СОДЕРЖАНИЕ

ВВЕДЕНИЕ…………………………………………………………………………3

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПРОЕКТИРОВАНИЕ……………...5

* 1. Постановка задачи на проектирование и разработку…………………..5
  2. Методы сбора требований к программному продукту…………………10
  3. Формирование функциональных и нефункциональных требований….15
  4. Моделирование программной системы на основе собранных данных..19

2. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА «ADMIN» ДЛЯ МКУ «СКО» Г. АРЗАМАСА……………………………………………………………………...24

1. Технологии разработки…………………………………………………...24
2. Архитектура программного продукта…………………………………...28
3. Пользовательский интерфейс…………………………………………….32
4. Программная логика……………………....…………………...…............42

ЗАКЛЮЧЕНИЕ………………………………………………………………...46

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ………………………………….48

ВВЕДЕНИЕ

С развитием информационных технологий и увеличением количества используемого оборудования в организациях возрастает необходимость в эффективной поддержке и управлении этими системами. В муниципальных казенных учреждениях, таких как МКУ «СКО» г. Арзамаса, системные администраторы сталкиваются с многочисленными задачами, связанными с поддержкой сетевой инфраструктуры, оборудования и программного обеспечения. Часто эти задачи включают устранение неполадок, обеспечение безопасности данных и поддержание стабильной работы всех систем.

Однако, с увеличением количества оборудования и сложностью инфраструктуры, ручное управление и устранение неполадок становится все более трудоемким и требует значительных ресурсов. В связи с этим возникает необходимость в разработке специализированного программного продукта, который автоматизирует процессы диагностики и устранения неполадок, тем самым облегчая работу системного администратора и повышая общую эффективность работы учреждения.

Целью данной дипломной работы является разработка программного продукта, ориентированного на помощь системному администратору в диагностике и устранении различных неполадок в здании МКУ «СКО» г. Арзамаса.

Задачи дипломной работы:

1. Провести анализ предметной области.
2. Определить требования к программному продукту.
3. Сформировать функциональные и нефункциональные требования.
4. Произвести моделирование на основе собранных данных.
5. Определить технологии разработки программного продукта.
6. Разработать архитектуру программного продукта.
7. Разработать пользовательский интерфейс программного продукта.
8. Реализовать программный продукт на основе разработанной архитектуры и интерфейса.

Объект исследования: МКУ «СКО» г. Арзамаса.

Предмет исследования: разработка программного продукта для автоматизации диагностики компьютерных рабочих мест и устранения неполадок в работе оборудования.

В ходе работы использованы научные труды отечественных и зарубежных ученых в области информационных систем, сетевых технологий и системного администрирования, нормативные акты, специализированная литература и материалы, предоставленные МКУ «СКО» г. Арзамаса.

Для достижения целей исследования использованы методы системного анализа, моделирования, программирования и тестирования программного обеспечения. Системный анализ позволил провести детальное изучение предметной области и определить основные требования к программному продукту. Моделирование использовалось для создания моделей программной системы, что облегчило процесс ее разработки. Программирование осуществлялось с использованием современных технологий и инструментов разработки. Тестирование проводилось с целью проверки работоспособности и эффективности программного продукта в реальных условиях.

Разработанный программный продукт позволит системному администратору МКУ «СКО» г. Арзамаса автоматизировать процессы диагностики и устранения неполадок, повысить точность данных и упростить работу, что, в свою очередь, повысит общую эффективность работы учреждения.

Дипломная работа состоит из введения, двух глав, заключения, списка используемых источников, которые рассматривают этапы анализа, проектирования и разработки программного продукта, содержит 53 листа пояснительного текста, 17 рисунков, 25 источников литературы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПРОЕКТИРОВАНИЕ
   1. Постановка задачи на проектирование и разработку

Современные муниципальные учреждения, такие как МКУ «СКО» г. Арзамаса, сталкиваются с необходимостью поддержания стабильной и безопасной работы своей ИТ-инфраструктуры. Системные администраторы играют ключевую роль в обеспечении бесперебойной работы всех технических средств и устранении возникающих неполадок. Однако с увеличением количества используемого оборудования и сложностью инфраструктуры, выполнение этих задач вручную становится все более трудоемким и неэффективным.

Приведем обоснование необходимости автоматизации.

ИТ-инфраструктура муниципального учреждения включает в себя разнообразное оборудование и программное обеспечение: серверы, рабочие станции, сетевые устройства, системы хранения данных, операционные системы и прикладные программы. Управление таким большим и разнообразным парком оборудования требует не только знаний и навыков, но и значительных временных затрат. Системные администраторы должны отслеживать состояние оборудования, оперативно реагировать на возникающие проблемы, проводить профилактические работы, обеспечивать безопасность данных и следить за обновлениями программного обеспечения.

Основные проблемы, с которыми сталкиваются системные администраторы в МКУ «СКО» г. Арзамаса, включают:

1. Большое количество оборудования: чем больше устройств нужно поддерживать, тем сложнее отслеживать их состояние и своевременно реагировать на возникающие проблемы.
2. Сложность инфраструктуры: сетевые устройства, серверы, системы хранения данных и рабочие станции должны работать слаженно и без сбоев, что требует сложной координации и управления.
3. Недостаток автоматизации: в ручном режиме системные администраторы не всегда могут оперативно выявлять и устранять неполадки, что может приводить к простою оборудования и снижению эффективности работы учреждения.
4. Безопасность данных: обеспечение защиты данных от несанкционированного доступа, вирусов и других угроз требует постоянного мониторинга и обновления систем безопасности.

Рассмотрим более подробно цели и задачи проекта.

Целью данного проекта является создание программного продукта, который автоматизирует процессы диагностики и устранения неполадок, облегчая работу системного администратора и повышая общую эффективность работы МКУ «СКО» г. Арзамаса. Для достижения этой цели необходимо решить следующие задачи:

1. Провести анализ текущего состояния системы управления ИТ-инфраструктурой в МКУ «СКО» г. Арзамаса.
2. Определить основные требования к новому программному продукту на основе анализа потребностей пользователей и существующих проблем.
3. Разработать архитектуру и интерфейс программного продукта, обеспечивающие удобство использования и соответствие требованиям пользователей.
4. Реализовать программный продукт на основе разработанной архитектуры и интерфейса.
5. Провести тестирование программного продукта в условиях реальной эксплуатации и оценить его эффективность.

Анализ текущего состояния системы управления ИТ-инфраструктурой.

Для разработки эффективного программного продукта необходимо провести детальный анализ текущего состояния системы управления ИТ-инфраструктурой в МКУ «СКО» г. Арзамаса. Этот анализ включает в себя.

1. Инвентаризация оборудования: составление списка всех устройств, используемых в учреждении, их характеристик, состояния и текущих проблем. Это позволяет понять объем работы и выявить наиболее проблемные участки.
2. Оценка существующих процессов управления: анализ текущих методов и инструментов, используемых для мониторинга и управления ИТ-инфраструктурой. Это включает изучение используемого программного обеспечения, протоколов и процедур работы системных администраторов.
3. Выявление основных проблем и недостатков: определение проблем, с которыми сталкиваются системные администраторы, и недостатков существующих методов управления. Это может включать частые сбои оборудования, низкую скорость реакции на проблемы, недостаток автоматизации и другие факторы, влияющие на эффективность работы[12, c.43].

Выполним определение требований к программному продукту.

На основе анализа текущего состояния и выявленных проблем определяются требования к новому программному продукту. Эти требования делятся на функциональные и нефункциональные.

Функциональные требования включают в себя следующие положения.

1. Мониторинг состояния оборудования: возможность отслеживать состояние всех устройств в реальном времени, получать уведомления о возникновении проблем и просматривать историю событий.
2. Диагностика неполадок: автоматическое обнаружение проблем и предоставление рекомендаций по их устранению. Это может включать анализ логов, тестирование соединений, проверку конфигураций и других параметров.
3. Управление устройствами: возможность удаленного управления устройствами, включая перезагрузку, обновление программного обеспечения и выполнение других операций.
4. Генерация отчетов и аналитика: создание отчетов о состоянии ИТ-инфраструктуры, анализ производительности и выявление тенденций и аномалий.

Нефункциональные требования включают:

1. Производительность: система должна обеспечивать быструю обработку данных и оперативное реагирование на запросы пользователей.
2. Надежность и безопасность: программный продукт должен быть устойчив к сбоям и обеспечивать защиту данных от несанкционированного доступа.
3. Масштабируемость: возможность адаптации системы к изменяющимся потребностям учреждения, включая добавление новых устройств и увеличение объема обрабатываемых данных.
4. Удобство использования: интуитивно понятный интерфейс, который позволяет пользователям легко выполнять необходимые операции без дополнительного обучения.

Рассмотрим процедуры проектирования и разработки.

Процесс разработки программного продукта включает несколько ключевых этапов, каждый из которых требует тщательного планирования и выполнения.

Этапы проектирования:

1. Сбор и анализ требований: включает детальное изучение потребностей пользователей, выявление функциональных и нефункциональных требований, проведение интервью и анкетирования сотрудников, анализ существующих систем и процессов.
2. Проектирование архитектуры системы: определение структуры системы, ее компонентов, их взаимодействия и используемых технологий. На этом этапе разрабатываются диаграммы классов, деятельности и последовательностей, описывающие функциональность и логику работы системы.
3. Проектирование интерфейса пользователя: разработка макетов и прототипов пользовательского интерфейса, обеспечение его удобства и интуитивности, проведение тестирования с участием конечных пользователей для получения обратной связи.

Этапы разработки:

1. Реализация серверной части: написание кода для обработки бизнес-логики, взаимодействия с базой данных и обеспечения API для клиентской части. Использование языка программирования C# и фреймворка WPF.
2. Реализация клиентской части: создание интерактивного и динамичного пользовательского интерфейса с использованием WPF и XAML. Обеспечение взаимодействия клиентской части с серверной через API.
3. Интеграция с базой данных: настройка базы данных Microsoft SQL Server для надежного и безопасного хранения данных, разработка схемы базы данных и реализация взаимодействия с серверной частью через ADO.NET или Entity Framework.

Тестирование и валидация включают в себя следующие виды.

1. Функциональное тестирование: проверка работоспособности всех функций и их соответствие требованиям. Проведение юнит-тестирования, интеграционного тестирования и системного тестирования.
2. Нагрузочное тестирование: оценка производительности системы под высокой нагрузкой, выявление и устранение узких мест.
3. Тестирование безопасности: проверка защиты данных и устойчивости системы к атакам. Проведение тестирования на проникновение и оценка механизмов аутентификации и авторизации.
4. Бета-тестирование: испытание системы в условиях реальной эксплуатации с участием конечных пользователей. Сбор обратной связи, выявление и исправление ошибок.

