

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS**

**Разработка клиентского приложения для системного администратора в системе
удаленного контроля и технической поддержки локальной инфокоммуникационной
структуры организации**

Обучающийся / Student Новожилова Анна Владимировна

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет

инфокоммуникационных технологий

Группа/Group K34212

Направление подготовки/ Subject area 11.03.02 Инфокоммуникационные технологии и
системы связи

Образовательная программа / Educational program Программирование в
инфокоммуникационных системах 2019

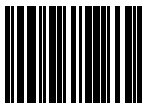
Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Руководитель ВКР/ Thesis supervisor Самохин Никита Юрьевич, Университет ИТМО,
факультет инфокоммуникационных технологий, ассистент (квалификационная категория
"ассистент")

Обучающийся/Student


Документ подписан	
Новожилова Анна Владимировна	
10.05.2023	

(эл. подпись/ signature)

Новожилова
Анна
Владимировна

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Самохин Никита Юрьевич	
10.05.2023	

(эл. подпись/ signature)

Самохин
Никита
Юрьевич

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Новожилова Анна Владимировна
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет
инфокоммуникационных технологий
Группа/Group K34212
Направление подготовки/ Subject area 11.03.02 Инфокоммуникационные технологии и
системы связи
Образовательная программа / Educational program Программирование в
инфокоммуникационных системах 2019
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Разработка клиентского приложения для системного
администратора в системе удаленного контроля и технической поддержки локальной
инфокоммуникационной структуры организации
Руководитель ВКР/ Thesis supervisor Самохин Никита Юрьевич, Университет ИТМО,
факультет инфокоммуникационных технологий, ассистент (квалификационная категория
"ассистент")

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Цель работы: повышение эффективности удаленного мониторинга и технической поддержки для трансформации рынка контроля за инфраструктурой организации посредством разработки клиентского приложения системного администратора программного комплекса для мониторинга сети «InBetween».

Техническое задание: моделирование и реализация клиентского приложения для системного администратора, являющегося частью программного комплекса удаленного мониторинга и технической поддержки «InBetween»

Задачи:

Изучить техническую литературу по теме исследования, и систематизировать полученную информацию

Составить техническое задание, определить основной функционал программного обеспечения

Разработать логическую модель системы

Изучить техническую документацию Python и PyQt6

Реализовать сетевое взаимодействие клиентского приложения с сервером системы

Реализовать API-клиент для взаимодействия программы с базой данных

Разработать функции для визуализации изменения параметров компьютеров во времени

Разработать графический интерфейс системы
Провести ручное тестирование итоговой программы и внести необходимые изменения

Рекомендуемые материалы:

1. Python 3.10 Documentation. - URL: <https://docs.python.org/3.10/>
2. Qt for Python Documentation. - URL: <https://doc.qt.io/qtforpython/>
3. S Lee, K Levanti, HS Kim. Network monitoring: Present and future. // Computer Networks. - 2014

Форма представления материалов ВКР / Format(s) of thesis materials:

Форма представления Приложений: программный код, конфигурационный файл сборки

Форма представления основных результатов: презентация, текст ВКР

Дата выдачи задания / Assignment issued on: 09.12.2022

Срок представления готовой ВКР / Deadline for final edition of the thesis 30.04.2023

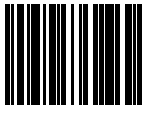
Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: да / yes

СОГЛАСОВАНО / AGREED:

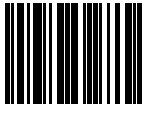
Руководитель ВКР/
Thesis supervisor

Документ подписан	
Самохин Никита Юрьевич	
30.03.2023	

(эл. подпись)

Самохин
Никита
Юрьевич

Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Новожилова Анна Владимировна	
09.04.2023	

(эл. подпись)

Новожилова
Анна
Владимировна

Руководитель ОП/ Head
of educational program

(эл. подпись)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Новожилова Анна Владимировна

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет

инфокоммуникационных технологий

Группа/Group K34212

Направление подготовки/ Subject area 11.03.02 Инфокоммуникационные технологии и системы связи

Образовательная программа / Educational program Программирование в инфокоммуникационных системах 2019

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Бакалавр

Тема ВКР/ Thesis topic Разработка клиентского приложения для системного администратора в системе удаленного контроля и технической поддержки локальной инфокоммуникационной структуры организации

Руководитель ВКР/ Thesis supervisor Самохин Никита Юрьевич, Университет ИТМО, факультет инфокоммуникационных технологий, ассистент (квалификационная категория "ассистент")

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Моделирование и разработка клиентского приложения для системного администратора, являющегося частью программного комплекса удаленного мониторинга и технической поддержки «InBetween».

Задачи, решаемые в ВКР / Research tasks

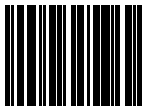
1) Исследование и сравнительный анализ систем мониторинга 2) Определение основных требований к ПО и его функционалу 3) Разработка логической модели системы 4) Реализация необходимых компонентов программы, функций и API 5) Разработка графический интерфейс системы

Краткая характеристика полученных результатов / Short summary of results/findings

В ходе выполнения практической части исследования была спроектирована логическая модель работы системы мониторинга сетей связи «InBetween». Разработанная структура взаимодействия всех элементов системы и диаграмма классов полностью соответствуют итоговому продукту. Приложение для системного администратора было написано на языке Python с использованием библиотеки для создания графических интерфейсов PyQt6. Оно может быть запущено с операционных систем Windows или Linux. Приложение позволяет

отслеживать состояние пользовательских компьютеров в локальной сети, управлять доступ к информации в системе мониторинга и взаимодействовать с пользователями системы через чат. Управление функциями приложения происходит через минимальный графический интерфейс. На момент написания отчёта система проходит тестирование в локальной сети МБОУ СОШ №2 поселка Марково. Таким образом, все задачи этого исследования выполнены, а цели достигнуты.

Обучающийся/Student

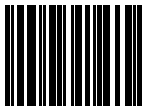
Документ подписан	
Новожилова Анна Владимировна	
10.05.2023	

(эл. подпись/ signature)

Новожилова
Анна
Владимировна

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Самохин Никита Юрьевич	
10.05.2023	

(эл. подпись/ signature)

Самохин
Никита
Юрьевич

(Фамилия И.О./ name
and surname)

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....	3
СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	7
ВВЕДЕНИЕ.....	8
1 ИСТОРИЯ И АНАЛИЗ СИСТЕМ МОНИТОРИНГА.....	10
1.1 История развития систем мониторинга	10
1.2 Принципы работы систем мониторинга	13
1.3 Анализ существующих систем мониторинга.....	14
2 МОДЕЛИРОВАНИЕ И РАЗРАБОТКА.....	24
2.1 Логическое проектирование приложения.....	24
2.1.1 Формирование требований к приложению	24
2.1.2 Создание блок-схемы работы приложения	24
2.1.3 Разработка диаграммы активностей	26
2.1.4 Разработка диаграммы классов	27
2.2 Организация хранения данных	29
2.3 Разработка приложения	31
ЗАКЛЮЧЕНИЕ	51
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	53

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Keyring	Сервис, предназначенный для безопасного хранения имен пользователей и паролей. Конфиденциальные данные хранятся в зашифрованном виде и пользователю достаточно вводить один мастер-пароль для получения к ним доступа.
Агент	Программа, которая вступает в отношения посредничества с пользователем или другой программой
Аутентификация	Процедура проверки личности пользователя программы, осуществляемая с помощью сравнения введенной информации с информацией, сохраненной в базе данных
База данных	Упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе
Библиотека	Сборник подпрограмм или объектов, используемых для разработки программного обеспечения [6]
Блок-схема	Тип схем, описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединённых между собой линиями, указывающими направление последовательности.
Виджет	Примитив графического интерфейса пользователя, имеющий стандартный внешний вид и выполняющий стандартные действия. [5]
Графический интерфейс	Система средств взаимодействия пользователя с электронными устройствами, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана [5]
Датакласс	Декоратор для классов в языке Python, предоставляющий

	функции для автоматического добавления сгенерированных специальных методов в определяемые пользователем классы. [6]
Декоратор	Функция, возвращающая другую функцию, обычно применяется для преобразования исходной функции [6]
Диаграмма активности	UML-диаграмма, на которой показана спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов [4]
Диаграмма классов	Структурная диаграмма языка UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними [4]
Класс	Шаблон для создания пользовательских объектов. Классы обычно содержат определения методов, которые работают с экземплярами класса. [7]
Клиент	Аппаратный или программный компонент вычислительной системы, посылающий запросы серверу [8]
Коммутатор	Устройство, предназначенное для соединения нескольких узлов компьютерной сети в пределах одного или нескольких сегментов сети.
Конструктор	Метод класса, вызываемый сразу после создания экземпляра класса [7]
Конфигурация	Совокупность настроек программы, задаваемая пользователем, а также процесс изменения этих настроек в соответствии с нуждами пользователя
Лог	Файл с записями о событиях в хронологическом порядке
Маршрутизатор	Специализированное устройство, которое пересылает

	пакеты между различными сегментами сети на основе правил и таблиц маршрутизации.
Метрика	Неформатированные данные об использовании ресурсов или поведении, которые можно отслеживать и собирать в системах
Мониторинг	Процесс постоянного отслеживания компьютерной сети на наличие медленных или неисправных компонентов, проверка состояния параметров сети, в том числе параметров качества предоставления сервиса [2]
Плагин	Независимо компилируемый программный модуль, динамически подключаемый к основной программе и предназначенный для расширения и/или использования её возможностей
Поле	Переменная, создающаяся при описании класса [7]
Протокол	Набор правил, задающих форматы сообщений и процедуры, которые позволяют компьютерам и прикладным программам обмениваться информацией
Сервер	Программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам. [8]
Сетевая инфраструктура	Совокупность специального оборудования и программного обеспечения, создающего основу для эффективного обмена информацией
Сеть связи	Технологическая система, включающая средства и линии связи, предназначенные для передачи всевозможной информации
Система мониторинга	Программа или программный комплекс, который позволяет осуществлять мониторинг сети

Топология сети	Это конфигурация графа, вершинам которого соответствуют конечные узлы сети, а рёбрам — физические или информационные связи между вершинами [9]
Триггер	Инструмент позволяющий оценить собранные данные и на основании настроенных условий, определить имеет ли данный элемент данных ошибки или определённые проблемы.
Фреймворк	программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

AD – Windows Active Directory

API – Application programming interface

CPU – Central Processing Unit

ICMP – Internet Control Message Protocol

IP – Internet Protocol

IT – Information Technology

JSON – JavaScript Object Notation

LDAP – Lightweight Directory Access Protocol

MPLS – Multiprotocol Label Switching

RMON – Remote Network Monitoring

SNMP – Simple Network Management Protocol

UML – Unified Modeling Language

USB – Universal Serial Bus

VoIP – Voice over Internet Protocol

VPN – Virtual Private Network

WAN – Wide Area Network

ВВЕДЕНИЕ

В современном мире сети связи играют важную роль во многих областях, от бизнеса до личного пользования. В связи с этим, обеспечение надежности и эффективности сетей связи является одним из главных приоритетов для технических специалистов. Для достижения этой цели необходимо постоянно осуществлять мониторинг и анализировать работу сетей связи. Для эффективного мониторинга и анализа используются различные системы мониторинга, которые позволяют оперативно обнаруживать и устранять проблемы в работе сетей связи.

Рынок мониторинговых систем для сетей связи постоянно растет, появляются все новые и новые продукты. Так, на 2022 год рынок систем мониторинга США оценивается в 2,2 миллиарда долларов, а по прогнозам аналитического агентства Markets and Markets к 2027 году эта сумма достигнет 3 миллиардов долларов [1].

Тем не менее, при разработке новых продуктов внимание чаще уделяется расширению функционала и гибким настройкам мониторинга, позволяющим осуществлять полномасштабное наблюдение за большими и сложными сетями. Популярны системы для мониторинга, такие как, например, Zabbix и Nagios, отличаются большим количеством разнообразных функций и способностью поддерживать большие, высоконагруженные сети. Однако в России в последние годы наблюдается тенденция к развитию малого и среднего бизнеса. Так, по данным Министерства экономики Российской Федерации доля малого бизнеса в ВВП выросла больше, чем на 10 процентов с 2017 года. Компьютерные сети, с которыми работают представители малого бизнеса, обычно не отличаются размером или сложной топологией, поэтому значительная часть функционала популярных систем мониторинга будет для них избыточной и неактуальной. Более того, предприятия, работающие на государственной поддержке, такие, как школы и поликлиники, а также небольшие компании очень часто не имеют возможностей нанять квалифицированного системного администратора,

который будет способен разобраться в сложных настройках системы мониторинга.

Таким образом, актуальность разработки простой в использовании системы для мониторинга, которая не будет требовать первичной настройки для полноценной работы, сложно переоценить. Введение такой системы в эксплуатацию позволит сократить затраты на содержание отдела системного администрирования и уменьшить нагрузку на специалистов, отвечающих за работу сети.

Целью данной работы является моделирование и разработка клиентского приложения для системного администратора, являющегося частью программного комплекса удаленного мониторинга и технической поддержки «InBetween».

