Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-ВОСТОЧНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ ИМ. М.К. АММОСОВА»
Институт математики и информатики
Кафедра «Информационные технологии»

ДОПУСТИТЬ К ЗАЩИТЕ:
Зав. кафедрой _____/Н.В. Николаева/
Протокол №____ от «____» _____ 2023г.

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ РАСЧЕТА КОМПЛЕКСНОГО ПОКАЗАТЕЛЯ ПОЖАРООПАСНОСТИ

БАКАЛАВРСКАЯ РАБОТА

Направление подготовки: 09.03.01 Информатика и вычислительная техника

Направленность (профиль): Технологии разработки программного обеспечения

Выполнил: студент 4 курса
группы БА-ИВТ-19-1ИМИ СВФУ
Федоров Дьулуур Андрианович
_____
подпись

Научный руководитель:
старший преподаватель
кафедры ИТ ИМИ СВФУ,
Петрова Е.А.
_____
подпись

Якутск 2023

Оглавление

ВВЕДЕНИЕ

Тема выпускной квалификационной работы посвящена созданию приложения для расчета комплексного показателя пожароопасности. Проект был предложен сотрудником лаборатории криогенных ландшафтов Института мерзлотоведения СО РАН – Петровой Александрой Николаевной. Необходимо было разработать приложение, которое сможет рассчитать показатель пожароопасности на основе данных, предоставленных Всероссийским научно-исследовательским институтом гидрометеорологической информации – мировым центром данных.

Актуальность данной темы обусловлена тем, что на текущий момент пожары являются одной из наиболее опасных и разрушительных стихийных бедствий, которые грозят жизни и имуществу людей. Они оказывают значительное влияние на экономику и окружающую среду. Дополнительно, данные о погоде, климате и других факторах, влияющих на пожароопасность, представляют собой большие объемы информации. Их обработка и анализ требуют специализированных инструментов и подходов.

Объектом исследования выступает пожари и их зависомость от метеоусловий, а предметом исследования является показатель пожароопасности.

Приложение, разработанное в рамках выпускной квалификационной работы, позволит исследователям и климатологам эффективно обрабатывать, анализировать и визуализировать сложные связи, и зависимости в климатических данных. Программная разработка предоставляет мощные инструменты для импорта, хранения, управления и визуализации данных, а также для выполнения сложных запросов и аналитики, необходимых для изучения климатических процессов. Приложение будет актуально среди экологов, биологов, гидрологов, а также студентов, использующих данные для исследования причин возникновения пожаров.

Цель бакалаврской работы: разработка приложения для расчета комплексного показателя пожароопасности.

Задачи работы:

1. изучить причины возникновение пожаров и степень влияния природных факторов на их возникновение;
2. спроектировать структуру и разработать приложение.

Бакалаврская работа состоит из введения, трех глав, выводов по каждой главе, заключения, списка использованной литературы и источников. В конце работы приведен программный код созданного приложения.

В первой главе проанализирована предметная область: пожары и причины их возникновения. Также был проведен обзор аналогов и обоснован выбор программных средств и инструментов разработки.

Во второй главе рассмотрена структура программного решения, приведено описание разработки пользовательского интерфейса и серверной части приложения.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Пожары и причины их возникновения

Лесной пожар — явление многофакторное. На возгорание массы растений влияют не только метеорологические, но и экологические факторы: видовой состав лесной экосистемы, возраст и санитарное состояние древостоя и другие. Лесные пожары, как правило, начинаются почти сразу после схода снежного покрова, если устанавливается сухая погода. Наличие сухой прошлогодней травы, подсохших эпигейных лишайников и мхов способствует возникновению пожара. В последующие месяцы влажность в этом ярусе значительно увеличивается, в частности, из-за новообразованной биомассы трав. Высокая летняя температура усиливает горимость леса, и возгорание сдерживается только из-за выпадения атмосферных осадков.

Пожары являются одной из наиболее серьезных и разрушительных природных и техногенных катастроф, которые причиняют значительный ущерб окружающей среде, жизням людей и экономике. Понимание причин возникновения пожаров является ключевым аспектом их предотвращения и эффективного управления.

Пожары возникают из-за различных причин, и их идентификация является важным аспектом для предотвращения и борьбы с пожарами. Вот некоторые из наиболее распространенных причин возникновения пожаров:

1. Человеческий фактор. Один из главных факторов, приводящих к пожарам, это небрежность или неосторожность людей. Это может включать неправильное обращение с огнем и источниками тепла, неправильное использование электрооборудования, курение в запрещенных местах или выбрасывание горящих предметов без должной осторожности.
2. Проводка и электрические неисправности. Неправильная установка, эксплуатация или обслуживание электрооборудования может привести

к короткому замыканию или перегреву, что может вызвать пожар. Пожары, связанные с электричеством, могут возникать из-за поврежденной проводки, неисправных электроприборов или перегрузки электрических сетей.

3. Природные причины. Природные явления, такие как молния, сильные ветры или высокая температура, могут способствовать возникновению лесных пожаров. Также пожары могут возникать из-за сухой растительности, которая может легко воспламеняться и распространяться.
4. Халатность при работе с огнем. Неправильное обращение с открытым огнем, таким как газовые горелки, костры или свечи, может привести к несчастным случаям и пожарам. Отсутствие контроля и недостаток осторожности могут привести к быстрому распространению огня.
5. Взрывоопасные материалы и химические вещества: Неправильное хранение, использование или транспортировка взрывоопасных материалов и химических веществ может привести к возникновению пожаров и взрывов. Химические реакции, несовместимость веществ или неправильное смешивание могут создавать опасные ситуации.

1.2 Постановка задачи

Постановка задачи для разработки приложения по расчету показателя пожароопасности с использованием СУБД Neo4j, языка программирования Python и библиотеки графического интерфейса PyQt5:

1. Общие требования:

- Разработать приложение для операционной системы Linux;
- Использовать язык программирования Python версии 3;
- Использовать библиотеку графического интерфейса PyQt5 для создания пользовательского интерфейса;
- Использовать СУБД Neo4j для хранения и обработки данных.

2. Загрузка исходных данных:

- Предусмотреть возможность загрузки исходных данных о погоде, климатических показателях и других факторах, необходимых для расчета показателя пожароопасности;
- Данные могут быть представлены в текстовом формате (например, в формате TXT);
- Разработать механизм загрузки данных из выбранного пользователем файла в программу.

3. Выгрузка результатов:

- Реализовать функциональность выгрузки результатов расчетов в файл или базу данных;
- Предусмотреть возможность выбора формата выгрузки данных (например, JSON).

4. Визуализация результатов:

- Разработать графический интерфейс, позволяющий визуализировать результаты расчетов показателя пожароопасности;
- Предусмотреть отображение данных в виде графиков, диаграмм и таблиц;
- Обеспечить интерактивность интерфейса, позволяющую пользователю взаимодействовать с визуализированными данными (например, выбирать периоды времени, масштабировать графики и т.д.).

5. Вычисление по индексу Нестерова:

- Реализовать алгоритм расчета показателя пожароопасности на основе модифицированного индекса Нестерова;
- Использовать соответствующую формулу и учитывать необходимые метеорологические и климатические параметры;

- Предусмотреть возможность задания пользователем параметров и настроек для расчетов.

6. Прогнозирование вероятности пожара:

- Вывести результаты расчетов в графическом интерфейсе с указанием вероятности пожара в регионе;
- Обеспечить понятную интерпретацию результатов для пользователя.

7. Тестирование и отладка:

- Провести тестирование приложения для проверки корректности расчетов и работоспособности интерфейса;
- Отладить возможные ошибки и неполадки, обеспечивая стабильную работу приложения.

8. Документация и установка:

- Подготовить документацию по использованию приложения, включая инструкцию по установке и настройке;
- Обеспечить удобный и понятный интерфейс пользователя, минимизируя необходимость в дополнительных пояснениях.

