МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(Наименование кафедры, центра, департамента)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка программного обеспечения для метеомониторинга линий электропередач с использованием искусственного интеллекта

Обучающийся М. Л. Порозов _____
(Инициалы Фамилия) (личная подпись)

Руководитель д.т.н., доцент, С.В. Мкртычев
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема выпускной квалификационной работы: «Разработка программного обеспечения для метеомониторинга линий электропередач с использованием искусственного интеллекта».

Объектом исследования являются процессы метеомониторинга и прогнозирования обледенения линий электропередач. Предметом исследования является программное обеспечение для анализа метеоданных, прогнозирования гололедообразования и уведомления диспетчеров.

Цель работы – разработка группы микросервисов, которые автоматизируют сбор метеоданных, анализируют их с использованием алгоритмов искусственного интеллекта и предоставляют результаты в удобной визуальной форме, помогая диспетчерам принимать вывод.

Работа включает введение, три главы, заключение и список используемой литературы.

В первой главе обосновывается актуальность разработки программного обеспечения для метеомониторинга, анализируются существующие решения, а также определяются задачи, которые необходимо решить в процессе работы.

Во второй главе представлено проектирование системы, включая разработку логической модели данных, структуры программного обеспечения и интерфейсов пользователя.

В третьей главе освещается процесс разработки группы микросервисов: выбор технологий и инструментов, описание модулей системы, а также этапы тестирования.

Бакалаврская работа содержит 67 страниц текста, 37 рисунков, 4 таблицы и 29 источников. В приложении представлены фрагменты кода.

Abstract

The title of the graduation work is Development of Software for Power Line Weather Monitoring Using Artificial Intelligence.

The graduation work consists of an introduction, three chapters, a conclusion, and a list of references. It includes 67 pages of text, 37 figures, 4 tables, and 29 references. Code fragments are presented in the appendix.

The aim of this graduation work is to develop a set of microservices that automate the collection of meteorological data, analyze it using artificial intelligence algorithms, and provide the results in a convenient visual format to support dispatchers in decision-making.

The object of the graduation work is the process of weather monitoring and icing prediction for power transmission lines.

The subject of the graduation work is software for meteorological data analysis, icing forecasting, and dispatcher notification.

The key issue of the graduation work is the development of intelligent microservice software that improves the efficiency of decision-making in the context of weather-related risks for power lines.

The graduation work may be divided into several logically connected parts: literature review, system design and implementation, and testing.

The first chapter provides a rationale for the development of weather monitoring software, analyzes existing solutions, and defines the main tasks to be addressed during the work.

The second chapter presents the system design, including the development of the logical data model, software architecture, and user interface.

The third chapter describes the process of microservices development: selection of technologies and tools, module description, and testing stages.

In conclusion, the work demonstrates the practical applicability of intelligent microservices for power line weather monitoring and highlights opportunities for further development and integration.

Оглавление

Введение

В современных условиях развитие информационных технологий играет ключевую роль в решении многих задач, связанных с повышением эффективности различных отраслей экономики. Одной из таких областей является энергетика, где прогнозирование и предотвращение аварийных ситуаций на линиях электропередач (ЛЭП) имеет особую важность.

Одним из наиболее критичных факторов, влияющих на надежность работы ЛЭП, является образование гололеда. Своевременное выявление опасных условий и прогнозирование их развития позволяет минимизировать риски обрушения ЛЭП, сократить время их восстановления и снизить финансовые потери компаний.

Объектом исследования является процесс автоматизированного мониторинга и прогнозирования обледенения на линиях электропередач.

Предметом исследования является программное обеспечение, предназначенное для обработки метеоданных, прогнозирования образования гололеда и уведомления диспетчеров.

Целью данной работы является разработка программного обеспечения для метеомониторинга линий электропередач с использованием искусственного интеллекта.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- выполнить постановку задачи по разработке программного обеспечения для прогнозирования обледенения на ЛЭП;
- спроектировать программное обеспечение, включающее логическую и концептуальную модель базы данных, алгоритмы обработки данных и интерфейс пользователя;
- реализовать все микросервисы и сбор данных, их обработку и визуализацию;
- провести тестирование программного обеспечения с использованием

контрольных примеров и реальных метеоданных.

Методы исследования включают использование технологий проектирования и разработки программного обеспечения, анализа данных и методов расчета экономической эффективности.

Практическая значимость работы заключается в том, что разработанное программное обеспечение позволяет энергетическим компаниям минимизировать затраты на обслуживание ЛЭП и снизить вероятность аварийных ситуаций.

Работа состоит из введения, трех глав, заключения и списка используемой литературы.

Во введении обоснована актуальность темы работы, определены объект, предмет, цель и задачи исследования.

В первой главе представлена постановка задачи, описаны текущие проблемы мониторинга ЛЭП и обоснование разработки программного обеспечения.

Во второй главе выполнено проектирование программного обеспечения. Разработаны концептуальная модель данных, алгоритмы обработки метеоданных, структура пользовательского интерфейса и схема интеграции с существующими системами.

В третьей главе описана реализация программного обеспечения, включая выбор технологий, описание основных модулей ПО, тестирование и оценку работы программы.

В заключении описываются результаты выполнения выпускной квалификационной работы.

Бакалаврская работа содержит 67 страниц текста, 37 рисунков, 4 таблицы и 29 источников. В приложении представлены фрагменты кода.

Глава 1 Постановка задачи на разработку программного обеспечения информации

1.1 Функциональные и архитектурные особенности программного обеспечения для метеомониторинга линий электропередач

В современном мире информационные технологии становятся неотъемлемой частью обеспечения надежности и безопасности инфраструктурных объектов, таких как линии электропередач [1][6]. Разработка специализированного программного обеспечения для метеомониторинга направлена на снижение риска аварий, связанных с обледенением проводов, и обеспечение бесперебойной работы энергосистемы [5][17]. Рассмотрим ключевые функциональные и архитектурные особенности, которые характеризуют успешное программное обеспечение для данной сферы [6][7][15].

Пользовательский опыт является важным аспектом разработки программного обеспечения для метеомониторинга [18]. Веб-приложения такого типа должны быть удобны и интуитивно понятны для диспетчеров и операторов [9][10]. Реализация интерактивного интерфейса, включающего визуализацию данных в виде сортированных таблиц по убыванию вероятности обледенения, позволяет оперативно оценивать текущую ситуацию и принимать обоснованные решения [6][15].

Полнота и актуальность представляемой информации играют важную роль [4][5]. Программное обеспечение должно обрабатывать данные о погодных условиях, таких как температура, влажность, скорость ветра, осадки и т.д. [4][5]. С использованием современных алгоритмов машинного обучения для прогнозирования вероятности обледенения [7][15][20]. Точность предоставляемых прогнозов зависит от надежности используемых источников данных и эффективности методов их обработки [7][15].

Визуальный дизайн программного обеспечения способствует

повышению удобства работы пользователей [9][10]. Система должна предоставлять четкую и структурированную информацию, а также использовать графические элементы, которые облегчают восприятие сложных данных [9]. Это повышает эффективность работы диспетчеров и минимизирует риск ошибок [18].

С увеличением объема данных и необходимости обработки информации в режиме реального времени архитектура системы должна быть адаптивной и масштабируемой [2][11]. Микросервисный подход позволяет разделить функционал системы на отдельные модули, отвечающие за сбор, обработку, прогнозирование и визуализацию данных [2][11]. Это обеспечивает гибкость при внедрении новых функций и повышает устойчивость к сбоям [11][12].

Приложение метеомониторинга должно быть оптимизировано для работы на различных устройствах, включая компьютеры, планшеты и мобильные устройства [9][10]. Адаптивный дизайн пользовательского интерфейса гарантирует доступность информации независимо от используемой платформы, что особенно важно для диспетчеров, работающих в полевых условиях [9][10].

Защита данных и безопасность системы являются критически важными аспектами [11][12]. Использование современных протоколов аутентификации и авторизации, таких как JWT-токены, обеспечивает надежную защиту от несанкционированного доступа [11]. Кроме того, регулярное резервное копирование данных и внедрение патчей безопасности поддерживают стабильность и надежность системы [11][12].

Поддержка пользователей включает такие функции, как консультации по действиям при выявлении рисков и инструкции по использованию системы [18]. Это позволяет повысить уровень взаимодействия с системой и укрепить доверие пользователей [18].

Эффективность программного обеспечения зависит от регулярного анализа данных и совершенствования алгоритмов прогнозирования [7][15]. Использование аналитических инструментов для мониторинга работы

системы, анализа пользовательской активности и оценки точности прогнозов позволяет выявлять слабые стороны и внедрять улучшения [6][7][15].

Поддержание успешной работы приложения требует постоянного внимания к обновлению контента, оптимизации производительности и соблюдению отраслевых стандартов [6][18]. Регулярные обновления программного обеспечения, исправление ошибок и адаптация к изменениям в метеорологических данных гарантируют его высокую надежность и эффективность [6][7][15].

1.2 Разработка требований к программному обеспечению для метеомониторинга линий электропередач

При разработке программного обеспечения для метеомониторинга линий электропередач важно сформировать четкие требования, которые соответствуют техническим задачам и бизнес-целям, а также удовлетворяют потребности конечных пользователей – диспетчеров и операторов энергетических компаний [6][7][18].

Для формулировки требований используется методология FURPS+, которая классифицирует требования к программным системам [18]. В данной методологии выделяются пять основных категорий:

- функциональные требования (Functionality) описывают основные функции системы, обеспечивающие выполнение задач пользователей. Для ПО метеомониторинга это сбор метеоданных [4][5], прогнозирование вероятности обледенения [7][15][20], визуализация данных [9][10];
- удобство использования (Usability) определяет пользовательский интерфейс, его структуру и внешний вид [9][10]. Система должна быть интуитивно понятной, адаптивной и удобной для работы на различных устройствах [9][10], обеспечивая доступ к информации в несколько кликов [6][18];

– надежность (Reliability) обеспечивает стабильность работы системы и быстрое восстановление в случае сбоев [11][12]. Программное обеспечение должно функционировать круглосуточно, минимизируя время простоя и ошибок [7][11];
– производительность (Performance) включает время отклика системы, скорость обработки и обновления данных [2][11]. Обновление данных должно происходить с интервалом не более 2 часов [6][7][15];
– поддерживаемость (Supportability) охватывает возможности по расширению функционала, адаптации под новые требования и оперативному устранению проблем [2][11][12].

В таблице 1 представлены требования к программному обеспечению для метеомониторинга линий электропередач в соответствии с методологией FURPS+ [18].

Таблица 1 – Требования к программному обеспечению для метеомониторинга линий электропередач в методологии FURPS+

Требование	Статус	Полезность	Риск	Стабильность
Функциональные требования				
Сбор метеоданных каждые 2 часа	Обязательное	Критическая	Средний	Высокая
Прогнозирование обледенения	Обязательное	Критическая	Высокий	Высокая
Визуализация зон риска	Обязательное	Критическая	Средний	Высокая
Требования к юзабилити				
Интуитивный интерфейс	Обязательное	Критическая	Средний	Средняя
Адаптивность под мобильные устройства	Обязательное	Важная	Средний	Средняя
Требования к надежности				
Среднее время восстановления системы: до 5 минут	Обязательное	Критическая	Средний	Средняя
Доступность системы: 24/7	Обязательное	Критическая	Средний	Высокая

Продолжение таблицы 1

Требование	Статус	Полезность	Риск	Стабильность
Требования к производительности				
Время отклика интерфейса: до 5 секунд	Желательное	Средняя	Низкий	Средний
Требования к поддерживаемости				
Время устранения критических ошибок: до 1 часа	Обязательное	Критическая	Высокий	Средняя

Сформулированные требования служат основой для проектирования, разработки и последующего тестирования программного обеспечения для метеомониторинга [6].

1.3 Анализ аналогов программного обеспечения для метеомониторинга линий электропередач

В условиях роста частоты обледенений воздушных линий электропередач (ВЛЭП), вызванных изменением климатических условий [5] и увеличением стоимости строительства, обслуживания и ремонта ВЛЭП [1], вопрос о мониторинге нарастания ледяной нагрузки на проводах становится особенно актуальным [6][17]. Так, в период с 2023 по 2024 год было проведено более 340 операций по плавке и механической очистке гололедно-изморозевых отложений на линиях электропередачи напряжением 6-500 кВ, расположенных в Поволжье, на Северном Кавказе и в других регионах [1][17].

Основное отличие разрабатываемого программного обеспечения от других систем мониторинга гололедообразования заключается в использовании исключительно метеоданных, получаемых от сторонних сервисов [4][5], и ретроспективных данных об обледенении на контролируемых линиях [15][20]. В то время как другие системы часто

опираются на данные физических датчиков [19][23], наше решение ориентировано на прогнозирование с использованием искусственного интеллекта [7][15], что значительно сокращает затраты на установку и обслуживание оборудования [1][6].

Одним из аналогов является программное обеспечение от компании ООО «НТЦ Инструмент-микро» [19]. Однако их решение основано на применении датчиков [19][23], что ограничивает гибкость масштабирования и повышает стоимость эксплуатации [1][6].

Данные метеомониторинга представлены в двух основных видах: на карте и в оперативной таблице [9][10]. Эти представления позволяют диспетчерам эффективно анализировать текущие метеоусловия [4][5] и оперативно реагировать на риски [6][18].

На карте посты телеметрии отображаются графическими элементами [9][10]: круглыми маркерами, которые показывают местоположение и текущее состояние поста, текстовыми метками температуры окружающей среды и номера опоры, а также графическими обозначениями направления ветра. На рисунке 1 представлена визуализация постов на карте, где маркеры позволяют быстро оценить текущее состояние линии [9][10][18].

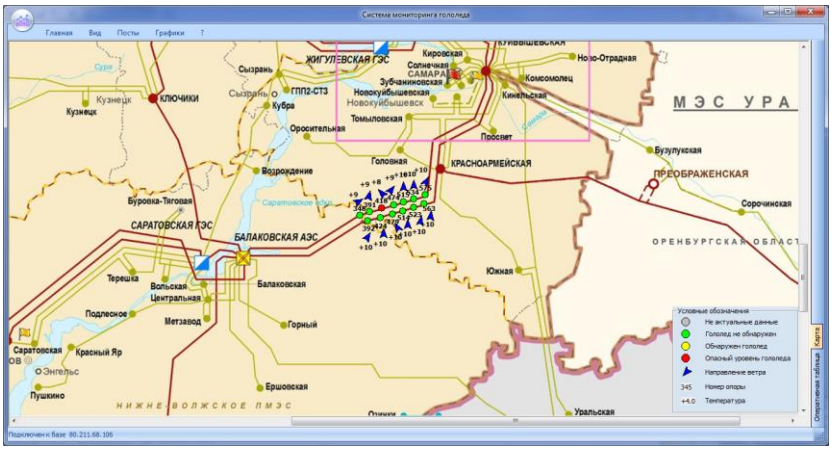

Рисунок 1 – Представление в виде карты

На рисунке 2 представлена карта телеметрии в виде таблицы [9][10].

