности физического представления системы и определить ее архитектуру.

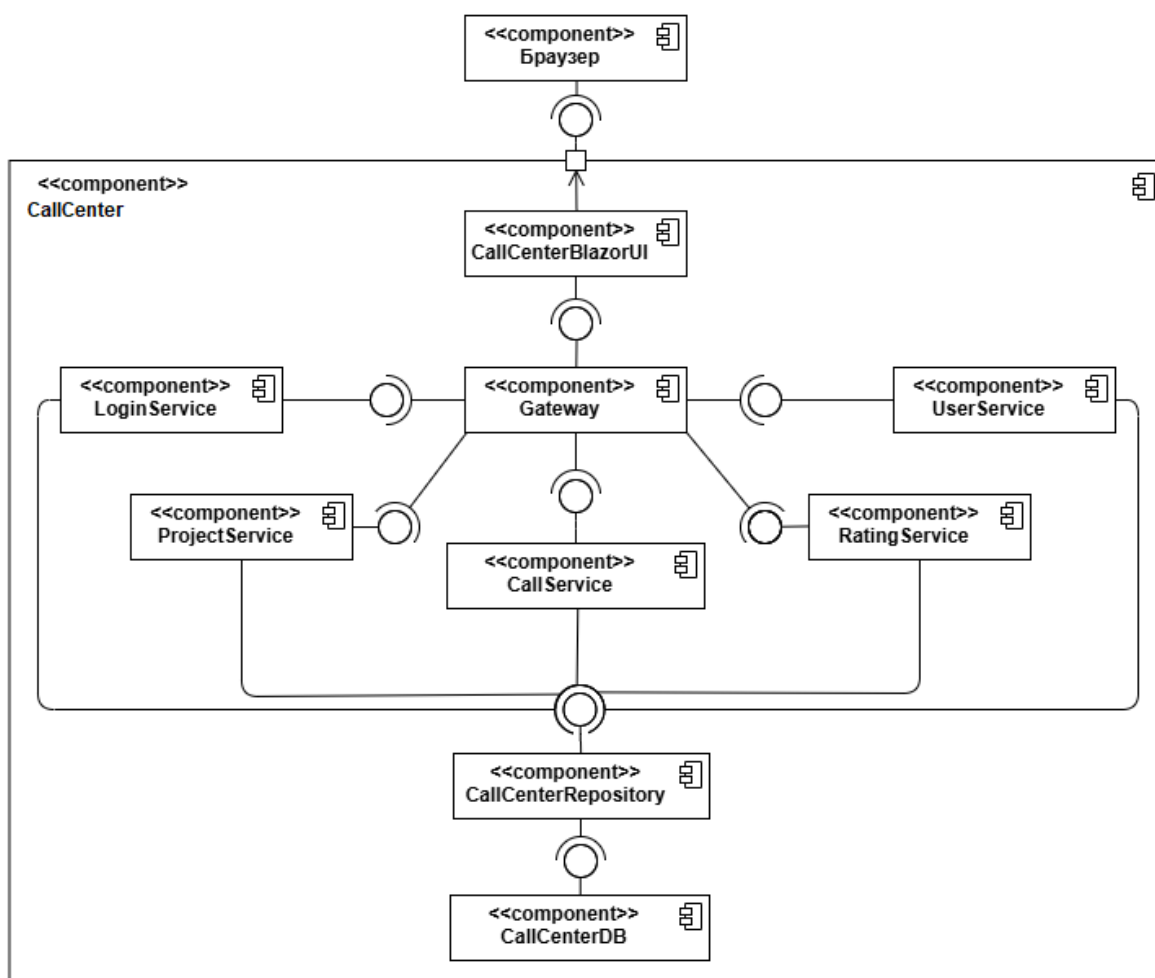


Рисунок 10 – Диаграмма компонентов приложения

На диаграмме показаны взаимосвязи между компонентами системы. Приложение CallCenterBlazorUI направляет запросы к маршрутизирующему сервису Gateway, который, в свою очередь, перенаправляет их пяти целевым сервисам:

- сервис авторизации – LoginService;
- сервис учетных записей – UserService;
- сервис проектов – ProjectService;
- сервис звонков – CallService;
- сервис рейтингов – RatingService.

Также на диаграмме отображена библиотека CallCenterRepository, которая содержит логику и настройки для работы с базой данных. Библиотека имеет связь с базой данных CallCenterDB.

Далее будет рассмотрена структура сервисов на примере сервиса авторизации LoginService и сервиса звонков CallService. Диаграмма классов на для сервиса авторизации изображена на рисунке 11.

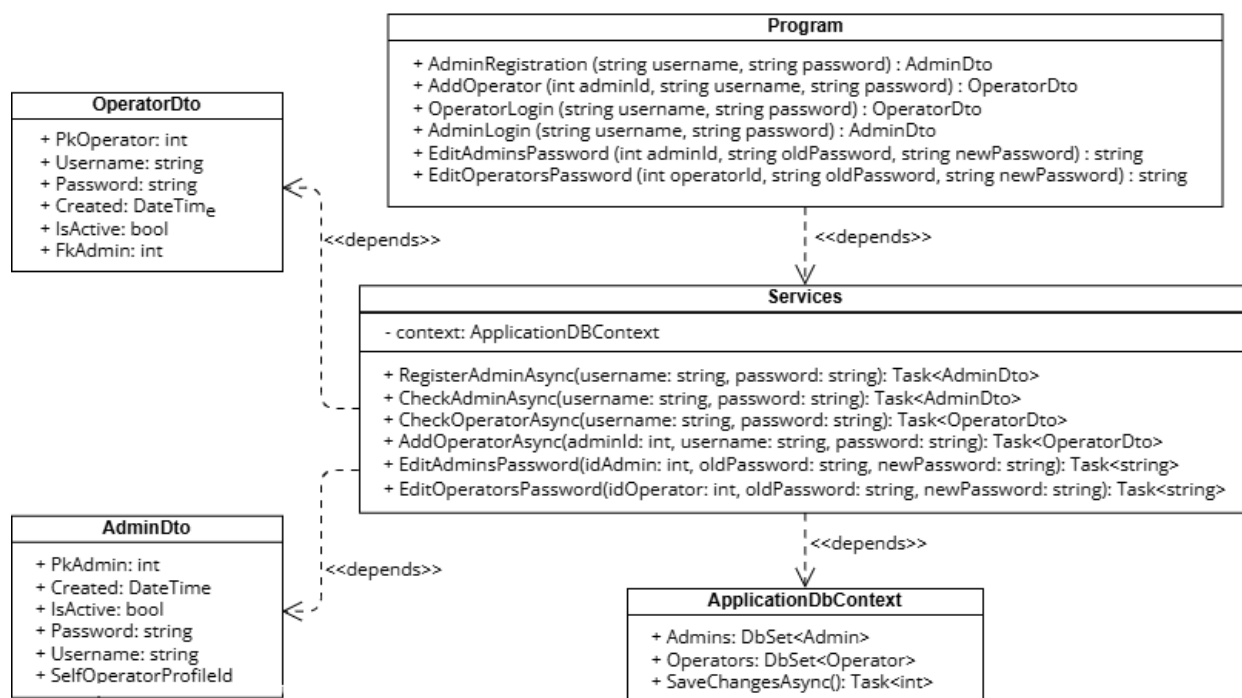


Рисунок 11 – Диаграмма классов сервиса авторизации

На диаграмме показаны взаимосвязи между классами сервиса, а также

отображены функции, им предоставляемые. К ним относятся:

- регистрация администратора;
- вход администратора в систему;
- вход оператора в систему;
- регистрация профиля оператора;
- смена пароля администратора;
- смена пароля оператора.

Эндпоинты главной программы используют функции для работы с базой данных класса Services. Для соединения с базой данных используется класс ApplicationDbContext библиотеки CallCenterRepository.

Диаграмма классов на для сервиса сервиса данных о звонках изображена на рисунке 12.

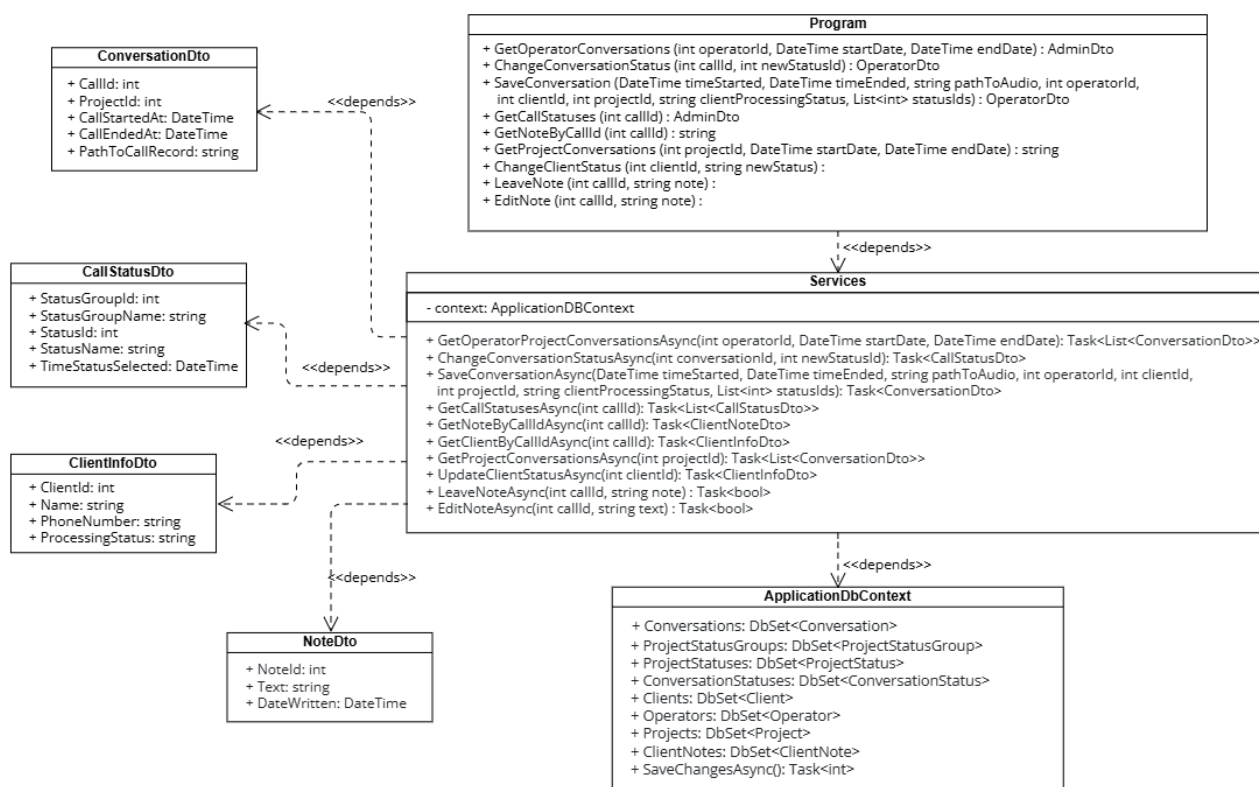


Рисунок 12 – Диаграмма классов сервиса данных о звонках

Как видно из диаграммы, к основным функциям данного сервиса относятся:

- сохранение звонка;

- получение всех записанных звонков, принадлежащих проекту;
- получение всех записанных звонков оператора;
- получение отмеченных оператором результатов диалога с клиентом (статусов звонка);
- редактирование отмеченного результата диалога;
- получение оставленной о звонке заметки, если таковая есть;
- получение информации о клиенте;
- изменение статуса обработки клиента («обработан», «не обработан», «перезвонить», «неверный номер»).

Другие сервисы системы имеют похожую общую структуру. Таким образом было проведено проектирование серверной части приложения.

2.3 Проектирование графического интерфейса

Проектирование пользовательского интерфейса – одна из важных задач в рамках создания приложения. К основным принципам построения грамотного интерфейса пользователя можно отнести следующее:

- интерфейс должен быть интуитивно понятным и удобным в использовании, стоит избегать нагромождения элементов;
- элементы интерфейса должны быть выполнены в едином стиле, одинаковые действия должны выполняться одинаково – это поможет избежать раздражения у пользователя и улучшит опыт взаимодействия с системой;
- пользователь должен видеть последствия своих действий, например, нажатие на кнопку может сопровождаться визуальным или звуковым эффектом;
- обобщающие разделы должны быть выделены размером шрифта и оформлением, чтобы пользователь мог быстро находить нужную вкладку;
- человек не должен запоминать много информации для работы с интерфейсом, поэтому меню и основные элементы управления должны всегда находиться перед глазами;
- интерфейс должен соответствовать тому оборудованию, на котором

будет использоваться (например, мобильный телефон, настольный компьютер, специализированное оборудование и т. д.) [7].

Следуя этим принципам, в рамках данной работы с помощью графического редактора были спроектированы макеты интерфейса приложения.

На рисунке 13 изображен макет веб-страницы входа в приложение.

Макет веб-страницы входа в приложение (администратор). В центре экрана находится форма с заголовком **Вход/Регистрация**. Над формой, выделенной красной линией с цифрой 1, находится ссылка [Администратор](#). Форма сама по себе выделена красной линией с цифрой 2. В форме есть два поля ввода: первое с меткой ***имя** (выделено красной линией с цифрой 3) и второе с меткой ***пароль** (выделено красной линией с цифрой 4). В нижней части формы находится ссылка [Войти / Зарегистрироваться](#), выделенная красной линией с цифрой 5.

Рисунок 13 – Макет страницы входа в приложение (администратор)

Цифрами на рисунке обозначены основные элементы:

- 1 – элемент переключения роли;
- 2 – поле для ввода имени пользователя;
- 3 – поле для ввода пароля;
- 4 – кнопка регистрации;
- 5 – кнопка входа в систему.

При нажатии на элемент, обозначенный как 1, пользователь может переключить роль, под которой он войдет в систему. Стартовое окно для роли «оператор» ничем не отличается от аналогичного для администратора, за исключением отсутствия возможности регистрации. Это связано с тем, что в рамках специфики системы аккаунты операторов может создавать только

администратор, а сотрудник затем входит в созданную учетную запись. При нажатии на кнопку «Войти», на макете обозначенную цифрой 4, администратор попадает на страницу профиля.

На рисунке 14 изображен макет страницы профиля администратора, на которую тот попадает в случае успешного входа или регистрации.

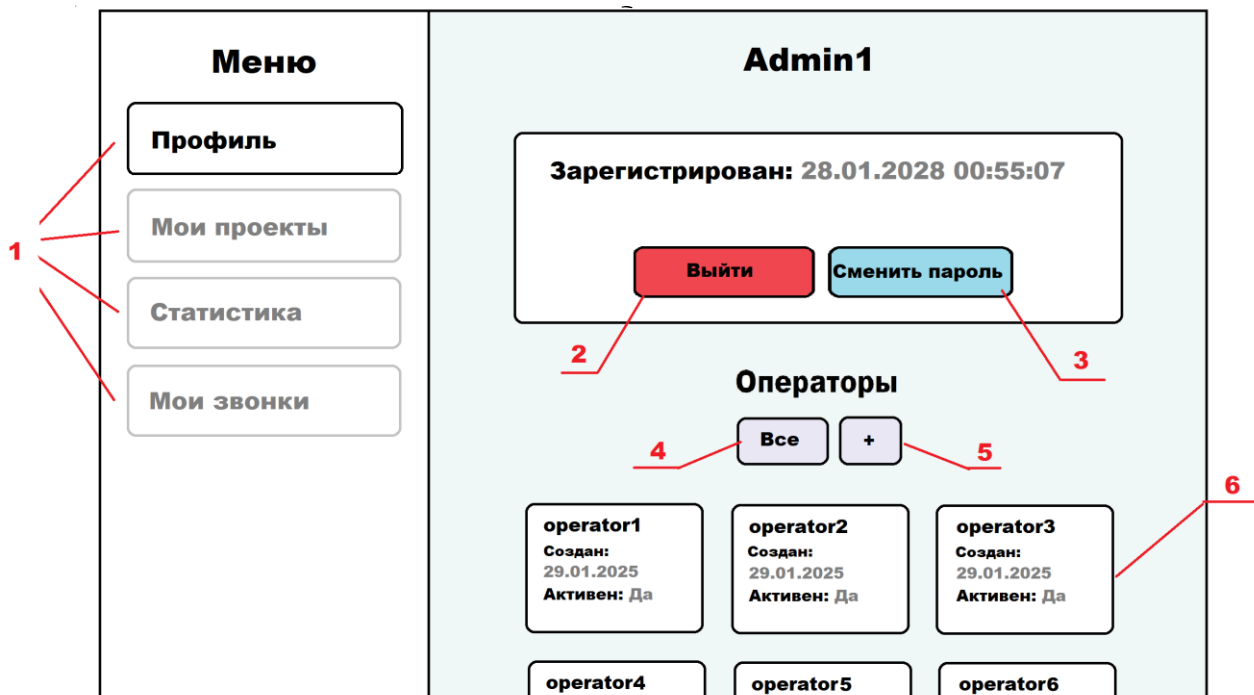


Рисунок 14 – Макет страницы профиля администратора

Цифрами обозначены основные элементы страницы:

- 1 – кнопки меню перехода между страницами;
- 2 – кнопка выхода из профиля;
- 3 – кнопка смены пароля;
- 4 – кнопка переключения отображения операторов с варианта «Только активные» на вариант «Все»;
- 5 – кнопка для добавления нового профиля оператора;
- 6 – элемент для отображения оператора, при нажатии на который происходит переход на страницу с информацией об операторе.

При нажатии на блок, отображающий информацию об операторе, пользователь переходит на страницу с информацией об этом сотруднике. На данной странице он может увидеть, в каких проектах задействован оператор, а

также при необходимости деактивировать профиль сотрудника (например, если он уже не работает в организации), или сменить имя профиля.

На рисунке 15 изображен макет страницы для отображения информации об операторе.



Рисунок 15 – Макет страницы информации об операторе

Основные элементы страницы обозначены цифрами:

- 1 – раскрываемая область – редактирование профиля оператора;
- 2 – поле для ввода нового имени;
- 3 – кнопка сохранения имени;
- 4 – кнопка переключения активности профиля;
- 5 – элемент перехода к звонкам оператора по выбранному проекту.

На рисунках 16, 17 изображен макет страницы проектов, которая отображает проекты, созданные данным администратором, а также модальное окно для добавления нового. На странице с помощью выпадающего списка можно выбрать категорию проектов, которая будет отображена.

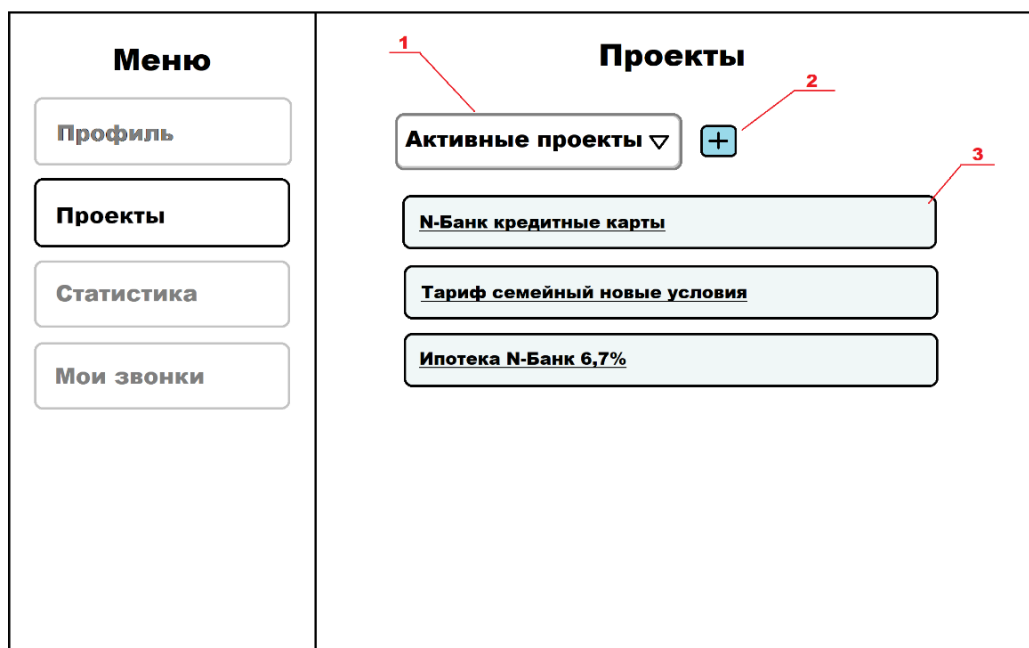


Рисунок 16 – Макет страницы проектов

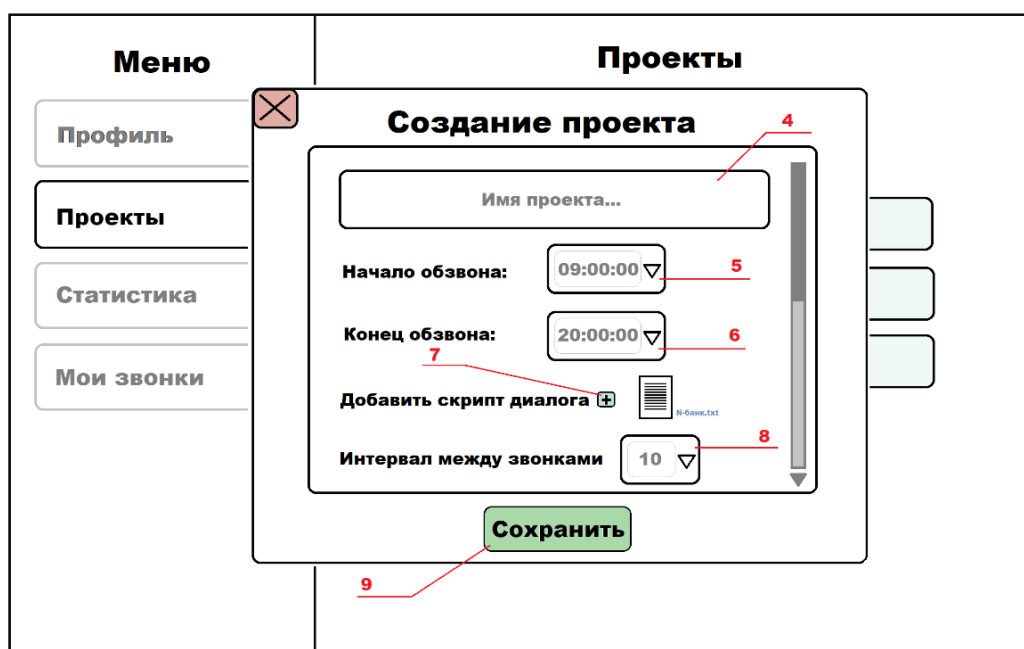


Рисунок 17 – Макет страницы проектов (с модальным окном)

На макете цифрами обозначены основные элементы страницы:

- 1 – выпадающий список для выбора категории отображаемых проектов (активные, приостановленные, закрытые);
- 2 – кнопка для добавления нового проекта, при нажатии на нее откроется модальное окно для ввода параметров нового проекта;
- 3 – кнопка перехода на страницу проекта;

- 4 – поле для ввода имени проекта;
- 5 – поле для ввода времени начала работы проекта;
- 6 – поле для ввода времени окончания работы проекта;
- 7 – кнопка для загрузки скрипта проекта;
- 8 – поле для ввода интервала времени между звонками;
- 9 – кнопка для сохранения нового проекта.

