

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Программные средства удаленного
управления конфигурацией сетевых
устройств

Студент _____ Шахманов Данила Дмитриевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Иванчуков А. Г.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

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

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

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

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

Пенза, 2025

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

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

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Шахманов Данила Дмитриевич гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства удаленного управления конфигурацией сетевых устройств

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

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» А. Г. Иванчуков

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

1) Разработать систему удаленного управления конфигурацией сетевых устройств

2) Среда выполнения системы: ОС Linux Debian 11

Языковые средства разработки: C#, UML

Инструментальные средства разработки: Microsoft Visual Studio, Apache Thrift, SQLite, ProjectLibre

Технологии разработки: Entity Framework, ООП

3) Основные функции системы:

а) добавление новых устройств в систему;

б) удаление устройств из системы;

в) удаленное управление конфигурацией устройств;

г) разрешение конфликтов при наличии локальных и удаленных изменений;

д) просмотр конфигураций устройств.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование

3) Реализация

4) Тестирование программного обеспечения

5) Оформление пояснительной записки

5. Календарный график выполнения работы

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

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 94 листа, 23 рисунка, 9 таблиц, 20 использованных источников, 4 приложения.

КОНФИГУРАЦИЯ, СЕТЕВОЕ УСТРОЙСТВО, УДАЛЕННОЕ УПРАВЛЕНИЕ, SQLITE, C#, THRIFT, UML

Объектом разработки является программные средства удаленного управления конфигурацией сетевых устройств.

Цель работы – разработка программных средств, позволяющих удаленно управлять конфигурацией сетевых устройств, а также добавлять новые устройства для управления их конфигурацией, и удалять эти устройства.

Разработка программных средств осуществлялась на языке программирования C# с использованием языка описания интерфейсов Thrift. Для реализации программных средств использовалась среда разработки Visual Studio 2022. В качестве СУБД использовалась SQLite.

Программные средства предназначены для работы на устройствах под управлением ОС Linux Debian 11 и выше.

					ПГУ 09.03.04 – 08ВП211.29 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.		Шахманов Д.Д.			Программные средства удаленного управления конфигурацией сетевых устройств. Пояснительная записка	Лит.	Лист	Листов
Пров.		Иванчуков А.Г.					3	94
Реценз.						Группа 21ВП1		
Н. контр.		Такташкин Д.В.						
Утверд.		Макарычев						

Содержание

Введение	7
1 Анализ предметной области и требований	9
1.1 Анализ предметной области и постановка задачи	9
1.2 Анализ доступных решений	10
1.2.1 Инструмент «Ansible»	10
1.2.2 Инструмент «Puppet»	12
1.2.3 Инструмент «Chef»	14
1.2.4 Инструмент «SaltStack»	15
1.3 Функционально-стоимостной анализ	16
2 Анализ требований к разрабатываемым средствам	19
2.1 Анализ функциональных требований	19
2.2 Анализ нефункциональных требований	26
3 Проектирование системы	28
3.1 Проектирование модуля конфигурации	33
3.2 Проектирование модуля управления	37
3.3 Проектирование модуля редактирования	42
4 Реализация системы	50
4.1 Реализация модуля конфигурации	54
4.2 Реализация модуля управления	56
4.3 Реализация модуля редактирования	60
5 Тестирование системы	65
5.1 Функциональное тестирование системы	65
5.2 Модульное тестирование	68
5.3 Интеграционное тестирование	71
5.4 Метрики качества кода	72
Заключение	76
Список использованных источников	77
Приложение А (справочное) Глоссарий	79
Приложение Б (обязательное) Исходный код приложения	81

Приложение В (рекомендуемое) Исходный код интеграционных тестов	87
Приложение Г (рекомендуемое) Акт о внедрении	93

Введение

Современная инфраструктура сетей представляет собой сложную систему взаимодействующих устройств, которая обеспечивает обмен данными между этими устройствами, обработку данных и их хранение. Из года в год системы, представляющие сетевую инфраструктуру, усложняются, в том числе за счет постоянного роста подключенных устройств, соответственно усложняется и управление их настройками и состоянием, формирующих конфигурацию этих устройств.

Конфигурация определяет поведение устройства. Ошибки, нарушения согласованности при ее изменении, несвоевременное применение изменений или конфликт между локальными и удаленными изменениями могут привести к сбоям работы сетевого устройства или всей инфраструктуры в целом, а также создать уязвимости для системы.

Разрабатываемая система предоставляет подсистему удаленного управления конфигурацией сетевых устройств. В качестве дополнительных требований, предъявляемых к разрабатываемым программным средствам – обеспечение универсальности в части возможности работы с конфигурацией любого вида, без привязки к каким-либо форматам данных. Разрабатываемая подсистема также должна обеспечивать атомарность применения конфигурации на удаленное устройство и обеспечивать возможность быстрого встраивания или поддержания сетевых устройств с ранее не поддерживаемым видом конфигурации.

Актуальность разработки заключается в необходимости разработки единой универсальной подсистемы доставки конфигурации до сетевых устройств при выполнении операции централизованного управления.

Универсальность подсистемы связана с необходимостью применения механизма централизованного управления в рамках нескольких подключенных устройств.

В ходе выполнения выпускной квалификационной работы планируется выполнение следующих задач:

- анализ предметной области;
- разработка требований ко всем модулям системы;
- выполнение функционально-стоимостного анализа разработки системы;
- проектирование архитектуры модулей и интерфейсов их взаимодействия;
- разработка и тестирование систем.

Целью выполнения выпускной квалификационной работы является разработка программных средств удаленного управления конфигурацией сетевых устройств.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

В ходе выполнения выпускной квалификационной работы необходимо разработать программные средства удаленного управления конфигурацией сетевых устройств, обеспечивающих возможность удаленного редактирования конфигурации конкретного подключенного устройства и сохранения изменений.

Удаленное управление конфигурацией сетевых устройств представляет из себя процесс модификации каких-либо параметров в настройках сетевого устройства с помощью отправки данных по сети, не требующей прямого физического подключения к сетевому устройству.

Сетевое устройство является элементом, участвующим в обработке или передаче данных в компьютерной сети.

Конфигурация сетевого устройства представляет из себя набор параметров и настроек, который определяет состояние и поведение данного сетевого устройства.

Разрабатываемое решение должно предоставлять единый интерфейс для изменения конфигураций всех подключенных устройств, а также учитывать возможность прямого локального изменения конфигурации на устройстве для корректного ее применения после удаленного изменения.

Единой точкой подключения сетевых устройств для управления их конфигурациями будет выступать управляющее устройство.

На основе проведенного анализа предметной области и существующих решений необходимо выполнить следующие задачи:

- определить требования к системе удаленного управления конфигурацией сетевых устройств с учетом анализа предметной области;
- спроектировать архитектуру системы;
- реализовать необходимые механизмы и функции для корректной работы системы;

- протестировать работоспособность решения.

Разрабатываемая система должна обеспечивать:

- возможность подключения к управляющему устройству новых устройств для изменения их конфигураций, а также возможность отключения подключенных устройств;
- единый удобный интерфейс для удаленного атомарного изменения конфигураций всех подключенных устройств;
- предоставление данных по конфигурациям всех подключенных устройств;
- возможность локального изменения конфигураций отдельных устройств, и обеспечение разрешение конфликтов при наличии удаленных и локальных изменений перед применением;
- универсальность в части возможности работы с конфигурацией любого вида, без привязки к каким-либо форматам данных;
- атомарность применения конфигурации на устройстве;
- быструю поддержку сетевых устройств с ранее не поддерживаемым видом конфигурации.

В результате разработки должна быть создана система для удаленного управления конфигурациями подключенных к системе устройств.

Цель разрабатываемого решения – обеспечить единый интерфейс для удаленного управления конфигурацией подключенных сетевых устройств

1.2 Анализ доступных решений

В соответствии с поставленной задачей был осуществлен анализ наиболее известных инструментов удаленного управления конфигурацией сетевых устройств. Далее приведен обзор некоторых из них.

1.2.1 Инструмент «Ansible»

Ansible является системой для управления устройствами, которая позволяет заранее описать желаемое состояние устройства с заданными

параметрами, и затем автоматически привести каждое подключенное к управляющему устройству сетевое устройство к этому состоянию [1].

В Ansible управляющее устройство инициирует выполнение операций на подключенных управляемых устройствах, которым не требуется устанавливать какие-либо специальные программы для получения конфигураций от управляющего устройства, все управление осуществляется через управляющее устройство [2].

С точки зрения разработки централизованной системы удаленного управления конфигурацией сетевых устройств, Ansible имеет преимущества в виде широкого распространения, надежности работы, а также предоставляет удобный способ описания конфигураций, который легко читается и поддерживается. Он является подходящим выбором в тех случаях, когда требуется настроить одинаковые параметры на множестве сетевых устройств без специфических требований.

При применении Ansible для удаленного управления конфигурацией сетевых устройств в более специализированных системах, можно выделить ряд ограничений.

Ansible не предоставляет встроенного механизма взаимодействия с внутренними компонентами сетевых устройств, то есть не предназначен для отправки данных во внутренние механизмы отдельных программ на устройстве, исходя из этой особенности требуется самостоятельная доработка системы, чтобы адаптировать ее под конкретные службы на управляемом устройстве. Также Ansible использует один определенный формат описания конфигурации на языке YAML, который может быть не всегда удобен для передачи в сервисы, ожидающие данные в других форматах.

Возможность изменения структуры конфигурации присутствует, но не проста в реализации, что может стать проблемой в системах, где формат конфигурации должен быть гибким и легко настраиваемым.

Ansible предоставляет базовые средства защиты передаваемых данных, но не предусматривает использование специализированных протоколов защиты каналов передачи данных, например, таких как IPsec.

Также Ansible ограничен отсутствием механизма отслеживания возможных конфликтов при удаленном и локальном изменении конфигурации сетевого устройства. Если устройство было настроено локально, в обход центрального управляющего устройства, Ansible не уведомляет о расхождениях и просто применяет новую конфигурацию, стирая при этом предыдущие изменения. Это затрудняет контроль над реальным состоянием системы и может привести к потере важных локальных настроек.

Таким образом, Ansible может использоваться как основа по удаленному управлению конфигурацией сетевых устройств, но не отвечает требованиям, предъявленным к разрабатываемой системе, которая должна поддерживать изменяемые форматы конфигураций, обеспечивать надежную защищенную передачу данных и контролировать состояние устройств с учетом возможных изменений, сделанных локально на самих устройствах и на управляемом устройстве.

1.2.2 Инструмент «Puppet»

Puppet является системой, созданной для автоматической настройки устройств и серверов [3]. В ее основе лежит идея, что все параметры и настройки можно описать в виде понятного и структурированного файла, который затем передается на устройства и применяется там автоматически.

Настройки хранятся в специализированном формате, который читается центральным управляющим устройством. Управляющие устройства регулярно запрашивают эти описания и приводят себя в соответствие с ними. Это помогает добиться единообразия и стабильности во всей системе.

Все параметры задаются в виде читаемого текста, в котором описан не алгоритм выполнения какого-либо действия, а какое состояние должно быть

достигнуто. Если некоторое устройство отклоняется от заданного состояния, то это обнаруживается, устраняется и фиксируется в отчетах.

Система подходит для управления сотнями и тысячами устройств одновременно, благодаря централизованному подходу и повторяемости операций [4].

Puppet передает конфигурацию в виде файлов или стандартных параметров, но не умеет напрямую направлять ее в интерфейсы конкретных служб на управляемом устройстве.

Также стоит отметить, что язык Puppet хоть и удобен, но он не дает свободы в выборе и легкой смене форматов. Для адаптации нужно создавать дополнительный код.

Связь между управляющим устройством и управляемыми устройствами защищена сертификатами, но не гарантирует передачу через строго настроенный безопасный канал, такой каким является IPsec.

Обработка конфликтов применения конфигураций ограничена стандартными сценариями, Puppet исправляет расхождения локальных изменений, но не предоставляет инструментов для взаимодействия с такими конфликтами. Если изменения сделаны локально перед применением новой конфигурации, они просто будут перезаписаны.

Также устройства сами проверяют, нужна ли им новая конфигурация, что не является полностью контролируемым централизованным процессом, где управляющее устройство инициирует рассылку.

Puppet является надежным инструментом для синхронизации большого количества устройств, с понятным описанием состояния и системой ведения отчетов по изменениям, но не удовлетворяет ключевым условиям архитектуры разрабатываемой системы и не может быть использован без серьезной модификации.

1.2.3 Инструмент «Chef»

Chef является системой управления конфигурациями, которая позволяет описывать желаемое состояние компьютеров и устройств с помощью так называемых рецептов и коллекций рецептов. Данная система предполагает хранение этих описаний на центральном управляющем устройстве и их периодическую загрузку на управляемые устройства.

Chef может работать одновременно с множеством устройств, что делает его удобным для крупных инфраструктур. Помимо этого, существует широкое сообщество, где пользователи публикуют готовые наборы настроек, ускоряющие процесс внедрения [5].

Стоит отметить, что Chef не знает, какие внутренние службы нужно настроить. Он лишь загружает файлы и запускает команды, поэтому точная отправка конфигурации в обработчики служб потребует дополнительных усилий.

Формат конфигурации не всегда гибкий, они хранятся в собственном формате Chef. Чтобы использовать, например, json или другой формат, потребуются посредники или преобразователи. В Chef нет встроенного защищенного канала вроде IPsec, если требуется уровень защиты, предусмотренный системой, его нужно организовать внешне, что добавляет сложности и может снизить контроль над безопасностью. Когда на устройстве вручную меняют настройки, Chef не фиксирует эти изменения и не ставит их в известность – он просто возвращает прежнее состояние, что может привести к потере важных данных.

Chef предлагает концепцию описания конфигураций в удобном и читаемом виде, подходящую для крупных систем, но не обеспечивает механизмы обработки конфликтов при локальных и удаленных изменениях конфигураций.

1.2.4 Инструмент «SaltStack»

SaltStack является платформой для удаленного управления устройствами, разработанной для высокой скорости, гибкости управления и масштабируемости. Она состоит из центрального управляющего устройства, отправляющего команды, и множества подключенных устройств, принимающих и исполняющих их [6].

Быстрая массовая рассылка позволяет отправлять одну и ту же настройку сразу на множество устройств, причем это делается эффективно и одновременно. Описания состояния описываются понятным текстовым способом и могут быть написаны в распространенных форматах, а также легко меняться.

SaltStack работает на уровне файлов или команд, но не направляет содержимое прямо в конкретную внутреннюю программу или службу на устройстве, соответственно требует расширения функционала системы [7].

Связь в системе защищена на уровне базовых протоколов, специальный туннель IPsec не является частью системы, и его нужно настраивать отдельно.

SaltStack проверяет и исправляет отклонения от заданной конфигурации, но не уведомляет и не сохраняет изменения, введенные напрямую на устройствах, что может привести к потере локальных правок.

Также чтобы направить настройку конкретной службе, нужно писать дополнительные скрипты или модули, которые преобразуют общий формат в специфичный для каждой службы.

Таким образом SaltStack подходит для быстрой и масштабируемой передачи настроек в однотипные системы, но не может направлять данные прямо во внутренние службы устройств без доработок, не обеспечивает защиту с помощью IPsec для передачи данных, не отслеживает и не решает конфликты локальных изменений, не гарантирует гибкую смену форматов, следовательно, он не удовлетворяет ключевым требованиям разрабатываемой системы.

1.3 Функционально-стоимостной анализ

Необходимо провести функционально-стоимостной анализ для определения стоимости и целесообразности разработки программного продукта [8]. Для этого определяем длительность разработки продукта и стоимость этапов разработки. Для определения стоимости разработки системы удаленного управления конфигурацией сетевых устройств был проведен анализ уровня заработной платы для работников сферы информационных технологий в 2025 году в городе Пенза. График приведен на рисунке 1.



Рисунок 1 – График заработной платы для работников сферы информационных технологий в 2025 году в городе Пенза

Исходя из графика получена средняя предлагаемая зарплата для работника сферы информационных технологий при 40-часовой рабочей недели 59845 рублей в месяц, что приблизительно составляет 374 рубля за каждый час работы.

Был составлен план разработки проекта для разбиения на этапы и учета стоимости выполнения каждого этапа, представленный на рисунке 2.

Название	Продолжи...	Начало	Окончание
☐ разработка системы удаленного управления конфигурацией	41 дней	08.04.25 8:00	03.06.25 17:00
анализ предметной области	3 дней	08.04.25 8:00	10.04.25 17:00
постановка задачи на разработку	2 дней	11.04.25 8:00	14.04.25 17:00
проектирование системы	9 дней	15.04.25 8:00	25.04.25 17:00
☐ реализация системы	11 дней	28.04.25 8:00	12.05.25 17:00
реализация ПО системы	10 дней	28.04.25 8:00	09.05.25 17:00
настройка взаимодействия модулей	1 день	12.05.25 8:00	12.05.25 17:00
тестирование системы	8 дней	13.05.25 8:00	22.05.25 17:00
подготовка документации	8 дней	23.05.25 8:00	03.06.25 17:00

Рисунок 2 – План разработки проекта

Для наглядного представления хода разработки проекта была создана диаграмма Ганта, представленная на рисунке 3.

Диаграмма Ганта представляет собой график для планирования и управления проектами, который показывает поставленные задачи и время, отведенное на их выполнение. Данная диаграмма подходит для планирования как небольших, так и масштабных проектов [9].