1.3 Обзор аналогов

В качестве аналогов, выявлено два основных проекта:

- SmartUnit(Сильван)
- FireWeatherIndex (FWI) System

SmartUnit(Сильван) (рис.1) – Система противодействия лесным пожарам «Сильван» — это отечественный программный комплекс, разработанный в 2022 году нашей компанией. Программа объединяет в себе все необходимые инструменты для борьбы с лесными пожарами любой

сложности. «Сильван» разрабатывался при непосредственном участии экспертов в тушении лесных пожаров [9].

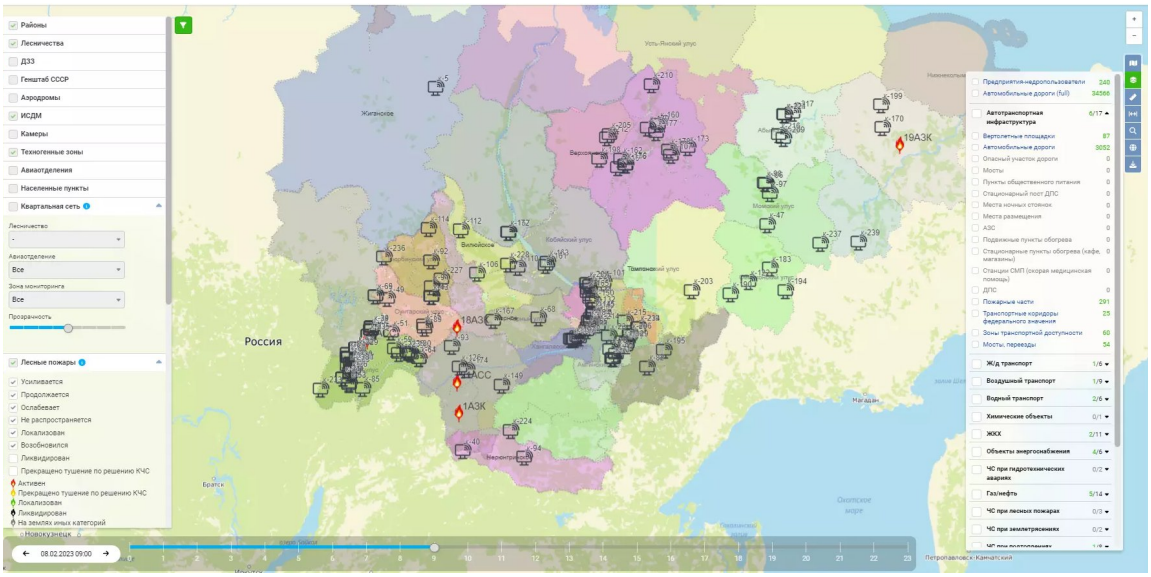

Рисунок 1. Приложение SmartUnit(Сильван)

Модуль позволяет вести реестр пожаров и карточки пожаров в режиме реального времени.

При обнаружении термической аномалии (термоточки), полученной из системы спутникового мониторинга, противопожарная служба уточняет природу возникновения термоточки путем авиамониторинга или иным доступным способом.

Отслеживание динамики пожара, сил и средств.

Если подтверждается пожар в реестр добавляется карточка пожара, содержащая подробную информацию.

В настоящий момент система “Сильван” успешно внедрена в самом большом регионе планеты – Республике Саха (Якутия) – 3 084 000 км².

FireWeatherIndex (FWI) System (рис.2) – это комплексная система для оценки пожароопасности в лесных районах, разработанная Канадским лесным службой. Она основана на анализе метеорологических данных и использует несколько индексов для оценки различных аспектов пожароопасности. Система использует трехкомпонентный индекс, который

включает индекс опасности пожара (Fire Danger Index, FDI), индекс распространения пожара (Fire Spread Index, FSI) и индекс поведения пожара (Fire Behavior Index, FBI). Каждый из этих индексов учитывает различные аспекты пожароопасности, такие как погодные условия, влажность топлива, скорость распространения огня и его поведение.

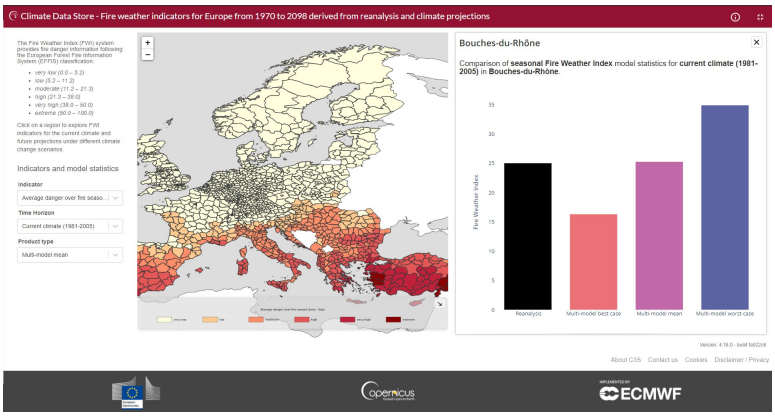

Рисунок 2. Приложение FireWare System Index

Для расчета индексов система использует метеорологические данные, такие как температура воздуха, относительная влажность, скорость ветра и осадки. Эти данные собираются с помощью автоматических метеостанций, расположенных в лесных районах. FWI System прошла множество исследований и тестов, чтобы обеспечить точность и надежность результатов. Она калибруется на основе реальных пожарных событий и периодически обновляется с использованием новых научных исследований и технологий.

Результаты расчетов индексов представляются в виде графиков, карт и отчетов, что обеспечивает информационную поддержку для специалистов по лесному хозяйству и пожарной безопасности. Это помогает принимать взвешенные решения в отношении контроля и тушения пожаров. FWI System широко используется в лесном хозяйстве и пожарной безопасности, а также в научных исследованиях. Она помогает предсказывать и оценивать пожароопасность, управлять ресурсами и разрабатывать стратегии предотвращения и тушения пожаров [8].

1.4 Выбор инструментов разработки

Для разработки приложения для комплексного расчета показателя пожароопасности были использованы следующие инструменты.

Python – является мощным языком программирования с богатым набором библиотек и фреймворков. Он предоставляет удобный и гибкий способ разработки приложений. Python отлично подойдет для обработки данных, разработки логики приложения и взаимодействия с базой данных Neo4j.

PyQt5 – библиотекой для разработки графического интерфейса пользователя (GUI) на основе фреймворка Qt. Она предоставляет широкие возможности для создания интерактивных и интуитивно понятных пользовательских интерфейсов. Вы можете использовать PyQt5 для разработки графической оболочки вашего приложения, включая окна, кнопки, таблицы, графики и другие элементы интерфейса.

Neo4j Python Driver: Neo4j Python Driver – это официальный драйвер для взаимодействия с базой данных Neo4j из приложений Python. Он предоставляет удобные методы для выполнения запросов и операций с данными в Neo4j. Вы можете использовать Neo4j Python Driver для подключения к базе данных Neo4j, выполнения запросов и получения результатов для отображения в вашем приложении.

Сочетание Python, PyQt5 и Neo4j Python Driver позволит вам создать функциональное приложение для проведения климатических исследований, с визуально привлекательным пользовательским интерфейсом и эффективным взаимодействием с базой данных Neo4j.

Исследования в области экологии и климатологии играют важную роль в понимании взаимосвязей между различными факторами, включая метеорологические данные, а также воздействие человеческой деятельности на окружающую среду. При анализе таких комплексных связей, где закономерности между данными не всегда очевидны, традиционные

реляционные системы управления базами данных (СУБД) могут стать неэффективными.

В этом контексте графовые СУБД становятся оптимальным выбором для проведения исследований связей между различными данными. Одной из таких систем является Neo4j – графовая СУБД с открытым исходным кодом, разработанная на языке программирования Java. Neo4j считается одной из самых популярных графовых СУБД на сегодняшний день.