Данный формат обеспечивает более детализированное и компактное представление информации о погодных условиях [4][5] и параметрах состояния линий электропередачи [6]. Таблица включает ключевые метеопоказатели (температура, влажность, скорость и направление ветра) [4][5], а также идентификационные данные опор и участков ВЛЭП [6][17]. Такое отображение позволяет диспетчерам быстро анализировать данные с различных постов одновременно [9][10], выявлять аномальные значения [7][15] и своевременно принимать решения без необходимости переключения между различными визуальными режимами [9][10][18]. Подобная структурированная форма особенно удобна при работе с большим числом объектов мониторинга [9][10][18].

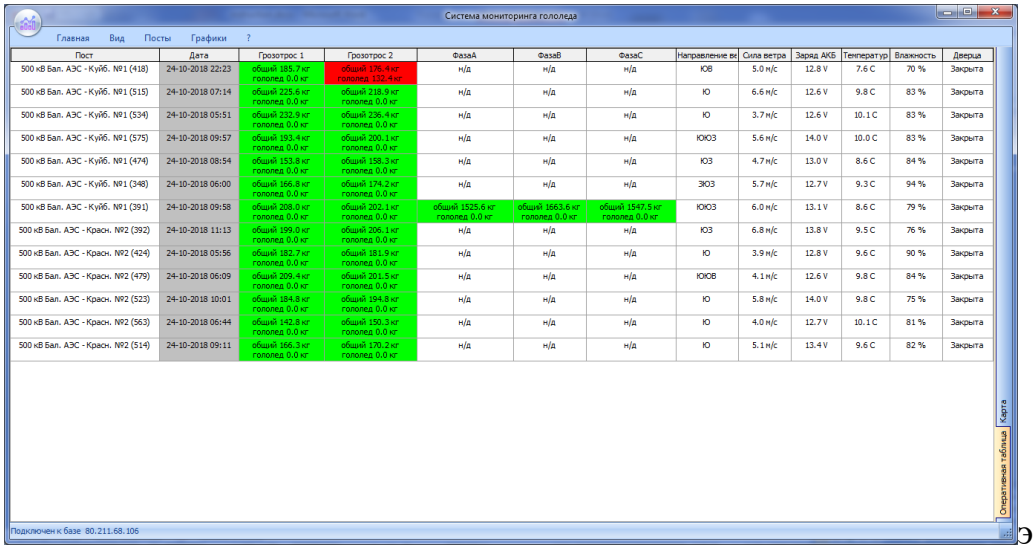

Рисунок 2 – Представление в виде оперативной таблицы

На рисунке 3 представлены условные обозначения, используемые на карте телеметрии для визуализации состояния постов мониторинга [9][10][18]. Цветовая индикация маркеров позволяет оперативно оценивать уровень гололедообразования на различных участках линий электропередачи [6][15]: зеленый цвет указывает на нормальные условия, желтый – на начало формирования ледяных отложений, а красный сигнализирует о критическом

уровне обледенения, требующем немедленного реагирования [6][17][18]. Дополнительные элементы, такие как стрелки направления ветра и текстовые метки температуры и номера опор, обеспечивают комплексное представление метеообстановки. Такой подход к визуализации повышает информативность интерфейса и ускоряет принятие решений в условиях ограниченного времени.

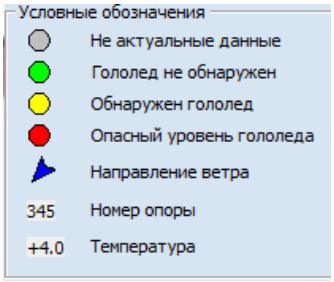

Рисунок 3 – Условные обозначения на карте телеметрии

В оперативной таблице данные представлены в структурированном табличном формате, что обеспечивает удобный доступ к ключевым параметрам состояния воздушных линий электропередачи. Для каждого контролируемого участка указывается общий вес провода, расчетный вес ледяных отложений, а также вспомогательные данные, такие как температура воздуха, скорость и направление ветра. В случаях отсутствия достоверных данных выводится обозначение «н/д», что позволяет диспетчеру учитывать степень полноты информации при анализе. Такой способ представления облегчает восприятие, способствует ускоренному принятию решений и позволяет отслеживать динамику изменения параметров в реальном времени.

На рисунке 4 представлено отображение данных в оперативной таблице.

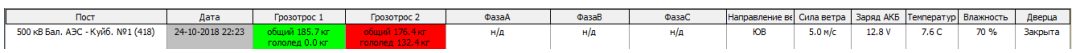

Рисунок 4 – Отображение данных в оперативной таблице

Переключение между различными форматами представления данных, картой и таблицей осуществляется через вкладку «Вид» или с помощью соответствующих кнопок, расположенных в интерфейсе программы. Это позволяет пользователю выбрать наиболее удобный способ визуализации в зависимости от текущих задач.

На рисунке 5 изображены кнопки переключения вида во вкладке “Вид”.

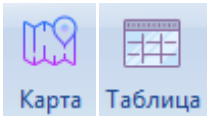

Рисунок 5 – Кнопки переключения вида во вкладке «Вид»

Альтернативный способ смены представления данных реализован через вкладки, расположенные в нижнем углу главного окна программы. Пользователь может переключаться между картой и таблицей, выбирая соответствующую вкладку. Такой интерфейс повышает удобство навигации и позволяет быстро адаптироваться к нужному формату отображения информации без лишних действий.

При наведении курсора на маркер, обозначающий пост телеметрии на карте, пользователь может вызвать всплывающее окно, содержащее подробную информацию о текущих показаниях. В окне отображаются значения температуры, направления и скорости ветра, уровня обледенения, а также другая метеоинформация, необходимая для оперативного анализа ситуации. Это решение обеспечивает быстрый доступ к детализированным данным без необходимости переключения между представлениями.

На рисунке 6 представлено всплывающее окно с показателями.

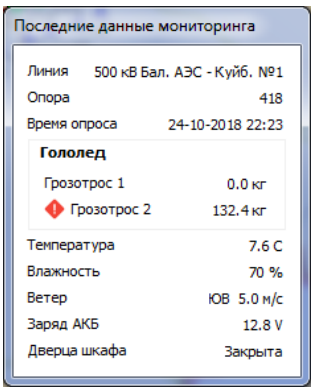

Рисунок 6 – Всплывающее окно с показаниями

Программное обеспечение предоставляет возможность анализа архивных данных для каждого поста телеметрии. Для этого предусмотрено отдельное окно, доступное через контекстное меню карты или таблицы, а также через вкладку «Графики». По умолчанию отображаются данные за текущие сутки в табличной форме, однако пользователь может выбрать произвольный временной интервал для анализа.

На рисунке 7 представлено окно отображения архивных данных

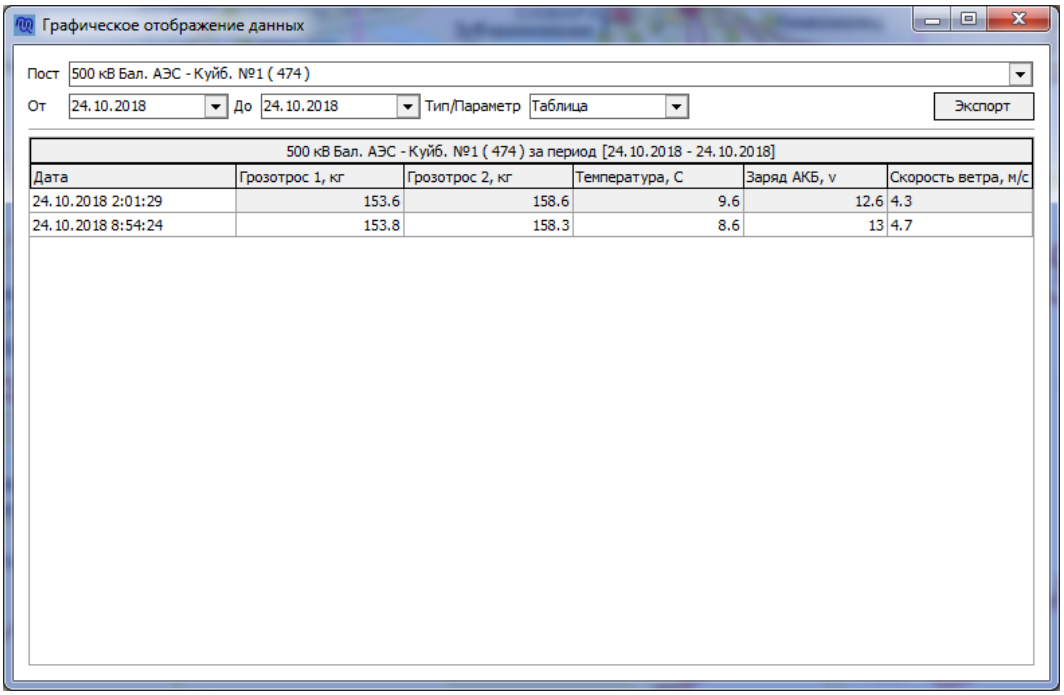

Рисунок 7 – Окно для отображения архивных данных

Для удобства визуального анализа изменений метеопараметров и состояния линий электропередач реализована возможность графического отображения данных. Пользователь может открыть соответствующее окно через интерфейс программы, выбрав специальную функцию “График”. Представлено на рисунке 8.

График отображает временную шкалу (по горизонтали) и шкалу значений параметров (по вертикали). На графике представлены три порога: порог обнаружения, ниже которого гололед отсутствует; порог тревоги, превышение которого указывает на наличие гололеда; и порог аварии, сигнализирующий об опасном уровне обледенения. Настройки отображения доступны через легенду графика.

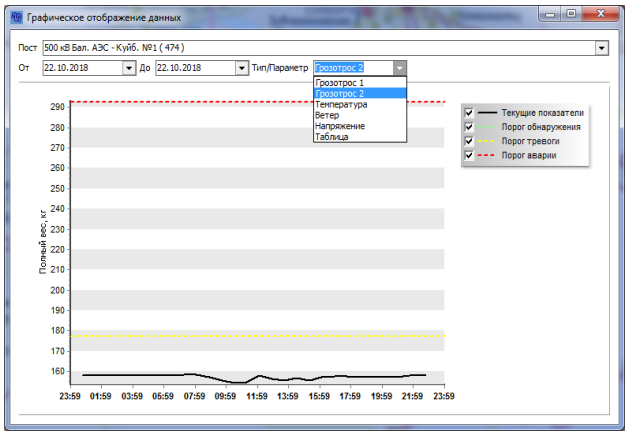

Рисунок 8 – Отображение архивных данных в виде графика

Интерфейс программного обеспечения спроектирован таким образом, чтобы обеспечить интуитивную навигацию и удобный доступ к основным функциям. Главное меню организовано в виде вкладок, каждая из которых отвечает за определённый набор действий: отображение карты, таблицы, графиков, настройки и справочная информация.

На рисунке 9 представлено меню программы, оформленное в виде вкладок, что позволяет пользователю быстро переключаться между

различными режимами работы.

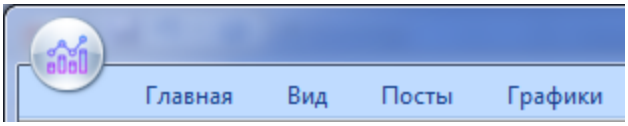

Рисунок 9 – Меню программы

Каждая вкладка меню программы выполняет определённую функцию и предназначена для эффективного взаимодействия пользователя с системой:

– главная – отвечает за настройку подключения к базе данных и выбор региона, по которому будет отображаться информация. Это первичная точка входа, с которой начинается работа пользователя. На рисунке 10 представлена вкладка “Главная”, содержащая элементы конфигурации подключения и выбора области мониторинга.

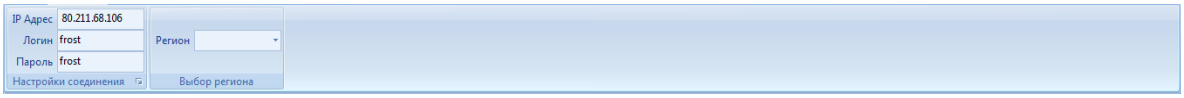

Рисунок 10 – Вкладка “Главная”

– вид – обеспечивает переключение между различными способами отображения информации (карта или таблица), масштабирование интерфейса, а также настройку отображения условных обозначений. На рисунке 11 показана вкладка “Вид”, с помощью которой можно настроить визуальное представление данных на экране.

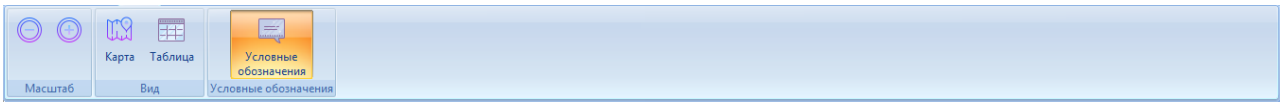

Рисунок 11 – Вкладка “Вид”

– посты – предназначена для управления отображением графических элементов телеметрических постов, их позиционирования и конфигурации. Также здесь можно включить или отключить звуковые уведомления о критических событиях. На рисунке 12 представлена вкладка “Посты”, позволяющая редактировать параметры каждого поста и управлять его индикацией и сигнализацией.

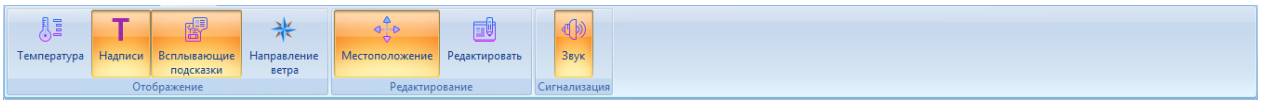

Рисунок 12 – Вкладка “Посты”

Позволяет настроить отображение графических элементов, составляющих индикацию поста. А также редактировать местоположение поста, и его настройки. И включать /отключать звуковую сигнализацию событий обнаружения гололеда.

– график – предоставляет доступ к архивным данным по каждому посту в графической или табличной форме, обеспечивая ретроспективный анализ погодных условий и состояния линии.
На рисунке 13 представлена вкладка “Графики”, открывающая окно для просмотра исторических данных.