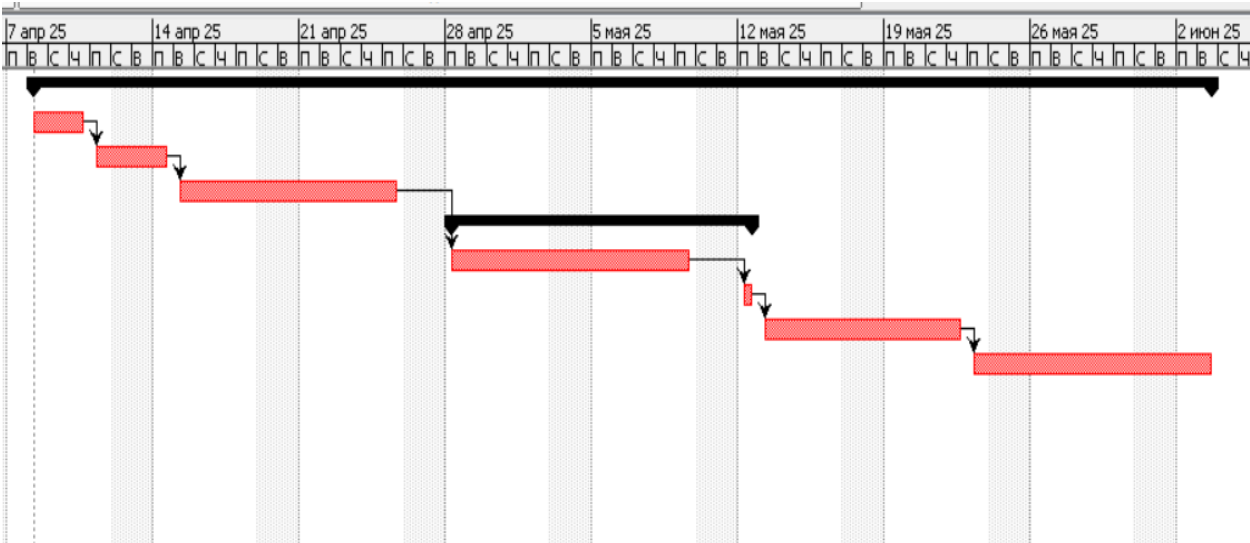


Рисунок 3 – Диаграмма Ганта

Для оценки стоимости проекта было проанализировано время на выполнение всех этапов разработки проекта и составлена таблица 1, показывающая ориентированную стоимость каждого этапа и его продолжительность, а также подведение итогов по денежным и трудовым затратам [10].

Таблица 1 – Денежные и трудовые затраты на реализацию проекта

Этап	Продолжительность	Зарплата	Стоимость
Анализ предметной области	24 ч	374 руб/ч	8 976 руб
Постановка задачи на разработку	16 ч	374 руб/ч	5 984 руб
Проектирование системы	72 ч	374 руб/ч	26 928 руб
Реализация ПО системы	80 ч	374 руб/ч	29 920 руб
Настройка взаимодействия компонентов	8 ч	374 руб/ч	2 992 руб
Тестирование системы	64 ч	374 руб/ч	23 936 руб
Подготовка документации	64 ч	374 руб/ч	23 936 руб

Проведя функционально-стоимостной анализ получили, что суммарные временные затраты на разработку системы удаленного управления конфигурацией сетевых устройств с оформлением документации составляют 328 часов, и стоимость их выполнения составляет 122 672 рублей.

2 Анализ требований к разрабатываемым средствам

2.1 Анализ функциональных требований

Функциональные требования описывают поведение разрабатываемой системы и определяют какие действия она должна выполнять в тех или иных случаях, а также определяют особенности взаимодействия пользователя и системы, описывая какие функции и сервисы должны быть реализованы для достижения целей, поставленных при разработке.

Функциональные требования закладывают основу разрабатываемого решения, так как они определяют какие задачи система должна выполнять с точки зрения конечного пользователя.

Четкая и детальная формулировка функциональных требований позволяет обеспечить корректность реализуемого решения и соответствие перечню необходимого функционала системы, тем самым снижая риск неоднозначностей между реализованным и требуемым функционалом.

В рамках данной работы необходимо разработать систему для удаленного управления конфигурацией сетевых устройств:

Далее приведены общие функциональные требования к системе:

- возможность добавления нового сетевого устройства для управления его конфигурацией;
- возможность подключения управляющего устройства к управляемому сетевому устройству с помощью защищенного соединения для взаимодействия с этим сетевым устройством;
- возможность отправлять данные конфигураций с внесенными изменениями для применения на подключенные сетевые устройства;
- гарантированная доставка конфигурации на сетевое устройство с управляющего устройства;
- возможность получать установленные на данный момент конфигурации с подключенных к управляющему устройству сетевых устройств;

- работа с конфигурациями сетевых устройств в рамках отдельных сессий, которые позволяют вносить некоторое число изменений в конфигурации устройств, а затем отправлять данные изменения для применения, а также отменять данные изменения при необходимости;
- возможность отмены внесенных изменений, сделанных в рамках текущей сессии, путем отката промежуточных хранимых конфигураций до состояния конфигураций, установленных на устройствах;
- возможность получать промежуточные хранимые конфигурации подключенных устройств, которые содержат внесенные в рамках текущей сессии изменения;
- предоставление функционала редактирования конфигураций с помощью атомарных действий, без необходимости работы с прямым редактированием полной конфигурации;
- возможность локального изменения конфигураций отдельных устройств, и обеспечение разрешения конфликтов при наличии удаленных и локальных изменений перед применением;
- универсальность в части возможности работы с конфигурацией любого вида, без привязки к каким-либо форматам данных;
- атомарность применения конфигурации на устройстве;
- быструю поддержку сетевых устройств с ранее не поддерживаемым видом конфигурации.

Исходя из описанных функциональных требований была построена диаграмма вариантов использования, представленная на рисунке 4 [11].

Диаграмма вариантов использования нужна для визуального описания взаимодействия между внешними акторами и системой, а также показывает какие функции система должна предоставить, и кто их будет задействовать.

Она обеспечивает понимание требований к системе с точки зрения пользователя и помогает в планировании необходимого функционала [12].

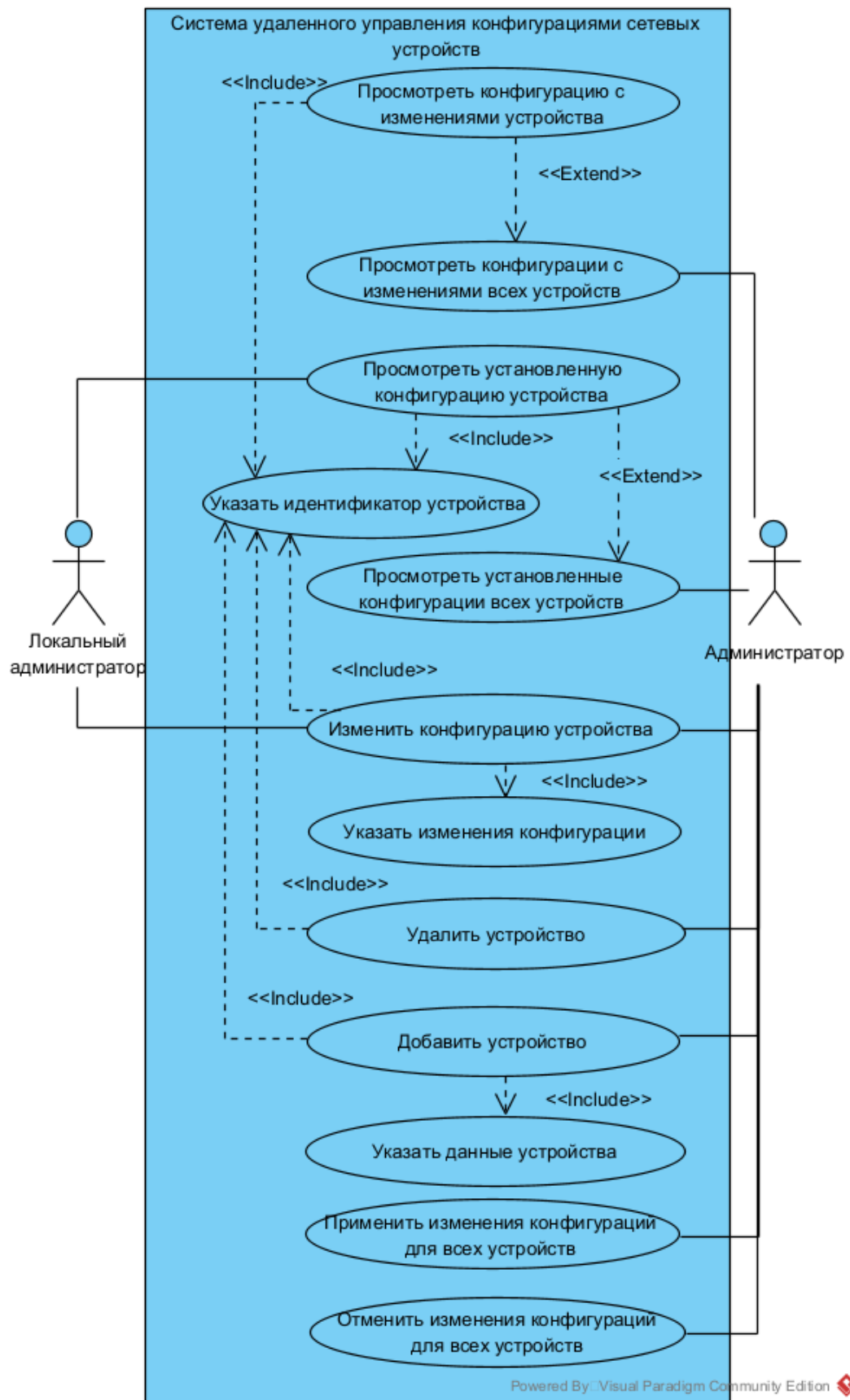


Рисунок 4 – Диаграмма вариантов использования

В таблице 2 приведено описание акторов системы удаленного управления конфигурацией сетевых устройств.

Таблица 2 – Описание акторов системы

Актор	Описание роли
Локальный администратор	Администратор, которому доступен функционал по работе с конфигурацией конкретного устройства. Изменения конфигураций происходит локально.
Администратор	Администратор, которому доступен функционал по работе с конфигурациями всех устройств системы со взаимодействием через управляющее устройство. Изменения конфигураций происходят удаленно.

Таким образом было выделено два актора разрабатываемой системы: локальный администратор, который локально управляет конфигурацией конкретного устройства, а также администратор, который удаленно управляет конфигурациями каждого подключенного к системе устройства.

Далее рассмотрен обзор вариантов использования акторов, подробное описание которых приведено в таблице 3.

Таблица 3 – Описание вариантов использования акторов

Актор	Наименование	Описание
Локальный администратор Администратор	Просмотреть установленную конфигурацию устройства	Администратору предоставляется описание конфигурации конкретного устройства, которая используется в настоящее время
Локальный администратор Администратор	Изменить конфигурацию устройства	Администратор указывает желаемые изменения в конфигурации конкретного устройства в рамках текущей сессии, изменения являются атомарными относительно всего содержимого конфигурации

Продолжение таблицы 3

Актор	Наименование	Описание
Администратор	Просмотреть конфигурации с изменениями всех устройств	Администратору предоставляется описание конфигураций всех подключенных устройств, которые содержат изменения, внесенные в рамках текущей сессии
Администратор	Просмотреть конфигурацию с изменениями устройства	Администратору предоставляется описание конфигурация конкретного устройства, которая содержит изменения, внесенные в рамках текущей сессии
Администратор	Просмотреть установленные конфигурации всех устройств	Администратору предоставляется описание конфигураций всех подключенных устройств, которые используются в настоящее время
Администратор	Добавить устройство	Администратор добавляет новое устройство в систему для управления его конфигурацией.
Администратор	Удалить устройство	Администратор удаляет подключенное устройство из системы для управления его конфигурацией.
Администратор	Применить изменения конфигураций для всех устройств	Администратор инициирует отправку конфигураций с изменениями в рамках текущей сессии для их дальнейшей установки на устройствах
Администратор	Отменить изменения конфигураций для всех устройств	Администратор инициирует отмену внесенных изменений в конфигурации в рамках текущей сессии

Также были описаны сценарии некоторых вариантов использования данной диаграммы.

Сценарий выполнения для варианта использования «Применить изменения конфигураций для всех устройств» представлен на рисунке 5.

Наименование: Применить изменения конфигураций для всех устройств
ID: 1
Краткое описание: Администратор запрашивает применение внесенных изменений в конфигурации на всех устройствах в рамках текущей сессии
Главные акторы: Администратор
Предусловия: Есть подключенные устройства
Основной поток: <ol style="list-style-type: none"> 1. Администратор внес все желаемые изменения в конфигурации устройств и инициирует применение изменений из конфигураций для всех устройств 2. Запрос передается управляющему устройству 3. Обработанные конфигурации отправляются на устройства 4. Устройства обновляют свои конфигурации
Альтернативный поток: <ol style="list-style-type: none"> 1. Администратор не вносил изменения после последнего применения конфигураций и инициирует применение изменений из конфигураций для всех устройств 2. Выполнения применения конфигураций не происходит
Постусловия: Установленные конфигурации на устройствах соответствуют внесенным изменениям, конфигурации на управляющем устройстве соответствуют установленным конфигурациям

Рисунок 5 – Описание варианта использования «Применить изменения конфигураций для всех устройств»

Сценарий выполнения для варианта использования «Отменить изменения конфигураций для всех устройств» представлен на рисунке 6.

Наименование: Отменить изменения конфигураций для всех устройств
ID: 2
Краткое описание: Администратор запрашивает отмену внесенных изменений в конфигурации для всех устройств в рамках текущей сессии
Главные акторы: Администратор
Предусловия: Есть подключенные устройства
Основной поток: <ol style="list-style-type: none"> 1. Администратор инициирует отмену изменений в конфигурациях для всех устройств 2. Запрос передается управляющему устройству 3. Проверяется наличие изменений в конфигурациях 4. При наличии изменений конфигурации откатываются до состояния установленных на устройствах конфигураций
Альтернативный поток: <ol style="list-style-type: none"> 1. Администратор не вносил изменения с момента последнего применения конфигураций и инициирует отмену изменений в конфигурациях для всех устройств 2. Отмены изменений в конфигурациях не происходит
Постусловия: Конфигурации на управляющем устройстве соответствуют установленным конфигурациям на устройствах

Рисунок 6 – Описание варианта использования «Отменить изменения конфигураций для всех устройств»

Сценарий выполнения для варианта использования «Добавить устройство» представлен на рисунке 7.

Наименование: Добавить устройство
ID: 3
Краткое описание: Администратор инициирует подключение нового устройства к системе
Главные акторы: Администратор
Предусловия: Устройство еще не добавлено
Основной поток: <ol style="list-style-type: none"> 1. Администратор инициирует подключение нового устройства к управляющему устройству и указывает необходимые данные для подключения 2. Запрос передается управляющему устройству 3. Управляющее устройство подключается к необходимому устройству 4. Инициализируются необходимые данные для подключенного устройства
Постусловия: Устройство подключено к системе и доступно для управления конфигурацией

Рисунок 7 – Описание варианта использования «Добавить устройство»

Сценарий выполнения для варианта использования «Удалить устройство» представлен на рисунке 8.

Наименование: Удалить устройство
ID: 4
Краткое описание: Администратор инициирует удаление подключенного устройства из системы
Главные акторы: Администратор
Предусловия: Устройство добавлено
Основной поток: <ol style="list-style-type: none"> 1. Администратор инициирует удаление подключенного устройства из системы, и указывает его идентификатор 2. Запрос передается управляющему устройству 3. Управляющее устройство удаляет связанные с устройством данные
Постусловия: Устройство удалено из системы и недоступно для управления конфигурацией

Рисунок 8 – Описание варианта использования «Удалить устройство»

Реализованная диаграмма вариантов использования точно описывает функциональные требования к системе, а описанные функциональные требования помогают сформировать представление о необходимых задачах, которые должна выполнять система.

2.2 Анализ нефункциональных требований

Нефункциональные требования описывают характеристики разрабатываемой системы, которые описывают каким образом она должна выполнять те или иные действия.

Нефункциональные требования применяют к описанию производительности, надежности, безопасности, удобству использования, масштабируемости и другим параметрам, которые влияют на опыт эксплуатации пользователя разрабатываемой системы.

Отличие от функциональных требований заключается в том, что функциональные требования задают конкретное поведение системы в тех или иных ситуациях, а нефункциональные требования сосредоточены на том, каким образом это поведение должно осуществляться.

Нефункциональные требования необходимы для построения систем с устойчивостью, удобством и безопасностью, так же влияют на выбор стека технологий и разработку архитектуры в целом.

Далее приведены общие нефункциональные требования к системе:

- гарантированное корректное сохранение и восстановление примененной конфигурации на устройстве после локальных изменений, в том числе при сбоях;
- гарантированное корректное сохранение и восстановление примененной конфигурации на устройстве после удаленных изменений, в том числе при сбоях;
- реализация согласованного интерфейса для работы с конфигурациями между управляющим устройством и подключенными устройствами;
- время обработки запросов получения, применения и отправки конфигурации должно стремиться к минимуму;
- гарантированное обеспечение изменений метки времени при установке измененной конфигурации локально или удаленно;

- обеспечение одновременного обслуживания более двадцати подключенных устройств с надлежащей скоростью;
- обеспечение связи с устройствами по защищенному каналу IPsec;
- предоставление набора определенных атомарных операций изменения конкретных частей в конфигурации устройства, чтобы избежать работы с полной версией конфигурации и обеспечить интуитивно понятный интерфейс для внесения изменений.

Перечисленные нефункциональные требования будут использоваться как основа для дальнейшего выбора архитектурных решений и проектирования системы.

3 Проектирование системы

На данном этапе будет рассмотрена архитектура и основные классы для каждого из модулей разрабатываемой системы удаленного управления конфигурацией сетевых устройств.

Исходная система была разбита на несколько модулей, которые будут установлены либо на управляемое удаленное устройство, либо на управляющее устройство:

- модуль конфигурации;
- модуль управления;
- модуль редактирования.

