МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Разработка социальных и экономических информационных систем
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка веб-приложения для единой дежурно-диспетчерской службы»

Обучающийся Д. В. Железко
(Инициалы Фамилия) (личная подпись)

Руководитель Кандидат педагогических наук, доцент, О.Ю. Копша
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Выпускная квалификационная работа посвящена разработке веб-приложения для единой дежурно-диспетчерской службы (далее – ЕДДС). Актуальность темы обусловлена необходимостью повышения эффективности работы диспетчерской службы, снижения вероятности потери данных и ускорения принятия решений при обработке инцидентов.

Целью работы является разработка веб-приложения, обеспечивающего автоматизированный учет и обработку вызовов, что позволит повысить оперативность реагирования и оптимизировать деятельность ЕДДС.

Структура работы включает введение, три главы, заключение и список литературы.

Введение раскрывает актуальность темы, обосновывает выбор направления исследования, определяет объект, предмет, цель и задачи работы, а также методы, используемые для ее выполнения.

Глава 1 посвящена анализу предметной области и объекта исследования. Рассматриваются структура ЕДДС, существующие методы работы, выявляются недостатки текущей системы и формулируются требования к разрабатываемому программному обеспечению.

Глава 2 описывает процесс разработки веб-приложения, включая выбор архитектуры, проектирование базы данных, создание клиентской и серверной частей, а также алгоритмы обработки данных.

Глава 3 содержит тестирование разработанного приложения. Рассматриваются контрольные примеры, сценарии тестирования и анализируется корректность работы системы.

Работа содержит 32 рисунка, 1 таблицу, 23 источника в списке литературы. Общий объём работы занимает 87 страниц.

Оглавление

Введение

Развитие информационных технологий оказывает значительное влияние на системы управления и оперативного реагирования, в том числе на деятельность единой дежурно-диспетчерской службы (далее – ЕДДС). В современных условиях для повышения эффективности работы ЕДДС необходимо внедрение автоматизированных систем, позволяющих оперативно фиксировать и обрабатывать поступающие вызовы, управлять инцидентами и обеспечивать координацию взаимодействия различных служб.

Выбор темы выпускной квалификационной работы (далее – ВКР) обусловлен необходимостью модернизации работы дежурно-диспетчерских служб, где традиционно используется бумажный документооборот и устаревшие методы учета происшествий. Это приводит к ряду проблем: низкой скорости обработки информации, высокой вероятности потери данных, сложностям в формировании отчетности и взаимодействии между службами. Разработка веб-приложения позволит автоматизировать ключевые процессы, улучшить доступ к информации и повысить надежность работы системы.

Актуальность темы обусловлена потребностью муниципальных организаций в эффективных средствах управления инцидентами и экстренными ситуациями. Внедрение веб-приложения для ЕДДС позволит:

- сократить время регистрации и обработки вызовов;
- минимизировать риски утраты данных;
- упростить процесс поиска информации и формирования отчетов;
- обеспечить безопасное и централизованное хранение данных.

Объектом исследования является процесс управления инцидентами в единой дежурно-диспетчерской службе, включая сбор, обработку и анализ информации о происшествиях.

Предмет исследования – разработка веб-приложения, обеспечивающего

автоматизацию процессов регистрации, обработки и управления инцидентами в ЕДДС.

Целью выпускной квалификационной работы является разработка веб-приложения для автоматизации работы единой дежурно-диспетчерской службы, направленного на повышение оперативности и надежности обработки вызовов и инцидентов.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ текущих бизнес-процессов ЕДДС и выявить их недостатки;
- изучить современные подходы к автоматизации дежурно-диспетчерских служб;
- разработать архитектуру и информационную модель веб-приложения.;
- спроектировать и создать базу данных для хранения информации о вызовах и инцидентах;
- реализовать серверную и клиентскую части веб-приложения;
- провести тестирование функциональности системы.

В ходе выполнения работы использовались следующие методы исследования:

- аналитический метод – для изучения существующих решений и выявления их недостатков;
- системный анализ – для рассмотрения процесса управления инцидентами в ЕДДС как комплексной системы;
- метод моделирования – для разработки архитектуры и информационной модели веб-приложения;
- экспериментальный метод – для тестирования работоспособности системы;

– сравнительный метод – для анализа эффективности предложенного решения по сравнению с традиционными подходами.

Выпускная работа состоит из трёх глав.

В первой главе проведён анализ предметной области и объекта исследования. Рассматриваются структура ЕДДС, текущие бизнес-процессы, их недостатки, предлагается улучшенная модель работы, обосновывается выбор веб-приложения и формулируются функциональные требования к системе.

Во второй главе описан процесс разработки веб-приложения. Приведён выбор архитектуры и технологий, спроектированы клиентская и серверная части, база данных, реализована информационная модель и программные модули для обработки данных, приведены диаграммы архитектуры и алгоритмов.

В третьей главе выполнено тестирование разработанного приложения. Представлены контрольные примеры, оценена корректность работы системы и рассчитана экономическая эффективность внедрения программного решения.

Заключение отражает основные результаты работы, подтверждает достижение поставленных целей и задач, а также подчёркивает практическую значимость проекта.

Глава 1 Анализ предметной области и объекта исследования

Администрация Томского района - это исполнительно-распорядительный орган местного самоуправления, который выполняет широкий спектр функций и обязанностей для управления и координации различными аспектами жизни и деятельности Томского района.

Цель существования Администрации Томского района заключается в обеспечении эффективного местного самоуправления, координации и управления различными аспектами жизни района. Основные задачи организации:

Администрация участвует в организации и проведении местных выборов, управляет местными государственными службами и структурами, чтобы обеспечить эффективное местное самоуправление в соответствии с законами и регуляциями.

Администрация заботится о социальном обеспечении населения в районе. Это включает в себя организацию и управление программами по здравоохранению, образованию, социальной помощи, и другим социальным услугам.

Администрация ответственна за управление инфраструктурой района, включая дороги, транспорт, жилищное строительство, и благоустройство территории.

Администрация работает над развитием экономики района, поддерживая местные предприятия, привлекая инвестиции и способствуя созданию новых рабочих мест.

Организация мер по охране окружающей среды, устойчивому природопользованию и поддержанию экологической устойчивости района.

Поддержка сельского хозяйства, развитие сельских территорий, и обеспечение продовольственной безопасности.

Развитие информационных систем и технологий для эффективного управления и обеспечения доступа к информации гражданам и предприятиям.

Организация культурных мероприятий и спортивных мероприятий для населения, поддержка местных художественных и спортивных инициатив.

1.1 Отдел по информатизации и кадровому обеспечению Управления Делами Администрации Томского района (АТР)

Цели и задачи отдела по информатизации и кадровому обеспечению Администрации Томского района заключаются в обеспечении эффективного функционирования информационных систем и кадрового управления в Администрации [12]. Основные цели отдела включают:

- обеспечение автоматизации деятельности Администрации с помощью современных информационных технологий;
- поддержка эффективного муниципального управления через информационно-технические решения;
- защита информации и информационных систем от несанкционированного доступа;
- обеспечение надежной работы информационно-коммуникационных систем и кадрового учета;
- разработка и внедрение программных средств (в том числе веб-приложений) и сервисов для оптимизации процессов управления и взаимодействия с гражданами.

Основными задачами отдела по направлению информационных технологий являются:

- обеспечение функционирования комплекса технических и программных средств автоматизации деятельности Администрации Томского района;

– обеспечение информационно-технической поддержки процесса муниципального управления;
– осуществление комплекса технических мероприятий по защите локальной вычислительной сети (далее - ЛВС) Администрации Томского района, информационных ресурсов, баз данных от несанкционированного доступа;
– проведение работ по технической защите информации и поддержанию достигнутого уровня защиты ЛВС и ее ресурсов на этапах промышленной эксплуатации и модернизации;
– участие в проектировании, создании и внедрении программных средств, включая системы для единой дежурно-диспетчерской службы, что позволяет автоматизировать процессы и улучшить обслуживание граждан.

В результате анализа деятельности отдела по информатизации и кадровому обеспечению установлено, что он играет ключевую роль в обеспечении эффективного функционирования информационных систем администрации.

1.2 Структура отдела ЕДДС Администрации Томского района

Организационная структура отдела единой дежурно-диспетчерской службы Администрации Томского района включает 5 сотрудников: начальника отдела и 4 оперативных дежурных. Ответственность и полномочия распределяются следующим образом:

Начальник отдела ЕДДС. Ответственность:

– общее руководство отделом и координация работы оперативных дежурных;
– разработка и контроль за выполнением планов и мероприятий по гражданской обороне, чрезвычайным ситуациям и управлению ЖКХ;

– принятие решений и оповещение должностных лиц в случае аварий и чрезвычайных ситуаций;
– обеспечение взаимодействия с другими отделами и организациями по вопросам чрезвычайных ситуаций;

Полномочия:

– принятие решений по управлению действиями дежурных и взаимодействие с внешними организациями;
– контроль за соблюдением правил безопасности и протоколов действий при чрезвычайных ситуациях.

Оперативный дежурный. Ответственность:

– круглосуточное дежурство и мониторинг ситуации в районе, связанной с авариями и чрезвычайными ситуациями;
– оперативное принятие мер по направлению служб для ликвидации последствий происшествий;
– сбор и передача информации о происшествиях начальнику отдела и другим ответственным структурам;
– ведение отчетности и документации по работе ЕДДС.

Полномочия:

– участие в управлении ликвидацией аварий и происшествий;
– взаимодействие с различными службами района (МЧС, скорой помощью, коммунальными службами) для координации действий в чрезвычайных ситуациях.

На рисунке 1 представлена структурная схема отдела ЕДДС.

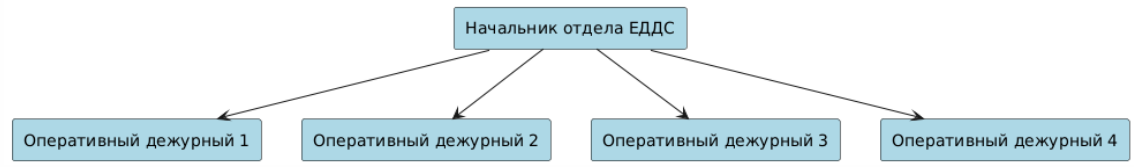


Рисунок 1 – Структурная схема отдела ЕДДС Администрации Томского района

Начальник отдела находится на вершине структуры и осуществляет координацию действий всех оперативных дежурных, которые равномерно распределяют дежурства и обязанности по мониторингу ситуации и реагированию на ЧС [13].

1.3 Процессная модель AS-IS (Как есть)

На текущий момент в ЕДДС Администрации Томского района все процессы по регистрации и учету происшествий выполняются вручную с использованием бумажных журналов. Для анализа текущего состояния системы или процесса необходимо составить модель «AS-IS». Она помогает понять, как именно сейчас работают процессы, выявить проблемы и узкие места, и собрать данные для дальнейшего улучшения. Модель «как есть» представлена на рисунке 2.

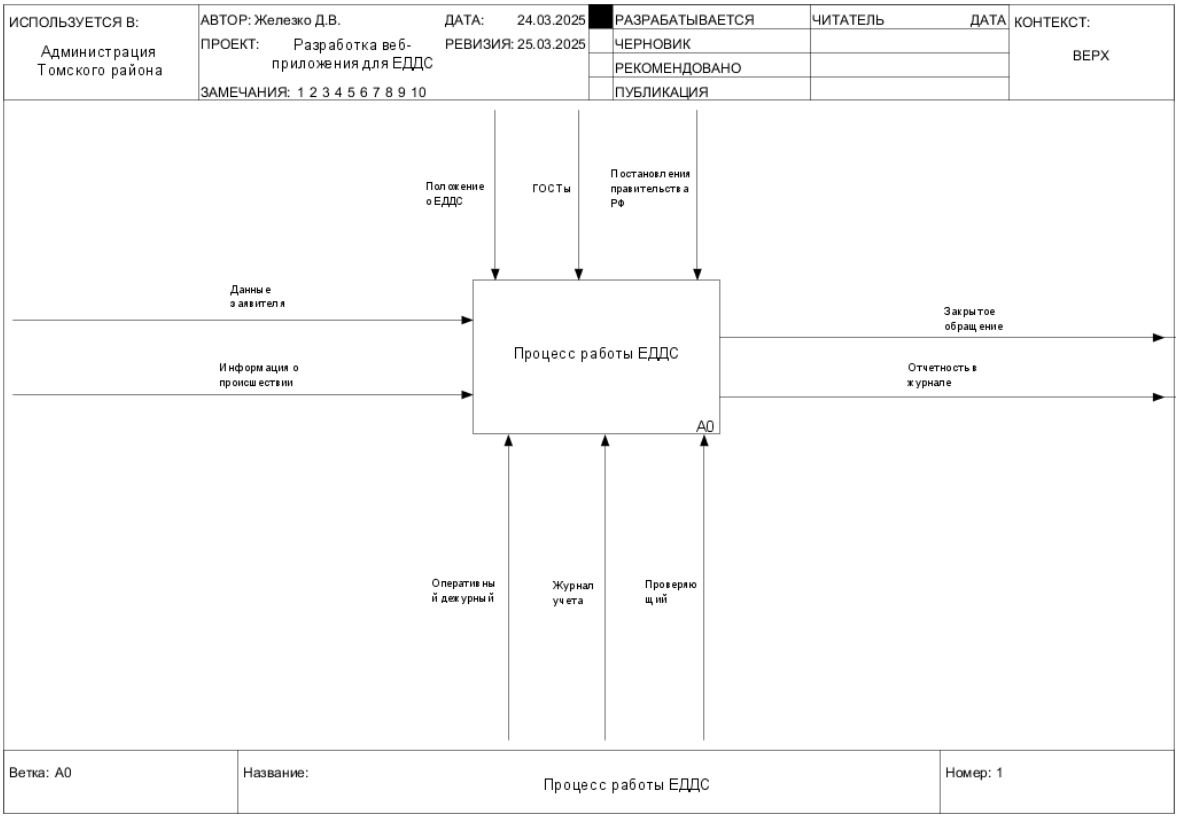


Рисунок 2 – Процессная модель работы «как есть» отдела ЕДДС

Процесс работы устроен следующим образом:

Получение звонка.

Оперативный дежурный поднимает трубку, когда поступает телефонный звонок.

Фиксация информации.

Вся полученная информация записывается вручную в бумажный журнал, куда дежурный фиксирует детали звонка, а также информацию о смене: кто принял и кто сдал смену, дату, время звонка, контактные данные заявителя и описание происшествия.

Поиск и проверка.

В случае необходимости проверки, проверяющие службы могут запросить доступ к бумажным журналам. Однако, если журнал текущий и еще не заполнен и находится у дежурного, его изъять нельзя. Поиск нужной информации в стопке журналов может занять несколько дней или даже недель.

Риски утраты данных.

Если журнал будет утерян или испорчен, данные восстановить будет невозможно, что повлечет за собой наказание для отдела. Утрата данных может серьёзно нарушить работу подразделения и привести к серьезным последствиям для ЕДДС.

Чтобы наглядно показать процесс работы в ЕДДС нужно составить декомпозицию модели. Декомпозиция позволяет разбить сложную систему или процесс на более мелкие, понятные части. Декомпозиция модели «как есть» представлена на рисунке 3.

Анализ модели «AS-IS» выявил основные проблемы:

- ручное ведение журналов замедляет работу дежурного;
- поиск информации по запросу занимает много времени;
- высокие риски утраты данных при порче или утере журнала;
- рабочие процессы дежурного прерываются во время проверок.

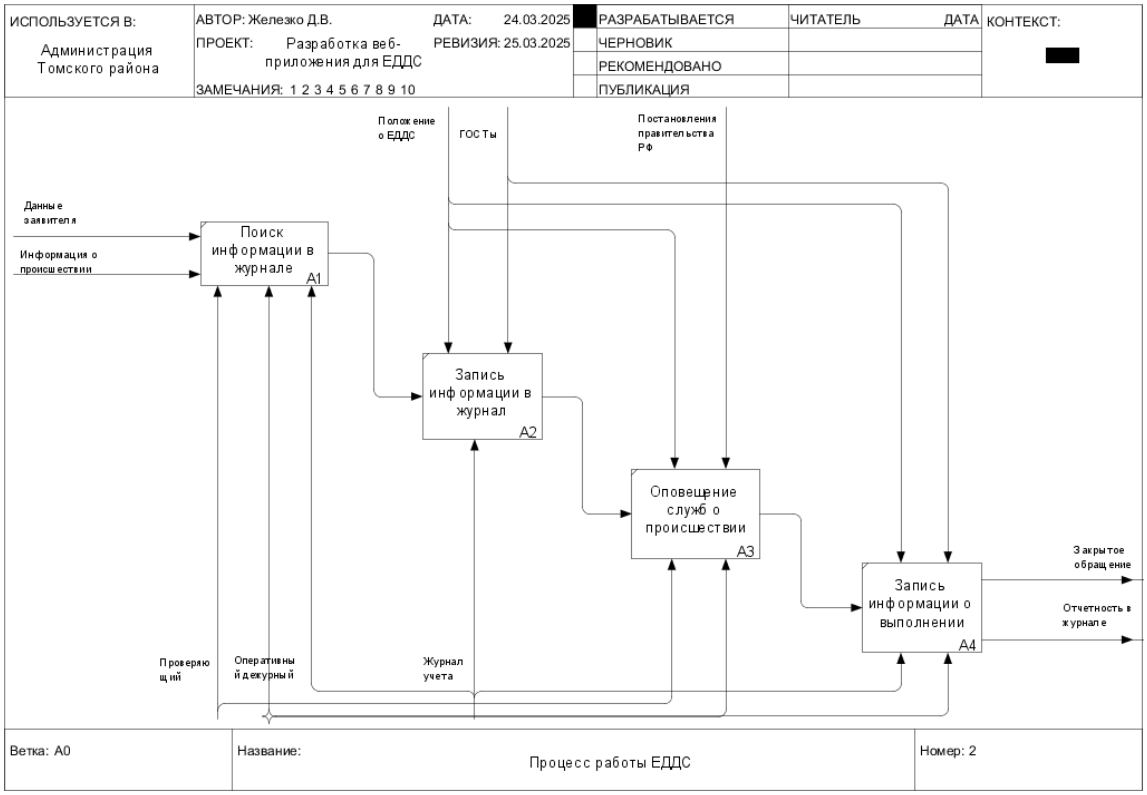


Рисунок 3 – Декомпозиция модели «как есть» отдела ЕДДС

Анализ текущей модели работы ЕДДС показал наличие существенных недостатков, связанных с ручной регистрацией вызовов, использованием бумажных журналов и высоким риском утраты данных. Процессы регистрации и поиска информации занимают много времени, что снижает оперативность реагирования и затрудняет проверку и контроль.

1.4 Процессная модель TO-BE (Как должно быть)

С внедрением автоматизированной системы для ЕДДС процессы значительно упрощаются и ускоряются благодаря переходу на электронный учет. Для описания оптимизированного состояния системы используется модель «TO-BE». Эта модель используется для разработки новых решений и улучшения процессов, устранения недостатков существующей системы и планирования внедрения изменений и автоматизации. Процессная модель «как должно быть» представлена на рисунке 4.

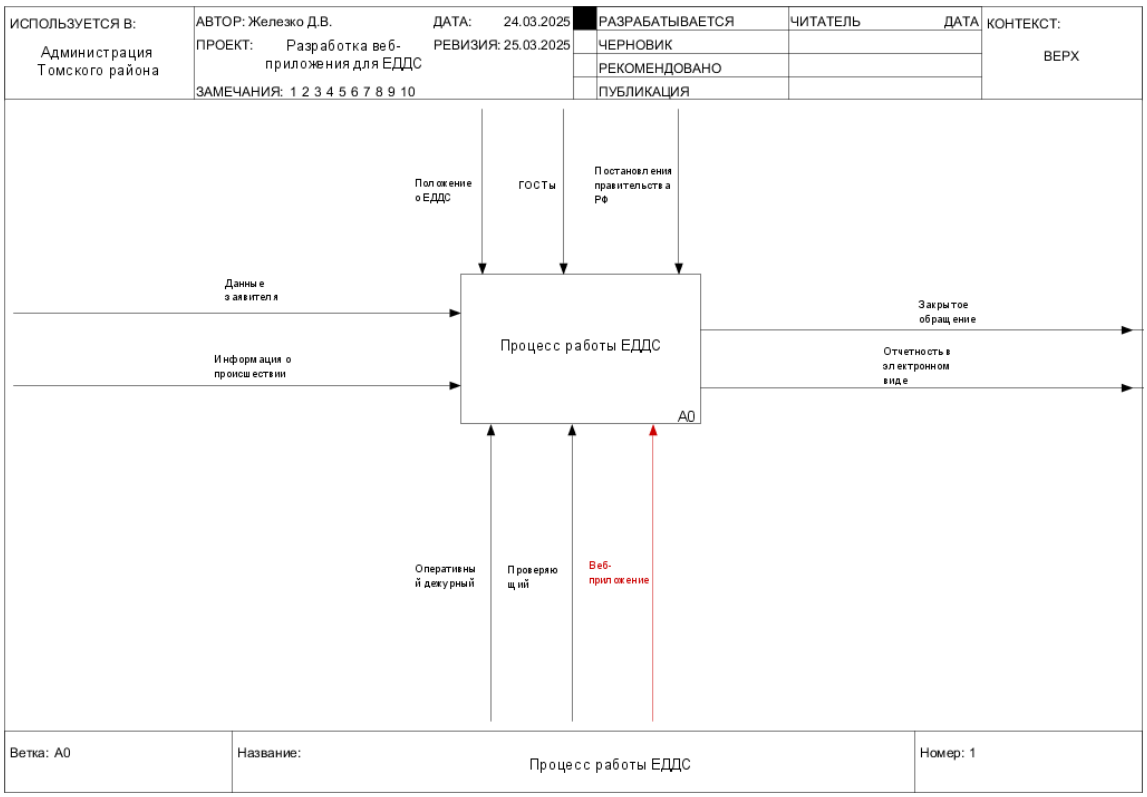


Рисунок 4 – Процессная модель работы «как должно быть» отдела ЕДДС

Вот как будет выглядеть работа в ЕДДС при применении модели «как должно быть»:

Получение звонка.

Оперативный дежурный принимает звонок, после чего на компьютере открывается веб-приложение с готовыми полями для ввода информации о происшествии.

Ввод информации.

Дежурный вносит все необходимые данные в веб-приложение: фиксирует, к какой категории относится происшествие, указывает дату, время и контактные данные. В случае, если звонок не является происшествием или ложный, это также можно указать в веб-приложении.

Очередь звонков.

Веб-приложение предоставляет возможность работы с несколькими звонками одновременно. Если поступает второй звонок, дежурный видит его

в очереди и может выбрать карточку для внесения нужной информации о звонке.