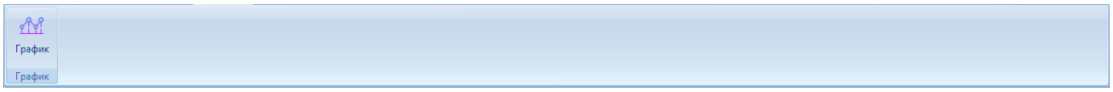

Рисунок 13 – Вкладка “Графики”

Редактирование графических элементов в системе реализовано таким образом, чтобы пользователи могли гибко настраивать отображение данных и адаптировать интерфейс под свои потребности. Функция «Местоположение»

позволяет изменять расположение маркера поста на карте простым перетаскиванием. Вместе с маркером автоматически перемещаются все связанные графические элементы, такие как текстовые и графические обозначения. Это особенно удобно для визуализации данных на карте, так как пользователю не нужно отдельно перемещать каждый объект. Если требуется переместить только один элемент, например, текстовую метку или графический индикатор, можно воспользоваться клавишей SHIFT, которая отключает привязку остальных элементов к маркеру.

Более сложные настройки постов выполняются через окно «Редактор постов». Это окно предоставляет доступ к широкому набору параметров, включая изменение характеристик опоры, настройку интервалов опроса данных, а также задание пороговых значений и сухого веса провода. Окно редактора поста, в котором отображаются все доступные параметры для редактирования, представлено на рисунке 14.

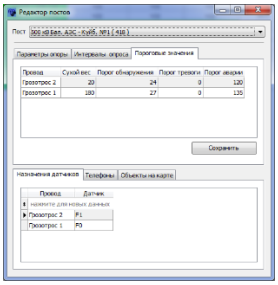

Рисунок 14 – Окно редактора поста

Одной из ключевых функций редактора является возможность изменения параметров опоры, таких как её идентификатор, местоположение и дополнительные характеристики. Пользователь может внести изменения через соответствующий интерфейс, пример которого представлен на рисунке 15.

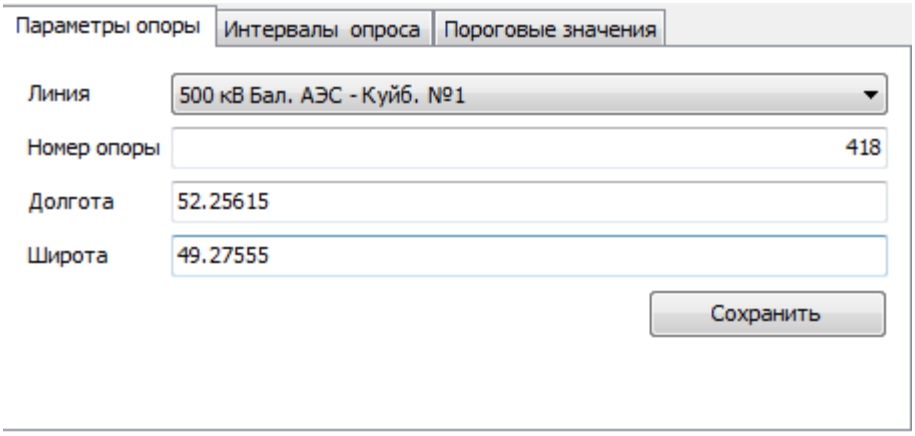

Рисунок 15 – Редактирование параметров опоры

Также доступна настройка интервалов опроса, что позволяет гибко задавать частоту обновления данных для каждой опоры. Это особенно важно для оптимизации работы системы в условиях изменяющихся погодных условий. Настройки интервалов отображены на рисунке 16.

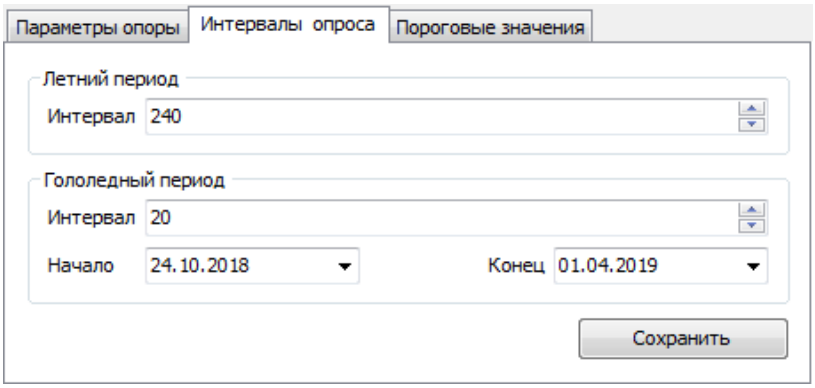

Рисунок 16 – Редактирование интервалов опроса

Кроме того, редактор постов предоставляет возможность задавать пороговые значения для обнаружения, тревоги и аварий, а также изменять сухой вес провода. Это позволяет учитывать индивидуальные характеристики каждой линии электропередач, что значительно повышает точность прогнозов. Пример интерфейса для редактирования пороговых значений и сухого веса провода приведён на рисунке 17.

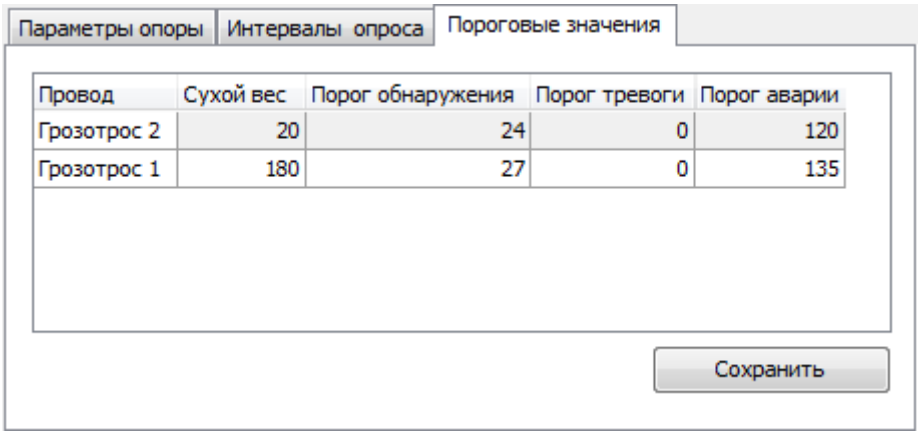

Рисунок 17 – Редактирование пороговых значений и сухой вес провода

В таблице 2 показан сравнительный анализ рассмотренных аналогов ИСУ с оценкой их характеристик по шкале от 0 до 3, где 0 – не соответствует разработанным требованиям, 3 – полностью соответствует.

Таблица 2 – Сравнительный анализ аналогов

Система/ Характеристика	Точность прогноза	Без датчиков	Стоимость внедрения	Адаптация под участки ЛЭП	Баллы (из 12)
SEL Weather Monitoring	3	0	0	2	5
Vaisala Tools	3	0	1	2	6
Гидрометцентр + SCADA	1	3	3	1	8
Разрабатываемое ПО	3	3	3	3	12

Проведённый анализ существующих решений в сфере прогнозирования обледенения ЛЭП показал, что, несмотря на высокую точность и надёжность некоторых зарубежных систем, они требуют установки дорогостоящих физических датчиков и имеют высокую стоимость внедрения и поддержки. Такие решения, как SEL и Vaisala, ориентированы на крупные предприятия с высоким бюджетом и не всегда подходят для широкого применения на всей

сети ЛЭП.

Отечественные комбинации, использование данных Гидрометцентра в связке с локальными SCADA-системами, напротив, не требуют дополнительных устройств, но при этом имеют низкую точность и не позволяют проводить точечный анализ по конкретным участкам линий электропередач.

Разработанное программное обеспечение выгодно отличается от аналогов: оно сочетает в себе высокую точность прогноза, не требует установки оборудования на линии, обладает высокой масштабируемостью и полностью адаптируется под конкретную инфраструктуру заказчика. Таким образом, по совокупности параметров оно обеспечивает лучший баланс между функциональностью, экономичностью и удобством внедрения.

На основании анализа можно сделать вывод о целесообразности разработки и внедрения собственного программного комплекса, который отвечает современным требованиям, снижает затраты на мониторинг ЛЭП и обеспечивает эффективное принятие решений на основе данных.

Выводы по главе 1

В первой главе были рассмотрены функциональные и архитектурные особенности программного обеспечения для метеомониторинга линий электропередач [6][7][11], проведён анализ аналогичных решений [19][23] и сформулированы требования к разрабатываемому программному обеспечению [18]. На основании анализа сделан вывод, что наше программное обеспечение должно иметь ряд преимуществ перед существующими аналогами, главным из которых является работа без использования физических датчиков на линиях [1][6]. Это значительно снижает расходы на установку и обслуживание системы, делая решение более доступным и актуальным [1][6][17].

Для достижения этой цели программное обеспечение для метеомониторинга должно соответствовать следующим требованиям

[6][7][18]:

- система должна собирать метеоданные каждые 2 часа [4][5][6]. Это обязательное требование с критической полезностью, средним риском и высокой стабильностью [6][11];
- программное обеспечение должно прогнозировать обледенение [7][15][20]. Это также обязательное требование с критической полезностью, высоким риском и высокой стабильностью [6][7][15];
- система должна визуализировать зоны риска [9][10][18]. Это обязательное требование с критической полезностью, средним риском и высокой стабильностью [6][9][18];
- интерфейс системы должен быть интуитивно понятным [9][10][18]. Это обязательное требование с критической полезностью, средним риском и средней стабильностью [9][10][18];
- интерфейс должен адаптироваться под мобильные устройства [9][10]. Это обязательное требование с важной полезностью, средним риском и средней стабильностью [9][10][18];
- среднее время восстановления системы должно составлять до 5 минут [11][12]. Это обязательное требование с критической полезностью, средним риском и средней стабильностью [11][12][18];
- доступность системы должна быть круглосуточной, 24/7 [11][12]. Это обязательное требование с критической полезностью, средним риском и высокой стабильностью [6][11][12];

Таким образом, сформулированные требования охватывают ключевые аспекты функциональности [6][7], доступности [11][12] и надёжности [6][11] программного обеспечения. Они обеспечивают основу для последующей архитектурной реализации системы, ориентированной на эффективную работу в реальных условиях эксплуатации энергетических объектов [1][6][17].

Глава 2 Проектирование программного обеспечения веб-приложения метеомониторинга линий электропередач

2.1 Логическое моделирование нейросети

На рисунке 18 представлена диаграмма архитектуры нейросети, используемой в веб-приложении метеомониторинга линий электропередач. Архитектура организована в виде отдельного сервиса – training_service, реализованного с использованием FastAPI [9]. Данный сервис предоставляет REST API для запуска обучения и получения предсказаний с помощью различных моделей машинного обучения.

В состав архитектуры входят три основных метода:

- полносвязная нейронная сеть для классификации (DenseNet), предназначенная для оценки риска гололедообразования на основе текущих и ретроспективных метеоданных [7, 15, 27];
- алгоритм градиентного бустинга (XGBoost), альтернативный метод для задачи бинарной классификации, используемый для повышения устойчивости модели и проведения сравнительного анализа качества [20, 27];
- полносвязная нейронная сеть для регрессии, применяемая для предсказания количественного прироста льда на проводах в зависимости от погодных условий.

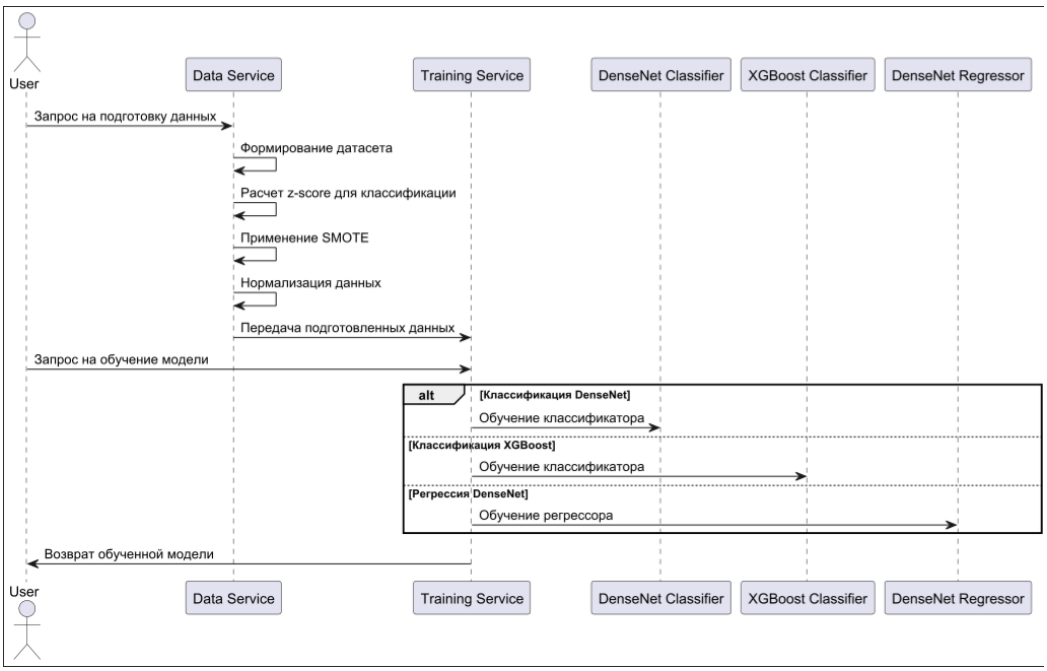

Рисунок 18 – Диаграмма архитектуры нейросети для приложения метеомониторинга

Ключевые особенности логической модели нейросети:

- модульность: разделение на сервисы подготовки данных и обучения позволяет гибко масштабировать систему;
- многозадачность: система способна решать задачи как классификации (выявление аномалий), так и регрессии (прогнозирование прироста льда);
- предобработка данных: включает расчет z-score для классификации, применение методов балансировки классов (Borderline SMOTE) и нормализацию данных;
- гибкость в выборе алгоритмов: возможность использования как нейросетевых подходов (DenseNet), так и классических алгоритмов машинного обучения (XGBoost);
- API-ориентированная архитектура: использование FastAPI обеспечивает удобный интерфейс для взаимодействия с сервисом обучения.

Такая архитектура позволяет эффективно решать задачи прогнозирования обледенения линий электропередач, обеспечивая при этом гибкость и масштабируемость системы.

2.2 Логическое моделирование веб-приложения для метеомониторинга линий электропередач

Для концептуального моделирования используется унифицированный язык моделирования UML, который предоставляет единый визуальный инструмент для проектирования ПО [22]. Диаграмма прецедентов является ключевым инструментом UML, предназначенным для описания вариантов использования системы и взаимодействия с внешними сущностями.