Далее приведено подробное описание каждого модуля системы удаленного управления конфигурацией сетевых устройств.

Модуль конфигурации представляет собой программное обеспечение для подключенного сетевого устройства, конфигурацией которого необходимо управлять с помощью управляющего устройства. Модуль конфигурации используется для отправки в модуль управления конфигурации устройства, на котором он установлен, а также для применения полученной конфигурации от модуля управления, содержащей изменения, которые требуется внести на данном устройстве. Также модуль служит точкой подключения локального администратора для локального внесения изменений в конфигурацию.

Модуль редактирования представляет собой программное обеспечение для управляющего устройства, выступающего единой точкой управления конфигурациями всех подключенных устройств сети. Модуль используется для изменения конфигураций подключенных сетевых устройств, предоставляя удобный интерфейс для их редактирования в рамках отдельных сессий, которые позволяют вносить некоторое число изменений в конфигурации устройств, а затем отправлять данные изменения для применения, а также отменять данные изменения при необходимости. Также

модуль предоставляет функционал редактирования конфигураций с помощью атомарных действий, без необходимости работы с прямым редактированием полных конфигураций.

Модуль управления представляет собой программное обеспечение для управляющего устройства, выступающего единой точкой подключения всех устройств сети. Модуль используется для реализации удаленного механизма отправки и получения конфигураций между управляющим устройством и подключенными к нему сетевыми устройствами. Модуль выступает посредником между модулем редактирования и модулем конфигурации конкретного устройства.

Спроектированные классы для модуля конфигурации были размещены в отдельных файлах, которые были использованы в сборке ConfigurationModule.dll. Данная сборка используется для запуска модуля в операционной системе Debian с помощью платформы .NET [13].

В состав модуля конфигурации на управляемом устройстве вошли 6 компонентов.

Спроектированные классы для модуля управления были размещены в отдельных файлах, которые были использованы в сборке MainModule.dll. Данная сборка используется для запуска модуля в операционной системе Debian с помощью платформы .NET.

В состав модуля управления на управляющем устройстве вошли 12 компонентов.

Спроектированные классы для модуля редактирования были размещены в отдельных файлах, которые были использованы в сборке EditModule.dll. Данная сборка используется для запуска модуля в операционной системе Debian с помощью платформы .NET.

В состав модуля редактирования на управляющем устройстве вошли 15 компонентов.

Общая диаграмма компонентов представлена на рисунке 9.

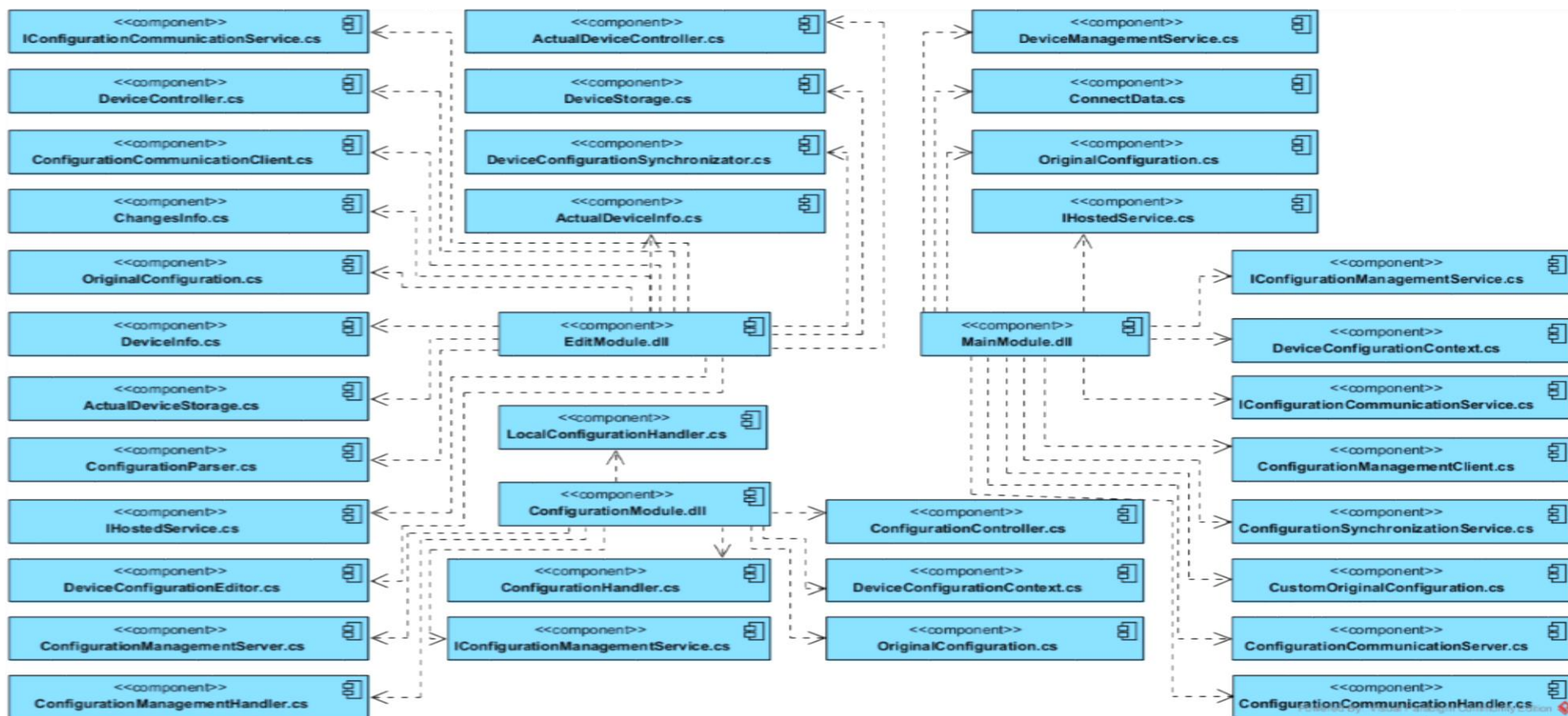


Рисунок 9 – Диаграмма компонентов системы

Рассмотрим подробнее компоненты модуля конфигурации:

- ConfigurationModule.dll является сборкой для запуска модуля и содержит в себе скомпилированные классы из остальных компонентов;
- ConfigurationManagementServer.cs является файлом для описания класса ConfigurationManagementServer;
- IConfigurationManagementService.cs является файлом для описания интерфейса IConfigurationManagementService;
- ConfigurationHandler.cs является файлом для описания класса ConfigurationHandler.
- DeviceConfigurationContext.cs является файлом для описания класса DeviceConfigurationContext;
- OriginalConfiguration.cs является файлом для описания класса OriginalConfiguration.
- LocalConfigurationHandler.cs является файлом для описания класса LocalConfigurationHandler;
- ConfigurationController.cs является файлом для описания класса ConfigurationController.
- ConfigurationManagementHandler.cs является файлом для описания класса ConfigurationManagementHandler.

Рассмотрим подробнее компоненты модуля управления:

- MainModule.dll является сборкой для запуска модуля и содержит в себе скомпилированные классы из остальных компонентов;
- CustomOriginalConfiguration.cs является файлом для описания класса CustomOriginalConfiguration;
- OriginalConfiguration.cs является файлом для описания класса OriginalConfiguration;
- DeviceConfigurationContext.cs является файлом для описания класса DeviceConfigurationContext;
- IConfigurationCommunicationService.cs является файлом для описания интерфейса IConfigurationCommunicationService;

- ConnectData.cs является файлом для описания класса ConnectData;
- ConfigurationCommunicationServer.cs является файлом для описания класса ConfigurationCommunicationServer;
- IHostedService.cs является файлом для описания интерфейса IHostedService;
- ConfigurationSynchronizationService.cs является файлом для описания класса ConfigurationSynchronizationService;
- DeviceManagementService.cs является файлом для описания класса DeviceManagementService;
- ConfigurationManagementClient.cs является файлом для описания класса ConfigurationManagementClient;
- IConfigurationManagementService.cs является файлом для описания интерфейса IConfigurationManagementService;
- ConfigurationCommunicationHandler.cs является файлом для описания класса ConfigurationCommunicationHandler.

Рассмотрим подробнее компоненты модуля редактирования:

- EditModule.dll является сборкой для запуска модуля и содержит в себе скомпилированные классы из остальных компонентов;
- DeviceController.cs является файлом для описания класса DeviceController;
- IConfigurationCommunicationService.cs является файлом для описания интерфейса IConfigurationCommunicationService;
- ConfigurationCommunicationClient.cs является файлом для описания класса ConfigurationCommunicationClient;
- ActualDeviceStorage.cs является файлом для описания класса ActualDeviceStorage;
- ConfigurationParser.cs является файлом для описания класса ConfigurationParser;
- IHostedService.cs является файлом для описания интерфейса IHostedService;

- DeviceConfigurationEditor.cs является файлом для описания класса DeviceConfigurationEditor;
- DeviceInfo.cs является файлом для описания класса DeviceInfo;
- OriginalConfiguration.cs является файлом для описания класса OriginalConfiguration;
- ActualDeviceInfo.cs является файлом для описания класса ActualDeviceInfo;
- ActualDeviceController.cs является файлом для описания класса ActualDeviceController;
- DeviceStorage.cs является файлом для описания класса DeviceStorage;
- DeviceConfigurationSynchronizator.cs является файлом для описания класса DeviceConfigurationSynchronizator;

Таким образом было проведено разбиение системы удаленного управления конфигурацией сетевых устройств на несколько модулей, которые устанавливаются либо на управляющее устройство, либо на управляемое сетевое устройство, в зависимости от назначения данных модулей.

3.1 Проектирование модуля конфигурации

Модуль конфигурации устанавливается на устройстве, конфигурацией которого необходимо управлять. Данный модуль необходим для взаимодействия устройства с управляющим устройством, чтобы предоставлять данные по конфигурации, установленной на устройстве, а также получать обновленную конфигурацию с изменениями для установки, и затем применять данные изменения.

Для данного модуля была спроектирована диаграмма классов, представленная на рисунке 10.

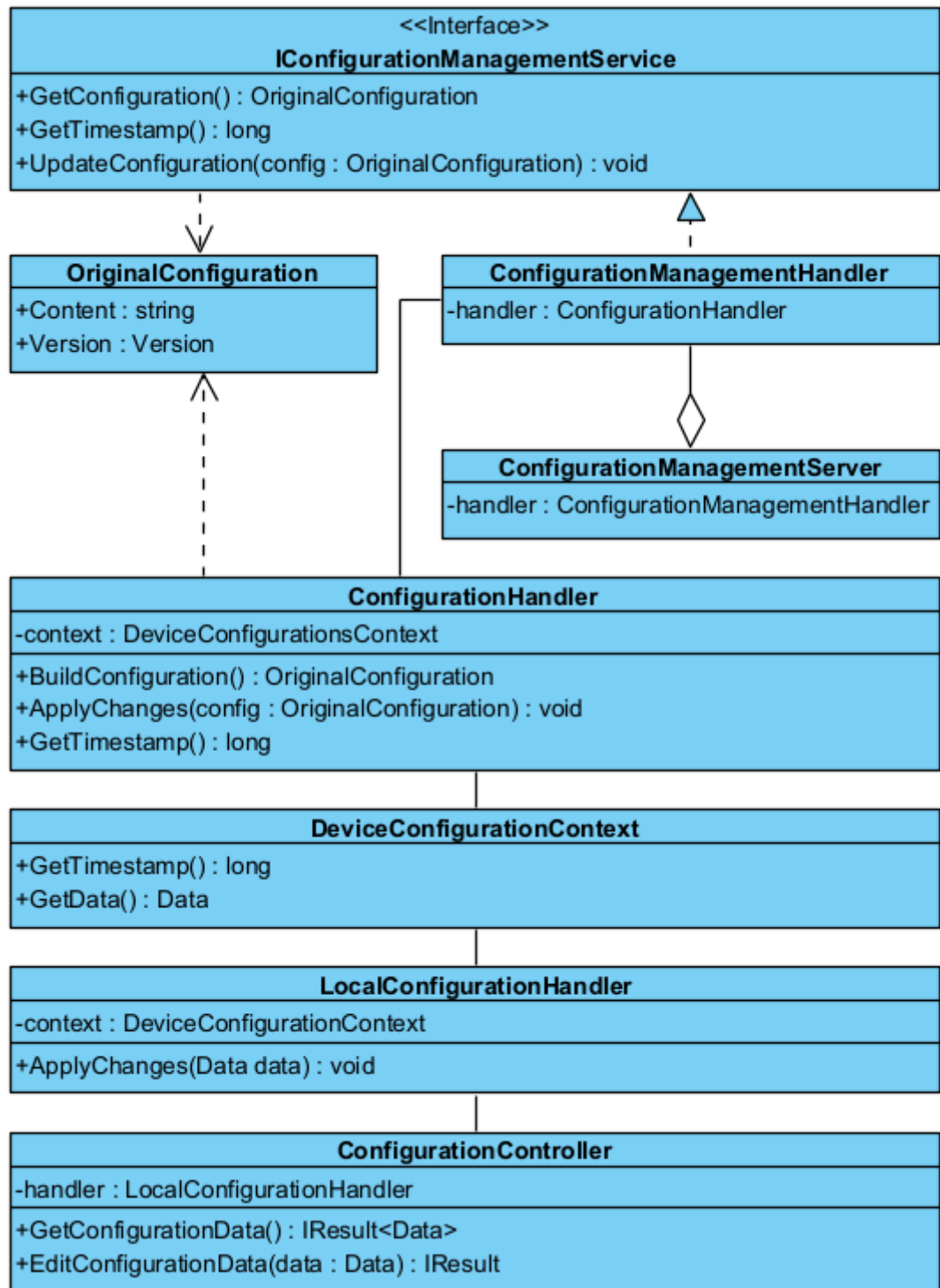


Рисунок 10 – Диаграмма классов модуля конфигурации

Далее приведено описание классов и интерфейсов, указанных на данной диаграмме.

DeviceConfigurationContext является классом для работы с базой данных и используется для хранения информации о данных конфигурации и метке времени конфигурации.

Класс DeviceConfigurationContext реализует метод GetTimestamp для получения метки времени, а также содержит специальные методы по получению и обновлению данных конфигурации.

ConfigurationHandler является классом для управления конфигурацией на устройстве, который принимает и применяет изменения конфигурации, полученной удаленно, а также формирует установленную, на данный момент, конфигурацию устройства по данным из базы данных для отправки.

Класс ConfigurationHandler содержит поле context, представляющий объект типа DeviceConfigurationContext, для работы с базой данных.

Данный класс реализует метод BuildConfiguration() для формирования установленной конфигурации устройства по данным из базы данных.

Также класс ConfigurationHandler реализует метод GetTimestamp() для получения метки времени установленной конфигурации из базы данных.

Помимо этого класс ConfigurationHandler реализует метод ApplyChanges(OriginalConfiguration config) для применения изменений из конфигурации config на устройстве с последующим сохранением измененных данных в базу данных.

OriginalConfiguration является классом конфигурации устройства.

Класс OriginalConfiguration содержит поле ConfigJson, представляющий собой строку в формате json, которая содержит сериализованные данные конфигурации.

Также класс OriginalConfiguration содержит поле Version, которое описывает информацию о версии конфигурации для определения ее формата.

IConfigurationManagementService является интерфейсом для обмена информации о конфигурации устройства между модулем конфигурации на устройстве и модулем управления на управляющем устройстве.

Интерфейс `IConfigurationManagementService` описывает методы `GetConfiguration()` для получения установленной на устройстве конфигурации, метод `GetTimestamp()` для получения метки времени установленной на устройстве конфигурации.

Также интерфейс `IConfigurationManagementService` описывает метод `UpdateConfiguration(OriginalConfiguration config)` для установки новой конфигурации `config`, которая содержит внесенные изменения, на устройстве.

`ConfigurationManagementHandler` является классом реализации функционала интерфейса `IConfigurationManagementService`.

Класс `ConfigurationManagementHandler` содержит поле `handler`, представляющее собой объект типа `ConfigurationHandler` для управления конфигурацией устройства.

`ConfigurationManagementServer` является классом сервером для предоставления функционала `ConfigurationManagementHandler`.

Класс `ConfigurationManagementServer` содержит поле `handler`, представляющее собой объект типа `ConfigurationHandler` для управления конфигурацией устройства.

`LocalConfigurationHandler` является классом для управления конфигурацией на устройстве, который принимает и применяет изменения конфигурации полученной локально на устройстве.

Класс `LocalConfigurationHandler` содержит поле `context`, представляющее собой объект типа `DeviceConfigurationContext`, для работы с базой данных, для доступа к метке времени и данным конфигурации.

Также класс `LocalConfigurationHandler` реализует метод `ApplyChanges(Data data)` для применения изменений данных конфигурации `data` с последующим сохранением измененных данных в базу данных.

`ConfigurationController` является классом контроллером для обработки запросов получения данных конфигурации и их изменения локально.

Класс `ConfigurationController` содержит поле `handler`, представляющее собой объект типа `LocalConfigurationHandler`, локального управления конфигурацией на устройстве.

Также класс `ConfigurationController` реализует методы `GetConfigurationData()` для получения данных конфигурации в виде HTTP ответа.

Помимо этого класс `ConfigurationController` реализует метод `EditConfigurationData(Data data)` для редактирования данных конфигурации `data` и применение изменений, после чего получения результата выполнения операции в виде HTTP ответа.

Таким образом были спроектированы классы, входящие в состав модуля конфигурации.

3.2 Проектирование модуля управления

Модуль управления устанавливается на управляющем устройстве. Данный модуль необходим для взаимодействия управляющего устройства со всеми подключенными устройствами, чтобы собирать данные конфигураций со всех устройств и передавать их в модуль редактирования, а затем, после изменений, отправлять их обратно на устройства для последующей установки.