Для достижения этой цели необходимо решить следующие задачи:

- 1) Исследовать и провести сравнительный анализ систем мониторинга
- 2) Определить основные требования к программному обеспечению и его функционал
- 3) Разработать логическую модель системы
- 4) Реализовать необходимые компоненты, функции и API
- 5) Разработать графический интерфейс системы

1 ИСТОРИЯ И АНАЛИЗ СИСТЕМ МОНИТОРИНГА

1.1 История развития систем мониторинга

Первая компьютерная сеть, способная объединять в себе устройства, находящиеся в относительной удаленности друг от друга была создана в 1969 году Агентством Министерства обороны США. Она получила название ARPANET и стала прототипом сети Интернет, использующейся в настоящее время. Системы мониторинга сетевой инфраструктуры начали появляться практически сразу же, как эта технология начала получать распространение.

Первые инструменты для контроля сети появились в 1980-тых годах. В то время системы мониторинга сетей были примитивными и управлялись через командную строку. Они предоставляли возможности для проверки доступности устройств и сервисов в сети. Наиболее популярными инструментами мониторинга сетей в те времена были SNMP (Simple Network Management Protocol) и ICMP (Internet Control Message Protocol). ICMP был разработан в 1981 году и предназначен для проверки доступности устройств в сети. Он позволял отправлять пакеты данных на устройства в сети и получать ответы от этих устройств для проверки доступности. SNMP был создан в 1988 году и стал первым протоколом управления сетями для мониторинга состояния устройств и сервисов в сети. Он позволял сетевым администраторам получать информацию о состоянии сетевых устройств, таких как маршрутизаторы, коммутаторы, серверы и другие устройства, и анализировать эту информацию для обнаружения проблем.

Эти инструменты мониторинга сетей в 1980-х годах были довольно ограниченными, но они создали основу для развития более продвинутых систем мониторинга в последующие десятилетия.

В 1990-х годах системы мониторинга сетей продолжали развиваться, и появились новые инструменты и технологии, которые расширили возможности мониторинга и управления сетью. Одним из ключевых инструментов мониторинга сетей в 1990-х годах был протокол RMON (Remote Network Monitoring). RMON был разработан как расширение

протокола SNMP и позволял собирать дополнительную информацию о состоянии сети, такую как данные о трафике, использовании пропускной способности и другие параметры. RMON был очень популярен в 1990-х годах и до сих пор используется в системах мониторинга сетей. В 1990-х годах также появились первые системы мониторинга сетей с графическим интерфейсом. Эти системы предоставляли удобный способ отображения и анализа данных о состоянии сети, что значительно упрощало работу с ними. Кроме того, в это время стали появляться специализированные системы мониторинга сетей, которые были нацелены на определенные сферы применения. Например, системы мониторинга сетей для мониторинга трафика в сетях передачи данных (WAN), системы мониторинга сетей для мониторинга производительности виртуальных частных сетей (VPN) и т.д.

В 2000-х годах произошел значительный рост в области сетевых технологий, что привело к увеличению сложности и размеров сетей. Было введено множество новых протоколов, в том числе IPv6, MPLS, VoIP и другие. Это привело к необходимости создания более сложных систем мониторинга, которые бы могли отслеживать эти новые протоколы и технологии. Одним из ключевых трендов этого десятилетия было появление систем мониторинга сетей с функцией автоматического обнаружения устройств и топологии сети. Это позволило администраторам сети автоматически находить новые устройства, отслеживать их и связи между ними, а также уведомлять об изменениях в топологии. Другой важной тенденцией стало увеличение возможностей мониторинга производительности приложений. Разработчики систем мониторинга начали предоставлять более точные и подробные отчеты о производительности приложений, которые могут помочь выявить проблемы и улучшить качество работы приложений. В это время также начали активно использоваться облачные технологии, что привело к появлению новых требований к системам мониторинга. Были созданы системы мониторинга, способные отслеживать производительность облачных ресурсов

и обнаруживать проблемы в облачных средах. Кроме того, в этот период стали появляться более сложные системы мониторинга, которые могут анализировать данные многих источников, включая логи, параметры сети и трассировку сетевых пакетов. Это помогло снизить время реакции на проблемы в сети и улучшить производительность и безопасность сети.

В 2010-х годах системы мониторинга сетей стали еще более распространенными и продвинутыми. Вместе с развитием облачных технологий, многие системы мониторинга начали переходить на облачные платформы. Кроме того, расширение возможностей сетевого оборудования и протоколов также повлияло на развитие систем мониторинга сетей. Одной из самых популярных систем мониторинга сетей в 2010-х годах стал Nagios, которая по-прежнему является одной из наиболее используемых систем мониторинга в настоящее время. Nagios была выпущена в 1999 году, но в 2010-х годах получила новый толчок развития и стала более гибкой и функциональной. Кроме Nagios, в 2010-х годах появилось множество других систем мониторинга сетей, таких как Zabbix, Cacti, Zenoss и другие. Они предоставляют различные функции, от простого мониторинга до более продвинутых возможностей, таких как анализ данных и прогнозирование. Также в 2010-х годах появилось множество облачных систем мониторинга сетей, таких как Amazon CloudWatch, Microsoft Azure Monitor и Google Cloud Monitoring. Эти системы предоставляют возможности мониторинга облачных сервисов и инфраструктуры, а также могут использоваться для мониторинга локальных сетей и приложений.

Сегодня системы мониторинга сетей имеют широкий спектр функций и возможностей, включая отслеживание доступности и производительности устройств и сервисов, анализ потока данных, управление событиями, оповещение об авариях и многое другое. Эти инструменты являются критически важными для поддержания надежности и производительности сетей в современном мире информационных технологий.

1.2 Принципы работы систем мониторинга

Мониторинг компьютерной сети — это процесс постоянного отслеживания компьютерной сети на наличие медленных или неисправных компонентов, проверка состояния показателей, в том числе метрик качества предоставления сервиса [2]. Соответственно, система мониторинга — это программа или программный комплекс, который позволяет осуществлять данное отслеживание. Основными возможностями, которые предоставляют системы мониторинга сетей связи, являются [3]:

- Сбор показателей с сетевых устройств – важных параметров, значение которых может значительно повлиять на работоспособность отдельного узла или всей сети, такие как загрузка CPU, температура аппаратных компонентов, доступность узла сети, количество потребляемой оперативной памяти и т.д. Эти показатели необходимо регулярно отслеживать.
- Агрегирование и анализ метрик. Современные системы мониторинга сетевой инфраструктуры не только собирают параметры состояния со всех узлов в сети (либо с тех, на которых установлен агент системы), но и имеют возможность объединять и анализировать их, представляя затем результаты анализа в виде графических элементов (графиков, гистограмм, диаграмм и т.д.).
- Система оповещений. При достижении одного или нескольких показателей критических, недопустимых значений, системы мониторинга отправляют сигнал системному администратору через электронную почту, СМС, мессенджер или уведомление на телефон. Наиболее продвинутые системы также обладают механизмами самовосстановления, что в некоторых ситуациях позволяет восстановить узел сети без вмешательства человека.

Несмотря на то, что мониторинг сети может осуществляться и в примитивном виде (например, периодические проверки сетевой доступности устройств с помощью утилиты ping), в настоящее время системы мониторинга чаще всего представляют собой сложную информационную систему, состоящую из нескольких компонентов

(см. Рисунок 1). На удаленные узлы в сети устанавливаются специальные программы: агенты. Именно агенты позволяют собирать большое количество различных метрик с серверов, сетевого оборудования и стационарных компьютеров. Агенты обмениваются информацией с сервером, на котором хранится база данных и обрабатываются сигналы в случае, если показатели достигли недопустимых значений. Информация из базы данных, в свою очередь, отображается на интерфейсе в виде графиков и сообщений. Интерфейс может быть представлен в виде веб-приложения, программы или скрипта для командной строки.

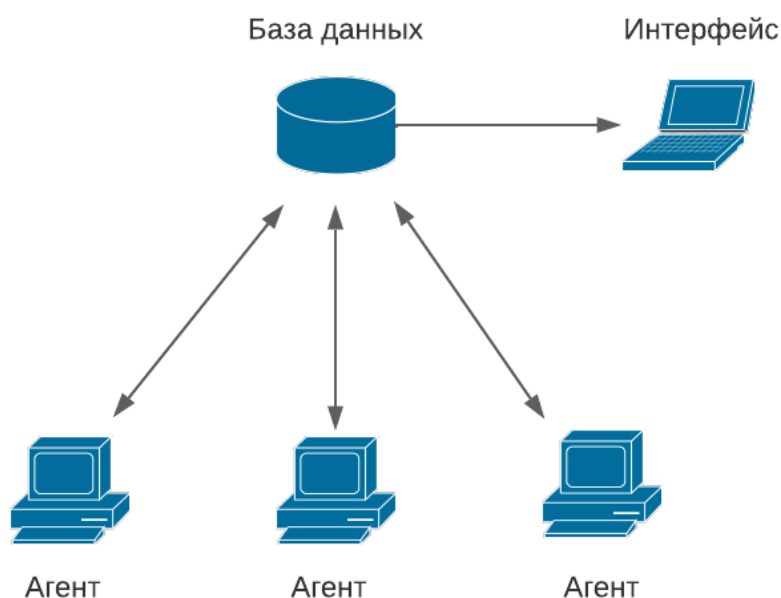


Рисунок 1 – Структура системы мониторинга

1.3 Анализ существующих систем мониторинга

Целью данного этапа было изучить наиболее популярные в русскоязычном сегменте системы мониторинга, сравнить их между собой, выделить недостатки и преимущества, чтобы затем использовать эти сведения при разработке функциональных требований к собственной системе мониторинга.

В сравнительном анализе принимались во внимание только наиболее популярные на рынке стран СНГ продукты. Это связано с тем,

что при выборе инструмента для контроля сетевой инфраструктуры компании обычно не рассматривают малоизвестные программные комплексы, предпочитая сразу же ориентироваться на лидеров рынка. Принимая во внимание этот факт, можно считать включение в работу непопулярных систем мониторинга нецелесообразным.

Для того чтобы отобрать объекты для сравнительного анализа было изучено содержание 50 статей о выборе системы мониторинга в русскоязычном сегменте сети Интернет. Всего в статьях встречались упоминания о 30 различных программных комплексах. Наиболее часто встречались упоминания о таких системах, как Zabbix, Nagios, Icinga, SolarWinds, Observium, PRTG Network Monitor, Prometheus и Cacti.

Следующим этапом при выборе систем мониторинга для анализа была сортировка отмеченных выше систем по популярности, опираясь на количество запросов в поисковой системе Яндекс. Выбор поисковой системы обусловлен следующими факторами:

- Нацеленность на русскоязычную аудиторию. Так как данная работа рассматривает системы мониторинга, наиболее популярные в русскоязычном сегменте сети Интернет, то анализ количества запросов следует проводить в поисковом сервисе, который нацелен на пользователей из России и СНГ.

- Наличие удобного инструмента. Яндекс предоставляет бесплатный инструмент Вордстат для анализа запросов в своих сервисах, которым можно пользоваться без регистрации и ограничений.

После получения результатов в сервисе Вордстат, они были отсортированы по убыванию количества запросов и представлены в виде таблицы (см. Таблицу 1).

Как видно из полученных результатов, наибольшей популярностью в России и СНГ пользуются такие системы мониторинга, как Zabbix, Cacti, Prometheus и Nagios. Именно они будут выступать в качестве объектов анализа.

Таблица 1 – Количество запросов о системах мониторинга в Яндексе

Система мониторинга	Количество запросов за последний месяц
Zabbix	103 380
Cacti	72 311
Prometheus	29 167
Nagios	2714
SolarWinds	2110
PRTG Network Monitor	586
Icinga	531
Observium	431

Так как решающую роль при выборе системы мониторинга предприятия и обычные пользователи отдают именно функционалу, подходящему под нужды бизнеса, в критерии не были включены технические характеристики продуктов, такие как язык разработки, используемая база данных и прочее. Тем не менее, если какая-либо особенность архитектуры оказывает влияние на работоспособность программы, это будет учтено при выставлении оценки.

Одним из наиболее важных критериев при выборе системы мониторинга можно считать качество визуализации, так как именно графики и гистограммы дают более наглядное и интуитивное представление о состоянии системы. Наличие качественных и подробных визуализаций позволит значительно сократить время на анализ ситуации в сети и поиск проблемы. Еще один критерий, который также может сильно повлиять на скорость обнаружения и устранения неполадок – это возможность системы мониторинга предоставлять наглядные и детальные отчёты о происшествиях. Наиболее ценной считается система, которая способна формировать отчеты для пользователей, которые не обладают высокой квалификацией в сфере ИТ-технологий. Также не менее важной является безопасная передача данных

системой. При работе с собираемой информацией система мониторинга должна хранить, передавать и обрабатывать ее с минимальным риском к потере, повреждению или утечке. Эти три функции необходимы при работе на предприятии любого профиля и размера, без них система мониторинга не может полноценно выполнять свои функции, поэтому в процентном соотношении всех параметров им уделено наибольшее значение (см. Таблицу 2).

Далее следует упомянуть критерии, которые не играют решающей роли для отдельного пользователя, но могут проявиться при установке системы мониторинга на предприятия среднего и крупного бизнеса. К ним относятся поддержка различных типов устройств и приложений, способность собирать большое количество разнообразных показателей и готовность системы к работе с большими, высоконагруженными сетями. Эти критерии оценки хоть и не являются экстренно необходимыми всем компаниям, оказывают значительное влияние на выбор продукта.