В веб-приложении для метеомониторинга линий электропередач можно выделить следующие основные акторы:

- диспетчеры энергетических компаний, которые используют систему для анализа вероятности обледенения линий электропередач и принятия решений по предотвращению аварий [1, 17];
- источники метеорологических данных, предоставляющие информацию о погодных условиях [4, 5].

Основное взаимодействие диспетчеров с веб-приложением заключается в просмотре текущих данных о вероятности обледенения в удобной табличной форме. Таблица отображает вероятности для всех участков линий электропередач. На основе этой информации диспетчеры могут оперативно принимать решения о необходимости проведения профилактических мероприятий [6, 17].

На фронтенде веб-приложения реализована единая веб-страница, где представлены данные в виде таблицы. В таблице отображаются:

- название участка и номер опоры;
- вероятность обледенения;
- время предсказания.

Для удобства использования интерфейс минималистичен, что позволяет диспетчерам быстро находить необходимую информацию без лишних действий [25, 26].

Метеорологические данные передаются через API и обрабатываются серверной частью веб-приложения [12]. Приложение обновляет таблицу каждые два часа, используя данные, проанализированные обученной нейросетью [7, 15], и предоставляет актуальные прогнозы.

Диаграмма вариантов использования веб-приложения метеомониторинга, представленная на рисунке 19 отображает основные сценарии использования [22]. Администраторы добавляют компании и их пользователей в систему, а диспетчеры получают информацию о текущей ситуации на линиях электропередач и принимают соответствующие меры при выявлении риска [1, 17, 28].

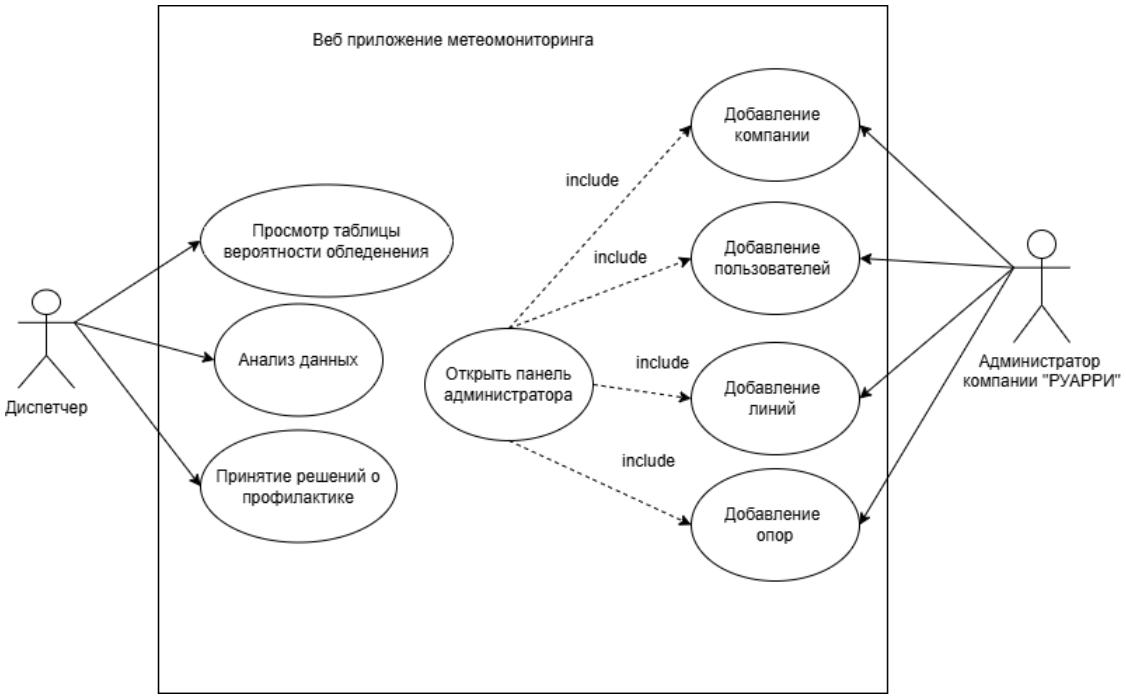

Рисунок 19 – Диаграмма вариантов использования приложения

Диаграмма классов является важным элементом языка моделирования UML (Unified Modeling Language) [22]. Она используется для отображения

статической структуры системы, включая её классы, их атрибуты, методы и взаимосвязи между объектами [3, 29]. Такие диаграммы помогают разработчикам и архитекторам лучше понять, как различные компоненты системы взаимодействуют друг с другом [2, 12], и выступают в качестве основы для дальнейшей реализации программного кода [22].

На основе требований и функций веб-приложения метеомониторинга, я выделил основные сущности и их взаимосвязи, которые представлены на рисунке 20.

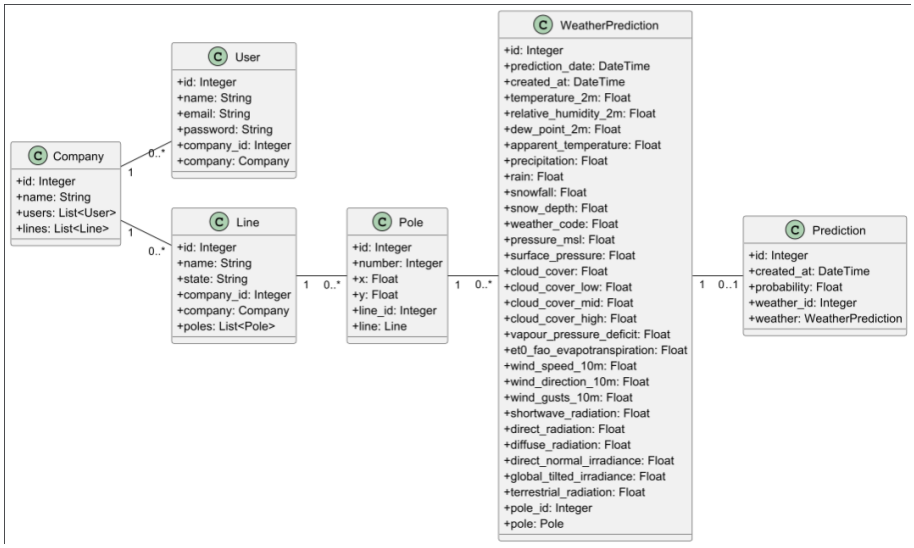

Рисунок 20 – Диаграмма классов

В таблице 3 представлена спецификация диаграммы классов, отражающая основные сущности программного обеспечения.

Таблица 3 – Спецификация диаграммы классов

Класс	Описание
Company	Класс объектов, моделирующих на логическом уровне энергетические компании
User	Класс объектов, моделирующих на логическом уровне пользователей системы (диспетчеров)
Line	Класс объектов, моделирующих на логическом уровне линии электропередач

Продолжение таблицы 3

Класс	Описание
Pole	Класс объектов, моделирующих на логическом уровне опоры линий электропередач
WeatherPrediction	Класс объектов, моделирующих на логическом уровне прогнозы погоды для конкретных опор
Prediction	Класс объектов, моделирующих на логическом уровне прогнозы вероятности обледенения

Диаграмма классов является важным инструментом для проектирования и разработки веб-приложения метеомониторинга. Она помогает визуализировать структуру системы, определить основные сущности и их взаимодействия, а также служит основой для написания кода. В результате, разработка веб-приложения становится более структурированной и понятной, что способствует созданию качественного продукта.

2.3 Логическое моделирование данных

В современных информационных системах существует два принципиально различных подхода к организации хранения данных - реляционные и нереляционные базы данных [8, 24]. Каждый из этих подходов имеет свои уникальные характеристики, преимущества и ограничения, которые необходимо тщательно учитывать при проектировании архитектуры программного обеспечения [6, 22].

Реляционные базы данных представляют собой классический подход к структурированному хранению информации [8]. Они основаны на математической модели реляционной алгебры и используют язык структурированных запросов SQL [24]. Основной принцип организации данных в таких системах - представление информации в виде взаимосвязанных таблиц, где каждая таблица описывает определенную

сущность, а связи между таблицами устанавливаются через первичные и внешние ключи [8].

Ключевым преимуществом реляционных баз данных является строгая структурированность и возможность установления сложных логических связей между различными сущностями [8, 24]. Это особенно важно для систем, где требуется высокая целостность и согласованность данных [6]. Транзакционная модель ACID гарантирует, что любые изменения в базе данных будут выполнены корректно и не приведут к нарушению целостности информации [8].

Не реляционные базы данных (NoSQL) возникли как альтернатива традиционным реляционным системам для решения задач, где классические подходы демонстрируют низкую эффективность [11]. Они принципиально по-другому подходят к вопросам хранения и обработки информации, предлагая гибкие схемы данных и горизонтальную масштабируемость [11].

NoSQL базы данных существуют в нескольких основных парадигмах: документные хранилища, key-value системы, графовые базы данных и колоночные базы [11]. Каждая из этих парадигм оптимизирована под решение специфических задач и имеет собственные механизмы оптимизации производительности [11].

Для метеомониторинга линий электропередач мы приняли решение использовать гибридный подход, объединяющий преимущества реляционной (PostgreSQL) и нереляционной (Redis) баз данных [8, 11]. Этот подход позволяет создать масштабируемое и эффективное приложение, используя Redis в качестве высокопроизводительного кэша и PostgreSQL для обработки транзакционных данных [11, 23].

PostgreSQL будет использоваться для хранения структурированной информации о компаниях, пользователях, линиях электропередач и опорах [8, 24]. Реляционная модель позволит нам поддерживать целостность данных, устанавливать сложные связи между сущностями и выполнять аналитические запросы [8]. PostgreSQL обеспечит надежное хранение и обработку

транзакций, гарантируя согласованность данных [24].

Redis выступит в роли высокопроизводительного хранилища для оперативного хранения прогнозов обледенения [11]. Использование in-memory базы данных типа ключ-значение обеспечит чрезвычайно быстрый доступ к актуальным метеорологическим прогнозам, что критически важно для систем мониторинга в режиме реального времени [11, 23].

Ключевое преимущество такого подхода – возможность сочетать надежность классических реляционных баз данных с высокой производительностью NoSQL решений [8, 13]. PostgreSQL гарантирует сохранение и согласованность базовой информации об инфраструктуре [24], а Redis обеспечивает мгновенный доступ к оперативным прогнозам [11].

Формат ключей в Redis будет максимально информативным: prediction:{номер линии}:{номер пролета}:{дата-время} [11]. Это позволит не только быстро получать актуальные прогнозы, но и выполнять эффективную выборку по различным временным диапазонам [16].

Такая архитектура не только решает текущие задачи системы метеомониторинга, но и создает гибкую платформу для будущего развития и масштабирования проекта. Использование облачных сервисов, таких как OCI, может дополнительно упростить управление, повысить безопасность и улучшить производительность системы.

2.4 Разработка внутренней структуры

Разработка веб-приложения для метеомониторинга линий электропередач начинается с создания структуры, представляющей собой логическую схему расположения элементов на странице [10, 21]. Хорошо продуманная структура позволяет избежать ошибок на последующих этапах разработки, что может существенно сократить время и затраты на исправление [22].

Требования к структуре веб-приложения могут варьироваться в

зависимости от решаемых бизнес-задач [1, 17]. В нашем случае приложение будет иметь одну веб-страницу, на которой будет представлена таблица с прогнозами обледенения [10]. Несмотря на простоту, структура должна учитывать потребности пользователей и обеспечивать удобный доступ к информации [25].

При создании структуры веб-приложения необходимо учитывать следующие аспекты:

- целевая аудитория: понимание потребностей пользователей, их возрастной категории и интересов поможет создать интуитивно понятный интерфейс;
- удобство навигации: структура должна облегчать поиск информации. Навигация должна быть интуитивной, а элементы интерфейса должны иметь логическую связь. Это гарантирует, что пользователи смогут быстро находить необходимую информацию и не покинут приложение из-за его непонятности.

Исходя из этих требований, мы выбрали иерархическую структуру для нашего веб-приложения [10, 14]. На главной странице будет размещена таблица прогнозов обледенения, которая будет содержать актуальные данные о вероятности обледенения для различных участков линий электропередач [15, 18].

Когда пользователь открывает приложение, он сразу попадает на главную страницу, где представлена таблица с прогнозами. Таблица включает следующие столбцы:

- название участка или номер опоры;
- вероятность обледенения;
- время прогноза.

Пользователь сможет легко просматривать данные и находить необходимую информацию благодаря простому и интуитивно понятному интерфейсу [25]. Также предусмотрены функции фильтрации и сортировки данных в таблице [10], что позволит пользователям быстро находить

актуальные прогнозы по определенным критериям [27].

Структура нашего веб-приложения является простой и удобной для пользователей [25], что является важным фактором для обеспечения положительного пользовательского опыта [21]. Это, в свою очередь, увеличит вероятность того, что пользователи захотят вернуться к приложению для получения актуальной информации о состоянии линий электропередач и прогнозах обледенения [1, 17].

Таким образом, архитектура нашего веб-приложения отвечает всем требованиям удобства и функциональности [22], создавая эффективный инструмент для мониторинга метеорологических условий в контексте управления линиями электропередач [17, 23].

Выводы по главе 2

В этой главе мы подробно рассмотрели этапы проектирования веб-приложения метеомониторинга ЛЭП. В рамках логического моделирования были созданы диаграммы вариантов использования, диаграммы деятельности, диаграммы классов. Эти артефакты позволили сформулировать ключевые функциональные требования и описать взаимосвязи между основными сущностями системы, что создаёт прочную основу для её последующей реализации.

На этапе проектирования внутренней структуры приложения обоснован выбор иерархической структуры, которая обеспечивает логичную организацию компонентов и способствует удобству работы как пользователей, так и разработчиков. При проектировании внешней структуры особое внимание уделено вопросам удобства и простоты интерфейса: описаны ключевые блоки пользовательского взаимодействия.

Таким образом, вся совокупность проектных решений в данной главе направлена на то, чтобы обеспечить надёжную, понятную и эффективную архитектуру программного продукта, соответствующую требованиям пользователей и спецификам предметной области.

Глава 3 Реализация и тестирование

3.1 Выбор и описание программных средств разработки

Выбор среды разработки (IDE) является важным этапом при создании программной системы [2, 3], так как от него зависит удобство работы, скорость разработки, производительность и надежность кода [29]. Разные технологии требуют специализированных инструментов, поэтому для каждого микросервиса будет выбрана наиболее подходящая среда разработки [12].