Также модуль является точкой подключения устройств к системе, а также их удаления из системы.

Таким образом модуль выступает посредником между модулем редактирования на этом же управляющем устройстве и модулями конфигурации конкретных сетевых устройств, которые подключены к системе удаленного управления конфигурацией сетевых устройств.

Для данного модуля была спроектирована диаграмма классов, представленная на рисунке 11.

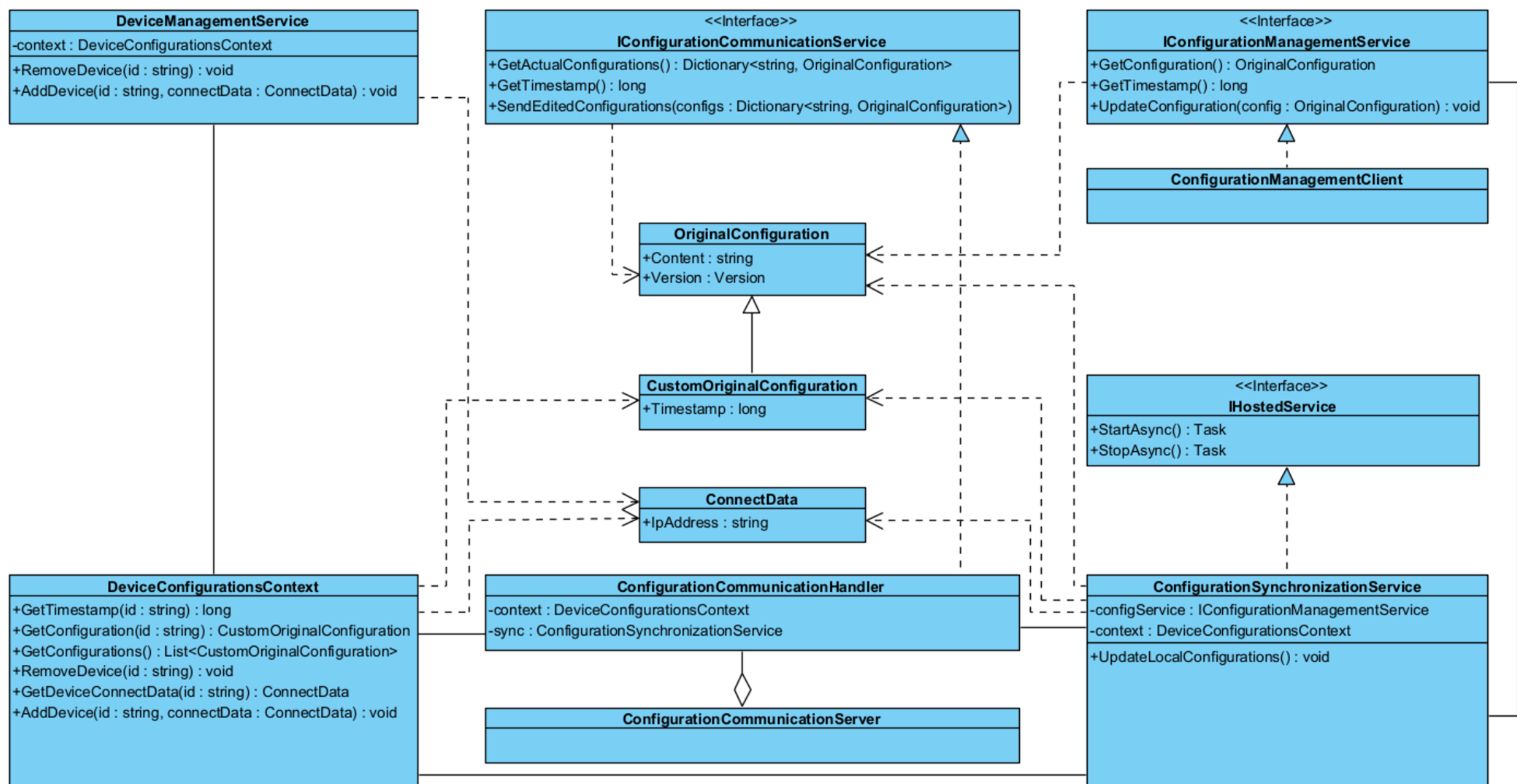


Рисунок 11 – Диаграмма классов модуля управления

Далее приведено описание классов и интерфейсов на данной диаграмме.

`DeviceManagementService` является классом для управления устройствами, которые подключены к системе.

Класс `DeviceManagementService` реализует метод `RemoveDevice(string id)` для удаления устройства с идентификатором `id` из системы.

Также данный класс реализует метод `AddDevice(string id, ConnectData connectData)` для добавления устройства с идентификатором `id` и данными для подключения к нему `connectData` в систему.

`ConnectData` является классом для описания данных подключения к устройству.

Класс `ConnectData` содержит поле `IpAddress`, представляющее собой строку с IP адресом устройства, а также остальные данные для подключения к устройству.

`DeviceConfigurationContext` является классом для работы с базой данных и используется для хранения информации об устройствах, включая данные их конфигураций и меток времени.

Класс `DeviceConfigurationContext` реализует метод `GetTimestamp(string id)` для получения метки времени конфигурации устройства с идентификатором `id`.

Также данный класс реализует метод `CustomOriginalConfiguration GetConfiguration(string id)` для получения конфигурации устройства с идентификатором `id` и метод `GetConfigurations()` для получения списка конфигураций устройств.

Помимо этого `DeviceConfigurationContext` реализует метод `RemoveDevice(string id)` для удаления устройства с идентификатором `id` из базы данных, метод `AddDevice(string id, ConnectData connectData)` для добавления устройства с идентификатором `id` и данными для подключения к нему `connectData` в базу данных и метод `ConnectData GetDeviceConnectData(string id)` для получения данных для подключения к устройству с идентификатором `id`.

IHostedService является интерфейсом для выполнения отдельного функционала при запуске модуля.

Интерфейс IHostedService описывает метод StartAsync(), выполнение которого происходит при запуске модуля.

Помимо этого, данный интерфейс описывает метод StopAsync(), выполнение которого выполняется при завершении работы модуля.

ConfigurationSynchronizationService является классом для синхронизации данных конфигураций, которые хранятся в модуле с актуальными данными конфигураций на устройствах.

Также класс ConfigurationSynchronizationService реализует интерфейс IHostedService.

Класс ConfigurationSynchronizationService содержит поле context, которое является объектом типа DeviceConfigurationContext, для работы с базой данных и поле configService, которое является объектом, реализующим интерфейс IConfigurationManagementService, для актуализации локальных конфигураций в рамках модуля.

Также данный класс реализует метод UpdateLocalConfigurations() для обновления данных конфигураций, которые хранятся в модуле при наличие изменений в конфигурациях на устройствах.

IConfigurationManagementService является интерфейсом для обмена информации о конфигурации устройства между модулем конфигурации на устройстве и модулем управления на управляющем устройстве.

Интерфейс IConfigurationManagementService описывает метод GetConfiguration() для получения установленной на устройстве конфигурации.

Также данный интерфейс описывает метод GetTimestamp() для получения метки времени установленной на устройстве конфигурации.

Помимо этого интерфейс IConfigurationManagementService описывает метод UpdateConfiguration(OriginalConfiguration config) для установки новой

конфигурации `config`, которая содержит внесенные изменения, на управляемом устройстве.

`ConfigurationManagementClient` является классом клиентом сервера `ConfigurationManagementServer` и реализует интерфейс `IConfigurationManagementService`.

`IConfigurationCommunicationService` является интерфейсом для обмена информацией о конфигурациях подключенных устройств между модулем управления и модулем редактирования.

Интерфейс `IConfigurationCommunicationService` описывает метод `GetActualConfigurations()` для получения конфигураций, установленных на устройствах.

Также данный интерфейс описывает метод `GetTimestamp` для получения общей метки времени конфигураций устройств для удобной проверки наличия изменений в устройствах и метод `SendEditedConfigurations(Dictionary<string, OriginalConfiguration> configs)` для отправки измененных конфигураций устройств `configs` для дальнейшего применения изменений на устройствах.

`ConfigurationCommunicationHandler` является классом для реализации интерфейса `IConfigurationCommunicationService`.

Класс `ConfigurationCommunicationHandler` содержит поле `sync`, которое является объектом типа `ConfigurationSynchronizationService`, для синхронизации данных конфигураций, которые хранятся в модуле с актуальными данными конфигураций на устройствах.

Также данный класс содержит поле `context`, которое является объектом типа `DeviceConfigurationContext`, для работы с базой данных.

`ConfigurationCommunicationServer` является классом сервером для предоставления функционала `ConfigurationCommunicationHandler`.

Класс `ConfigurationCommunicationServer` содержит поле `handler`, которое является объектом типа `ConfigurationCommunicationHandler`, для

обмена информацией о конфигурациях подключенных устройств между модулем управления и модулем редактирования.

OriginalConfiguration является классом конфигурации устройства.

Класс OriginalConfiguration содержит поле ConfigJson, представляющее собой строку в формате json, которая содержит сериализованные данные конфигурации.

Также данный класс содержит поле Version, представляющее собой информацию о версии конфигурации для определения ее формата.

CustomOriginalConfiguration является модифицированным классом конфигурации устройства с меткой времени. Наследуется от класса OriginalConfiguration.

Класс CustomOriginalConfiguration содержит поле Timestamp, представляющее собой метку времени конфигурации.

Таким образом были спроектированы классы, входящие в состав модуля управления.

3.3 Проектирование модуля редактирования

Модуль редактирования устанавливается на управляющем устройстве. Данный модуль необходим для предоставления удобного функционала по редактированию конфигураций, а также предоставления данных конфигураций администратору.

Модуль также реализует функционал редактирования конфигураций подключенных сетевых устройств в виде сессий, позволяя вносить атомарные изменения в конфигурации, а затем уже отправлять их на отправку.

Для данного модуля была спроектирована диаграмма классов, представленная на рисунке 12.

Далее приведено описание классов и интерфейсов на данной диаграмме.

ChangesInfo является классом для получения общей информации об изменениях в конфигурациях.

Класс ChangesInfo содержит поле HasChanges, значение типа bool, которое используется как флаг наличия изменений в конфигурациях устройств, поле Timestamp, значение типа long, которое используется как метка времени применения конфигураций на устройствах, поле ChangesTimestamp, значение типа long, которое используется как метка времени последних внесенных изменений в конфигурации устройств.

DeviceInfo является классом с информацией об устройстве и его конфигурацией в удобном для изменений виде.

Класс DeviceInfo содержит поле Id, значение типа string, которое представляет собой идентификатор устройства, поле ParsingDetails, значение типа ConfigurationParsingDetails, которое представляет собой детали десериализации входящей начальной конфигурации, а именно: статус десериализации конфигурации устройства и список ошибок при разборе конфигурации данного устройства,

Помимо этого, класс DeviceInfo содержит поле Status, значение типа DeviceStatus, которое представляет собой статус устройства в модуле редактирования, поле OriginalConfig, значение типа OriginalConfiguration, которое представляет собой полученную конфигурацию устройства от модуля управления.

Также класс устройства DeviceInfo содержит разобранные данные EditableData из конфигурации в удобном виде для изменений.

OriginalConfiguration является классом конфигурации устройства.

Класс OriginalConfiguration содержит поле ConfigJson, представляющий собой строку в формате json, которая содержит сериализованные данные конфигурации.

Также класс OriginalConfiguration содержит поле Version, которое описывает информацию о версии конфигурации для определения ее формата.

ActualDeviceInfo является классом с информацией об устройстве и его конфигурацией, которая в данный момент установлена на устройстве.

Класс ActualDeviceInfo содержит поле Id, значение типа string, которое представляет собой идентификатор устройства.

Также класс ActualDeviceInfo содержит разобранные данные Data из конфигурации в удобном виде для представления администратору.

DeviceStorage является классом хранилищем данных об устройствах с конфигурациями, содержащих изменения.

Класс DeviceStorage содержит поле, infos, объект типа Dictionary<string, DeviceInfo>, который представляет собой данные об устройствах с конфигурациями, содержащие изменения.

Класс DeviceStorage реализует метод GetDeviceInfo(string id) для получения информации об устройстве с идентификатором id с конфигурацией, которая содержит изменения, внесенные в рамках данной сессии, метод GetDeviceInfos() для получения информации о всех подключенных устройствах с конфигурациями, которые содержат изменения, внесенные в рамках данной сессии.

ActualDeviceStorage является классом хранилищем данных об устройствах с конфигурациями, установленными на устройствах.

Класс ActualDeviceStorage содержит поле infos, объект типа Dictionary<string, ActualDeviceInfo>, который представляет собой данные об устройствах с конфигурациями, установленными на устройствах.

Класс ActualDeviceStorage содержит метод GetDeviceActualInfo(string id) для получения информации об устройстве с идентификатором id и конфигурацией, которая установлена на устройстве, метод GetDeviceInfos() для получения информации о всех подключенных устройствах с конфигурациями, которые установлены на устройствах, метод GetDevicesCount() для получения количества подключенных к системе устройств.

ConfigurationParser является классом для десериализации полученных от модуля управления конфигураций устройств.

Класс ConfigurationParser реализует метод Parse(OriginalConfiguration), который производит десериализацию полученной от модуля управления конфигураций устройства информацию об устройстве ActualDeviceInfo, метод Parse(OriginalConfiguration), который производит десериализацию полученной от модуля управления конфигураций устройства информацию об устройстве DeviceInfo.

IConfigurationCommunicationService является интерфейсом для обмена информацией о конфигурациях подключенных устройств между модулем управления и модулем редактирования.

Интерфейс описывает метод GetActualConfigurations() для получения конфигураций, установленных на устройствах, метод GetTimestamp для получения общей метки времени конфигураций устройств для удобной проверки наличия изменений в устройствах, метод SendEditedConfigurations(Dictionary<string, OriginalConfiguration> configs) для отправки измененных конфигураций устройств configs для дальнейшего применения изменений на устройствах.

ConfigurationCommunicationClient является классом клиентом сервера ConfigurationCommunicationServer и реализует интерфейс IConfigurationCommunicationService.

IHostedService является интерфейсом для выполнения отдельного функционала при запуске модуля.

IHostedService является интерфейсом для выполнения отдельного функционала при запуске модуля.

Интерфейс IHostedService описывает метод StartAsync(), выполнение которого происходит при запуске модуля.

Помимо этого данный интерфейс описывает метод StopAsync(), выполнение которого выполняется при завершении работы модуля.

DeviceConfigurationsSynchronizator является основным классом управления конфигурациями в модуле. Реализует интерфейс IHostedService для первичной загрузки конфигураций.

Класс DeviceConfigurationsSynchronizator содержит поле actualStorage, объект типа ActualDeviceStorage, который представляет собой хранилище с информацией об устройствах и их конфигурациях, установленных на них, поле storage, объект типа DeviceStorage, который представляет собой хранилище с информацией об устройствах и их конфигурациях с внесенными изменениями.

Также данный класс содержит поле parser, объект типа ConfigurationParser, для разбора полученных конфигураций устройств, поле communicationService, объект, реализующий интерфейс IConfigurationCommunicationService, для обмена информацией о конфигурациях подключенных устройств между модулем управления и модулем редактирования.

Класс DeviceConfigurationsSynchronizator реализует метод UpdateDevicesConfigs() для отправки конфигураций всех устройств с изменениями на модуль управления для их дальнейшей установки на подключенные устройства, метод DiscardDevicesConfigsChanges() для отмены всех внесенных изменений в конфигурации всех устройств в рамках данной сессии, механизм которого осуществляется с помощью отката до состояния актуальной конфигурации соответствующего устройства.

Помимо этого, класс DeviceConfigurationsSynchronizator содержит метод GetConfigChangesInfo() для получения информации об изменениях в конфигурациях, метод UpdateStorages() для обновления информации об устройствах в хранилищах.

DeviceController является классом контроллером для обработки запросов получения информации о конфигурациях, которые содержат изменения в рамках данной сессии, а так же обработки запросов для изменения конфигураций.

Класс DeviceController содержит поле storage, объект типа DeviceStorage, который представляет собой хранилище информации об устройствах, поле provider, объект типа DeviceConfigurationsSynchronizator, для управления конфигурациями в модуле.

Класс DeviceController реализует метод GetDeviceInfo(string id) для получения информации о подключенном устройстве с идентификатором id с конфигурацией, которая содержит изменения, внесенные в рамках данной сессии, в виде HTTP ответа, метод GetDeviceInfos() для получения информации о всех подключенных устройствах с конфигурациями, которые содержат изменения, внесенные в рамках данной сессии, в виде HTTP ответа.

Также данный класс содержит метод UpdateDevicesConfigs() для отправки конфигураций всех устройств с изменениями на модуль управления для дальнейшей установки на подключенные устройства, и получение результата выполнения операции в виде HTTP ответа, метод DiscardDevicesConfigsChanges() для отмены всех внесенных изменений в конфигурации всех устройств в рамках данной сессии, и получение результата выполнения операции в виде HTTP ответа, метод GetConfigChangesInfo() для получения информации об изменениях конфигураций устройств в виде HTTP ответа.

ActualDeviceController является классом контроллером для обработки запросов в рамках получения информации об устройствах с конфигурациями, которые установлены на устройствах.

Класс ActualDeviceController содержит поле actualStorage, объект типа ActualDeviceStorage, который представляет собой хранилище с информацией об устройствах с конфигурациями, установленными на устройствах.

Класс ActualDeviceController реализует метод GetDeviceActualInfo(string id) для получения информации об устройстве с идентификатором id с конфигурацией, установленной на устройстве, в виде HTTP ответа, метод GetDeviceActualInfos() для получения информации о всех устройствах с конфигурациями, установленными на них, в виде HTTP

ответа, метод `GetDevicesCount()` для получение количества устройств для управления их конфигурацией, в виде HTTP ответа.

