**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(ВлГУ)**

**ДИПЛОМНАЯ РАБОТА**

Студент Лаврухин Максим Дмитриевич

Колледж инновационных технологий и предпринимательства

Специальность 09.02.07 Информационные системы и программирование

**Тема дипломной работы**

**РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ УЧЁТА И КОНТРОЛЯ ЗАЯВОК ПОЛЬЗОВАТЕЛЕЙ ИНТЕРНЕТ-ПРОВАЙДЕРА**

Руководитель    А.В. Лоханов

Студент    М.Д. Лаврухин

**Допустить дипломную работу к защите**

**в государственной экзаменационной комиссии**

Заведующий кафедрой ФиПМ      Д.В. Абрамов

«15» июня 2025 г.

**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

**(ВлГУ)**

УТВЕРЖДАЮ

Зав. кафедрой ФиПМ

\_\_\_\_\_\_\_\_\_\_ Д.В. Абрамов

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**ЗАДАНИЕ**

**НА ДИПЛОМНУЮ РАБОТУ**

Студенту   *Лаврухину Максиму Дмитриевичу*

1. Тема работы   *Разработка веб-приложения учёта и контроля заявок пользователей интернет-провайдера.*

утверждена приказом по университету № *073/15 от 18.02.2025 года*

2. Срок сдачи студентом законченной работы    *15.06.2025 года*

3. Исходные данные к работе   *Веб-Фреймворк Blazor; СУБД SQLite*

4. Содержание расчетно-пояснительной записки (перечень подлежащих

разработке вопросов):   
*1) Анализ современных систем обработки обращений.   
   2) Формирование тестовых требований к системе.   
   3) Разработка архитектуры приложения.   
   4) Реализация пользовательского интерфейса.*

*5) Проведение тестирования приложения и анализ полученных результатов.*

5. Перечень графического материала (с указанием обязательных чертежей):

*1) Название работы.*

*2) Постановка задачи.*

*3) Диаграммы UML.*

*4) Макеты пользовательского интерфейса.*

*5) Разработанный интерфейс пользователя.*

*6) Результаты тестирования.*

6. Консультанты *Кабанова М.Ю. (нормоконтроль)*

Дата выдачи задания    *18.02.2025 года*

Руководитель А.В. Лоханов

(подпись)

Задание принял к исполнению М.Д. Лаврухин

(подпись студента)

АННОТАЦИЯ

Целью дипломной работы являлась разработка веб-приложения для автоматизации обработки заявок интернет-провайдера. Актуальность темы обусловлена ростом спроса на цифровые услуги и необходимостью оптимизации управления ресурсами компании. В процессе работы изучены существующие системы, определены требования, спроектирована архитектура и база данных, разработан пользовательский интерфейс с использованием современных технологий, реализованы функции подачи и распределения заявок, проведено тестирование. Результатом стало веб-приложение, способное повысить качество обслуживания абонентов.

Практическая значимость работы заключается во внедрении решения, способного снизить затраты компании и повысить удовлетворенность клиентов.

Пояснительная записка содержит 65 с., 45 рис., 12 источников, 2 прил.

SUMMARY

The aim of the thesis was to develop a web application for automating internet provider’s request processing. The relevance stems from the growing demand for digital services and the need to optimize resource management. The work included analysis of existing systems, requirement specification, database and architecture design, UI development with modern technologies, implementation of request submission and allocation features, as well as comprehensive testing. The resulting application improves customer service quality.

The practical value lies in deploying a solution that reduces costs and increases customer satisfaction.

The explanatory note contains 65 pages, 45 figures, 12 sources, 2 appendicies.

СОДЕРЖАНИЕ

[Введение 6](#_Toc200963384)

[1 Анализ предметной области 7](#_Toc200963385)

[1.1 Описание предметной области 7](#_Toc200963386)

[1.2 Анализ существующих решений 8](#_Toc200963387)

[1.3 Потребности пользователей и заинтересованных лиц 13](#_Toc200963388)

[2 Постановка задачи 15](#_Toc200963389)

[2.1 Требования к разрабатываемой системе 15](#_Toc200963390)

[2.1.1 Функциональные требования 15](#_Toc200963391)

[2.1.2 Нефункциональные требования 17](#_Toc200963392)

[2.2 Обоснование выбора инструментальных средств 18](#_Toc200963393)

[3 Проектирование приложения 20](#_Toc200963394)

[3.1 Архитектурное проектирование 20](#_Toc200963395)

[3.2 Проектирование базы данных 21](#_Toc200963396)

[3.3 Проектирование интерфейса пользователя 22](#_Toc200963397)

[3.5 Диаграммы UML 25](#_Toc200963398)

[3.5.1 Диаграмма классов 25](#_Toc200963399)

[3.5.2 Диаграмма прецедентов 26](#_Toc200963400)

[3.5.3 Диаграммы деятельности 28](#_Toc200963401)

[3.5.4 Диаграмма развертывания 30](#_Toc200963402)

[3.5.5 Диаграмма компонентов 31](#_Toc200963403)

[4 Разработка приложения 33](#_Toc200963404)

[4.1 Реализация архитектуры 33](#_Toc200963405)

[4.2 Реализация базы данных 34](#_Toc200963406)

[4.3 Реализация модулей 37](#_Toc200963407)

[4.4 Реализация пользовательского интерфейса 37](#_Toc200963408)

[5 Тестирование приложения 45](#_Toc200963409)

[5.1 Тест-требования 45](#_Toc200963410)

[5.2 Тест-план 46](#_Toc200963411)

[5.3 Результаты тестирования 48](#_Toc200963412)

[Заключение 53](#_Toc200963413)

[Список использованных источников 54](#_Toc200963414)

[Приложение А 56](#_Toc200963415)

[Приложение Б 61](#_Toc200963416)

# Введение

Активное расширение оптоволоконных сетей в регионах приводит к росту спроса на цифровые услуги. В связи с этим происходит увеличение числа заявок на подключение и техническое обслуживание, создавая повышенную нагрузку на интернет-провайдеров. В таких условиях традиционные методы управления персоналом становятся неэффективными: возникают ошибки и задержки при обработке запросов. Разрабатываемое веб-приложение решает эту проблему за счет автоматизации процессов регистрирования заявок, распределения задач между инсталляторами. Это позволит повысить скорость и качество обслуживания клиентов, снизить затраты и улучшить управление ресурсами компании. Таким образом, проект актуален как для интернет-провайдеров, стремящихся оптимизировать свою работу, так и для абонентов, заинтересованных в быстром и профессиональном сервисе.

Для достижения поставленной цели необходимо решить следующие задачи:

* Исследовать современные системы и платформы для управления заявками, выделив их преимущества.
* Составить перечень функциональных и технических требований к приложению.
* Разработать модульную структуру приложения.
* Разработать удобный интерфейс с адаптивным дизайном, учитывающий особенности работы разных категорий пользователей.
* Реализовать ключевые подсистемы, включающие прием и обработку заявок, распределение задач, механизмы формирования отчетности.
* Провести проверку системы.

# 1 Анализ предметной области

## 1.1 Описание предметной области

Разрабатываемое приложение является онлайн-платформой, предназначенной для эффективного распределения заявок от новых клиентов на подключение цифровых услуг, а также запросов от текущих абонентов на устранение неполадок, связанных с использованием интернета, интерактивного телевидения, мобильной связи и видеонаблюдения.

В приложении предусмотрены три роли пользователей: диспетчер, инсталлятор и руководитель.

Диспетчер связывается с клиентом по номеру телефона, указанному в форме обратной связи, чтобы согласовать удобную дату и время визита инсталлятора. При необходимости уточняются детали заявки, после чего она подтверждается.

Подтвержденные заявки поступают в систему учета, где распределяются между инсталляторами, формируя их рабочий график на день вперед, при этом в целях сокращения времени, затрачиваемого на транспортировку от адреса к адресу, система выделяет для сотрудников набор запросов, поступивших в радиусе одного района.

Руководители имеют расширенный доступ к управлению персоналом и аналитике. Они могут добавлять новых сотрудников, настраивать их права доступа, а также отслеживать статистику по выполненным заявкам, оценивая эффективность работы подчиненных.

Кроме прочих, задачей сервиса является предоставление статистики о характере проблем, возникающих у пользователей.

Таким образом, веб-приложение служит рабочим инструментом как для диспетчеров, так и для инсталляторов, которые фиксируют свои результаты работы: проблема устранена или требуется повторный выезд, абонент подключен к услуге(ам) или отказался по определенным причинам.

## 1.2 Анализ существующих решений

Приложение «МТС Координатор» позволяет легко организовать эффективную работу разъездного персонала – торговых представителей, курьеров водителей, сервисных инженеров и др.

В возможности пользователя личного кабинета входит:

* Планирование рабочего дня с помощью назначения задач в автоматическом и ручном режимах,
* Передача требуемой информации в задаче (адрес, телефон, комментарий), а также её кастомизация с помощью дополнительных пользовательских полей,
* Контроль выполнения задач в режиме реального времени или с использованием настраиваемой отчетности по факту выполнения задачи или посещения объекта,
* Онлайн нотификация (E-mail, SMS, Telegram) по настраиваемым событиям работы приложения,
* Построение маршрутов и контроль их соблюдения между задачами для минимизации пробега и времени,
* Контроль местоположения и истории перемещений сотрудников,
* Контроль посещений рабочих объектов и гео-зон

Интерфейс «МТС Координатора» представлен на рисунке 1.1.

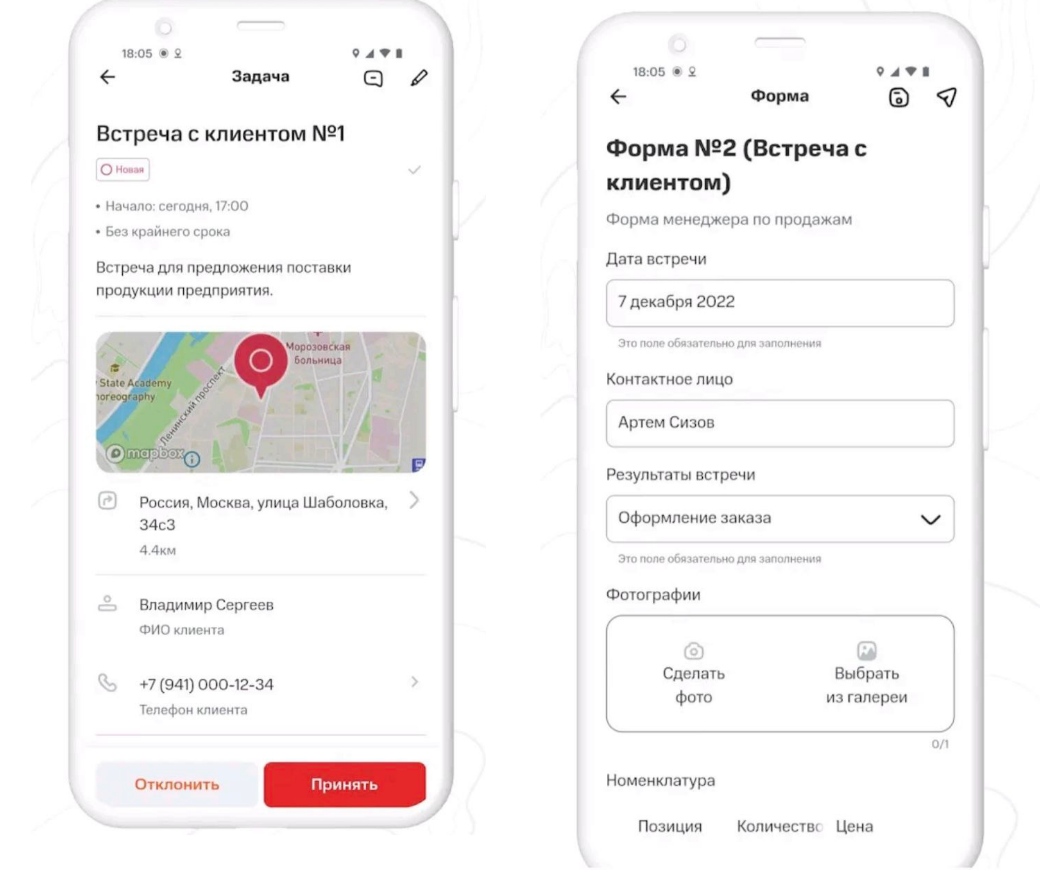


Рисунок 1.1. МТС Координатор

Мобильный агент предназначен для внутреннего использования сотрудниками компании ПАО «Ростелеком».

Среди функционала можно выделить следующее:

* Просмотр маршрутного листа и выбор домов для обхода,
* Создание заявок на подключение,
* Отслеживание статуса принятых в работу заявок,
* Просмотр статистики агента

Интерфейс «Мобильного агента Ростелеком» представлен на рисунке 1.2.

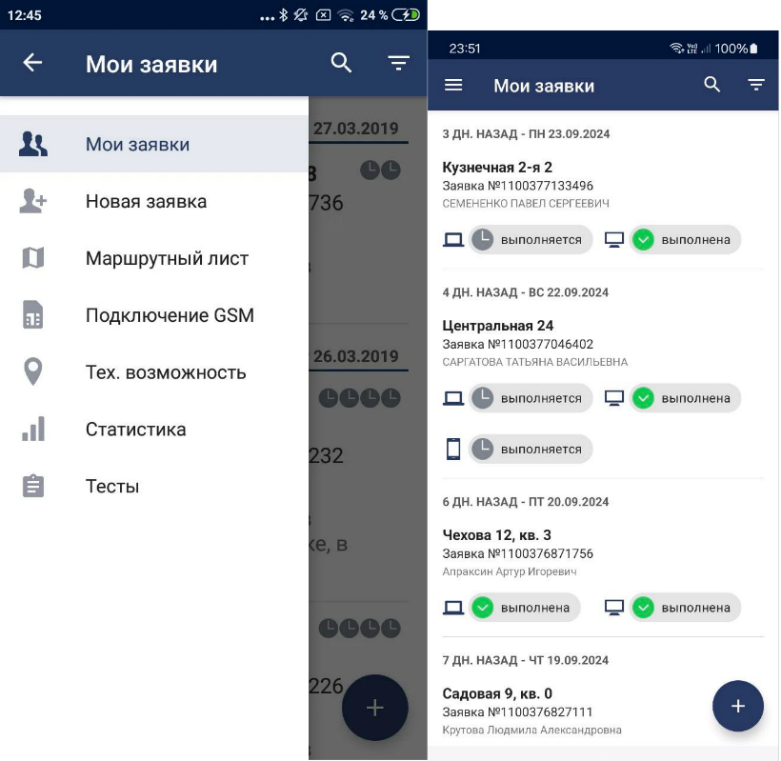


Рисунок 1.2. Мобильный агент (Ростелеком)

Дом.ру разработан для специалистов компании Эр-Телеком Холдинг. Приложение создано для автономной работы сотрудников, объединяет в себе инструменты АРМа, EQM и сервис геолокации для выполнения заказов

В функционал входит:

* Интеграция с АРМ (автоматизированное рабочее место)
* Доступ к системе EQM (управление качеством обслуживания),
* Сервис геолокации и навигации к месту выполнения заказа,
* Просмотр и прием заявок на подключение/ремонт,
* Ввод результатов выполненных работ,
* Доступ к технической документации и регламентам,
* Оффлайн-режим работы

Интерфейс «Дом.ру» представлен на рисунке 1.3.

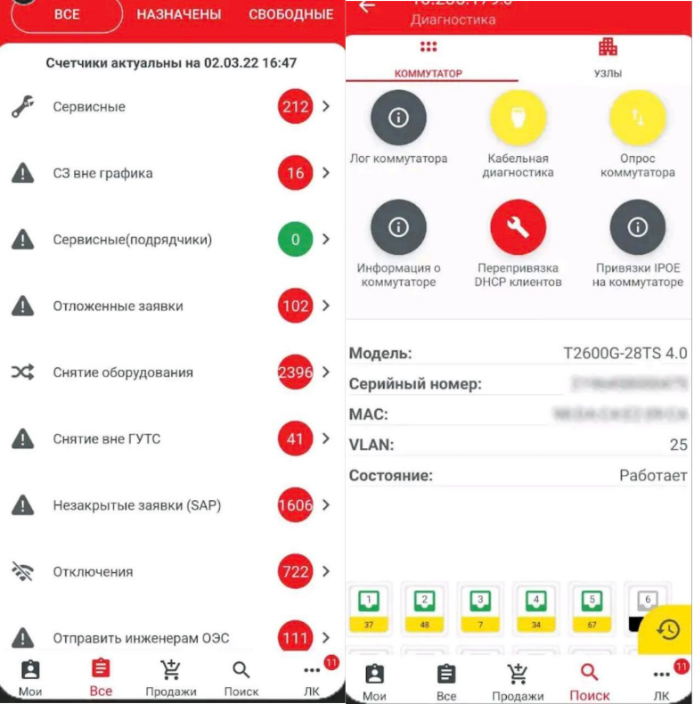


Рисунок 1.3. Рабочее место специалиста (Дом.ру)

BeeTrack предназначен для корпоративных клиентов Билайна. BeeTrack помогает сотрудникам организовать свой рабочий день и работать с задачами, назначенными диспетчером или руководителем. Функционал:

* Просмотр списка задач,
* Установка статуса работы,
* Обмен сообщений с диспетчером,
* Заполнение отчетов по выполненной работе

Интерфейс «BeeTrack» представлен на рисунке 1.4.

