ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **«ДОСЛІДЖЕННЯ АЛГОРИТМІВ МАШИННОГО НАВЧАННЯ ДЛЯ КЛАСИФІКАЦІЇ МЕРЕЖЕВОГО ТРАФІКУ»**

на здобуття освітнього ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія

освітньо-професійної програми Комп'ютерні системи та мережі

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Олексій СОЛОД

Виконав: здобувач вищої освіти групи КСДМ-62

Олексій СОЛОД

Керівник: Андрій ЛЕМЕШКО
PHD, доцент
Рецензент: Вікторія ЖЕБКА
д.т.н, професор

Київ - 2024

ВСТУП

Використання методів машинного навчання у сучасності виявляється надзвичайно актуальним. Ці методи широко використовуються для створення різноманітних інтелектуальних систем. Машинне навчання - це сфера штучного інтелекту, яка стрімко розвивається. За визначенням Тома М. Мітчелла, відомого фахівця з комп'ютерних наук та піонера машинного навчання, це вивчення комп'ютерних алгоритмів, які дозволяють програмам автоматично покращуватися через досвід.

Алгоритм можна розглядати як набір інструкцій, які програміст задає, і які комп'ютер може обробити. Іншими словами, алгоритми машинного навчання вивчаються на основі досвіду, так само, як це робить людина. Наприклад, після певної кількості прикладів об'єкта, алгоритм машинного навчання зможе розпізнати цей об'єкт у нових, раніше невиданих сценаріях.

Машинне навчання використовує різноманітні методи для інтелектуальної обробки великої та складної інформації для прийняття рішень та/або прогнозування.

На практиці, закономірності, які вивчає комп'ютер (система машинного навчання), можуть бути дуже складними для пояснення. Наприклад, пошук зображень автомобіля у Google. Google добре працює у досягненні відповідних результатів, але як саме це відбувається? Пошук Google спочатку отримує велику кількість прикладів фотографій з написом "автомобіль", після чого комп'ютер (система машинного навчання) аналізує візерунки пікселів і кольорів, щоб визначити, що це справді автомобіль.

Машинне навчання сьогодні надзвичайно важливе, оскільки воно здатне розв'язувати складні реальні проблеми. Крім того, протягом останнього десятиліття воно вплинуло на різні галузі і продовжує це робити, оскільки все більше лідерів та дослідників спеціалізуються на машинному навчанні для подальших досліджень та розробки інструментів з позитивним впливом на різні сфери. Швидкість розвитку машинного навчання вражає завдяки змінам у зберіганні та обчисленні

даних, а також зростанню потужності обробки цих даних. Чим більше людей втягнено в цей процес, тим більше можемо очікувати прогресу у різних галузях завдяки машинному навчанню.

Один з прикладів використання машинного навчання - створення нейронних мереж для прогнозування. У цій роботі я буду використовувати машинне навчання для цієї мети. "Прогноз" - це результат роботи алгоритму після навчання на історичних даних та їх застосування до нових. Алгоритм генерує ймовірні значення для кожного запису у нових даних.

Термін "прогноз" може викликати певне непорозуміння. В певних ситуаціях це насправді означає передбачення майбутнього результату, наприклад, коли ви використовуєте машинне навчання для визначення найкращого кроку у маркетинговій стратегії. Але у інших випадках "прогноз" стосується, наприклад, того, чи була вже здійснена транзакція шахрайською. У цьому контексті транзакція вже відбулася, але ви робите обґрунтовані припущення стосовно її законності, що допомагає прийняти відповідне рішення.

Для різних видів прогнозу можуть застосовуватися різні типи нейронних мереж, такі як багатошарові персептрони, згорткові та повторювані нейронні мережі.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Протягом останніх років значна увага була приділена методам моніторингу та аналізу мережевого трафіку (МАМТ), які стали важливою темою для досліджень, спрямованих на підтримку продуктивності мережі. Ці методи вже широко використовуються як у промисловості, так і наукових дослідженнях у сфері управління мережею. Незважаючи на впровадження різноманітних методів МАМТ, розвиток нових технологій та парадигм у мережевій сфері ускладнює створення ефективних мереж. Нові мережі з великою кількістю вузлів, такі як Інтернет речей, потребують систематичного моніторингу для забезпечення їхньої ефективності.