`DeviceConfigurationEditor` является классом для редактирования данных конфигурации устройства.

Класс `DeviceConfigurationEditor` реализует метод `Edit(EditableData data)` для редактирования изменяемых данных конфигурации устройства.

Таким образом были спроектированы классы, входящие в состав модуля редактирования.

4 Реализация системы

В состав разрабатываемой системы удаленного управления конфигурацией сетевых устройств входит несколько подсистем, которые приведены далее.

Подсистема устройство представляет собой устройство, которое подключают к управляющему устройству для дальнейшего управления его конфигурацией с помощью реализованных функций.

Подсистема содержит в себе модуль конфигурации, который реализует серверную часть взаимодействия с модулем управления на управляющем устройстве с помощью Apache Thrift для отправки необходимых данных на управляющее устройство, таких как: метка времени изменения используемой конфигурации и конфигурация, а также получения данных с управляющего устройства, таких как: измененная конфигурация [14].

Frontend-приложение является веб-интерфейсом локального администратора, который реализует удобное визуальное представление интерфейса для локального редактирования конфигурации устройства.

Веб-сервер Nginx обеспечивает прием HTTPS-запросов от клиента и перенаправляет их к frontend-приложению.

Также модуль отвечает за обработку полученной измененной конфигурации для корректного применения внесенных изменений. Данные, которые используются в конфигурации локально хранятся в базе данных SQLite, а взаимодействие с базой данных происходит через ORM Entity Framework.

Подсистема управляющее устройство представляет собой управляющее устройство, выступающее единой точкой подключения всех устройств сети, для реализации удаленного механизма управления конфигурацией. Содержит в себе несколько модулей таких, как:

Модуль управления реализует клиентскую часть интерфейса взаимодействия с модулем конфигурации на устройстве с помощью Apache

Thrift для получения необходимых данных, таких как: метка времени изменения используемой конфигурации на устройстве и конфигурация, а также отправки данных на устройство, таких как: измененная конфигурация. Так же реализует серверную часть интерфейса взаимодействия с модулем редактирования на этом же управляющем устройстве с помощью Apache Thrift для отправки актуальных конфигураций, а также получения измененных конфигураций для дальнейшей отправки на устройства.

Так же модуль отвечает за управление списком сетевых устройств. Данные, которые используются для подключения к устройствам хранятся в базе данных SQLite, а взаимодействие с базой данных происходит через ORM Entity Framework.

Таким образом модуль является основным компонентом распределения измененных конфигураций на устройства.

Модуль редактирования реализует клиентскую часть интерфейса взаимодействия с модулем управления на управляющем устройстве с помощью Apache Thrift для получения актуальных конфигураций, а также отправки измененных конфигураций в модуль управления управляющего устройства. Реализует логику по обработке конфликтов в изменениях конфигураций, которые были сделаны локально и удаленно, а также предоставляет удобный интерфейс для изменения конфигураций устройств без необходимости редактирования полной конфигурации с помощью реализации специальных функций.

Frontend-приложение является веб-интерфейсом администратора, который реализует удобное визуальное представление интерфейса модуля редактирования, связь с модулем редактирования обеспечивается за счет HTTP протокола.

Веб-сервер Nginx обеспечивает прием HTTPS-запросов от клиента и перенаправляет их к frontend-приложению.

Подсистема ПК администратора используется для взаимодействия с управляющим устройством для редактирования конфигураций

подключенных устройств через веб-браузер по защищенному протоколу HTTPS, обеспечивая авторизованный доступ к функциональности Frontend-приложения.

Подсистема ПК локального администратора используется для взаимодействия с устройством для внесения изменений в его конфигурацию через веб-браузер, что происходит локально по защищенному протоколу HTTPS, обеспечивая авторизованный доступ к функциональности Frontend-приложения.

Коммуникация между управляющим устройством и удаленными устройствами осуществляется через Thrift интерфейс, что обеспечивает независимость от платформ и языков программирования, и позволяет обеспечить удобный согласованный интерфейс.

Для обеспечения безопасности передаваемых данных между управляющим устройством и удаленным сетевыми устройствами используется протокол IPsec, который обеспечивает надежное шифрование передаваемых данных на сетевом уровне, что особенно важно при передаче данных конфигурации между удаленными устройствами.

Диаграмма развертывания для разрабатываемой системы приведена на рисунке 13.

Диаграмма развертывания представляет собой вид диаграммы UML, который показывает физическое размещение программных компонентов на модулях всей системы в целом. Диаграмма развертывания отражает, как программные компоненты, такие как: сервисы, базы данных, развернуты на модулях системы, представляющие собой серверы, компьютеры или другие устройства в системе.

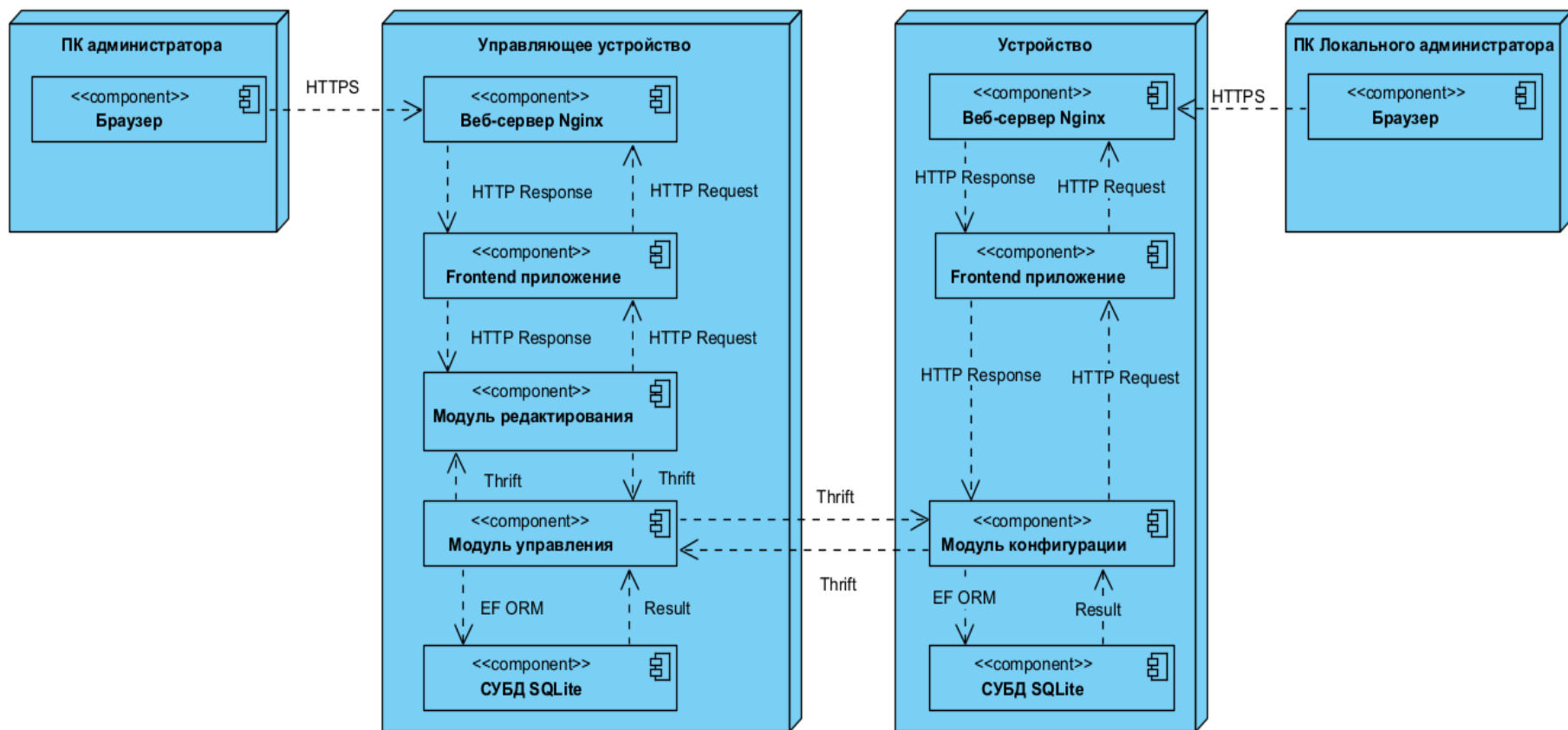


Рисунок 13 – Диаграмма развертывания

Таким образом была разработана диаграмма развертывания для описания реализуемой системы.

4.1 Реализация модуля конфигурации

Для организации взаимодействия модуля конфигурации и модуля управления используется интерфейс `IConfigurationManagementService`, который был описан на языке Thrift и скомпилирован в сборку с помощью инструмента Apache Thrift.

В листинге 1 приведено описание данного интерфейса на языке Thrift.

Листинг 1 – Описание интерфейса `IConfigurationManagementService`

```
service ConfigurationManagementService{
    i64 GetTimestamp()
    OriginalConfiguration GetConfiguration()
    void UpdateConfiguration(
        1: OriginalConfiguration config
    )
}
```

Также в файл для компиляции через Apache Thrift было добавлено описание структуры `OriginalConfiguration`, описание которой приведено в листинге 2.

Листинг 2 – Описание класса `OriginalConfiguration`

```
struct OriginalConfiguration{
    1: string content
    2: Version version
}
```

Описание данных компонентов было сохранено в thrift-файле `configuration_module.thrift` и скомпилировано в сборки модуля конфигурации и модуля управления.

Для генерации в классы на языке C# была выполнена команда, представленная в листинге 3 [15].

Листинг 3 – Команда генерации Thrift

```
thrift -gen netstd:net6 -r -out <путь для сгенерированных
компонентов> <путь к thrift файлу>
```

После выполнения данной команды был создан интерфейс `ConfigurationManagementService.IAsync`, который описывает методы `IConfigurationManagementService`.

От данного интерфейса наследуется сгенерированный класс `Client`, который будет использоваться для выполнения описанных методов в других модулях.

Также от данного интерфейса реализуется класс обработчик `ConfigurationManagementHandler`, который далее передается для создания сгенерированного класса `ConfigurationManagementService.Processor`, используемого для создания Thrift сервера для принятия запросов на выполнение данных методов. Сервер реализован через Unix-сокеты.

Содержимое конфигураций и информация по метке времени хранится в базе данных, работа с которой осуществляется с помощью ORM системы `Entity Framework`.

Для этого класс `DeviceConfigurationContext` был унаследован от базового класса `Microsoft.EntityFrameworkCore.DbContext`. В классе добавлены свойства типа `DbSet` для данных конфигурации и метки времени, представляющие собой коллекцию объектов, которая сопоставляется с определенной таблицей в базе данных.

Модуль реализован в виде Linux-демона, который представляет собой фоновый процесс в операционной системе и работает независимо от взаимодействия с пользователем. Автоматический запуск при загрузке системы осуществляет `systemd`, для чего понадобилось описать модуль в файле `configuration_module.service`, содержимое которого представлено в листинге 4.

Листинг 4 – Описание модуля конфигурации

```
[Unit]
Description=Модуль конфигурации устройства
[Service]
WorkingDirectory=/usr/local/bin/ConfigurationModule
ExecStart=/usr/bin/dotnet
/usr/local/bin/ConfigurationModule/ConfigurationModule.dll
```

```
Restart=always
RestartSec=10
```

В данном файле описаны следующие параметры:

- Description – краткое описание модуля;
- WorkingDirectory – рабочий каталог для запуска процесса;
- ExecStart – команда для запуска модуля;
- Restart – перезапуск службы при завершении;
- RestartSec – время ожидания перед перезапуском службы.

Таким образом был реализован модуль конфигурации.

4.2 Реализация модуля управления

Для организации взаимодействия модуля управления и модуля редактирования используется интерфейс `IConfigurationCommunicationService`, который был описан на языке Thrift и скомпилирован в сборку с помощью инструмента Apache Thrift.

В листинге 5 приведено описание данного интерфейса на языке Thrift.

Листинг 5 – Описание интерфейса `IConfigurationCommunicationService`

```
service ConfigurationCommunicationService{
    map<string, OriginalConfiguration> GetActualConfigurations()
    long GetTimestamp()
    void SendEditedConfigurations(
        1: map<string, OriginalConfiguration> configs
    )
}
```

Также было добавлено описание структуры `OriginalConfiguration`.

Описание компонентов было сохранено в thrift файле `main_module.thrift` и скомпилировано в сборки модуля конфигурации и модуля управления.

После выполнения команды генерации был создан интерфейс `ConfigurationCommunicationService.IAsync`, который описывает методы в `IConfigurationCommunicationService`.

От данного интерфейса наследуется сгенерированный класс `Client`, который будет использоваться для выполнения описанных методов в других модулях.

Также от данного интерфейса реализуется класс обработчик `ConfigurationCommunicationHandler`, который далее передается для создания сгенерированного класса `ConfigurationCommunicationService.Processor`, используемого для создания Thrift сервера для принятия запросов на выполнение данных методов. Сервер реализован через Unix-сокеты.

Для передачи конфигураций подключенных устройств была выбрана структура `map`, которая эффективно позволяет получать значения по ключу. В данном случае ключом будет являться идентификатор устройства, а значением его конфигурация. Данная структура после генерации имеет тип `Dictionary<string, OriginalConfiguration>`, что является ее аналогом в языке программирования C# [16].

Также было принято решение использовать общую метку времени для передачи клиенту данного сервера, чтобы обеспечить эффективную и быструю проверку на наличие изменений в установленных на устройствах конфигурациях, при обнаружении изменений хотя бы на одном устройстве данная метка меняется.

Конфигурации и данные об устройствах хранятся в базе данных, работа с которой осуществляется с помощью ORM системы `Entity Framework`.

Для этого класс `DeviceConfigurationContext` был унаследован от базового класса `Microsoft.EntityFrameworkCore.DbContext`. В классе добавлены свойства типа `DbSet` для конфигураций и данных об устройствах, представляющие собой коллекцию объектов, которая сопоставляется с определенной таблицей в базе данных [17].

Модуль реализован в виде Linux-демона, который представляет собой фоновый процесс в операционной системе и работает независимо от взаимодействия с пользователем. Автоматический запуск при загрузке

системы осуществляет systemd, для чего понадобилось описать модуль в файле `main_module.service`, содержимое которого представлено в листинге 6.

Листинг 6 – Описание модуля управления

```
[Unit]
Description=Модуль управления
[Service]
WorkingDirectory=/usr/local/bin/MainModule
ExecStart=/usr/bin/dotnet
/usr/local/bin/MainModule/MainModule.dll
Restart=always
RestartSec=10
```

В данном файле описаны следующие параметры:

- Description – краткое описание модуля;
- WorkingDirectory – рабочий каталог для запуска процесса;
- ExecStart – команда для запуска модуля;
- Restart – перезапуск службы при завершении;
- RestartSec – время ожидания перед перезапуском службы.

При запуске объект класса `ConfigurationSynchronizationHandler` выполняет получение данных о конфигурациях для актуализации с подключенных устройств благодаря реализации `Microsoft.Extensions.Hosting.IHostedService` интерфейса, в методе `StartAsync` которого и происходит вызов метода актуализации.

Также модуль содержит сгенерированный через инструмент `Apache Thrift` клиент сервера выполнения функционала, описанного в `ConfigurationManagementService.IAsync`.

Интерфейс по добавлению новых устройств в систему удаленного управления конфигурацией представлен на рисунке 14.

Добавление узла сети

Добавление узла сети

* Имя centcgw

Описание

* IP-адреса 20 . 0 . 0 . 3 +

Далее

Рисунок 14 – Интерфейс добавления сетевых устройств

Интерфейс по созданию соединений между управляющим устройством и подключенными устройствами представлен на рисунке 15.

Добавление соединения

Данные о соединении

* Первый узел centcgw (Центральный)

* Второй узел gw-2

Создать Отмена

Рисунок 15 – Интерфейс создания соединения

Таким образом был реализован модуль управления.

4.3 Реализация модуля редактирования

Модуль содержит сгенерированный через инструмент Apache Thrift клиент сервера выполнения функционала описанного в ConfigurationCommunicationService.IAsync.

Модуль реализован в виде Linux-демона, который представляет собой фоновый процесс в операционной системе и работает независимо от взаимодействия с пользователем. Автоматический запуск при загрузке системы осуществляет systemd, для чего понадобилось описать модуль в файле edit_module.service, содержимое которого представлено в листинге 7.

Листинг 7 – Описание модуля редактирования

```
[Unit]
Description=Модуль редактирования конфигурации устройства
[Service]
WorkingDirectory=/usr/local/bin/EditModule
ExecStart=/usr/bin/dotnet
/usr/local/bin/EditModule/EditModule.dll
Restart=always
RestartSec=10
```

В данном файле описаны следующие параметры:

- Description – краткое описание модуля;
- WorkingDirectory – рабочий каталог для запуска процесса;
- ExecStart – команда для запуска модуля;
- Restart – перезапуск службы при завершении;
- RestartSec – время ожидания перед перезапуском службы.

Модуль редактирования при запуске получает структуру Dictionary<string, OriginalConfiguration> от модуля управления с конфигурациями OriginalConfiguration с версией конфигурации и производит для каждого устройства десериализацию содержимого конфигурации и фиксирует все возникающие ошибки. После десериализации создается структура с информацией об устройстве, в которую записываются десериализованные данные, возникшие при десериализации ошибки, статус десериализации.

Для присваивания статуса десериализации полученной начальной конфигурации устройства было реализовано перечисление Enum, представленное в листинге 8.

Листинг 8 – Перечисление статусов десериализации конфигурации

```
public enum ParsingStatus
{
    Ok,
    Error,
    Warning
}
```

Статусы означают:

- Ok – начальная конфигурация была успешно десериализована;
- Error – возникли критические ошибки при десериализации конфигурации;
- Warning – начальная конфигурация была успешно десериализована, но конфигурация не соответствует требованиям.