Рассмотрим основные вопросы внедрения и поддержки системы.

После завершения разработки и тестирования система готова к внедрению. Этот этап включает:

1. Установка программного продукта: настройка и конфигурация системы на серверах и рабочих станциях учреждения.
2. Обучение пользователей: проведение тренингов и семинаров для системных администраторов и других сотрудников, демонстрация возможностей системы и обучение работе с ней.
3. Поддержка и обслуживание: обеспечение технической поддержки, регулярные обновления и улучшения системы, оперативное решение возникающих проблем.

Ожидаемые результаты и преимущества. Внедрение автоматизированной системы диагностики и устранения неполадок в МКУ «СКО» г. Арзамаса позволит:

1. Сократить время реакции на проблемы: автоматическое обнаружение и уведомление о проблемах позволяет оперативно реагировать на неполадки и минимизировать простой оборудования.
2. Повысить точность диагностики: автоматические алгоритмы диагностики обеспечивают более точное выявление причин проблем и предоставление рекомендаций по их устранению.
3. Снизить трудозатраты системных администраторов: автоматизация рутинных задач освобождает время для более сложных и критически важных задач.
4. Обеспечить надежность и безопасность данных: интеграция с MSSQL и использование современных технологий безопасности обеспечивает защиту данных и устойчивость системы к сбоям[12, c.141].

Таким образом, разработка и внедрение программного продукта для автоматизации процессов диагностики и устранения неполадок в МКУ «СКО» г. Арзамаса позволит существенно повысить эффективность работы учреждения и обеспечить стабильную и безопасную работу его ИТ-инфраструктуры.

1.2 Методы сбора требований к программному продукту

Для успешной реализации проекта по автоматизации процессов диагностики и устранения неполадок в ИТ-инфраструктуре МКУ «СКО» г. Арзамаса необходимо собрать и проанализировать требования к программному продукту. Сбор требований является одним из ключевых этапов разработки, так как именно на этом этапе определяется, каким должен быть программный продукт, чтобы удовлетворять потребности пользователей и решать поставленные задачи.

Рассмотрим аспкет важности сбора требований.

Сбор требований – это критически важный этап в процессе разработки программного обеспечения, так как от качества и полноты собранных требований напрямую зависит успех проекта. Неправильно или неполно собранные требования могут привести к созданию системы, которая не соответствует ожиданиям пользователей и не решает поставленных задач, что в итоге приведет к дополнительным затратам на доработку и улучшение продукта.

В данной работе использованы следующие методы сбора требований:

1. Интервью: проведение интервью с системными администраторами и техническим персоналом МКУ «СКО» г. Арзамаса позволило получить подробную информацию о текущих процессах управления ИТ-инфраструктурой, проблемах и ожиданиях от нового программного продукта. Интервью проводились с целью выявления основных проблем, с которыми сталкиваются системные администраторы, и определения функциональных и нефункциональных требований к системе.

Подготовка к интервью: перед проведением интервью были подготовлены списки вопросов, направленных на выявление текущих проблем, используемых инструментов, желаемых функций и характеристик нового программного продукта. Вопросы были разделены на несколько категорий:

1. Текущие проблемы и вызовы: какие проблемы наиболее часто возникают при управлении ИТ-инфраструктурой? Каковы основные вызовы в повседневной работе?
2. Используемые инструменты и процессы: какие инструменты и процессы используются для мониторинга, диагностики и устранения неполадок? Насколько они эффективны?
3. Потребности и ожидания: какие функции и возможности должны быть реализованы в новом программном продукте? Какие задачи должны быть автоматизированы в первую очередь?

Проведение интервью: Интервью проводились в формате личных встреч и телефонных разговоров. В процессе интервью участники делились своим опытом, обсуждали текущие проблемы и предлагали свои идеи по улучшению системы. Все ответы и предложения фиксировались для дальнейшего анализа.

1. Анкетирование: Анкетирование сотрудников учреждения позволило собрать мнения и предложения по улучшению процессов управления ИТ-инфраструктурой[9, c.56]. Анкеты были разработаны с учетом специфики работы различных отделов и включали вопросы о текущих проблемах, пожеланиях и требованиях к новому программному продукту.

Разработка анкеты: Анкета включала вопросы как закрытого, так и открытого типа.

Вопросы закрытого типа позволяли быстро получить статистические данные, тогда как вопросы открытого типа давали возможность сотрудникам выразить свои мнения и предложения.

Примеры вопросов анкеты:

* + Как часто возникают проблемы с ИТ-оборудованием?
  + Какие типы проблем возникают чаще всего (аппаратные сбои, сетевые проблемы, проблемы с программным обеспечением)?
  + Какие инструменты и методы вы используете для устранения проблем?
  + Какие функции вы хотели бы видеть в новом программном продукте?

Распространение и сбор данных: Анкеты распространялись среди сотрудников через электронную почту и внутреннюю сеть учреждения. Для повышения отклика сотрудников было организовано несколько напоминаний о необходимости заполнения анкет. Собранные данные обрабатывались и анализировались для выявления общих тенденций и проблем.

1. Анализ документов: изучение существующих отчетов, регламентов и других документов, используемых в МКУ «СКО», позволило понять текущие процессы управления ИТ-инфраструктурой и выявить основные проблемы и недостатки. Анализ документов включал изучение отчетов по управлению сетевым оборудованием, программного обеспечения, планов резервного копирования и других нормативных документов.

Приведем описание типов анализируемых документов.

1. Отчеты о состоянии оборудования: регулярные отчеты, создаваемые системными администраторами, содержащие информацию о текущем состоянии оборудования, частоте и типах возникающих проблем, а также времени, затрачиваемом на их устранение.
2. Планы резервного копирования и восстановления данных: документы, описывающие процедуры и графики резервного копирования данных, а также планы действий в случае сбоев и утраты данных.
3. Регламенты и инструкции: внутренние регламенты и инструкции по управлению ИТ-инфраструктурой, включая протоколы мониторинга, диагностики и устранения неполадок, а также процедуры безопасности[10, c.78].

Анализ данных: Все данные, собранные из документов, были структурированы и проанализированы для выявления частых проблем, эффективности существующих процессов и областей, требующих улучшения. Результаты анализа помогли уточнить требования к новому программному продукту.

Анализ и обработка данных. После сбора всех данных был проведен их тщательный анализ с целью выявления основных тенденций, проблем и требований. Для этого использовались следующие методы.

1. Классификация данных: все собранные данные были классифицированы по категориям (например, типы проблем, используемые инструменты, желаемые функции) для удобства анализа.
2. Статистический анализ: для количественных данных (например, частота возникновения проблем, время на устранение) был проведен статистический анализ, который позволил выявить наиболее критические области и приоритеты.
3. Качественный анализ: открытые ответы и комментарии сотрудников были проанализированы с использованием методов качественного анализа для выявления ключевых проблем и предложений по улучшению системы.

Формирование требований. На основе анализа собранных данных были сформированы функциональные и нефункциональные требования к программному продукту. Функциональные требования описывают основные функции, которые должен выполнять продукт, а нефункциональные требования определяют характеристики, связанные с производительностью, надежностью и удобством использования.

Верификация требований. После формирования требований был проведен этап их верификации, включающий:

1. Обратная связь от пользователей: требования были представлены ключевым пользователям (системным администраторам и техническому персоналу) для получения обратной связи. Это позволило убедиться, что требования правильно отражают потребности пользователей и не содержат противоречий.
2. Приоритизация требований: все требования были приоритизированы по степени важности и срочности. Это помогло определить, какие функции должны быть реализованы в первую очередь, а какие могут быть отложены на более поздние этапы разработки.
3. Документирование требований: окончательно утвержденные требования были задокументированы и представлены в виде спецификации требований, которая служит основой для дальнейшей разработки и тестирования программного продукта.

Таким образом, сбор и анализ требований к программному продукту позволили получить исчерпывающую информацию о потребностях и ожиданиях пользователей, выявить основные проблемы и определить ключевые функции и характеристики новой системы.

Этот этап является критически важным для успешной реализации проекта, так как от качества и полноты собранных требований зависит эффективность и результативность разработанного программного продукта.

1.3 Формирование функциональных и нефункциональных требований

На основе собранных данных и проведенного анализа были сформированы функциональные и нефункциональные требования к программному продукту. Функциональные требования описывают основные функции, которые должен выполнять продукт, а нефункциональные требования определяют характеристики, связанные с производительностью, надежностью и удобством использования. Правильное формирование требований является основой успешной разработки, так как определяет, каким должен быть конечный продукт и как он будет функционировать.

Функциональные требования описывают конкретные функции и возможности, которые должны быть реализованы в программном продукте. В данном проекте основные функциональные требования включают:

1. Мониторинг состояния оборудования:

* возможность отслеживания состояния всех устройств в реальном времени;
* отображение информации о текущем состоянии каждого устройства (включено/выключено, доступно/недоступно, статус работы);
* регулярное обновление данных с использованием протоколов snmp (simple network management protocol) и wmi (windows management instrumentation);
* генерация уведомлений при возникновении проблем или изменении состояния устройства.

1. Диагностика неполадок:

* автоматическое обнаружение проблем с оборудованием и программным обеспечением;
* анализ логов и событий для выявления причин проблем;
* тестирование соединений и проверка конфигураций устройств;
* предоставление рекомендаций по устранению выявленных проблем, включая шаги по их решению и ссылки на соответствующие инструкции.

1. Управление устройствами:

* возможность удаленного управления устройствами (включение/выключение, перезагрузка, обновление программного обеспечения);
* управление настройками и конфигурациями устройств через централизованный интерфейс;
* поддержка массовых операций (например, обновление программного обеспечения на нескольких устройствах одновременно).

1. Генерация отчетов и аналитика:

* автоматическое создание отчетов о состоянии ит-инфраструктуры;
* анализ производительности устройств и выявление тенденций и аномалий;
* поддержка различных типов отчетов (ежедневные, еженедельные, ежемесячные) и возможность настройки их параметров;
* экспорт отчетов в различные форматы (pdf, excel, csv).

1. Интеграция с существующими системами:

* поддержка интеграции с другими системами управления ит-инфраструктурой, такими как системами управления активами (asset management) и системами управления конфигурацией (configuration management);
* обеспечение обмена данными и синхронизации информации между системами;
* использование стандартных протоколов и api для интеграции.

Нефункциональные требования описывают характеристики программного продукта, которые влияют на его производительность, надежность, безопасность и удобство использования. В данном проекте основные нефункциональные требования включают:

1. Производительность:

* программный продукт должен обеспечивать быструю обработку данных и оперативное реагирование на запросы пользователей;
* время реакции на запросы не должно превышать нескольких секунд;
* система должна поддерживать работу с большим количеством устройств без снижения производительности.