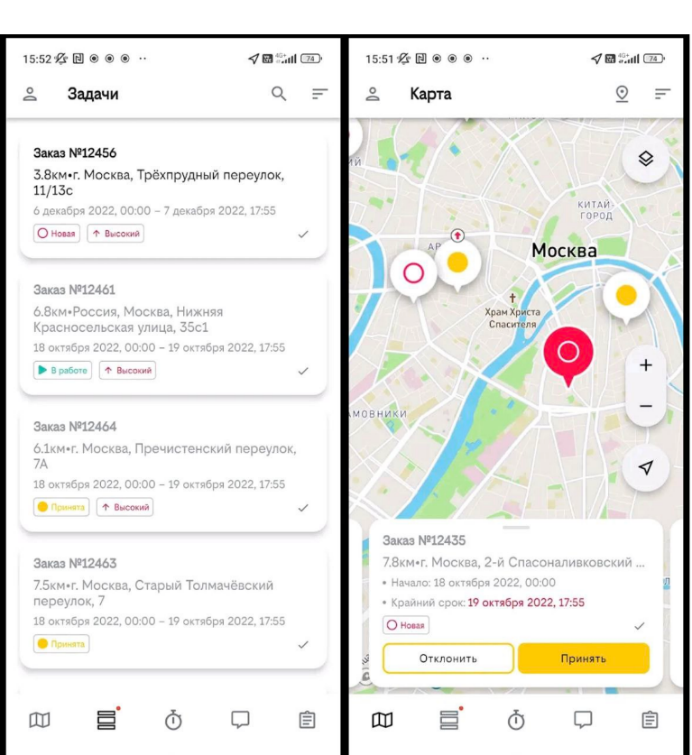


Рисунок 1.4. BeeTrack (Билайн)

ГПБМ разработан специально для услуги «Мониторинг кадров и транспорта», которая доступна для корпоративных клиентов ГПБ Мобайл. Приложение позволяет работать с задачами от диспетчера или руководителя и помогает разъездному сотруднику спланировать свой рабочий день.

Обладает следующим функционалом:

* Прием и выполнение задач от диспетчера/руководителя,
* Планирование рабочего дня и маршрутов,
* Отметки о выполнении задач,
* Функции временного учета,
* Интеграция с системой мониторинга транспорта,
* Возможность оперативной связи с диспетчером

Интерфейс «ГПБМ» представлен на рисунке 1.5.

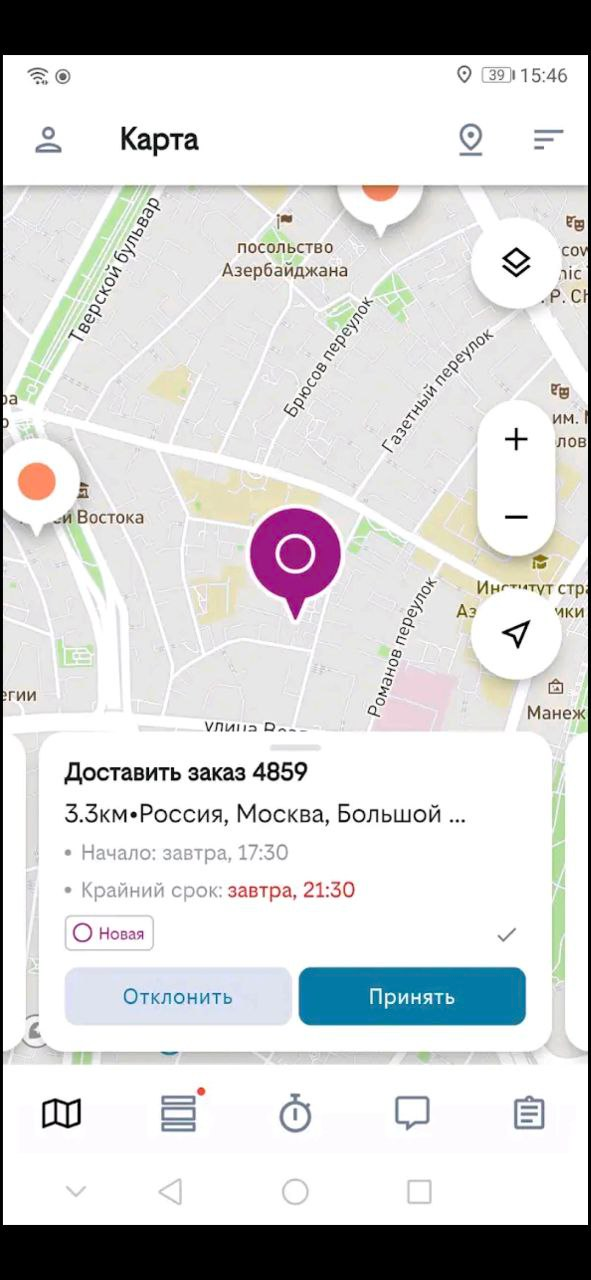


Рисунок 1.5. Газпромбанк Мобайл

Рассмотренные аналоги позволили выделить ключевые подходы к реализации подобных веб-приложений на C#.

## 1.3 Потребности пользователей и заинтересованных лиц

Пользователями разрабатываемого веб-приложения являются сотрудники интернет-провайдера, включая:

Инсталляторов (технических специалистов) – монтажников и сервисных инженеров, которым требуется:

* четкое распределение задач
* мобильный доступ к информации о заявках
* простой способ фиксации результатов работы

Диспетчеров – сотрудников call-центра и технической поддержки, которым важно:

* обрабатывать входящие запросы
* контролировать очередь заявок
* оперативно связываться с клиентами

Руководителей подразделений, заинтересованных в:

* анализе загруженности сотрудников
* контроле качества обслуживания
* получении статистики по типам и частоте обращений

Ключевые потребности всех категорий пользователей:

* Простота и интуитивность интерфейса
* Достаточный функционал для выполнения задач
* Возможность быстрого обучения работе с системой
* Доступ к актуальной информации в режиме реального времени

Система должна обеспечивать удобное взаимодействие между всеми участниками процесса – от момента создания заявки до ее выполнения техническим специалистом и последующего анализа руководством.

# 2 Постановка задачи

## 2.1 Требования к разрабатываемой системе

### 2.1.1 Функциональные требования

Системные характеристики.

* 1. СХ-1: Приложение является веб-приложением, с использованием фреймворка Blazor Server.
  2. СХ-2: Приложение написано на высокоуровневом языке программирования C#.
  3. СХ-3: Приложение взаимодействует с базой данных SQLite.
  4. СХ-4: Работа с базой данных осуществляется с использованием LINQ-запросов.

Пользовательские требования.

1. ПТ-1: Запуск и выход из приложения.
   1. ПТ-1.1: Запуск приложения осуществляется через браузер по указанному URL.
   2. ПТ-1.2: Выход из приложения осуществляется закрытием вкладки браузера или завершением сессии.
2. ПТ-2: Интерфейс.
   1. ПТ-2.1: Главное меню представляет собой горизонтальную панель с кнопками для навигации между разделами приложения.
   2. ПТ-2.2: Данные отображаются в виде таблиц или списков, в зависимости от контекста.
   3. ПТ-2.3: Интерфейс адаптируется под разные разрешения экранов (десктоп и мобильные устройства).
3. БП-1: Работа с заявками.
   1. БП-1.1: Диспетчер может создавать заявки на устранение технических проблем, смену тарифа или подключение новых функций
   2. БП-1.2: Инсталлятор может просматривать список назначенных заявок и обновлять их статус.
   3. БП-1.3: Руководитель может переназначать заявки между инсталляторами.
4. БП-2: Отчеты инсталляторов.
   1. БП-2.1: Инсталлятор создает отчет по выполненной заявке, указав результат и комментарий (опционально).
   2. БП-2.2: Статус заявки должен автоматически обновляться после создания отчета.
5. ДС-1: Детальные спецификации.
   1. ДС-1.1: Язык C#.
   2. ДС-1.2: Минимальная версия .NET — 6.0.
   3. ДС-1.3: Устанавливаемые пакеты NuGet должны быть совместимы с .NET 6.0.
6. ДС-2: Сообщения об ошибках.
   1. ДС-2.1: Сообщение о незаполненных обязательных полях: «Заполните все обязательные поля!».
   2. ДС-2.2: Сообщение о некорректном вводе данных: «Номер телефона должен содержать 11/12 символов и начинаться с +7/8».
   3. ДС-2.3: Сообщение о некорректном пароле: «Пароль должен содержать не менее 8 символов».
7. ДС-3: Отображаемые статусы заявок.
   1. ДС-3.1: Перечень допустимых статусов заявок: «Новая», «Ожидает», «Завершена», «Отложена».
8. ДС-4: Работа с базой данных.
   1. ДС-4.1: Все запросы к базе данных выполняются через LINQ.
   2. ДС-4.2: Данные в интерфейсе обновляются автоматически при изменении данных в базе.
   3. ДС-4.3: Каждая запись в базе данных должна иметь уникальный идентификатор

### 2.1.2 Нефункциональные требования

Атрибуты качества.

1. АК-1: Производительность.
   1. АК-1.1: Запуск приложения должен занимать не более 5 секунд на конфигурации: i3-6100, 4 ГБ ОЗУ, Intel HD 530.
   2. АК-1.2: Выполнение LINQ-запросов к базе данных должно занимать не более 2 секунд на той же конфигурации.
2. АК-2: Интерфейс.
   1. АК-2.1: Интерфейс должен быть интуитивно понятным и удобным для пользователей.
   2. АК-2.2: Основные функции должны быть доступны в 1-2 клика от главного меню.
   3. АК-2.3: Интерфейс должен корректно отображаться на устройствах с разными разрешениями экрана.
3. АК-3: Надежность и целостность данных.
   1. АК-3.1: Система сохраняет данные при неожиданном разрыве соединения.
   2. АК-3.2: Некорректный ввод не приводит к критическим ошибкам.
4. АК-4: Кросс-платформенность.
   1. АК-4.1: Приложение должно корректно функционировать в наиболее популярных браузерах (Chrome, Firefox, Edge, Safari).
5. АК-5: Безопасность.
   1. АК-5.1: Доступ к функционалу ограничен в соответствии с ролями (диспетчер, инсталлятор, руководитель).
   2. АК-5.2: Защита от SQL-инъекций реализована для всех полей ввода.
6. АК-6: Обработка ошибок
   1. АК-6.1: Пользователь получает понятные сообщения при возникновении проблемы.

## 2.2 Обоснование выбора инструментальных средств

Средой разработки приложения выбрана Microsoft Visual Studio – интегрированная среда разработки, предоставляющая удобный отладчик, систему контроля версий, богатую систему расширений и функции IntelliSense для автодополнения кода.

Приложение разрабатывается на платформе Blazor Server, которая обеспечивает тесную интеграцию с инфраструктурой .NET. Компонентная архитектура фреймворка способствует созданию решений, где критически важная логика безопасно выполняется на стороне сервера.

Серверная часть приложения будет написана на языке программирования C# – современном объекто-ориентированном языке, который сочетает высокую производительность с кроссплатформенностью. Для реализации бизнес-логики служит богатая библиотека .NET и экосистема пакетов NuGet.

Для работы с данными используется СУБД SQLite с инструментом DB Browser for SQLite. Эта система управления базами данных была выбрана благодаря своей компактности, надежности и простоте развертывания. DB Browser for SQLite предоставляет интуитивно понятный графический интерфейс для администрирования, а возможность хранения всей базы данных в одном файле значительно упрощает перенос и развертывание приложения.

В процессе разработки проекта используется система контроля версий Git и веб-сервис для хостинга проектов GitHub, обеспечивающий надежное хранение исходного кода с полным сохранением истории всех изменений, что позволяет в любой момент вернуться к предыдущим версиям проекта.

# 3 Проектирование приложения

## 3.1 Архитектурное проектирование

Разрабатываемая система представляет собой веб-приложение, построенное на архитектурном шаблоне Model-View-ViewModel с использованием Entity Framework Core для взаимодействия с базой данных.

Структура организована следующим образом:

* Модели – классы, описывающие основные сущности системы (пользователи, заявки, услуги, тарифы и др.) и их взаимосвязи.
* Представления – компоненты пользовательского интерфейса, отвечающие за отображение данных и взаимодействие с пользователем. Реализованы с использованием Razor Pages..
* Модели представления – промежуточные классы, которые адаптируют данные моделей для отображения в представлениях, содержат логику преобразования данных и команды для обработки пользовательских действий.

Таким образом, бизнес-логика приложения распределена между моделями и моделями представления, что обеспечивает удобство дальнейшего развития системы без необходимости кардинального изменения архитектуры.

## 3.2 Проектирование базы данных

Разработанная ER – диаграмма в нотации Мартина представлена на рисунке 3.1.

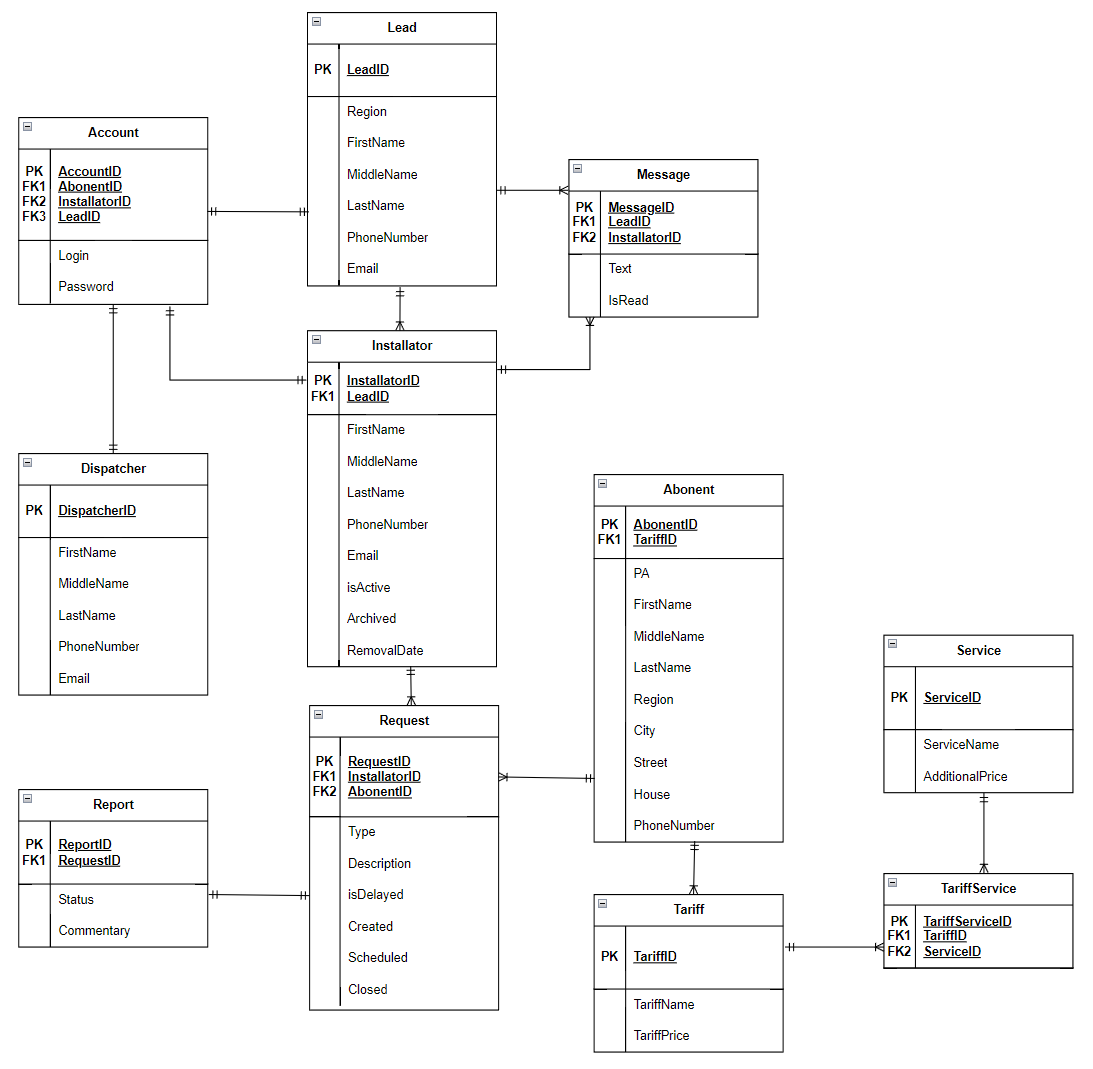


Рисунок 3.1 − ER − диаграмма в нотации Мартина

Созданная ER-диаграмма четко демонстрирует объекты предметной области, их характеристики и взаимосвязи.