Різні аспекти мережі вимагають від адміністраторів оцінювати її з різних позицій, таких як безпека, вимоги до якості обслуговування та оптимізація використання ресурсів. Для досягнення цих цілей використовуються методи МАМТ, такі як виявлення аномалій, класифікація мережевого трафіку, управління несправностями та прогнозування трафіку.

Методи МАМТ розділяються на дві основні групи: активні та пасивні. Активні методи передбачають генерацію та введення тестового трафіку в мережу для вивчення її стану. Вони дозволяють отримувати дані про продуктивність мережі у реальному часі та є основою для контролю обслуговування на основі угод про рівень обслуговування. З іншого боку, пасивні методи використовуються для аналізу реального мережевого трафіку без втручання у його потік. Вони корисні для вирішення завдань управління та планування, зокрема виявлення несправностей та оцінки якості досвіду користувача.

Таблиця 1.1 узагальнює застосування активних та пасивних методів МАМТ.

Таблиця 1.1 – Порівняння активних та пасивних методів

Активні методи	Пасивні методи

Прямий і наскрізний аналіз	Кроки для усунення несправностей
Якість обслуговування	Якість досвіду
Моніторинг у режимі реального часу	Діагностика проблем протоколу
Моніторинг ефективності мережі та послуг	Моніторинг у режимі реального часу
Моніторинг наскрізних транспортних процесів у режимі реального часу	Моніторинг обслуговування та досвіду клієнтів

Розширення комунікаційних систем і мереж у контексті зростання кількості користувачів та обсягу згенерованого трафіку створює для методів моніторингу та аналізу мережевого трафіку (МАМТ) низку щоденних проблем, які включають у себе:

1. Збереження та обробка даних про трафік;
2. Використання даних про трафік для досягнення бізнес-цілей шляхом отримання інформації;
3. Інтеграція даних трафіку;
4. Перевірка достовірності даних трафіку;
5. Забезпечення безпеки даних трафіку;
6. Отримання даних про трафік.

Надзвичайне зростання кількості підключених вузлів і обсягу даних підсилює складність мережі, що потребує подальших досліджень для аналізу та контролю продуктивності мережі. Крім того, наявність великого й різноманітного обсягу даних трафіку потребує нових підходів до моніторингу та аналізу даних управління мережею. Оскільки ці проблеми є важливими, більшість досліджень

фокусується саме на одному аспекті МАМТ: виявленні аномалій та класифікації трафіку.

Отримання даних про трафік викликає великі технічні труднощі для методів моніторингу та аналізу мережевого трафіку (МАМТ), особливо для активних вимірювань. Це означає використання зондів для оцінки ключових мережевих параметрів протягом часу. Зонди представляють собою ефективний спосіб отримання уявлень про наскрізну продуктивність, відчувану кінцевими користувачами.

Активні та пасивні зонди є двома розповсюдженими стратегіями, що поліпшують продуктивність мережевого вимірювання та визначають якість обслуговування та ефективність, збираючи детальні дані про трафік. Активний зонд спробовує емулювати мережевий трафік, а потім відправляє його всередину мережі для вимірювання наскрізної продуктивності, наприклад, затримки. Пасивні зонди, натомість, розташовані на каналах мережі і пропускають увесь трафік, що проходить через контрольовані з'єднання.

Конкретні мережеві сценарії та цілі збору даних про трафік (наприклад, класифікація трафіку та виявлення вторгнень) встановлюють різні вимоги до отримання даних. Таким чином, необов'язково збирати всі доступні дані з мережі. Отже, мережеві пакети зазвичай розглядаються як основна ціль збору трафіку.