Далее будут приведены требования к полям структуры конфигурации.

Требования, предъявленные к конфигурации, в рамках формата данных поля версии:

- версия должна быть обозначена;
- версия должна быть поддерживаема в модуле редактирования.

Требования, предъявленные к конфигурации, в рамках формата данных поля содержимое конфигурации:

- строка не должна быть пустой;
- строка не должна быть null;
- строка должна представлять собой формат json;
- json не должен представлять из себя значение «null»
- структура json должна совпадать со структурой для десериализации.

Если конфигурация не соответствует какому-либо требованию необходимо зафиксировать это в структуре информации об устройстве.

Интерфейс с описанием конфигураций устройств, содержащих изменения в рамках текущей сессии представлен на рисунке 16 [18].

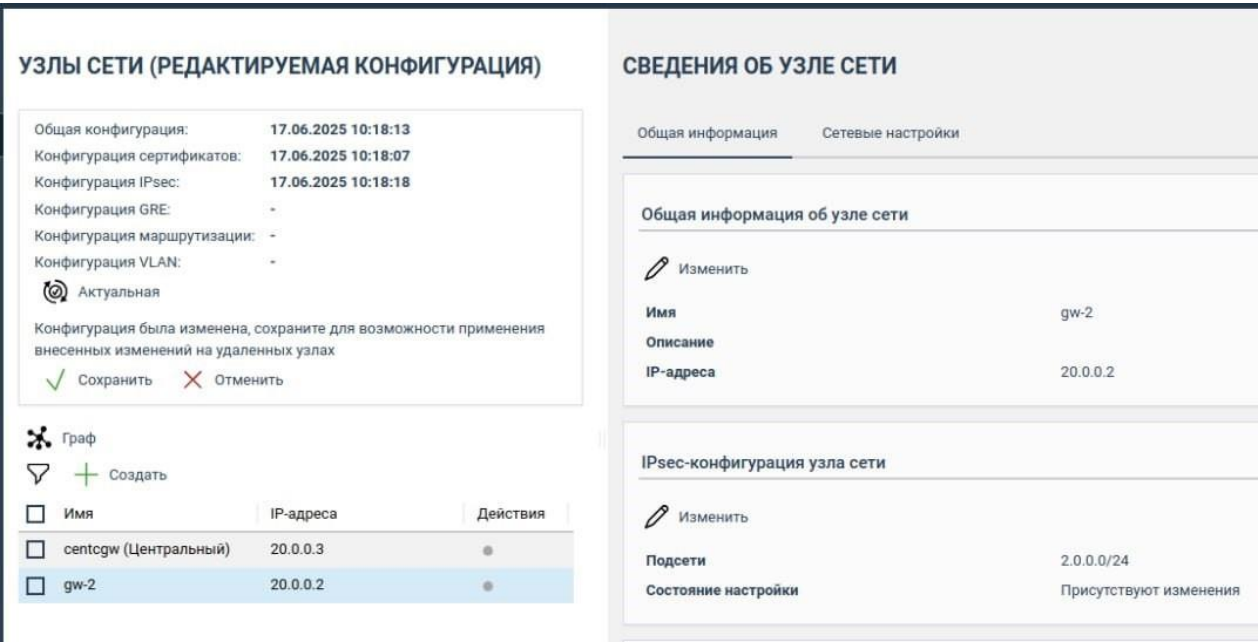


Рисунок 17 – Интерфейс с описанием редактируемых конфигураций

Для формирования коллекции типа Dictionary<string, DeviceInfo> с информацией об устройствах, которая содержит все необходимые данные для редактирования. Данный тип был выбран для быстрого доступа к информации об устройстве по идентификатору.

Также подобная обработка проводится для формирования коллекции типа Dictionary<string, ActualDeviceInfo> с информацией об устройствах, которая содержит данные по установленным на устройствах конфигурациях на данный момент.

Обе коллекции хранятся в памяти в полях классов-хранилищ DeviceStorage и ActualDeviceStorage, что избавило от необходимости подключения базы данных к модулю, все данные хранятся в модуле управления и запрашиваются оттуда по необходимости.

Диаграмма деятельности для запуска модуля редактирования представлена на рисунке 17.

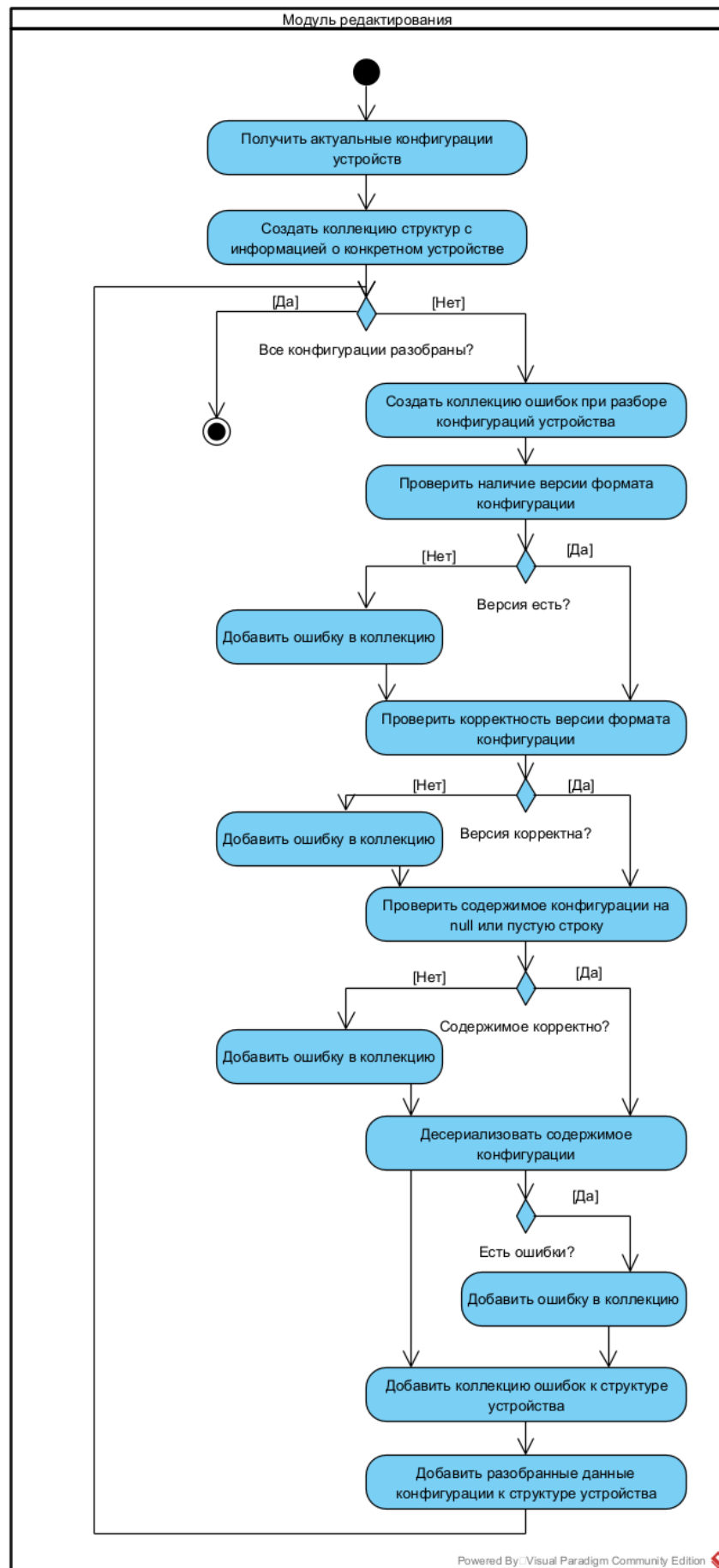


Рисунок 17 – Диаграмма деятельности запуска модуля редактирования

Интерфейс с описанием конфигураций устройств, которые на данный момент установлены на устройствах, представлен на рисунке 18.

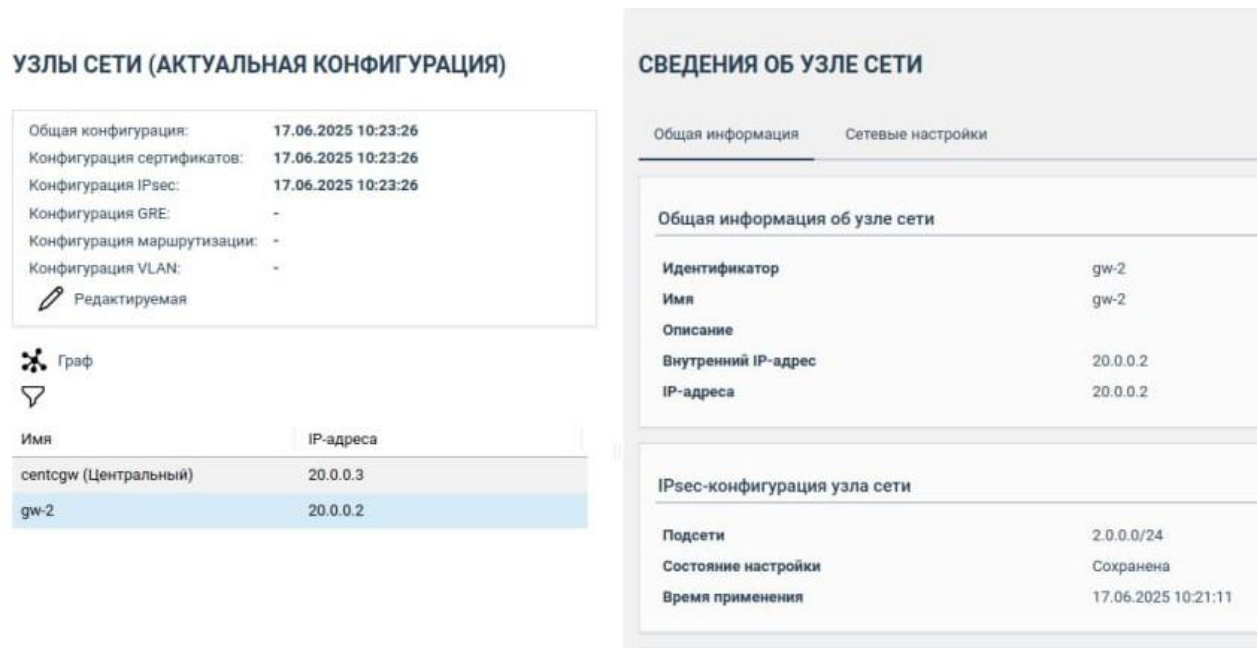


Рисунок 18 – Интерфейс с описанием установленных конфигураций

Таким образом был реализован модуль редактирования.

5 Тестирование системы

При тестировании функционала разрабатываемой системы использовались методы как ручного, так и автоматизированного тестирования. Для ручного тестирования был применен метод черного ящика. Для автоматизированного – фреймворк NUnit для создания Unit-тестов [19].

Тестирование методом черного ящика – это подход к тестированию программного обеспечения, при котором тестировщик оценивает функциональность приложения без знания его внутренней структуры или кода. При применении данного метода основное внимание уделяется входным данным и ожидаемым выходным результатам. Тестировщик создает тест-кейсы на основе требований и спецификаций, проверяя, соответствует ли поведение программы заданным критериям. Этот метод позволяет сосредоточиться на том, как система работает с точки зрения пользователя, и выявлять ошибки, которые могут быть незаметны при анализе внутренней работы программы [20].

5.1 Функциональное тестирование системы

При получении конфигураций OriginalConfiguration в модуле редактирования из модуля управления происходит их разбор для того, чтобы сформировать структуру с информацией об устройстве DeviceInfo и осуществлять редактирование содержимого конфигурации в дальнейшем.

Таблица 4 описывает проведенные тесты для разбора полученной конфигурации от модуля управления в модуле редактирования, в ней указаны какие входные данные конфигурации были получены и какие сообщения о несоответствии требованиям были добавлены в структуру устройства. В модуле установлена используемая версия конфигураций 1.0.

Таблица 4 – Описание проведенных тестов десериализации конфигурации

№	Входные данные конфигурации	Зафиксированные несоответствия требованиям в описании структуры информации устройства
1	содержимое: "" версия: 1.0	Конфигурация устройства не содержит данных
2	содержимое: "" версия: -1	Конфигурация устройства не содержит данных Конфигурация устройства имеет версию -1, отличающуюся от используемой 1.0
3	содержимое: "" версия: null	Конфигурация устройства не содержит данных Конфигурация устройства не содержит данных о версии
4	содержимое: null версия: 1.0	Конфигурация устройства не содержит данных
5	содержимое: null версия: -1	Конфигурация устройства не содержит данных Конфигурация устройства имеет версию -1, отличающуюся от используемой 1.0
6	содержимое: null версия: null	Конфигурация устройства не содержит данных Конфигурация устройства не содержит данных о версии
7	содержимое: "null" версия: 1.0	При десериализации конфигурации устройства получено пустое значение
8	содержимое: "null" версия: -1	При десериализации конфигурации устройства получено пустое значение Конфигурация устройства имеет версию -1, отличающуюся от используемой 1.0
9	содержимое: "null" версия: null	При десериализации конфигурации устройства получено пустое значение Конфигурация устройства не содержит данных о версии
10	содержимое: ""field": "some_value"" версия: 1.0	-

Продолжение таблицы 4

№	Входные данные конфигурации	Зафиксированные несоответствия требованиям в описании структуры информации устройства
11	содержимое: “”field”: “some_value”” версия: -1	Конфигурация устройства имеет версию -1, отличающуюся от используемой 1.0
12	содержимое: “”field”: “some_value”” версия: null	Конфигурация устройства не содержит данных о версии
13	содержимое: “”invalid_field”: 1” версия: 1.0	Конфигурация устройства содержит неверный формат данных
14	содержимое: “”invalid_field”: 1” версия: -1	Конфигурация устройства содержит неверный формат данных Конфигурация устройства имеет версию -1, отличающуюся от используемой 1.0
15	содержимое: “”invalid_field”: 1” версия: null	Конфигурация устройства содержит неверный формат данных Конфигурация устройства не содержит данных о версии
16	содержимое: ”not_json” версия: 1.0	Конфигурация устройства содержит неверный формат данных
17	содержимое: “not_json” версия: -1	Конфигурация устройства содержит неверный формат данных Конфигурация устройства имеет версию -1, отличающуюся от используемой 1.0
18	содержимое: “not_json” версия: null	Конфигурация устройства содержит неверный формат данных Конфигурация устройства не содержит данных о версии

Исходя из результатов проведенного ручного тестирования можно сказать, что разработанная система успешно его прошла.

5.2 Модульное тестирование

Также для десериализации конфигураций были разработаны Unit-тесты для класса `ConfigurationParser`, исходный код которых приведен в листинге Б.1 в приложении Б.

В таблице 5 приведено описание разработанных Unit-тестов для десериализации конфигурации.

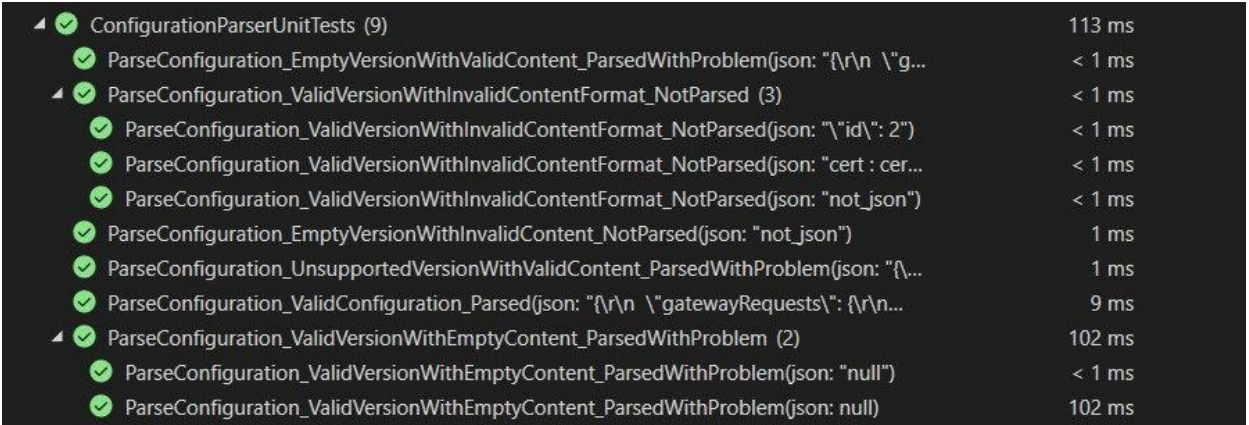
Таблица 5 – Описание Unit-тестов для десериализации конфигурации

Тест	Описание	Ожидание	Актуальный результат
<code>ParseConfiguration_EmptyVersionWithInvalidContent_Error</code>	Проверка десериализации конфигурации с неуказанной версией и некорректным содержимым. Статус разбора <code>Error</code>	Пройден	Пройден (рисунок 30)
<code>ParseConfiguration_ValidVersionWithInvalidContentFormat_Error</code>	Проверка десериализации конфигурации с некорректной версией и некорректным содержимым. Статус разбора <code>Error</code>	Пройден	Пройден (рисунок 30)
<code>ParseConfiguration_ValidVersionWithEmptyContent_Warning</code>	Проверка десериализации конфигурации с корректной версией и пустым содержимым. Статус разбора <code>Warning</code>	Пройден	Пройден (рисунок 30)
<code>ParseConfiguration_EmptyVersionWithValidContent_Warning</code>	Проверка десериализации конфигурации с пустой версией и корректным содержимым. Статус разбора <code>Warning</code>	Пройден	Пройден
<code>ParseConfiguration_UnsupportedVersionWithValidContent_Warning</code>	Проверка десериализации конфигурации с некорректной версией и корректным содержимым. Статус разбора <code>Warning</code>	Пройден	Пройден

Продолжение таблицы 5

Тест	Описание	Ожидание	Актуальный результат
ParseConfiguration_ValidConfiguration_Ok	Проверка десериализации конфигурации с корректной версией и корректным содержимым. Статус разбора Ok	Пройден	Пройден

Результат выполненных Unit-тестов для класса десериализации конфигурации представлен на рисунке 19.