Категоризация и передача информации:

Система позволяет дежурному классифицировать происшествие по категориям, указать организации, которым нужно передать информацию, и внести дополнительные комментарии.

Обеспечение доступности данных.

Если требуется проверка, проверяющие службы могут получить доступ к базе данных веб-приложения на другом рабочем месте, не прерывая работы дежурного. Доступ к системе осуществляется через уникальные данные дежурного или начальника отдела, а поиск нужной информации занимает секунды.

Резервное копирование и безопасность.

Все данные сохраняются в базе данных с регулярным резервным копированием, что исключает риск потери информации при сбоях или авариях. В случае инцидентов база данных может быть восстановлена.

Более наглядное представление модели «как должно быть» показывает её декомпозиция, представленная на рисунке 5. Декомпозиция модели делает процесс понятным, управляемым и эффективным для анализа и оптимизации системы.

Преимущества модели «TO-BE»:

- автоматизация сокращает время на обработку и регистрацию звонков;
- легкость поиска информации для проверяющих служб;
- отсутствие риска потери данных благодаря резервному копированию;
- работа дежурного не прерывается во время проверок, что повышает его эффективность.

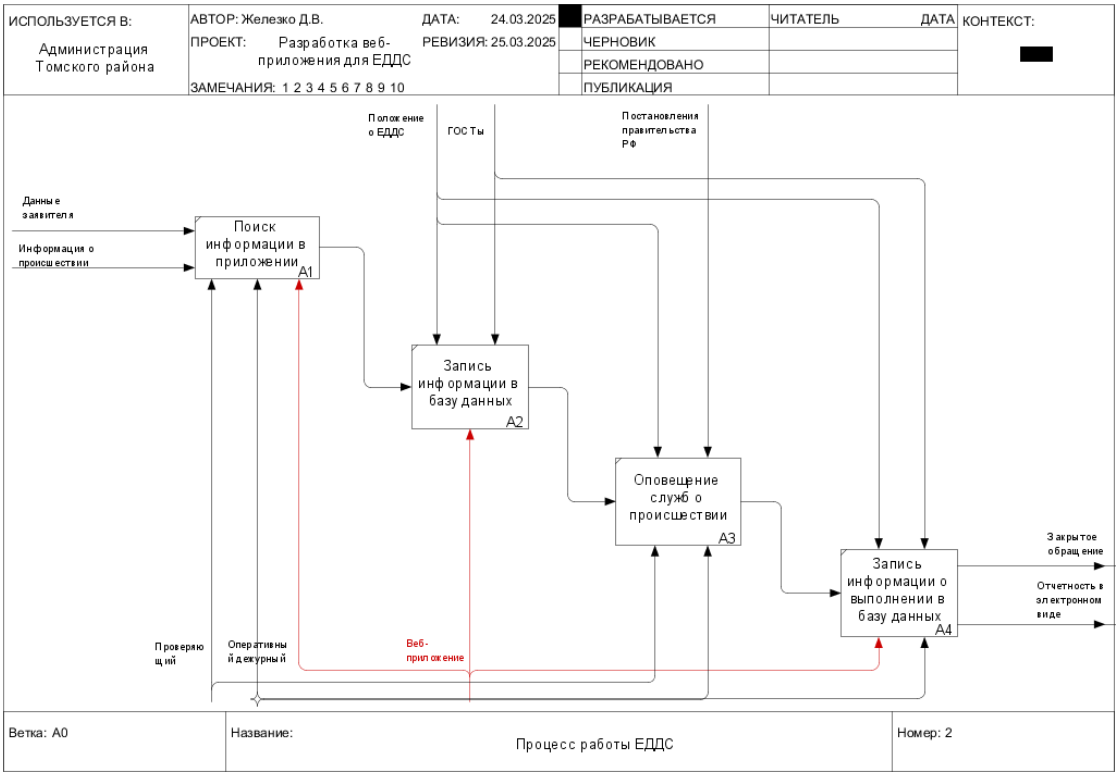


Рисунок 5 – Декомпозиция модели «Как должно быть» отдела ЕДДС

Разработанная модель TO-BE демонстрирует, как автоматизация процессов с помощью веб-приложения способна существенно повысить эффективность работы ЕДДС. Новая система позволяет оперативно регистрировать вызовы, обрабатывать несколько обращений одновременно, исключает риск потери данных и обеспечивает быстрый доступ к информации.

1.5 Обоснование выбора веб-приложения для ЕДДС

Опираясь на современные практики и проблемы, выявленные в текущем (AS-IS) процессе работы ЕДДС, разработка веб-приложения является оптимальным решением [4]. Вот ключевые причины, которые обосновывают данный выбор:

– эффективность и скорость работы – в настоящее время ведение бумажных журналов значительно замедляет работу дежурного,

особенно в моменты пиковых нагрузок или во время проверок. Веб-приложение позволит оперативно вводить данные, обрабатывать несколько звонков одновременно, а также мгновенно передавать информацию по категориям и организациям, которые должны быть уведомлены;

- уменьшение риска потери данных – бумажные журналы подвержены физическим рискам: их можно потерять, испортить или повредить. Использование веб-приложения с централизованным хранением данных в базе данных и резервным копированием минимизирует риск утраты информации. В случае сбоя данные легко восстанавливаются из резервных копий;
- упрощение поиска и отчетности – поиск информации в бумажных журналах занимает много времени, что может привести к задержкам при проверках или запросах. В веб-приложении поиск данных осуществляется мгновенно благодаря возможностям фильтрации и категоризации записей. Кроме того, система может автоматически формировать отчеты для предоставления проверяющим органам;
- поддержка многозадачности – веб-приложение позволит оперативным дежурным одновременно работать с несколькими вызовами, видеть их в очереди, вести записи по каждому из них и своевременно передавать информацию. Это особенно важно для служб, где одновременно может поступать большое количество звонков;
- безопасность и доступ – важным аспектом является информационная безопасность. Веб-приложение позволит ограничить доступ к данным с использованием индивидуальных логинов и паролей для дежурных и начальника отдела. Все действия пользователей будут логироваться, что повышает прозрачность работы. Кроме того, резервное копирование данных обеспечивает сохранность информации;

– экономия ресурсов – переход на веб-приложение сократит использование бумажных носителей, а также уменьшит затраты на архивирование и хранение бумажных журналов. Это также способствует экологической устойчивости и поддержке цифровой трансформации в государственных и муниципальных учреждениях.

Выбор веб-приложения в качестве решения для автоматизации работы отдела обоснован его универсальностью, доступностью и соответствием задачам службы.

1.6 Анализ существующих решений

Для разработки веб-приложения для единой дежурно-диспетчерской службы важно рассмотреть уже существующие решения, используемые в аналогичных системах, и проанализировать их с точки зрения функциональности, надежности и удобства использования [2]. Ниже представлен анализ существующих разработок и систем, которые могут быть применены для решения обозначенной задачи.

Система 112.

Система "112" – это комплексная система экстренной помощи, которая интегрирует работу различных служб (полиция, скорая помощь, пожарные и т.д.) [14]. Она используется в ряде стран Европы и в России. Основные функции этой системы:

– единая база данных происшествий;
– оперативная маршрутизация звонков и передача данных службам реагирования;
– электронный учет всех обращений;
– аналитика по происшествиям и вызовам.

Плюсы:

– интеграция с несколькими экстренными службами;
– высокая надежность и безопасность данных;

– мгновенный доступ к информации для всех участников процесса.

Минусы:

– высокая стоимость внедрения;
– ориентация на большие города и крупные диспетчерские центры, что может быть избыточным для небольших служб.

Центр управления вызовами 911 (США).

Система 911 используется для экстренной связи в США [20]. Она также объединяет несколько служб экстренного реагирования и включает в себя:

– локацию звонящего (по GPS);
– моментальное распределение вызовов по категориям;
– анализ вызовов на ложность и корректность;
– ведение полной истории обращений и возможность быстрой генерации отчетов.

Плюсы:

– высокий уровень автоматизации и интеграция с внешними системами;
– возможность работы с геолокационными данными;
– многоуровневая защита данных.

Минусы:

– требуется развитая инфраструктура для поддержки геолокации;
– высокие затраты на внедрение и поддержание системы.

Системы управления вызовами в коммунальных и аварийных службах.

В российских условиях аналогичные системы применяются в ЖКХ для управления заявками и авариями. Примеры таких решений описаны ниже.

СКДП "Город" – система для автоматизации аварийно-диспетчерских служб в сфере ЖКХ. Она интегрируется с базами данных управляющих компаний и позволяет быстро передавать информацию о заявках и авариях;

AmoCRM и 1С:Диспетчерская служба – системы учета заявок и управления процессами внутри организаций [15] [16].

Плюсы:

– простота в настройке и использовании;
– возможность быстрой адаптации под нужды конкретной организации;
– интеграция с бухгалтерскими и учетными системами.

Минусы:

– ограниченные возможности для масштабирования;
– отсутствие полной интеграции с системами экстренного реагирования.

На основе анализа вышеупомянутых решений можно выделить ключевые требования для разработки веб-приложения для ЕДДС, представленные в таблице 1.

Таблица 1 – Сравнительный анализ готовых решений для ЕДДС

Функция	112	911	СКДП "Город"	AmoCRM/1С:Диспетчерская
Интеграция с экстренными службами	Полная интеграция	Полная интеграция	Частичная интеграция	Нет интеграции
Управление очередью вызовов	Да	Да	Да	Нет
Автоматизация процессов	Полная	Полная	Частичная	Частичная
Работа с геолокацией	Нет	Да	Нет	Нет
Электронная база данных	Да	Да	Да	Да
Генерация отчетов	Да	Да	Да	Да
Стоимость внедрения	Высокая	Высокая	Средняя	Низкая
Время внедрения	Длительное	Длительное	Среднее	Короткое

Веб-приложение для ЕДДС Администрации Томского района должно быть адаптировано под специфику небольшого муниципального

образования, где основная нагрузка идет на управление экстренными происшествиями и взаимодействие с коммунальными службами. Пример систем, таких как "112" или "911", ориентированы на крупные города и требуют значительных ресурсов для внедрения. Это делает их избыточными для муниципальных служб.

Использование CRM-систем (например, AmoCRM или 1С) может оказаться слишком общим для специфических задач ЕДДС. Однако такие решения показывают важность автоматизации заявок и их обработки. Внедрение веб-приложения, специально разработанного для нужд ЕДДС, позволит автоматизировать все процессы приема и обработки вызовов, сохраняя простоту использования для дежурных.

Разработка собственного веб-приложения позволит гибко подстраивать его под потребности службы. Это позволит решить задачи оптимизации процессов обработки вызовов и поиска информации в кратчайшие сроки. При этом система может быть легко масштабирована и дополнена новыми функциями (например, интеграцией с внешними системами).

Важно обеспечить безопасность данных. Пример успешного использования электронных систем в других странах показывает, что веб-приложение с централизованным хранилищем данных, резервным копированием и разграничением прав доступа минимизирует риски утраты информации. Это особенно важно в случае проверок и аудитов.

1.7 Функциональные требований проекта

Функциональные требования – это важная часть проектирования веб-приложения для единой дежурно-диспетчерской службы. Эти требования определяют основные функции, которые система должна выполнять, и включают взаимодействие пользователей с приложением, обработку данных и обеспечение безопасности [6].

Функции приёма и обработки звонков.

Регистрация вызовов:

– оперативный дежурный должен иметь возможность регистрировать входящие звонки с указанием времени поступления, номера телефона, имени заявителя и краткого описания происшествия;
– в случае ложного вызова дежурный может пометить его как ложный.

Обработка нескольких звонков:

– приложение должно поддерживать работу с несколькими вызовами одновременно, предоставляя дежурному интерфейс с очередью поступивших звонков;
– дежурный должен иметь возможность выбрать звонок из очереди и начать обработку.

Классификация происшествий:

– при регистрации вызова дежурный должен иметь возможность выбрать категорию происшествия (например, авария, пожар, чс и т.д.);
– приложение должно предлагать предустановленный список категорий и возможность добавления дополнительных комментариев.

Хранение и управление данными.

База данных происшествий:

– все зарегистрированные звонки и происшествия должны сохраняться в базе данных; информация должна включать дату, время, описание происшествия, категорию и контактные данные.

Сортировка и фильтрация данных:

– дежурный и проверяющие должны иметь возможность быстро искать и фильтровать данные по категориям, дате, типу вызова (истинный или ложный) и другим параметрам.

Хранение истории изменений:

– система должна хранить историю изменений по каждому инциденту

(кто и когда вносил изменения, что именно было изменено).

Резервное копирование данных:

– необходимо регулярно производить резервное копирование базы данных, чтобы избежать утраты информации в случае сбоя системы.

Управление пользователями и доступами.

Регистрация и авторизация пользователей:

– каждый дежурный, начальник отдела и проверяющие должны иметь личные учетные записи для доступа к системе;
– для входа в систему пользователи должны проходить авторизацию по логину и паролю.

Разграничение прав доступа:

– приложение должно предоставлять разные уровни доступа: дежурные могут регистрировать и редактировать звонки, начальник отдела имеет доступ к изменению категорий и некоторых параметров приложения, а также статистике и отчетам; проверяющие могут только просматривать и запрашивать данные.

Журналирование действий:

– веб-приложение должно вести журнал действий всех пользователей: кто вошел в систему, какие данные были изменены, какие запросы к базе данных были выполнены.

Отчёты и аналитика.

Генерация отчетов:

– приложение должно позволять автоматически формировать отчеты по определенным критериям (например, количество происшествий за месяц, количество ложных вызовов);
– отчеты должны быть экспортируемы в формате PDF или Excel для предоставления проверяющим органам.

Статистика по вызовам:

– начальник отдела должен иметь доступ к статистике по звонкам:

количество принятых вызовов, время ответа, количество ложных звонков и т.д.

Мониторинг очереди вызовов:

– приложение должно предоставлять информацию о текущей загруженности дежурных (количество активных вызовов, время обработки).

Интерфейс пользователя.

Удобный интерфейс для дежурного:

– интерфейс должен быть интуитивно понятным и легким в использовании, с минимальным количеством шагов для регистрации и обработки вызовов.

Уведомления и напоминания:

– дежурный должен получать уведомления о поступлении новых звонков, необходимости закрытия старых инцидентов и других важных событиях.

Адаптивный интерфейс:

– веб-приложение должно корректно отображаться на различных устройствах, включая настольные компьютеры, планшеты и мобильные телефоны.

Безопасность данных.

Шифрование данных:

– все передаваемые данные должны быть защищены с использованием протоколов шифрования (например, SSL/TLS) для предотвращения утечек данных.

Резервное копирование:

– система должна регулярно выполнять резервное копирование всех данных для защиты от потерь при сбоях или атаках.

Масштабируемость и производительность.

Масштабируемость системы:

– приложение должно быть разработано с возможностью масштабирования для обработки большого числа вызовов при увеличении нагрузки.

Высокая производительность:

– система должна обрабатывать запросы и поиск данных в течение нескольких секунд, обеспечивая быстрое реагирование.

Технические требования.

Поддержка современных браузеров:

– приложение должно корректно работать во всех популярных веб-браузерах (Google Chrome, Firefox, Яндекс Браузер).

Надежная серверная архитектура:

– серверная часть должна быть надежной и устойчивой к нагрузкам, поддерживая одновременную работу нескольких пользователей без сбоев.

Перечисленные требования помогут разработать веб-приложение, которое повысит эффективность работы дежурно-диспетчерской службы, обеспечив быстрый и безопасный процесс обработки вызовов, хранение данных и взаимодействие с внешними службами.

Выводы по главе 1

В ходе анализа предметной области и объекта исследования были рассмотрены структура и задачи Администрации Томского района, особенности работы отдела по информатизации и кадровому обеспечению, а также деятельность единой дежурно-диспетчерской службы. Было выявлено, что текущая система регистрации и обработки вызовов в ЕДДС основывается на бумажном документообороте, что приводит к низкой оперативности работы, затрудненному поиску информации и высоким рискам потери данных.

В результате исследования были сформированы ключевые проблемы существующей системы (AS-IS) и предложена оптимизированная модель работы (TO-BE), основанная на автоматизации процессов с помощью веб-

приложения.

Анализ процессной модели «AS-IS» позволил выявить ключевые недостатки существующего подхода: отсутствие автоматизации, высокая трудоёмкость при ведении учета, невозможность одновременной обработки нескольких обращений и сложность в обеспечении прозрачности и контроле со стороны руководства. Модель «TO-BE», предложенная в рамках исследования, демонстрирует, как автоматизация процессов на базе веб-приложения способна кардинально улучшить функционирование ЕДДС, повысив скорость, точность и надёжность обработки обращений.

Обоснование выбора веб-приложения как основного инструмента автоматизации подкреплено сравнительным анализом существующих решений, включая международные и отечественные практики. Было установлено, что разработка собственного адаптированного решения под нужды муниципальной службы является наиболее целесообразным вариантом с точки зрения функциональности, стоимости и гибкости внедрения.

Также были сформированы функциональные требования к системе, охватывающие регистрацию и классификацию происшествий, хранение данных, отчётность, разграничение прав доступа, безопасность и адаптивность интерфейса. Это создало прочную основу для последующего проектирования и реализации приложения.

Таким образом, разработка веб-приложения для ЕДДС является актуальной задачей, способствующей повышению эффективности работы службы.

Глава 2 Разработка приложения

Для разработки веб-приложения была выбрана трёхуровневая клиент-серверная архитектура, предполагающая разделение системы (проекта веб-приложения) на три основных уровня:

- клиентская часть (Frontend): отвечает за пользовательский интерфейс и взаимодействие с пользователем;
- серверная часть (Backend): логика обработки данных и взаимодействие с базой данных;
- база данных (Database): хранение и управление данными.

Такое разделение позволяет четко распределить функциональные задачи между уровнями, упрощает разработку и дальнейшее сопровождение системы, а также повышает безопасность и устойчивость проекта [10].

2.1 Клиентская часть

Клиентская часть проекта реализована с использованием следующих технологий:

- Bootstrap 3 с темой Superhero для создания темного интерфейса [17];
- DataTables для отображения табличных данных, таких как отчеты, списки и история действий [19];
- jQuery и jQuery-UI для удобного манипулирования элементами интерфейса и улучшения пользовательского опыта [9];
- Moment.js и Bootstrap-datetimepicker для работы с датами и временем [18] [21].

Выбранные решения обеспечивают несколько ключевых преимуществ.

Ускорение разработки интерфейса - Bootstrap и jQuery предлагают обширные наборы готовых компонентов и функций, это позволяет быстрее собирать интерфейс и фокусироваться на функциональности.

Адаптивность интерфейса - Bootstrap упрощает создание адаптивных

макетов, которые корректно отображаются на различных устройствах, включая мобильные.

Улучшение UX и снижение нагрузки на глаза - тёмная тема Superhero помогает снизить усталость глаз, особенно для пользователей, которые работают с системой длительное время.

2.2 Серверная часть

Серверная часть реализована на PHP 7.4 без использования дополнительных фреймворков, это обеспечивает высокую производительность и гибкость [23]. Выбор Apache в качестве веб-сервера и MySQL в качестве СУБД [22] также обоснован с точки зрения требований и предполагаемых нагрузок на систему.

Стабильность и безопасность. PHP 7.4 предоставляет обновленные функции безопасности, а также возможность реализовывать объектно-ориентированные подходы, улучшающие код и делающие его более читаемым и поддерживаемым.

Производительность. Прямое написание кода без фреймворков снижает накладные расходы, характерные для большинства фреймворков. Это может быть особенно полезно в средах с ограниченными ресурсами.

Сильная интеграция с MySQL. СУБД MySQL широко используется в веб-разработке и хорошо подходит для структурированных данных и построения сложных запросов, что особенно важно для работы с данными веб-приложения [8].

2.3 СУБД и структура базы данных

Для хранения данных использована база данных MySQL, состоящая из 13 таблиц, которые включают пользователей, карточки, записи разговоров и различные справочные данные.

Такая структура позволяет организовать хранение данных в удобном формате. Таблицы и связи между ними структурируют данные, такой вариант облегчает поиск, обновление и удаление информации.

Кроме того, сделать систему расширяемой. Благодаря гибкой структуре базы данных возможно добавление новых таблиц и отношений, если в будущем потребуется расширить функциональность.

А также обеспечить быстрое извлечение данных. MySQL оптимизирована для выполнения запросов с высокими показателями производительности, это особенно полезно для отчетности и поиска по параметрам.

2.4 Обоснование безопасности

Проект использует уровневую модель доступа к данным и строгое разграничение прав между пользователями с различными уровнями доступа. Все запросы на сервер проверяются на наличие разрешений, а данные, передаваемые на сервер, защищаются шифрованием, чтобы предотвратить несанкционированный доступ. В серверной части также используется белый список запросов, что позволяет обработчику отклонять подозрительные или несанкционированные операции.

Как итог, трёхуровневая архитектура проекта и выбранные технологии позволяют создать производительное, масштабируемое и легко поддерживаемое приложение. Разделение задач на клиентскую, серверную части и базу данных способствует поддержке четкой структуры системы и улучшает ее надежность. Использование популярных и проверенных технологий, таких как Bootstrap, jQuery и PHP, облегчает разработку, обеспечивает быструю реакцию системы и хорошую интеграцию с СУБД MySQL для работы с данными, а также предоставляет гибкость для дальнейших улучшений и расширений приложения.

2.5 Построение и описание информационной модели

Для описания информационной модели проекта рассмотрим каждый из перечисленных этапов и задачи, которые необходимо выполнить для создания полноценной и эффективно функционирующей информационной системы. При этом важно учитывать, что проект должен быть ориентирован на четкую структуру данных, удобные интерфейсы для пользователей и интеграцию с внешними системами, если это требуется.

Для построения информационной модели важно разделить данные на несколько уровней, включая концептуальную схему и подсхемы приложений. Концептуальная схема представляет собой общее представление данных и их взаимосвязей в системе. На этом уровне определяется, какие основные сущности и связи между ними существуют в системе, какова их иерархия и структура.

Рассмотрим расширенную диаграмму классов, представленную в приложении А, на рисунке А.1.

Расширенная диаграмма классов [7] отражает архитектуру приложения, разделенного на бэкенд (Backend - серверная часть) и фронтенд (Frontend - клиентская часть), которые взаимодействуют друг с другом через обработчик запросов index. Также показана связь с СУБД MySQL, однако взаимодействие с ней рассматривается кратко, без подробностей.

Для начала рассмотрим серверную части диаграммы.

На сервере расположены классы, реализующие основные функции работы с базой данных и бизнес-логикой приложения. Основной класс бэкенда — edds. Этот и другие классы бэкенда написаны на языке PHP. Класс edds выполняет все действия, связанные с подключением и работой с базой данных MySQL.