Для моніторингу мережевого трафіку та оцінки його продуктивності існують два основні методи: перевірка неглибоких пакетів і глибока перевірка пакетів. Перший збирає інформацію лише з заголовків пакетів, а другий - із всього вмісту пакета, включаючи дані користувача. Зонди можуть використовувати обидва ці методи, але глибока перевірка пакетів має свої недоліки, такі як порушення конфіденційності користувачів, більший обсяг часу та ресурсів, що потрібні для обробки пакетів, а також неможливість застосування до певних типів мережевого трафіку, наприклад, зашифрованого мережевого трафіку або віртуальних приватних мереж.

Зонди можуть використовувати обидва методи для отримання інформації про мережу. Однак глибока перевірка пакетів має свої недоліки, такі як:

1. ризик порушення конфіденційності користувачів через аналіз їх даних;
2. вимагає більше часу та ресурсів для обробки усього пакету порівняно з обробкою лише заголовка;
3. не можлива у деяких типах мережевого трафіку, наприклад, у віртуальних приватних мережах або при зашифрованому мережевому трафіку.

Враховуючи зазначені вище проблеми глибокої перевірки пакетів, більшість зондів у нових методах моніторингу та аналізу мережевого трафіку (MAMT) використовують неглибоку перевірку пакетів. На жаль, щоб вирішити цю проблему, останніми роками було запропоновано деякі інструменти. Проте до цього часу не існує адаптивного та ефективного підходу до збору даних, який можна було б широко використовувати в різнорідних і великих сучасних мережах.

Мережеві пакети залишаються основним форматом даних для збору мережевого трафіку на сьогодні. Проте оптимальні методи збору мережевих пакетів стикаються з проблемою втрати пакетів, особливо тих, що мають проблеми. Ці методи також мають складнощі у високошвидкісних каналах і можуть стати неефективними через низьку пропускну здатність. Методи збору даних на основі потоків — це інший поширений спосіб збору даних. Потокова мережа представляє собою набір мережевих пакетів з однаковими характеристиками, такими як адреса джерела/призначення та порти джерела/призначення [5]. Порівняно з іншими методами, збір даних на основі потоків може зменшити кількість необхідних завдань для аналізу пакетів та забезпечити кращу продуктивність, особливо в гігабітних мережах. Однак фільтрація пакетів і потоків може вплинути на ефективність цих методів.

Сучасні мережеві рішення постійно стикаються з впливом "великих даних". Це обумовлено особливими характеристиками даних керування мережею, такими як великий обсяг, висока швидкість, висока точність і різноманітність. Дані керування мережею охоплюють всі дані, що відображають стан мережі, переважно витягнуті із заголовків пакетів, такі як затримка пакета, часові мітки та тип пакету. Методи моніторингу та аналізу мережевого трафіку є ключовими споживачами "великих даних". Ця область також є важливою для дослідження у контексті

аналізу великих обсягів даних. Звичайні методи обробки даних для моніторингу та аналізу мережевого трафіку включають:

1. математичні та статистичні методи (наприклад, регресія для аналізу часових рядів);
2. алгоритми машинного навчання та стратегії обробки великих обсягів даних (наприклад, контрольоване навчання для виявлення вторгнень).

Методи МАМТ повинні пройти послідовність кроків для перетворення необроблених даних про трафік у корисну інформацію. Використання традиційних методів для аналізу великих даних стикається з рядом проблем, включаючи точність, швидкість аналізу та ефективну обробку великих обсягів даних у реальному часі. Крім того, з розвитком нових парадигм, таких як Інтернет речей, щоденно підключається велика кількість пристроїв, що генерують значні обсяги необроблених даних, і для їх ефективного моніторингу та аналізу потрібні більш продуктивні методології обробки даних з точки зору часу та обсягу.

Як було вказано раніше, методи машинного навчання (МН) отримали велику увагу в контексті МАМТ. Методи МН поділяються на чотири групи, а саме:

1) Навчання з наглядом;

2) Навчання з напівнаглядом;

3) Навчання без нагляду;

4) Навчання з підкріпленням.