К последней категории критериев относятся параметры, которые чаще всего не влияют на общую работоспособность системы, но при этом являются приятным дополнением, которое ускоряет и упрощает работу отдела системных администраторов. К таким критериям относятся системы оповещения пользователей, способность к прогнозированию трендов для метрик и простота первичной настройки. Эти критерии не оказывали большого влияния на конечную оценку продукта при сравнительном анализе, но все же имели значение при почти одинаковых результатах остальных параметров (см. Таблицу 2).

Итоговая шкала оценивания приведена в Таблице 2.

Далее был проведен сравнительный анализ выбранных систем с использованием определенных ранее критериев. Далее будут пояснены выставленные оценки. Итоговый результат анализа представлен в Таблице 3.

Таблица 2 – Шкала оценивания для сравнительного анализа

Критерий оценки	Минимальное значение	Максимальное значение	Доля в итоговой оценке
Качество визуализации	0	10	0,20
Качество отчетности	0	10	0,15
Безопасность системы	0	10	0,20
Поддержка устройств разного типа	0	10	0,1
Количество метрик	0	10	0,1
Работа в нагруженных сетях	0	10	0,1
Система оповещения	0	10	0,05
Способность к прогнозированию	0	10	0,05
Простота первичной настройки	0	10	0,05
Общая оценка	0	100	1

Качество визуализации

Наиболее разнообразными инструментами визуализации среди рассматриваемых систем обладает Zabbix, предоставляя возможность не только строить графики по результатам сбора метрик, но и создавать пользовательские информационные панели, размещая на них только необходимую информацию. Cacti тоже обладает сильным инструментом визуализации, позиционируя это, как свою основную функцию, однако

разнообразие графиков не так велико, как в Zabbix. Также Cacti не предоставляет возможностей для визуального сравнения данных из разных источников.

В Prometheus и Nagios визуализация данных не предусмотрена совсем, и может быть получена либо с установкой дополнительного плагина NagVis, в случае с Nagios, либо установкой стороннего программного обеспечения Grafana.

Качество отчетности

Prometheus и Zabbix являются лидерами в категории составления отчётности. Функции Prometheus позволяют не только составлять отчёты по техническим метрикам, но и собирать данные об успешности бизнес-процессов с помощью настройки специальных встроенных программ. Недостатком отчётов Zabbix является то, что они предоставляются в достаточно вольном формате и не являются документом, который можно предоставить коллеге или пользователю. Nagios в свою очередь предоставляет очень ограниченные отчеты, а Cacti и вовсе не имеет такой функции.

Безопасность системы

Во время анализа данных в сети Интернет не было обнаружено информации о каких-либо серьёзных уязвимостях в системах мониторинга Zabbix и Prometheus. Имеется информация о незначительных ошибках, но они исправляются достаточно быстро. Более того, официальная документация систем содержит информацию о способах повышения безопасности их работы. Cacti является относительно стабильной системой, однако часть используемых системой серверов устарела и является желанной целью для атак злоумышленников. Система Nagios не считается безопасной и активно критикуется сообществом IT-специалистов. Существуют открытые инструкции о том, как обойти защиту Nagios, а скрипты для взлома находятся в публичном доступе.

Поддержка устройств

В плане поддержки устройств все системы находятся примерно на одном уровне, так как имеют очень широкий спектр устройств и сервисов, мониторинг которых можно осуществлять. Все, что невозможно мониторить по умолчанию, может быть добавлено с плагинами.

Количество метрик

Системы мониторинга Nagios и Zabbix умеют оперировать любыми необходимыми форматами данных – числовые, строковые, JSON-объекты, файлы и так далее. Prometheus, в свою очередь, не позиционирует себя, как система для хранения логов, а потому, за редким исключением, собирает только численные и строковые метрики без возможности собрать файлы логов или JSON-объекты. Это связано с тем, что для хранения данных Prometheus использует временные ряды и словари значений. Cacti же не имеет варианта со словарями, а потому по умолчанию позволяет сохранять только числовые значения.

Работа в высоконагруженных сетях и масштабируемость

При работе в крупных корпорациях часто приходится иметь дело с наблюдением за несколькими тысячами машин. Zabbix прекрасно справляется с работой в больших сетях и регулярно становится темой для обсуждения на конференции о высоконагруженных сетях Highload. Prometheus тоже хорошо работает даже при резком увеличении размеров сети, но для этого требуется установка дополнительных плагинов. В системе мониторинга Nagios показатели гораздо хуже. Несмотря на имеющиеся примеры поддержки сетей из 15 тысяч узлов, Nagios является очень статичной системой и любые изменения требуют их внесения в конфигурационные файлы вручную. Хуже всего справляется Cacti, так как даже на официальном сайте не дает информации о поддержке более, чем нескольких сотен машин.

Система оповещения

Система оповещений присутствует по всем рассматриваемым программам для мониторинга, что уже делает администрирование сети проще и мобильнее. Cacti и Nagios могут отправлять уведомления по смс и электронной почте, причем в Cacti для этого требуется отдельный плагин. Zabbix и Prometheus также имеют интеграцию с популярными мессенджерами, такими как Slack и Telegram.

Способность к прогнозированию

Прогнозирование появления аномалий на основе анализа предыдущих значений метрик имеется во всех системах, кроме Cacti, по умолчанию. Cacti не имеет данной функции, однако ее можно дополнительно установить с помощью плагинов.

Простота первичной настройки

Nagios славится в сообществе сложностью своей настройки, требующей от пользователя глубоких знаний системного администрирования и программирования. Почти вся настройка осуществляется через конфигурационные файлы, а возможности интерфейса ограничены. Cacti также в основном настраивается без использования графического интерфейса, но за счет большей стабильности система является более простой в настройке. Zabbix и Prometheus предоставляют графический интерфейс практически для любой настройки, однако требуют усилий и времени на первичную настройку всех триггеров и информационных панелей со стороны системного администратора.

Таблица 3 – Результаты сравнительного анализа

Критерий	Zabbix	Cacti	Prometheus	Nagios
1	2	3	4	5
Качество визуализации	9	7	2	3
Качество отчетности	8	0	10	4

Продолжение Таблицы 3

1	2	3	4	5
Безопасность системы	10	6	10	2
Поддержка устройств разного типа	8	8	8	8
Количество метрик	10	4	7	10
Работа в высоконагруженных сетях	10	2	8	5
Система оповещения	10	6	10	8
Способность к прогнозированию	10	5	10	10
Простота первичной настройки	7	4	7	0
Общая оценка после применения весов	9,15	4,75	7,55	4,8

По результатам сравнительного анализа лучшей системой оказался программный комплекс Zabbix, набрав 9,15 баллов из 10 возможных. Zabbix считается наиболее популярной на текущий момент времени системой для мониторинга сети, и является универсальным инструментом. Он позволяет осуществлять наблюдение не только за работой серверов, стационарных компьютеров и сетевого оборудования, но и контролировать IoT-устройства, сервисы, веб-сайты и собственные приложения. Zabbix регулярно обновляется и имеет большое сообщество разработчиков и пользователей. Prometheus может послужить альтернативой для Zabbix, так как единственным выявленным недостатком этой системы стало

отсутствие полноценной визуализации метрик, что легко решается синхронизацией с Grafana. Обе упомянутые системы мониторинга могут рассматриваться в качестве примера качественного продукта при дальнейшей разработке.

Однако, другим значительным результатом анализа существующих систем мониторинга стало наблюдение об отсутствии среди популярных на рынке простых в использовании и настройке инструментов. При поиске подходящего решения в сети Интернет, в результатах не фигурирует программное обеспечение, позволяющее осуществлять мониторинг сразу же после установки программы на компьютер и не требующее от пользователя навыков системного администрирования. Можно предположить, что подобное решение на рынке отсутствует или же недостаточно распространено. В связи с этим, разработка программного комплекса, позволяющего осуществлять наблюдение за небольшой организацией и ее контроль даже без специфических знаний в сфере технологий, представляется актуальной и имеющей практическое применение. С учетом указанной информации в данной работе при разработке системы мониторинга упор был сделан на простоту наблюдения за сетевой инфраструктурой организации и управления ею, а также отсутствие необходимости в первичной настройке.

2 МОДЕЛИРОВАНИЕ И РАЗРАБОТКА

2.1 Логическое проектирование приложения

2.1.1 Формирование требований к приложению

Анализ преимуществ и недостатков различных систем мониторинга позволил выработать требования к функциональным возможностям приложения, к которым относятся:

- Аутентификация пользователя с помощью ключа
- Ограничение доступа к приложению при отсутствии ключа
- Просмотр всех устройств, на которых установлен агент
- Просмотр информации об устройстве, включая статус в сети, сетевое имя, идентификатор материнской платы, характеристики аппаратного обеспечения, операционную систему
- Просмотр логов за определенный промежуток времени
- Сортировка логов по дате получения и типу
- Наблюдение за изменением нагрузки на процессор и оперативную память устройства во времени
- Связь с пользователями устройств посредством чата
- Отключение устройства от системы мониторинга
- Создание нового ключа для пользователя системы мониторинга
- Просмотр информации о ключе, включая имя пользователя, для которого он был создан, уникальный идентификатор, настройки доступа для данного ключа и его назначение
- Преобразование ключа в QR-код для дальнейшего использования в мобильном приложении для системного администратора
- Изменение настроек доступа для существующего ключа

2.1.2 Создание блок-схемы работы приложения

С опорой на составленный ранее список функциональных требований, была создана блок-схема работы приложения (см. Рисунок 2). При открытии приложения первым делом проверяется доступность сервера, на котором

хранятся и обрабатываются все данные. Если по какой-либо причине подключение к серверу невозможно осуществить, то приложение выводит на экран ошибку с возможностью повторить попытку подключения. Затем проверяется факт наличия предыдущего подключения, и если оно не обнаружено, перед полным доступом к функциям предложения необходимо будет ввести уникальный ключ. Ключ проверяется на наличие в базе данных и сохраняется на устройстве. Больше ввод ключа не потребуется. После проверки доступности сервера и аутентификации пользователя будет получен доступ ко всем экранам приложения. При открытии экрана с подробно информацией о пользователе будет открыто соединение с сервером, через которое осуществляется чат. Также этот экран подразумевает возможность посмотреть логи событий и визуализацию метрик.

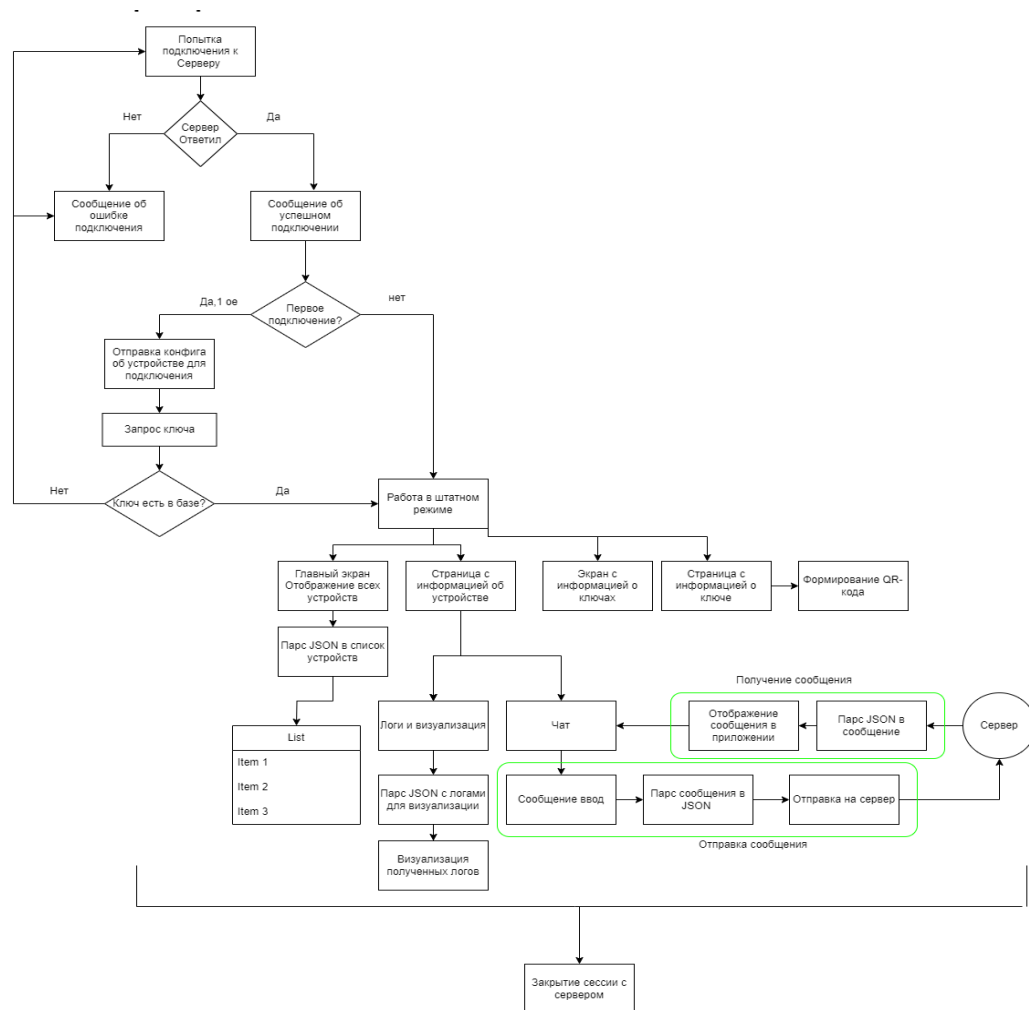


Рисунок 2 — Блок-схема работы приложения

2.1.3 Разработка диаграммы активностей

Затем была разработана диаграмма активности, демонстрирующая принцип взаимодействия всех компонентов системы мониторинга «InBetween» (см. Рисунок 3). При рассмотрении диаграммы важно учесть, что взаимодействие рассматривается со стороны системного администратора, работающего с операторским программным обеспечением (на диаграмме оно обозначено как «Оператор»). Как видно, подавляющее большинство сетевых взаимодействий производится с участием сервера. Это связано с тем, что сервер является связующим звеном между элементами системы, контролирующим исходящие и входящие данные. Даже при обмене информацией с клиентом в чате все сообщения сначала поступают на сервер, сохраняются в базу данных и только затем отсылаются сервером адресату для последующего отображения. Подобный принцип, запрещающий прямое взаимодействие двух конечных узлов, позволяет синхронизировать информацию на всех устройствах и увеличить уровень безопасности системы.