Для разработки микросервисов на Java выбрана IntelliJ IDEA (рисунок 21). Данная IDE предоставляет мощные инструменты для работы с Spring Boot, встроенные средства отладки, авто дополнение кода, удобную систему работы с зависимостями и поддержку баз данных. Это делает IntelliJ IDEA оптимальным выбором для разработки серверной части системы.

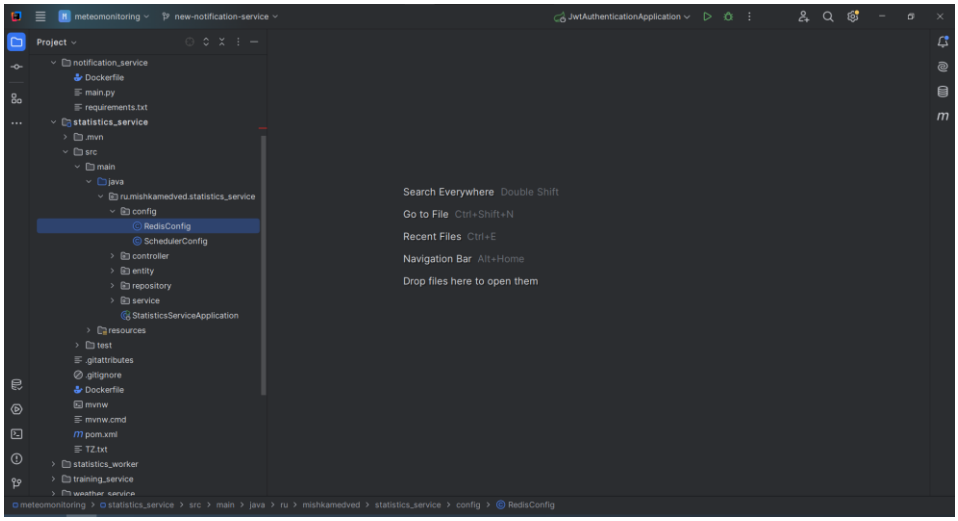

Рисунок 21 – IntelliJ IDEA

Для сервисов, написанных на Python, используется PyCharm (рисунок 22). Данный инструмент предлагает встроенную поддержку FastAPI, Django, Flask, мощные средства рефакторинга, удобную работу с виртуальными

окружениями и автоматическое управление зависимостями. Это позволяет эффективно разрабатывать и тестировать микросервисы на Python.

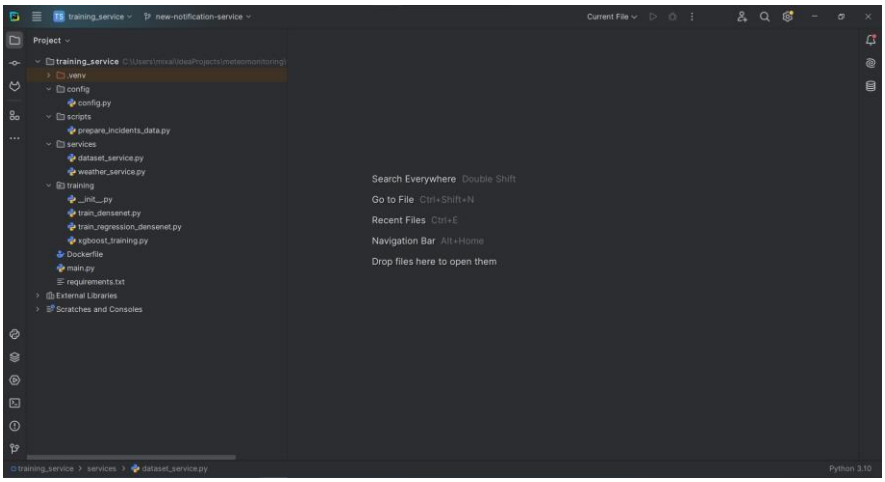

Рисунок 22 – PyCharm

Фронтенд-часть, написанная на TypeScript, разрабатывается в Visual Studio Code (рисунок 23). Этот редактор кода предоставляет удобную интеграцию с npm, автодополнение кода, поддержку расширений для работы с React и CSS, а также встроенные инструменты для работы с системами контроля версий, такими как Git. Благодаря простоте и гибкости VS Code является оптимальным выбором для написания клиентской части приложения.

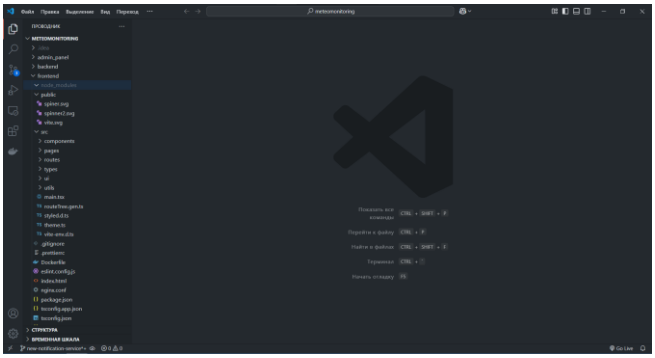

Рисунок 23 – Visual Studio Code

Использование разных IDE для различных сервисов позволяет максимально эффективно использовать возможности каждой технологии и упрощает процесс разработки. Выбор IntelliJ IDEA для Java, PyCharm для Python и Visual Studio Code для фронтенда обусловлен их возможностями и удобством работы с соответствующими технологиями. Такой подход повышает производительность команды и делает процесс разработки более удобным и структурированным.

3.2 Выбор технологии реализации проекта веб-приложения

Для разработки веб-приложения мониторинга и прогнозирования обледенения линий электропередач мы используем стек технологий, включающий как серверные, так и клиентские решения [10, 12]. В этом разделе мы рассмотрим выбранные инструменты, их назначение и используемые среды разработки.

Клиентская часть отвечает за отображение данных и взаимодействие пользователя с системой [3], [21]. Мы выбрали следующий стек технологий:

- TypeScript [26] – язык программирования, расширяющий JavaScript за счет статической типизации. Это снижает вероятность ошибок и улучшает читаемость кода [25];
- React [10] – библиотека для создания компонентов пользовательского интерфейса. React позволяет эффективно работать с динамическими данными и обновлять интерфейс без полной перезагрузки страницы [21];
- NGINX – веб-сервер, который используется для раздачи статических файлов и проксирования запросов между клиентом и сервером;
- Vite [14] – инструмент сборки, ускоряющий процесс разработки за счет мгновенной перезагрузки и оптимизированной компиляции.

Серверная часть отвечает за обработку данных, прогнозирование и API для обмена информацией с клиентом [2, 12]. Мы используем:

- Python [9, 16] – основной язык программирования, обладающий богатой экосистемой библиотек для обработки данных и машинного обучения;
- FastAPI [28] – веб-фреймворк, который поддерживает асинхронное программирование, что ускоряет обработку запросов;
- Java Spring Boot [2, 12] – используется для микросервиса аутентификации и авторизации (JWT) и для сохранения истории показаний по линии за час.

Обработка метеорологических данных требует работы с различными инструментами:

- Pandas [9] – библиотека для анализа данных, используемая при обработке метеоданных;
- PostgreSQL [8, 24] – основная реляционная база данных, в которой хранятся данные о пользователях, линиях электропередач и прогнозах;
- Redis [11] – NoSQL-хранилище для кэширования данных, что ускоряет доступ к предсказанным значениям вероятности обледенения.

Выбранные технологии и среды разработки обеспечивают высокую производительность, удобство работы и надежность системы. Использование IntelliJ IDEA для Java, PyCharm для Python и Visual Studio Code для фронтенда позволяет нам эффективно разрабатывать и тестировать микросервисы.

3.3 Выбор и описание средств работы с базой данных

В мире реляционных и нереляционных баз данных выделяются два основных игрока: PostgreSQL и Redis [8, 11, 24]. Каждая из этих систем обладает уникальными особенностями и преимуществами, что делает их подходящими для различных типов приложений [6, 22].

PostgreSQL – это мощная реляционная система управления базами

данных, известная своей надежностью и поддержкой сложных запросов. Она использует язык SQL для взаимодействия с данными и обеспечивает высокую степень согласованности благодаря соблюдению принципов ACID (атомарность, согласованность, изоляция, долговечность). PostgreSQL отлично подходит для приложений, требующих структурированного хранения данных, таких как учетные системы или аналитические платформы. Она поддерживает множество типов данных, включая JSON, что позволяет использовать её в гибридных сценариях. Благодаря своей способности обрабатывать сложные запросы и обеспечивать целостность данных, PostgreSQL является идеальным выбором для хранения информации о пользователях, компаниях и линиях электропередач в нашем проекте.

Redis, с другой стороны, представляет собой NoSQL базу данных типа ключ-значение, которая известна своей высокой производительностью и низкой задержкой при доступе к данным. Она хранит данные в оперативной памяти, что обеспечивает мгновенный доступ к часто запрашиваемой информации. Redis идеально подходит для задач кэширования и временного хранения данных, таких как сессии пользователей или результаты запросов к базе данных. В нашем проекте Redis будет использоваться для хранения прогнозов обледенения, что позволит быстро получать актуальные данные без задержек. Благодаря своей простоте и высокой скорости работы Redis часто используется в сочетании с реляционными базами данных, такими как PostgreSQL, что позволяет оптимизировать производительность приложения.

Таким образом, выбор PostgreSQL и Redis для нашего проекта обеспечивает надежное хранение структурированных данных с высокой степенью согласованности и мгновенный доступ к временным данным. Это сочетание технологий позволяет эффективно управлять данными о метеорологических условиях и обеспечивать пользователям актуальную информацию в реальном времени.

3.4 Реализация веб-приложения метеомониторинга

Рассмотрим разработку backend микросервиса

Определившись с необходимыми инструментами, можно приступить к реализации программной части. Разработка начинается с создания микросервиса backend, отвечающего за обработку данных, взаимодействие с базой данных и предоставление API для фронтенда. В рамках мультимодульного проекта, созданного в среде разработки PyCharm, был сформирован отдельный модуль backend, в который добавлена базовая структура каталогов и файлов, обеспечивающих модульность и расширяемость системы.

На рисунке 24 представлена структура модуля backend.

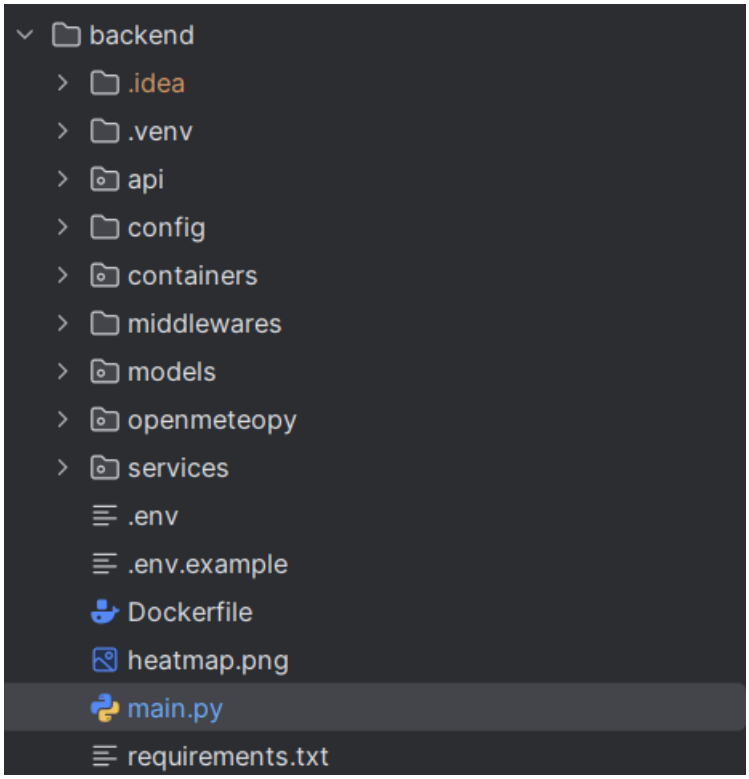

Рисунок 24 – Структура модуля backend

В папке api находятся ендпоинты для общения с другими сервисами. Папка config хранит конфигурационные настройки для данного микросервиса.

В containers содержится реализация инверсии зависимостей. В папке middlewares находится промежуточное ПО, такое как auth_middleware.py, которое отвечает за обработку аутентификации запросов. Папка models содержит описание моделей данных. В файле anomaly_prediction.py определены структуры для работы с прогнозом аномалий, db_models.py содержит модели базы данных, а response_models.py описывает форматы данных для ответов API. В папке openmeteopy реализован клиент для работы с OpenMeteo API. Файл client.py отвечает за взаимодействие с API, exceptions.py – за обработку ошибок. Подпапки daily, fifteen_minutes, hourly и options содержат настройки запросов для разных временных интервалов. В папке services расположена основная бизнес-логика приложения. inference_service.py реализует сервис прогнозирования обледенения, interpolation_service.py выполняет интерполяцию данных, а plots_service.py отвечает за построение графиков. redis_client.py настраивает подключение к Redis, weather_params.py обрабатывает параметры метеоданных, а weather_service.py занимается их получением и обработкой.

Таким образом, структура кода организована так, чтобы каждая папка отвечала за отдельный аспект работы приложения.

Файл api.py представляет собой главный класс для общения с сервисом обучения и фронтендом.

Main.py отвечает за инициализацию и запуск приложения. В нем создается экземпляр FastAPI, настраиваются зависимости, маршруты и промежуточное ПО. После этого приложение запускается локально с помощью Uvicorn, обеспечивая точку входа для обработки запросов.

Requirements.txt – хранит в себе название библиотек и их версии которые используются для этого микросервиса.

Dockerfile – описывает процесс создания образа для приложения на Python. Позволит запускать его в docker-compose файле в отдельном контейнере. В итоге в этом сервисе реализована логика.

Рассмотрим разработку микросервиса нейросети.

Мы начали с подготовки данных для прогнозирования степени обледенения линий электропередачи использовался сервис Open-Meteo, предоставляющий погодные данные для точек расположения опор ЛЭП. Сбор данных осуществлялся с почасовой частотой, а затем они проходили предобработку для адаптации к задачам машинного обучения.

Для этого мы инициализировали новый сервис wether_service и создали следующую структуру, которая изображена на рисунке 25.