Серед різних методів МН, глибоке навчання (ГН) виступає як ключовий етап для спрощення аналізу та отримання знань у сфері великих даних. ГН застосовується у багатьох сферах, включаючи комп’ютерне зорове сприйняття, охорону здоров’я, транспорт і розумне сільське господарство. Крім того, технологічні компанії активно зацікавлені у ГН. Великі компанії, такі як Twitter, YouTube і Facebook, щоденно генерують значні обсяги даних, і для них надзвичайно важливо обробляти ці великі потоки інформації. Алгоритми ГН застосовуються для аналізу цих даних та вилучення суттєвої інформації, оскільки традиційні методи обробки даних майже неможливо застосувати до таких великих обсягів інформації. Найважливіші напрями МАМТ включають класифікацію

трафіку, прогнозування трафіку, управління помилками та забезпечення безпеки мережі (див. рисунок 1.1).

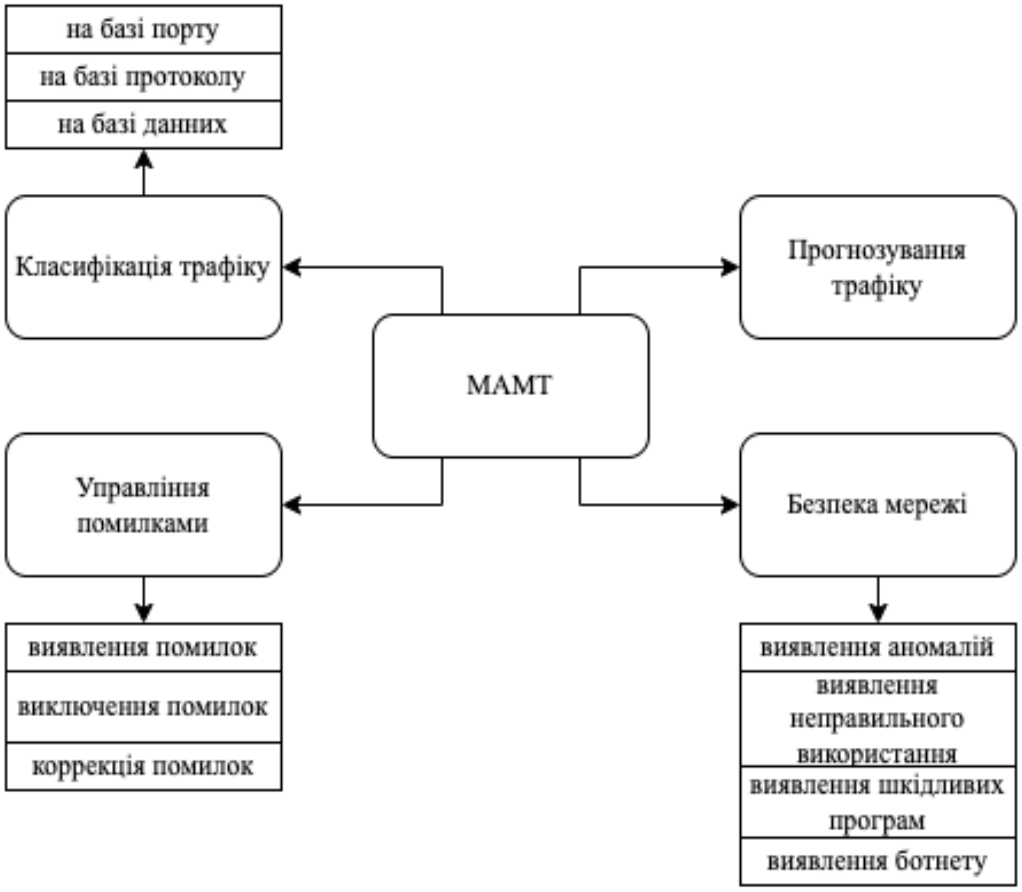

Рисунок 1.1 − Структура МАМТ

1.2 Структура методів моніторингу та аналізу мережевого трафіку

Моніторинг, аналіз і управління мережевим трафіком (МАМТ) охоплює різноманітні методи, які спостерігають за рівнем деталізації мережі, таким як дані на рівні пакетів. Методи МАМТ працюють із даними та інформацією щодо ефективності мережі, поведінки користувачів та функціонування системи в цілому. У сфері комунікаційних систем МАМТ грає важливу роль у розумінні роботи мереж, моніторингу їх продуктивності, використання ресурсів кінцевими споживачами, а також у керуванні телекомунікаційними структурами для досягнення SLA.