1. Надежность:

* программный продукт должен быть устойчив к сбоям и обеспечивать непрерывную работу;
* в случае сбоя система должна автоматически восстанавливаться и сохранять данные;
* регулярное создание резервных копий данных и возможность их восстановления.

1. Безопасность:

* программный продукт должен обеспечивать защиту данных от несанкционированного доступа и возможных угроз;
* поддержка аутентификации и авторизации пользователей;
* шифрование данных при передаче и хранении.

1. Масштабируемость:

* программный продукт должен быть легко масштабируемым, что позволит адаптировать его к изменяющимся потребностям учреждения;
* возможность добавления новых устройств и увеличения объема обрабатываемых данных без значительных изменений в системе.

1. Удобство использования:

* интуитивно понятный интерфейс, который позволяет пользователям легко выполнять необходимые операции без дополнительного обучения;
* поддержка нескольких языков интерфейса;
* возможность настройки интерфейса под потребности конкретных пользователей и отделов.

Документирование требований. После формирования требований они были задокументированы в виде спецификации требований.

Спецификация включает детальное описание всех функциональных и нефункциональных требований, а также примеры использования и сценарии тестирования. Документирование требований позволяет:

* + 1. Обеспечить ясное понимание задач и целей проекта для всех участников разработки.
    2. Снизить риск недопонимания и ошибок при реализации программного продукта.
    3. Облегчить процесс тестирования и валидации системы.

После документирования требований была проведена их верификация и утверждение. Этот этап включал:

1. Обратная связь от пользователей: требования были представлены ключевым пользователям (системным администраторам и техническому персоналу) для получения обратной связи. Это позволило убедиться, что требования правильно отражают потребности пользователей и не содержат противоречий.
2. Приоритизация требований: все требования были приоритизированы по степени важности и срочности. Это помогло определить, какие функции должны быть реализованы в первую очередь, а какие могут быть отложены на более поздние этапы разработки.
3. Анализ на предмет полной и непротиворечивости: были проведены проверки на предмет полноты и непротиворечивости требований, чтобы избежать пропуска важных аспектов и устранить возможные конфликты между требованиями[3, c.456].

Для успешной реализации проекта было важно определить приоритеты внедрения различных функций и характеристик системы. Приоритизация проводилась с учетом следующих критериев:

* + 1. Критичность для пользователей: насколько важно данное требование для ежедневной работы системных администраторов и других сотрудников.
    2. Влияние на производительность и безопасность: насколько сильно данное требование влияет на общую производительность системы и безопасность данных.
    3. Сложность реализации: оценка трудозатрат и временных затрат на реализацию требования.
    4. Зависимости от других требований: выявление зависимостей между требованиями, чтобы избежать ситуаций, когда реализация одного требования зависит от выполнения другого.

Формирование функциональных и нефункциональных требований является ключевым этапом в разработке программного продукта. Правильное определение и документирование требований обеспечивает ясное понимание целей и задач проекта, снижает риск ошибок и недопонимания, а также облегчает процесс разработки, тестирования и внедрения системы. Выполненные в рамках данного проекта этапы сбора и анализа требований позволили создать детальную и полную спецификацию, которая служит надежной основой для дальнейшей работы.

1.4 Моделирование программной системы на основе собранных данных

На основании собранных данных и сформированных требований была разработана модель программной системы. Моделирование включало создание диаграмм и схем, которые наглядно представляют структуру и функциональные возможности будущего программного продукта. Этот этап позволяет визуализировать архитектуру системы, ее компоненты и их взаимодействие, что является важным шагом для успешной реализации проекта.

Цели моделирования. Моделирование программной системы преследует несколько ключевых целей:

1. Определение структуры системы: выявление основных компонентов системы и их взаимосвязей.
2. Описание бизнес-логики: моделирование основных процессов и алгоритмов, которые будут реализованы в системе.
3. Упрощение разработки: предоставление разработчикам четкого представления о структуре и функциональности системы, что упрощает процесс кодирования.
4. Выявление и устранение недостатков: обнаружение возможных проблем и недостатков на ранних этапах разработки, что позволяет их оперативно исправить.

Для моделирования программной системы были использованы различные типы диаграмм и схем, каждая из которых имеет свою специфику и назначение.

Диаграммы классов представляют собой схему, отображающую основные классы системы и их взаимосвязи. Диаграммы классов позволяют определить основные объекты системы, их свойства и методы, а также взаимосвязи между ними. В данном проекте диаграммы классов использовались для моделирования основных объектов управления ИТ-инфраструктурой, таких как серверы, сетевые устройства, рабочие станции и т.д.

Диаграммы вариантов использования (Use Case). Диаграммы вариантов использования отображают функциональные требования к системе и взаимодействие пользователей с системой. Они помогают определить, какие функции должна выполнять система и как пользователи будут взаимодействовать с этими функциями. Представлена на рисунке 1.

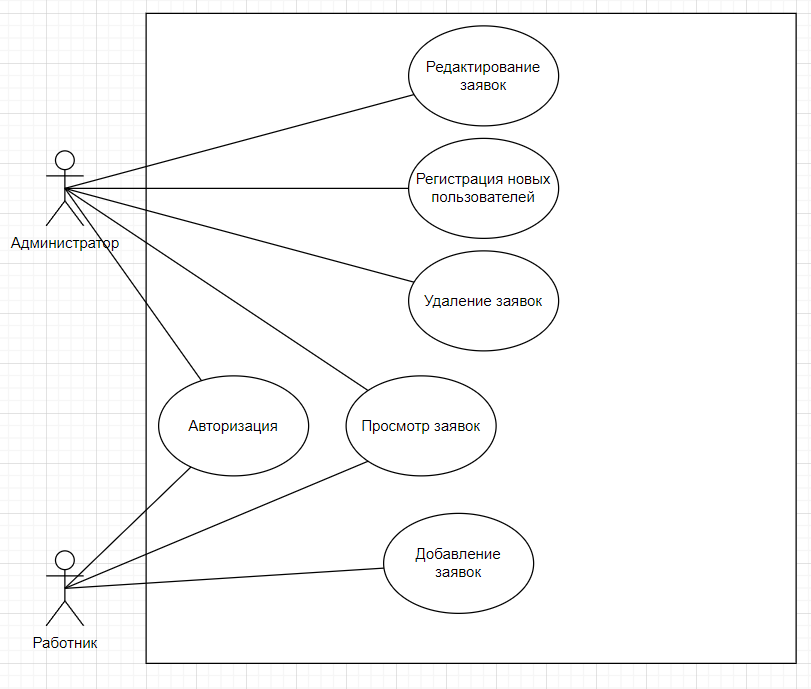


Рисунок 1 – USE-CASE диаграмма

ERD (Entity-Relationship Diagram) используется для моделирования структуры базы данных. Диаграмма представляет собой визуальное отображение сущностей, их атрибутов и связей между ними, что позволяет понять, как данные будут организованы и взаимодействовать в системе. Представлена на рисунке 2.

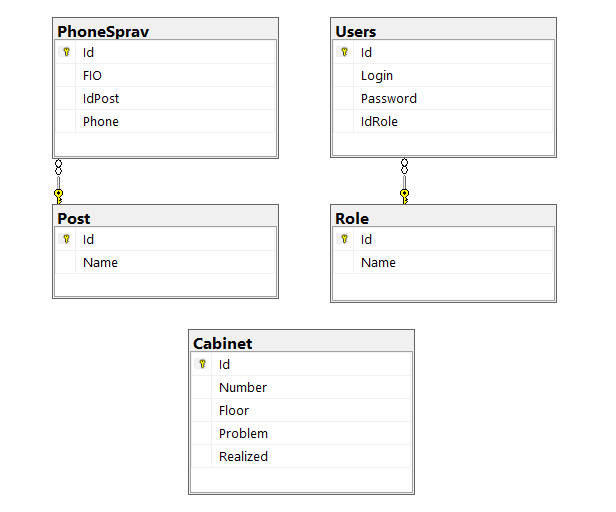


Рисунок 2 – ERD диаграмма

Диаграммы компонентов

Диаграммы компонентов показывают физическое распределение программных компонентов и их взаимодействие. В данном проекте диаграммы компонентов использовались для представления архитектуры системы, включая серверную часть, клиентскую часть и базу данных.

Пример диаграммы компонентов:

* + 1. Компоненты:
    2. ClientApplication: включает в себя пользовательский интерфейс и логику взаимодействия с сервером.
    3. WebServer: обрабатывает запросы от клиентского приложения, взаимодействует с базой данных.
    4. DatabaseServer: отвечает за хранение и управление данными.
    5. MonitoringService: сервис для мониторинга состояния устройств и сбора данных.
    6. Взаимодействие:
    7. ClientApplication отправляет запросы на WebServer.
    8. WebServer взаимодействует с DatabaseServer для чтения и записи данных.
    9. MonitoringService собирает данные с устройств и отправляет их на WebServer.

Моделирование бизнес-логики системы включает описание алгоритмов и процессов, которые будут реализованы в программном продукте[5, c.42]. Это включает:

* + 1. Алгоритмы мониторинга и диагностики: описание процессов сбора данных с устройств, анализа логов и событий, выявления проблем и предоставления рекомендаций по их устранению.
    2. Алгоритмы управления устройствами: описание процессов удаленного управления устройствами, включая перезагрузку, обновление программного обеспечения и изменение настроек.
    3. Алгоритмы генерации отчетов и аналитики: описание процессов создания отчетов о состоянии ИТ-инфраструктуры, анализа производительности устройств и выявления тенденций и аномалий.

Пример алгоритма диагностики неполадок:

1. Сбор данных: система собирает логи и события с устройства.
2. Анализ данных: логи и события анализируются для выявления отклонений от нормального состояния.
3. Выявление проблемы: на основе анализа данных определяется тип проблемы (например, сбой оборудования, проблема с сетью).
4. Предоставление рекомендаций: система генерирует рекомендации по устранению проблемы, включая конкретные шаги и ссылки на соответствующие инструкции.

После создания модели системы была проведена ее валидация для проверки правильности и полноты. Валидация включала:

* + 1. Рецензирование модели: модель была представлена ключевым пользователям и техническим специалистам для получения обратной связи и выявления возможных ошибок и недостатков.
    2. Проверка на соответствие требованиям: модель проверялась на соответствие сформированным требованиям, чтобы убедиться, что все функциональные и нефункциональные требования правильно отражены в модели.
    3. Тестирование сценариев использования: были разработаны и протестированы различные сценарии использования системы для проверки ее работоспособности и выявления возможных проблем.