Также на сервере расположена база данных, в которой хранится вся имеющаяся в системе информация. Это позволяет уменьшить объем памяти, которую приложение занимает в системе, так как информация не сохраняется локально на компьютере пользователя.

Единственными действиями пользователя, в которых не задействован сервер, являются отображение на экране QR-кода с зашифрованным ключом от базы данных для дальнейшего сканирования мобильным устройством и закрытие приложения. Это связано с тем, что при выполнении этих действий никакая новая информация в систему не сохраняется: операторское программное обеспечение не отправляет на сервер данные о том, работает оно или нет, а QR-код генерируется на основе уже ранее запрошенной приложением информации и сохраняется только на время открытия диалогового окна.

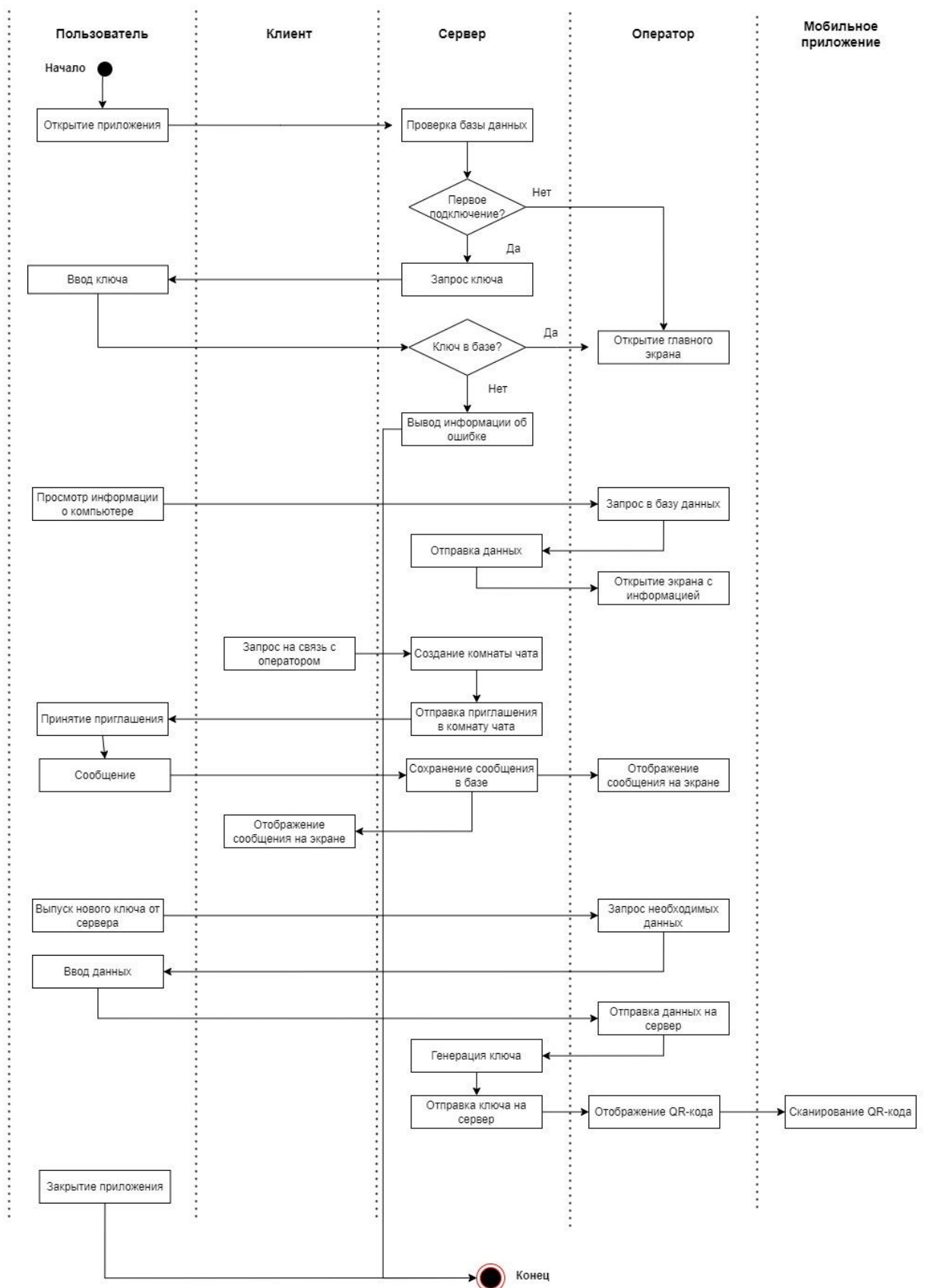


Рисунок 3 — Диаграмма активностей

2.1.4 Разработка диаграммы классов

Диаграмма классов — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы,

их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними [4]. Диаграмма классов является очень важным этапом проектирования, так как позволяет распланировать не только набор классов для создания, но также их содержимое и то, как они связаны между собой. Подобный этап планирования позволяет сократить время на разработку и количество правок, которые вносятся в код.

В ходе создания диаграммы (см. Рисунок 4) был определен 21 класс. К ним относятся 4 статических класса, которые предназначены только для использования в других частях программы и не предполагают создания отдельных экземпляров; 5 классов-виджетов, которые представляют собой элемент графического интерфейса программы, а их методы обычно предназначены для вывода на экран данных, передаваемых из других классов; 3 классов, предназначенных исключительно для хранения данных и не предполагающих создание методов для них. Остальные классы составляют основную структуру приложения и, предположительно, будут представлены в виде окон, которые обрабатывают и выводят на экран необходимую информацию. Некоторые такие классы не содержат полей, так как работают с информацией, получаемой из других элементов программы.

На диаграмме отображены 3 вида отношений между классами. Прямой линией обозначено отношение ассоциации, когда элементы существуют независимо друг от друга, но взаимодействуют в ходе работы приложения. Для таких отношений указана кратность. Белым ромбом обозначены отношения ассоциации: класс, в сторону которого направлен ромб, включает в себя второй класс, являющийся его частью, но при этом второй класс может также существовать независимо от первого. Черным ромбом обозначены отношения композиции или строгой ассоциации. В этом случае зависимый объект не может существовать без основного.

Основным классом, с которого начинается работа программы, является MainWindow, все остальные окна и виджеты, как и связанные с ними классы, открываются в ходе работы методов класса MainWindow.

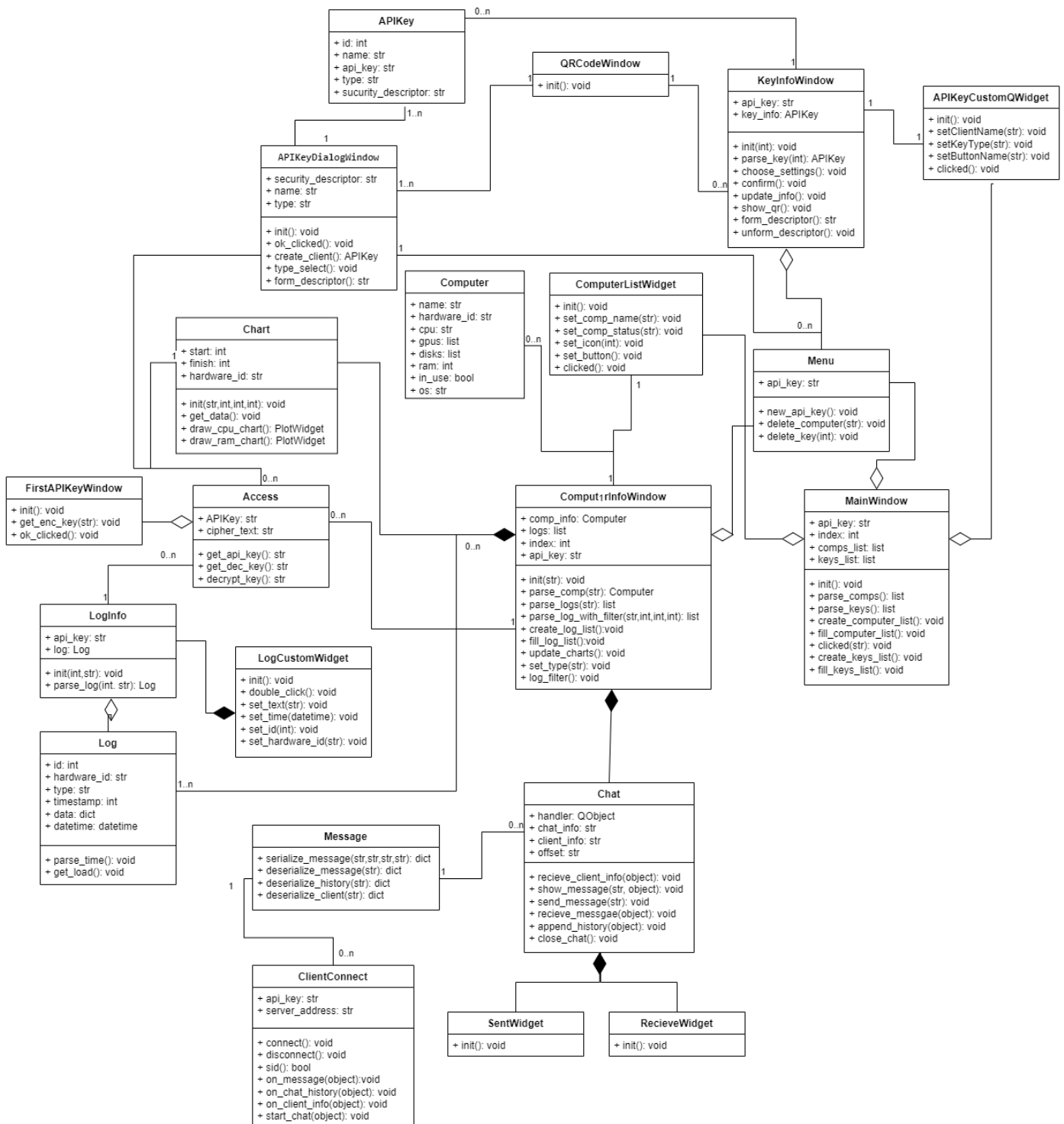


Рисунок 4 — Диаграмма классов

2.2 Организация хранения данных

Так как данные, получаемые пользователями приложения, должны быть одинаковыми и актуальными, было принято решение хранить их в базе

данных на сервере. Доступ к данным клиенты получают по API или, в случае необходимости постоянного и незамедлительного обновления данных, посредством веб-сокетов.

На сервере вся информация хранится в реляционной базе данных, так как все элементы системы не только предполагают строгую, неизменную структуру, но и связаны между собой. В качестве базы данных было выбрано решение PostgreSQL, так как это бесплатный, регулярно обновляющийся инструмент, поддерживающий объектно-ориентированные функции и предлагающий пользователям большое количество типов данных для работы. Также PostgreSQL поддерживается многими операционными системами из семейств Linux, Windows, MacOS, а значит, конечный пользователь не будет ограничен в выборе оборудования.

Структура базы данных представлена на рисунке 5.