Архитектура Neo4j обеспечивает хранение данных в специализированном формате, оптимизированном для представления графовой информации. Этот подход позволяет использовать дополнительные оптимизации для более сложных структур данных. Более того, Neo4j предоставляет специальные оптимизации для работы с SSD-накопителями и обработки больших графов без необходимости полного помещения данных в оперативную память.

В графовой модели данных, используемой Neo4j, основными компонентами являются узлы и ребра. Каждый узел и ребро могут быть дополнены своим собственным набором полей, что позволяет более гибко представлять информацию. Графовая структура базы данных позволяет легко выражать и анализировать сложные связи и отношения между сущностями, что важно для исследований в области климатологии.

В заключение, использование графовой СУБД Neo4j в приложении для климатических исследований позволяет ученым эффективно работать с данными, исследовать их взаимосвязи и визуализировать результаты. Это предоставляет новые возможности для более глубокого понимания климатических процессов и принятия обоснованных решений в области экологии и устойчивого развития.

Модель БД Neo4j схематично изображена на рисунке 3.

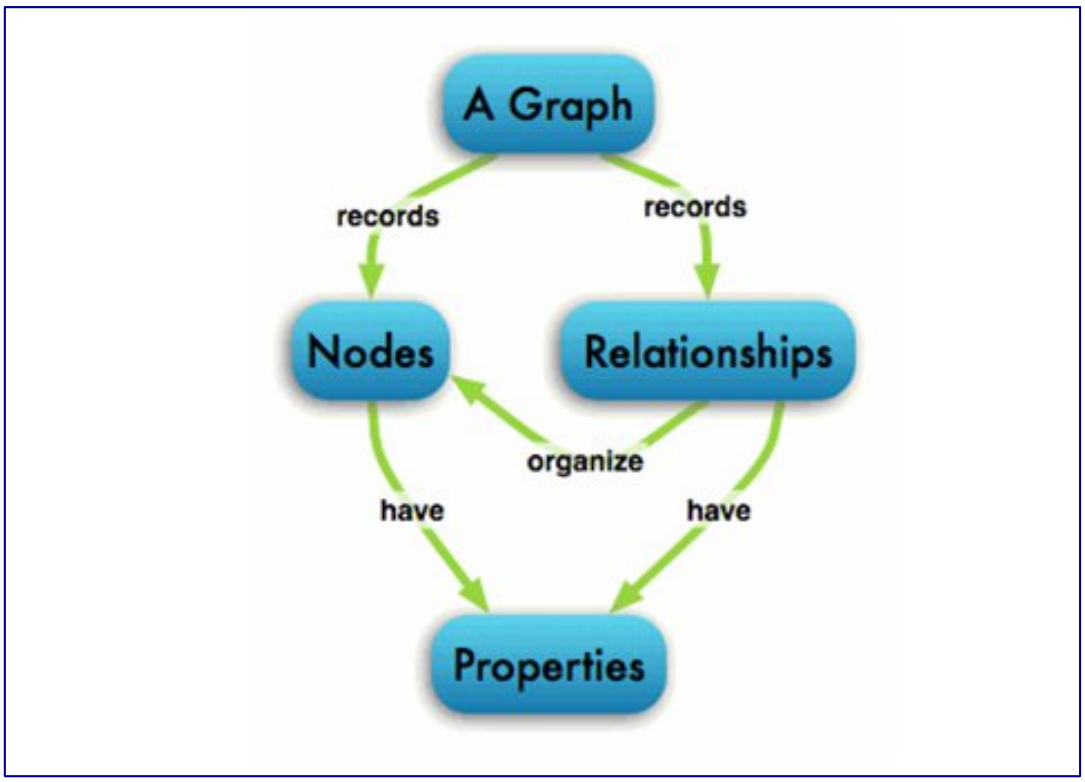

Рисунок 3. Модель базы данных

1.5 Графовая модель хранения данных

Neo4j – это графовая база данных, которая основана на графовой модели данных. Она предоставляет ряд принципов и возможностей, которые делают ее мощным инструментом для работы с графовыми данными.

Графовая модель данных в Neo4j состоит из узлов, связей и свойств. Узлы представляют объекты или сущности в вашей системе, связи определяют отношения между узлами, а свойства содержат атрибуты или данные, связанные с узлами и связями, которая позволяет легко представлять и обрабатывать сложные структуры данных и связи.

Для этого имеется связь-центричность, которая ориентирована на связи между данными. Она позволяет легко и эффективно обрабатывать и анализировать связи и зависимости в данных. Связи в Neo4j могут быть направленными или ненаправленными, а также могут иметь вес или атрибуты,

что позволяет более точно описывать взаимодействия между элементами данных.

Язык запросов Cypher является частью Neo4j и используется для работы с графовой базой данных. Cypher предоставляет интуитивно понятный и выразительный способ выполнения запросов к графовым данным. С его помощью вы можете создавать запросы для поиска, фильтрации, агрегации и анализа данных.

В дополнении данная база данных обладает высокой производительностью с помощью своей оптимизированной структуре данных и алгоритма. Благодаря индексам и кэшированию, обеспечивает быстрый доступ к данным и эффективное выполнение запросов даже для больших графовых баз данных.

Посредством горизонтального масштабирования базы данных путем добавления дополнительных серверов и узлов возможна обработка больших объемов данных и последующее увеличение пропускной способности системы.

Neo4j обеспечивает ACID-совместимые транзакции, что гарантирует надежность и целостность данных. Дополнительно предоставляет мощные возможности для работы с графовыми данными и обработки сложных связей и зависимостей.

Эти принципы и возможности делают Neo4j идеальным выбором для разработки приложений для проведения климатических исследований, где важно анализировать и визуализировать взаимосвязи между различными климатическими параметрами и факторами.

Основные элементы графовой модели данных для такого приложения:

- Узлы (Nodes): В графовой модели данных узлы представляют объекты или сущности в вашей системе. В контексте климатических исследований узлы могут представлять такие элементы, как географические регионы, метеостанции, климатические параметры

(температура, влажность, осадки и т.д.), временные точки и другие связанные сущности.

- Связи (Relationships): Связи определяют отношения между узлами и представляются направленными или ненаправленными ребрами. В контексте климатических исследований связи могут представлять такие отношения, как влияние одного климатического параметра на другой, географическую близость между регионами, исторические данные о климатических изменениях и т.д.
- Свойства (Properties): Узлы и связи могут иметь свойства, которые содержат атрибуты или данные, связанные с ними. В случае климатических исследований свойства могут включать числовые значения климатических параметров, географические координаты, временные метки, атрибуты связей и другие соответствующие данные.

Графовая модель данных позволяет представить сложные сети и зависимости в климатической системе, а также обеспечивает эффективное выполнение запросов для анализа, поиска и визуализации данных, позволит исследователям и климатологам легко обнаруживать скрытые паттерны, анализировать влияние различных факторов на климатические процессы и принимать информированные решения на основе полученных результатов.

1.6 Преимущества Neo4j для климатических исследований

Neo4j предоставляет ряд преимуществ, которые делают его полезным инструментом для проведения климатических исследований:

Гибкая модель данных: Графовая модель данных Neo4j позволяет представлять сложные связи и зависимости между различными климатическими параметрами, регионами, временем и другими факторами.

Это обеспечивает гибкость и масштабируемость при анализе и моделировании климатических данных.

Высокая производительность: Neo4j предоставляет высокую производительность при работе с графовыми данными. Он использует оптимизированные алгоритмы и структуры данных, что позволяет эффективно обрабатывать большие объемы данных и выполнение сложных запросов. Это особенно важно при анализе и обработке сложных климатических сетей и взаимосвязей.

Глубокий анализ связей: Графовая модель Neo4j позволяет проводить глубокий анализ связей между различными климатическими факторами. Он облегчает обнаружение скрытых паттернов, взаимосвязей и зависимостей между параметрами, что помогает исследователям и климатологам получить более полное понимание климатических процессов.