Основные компоненты класса edds.

Приватные переменные: dbr_conn, dbr_host, dbr_base, dbr_user, dbr_pass — данные для подключения к базе данных.

Методы:

- connectDB(DB_TYPE) — обеспечивает подключение к СУБД, поддерживает различные типы баз данных;
- create_base — создает базу данных и все необходимые таблицы, если в index значение булевой константы INIT_SITE равно true;
- createCookie(login, msg) — создает куки для пользователей после проверки логина и выводит системное сообщение;
- checkCookie() — проверяет соответствие куки сессии клиента;
- logOut() — завершает сессию пользователя;
- дополнительные методы, такие как get_level(), get_users(), get_fio(username), и другие, обеспечивают выполнение специфических задач, связанных с данными пользователей, справочниками, историей вызовов и отчетами.

Класс modal.

Modal предназначен для вывода данных в модальные окна веб-приложения, обеспечивая интерактивные функции для пользователей. Его методы позволяют работать с данными в контексте различных модальных окон.

Методы:

- datatable(table) — выводит данные для модального окна, используя указанную таблицу. Это может быть полезно для отображения информации из базы данных в клиентской части;
- register() — отображает информацию для модального окна регистрации, предоставляя администратору или начальнику отдела интерфейс для создания новых аккаунтов;
- login() — выводит интерфейс авторизации, обеспечивая доступ к системе для зарегистрированных пользователей;
- changeUser(users) — позволяет администратору или начальнику отдела редактировать учетные записи пользователей. Это включает

изменение информации о пользователе и настройку его прав доступа;

– support() и about() — выводят данные для модального окна технической поддержки и окна "О программе". Эти окна содержат информацию о текущей версии системы и контактные данные для поддержки.

Класс auth.

Auth реализует функционал, связанный с авторизацией и регистрацией пользователей, обеспечивая безопасность данных с использованием криптографических алгоритмов.

Основные методы:

– zrandom_bytes(length) — приватный метод, генерирует случайные значения, необходимые для криптографической обработки данных, таких как пароли. Используется для создания соли при шифровании;

– CalculateVerifier(username, password, salt) — рассчитывает и проверяет криптографические данные пароля. Используется для обеспечения безопасности при хранении паролей;

– GetRegistrationData(username, password) — извлекает криптографические данные для регистрации пользователя, обеспечивая безопасное хранение пароля и других учетных данных;

– VerifyLogin(username, password, salt, verifier) — проверяет соответствие логина и пароля. Этот метод сравнивает введенные пользователем данные с зашифрованными значениями в базе данных.

Класс website.

Website обеспечивает дополнительные функции серверной части веб-приложения, связанные с безопасностью и обработкой данных.

Приватная переменная *key* используется в качестве ключа для шифрования данных, что обеспечивает защиту информации, передаваемой между клиентом и сервером.

Методы:

- encrypt(encrypt) и decrypt(decrypt) — выполняют шифрование и дешифрование данных. Эти методы используют приватный ключ класса для защиты информации от несанкционированного доступа;
- sanitize_output(buffer) — сжимает данные, что позволяет оптимизировать производительность и уменьшить нагрузку на сеть;
- err_handler(errno, errmsg, filename, linenum) — обрабатывает ошибки и записывает их в файл журнала. Файл для записи ошибок указан в приватной переменной php_log. Этот метод позволяет отслеживать ошибки в системе и устранять их;
- return_bytes(sSize) — возвращает размер в байтах для указания максимального объема файла, который может быть загружен с сервера АТС. Этот метод полезен для ограничения размера загружаемых файлов и предотвращения перегрузки сервера.

Обработчик запросов index.

Index выполняет роль центрального обработчика запросов серверной части веб-приложения, связывая клиентскую часть с классами бэкенда. Он отвечает за обработку запросов от клиента, распределяя их по соответствующим классам и методам, а также обеспечивает управление ресурсами сервера, такими как оперативная память и тайм-ауты для запросов. Кроме того, он контролирует доступ к серверу, поддерживая безопасность системы и её стабильность.

Обработчик index имеет связь "один ко многим" с другими классами бэкенда, что обозначает возможность взаимодействия с несколькими экземплярами других классов. В свою очередь, другие классы связаны с index отношением "многие к одному", что указывает на возможность отправлять и обрабатывать запросы от различных компонентов системы.

Index имеет четыре константы:

- INIT_SITE — защищает серверную часть от несанкционированного доступа, запрещая клиентам доступ к внутренним файлам классов.

Это повышает безопасность системы, предотвращая попытки ботов открыть файлы приложения на веб-сервере;

- DB_FIRST — активируется при первой установке веб-приложения, позволяя создать базу данных и таблицы в ней. Эта константа важна для первоначальной инициализации системы;
- FLOGIN_LIMIT — определяет максимально допустимое количество неудачных попыток ввода пароля, предотвращая атаки перебором пароля и повышая безопасность авторизации;
- SITE_LOGIN_LIFE — задаёт время жизни куки, что позволяет контролировать, как долго пользователь остаётся авторизованным в системе. Эта настройка влияет на удобство использования и безопасность веб-приложения, регулируя время автоматического выхода пользователя.

Рассмотрим клиентскую часть диаграммы.

Клиенты взаимодействуют с серверной частью через обработчик index, обеспечивающий связь между фронтендом и бэкендом. На диаграмме класс main фронтенда связан с index по схеме "многие ко многим", что позволяет обмениваться данными между клиентом и сервером.

Класс main.

Main – основной класс клиентской части веб-приложения, обеспечивающий вызов других классов фронтенда и управление функционалом клиентской части.

Константы:

- name_app — задаёт имя веб-приложения;
- back_urls — содержит адреса серверной части для различных сетей (внутренних и внешних);
- name_params — объект для получения параметров от бэкенда;
- option_params — настройки для корректной отправки данных на сервер.

back_url — публичная переменная, указывает текущий адрес серверной части в зависимости от используемой сети.

Методы:

- $() — метод jQuery, выполняющийся после полной загрузки страницы;
- dateMask(elem) — создаёт маску для ввода даты и инициализирует datetimepicker;
- getAuth() — проверяет авторизацию, и при успехе загружает основной функционал приложения;
- jsonToTable(json, response) — преобразует JSON-данные, полученные от серверной части в табличный вариант HTML;
- paramToOption(param, item, response) — возвращает текстовое представление параметров с помощью сопоставление их с константой option_params;
- setCards() — устанавливает интервал проверки звонков на сервере в зависимости от уровня доступа пользователя системы;
- checkCards() — проверяет наличие новых звонков и вызывает модальное окно;
- notifAudioMsg() — оповещает пользователя о новых звонках звуковым уведомлением.

Класс modal.

Modal отвечает за управление модальными окнами (на основе Bootstrap) и вызывается из класса main по отношению "один ко многим".

Методы:

- modalFade(btnText, param, modClose) — отображает модальное окно, наполняемое JSON-данными с сервера;
- loginForm() — вызывает модальное окно для авторизации;
- dataTables() — данный метод формирует таблицы с использованием библиотеки DataTables, поддерживая добавление, редактирование и

удаление строк.

Класс menu.

Класс menu управляет центральным меню веб-приложения, расположенным в верхней части интерфейса, и вызывается через класс main.

Методы:

- getLevel() — запрашивает уровень доступа пользователя и формирует меню;
- menuEvents() — обрабатывает клики по элементам меню, отправляя запросы на сервер и вызывая другие методы.

Класс base.

Base представляет правую часть интерфейса приложения, предназначенную для фильтрации, поиска, просмотра связей и истории карточек. Класс связан с menu по схеме композиции и отношению "один ко многим".

Методы:

- baseLoader(param) — загружает интерфейс блока base;
- baseClickCards() — обрабатывает клики по карточкам в блоке base;
- showHookup() — показывает вкладку связей с карточками;
- showSearch() — показывает вкладку поиска;
- inputSelectSearch(elem, items, selected) — наполняет выпадающий список для параметров поиска, используя константу name_params из main;
- baseCards(items, deleted) — формирует HTML-код карточек в блоке base;
- getBase(response) — создаёт интерфейс базового блока base в формате HTML.

Класс cards.

Cards отвечает за функциональность левого блока приложения, где отображается активная карточка в системе. Связан с классом menu по

композиции.

Методы:

– cardLoader(param) — загружает интерфейс карточки;
– inputAutocomplete(elem, query) — выполняет функционал автодополнения некоторых полей карточки;
– inputSelect(elem, items, selected) — наполняет выпадающий список некоторых полей карточки;
– closeCard() — закрывает карточку и отправляет её в таблицу удалённых карточек;
– hookupCard(card) — выполняет связывание карточек;
– cardHistory(id, items) — формирует историю карточки в формате HTML;
– cardHookup(items) — формирует список связанных карточек в формате HTML;
– newCard(card, tel, audio, deleted) — создаёт список вызовов и карточек для них;
– getCard(response, tel, audio_id) — генерирует HTML-код для отображения интерфейса карточки.

Диаграмма пакетов.

Для удобства структурирования и организации системы рассмотрим диаграмму пакетов [5], представленную на рисунке 6.

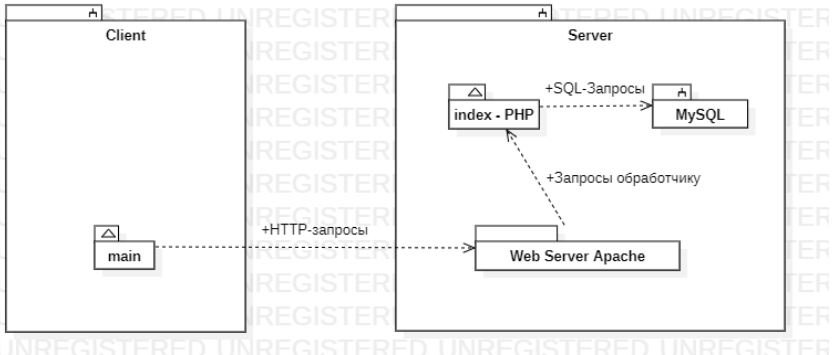


Рисунок 6 – Диаграмма пакетов веб-приложения для ЕДДС

Пакет Client.

Main - Главный класс клиентской части веб-приложения. Он является точкой входа для клиентских запросов и управляет взаимодействием с сервером через веб-сервер Apache.

Пакет Server.

index - PHP:

- выполняет роль обработчика запросов серверной части приложения;
- принимает запросы от клиентского класса main через веб-сервер Apache;
- обрабатывает запросы, выполняя соответствующую логику на серверной стороне;
- может генерировать SQL-запросы для взаимодействия с базой данных.

MySQL:

- служит базой данных для хранения информации, необходимой веб-приложению;
- принимает SQL-запросы от обработчика index - PHP и возвращает результаты, которые используются для обработки и отправки данных клиенту.

Web Server Apache:

- обеспечивает инфраструктуру для связи между клиентом и сервером;
- получает запросы от клиентского класса main и передает их серверному обработчику index – PHP;
- возвращает клиенту обработанные данные от сервера.

Основные взаимодействия.

- Client → Server: клиентский класс main отправляет запросы к веб-серверу Apache, который обрабатывает их и перенаправляет на серверную часть;
- Server (index - PHP) ↔ MySQL: класс index - PHP выполняет SQL-

запросы к базе данных MySQL для получения или сохранения необходимых данных.

Диаграмма развертывания и компонентов.

Для визуализации архитектуры системы на уровне структурных элементов и их взаимодействий воспользуемся диаграммой компонентов [5], представленной на рисунке 7.

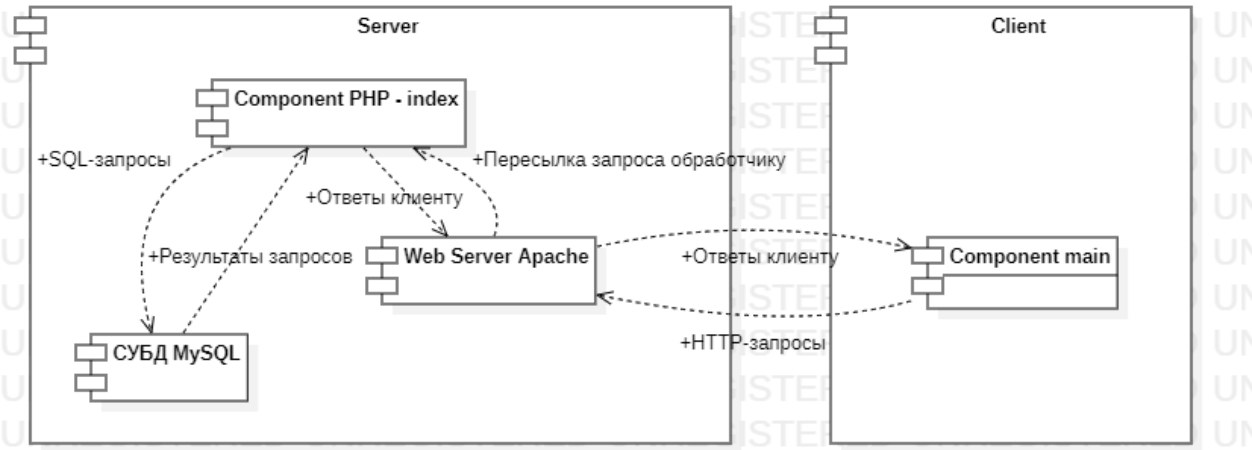


Рисунок 7 – Диаграмма компонентов веб-приложения для ЕДДС

Клиентская часть включает компонент main, который представляет клиентскую часть приложения.

Серверная часть содержит компоненты PHP - index (серверная логика), MySQL (база данных) и Web Server Apache (веб-сервер).

Компонент main отправляет HTTP-запросы к Web Server Apache.

Web Server Apache перенаправляет запросы к PHP - index, который выполняет логику обработки данных.

PHP - index взаимодействует с MySQL для получения и сохранения данных.

Web Server Apache возвращает обработанные ответы клиентскому компоненту main.

Для отображения физического распределения компонентов системы на

аппаратные или виртуальные узлы (узлы развертывания), а также связи между ними нам потребуется диаграмма развертывания [5]. Она описывает, на каких серверах, устройствах и в каких средах развернуты различные компоненты системы и как эти компоненты взаимодействуют друг с другом на физическом уровне. Кроме того, диаграмма развертывания помогает визуализировать, как программные компоненты взаимодействуют с оборудованием, а также выявить аппаратные и сетевые требования для системы. Рассмотрим диаграмму развертывания на рисунке 8.

На данной диаграмме развертывания рассматриваются три основных узла, которые представляют инфраструктуру для работы веб-приложения.

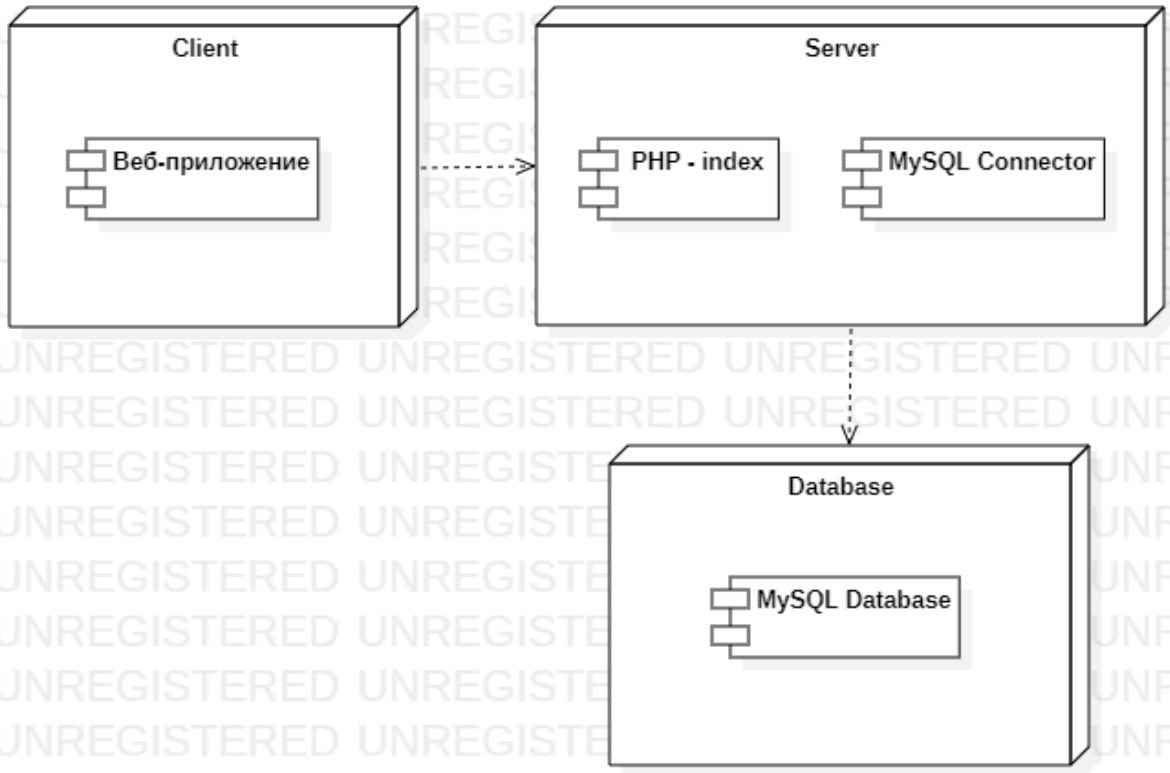


Рисунок 8 – Диаграмма развертывания веб-приложения для ЕДДС

Client (Клиент).

Этот узел представляет пользовательскую часть приложения.

На клиентском узле работает Веб-приложение, которое загружается в

веб-браузере пользователя. Веб-приложение взаимодействует с сервером, отправляя HTTP-запросы и получая ответы для отображения данных пользователю.

Server (Сервер).

Этот узел представляет серверную часть приложения, которая обрабатывает запросы от клиента и взаимодействует с базой данных.

На сервере развернуты два компонента:

– PHP - index – основной обработчик запросов. Этот компонент отвечает за логику работы приложения, обработку пользовательских данных, управление сессиями и т.д;
– MySQL Connector – интерфейс для подключения и обмена данными с базой данных. Он выполняет SQL-запросы, направляемые в базу данных для получения, добавления, изменения или удаления информации.

Database (База данных).

Узел базы данных, на котором хранится вся информация, используемая системой.

На этом узле находится MySQL Database – реляционная база данных, в которой сохраняются структурированные данные веб-приложения, такие как данные пользователей, информация о звонках и другие сущности, необходимые для работы системы.

Диаграммы деятельности.

Для того чтобы понять последовательность действий в определённых ситуациях воспользуемся диаграммами деятельности для двух прецедентов – поступление звонка и связывание карточки.

Рассмотрим диаграмму деятельности для прецедента «Поступление звонка», она иллюстрирует процесс обработки входящего звонка в системе, и представлена на рисунке 9.

– процесс начинается с запроса пользователя клиентом веб-приложения к серверу о поступлении новых звонков;

– сервер получает информацию от базы данных;
– результат проверки формируется и обрабатывается сервером;
– если звонков нет, процедура завершает свою работу;
– если звонки есть, на клиент приходит уведомление с модельным окном и списком звонков, а также звуковым уведомлением;
– пользователь может закрыть модальное окно клиента со списком – это нужно, если уже вызов поступил и клиент должен заполнить предыдущую карточку. В таком случае деятельность завершается;
– пользователь может выбрать в клиенте необходимый звонок для создания карточки;
– пользователь заполняет в клиентской части необходимые поля карточки и отправляет их на сервер;
– сервер обрабатывает запрос из базы данных;
– дальнейшая работа сервера разделяется на 2 потока – один производит запись истории карточки в базу данных а другой отправляет результат сохранения карточки клиенту;
– клиент закрывает карточку пользователя, показывая тем самым что данные сохранены;
– на этом обработка данного прецедента завершается.

Теперь рассмотрим прецедент – связывание карточки. Диаграмма деятельности для прецедента «Связывание карточки» описывает процесс соединения или связывания карточки с другими карточками в системе. Эта функция может понадобиться для управления связанными объектами, такими как контакты, задачи, звонки или записи в системе управления. Диаграмма помогает понять шаги, которые выполняет система и пользователь, когда необходимо связать одну карточку с другой. Диаграмма проиллюстрирована на рисунке 10.