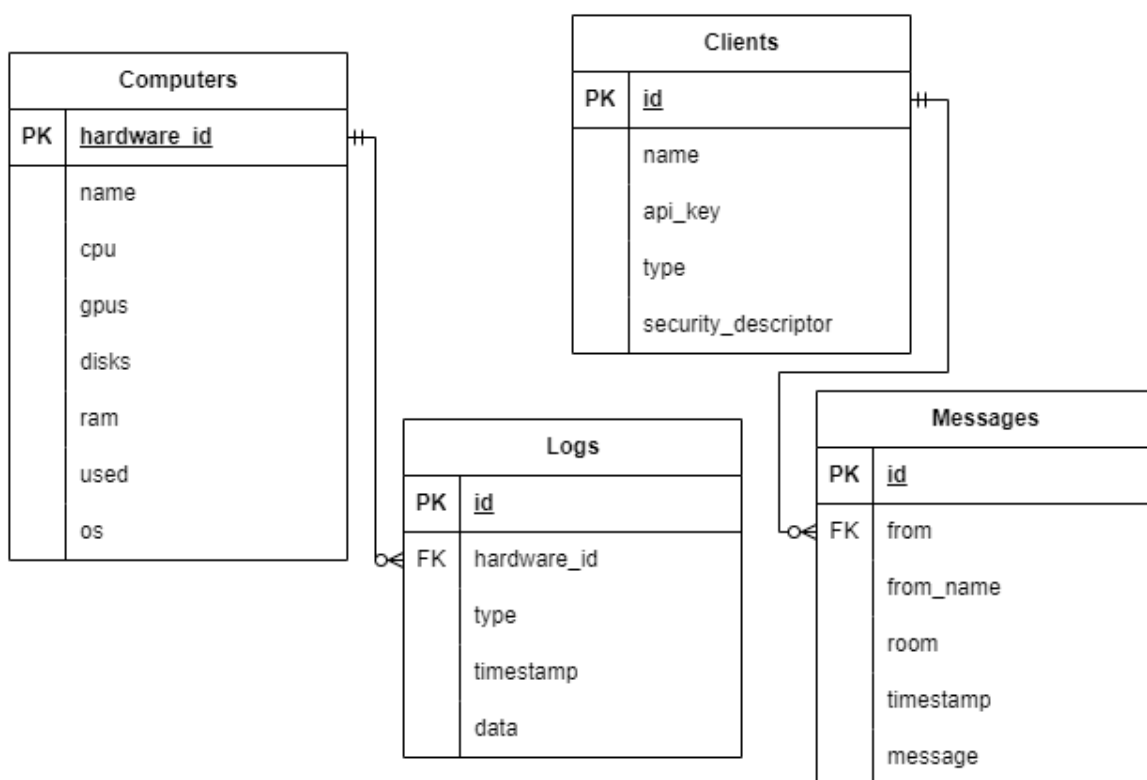


Рисунок 5 – Диаграмма базы данных

2.3 Разработка приложения

Разработка приложения велась на языке Python с использованием библиотеки для приложений с графическим интерфейсом PyQt6. Выбор языка обоснован следующими факторами:

- Синтаксис языка Python значительно проще и короче, чем остальных языков, которые рассматривались для разработки. Также этот язык менее чувствителен к ошибкам. В совокупности это позволяет сократить время на разработку продукта
- Для языка Python написано большое количество библиотек и фреймворков для визуализации данных, которая является основной функцией системы мониторинга
- Python хорошо подходит для анализа данных и построения математических моделей, что позволит в дальнейшем реализовать функцию прогнозирования метрик
- Графическая библиотека PyQT распространяется бесплатно с 1998 года, регулярно обновляется и дополняется, имеет большое сообщество разработчиков, как в русскоязычном, так и в англоязычном сегментах интернета

Так как система мониторинга непрерывно работает с сервером, хранящим большое количество информации о сети, в целях безопасности доступ к приложению для системного администратора предоставляется по специальному ключу. Этот же ключ позволяет получить доступ к базе данных. Когда пользователь впервые запускает приложение, открывается диалоговое окно, требующее ввести ключ доступа (см. Рисунок 7). Когда пользователь вводит известный ему ключ, начинается процесс шифрования и занесение зашифрованного ключа в базу паролей keyring (см. Рисунок 6). Эта технология позволяет безопасно сохранять пароли на устройстве, и повторное введение ключа не потребуется.

```
def enc_key_get(api_key):
    key = Fernet.generate_key()
    try:
        keyring.set_password('VKR_API_ENC_KEY', 'api_key', key.decode())
    except keyring.errors.KeyringError:
        print('Failed to access the keyring.')
    api_key = api_key.encode()
    cipher = Fernet(key)
    encrypted_api_key = cipher.encrypt(api_key)
    with open('encrypted_api_key.bin', 'wb') as f:
        f.write(encrypted_api_key)
```

Рисунок 6 — Шифрование и сохранение ключа

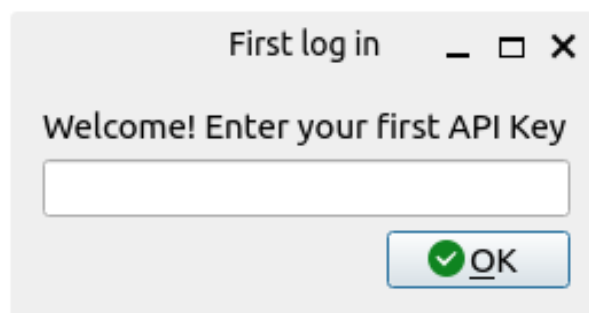


Рисунок 7 — Диалоговое окно ввода ключа

Работа программы начинается с главного экрана, представленного классом `MainWindow`. Этот класс наследует свойства от встроенного в библиотеку `PyQT6` класса `QMainWindow`. Главный экран представляет собой список, содержимое которого зависит от нажатия одной из трёх кнопок в верхней части экрана (см. Рисунок 8). Все три кнопки обрабатываются с помощью функции `clicked`, поведение которой зависит от переданного в качестве параметра сигнала (см. Рисунки 9-10). На экран выводятся список компьютеров в сети, оповещение о скором появлении новой функции (синхронизация с `Active Directory` через `LDAP`) и список `API`-ключей, которые есть в базе данных соответственно. Также предусмотрено предупреждение об ошибке в случае сбоя в системе и передачи в функцию несуществующего сигнала.

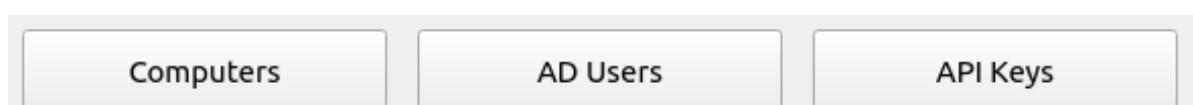


Рисунок 8 – Кнопки изменения содержимого экрана

```

computers_button = QPushButton("Computers")
computers_button.setFixedSize(180, 40)
computers_button.clicked.connect(lambda: self.clicked('computers'))

users_button = QPushButton("AD Users")
users_button.setFixedSize(180, 40)
users_button.clicked.connect(lambda: self.clicked('users'))

keys_button = QPushButton("API Keys")
keys_button.setFixedSize(180, 40)
keys_button.clicked.connect(lambda: self.clicked('keys'))

```

Рисунок 9 – Создание кнопок и передача сигнала

```

def clicked(self, text):
    item = self.grid.itemAtPosition(1, 1)
    item.widget().deleteLater()
    match text:
        case 'computers':
            self.CreateComputerListWidget()
            self.grid.addWidget(self.computersQListWidget, 1, 0, 1, 3)
        case 'users':
            users_msg = QLabel("Here will be list of Windows Active Directory Users soon!")
            users_msg.setAlignment(Qt.AlignmentFlag.AlignHCenter)
            self.grid.addWidget(users_msg, 1, 0, 1, 3)
        case 'keys':
            self.CreateAPIKeysListWidget()
            self.grid.addWidget(self.keysQListWidget, 1, 0, 1, 3)
        case _:
            error = QLabel("Something went wrong! Please try to click the button again!")
            error.setAlignment(Qt.AlignmentFlag.AlignHCenter)
            self.grid.addWidget(error, 1, 0, 1, 3)

```

Рисунок 10 – Обработка сигналов в функции

По умолчанию и при нажатии кнопки Computers на главный экран выводится список компьютеров, на которые было установлено клиентское приложение и которые, соответственно, были сохранены в базе данных. Запрос к базе отправляется каждый раз, когда необходимо создать список. Это позволяет отображать актуальную информацию сразу же после обновления базы.

Для получения информации с сервера была написана функция, отправляющая JSON-запрос на сервер (см. Рисунок 12). Для доступа к базе

потребуется ключ, ранее сохраненный в `keyring`. После получения ответа от сервера JSON-объект преобразуется в список словарей, которые, в свою очередь, передаются в качестве параметра конструктора датакласса `Computer` (см. Рисунок 11). Датакласс — декоратор для классов в языке Python, предоставляющий функции для автоматического добавления сгенерированных специальных методов, таких как `__init__()` и `__repr__()`, в определяемые пользователем классы. Декоратор позволяет указать только поля и тип находящихся в них данных при создании класса. Эта конструкция подходит для работы с классами, предназначенными для хранения данных.

```
4 usages  ▸ Anna Novozhilova
@dataclass
class Computer:
    name: str
    hardware_id: str
    cpu: str
    gpus: list
    disks: list
    ram: int
    used: int
    OS: str
```

Рисунок 11 — Датакласс `Computer`

```
def parse_all():
    message = 'http://46.151.30.76:5000/api/computers' + '?api_key=' + API_KEY
    all_comps_json = requests.get(message)
    list_of_dicts = all_comps_json.json()
    list_of_dicts = list_of_dicts['computers']
    list_of_comps = list()
    for el in list_of_dicts:
        tmp = Computer(**el)
        list_of_comps.append(tmp)
    return list_of_comps
```

Рисунок 12 — Создание списка компьютеров

Для отображения полученной информации был создан пользовательский виджет. Виджет позволяет выводить на экран информацию о сетевом имени устройства, статусе в сети, изображение, соответствующее типу устройства (на данный момент система поддерживает только пользовательские компьютеры, однако в будущем планируется добавить мониторинг маршрутизаторов и коммутаторов). Также в макет виджета включена кнопка, открывающая окно с более подробной информацией

о компьютере и передающая в него, как параметр, идентификатор материнской платы компьютера (см. Рисунок 13).

```
def clicked(self):
    sender = self.sender()
    self.window = ComputerInfoWindow(sender.objectName())
    self.window.show()
```

Рисунок 13 — Открытие окна с информацией о компьютере

Список устройств заполняется с помощью функций CreateComputerListWidget (см. Рисунок 14), которая очищает предыдущий список, заново получает информацию из базы данных и запускает процесс создания виджетов, и FillComputerListWidget (см. Рисунок 14), которая создает виджеты, заполняет их нужной информацией и добавляет их в список.

Аналогичным образом создается и заполняется список ключей, которые есть в базе данных. Отличием является только содержимое виджета: для ключа на экране отображается имя владельца, его роль в системе (рядовой пользователь, администратор, работающий с компьютера или мобильный администратор) и кнопка открытия окна с более подробной информацией.

```
def CreateComputerListWidget(self):
    self.computersQListWidget = QListWidget()
    self.list_of_comps = parse_all()
    self.timer = QTimer()
    self.timer.timeout.connect(self.fillComputerListWidget)
    self.timer.start(50)

def fillComputerListWidget(self):
    if self.index < len(self.list_of_comps):
        compLineWidget = QCustomQWidget()
        compLineWidget.setComputerName(str(self.list_of_comps[self.index].name))
        compLineWidget.setComputerStatus(self.list_of_comps[self.index].used)
        compLineWidget.setIcon()
        compLineWidget.setButtonName(self.list_of_comps[self.index].hardware_id)
        computersQListWidgetItem = QListWidgetItem(self.computersQListWidget)
        computersQListWidgetItem.setSizeHint(compLineWidget.sizeHint())
        self.computersQListWidget.addItem(computersQListWidgetItem)
        self.computersQListWidget.setItemWidget(computersQListWidgetItem, compLineWidget)
        self.index += 1

    if self.index >= len(self.list_of_comps):
        self.timer.stop()
        self.index = 0
```

Рисунок 14 — Создание списка компьютеров

В результате работы представленного выше кода на экране отображаются списки компьютеров и ключей при нажатии на соответствующие кнопки на экране (см. Рисунки 16-17). Кнопка AD Users на момент написания работы выводит на экран сообщение с сообщением о скором появлении новой функции (см. Рисунок 15).