Легкость визуализации данных: Neo4j предоставляет возможности для визуализации графовых данных, что позволяет легко и наглядно представлять сложные взаимосвязи между климатическими параметрами. Визуализация помогает исследователям визуально исследовать и анализировать данные, выявлять паттерны и тренды, а также коммуницировать результаты исследования с другими учеными и заинтересованными сторонами.

Расширяемость и интеграция: Neo4j предоставляет возможность расширения функциональности с помощью пользовательских процедур и плагинов. Это позволяет адаптировать базу данных к специфическим требованиям климатических исследований. Кроме того, Neo4j обеспечивает интеграцию с другими инструментами и технологиями, такими как Python и PyQt5, что упрощает разработку и интеграцию приложения для проведения климатических исследований.

Применение Neo4j в климатических исследованиях позволяет исследователям лучше понимать сложные взаимодействия и зависимости в климатических системах, а также принимать более обоснованные решения на основе полученных результатов.

1.6 Модифицированный индекс Нестерова

Для вычисления индекса пожароопасности было предложено использовать модифицированный индекс Нестерова:

$$g_i = K(R_i)g_{i-1} + T_i d_i \quad (1)$$

При этом значение коэффициента K(R) задается в зависимости от суточной суммы осадков. Размерность g есть (°C) 2 · сут. Вычисление индекса g проводится в теплую половину года по суткам с положительной температурой [4].

Модифицированный индекс Нестерова является методом для выявления пожароопасности в регионе. В нашем случае был выбран город Мирный.

Индекс основан на анализе различных факторов, включая климатические показатели, погодные условия, географические особенности и другие важные параметры, которые могут влиять на возникновение и распространение пожаров. Он является числовым показателем, который позволяет оценить уровень пожароопасности в определенном регионе на основе анализа различных факторов и параметров. Представляет собой комплексный индикатор, учитывающий взаимодействие множества факторов, связанных с возникновением и распространением пожаров.

Для выявления пожароопасности необходимо учесть следующие шаги:

1. Сбор данных: Первоначально, были собраны данные с метеоцентра, который предоставил заказчик, температура воздуха, температура точки росы, скорость и направление ветра, а также географические особенности региона, такие как рельеф местности, наличие лесов, растительности.
2. Подготовка данных: После сбора данных, обработать и подготовить для дальнейшего анализа. Включает очистку данных от выбросов и

ошибок, преобразование данных в нужный формат, агрегацию данных и установление взаимосвязей между различными параметрами.

3. Определение весовых коэффициентов: Для вычисления модифицированного индекса Нестерова необходимо определить весовые коэффициенты для каждого параметра. Весовые коэффициенты определяют степень важности каждого параметра в контексте пожароопасности.
4. Расчет индекса: После определения весовых коэффициентов, производится расчет модифицированного индекса Нестерова. Это может включать комбинацию взвешенных параметров с использованием соответствующих математических формул. Результатом расчета будет числовое значение, отражающее уровень пожароопасности в регионе.
5. Интерпретация результатов: Полученное числовое значение индекса пожароопасности может быть интерпретировано для принятия соответствующих мер по предотвращению и борьбе с пожарами. Например, высокое значение индекса может указывать на высокий уровень пожароопасности, что требует усиленной мониторинговой деятельности, эвакуации или предоставления дополнительных ресурсов для пожаротушения.

Интеграция модифицированного индекса Нестерова в приложение для расчета комплексного показателя пожароопасности позволит эффективно анализировать и оценивать уровень пожароопасности в заданном регионе на основе различных факторов. Это поможет пользователям принимать информированные решения и предпринимать соответствующие меры для предотвращения пожаров и минимизации их последствий.

Вывод по главе 1

В данной главе был проведен анализ предметной области, связанной с пожарами и показателем их пожароопасности. Были рассмотрены основные причины возникновения пожаров, что позволило получить общее представление о факторах, влияющих на пожарную безопасность.

Задача разработки приложения для расчета комплексного показателя пожароопасности была поставлена. Это приложение должно позволить загружать исходные данные о климатических показателях, а затем вычислять показатель пожароопасности с использованием модифицированного индекса Нестерова.

В ходе обзора аналогов были рассмотрены существующие приложения и системы, предназначенные для расчета показателя пожароопасности. Это позволило выявить основные преимущества и недостатки существующих решений и определить направления для дальнейшей разработки.

Были выбраны инструменты разработки, включая язык программирования Python и библиотеку PyQt5 для создания графического интерфейса. Также была выбрана СУБД Neo4j для хранения и обработки данных. Эти инструменты обладают необходимыми функциональностями и возможностями для разработки приложения.

В главе была рассмотрена графовая модель хранения данных, которая позволяет эффективно организовать информацию о климатических показателях и связях между ними. Это способствует удобному и быстрому доступу к данным при расчете показателя пожароопасности.

Также были выделены преимущества СУБД Neo4j для климатических исследований, включая возможность эффективной обработки связей и взаимосвязей между данными. Это позволяет более точно учитывать различные факторы и условия, влияющие на пожароопасность.

Наконец, был представлен модифицированный индекс Нестерова, который является основой для расчета показателя пожароопасности. Этот индекс учитывает различные параметры и факторы, включая метеорологические

данные, что позволяет более точно оценить вероятность возникновения пожаров.

В целом, проведенный анализ предметной области и выбранные инструменты позволяют утверждать о целесообразности разработки приложения для расчета комплексного показателя пожароопасности. Это приложение будет иметь преимущества в сравнении с существующими аналогами и обеспечит возможность прогнозирования вероятности пожаров на основе собранных и анализируемых данных о климатических показателях.

ГЛАВА 2. РАЗРАБОТКА ПРИЛОЖЕНИЯ

2.1 Общая структура приложения

Общая структура приложения для расчета комплексного показателя пожароопасности организована следующим образом:

Интерфейс пользователя: В приложении есть интерфейс, который позволяет исследователям взаимодействовать с данными, проводить анализ и визуализацию результатов исследований. Интерфейс пользователя может быть реализован с использованием PyQt5, предоставляя удобный и интуитивно понятный интерфейс для работы с приложением.

База данных Neo4j: Neo4j является основным хранилищем данных для приложения. Она содержит графовую модель данных, в которой хранятся данные для вычисления показателя пожароопасности, такие как:

- Синоптический индекс станции;
- Год по Гринвичу;
- Месяц по Гринвичу;
- День по Гринвичу;
- Срок по Гринвичу;
- Направление ветра;
- Средняя скорость ветра;
- Средняя скорость ветра;
- Максимальная скорость ветра;
- Сумма осадков;
- Температура поверхности почвы;
- Температура воздуха по сухому терм-ру;
- Относительная влажность воздуха;
- Температура точки росы;
- Атмосферное давление на уровне станции;
- Величина барической тенденции.

Ввод и хранение данных: Приложение поддерживает механизм ввода и хранения данных. Это может включать импорт данных из внешних источников, ввод данных вручную или автоматическую загрузку данных с метеостанций. Полученные данные сохраняются в базе данных Neo4j для последующего анализа и обработки.

Анализ и визуализация данных: Приложение должно предоставлять возможности для анализа и визуализации климатических данных. Это может включать выполнение сложных запросов к базе данных Neo4j для извлечения нужной информации, а также использование графических библиотек для создания графиков, диаграмм и карт для наглядного представления результатов исследований.

Импорт и экспорт данных: Приложение может поддерживать функциональность импорта и экспорта данных. Это позволяет пользователям загружать данные из внешних источников и сохранять результаты исследований в удобном формате, например, в CSV или Excel.

Общая структура приложения обеспечивает удобный интерфейс для исследователей, эффективное взаимодействие с базой данных Neo4j и мощные инструменты для анализа и визуализации климатических данных. Вся программа запускается в файле app.py в файле mainwindow.py (рис. 4).

Рисунок 4. Главное окно приложения

Структура данных и моделирования данных для расчета комплексного показателя пожароопасности:

Создание узлов:

Узел "Станция" – содержит информацию о синоптическом индексе станции.

Узел "Дата" – содержит информацию о годе, месяце, дне и сроке по Гринвичу.