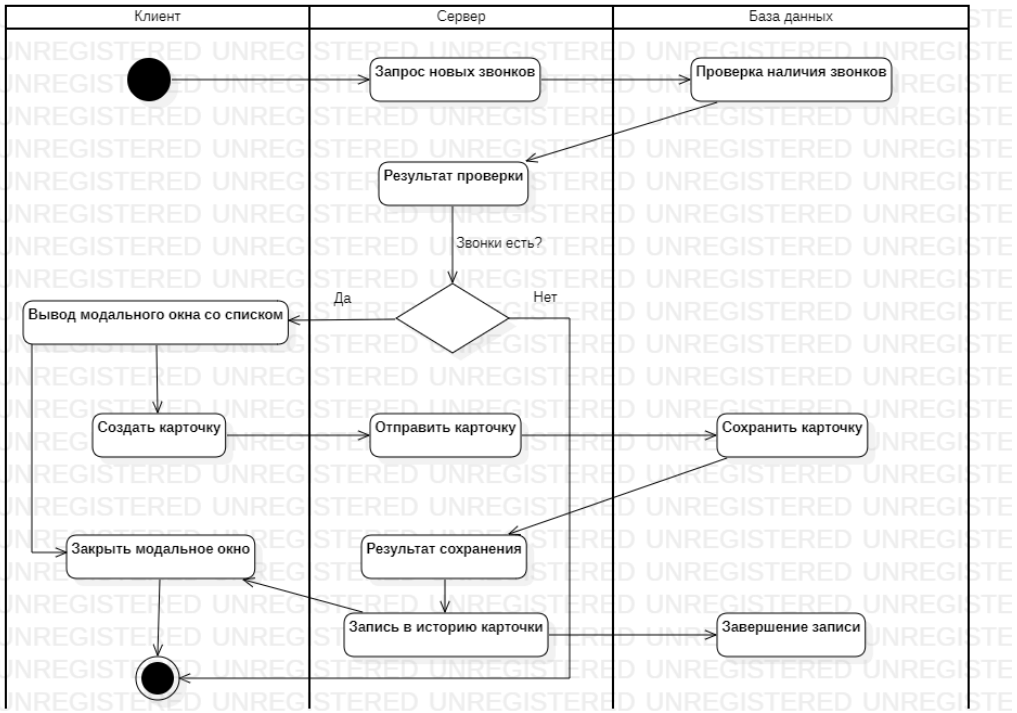


Рисунок 9 – Диаграмма деятельности для прецедента «Поступление звонка»

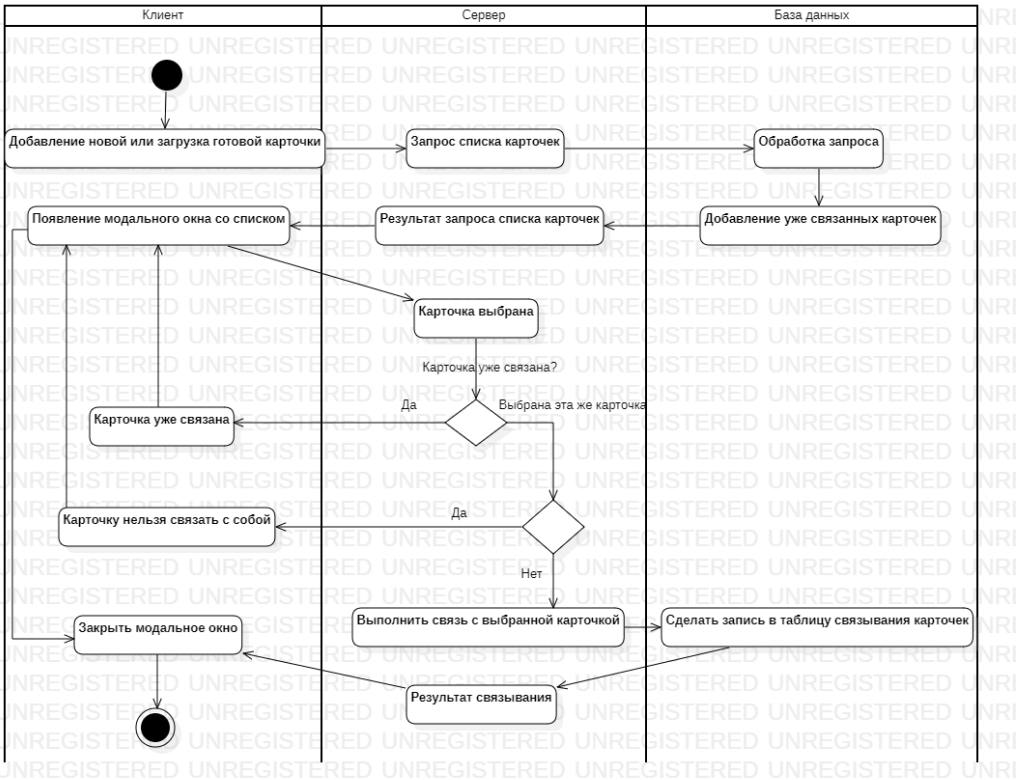


Рисунок 10 – Диаграмма деятельности для прецедента «Связывание карточки»

– процесс начинается, когда пользователь добавляет новую карточку, или же загружает готовую карточку. Для связывания есть специальная кнопка;
– кликая на кнопку «связать» клиент отправляет на сервер запрос;
– сервер запрашивает у базы данных список карточек для связывания;
– база данных кроме списка карточек добавляет к результату запросу карточки, которые уже связаны, т.е. из таблицы связанных карточек;
– пользователь наблюдает как в клиенте появляется модальное окно со списком карточек;
– пользователь делает выбор и отправляет клиентом на сервер свой выбор;
– сервер производит обработку запроса и сравнивает его со списком уже связанных карточек;
– если карточка уже связана, пользователь получает уведомление в веб-приложении и процедура возвращается на этап появившегося модального окна со списком карточек;
– в другом случае сервер производит проверку на соответствие той же карточки, которую заполняет пользователь в клиенте;
– если проверка выявляет соответствие, пользователь получает уведомление об этом, и процедура возвращается на этап появившегося модального окна со списком карточек;
– в ином случае сервер отправляет запрос на связывание карточек в базу данных;
– база данных обрабатывает запрос и возвращает серверу результат.
– сервер отправляет результат клиенту;
– пользователь видит, как модальное окно закрывается, понимая при этом, что связь с карточкой завершена;
– на этом обработка прецедента завершается.

Диаграммы последовательности.

Кроме диаграмм деятельности, для того, чтобы показать взаимодействие между объектами в рамках выполнения прецедентов воспользуемся диаграммами последовательности [5].

Рассмотрим диаграмму последовательности для прецедента «Поступление звонка». Она представлена на рисунке 11 и поможет лучше понять взаимодействие между объектами в системе

Диаграмма показывает последовательность взаимодействий между пользователем, клиентом, сервером и базой данных для обработки входящего звонка. В зависимости от наличия звонков, пользователь либо закрывает уведомление, либо выбирает звонок и заполняет карточку.

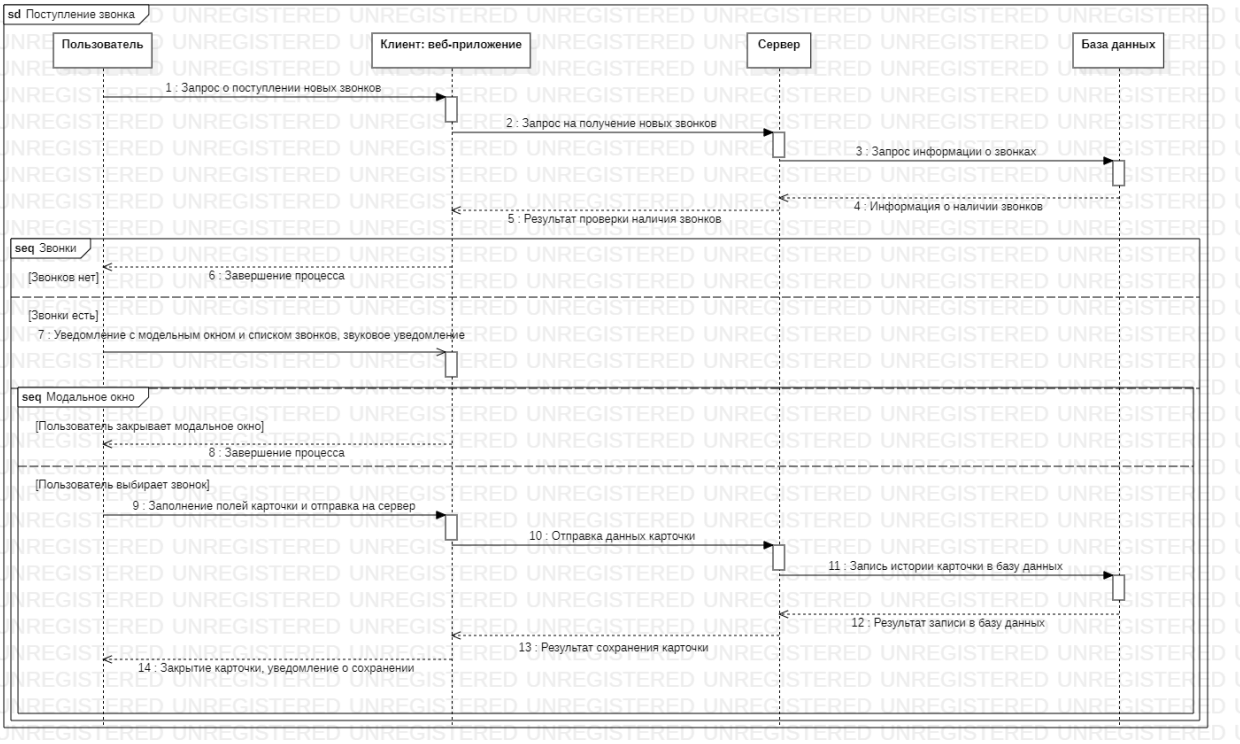


Рисунок 11 – Диаграмма последовательности для прецедента «Поступление звонка»

Теперь рассмотрим диаграмму последовательности для прецедента «Связывание карточки», представленную на рисунке 12. Диаграмма описывает последовательность действий для связывания карточек. Здесь реализована проверка, чтобы избежать дублирования связей и связанных с

этим уведомлений. Пользователь получает возможность выбрать нужную карточку для связывания и видит результат после успешного связывания.

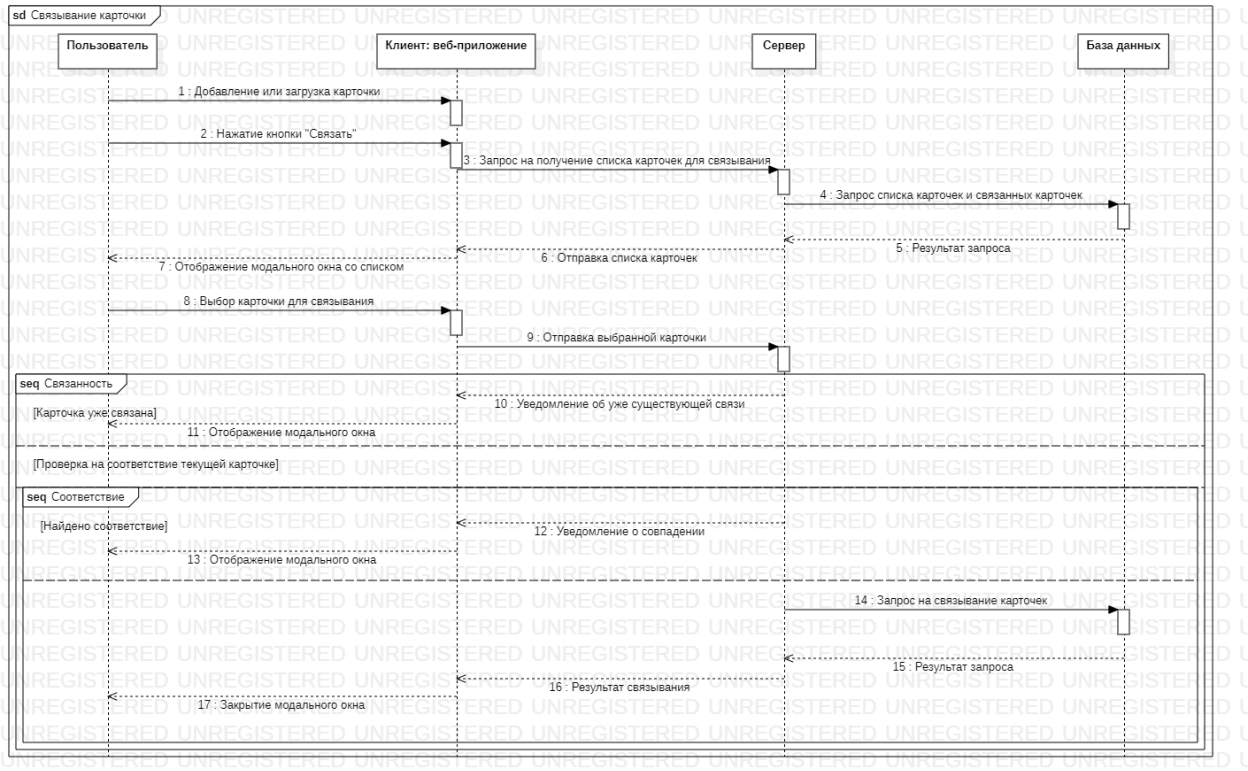


Рисунок 12 – Диаграмма последовательности для прецедента «Связывание карточки»

Описанные диаграммы могут быть полезны для визуализации шагов и обмена сообщениями между компонентами системы при выполнении соответствующих прецедентов.

Диаграмма вариантов использования.

Такая диаграмма применяется для моделирования взаимодействия пользователей с системой, определяя, какие функции доступны различным ролям. Диаграмма представлена на рисунке 13.

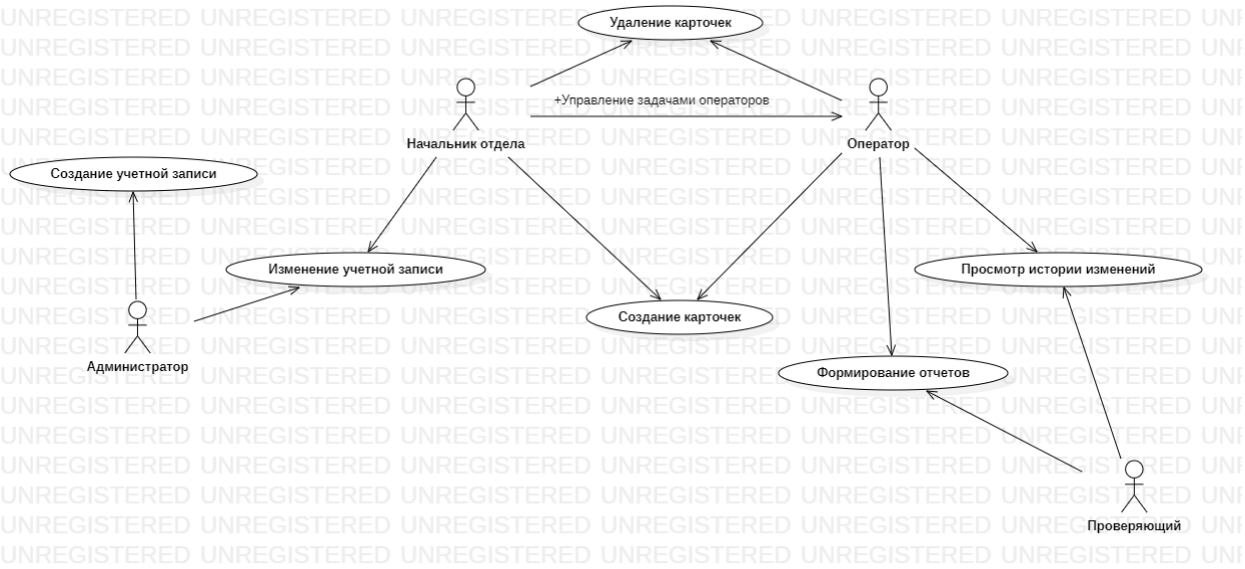


Рисунок 13 – Диаграмма вариантов использования веб-приложения для ЕДДС

Описание диаграммы:

- администратор управляет пользователями (создание, изменение, удаление учетных записей;
- начальник отдела управляет задачами операторов, анализирует данные и может управлять карточками, в период отсутствия операторов; кроме того, начальник отдела редактирует учетные данные пользователей после увольнения и приёма операторов;
- проверяющий проверяет карточки, фильтрует их и анализирует историю изменений;
- оператор работает с карточками: создает, редактирует, связывает, ищет данные.

Диаграмма вариантов использования служит основой для дальнейшего проектирования и разработки, обеспечивая наглядное представление функциональных возможностей системы.

Диаграмма потоков данных.

Наконец, чтобы определить, как данные перемещаются внутри системы, какие процессы обрабатывают эти данные, какие внешние и

внутренние сущности взаимодействуют друг с другом потребуется диаграмма потоков данных (DFD Диаграмма) [5]. Диаграмма представлена на рисунке 14.

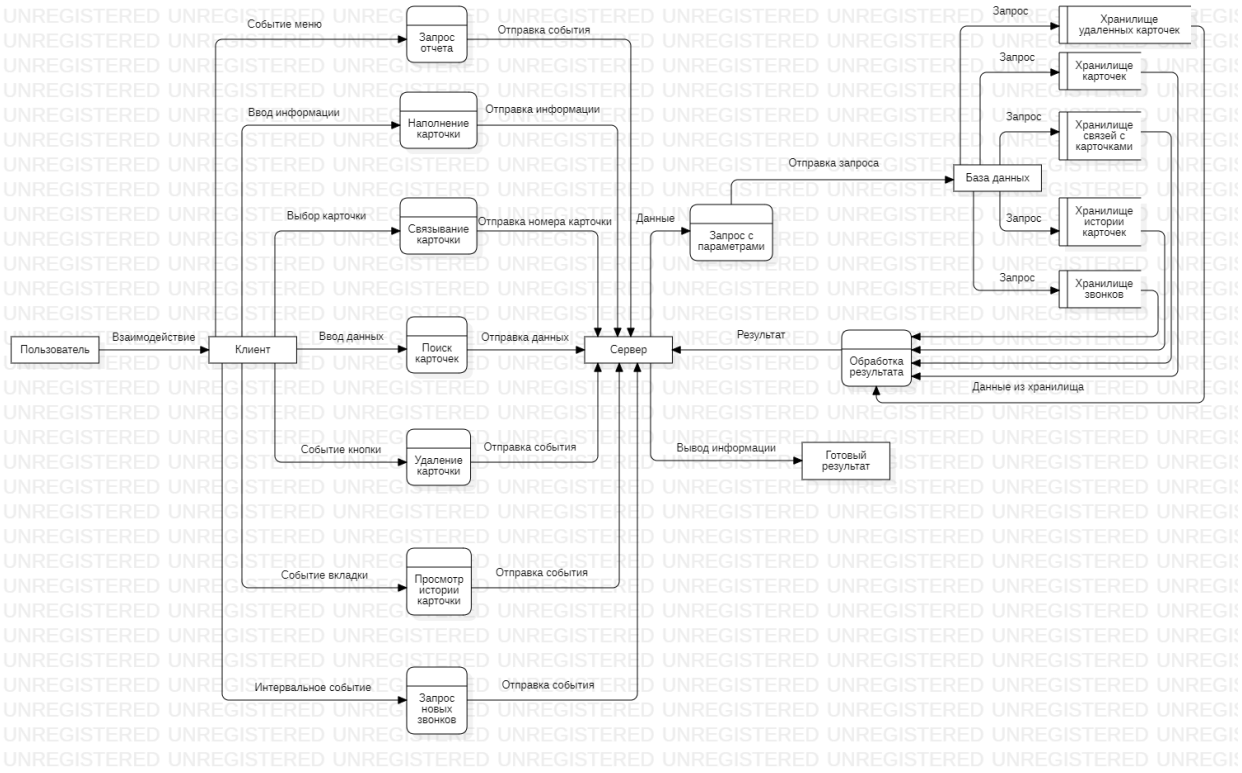


Рисунок 14 – Диаграмма потоков данных веб-приложения для ЕДДС

Диаграмма потоков данных описывает основные процессы и потоки данных в разрабатываемом веб-приложении.

Пользователь – внешняя сущность, взаимодействующая с клиентом. Пользователь взаимодействуя с клиентом отправляет данные на сервер и получает результат.

Клиент – внешняя сущность, взаимодействующая с сервером. Клиентская часть имеет функционал взаимодействия с сервером через различные протоколы.

Запрос отчета – процесс, инициируемый пользователем, выполняющийся с помощью клиента. Клиент отправляет событие меню на сервер. Сервер взаимодействует с хранилищем карточек через базу данных и

специальный запрос. Полученный результат возвращается клиенту, клиент выводит информацию в виде отчета в модальном окне.

Наполнение карточки – процесс, инициируемый пользователем, выполняющийся с помощью клиента. Когда клиент отправляет информацию на сервер, он взаимодействует с хранилищем карточек с помощью базы данных. Полученную информацию сервер обрабатывает и отправляет готовый результат.

Связывание карточки – процесс. Связывание карточки рассматривалось при составлении диаграмм последовательности и деятельности. Сервер запрашивает данные запросом через базу данных. И кроме списка карточек в результате будет ещё информация уже ранее связанных карточек из хранилища связей с карточками. Во время исполнения процесса база данных обращается к двум хранилищам, чтобы объединить готовый результат.

Поиск карточки – процесс, пользователь указывает в клиенте необходимый параметр для поиска и отправляет его на сервер. Сервер обрабатывая запрос от клиента обращается к базе данных, а база данных уже к хранилищу карточек. Полученные данные обрабатываются сервером и результат возвращается клиенту.

Удаление карточек – процесс. Сервер получив такой запрос от клиента обращается к базе данных, а она в свою очередь к хранилищу удаленных карточек и производит запись в этом хранилище, возвращая результат записи. Сервер обрабатывает результат и отправляет его клиенту. Клиент инициирует закрытие карточки, визуализируя его пользователю.

Просмотр истории карточки – процесс, который инициализирует пользователь в клиентской части веб-приложения. Сервер, получив запрос от клиента обращается к базе данных, а она в свою очередь к хранилищу истории карточек. Полученный результат обрабатывается сервером и выводится клиентом пользователю.

Запрос новых звонков – процесс, подробно описанный на диаграммах последовательности и деятельности. Сервер при обработке запроса от

клиента обращается к базе данных. База данных считывает информацию с хранилища звонков и полученный результат возвращает серверу. Сервер обрабатывает результат и отправляет его клиенту, который уже преобразовывает данные в читаемый результат и выводит необходимую информацию в модальном окне.

Сервер – внешняя сущность. Сервер находится во взаимодействии с клиентом и базой данных. Сервер принимает на себя всю нагрузку по обработке информации от клиента и базы данных генерируя необходимые запросы и обрабатывая полученные данные. Кроме того, сервер генерирует результат для клиента в формате JSON, который потом преобразуется клиентом в понятный пользователю вид.

Запрос с параметрами – процесс, исполняемый сервером и отправляемый базе данных для обработки. Параметры зависят от поступивших данных клиента.

База данных – внешняя сущность, которая обрабатывает запросы и обращается к хранилищам, полученные результаты запросов к хранилищам поступают серверу для обработки через процесс «Обработка результата».

Обработка результата – процесс, исполняемый базой данных и полученный от хранилищ базы данных. Результат обрабатывается и преобразуется в формат JSON [11]. Который занимает меньший объём информации и может быть сжат специальными средствами для оптимизации работы системы.

Готовый результат – внешняя сущность, создаваемая сервером. Результат отправляется клиенту, после чего клиент преобразует его в читаемый пользователем формат в виде таблиц, модальных окон с наполнением или обыкновенного текста.

2.6 Разработка комплекса программ для реализации алгоритмов обработки данных

На этапе разработки комплекса программ для реализации алгоритмов обработки данных необходимо создать набор программных модулей, которые смогут выполнять поставленные задачи по обработке данных в соответствии с требованиями проекта [3]. Этот этап предполагает:

- определение и описание алгоритмов обработки данных. Уточнение алгоритмов, которые будут использованы для обработки информации, будь то сортировка, фильтрация, анализ или иные действия с данными;
- разработка кода программных модулей. Написание программного кода, реализация логики обработки данных, а также настройка взаимодействия между компонентами системы;
- интеграция с базой данных и серверной частью. Настройка связи между алгоритмами обработки данных и источником данных — базой данных, сервером и другими компонентами системы. Важно учесть особенности обмена данными и формат запросов, если предполагается взаимодействие с другими частями приложения.

Определение и описание алгоритмов обработки данных входящих звонков.

Цель: определение и отображение новой информации о звонках для операторов системы.

Входные данные: JSON-данные от сервера, включающие ID звонка, номер телефона и время звонка.

Выходные данные: обновленный список звонков на клиентской стороне, включающий новые или измененные записи.