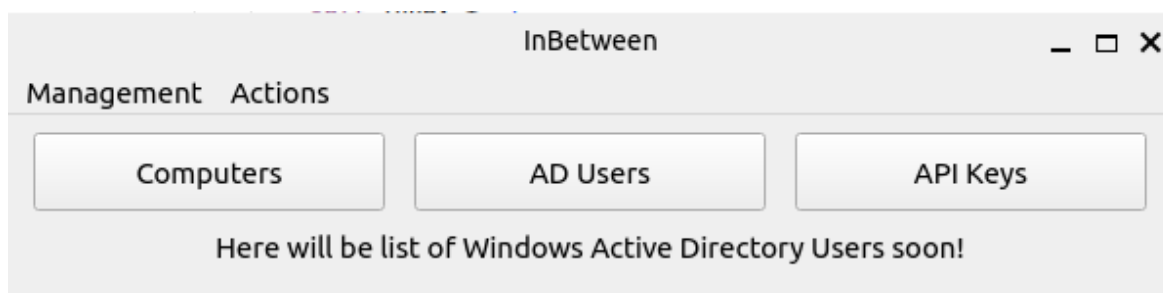


Рисунок 15 — Сообщение об отсутствии функции

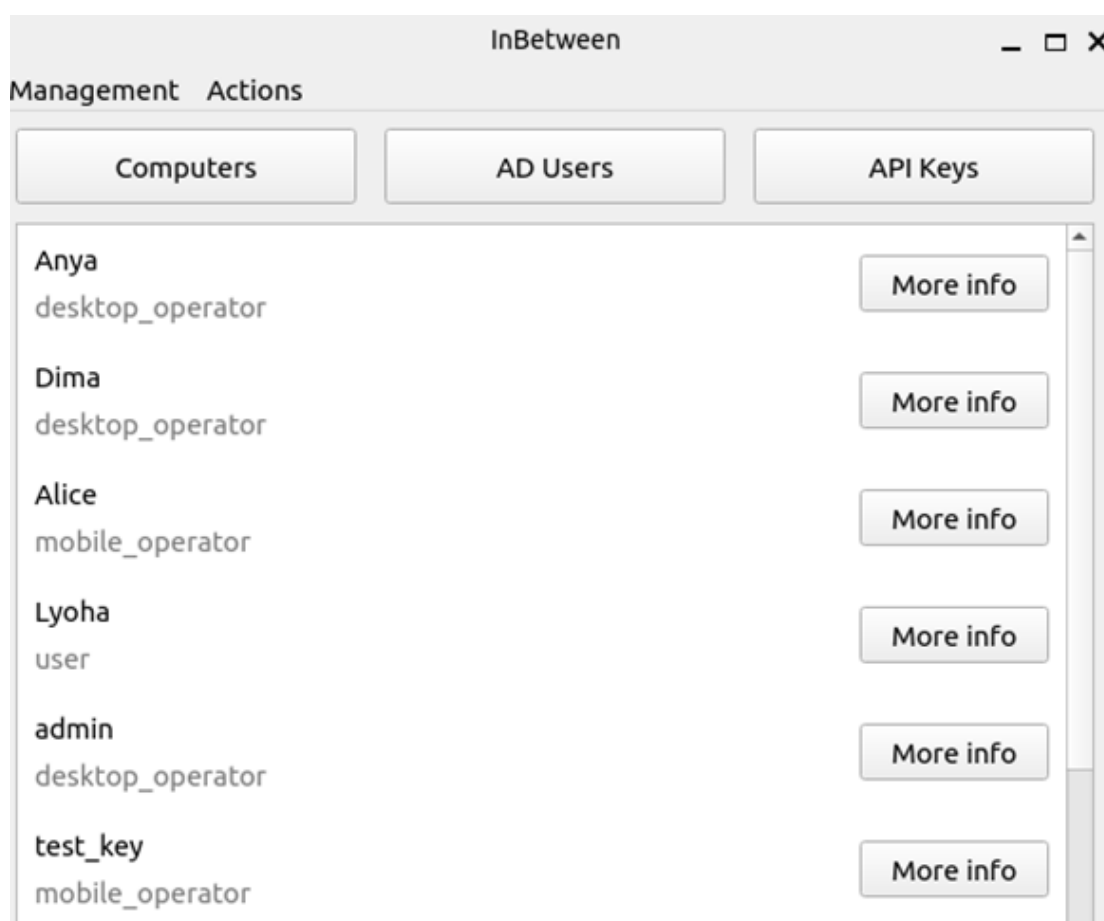


Рисунок 17 — Список API-ключей

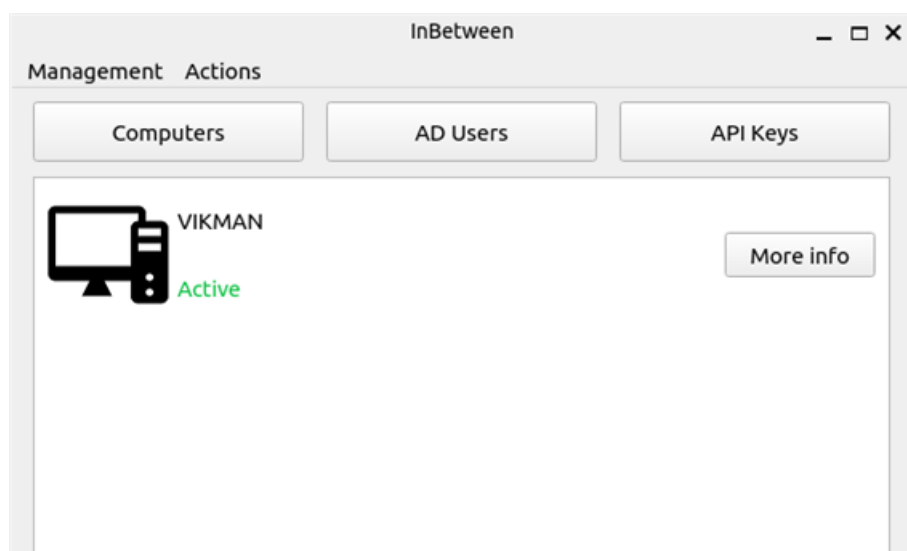


Рисунок 16 — Список устройств, подключенных к системе

Возможности системы мониторинга не предполагают создание новых устройств вручную, так как они автоматически добавляются в базу при установке на компьютер клиентского приложения «InBetween». Однако, через верхнее меню есть возможность добавить в базу данных новый ключ от базы данных. Этот функционал реализован с помощью диалогового окна (см. Рисунок 20) — специального класса, созданного для получения информации от пользователя. От пользователя требуется ввести имя, на которое будет записан ключ, на каком устройстве он будет использоваться, а также задать права безопасности. Права задаются с помощью переключателей, каждый из которых обозначает одну из возможностей: чтение, обновление данных, добавление новой записи, удаление записей для таблиц с компьютерами, API-ключами и логами. После нажатия кнопки ОК первым делом данные из переключателя формируются в строку определенного формата (см. Рисунок 18), которая, затем, вместе с остальными параметрами, передается JSON-объектом в POST-запрос к базе данных (см. Рисунок 19).

```

def form_descriptor(self):
    desc_string = ""
    if self.comp_read.isChecked():
        desc_string += "1"
    else:
        desc_string += "0"
    if self.comp_update.isChecked():
        desc_string += "1"
    else:
        desc_string += "0"
    if self.comp_add.isChecked():
        desc_string += "1"
    else:
        desc_string += "0"
    if self.comp_delete.isChecked():
        desc_string += "1"
    else:
        desc_string += "0"
    desc_string += ":"

    if self.client_read.isChecked():
        desc_string += "1"
    else:
        desc_string += "0"
    if self.client_update.isChecked():
        desc_string += "1"

```

Рисунок 18 – Формирование security_descriptor

```

def create_client(self):
    url = 'http://4          0/api/client?api_key='
    client_json_dict = {"id": 0,
                        "name": self.name,
                        "api_key": "",
                        "type": self.type,
                        "security_descriptor": self.security_descriptor}
    request = json.dumps(client_json_dict)
    page = requests.post(url, json=request, verify=False)
    page = page.json()
    client = APIKey(**page)
    return client

```

Рисунок 19 – Создание новой записи в базе данных

New API Key Creation

Enter the name of API Key owner

Select key user type

desktop_operator

Select the set of rights that the key grants:

Computers	API Keys	Logs
<input type="checkbox"/> Read	<input type="checkbox"/> Read	<input type="checkbox"/> Read
<input type="checkbox"/> Update	<input type="checkbox"/> Update	<input type="checkbox"/> Update
<input type="checkbox"/> Add	<input type="checkbox"/> Add	<input type="checkbox"/> Add
<input type="checkbox"/> Delete	<input type="checkbox"/> Delete	<input type="checkbox"/> Delete

Cancel OK

Рисунок 20 – Окно создания ключа

После того, как от сервера будет получен положительный ответ, только что сгенерированный ключ преобразуется в изображение, содержащее QR-код с помощью модуля `qrcode` (см. Рисунок 21). Это изображение выводится на экран в новом окне (см. Рисунок 22), чтобы пользователь мобильного приложения мог его отсканировать. После закрытия окна изображение удаляется в целях безопасности.

```
def ok_clicked(self):
    self.name = self.name_input_box.text()
    self.security_descriptor = self.form_descriptor()
    req = self.create_client()
    filename = "api_key_qr.png"
    img = qrcode.make(req.api_key)
    img.save(filename)
    self.accept()
```

Рисунок 21 – Формирование QR-кода



Рисунок 22 – Вывод изображения на экран

Для всех имеющихся в базе ключей на экран можно вывести в отдельном окне более подробную информацию (см. Рисунок 23), которая включает в себя имя пользователя ключа, его идентификационный номер в базе, тип, непосредственно сам ключ и настройки доступа, связанные с этим ключом. По умолчанию переключатели настроек безопасности заблокированы, чтобы избежать случайного нажатия.

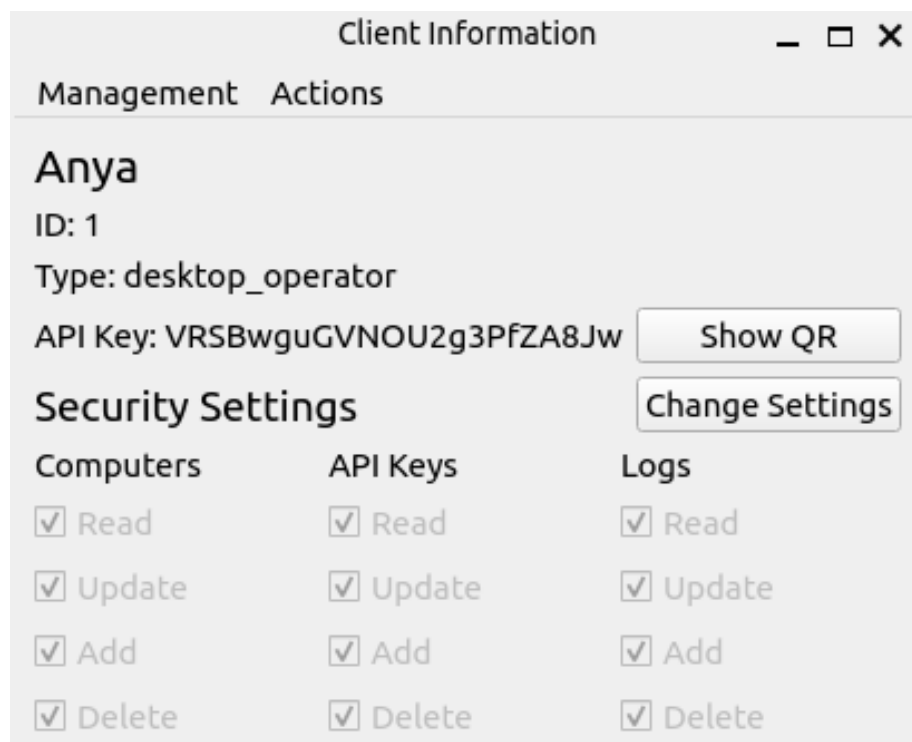


Рисунок 23 – Окно подробной информации о ключе

Разблокировать переключатели для изменения возможно с помощью нажатия кнопки «Change Settings». Во время нажатия кнопки обрабатывается функция (см. Рисунок 24), снимающая статус Blocked с каждого переключателя, сохраняя при этом их состояние, также кнопка «Change Settings» заменяется кнопкой «Confirm». После этого можно будет изменить состояния переключателей (см. Рисунок 25). Важно отметить, что со стороны графического интерфейса приложения не налагается никаких ограничений на права работающего в системе пользователя, так как если их будет недостаточно для выполнения необходимой операции, то сервер откажет пользователю в выполнении запроса после нажатия кнопки «Confirm». Во время нажатия на нее срабатывает функция (см. Рисунок 26), которая изменяет интерфейс приложения, заменяя одну кнопку другой, блокирует изменение переключателей и отправляет на сервер POST-запрос с измененной информацией о ключе.

```
def choose_settings(self):  
    self.confirm_button = QPushButton('Confirm')  
    self.confirm_button.clicked.connect(self.confirmed)  
    self.settings_button.deleteLater()  
    self.layout.addWidget(self.confirm_button, 6, 1)  
    self.layout.update()  
  
    self.comp_read.setDisabled(False)  
    self.comp_add.setDisabled(False)  
    self.comp_delete.setDisabled(False)  
    self.comp_update.setDisabled(False)
```

Рисунок 24 – Функция разблокировки переключателей

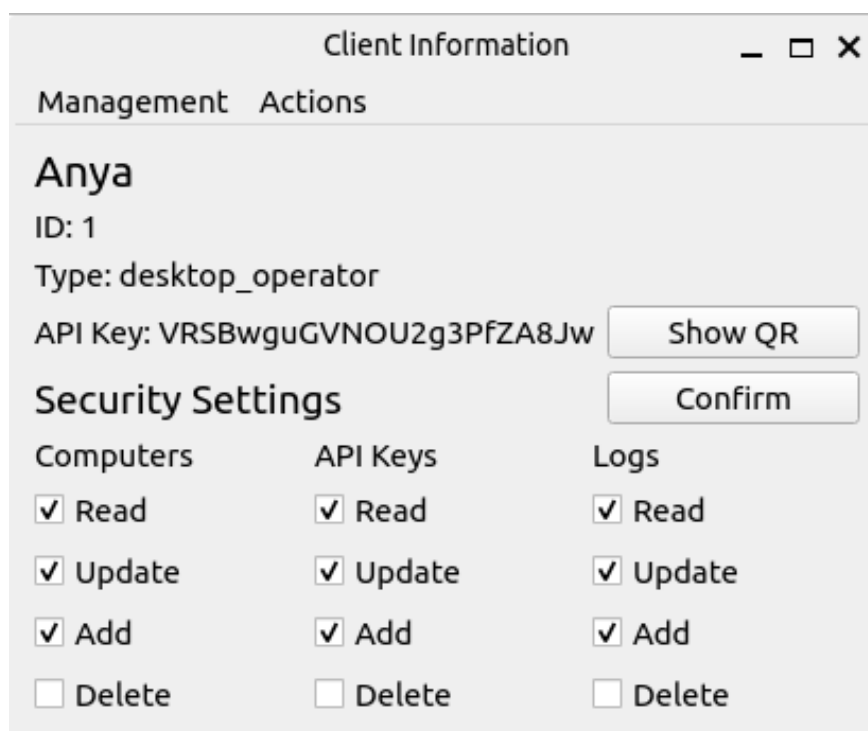


Рисунок 25 – Изменение настроек доступа

```
def confirmed(self):
    self.settings_button = QPushButton('Change Settings')
    self.settings_button.clicked.connect(self.choose_settings)
    self.confirm_button.deleteLater()
    self.layout.addWidget(self.settings_button, 6, 1)
    self.layout.update()
    self.update_info()

    self.comp_read.setDisabled(True)
    self.comp_add.setDisabled(True)
    self.comp_delete.setDisabled(True)
    self.comp_update.setDisabled(True)

def update_info(self):
    security_descriptor = self.form_descriptor()
    self.key_info.security_descriptor = security_descriptor
    url = 'http://46.151.30.76:5000/api/client?api_key=test_api_key'
    client_json_dict = {"id": self.key_info.id,
                        "name": self.key_info.name,
                        "api_key": self.key_info.api_key,
                        "type": self.key_info.type,
                        "security_descriptor": self.key_info.security_descriptor}
    request = json.dumps(client_json_dict)
    page = requests.post(url, json=request, verify=False)
    page = page.json()
```

Рисунок 26 – Функции изменения информации о ключе

Основной функционал приложения сосредоточен в окне с подробной информацией о компьютере (см. Рисунок 27). На нем отображается не только краткая сводка о характеристиках компьютера – имя в сети, идентификационный номер материнской платы, установленная на компьютере операционная система (если на компьютере несколько операционных систем, будет выведена только информация о той, на которую установлено клиентское приложение), процессор, количество оперативной памяти в мегабайтах, все размеченные в операционной системе тома (их буквенное обозначение и количество свободного места), видеокарты – но также и полученные за последнее время логи, линейные графики для показателей оперативной памяти и загрузки процессора, история сообщений чата.