На рисунках 18, 19 изображен макет раздела информации о проекте страницы проекта.

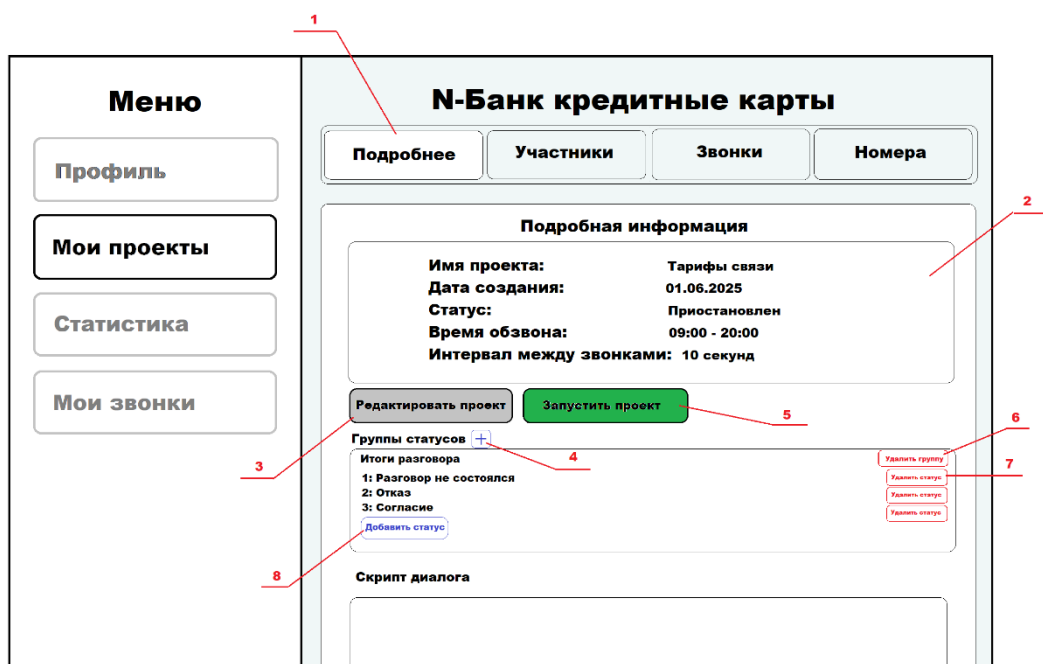


Рисунок 18 – Макет страницы проекта (часть 1)

Далее будут приведены обозначенные цифрами элементы страницы:

- 1 – кнопка переключения раздела страницы;
- 2 – краткая информация о проекте;
- 3 – кнопка для редактирования информации о проекте;
- 4 – кнопка добавления новой группы статусов;
- 5 – кнопка запуска проекта;
- 6 – кнопка удаления группы статусов;
- 7 – кнопка удаления статуса из группы статусов;
- 8 – кнопка добавления нового статуса в существующую группу статусов.

Меню

- Профиль
- Мои проекты
- Статистика
- Мои звонки

Подробная информация

Имя проекта: Тарифы связи
 Дата создания: 01.06.2025
 Статус: Приостановлен
 Время обзвона: 09:00 - 20:00
 Интервал между звонками: 10 секунд

Редактировать проект Запустить проект

Группы статусов +

Итоги разговора

1: Разговор не состоялся
 2: Отказ
 3: Согласие

Добавить статус

Удалить группу
 Удалить статус
 Удалить статус
 Удалить статус

Скрипт диалога

Сохранить Выбрать файл

Закрыть проект

9 10 11 12

Рисунок 19 – Макет страницы проекта (часть 2)

На данном рисунке цифрами обозначены остальные элементы страницы проекта:

- 9 – поле для ввода или редактирования скрипта диалога с клиентом;
- 10 – кнопка сохранения скрипта диалога;
- 11 – кнопка выбора файла, из которого будет загружен скрипт диалога для проекта;
- 12 – кнопка закрытия проекта.

Таким образом были спроектированы основные страницы веб-приложения.

Данные макеты страниц были использованы для создания реального интерфейса приложения.

2.4 Оценка бюджета разработки web-приложения

Для оценки и визуализации трудовых и финансовых затрат на разработку приложения «Автоматизированное рабочее место администратора отдела информирования клиентов организации» была разработана диаграмма Ганта с помощью программы Microsoft Office Project [8].

Диаграмма Ганта – это инструмент для визуального представления графика выполнения проекта. Она позволяет наглядно увидеть временные рамки задач, их последовательность, взаимосвязи, а также текущее состояние выполнения. Она получила свое название в честь американского инженера Генри Ганта, который разработал этот метод в начале 20 века [9].

В таблице 3 представлены задачи проекта и время на их выполнение.

Таблица 3 – Задачи проекта

Название задачи	Длительность	Начало	Окончание
<i>Анализ предметной области</i>	<i>6 дней</i>	<i>08.04.2025</i>	<i>13.04.2025</i>
Анализ существующих систем аналогичного назначения	1 день	08.04.2025	08.04.2025
<i>Анализ требований</i>	<i>4 дня</i>	<i>09.04.2025</i>	<i>12.04.2025</i>
Анализ функциональных требований	2 дня	09.04.2025	10.04.2025
Анализ нефункциональных требований	2 дня	11.04.2025	12.04.2025
Постановка задачи	1 день	13.04.2025	13.04.2025
<i>Проектирование web-приложения</i>	<i>14 дней</i>	<i>14.04.2025</i>	<i>27.04.2025</i>
Выбор технологии разработки	2 дня	14.04.2025	15.04.2025
Проектирование архитектуры приложения	6 дней	16.04.2025	21.05.2025
Проектирование графического интерфейса	4 дня	22.04.2025	25.04.2025
Оценка бюджета разработки приложения	2 дня	26.04.2025	27.04.2025
<i>Разработка и реализация приложения</i>	<i>21 день</i>	<i>28.04.2025</i>	<i>18.05.2025</i>

Продолжение таблицы 3

Название задачи	Длительность	Начало	Окончание
Разработка модуля учетных записей	5 дней	28.04.2025	02.05.2025
Разработка модуля проектов	6 дней	03.05.2025	08.05.2025
Разработка модуля звонков	5 дней	09.05.2025	13.05.2025
Разработка модуля статистики	5 дней	14.05.2025	18.05.2025
<i>Тестирование</i>	<i>10 дней</i>	<i>19.05.2025</i>	<i>28.05.2025</i>
Функциональное тестирование	6 дней	19.05.2025	24.05.2025
Автоматизированное тестирование	3 дня	25.05.2025	27.05.2025
Расчет метрик кода	1 день	28.05.2025	28.05.2025
<i>Оформление документации</i>	<i>7 дней</i>	<i>29.05.2025</i>	<i>03.06.2025</i>
Написание документации	6 дней	29.05.2025	02.06.2025
Оформление и печать документации	1 день	03.05.2025	03.06.2025

Так как данный проект разрабатывается одним человеком на некоммерческой основе, количество трудовых ресурсов было рассчитано как отношение количества часов работы определенного ресурса (аналитик, проектировщик, программист, тестировщик) к общему количеству рабочих часов для всех трудовых ресурсов. Число часов работы каждого ресурса было автоматически подсчитано приложением Microsoft Office Project. На рисунке 18 представлен расчет трудовых затрат.

В таблице 4 представлены ресурсы, затраченные в ходе выполнения проекта. На рисунке 19 представлена диаграмма Ганта для разрабатываемой системы.

Таблица 4 – Ресурсы проекта

Название	Тип	Количество	Стоимость
Аналитик	Трудовой	0,23	-
Проектировщик	Трудовой	0,24	-
Программист	Трудовой	0,36	-
Тестировщик	Трудовой	0,17	-
Компьютер	Материальный	1	21000 рублей
Бумага	Материальный	Пачка	500 рублей
Оплата сервиса Voximplant	Затраты	1 месяц	699 рублей/мес.
Принтер	Затраты	1	550 рублей

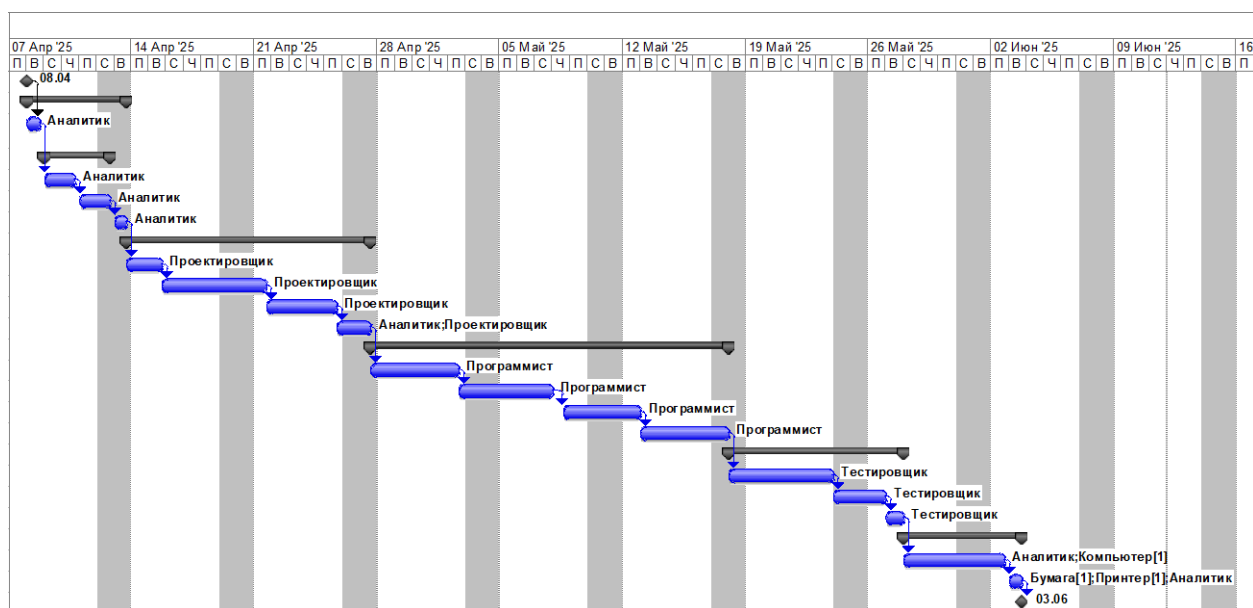


Рисунок 19 – Диаграмма Ганта

Расчеты, представленные в данном разделе, показали, что затраты на реализацию проекта составят 22749 рублей.

3 Разработка и реализация приложения

В данном разделе будет представлен ход разработки и реализации приложения «Автоматизированное рабочее место администратора отдела информирования клиентов организации».

3.1 Реализация базы данных приложения

Как было отмечено в разделе 2.1 – «Выбор технологий разработки», для интеграции базы данных в систему используется Entity Framework Core и СУБД PostgreSQL. Entity Framework позволяет автоматически связывать классы языка #C с объектами базы данных. Модели базы данных и настройки для работы с ней были вынесены в отдельный проект-библиотеку – CallCenterRepository.

Для того, чтобы использовать Entity Framework в проекте, необходимо установить пакеты Microsoft.EntityFrameworkCore.Design (для инструментов миграций) и Npgsql.EntityFrameworkCore.PostgreSQL (для работы с PostgreSQL). Затем необходимо определить сущности базы данных. Для этого создаются классы таблиц. В листинге 1 представлен пример создания класса таблицы администраторов (полный код класса Admin, включающий конструктор, представлен в приложении А.1).

Листинг 1 – Класс таблицы администраторов

```
[Table("Admins")]
public class Admin
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int PkAdmin { get; set; }

    [Required(ErrorMessage = "Username is required.")]
    [MaxLength(30)]
    public string Username { get; set; }

    [Required(ErrorMessage = "Password is required.")]
    [MaxLength(100)]
    public string Password { get; set; }

    [Required]
    public DateTime Created { get; set; } = DateTime.UtcNow;
```

```

[Required]
public bool IsActive { get; set; } = true;

[Required]
public int SelfOperatorProfile { get; set; }

}

```

В классе используются следующие атрибуты:

- [Table («Admins»)] указывает, что этот класс соответствует таблице с именем «Admins»;
- [Key] указывает, что свойство является первичным ключом;
- [DatabaseGenerated (DatabaseGeneratedOption.Identity)] означает, что значение будет автоматически генерироваться базой данных при вставке;
- [Required] определяет поле как обязательное для заполнения;
- [MaxLength ()] задает максимальную длину строки в базе данных.

Указание данных атрибутов является опциональным и определяется нуждами проекта.

После создания классов таблиц необходимо создать класс контекста базы данных, который будет унаследован от класса DbContext пространства имен Microsoft.EntityFrameworkCore. Для разрабатываемого приложения был создан класс ApplicationDbContext (полный его код представлен в приложении А.2). В листинге 2 представлен пример объявления коллекций DbSet, представляющих сущности определенного типа в контексте базы данных. Типами объектов коллекций в данном случае являются созданные ранее классы таблиц.

Листинг 2 – Фрагмент класса ApplicationDbContext – объявление DbSet

```

public DbSet<Project> Projects { get; set; }
public DbSet<OperatorProject> OperatorProjects { get; set; }
public DbSet<Admin> Admins { get; set; }
public DbSet<Operator> Operators { get; set; }

```

Свойство Projects представляет таблицу проектов в базе данных, Operators – таблицу операторов, Admins – таблицу администраторов, OperatorProjects – таблицу связей между проектами и операторами, через которую реализована

связь «многие-ко-многим».

После определения всех таблиц необходимо задать отношения между ними и необходимые ограничения. Это осуществляется путем переопределения метода `OnModelCreating`. В листинге 3 показан пример определения отношений между таблицами.

Листинг 3 – Фрагмент класса `ApplicationDbContext` – ограничения и отношения между таблицами

```
modelBuilder.Entity<OperatorProject>()
    .HasKey(op => new { op.FkOperator, op.FkProject });

modelBuilder.Entity<OperatorProject>()
    .HasOne(op => op.Operator)
    .WithMany()
    .HasForeignKey(op => op.FkOperator);

modelBuilder.Entity<OperatorProject>()
    .HasOne(op => op.Project)
    .WithMany(p => p.OperatorProjects)
    .HasForeignKey(op => op.FkProject);

modelBuilder.Entity<ConversationGrade>()
    .HasOne(cg => cg.Conversation)
    .WithMany()
    .HasForeignKey(cg => cg.FkConversation)
    .onDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<Admin>()
    .HasIndex(a => a.Username)
    .IsUnique();
```

В данном фрагменте кода метода `OnModelCreating` показано:

- создание связи «многие-ко-многим» на примере промежуточной таблицы `OperatorProjects` с составным ключом (один оператор может быть задействован во многих проектах, в одном проекте может принимать участие много операторов);
- настройка каскадного удаления на примере таблиц звонков и оценок звонка (при удалении объекта звонка происходит автоматическое удаление всех связанных с ним оценок);
- создание ограничений уникальности на примере имени пользователя

администратора (гарантирует, что не будет двух администраторов с одинаковым именем пользователя).

Для настройки подключения к базе данных используется перегруженный метод `OnConfiguring`, в котором задаются параметры подключения [11].

Чтобы создать или обновить базу данных с помощью Entity Framework необходимо создать и применить миграцию. Миграция – это механизм управления изменениями в структуре базы данных в Entity Framework Core. Они позволяют легко применять и откатывать изменения по мере развития приложения без потери данных [12]. В листинге 4 представлены команды для создания и применения миграции.

Листинг 4 – Создание и применение миграции

```
dotnet ef migrations add InitialCreate  
dotnet ef database update
```

Все созданные миграции сохраняются в папку `Migrations` проекта. На рисунке 20 представлена папка миграций проекта-библиотеки `CallCenterRepository`.

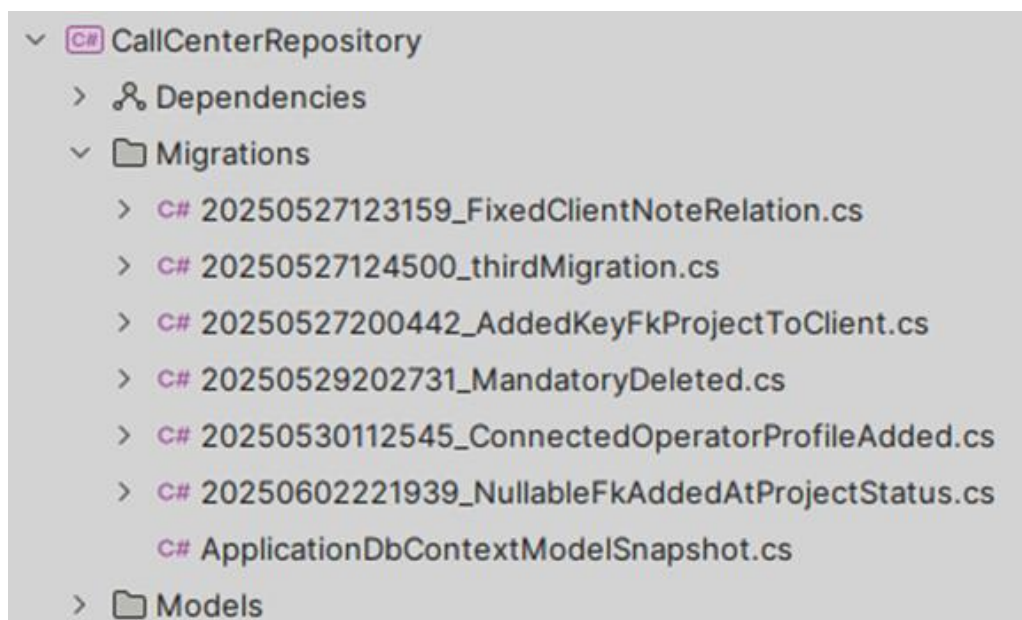


Рисунок 20 – Папка `Migrations` библиотеки `CallCenterRepository`

На рисунке 21 представлена схема созданной с помощью Entity Framework базы данных (автоматически сгенерированная средствами ide DataGrip).

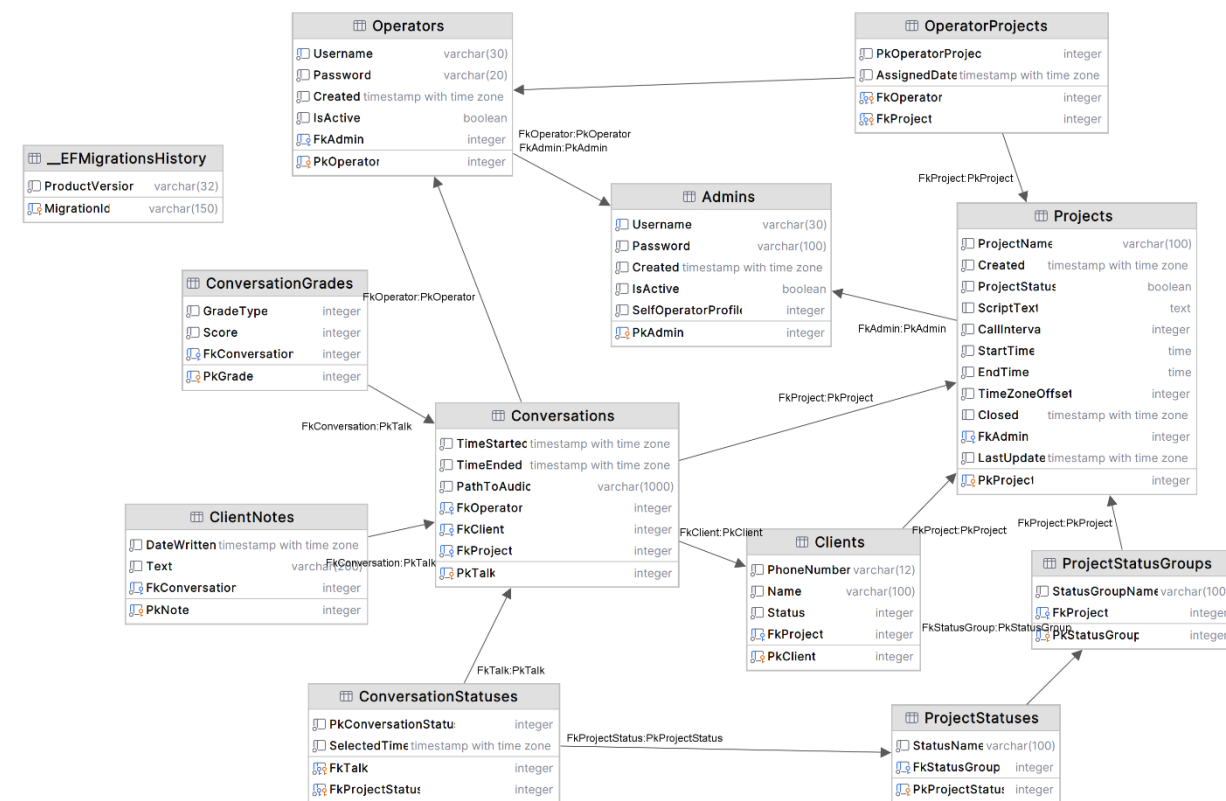


Рисунок 21 – Схема базы данных приложения, созданной с помощью Entity Framework

Таблица _EFMigrationHistory хранит историю миграций. Она не связана с другими сущностями базы данных.

Таким образом была создана база данных приложения «Автоматизированное рабочее место администратора отдела информирования клиентов организации».

3.2 Реализация сервисов

Как было отмечено в подразделе 2.2.2 – «Проектирование сервисов», логика приложения разделена между пятью независимыми сервисами:

- сервис авторизации – LoginService;
- сервис учетных записей – UserService;
- сервис проектов – ProjectService;
- сервис звонков – CallService;
- сервис рейтингов – RatingService.

Так как все сервисы организованы похожим образом, в данном разделе будет подробно описано устройство сервисов на примере UserService, представляющим функционал для управления учетными записями пользователей. Полный код сервиса UserService представлен в приложении А. Ссылка на репозиторий с полным кодом приложения содержится в находится по ссылке, указанной в источнике [10].

Сервис содержит основной класс Services.cs, который предоставляет бизнес-логику работы с пользователями. Он использует Entity Framework и библиотеку CallCenterRepository для взаимодействия с базой данных через класс ApplicationDbContext. Пример метода класса Services для получения всех операторов, закрепленных за конкретным администратором, представлен в листинге 5.

Листинг 5 – Метод класса Services – получить всех операторов администратора

```
public async Task<List<OperatorDto>>
GetAllOperatorsByAdminAsync(int adminId)
{
    var admin = await context.Admins.FirstOrDefaultAsync(a =>
a.PkAdmin == adminId);
    if (admin==null)throw new ArgumentException($"Администратор с
ID {adminId} не найден.");

    return await context.Operators
        .Where(op => op.FkAdmin == adminId && op.PkOperator!=
admin.SelfOperatorProfile)
        .Select(op => new OperatorDto
        {
            PkOperator = op.PkOperator,
            Username = op.Username,
            Password = op.Password,
            Created = op.Created,
            IsActive = op.IsActive,
            FkAdmin = op.FkAdmin
        })
        .ToListAsync();
}
```

Метод возвращает объект OperatorDto. Он представляет из себя модель данных типа record и используется с целью удобства сериализации и десериализации. В листинге 6 представлен его код.