Основные этапы:

- клиент отправляет запрос на сервер для проверки новых звонков;
- сервер получает данные о звонках из базы данных и отправляет их

клиенту;

– данные преобразуются в таблицу и отображаются в модальном окне на клиентской стороне;
– если звонки присутствуют, клиент активирует звуковое уведомление и отображает модальное окно со списком звонков;
– если данные не получены, клиент не выводит модальное окно и пользователь продолжает работу в штатном режиме.

Разработка кода программных модулей входящих звонков.

Для выполнения данного алгоритма был разработан код как для клиентской (фронтенд) части на языке JavaScript, так и для серверной (бэкенд) части на языке PHP.

Код для фронтенда представлен на рисунке 15.

```
199  function checkCards(){
200      let oldcards = localStorage.getItem(name_app + '_cards');
201      $.post(back_url, {datatable: 'audio'}, function(response){
202          let newcards =  response['data'][response['data'].length - 1]['id'];
203          if(newcards > oldcards){
204              localStorage.setItem(name_app + '_cards', newcards);
205              modalFade('Карточку из списка');
206              notifAudioMsg();
207          }
208          if(oldcards == null) localStorage.setItem(name_app + '_cards', newcards);
209      });
210  }
```

Рисунок 15 – Код на языке JavaScript для клиента – получение входящих звонков

Во время интерпретации кода клиентское веб-приложение проверяет последний идентификатор звонка в локальном хранилище и вносит его в переменную oldcards.

Далее, клиент отправляет POST-запрос на сервер для получения последнего идентификатора звонка, и если запрос завершается успешно, то заносит полученное число в переменную newcards.

После этого клиент производит сравнение чисел в oldcards и newcards. Если значение, полученное от сервера больше значения в локальном

хранилище, тогда вызывается модальное окно, появляется звуковое уведомление, а так же производится запись большего значения в локальное хранилище.

Если локальное хранилище пусто, клиент производит запись значения, полученного от сервера.

Код для бэкенда представлен на рисунке 16.

```
390 if(isset($_POST['datatable'])){
391     $allow = array(
392         'edited',
393         'audio',
394         'hookup_card',
395         'report_cards_week',
396         'report_cards_month',
397         'report_cards_year',
398         'report_cards',
399         'report_cards_disable',
400         'report_hookup',
401         'report_calls',
402         'digests_settlements',
403         'digests_villages',
404         'digests_categories',
405         'digests_organizations',
406         'digests_problems',
407         'digests_problems2',
408         'digests_states',
409     );
410 
411     if(!in_array($_POST['datatable'], $allow)) die(header('HTTP/1.0 404 Not Found'));
412     header('Content-type: application/json; charset="UTF-8"');
413 
414     if($_POST['datatable'] == 'edited' && isset($_POST['action']) && $_POST['action'] == 'edit'){
445 
446     if($_POST['datatable'] == 'edited' && isset($_POST['action']) && $_POST['action'] == 'create'){
482 
483     if($_POST['datatable'] == 'edited' && isset($_POST['action']) && $_POST['action'] == 'remove'){
502 
503     if($_POST['datatable'] == 'audio'){
504         if(edds::get_level() < 0) die(header('HTTP/1.0 403 Forbidden'));
505         $json = edds::get_calls();
506         echo json_encode($json);
507         exit();
508     }
```

Рисунок 16 – Код на языке PHP для сервера – получение входящих звонков

После отправки POST-запроса от клиента веб-сервер перенаправляет запрос программе-обработчику (в данном случае PHP).

PHP обработчик index обнаруживая POST-запрос сравнивает его с белым списком запросов из массива $allow.

Если совпадения с белым списком не обнаруживается, тогда программа завершает работу, выдав ошибку на языке HTTP – код ответа 404 (не

найдено).

В ином случае работа программы продолжается, и на этом этапе к ответу будет присвоен HTTP заголовок для вывода JSON данных.

Далее, обработчик, повторно убедившись, что ему поступил верный текст POST-запроса, производит проверку уровня доступа к системе, обращаясь к методу get_level() класса edds. Если уровень к системе не соответствует заявленному, тогда программа завершается, выдав ошибку на языке HTTP – код ответа 403 (доступ запрещён).

Пройдя проверку по уровню доступа, сервер обращается к классу edds и методу get_calls() для получения списка звонков.

Последним этапом обработчик формирует полученные от метода get_calls() данные в формат json и выводит их клиенту. После чего работа обработчика завершается стандартной функцией PHP – exit().

Интеграция с базой данных и серверной частью.

Для интеграции с базой данных, в данном случае с СУБД MySQL служит класс edds и метод get_calls(). Код метода представлен на рисунке 17.

Данный метод имеет параметр $param, с помощью которого будет задан дополнительный параметр обращения к базе данных. Параметр имеет уже значение по умолчанию, которое будет использоваться при вызове метода без указания параметра. Стандартное значение параметра подразумевает дополнение к выборочному запросу базы данных, где указывается пустая карточка и время поступления звонка.

Кроме того, у метода есть глобальная переменная, необходимая для формирования пути к аудиозаписи для возможности её воспроизведения.

Метод содержит условие пустого параметра, в таком случае метод выводит данные, необходимые для отчета по входящим звонкам.

Переменная $date_time внутри метода высчитывает дату и время в формате Unix timestamp прошедших суток от текущей даты и времени.

Далее, метод обращается к базе данных с необходимым запросом, для формирования данных о звонках в массив $items.

```
public static function get_calls($param = "WHERE `date_time` > :date_time AND `card_id` = '0'"){
    global $SITEURL;
    $report = empty($param);
    $db = self::connectDB();
    $date_time = (time() - 86400); // 86400 - за сутки
    $result = $db->prepare("SELECT * FROM `audio` {$param} LIMIT 1000000");
    $result->bindParam(':date_time', $date_time);
    $result->execute();
    $items = array();
    foreach($result->fetchAll(PDO::FETCH_ASSOC) as $row){
        $card = '<a href="#" class="btn btn-xs btn-success" onclick="return newCard(\''.$row['card_id'].'\', \''.$row['chat_id'].'\', \''.$row['id'].'\');">';
        $file = !empty($row['audio']) ? '<audio data="'.$row['card_id'].'" src="'.$SITEURL.'/edds_region_tomsk/back?getAudio='.$row['file'].'" preload="none" controls>' : '';
        $items[] = array(
            'id' => $row['id'],
            'card_id' => (!empty($row['card_id']) ? $card.'№ '.$row['card_id'].'</a>' : ($report ? '—' : $card.'Добавить'.'</a>')),
            'chat_id' => $row['chat_id'],
            'file' => $file,
            'date_time' => (!empty($row['date_time']) ? date('Y-m-d H:i:s', $row['date_time']) : $row['date_time']),
        );
    }
    $head = array(
        array('data' => 'id', 'title' => '№ пп', 'className' => 'on-selected'),
        array('data' => 'card_id', 'title' => 'Карточка'),
        array('data' => 'chat_id', 'title' => 'Телефон'),
        array('data' => 'file', 'title' => 'Запись'),
        array('data' => 'date_time', 'title' => 'Поступил'),
    );
    $edit = array(
        array('name' => 'id', 'label' => '№ пп', 'type' => 'hidden'),
    );
    $result = null;
    $db = null;
    return array(
        'cols' => !empty($items) ? $head : array(array('data' => 'chat_id', 'title' => 'Телефон')),
        'data' => !empty($items) ? $items : array(array('chat_id' => 'Ничего не найдено')),
        'edit' => $edit,
        'order' => 'desc',
        'buttons' => ($report ? array('copy', 'csv', 'excel', 'print') : array()),
    );
}
```

Рисунок 17 – Код метода get_calls() – получение входящих звонков

При формировании массива, во время перебора полученных данных от СУБД происходит замена некоторых данных. Если в строке таблицы отсутствует путь до файла аудиозаписи, тогда в массив записывается пустая строка, в ином случае HTML код для прослушивания аудиозаписи разговора.

Так же при формировании массива учитывается привязка звонка к карточке, в основном это требуется для отчетности, в таком случае массив заполняется прочерками при отсутствии связи звонка и карточки.

После формирования массива $items метод формирует ещё несколько массивов, необходимых для работы библиотеки DataTables. Эти данные требуются при создании табличной формы DataTables.

Полученные массивы сводятся в один многомерный массив и выводятся методом. Перед выводом метод закрывает подключение к базе данных и очищает переменные, для экономии памяти сервера.

Определение и описание алгоритмов обработки данных связывания карточек.

Цель: связывание текущей карточки с другими, чтобы установить логическую связь между связанными данными.

Входные данные: запрос от клиента, включающий ID текущей карточки и выбранной карточки для связи.

Выходные данные: обновленный список связанных карточек в базе данных и на клиенте.

Основные этапы:

– клиент отправляет запрос на сервер для получения доступных карточек для связывания;
– сервер проверяет, связана ли уже текущая карточка с выбранной и не совпадает ли она с текущей;
– если связь отсутствует, сервер создает новую запись связи в базе данных;
– результат отправляется клиенту, модальное окно закрывается и список связанных карточек обновляется.

Разработка кода программных модулей связывания карточек.

Для выполнения указанного алгоритма был разработан код для фронтенда на языке JavaScript, и для бэкенда на языке PHP.

Код для фронтенда представлен на рисунке 18.

В клиенте функция, отвечающая за функционал связывания, имеет параметр – идентификатор карточки, по умолчанию этот параметр равен нулю.

```
148 function hookupCard(card = '0'){
149     if(card == '0') return false;
150     $.post(back_url, {hookupid: $('#input-newcard').attr('data'), hookupcard: card}, function(data){
151         if(data == 'true'){
152             $('#base').slideUp('fast');
153             setTimeout(function(){
154                 baseLoader();
155                 setTimeout(function(){
156                     showHookup();
157                     $('#modalFade .modal-header .close').click();
158                 }, 200);
159             }, 200);
160         }else{
161             alert(data);
162         }
163     });
164     return false;
165 }
```

Рисунок 18 – Код на языке JavaScript для клиента – связывание карточек

Если во время вызова функции параметр равен нулю или не задан, тогда функция возвращает отрицательное булевое значение, не позволяющее модальному окну закрыться, или выполнить какое-то действие по отправке данных на сервер.

В ином случае клиент исполняет POST-запрос с указанием карточки которую надо связать (hookupid) с карточкой которую связываем (hookupcard).

После отправки запроса клиент будет ожидать от обработчика текстового значения «true», означающее, что запрос выполнен корректно.

В этом случае происходит обновление информации базового блока, а так же открытие вкладки со связанными карточками. Модальное окно в этом случае закрывается.

Для корректного и визуально правильного отображения информации веб-приложения существует задержка перед выполнением обновления базового блока и отображения вкладки связей с карточками с помощью стандартной функции JavaScript – setTimeout().

Если же сервер обнаружил какую то ошибку или несоответствие, клиент уведомляет об этом пользователя с помощью всплывающего сообщения стандартной JavaScript функции alert(). Сообщение блокирует дальнейшую работу веб-приложения и требует от пользователя обязательного ознакомления с текстом сообщения.

Код для бэкенда представлен на рисунке 19.

После обнаружения обработчиком на PHP POST-запросов hookupid и hookupcard производится проверка клиента, приславшего запрос на соответствие необходимого доступа. Если доступ не соответствует, обработчик отправляет код HTTP 403.

При дальнейшей интерпретации кода POST-запросы присваиваются переменным для более удобной работы с кодом.

Если во время проверка выяснится, что числовые значения запросов совпадает, тогда обработчик завершит работу сообщением «Карточку нельзя

связать с собой». Это будет значить, что клиент пытается связать карточку с той же карточкой, которую заполняет.

```php
if(isset($_POST['hookupid']) && isset($_POST['hookupcard'])){
    if(edds::get_level() < 0) die(header('HTTP/1.0 403 Forbidden'));
    $card_id = !empty($_POST['hookupid']) ? $_POST['hookupid'] : null;
    $card_hookup = !empty($_POST['hookupcard']) ? $_POST['hookupcard'] : null;

    if($card_id == $card_hookup) die('Карточку нельзя связать с собой');

    $db = edds::connectDB();

    $select = $db->prepare("SELECT * FROM `hookup` WHERE `card_id` = ? AND `card_hookup` = ?");
    $select->execute(array($card_id, $card_hookup));
    $item = $select->fetch(PDO::FETCH_ASSOC);
    $select = null;

    if(!empty($item)) die('Такая связь уже существует');

    $insert = $db->prepare("INSERT INTO `hookup` (`card_id`, `card_hookup`) VALUES (:card_id, :card_hookup)");
    $insert->bindParam(':card_id', $card_id);
    $insert->bindParam(':card_hookup', $card_hookup);
    $result = $insert->execute();
    $insert = null;
    $db = null;

    $history = array('card_id' => $card_id, 'card_hookup' => $card_hookup);
    edds::add_history($card_id, 'Добавление связи', json_encode($history));

    echo !empty($result) ? 'true' : 'false';
    exit();
}
```

Рисунок 19 – Код на языке PHP для сервера – связывание карточек

После прохождения этой проверки обработчик выполняет подключение к базе данных, для получения информации о существующей связи.

Если такая связь в таблице обнаруживается, тогда обработчик завершает работу сообщением «Такая связь уже существует».

В ином случае обработчик производит запись в базу данных с указанием связи карточек.

После завершения записи в базу подключение к базы данных завершается, а некоторые переменные очищаются для экономии оперативной памяти сервера.

Дальше происходит процесс формирования данных для добавления истории карточки с сообщением о создании новой связи с карточкой.

И с помощью метода add_history класса edds производится добавление новой записи в историю карточки.

Завершается работа обработчика выводом результатов записи в таблицу связей карточек. Если запись была произведена, тогда результирующее значение не может быть пустым и обработчик выведет по условиям кода текстовое значение «true». В ином случае выводится «false».

Интеграция с базой данных и серверной частью.

Интеграция с базой данных отображения в обработчике, на рисунке 14. Ещё одна интеграция происходит при добавлении истории карточки. Она осуществляется с помощью метода add_history класса edds. Данный метод представлен на рисунке 20.

```
393     public static function add_history($card_id = '0', $description = '', $data = '', $date_time = null){
394         $db = self::connectDB();
395         $date_time = empty($date_time) ? time() : $date_time;
396         $result = $db->prepare("INSERT INTO `history` (`card_id`, `description`, `data`, `date_time`) VALUES (:card_id, :description, :data, :date_time)");
397         $result->bindParam(':card_id', $card_id);
398         $result->bindParam(':description', $description);
399         $result->bindParam(':data', $data);
400         $result->bindParam(':date_time', $date_time);
401         $result->execute();
402         $last_id = $db->lastInsertId();
403         $result = null;
404         $db = null;
405         return $last_id;
406     }
```

Рисунок 20 – Интеграция с базой данных – метод добавления истории карточки

Метод имеет 4 параметра, три из которых желательны к указанию, в порядке слева на права это – идентификатор карточки, описание события, данные в формате JSON. Последний параметр – это дата и время в формате Unix timestamp, и может быть задан автоматически по текущим часам сервера. Конечным этапом добавления карточки служит возврат идентификатора записи в таблице истории карточек.

2.7 Разработка интерфейса конечного пользователя

Основные элементы интерфейса состоят из трех ключевых блоков: верхний блок с меню, левый блок с карточками и правый блок с вкладками управления.

Верхний блок (Меню).

Расположение: верхняя часть страницы, содержит логотип, ФИО пользователя и пункт выхода из аккаунта (рисунок 21).

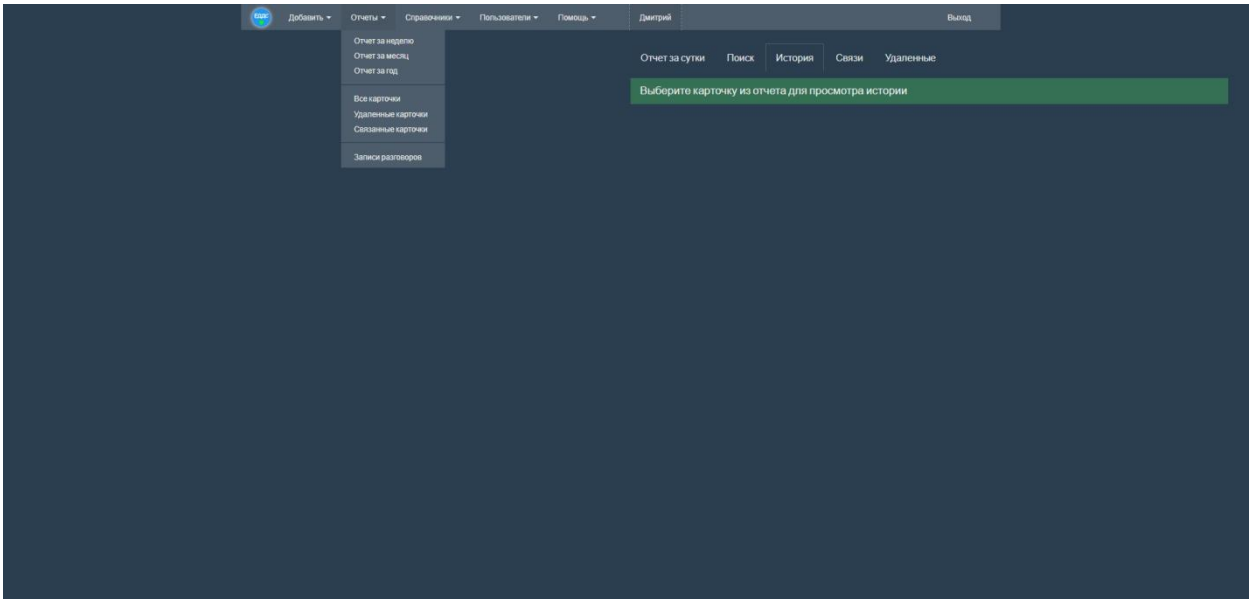


Рисунок 21 – Меню веб-приложения для ЕДДС

Функционал меню:

– добавить - пункты для создания новой карточки или выбора карточки из списка;
– отчеты - подразделы для просмотра отчетов за неделю, месяц, год и другие карточки, включая удаленные и связанные карточки, а так же записи разговоров;
– справочники - доступ для администраторов или начальников отдела к спискам справочников (категории, организации, населенные пункты);
– пользователи - функции для регистрации новых пользователей и изменения данных существующих пользователей;
– помощь - документация, техническая поддержка, информация о приложении.

UI и стили:

– используется темная тема Superhero для Bootstrap для снижения нагрузки на глаза;
– поддержка выпадающих меню с подкатегориями для организации доступа к различным функциям;
– меню меняется в зависимости от уровня доступа (например, операторы не видят пункты «Справочники» и «Пользователи»).

Левый блок (Карточки).

Расположение: левая часть страницы, ниже меню (рисунок 22).

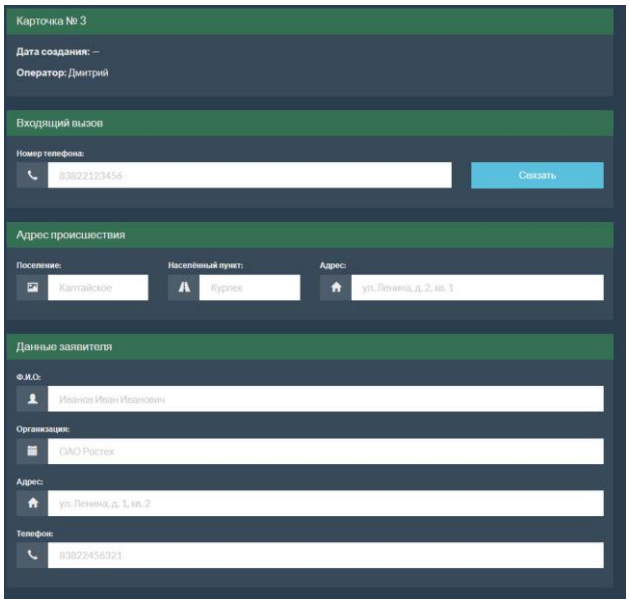


Рисунок 22 – Карточки веб-приложения для ЕДДС

Функционал. Блок отображает карточку, в которой пользователи вводят информацию о звонках и инцидентах. Карточки содержат такие поля, как:

– контактные данные - номер телефона, ФИО, адрес происшествия;
– информация о происшествии - категория, описание, организация, проблематика, состояние;
– детали инцидента - дата и время, аудиозапись (если доступна), отчет о выполнении.

Интерактивные функции:

– автозаполнение (автодополнение) - поля «Организация», «Населенный пункт», «Категория» поддерживают автозаполнение (рисунок 23);

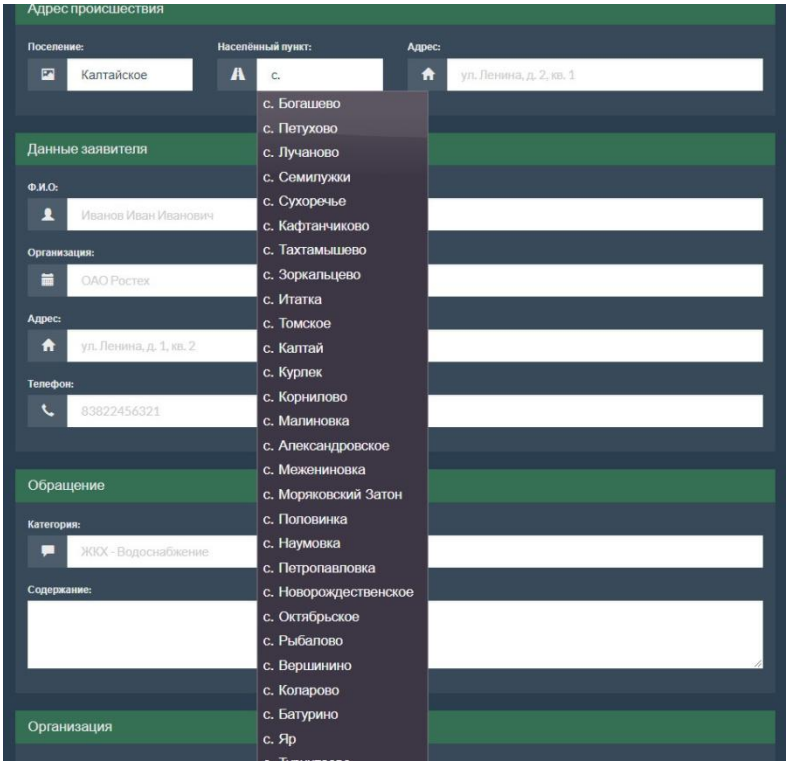


Рисунок 23 – Автозаполнение поля «Населенный пункт»

– выпадающие списки - поля «Состояние», «Проблематика» и «Подпроблема» заполняются данными из справочников, также доступен фильтр для выбора нужного значения (рисунок 24).