## 3.3 Проектирование интерфейса пользователя

Перед началом разработки приложения были спроектированы макеты пользовательского интерфейса.

Макет страницы авторизации представлен на рисунке 3.2 Пользователю необходимо ввести электронную почту или номер телефона и пароль. Приложение предназначено только для сотрудников, поэтому функция регистрации недоступна. В случае утери пароля предусмотрено его восстановление через письмо, отправленное на электронную почту.

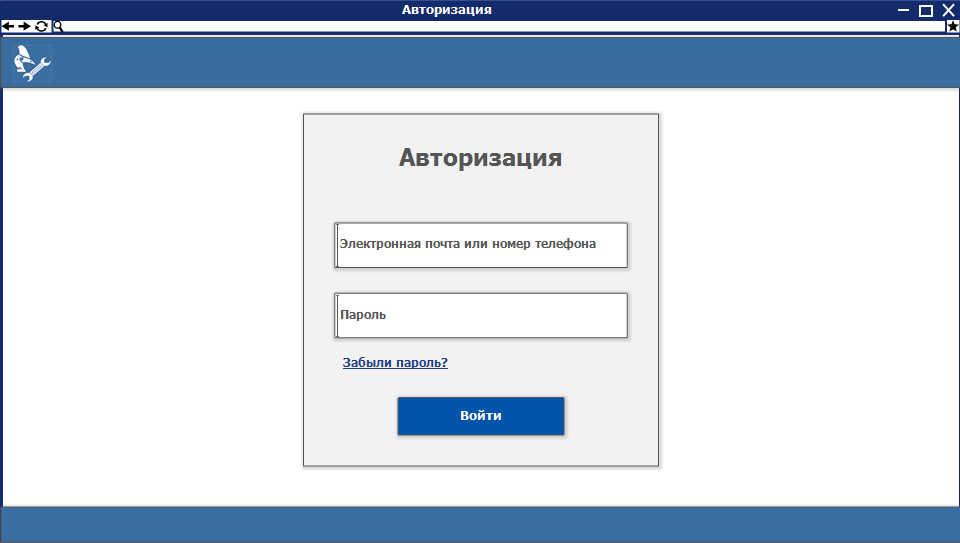


Рисунок 3.2 – Макет страницы авторизации

Макет страницы с расписанием выездов представлен на рисунке 3.3. На этой странице отображается список вызовов, назначенных сотруднику.

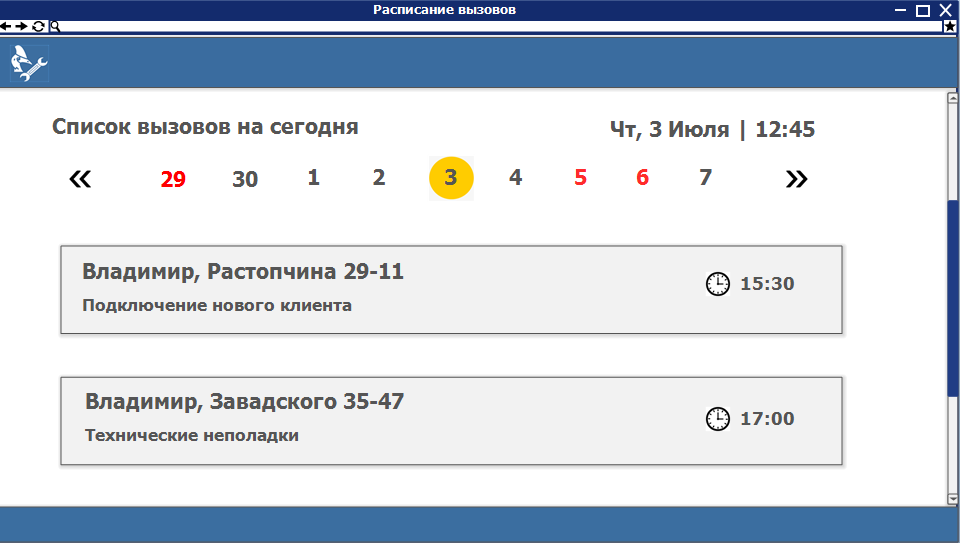


Рисунок 3.3 – Макет страницы с расписанием выездов

Макет страницы с информацией о выезде представлен на рисунке 3.4. Здесь представлены детали заявки.

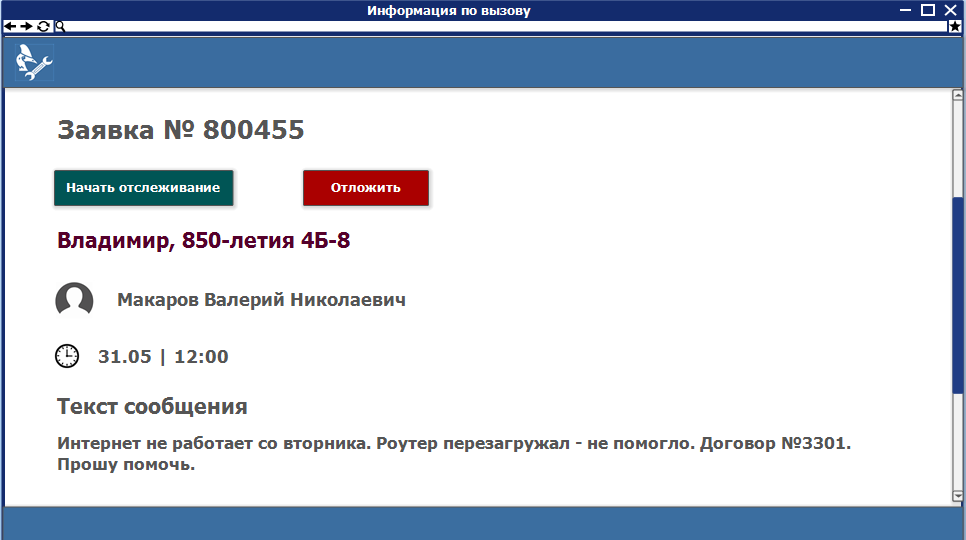


Рисунок 3.4 – Макет страницы с информацией о выезде

Макет страницы руководителя представлен на рисунке 3.5. В этом разделе отображен список сотрудников отдела.

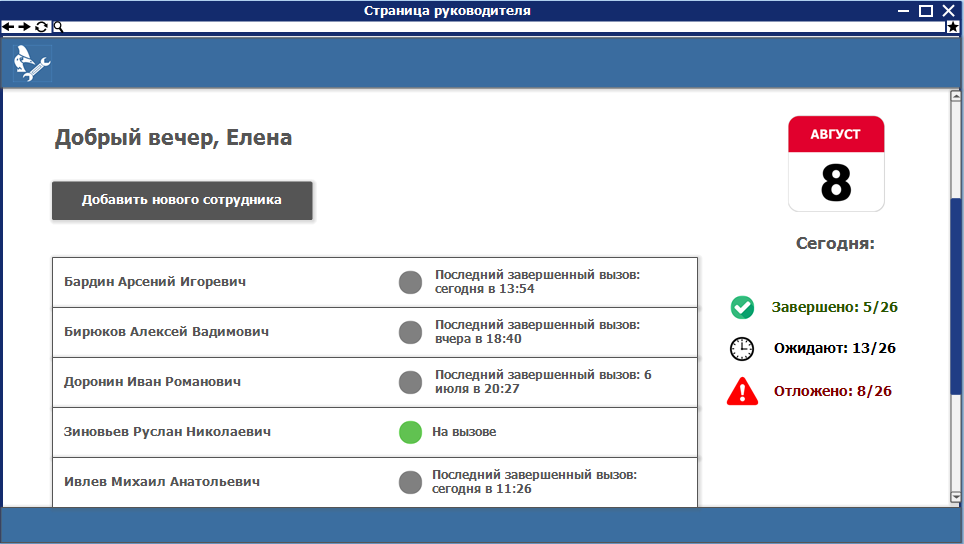


Рисунок 3.5 – Макет страницы руководителя

Макет страницы с сотрудником представлен на рисунке 3.6. Здесь указаны контактные данные подчиненного и его график работы.

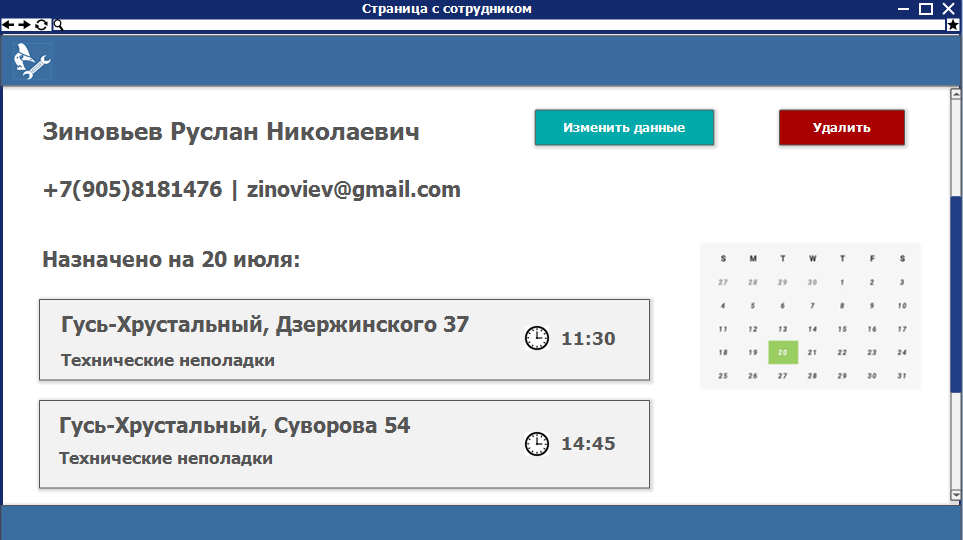


Рисунок 3.6 – Макет страницы с сотрудником

За каждым сотрудником ведется учет отработанных часов. Макет страницы со статистикой представлен на рисунке 3.7.

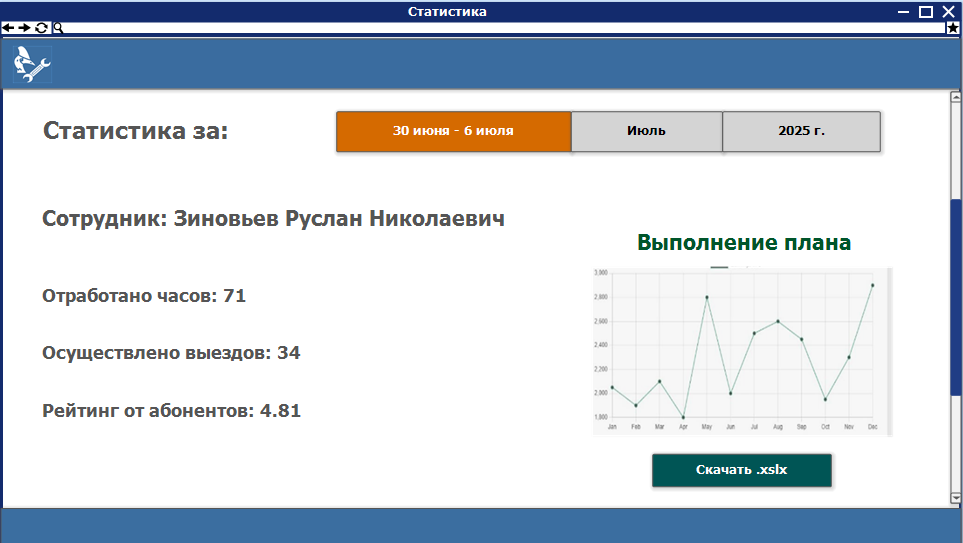


Рисунок 3.7 – Макет страницы со статистикой

В итоге был разработан прототип пользовательского интерфейса, который иллюстрирует ключевые функциональные возможности и сценарии использования приложения

## 3.5 Диаграммы UML

### 3.5.1 Диаграмма классов

Диаграмма классов (англ. class diagram) — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними [8].

На рисунке 3.8 представлена диаграмма классов.

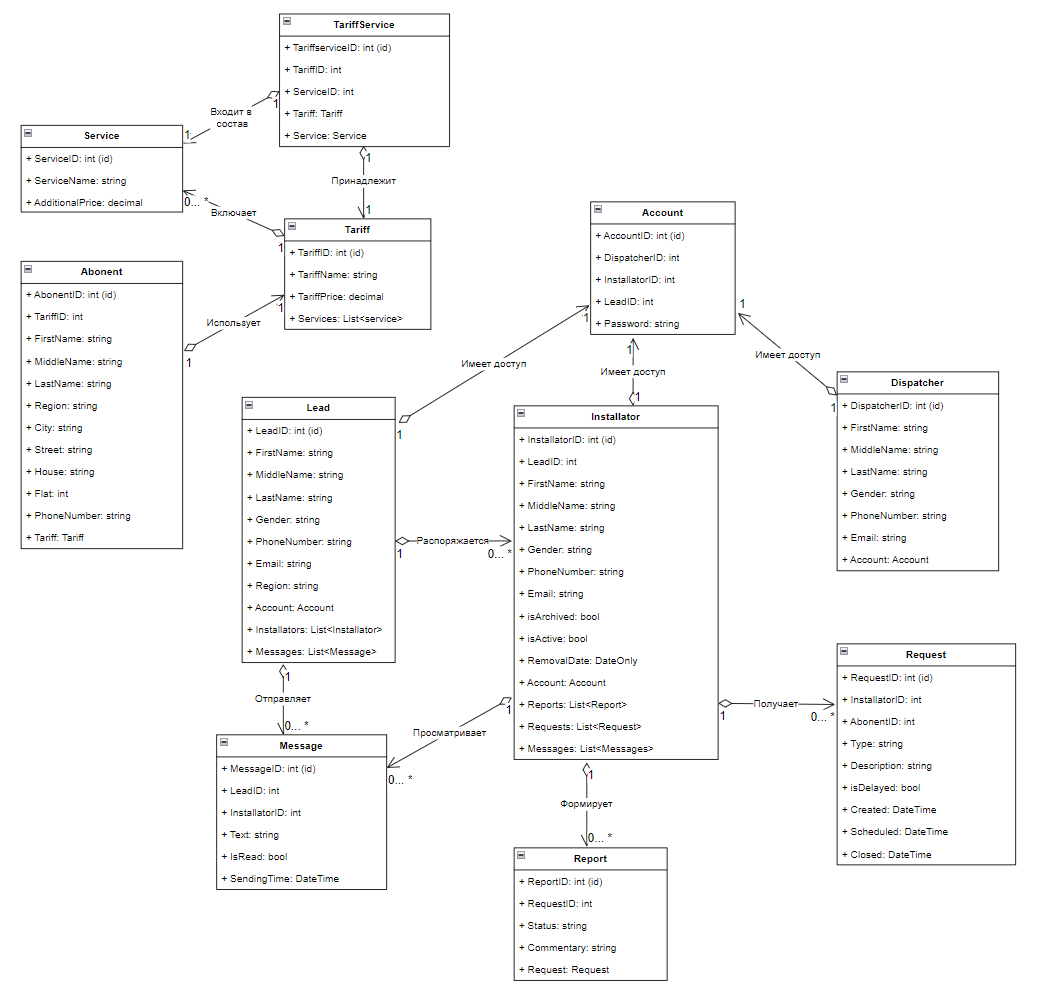


Рисунок 3.8 – Диаграмма классов

Таким образом диаграмма классов предоставляет четкое представление о структуре системы и взаимосвязях между её компонентами.