Листинг 6 – Код record-типа представления данных OperatorDto

```
public record OperatorDto
{
    public int PkOperator { get; init; }
    public string? Username { get; init; }
    public string? Password { get; init; }
    public DateTime Created { get; init; }
    public bool IsActive { get; init; }
    public int FkAdmin { get; init; }
}
```

Главная программа содержит endpoints – адреса, по которым можно получить доступ к определенному ресурсу. Пример endpointa GetAllOperators для получения всех операторов, закрепленных за администратором, приведен в листинге 7.

Листинг 7 – Эндпоинт GetAllOperators

```
app.MapGet("admin/{adminId}/operators", async (int adminId) =>
{
    try
    {
        var operators = await
services.GetAllOperatorsByAdminAsync(adminId);
        Console.WriteLine("UserService method GetAllOperators may
be completed (or maybe not...)");
        return Results.Ok(operators); //возвращаем операторов
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine(ex.Message);
        return Results.NotFound(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.Data);
        return Results.StatusCode(500);
    }
}).WithName("GetAllOperators");
```

При обращении к нему клиенту возвращается json-объект – результат сериализации объекта OperatorDto.

Для маршрутизации запросов от приложения-клиента используется YARP – настраиваемый обратный прокси-сервер от Microsoft, задача которого

состоит в перенаправлении клиентских запросов на соответствующий сервис серверной части и последующем возврате ответа сервера клиенту [13].

Для подключения библиотеки YARP был создан отдельный ASP.NET Core проект Gateway, так как клиентская часть представляет собой приложение Blazor WASM, которое выполняется в браузере и не может само выступать в роли обратного прокси-сервера.

Использование YARP позволяет избавиться от необходимости настраивать CORS для каждого сервиса. CORS (Cross-Origin Resource Sharing) – это механизм, реализованный в браузерах для безопасности, который позволяет или запрещает веб-страницам делать запросы к серверам с домена, отличного от того, с которого была загружена страница [14]. В случае, когда сервисы находятся за Gateway, использующим YARP, можно настроить CORS только для него и отключить на внутренних сервисах. Кроме того, для направления запросов к разным сервисам можно использовать префиксы, что делает обращение к ним из клиентского приложения удобнее, а масштабирование системы (добавление новых сервисов, изменение внутренней топологии) проще. Фрагмент файла Gateway\appsettings.json – пример настройки пути для сервиса UserService представлен в листинге 8.

Листинг 8 – Пример настройки пути с помощью YARP

```
{
  "ReverseProxy": {
    "Routes": {
      "user-service": {
        "ClusterId": "user-cluster",
        "Match": {
          "Path": "/usr-srv/{**catch-all}",
          "Methods": ["GET", "POST", "PUT", "DELETE"]
        },
        "Transforms": [
          { "PathRemovePrefix": "/usr-srv" }
        ]
      },
    },
  },
  "Clusters": {
    "login-cluster": {
      "Destinations": {
        "login": {
```


проекта представлена в источнике [10]. В листинге 9 представлен метод `RenewOperatorsList` класса `AdminProfileClass`, обновляющий свойство `MyOperators`, которое содержит список всех операторов данного администратора.

Листинг 9 – Обновление свойства `MyOperators` класса `AdminProfileClass`

```
public async Task RenewOperatorsList(HttpClient http)
{
    if (IdAdmin <= 0)
    {
        throw new InvalidOperationException("Id не
инициализирован");
    }

    if (http == null) throw new
InvalidOperationException("HttpClient не инициализирован");
    try
    {
        var url = $"/usr-srv/admin/{IdAdmin}/operators";
        var operators = await
http.GetFromJsonAsync<List<OperatorProfileClass>>(url);
        MyOperators = operators?.OrderBy(o =>
o.OperatorProfileCreated).ToList() ?? [];
    }
    catch
    {
        MyOperators = new List<OperatorProfileClass>();
        throw;
    }
}
```

На рисунке 22 представлена диаграмма последовательности действий на примере вызова метода `RenewOperatorsList` для получения всех операторов конкретного администратора.

На рисунке 22 показаны два сценария:

- успешный запрос списка операторов;
- ошибка при отсутствии администратора с указанным `id` в базе данных.

Система корректно обрабатывает оба случая, возвращая HTTP-код 200 или 404.

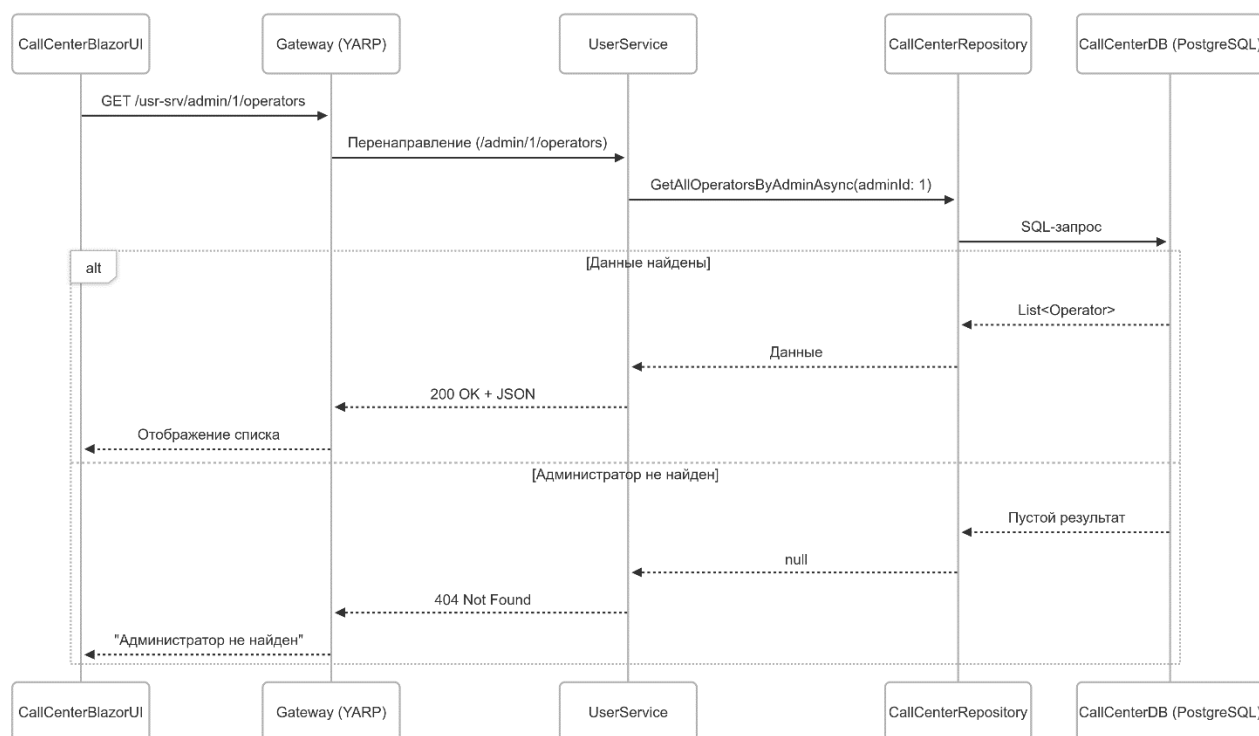


Рисунок 22 – Диаграмма последовательности действий для получения всех операторов администратора

Для того, чтобы пользователь не терял данные своего профиля при перезагрузке страницы, в проект CallCenterBlazorUI был добавлен сервис Local Storage с помощью библиотеки Blazored.LocalStorage. В Blazor WASM Local Storage позволяет сохранять данные на стороне клиента (в браузере) после обновления страницы [15]. В листинге 10 представлен фрагмент кода главной программы Program.cs – регистрация и настройка сервиса Local Storage для сохранения информации при успешной авторизации администратора.

Листинг 10 – Регистрация и настройка сервиса Local Storage

```

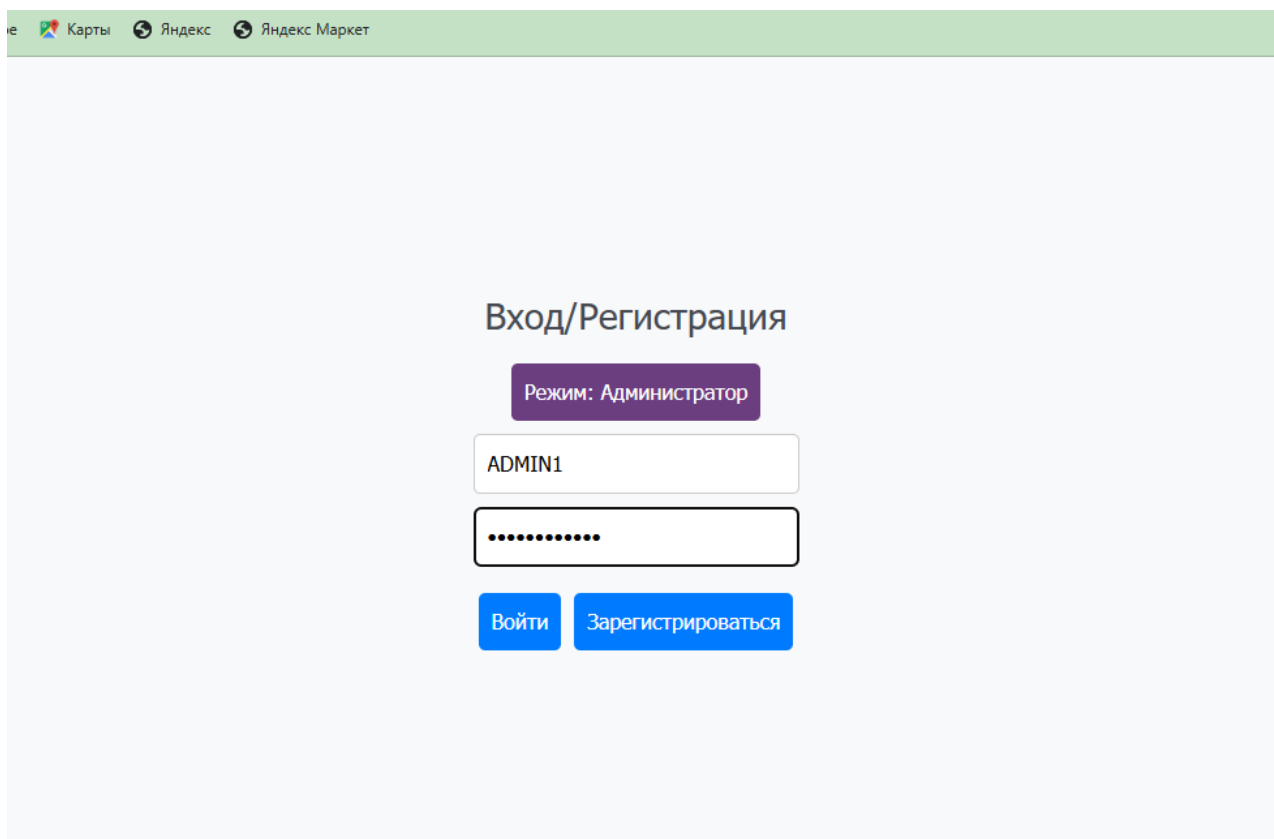
builder.Services.AddBlazoredLocalStorage();

builder.Services.AddScoped<AdminProfileClass>(sp =>
{
    var localStorage =
sp.GetRequiredService<ILocalStorageService>();
    var http = sp.GetRequiredService<HttpClient>();
    var profile = new AdminProfileClass();
    profile.InitializeServices(localStorage, http);
    return profile;
});
  
```

При загрузке приложения происходит проверка авторизации. После выхода пользователя из аккаунта данные удаляются из локального хранилища. Ссылка на репозиторий с полным кодом программы, а также всех классов для хранения состояния объектов системы представлена в источнике [10].

Для отображения данных и предоставления их пользователю в удобном виде были разработаны различные страницы. При их создании в качестве ориентира использовались макеты страниц, которые были представлены в разделе 2.3 – «Проектирование графического интерфейса».

При загрузке приложения неавторизованный пользователь видит экран ввода данных для входа в систему (рисунок 23).



Вход/Регистрация

Режим: Администратор

ADMIN1

.....

Войти Зарегистрироваться

Рисунок 23 – Страница входа в приложение

После успешной авторизации пользователь перенаправляется на страницу профиля (рисунок 24). В профиле он может видеть данные своего аккаунта, а также список закрепленных за ним операторов.

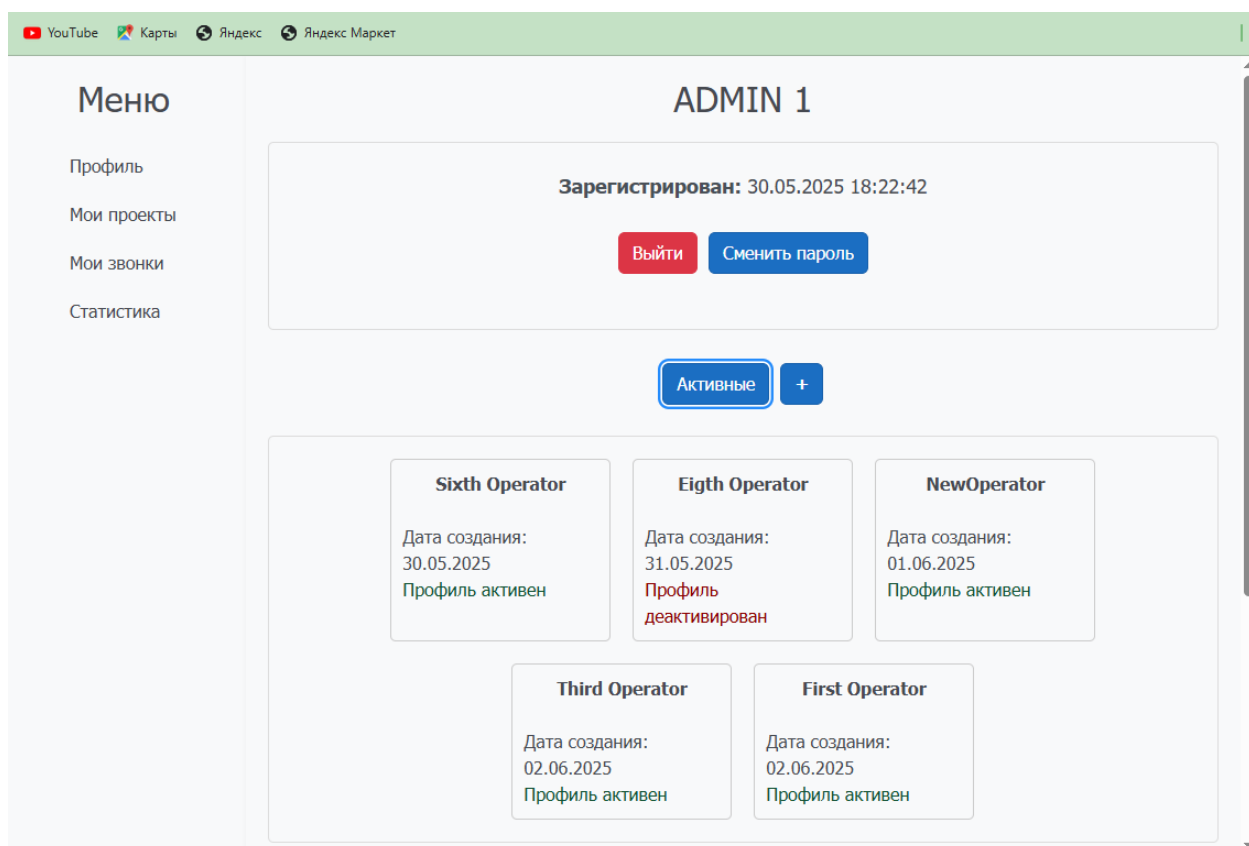


Рисунок 24 – Страница профиля администратора

При необходимости пользователь может сменить свой пароль, для этого нужно нажать на соответствующую кнопку на странице профиля. Откроется модальное окно для редактирования пароля (рисунок 25).

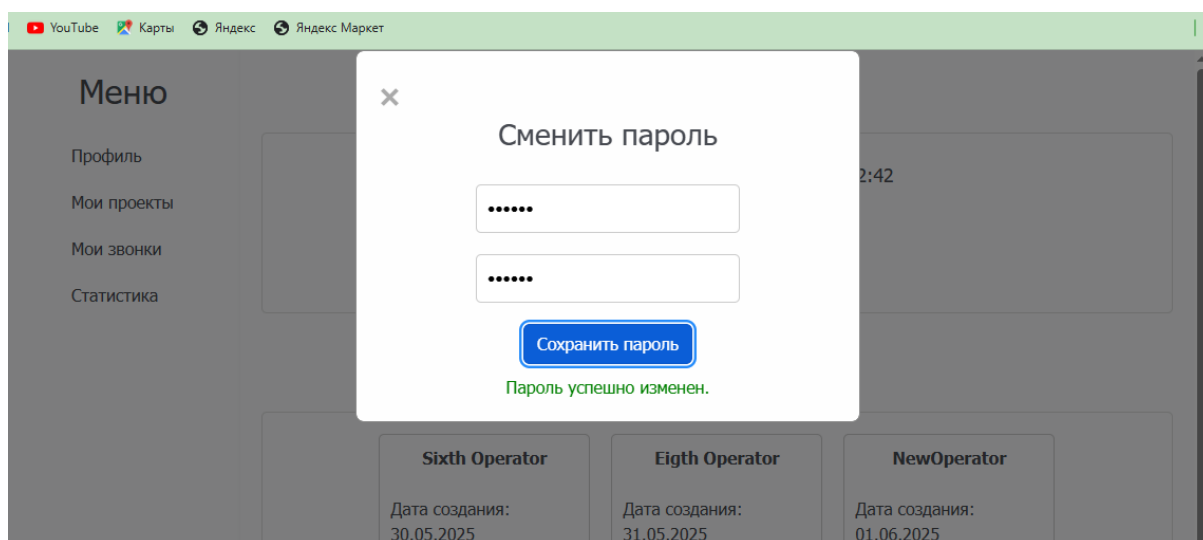


Рисунок 25 – Редактирование пароля администратора

При нажатии на кнопку «+» администратор может создать новый профиль оператора (рисунок 26).

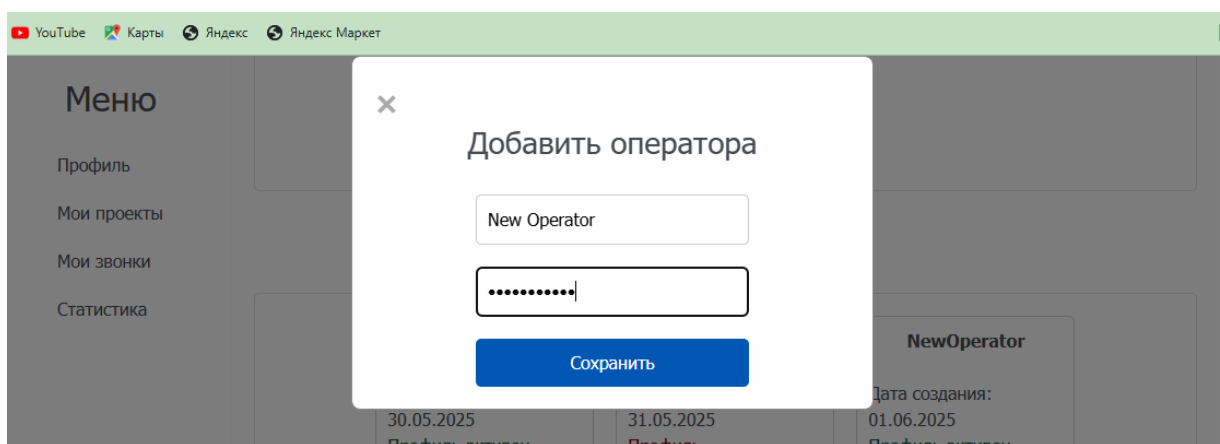


Рисунок 26 – Добавление нового оператора

При нажатии на блок, отображающий краткую информацию об операторе, пользователь попадает на страницу с подробной информацией об этом операторе (рисунок 27). На данной странице администратор может при необходимости деактивировать аккаунт оператора, изменить имя, а также увидеть, в каких действующих проектах задействован сотрудник.

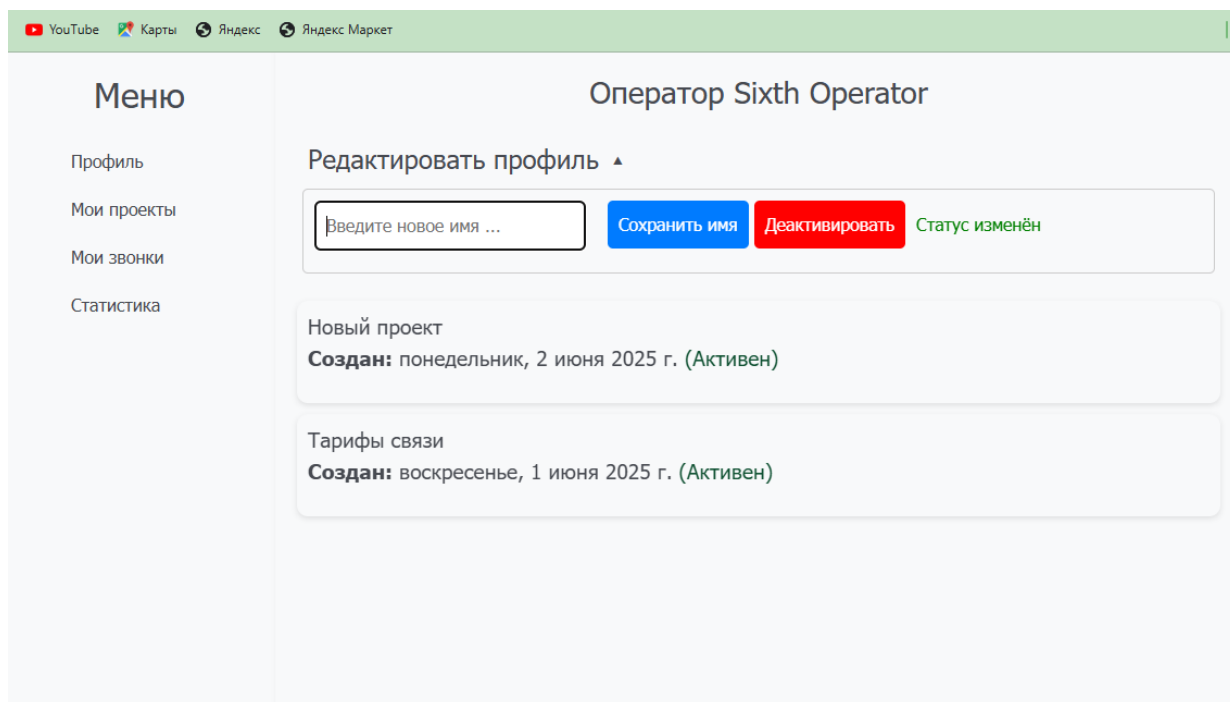


Рисунок 27 – Подробная информация об операторе

При нажатии на блок, отображающий краткую информацию о проекте, администратор может перейти на страницу проекта (рисунок 30). На данную страницу можно также прийти иным путём.

При нажатии на кнопку «Мои проекты» в боковом меню пользователь

попадает на страницу со списком созданных им проектов (рисунок 28). Здесь он может выбрать в выпадающем списке категорию проектов, которую хочет просмотреть (активные, приостановленные или закрытые).