Моделирование программной системы является важным этапом в процессе разработки, который позволяет наглядно представить структуру и функциональные возможности будущего продукта, выявить и устранить возможные проблемы на ранних этапах, а также обеспечить четкое понимание задач и целей проекта для всех участников разработки. Созданные диаграммы и схемы обеспечивают разработчикам и техническим специалистам ясное представление о структуре системы, ее компонентах и взаимодействиях, что упрощает процесс реализации и тестирования программного продукта.

2. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА «ADMIN» ДЛЯ МКУ «СКО» Г. АРЗАМАСА

2.1. Технологии разработки

Для успешной реализации программного продукта «ADMIN» были выбраны современные и надежные технологии, которые обеспечивают высокую производительность, безопасность и удобство использования. В этом разделе рассмотрим основные технологии и инструменты, используемые для разработки программного продукта.

Дадим краткую характеристику используемым языкам программирования.

1. C#: основной язык программирования для реализации программного продукта. C# выбран за его мощные возможности, поддержку объектно-ориентированного программирования и широкую применяемость в разработке настольных приложений. C# предоставляет мощные средства для работы с данными, управления потоками, сетевого программирования и взаимодействия с операционной системой, что делает его идеальным выбором для разработки сложных корпоративных приложений.
2. XAML (Extensible Application Markup Language): язык разметки, используемый в WPF для описания пользовательского интерфейса. XAML позволяет отделить дизайн интерфейса от логики приложения, что облегчает разработку и поддержку. Использование XAML упрощает создание сложных и интерактивных интерфейсов, поддерживая привязку данных, стили и шаблоны[8, c.73].

Характеристика среды разработки Visual Studio: это мощная интегрированная среда разработки (IDE) от Microsoft, предоставляющая все необходимые инструменты для разработки, отладки и тестирования приложений на языке C#. Visual Studio обеспечивает удобство и эффективность разработки, предлагая широкий набор функций, таких как IntelliSense, отладка в реальном времени, интеграция с системами контроля версий и инструменты для профилирования и тестирования производительности[18, c.251].

Характеристика применяемого фреймворка. Windows Presentation Foundation (WPF): современный фреймворк для создания настольных приложений на платформе .NET. WPF предоставляет мощные инструменты для разработки гибких и интерактивных пользовательских интерфейсов, поддерживая работу с мультимедиа, графикой и анимацией. WPF также обеспечивает высокую производительность и возможность использования аппаратного ускорения для отрисовки графики.

Используемая система управления базами данных. Microsoft SQL Server (MSSQL): мощная реляционная база данных, обеспечивающая высокую производительность, масштабируемость и надежность. MSSQL поддерживает сложные запросы, транзакции и интеграцию с другими продуктами Microsoft, что делает ее идеальным выбором для корпоративных приложений. SQL Server также предоставляет средства для анализа данных, резервного копирования и восстановления, а также обеспечения безопасности данных.

Инструменты разработки. Git: система контроля версий, которая позволяет отслеживать изменения в коде и управлять различными версиями проекта. Использование Git обеспечивает командную работу, упрощает процесс разработки и тестирования, позволяет разработчикам работать над различными частями проекта независимо друг от друга, а затем объединять свои изменения. Git также предоставляет инструменты для управления ветками, слияния изменений и разрешения конфликтов, что делает его незаменимым инструментом для командной разработки.

Дополнительные библиотеки и инструменты. Entity Framework: ORM (Object-Relational Mapping) библиотека для работы с базой данных. Entity Framework позволяет разработчикам работать с данными в базе данных через объектно-ориентированные модели, что упрощает процесс взаимодействия с данными и обеспечивает автоматическое создание и управление схемой базы данных. Entity Framework также поддерживает запросы LINQ, что делает код более читаемым и поддерживаемым.

Newtonsoft.Json: библиотека для работы с JSON. Использование Newtonsoft.Json позволяет легко сериализовать и десериализовать данные, что упрощает взаимодействие между клиентской и серверной частями приложения, а также интеграцию с внешними сервисами[21, c.43].

NUnit: фреймворк для модульного тестирования. NUnit используется для написания и выполнения тестов, что помогает обеспечивать качество и надежность кода, а также упрощает процесс обнаружения и исправления ошибок на ранних этапах разработки.

Рассмотрим применяемые архитектурные подходы и паттерны.

MVVM (Model-View-ViewModel): паттерн проектирования, широко используемый в WPF-приложениях. MVVM позволяет разделить логику представления, бизнес-логику и данные, что делает код более структурированным и поддерживаемым. Использование MVVM также облегчает тестирование и повторное использование компонентов.

Dependency Injection: подход, позволяющий создавать гибкие и легко тестируемые приложения. Dependency Injection обеспечивает инверсию управления и позволяет упростить замену компонентов, что делает систему более модульной и облегчает её модификацию.

Repository Pattern: паттерн проектирования, который абстрагирует логику доступа к данным, что делает код более чистым и тестируемым. Repository Pattern также упрощает работу с базой данных и позволяет изменять способ хранения данных без изменения бизнес-логики приложения.

Процессы разработки. Agile: гибкий метод управления проектами, который позволяет оперативно реагировать на изменения требований и быстро адаптироваться к новым условиям. Agile методология включает итеративный подход к разработке, регулярные встречи команды (Scrum), и непрерывное улучшение процесса разработки.

Continuous Integration/Continuous Deployment (CI/CD): практика, направленная на автоматизацию процесса интеграции и развертывания кода. CI/CD позволяет регулярно интегрировать изменения в коде, автоматически выполнять тесты и развертывать приложение в различных средах, что снижает риск ошибок и упрощает процесс выпуска новых версий.

Инструменты мониторинга и логирования.

Serilog: библиотека для логирования, которая позволяет записывать события и ошибки в различных форматах и источниках (файлы, базы данных, консоль и т.д.). Использование Serilog помогает отслеживать работу приложения, выявлять и устранять проблемы на ранних этапах.

Application Insights: сервис от Microsoft для мониторинга и анализа приложений. Application Insights предоставляет подробную информацию о производительности, использовании и ошибках в приложении, что помогает улучшить качество и надежность программного продукта[22, c.76].

Преимущества выбранных технологий:

1. Высокая производительность: использование C#, WPF и MSSQL обеспечивает высокую производительность приложения, что особенно важно при работе с большим количеством данных и сложными операциями.
2. Масштабируемость и надежность: MSSQL предоставляет возможности для масштабирования и обеспечения высокой надежности данных, что позволяет системе расти вместе с потребностями учреждения.
3. Удобство разработки: Visual Studio и интеграция с Git облегчают процесс разработки, отладки и тестирования, что позволяет команде работать более эффективно и снижать количество ошибок.
4. Интерактивный и гибкий интерфейс: WPF и XAML позволяют создавать современные и удобные пользовательские интерфейсы, которые могут быть легко адаптированы под потребности пользователей.
5. Безопасность данных: MSSQL и современные методы аутентификации и авторизации обеспечивают высокий уровень безопасности данных, что особенно важно для защиты информации в муниципальных учреждениях.

Выбор этих технологий обусловлен их широким применением в разработке настольных приложений, высокой производительностью, безопасностью и удобством использования. Эти инструменты и технологии позволяют разработчикам создавать масштабируемые, надежные и поддерживаемые системы, которые соответствуют современным требованиям и стандартам.

2.2. Архитектура программного продукта

Архитектура программного продукта разрабатывалась с учетом необходимости обеспечения высокой производительности, надежности и безопасности системы. В этом разделе рассмотрим основные компоненты архитектуры и их взаимодействие.

Рассмотрим общие принципы архитектуры.

Разработка архитектуры программного продукта основывается на нескольких ключевых принципах[23, c.142]:

1. Модульность: программа разделена на независимые модули, каждый из которых отвечает за выполнение конкретных функций. Это облегчает поддержку и обновление системы.
2. Масштабируемость: архитектура должна позволять легко расширять и масштабировать систему по мере роста требований и нагрузки.
3. Надежность и отказоустойчивость: система должна быть устойчивой к сбоям и обеспечивать непрерывную работу даже при возникновении ошибок или отказе отдельных компонентов.
4. Безопасность: обеспечение защиты данных от несанкционированного доступа и обеспечение безопасности взаимодействия между компонентами.

Назовем основные компоненты архитектуры.

Архитектура программного продукта включает следующие основные компоненты:

1. Клиентская часть (Frontend)
2. Серверная часть (Backend)
3. База данных (Database)
4. Службы мониторинга и уведомлений
5. Интерфейсы и API

1. Клиентская часть (Frontend).

Клиентская часть представляет собой пользовательский интерфейс, который взаимодействует с серверной частью через API. Она реализована с использованием WPF и XAML, что обеспечивает динамичность и интерактивность интерфейса[6, c.132]. Основные задачи клиентской части:

* отображение информации о состоянии оборудования и системных ресурсов;
* обеспечение средств для управления устройствами и выполнения операций;
* обеспечение интуитивно понятного и удобного интерфейса для пользователей.

Клиентская часть взаимодействует с серверной частью через HTTP-запросы, отправляемые на RESTful API, реализованный в серверной части. Это обеспечивает независимость клиентской и серверной частей и позволяет легко изменять и обновлять интерфейс без необходимости внесения изменений в серверную логику.

2. Серверная часть (Backend).

Серверная часть отвечает за обработку бизнес-логики, взаимодействие с базой данных и обеспечение API для клиентской части. Она реализована с использованием C# и .NET. Основные задачи серверной части:

1. Обработка запросов от клиентской части.
2. Взаимодействие с базой данных для чтения и записи данных.
3. Выполнение бизнес-логики, связанной с мониторингом и управлением устройствами.
4. Генерация отчетов и аналитических данных.

Серверная часть использует ASP.NET Core для создания веб-сервиса, который обрабатывает HTTP-запросы и возвращает ответы в формате JSON. Это позволяет клиентской части взаимодействовать с сервером независимо от платформы и языка программирования.

3. База данных (Database).

База данных реализована с использованием Microsoft SQL Server (MSSQL)[4, c.56], что обеспечивает надежное и безопасное хранение данных, а также поддержку сложных запросов и транзакций. Основные задачи базы данных:

1. Хранение информации о пользователях, устройствах, конфигурациях и событиях.
2. Обеспечение целостности и консистентности данных.
3. Поддержка резервного копирования и восстановления данных.

4. Службы мониторинга и уведомлений.

Службы мониторинга и уведомлений играют ключевую роль в обеспечении оперативного реагирования на проблемы и поддержании высокой доступности системы. Основные задачи этих служб:

1. Непрерывный мониторинг состояния устройств и системных ресурсов.
2. Сбор и анализ данных в реальном времени.
3. Генерация уведомлений при возникновении проблем или отклонений от нормы.
4. Предоставление рекомендаций по устранению выявленных проблем.
5. Для реализации служб мониторинга используются следующие технологии:
6. SNMP (Simple Network Management Protocol): для мониторинга сетевых устройств.
7. WMI (Windows Management Instrumentation): для мониторинга систем на базе Windows.
8. Планировщик задач Windows: для выполнения регулярных проверок и сборов данных.