### 3.5.2 Диаграмма прецедентов

Диаграмма вариантов использования (англ. use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей [9].

Диаграмма прецедентов представлена на рисунке 3.9.

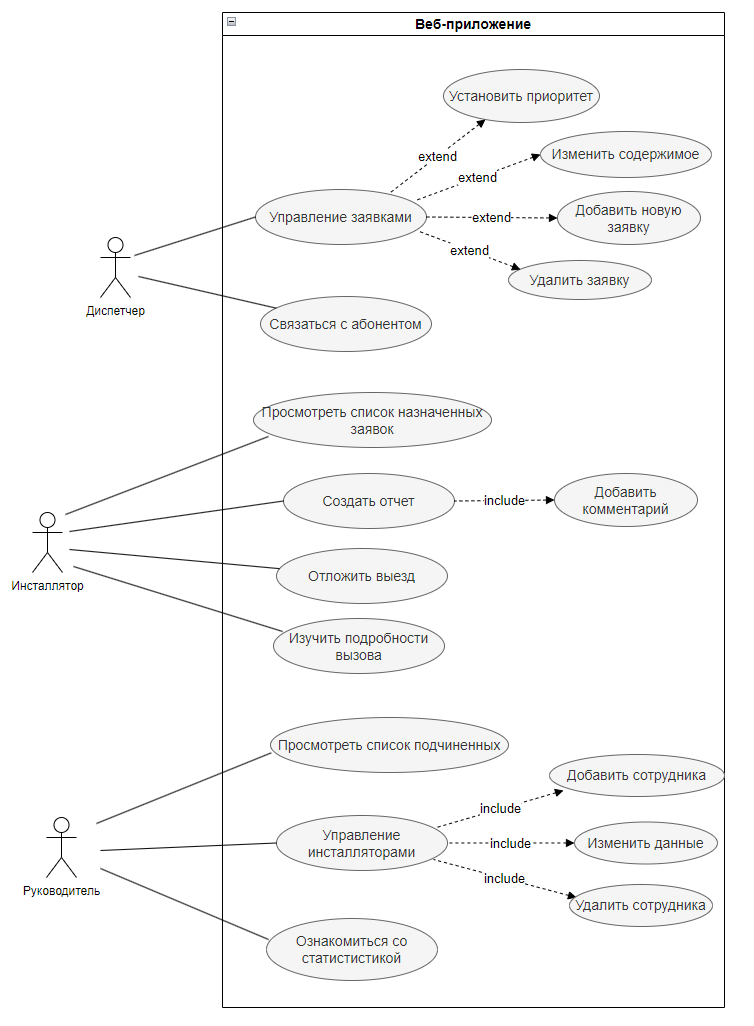


Рисунок 3.9 – Диаграмма прецедентов

В диаграмме прецедентов представлены три актера: инсталлятор, диспетчер, руководитель. Инсталлятор смотрит расписание, фиксирует результаты и заполняет отчёты. Диспетчер добавляет новые заявки. Руководитель управляет сотрудниками и отслеживает статистику.

### 3.5.3 Диаграммы деятельности

Диаграмма деятельности отражает динамические аспекты поведения системы и представляет собой блок-схему, которая наглядно показывает, как поток управления переходит от одной деятельности к другой [10].

Диаграмма деятельности для обработки заявки представлена на рисунке 3.10.

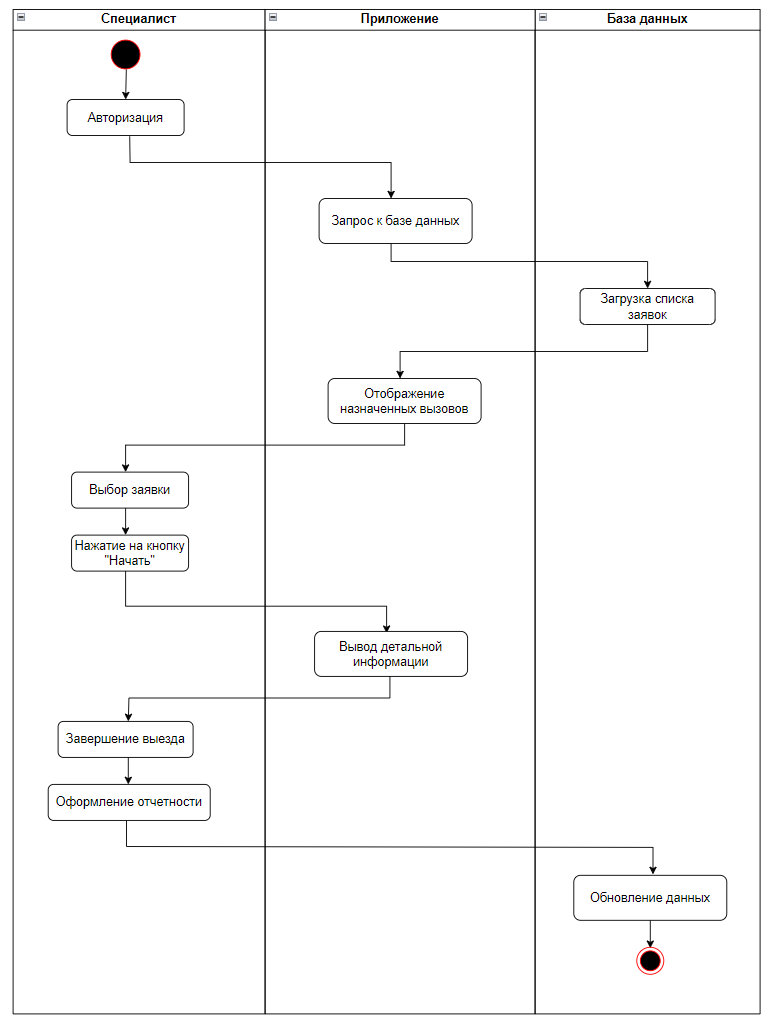


Рисунок 3.10 – Диаграмма активности. Обработка заявки

Перед завершением выезда инсталлятор указывает причину проблемы и способ ее устранения во всплывающем окне. Для заявок на подключение достаточно нажать «Завершить».

Диаграмма деятельности для добавления нового сотрудника на рисунке 3.11.

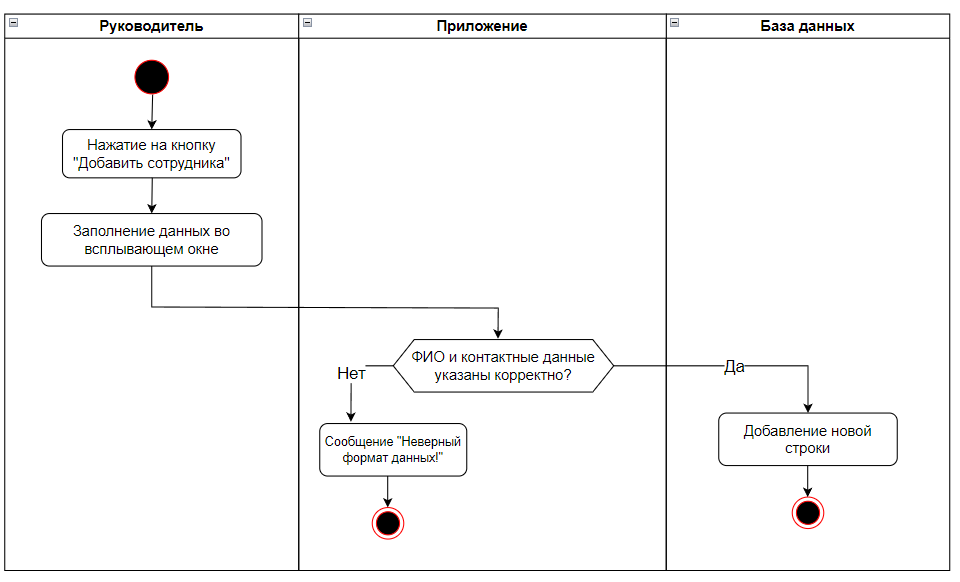


Рисунок 3.11 – Диаграмма активности. Добавление нового сотрудника

Приложение проверяет корректность вводимых данных. Фамилия и имя не должны содержать цифр, номер телефона должен начинаться с +7 или 8.

Диаграмма деятельности для передачи заявки другому лицу представлена на рисунке 3.12.

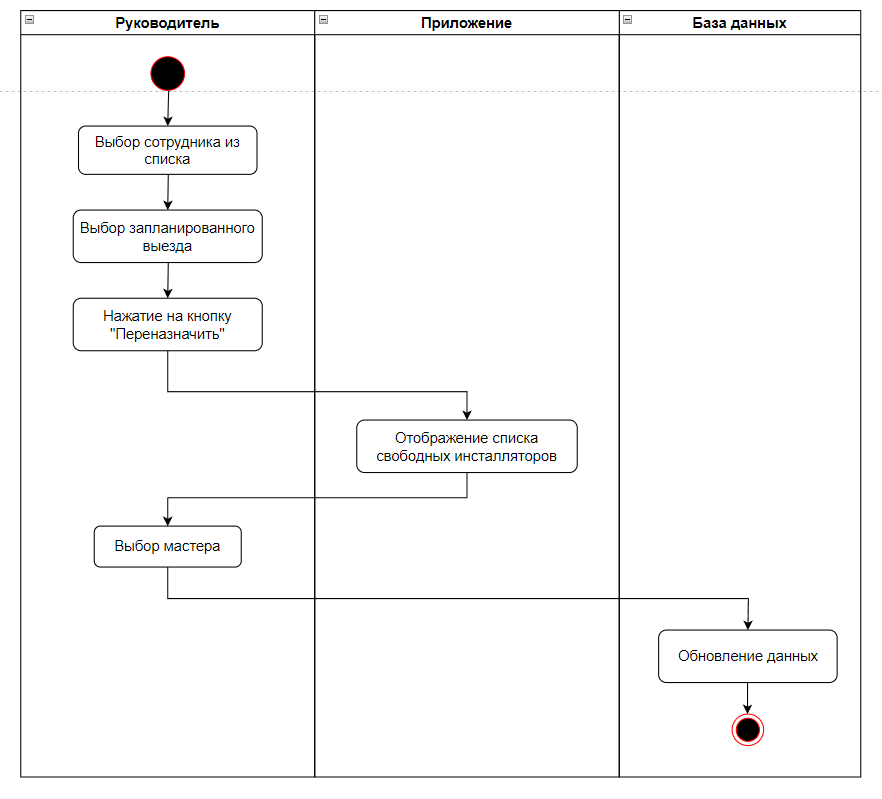


Рисунок 3.12 – Диаграмма активности. Переназначение заявки другому специалисту

Приложение учитывает запланированное время выезда и выбирает только тех специалистов, у которых в расписании в этот момент еще есть свободное время.

### 3.5.4 Диаграмма развертывания

Диаграмма развертывания – это тип UML-диаграммы, которая показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их [11].

Диаграмма развертывания представлена на рисунке 3.13.

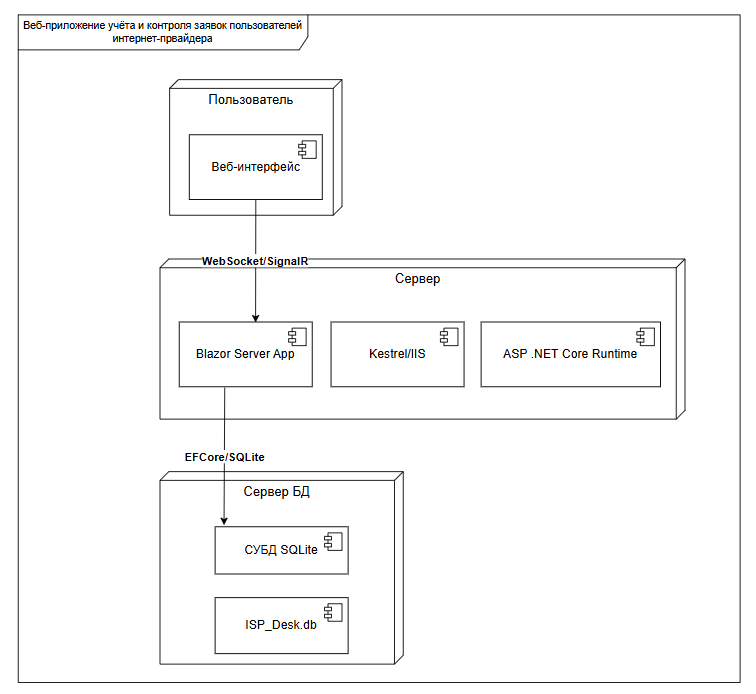


Рисунок 3.13 – Диаграмма развертывания

Итак, диаграмма развертывания отображает физическую архитектуру системы.

### 3.5.5 Диаграмма компонентов

Диаграмма компонентов (англ. Component diagram) — элемент языка моделирования UML, статическая структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами [12].

Диаграмма компонентов представлена на рисунке 3.14.

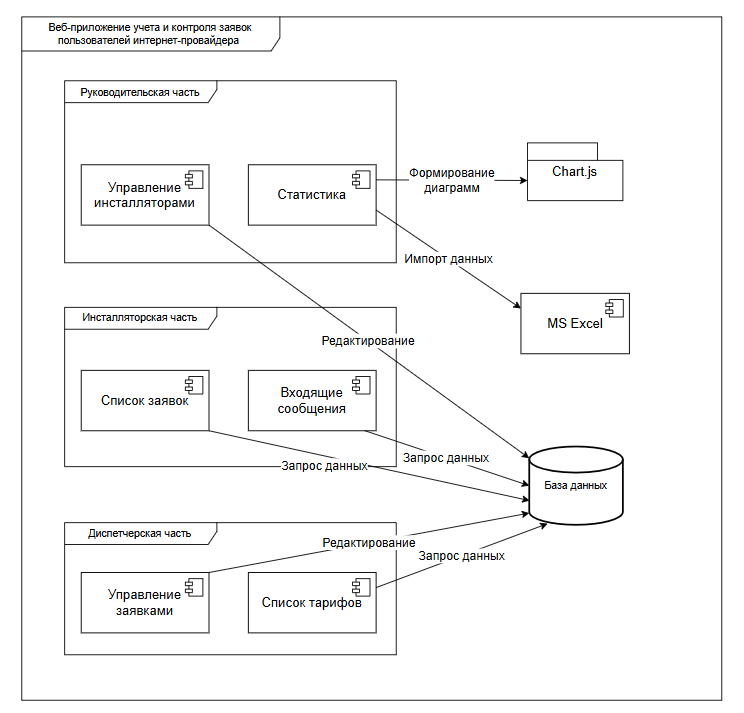


Рисунок 3.14 – Диаграмма компонентов

Таким образом. диаграмма компонентов описывает логическую организацию системы.

# 4 Разработка приложения

## 4.1 Реализация архитектуры

Для реализации функционала работы с данными в системе было создано одиннадцать моделей, описывающих основные сущности приложения:

* Abonent
* Account
* Dispatcher
* Installator
* Lead
* Message
* Report
* Request
* Service
* Tariff
* TariffService

Листинг кода реализованных моделей представлен в приложении А.

Классы моделей и класс AppDbContext.cs представлены на рисунке 4.1.

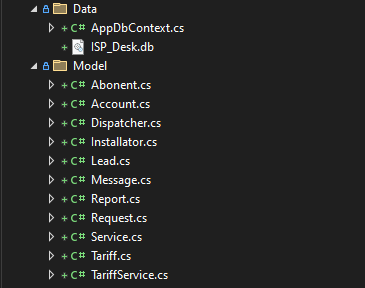


Рисунок 4.1 – Модели данных

Таким образом, в системе определены одиннадцать моделей данных, охватывающих все сущности приложения.

## 4.2 Реализация базы данных

Реализованная в SQLite база данных имеет структуру, соответствующую разработанной ER-диаграмме. Разработанная база данных представлена на рисунках 4.2 – 4.3.