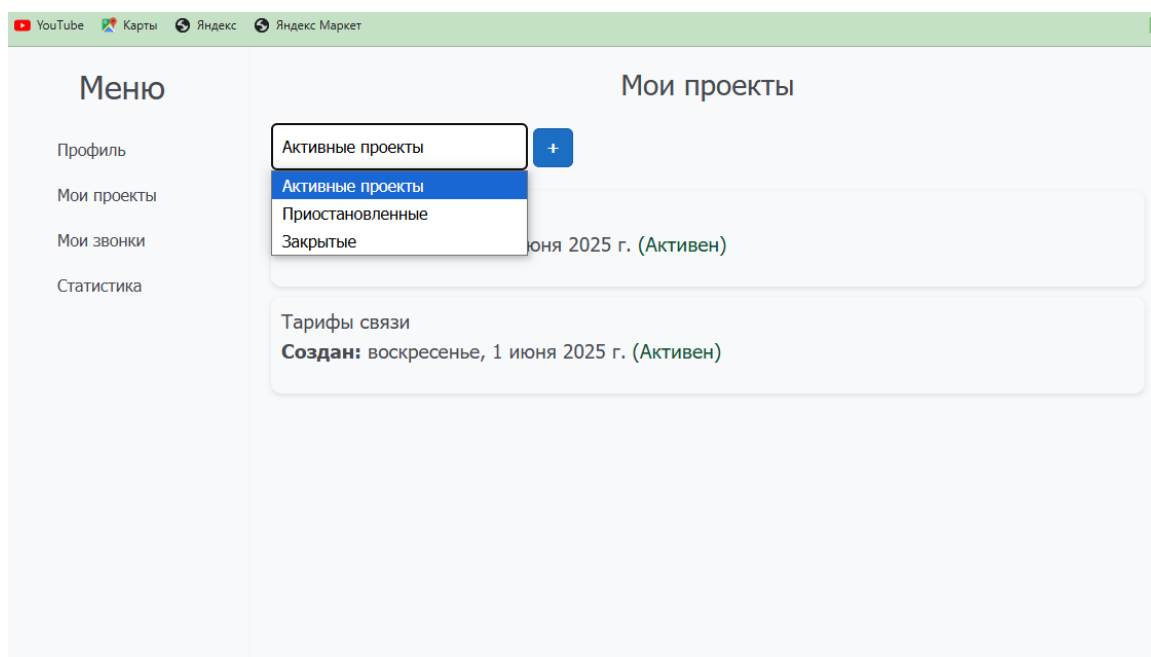


Рисунок 28 – Список проектов администратора

При нажатии на кнопку «+» откроется модальное окно добавления нового проекта (рисунок 29).

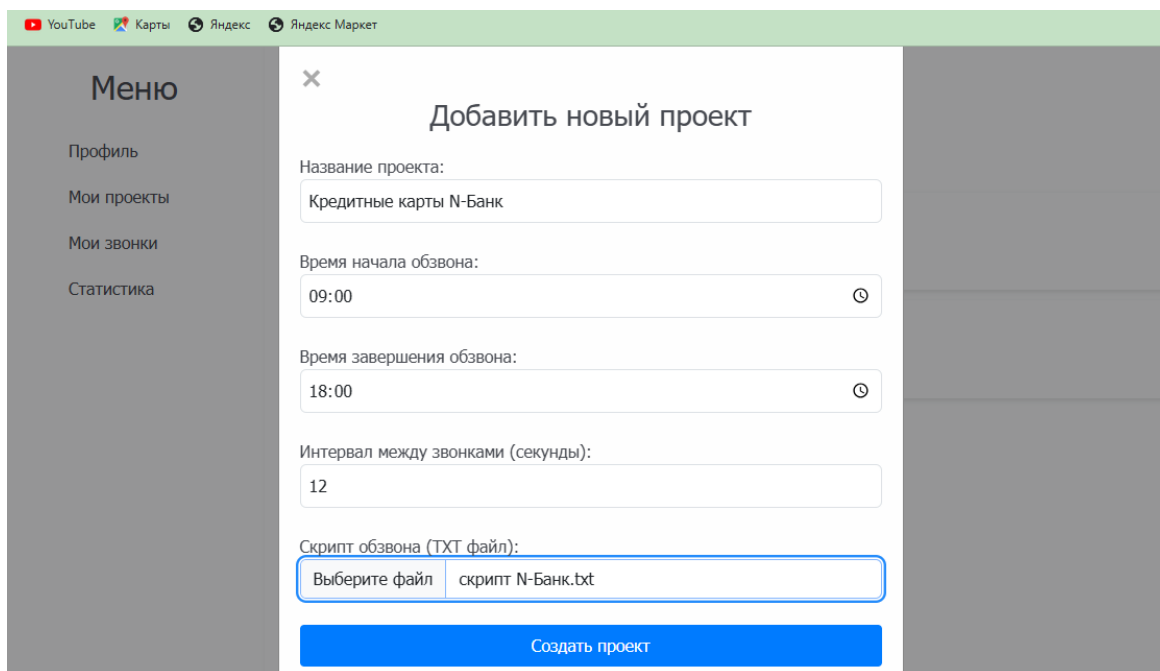


Рисунок 29 – Добавление нового проекта

Здесь администратор должен указать основные данные проекта, после чего

сохранить его. Далее он появится в списке. По умолчанию вновь созданный проект имеет статус «Приостановлен».

При нажатии на блок с краткой информацией о проекте пользователь попадает на страницу проекта (рисунок 30).

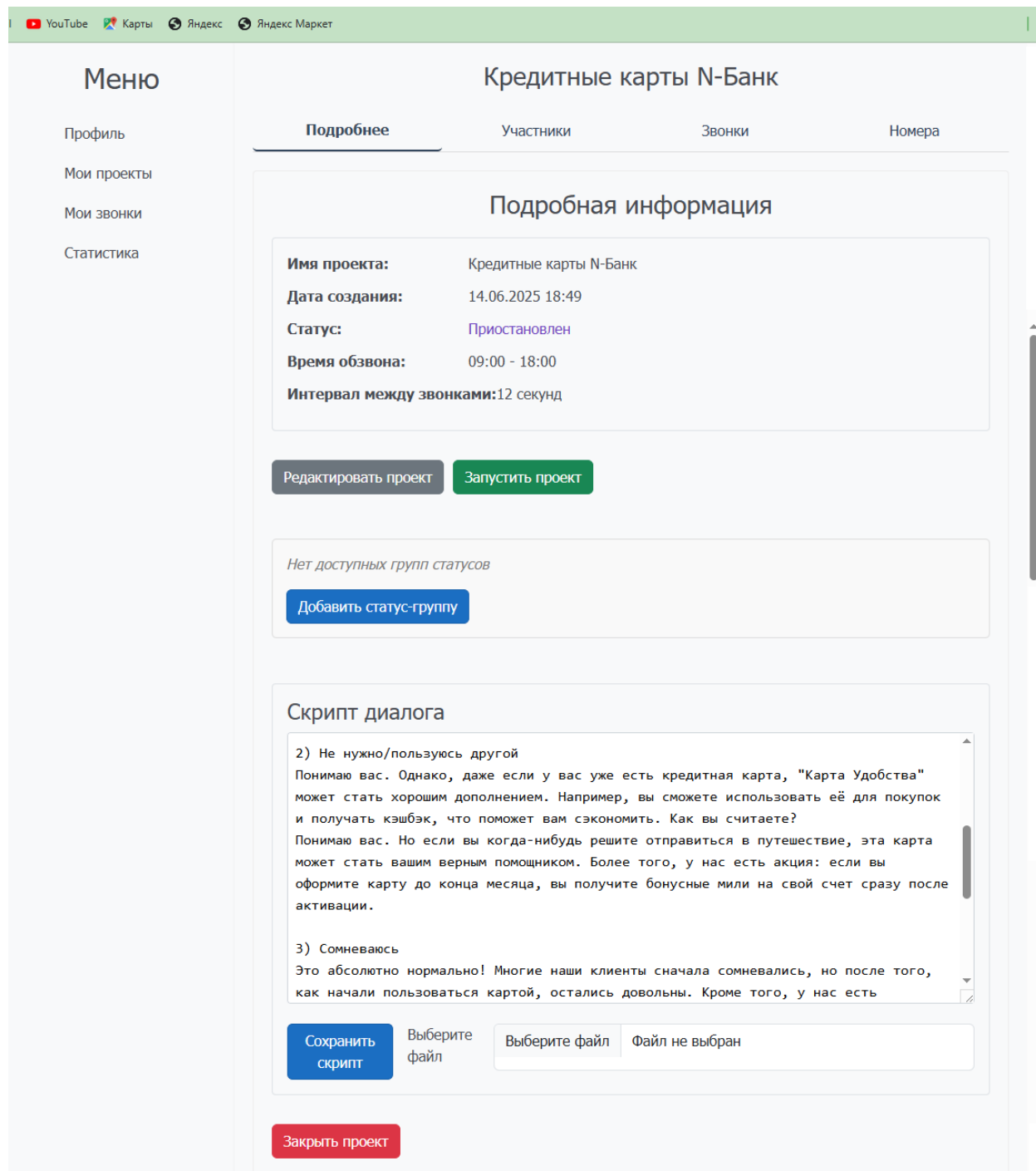


Рисунок 29 – Основная информация о проекте

Здесь он может редактировать основные данные, назначать и откреплять операторов, а также добавлять базу номеров клиентов, подлежащих обзвону. Для

этих целей внутри страницы организовано горизонтальное меню с переключаемыми панелями контента.

При переходе на страницу редактирования проекта по умолчанию открыта панель с основной информацией. С помощью этой панели администратор редактировать основные данные проекта (рисунок 31), редактировать загруженный скрипт диалога или загрузить новый из файла.

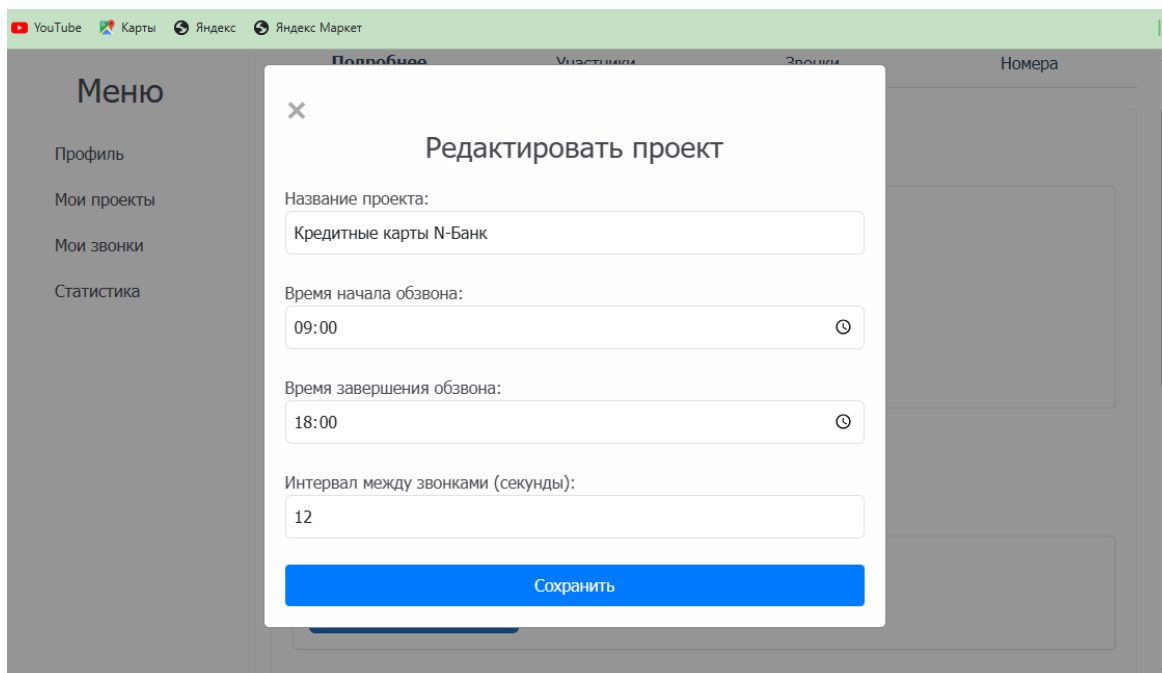


Рисунок 31 – Редактирование основных данных проекта

Также он может добавлять, удалять и редактировать статусы и группы статусов. Модальное окно добавления группы статусов представлено на рисунке 32.

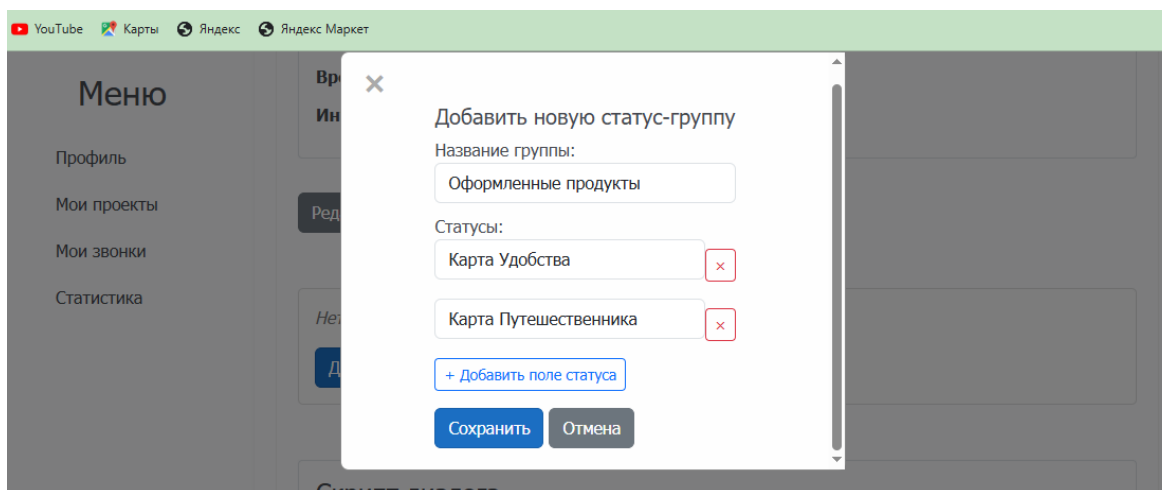


Рисунок 32 – Добавление группы статусов

Отображение добавленных к проекту статусов итогов звонка представлено на рисунке 33.

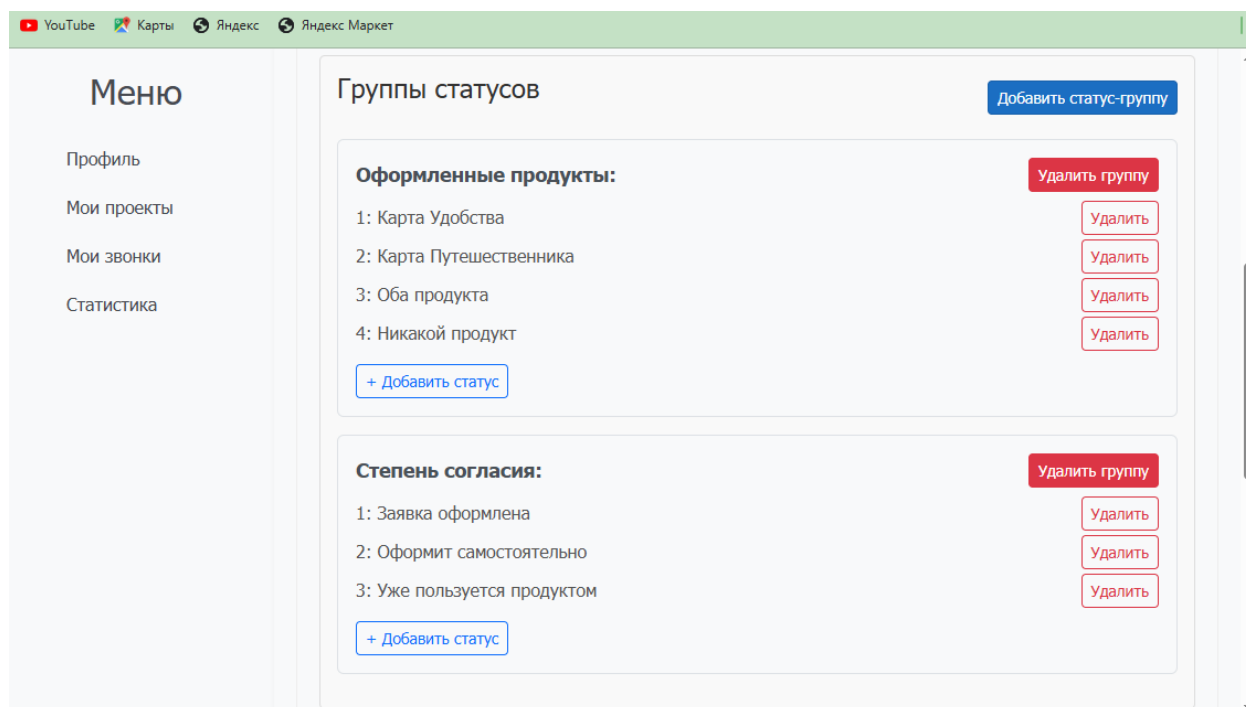


Рисунок 33 – Добавленные статусы итогов звонка

Для того, чтобы отредактировать имя статуса или группы статусов, пользователь может нажать на текущее название, после чего откроется модальное окно для его редактирования (рисунок 34).

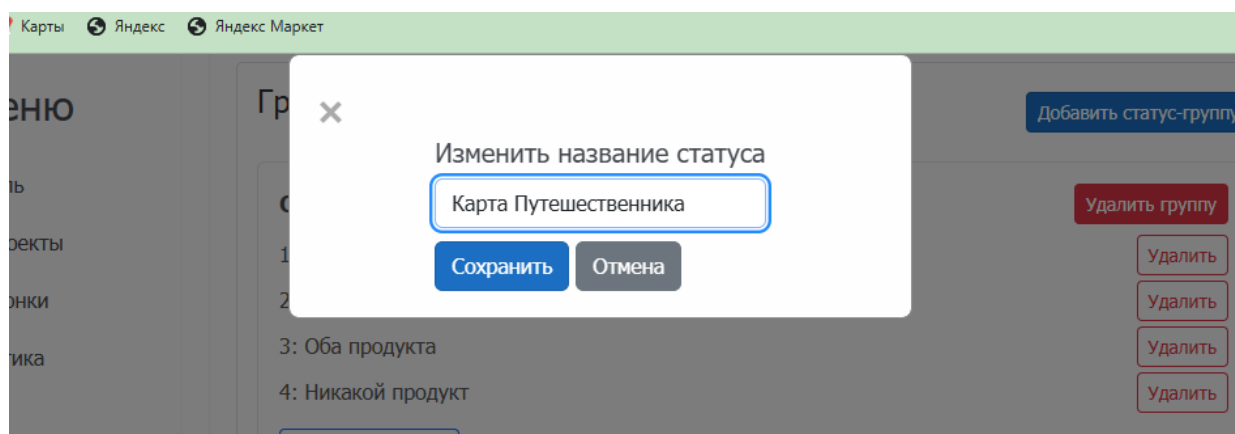


Рисунок 34 – Редактирование имени статуса

При запуске проекта поля с информацией о нем доступны только для просмотра, так как редактировать можно только приостановленный проект, по которому не осуществляется обзвон клиентов в текущий момент времени. Администратор также может закрыть проект, после чего он станет недоступен

для запуска и редактирования (рисунок 35).

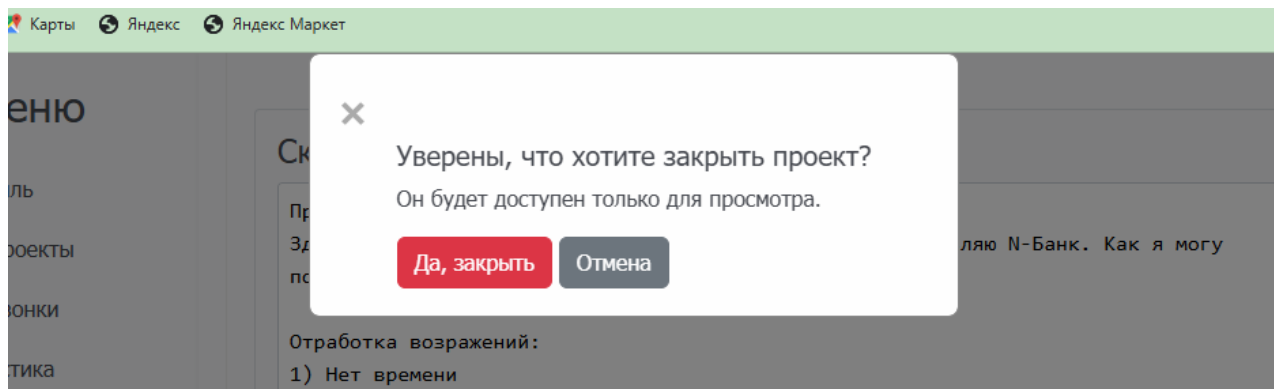


Рисунок 35 – Закрытие проекта

Для просмотра назначенных на проект операторов нужно открыть панель «Участники» (рисунок 36).

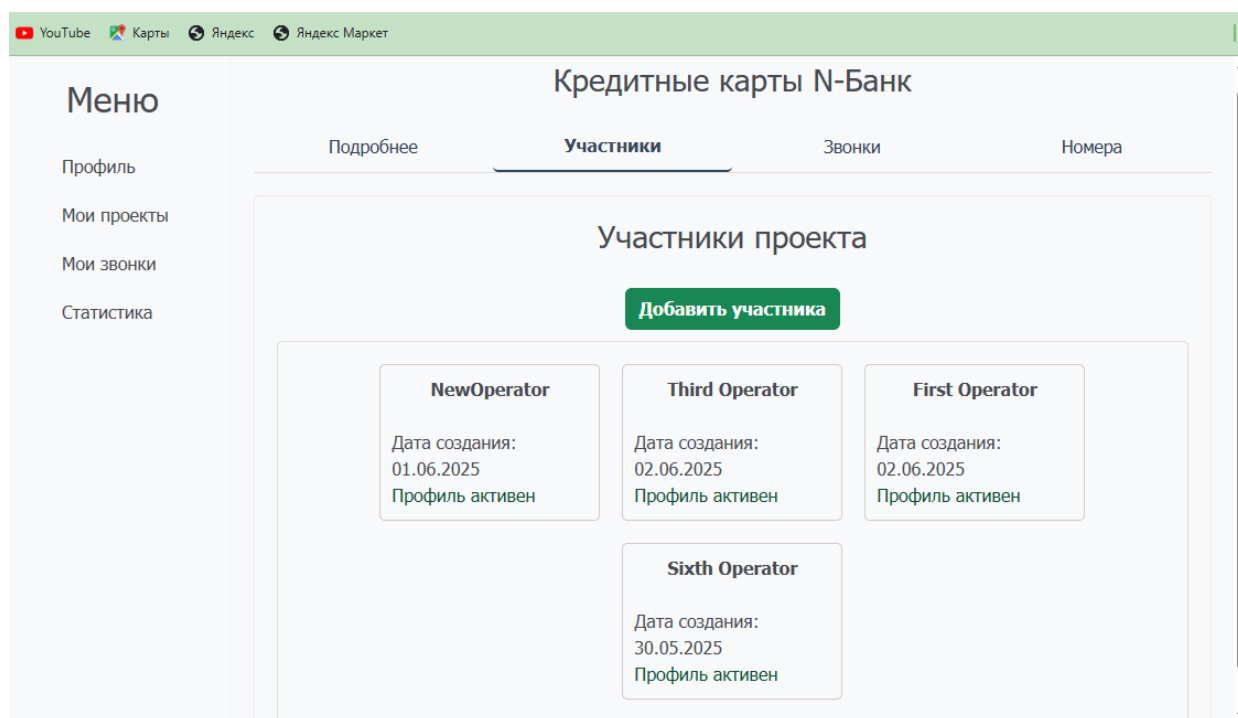


Рисунок 36 – Участники проекта

Для добавления новых участников необходимо нажать на соответствующую кнопку, после чего в окне отобразится список всех доступных для добавления операторов (рисунок 37). Для того, чтобы открепить оператора от проекта, нужно нажать на соответствующий блок и в появившемся окне выбрать опцию «Открепить от проекта» (рисунок 38).