Узел "Погодные параметры" – содержит информацию о направлении ветра, средней скорости ветра, максимальной скорости ветра, сумме осадков, температуре поверхности почвы, температуре воздуха, относительной влажности воздуха, температуре точки росы, атмосферном давлении и величине барической тенденции.

Определение отношений

Отношение "ИМЕЕТ_ПАРАМЕТРЫ" связывает узлы "Станция" и "Погодные параметры". Это отношение указывает, что погодные параметры относятся к определенной станции.

Отношение "СООТВЕТСТВУЕТ_ДАТЕ" связывает узлы "Погодные параметры" и "Дата". Это отношение указывает, какие погодные параметры относятся к конкретной дате.

Определение этих основных сущностей и их атрибутов поможет структурировать данные для расчета пожароопасности индекса в базе данных Neo4j. Взаимосвязи между регионами, метеостанциями, показателями пожароопасности и другими сущностями позволят эффективно хранить и анализировать данные, а также проводить расчеты и визуализацию результатов.

Основные данные берутся с сайта meteo.ru — Федеральная служба по гидрометеорологии и мониторингу окружающей среды Технологии Аисори Специализированные массивы для климатических исследований.

Выборка данных обеспечивается Web-технологией:

«Аисори – Удаленный доступ к ЯОД-архивам» (рис.5).

Технологии Аисори – это общее название семейства программных продуктов, предназначенных для эффективной работы с архивами Государственного фонда данных о состоянии природной среды (Госфонд). Госфонд содержит десятки архивов по различным разделам изучения природной среды (метеорология, гидрология, аэрология, океанография, загрязнения сред и т.п.) за период с 1874 г. по настоящее время. Объем данных в большинстве архивов составляет от 1 до 20 Гбайт.

Все технологии Аисори имеют в своей основе две общие черты:

Структура архивов описывается средствами Языка описания гидрометеорологических данных (ЯОД), который реализует иерархическую модель данных. Этот язык принят в качестве отраслевого стандарта Гидрометеослужбы в 1978 г.

В качестве средства управления данными в технологиях используется ядро Аисори – специализированный пакет программного обеспечения, включающий Транслятор ЯОД, Компилятор запросов и Процессор запросов.

Семейство технологий Аисори состоит из пяти программных продуктов:

Аисори – Удаленный доступ к ЯОД-архивам” – Web технология в среде Байконура, обеспечивающая удаленный доступ пользователей к ЯОД-архивам для получения из них любых выборок табличной структуры. Доступ к ЯОД-архивам осуществляется по сетям Интернет или Интранет. Пользователь обращается за выборкой данных, используя обычный браузер. Таким образом, это клиент серверная технология, в которой пользователем является тонкий клиент [5].

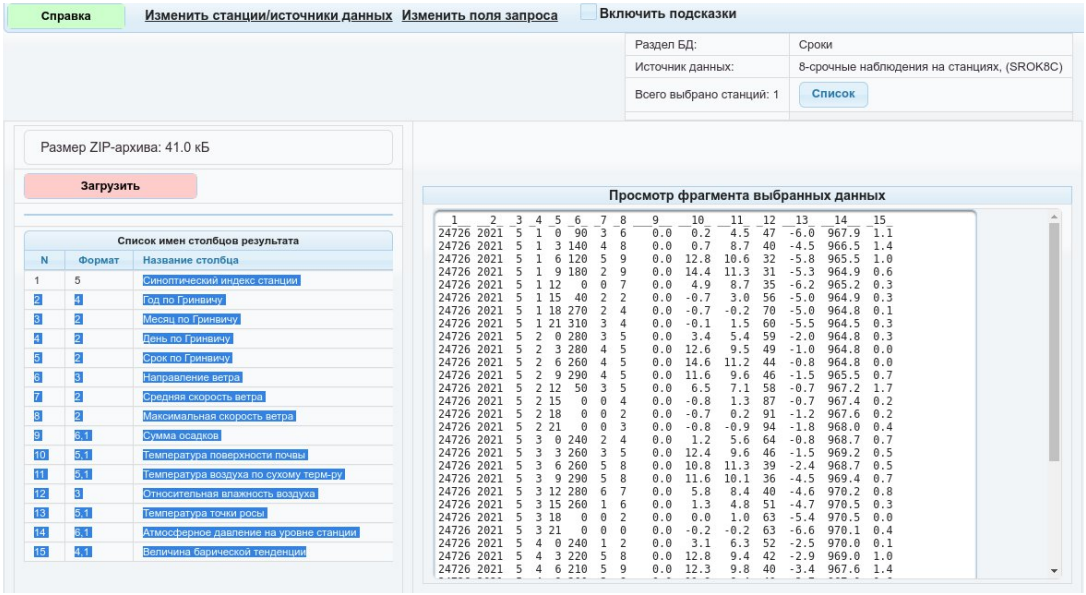

Рисунок 5. Удаленный доступ к ЯОД архивам

Аисори – Интерактивная оболочка для табличных данных” – локальное приложение, предназначенное для представления различных расчетов, результаты которых имеют табличную форму (например, справочник по климату, сведения о полноте и пропусках в архивах и т.п.). Имеет удобный для пользователей непрофессионалов интерактивный интерфейс, в том числе средства картографического отображения пунктов наблюдений.

Технологии Аисори разработаны в Лаборатории автоматизированной информационной системы ФГБУ "ВНИИГМИ-МЦД" Росгидромета (г. Обнинск Калужской области).

2.2 Реализация функциональности приложения

Ввод и хранение данных о климатических параметрах

Для реализации функциональности ввода и хранения данных о климатических параметрах в приложении, предлагается использовать следующие методы:

1. Выгрузка данных: Данные выгружаются с нужными параметрами с веб-сайта aisori-m-meteo.ru.

2. Импорт и конвертация данных: Реализовать механизм, который позволяет одновременно выполнять импорт данных с веб-сайта и их конвертацию в формат, совместимый с базой данных Neo4j. Для этого используются библиотеки json, threading, neo4j (рис. 6).

```
def send_to_database(self):
    # Получение настроек подключения к базе данных
    settings_win = Settings()
    address = settings_win.get_setting('Connection', 'Address')
    login = settings_win.get_setting('Connection', 'Login')
    password = settings_win.get_setting('Connection', 'Password')

    if not self.data:
        QMessageBox.warning(self, 'Предупреждение', 'Нет данных для отправки в базу данных.')
        return

    try:
        # Подключение к базе данных Neo4j
        driver = GraphDatabase.driver(f"neo4j+s://{address}", auth=(login, password))

        # Создание диалогового окна с ползунком загрузки
        progress_dialog = QProgressDialog("Отправка данных...", "Отмена", 0, len(self.data))
        progress_dialog.setWindowModality(Qt.WindowModal)
        progress_dialog.setWindowTitle("Отправка данных")
        progress_dialog.setAutoClose(False)
        progress_dialog.setAutoReset(False)

        with driver.session() as session:
            # Отправка данных в базу данных
            for index, values in enumerate(self.data):
                if progress_dialog.wasCanceled():
                    break

                query = """
                CREATE (s:Station {SynopticIndex: $synopticIndex})
                MERGE (w:WeatherData {
                    YearInGreenwich: $yearInGreenwich,
                    MonthInGreenwich: $monthInGreenwich,
                    DayInGreenwich: $dayInGreenwich,
                    TimeInGreenwich: $timeInGreenwich,
                    YearFromSourceLocal: $yearFromSourceLocal,
                    MonthFromSourceLocal: $monthFromSourceLocal,
                    DayFromSourceLocal: $dayFromSourceLocal,
                    TimeFromSourceLocal: $timeFromSourceLocal,
                    LocalTime: $localTime,
                    AverageWindSpeed: $averageWindSpeed,
                    MaximumWindSpeed: $maximumWindSpeed,
                    PrecipitationSum: $precipitationSum,
                    MinSoilSurfaceTemperature: $minSoilSurfaceTemperature,
                    MaxSoilSurfaceTemperature: $maxSoilSurfaceTemperature
                })
```

Рисунок 6. Код отправки данных

3. Хранение данных: Neo4j для хранения данных о климатических параметрах. Neo4j предлагает графовую модель данных, что позволяет

эффективно организовать хранение и структурирование данных, а также проводить различные операции с графами (рис.7).