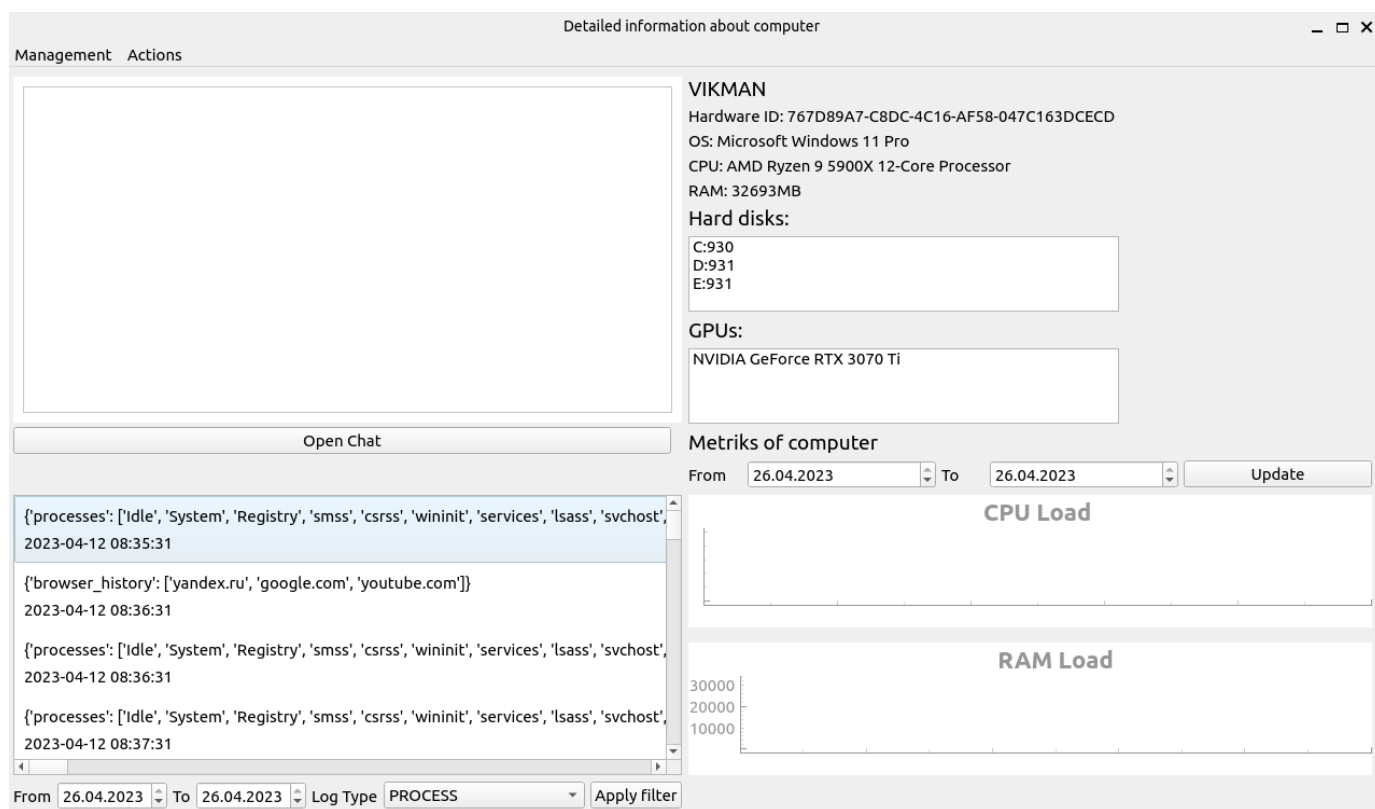


Рисунок 27 – Окно с подробной информацией о компьютере

Далее подробно будет рассмотрена работа каждой из обозначенных функций.

Логи выводятся прокручиваемым списком в нижнем левом углу окна (см. Рисунок 28). Для датакласса Log (см. Рисунок 29) определены такие поля,

как идентификатор в базе, идентификатор компьютера, к которому он относится, тип лога, время его получения и полученные данные. Также для него определены две функции: функция `parse_time` преобразует дату из формата `timestamp` в формат `datetime`, а функция `get_load` добавляет новые поля в класс – нагрузку на процессор и нагрузку на оперативную память. Эта функция имеет смысл и запускается только в том случае, если лог относится к типу `LOAD` и содержит информацию о нагрузке в указанный момент времени. Этот тип логов по умолчанию скрыт из списка на экране, и его можно вывести только намеренно указав его в фильтре.

Каждый элемент списка логов представляет собой пользовательский виджет с текстом лога и временем, когда он был получен клиентским приложением. Список заполняется аналогично спискам компьютеров и ключей – в функции `CreateLogListWidget` (см. Рисунок 30) список пересоздается, также актуализируется информация из базы, затем каждые 50 миллисекунд в цикле создается новый экземпляр пользовательского виджета. В функции `FillLogListWidget` (см. Рисунок 30) он заполняется информацией из полученных на предыдущем шаге объектов типа `Log` и добавляется в список.

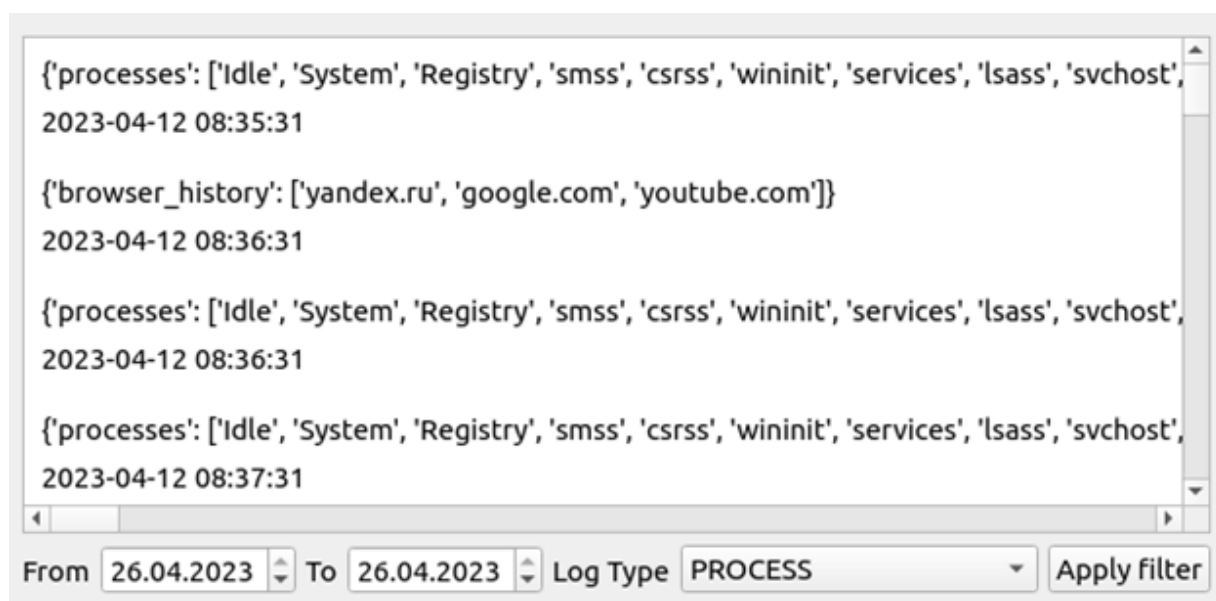


Рисунок 28 – Список логов


```

@dataclass
class Log:
    id: int
    hardware_id: str
    type: int
    timestamp: int
    data: dict

    Anna Novozhilova
def parse_time(self):
    self.datetime = datetime.datetime.fromtimestamp(self.timestamp)

    Anna Novozhilova
def get_load(self):
    self.cpu = str(self.data['cpu'])
    self.cpu = float(self.cpu.replace('%', ''))
    self.ram = str(self.data['ram'])
    self.ram = int(self.ram.replace('MB', ''))
    print('ram ', self.ram, 'cpu ', self.cpu)

```

Рисунок 29 – Класс Log

```

def CreateLogListWidget(self):
    self.logListWidget = QListWidget()
    self.timer = QTimer()
    self.timer.timeout.connect(self.fillLogListWidget)
    self.timer.start(50)

    Anna Novozhilova
def fillLogListWidget(self):
    if self.index < len(self.logs):
        if self.logs[self.index].type == 1:
            self.index += 1
        else:
            LogLineWidget = LogCustomQWidget()
            LogLineWidget.setText(str(self.logs[self.index].data))
            LogLineWidget.setTime(str(self.logs[self.index].datetime))
            #LogLineWidget.setIcon(self.logs[self.index].type)
            LogLineWidget.setId(self.logs[self.index].id)
            LogLineWidget.setHardwareId(self.logs[self.index].hardware_id)
            logListWidgetItem = QListWidgetItem(self.logListWidget)
            logListWidgetItem.setSizeHint(LogLineWidget.sizeHint())
            self.logListWidget.addItem(logListWidgetItem)
            self.logListWidget.setItemWidget(logListWidgetItem, LogLineWidget)
            self.index += 1
    if self.index >= len(self.logs):
        self.timer.stop()
        self.index = 0

```

Рисунок 30 – Функции для заполнения списка логов

При двойном щелчке на элемент списка срабатывает функция (см. Рисунок 31), которая передает информацию о выбранном логе в новое окно (см. Рисунок 32). Это позволяет удобно читать длинные логи, которые не помещаются на панель списка.

```
def mouseDoubleClickEvent(self, event):  
    print(self.id)  
    self.window = LogInfo(self.id, self.hardware_id)  
    self.window.show()
```

Рисунок 31 – Функция открытия нового окна



Рисунок 32 – Окно с полным текстом лога

Список логов можно отфильтровать по двум параметрам – тип лога и время его получения (см. Рисунок 33). Всего в системе определено пять типов логов – список запущенных системой процессов, нагрузка на систему, информация о подключении нового USB-устройства, информация об изменении конфигурации, список открытых в браузере вкладок. По умолчанию фильтр настроен на выдачу всех результатов за сегодняшний день. Также для полей выбора даты есть три ограничения: дата не может быть позже текущей, не может быть раньше 1 января 1970 года и дата начала не может быть позже даты окончания.