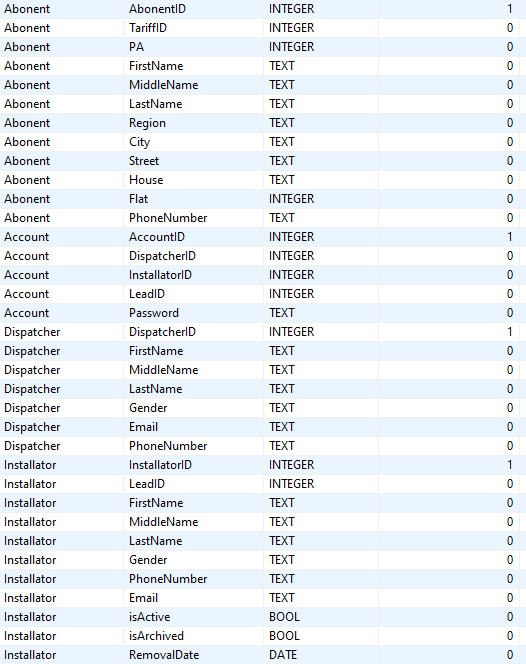


Рисунок 4.2 – Структура базы данных



Рисунок 4.3 – Структура базы данных

Таким образом, все сущности, их атрибуты и взаимосвязи, представленные на диаграмме, нашли свое точное отражение в структуре созданной базы данных.

## 4.3 Реализация модулей

При разработке приложения было создано три класса-сервиса.

1. ChartService – предназначен для визуализации данных в виде круговых диаграмм. Он представляет функционал для отображения двух диаграмм: верхней (DrawTopChart), отображающей общую статистику по статусам заявок и нижней (DrawBottomChart), показывающей распределение заявок по типам проблем.
2. CryptoService – служит для симметричного шифрования и дешифрования паролей с использованием алгоритма Advanced Encryption Standard. Используется ключ длиной 256 бит.
3. FormatChecker – является сервисом для валидации форматов данных. Он содержит методы validateMail() и validatePhone(), проверяющие корректность ввода электронной почты и телефона соответственно.

Листинг кода сервисов представлен в приложении Б.

## 4.4 Реализация пользовательского интерфейса

Запуск приложения можно осуществить либо с помощью ярлыка на рабочем столе, либо введя URL в адресную строку браузера. Страница авторизации изображена на рисунке 4.4.

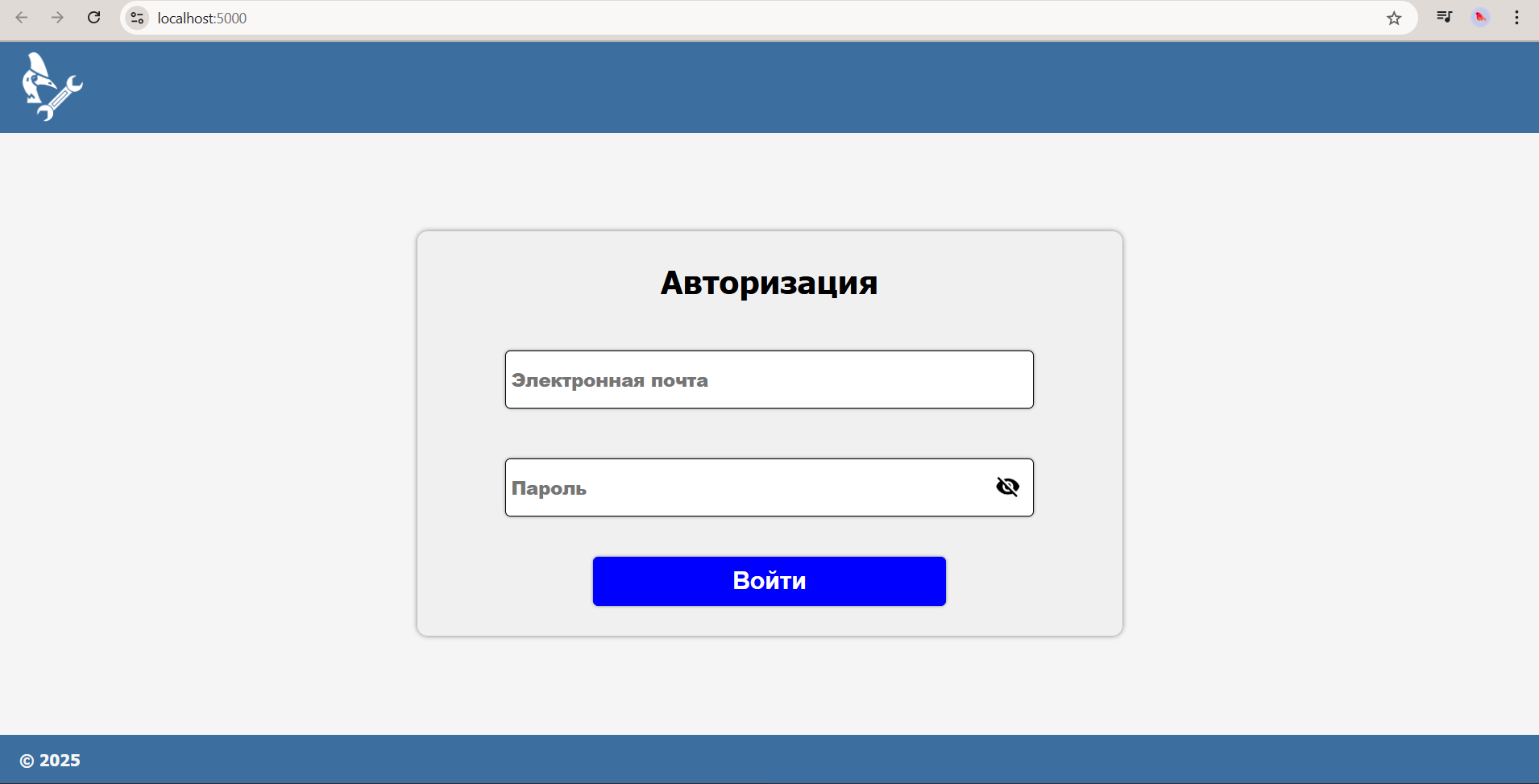


Рисунок 4.4 – Страница авторизации

Страница, на которую попадает инсталлятор после авторизации, представлена на рисунке 4.5.

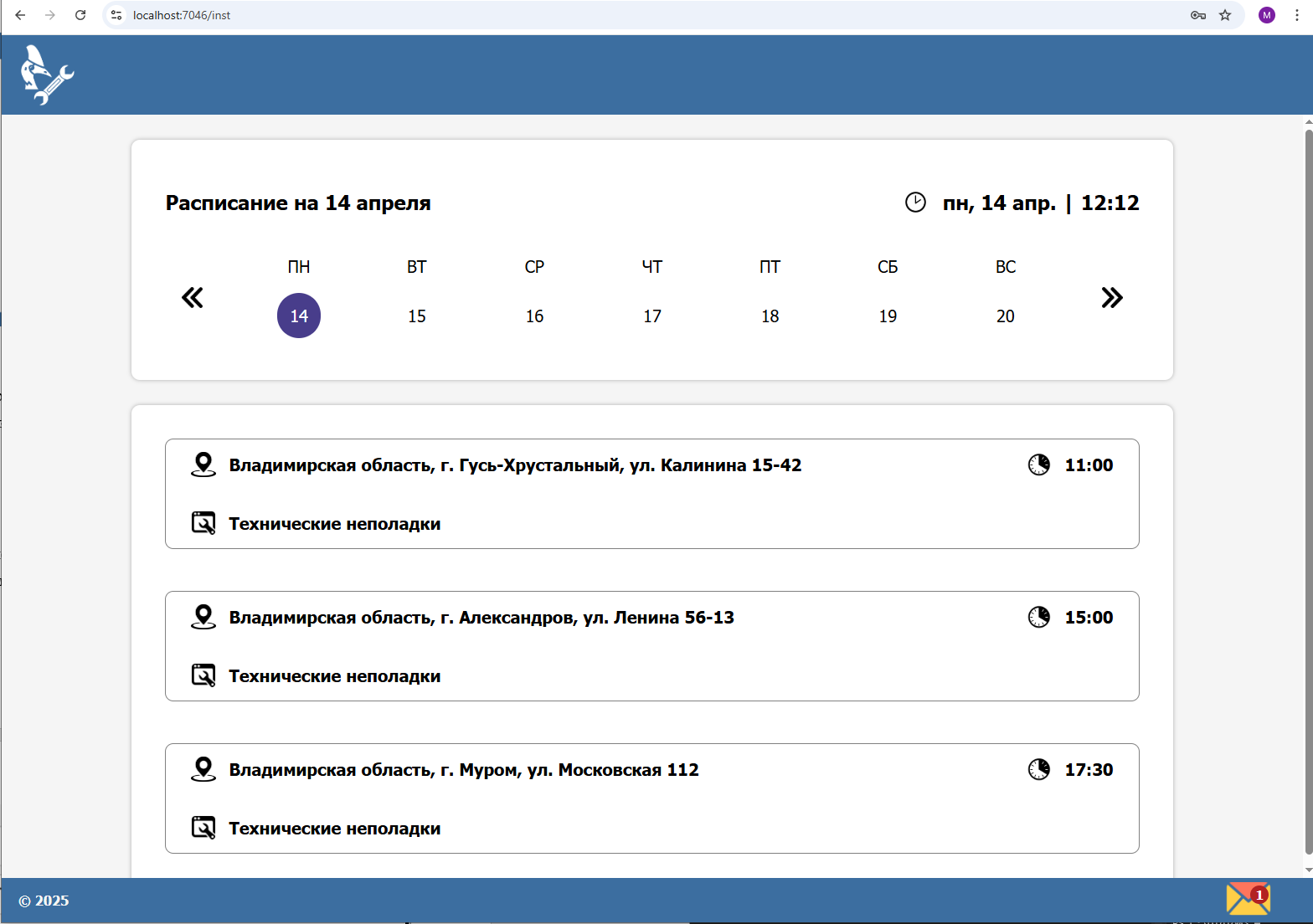


Рисунок 4.5 – Страница инсталлятора

Всплывающее окно с вызовом представлено на рисунке 4.6.

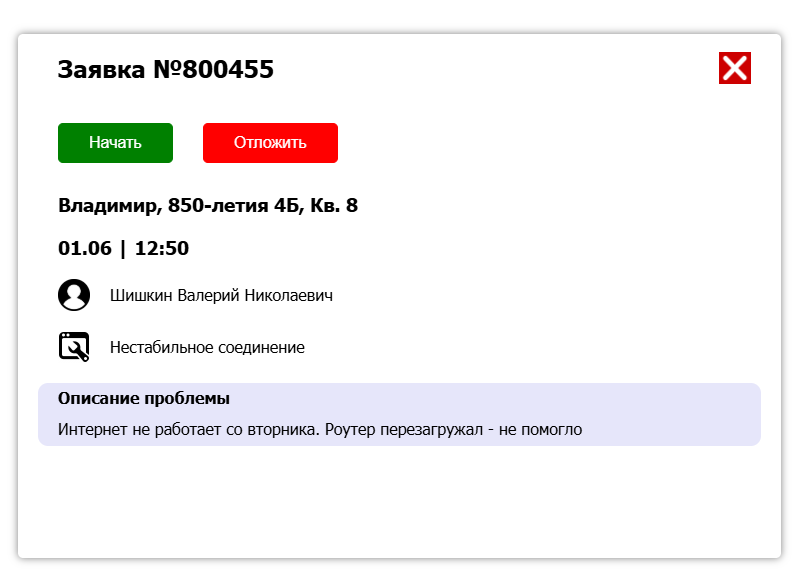


Рисунок 4.6 – Всплывающее окно вызова

Окно для переноса заявки на другое время представлено на рисунке 4.7.

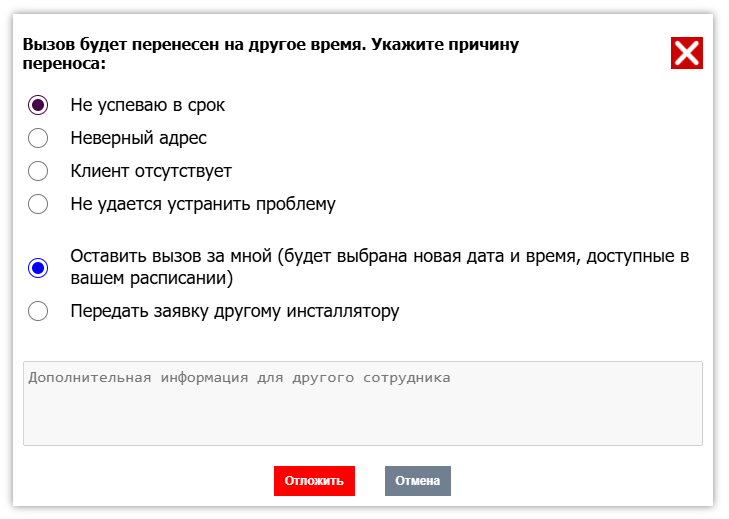


Рисунок 4.7 – Всплывающее окно для переноса заявки

Главная страница руководителя изображена на рисунке 4.8.

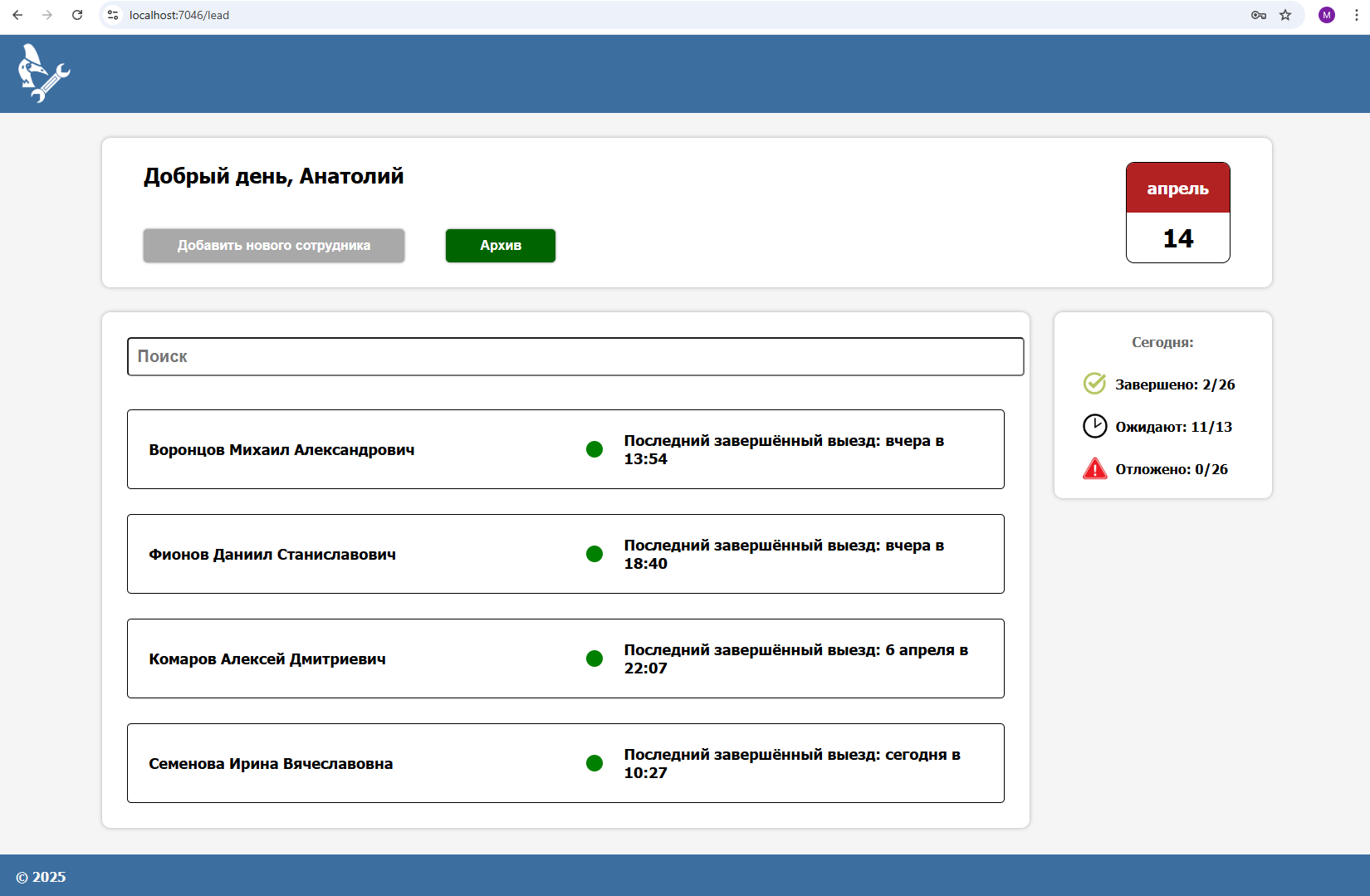


Рисунок 4.8 – Страница руководителя

Страница со списком сотрудников представлена на рисунке 4.9.