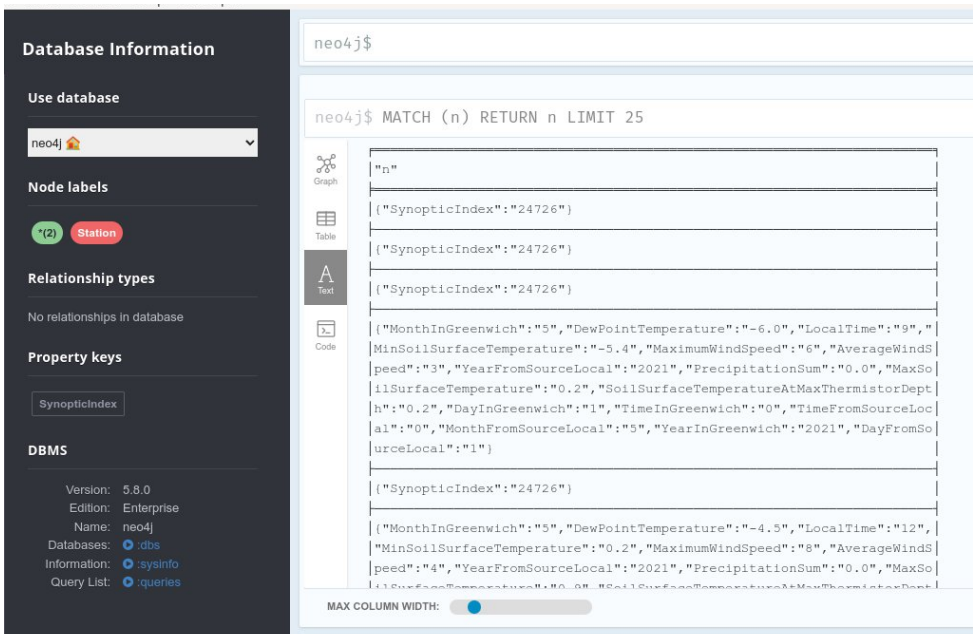

Рисунок 7. Neo4j Browser

4. Отображение данных: Реализовано отображение данных в виде таблицы. Пользователь может просмотреть информацию по следующим столбцам:

- Синоптический индекс станции
- Год по Гринвичу
- Месяц по Гринвичу
- День по Гринвичу
- Срок по Гринвичу
- Направление ветра
- Средняя скорость ветра
- Максимальная скорость ветра
- Сумма осадков
- Температура поверхности почвы
- Температура воздуха по сухому термометру

- Относительная влажность воздуха
- Температура точки росы
- Атмосферное давление на уровне станции
- Величина барической тенденции

Код отображения данных. (рис. 8)

```
def __init__(self):
    super().__init__()
    self.setWindowTitle("Load")
    self.setObjectName("Load")
    self.resize(1900, 700)
    self.setStyleSheet("background-color: rgb(108, 136, 177);\n"
                       "color: rgb(255, 255, 255);")
    # Создание таблицы для отображения данных
    self.table_widget = QTableWidget()
    self.table_widget.setColumnCount(15)
    self.table_widget.setHorizontalHeaderLabels([
        'Synoptic Index', 'Year in Greenwich', 'Month in Greenwich', 'Day in Greenwich', 'Time in Greenwich',
        'Year from Source Local', 'Month from Source Local', 'Day from Source Local', 'Time from Source Local',
        'Local Time', 'Average Wind Speed', 'Maximum Wind Speed',
        'Precipitation Sum', 'Min Soil Surface Temperature', 'Max Soil Surface Temperature'
    ])
    self.table_widget.resizeColumnsToContents()
    # Создание кнопок
    self.btn_select_file = QPushButton('Choose a file')
    self.btn_convert_to_json = QPushButton('Convert to JSON')
    self.btn_send_to_database = QPushButton('Send to DB')
    self.btn_show_table = QPushButton('Calculate')

    # Назначение обработчиков событий для кнопок
    self.btn_select_file.clicked.connect(self.select_file)
    self.btn_convert_to_json.clicked.connect(self.convert_to_json)
    self.btn_send_to_database.clicked.connect(self.send_to_database)
    self.btn_show_table.clicked.connect(self.open_data_window)

    # Создание вертикального компоновщика и добавление кнопок и таблицы
    layout = QVBoxLayout()
    layout.addWidget(self.btn_select_file)
    layout.addWidget(self.btn_convert_to_json)
    layout.addWidget(self.btn_send_to_database)
    layout.addWidget(self.btn_show_table)
    layout.addWidget(self.table_widget)
    self.btn_save_excel = QPushButton('Save Excel')
    self.btn_save_excel.clicked.connect(self.save_as_excel)
    # Добавление кнопки "Save Excel" в компоновщик
    layout.addWidget(self.btn_save_excel)
    # Установка компоновщика в диалоговое окно
    self.setLayout(layout)

    # Переменная для хранения данных
    self.data = []
```

Рисунок 8. Код отображение данных

5. Расчет: Функциональность расчета комплексного показателя пожароопасности с использованием модифицированного индекса Нестерова.

Пользователь может ввести необходимые данные для расчета. После ввода данных пользователь может нажать кнопку "Вычислить", чтобы выполнить расчет и получить результат.

Код расчета комплексного показателя пожароопасности и отображения результата. (рис. 9)

```
def calculate(self):
    # Индексы столбцов в таблице
    wind_direction_column = 4
    average_wind_speed_column = 5
    maximum_wind_speed_column = 6
    precipitation_sum_column = 7
    soil_surface_temperature_column = 8
    air_temperature_column = 10
    relative_humidity_column = 11
    dew_point_temperature_column = 12

    # Получение данных из таблицы
    data = []
    for row in range(self.table_widget.rowCount()):
        values = [
            self.table_widget.item(row, wind_direction_column).text(),
            self.table_widget.item(row, average_wind_speed_column).text(),
            self.table_widget.item(row, maximum_wind_speed_column).text(),
            self.table_widget.item(row, precipitation_sum_column).text(),
            self.table_widget.item(row, soil_surface_temperature_column).text(),
            self.table_widget.item(row, air_temperature_column).text(),
            self.table_widget.item(row, relative_humidity_column).text(),
            self.table_widget.item(row, dew_point_temperature_column).text()
        ]
        data.append(values)

    # Выполнение вычислений
    results = []
    for row, values in enumerate(data):
        result = self.perform_calculations(values)
        results.append(result)

        # Отображение данных и результатов
        self.display_result(row, values, result)

    # Вывод общего результата
    total_result = sum(results) / len(results)
    QMessageBox.information(self, 'Общий результат', f'Общий результат вычислений: {total_result}')
```

Рисунок 9. Код вычисления данных

Для реализации функциональности анализа и визуализации данных в приложении были использованы следующие подходы.

Анализ данных: Применены статистические методы и алгоритмы для извлечения информации из данных о климатических параметрах.

Визуализация данных: Использованы библиотеки визуализации данных, такие как Matplotlib, Seaborn или Plotly, для создания графиков, диаграмм,

тепловых карт и других визуальных представлений данных о климатических параметрах.