Взаимодействие между компонентами.

Компоненты архитектуры взаимодействуют между собой следующим образом:

1. Клиентская часть: пользователь взаимодействует с клиентским приложением, которое отправляет HTTP-запросы на серверную часть через API.
2. Серверная часть: серверная часть обрабатывает запросы, выполняет бизнес-логику и взаимодействует с базой данных для чтения и записи данных.
3. База данных: сохраняет и управляет данными, обеспечивая их целостность и доступность.
4. Службы мониторинга и уведомлений: непрерывно собирают и анализируют данные о состоянии устройств, генерируют уведомления и предоставляют рекомендации.
5. Интерфейсы и API: обеспечивают стандартизированное и безопасное взаимодействие между компонентами системы и внешними сервисами.

Архитектурные решения и паттерны.

Для обеспечения гибкости, модульности и масштабируемости системы были использованы следующие архитектурные решения и паттерны[11, c.98]:

1. Микросервисная архитектура: система разделена на отдельные сервисы, каждый из которых выполняет свою функцию. Это обеспечивает гибкость и масштабируемость, так как каждый сервис можно разрабатывать, развертывать и масштабировать независимо.
2. MVVM (Model-View-ViewModel): использование паттерна MVVM в клиентской части позволяет отделить логику представления от бизнес-логики и данных, что облегчает тестирование и поддержку кода.
3. Repository Pattern: паттерн репозитория используется для абстрагирования логики доступа к данным и обеспечения чистого и тестируемого кода. Репозиторий предоставляет унифицированный интерфейс для взаимодействия с базой данных и упрощает замену источника данных при необходимости.
4. Dependency Injection (DI): использование DI для управления зависимостями между компонентами обеспечивает инверсию управления, улучшает тестируемость и упрощает замену и конфигурацию компонентов.
5. Event-Driven Architecture: использование событий для взаимодействия между компонентами системы позволяет обрабатывать события асинхронно и повышает отказоустойчивость и масштабируемость системы[15, c.78].

Архитектура программного продукта «ADMIN» обеспечивает высокую производительность, надежность и безопасность системы, соответствуя современным требованиям и стандартам. Выбранные технологии и архитектурные решения позволяют создавать масштабируемую, гибкую и поддерживаемую систему, которая эффективно решает задачи мониторинга и управления ИТ-инфраструктурой в МКУ «СКО» г. Арзамаса.

2.3. Пользовательский интерфейс

Разработка пользовательского интерфейса информационной системы осуществляется с помощью платформы Visual Studio в среде Windows Presentation Foundation (WPF). Согласно заданию и макету пользовательского интерфейса, разработанного ранее, была начата работа по созданию Главной формы приложения ИС «МКУ «СКО» г. Арзамаса», которая представлена на рисунке 3.

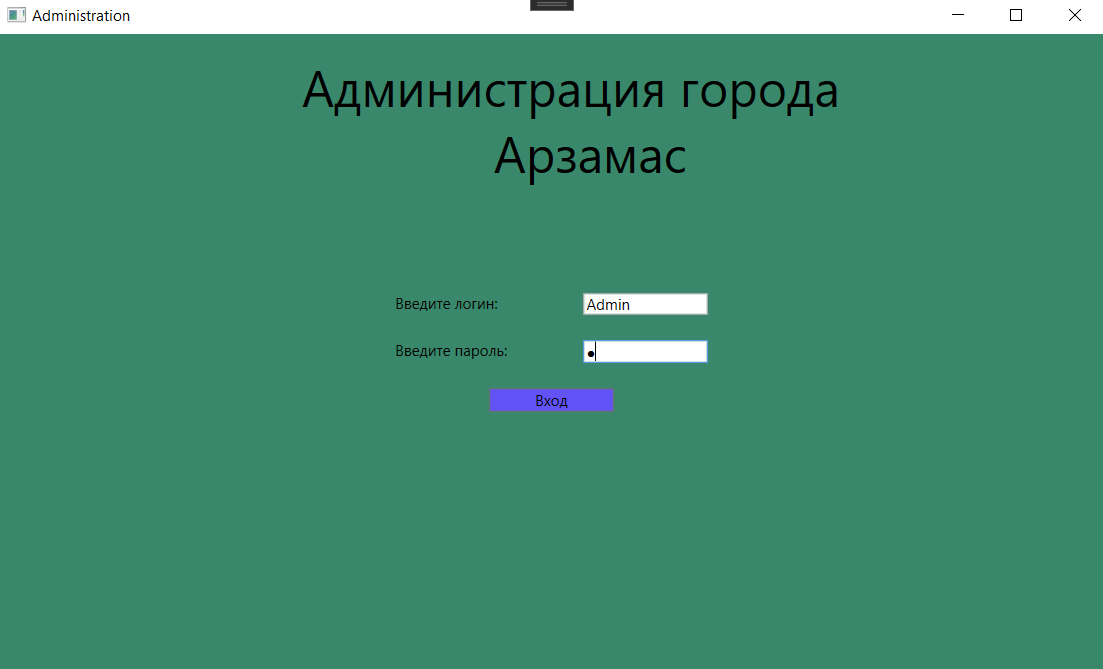


Рисунок 3 - Главная форма

Рассмотрим фрагмент разметки главной формы (рисунок 4), которая отвечает за дизайн формы авторизации в нашем приложение. На странице авторизации расположены 2 поля для входа, одна кнопка, текст для взаимодействия с пользователем. Шрифт и цвет подобраны под деловой стиль нашего приложения.



Рисунок 4 – Фрагмент разметки главной формы

По нажатию на кнопку «Вход» происходит поиск в базе данных такого пользователя, если его нет, то выводится сообщение, что такого пользователя не найдено, если он найден, то в связи с его ролью открывается то или иное окно. Реализация авторизации представлена в листинге 1.

Листинг 1 – Код страницы авторизации

try

{

var userObj = DbConnect.entObj.Users.FirstOrDefault(x => x.Login == TxbLogin.Text && x.Password == PsbPassword.Password);

if (userObj == null)

{

MessageBox.Show("Такой пользователь не найден!",

"Уведомление",

MessageBoxButton.OK,

MessageBoxImage.Information);

Продолжение листинга 1

}

else

{

switch (userObj.IdRole)

{

case 1:

MessageBox.Show("Здравствуйте, администратор " + userObj.Login + "!",

"Уведомление",

MessageBoxButton.OK,

MessageBoxImage.Information);

FrameApp.frmObj.Navigate(new Admin());

break;

case 2:

MessageBox.Show("Здравствуйте, работник " + userObj.Login + "!",

"Уведомление",

MessageBoxButton.OK,

MessageBoxImage.Information);

FrameApp.frmObj.Navigate(new User());

break;

default:

MessageBox.Show("Такой пользователь не найден!",

"Уведомление",

MessageBoxButton.OK,

MessageBoxImage.Information);

break;

}

}

}

catch (Exception ex)

{

MessageBox.Show("Критический сбой в работе предложения: " + ex.Message.ToString(),

"Уведомление",

MessageBoxButton.OK,

MessageBoxImage.Warning);

}

После была разработана архитектура папок в приложении, это является одним из важных моментов для дальнейшей разработки (рисунок 5).

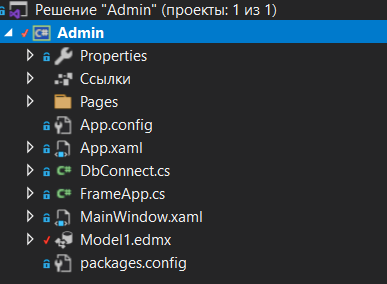


Рисунок 5 - Архитектура папок в приложение

Спроектировано окно навигации. Оно содержит кнопки, позволяющие быстро переключаться между различными модулями администратора (например, список кабинетов, просмотр справочника и дообавление сотрудника). Пользовательский интерфейс для Администратора, представленного на рисунке 6.

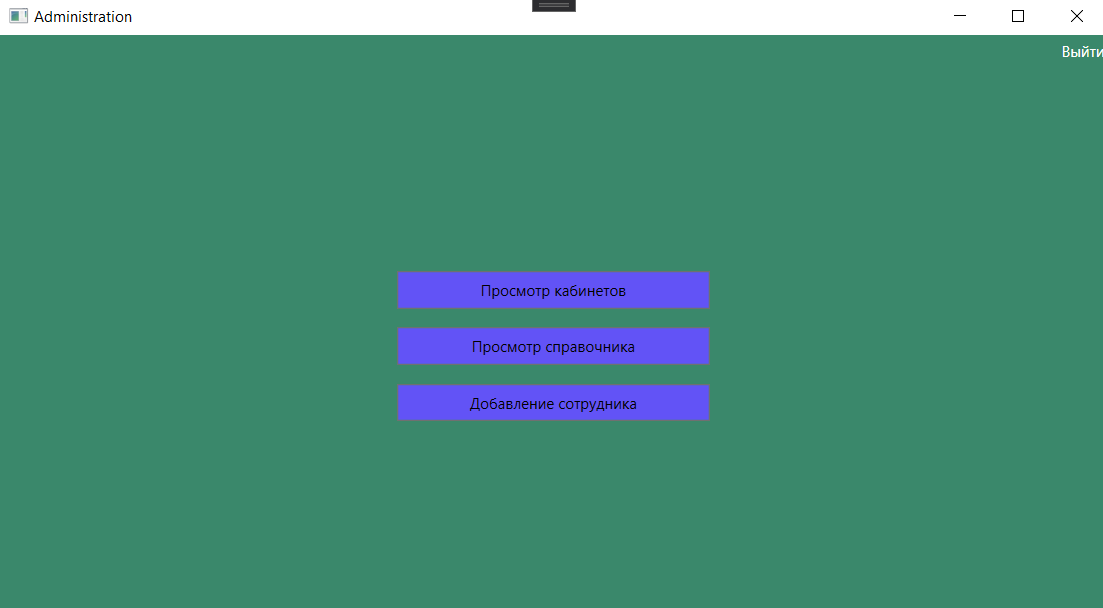


Рисунок 6 – Пользовательский интерфейс для администратора

После рассмотрим фрагмент разметки пользовательский интерфейс для администратора. На котором изображено 4 кнопки, один рисунок, представлено на рисунке 7.