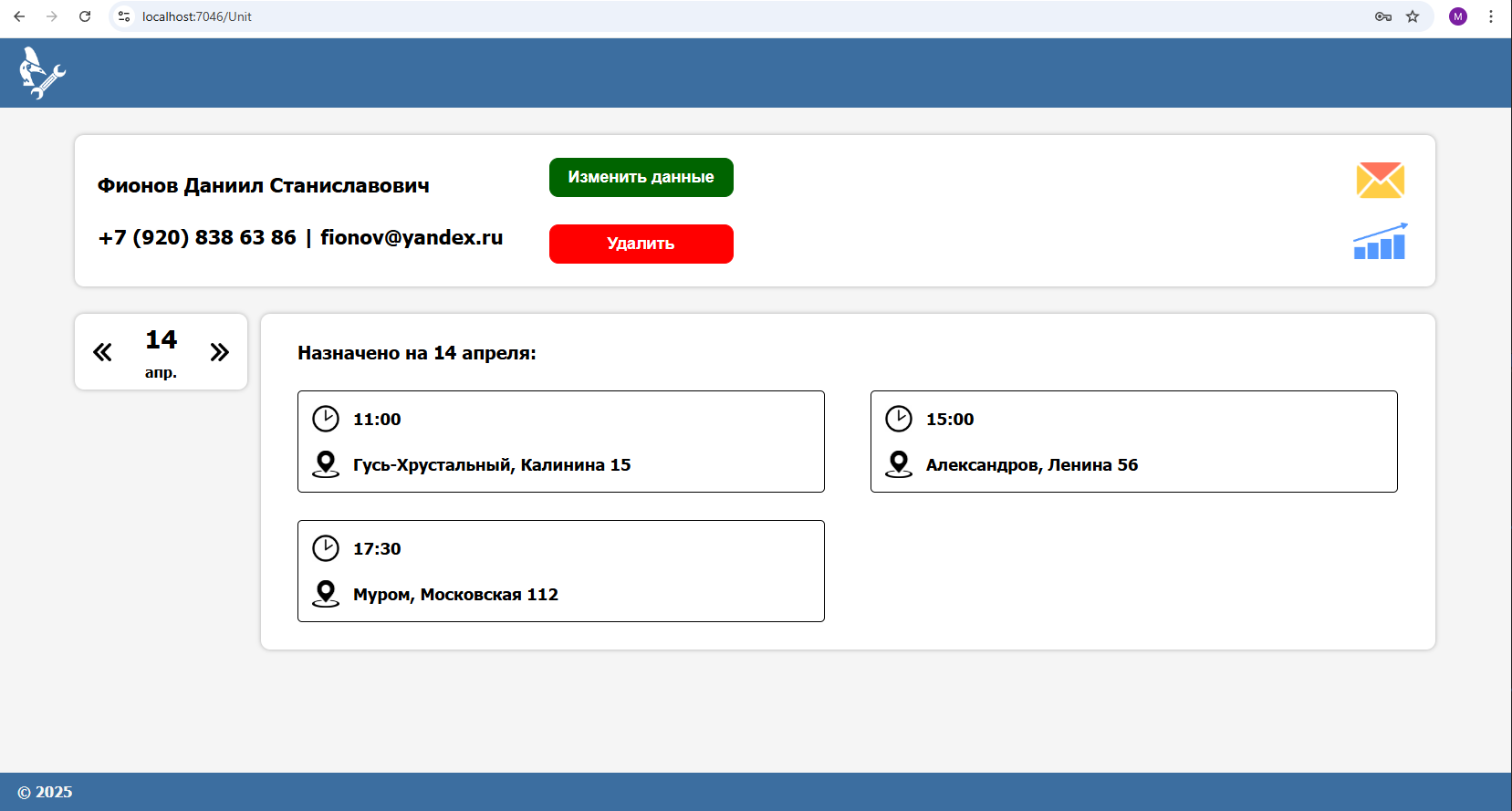


Рисунок 4.9 – Сотрудник

Всплывающее окно для добавления нового специалиста изображен на рисунке 4.10.

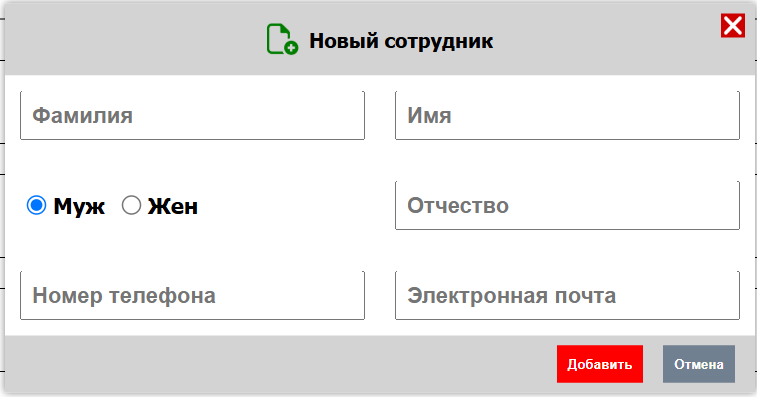


Рисунок 4.10 – Попап для добавления

Архив с удаленными специалистами представлен на рисунке 4.11.

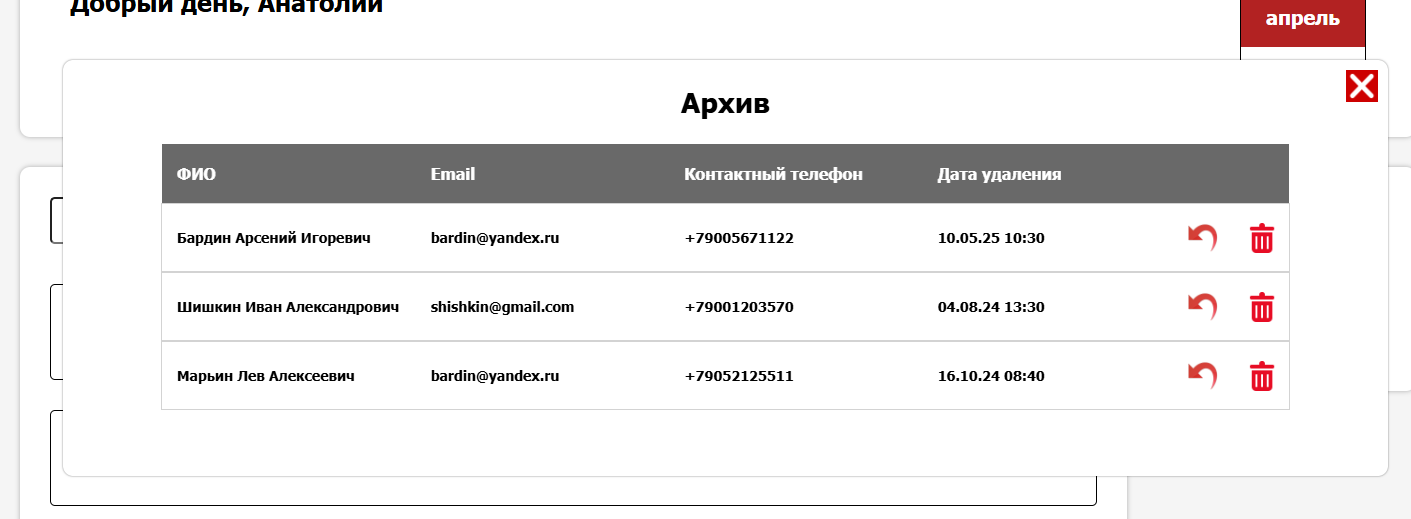


Рисунок 4.11 – Архив

Всплывающие окна для отправки и получения сообщений представлены на рисунках 4.12 – 4.13.

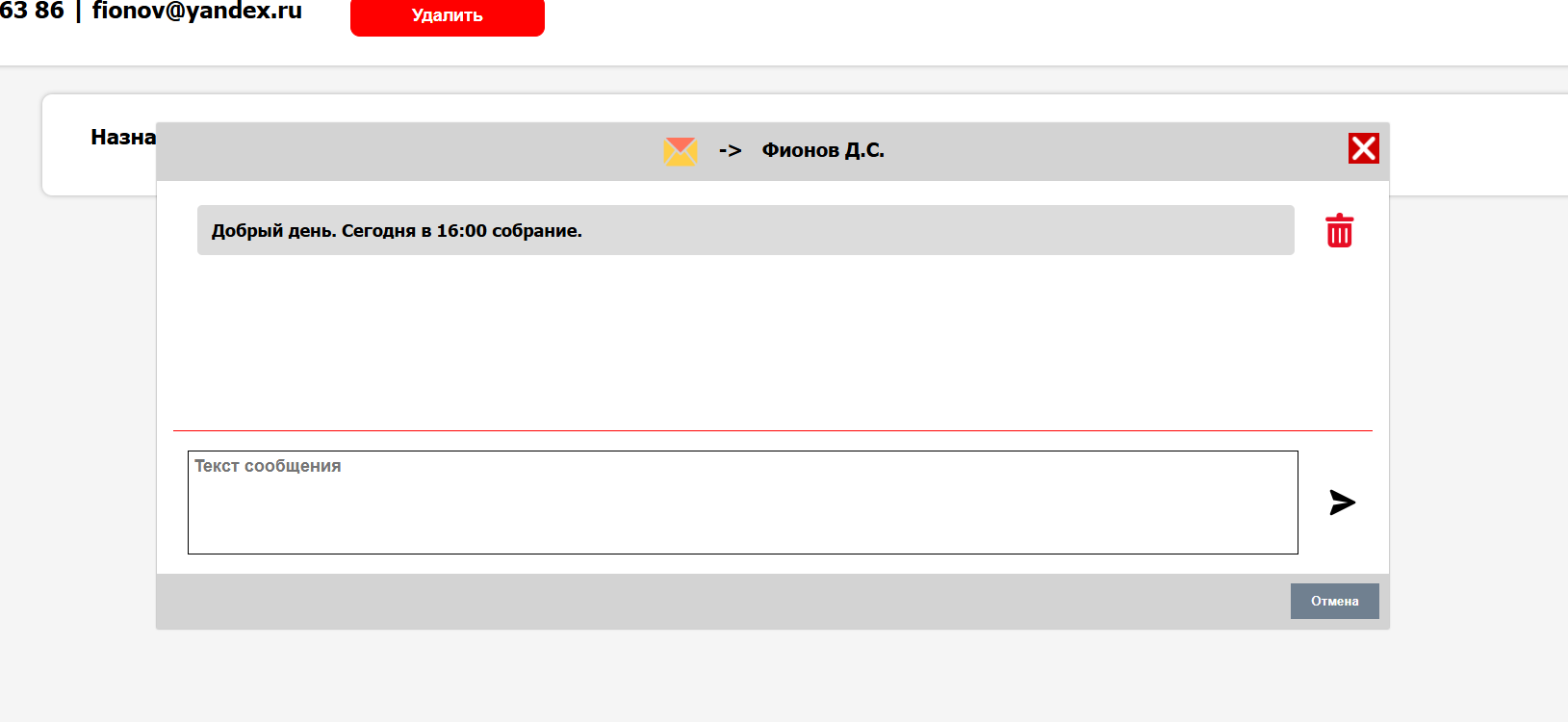


Рисунок 4.12 – Отправленные

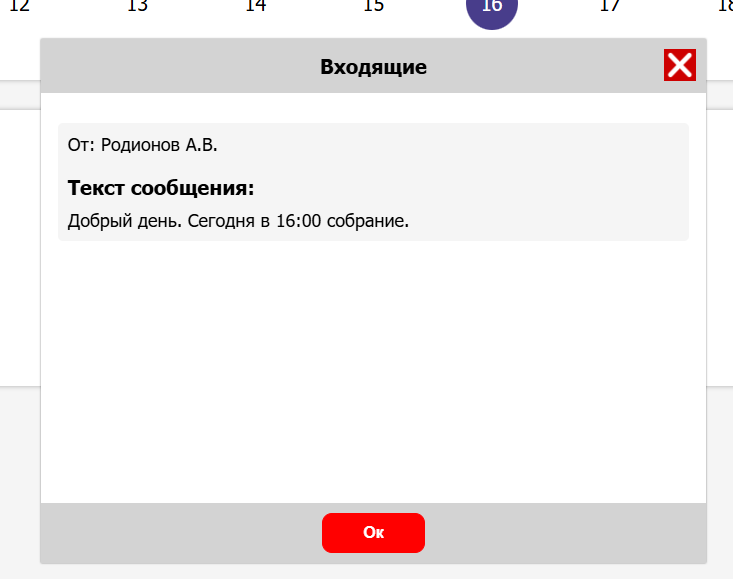


Рисунок 4.13 – Входящие

Окно со статистикой обработанных вызовов изображено на рисунке 4.14.

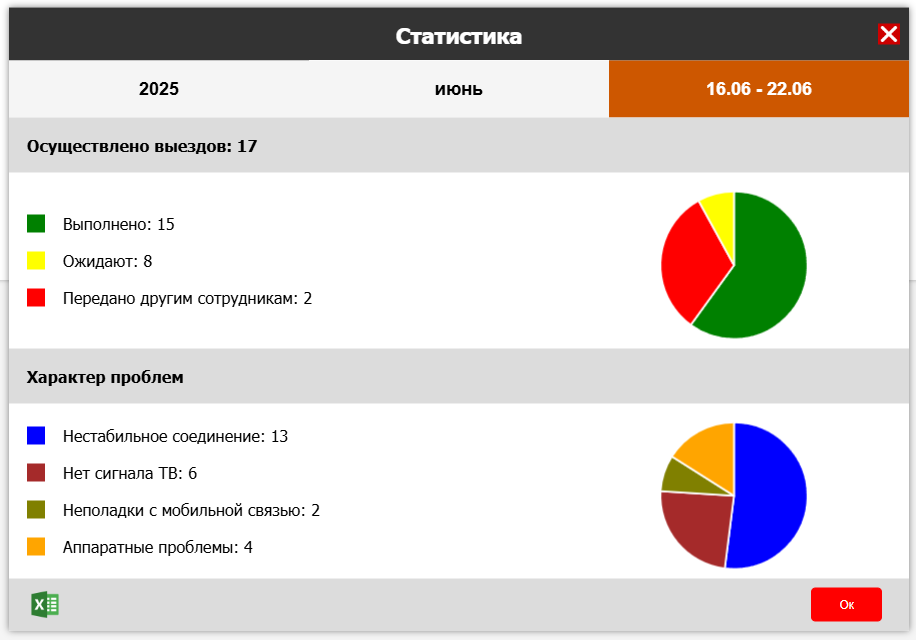


Рисунок 4.14 – Статистика

Всплывающее окно для импорта статистики в лист Excel представлено на рисунке 4.15.

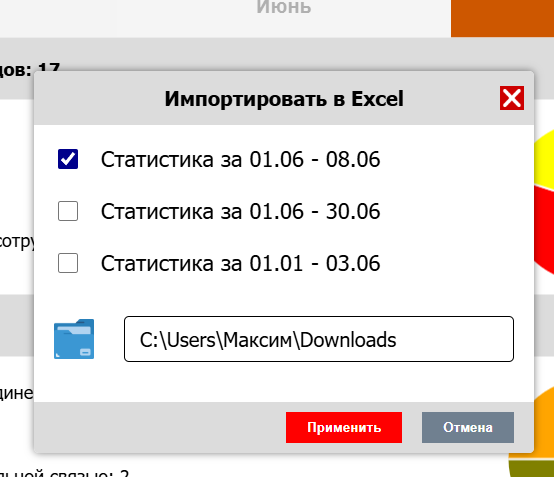


Рисунок 4.15 – Импорт статистики

На рисунке 4.16 представлена главная страница диспетчера.