Сохранение в Excel формате: Реализована функциональность сохранения визуализированных данных в формате Excel (рис.10).

```
def calculate(self):
    # Индексы столбцов в таблице
    wind_direction_column = 4
    average_wind_speed_column = 5
    maximum_wind_speed_column = 6
    precipitation_sum_column = 7
    soil_surface_temperature_column = 8
    air_temperature_column = 10
    relative_humidity_column = 11
    dew_point_temperature_column = 12

    # Получение данных из таблицы
    data = []
    for row in range(self.table_widget.rowCount()):
        values = [
            self.table_widget.item(row, wind_direction_column).text(),
            self.table_widget.item(row, average_wind_speed_column).text(),
            self.table_widget.item(row, maximum_wind_speed_column).text(),
            self.table_widget.item(row, precipitation_sum_column).text(),
            self.table_widget.item(row, soil_surface_temperature_column).text(),
            self.table_widget.item(row, air_temperature_column).text(),
            self.table_widget.item(row, relative_humidity_column).text(),
            self.table_widget.item(row, dew_point_temperature_column).text()
        ]
        data.append(values)

    # Выполнение вычислений
    results = []
    for row, values in enumerate(data):
        result = self.perform_calculations(values)
        results.append(result)

        # Отображение данных и результатов
        self.display_result(row, values, result)

    # Вывод общего результата
    total_result = sum(results) / len(results)
    QMessageBox.information(self, 'Общий результат', f'Общий результат вычислений: {total_result}')
```

Рисунок 10. Код сохранения данных в таблицу

Сохранение в JPEG формате: Реализовать функциональность сохранения визуализированных данных в формате JPEG. Для этого можно использовать функции библиотеки Matplotlib, которые позволяют сохранять графики и диаграммы в различных графических форматах, включая JPEG.

Вывод по главе 2

В данной главе было представлено разработанное приложение для расчета комплексного показателя пожароопасности. Основной целью приложения является расчет комплексного показателя пожароопасности.

В рамках главы была рассмотрена общая структура приложения, которая включает компоненты пользовательского интерфейса, моделирование данных, реализацию функциональности и анализ данных. Был выбран язык программирования Python и библиотека PyQt5 для разработки приложения, а также СУБД Neo4j для хранения и обработки данных.

Особое внимание уделено моделированию данных, где были определены сущности и атрибуты, необходимые для хранения информации о погодных параметрах. Загрузка и выгрузка данных в приложение, их преобразование и импорт в СУБД Neo4j также были реализованы.

Важным аспектом разработанного приложения является анализ и визуализация данных. Был использован модифицированный индекс Нестерова для расчета показателя пожароопасности, который позволяет оценить вероятность возникновения пожаров в регионе на основе имеющихся данных. Результаты анализа данных были визуализированы в виде таблицы, что облегчает восприятие и позволяет пользователям легко оценить степень пожароопасности.

Заключение

Разработка приложения для проведения климатических исследований с использованием СУБД Neo4j представляет собой важный шаг в направлении эффективного анализа и управления данными о климатических параметрах.

Neo4 предоставляет удобные и гибкие возможности хранения, структурирования и анализа связей между сущностями.

Применение Neo4j в климатических исследованиях позволяет исследователям и специалистам в области климата эффективно работать с данными, анализировать их, выявлять закономерности и зависимости, а также проводить визуализацию результатов. Это способствует более глубокому пониманию климатических процессов и помогает принимать обоснованные решения в области климатической политики и окружающей среды.

Разработка приложения позволяет удобно вводить и хранить данные о климатических параметрах, а также обеспечивает возможность импорта и экспорта данных в различных форматах, таких как CSV и Excel, что упрощает обмен данными с другими системами и исследовательскими группами.

В целом, использование СУБД Neo4j в разработке приложения для климатических исследований демонстрирует преимущества графовой модели данных, предоставляя удобство, гибкость и эффективность в работе с данными о климатических параметрах. Такое приложение может стать ценным инструментом для исследователей и специалистов, способствуя улучшению понимания и принятию решений в области климата и окружающей среды.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рейтинг систем управления графовыми базами данных DB-Engines [Электронный ресурс] – URL: https://db-engines.com/en/ranking/graph+dbms (Дата обращения: 22 мая 2023 года).
2. Официальный сайт. Руководство по эксплуатации Neo4j [Электронный ресурс] – URL: https://Neo4j.com/docs/operations-manual/3.2/ (Дата обращения: 22 мая 2023 года).
3. Лесные пожары на территории России [Электронный ресурс] URL:http://downloads.igce.ru/publications/pemem/PEMEM25/15_Sherstyukov_Sherstyukov.pdf (Дата обращения: 22 мая 2023 года).
4. Удаленный доступ к ЯОД архивам Аисори – [Электронный ресурс] – URL: http://meteo.ru/it/178-aisori (Дата обращения: 22 мая 2023 года).
5. Официальная документация по Python [Электронный ресурс] – URL: https://docs.python.org/3.9/ (Дата обращения: 22 мая 2023 года).
6. Официальная документация по PyQt5 [Электронный ресурс] – URL: https://doc.qt.io/qtforpython-6/ (Дата обращения: 22 мая 2023 года).
7. Сайт FireWare Index System [Электронный ресурс] – URL: https://climate.copernicus.eu/fire-weather-index (Дата обращения: 22 мая 2023 года).
8. SmartUnit(Сильван)[Электронный ресурс] – URL: https://smartplatform.pro/product/ (Дата обращения: 22 мая 2023 года).

Программный код приложения

app.py

```
import sys
from PyQt5 import QtWidgets
from utils.MainWindow import MainWindow

class App(QtWidgets.QMainWindow):
    def __init__(self):
        super(App, self).__init__()
        self.ui = MainWindow()
        self.ui.setupUi(self)

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = App()
    window.show()
    sys.exit(app.exec_())
```

mainwindow.py

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import QThread, QObject, pyqtSignal
from PyQt5.QtWidgets import QMessageBox, QFileDialog
from utils.About import About
from utils.Settings import Settings
from utils.LoadData import Load, DataWindow
from utils.Calculate import FireSafetyCalculator
from neo4j import GraphDatabase
import logging
import sys


# Создание логгера
logger = logging.getLogger('connection_logger')
logger.setLevel(logging.INFO)

# Создание обработчика для вывода в терминал
handler = logging.StreamHandler(sys.stdout)
handler.setLevel(logging.INFO)

# Создание форматировщика логов
```

```
formatter = logging.Formatter('%(asctime)s – %(levelname)s – %(message)s')
handler.setFormatter(formatter)

# Добавление обработчика в логгер
logger.addHandler(handler)

class MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(888, 563)

        # Устанавливаем шрифт
        font = QtGui.QFont()
        font.setFamily("URW Gothic")
        font.setKerning(True)
        MainWindow.setFont(font)

        # Устанавливаем стиль главного окна
        MainWindow.setStyleSheet("background-color: rgb(108, 136, 177);\n"
                                 "color: rgb(255, 255, 255);")

        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setStyleSheet("")
        self.centralwidget.setObjectName("centralwidget")

        self.MainWindow_text = QtWidgets.QLabel(self.centralwidget)
        self.MainWindow_text.setGeometry(QtCore.QRect(490, 60, 261, 61))
        font = QtGui.QFont()
        font.setFamily("URW Gothic")
        font.setPointSize(40)
        self.MainWindow_text.setFont(font)
        self.MainWindow_text.setStyleSheet("\n"
                                           "background-color: rgb(108, 136, 177);")
        self.MainWindow_text.setAlignment(QtCore.Qt.AlignCenter)
        self.MainWindow_text.setObjectName("MainWindow_text")


        self.load_btn = QtWidgets.QPushButton(self.centralwidget)
        self.load_btn.setGeometry(QtCore.QRect(530, 150, 190, 51))
        font = QtGui.QFont()
        font.setPointSize(20)
        self.load_btn.setFont(font)
        self.load_btn.setObjectName("load_btn")
```