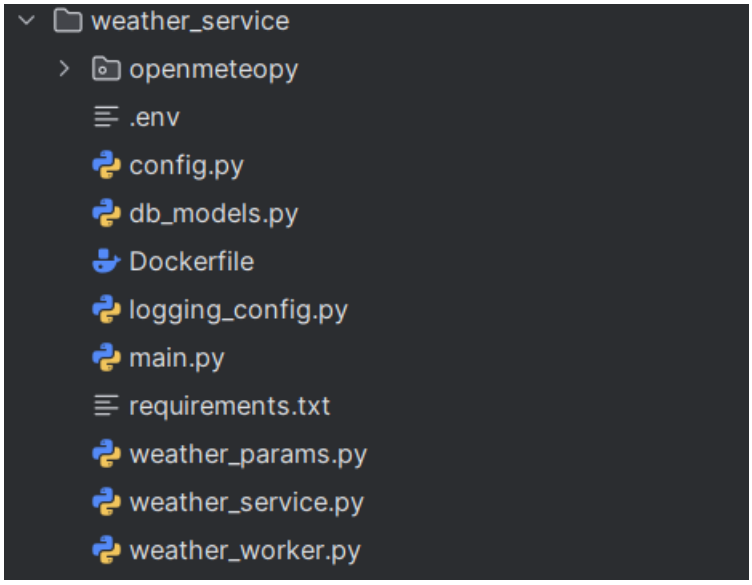

Рисунок 25 – Структура сервиса получение погодных данных

На первом этапе предобработки категориальный параметр погодных условий (weather_code) был преобразован в формат one-hot encoding. Это позволило сделать данные совместимыми с моделями глубокого обучения. Далее был вычислен параметр wind_line_angle, отражающий угловое соотношение между направлением ветра и ориентацией линии электропередачи. Он рассчитывался как абсолютное значение синуса угла между направлением ветра (0–360 градусов) и направлением линии, определяемым координатами двух ближайших опор. Параметр принимал значение 1 при перпендикулярном ветре и 0 при параллельном.

Дополнительно в качестве характеристики была включена высота над уровнем моря, учитывающая географические условия. Кроме того, использовались технические характеристики линий: масса провода без льда и критическая нагрузка, которую он может выдержать. Эти параметры позволили выделить четыре категории риска обледенения:

- нормальный (0–25% критической нагрузки);
- умеренный (25–50%);
- опасный (50–75%);
- критический (75–100%).

Первичный анализ выявил сильный дисбаланс классов (96,85% примеров относились к нормальному классу). Для балансировки данных были применены методы уменьшения выборки (undersampling) с алгоритмом NearMiss и увеличения выборки (oversampling) с BorderlineSMOTE. В результате все классы получили равномерное распределение (25% каждый).

Далее мы приступаем к обучению модели, данные были разделены на обучающую, валидационную и тестовую выборки (60/20/20), при этом записи, относящиеся к одной линии электропередачи, не пересекались между выборками.

Для классификации степени обледенения была разработана модель многослойного перцептрона (MLP) со следующей архитектурой:

- полносвязный слой (1024 нейрона, ReLU, L2-регуляризация, Dropout 50%);
- полносвязный слой (1024 нейрона, ReLU, L2-регуляризация, Dropout 50%);
- полносвязный слой (512 нейронов, ReLU, L2-регуляризация, Dropout 50%);
- полносвязный слой (256 нейронов, ReLU, L2-регуляризация, Dropout 50%);
- полносвязный слой (128 нейронов, ReLU, L2-регуляризация, Dropout 50%);

– выходной слой (4 нейрона, softmax).

Для оценки качества работы классификационной модели использовался отдельный тестовый набор данных, не участвующий в процессе обучения. В результате проведения тестирования были получены следующие метрики:

– потери (Loss): 1.0143;

– точность (Accuracy): 81.08%.

Для более детального анализа была сформирована таблица отчета по классификации, в которую включены значения precision (точности), recall (полноты) и F1-меры для каждого класса. Эти показатели позволяют судить о сбалансированности модели и качестве прогнозов для различных типов входных данных (таблица 4).

Таблица 4 – Отчет по классификации

Класс	Точность	Полнота	F1-мера
Нормальный	0.75	0.73	0.74
Умеренный	0.82	0.84	0.83
Опасный	0.84	0.75	0.79
Критический	0.84	0.97	0.90

Модель достигла высокой точности, но возможны ошибки в разграничении классов "Нормальный" и "Умеренный" из-за колебаний проводов под воздействием ветра. В дальнейшем планируется улучшение модели путем добавления дополнительных метеорологических параметров и экспериментов с альтернативными архитектурами.

Рассмотрим разработку frontend микросервиса.

После завершения разработки микросервисов backend и обучения нейросети, следующим этапом является создание микросервиса frontend. Данный сервис отвечает за отображение данных пользователям, обеспечение удобного интерфейса для работы с прогнозами обледенения линий электропередач и взаимодействие с backend-сервисами.

После инициализации проекта структура сервиса выглядит следующим образом.

На рисунке 26 изображена структура frontend сервиса

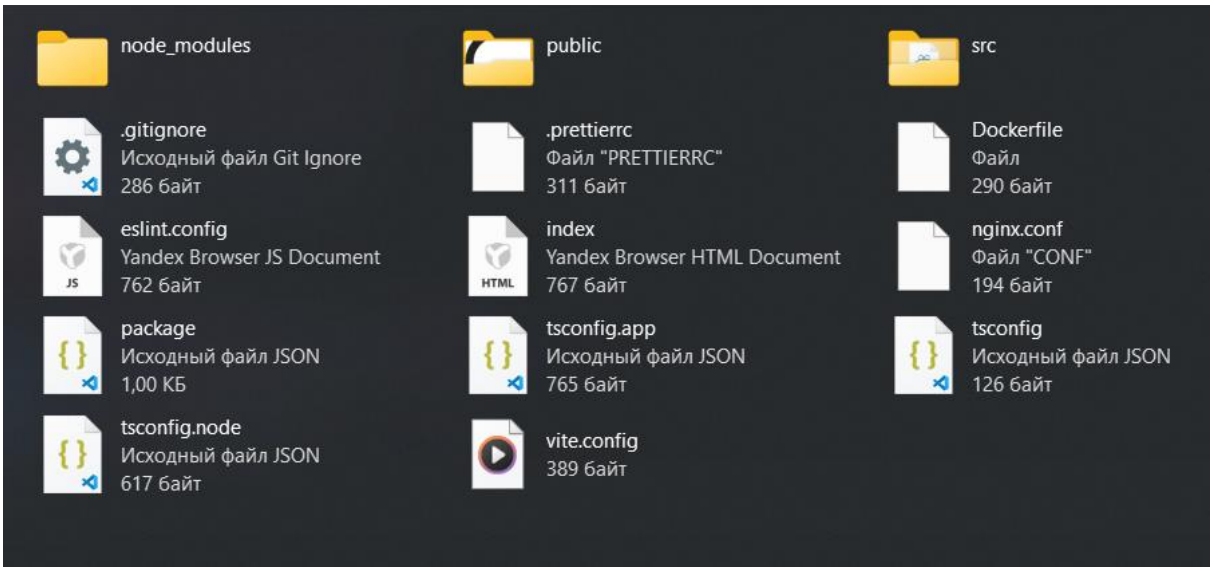

Рисунок 26 – Структура frontend сервиса

Фронтенд-сервис был реализован с использованием библиотеки React в сочетании с инструментом сборки Vite, что обеспечило высокую производительность, быструю сборку и удобную настройку проекта. Vite выбран в качестве среды разработки благодаря поддержке горячей перезагрузки (hot-reload) и простой конфигурации, что существенно ускоряет итерации при разработке интерфейса.

Основной файл index.html содержит минимальную базовую разметку, а вся логика и конфигурация маршрутизации приложения сосредоточена во входной точке – файле main.tsx, где используется библиотека @tanstack/react-router для управления переходами между страницами.

Листинг кода представлен в Приложении А.

Для получения прогнозных данных о вероятности обледенения веб-приложение отправляет запрос к backend-сервису, который, в свою очередь, взаимодействует с моделью нейросети и возвращает результаты

пользователю. Этот процесс реализуется через API, обеспечивающий асинхронную передачу данных и быструю обработку запросов.

На рисунке 27 представлено, как осуществляется API-запрос от фронтенда к backend-сервису для получения прогнозов, сформированных нейросетью.

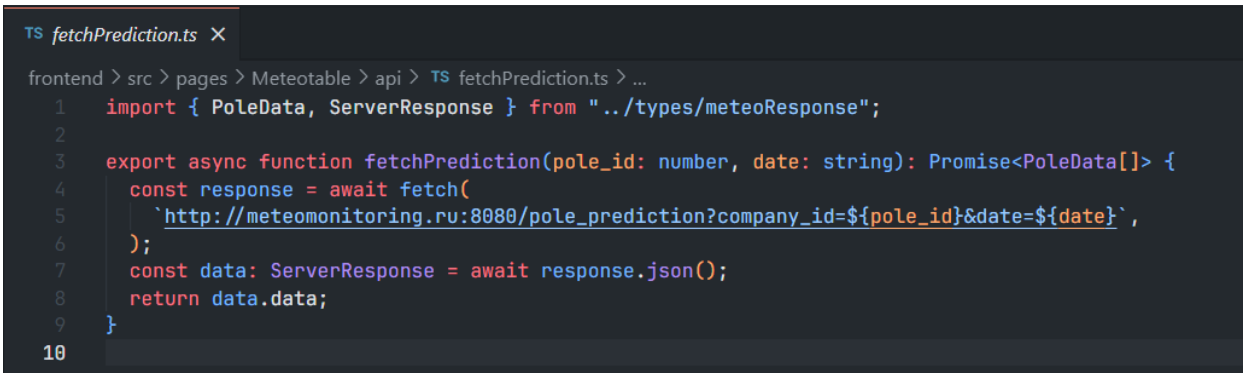

Рисунок 27 – Запрос в сервис backend для получения прогнозов от нейросети

Для визуализации прогнозов используется таблица, созданная с помощью @tanstack/react-table. Основные данные включают номер опоры, наименование линии, дату последнего прогноза и вероятность обледенения:

Также реализована серверная пагинация с использованием @tanstack/react-table, а для фильтрации данных по дате применяется библиотека date-fns.

Фронтенд-сервис развернут в Docker-контейнере и обслуживается через Nginx, который выполняет роль обратного прокси-сервера.

На рисунке 28 представлена конфигурация Nginx, обеспечивающая маршрутизацию запросов к основному веб-приложению и административной панели

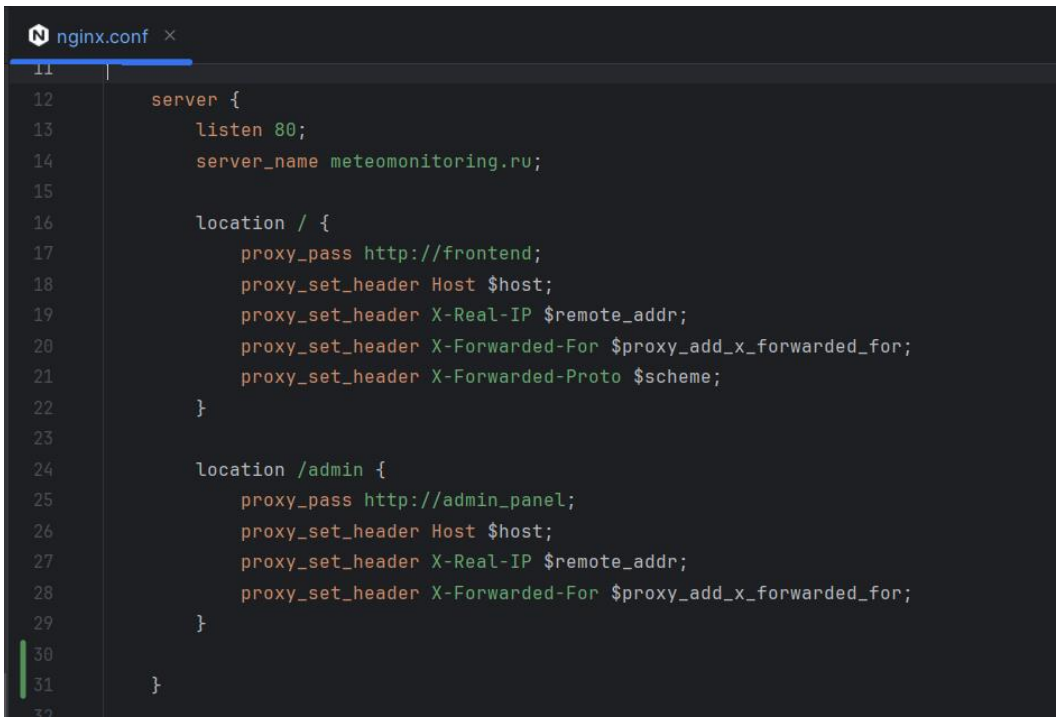

Рисунок 28 – Конфигурация Nginx

Для контейнеризации фронтенд-сервиса используется следующий Dockerfile, в котором описаны этапы сборки проекта на базе Vite и его последующего запуска через Nginx. На рисунке 29 представлено содержимое Dockerfile, используемого для сборки и развёртывания frontend.

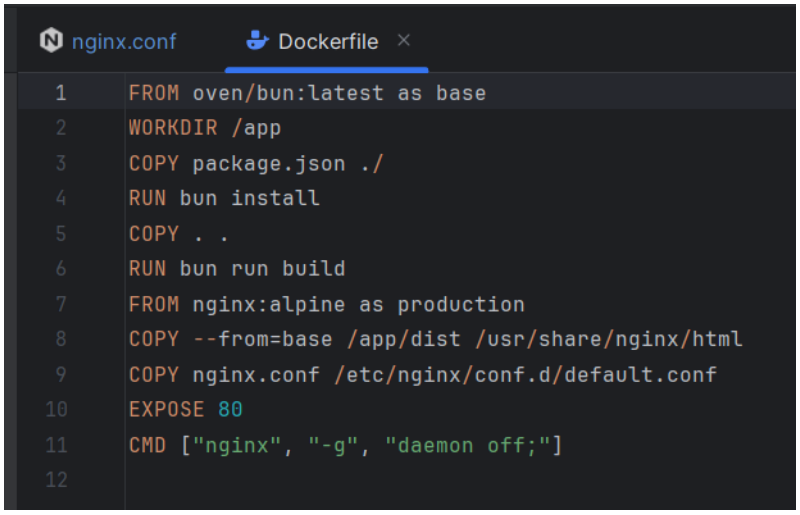

Рисунок 29 – Dockerfile frontend модуля

Фронтенд-сервис интегрирован в систему через docker-compose.yml, где определены его зависимости, сетевые настройки и параметры взаимодействия с другими микросервисами системы.

На рисунке 30 представлено содержание docker-compose.yml, описывающее настройку и запуск фронтенд-сервиса в контейнере.