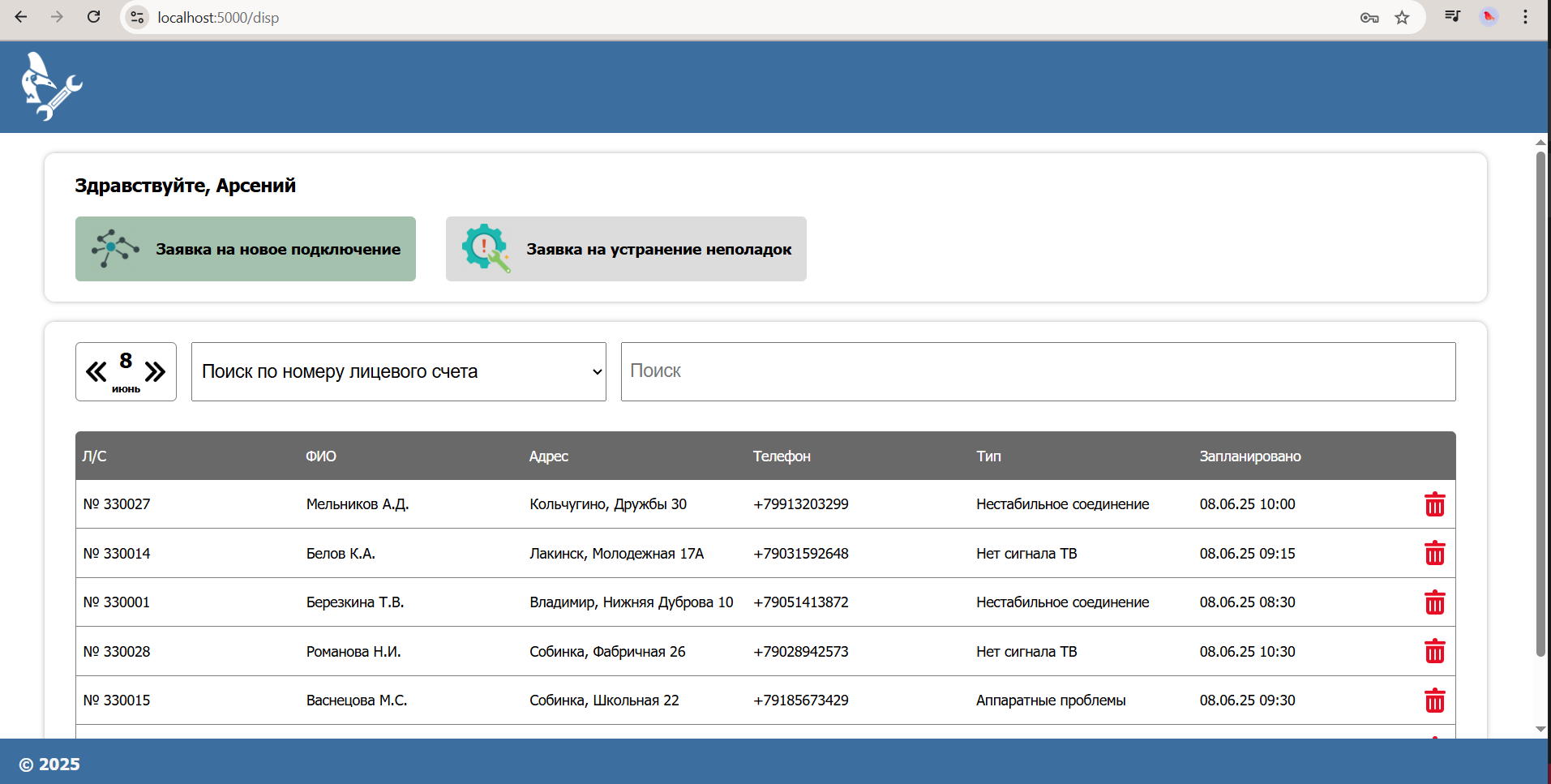


Рисунок 4.16 – Страница диспетчера

Страница для добавления новой заявки представлена на рисунке 4.17

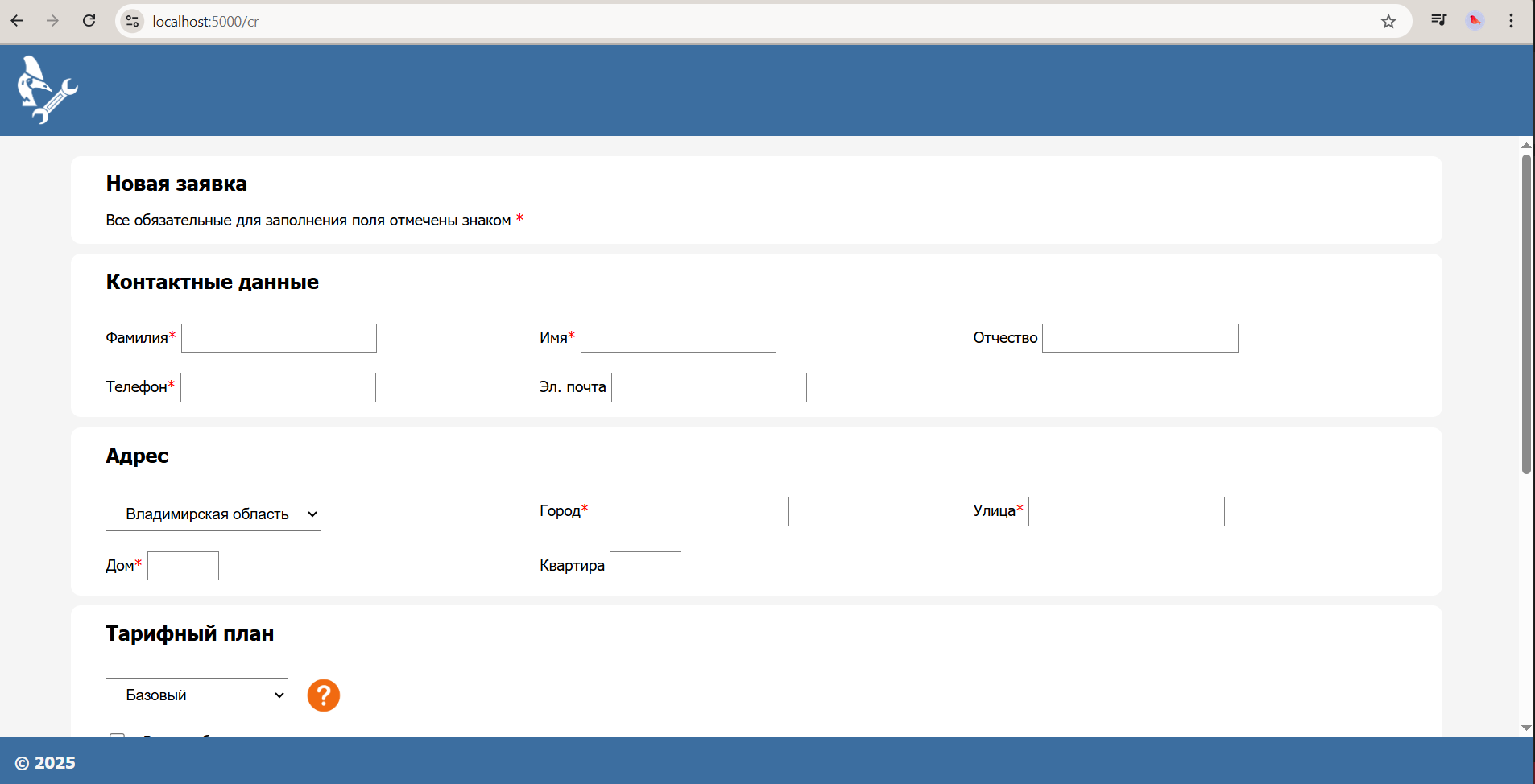


Рисунок 4.17 – Страница добавления заявки

В данном разделе был представлен спроектированный пользовательский интерфейс.

# 5 Тестирование приложения

5.1 Тест-требования

Тест-требование 1. Удаленный сотрудник должен быть перемещен в архив.

Проверка, что при удалении учетной записи сотрудника он корректно переносится в архив без потери данных.

Тест-требование 2. Список сотрудников (полный и отфильтрованный) должен быть отсортирован в алфавитном порядке.

Проверка, что сортировка по ФИО применяется ко всем вариантам отображения списка.

Тест-требование 3. Поле «ФИО» должно принимать только буквенные символы.

Проверка, что ввод цифр или некорректных символов недоступен

Тест-требование 4. Новая заявка должна автоматически назначаться свободному сотруднику.

Проверка, что система корректно распределяет заявки, учитывая текущую загрузку сотрудников.

Тест-требование 5 Добавление нового сотрудника возможно только при заполнении всех обязательных полей.

Проверка, что форма не отправляется, если хотя бы одно поле осталось пустым.

Тест-требование 6. Экспортированный файл в формате .xlsx должен содержать полную таблицу сотрудников.

Проверка, что экспорт включает все данные без искажений.

5.2 Тест-план

Тестовый пример 1.

Номер тест-требования: 1.

Описание теста: Данный тест проверяет корректность переноса удаленного сотрудника в архив.

Входные данные: логин – rodionov@gmail.com, пароль – Space940.

Ожидаемый результат: сотрудник исчезает из основного списка, появляется запись в архиве с сохранением всех данных.

Сценарий теста:

− Авторизация под учетной зписью руководителя.

− Выбор сотрудника.

− Нажатие «Удалить».

− Проверка архива на наличие удаленной записи.

Тестовый пример 2.

Номер тест-требования: 2.

Описание теста: Данный тест проверяет алфавитную сортировку списка сотрудников.

Входные данные: логин – rodionov@gmail.com, пароль – Space940, фильтр – "Б".

Ожидаемый результат: отсортированный список: «Борисов Николай Алексеевич», «Ярославцев Борис Михайлович», «Кушниренко Сергей Борисович».

Сценарий теста:

− Авторизация под учетной записью руководителя.

− Фильтрация с помощью поисковой строки.

− Проверка порядка отображения.

Тестовый пример 3.

Номер тест-требования: 3.

Описание теста: Данный тест проверяет валидацию ФИО абонента при создании заявки.

Входные данные: логин – gromov@yandex.ru, пароль – Wind810, ФИО – «Полянская Анна 123Владимировна».

Ожидаемый результат: сообщение об ошибке: «ФИО должно содержать только буквы».

Сценарий теста:

− Авторизация под учетной записью диспетчера.

− Нажатие на кнопку «Новая заявка».

− Попытка создания заявки с некорректным ФИО.

Тестовый пример 4.

Номер тест-требования: 4.

Описание теста: Данный тест проверяет автоматическое назначение заявки сотруднику.

Входные данные: логин – gromov@yandex.ru, пароль – Wind810, Новая заявка (приоритет: высокий).

Ожидаемый результат: заявка появляется в расписании одного из свободных сотрудников.

Сценарий теста:

− Авторизация под учетной записью диспетчера.

− Нажатие на кнопку «Новая заявка».

− Создание заявки.

Тестовый пример 5.

Номер тест-требования: 5.

Описание теста: Данный тест проверяет необходимость заполнения всех полей при добавлении сотрудника.

Входные данные: логин – rodionov@gmail.com, пароль – Space940, пустое поле «Электронная почта».

Ожидаемый результат: cообщение: «Заполните все обязательные поля».

Сценарий теста:

− Авторизация под учетной записью руководителя.

− Нажатие на кнопку «Добавить сотрудника».

− Попытка сохранения формы с пустым полем.

Тестовый пример 6.

Номер тест-требования: 6.

Описание теста: Данный тест проверяет экспорт данных в .xlsx.

Входные данные: логин – rodionov@gmail.com, пароль – Space940

Ожидаемый результат: файл содержит всех сотрудников.

Сценарий теста:

− Авторизация под учетной записью руководителя.

− Переход в раздел «Статистика».

− Экспорт данных.

5.3 Результаты тестирования

Тест 1. Проверка, что при удалении учетной записи сотрудника он корректно переносится в архив без потери данных.

Результат теста представлен на рисунках 4.1 – 4.2.

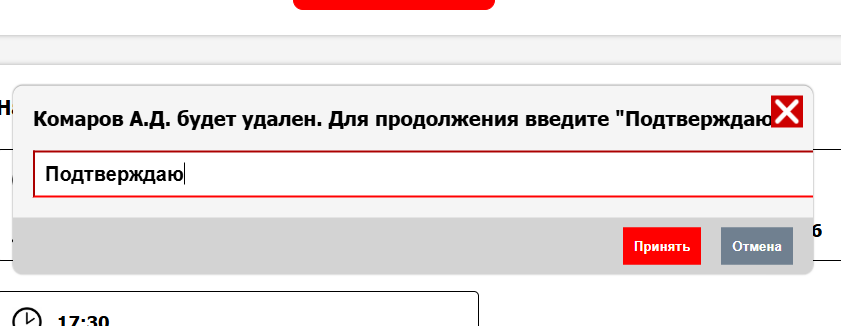


Рисунок 4.1 – Удаляемый сотрудник

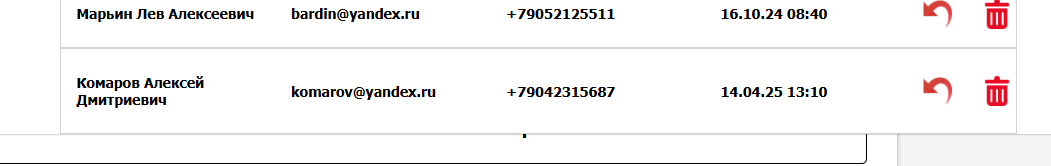


Рисунок 4.2 – Архив

Тест пройден успешно: удаленный сотрудник корректно перенесен в архив.

Тест 2. Проверка, что сортировка по ФИО применяется ко всем вариантам отображения списка.

Результат теста представлен на рисунке 4.3.

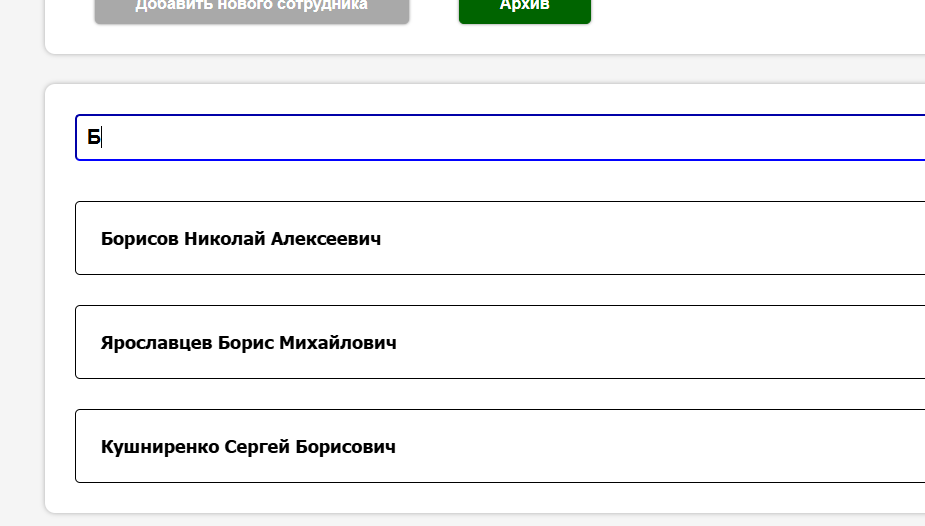


Рисунок 4.3 – Фильтрация сотрудников

Тест пройден успешно: список сотрудников корректно сортируется в алфавитном порядке.

Тест 3. Проверка, что ввод цифр или некорректных символов недоступен.

Результат теста представлен на рисунке 4.4.

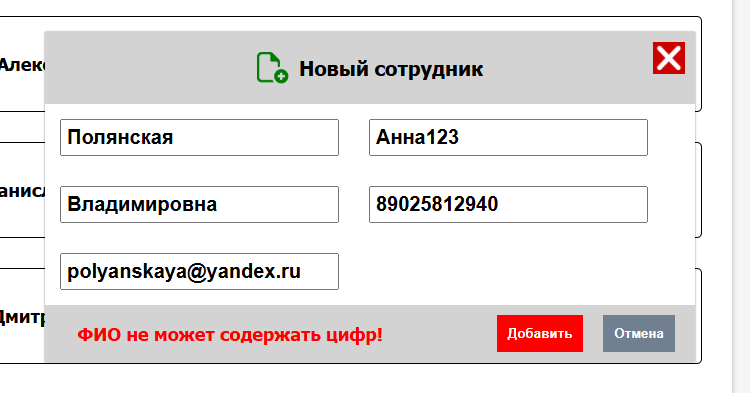


Рисунок 4.4 – Сообщение об ошибке

Тест пройден успешно: система блокирует добавление при наличии цифр в поле «ФИО сотрудника», выводя соответствующее сообщение об ошибке.