Оскільки кількість підключених пристроїв і обсяг трафіку стрімко зростають, настає необхідність розробки більш вдосконалених методів МАМТ, що забезпечать стабільність і доступність систем зв’язку. У наступних кроках ми розглянемо загальну структуру МАМТ, яка складається з п'яти етапів [6]. Більшість існуючих досліджень повністю або частково відповідають представленій структурі, зображеній на рисунку 1.2.

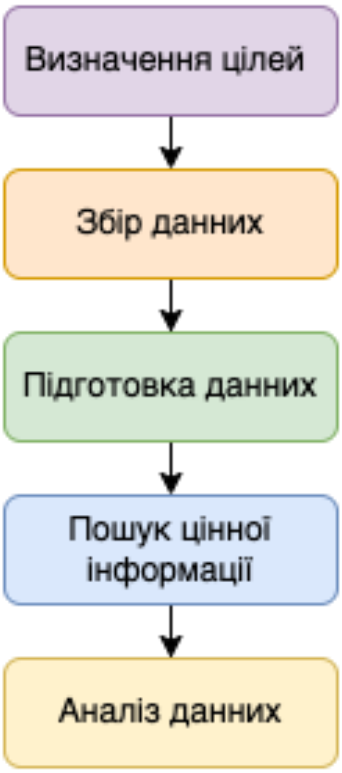

Рисунок 1.2 − Загальна структура МАМТ

Першим етапом МАМТ є чітка формулювання цілей. Як зазначалося раніше, типові мети включають у себе класифікацію трафіку, прогнозування об'єму трафіку, вирішення помилок і забезпечення безпеки мережі. Залежно від мети аналізу трафіку може виникнути необхідність працювати з різними підметами для досягнення загальної мети. Наприклад, якщо основною ціллю МАМТ є класифікація мережевого трафіку, підметом може бути категоризація трафіку на різні класи на основі їх міток, таких як VPN і не-VPN або застосунки типу Firefox і Chrome.

Другий етап включає збір даних управління мережею за допомогою пасивних або активних методів моніторингу. Оскільки ці два методи дають різні перспективи на стан мережі, їх можна комбінувати для отримання переваг обох підходів.

Обробка та очищення даних перед використанням методів аналізу мережі може суттєво вплинути на ефективність МАМТ, особливо при використанні методів машинного навчання. Видалення деяких даних керування мережею, таких як повторні передачі пакетів або подвійні підтвердження прийому (ACK), може покращити продуктивність додатків МАМТ, наприклад, при прогнозуванні трафіку.

Нормалізація є ще одним методом попередньої обробки, який допомагає поліпшити продуктивність МАМТ, особливо для підходів, що використовують методи машинного навчання та глибокого навчання.

Після передпроцесингу даних, МАМТ переходить до стадії відбору ознак, де визначаються найбільш важливі характеристики для досягнення поставленої мети. Цей вибір може бути здійснений або автоматично за допомогою алгоритмів, які виділяють найінформативніші характеристики, або вручну, використовуючи експертні знання для відбору певних ознак [7].

Після зазначених вище кроків, експерти з аналізу даних проводять глибокий аналіз передопрацьованих даних з метою отримання значущої інформації. Як було вказано в вступному розділі, математичні та статистичні методи, алгоритми машинного навчання та підходи до великих даних вважаються традиційними для видобування значущих знань із необроблених даних. Вибір належної моделі чи методики з наявних підходів є критичним для отримання надійних та відтворюваних статистичних висновків.

Підходи, що базуються на методах машинного навчання, переважають математичні та статистичні методи завдяки їх здатності розкривати приховані закономірності у вихідних даних. У сферах комунікаційних систем та мереж, методи машинного навчання використовуються в різних програмах, таких як системи виявлення вторгнень, виявлення аномалій, моніторинг та виявлення шаблонів.