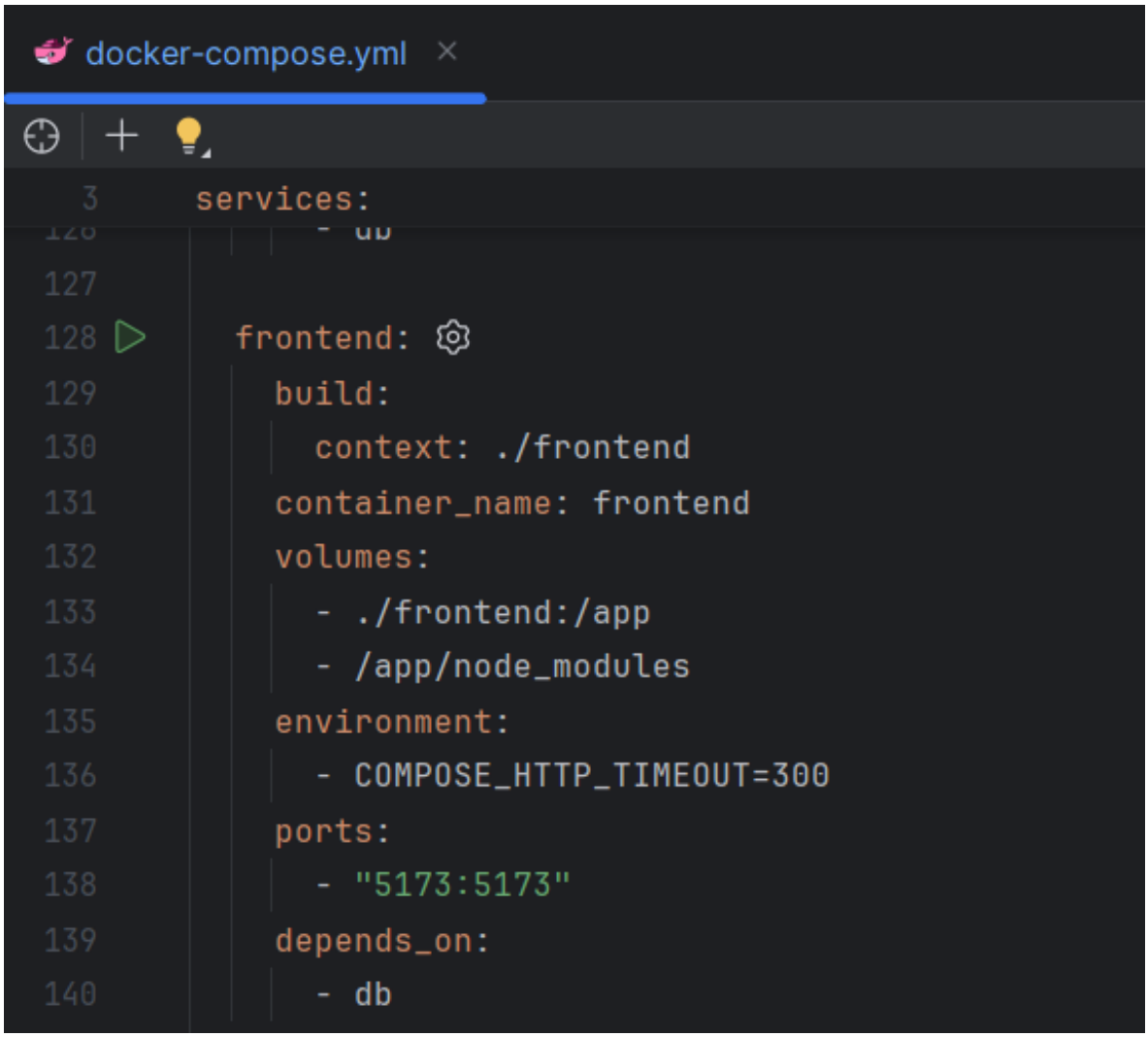

Рисунок 30 – Конфигурация docker-compose для фронтенд-сервиса

Таким образом, микросервис frontend предоставляет удобный пользовательский интерфейс для мониторинга данных и взаимодействует с backend-сервисами для получения прогнозов обледенения.

3.5 Тестирование веб приложения

После разработки всех микросервисов необходимо выполнить серию тестов для проверки корректности их работы. В этом разделе приведены результаты тестирования, описание использованных тестовых данных, этапы обработки информации и анализ полученных итогов.

Для проверки корректной работы подсистемы были проведены тесты:

- соответствие сайта функциональным требованиям;
- просмотр результатов прогнозирования на нашем фронтенде;
- корректность отображения и функционирования веб-приложения на всех;
- отсутствие ошибок в коде;
- Процесс тестирования нашего веб-приложения состоит из нескольких этапов:
- функциональное тестирование, на котором проверяются функции на их соответствие требованиям заказчика;
- конфигурационное тестирование, которое проверяет корректное отображение на различных экранах и браузерах;
- нагрузочное тестирование, заключающееся в проверке уровня критических нагрузок на сервер;
- тестирование пользователем, в процессе которого продукт тестируется конечным пользователем на удобство использования.

Переходим к тестированию, для этого перейдем в админку нашего приложения, так как сначала администратор компании “РУРИКО” должен добавить компанию в систему и создать аккаунт для сотрудника компании. Добавим тестовую компанию с названием “Демо компания”

На рисунке 31 представлен список компаний с нашей тестовой компанией

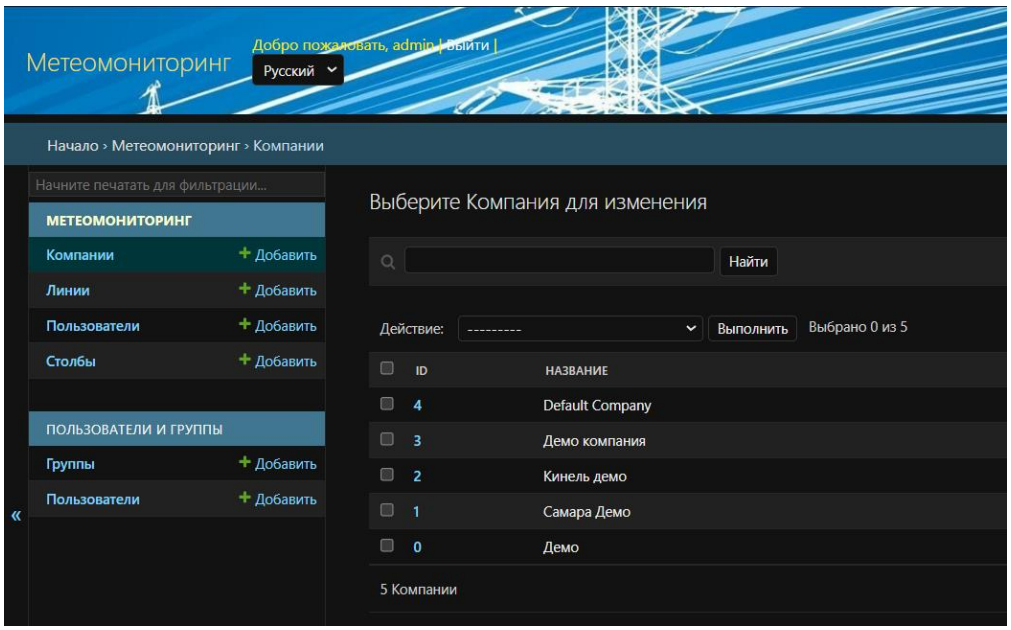

Рисунок 31 – Список компаний

Далее нам нужно, добавить аккаунты пользователей, через которые диспетчера будут входить в наше приложение. Добавим 2 аккаунта для компании.

На рисунке 32 представлен список пользователей для компании “Демо компания”

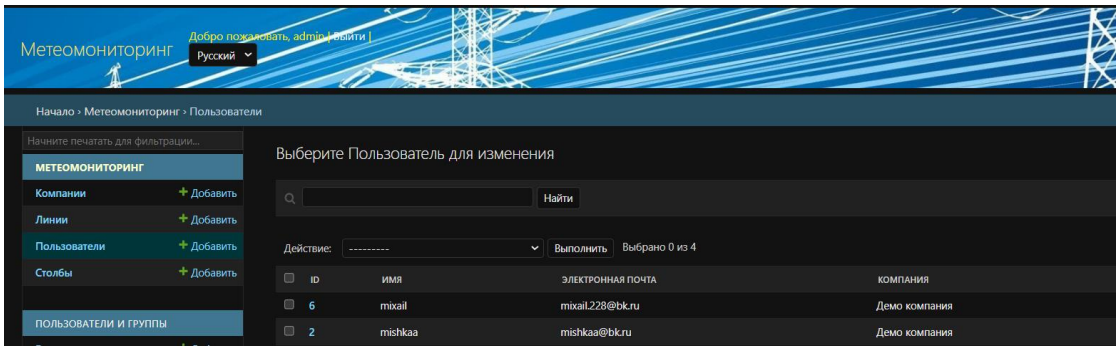

Рисунок 32 – Список пользователей компании “Демо компания”

У каждой компании есть свои ЛЭП и они имеют свои названия, так же им нужно добавить столбы добавим всего 2 для тестов этого вполне хватит.

На рисунке 33 представлен список ЛЭП с нашей линией.

Рисунок 33 – Список линий

Теперь необходимо добавить опоры, которые принадлежат выбранной линии электропередач. Для целей тестирования будет достаточно создать две опоры, что позволит проверить корректность работы интерфейса, отобразить информацию на карте, а также убедиться в правильной связи между линией и опорами.

Добавим две опоры с координатами:

- первая опора: x = 69.343985, y = 88.210393;
- вторая опора: x = 67.494690, y = 63.977495.

На рисунке 34 представлен список опор, добавленных для тестовой линии электропередач

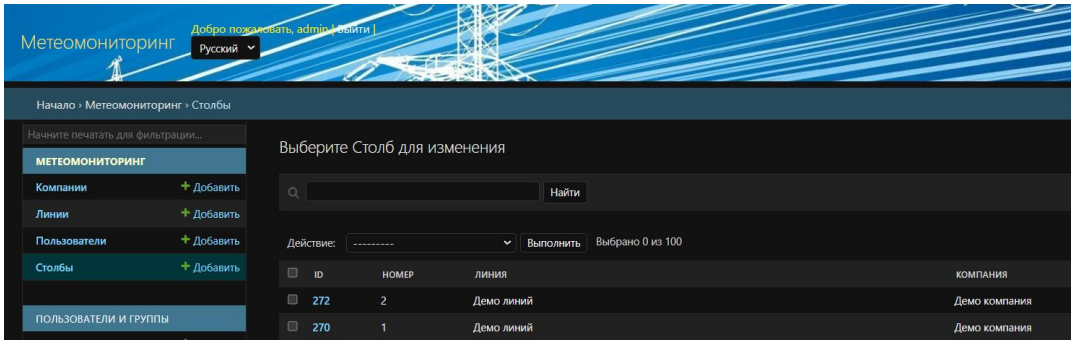

Рисунок 34 – Список опор, принадлежащих тестовой ЛЭП

После того как были добавлены все необходимые тестовые данные (компания, пользователи, линии электропередач и опоры), необходимо подождать, пока система выполнит автоматический запрос метеоданных по заданным координатам. После получения актуальных погодных данных и их обработки с помощью нейросети, в таблице отобразится прогноз вероятности обледенения ЛЭП. Это позволяет убедиться в корректности работы механизма интеграции с метеосервисом и логики прогнозирования.

Дальнейшее тестирование выполняется от лица пользователя приложения "Метеомониторинг". При входе в систему пользователь в первую очередь попадает на страницу аутентификации, где необходимо ввести логин и пароль для доступа к функционалу системы.

На рисунке 35 представлена форма логина для входа в систему.

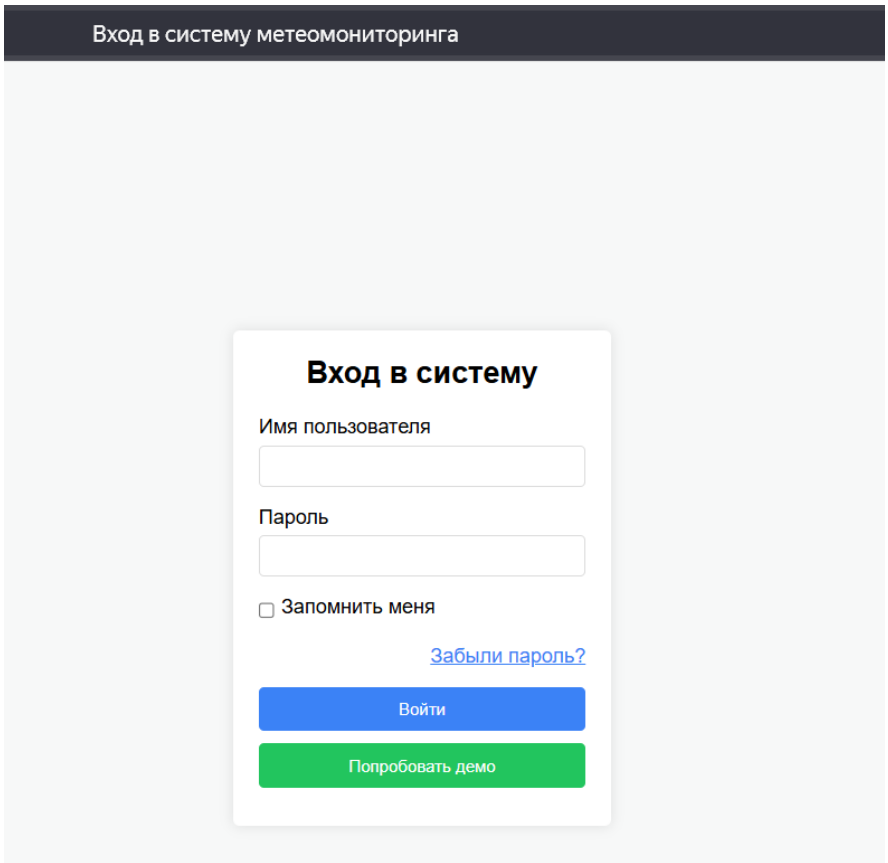

Рисунок 35 – Форма входа в систему

После успешной аутентификации пользователь попадает в основное окно приложения. Перед тем как отобразится информация о прогнозах по добавленным опорам, система выполняет фоновую загрузку данных. В этот момент отображается анимация загрузки, свидетельствующая о том, что происходит получение и обработка данных с сервера.

На рисунке 36 представлена анимация загрузки таблицы данных, появляющаяся до отображения прогноза вероятности обледенения.