```
self.view_btn = QtWidgets.QPushButton(self.centralwidget)
self.view_btn.setGeometry(QtCore.QRect(530, 230, 190, 51))
font = QtGui.QFont()
font.setPointSize(20)
self.view_btn.setFont(font)
self.view_btn.setObjectName("view_btn")

self.calculate_btn = QtWidgets.QPushButton(self.centralwidget)
self.calculate_btn.setGeometry(QtCore.QRect(530, 310, 190, 51))
font = QtGui.QFont()
font.setPointSize(20)
self.calculate_btn.setFont(font)
self.calculate_btn.setObjectName("calculate_btn")

self.settings_btn = QtWidgets.QPushButton(self.centralwidget)
self.settings_btn.setGeometry(QtCore.QRect(530, 390, 190, 51))
font = QtGui.QFont()
font.setPointSize(20)
self.settings_btn.setFont(font)
self.settings_btn.setObjectName("settings_btn")

self.image_1 = QtWidgets.QLabel(self.centralwidget)
self.image_1.setGeometry(QtCore.QRect(50, 80, 400, 400))
self.image_1.setText("")
self.image_1.setPixmap(QtGui.QPixmap("assets/earth.png"))
self.image_1.setScaledContents(True)
self.image_1.setObjectName("image_1")
self.image_1.raise_()
self.settings_btn.raise_()
# self.data_btn.raise_()
self.view_btn.raise_()
self.calculate_btn.raise_()
self.MainWindow_text.raise_()

# Установка центрального виджета главного окна
MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 888, 24))
self.menubar.setObjectName("menubar")

self.menu_about = QtWidgets.QMenu(self.menubar)
self.menu_about.setObjectName("menu_about")
```

```
self.menu_settings = QtWidgets.QMenu(self.menubar)
font = QtGui.QFont()
font.setFamily("URW Gothic")
self.menu_settings.setFont(font)
self.menu_settings.setObjectName("menu_settings")

self.menu_open = QtWidgets.QMenu(self.menubar)
font = QtGui.QFont()
font.setFamily("URW Gothic")
self.menu_open.setFont(font)
self.menu_open.setObjectName("menu_open")

self.menuSave = QtWidgets.QMenu(self.menubar)
self.menuSave.setObjectName("menuSave")

# Установка строки меню в главном окне
MainWindow.setMenuBar(self.menubar)

self.file_btn = QtWidgets.QAction(MainWindow)
self.file_btn.setObjectName("file_btn")

self.data_btn_2 = QtWidgets.QAction(MainWindow)
self.data_btn_2.setObjectName("data_btn_2")

self.save_btn = QtWidgets.QAction(MainWindow)
self.save_btn.setObjectName("save_btn")

self.saveAs_btn = QtWidgets.QAction(MainWindow)
self.saveAs_btn.setObjectName("saveAs_btn")

self.actionData_2 = QtWidgets.QAction(MainWindow)
font = QtGui.QFont()
font.setFamily("URW Gothic")
self.actionData_2.setFont(font)
self.actionData_2.setObjectName("actionData_2")

self.actionData_3 = QtWidgets.QAction(MainWindow)
self.actionData_3.setObjectName("actionData_3")

self.plot_action = QtWidgets.QAction(MainWindow)
self.plot_action.setObjectName("plot_action")

self.settings_action = QtWidgets.QAction(MainWindow)
```

```
self.settings_action.setObjectName("settings_action")
self.settings_action.setText("Settings")

self.about_action = QtWidgets.QAction(MainWindow)
self.about_action.setObjectName("about_action")
self.about_action.setText("About")

self.menu_settings.addAction(self.settings_action)
self.menu_about.addAction(self.about_action)
self.menu_open.addAction(self.file_btn)
self.menu_open.addAction(self.data_btn_2)
self.menuSave.addAction(self.save_btn)
self.menuSave.addAction(self.saveAs_btn)

self.menubar.addAction(self.menu_open.menuAction())
self.menubar.addAction(self.menuSave.menuAction())
self.menubar.addAction(self.menu_settings.menuAction())
self.menubar.addAction(self.menu_about.menuAction())

MainWindow.setWindowTitle("MainWindow")
self.MainWindow_text.setText("ClimStore")
self.settings_btn.setText("Settings")
# self.data_btn.setText("Data")
self.view_btn.setText("View")
self.calculate_btn.setText("Calculate")
self.menu_about.setTitle("About")
self.menu_settings.setTitle("Settings")
self.menu_open.setTitle("Open")
self.menuSave.setTitle("Save")
self.file_btn.setText("File")
self.data_btn_2.setText("Data")
self.save_btn.setText("Save")
self.saveAs_btn.setText("Save as")
self.actionData_2.setText("Data")
self.load_btn.setText("Load")
self.actionData_3.setText("Data")
self.plot_action.setText("Plot")
self.neo4j_switch = QtWidgets.QPushButton(self.centralwidget)
self.neo4j_switch.setGeometry(QtCore.QRect(20, 500, 100, 30))
self.neo4j_switch.setCheckable(True)
self.neo4j_switch.setObjectName("neo4j_switch")
self.neo4j_switch.setText("Connect")
self.load_btn.clicked.connect(self.load_btn_clicked)
self.calculate_btn.clicked.connect(self.open_calculation_window)
```

```
        self.settings_btn.clicked.connect(self.open_settings)
        self.menu_settings.triggered.connect(self.open_settings)
        self.menu_about.triggered.connect(self.show_about)
        self.neo4j_switch.clicked.connect(self.toggle_neo4j_connection)
    def toggle_neo4j_connection(self):
        if self.neo4j_switch.isChecked():
            if self.connect_to_neo4j():
                self.neo4j_switch.setText("Disconnect")
            else:
                QMessageBox.critical(self.centralwidget, "Connection Error", "Failed to
connect to Neo4j.")
                self.neo4j_switch.setChecked(False)
        else:
            self.disconnect_from_neo4j()
            self.neo4j_switch.setText("Connect")
    def open_calculation_window(self):
        self.calculation_window = FireSafetyCalculator()
        self.calculation_window.show()

    def connect_to_neo4j(self):
        # Установка соединения с базой данных Neo4j
        settings_win = Settings()
        address = settings_win.get_setting('Connection', 'Address')
        login = settings_win.get_setting('Connection', 'Login')
        password = settings_win.get_setting('Connection', 'Password')
        # Установка соединения с базой данных Neo4j
        try:
            logger.info(f"Connecting to Neo4j – Address: {address}, Login: {login}")
            self.driver = GraphDatabase.driver(f"neo4j+s://{address}", auth=(login,
password), connection_timeout=10)
            self.session = self.driver.session()
            if self.is_neo4j_connected():
                print("Connected to Neo4j")
                return True
            else:
                print("Not Connected")
                return False
        except Exception as e:
            error_message = "Connection failed: " + str(e)
            logger.error(error_message)
            self.show_error_message(error_message)
            return False

    def disconnect_from_neo4j(self):
```

```
        # Разрыв соединения с базой данных Neo4j
        self.session.close()
        self.driver.close()
        print('Disconnected')

    def open_settings(self):
        settings_win = Settings()
        settings_win.open_dialog()


    def show_about(self):
        about_ui = About()
        about_ui.open_dialog()

    def show_error_message(self, message):
        error_box = QMessageBox()
        error_box.setIcon(QMessageBox.Critical)
        error_box.setWindowTitle("Error")
        error_box.setText(message)
        error_box.setStandardButtons(QMessageBox.Ok)
        error_box.button(QMessageBox.Ok).setStyleSheet("QPushButton { text-
align: center; }")
        error_box.exec_()

    def is_neo4j_connected(self):
        try:
            # Проверка наличия активного соединения с базой данных Neo4j
            # Если соединение отсутствует, возникнет исключение
            self.session.run("MATCH (n) RETURN n LIMIT 1")
            return True
        except Exception:
            return False
    def load_btn_clicked(self):
        if self.neo4j_switch.isChecked():
            if self.is_neo4j_connected():
                self.load_window = Load()
                self.load_window.show()
            else:
                # Предупреждаем пользователя, что он должен сначала
подключиться к Neo4j
                QMessageBox.information(self.centralwidget, "Connection Required",
"Please connect to Neo4j first.")
        else:
```

```
        # Предупреждаем пользователя, что он должен сначала подключиться к Neo4j
        QMessageBox.information(self.centralwidget, "Connection Required", "Please connect to Neo4j first.")
```