1.3 Постановка задачі

Для вирішення проблем моніторингу та аналізу мережевого трафіку важливо, щоб методи та засоби відповідали поточним вимогам. У цій роботі буде проведено порівняльний аналіз класичних та машинного навчання методів моніторингу та аналізу мережевого трафіку. З різноманітних методів МАМТ будуть обрані найбільш підходящі, які здатні обробляти великі обсяги даних у реальному часі, а також легко інтегруватися у веб-орієнтовані мережі.

Система МАМТ має працювати з часово-просторовими даними та фіксувати відповідні залежності. Зокрема, моделі глибокого навчання зможуть точно аналізувати багато данних управління мережею, представлених у вигляді часових рядів. Розгортання точних та ефективних методів для різних застосунків МАМТ є критичним.

Система МАМТ отримуватиме постійний потік даних, аналізуватиме їх та автоматично виявлятиме можливі аномалії в мережевому трафіку веб-орієнтованого додатку, надаючи можливість живого моніторингу мережі.

Мета цієї кваліфікаційної роботи полягає у вивченні існуючих методів машинного навчання для прогнозування навантаження у веб-орієнтованих високонавантажених системах. Для досягнення цієї мети необхідно виконати наступні завдання:

1) Провести аналіз предметної області;

2) Вивчити та порівняти існуючі методи;

3) Розробити та описати власний підхід до прогнозування навантаження веб-орієнтованих високонавантажених систем;

4) Навчити та протестувати розроблену модель рекурентної нейронної мережі з довгою короткостроковою пам'яттю для прогнозування навантаження;

5) Проаналізувати отримані результати.

2. ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ ТА ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ МОНІТОРИНГУ ТРАФІКУ

2.1 Методи моніторингу та аналізу мережевого трафіку

Останніми роками інтерес до штучного інтелекту (ШІ) зросла завдяки його застосуванню в різних сферах, таких як самокеровані автомобілі, чат-боти, віртуальні помічники та інші. Історія розвитку ШІ починається ще з 50-х років минулого століття, коли вчені намагалися автоматизувати інтелектуальні завдання, які зазвичай виконували люди. Протягом цього періоду багато експертів вважали, що за допомогою великої кількості явних правил для обробки знань можна створити штучний інтелект, аналогічний людському. Цей підхід, відомий як символічний ШІ, був основним методом досягнення рівня інтелекту, подібного до людського, з 1950-х по кінець 1980-х років. Хоча символічний ШІ успішно справлявся з конкретними завданнями, такими як шахи, він мав проблеми з більш складними завданнями, наприклад, розпізнаванням мовлення чи класифікацією зображень. Для вирішення цих проблем виникло машинне навчання.

Машинне навчання принесло нову парадигму в програмування. У символічному ШІ людина-агент формулює правила (програму) та обробляє дані за цими правилами, отримуючи результати. У машинному навчанні людина-агент вводить дані та очікувані результати від цих даних, і потім модель навчання формує правила. Ці правила використовуються для обробки нових даних для отримання бажаних результатів. Системи машинного навчання можна навчити, а не програмувати в явному вигляді. Це означає, що в цих системах використовується велика кількість даних для виявлення ключових характеристик у цих даних. Потім ці характеристики можна використовувати для створення правил для автоматизації завдань. Однак машинне навчання часто має проблеми з обробкою великих та складних наборів даних, таких як великі масиви зображень. Для класичного статистичного аналізу, такого як байєсівський аналіз, обробка таких об'ємів даних

майже неможлива. Таким чином, машинне навчання, зокрема глибоке навчання, має більш інженерний характер і мало пов'язане з теорією математики.

Глибоке навчання є конкретною галуззю машинного навчання, в якій глибока нейронна мережа використовується для отримання представлення даних на кожному рівні. Глибоке визначення в цьому контексті стосується ідеї послідовних шарів уявлень. Наприклад, моделі глибокого навчання для завдань, таких як розпізнавання зображень, часто мають десятки або сотні послідовних шарів уявлень. У порівнянні з іншими моделями машинного навчання, які зазвичай мають один або два шари для представлення даних, архітектура глибокого навчання подана на рисунку 2.1.