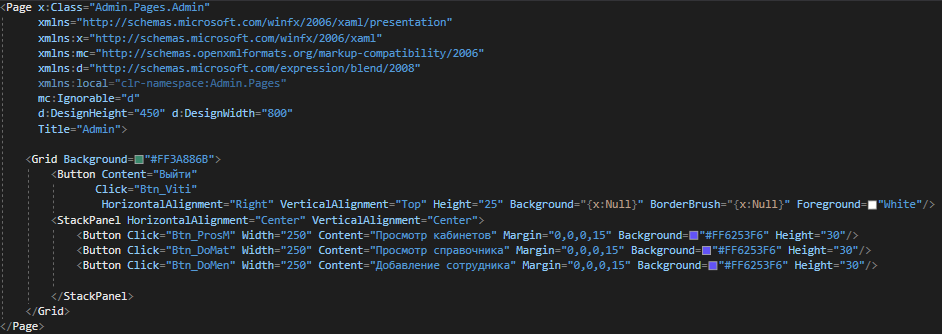


Рисунок 7 – Фрагмент разметки главной формы

По нажатию на кнопку происходит событие обработки кнопки «Список кабинетов» (листинг 2), которое открывает новую страницу.

Листинг 2 – Код кнопки «Список кабинетов»

private void Btn\_ProsM(object sender, RoutedEventArgs e)

{

FrameApp.frmObj.Navigate(new SpisokSotr());

}

По нажатию на кнопку происходит событие обработки кнопки «просмотр справочника», (листинг 3), которое открывает новое окно.

Листинг 3 – Код кнопки «Просмотр справочника».

private void Btn\_DoMat(object sender, RoutedEventArgs e)

{

FrameApp.frmObj.Navigate(new Spravochnik());

}

По нажатию на кнопку происходит событие обработки кнопки «Добавление сотрудника» (листинг 4), которое открывает новое окно.

Листинг 4 – Код кнопки «Добавление сотрудника»

private void Btn\_DoMen(object sender, RoutedEventArgs e)

{

FrameApp.frmObj.Navigate(new DobavSotr(null));

}

После рассмотрим каждую страницу по отдельности, начнем с «Список кабинетов». интерфейс представлен на рисунке 8.



Рисунок 8- Интерфейс на страницы «Список кабинетов»

После рассмотрим фрагмент разметки страницы «Список кабинетов», на которой изображено 3 кнопки, одна таблицы (рисунок 9).

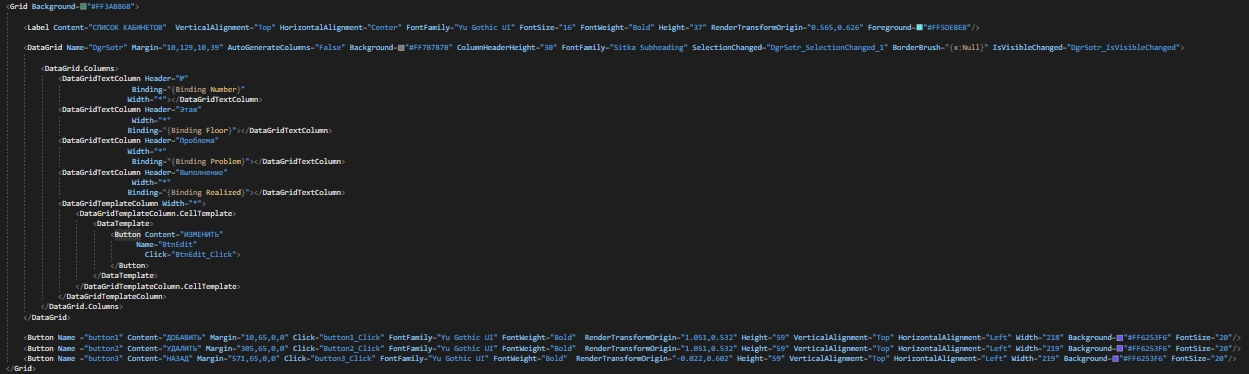


Рисунок 9 – Фрагмент разметки страницы «Список кабинетов»

После рассмотрим страницу добавления проблем, которая представлена на рисунке 10.

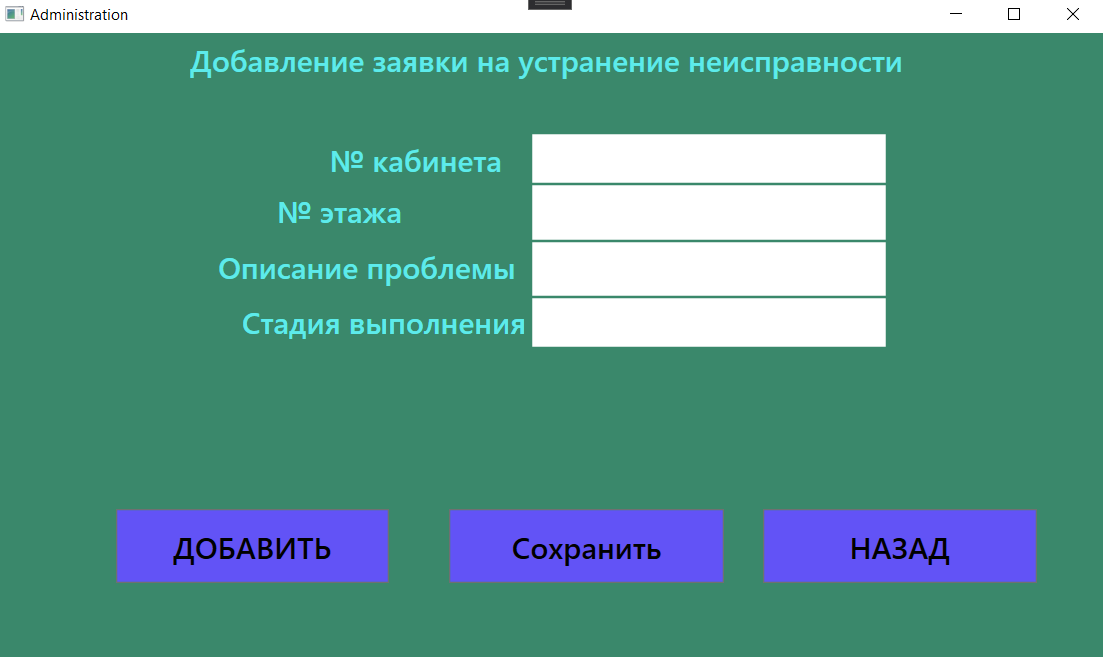


Рисунок 10 – Интерфейс страницы «Добавление заявок»

После рассмотрим фрагмент разметки страницы «Добавление», который представлен на рисунке 11.

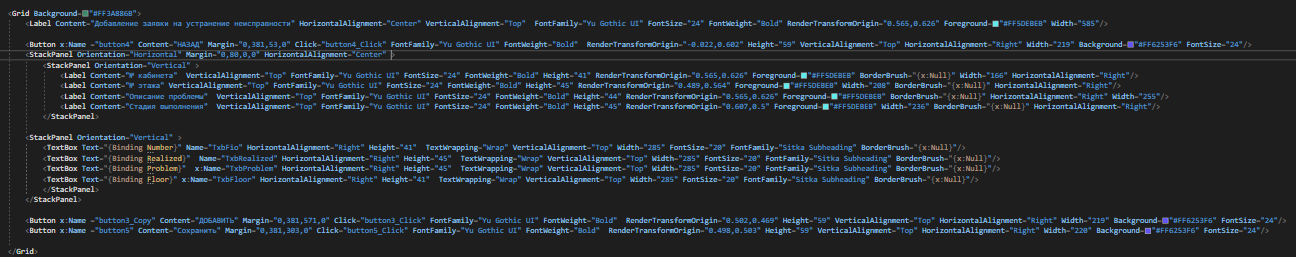


Рисунок 11 – Фрагмент разметки страницы «Добавление»

После нажатия на кнопку происходит событие обработки кнопки «Добавить», код которого представлен в листинге 5.

Листинг 5 – Код кнопки «Добавить».

private void Button\_Click(object sender, RoutedEventArgs e)

try

{

Cabinet SpravochnikObj = new Cabinet()

Продолжение листинга 5

{

Number = TxbFio.Text,

Floor = TxbFloor.Text,

Problem = TxbProblem.Text,

Realized = TxbRealized.Text

};

DbConnect.entObj.Cabinet.Add(PersonalObj);

DbConnect.entObj.SaveChanges();

MessageBox.Show("Результат добавлен в базу данных!", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка работы приложения: " + ex.Message.ToString(), "Критический сбой работы приложения", MessageBoxButton.OK, MessageBoxImage.Warning);

}

После перейдем на главную страницу администратора и перейдем по кнопке «Справочник сотрудников» интерфейс которой представлен на рисунке 12.



Рисунок 12 - Интерфейс страницы «Справочник сотрудников»

Затем рассмотрим фрагмент разметки страницы «Справочник сотрудников», который на рисунке 13.

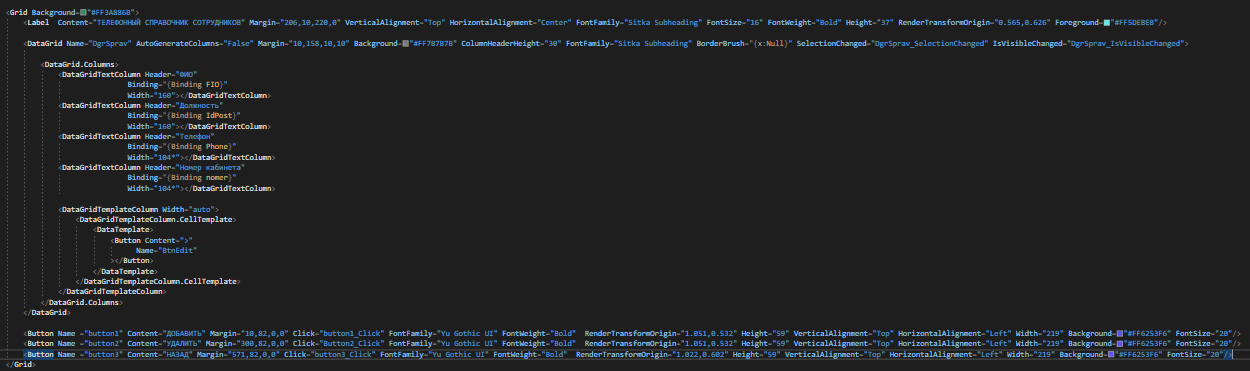


Рисунок 13 – Фрагмент разметки страницы «Справочник сотрудников»

Перейдем по кнопке «Добавить» интерфейс которой представлен на рисунке 14.