Рисунок 36 – Анимация загрузки таблицы данных перед отображением прогноза

После того как данные о погоде были получены и обработаны, система отображает таблицу с рассчитанными значениями вероятности обледенения для каждой опоры линии электропередачи. Прогноз формируется с учётом текущих и прогнозируемых метеоусловий на протяжении суток, включая температуру воздуха, влажность, осадки и другие параметры.

На рисунке 37 представлена таблица с прогнозом вероятности обледенения по каждой опоре.

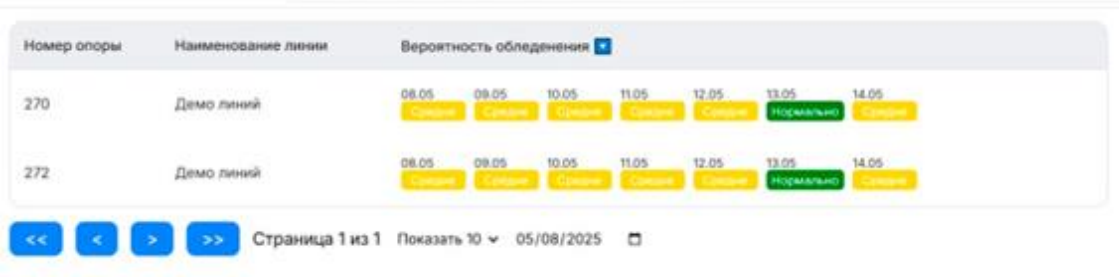

Рисунок 37 – Таблица с прогнозом вероятности обледенения по опорам ЛЭП

Таким образом, в результате тестирования была подтверждена корректная работа всех ключевых компонентов системы метеомониторинга:

- данные о компаниях, линиях и опорах успешно добавляются и отображаются через административную панель;
- прогнозирование обледенения выполняется автоматически на основе метеоданных, получаемых по координатам опор;
- отображение прогнозов в пользовательском интерфейсе реализовано корректно – данные отображаются в виде таблицы после обработки и загрузки;
- пользовательский интерфейс реагирует предсказуемо, включая аутентификацию, отображение загрузки и результаты прогнозов;
- взаимодействие между микросервисами осуществляется стабильно, что говорит о правильно выстроенной архитектуре и настройке системы.

Данные результаты подтверждают готовность системы к использованию в демонстрационных и пилотных режимах. При дальнейшем внедрении следует учитывать производственные условия и масштабируемость.

Выводы по главе 3

В данной главе была рассмотрена реализация и тестирование веб-приложения метеомониторинга линий электропередач. В процессе разработки была использована современная стековая архитектура: фронтенд-сервис

реализован с использованием React и Vite, а бекэнд-часть – в виде микросервисов на базе Python (FastAPI) и Java (Spring Boot). Для маршрутизации на клиенте применён пакет @tanstack/react-router, а для развёртывания компонентов системы использована контейнеризация с помощью Docker и docker-compose.

Система прошла этап функционального тестирования: была добавлена тестовая компания, линии электропередач, опоры и пользователи, после чего были получены прогнозы обледенения на основе координат и метеоданных. Пользовательский интерфейс проверен на корректность отображения информации, загрузку и навигацию.

Приложение успешно развернуто в изолированной среде, фронтенд обслуживается через Nginx, и обеспечена интеграция всех микросервисов. Тестирование подтвердило корректность обработки данных, визуализации результатов и устойчивость архитектуры.

Таким образом, веб-приложение метеомониторинга реализовано и протестировано в соответствии с поставленными требованиями, и готово к дальнейшему развертыванию и использованию в реальных условиях.

Заключение

В данной выпускной квалификационной работе было разработано программное обеспечение для метеомониторинга линий электропередач с использованием искусственного интеллекта. В ходе работы был проведён анализ предметной области, определены потребности конечных пользователей – диспетчеров энергетических компаний, а также сформулированы функциональные и нефункциональные требования к системе.

На этапе проектирования была построена информационная модель, разработаны схемы базы данных, определены ключевые компоненты архитектуры, включая микросервисы для сбора метеоданных, прогнозирования обледенения, визуализации и управления пользователями. Также были реализованы механизмы аутентификации и авторизации, а интерфейс спроектирован с учётом требований к интуитивной понятности и адаптивности.

Для реализации системы использовались современные технологии: фронтенд разработан с использованием React и Vite, микросервисы написаны на Python (FastAPI) и Java (Spring Boot), развертывание компонентов организовано с помощью Docker и docker-compose. Прогнозирование вероятности обледенения осуществляется на основе моделей машинного обучения, обученных на исторических метеоданных.

Проведённое тестирование подтвердило корректность функционирования всех компонентов системы: от создания компаний, ЛЭП и опор до получения прогнозов и их отображения в интерфейсе. Все требования, предъявленные к системе, были выполнены.

Разработанное программное обеспечение является надёжным инструментом для мониторинга и прогнозирования обледенения ЛЭП. Оно помогает диспетчерам принимать обоснованные решения, повышает оперативность реагирования и снижает риск аварийных ситуаций.

Список используемой литературы и используемых источников

1. АО «Россети». Итоги зимнего периода 2023–2024 гг. // https://www.rosseti.ru

2. Бен-Аврам Г. Spring в действии. 6-е издание. – М.: Диалектика, 2022. – 500 с.

3. Блинов А. В., Соловьёв Г. А. Java. Библиотека профессионала. Том 1. Основы. – СПб.: Питер, 2021. – 800 с.

4. Вайсала (Vaisala). Weather and Environmental Monitoring for Power Transmission Systems. URL: https://www.vaisala.com

5. Гидрометцентр России. Погодные условия и режимы гололедообразования на территории РФ. – http://www.meteoinfo.ru

6. ГОСТ Р 56764-2015. Надежность энергетических объектов. Общие положения.

7. Гудфеллоу И., Бенджио Й., Курвил А. Глубокое обучение. – М.: Диалектика, 2020. – 752 с.

8. Документация по React и TypeScript: URL: https://react-typescript-cheatsheet.netlify.app/

9. Документация по Redis. URL: https://redis.io/docs/

10. Документация по Spring Framework. URL: https://docs.spring.io/spring-framework/

11. Документация по Vite. URL: https://vitejs.dev/

12. Козлов И.А., Назаров А.Б. Прогнозирование гололедных нагрузок на ВЛ с использованием искусственных нейронных сетей // Энергетика и промышленность России. – 2023. – №3. – С. 25–28.

13. Лутц М. Изучаем Python. 5-е изд. – СПб.: Символ-Плюс, 2021. – 1600 с.

14. Методические рекомендации по борьбе с гололедно-изморозевыми отложениями на ВЛ 6–750 кВ / Под ред. В.М. Кармазина. – М.: Энергия, 2019.

15. Назаренко Е.А. Использование погодных данных в энергетике:

современные подходы // Автоматизация и IT в энергетике. – 2021. – №6. – С. 33–36.

16. ООО «НТЦ Инструмент-Микро». Официальный сайт. Раздел: Системы диагностики ВЛ. URL: http://www.instrument-micro.ru

17. Рашид В. Python и машинное обучение. – СПб.: Питер, 2021. – 416 с.

18. Сидорова И. Ю. Методологии разработки требований к программному обеспечению. – М.: Инфра-М, 2019. – 256 с.

19. Флэнаган Д. JavaScript. Подробное руководство. 7-е изд. – СПб.: Питер, 2021. – 1088 с.

20. Фримен Э., Робсон Э. Изучаем TypeScript. – СПб.: Питер, 2023. – 416 с.

21. Чолле А. Нейронные сети: создание и обучение на Python. – СПб.: Питер, 2022. – 368 с.

22. Чурсин Ю.Г., Сафонов А.В. Применение SCADA-систем в энергетике и вопросы интеграции с метеоданными // Вестник энергетики. – 2022. – №8. – С. 14–19.

23. Шилдт Г. Java. Полное руководство. 12-е изд. – М.: Вильямс, 2022. – 1350 с.

24. François Chollet. Deep Learning with Python. 2nd ed. – Manning, 2021. – 504 p.

25. Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning. – MIT Press, 2016. – 800 p.

26. Mark Lutz. Learning Python. 5th ed. – O’Reilly Media, 2013. – 1648 p.

27. React Contributors. React TypeScript Cheatsheet. – https://react-typescript-cheatsheet.netlify.app/

28. SEL (Schweitzer Engineering Laboratories). Line Monitoring and Icing Detection Systems. URL: https://selinc.com

29. The PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/

Приложение А

Листинг микросервиса фронтенда

```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <link
      rel="icon"
      type="image/svg+xml"
      href="/vite.svg" />
    <meta
      name="viewport"
      content="width=device-width, initial-scale=1.0" />
    <link
      rel="preconnect"
      href="https://fonts.googleapis.com" />
    <link
      rel="preconnect"
      href="https://fonts.gstatic.com"
      crossorigin />
    <link
       href="https://fonts.googleapis.com/css2?family=Inter:ital,opsz,wght@0,
14..32,100..900;1,14..32,100..900&display=swap"
      rel="stylesheet" />
    <title>Meteomonitoring</title>
  </head>
  <body>
    <div id="root"></div>
    <script
```

```
   type="module"
     src="/src/main.tsx"></script>
  </body>
</html>


import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import { RouterProvider, createRouter } from "@tanstack/react-router";
import { routeTree } from "./routeTree.gen";
import PendingComponent from "./components/PendingComponent";

const router = createRouter({
  routeTree,
  defaultPendingComponent: PendingComponent,
});

createRoot(document.getElementById("root")!).render(
  <StrictMode>
    <RouterProvider router={router} />
  </StrictMode>,
);

import {
  ColumnDef,
  PaginationState,
  flexRender,
  getCoreRowModel,
  getFilteredRowModel,
```

```
  getPaginationRowModel,
  getSortedRowModel,
  useReactTable,
} from "@tanstack/react-table";
import { PoleData } from "../types/meteoResponse";
import { getRouteApi } from "@tanstack/react-router";
import { useMemo, useState } from "react";
import Button from "@ui/Button";
import styled from "styled-components";
import { fetchPrediction } from "../api/fetchPrediction";

const TableContainer = styled.div`
  border: 1px solid ${(props) => props.theme.colors.gray[200]};
  border-radius: 0.8rem;
  margin: 0.8rem;
  overflow: auto;
`;

const ButtonContainer = styled.div`
  display: flex;
  gap: 12px;
  margin: 0 0.8rem 0 0.8rem;
  align-items: center;
`;

const Table = styled.table`
  width: 100%;
  border-spacing: 0;
```

```
  border-collapse: collapse;
  text-align: left;
  color: #222222;
  font-size: 0.833rem;
`;
const TableHeader = styled.thead``;
const TableRow = styled.tr`
  background-color: ${(props) => props.theme.colors.gray[100]};
`;
const TableHead = styled.th`
  font-weight: 500;
  padding: 0.8rem 0 0.8rem 0.8rem;
  border-bottom: 1px solid ${(props) => props.theme.colors.gray[200]};
`;

const TableData = styled.td`
  padding: 1rem 0 1rem 0.8rem;
`;

export function Meteotable() {
  const routeApi = getRouteApi("/table");
  const fetchData = routeApi.useLoaderData();

      const [selectedDate, setSelectedDate] = useState(new
Date().toISOString().split('T')[0]);
  const [data, setData] = useState(() => [...fetchData]);
  const [pagination, setPagination] = useState<PaginationState>({
    pageIndex: 0,
```

```
  pageSize: 10,
 });

 const updateData = async (date: string) => {
  try {
   const newData = await fetchPrediction(1, date);
   setData(newData);
  } catch (error) {
   console.error('Ошибка при загрузке данных:', error);
  }
 };

 const handleDateChange = (e: React.ChangeEvent<HTMLInputElement>)
=> {
  const newDate = e.target.value;
  setSelectedDate(newDate);
  updateData(newDate);
 };

 const columns = useMemo<ColumnDef<PoleData>[]>(
  () => [
   {
    accessorKey: "pole_id",
    header: "Номер опоры",
   },
   {
    accessorKey: "line_name",
    header: "Наименование линии",
```

```
    },
    {
      accessorKey: "created_at",
      header: "Последнее время предсказания",
    },
    {
      accessorKey: "max_probability",
      header: "Вероятность обледенения",
    },
  ],
  [],
);

const table = useReactTable({
  columns,
  data,
  debugTable: true,
  getCoreRowModel: getCoreRowModel(),
  getSortedRowModel: getSortedRowModel(),
  getFilteredRowModel: getFilteredRowModel(),
  getPaginationRowModel: getPaginationRowModel(),
  onPaginationChange: setPagination,
  initialState: {
    sorting: [
      {
        id: "max_probability",
        desc: true,
      },
```

```
    ],
  },
  state: {
    pagination,
  },
});

return (
  <>
    <TableContainer>
      <Table>
        <TableHeader>
          {table.getHeaderGroups().map((headerGroup) => (
            <TableRow key={headerGroup.id}>
              {headerGroup.headers.map((header) => {
                return (
                  <TableHead
                    key={header.id}
                    colSpan={header.colSpan}
                    onClick={header.column.getToggleSortingHandler()}>
                    {flexRender(
                      header.column.columnDef.header,
                      header.getContext(),
                    )}
                    {{
                      asc: " ▲",
                      desc: " ▼",
                    }[header.column.getIsSorted() as string] ?? null}
```

```
            </TableHead>
          );
        })}
      </TableRow>
    ))}
  </TableHeader>
  <tbody>
    {table.getRowModel().rows.map((row) => {
      return (
        <tr key={row.id}>
          {row.getVisibleCells().map((cell) => {
            return (
              <TableData key={cell.id}>
                {flexRender(
                  cell.column.columnDef.cell,
                  cell.getContext(),
                )}
              </TableData>
            );
          })}
        </tr>
      );
    })}
  </tbody>
 </Table>
</TableContainer>
<ButtonContainer>
 <Button onClick={() => table.firstPage()}>{"<<"}</Button>
```

```
<Button onClick={() => table.previousPage()}>{"<"}</Button>
    <Button onClick={() => table.nextPage()}>{">"}</Button>
    <Button onClick={() => table.lastPage()}>{">>"}</Button>
    <span>
     Страница {table.getState().pagination.pageIndex + 1} из{" "}
     {table.getPageCount().toLocaleString()} </span>
    <select
     value={table.getState().pagination.pageSize}
     onChange={(e) => {
      table.setPageSize(Number(e.target.value));
     }}>
     {[10, 20, 30, 40, 50].map((pageSize) => (
      <option
       key={pageSize}
       value={pageSize}>
       Показать {pageSize}
      </option>
     ))}
    </select>
    <input
     type="date"
     value={selectedDate}
     onChange={handleDateChange}
    />
   </ButtonContainer>
  </>
);
```