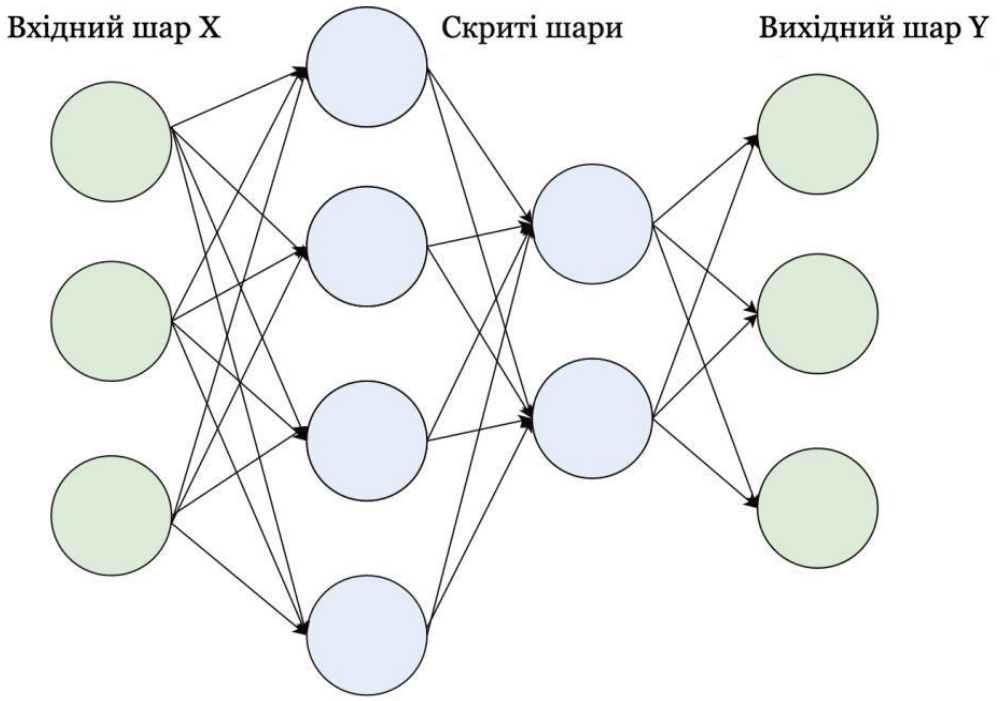

Рисунок 2.1 − Структура глибокої нейронної мережі

Загальною дефініцією можна описати машинне навчання як процес відображення початкових даних (таких як відео або зображення) на певні цільові показники (наприклад, мітка "собака") за допомогою навчання моделі на безлічі прикладів введення та цілей. Зрозуміло, що глибоке навчання (ГН) реалізує цей процес через послідовні шари глибоких перетворень даних для відображення

вхідних даних та цілей. В моделі глибокого навчання ваги кожного шару, відомі також як параметри, визначають, як саме дані будуть перетворені перед проходженням через кожен шар. У зв'язку з тим, що ці ваги представляють собою набір числових значень, навчання в глибокому навчанні полягає в пошуку оптимальних значень цих параметрів для кожного шару моделі, щоб вона ефективно відображала вхідні дані на відповідні цілі. Зважаючи на те, що моделі глибокого навчання можуть включати десятки мільйонів параметрів, пошук правильних значень для цих параметрів є складним завданням. На рисунку 2.2 представлений узагальнений зв'язок між штучним інтелектом (ШІ), машинним навчанням і глибоким навчанням. Далі ми докладніше розглянемо основні моделі глибокого навчання.