Тест 4. Проверка, что система корректно распределяет заявки, учитывая текущую загрузку сотрудников.

Результат теста представлен на рисунках 4.5 – 4.7.

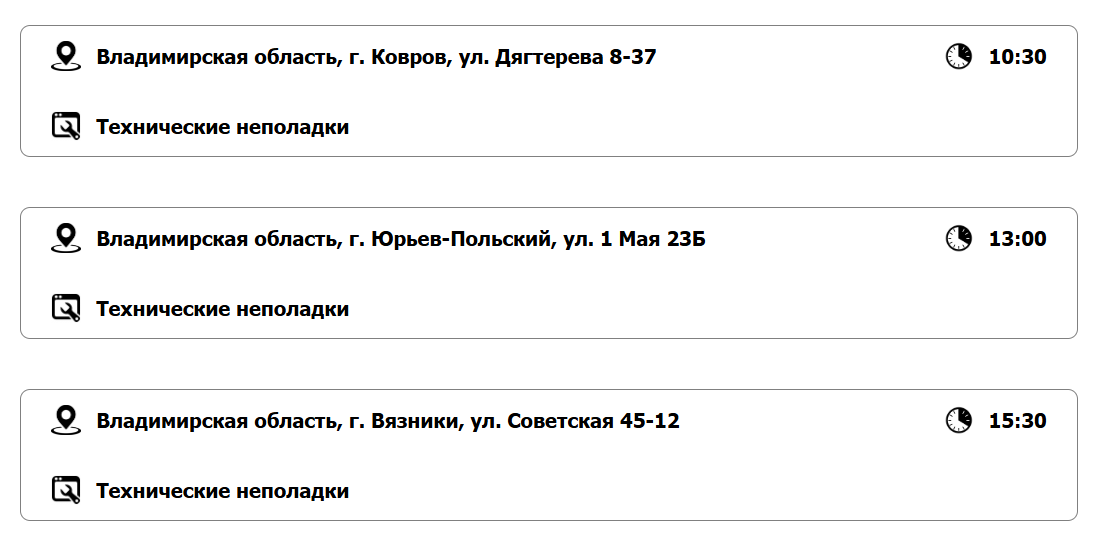


Рисунок 4.5 – Расписание (свободно после 16:00)

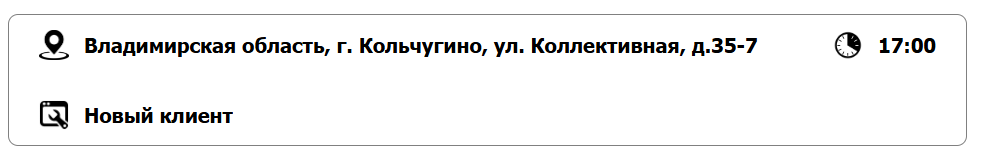


Рисунок 4.6 – Новая заявка (назначается на 17:00)

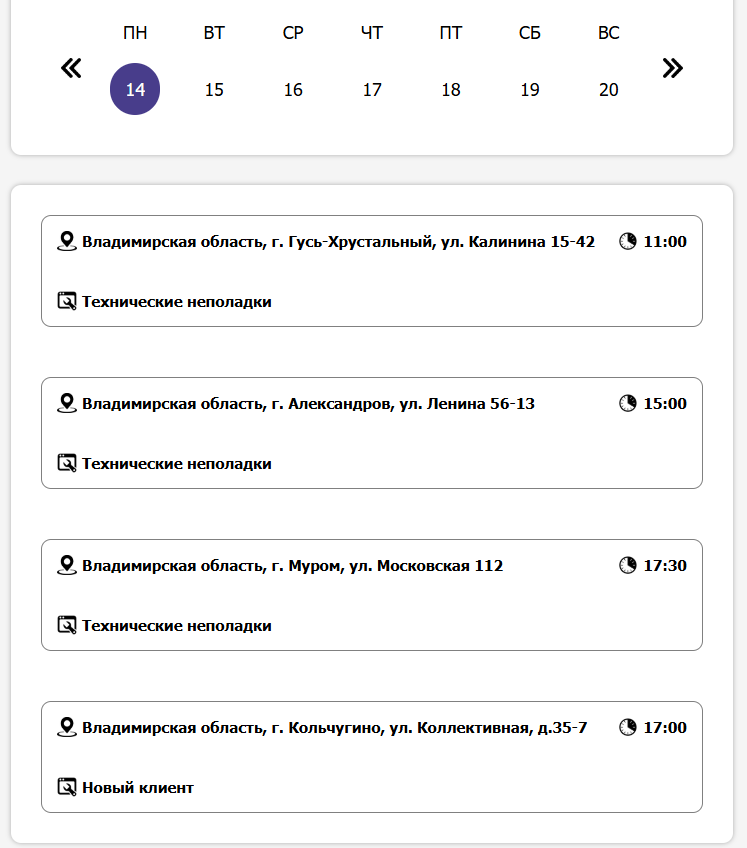


Рисунок 4.7 – Обновленное расписание

Тест пройден успешно: новая заявка автоматически назначается свободному сотруднику согласно алгоритму распределения.

Тест 5. Проверка, что форма не отправляется, если хотя бы одно поле осталось пустым.

Результат теста представлен на рисунке 4.8.

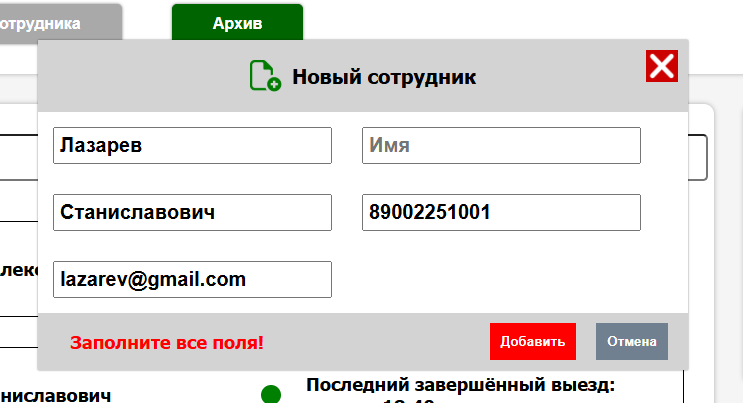


Рисунок 4.8 – Поле «Имя» оставлено пустым

Тест пройден успешно: система не позволяет сохранить нового сотрудника при наличии незаполненных обязательных полей, выводя сообщение об ошибке.

Тест 6. Проверка, что экспорт включает все данные без искажений.

Результат теста представлен на рисунке 4.9.

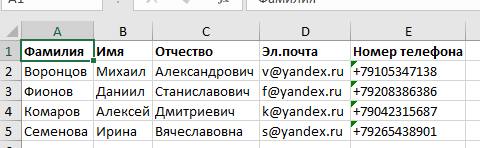


Рисунок 4.9 – Сгенерированный документ MS Excel

Тест пройден успешно: экспортированный файл .xlsx содержит полные и корректные данные о всех выездах за указанный период.

На основании проведенных тестов можно убедиться, что программа работает в соответствии с ожиданиями.

# Заключение

В результате выполнения дипломной работы был проведен анализ современных систем обработки обращений, разработана архитектура и реализовано веб-приложение с удобным интерфейсом. Сформированы тестовые требования, проведено тестирование, подтвердившее корректность работы системы. Результатом стало готовое к использованию решение для интернет-провайдера, отвечающее поставленным требованиям. Все поставленные задачи были реализованы.

В будущих обновлениях планируется добавить следующий функционал:

* Возможность восстановления пароля с помощью электронной почты.
* Интеграция с картографическими сервисами для построения оптимальных маршрутов.
* Уведомления через API для сотрудников о скорых предстоящих вызовах.

# Список использованных источников

1. Сэйнти К. Blazor в действии. – М.: ДМК Пресс, 2022. – 320 c. – ISBN: 978-5-93700-179-5.
2. Троелсен Э. Язык программирования C# 10 и платформа .NET 6. – СПб.:Питер, 2022. – 1088 c. – ISBN: 978-5-4461-1849-6.
3. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# (4-е изд.). – СПб.: Питер, 2021. –896 с. – ISBN: 978-5-4461-1102-2.
4. Фримен А. Pro ASP.NET Core MVC 2. – Apress, 2017. –992 c. – ISBN: 978-5-9908910-4-3.
5. Дакетт Дж. HTML и CSS. Разработка и дизайн веб-сайтов. – М.: Эксмо, 2022. – 480 c. – ISBN: 978-5-04-105805-6.
6. Эспозито Д. Blazor. Разработка веб-приложений на C#. – СПб.: Питер, 2023. – 352 c. – ISBN: 978-5-4461-2034-5.
7. Скит Дж. C# для профессионалов. Тонкости программирования. – М.: Вильямс, 2021. – 608 c. – ISBN: 978-5-9909445-8-3.

Интернет-ресурсы:

1. Диаграмма классов [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма\_классов (дата обращения: 17.05.2025).
2. Использование диаграммы вариантов использования UML при проектировании программного обеспечения [Электронный ресурс]. URL: https://habr.com/ru/articles/566218/ (дата обращения: 17.05.2025).
3. Диаграмма активностей (Activity diagram) [Электронный ресурс]. URL: https://flexberry.github.io/ru/fd\_activity-diagram.html (дата обращения: 17.05.2025).
4. Простое руководство к диаграммам развертывания UML [Электронный ресурс]. URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-развёрт/ (дата обращения: 18.05.2025).
5. Диаграмма компонентов [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма\_компонентов (дата обращения: 18.05.2025).

# Приложение А

Листинг класса Abonent:

public class Abonent

{

public int AbonentID { get; set; }

public int TariffID { get; set; }

public int PA { get; set; }

public string FirstName { get; set; }

public string MiddleName { get; set; }

public string LastName { get; set; }

public string Region { get; set; }

public string City { get; set; }

public string Street { get; set; }

public string House { get; set; }

public int? Flat { get; set; }

public string PhoneNumber { get; set; }

public Tariff Tariff { get; set; }

}

Листинг класса Account:

public class Account

{

public int AccountID { get; set; }

public int? DispatcherID { get; set; }

public int? InstallatorID { get; set; }

public int? LeadID { get; set; }

public string Password { get; set; }

}

Листинг класса Dispatcher:

public class Dispatcher

{

public int DispatcherID { get; set; }

public string FirstName { get; set; }

public string MiddleName { get; set; }

public string LastName { get; set; }

public string Gender { get; set; }

public string PhoneNumber { get; set; }

public string Email { get; set; }

public Account Account { get; set; }

}

Листинг класса Installator:

public class Installator

{

public int InstallatorID { get; set; }

public int LeadID { get; set; }

public string FirstName { get; set; }

public string MiddleName { get; set; }

public string LastName { get; set; }

public string Gender { get; set; }

public string PhoneNumber { get; set; }

public string Email { get; set; }

public bool isArchived { get; set; }

public bool isActive { get; set; }

public DateOnly? RemovalDate { get; set; }

public Account Account { get; set; }

public List<Report> Reports { get; set; }

public List<Request> Requests { get; set; }

public List<Message> Messages { get; set; }

}

Листинг класса Lead:

public class Lead

{

public int LeadID { get; set; }

public string FirstName { get; set; }

public string MiddleName { get; set; }

public string LastName { get; set; }

public string Gender { get; set; }

public string PhoneNumber { get; set; }

public string Email { get; set; }

public string Region { get; set; }

public Account Account { get; set; }

public List<Installator> Installators { get; set; }

public List<Message> Messages { get; set; }

}

Листинг класса Message:

public class Message

{

public int MessageID { get; set; }

public int LeadID { get; set; }

public int InstallatorID { get; set; }

public string Text { get; set; }

public bool isRead { get; set; }

public DateTime SendingTime { get; set; }

}

Листинг класса Report:

public class Report

{

public int ReportID { get; set; }

public int RequestID { get; set; }

public string Status { get; set; }

public string Commentary { get; set; }

public Request Request { get; set; }

}

Листинг класса Request:

public class Request

{

public int RequestID { get; set; }

public int InstallatorID { get; set; }

public int AbonentID { get; set; }

public string Type { get; set; }

public string? Description { get; set; }

public bool isDelayed { get; set; }

public DateTime Created { get; set; }

public DateTime Scheduled { get; set; }

public DateTime? Closed { get; set; }

}

Листинг класса Service:

public class Service

{

public int ServiceID { get; set; }

public string ServiceName { get; set; }

public int AdditionalPrice { get; set; }

}

Листинг класса Tariff:

public class Tariff

{

public int TariffID { get; set; }

public string TariffName { get; set; }

public int TariffPrice { get; set; }

public List<Service> Services { get; set; }

}

Листинг класса TariffService:

public class TariffService

{

public int TariffServiceID { get; set; }

public int TariffID { get; set; }

public int ServiceID { get; set; }

public Tariff Tariff { get; set; }

public Service { get; set; }

}

# Приложение Б

Листинг кода сервиса CryptoService:

public class CryptoService

{

private static readonly byte[] Key = Encoding.UTF8.GetBytes("1234567890ABCDEFGHIJKLMNOPQRSTUV");

private static readonly byte[] IV = Encoding.UTF8.GetBytes("1234567890ABCDEF");

public static string Encrypt(string plainText)

{

using (Aes aesAlg = Aes.Create())

{

aesAlg.Key = Key;

aesAlg.IV = IV;

ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

using (var msEncrypt = new System.IO.MemoryStream())

{

using (var csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))

{

using (var swEncrypt = new System.IO.StreamWriter(csEncrypt))

{

swEncrypt.Write(plainText);

}

return Convert.ToBase64String(msEncrypt.ToArray());

}

}

}

}

public static string Decrypt(string cipherText)

{

using (Aes aesAlg = Aes.Create())

{

aesAlg.Key = Key;

aesAlg.IV = IV;

ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

using (var msDecrypt = new System.IO.MemoryStream(Convert.FromBase64String(cipherText)))

{

using (var csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))

{

using (var srDecrypt = new System.IO.StreamReader(csDecrypt))

{

return srDecrypt.ReadToEnd();

}

}

}

}

}

}

Листинг кода сервиса FormatChecker

public static class FormatChecker

{

public static bool validateMail(string email)

{

if (string.IsNullOrEmpty(email))

{

return true;

}

var regex = new Regex(@"^[^@\s]+@[^@\s]+\.[^@\s]+$");

bool isValid = regex.IsMatch(email);

return isValid;

}

public static bool validatePhone(string phone)

{

var phonePattern = @"^(?:\+7|8)\d{10}$";

bool isValid = System.Text.RegularExpressions.Regex.IsMatch(phone, phonePattern);

return isValid;

}

}

Листинг кода сервиса ChartService

public static class ChartService

{

public static List<Request> userRequests = new List<Request>();

public static List<Report> userReports = new List<Report>();

public static PieConfig InstData = new PieConfig();

public static PieConfig GeneralData = new PieConfig();

public static PieDataset<int> instset;

public static PieDataset<int> genset;

public static void DrawTopChart()

{

if (InstData.Data.Datasets.Contains(instset))

{

InstData.Data.Datasets.Remove(instset);

}

instset = new PieDataset<int>(new[] {

userReports.Where(r => r.Status == "Завершен").Count(),

userRequests.Where(u => u.Closed == null).Count(),

userReports.Where(r => r.Status == "Передан").Count()

})

{

BackgroundColor = new[]

{

ColorUtil.ColorHexString(0, 128, 0),

ColorUtil.ColorHexString(255, 0, 0),

ColorUtil.ColorHexString(255, 255, 0),

}

};

InstData.Data.Datasets.Add(instset);

}

public static void DrawBottomChart()

{

if (GeneralData.Data.Datasets.Contains(genset))

{

GeneralData.Data.Datasets.Remove(genset);

}

genset = new PieDataset<int>(new[] {

userRequests.Where(u => u.Type == "Нестабильное соединение").Count(),

userRequests.Where(u => u.Type == "Нет сигнала ТВ").Count(),

userRequests.Where(u => u.Type == "Неполадки с мобильной связью").Count(),

userRequests.Where(u => u.Type == "Аппаратные проблемы").Count()

})

{

BackgroundColor = new[]

{

ColorUtil.ColorHexString(0, 0, 255),

ColorUtil.ColorHexString(165, 42, 42),

ColorUtil.ColorHexString(128, 128, 0),

ColorUtil.ColorHexString(255, 165, 0),

}

};

GeneralData.Data.Datasets.Add(genset);

}

}