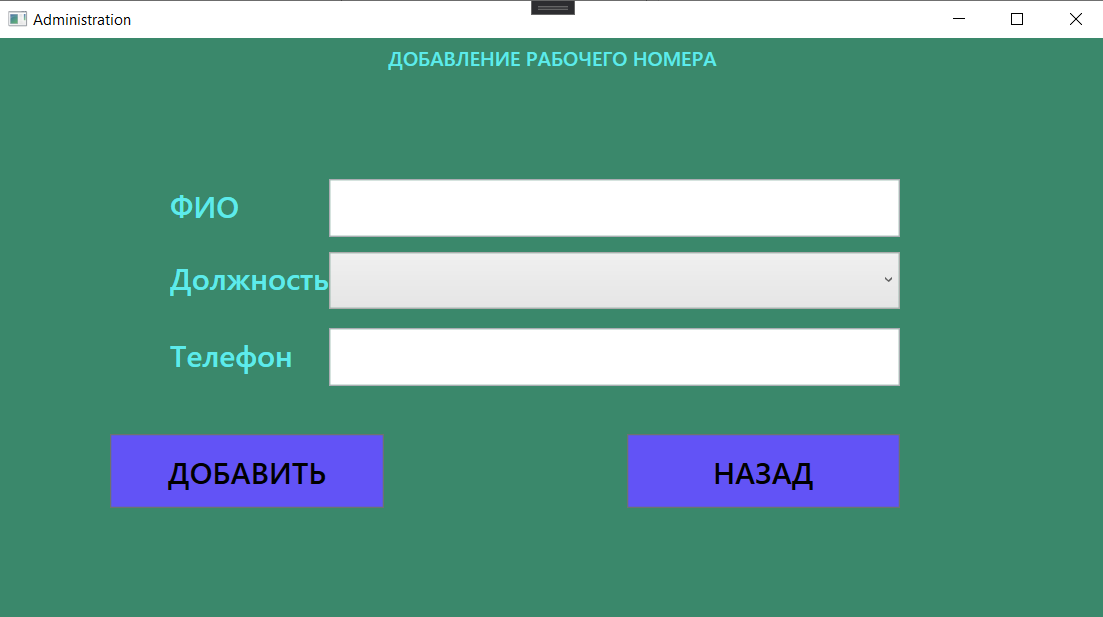


Рисунок 14 – Интерфейс страницы «Добавление номера»

После рассмотрим фрагмент разметки страницы «Добавление номера» на рисунке 15.



Рисунок 15 – Фрагмент разметки страницы «Добавление номра»

Теперь перейдем на страницу работника, который может только создавать и редактировать свои заявки на исправление неисправности представленную на рисунке 16.

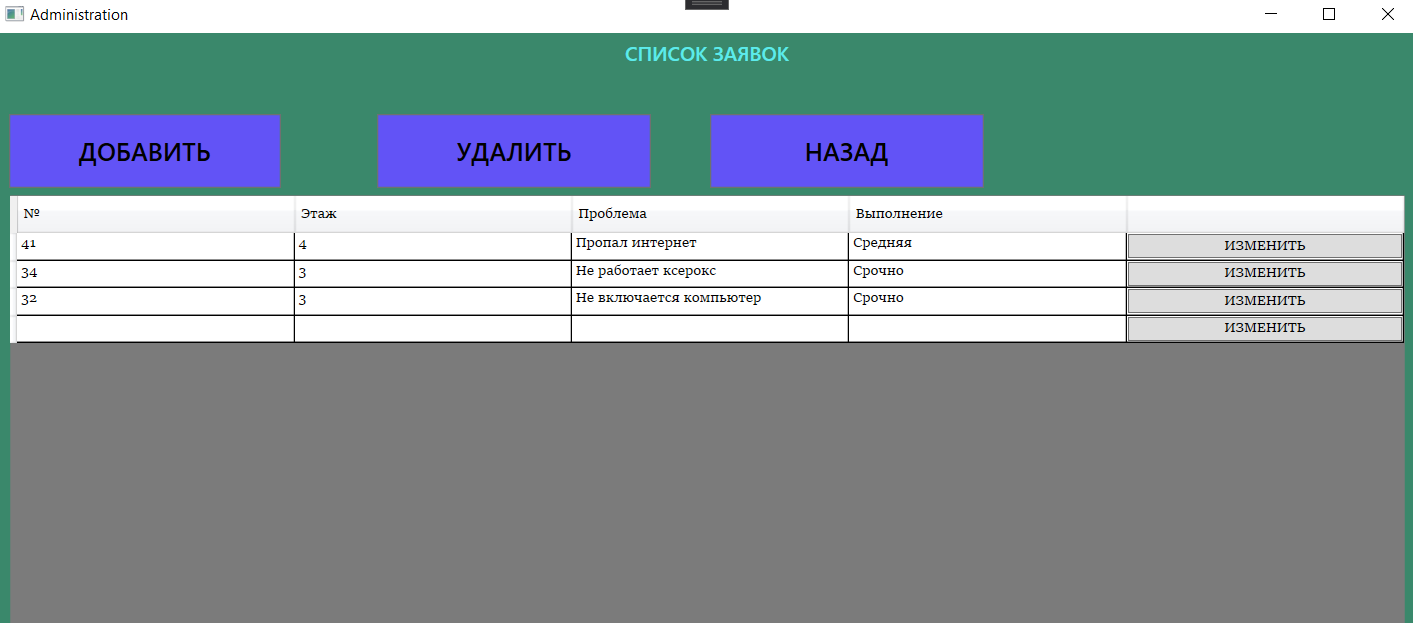


Рисунок 16 - Интерфейс страницы работника

После рассмотрим фрагмент разметки работника показана на рисунке 17.

На этой странице используются такие элементы управления, как DataGrid, DataTemplate, Button.

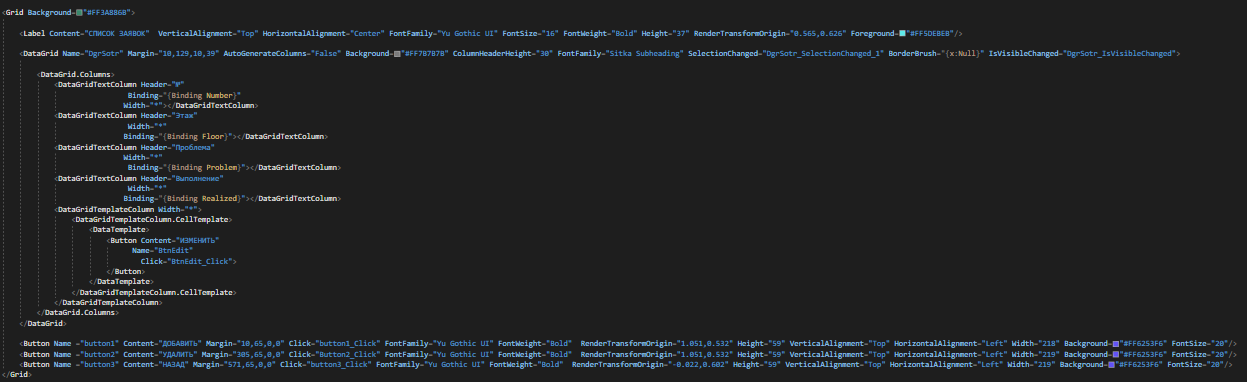


Рисунок 17 – Фрагмент разметки работника

В данной работе была продемонстрирована разработка пользовательского интерфейса информационной системы «МКУ «СКО» г. Арзамаса» с использованием платформы Visual Studio и среды Windows Presentation Foundation (WPF). Были созданы ключевые элементы интерфейса, такие как главная форма приложения и форма авторизации, а также реализованы механизмы взаимодействия с базой данных для авторизации пользователей. Разработка включала в себя проектирование архитектуры папок, создание навигационного окна для администратора и реализацию различных страниц и кнопок для выполнения соответствующих функций. Все компоненты интерфейса оформлены в деловом стиле, обеспечивая удобство и функциональность для пользователей различных ролей в системе.

2.4. Программная логика

Программная логика программного продукта «ADMIN» является ядром системы и отвечает за обработку бизнес-логики, управление данными и обеспечение взаимодействия между компонентами системы. В этом разделе рассмотрим основные аспекты программной логики, используемые подходы и алгоритмы, а также ключевые функции и методы.

Основные компоненты программной логики. Программная логика включает несколько ключевых компонентов:

1. Обработка данных
2. Бизнес-логика
3. Взаимодействие с базой данных
4. Управление устройствами
5. Генерация отчетов

1. Обработка данных

Основные задачи обработки данных включают сбор, валидацию, трансформацию и хранение данных. Обработка данных начинается с получения данных от различных источников, включая устройства, пользовательские вводы и внешние системы.

Пример алгоритма обработки данных:

1. Сбор данных: Система собирает данные с устройств с помощью протоколов SNMP и WMI.
2. Валидация данных: Проверка корректности и целостности данных.
3. Трансформация данных: Преобразование данных в необходимый формат для дальнейшей обработки.
4. Хранение данных: Сохранение данных в базе данных MSSQL.

2. Бизнес-логика

Бизнес-логика включает реализацию основных функций и процессов, которые обеспечивают выполнение задач системы. Это может включать диагностику неполадок, управление устройствами и генерацию отчетов[17, c.54].

Пример алгоритма диагностики неполадок:

1. Сбор логов и событий: Система собирает логи и события с устройств.
2. Анализ данных: Логи и события анализируются для выявления отклонений от нормального состояния.
3. Выявление проблемы: На основе анализа данных определяется тип проблемы (например, сбой оборудования, проблема с сетью).
4. Предоставление рекомендаций: Система генерирует рекомендации по устранению проблемы, включая конкретные шаги и ссылки на соответствующие инструкции.

3. Взаимодействие с базой данных

Программная логика включает методы для взаимодействия с базой данных, включая чтение, запись и обновление данных. Для этого используется Entity Framework, который обеспечивает удобный и эффективный доступ к данным через объектно-ориентированные модели.

Пример метода для взаимодействия с базой данных представлен в листинге 6.

Листинг 6 – Пример реализации метода взаимодействия с базой данных

public List<Device> GetAllDevices()

{

using (var context = new DatabaseContext())

{

return context.Devices.ToList();

}

}

4. Управление устройствами.

Управление устройствами включает функции для удаленного управления устройствами, включая перезагрузку, обновление программного обеспечения и изменение конфигураций.

Программная логика для управления устройствами также включает следующие функции:

1. Обновление программного обеспечения: функция, которая позволяет автоматически обновлять прошивку и программное обеспечение устройств.
2. Изменение конфигурации: функция, которая позволяет изменять параметры конфигурации устройств удаленно.
3. Мониторинг состояния: функция, которая постоянно отслеживает состояние устройств и сообщает об изменениях или проблемах в реальном времени.

5. Генерация отчетов.