Кнопки управления (рисунки 22 и 24):

– сохранить карточку;
– удалить карточку (если звонок ложный);
– «Связать» карточку с другой (вызывает модальное окно для выбора карточки, представлено на рисунке 25).

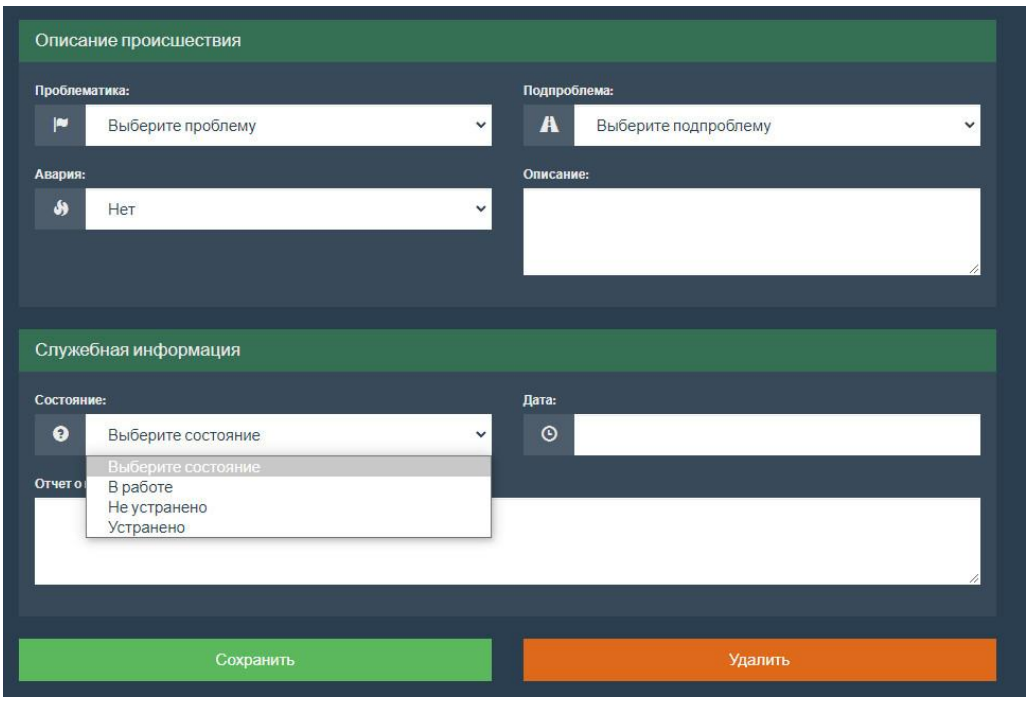


Рисунок 24 – Выпадающий список поля «Состояние»

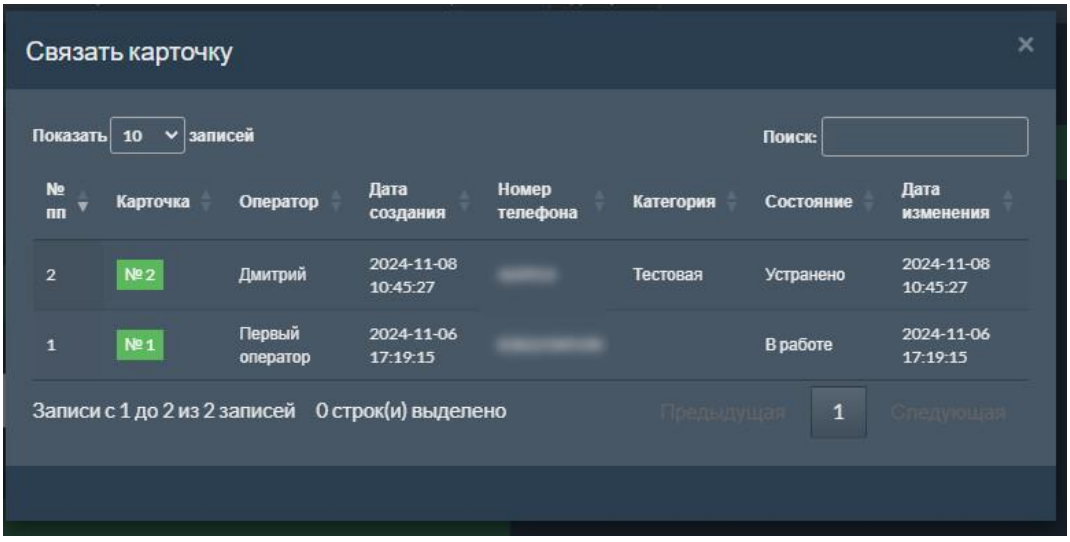


Рисунок 25 – Модальное окно «Связать карточку»

Правый блок (Управление, Вкладки).

Функционал. Управление карточками, просмотр и фильтрация данных, включающие (рисунок 26):

– отчет за сутки - список всех карточек за день;

– поиск - поля для поиска карточек по параметрам;

– история - таблица с историей изменений карточек, доступна для подробного просмотра;
– связи - список связанных карточек;
– удаленные карточки - список удаленных карточек с возможностью просмотра.

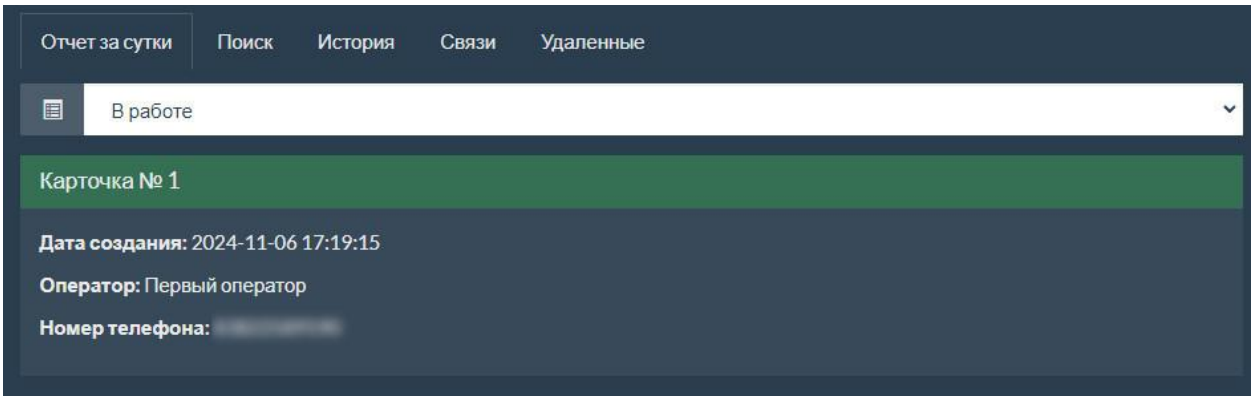


Рисунок 26 – Правый блок веб-приложения – управление, вкладки

Дополнительные функции:
– фильтрация и сортировка - доступен фильтр для карточек на основе их статуса и категории;
– DataTables - используется библиотека DataTables для отображения данных, позволяя упрощать добавление, редактирование и удаление строк в таблице;
– навигация - вкладки для быстрого переключения между разделами.

Прочий функционал веб-приложения.

В приложении Б, на рисунке Б.1 представлена полноценная экранная форма модального окна с отчетностью за месяц.

В приложении Б, на рисунке Б.2 представлена экранная форма изменения данных пользователя администратором системы.

Использование темной темы, адаптивного дизайна и визуальных компонентов позволяет снизить нагрузку на зрение и повысить комфорт работы дежурных. Интерфейс обеспечивает интуитивное взаимодействие с

системой, минимизирует количество действий при регистрации вызовов и поддерживает эффективную работу в условиях многозадачности.

2.8 Создание прототипа базы данных

Для создания прототипа базы данных на основе описанных функций и взаимодействий необходимо начать с определения основных таблиц и связей. Разработку прототипа будем производить в программе MySQL Workbench [5]. Прототип представлен в приложении В, на рисунке В.1.

Описание прототипа базы данных.

Таблица account хранит информацию о пользователях системы, включая учетные данные и уровень доступа.

Основные поля:

- id: идентификатор пользователя (первичный ключ);
- username, email, fio: учетные данные пользователя, имя и контактная информация;
- v, s: поля, необходимые для шифрования пароля в специальный хэш.
- level: уровень доступа пользователя (например, оператор, администратор);
- failed_logins, locked: счетчики для ограничения доступа при превышении количества неверных попыток входа;
- joindate, bandate: дата регистрации и дата блокировки, если применимо.

Таблица cards - основная таблица для хранения информации о карточках звонков.

Основные поля:

- id: уникальный идентификатор карточки;
- operator: имя оператора, заполнившего карточку;
- createdate, modifdate: дата создания и последнего изменения карточки;

– phone, settle, village, address: контактные и адресные данные;
– reason, keep, audio: категория происшествия, содержание и идентификатор аудиофайла записи разговора;
– problem, problem2: проблема и подпроблема, указанные для классификации происшествия;
– state: текущий статус карточки;
– report: отчет о выполнении или статусе работ.

Таблица cards_disable хранит информацию о карточках, которые были удалены или деактивированы.

Основные поля: идентичны полям таблицы cards, что позволяет полностью сохранять информацию деактивированных карточек.

Таблицы справочников (digests_settlements, digests_villages, digests_categories, digests_organizations, digests_problems, digests_problems2, digests_states) содержат справочные данные для автозаполнения и классификации в карточках.

Основные поля:

– id: уникальный идентификатор;
– param, named: параметр (например, код) и название элемента справочника (например, название населенного пункта, организации, категории).

Таблица history сохраняет историю изменений каждой карточки.

Основные поля:

– id: уникальный идентификатор записи истории;
– card_id: идентификатор карточки, к которой относится запись;
– description, data: описание изменения и его данные в JSON формате;
– date_time: дата и время записи.

Таблица audio хранит данные о записи разговоров.

Основные поля:

– id: идентификатор записи;

– card_id: идентификатор карточки, связанной с записью;
– audio, file: поля для хранения данных о звуковом файле.

Таблица hookup хранит информацию о связях между карточками.

Основные поля:

– card_id, card_hookup: поля для записи идентификаторов связанных карточек;
– createdate: дата создания связи.

Связи между таблицами.

Таблица cards:

– связана с таблицей history по полю card_id, эта связь позволяет хранить историю изменений для каждой карточки;
– связана с таблицей audio также по полю card_id, это дает возможность привязывать аудиозаписи к конкретной карточке;
– связана с таблицей hookup через поля card_id и card_hookup, и позволяет устанавливать связи между различными карточками;
– имеет внешние ключи problem, problem2 и state, которые ссылаются на таблицы digests_problems, digests_problems2 и digests_states, соответственно; эти поля используются для хранения информации о проблемах и состоянии карточки, используя справочные данные из связанных таблиц.

Таблица cards_disable:

– структурно аналогична таблице cards, но служит для хранения удаленных карточек;
– может также иметь аналогичные связи с таблицами digests_problems, digests_problems2 и digests_states через соответствующие внешние ключи, и позволяет сохранять информацию о проблемах и состоянии архивированных карточек.

Таблица history связана с таблицей cards через поле card_id и позволяет отслеживать изменения, связанные с конкретной карточкой.

Таблица audio связана с таблицей cards по полю card_id - она дает возможность привязывать аудиозаписи к определенной карточке для хранения пути записей разговоров.

Таблица hookup имеет связь с таблицей cards по полям card_id и card_hookup. Это позволяет создавать ассоциации между карточками, чтобы, например, связывать инциденты или другие взаимосвязанные данные.

Таблицы digests_settlements, digests_villages, digests_categories, digests_organizations содержат справочные данные и не имеют прямых связей с другими таблицами, но используются для хранения информации о населенных пунктах, категориях, организациях и других атрибутах, которые могут быть полезны в карточках и используются для автозаполнения.

Таблица digests_problems и digests_problems2 связаны с таблицами cards и cards_disable через поля problem и problem2, и указывают типы проблем для каждой карточки.

Таблица digests_states связана с таблицами cards и cards_disable через поле state, она определяет текущее состояние карточки с использованием справочной информации.

Выводы по главе 2

Во второй главе была выполнена разработка веб-приложения для единой дежурно-диспетчерской службы, включая выбор архитектуры, технологий и реализацию ключевых компонентов системы.

В ходе работы была выбрана трёхуровневая клиент-серверная архитектура, включающая клиентскую часть, серверную часть и базу данных. Данное разделение позволило создать гибкую и масштабируемую систему, обеспечивающую надежность и безопасность работы.

Для клиентской части использованы современные технологии, включая Bootstrap, DataTables, jQuery и Moment.js, что позволило создать удобный и адаптивный интерфейс. Серверная часть реализована на PHP 7.4 с использованием Apache и MySQL, что обеспечивает высокую производительность и стабильность системы.

Разработана структура базы данных, включающая 13 таблиц, оптимизированных для хранения и быстрого поиска информации о вызовах и инцидентах. Предусмотрены механизмы разграничения прав доступа и защиты данных, а также резервное копирование, снижающее риски потери информации.

Особое внимание уделено вопросам безопасности. Внедрены механизмы авторизации и разграничения прав доступа по ролям (оператор, начальник, проверяющий), реализовано шифрование передаваемых данных, журналирование действий пользователей и регулярное резервное копирование информации. Это позволяет обеспечить сохранность данных и прозрачность работы системы.

Также разработан комплекс программных модулей для реализации алгоритмов обработки данных — от регистрации вызовов до связывания карточек и фильтрации записей. Использование диаграмм классов, компонентов, вариантов использования, а также диаграмм потоков данных и последовательностей позволило структурировать проект на уровне проектной документации и обеспечить его визуальное представление.

Таким образом, в результате проделанной работы было создано полноценное веб-приложение, удовлетворяющее всем поставленным функциональным и техническим требованиям. Реализация архитектуры, интерфейса, базы данных и алгоритмов обработки данных завершает этап проектирования и разработки и обеспечивает основу для последующего тестирования системы и оценки её эффективности в условиях реального использования.

Глава 3 Тестирование приложения и расчет экономической эффективности

3.1 Разработка контрольных примеров

Для разработки контрольных примеров, обеспечивающих многоцелевое тестирование базы данных и прикладных программ были определены несколько сценариев [1].

Добавление новой записи в таблицу cards. Цель данного теста — проверить, что в таблицу cards можно корректно добавить новую запись с полными данными. Также проверяется выполнение ограничений и обязательность полей.

Предварительные условия:

- таблица cards существует и доступна для записи данных;
- есть доступ к полям operator, phone, reason, и другим необходимым полям.

Шаги выполнения:

- получить уведомление с модальным окном и списком вызовов;
- открыть интерфейс для добавления новой записи (блок карточки);
- ввести данные в поля;
- сохранить запись.

Ожидаемый результат:

- запись успешно добавлена в таблицу cards;
- значение в поле createdate автоматически установлено на текущую дату и время;
- данные корректно сохранены во всех полях;
- поля id и другие, имеющие автоинкремент или дефолтные значения, автоматически заполнены.

Обновление записи в таблице account. Цель теста — проверить, что запись в таблице account можно корректно обновить. В ходе теста будут

проверены обновление полей, таких как username, email, и locked, а также сохранение и корректное обновление значений.

Предварительные условия:

- таблица account существует и доступна для обновления данных;
- в таблице account есть хотя бы одна запись для обновления;
- известен id записи, которую нужно обновить (например, id = 1). Шаги выполнения:
- открыть под ролью администратора через меню пункт «Сменить данные пользователя»;
- выбрать логин пользователя;
- внести изменения в данные путем заполнения необходимых полей;
- сохранить запись, кликнув на кнопку «Сменить».

Ожидаемый результат:

- запись с уникальным id успешно обновлена;
- поле fio теперь содержит новое значение;
- поле email содержит новое значение;
- поле locked содержит значение 0.

Выполнение поиска по параметру состояния в таблице cards. Цель теста — проверить, что можно корректно выполнять поиск записей в таблице cards по определенному значению состояния (state). Этот тест важен для проверки фильтрации записей на основе состояния и для обеспечения корректной работы системы при выборке данных по этому параметру.

Предварительные условия:

- таблица cards существует и содержит несколько записей с различными значениями в поле state;
- известны значения state, которые будут использоваться в тесте (например, state = 1 и state = 2);
- текстовые значения корректно преобразуются в числовые значения состояния.

Шаги выполнения:

– открыть у базовом блоке вкладку «Поиск»;
– выбрать параметра для поиска;
– указать слово для поиска;
– кликнуть на кнопку «Искать»;
– получить ниже ожидаемый и корректный результат.

Ожидаемый результат:

– поиск по текстовым точным значением возвращает все записи, у которых значение state равно числовым, и только им;
– поиск по числовым значениям возвращает все записи, у которых значения state соответствует заданным;
– поиск неверных значений или значений которые не были найдены должен выдавать сообщение «Ничего не найдено...».

Примеры охватывают типовые сценарии работы ЕДДС, включая регистрацию, обработку и поиск вызовов. Проведённые проверки подтвердили стабильную работу системы и её готовность к применению в реальных условиях.

3.2 Контрольный пример реализации проекта

Для представления контрольного примера реализации проекта рассмотрим основной сценарий, включающий тестовые данные, процесс обработки и результаты обработки.

Добавление новой записи в таблицу cards.

Прежде чем начать заполнение полей, нам необходимо дождаться уведомления о поступлении нового вызова. Когда это происходит, на экране пользователя всплывает модальное окно (приложение Г, рисунок Г.1).

После чего нам нужно выбрать интересующий нас звонок и заполнить по нему карточку, кликнув на соответствующую кнопку «добавить», в строке с нужным звонком.

Далее, мы вводим данные в поля карточке, и в нижней части карточки кликаем на кнопку «Сохранить» (приложение Г, рисунок Г.2).

После этого модальное окно закрывается, и данные отправляются на сервер обработчику. Чтобы убедиться в правильности отправки данных используем в клиенте (в данном примере используется браузер) инструмент разработчика. Где мы может посмотреть отправку наших данных. На рисунке 27 представлена экранная форма инструмента разработчика, где указано какие данные были отправлены обработчику через POST-запрос.

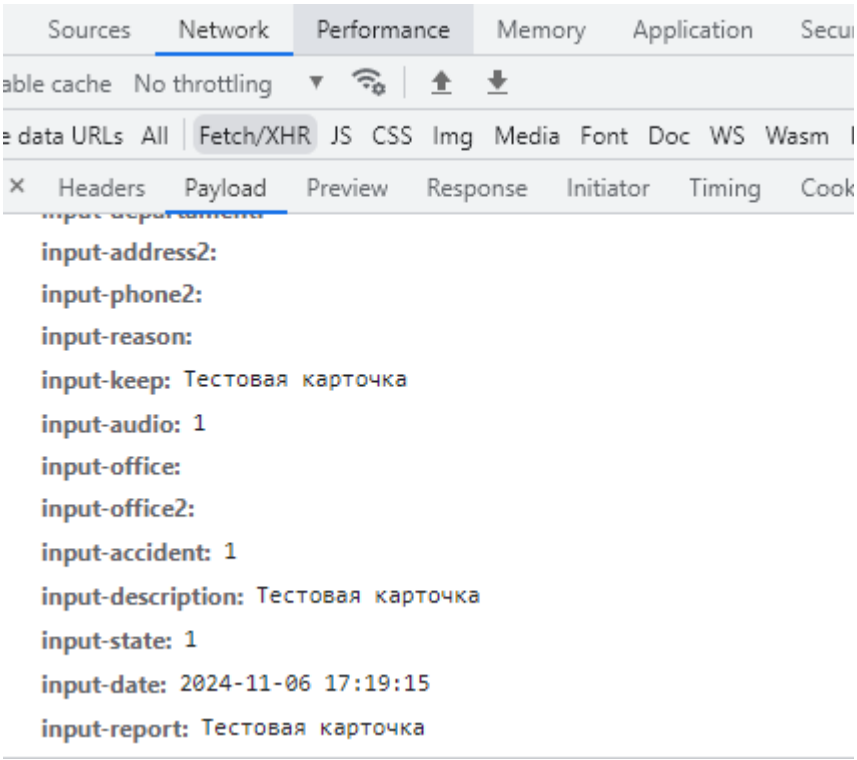


Рисунок 27 – Экранная форма инструмента разработчика с отправкой данных на сервер

На этом работу с фронд частью можно завершить. Теперь проверим, что данные в бэкенде были отправлены в базу. Воспользуемся программой Navicat для удобной визуализации и проверки нужной нам таблицы на факт появления там новой записи. На рисунке 28 представлена экранная форма таблицы cards базы данных веб-приложения в программе Navicat. Кроме того, на форме видны параметры этой таблицы.

Так как это первая запись в таблице, ей автоматически присвоился идентификатор 1. В поле createdate верно указаны дата и время тестового запроса. А параметр таблицы AUTO INCREMENT соответственно равен 2.

Остальные поля заполнены верно, можно считать что тест успешен.

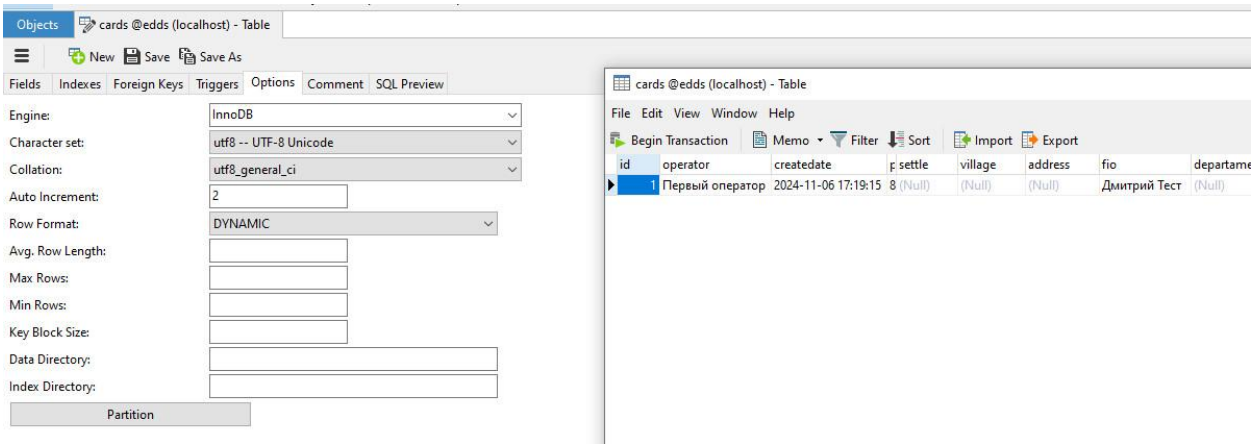


Рисунок 28 – Экранная форма таблицы cards и её параметров в программе Navicat

Обновление записи в таблице account.