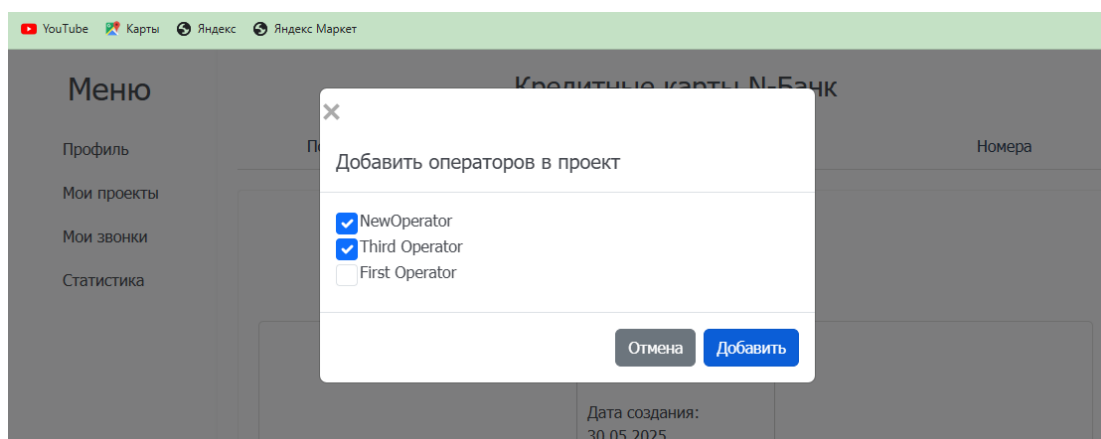


Рисунок 37 – Назначение операторов на проект

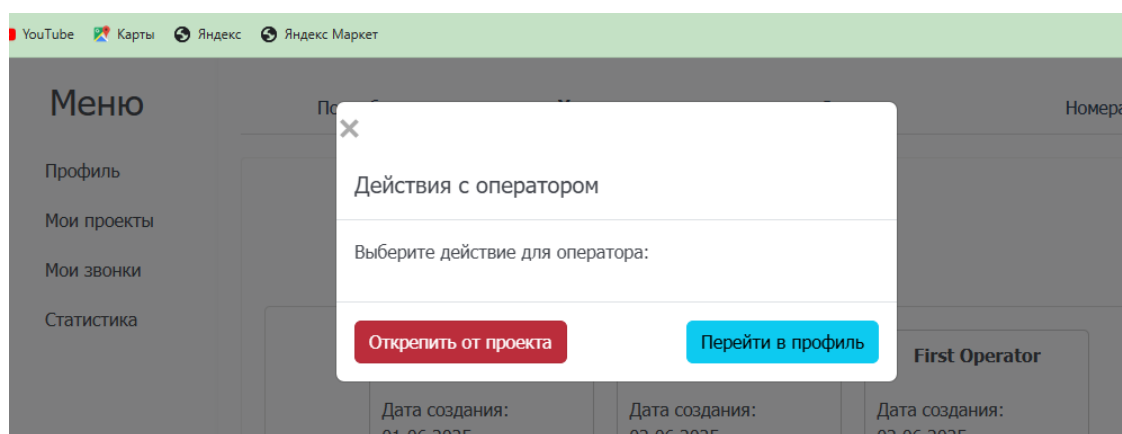


Рисунок 38 – Открепление оператора от проекта

Для того, чтобы добавить базу клиентских номеров, нужно перейти на панель «Номера» и выбрать Excel-файл для загрузки (рисунок 39).

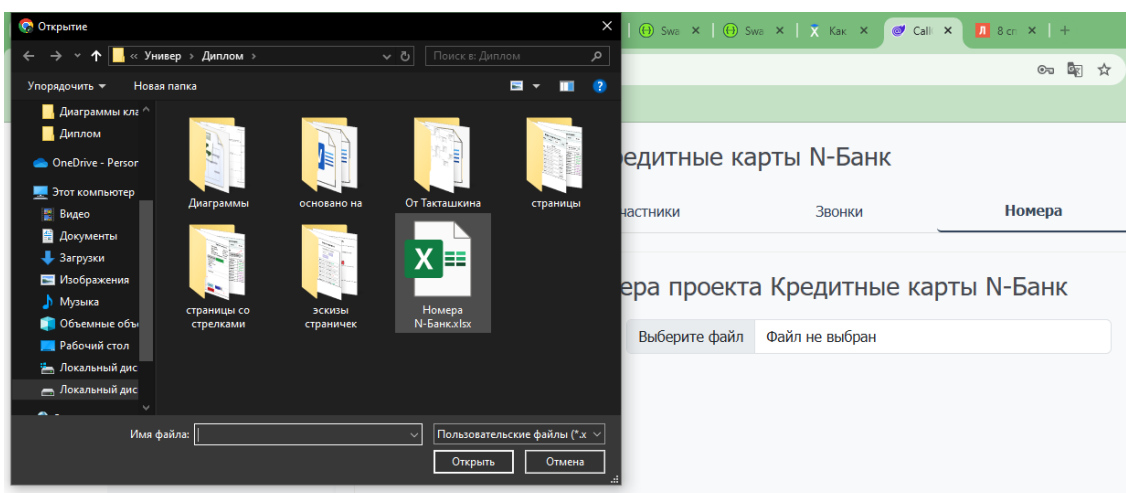


Рисунок 39 – Добавление номеров клиентов

Результат загрузки базы телефонных номеров клиентов, подлежащих обзвону, показан на рисунке 40.

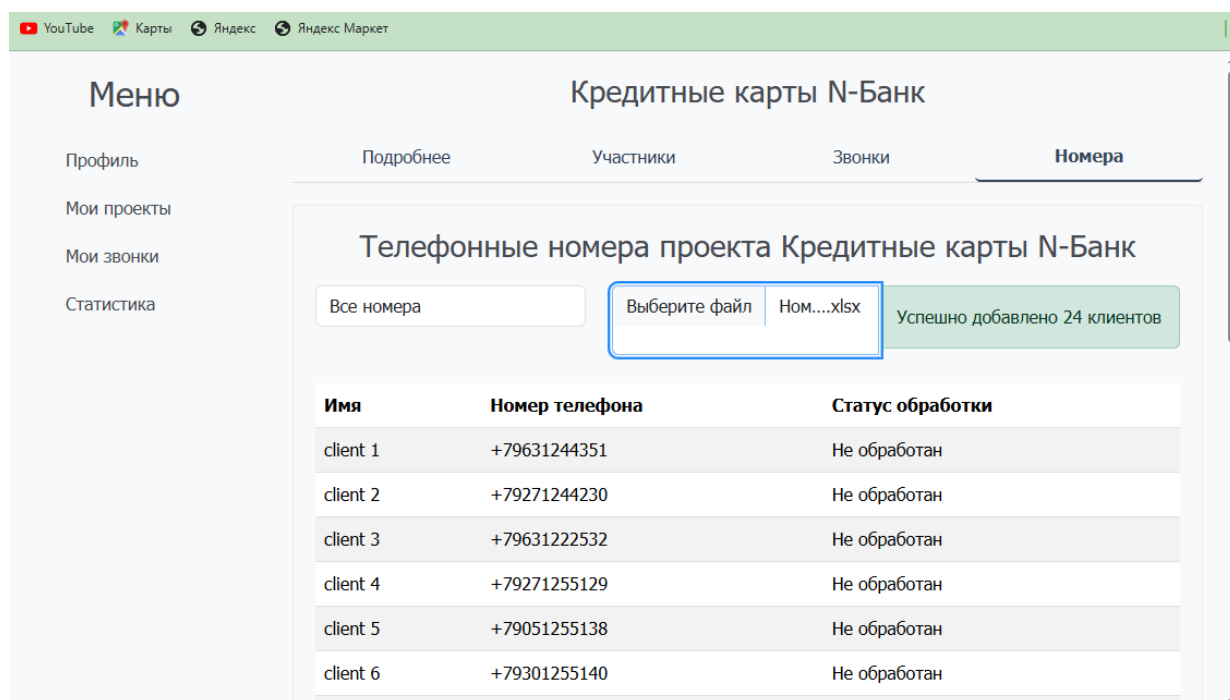


Рисунок 40 – Результат загрузки базы телефонных номеров

При загрузке все номера имеют статус «Не обработан». По мере того, как клиентская база будет обзваниваться, номера в системе будут приобретать иные статусы («Обработан», «Неверный номер», «Перезвонить»). На странице предусмотрена фильтрация по данному параметру.

Для того, чтобы просмотреть подробную информацию о звонках, совершенных операторами в рамках данного проекта, необходимо перейти на панель «Звонки» (рисунок 41).

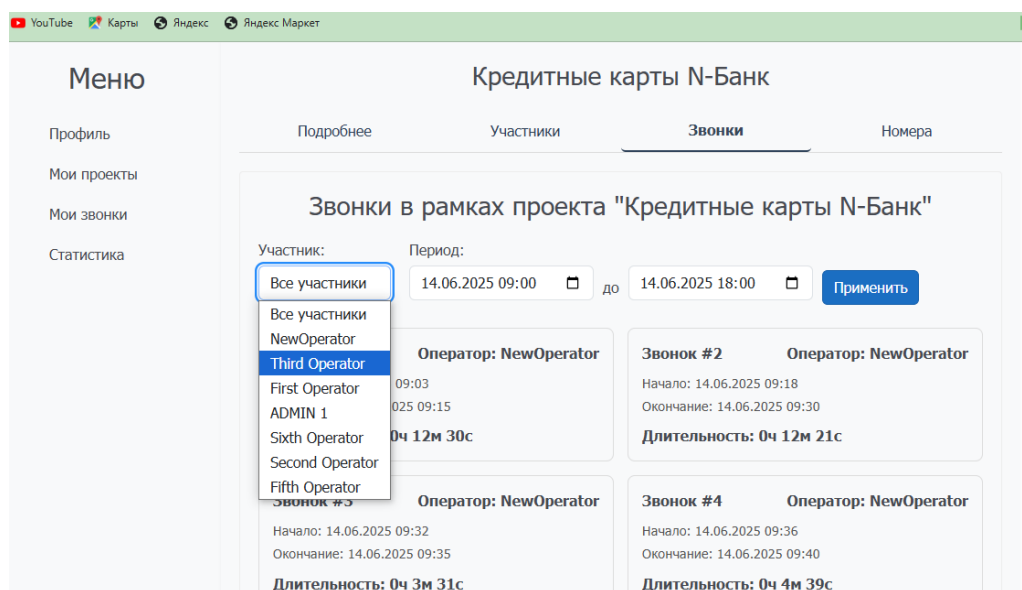


Рисунок 41 – Звонки в рамках проекта

На данной странице предусмотрена фильтрация по дате совершенного звонка. При открытии страницы по умолчанию выставлена текущая дата, а временной интервал по умолчанию соответствует временному интервалу работы данного проекта. На странице просмотра также можно отфильтровать записанные звонки по определенному сотруднику.

При нажатии на блок, отображающий краткую информацию о звонке, происходит переход на страницу с подробной информацией о нем (рисунок 42). На данной странице для удобства навигации области редактирования различных параметров сворачиваются. Для этого необходимо нажать на символ треугольника.

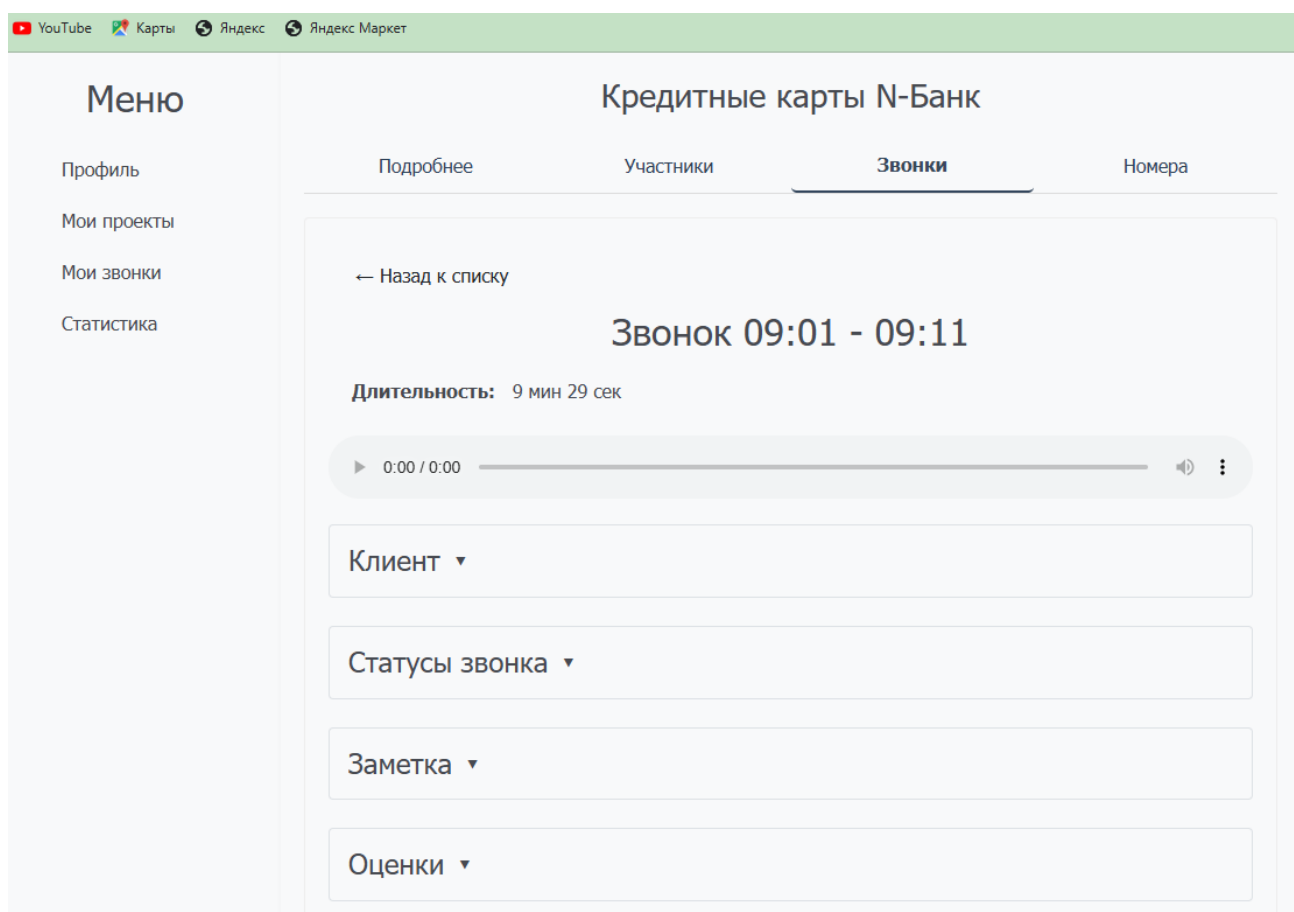


Рисунок 42 – Страница звонка

На данной странице администратор может оценить звонок (рисунок 43) или отредактировать выставленные оператором статусы завершения диалога (рисунок 44).

Меню

- Профиль
- Мои проекты
- Мои звонки
- Статистика

Оценки ▲

Речь:

4

Следование скрипту:

5

Безошибочность:

5

Корректность статусов:

3

Сохранить оценки

Рисунок 43 – Оценка звонка

Меню

- Профиль
- Мои проекты
- Мои звонки
- Статистика

Статусы звонка ▲

Оформленные продукты

Карта Удобства

Степень согласия

Оформит самостоятельно

Сохранить статусы

Рисунок 44 – Редактирование статусов

При необходимости администратор может изменить статус обработки номера клиента (рисунок 45), а также оставить комментарий о данном звонке для оператора.

Мои проекты

- Мои проекты
- Мои звонки
- Статистика

Клиент ▲

Имя клиента: client 4

Номер телефона: +79271255129

Статус клиента:

- Обработан
- Не обработан
- Обработан
- Неверный номер
- Перезвонить

Рисунок 45 – Редактирование статуса обработки клиента

Чтобы посмотреть статистику операторов, администратор должен перейти на вкладку «Статистика» в боковом меню, после чего произойдет переход на соответствующую страницу (рисунок 46).

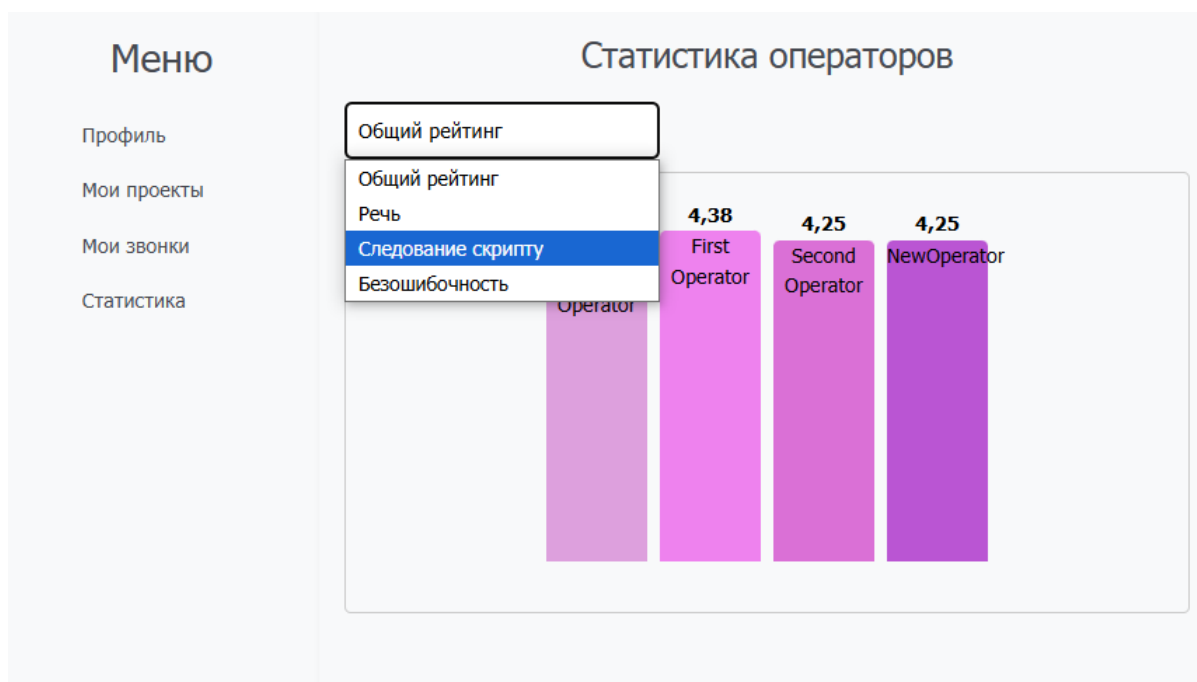


Рисунок 46 – Статистика операторов

Таким образом была реализована клиентская часть приложения «Автоматизированное рабочее место администратора отдела информирования клиентов организации», созданы страницы, предоставляющие доступ к функционалу системы.

4 Функциональное тестирование приложения

При тестировании функционала разрабатываемой системы использовались методы как ручного, так и автоматизированного тестирования. Для автоматизированного тестирования был использован фреймворк для создания unit-тестов NUnit.

4.1 Ручное тестирование

Ручное тестирование – это процесс поиска ошибок в программе без помощи специального программного обеспечения. Тестировщик при этом имитирует действия реального пользователя. Ручное тестирование может дополнять автоматизированное, чтобы обеспечить более глубокую проверку [16].

Пример результатов тестирования представлен в таблице 5. Скриншоты проведенных тестов представлены в приложении В.3.

Таблица 5 – Результаты ручного тестирования

Вариант использования	Состав теста	Тестовые данные	Результат
Войти в систему	Ввести корректные данные	Логин: ADMIN 1 QA5hWSZX@	Тест пройден. Осуществлен вход администратора в систему (рисунок В.1).
	Ввести неверные данные	Логин: ADMIN 1 QA5hZX@	Тест пройден. Выведено уведомление о неверных данных (рисунок В.2)

Продолжение таблицы 5

Вариант использования	Состав теста	Тестовые данные	Результат
Создать проект	Попытка сохранения проекта	Название: Кредитные карты N-Банк; Время начала работы: 09:00; Время окончания работы: 18:00; Интервал: 12 секунд; Скрипт: файл Скрипт N-Банк.txt	Тест пройден. Проект создан (рисунок В.3).
Редактировать основные данные проекта	Попытка сохранить изменения	Интервал: 10 секунд.	Тест пройден. Изменения сохранены. (рисунок В.4, В.5).
Приостановить проект	Попытка приостановить проект в рабочее время	Время системы: 12:30; Время работы проекта: 09:00 – 18:00.	Тест пройден. Выведено сообщение об отказе приостановки проекта. (рисунок В.6).
	Попытка приостановить проект в нерабочее время	Время системы: 18:30; Время работы проекта: 09:00 – 18:00.	Тест пройден. Проект приостановлен (рисунок В.7).

Продолжение таблицы 5

Вариант использования	Состав теста	Тестовые данные	Результат
Добавить оператора в проект	Попытка добавления доступного оператора в проект	Оператор не добавлен в проект.	Тест пройден. Оператор добавлен в проект (рисунок В.8, В.9).
Открепить оператора от проекта	Попытка открепления оператора от проекта	Оператор ранее был добавлен в проект.	Тест пройден. Оператор откреплен от проекта и снова в списке доступных для прикрепления (рисунок В.10, В.11)
Загрузить клиентскую базу	Выбрать файл верного формата	Файл: Номера N- Банк.xlsx	Тест пройден. Клиентская база добавлена (рисунок В.12).
	Выбрать файл неверного формата	Файл: Скрипт N-Банк.txt	Тест пройден. Выведено сообщение об ошибке формата файла (рисунок В.13).
Выставить оценки звонку	Попытка сохранить оценки	Речь: 4. Следование скрипту: 5. Безошибочность: 5. Корректность: 3.	Тест пройден. Оценки сохранены (рисунок В.14).

Окончание таблицы 5

Вариант использования	Состав теста	Тестовые данные	Результат
Оставить заметку о звонке	Попытка сохранить заметку	Текст заметки: «Стоит поработать над речью, будьте внимательнее при выставлении статусов.»	Тест пройден. Заметка сохранена (рисунок В.15).
Редактировать статус	Попытка сохранить новый статус	Старый статус: «Не заинтересовался». Новый статус: «Оформит самостоятельно».	Тест пройден. Изменения сохранены (рисунок В.16).