ConfigurationParserUnitTests (9)	113 ms
ParseConfiguration_EmptyVersionWithValidContent_ParsedWithProblem(json: "{\r\n \"g...	< 1 ms
ParseConfiguration_ValidVersionWithInvalidContentFormat_NotParsed (3)	< 1 ms
ParseConfiguration_ValidVersionWithInvalidContentFormat_NotParsed(json: "\"id\": 2")	< 1 ms
ParseConfiguration_ValidVersionWithInvalidContentFormat_NotParsed(json: "cert : cer...	< 1 ms
ParseConfiguration_ValidVersionWithInvalidContentFormat_NotParsed(json: "not_json")	< 1 ms
ParseConfiguration_EmptyVersionWithInvalidContent_NotParsed(json: "not_json")	1 ms
ParseConfiguration_UnsupportedVersionWithValidContent_ParsedWithProblem(json: "{\r...	1 ms
ParseConfiguration_ValidConfiguration_Parsed(json: "{\r\n \"gatewayRequests\": {\r\n...	9 ms
ParseConfiguration_ValidVersionWithEmptyContent_ParsedWithProblem (2)	102 ms
ParseConfiguration_ValidVersionWithEmptyContent_ParsedWithProblem(json: "null")	< 1 ms
ParseConfiguration_ValidVersionWithEmptyContent_ParsedWithProblem(json: null)	102 ms

Рисунок 19 – Результаты Unit-тестов для класса десериализации конфигурации

Для разработанной системы было также проведено модульное тестирование контроллеров для обработки запросов к модулю редактирования с помощью NUnit, исходный код тестов приведен в листингах Б.2 и Б.3 в приложении Б.

В таблице 6 приведено описание разработанных Unit-тестов для контроллера DeviceController, который используется для получения информации о конфигурациях с изменениями, внесенными в рамках текущей сессии, а также для применения и отмены внесенных изменений в рамках текущей сессии.

Таблица 6 – Описание Unit-тестов контроллера DeviceController

Тест	Описание	Ожидание	Актуальный результат
GetDevice	Проверка получения информации о конкретном устройстве с конфигурацией, содержащей внесенные изменения	Пройден	Пройден (рисунок 30)
GetDevices	Проверка получения информации о всех устройствах с конфигурациями, содержащими внесенные изменения	Пройден	Пройден (рисунок 30)

Результат выполненных Unit-тестов для контроллера DeviceController представлен на рисунке 20.

DeviceControllerUnitTests (4)	131 ms
GetDevices	4 ms
GetDevice (3)	127 ms
GetDevice(id: "1", exists: True)	< 1 ms
GetDevice(id: "4", exists: False)	3 ms
GetDevice(id: "3", exists: True)	124 ms

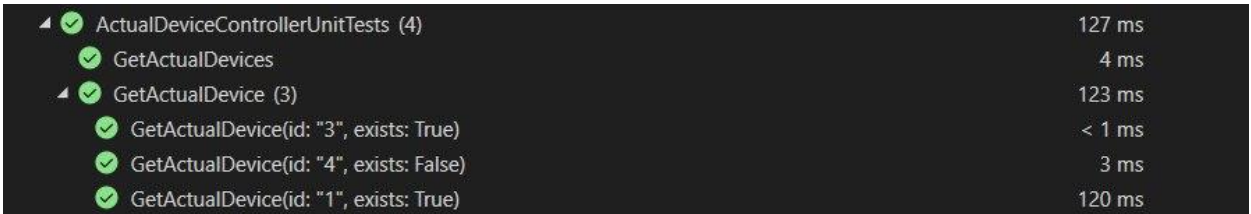
Рисунок 20 – Результаты Unit-тестов для контроллера DeviceController

В таблице 7 приведено описание разработанных Unit-тестов для контроллера ActualDeviceController.

Таблица 7 – Описание Unit-тестов контроллера ActualDeviceController

Тест	Описание	Ожидание	Актуальный результат
GetActualDevice	Проверка получения информации о конкретном устройстве с конфигурацией, установленной на устройстве	Пройден	Пройден (рисунок 30)
GetActualDevices	Проверка получения информации о всех устройствах с конфигурациями, установленными на устройствах	Пройден	Пройден (рисунок 30)

Результат выполненных Unit-тестов для контроллера ActualDeviceController представлен на рисунке 21.



ActualDeviceControllerUnitTests (4)	127 ms
GetActualDevices	4 ms
GetActualDevice (3)	123 ms
GetActualDevice(id: "3", exists: True)	< 1 ms
GetActualDevice(id: "4", exists: False)	3 ms
GetActualDevice(id: "1", exists: True)	120 ms

Рисунок 21 – Результаты Unit-тестов для контроллера ActualDeviceController

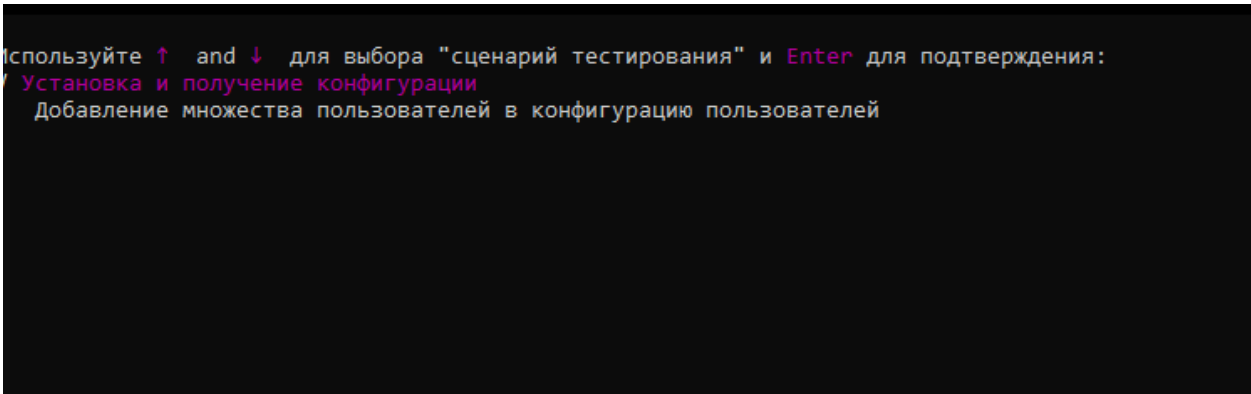
Исходя из результатов проведенного модульного тестирования можно сказать, что разработанная система успешно его прошла.

5.3 Интеграционное тестирование

Для тестирования применения конфигурации на управляемом сетевом устройстве была разработана консольная утилита с интеграционными тестами.

В утилите был реализован класс сценария тестирования ConfigurationSettingScenario, код которого приведен в приложении В.1, по применению на устройстве, путем создания клиента сервера ConfigurationManagementServer, работающего на Unix-сокете. Сценарий предполагает формирование конфигурации для устройства и ее отправки с помощью метода IConfigurationManagementService.UpdateConfiguration.

На рисунке 22 представлен консольный интерфейс утилиты интеграционных тестов.

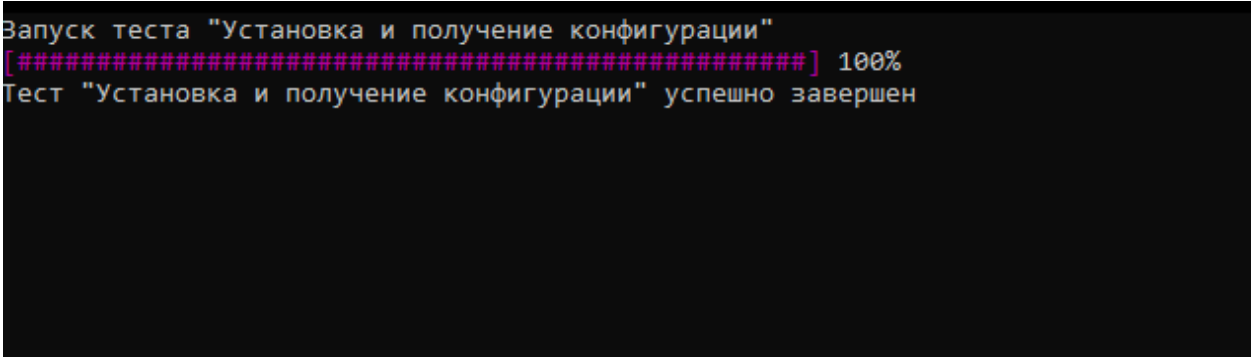


```

Используйте ↑ and ↓ для выбора "сценарий тестирования" и Enter для подтверждения:
/ Установка и получение конфигурации
  Добавление множества пользователей в конфигурацию пользователей
  
```

Рисунок 22 – Консольный интерфейс утилиты интеграционных тестов

На рисунке 23 представлен результат выполнения интеграционного теста по применению конфигурации на модуль конфигурации.



```

Запуск теста "Установка и получение конфигурации"
[#####] 100%
Тест "Установка и получение конфигурации" успешно завершен

```

Рисунок 23 – Консольный интерфейс утилиты интеграционных тестов

Исходя из результатов проведенного интеграционного тестирования можно сказать, что разработанная система успешно его прошла.

Полный код утилиты интеграционных тестов приведен в приложении В.

5.4 Метрики качества кода

Также был проведен анализ качества кода реализованной системы удаленного управления конфигурацией сетевых устройств, в ходе которого было проведено вычисление следующих метрик:

- количественная метрика LoC (количество строк кода);
- метрика Холстеда HV (вычислительная сложность);
- цикломатическая сложность CC (количество ветвлений);
- индекс удобства поддержки MI;

Для рассмотрения данных метрик был использован класс ConfigurationParser в качестве примера.

Для вычисления количественной метрики LoC было подсчитано количество строк, занимаемого исходным кодом класса ConfigurationParser и получено значение LoC равное 101.

Для вычисления метрики Холстеда были определены словари операторов и операндов, используемые в исходном коде класса ConfigurationParser. В таблице 8 приведен словарь операторов для класса ConfigurationParser.

Таблица 8 – Словарь операторов

Операторы	Число вхождений
=	24
.	41
?.	8
()	41
{}	6
if	5
return	3
try/catch	1
&&	1
!=	1
>	1
!	1
var	3
new	11

В таблице 9 приведен словарь операндов для класса ConfigurationParser.

Таблица 9 – Словарь операндов

Операнды	Число вхождений
parsingDetails	13
problems	14
_logger	9
configuration	5
version	7
id	10

Значение количества операторов в словаре η_1 равно 14, значение количества операндов в словаре η_2 равно 6. Общее значение словаря вычисляется по формуле:

$$\eta = \eta_1 + \eta_2,$$

где η_1 – словарь операторов;

η_2 – словарь операндов;

η – общий словарь.

Значение словаря равно:

$$\eta = 14 + 6 = 20.$$

Общее количество использований элементов словаря операторов N_1 равно 147, а общее количество использований элементов словаря операндов N_2 равно 58. Общее количество использований элементов словаря вычисляется по формуле:

$$N = N_1 + N_2,$$

где N_1 – количество использований элементов словаря операторов;

N_2 – количество использований элементов словаря операндов;

N – общее количество использований элементов словаря.

Количество использований элементов словаря равно:

$$N = 147 + 58 = 205$$

Метрика Холстеда вычисляется по формуле:

$$HV = N \log_2 \eta,$$

где HV – вычислительная сложность;

N – общее количество использований элементов словаря;

η – общий словарь.

Таким образом, значение метрики Холстеда будет равно 886.

Для определения цикломатической сложности CC необходимо найти количество ветвлений в исходном коде приложения. Для класса `ConfigurationParser` количество ветвлений равно 5. На основе вычисленных метрик исходного кода можно определить индекс удобства поддержки кода MI .

Данное значение вычисляется по формуле:

$$MI = \max \left(0, (171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LoC)) * \frac{100}{171} \right),$$

где HV – вычислительная сложность (Halstead Volume);

CC – цикломатическая сложность (Cyclomatic Complexity);

LoC – количество строк кода (Lines of Code).

Тогда, индекс удобства поддержки будет равен:

$$MI = (171 - 35 - 1.15 - 75) * \frac{100}{171} = 35.$$

Нормой индекса удобства поддержки является значение от 20 до 100. В данном случае значение индекса равно 35, что означает, что исходный код приложения достаточно удобен для поддержки.

Заключение

Результатом выполнения выпускной квалификационной работы стали разработанные программные средства удаленного управления конфигурацией сетевых устройств.

Также был проведен анализ существующих решений для удаленного управления конфигурацией сетевых устройств и сравнены характеристики этих решений в рамках этапа анализа требований и предметной области. В ходе анализа были выявлены основные преимущества и недостатки аналоговых решений, на основе которых были определены требования, которым должна соответствовать разрабатываемая система.

Был осуществлен функционально-стоимостной анализ, в ходе которого был определен бюджет и затраты на разработку. В результате анализа было определено, что разработка системы займет 41 день, и затраты на ее реализацию составят сумму 122 672 рублей.

В ходе проектирования были разработаны диаграммы классов для каждого модуля системы и диаграмма развертывания всей системы в нотации UML.

Для написания модулей использовался язык C#, а также инструмент Apache Thrift для удаленного вызова процедур между разными модулями и элементами системы.

Для разработанной системы было осуществлено функциональное, интеграционное и модульное тестирование, в ходе которых ошибок работы системы обнаружено не было. Также корректную работу разработанной системы подтверждает ее внедрение в технологическую платформу в компании «ИнфоТеКС», что подтверждается актом о внедрении, приведенном в приложении Г.

Список использованных источников

1. Хохштайн Л. Ansible. Подробное руководство по автоматизации администрирования. – СПб.: Питер, 2019. – 416 с.
2. Ansible. Документация [Электронный ресурс]. – URL: <https://docs.ansible.com/ansible/latest/index.html/> – (Дата обращения: 16.04.2025).
3. Маршалл М. Puppet. Эффективное управление инфраструктурой. – СПб.: Питер, 2018. – 400 с.
4. Puppet. Официальная документация [Электронный ресурс]. – URL: <https://puppet.com/ru/docs/> – (Дата обращения: 16.04.2025).
5. Прокди К. Chef. Управление инфраструктурой. – М.: ДМК Пресс, 2018. – 328 с.
6. Хаусенблас М. SaltStack. Практическое руководство. – М.: ДМК Пресс, 2017. – 280 с.
7. SaltStack: статьи на Habr [Электронный ресурс]. – URL: <https://habr.com/ru/search/?q=saltstack> – (Дата обращения: 16.04.2025).
8. Шеремет А. Д., Ковалев А. П. Функционально-стоимостной анализ: Учебное пособие / Под ред. А. Д. Шеремета. – М.: Экономический факультет МГУ имени М. В. Ломоносова, 2017. – 204 с.
9. Гантт, Г. Планирование работы. Диаграммы Ганта / Генри Л. Гантт. – М.: Либроком, 2010. – 128 с.
10. Методика «Функционально-стоимостной анализ и имитационное моделирование» [Электронный ресурс]. – URL: https://www.businessstudio.ru/articles/article/metodika_funktsionalno_stoimostno_y_analiz_i_imitats/ (Дата обращения: 17.04.2025).
11. Прошкина Е. Н. Язык моделирования информационных процессов UML 2.0 (учебное пособие) / Прошкина Е. Н., Балашова И. Ю. – Пенза, Издательство ПГУ, 2018. – 112 с.

12. Диаграмма вариантов использования [Электронный ресурс]. – URL: <http://www.intuit.ru/studies/courses/32/32/lecture/1004> (дата обращения 21.05.2025).
13. Гудхарт Б., Кокс Дж. Linux. Программирование системных приложений. – СПб.: БХВ-Петербург, 2018. – 832 с.
14. Свенсон Э. Thrift. Кросс-языковое взаимодействие сервисов. – М.: ДМК Пресс, 2020. – 256 с.
15. Мартин Р., Мартин М. Принципы, паттерны и методики гибкой разработки на языке C# / Мартин Р., Мартин М. – пер. с англ. – СПб: Символ-Плюс, 2011. – 768 с.
16. Microsoft Docs. Dictionary<TKey, TValue>.NET [Электронный ресурс]. – URL: <https://learn.microsoft.com/ru-ru/dotnet/api/system.collections.generic.dictionary-2/> (Дата обращения: 17.04.2025).
17. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2022. – 367 с.
18. Принципы разработки интерфейса [Электронный ресурс]. – URL: <http://www.cmsmagazine.ru/library/items/usability/principles-of-user-interface-design/> (дата обращения: 08.06.2025).
19. Виды тестирования ПО [Электронный ресурс]. – URL: http://wiki.rosalab.ru/ru/index.php/Виды_тестирования_ПО (Дата обращения: 08.06.2025).
20. Классификация тестирования по степени автоматизации [Электронный ресурс]. – URL: <https://qastart.by/class/vidi-testirovaniya-m/40-po-stepeni-avtomatizatsii> (Дата обращения: 08.06.2025).

Приложение А
(справочное)
Глоссарий

Json – легковесный формат обмена данными.

C# – строго типизированный объектно-ориентированный язык программирования общего назначения.

СУБД – система управления базами данных.

IPsec – набор протоколов для защиты IP-трафика.