Генерация отчетов включает создание и экспорт отчетов о состоянии ИТ-инфраструктуры, анализе производительности и выявлении тенденций и аномалий.

Программная логика для генерации отчетов включает следующие аспекты:

1. Агрегация данных: сбор данных из различных источников, включая логи устройств, события системы и пользовательские вводы.
2. Анализ данных: использование алгоритмов анализа данных для выявления тенденций, аномалий и проблем в работе ИТ-инфраструктуры.
3. Формирование отчетов: создание отчетов в различных форматах (PDF, Excel, HTML) для удобного представления и анализа данных.
4. Планирование отчетов: возможность автоматического планирования и генерации отчетов по заданному расписанию.

Интеграция и взаимодействие. Программная логика системы тесно интегрирована с другими компонентами системы, обеспечивая их взаимодействие и координацию. Для этого используются различные интерфейсы и протоколы:

1. API: реализация RESTful API для взаимодействия между клиентской и серверной частями системы.
2. Событийная модель: использование событийной модели для обработки и уведомления о различных событиях в системе.
3. Системные службы: использование системных служб для выполнения фоновых задач, таких как мониторинг состояния устройств и генерация отчетов.

ЗАКЛЮЧЕНИЕ

Разработанный программный продукт для МКУ «СКО» г. Арзамаса представляет собой современное решение для автоматизации процессов диагностики и устранения неполадок в ИТ-инфраструктуре. В ходе выполнения дипломного проекта были достигнуты следующие результаты:

1. Проведен анализ предметной области.
2. Определены требования к программному продукту.
3. Сформированы функциональные и нефункциональные требования.
4. Произведены моделирование на основе собранных данных.
5. Определены технологии разработки программного продукта.
6. Разработаны архитектуру программного продукта.
7. Разработаны пользовательский интерфейс программного продукта.
8. Реализованы программный продукт на основе разработанной архитектуры и интерфейса.

Программный продукт позволяет значительно сократить время реакции на проблемы, повысить точность диагностики, снизить трудозатраты системных администраторов и обеспечить надежность и безопасность данных. Внедрение системы в МКУ «СКО» г. Арзамаса повысит общую эффективность работы учреждения и обеспечит стабильную и безопасную работу его ИТ-инфраструктуры.

Внедрение программного продукта оказывает значительное влияние на повышение эффективности работы учреждения. Системные администраторы получают возможность автоматизировать рутинные задачи, что освобождает время для выполнения более сложных и стратегически важных задач. Автоматическое обнаружение и диагностика проблем позволяет оперативно реагировать на неполадки, снижая время простоя оборудования и обеспечивая бесперебойную работу всех систем.

Программный продукт обеспечивает высокую точность диагностики благодаря использованию современных алгоритмов и методов анализа данных. Система собирает и анализирует данные в реальном времени, выявляя отклонения от нормы и предоставляя рекомендации по их устранению. Это позволяет значительно повысить надежность ИТ-инфраструктуры и снизить вероятность возникновения критических сбоев.

Автоматизация процессов управления ИТ-инфраструктурой позволяет значительно сократить трудозатраты и ресурсы, необходимые для поддержания системы в рабочем состоянии. Системные администраторы могут уделять больше времени на стратегическое планирование и оптимизацию инфраструктуры, что в конечном итоге приводит к улучшению качества обслуживания и повышению удовлетворенности пользователей.

Программный продукт обеспечивает высокий уровень безопасности данных, что является критически важным аспектом для муниципальных учреждений. Использование современных методов шифрования, аутентификации и авторизации обеспечивает защиту данных от несанкционированного доступа и угроз. Регулярное резервное копирование и возможность восстановления данных гарантируют сохранность информации в случае сбоев или атак.

Система спроектирована таким образом, чтобы легко адаптироваться к изменяющимся потребностям учреждения. Масштабируемая архитектура позволяет добавлять новые устройства и расширять функциональность без значительных изменений в коде. Это делает систему гибкой и готовой к будущим изменениям и улучшениям.

Во время тестирования системы была получена ценная обратная связь от пользователей, которая позволила выявить области для улучшения и оптимизации. Внедрение новых функций и доработка существующих компонентов будут продолжаться на основе этой обратной связи, что позволит сделать систему еще более эффективной и удобной в использовании.

Разработанный программный продукт может быть адаптирован для использования в других муниципальных учреждениях и организациях. Гибкость и масштабируемость системы позволяют легко настроить ее под специфические требования и условия работы различных организаций. Это открывает возможности для широкого применения и распространения системы, что в свою очередь повысит общую эффективность и надежность ИТ-инфраструктуры в различных учреждениях.

В заключение, разработанный программный продукт отвечает современным требованиям и стандартам, предоставляет широкие возможности для автоматизации процессов управления ИТ-инфраструктурой и может быть адаптирован для использования в других муниципальных учреждениях и организациях. Внедрение данной системы в МКУ «СКО» г. Арзамаса уже показало значительные улучшения в эффективности работы, точности диагностики и надежности ИТ-инфраструктуры. Дальнейшее развитие и оптимизация системы на основе обратной связи от пользователей обеспечат ее успешное использование и в будущем.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ 19.201-78. Техническое задание. Требования к содержанию и оформлению.
2. ГОСТ 19.504-78. Руководство программиста. Требования к содержанию и оформлению.
3. Алан Бьюли Изучаем SQL / Алан Бьюли. - М.: Символ-плюс, 2022. — 851 c. —ISBN—9785907365544
4. Арсеньев, Б. П. Интеграция распределенных баз данных / Б.П. Арсеньев, С. А Яковлев. - Москва: Гостехиздат, 2019. - 464 c. — ISBN—5-8114-0321-6
5. Астахова, И. Ф., Объектные базы данных : учебное пособие / И. Ф. Астахова, Д. В. Борисенков, Е. И. Киселева, Н. К. Самойлов. — Москва : Русайнс, 2022. — 93 с. — ISBN 978-5-4365-9779-9.
6. Гордиенко, А. П., Функциональное программирование : учебник / А. П. Гордиенко. — Москва : КноРус, 2022. — 277 с. — ISBN 978-5-406-08432-8.
7. Грекул, В. И. Проектирование информационных систем : учебное пособие / В. И. Грекул, Г. Н. Денищенко, Н. Л. Коровкина. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 299 c. — ISBN 978-5-4497-0689-8.
8. Грекул, В. И. Управление внедрением информационных систем : учебное пособие для СПО / В. И. Грекул, Г. Н. Денищенко, Н. Л. Коровкина. — Саратов : Профобразование, 2021. — 277 c. — ISBN 978-5-4488-1016-9.
9. Зиангирова, Л. Ф. Инфокоммуникационные системы и сети : учебное пособие для СПО / Л. Ф. Зиангирова. — Саратов : Профобразование, Ай Пи Ар Медиа, 2019. — 128 c. — ISBN 978-5-4488-0302-4
10. Иванова, Г. С., Программирование. : учебник / Г. С. Иванова. — Москва : КноРус, 2022. — 426 с. — ISBN 978-5-406-09829-5.
11. Киселева, Т.В.. Проектирование информационных систем. Часть I : Учебное пособие / Т.В. Киселева — Ставрополь : Северо-Кавказский федеральный университет, 2019. — 150 с. — ISBN**:**9785534159233
12. Козлов, А. Д. Методы анализа предметных областей : 2019-12-06 / А. Д. Козлов, В. А. Лекае, М. С. Шаповалова. — 3-е изд. (эл.). — Москва : РГГУ, 2019. — 203 с. — ISBN 978-5-7281-2489-4.
13. Котляров, В. П. Основы тестирования программного обеспечения : учебное пособие для СПО / В. П. Котляров. — Саратов: Профобразование, 2019. — 335 с. - ISBN 978-5-4488-0364-2.
14. Кудинов, Ю. И. Интеллектуальные информационные системы : учебное пособие для СПО / Ю. И. Кудинов. — 2-е изд. — Липецк, Саратов : Липецкий государственный технический университет, Профобразование, 2020. — 63 c. — ISBN 978-5-88247-961-8,
15. Мижгородская И.А. Информатика: Технология создания и преобразования информационных объектов. Практикум : учебное пособие / Мижгородская И.А. — Москва : Русайнс, 2022. — 146 с. — ISBN 978-5-4365-1352-2.
16. Молдованова, О. В. Информационные системы и базы данных : учебное пособие для СПО / О. В. Молдованова. — Саратов : Профобразование, 2021. — 177 c. — ISBN 978-5-4488-1177-7.
17. Н. К. Лисяк, В. В. Лисяк Моделирование информационных систем на платформе SOFTWARE IDEAS MODELER, 2019. – 90 с. – ISBN 978-5-16-107692-7.
18. Кунал Чоудхури Освоение Visual Studio 2019: платформы .NET Framework и .NET Core — Packt Publishing, 2019. — 354 с. —ISBN-13 ‏ : ‎ 978-1789530094.
19. Проектирование информационных систем : учебник и практикум для среднего профессионального образования / Д. В. Чистов, П. П. Мельников, А. В. Золотарюк, Н. Б. Ничепорук. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2023. — 293 с. — (Профессиональное образование). — ISBN 978-5-534-16217-2.
20. Скороход, С. В. Программирование на платформе 1С: Предприятие 8.3 : учебное пособие / С. В. Скороход. — Ростов-на-Дону, Таганрог : Издательство Южного федерального университета, 2019. — 135 c. — ISBN 978-5-9275-3315-2.
21. Спицина И. А. Системный анализ и моделирование информационных систем : учебное пособие : Рекомендовано методическим советом Уральского федерального университета для студентов вуза, обучающихся по направлению подготовки 09.04.01 — Информатика и вычислительная техника / И. А. Спицина, К. А. Аксенов ; научный редактор Л. Г. Доросинский ; Министерство науки и высшего образования Российской Федерации, Уральский федеральный университет имени первого Президента России Б. Н. Ельцина. — Екатеринбург : Издательство Уральского университета, 2021. — 118 с. — ISBN 978-5-7996-3196-3.
22. Флегонтов, А. В. Моделирование информационных систем. Unified Modeling Language : учебное пособие / А. В. Флегонтов, И. Ю. Матюшичев. — 2-е изд., стер. — Санкт-Петербург : Лань, 2022. — 112 с. — ISBN 978-5-8114-2907-3.
23. Боб Уорд., Инновации SQL Server 2019. Использование технологий больших данных и машинного обучения— Издательство: Дмк Пресс 2020— 408— ISBN : 978-5-98060-695-0
24. Чулюков, В. А., Проектирование баз данных. Практический курс : учебное пособие / В. А. Чулюков, И. Ф. Астахова, С. О. Башарина, О. А. Сидорова. — Москва : Русайнс, 2022. — 163 с. — ISBN 978-5-4365-9130-8.