Ручное тестирование показало корректность работы протестированных функций приложения.

4.2 Автоматизированное тестирование

Попарное тестирование или попарное покрытие – это метод тестирования, который фокусируется на проверке всех возможных пар значений входных параметров. Основная его идея заключается в том, что большинство дефектов в программном обеспечении возникает из-за взаимодействия между парами параметров, а не из-за их индивидуальных значений. Этот метод позволяет сократить количество тестов в сравнении с полным перебором всех возможных комбинаций входных данных.

В данной дипломной работе попарное тестирование проводилось с использованием инструмента для автоматизации процесса попарного

тестирования Pairwise. Он генерирует набор тестовых случаев, охватывающих все возможные пары значений для заданного набора параметров [17].

Для тестирования функционала добавления нового оператора определим основные случаи, которые необходимо протестировать.

Для поля имени:

- имя пользователя содержит цифры;
- имя пользователя равно null;
- имя пользователя содержит символы;
- имя пользователя корректно.

Для поля пароля:

- содержит верные символы, длинный;
- содержит верные символы, короткий;
- верная длина, неверный формат;
- верная длина, верный формат.

Файл для инструмента Pairwise приведен на рисунке 47.

	×	×		
	Name	Password	+	
×	Row 1	Ир1на	Sfth46&#ghdj666rfgGG:	+
×	Row 2	null	ρ2#8H	+
×	Row 3	Ирин@	hjhjhjhjhjh	+
×	Row 4	Ирина	D56f34?g#21	+
+	+	+	+	

Рисунок 47 – Файл для инструмента Pairwise

Скриншот результатов выполнения – тест кейсов, приведен на рисунке 48.

	Name	Password
1	Ир1на	Sfth46&#ghdj666rfgGGsa
2	Ир1на	p2#8H
3	Ир1на	hjhjhjhjhjhjh
4	Ир1на	D56f34?g#21
5	null	p2#8H
6	null	hjhjhjhjhjhjh
7	null	D56f34?g#21
8	null	Sfth46&#ghdj666rfgGGsa
9	Ирин@	hjhjhjhjhjhjh
10	Ирин@	D56f34?g#21
11	Ирин@	Sfth46&#ghdj666rfgGGsa
12	Ирин@	p2#8H
13	Ирина	D56f34?g#21
14	Ирина	Sfth46&#ghdj666rfgGGsa
15	Ирина	p2#8H
16	Ирина	hjhjhjhjhjhjh

Рисунок 48 – Тест-кейсы, сгенерированные инструментом Pairwise

Получено комбинаторных переборов данных – 16 наборов, составлено тест-кейсов – 16. Для проведения автоматизированного тестирования был использован фреймворк NUnit – открытый фреймворк для модульного тестирования, созданный для платформы .NET [18,19]. В листинге 11 приведен код тестируемого метода.

Листинг 11 – Тестируемый код

```
public string? ValidateInputs()
{
    // Проверка на пустоту
    if (string.IsNullOrEmpty>Password) ||
string.IsNullOrEmpty>Username))
    {
        return "Заполните поля...";
    }

    // Проверка имени
    if (Username.Length > 60 || !Regex.IsMatch(Username, @"^[a-zA-
Za-яA-ЯёЁs]+$"))
    {
        return "Имя должно содержать только буквы и пробелы, и не
превышать 60 символов.";
    }

    // Проверка пароля
    if (Password.Length < 8 || Password.Length > 20 ||
        Regex.IsMatch>Password, @"[a-яA-ЯёЁ]" ) ||
```

```

        !Regex.IsMatch(Password, @"[A-Z]") ||
        !Regex.IsMatch(Password, @"[0-9]") ||
        !Regex.IsMatch(Password, @"[@&?$#%^*!() ]"))
    {
        return "Неверный формат пароля (Длина 8-20 символов,
должен содержать спецсимволы и цифры).";
    }

    return null; // Все проверки пройдены успешно
}

```

В листинге 12 приведен код тест-кейсов, выявленных в предыдущем подразделе, и класса, содержащего необходимые методы.

Листинг 12 – Тест-кейсы

```

[TestFixture]
public class OperatorTests
{
    [TestFixture]
    public class OperatorsTests
    {
        private Operators _operators;

        [SetUp]
        public void Setup()
        {
            _operators = new Operators();
        }

        [TestCase("Ир1на", "Sfth46&#ghdj666rfgGGsa", "Имя должно
содержать только буквы и пробелы, и не превышать 60 символов.")]
        [TestCase("Ир1на", "р2#8н", "Имя должно содержать только буквы и
пробелы, и не превышать 60 символов.")]
        [TestCase("Ир1на", "hjhjhjhjhjhjh", "Имя должно содержать
только буквы и пробелы, и не превышать 60 символов.")]
        [TestCase("Ир1на", "D56f34?g#21", "Имя должно содержать
только буквы и пробелы, и не превышать 60 символов.")]
        [TestCase(null, "р2#8н", "Заполните поля...")]
        [TestCase(null, "hjhjhjhjhjhjh", "Заполните поля...")]
        [TestCase(null, "D56f34?g#21", "Заполните поля...")]
        [TestCase(null, "Sfth46&#ghdj666rfgGGsa", "Заполните
поля...")]
        [TestCase("Ирин@", "hjhjhjhjhjhjh", "Имя должно содержать
только буквы и пробелы, и не превышать 60 символов.")]
        [TestCase("Ирин@", "D56f34?g#21", "Имя должно содержать
только буквы и пробелы, и не превышать 60 символов.")]
        [TestCase("Ирин@", "Sfth46&#ghdj666rfgGGsa", "Имя должно
содержать только буквы и пробелы, и не превышать 60 символов.")]
        [TestCase("Ирин@", "р2#8н", "Имя должно содержать только
буквы и пробелы, и не превышать 60 символов.")]
        [TestCase("Ирина", "D56f34?g#21", null)]
    }
}

```

```

[TestCase("Ирина", "Sfth46&#ghdj666rfgGGsa", "Неверный
формат пароля (Длина 8-20 символов, должен содержать спецсимволы и
цифры).")]
[TestCase("Ирина", "p2#8H", "Неверный формат пароля (Длина
8-20 символов, должен содержать спецсимволы и цифры).")]
[TestCase("Ирина", "hjhjhjhjhjhjh", "Неверный формат пароля
(Длина 8-20 символов, должен содержать спецсимволы и цифры).")]

public void
ValidateInputs_ShouldReturnExpectedMessage(string username, string
password, string expectedMessage)
{
    // Arrange
    _operators.Username = username;
    _operators.Password = password;

    // Act
    var result = _operators.ValidateInputs();

    // Assert
    Assert.That(expectedMessage, Is.EqualTo(result));
}
}
}

```

На рисунке 50 приведены результаты запуска автоматизированного тестирования.

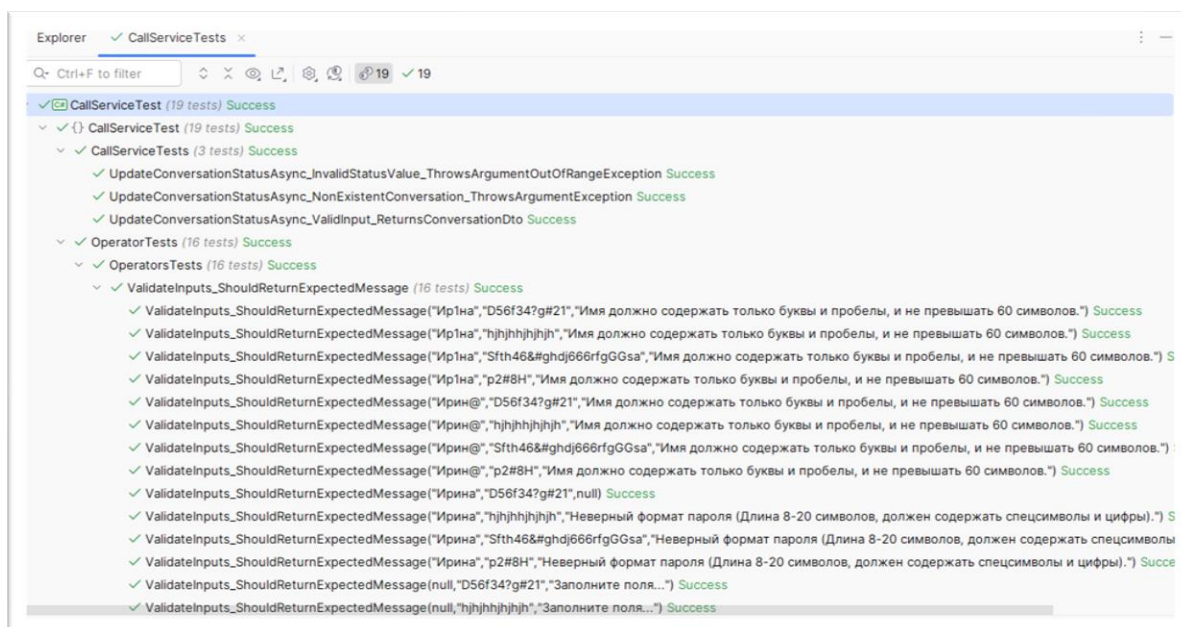


Рисунок 50 – Результаты автоматизированного тестирования

Успешно выполненное автоматизированное тестирование свидетельствует о том, что рассматриваемый блок кода работает корректно.

4.3 Анализ сложности кода модуля регистрации

Для измерения сложности кода были использованы метрики Холстеда и метрики размера программы.

В листинге 13 приведен код метода для валидации введенных данных нового сотрудника и метода добавления новой записи в базу данных, который обращается к соответствующему сервису системы. На его основании будет производиться расчет метрик.

Метрики размера программы – это числовые значения, которые используются для оценки сложности программного кода. Их расчет предполагает подсчет количества строк программного кода, а также числа строк с комментариями [20].

Листинг 13 – Метод валидации данных нового сотрудника и метод для добавления новой записи в базу данных

```
private async Task SaveOperator()
{
    var validationMessage = ValidateInputs();

    if (validationMessage != null)
    {
        _message = validationMessage;
        _messageClass = "error-message";
        return;
    }

    var adminId = await IdService.GetIdAsync();
    var url =
$"api/admin/{adminId}/operators?username={Uri.EscapeDataString(Use
rname)}&password={Uri.EscapeDataString>Password)}";
    var response = await HttpClient.PostAsync(url, null);

    if (response.IsSuccessStatusCode)
    {
        _message = "Оператор успешно добавлен.";
        _messageClass = "success-message";
        await LoadOperators(); // Перезагрузить основную страничку
    }
    else if (response.StatusCode == HttpStatusCode.Conflict)
    {
        _message = "Оператор с таким именем уже существует.";
        _messageClass = "error-message";
    }
}
```

```

        else
        {
            _message = $"Произошла ошибка при добавлении нового
оператора: {response.StatusCode.ToString()}";
            _messageClass = "error-message";
            Console.WriteLine($"Произошла ошибка при добавлении нового
оператора: {response.StatusCode.ToString()}");
        }
    }

public string? ValidateInputs()
{
    // Проверка на пустоту
    if (string.IsNullOrEmpty>Password) ||
string.IsNullOrEmpty(Username))
    {
        return "Заполните поля...";
    }

    // Проверка имени
    if (Username.Length > 60 || !Regex.IsMatch(Username, @"^[a-zA-
Za-яА-ЯЁЁs]+$"))
    {
        return "Имя должно содержать только буквы и пробелы, и не
превышать 60 символов.";
    }

    // Проверка пароля
    if (Password.Length < 8 || Password.Length > 20 ||
        Regex.IsMatch>Password, @"[a-яА-ЯЁЁ]") || // наличие
русских букв
        !Regex.IsMatch>Password, @"[A-Z]") || // минимум одна
заглавная буква
        !Regex.IsMatch>Password, @"[0-9]") || // минимум одна
цифра
        !Regex.IsMatch>Password, @"[!@&?${}%^*!()]") || // минимум
один спецсимвол
    {
        return "Неверный формат пароля (Длина 8-20 символов,
должен содержать спецсимволы и цифры).";
    }

    return null; // Все проверки пройдены успешно
}

```

Физическая строка – это каждая строка кода, включая пустые строки и строки с комментариями; логические строки – это строки, которые содержат исполняемый код или выражения, игнорируя пустые и строки с комментариями. Далее будет приведено вычисление метрик размера программы для

приведенного выше программного кода.

В данном фрагменте кода имеется:

- 53 физических строки;
- 39 логических строк;
- 9 строк-комментариев.

Метрики Холстеда представляют собой набор количественных показателей, разработанных Норманом Холстедом в 1977 году для оценки сложности программного кода. Эти метрики помогают разработчикам и понять качество и сложность программного обеспечения [21]. Для вычисления метрик Холстеда необходимо измерить четыре характеристики программы:

- число уникальных операторов;
- число уникальных операндов;
- общее число операторов;
- общее число операндов.

В таблицах 9 – 10 приведены словарь операторов и словарь операндов для рассматриваемого фрагмента кода.

Таблица 9 – Словарь операторов

Операторы	Число вхождений
=	10
.	17
==	1
()	24
{}	14
;	19
!=	1
	7
?	2
!	4
>	2
<	1
await	3
if	6
else if	1
else	1

Продолжение таблицы 9

Операторы	Число вхождений
return	5
var	4
ToString	2
n1 = 19	N1 = 124

Таблица 10 – Словарь операндов

Операнды	Число вхождений
_message	4
_messageClass	4
adminId	2
url	2
response	5
Password	8
Username	4
validationMessage	3
n2 = 8	N2 = 32

Таким образом, число уникальных операторов программы составляет $n1 = 19$, число уникальных операндов программы $n2 = 8$, общее число операторов в программе $N1 = 124$, общее число операндов в программе $N2 = 32$.

На основе метрик Холстеда можно вычислить ряд характеристик программы. Далее будут рассмотрены некоторые из них.

Расчёт словаря программы производится по формуле:

$$n = n1 + n2 = 8 + 19 = 27,$$

где $n1$ – количество операторов в программе, $n2$ – количество операндов программы.

Расчёт длины программы производится по формуле:

$$N = N1 + N2 = 124 + 32 = 156,$$

где $N1$ – число вхождений всех операторов в программе, $N2$ – число вхождений всех операндов программы.

Расчёт объёма программы производится по формуле:

$$V = N * \log_2 n = 156 * \log_2 27 = 741,78,$$

где N – длина программы, n – словарь программы.

Расчёт сложности программы производится по формуле:

$$D = (n1/2) * (N2/n2) = (19 / 2) * (32/ 14) = 25,5,$$

где $n1$ – количество операторов в программе, $n2$ – количество операндов программе, $N2$ – число вхождений всех операндов программе.

Расчёт оценки интеллектуальных усилий производится по формуле:

$$E = D * V = 37,93 * 900 = 18915,39,$$

где D – сложность программы, V – объем программы.

По результатам расчета метрик Холстеда получены потенциальные значения длины программы – 156, объема – 741,78, сложности 25,5 и оценки интеллектуальных усилий – 18915,39.

Заключение

В результате выполнения выпускной квалификационной работы было создано веб-приложение «Автоматизированное рабочее место администратора отдела информирования клиентов организации». В ходе его реализации были выполнены следующие задачи:

- проведен анализ предметной области;
- проанализированы существующие на рынке предложения по смежной тематике;
- определены функциональные и нефункциональные требования к разрабатываемой системе;
- разработана UML-диаграмма вариантов использования;
- определены технологии и средства разработки;
- спроектированы клиентская и серверная части системы, в рамках проектирования созданы диаграммы классов и диаграмма компонентов;
- спроектирован графический интерфейс клиентского приложения;
- произведен расчет бюджета на разработку системы;
- составлен график работ с помощью диаграммы Ганта;
- реализована серверная и клиентская части системы;
- произведено ручное и автоматизированное тестирование функций системы, в результате чего был сделан вывод о корректности их работы;
- рассчитаны метрики размера кода и метрики Холстеда, на основе которых определена сложность кода разработанного приложения.

В результате выполнения всех вышеперечисленных пунктов была получена система, предоставляющая администратору отдела информирования клиентов организации удобный и эффективный способ управления проектами и учетными записями сотрудников.

Список использованных источников

1. WebAssembly – Совместимость с браузерами // MDN Web Docs [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/WebAssembly/Reference/JavaScript_interface (дата обращения: 26.05.2025).
2. Что такое нефункциональные требования: примеры, определения, полное руководство // Visure [Электронный ресурс]. URL: <https://visuresolutions.com/ru/blog/non-functional-requirements> (дата обращения: 26.05.2025).
3. Обзор Entity Framework Core // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/ef/core> (дата обращения: 26.05.2025).
4. Введение в Blazor. Что такое Blazor // Metanit [Электронный ресурс]. URL: <https://metanit.com/sharp/blazor/1.1.php> (дата обращения: 27.05.2025).
5. Ричардс, М. Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы / М. Ричардс, Н. Форд – СПб.: Питер, 2023. – 448 с.: ил. – (Серия «Для профессионалов»).
6. Дейт, К. Дж. Введение в системы баз данных, 8-е издание / К. Дж. Дейт – Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 1328 с.: ил. – Парал. тит. англ. ISBN 5-8459-0788-8 (рус.).
7. Проектирование интерфейса: 8 принципов, которые должен знать каждый UX-дизайнер // Яндекс Практикум [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/proektirovanie-interfejsov> (дата обращения: 27.05.2025).
8. Гультияев, А. К. Microsoft Office Professional 2007. Управление проектами: Практическое пособие / А. К. Гультияев – СПб.: КОРОНА-Век, 2008. – 480 с., ил.
9. Что такое диаграмма Ганта // Weeeek [Электронный ресурс]. URL: <https://weeeek.net/ru/blog/diagramma-ganta> (дата обращения: 01.06.2025).
10. Репозиторий с полным кодом проекта // GitHub [Электронный ресурс]. URL: <https://github.com/motherpleasedonotlook/DiplomaProjectCallCenter> (дата обращения: 01.06.2025).

11. Работа с базой данных: Подключение Entity Framework Core // Metanit [Электронный ресурс]. URL: <https://metanit.com/sharp/aspnet6/12.1.php> (дата обращения 27.05.2025).
12. EF Core Migrations // Learn Entity Framework [Электронный ресурс]. URL: <https://www.learnentityframeworkcore.com/migrations> (дата обращения 27.05.2025).
13. Обзор YARP // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/aspnet/core/fundamentals/servers/yarp/yarp-overview?view=aspnetcore-9.0> (дата обращения 28.05.2025).
14. Cross-Origin Resource Sharing (CORS) // MDN Web Docs [Электронный ресурс]. URL: <https://developer.mozilla.org/ru/docs/Web/HTTP/Guides/CORS> (дата обращения: 29.05.2025).
15. Как научиться работать в Blazor, делая что-то полезное // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/first/articles/583042/> (дата обращения: 29.05.2025).
16. Что такое ручное тестирование и почему без него не бывает качественного ПО // Skillfactory Media [Электронный ресурс]. URL: <https://blog.skillfactory.ru/chto-takoe-ruchnoe-testirovanie> (дата обращения: 29.05.2025).
17. Попарное тестирование: суть техники, инструменты и примеры // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/otus/articles/592575> (дата обращения: 29.05.2025).
18. Модульное тестирование C# с NUnit и .NET Core // Microsoft Learn [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-csharp-with-nunit> (дата обращения: 01.06.2025).
19. Вступление в NUnit // ITVDN [Электронный ресурс]. URL: <https://itvdn.com/ru/blog/article/entry-into-nunit> (дата обращения: 01.06.2025).
20. Метрики сложности программ // StudFiles [Электронный ресурс]. URL: <https://studfile.net/preview/3221055/page:2> (дата обращения: 01.06.2025).

21. Метрики Холстеда // StudFiles [Электронный ресурс].
URL: <https://studfile.net/preview/1568115/page:2> (дата обращения: 01.06.2025).