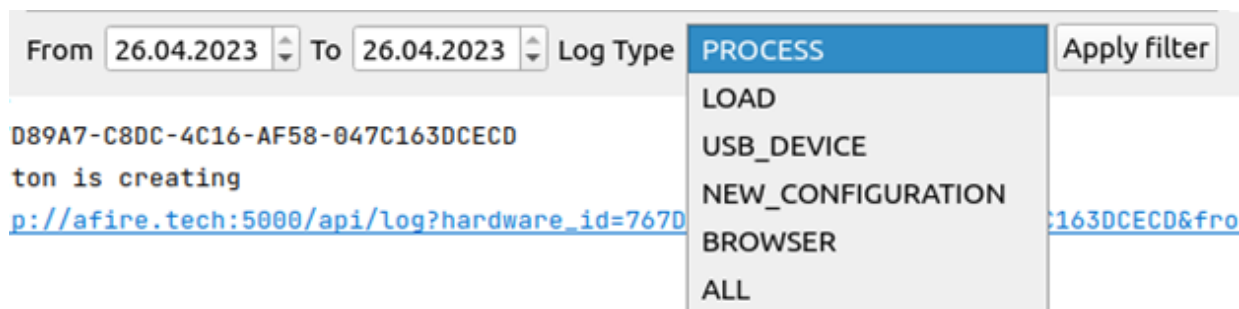


Рисунок 33 – Фильтрация логов

После нажатия кнопки «Apply Filter» значения из полей передаются в функцию `filter` (см. Рисунок 34), которая инициирует проверку заданных условий и, если все корректно, то вызывает еще одну функцию – `parse_log_with_filter`, которая заново заполняет список логов, но уже с учетом выставленных фильтров (см. Рисунок 35). Если же значения выставлены некорректно, то функция `filter` вызывает окно об ошибке.

```
def filter(self):
    to_time = int(datetime.timestamp(self.to_time_widget.dateTime().toPyDateTime()))
    from_time = int(datetime.timestamp(self.from_time_widget.dateTime().toPyDateTime()))
    if (to_time >= from_time):
        self.logs = parse_logs_with_filters(self.comp_info.hardware_id, from_time, to_time, self.log_type)
        self.logListWidget.clear()
        self.CreateLogListWidget()
        self.grid.addWidget(self.logListWidget, 6, 0, 7, 2)
        self.grid.update()
    else:
        msg = QMessageBox()
        msg.setWindowTitle("Filter error")
        msg.setText("Date setting is invalid")
        #msg.setIcon(QMessageBox.Warning)
        msg.exec()
```

Рисунок 34 – Функция filter

```
def parse_logs_with_filters(hardware_id, from_timestamp, to_timestamp, log_type):
    api_url = ''
    if(log_type == 5):
        api_url = 'http://afire.tech:5000/api/log?hardware_id=' + hardware_id + '&from=' + str(
            from_timestamp) + '&to=' + str(to_timestamp) + '&api_key=' + API_KEY
    else:
        if(log_type == 4):
            log_type+=1
        api_url = 'http://afire.tech:5000/api/log?hardware_id=' + hardware_id + '&from=' + str(
            from_timestamp) + '&to=' + str(to_timestamp) + '&type=' + str(log_type) + '&api_key='
    print(api_url)
    log_list_json = requests.get(api_url)
    log_list_dict = log_list_json.json()
    log_list_dict = log_list_dict['logs']
    log_list = list()
    for el in log_list_dict:
        tmp = Log(**el)
        tmp.parse_time()
        log_list.append(tmp)
    return log_list
```

Рисунок 35 – Функция parse_logs_with_filters

Еще одним важным элементом окна с информацией о компьютере являются графики. Графики отражают изменения нагрузки на систему (процессор и оперативная память) во времени, и значительно упрощают процесс мониторинга состояния компьютера.

Графики в приложении создаются в отдельном классе Chart, включающем в себя следующие поля – начальная временная точка, конечная временная точка, идентификатор устройства, максимальный объем оперативной памяти (для корректного отображения графика). Если при создании графика пользователь не указывает определенный временной промежуток, то по умолчанию графики составляются от полуночи сегодняшнего дня, до текущего времени (см. Рисунок 36).

```
def __init__(self, hardware_id, ram, start=0, finish=0):
    if start == 0:
        dt = datetime.combine(date.today(), datetime.min.time())
        self.start = int(datetime.timestamp(dt))
    else:
        self.start = start
    if finish == 0:
        self.finish = int(datetime.timestamp(datetime.now()))
    else:
        self.finish = finish
    self.hardware_id = hardware_id
    self.max_ram = ram
```

Рисунок 36 – Конструктор класса Chart

Нагрузка на систему, на основе которой строятся графики, хранится на сервере вместе с другими логами, поэтому, чтобы получить необходимые данные функция `get_data` (см. Рисунок 37) обращается к API сервера, запрашивая логи только первого типа за указанный промежуток времени. Затем функция разделяет полученные данные на три списка – временные отметки, показатели оперативной памяти и проценты загрузки процессора.

```

def get_gata(self):
    global API_KEY
    API_KEY = get_api_key()
    api_url = 'http://afire.tech:5000/api/log?hardware_id=' + self.hardware_id
    self.start) + '&to=' + str(self.finish) + '&type=1&api_key=' + API_KEY
    log_list_json = requests.get(api_url)
    log_list_dict = log_list_json.json()
    log_list_dict = log_list_dict['logs']
    self.cpus = list()
    self.rams = list()
    self.times = list()
    for el in log_list_dict:
        tmp = Log(**el)
        tmp.parse_time()
        tmp.get_load()
        self.cpus.append(tmp.cpu)
        self.rams.append(tmp.ram)
        self.times.append(tmp.datetime)

```

Рисунок 37 – Функция получения данных

Графики рисуются с помощью библиотеки rugarph, являющейся частью фреймворка PyQt. В конструктор (см. Рисунок 38) передаются полученные данные, настраивается внешний вид графиков (размер сетки, цвет линий и цвет фона), устанавливается название. Графики наследуют поведение класса QWidget и могут быть вставлены в сетку графического интерфейса приложения (см. Рисунок 39).

<pre> def draw_cpu_chart(self): self.date_axis = TimeAxisItem(orientation='bottom') self.cpu_plot_widget = pg.PlotWidget(axisItems={'bottom': self.date_axis}, title='<h2>CPU Load</h2>') pen = pg.mkPen(color=(255, 0, 0), width=2) self.cpu_plot_widget.plot(x=[x.timestamp() for x in self.times], y=self.cpus, pen=pen) self.cpu_plot_widget.setBackground('w') self.cpu_plot_widget.setYRange(0, 1) return self.cpu_plot_widget </pre>	<pre> def draw_ram_chart(self): self.date_axis = TimeAxisItem(orientation='bottom') self.ram_plot_widget = pg.PlotWidget(axisItems={'bottom': self.date_axis}, title='<h2>RAM Load</h2>') pen = pg.mkPen(color=(255, 0, 0), width=2) self.ram_plot_widget.plot(x=[x.timestamp() for x in self.times], y=self.rams, pen=pen) self.ram_plot_widget.setBackground('w') self.ram_plot_widget.setYRange(0, int(self.max_ram)) return self.ram_plot_widget </pre>
---	---

Рисунок 38 – Конструктор графиков

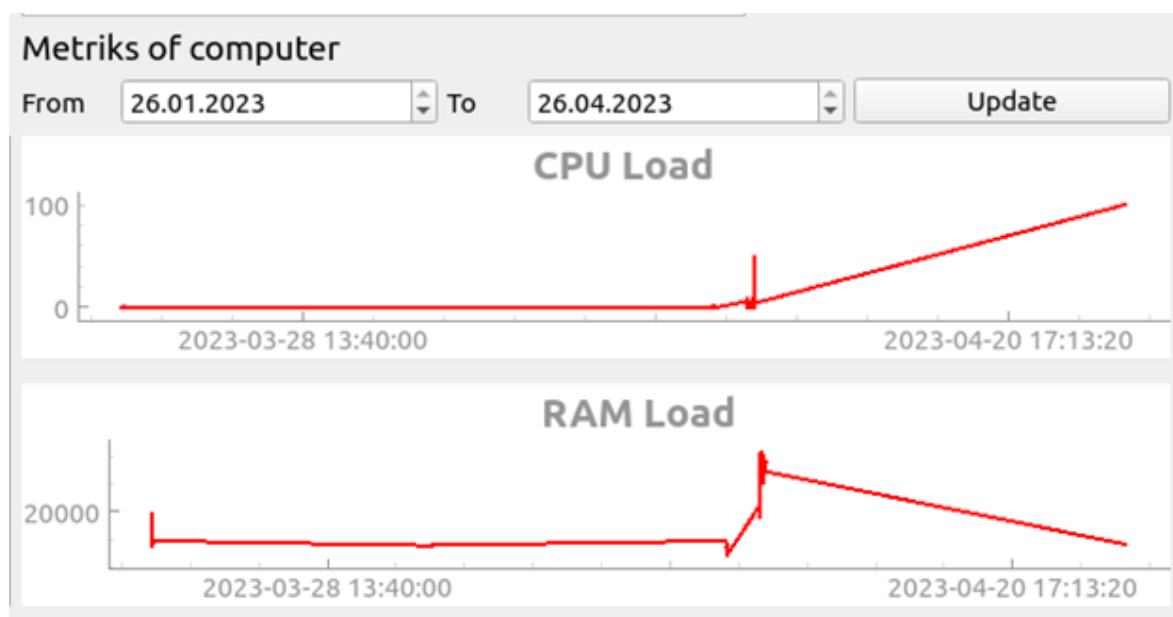


Рисунок 39 – Отображение графиков на экране

По нажатию на кнопку «Update» графики перестраиваются с учетом указанных промежутков времени (см. Рисунок 40). Ограничения на временные рамки точно такие же, как для фильтра логов. Если ограничения не соблюдены, то отображается сообщение об ошибке, а графики на экране остаются неизменными.

```
def update_charts(self):
    to_time = int(datetime.timestamp(self.chart_to_time_widget.dateTime().toPyDateTime()))
    from_time = int(datetime.timestamp(self.chart_from_time_widget.dateTime().toPyDateTime()))
    if (to_time >= from_time):
        self.cpu_chart.close()
        self.ram_chart.close()
        self.charts = Chart(self.comp_info.hardware_id, self.comp_info.ram, from_time, to_time)
        self.charts.get_gata()
        self.cpu_chart = self.charts.draw_cpu_chart()
        self.ram_chart = self.charts.draw_ram_chart()
        self.grid.addWidget(self.cpu_chart, 11, 3, 4, 2)
        self.grid.addWidget(self.ram_chart, 16, 3, 4, 2)
        self.grid.update()
    else:
        msg = QMessageBox()
        msg.setWindowTitle("Filter error")
        msg.setText("Date setting is invalid")
        #msg.setIcon(QMessageBox.warning)
        msg.exec()
```

Рисунок 40 – Обновление графиков

ЗАКЛЮЧЕНИЕ

Системы мониторинга сетей связи являются неотъемлемой частью сетевой инфраструктуры любого предприятия. Даже небольшие организации при работе сетями прибегают к использованию программ для мониторинга. Это позволяет не только сократить время, необходимое на обнаружение неполадок или сетевых угроз, но также дает возможность предотвратить сбой в работе инфраструктуры предприятия.

За последние десятилетия было разработано множество систем мониторинга. В ходе выполнения работы был произведен сравнительный анализ систем мониторинга, наиболее популярных в русскоязычном сегменте сети Интернет. Этот анализ показал, что, несмотря на наличие разнообразных бесплатных инструментов, большинство из них предполагает сложную настройку и избыточный для небольших, не испытывающих экстремальной нагрузки сетей функционал. В связи с этим было принято решение разработать простую в использовании систему мониторинга, не требующую настройки перед началом работы.

В ходе выполнения практической части исследования была спроектирована логическая модель работы системы мониторинга сетей связи «InBetween». Разработанная структура взаимодействия всех элементов системы и диаграмма классов полностью соответствуют итоговому продукту.

Приложение для системного администратора было написано на языке Python с использованием библиотеки для создания графических интерфейсов PyQt6. Оно может быть запущено с операционных систем Windows или Linux. Приложение позволяет отслеживать состояние пользовательских компьютеров в локальной сети, управлять доступ к информации в системе мониторинга и взаимодействовать с пользователями системы через чат. Управление функциями приложения происходит через минимальный графический интерфейс.

Интерес к системе мониторинга «InBetween» проявили несколько ИП и небольших компаний, являющихся целевой аудиторией проекта,

что подтверждает его практическую значимость. На момент написания отчёта система проходит тестирование в локальной сети МБОУ СОШ №2 поселка Марково.

Таким образом, все задачи этого исследования выполнены, а цели достигнуты.

В последующих исследованиях планируется доработать функциональные возможности системы мониторинга «InBetween», включающие в себя мониторинг маршрутизаторов и коммутаторов, визуализацию большего количества показателей нагрузки системы, отслеживание сетевых атак и нежелательных приложений, возможность наблюдения за сессией клиента, интеграция с Active Directory. Также планируется усовершенствовать графический интерфейс системы, сделав его соответствующим современным трендам в дизайне.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Network Monitoring Market by Offering (Equipment, Software and Services), Bandwidth, Technology, End User and Geography – Global Forecast to 2027. URL: <https://www.marketsandmarkets.com/Market-Reports/network-monitoring-market-51888593.html> (дата обращения: 25.03.2023)
2. Pang-Wei Tsai, Chun-Wei Tsai, Chia-Wei Hsu, Chu-Sing Yang. Network Monitoring in Software-Defined Networking //A Review. IEEE - 2018. – p.3958 - 3969
3. Шардаков К.С. Сравнительный анализ популярных систем мониторинга сетевого оборудования, распространяемых по лицензии GPL // Intellectual Technologies on transport, 2018. – с.44-47
4. Г. Буч, Д. Рамбо, И. Якобсон. Язык UML. Руководство. // ДМК Пресс, 2006. — 496 с.
5. Алан Купер. Об интерфейсе. Основы проектирования // Символ-Плюс, 2009. — 688 с.
6. Python 3.10 Documentation. URL: <https://docs.python.org/3/index.html> (дата обращения 01.05.2023)
7. Майер Б. Объектно-ориентированное конструирование программных систем. // Русская редакция, 2019. – с.44-56
8. Comer, Douglas E.; Stevens, David L. Client-Server Programming and Applications. Internetworking with TCP/IP. // Prentice Hall, 1993. – p.13-21
9. В. Олифер, Н. Олифер. Компьютерные сети. Принципы, технологии, протоколы. // Питер, 2020. – 51 с.