Чтобы изменить данные пользователя веб-приложения, нам необходима роль администратора. Пользователь с такой ролью должен вызвать соответствующий пункт меню - Сменить данные пользователя. После чего должно появиться модальное окно со списком пользователей веб-приложения (рисунок 29).

После открытия модального окна нам необходимо выбрать логин пользователя, произвести изменения полей и отправить данные обработчику на сервер с помощью кнопки «Сменить».

Попробуем изменить данные логина operator3 – ФИО и Email. Проконтролируем эти изменения с помощью инструмента разработчика и убедимся, что данные отправлены на сервер (рисунок 30).

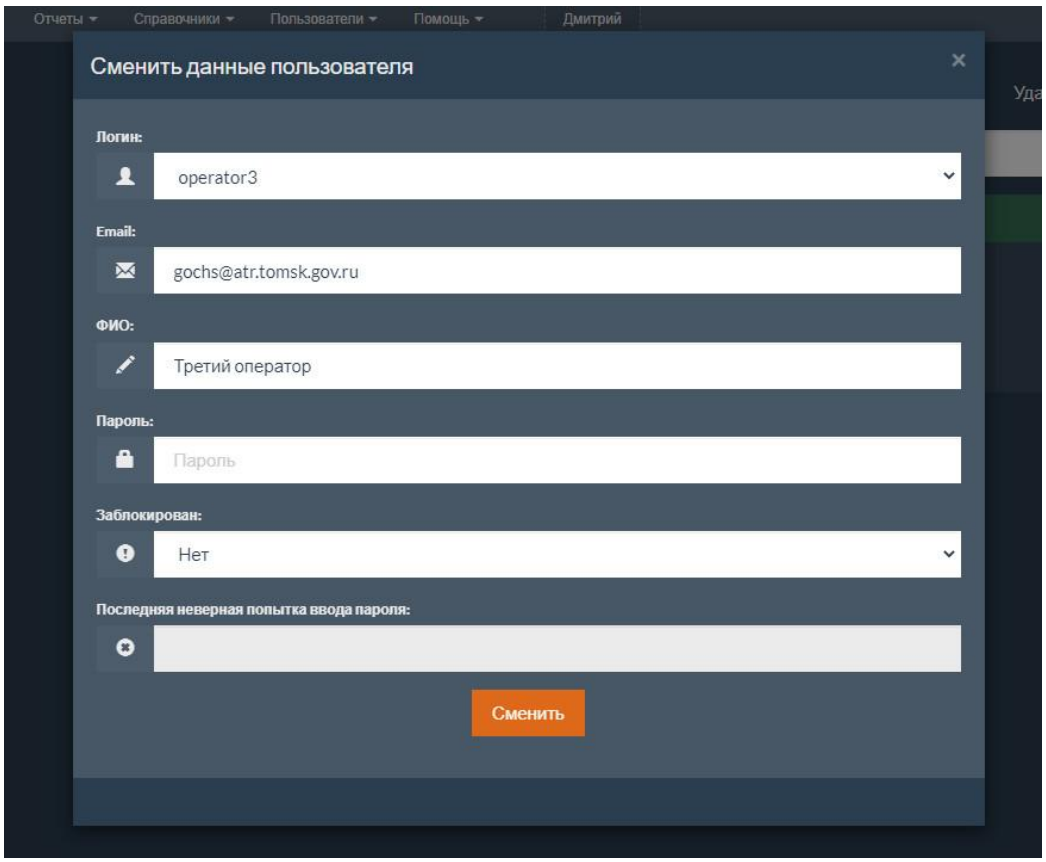


Рисунок 29 – Модальная форма изменения данных пользователя

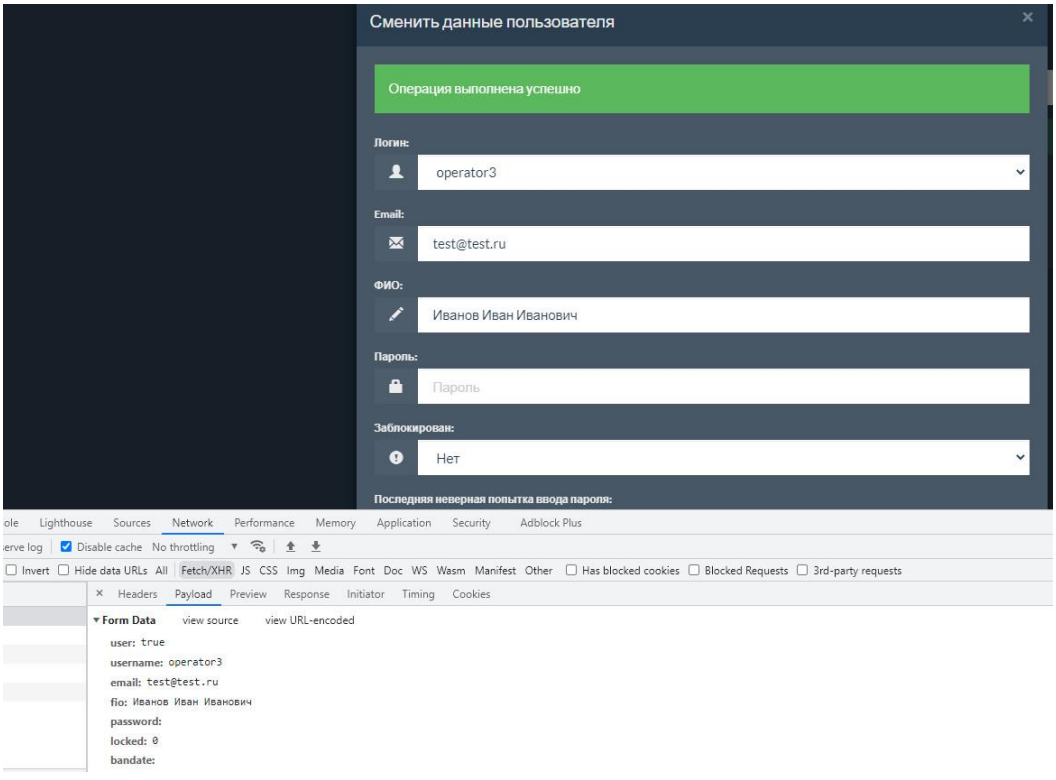
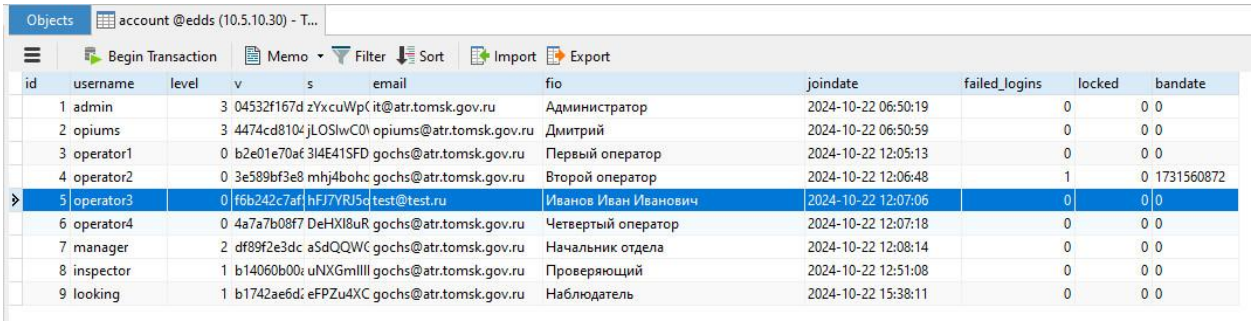


Рисунок 30 – Отправка новых данных пользователя на сервер

После получения сообщения об успехе, убедимся, что данные попали в базу данных через обработчик, воспользуемся программой Navicat (рисунок 31).



id	username	level	v	s	email	fio	joindate	failed_logins	locked	bandate
1	admin	3	04532f167d	zYxcuWp(	it@atr.tomsk.gov.ru	Администратор	2024-10-22 06:50:19	0	0	0
2	opiums	3	4474cd8104	jLOSlwC0\	opiums@atr.tomsk.gov.ru	Дмитрий	2024-10-22 06:50:59	0	0	0
3	operator1	0	b2e01e70a6	3l4E41SFD	gochs@atr.tomsk.gov.ru	Первый оператор	2024-10-22 12:05:13	0	0	0
4	operator2	0	3e589bf3e8	mhj4bohc	gochs@atr.tomsk.gov.ru	Второй оператор	2024-10-22 12:06:48	1	0	1731560872
5	operator3	0	f6b242c7af	hFJ7YRJ5c	test@test.ru	Иванов Иван Иванович	2024-10-22 12:07:06	0	0	0
6	operator4	0	4a7a7b08f7	DeHXI8uR	gochs@atr.tomsk.gov.ru	Четвертый оператор	2024-10-22 12:07:18	0	0	0
7	manager	2	df89f2e3dc	aSdQQWC	gochs@atr.tomsk.gov.ru	Начальник отдела	2024-10-22 12:08:14	0	0	0
8	inspector	1	b14060b00a	uNXGmIII	gochs@atr.tomsk.gov.ru	Проверяющий	2024-10-22 12:51:08	0	0	0
9	looking	1	b1742ae6d2	eFPZu4XC	gochs@atr.tomsk.gov.ru	Наблюдатель	2024-10-22 15:38:11	0	0	0

Рисунок 31 – Экранная форма таблицы account в программе Navicat

Убедившись, что данные по этому логину были изменены в соответствии с запросом, можно считать тест успешно завершенным.

Выполнение поиска по параметру состояния в таблице cards.

Поиск по параметрам находится в базовом блоке, вкладке «поиск». Открыв данную вкладку, мы можем выбрать необходимый для тестирования параметр – состояние. И указать необходимое состояние, как в текстовом (рисунок 32), так и в числовом значении.

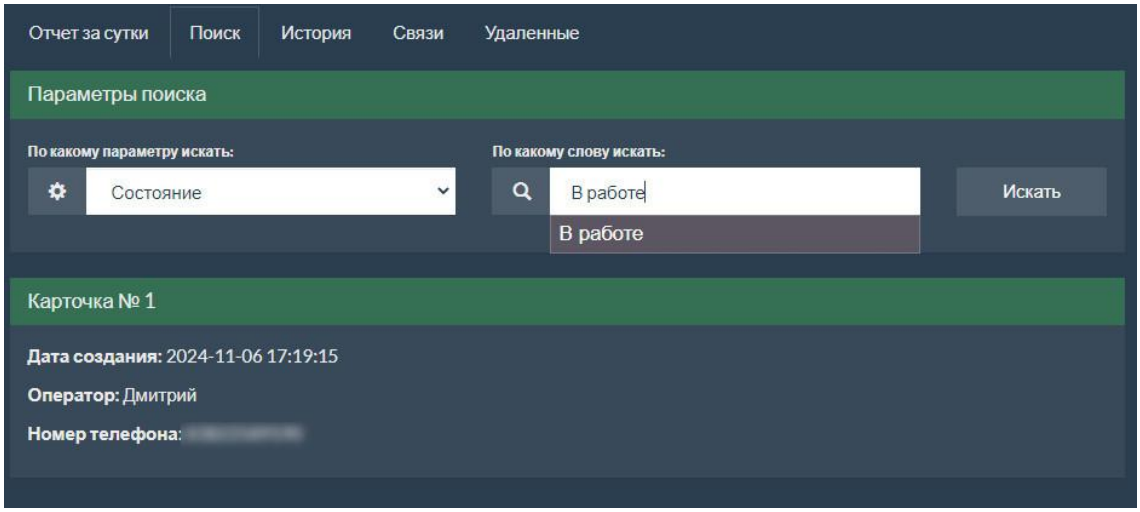


Рисунок 32 – Экранная форма поиска веб-приложения для ЕДДС

В настоящий момент существует только 3 состояния, тестовые карточки есть в 2-х состояниях, соответственно по 2-м карточки должны выводиться в поиске. На рисунке 35 поиск уже дал результаты и вывел карточку. Специально для тестов, обработчик на сервере возвращает запрос к базе данных, и мы его можем посмотреть с помощью инструмента разработчика на рисунке Г.3, в приложении Г.

Продолжим тестирование, проверив другие состояния карточек по поиску. Результаты поиска представлены на рисунке Г.4, в приложении Г.

После получения положительных результатов, убедимся, что также работает поиск по числовым значениям состояния – 1, 2, и 3. Результаты поиска представлены на рисунке Г.5, в приложении Г.

Все этапы тестирования показали правильный и безошибочный, ожидаемый результат. Можно считать тестирование завершенным и подтвердить, что разработанная система соответствует заявленным требованиям и обеспечивает корректную работу основных функций

3.3 Расчет экономической эффективности проекта

Для расчета экономической эффективности проекта потребуется несколько шагов:

- расчет затрат на реализацию проекта;
- экономия от внедрения;
- расчет экономической эффективности.

Расчет затрат на реализацию проекта

Затраты на трудовые ресурсы:

- часовая ставка: 300 рублей/час;
- общее количество потраченных на разработку дней: 60;
- продолжительность разработки в сутки: 4 часа.

Общие затраты на труд: Часовая ставка × Количество рабочих дней × Часы в день = 300 × 60 × 4 = 72 000 рублей.

Затраты на оборудование и программное обеспечение:

- все программное обеспечение использовалось с открытым исходным кодом и бесплатное;
- серверы и компьютеры уже имеются, и нет необходимости в дополнительных затратах на оборудование.

Итого: 0 рублей

Накладные расходы (аренда помещения, хозяйственные нужды и поддержка инфраструктуры) в проекте не учитываются, так как администрация уже покрывает эти расходы.

Итого: 0 рублей

Прочие расходы:

- привлечение стороннего специалиста для настройки АТС: 30 000 рублей;
- обучение сотрудников: 50 000 рублей.

Общие прочие расходы: 30 000 + 50 000 =8 0 000 рублей

Итоговые затраты на реализацию проекта

Суммируем все категории затрат:

72,000 + 0 + 0 + 80 000 = 152 000 рублей

Итого, фактические затраты на реализацию проекта составляют 152 000 рублей.

Расчет экономии от внедрения проекта.

На основе полученных данных рассчитаем текущие расходы и новые фактические расходы, а также определим ежегодную экономию от перехода на электронные журналы.

Текущие расходы на ведение бумажных журналов.

Ежемесячные затраты на бумажные журналы и канцелярию:

- журнал — 400 рублей/месяц;
- канцелярия — 100 рублей/месяц;
- корректоры — 100 рублей/месяц;

– прочие расходы на канцелярию — 1000 рублей/месяц.

Итого на канцелярию: 400 + 100 + 100 + 1000 = 1 600 рублей/месяц

Затраты на составление отчетов и привлечение сторонних специалистов:

– привлечение специалистов для отчетов — 50 000 рублей/месяц;
– прочие расходы (штрафы, организационные потери) — 10 000 рублей/месяц.

Итого на отчетность и прочие расходы: 50 000 + 10 000 = 60 000 рублей/месяц.

Общие текущие расходы на процесс: 1 600 + 60,000 = 61 600 рублей/месяц.

Ежегодные расходы на бумажные журналы: 61 600 × 12 = 739 200 рублей/год.

Новые расходы после внедрения электронного журнала. Так как проект был разработан за 4 месяца, месячные фактические затраты составили: 152 000 × 4 = 38 000 рублей/месяц.

Ежегодные новые расходы: 8 000 × 12 = 456 000 рублей/год

Ежегодная экономия: 739 200 – 456 000 = 283 200 рублей/год

Вывод: внедрение веб-приложения позволяет сократить расходы на 283 200 рублей в год за счет автоматизации процесса, уменьшения затрат на покупку журналов и канцелярию, а также снижения потерь, связанных с ошибками и привлечением сторонних специалистов.

Расчет экономической эффективности

Для расчета экономической эффективности проекта воспользуемся несколькими показателями: сроком окупаемости, чистым приведённым доходом (NPV) и рентабельностью инвестиций (ROI).

Срок окупаемости.

Срок окупаемости показывает, за какое время проект вернет вложенные средства.

$$\text{Срок окупаемости} = \frac{\text{Общие затраты на проект}}{\text{Ежегодная экономия}}, \quad (1)$$

где общие затраты на проект: 152 000 рублей;

ежегодная экономия: 283 200 рублей.

$$\text{Срок окупаемости} = \frac{152\,000}{283\,200} \sim 0.54 \text{ года} \sim 6.5 \text{ месяцев}$$

Чистый приведённый доход (NPV).

Чтобы рассчитать NPV, принимаем дисконтную ставку 10% для корректировки денежных потоков с учетом временной стоимости денег. Предположим, что проект будет приносить экономию в течение 5 лет.

Формула для расчета NPV:

$$NVP = \sum_{t=1}^{T} \frac{\text{Экономия}}{(1+r)^t} - \text{Общие затраты}, \quad (2)$$

где T = 5 лет;

r = 0.1 (ставка дисконтирования);

экономия = 283 200 рублей ежегодно.

$$\text{NPV} = \frac{283\,200}{(1+0.1)^1} + \frac{283\,200}{(1+0.1)^2} + \frac{283\,200}{(1+0.1)^3} + \frac{283\,200}{(1+0.1)^4} + \frac{283\,200}{(1+0.1)^5} - 152\,000$$

NPV = 257 455 + 234 050 + 212 773 + 193 430 + 175 846 – 152 000 = 921 554 рублей

Вывод: чистый приведённый доход за 5 лет составит 921,554 рублей, что указывает на высокую привлекательность проекта.

Рентабельность инвестиций (ROI).

Рентабельность инвестиций (ROI) показывает процентную доходность от вложений.

$$\text{ROI} = \frac{\text{Экономия за 5 лет} - \text{Общие затраты}}{\text{Общие затраты}} \times 100\%, \quad (3)$$

где экономия за 5 лет: 283 200 × 5 = 1 416 000 рублей;

общие затраты: 152 000 рублей.

$$\text{ROI} = \frac{1\,416\,000 - 152\,000}{152\,000} \times 100\% = 832.89\%$$

Вывод: рентабельность инвестиций составляет 832.89%, что подтверждает высокую экономическую эффективность проекта.

Общий итог. Все показатели указывают на высокую экономическую

эффективность проекта. Срок окупаемости составляет всего 6.5 месяцев, а ROI показывает значительный уровень доходности вложений, что делает проект экономически целесообразным для реализации.

Выводы по главе 3

В третьей главе было проведено тестирование разработанного веб-приложения для единой дежурно-диспетчерской службы с целью проверки его работоспособности и соответствия функциональным требованиям.

Были разработаны контрольные примеры для тестирования основных функций системы, включая добавление новой записи в базу данных, обновление пользовательских данных и поиск по параметрам. В ходе тестирования использовались инструменты разработчика в браузере и программа Navicat для анализа содержимого базы данных.

Тестирование показало, что все ключевые функции приложения работают корректно. Добавление и обновление записей выполняются без ошибок, обеспечивается сохранность и целостность данных, а поиск по параметрам успешно возвращает ожидаемые результаты. Проверенные сценарии подтвердили правильность работы серверной и клиентской частей, а также взаимодействие с базой данных.

Так же был выполнен расчет экономической эффективности внедрения веб-приложения. В ходе анализа были оценены затраты на разработку, развертывание и сопровождение системы, а также экономия, достигаемая за счет сокращения времени обработки вызовов, уменьшения бумажного документооборота и снижения нагрузки на операторов. Расчеты показали, что внедрение системы позволяет существенно сократить трудозатраты персонала и повысить скорость реагирования на инциденты, что подтверждает целесообразность ее использования.

Заключение

В процессе выполнения выпускной квалификационной работы была разработана и протестирована автоматизированная система в виде веб-приложения для единой дежурно-диспетчерской службы Администрации Томского района. Проведенный анализ предметной области позволил выявить основные проблемы существующей системы, основанной на ручном ведении журналов, что приводило к низкой скорости обработки вызовов, затруднениям в поиске информации и высоким рискам потери данных.

Для решения данных проблем была выбрана трехуровневая клиент-серверная архитектура, включающая клиентскую часть, серверную часть и базу данных. Реализация клиентской части выполнена с использованием Bootstrap, DataTables и jQuery, что позволило создать удобный и адаптивный интерфейс. Серверная часть разработана на PHP с использованием Apache и MySQL, это решение обеспечило высокую производительность, надежность и безопасность системы.

В рамках работы была создана структура базы данных, включающая 13 таблиц, предназначенных для хранения информации о вызовах, карточках инцидентов, пользователях и справочных данных. Были разработаны алгоритмы обработки входящих звонков, связывания карточек и работы с отчетностью, такой подход позволил автоматизировать ключевые процессы диспетчерской службы.

Проведенное тестирование подтвердило корректность работы всех функциональных модулей системы. В ходе испытаний были проверены операции добавления и обновления данных, поиск по параметрам и безопасность передачи информации. Результаты тестирования показали, что веб-приложение стабильно функционирует, обеспечивая быструю обработку данных, удобный доступ к информации и защиту от потери данных.

Выполненная работа показывает, что внедрение разработанной системы позволит значительно повысить эффективность работы ЕДДС,

сократить расходы и время обработки инцидентов, улучшить доступность информации и обеспечить необходимый уровень безопасности данных.

Подведя итог работы, можно сделать вывод, что поставленная цель – разработка веб-приложения для автоматизации процессов в ЕДДС – была полностью достигнута. Все поставленные задачи были успешно решены, что подтверждается корректной работой системы, ее соответствием техническим требованиям и успешным прохождением тестирования. Разработанная система готова к внедрению и дальнейшему развитию, включая возможную интеграцию с внешними сервисами и расширение аналитических возможностей.

Список используемых источников информации

1. Аграновский, А. В. Тестирование веб-приложений : учебное пособие / А. В. Аграновский. — Санкт-Петербург : ГУАП, 2020. — 155 с. — ISBN 978-5-8088-1515-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/216533
2. Баланов, А. Н. Управление IT-проектами : учебное пособие для вузов / А. Н. Баланов. — Санкт-Петербург : Лань, 2024. — 616 с. — ISBN 978-5-507-49698-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/428081
3. Гостин, А. М. Интернет-технологии : учебное пособие / А. М. Гостин, А. Н. Сапрыкин. — Рязань : РГРТУ, 2017 — Часть 2 — 2017. — 64 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/168158
4. Грекул, В. И. Проектирование информационных систем : учебное пособие / В. И. Грекул, Г. Н. Денищенко, Н. Л. Коровкина. — 4-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2024. — 299 с. — ISBN 978-5-4497-3335-1. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/142298.html
5. Деваев, В. М. Методы структурного моделирования информационных систем : учебное пособие / В. М. Деваев. — Казань : КНИТУ-КАИ, 2017. — ISBN 978-5-7579-2296-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/156056
6. Доррер, Г. А. Методология программной инженерии : учебное пособие / Г. А. Доррер. — Красноярск : СибГУ им. академика М. Ф.