Приложение А
(обязательное)
Исходный код приложения

Листинг А.1 – Класс Admin

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace CallCenterRepository.Models;
[Table("Admins")]
public class Admin
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int PkAdmin { get; set; }

    [Required(ErrorMessage = "Username is required.")]
    [MaxLength(30)]
    public string Username { get; set; }

    [Required(ErrorMessage = "Password is required.")]
    [MaxLength(100)]
    public string Password { get; set; }

    [Required]
    public DateTime Created { get; set; } = DateTime.UtcNow;

    [Required]
    public bool IsActive { get; set; } = true;

    [Required]
    public int SelfOperatorProfile { get; set; }

    public Admin(string username, string password, int
selfOperatorProfile)
    {
        if (string.IsNullOrEmpty(username))
        {
            throw new ArgumentException("Username cannot be null
or empty.", nameof(username));
        }

        if (string.IsNullOrEmpty(password))
        {
            throw new ArgumentException("Password cannot be null
or empty.", nameof(password));
        }

        Username = username;
        Password = password;
        SelfOperatorProfile = selfOperatorProfile;
    }
}

```

Листинг А.2 – Класс ApplicationDbContext

```
using Microsoft.EntityFrameworkCore;
using CallCenterRepository.Models;

namespace CallCenterRepository;

public class ApplicationDbContext : DbContext
{
    public DbSet<Project> Projects { get; set; }
    public DbSet<ProjectStatusGroup> ProjectStatusGroups { get;
set; }
    public DbSet<ConversationStatus> ConversationStatuses { get;
set; }
    public DbSet<ProjectStatus> ProjectStatuses { get; set; }
    public DbSet<OperatorProject> OperatorProjects { get; set; }
    public DbSet<Admin> Admins { get; set; }
    public DbSet<Operator> Operators { get; set; }
    public DbSet<Client> Clients { get; set; }
    public virtual DbSet<Conversation> Conversations { get; set; }
    public DbSet<ConversationGrade> ConversationGrades { get; set;
}

    public DbSet<ClientNote> ClientNotes { get; set; }

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        // Настройка связи многие-ко-многим между Operator и
Project
        modelBuilder.Entity<OperatorProject>()
            .HasKey(op => new { op.FkOperator, op.FkProject });

        modelBuilder.Entity<OperatorProject>()
            .HasOne(op => op.Operator)
            .WithMany()
            .HasForeignKey(op => op.FkOperator);

        modelBuilder.Entity<OperatorProject>()
            .HasOne(op => op.Project)
            .WithMany(p => p.OperatorProjects)
            .HasForeignKey(op => op.FkProject);
        //при удалении
        modelBuilder.Entity<Operator>()
            .HasOne(o => o.Admin)
            .WithMany()
            .HasForeignKey(o => o.FkAdmin)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<Conversation>()
            .HasOne(c => c.Operator)
            .WithMany()
            .HasForeignKey(c => c.FkOperator)
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

```

modelBuilder.Entity<Conversation>()
    .HasOne(c => c.Client)
    .WithMany()
    .HasForeignKey(c => c.FkClient)
    .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<ConversationGrade>()
    .HasOne(cg => cg.Conversation)
    .WithMany()
    .HasForeignKey(cg => cg.FkConversation)
    .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<ClientNote>()
    .HasOne(cn => cn.Conversation)
    .WithMany(c => c.Notes) // навигационное свойство в
Conversation
    .HasForeignKey(cn => cn.FkConversation)
    .OnDelete(DeleteBehavior.Cascade);

//уникальность имен пользователей
modelBuilder.Entity<Admin>()
    .HasIndex(a => a.Username)
    .IsUnique();
modelBuilder.Entity<Operator>()
    .HasIndex(op => op.Username)
    .IsUnique();

modelBuilder.Entity<ConversationStatus>()
    .HasKey(cs => new { cs.FkTalk, cs.FkProjectStatus });
// Составной ключ

modelBuilder.Entity<ConversationStatus>()
    .HasOne(cs => cs.Conversation)
    .WithMany(c => c.SelectedStatuses)
    .HasForeignKey(cs => cs.FkTalk);

modelBuilder.Entity<ConversationStatus>()
    .HasOne(cs => cs.ProjectStatus)
    .WithMany()
    .HasForeignKey(cs => cs.FkProjectStatus);

base.OnModelCreating(modelBuilder);
}

protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
    optionsBuilder.UseNpgsql("" +
        "Host=localhost;" +
        "Port=5432;" +
        "Database=CallCenterDB;" +
        "Username=postgres;" +

```

```

        "Password=*****"
    );
}
}

```

Листинг А.3 – UserService/Program.cs

```

using CallCenterRepository;
using Microsoft.AspNetCore.Mvc;
using UserService;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddOpenApi();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

var context = new ApplicationDbContext();
var services = new Services(context);

// #1 получить профиль админа
app.MapGet("admin/{adminId}", async (int adminId) =>
{
    try
    {
        var existingAdmin = await
services.GetAdminProfile(adminId);
        Console.WriteLine("UserService method Admin may be
completed (or maybe not...)");
        return Results.Ok(existingAdmin); //возвращаем данные
профиля
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine(ex.Message);
        return Results.NotFound(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.Data);
    }
}

```

```

        return Results.StatusCode(500);
    }
}).WithName("Admin");

//#2 все операторы админа
app.MapGet("admin/{adminId}/operators", async (int adminId) =>
{
    try
    {
        var operators = await
services.GetAllOperatorsByAdminAsync(adminId);
        return Results.Ok(operators); //возвращаем операторов
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine(ex.Message);
        return Results.NotFound(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.Data);
        return Results.StatusCode(500);
    }
}).WithName("GetAllOperators");

//#3 получить профиль оператора по id
app.MapGet("operators/{operatorId}", async (int operatorId) =>
{
    try
    {
        var operator_ = await
services.GetOperatorByAdminAsync(operatorId);
        return Results.Ok(operator_); //возвращаем оператора
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine(ex.Message);
        return Results.NotFound(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.Data);
        return Results.StatusCode(500);
    }
}).WithName("GetOperator");

//#4 изменить имя оператора
app.MapPut("operators/{operatorId}/username", async (int
operatorId, string newUsername) =>
{
    try

```

```

    {
        var editedOperatorId = await
services.EditOperatorUsernameAsync(operatorId, newUsername);
        return Results.Ok(editedOperatorId); //возвращаем id
оператора
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine(ex.Message);
        return Results.NotFound(ex.Message);
    }
    catch (InvalidOperationException ex)
    {
        Console.WriteLine(ex.Message);
        return Results.Conflict(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.Data);
        return Results.StatusCode(500);
    }
}).WithName("EditOperatorUsername");

//#5 переключить статус оператора
app.MapPut("operators/{operatorId}/status", async (int operatorId)
=>
{
    try
    {
        var editedOperatorId = await
services.SwitchOperatorStatusAsync(operatorId);
        return Results.Ok(editedOperatorId); //возвращаем id
оператора
    }
    catch (ArgumentException ex)
    {
        Console.WriteLine(ex.Message);
        return Results.NotFound(ex.Message);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.Data);
        return Results.StatusCode(500);
    }
}).WithName("SwitchOperatorStatus");

//#6 Получение операторов проекта
app.MapGet("project/operators", async (int projectId) =>
{
    try
    {

```

```

        var operators = await
services.GetOperatorsByProjectIdAsync(projectId);
        return Results.Ok(operators);
    }
    catch (ArgumentException ex)
    {
        return Results.BadRequest(ex.Message);
    }
    catch (Exception ex)
    {
        return Results.Problem(ex.Message);
    }
}).WithName("GetProjectOperators");

//#7 Активные операторы админа, не принадлежащие проекту N
app.MapGet("admin/available-operators", async (int adminId, int
projectId)=>
{
    try
    {
        var operators = await
services.GetAvailableOperatorsForProjectAsync(adminId, projectId);
        return Results.Ok(operators);
    }
    catch (Exception ex)
    {
        return Results.Problem(ex.Message);
    }
}).WithName("AvailableOperatorsToAdd");

app.Run();

```

Листинг A.4 – UserService/Services.cs

```

using CallCenterRepository;
using CallCenterRepository.Models;
using Microsoft.EntityFrameworkCore;

namespace UserService;

public class Services(ApplicationDbContext context)
{
    //№1 получить профиль админа по id
    public async Task<AdminDto> GetAdminProfile(int adminId)
    {
        var admin = await context.Admins
            .FirstOrDefaultAsync(a => a.PkAdmin==adminId);
        if(admin==null)throw new ArgumentException($"Администратор
с id {adminId} не найден.");
        var result = new AdminDto
        {
            PkAdmin = admin.PkAdmin,

```

```

        Created = admin.Created,
        IsActive = admin.IsActive,
        Password = admin.Password,
        Username = admin.Username,
        SelfOperatorProfileId = admin.SelfOperatorProfile
    };
    return result;
}

//№3 возвращает всех операторов данного админа
public async Task<List<OperatorDto>>
GetAllOperatorsByAdminAsync(int adminId)
{
    var admin = await context.Admins.FirstOrDefaultAsync(a =>
a.PkAdmin == adminId);
    if (admin==null)throw new
ArgumentException($"Администратор с ID {adminId} не найден.");

    return await context.Operators
        .Where(op => op.FkAdmin == adminId && op.PkOperator!=
admin.SelfOperatorProfile)
        .Select(op => new OperatorDto
        {
            PkOperator = op.PkOperator,
            Username = op.Username,
            Password = op.Password,
            Created = op.Created,
            IsActive = op.IsActive,
            FkAdmin = op.FkAdmin
        })
        .ToListAsync();
}

//№4 возвращает оператора по id
public async Task<OperatorDto> GetOperatorByAdminAsync(int
operatorId)
{
    var operator_ = await context.Operators
        .FirstOrDefaultAsync(op => op.PkOperator==operatorId);
    if(operator_==null)throw new ArgumentException($"Оператор
с id {operatorId} не найден.");
    return new OperatorDto
    {
        PkOperator = operator_.PkOperator,
        Username = operator_.Username,
        Password = operator_.Password,
        Created = operator_.Created,
        IsActive = operator_.IsActive,
        FkAdmin = operator_.FkAdmin
    };
}

//№5 изменяет имя оператора и возвращает профиль

```

```

    public async Task<OperatorDto> EditOperatorUsernameAsync(int
operatorId, string newUsername)
    {
        var operator_ = await context.Operators
            .FirstOrDefaultAsync(op => op.PkOperator==operatorId);
        if(operator_==null)throw new ArgumentException($"Оператор
с id {operatorId} не найден.");
        var sameUsername = await context.Operators
            .AnyAsync(o => o.Username == newUsername &&
o.PkOperator!=operatorId);
        if (sameUsername)throw new InvalidOperationException($"Имя
{newUsername} занято.");
        operator_.Username = newUsername;
        await context.SaveChangesAsync();
        return new OperatorDto
        {
            PkOperator = operator_.PkOperator,
            Username = operator_.Username,
            Password = operator_.Password,
            Created = operator_.Created,
            IsActive = operator_.IsActive,
            FkAdmin = operator_.FkAdmin
        };
    }
    //№6 переключение статуса оператора
    public async Task<OperatorDto> SwitchOperatorStatusAsync(int
operatorId)
    {
        var operator_ = await context.Operators
            .FirstOrDefaultAsync(op => op.PkOperator==operatorId);
        if(operator_==null)throw new ArgumentException($"Оператор
с id {operatorId} не найден.");
        operator_.IsActive = operator_.IsActive != true;
        await context.SaveChangesAsync();
        return new OperatorDto
        {
            PkOperator = operator_.PkOperator,
            Username = operator_.Username,
            Password = operator_.Password,
            Created = operator_.Created,
            IsActive = operator_.IsActive,
            FkAdmin = operator_.FkAdmin
        };
    }

    //№7 добавление клиентов в проект
    public async Task<int> AddClientsToProject(
int fkProject,
List<Dictionary<string, string>> clientData)
    {
        if (clientData == null || clientData.Count == 0)
            throw new ArgumentException("Client data list cannot
be null or empty.", nameof(clientData));
    }

```

```

        // Проверяем существование проекта
        var projectExists = await context.Projects.AnyAsync(p =>
p.PkProject == fkProject);
        if (!projectExists)
            throw new ArgumentException($"Project with id
{fkProject} not found.");

        // Получаем существующие номера для проекта (для проверки
дубликатов)
        var existingNumbers = await context.Clients
            .Where(c => c.FkProject == fkProject)
            .Select(c => c.PhoneNumber)
            .ToListAsync();

        var clients = new List<Client>();
        foreach (var data in clientData)
        {
            try
            {
                var phoneNumber = data["PhoneNumber"].Trim();
                var name = data["Name"].Trim();

                if (string.IsNullOrEmpty(phoneNumber) ||
phoneNumber.Length < 12)
                    throw new ArgumentException("Invalid phone
number format.");

                // проверка на дубликат в текущем проекте
                if (existingNumbers.Contains(phoneNumber))
continue; //пропускаем его

                var client = new Client(
                    phoneNumber: phoneNumber,
                    name: name ?? throw new
InvalidOperationException("Name field is empty."),
                    fkProject: fkProject
                );
                clients.Add(client);
            }
            catch (KeyNotFoundException ex)
            {
                throw new ArgumentException($"Missing required
field: {ex.Message}");
            }
        }

        if (clients.Count == 0)
            return 0;

        // Сохраняем в БД в транзакции
        await using var transaction = await
context.Database.BeginTransactionAsync();

```

```

        try
        {
            await context.Clients.AddRangeAsync(clients);
            await context.SaveChangesAsync();
            await transaction.CommitAsync();
            return clients.Count;
        }
        catch
        {
            await transaction.RollbackAsync();
            throw;
        }
    }

    //№8 получение списка клиентов проекта
    public async Task<List<ClientInfoDto>> GetProjectClients(int
projectId)
    {
        // Проверяем существование проекта
        var projectExists = await context.Projects
            .AnyAsync(p => p.PkProject == projectId);

        if (!projectExists)
        {
            throw new ArgumentException($"Проект с id {projectId}
не найден.");
        }

        // Получаем клиентов проекта
        var clients = await context.Clients
            .Where(cl => cl.Project.PkProject == projectId)
            .Select(cl => new ClientInfoDto
            {
                ClientId = cl.PkClient,
                PhoneNumber = cl.PhoneNumber,
                Name = cl.Name,
                ProcessingStatus = cl.Status.ToString()
            })
            .ToListAsync();

        return clients;
    }

    //№9 связать операторов с проектом
    public async Task<int> AddOperatorsToProjectAsync(int
projectId, List<int> operatorIds)
    {
        // проект есть
        var project = await context.Projects
            .FirstOrDefaultAsync(p => p.PkProject == projectId);

        if (project == null)
            throw new ArgumentException($"Проект с ID {projectId}
не найден.");
    }

```

```

        // проверка операторов
        var existingOperators = await context.Operators
            .Where(o => operatorIds.Contains(o.PkOperator))
            .Where(o => o.FkAdmin == project.FkAdmin) //тот же
админ
            .Select(o => o.PkOperator)
            .ToListAsync();

        var missingOperators =
operatorIds.Except(existingOperators).ToList();
        if (missingOperators.Any())
        {
            throw new ArgumentException(
                $"Операторы не найдены или не принадлежат
администратору: {string.Join(", ", missingOperators)}",
                nameof(operatorIds));
        }

        // кто из списка уже связан с проектом
        var existingLinks = await context.OperatorProjects
            .Where(op => op.FkProject == projectId)
            .Select(op => op.FkOperator)
            .ToListAsync();
        // кого нет в проекте
        var operatorsToAdd =
operatorIds.Except(existingLinks).ToList();

        // добавляем ток тех, кого не было
        foreach (var operatorId in operatorsToAdd)
        {
            context.OperatorProjects.Add(new OperatorProject
            {
                FkOperator = operatorId,
                FkProject = projectId,
                AssignedDate = DateTime.UtcNow
            });
        }
        project.LastUpdate = DateTime.UtcNow;

        // Сохраняем изменения и возвращаем количество добавленных
операторов
        await context.SaveChangesAsync();
        return operatorsToAdd.Count;
    }

    //№10 получить всех операторов, связанных с данным проектом
    public async Task<List<OperatorDto>>
GetOperatorsByProjectIdAsync(int projectId)
    {
        if (!await context.Projects.AnyAsync(p => p.PkProject ==
projectId))
            throw new ArgumentException($"Проект с ID {projectId}

```

не найден");

```

        return await context.OperatorProjects
            .Where(op => op.FkProject == projectId)
            .Include(op => op.Operator)
            .Select(op => new OperatorDto
            {
                PkOperator = op.Operator.PkOperator,
                Username = op.Operator.Username,
                Password = op.Operator.Password,
                Created = op.Operator.Created,
                IsActive = op.Operator.IsActive,
                FkAdmin = op.Operator.FkAdmin
            })
            .ToListAsync();
    }

    // #11 все активные операторы, не связанные с проектом N
    public async Task<List<OperatorDto>>
    GetAvailableOperatorsForProjectAsync(int adminId, int projectId)
    {
        return await context.Operators
            .Where(o => o.FkAdmin == adminId && o.IsActive)
            .Where(o => !context.OperatorProjects
                .Any(op => op.FkProject == projectId &&
                    op.FkOperator == o.PkOperator))
            .Select(o => new OperatorDto
            {
                PkOperator = o.PkOperator,
                Username = o.Username,
                Password = o.Password,
                Created = o.Created,
                IsActive = o.IsActive,
                FkAdmin = o.FkAdmin
            })
            .ToListAsync();
    }
}

```

Листинг A.5 – UserService/Records.cs

```

namespace UserService;

public record ClientInfoDto
{
    public int ClientId { get; init; }
    public string? Name { get; init; }
    public string? PhoneNumber { get; init; }
    public string? ProcessingStatus { get; init; }
}

```

```
public record AdminDto
{
    public int PkAdmin { get; init; }
    public DateTime Created { get; init; }
    public bool IsActive { get; init; }
    public string? Password { get; init; }
    public string? Username { get; init; }
    public int SelfOperatorProfileId { get; init; }
}

public record OperatorDto
{
    public int PkOperator { get; init; }
    public string? Username { get; init; }
    public string? Password { get; init; }
    public DateTime Created { get; init; }
    public bool IsActive { get; init; }
    public int FkAdmin { get; init; }
}
```

Приложение Б
(рекомендуемое)
Графическая часть

Наименование: Войти
ID: 1
Краткое описание: администратор колл-центра авторизуется в системе.
Действующие лица: администратор, система.
Предусловие: администратор не авторизован.
<p>Основной поток:</p> <ul style="list-style-type: none"> – администратор переходит на страницу входа в приложение; – администратор вводит данные для входа в аккаунт (имя пользователя и пароль); – система проверяет корректность введенных данных; <p>система предоставляет администратору доступ к функционалу приложения.</p>
Постусловие: администратор вошел в систему, ему доступен функционал.

Рисунок Б.1 – Спецификация варианта использования «Войти»

Наименование: Создать аккаунт оператора.
ID: 2
Краткое описание: администратор колл-центра создает новый аккаунт оператора.
Действующие лица: администратор, система.
Предусловие: администратор авторизован, аккаунт оператора с данным именем не создан.
<p>Основной поток:</p> <ul style="list-style-type: none"> – администратор переходит на страницу добавления нового оператора; – администратор вводит данные новой учетной записи (имя пользователя и временный пароль); – система проверяет корректность введенных данных; <p>система создает учетную запись оператора.</p>
Постусловие: учетная запись оператора создана, доступна, и отображается в списке операторов.

Рисунок Б.2 – Спецификация варианта использования «Создать аккаунт оператора»

Наименование: Создать проект.
ID: 3
Краткое описание: администратор создает проект.
Действующие лица: администратор, система.
Предусловие: администратор авторизован, проект не создан.
<p>Основной поток:</p> <ul style="list-style-type: none"> – администратор входит в учетную запись и переходит на страницу проектов; – администратор нажимает на кнопку добавления проекта; – администратор заполняет все поля формы создания проекта; – администратор нажимает кнопку «Сохранить проект».
Постусловие: проект создан и отображается в списке проектов.

Рисунок Б.3 – Спецификация варианта использования «Создать проект»

Наименование: Оценить запись звонка.
ID: 4
Краткое описание: администратор колл-центра оценивает запись диалога оператора с клиентом в соответствии с критериями.
Действующие лица: администратор, система.
Предусловие: администратор авторизован, запись звонка существует.
<p>Основной поток:</p> <ul style="list-style-type: none"> – администратор переходит на страницу подробной информации о совершенном звонке; – администратор разворачивает раздел оценок звонка; – администратор выставляет оценку записи звонка от 1 до 5 баллов по каждому из четырех критериев; – администратор нажимает кнопку «Сохранить»; – система сохраняет оценки и выводит соответствующее оповещение администратору.
Постусловие: оценки звонка сохранены.

Рисунок Б.4 – Спецификация варианта использования «Оценить запись звонка»

Наименование: Редактировать проект.
ID: 5
Краткое описание: администратор изменяет параметры существующего проекта.
Действующие лица: администратор, система.
Предусловие: администратор авторизован, проект существует.
<p>Основной поток:</p> <ul style="list-style-type: none"> – администратор открывает страницу проекта; – администратор вносит изменения в скрипт, статусы звонков, базу клиентов, список назначенных сотрудников или время работы; – система сохраняет изменения.
Постусловие: изменения сохранены.

Рисунок Б.5 – Спецификация варианта использования «Редактировать проект»

Наименование: Загрузить базу клиентов
ID: 6
Краткое описание: администратор загружает базу клиентов из Excel-файла.
Действующие лица: администратор, система.
Предусловие: администратор авторизован, проект создан.
<p>Основной поток:</p> <ul style="list-style-type: none"> – администратор переходит на страницу проекта и открывает вкладку загрузки клиентской базы; – администратор нажимает на кнопку «Загрузить файл»; – администратор выбирает и загружает Excel-файл; – система проверяет файл на корректность формата; – система извлекает данные из файла и сохраняет их; – система отображает список добавленных клиентов.
Постусловие: база клиентов добавлена в проект.

Рисунок Б.6 – Спецификация варианта использования
«Загрузить базу клиентов»

Наименование: Посмотреть список операторов
ID: 7
Краткое описание: администратор просматривает список закрепленных за ним операторов.
Действующие лица: администратор, система.
Предусловие: администратор авторизован.
<p>Основной поток:</p> <ul style="list-style-type: none"> – администратор переходит на страницу списка операторов; – администратор выбирает критерий фильтрации списка; – система отображает всех операторов списка в соответствии с выбранным критерием.
Постусловие: список операторов отображен.

Рисунок Б.7 – Спецификация варианта использования «Посмотреть список операторов»

Наименование: Оценить звонок
ID: 8
Краткое описание: администратор оценивает запись звонка оператора по критериям.
Действующие лица: администратор, система.
Предусловие: администратор авторизован, звонок сохранен.
<p>Основной поток:</p> <ul style="list-style-type: none"> – администратор переходит на страницу звонка; – администратор выставляет оценки в соответствии с определенными критериями; – администратор нажимает «Сохранить»; – система сохраняет оценки; – система выводит сообщение об успешном сохранении выбранных оценок.
Постусловие: оценки сохранены.

Рисунок Б.8 – Спецификация варианта использования «Оценить звонок»

Наименование: Настроить перечень статусов звонка.
ID: 9
Краткое описание: администратор настраивает статусы звонка для проекта.
Действующие лица: администратор, система.
Предусловие: администратор авторизован, проект создан.
<p>Основной поток:</p> <ul style="list-style-type: none"> – администратор переходит на страницу проекта; – администратор нажимает на кнопку «Добавить статус-группу»; – администратор вводит имя статус-группы; – администратор вводит перечень статусов; – администратор нажимает на кнопку «Сохранить»; – система сохраняет статус-группу с перечнем статусов.
Постусловие: перечень статусов звонка настроен.