Приложение Б
(обязательное)
Исходный код приложения

Листинг Б.1 – Класс Unit-тестов для ConfigurationParser

```

public class ConfigurationParserUnitTests
{
    [Theory]
    [JsonFileContent("Configuration.json")]
    public void ParseConfiguration_ValidConfiguration_Ok(string json)
    {
        // Подготовка
        const string id = "1";
        var parser = _factory.Build();

        // Действие
        (var parsedConfiguration, var parsingDetails) =
        parser.ParseConfiguration(json, ConfigurationVersionInfo.UsedVersion, id);

        // Проверка
        Assert.NotNull(parsedConfiguration);
        Assert.Equal(ConfigurationParsingStatus.Ok, parsingDetails.Status);
        Assert.Null(parsingDetails.Problems);
    }

    [Theory]
    [JsonFileContent("Configuration.json")]
    public void
    ParseConfiguration_UnsupportedVersionWithValidContent_Warning(string json)
    {
        // Подготовка
        const string id = "1";
        var parser = _factory.Build();

        // Действие
        (var parsedConfiguration, var parsingDetails) =
        parser.ParseConfiguration(json, new Version(), id);

        // Проверка
        Assert.NotNull(parsedConfiguration);
        Assert.Equal(ConfigurationParsingStatus.Warning,
        parsingDetails.Status);
        Assert.NotNull(parsingDetails.Problems);
        Assert.True(parsingDetails.Problems.Count == 1);
        Assert.Contains(parsingDetails.Problems, problem => problem.Status ==
        (int)ConfigurationParsingProblemStatus.UnsupportedVersion);
    }

    [Theory]
    [JsonFileContent("Configuration.json")]
    public void
    ParseConfiguration_EmptyVersionWithValidContent_Warning(string json)
    {
        // Подготовка
        const string id = "1";
        var parser = _factory.Build();

        // Действие
        (var parsedConfiguration, var parsingDetails) =
        parser.ParseConfiguration(json, null, id);

        // Проверка
        Assert.NotNull(parsedConfiguration);
    }
}

```

```

        Assert.Equal(ConfigurationParsingStatus.Warning,
parsingDetails.Status);
        Assert.NotNull(parsingDetails.Problems);
        Assert.True(parsingDetails.Problems.Count == 1);
        Assert.Contains(parsingDetails.Problems, problem => problem.Status ==
(int)ConfigurationParsingProblemStatus.EmptyVersion);
    }

    [Theory]
    [InlineData("null")]
    [InlineData(null)]
    public void
ParseConfiguration_ValidVersionWithEmptyContent_Warning(string? json)
    {
        // Подготовка
        const string id = "1";
        var parser = _factory.Build();

        // Действие
        (var parsedConfiguration, var parsingDetails) =
parser.ParseConfiguration(json, ConfigurationVersionInfo.UsedVersion, id);

        // Проверка
        Assert.NotNull(parsedConfiguration);
        Assert.Equal(ConfigurationParsingStatus.Warning,
parsingDetails.Status);
        Assert.NotNull(parsingDetails.Problems);
        Assert.True(parsingDetails.Problems.Count == 1);
        Assert.Contains(parsingDetails.Problems, problem => problem.Status ==
(int)ConfigurationParsingProblemStatus.EmptyContent);
    }

    [Theory]
    [InlineData("not_json")]
    [InlineData("incorrect : incorrect")]
    [InlineData("\"id\": 2")]
    public void
ParseConfiguration_ValidVersionWithInvalidContentFormat_Error(string json)
    {
        // Подготовка
        const string id = "1";
        var parser = _factory.Build();

        // Действие
        (var parsedConfiguration, var parsingDetails) =
parser.ParseConfiguration(json, ConfigurationVersionInfo.UsedVersion, id);

        // Проверка
        Assert.NotNull(parsedConfiguration);
        Assert.Equal(ConfigurationParsingStatus.Error,
parsingDetails.Status);
        Assert.NotNull(parsingDetails.Problems);
        Assert.True(parsingDetails.Problems.Count == 1);
        Assert.Contains(parsingDetails.Problems, problem => problem.Status ==
(int)ConfigurationParsingProblemStatus.InvalidContent);
    }

    [Theory]
    [InlineData("not_json")]
    public void
ParseConfiguration_EmptyVersionWithInvalidContent_Error(string json)
    {

```

```

        // Подготовка
        const string id = "1";
        var parser = _factory.Build();

        // Действие
        (var parsedConfiguration, var parsingDetails) =
        parser.ParseConfiguration(json, null, id);

        // Проверка
        Assert.NotNull(parsedConfiguration);
        Assert.Equal(ConfigurationParsingStatus.Error,
        parsingDetails.Status);
        Assert.NotNull(parsingDetails.Problems);
        Assert.True(parsingDetails.Problems.Count == 2);
        Assert.Contains(parsingDetails.Problems, problem => problem.Status ==
        (int)ConfigurationParsingProblemStatus.InvalidContent);
        Assert.Contains(parsingDetails.Problems, problem => problem.Status ==
        (int)ConfigurationParsingProblemStatus.EmptyVersion);
    }

    private readonly ConfigurationParserFactory _factory = new();
}

```

Листинг Б.2 – Класс Unit-тестов для контроллера DeviceController

```

public class DeviceControllerUnitTests
{
    [Theory]
    [InlineData("1", true)]
    [InlineData("3", true)]
    [InlineData("4", false)]
    public async void GetDevice(string id, bool exists)
    {
        // Подготовка
        var configurations = new Dictionary<string, OriginalConfiguration>
        {
            { "1", new() },
            { "2", new() },
            { "3", new() }
        };

        var controller = await BuildAsync(configurations);

        // Действие
        var answer = await controller.GetDeviceInfo(id);

        // Проверка
        if (exists)
            ControllerAssert.OkObject<DeviceInfo>(answer);
        else
            ControllerAssert.NotFoundProblem(answer);
    }

    [Fact]
    public async void GetDevices()
    {
        // Подготовка
        const int devicesCount = 3;
        var configurations = new Dictionary<string, OriginalConfiguration>
        {
            { "1", new() },
            { "2", new() },
            { "3", new() }
        };
    }
}

```

```

};

var controller = await BuildAsync(configurations);

// Действие
var answer = await controller.GetDeviceInfos();

// Проверка
var okResult = Assert.IsType<OkObjectResult>(answer);
var devices =
Assert.IsAssignableFrom<IEnumerable<DeviceInfo>>(okResult.Value);
Assert.Equal(devicesCount, devices.Count());
}

private async Task<DeviceController> BuildAsync(Dictionary<string,
OriginalConfiguration> configurations)
{
    _provider.Factory.ConfigurationCommunicationClient.Setup(configurations);

    var controller = _provider.Factory.Build();

    await
    _provider.Factory.DeviceConfigurationSynchronizator.StartAsync(new());

    return controller;
}

private readonly DeviceControllerTestsProvider _provider = new();
}

```

Листинг Б.3 – Класс Unit-тестов для контроллера ActualDeviceController

```

public class ActualDeviceControllerUnitTests
{
    [Theory]
    [InlineData("1", true)]
    [InlineData("3", true)]
    [InlineData("4", false)]
    public async void GetActualDevice(string id, bool exists)
    {
        // Подготовка
        var configurations = new Dictionary<string, OriginalConfiguration>
        {
            { "1", new() },
            { "2", new() },
            { "3", new() }
        };

        var controller = await BuildAsync(configurations);

        // Действие
        var answer = await controller.GetActualDeviceInfo(id);

        // Проверка
        if (exists)
            ControllerAssert.OkObject<ActualDeviceInfo>(answer);
        else
            ControllerAssert.NotFoundProblem(answer);
    }

    [Fact]

```

```

public async void GetActualDevices()
{
    // Подготовка
    const int devicesCount = 3;
    var configurations = new Dictionary<string,
GatewayCertificatesConfiguration>
    {
        { "1", new() },
        { "2", new() },
        { "3", new() }
    };

    var controller = await BuildAsync(configurations);

    // Действие
    var answer = await controller.GetActualDevicesInfos();

    // Проверка
    var okResult = Assert.IsType<OkObjectResult>(answer);
    var nodes =
Assert.IsAssignableFrom<IEnumerable<ActualDeviceInfo>>(okResult.Value);
    Assert.Equal(devicesCount, nodes.Count());
}

private async Task<ActualDeviceController> BuildAsync(Dictionary<string,
GatewayCertificatesConfiguration> configurations)
{
    _factory.ConfigurationCommunicationClient.Setup(configurations);

    var controller = _factory.Build();

    await _factory.DeviceConfigurationSynchronizator.StartAsync(new());

    return controller;
}

private readonly ActualDeviceControllerFactory _factory = new();
}

```

Приложение В
(рекомендуемое)
Исходный код интеграционных тестов

Листинг В.1 – Класс сценария применения конфигурации

```
public class ConfigurationSettingScenario : ITestScenario
{
    public ConfigurationSettingScenario()
    {
        _configModuleClient = new ThriftClient<ConfigurationManagementService.Client>(_InternalServicePort,
            p => new ConfigurationManagementService.Client(p));

        _internalClient = new ThriftClient<InternalService.Client>(_UsersServicePort,
            p => new InternalService.Client(p));
    }
    /// <inheritdoc>
    public string Description { get; } = "Установка и получение конфигурации";

    /// <inheritdoc>
    public async Task InitAsync()
    {
        // Очистка базы для корректной установки конфигурации
        await _internalClient.Execute(c => c.Clear());

        var config = FakesFactory.GetConfiguration();

        Configuration.Version = new Version() { Major = 1, Minor = 0 };
        Configuration.Content = JsonSerializer.Serialize(config, SerializationUtils.Options);
    }

    /// <inheritdoc>
    public async Task ExecuteAsync()
    {
        await _configModuleClient.Execute(c => c.UpdateConfiguration(Configuration));

        var configuration = await _configModuleClient.Execute(c => c.GetConfiguration());
    }

    private OriginalConfiguration Configuration { get; set; } = new();

    private readonly ThriftClient<ConfigurationManagementService.Client> _configModuleClient;

    private readonly ThriftClient<InternalService.Client> _internalClient;

    private const int _ConfigModulePath = "/run/ConfigModule/config.sock";

    private const int _InternalServicePath = "/run/ConfigModule/internal.sock";
}
```

Листинг В.2 – Интерфейс сценария

```
/// <summary>
/// Интерфейс сценария тестирования.
/// </summary>
public interface ITestScenario
{
    /// <summary>
    /// Описание теста.
    /// </summary>
    public string Description { get; }

    /// <summary>
    /// Инициализация сценария тестирования.
    /// </summary>
    public Task InitAsync();

    /// <summary>
```

```

    /// Выполнение сценария тестирования.
    /// </summary>
    public Task ExecuteAsync();
}

```

Листинг В.3 – Вспомогательные методы работы с консолью

```

public static class ConsoleUtils
{
    /// <summary>
    /// Показывает прогресс в консоли.
    /// </summary>
    /// <param name="current">Текущее значение.</param>
    /// <param name="total">Максимальное значение.</param>
    public static void ShowProgress(int current, int total)
    {
        int width = 50;
        int filled = current * width / total;
        string bar = "\u001b[95m[" + new string('#', filled) + new string('-', width - filled) + "]";
        Console.Write($"{bar}\u001b[0m {current * 100 / total}%");
    }

    /// <summary>
    /// Получает выбранную опцию из консоли.
    /// </summary>
    /// <param name="description">Описание.</param>
    /// <param name="options">Опции.</param>
    /// <returns>Индекс выбранной опции.</returns>
    public static int GetChosenOption(string description, string[] options)
    {
        Console.Clear();
        Console.OutputEncoding = Encoding.Unicode;
        Console.CursorVisible = false;
        Console.WriteLine($"{description}\nИспользуйте \u001b[95m↑ \u001b[0m и \u001b[95m↓ \u001b[0m для выбора\n" +
            $"{description}\n" и \u001b[95mEnter\u001b[0m для подтверждения:");
        (int left, int top) = Console.GetCursorPosition();
        var optionIndex = 0;
        var decorator = "\u001b[95m";

        ConsoleKeyInfo key;
        bool isSelected = false;

        while (!isSelected)
        {
            Console.SetCursorPosition(left, top);

            for (int i = 0; i < options.Length; i++)
            {
                Console.WriteLine($"{(optionIndex == i ? decorator : " ")}{options[i]} \u001b[0m");
            }

            key = Console.ReadKey(false);

            switch (key.Key)
            {
                case ConsoleKey.UpArrow:
                    optionIndex = optionIndex == 0 ? options.Length - 1 : optionIndex - 1;
                    break;

                case ConsoleKey.DownArrow:
                    optionIndex = optionIndex == options.Length - 1 ? 0 : optionIndex + 1;
                    break;

                case ConsoleKey.Enter:
                    isSelected = true;

```

```

        break;
    }
}
return optionIndex;
}
}

```

Листинг В.4 – Класс для запуска сценариев тестов

```

public class TestScenariosRunner
{
    /// <summary>
    /// Сценарии тестов.
    /// </summary>
    public List<ITestScenario> TestScenarios { get; private set; } =
        new(_TestScenarioCount)
        {
            new ConfigurationSettingScenario(),
        };

    /// <summary>
    /// Запуск сценария теста.
    /// </summary>
    /// <param name="testScenarioId">Идентификатор сценария теста.</param>
    /// <param name="cancellationToken">Токен отмены.</param>
    public Task Run(int testScenarioId, Cancellation_token cancellationToken)
    {
        return RunWrapper(testScenarioId,
            async (ITestScenario scenario) =>
            {
                Console.Clear();

                var progressAnimator = new ProgressAnimator($"Выполнение теста
                \u001b[95m{scenario.Description.ToLowerInvariant()}\u001b[0m");
                progressAnimator.Update();

                while (!cancellationToken.IsCancellationRequested)
                {
                    await scenario.ExecuteAsync();
                    progressAnimator.Update();
                }
            });
    }

    /// <summary>
    /// Запуск теста заданное количество раз.
    /// </summary>
    /// <param name="testScenarioId">Идентификатор сценария теста.</param>
    /// <param name="executionCount">Количество запусков теста.</param>
    /// <param name="cancellationToken">Токен отмены.</param>
    public Task Run(int testScenarioId, int executionCount, Cancellation_token cancellationToken)
    {
        return RunWrapper(testScenarioId,
            async (ITestScenario scenario) =>
            {
                for (int i = 0; i < executionCount && !cancellationToken.IsCancellationRequested; i++)
                {
                    ConsoleUtils.ShowProgress(i, executionCount - 1);
                    await scenario.ExecuteAsync();
                }
            });
    }

    private async Task RunWrapper(int testScenarioId, Func<ITestScenario, Task> runner)
    {

```

```

var scenario = TestScenarios[testScenarioId];

Console.WriteLine($"Запуск теста \"{scenario.Description}\"");

await scenario.InitAsync();

await runner(scenario);

Console.WriteLine($"Завершение теста \"{scenario.Description}\"");
}

private const int _TestScenarioCount = 1;
}

```

Листинг В.5 – Класс для запуска консольной утилиты

```

class Program
{
    public static async Task Main(string[] args)
    {
        var testsRunner = new TestScenariosRunner();

        var scenarioDescriptions = testsRunner.TestScenarios.Select(s => s.Description).ToArray();

        while (true)
        {
            try
            {
                using var cancellationTokenSource = new CancellationTokencSource();

                ConsoleCancelEventHandler handler = (sender, e) =>
                {
                    e.Cancel = true;
                    cancellationTokenSource.Cancel();
                };

                Console.CancelKeyPress += handler;

                var scenarioOption = ConsoleUtils.GetChooosenOption("сценарий тестирования", scenarioDescriptions);

                var runModOption = ConsoleUtils.GetChooosenOption("сценарий запуска теста", _runModDescriptions);

                Console.Clear();

                switch (runModOption)
                {
                    case 0:
                        await testsRunner.Run(scenarioOption, cancellationTokenSource.Token);
                        break;
                    case 1:
                        Console.WriteLine("Введите количество запусков теста: ");
                        var executionCount = int.Parse(Console.ReadLine() ?? "1");
                        await testsRunner.Run(scenarioOption, executionCount, cancellationTokenSource.Token);
                        break;
                    default:
                        throw new NotImplementedException();
                }

                Console.CancelKeyPress -= handler;

                Thread.Sleep(3000);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Ошибка: {ex.Message}: {ex}");
                throw;
            }
        }
    }
}

```

```
    }  
  }  
}  
  
private static string[] _runModDescriptions =  
  new string[_RunModCount]  
  {  
    "Непрерывное тестирование по выбранному сценарию",  
    "Спрашивать сценарий тестирования на каждый запуск"  
  };  
  
private const int _RunModCount = 2;  
}
```

Приложение Г
(рекомендуемое)
Акт о внедрении

infotecs

Акционерное общество
«Информационные технологии
и коммуникационные системы»
(АО «ИнфоТеКС»)

Юридический адрес: ул. Мишина,
дом 56, стр. 2, этаж 2, пом. IX, комната 29
Москва, 127083

Почтовый адрес: ая № 80,
улица Отрадная, дом 2Б, стр. 1
Москва, 127273

Тел. (495) 737-61-92 Факс (495) 737-72-78

ОГРН 1027739185066
ИНН КПП 7710013769/771401001
<http://www.infotecs.ru>

Акционерное общество «Информационные
технологии и коммуникационные системы»
(АО «ИнфоТеКС»)

127083, Москва, улица Мишина, д. 56, стр. 2,
этаж 2, помещение IX, комната 29

.....17.06.2024..... №

Акт о внедрении

Настоящим подтверждаю, что Шахманов Данила Дмитриевич является сотрудником компании АО «ИнфоТеКС». Решения, описанные в выпускной квалификационной работе на тему «Программные средства удаленного управления конфигурацией сетевых устройств», вошли в состав разрабатываемой технологической платформы.

Выдано Шахманову Даниле Дмитриевичу для предъявления по требованию.

Руководитель обособленного
подразделения в г. Пенза



М.М. Грунтович