Решетнёва, 2021. — 190 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/195097

7. Информационное обеспечение и базы данных : учебное пособие / составитель А. Ф. Похилько. — Ульяновск : УлГТУ, 2019. — ISBN 978-5-9795-1964-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/165031
8. Калиберда, Е. А. Разработка web-приложений : учебное пособие / Е. А. Калиберда, К. В. Кравченко. — Омск : ОмГТУ, 2023. — ISBN 978-5-8149-3679-0. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/421766
9. Кашкин, Е. В. Разработка динамических страниц на языке JavaScript с использованием библиотеки jQuery : учебно-методическое пособие / Е. В. Кашкин. — Москва : РТУ МИРЭА, 2020. — 86 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/
10. Лёзин, И. А. Разработка веб-приложений с использованием Spring Boot : учебное пособие / И. А. Лёзин, И. В. Лёзина. — Самара : Самарский университет, 2023. — 96 с. — ISBN 978-5-7883-1989-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/406517
11. Павлов, Л. А. Структуры и алгоритмы обработки данных : учебник для вузов / Л. А. Павлов, Н. В. Первова. — 3-е изд., стер. — Санкт-Петербург : Лань, 2021. — 256 с. — ISBN 978-5-8114-7259-8. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/156929
12. Положение об отделе по информатизации и кадровому обеспечению Управления Делами Администрации Томского района.
13. Положения о единой дежурно-диспетчерской службе отдела по ГО

и ЧС Управления Делами Администрации Томского района

14. Постановление Правительства Российской Федерации от 21 ноября 2011 г. № 958 "Положение о системе обеспечения вызова экстренных оперативных служб по единому номеру «112»" (в ред. Постановлений Правительства РФ от 04.09.2012 № 882, от 06.03.2015 № 201, от 20.11.2018 № 1391) // Собрание законодательства РФ. – 2011. – № 48. – Ст. 6932. — URL: https://www.consultant.ru/document/cons_doc_LAW_122161/
15. 1С:Диспетчерская служба: программный продукт: электронный ресурс. — URL: https://www.1cbit.ru/1csoft/1s-dispetcherskaya-zhkkh/
16. amoCRM: система управления взаимоотношениями с клиентами (CRM): официальный сайт. — URL: https://www.amocrm.ru
17. Bootstrap CSS 3.4 // Documentation. URL: https://getbootstrap.com/docs/3.4/css/
18. Bootstrap-datetimepicker: библиотека для выбора даты и времени в Bootstrap: электронный ресурс. — URL: https://github.com/Eonasdan/tempus-dominus
19. DataTables: библиотека для работы с таблицами в JavaScript: электронный ресурс. — URL: https://datatables.net
20. Federal Communications Commission (FCC). Enhanced 911 (E911) Requirements for Wireless Services [Electronic resource]. – Washington, D.C.: FCC, 2023. — URL: https://www.fcc.gov/general/9-1-1-and-e9-1-1-services
21. Moment.js: библиотека для работы с датами и временем в JavaScript: электронный ресурс. — URL: https://momentjs.com/
22. MySQL: система управления базами данных: официальная документация: электронный ресурс. — URL: https://dev.mysql.com/doc
23. The PHP Manual // Documentation. URL: https://www.php.net/

Приложение А

Расширенная диаграмма классов

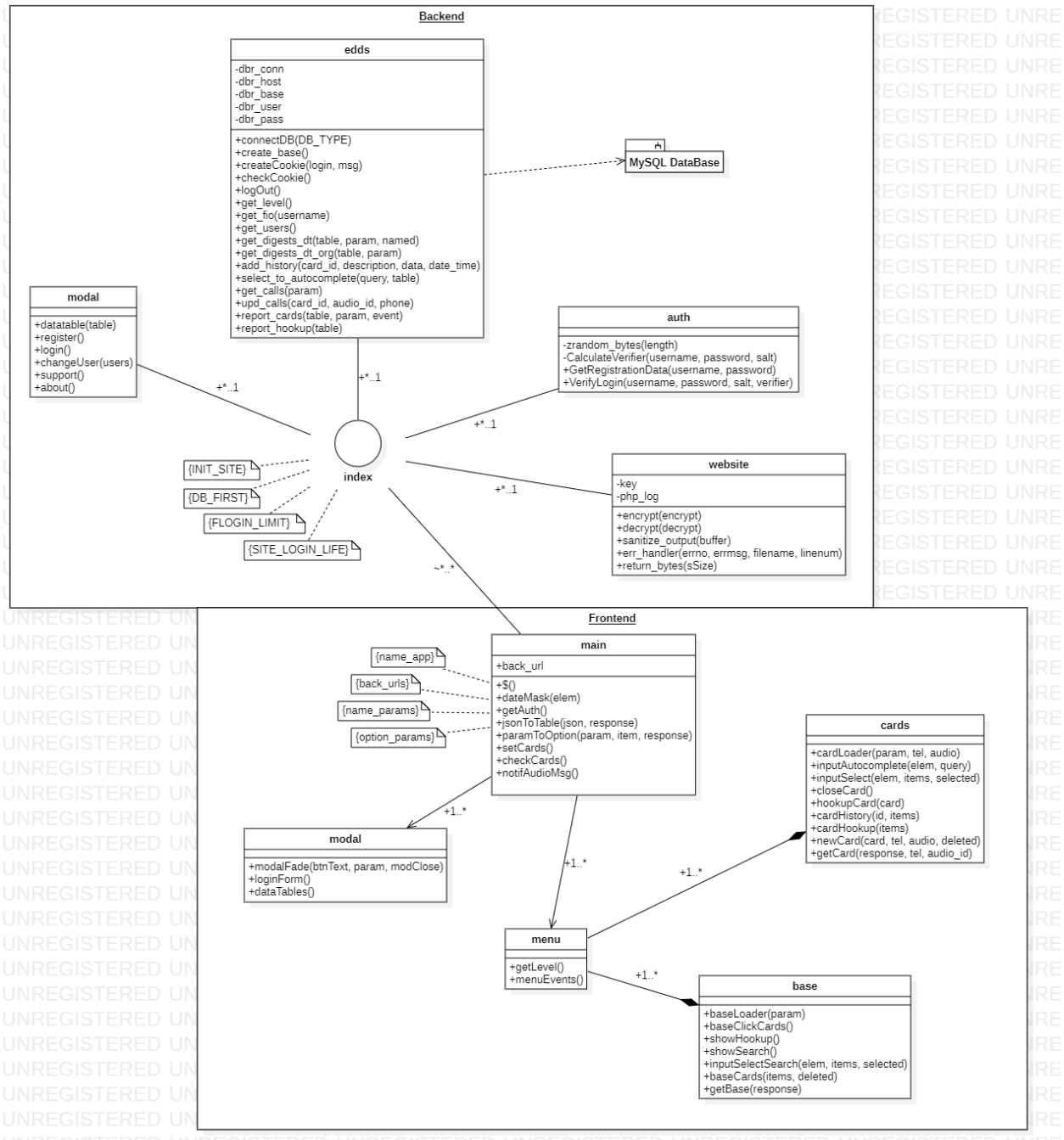


Рисунок А.1 – Расширенная диаграмма классов

Приложение Б

Прочий функционал веб-приложения

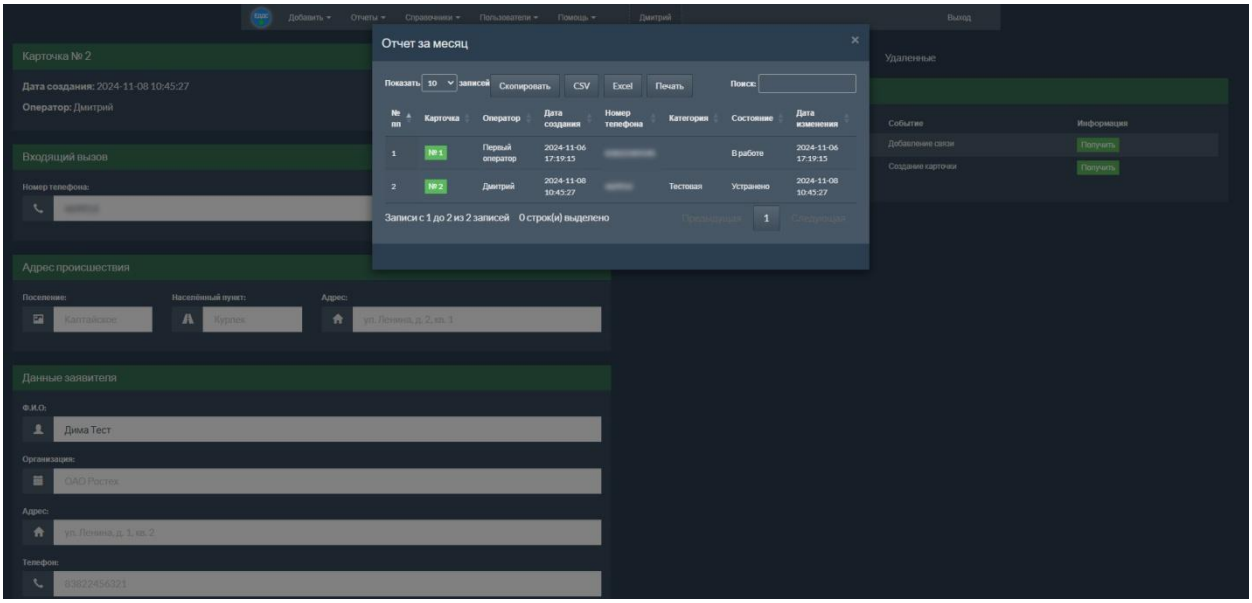


Рисунок Б.1 – Экранная форма модального окна с отчетностью за месяц

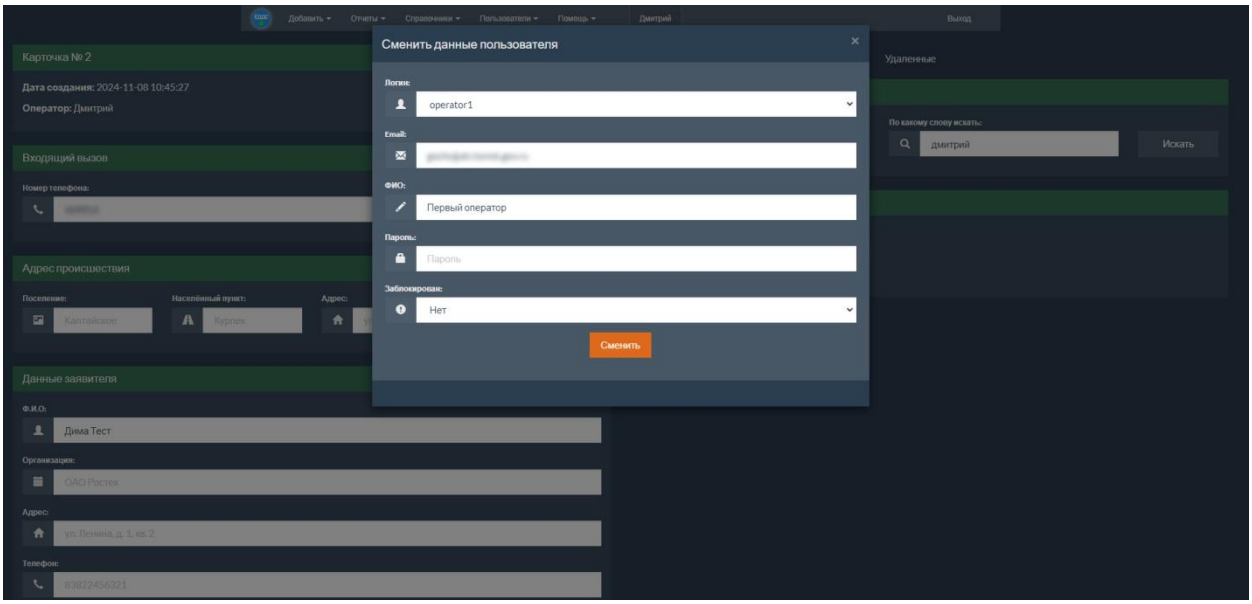


Рисунок Б.2 – Экранная форма изменения данных пользователя администратором системы

Приложение В

Прототип базы данных веб-приложения

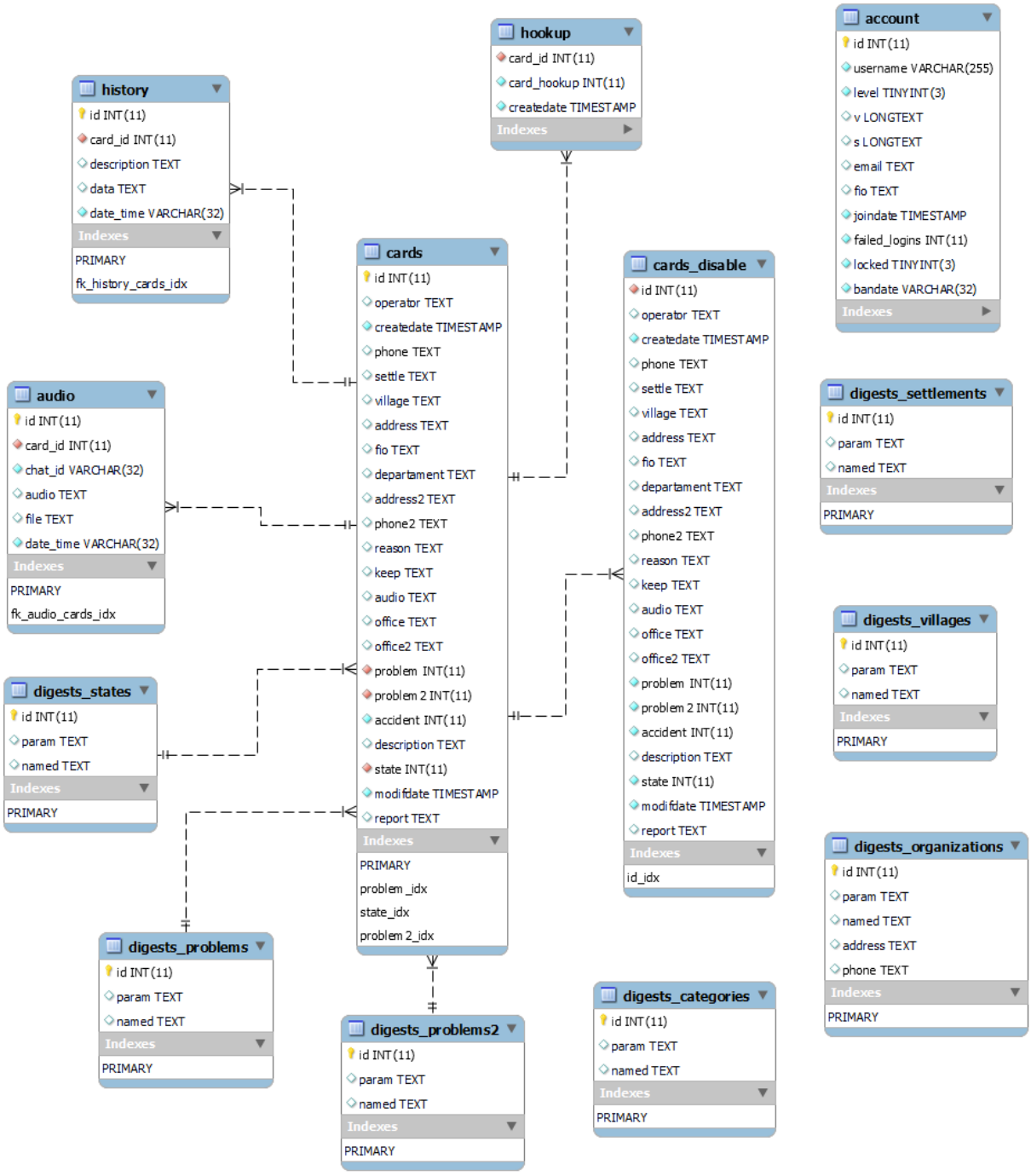


Рисунок В.1 – Прототип базы данных веб-приложения в программе MySQL Workbench

Приложение Г

Контрольный пример реализации проекта

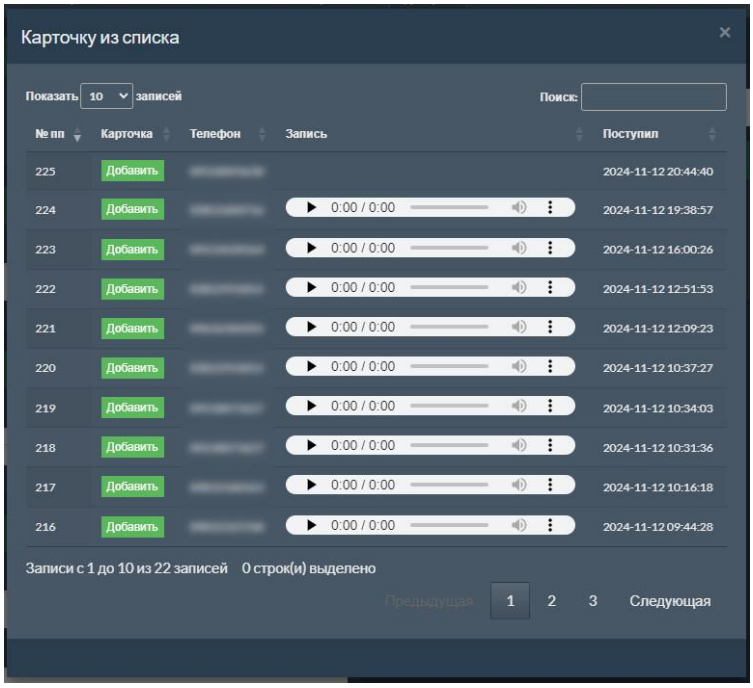


Рисунок Г.1 – Экранная форма со списком звонков в момент поступления нового вызова

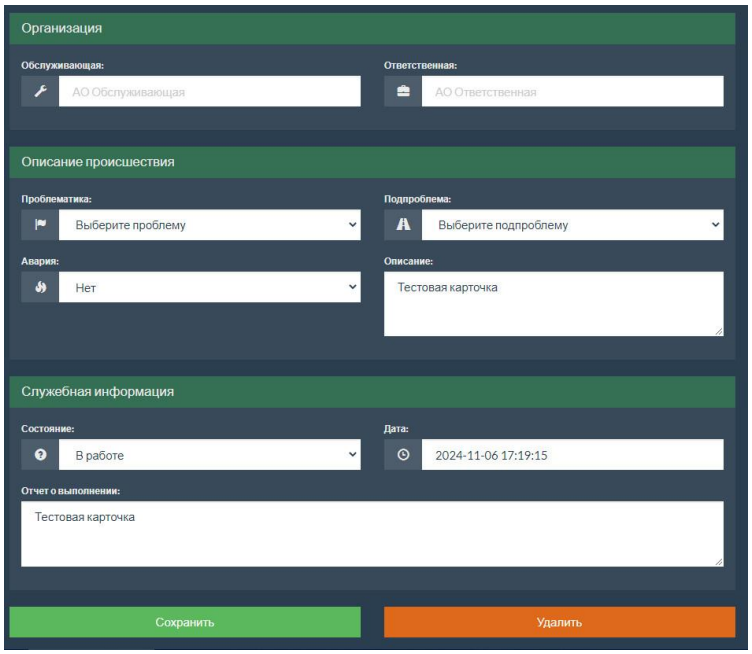


Рисунок Г.2 – Экранная форма с заполнением карточки

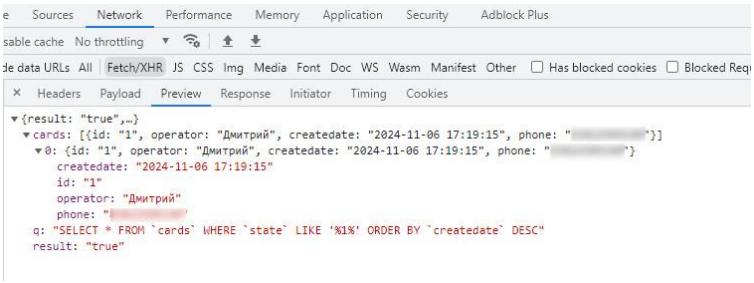


Рисунок Г.3 – Результат поиска и вернувшийся запрос к базе данных

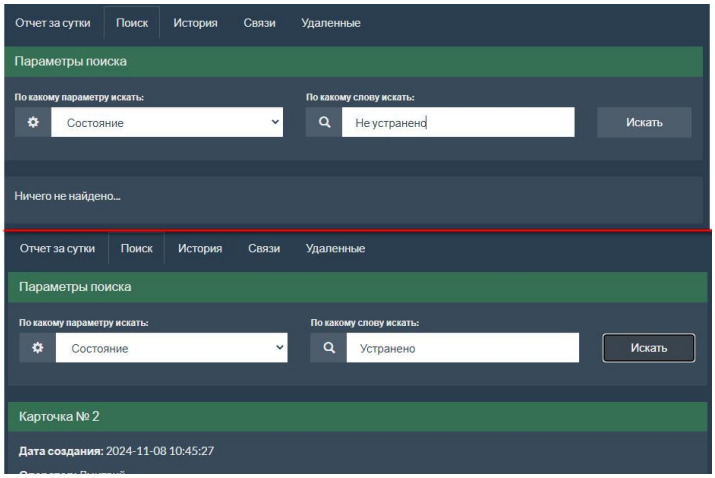


Рисунок Г.4 – Результат поиска текстовых запросов по двум состояниям

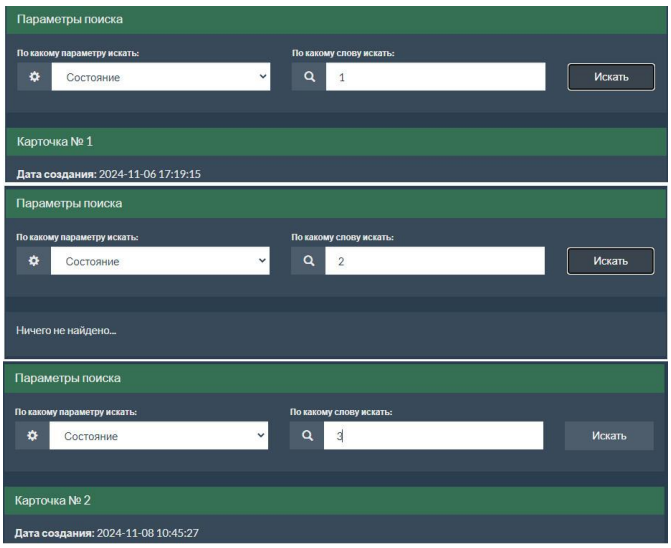


Рисунок Г.5 – Результат поиска числовых запросов состояний