Рисунок Б.9 – Спецификация варианта использования «Настроить перечень статусов звонка»

Приложение В
(рекомендуемое)
Результаты ручного тестирования

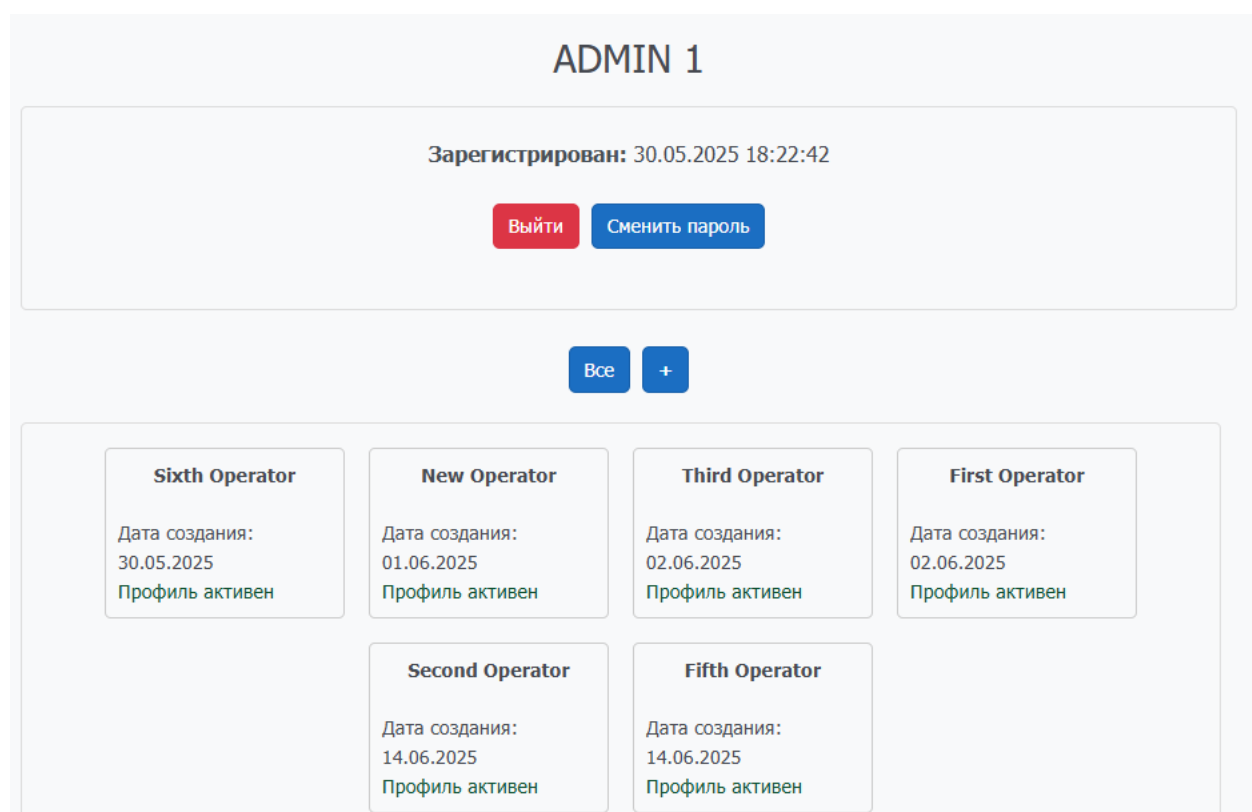


Рисунок В.1 – Успешный вход в систему

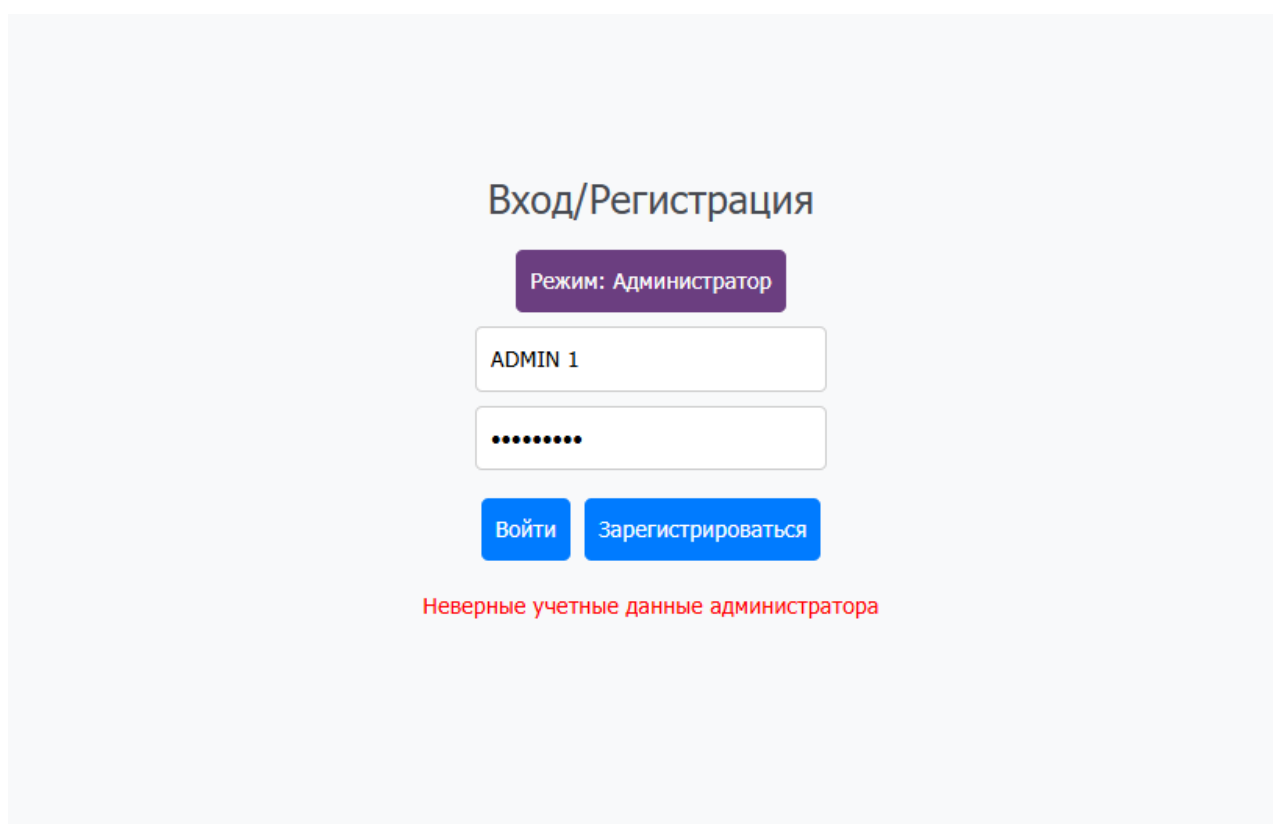


Рисунок В.2 – Ошибка входа в систему

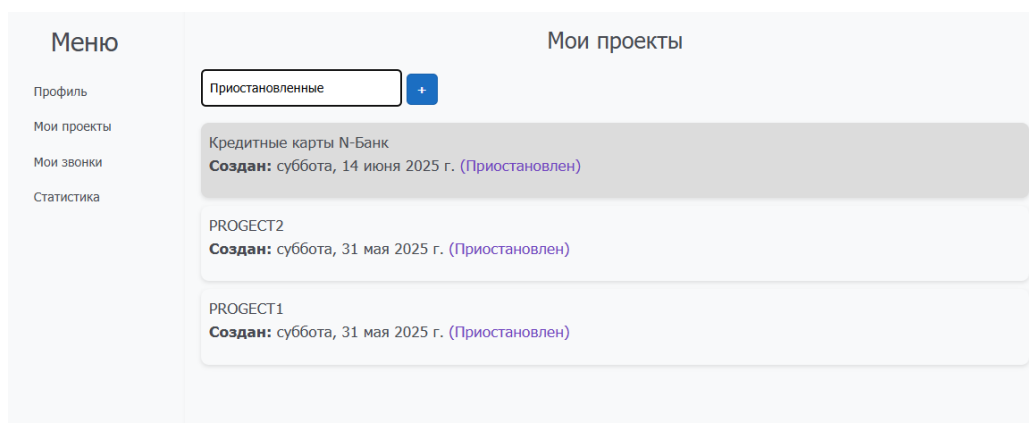


Рисунок В.3 – Сохраненный проект

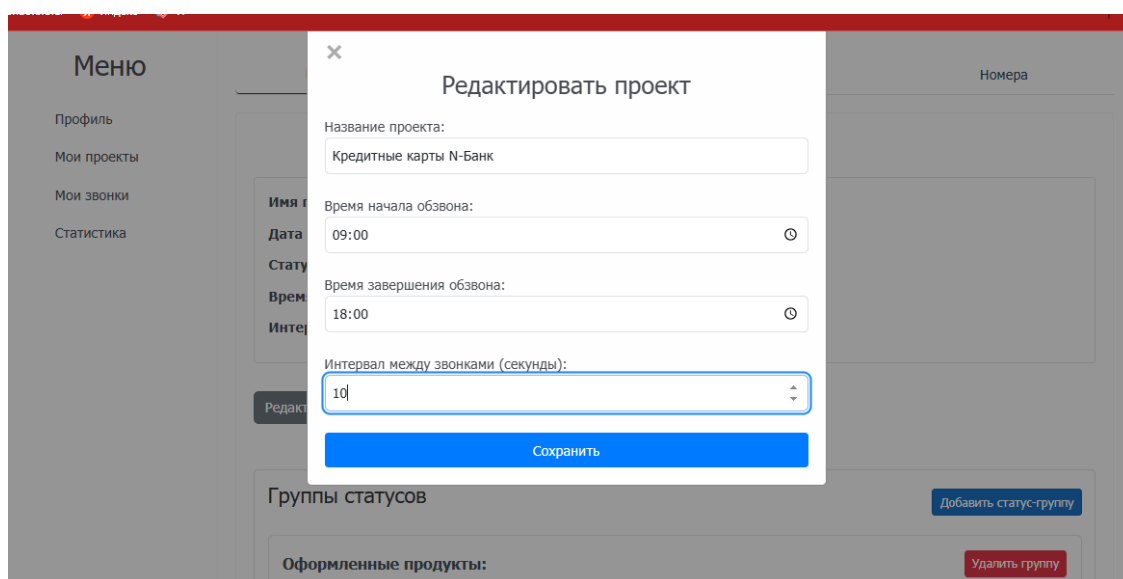


Рисунок В.4 – Редактирование проекта

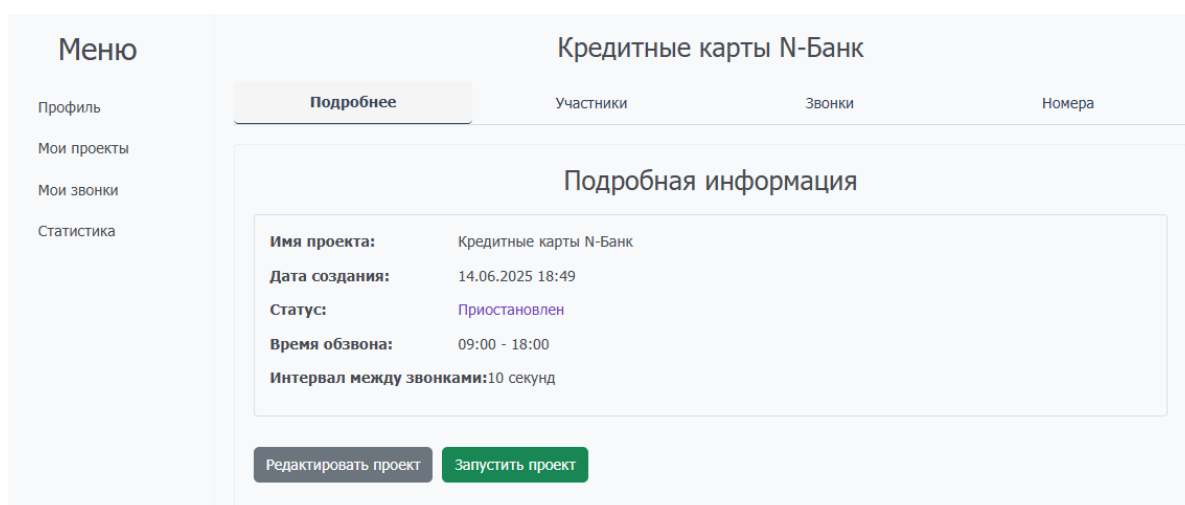


Рисунок В.5 – Результат редактирования проекта

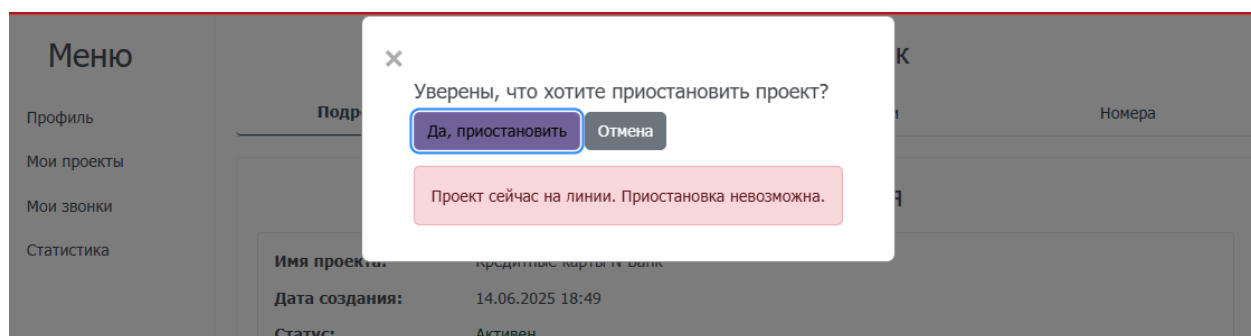


Рисунок В.6 – Попытка приостановить проект в рабочее время

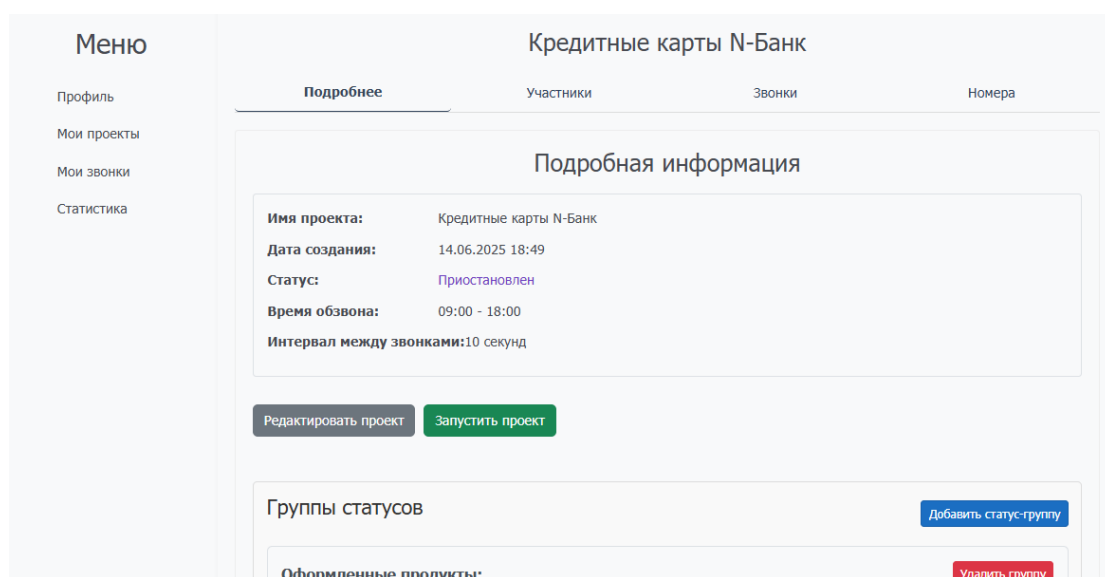


Рисунок В.7 – Приостановленный проект

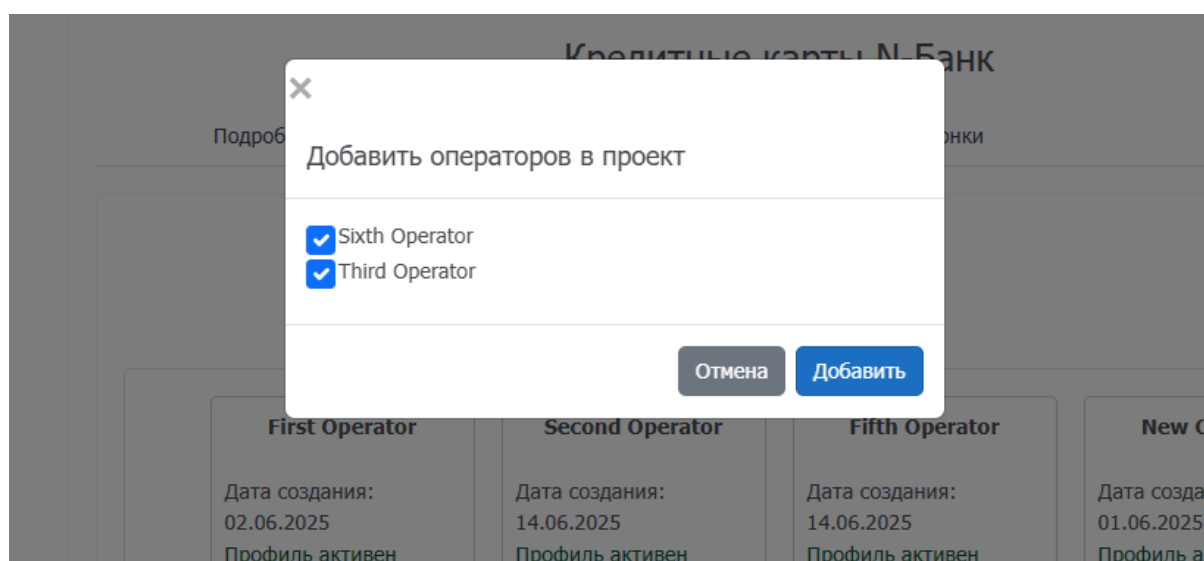


Рисунок В.8 – Добавление операторов в проект

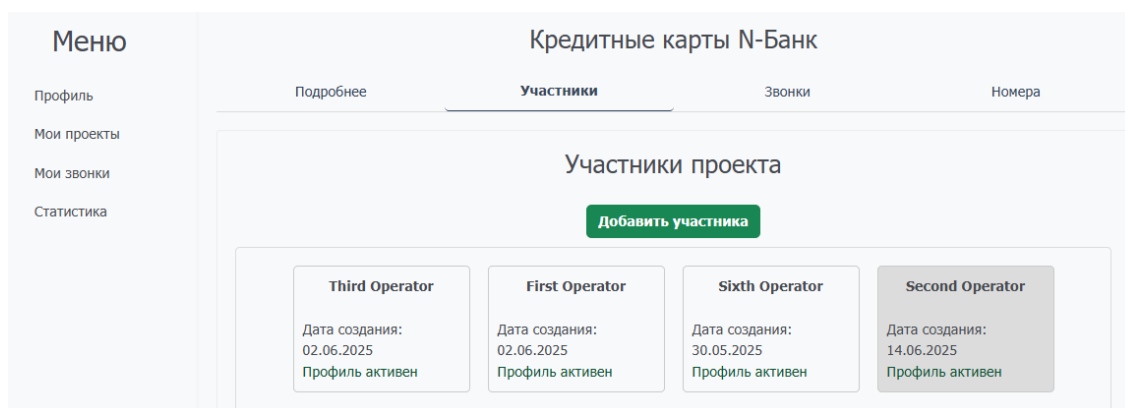


Рисунок В.9 – Результат добавления операторов в проект

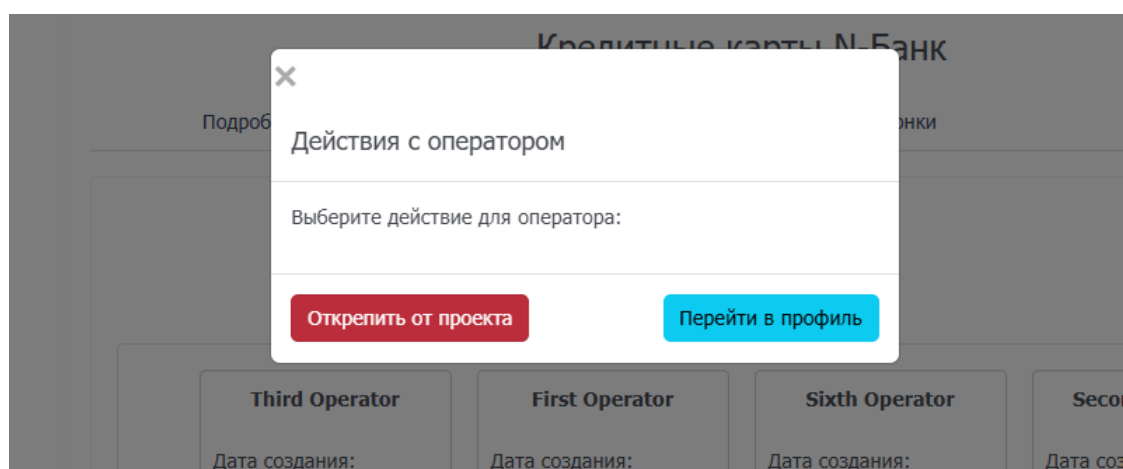


Рисунок В.10 – Открепление оператора Sixth Operator от проекта

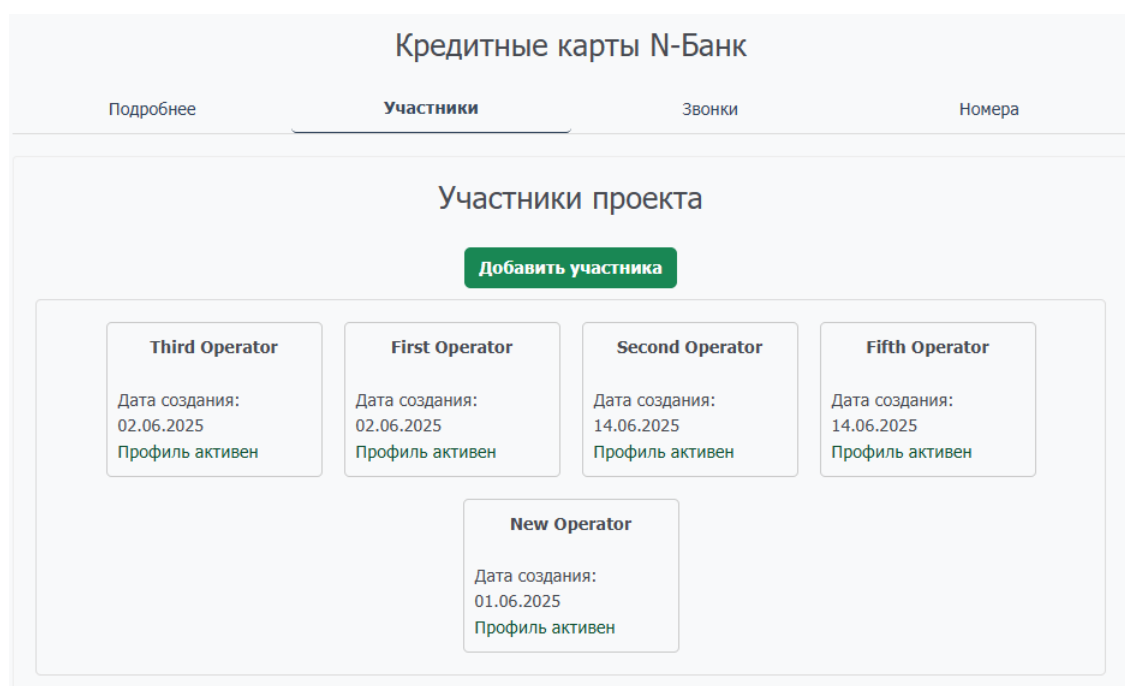


Рисунок В.11 – Результат открепления оператора от проекта

Кредитные карты N-Банк

Подробнее
Участники
Звонки
Номера

Телефонные номера проекта Кредитные карты N-Банк

Все номера

Выберите файл
Ном....xlsx

Успешно добавлено 24 клиентов

Имя	Номер телефона	Статус обработки
client 1	+79631244351	Не обработан
client 2	+79271244230	Не обработан
client 3	+79631222532	Не обработан
client 4	+79271255129	Не обработан
client 5	+79051255138	Не обработан
client 6	+79301255140	Не обработан

Рисунок В.12 – Успешная загрузка клиентской базы

Кредитные карты N-Банк

Подробнее
Участники
Звонки
Номера

Телефонные номера проекта Кредитные карты N-Банк

Все номера

Выберите файл
скрипт N-Банк.txt

Ошибка: The file is not a valid Package file. If the file is encrypted, please supply the password in the constructor.

Имя	Номер телефона	Статус обработки
client 7	+78411255150	Не обработан
client 8	+79301255167	Не обработан
client 11	+79041255190	Не обработан
client 12	+79301255205	Не обработан
client 23	+79251255310	Не обработан
client 24	+79941255320	Не обработан

Рисунок В.13 – Попытка загрузить файл неверного формата

Меню

- Профиль
- Мои проекты
- Мои звонки
- Статистика

Редактировать

Оценки ▾

Речь:

4

Следование скрипту:

5

Безошибочность:

5

Корректность статусов:

3

Сохранить оценки

Оценки успешно сохранены

Рисунок В.14 – Сохранение оценок

Меню

- Профиль
- Мои проекты
- Мои звонки
- Статистика

карта удобства

Степень согласия

Оформит самостоятельно

Сохранить статусы

Заметка ▾

Стоит поработать над речью, будьте внимательнее при выставлении статусов

Редактировать

Заметка успешно сохранена

Рисунок В.15 – Сохранение заметки

Меню

- Профиль
- Мои проекты
- Мои звонки
- Статистика

Статусы звонка ▾

Оформленные продукты

Карта Удобства

Степень согласия

Оформит самостоятельно

Сохранить статусы

Статусы успешно сохранены

Рисунок В.16 – Редактирование