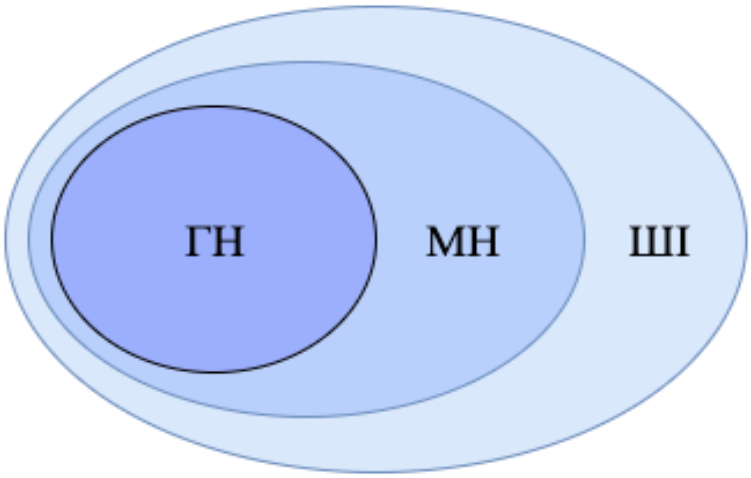

Рисунок 2.2 − Зв'язок між ШІ там ГН

2.2 Багатошаровий персептрон

Модель глибокої нейронної мережі, відома як багатошаровий персептрон, є важливим типом ГН. Вона перетворює вхідні дані у цільові значення шляхом пройдення через кілька шарів (не менше трьох). Кожен шар мережі відповідає за створення нового представлення для кожної точки даних. Головною метою цієї моделі є наближення функції f, наприклад, у випадку класифікатора $y = f(x)$, де x

є вхідними даними, а y - міткою. Багатошаровий персептрон встановлює відображення $y = f(x;\theta)$ та знаходить оптимальні значення параметрів θ, які найкраще апроксимують функцію. У цій моделі відсутній зворотний зв'язок, де вихідні дані повертаються до входу мережі. Багатошаровий персептрон має щонайменше три шари, де обчислювальні блоки щільно з'єднані з одиницями наступного шару. Ця архітектура виконує послідовну операцію обробки даних.

$$\mathbf{y} = \sigma(W*x+b), \quad (2.1)$$

У цьому виразі, y є результатом вихідного сигналу шару, W позначає ваги навчання, а b вказує на зміщення нейронів. Крім того, $\sigma()$ - це функція активації, спрямована на покращення навчання моделі шляхом додавання нелінійності. Найбільш поширеними нелінійними функціями активації є:

1) сигмовидна (логістична);

2) гіперболічний тангенс (Tanh);

3) ReLU (випрямлений лінійний блок);

4) Leaky ReLU.

Функції активації ReLU та Leaky ReLU використовуються для подолання проблеми градієнтного зникнення, яка виникає у інших функціях активації, коли градієнти функції втрачають свою інтенсивність і не можуть ефективно поширюватися через різні шари мережі.

2.3 Згорткові нейронні мережі

Згорткові мережі, також відомі як згорткові нейронні мережі, є спеціалізованим типом нейронних мереж, призначених для обробки даних, подібних до сітки. Наприклад, часові ряди та зображення вважаються такими даними, які можна інтерпретувати як одновимірну та двовимірну сітку відповідно. Згорткові мережі широко використовуються в різних реальних задачах, таких як обробка природної мови, комп'ютерний зір, розпізнавання мовлення тощо [8].

Термін "згортка" у згорткових нейронних мережах відображає використання математичної операції, відомої як згортка. У своїй базовій формі операція згортки

- це певний вид лінійної операції, яка здійснює інтеграл добутку двох функцій або сигналів. Згорткові нейронні мережі використовують оператори згортки замість загального множення матриці, принаймні на одному з рівнів їхньої мережі.

Ці мережі застосовують три ключові принципи для оптимізації роботи систем машинного навчання: обмежене використання параметрів або ваг, розріджені взаємодії та еквіваріантні уявлення.

Архітектура глибоких нейронних мереж має недолік у великій кількості параметрів, особливо коли вхідні дані є великими та складними, наприклад, зображення. Для вирішення цієї проблеми був розроблений оператор згортки (або рівень згортки) як альтернатива повному підключенню в архітектурі глибоких нейронних мереж [9]. Схематичне зображення архітектури згорткових нейронних мереж подано на рисунку 2.3.

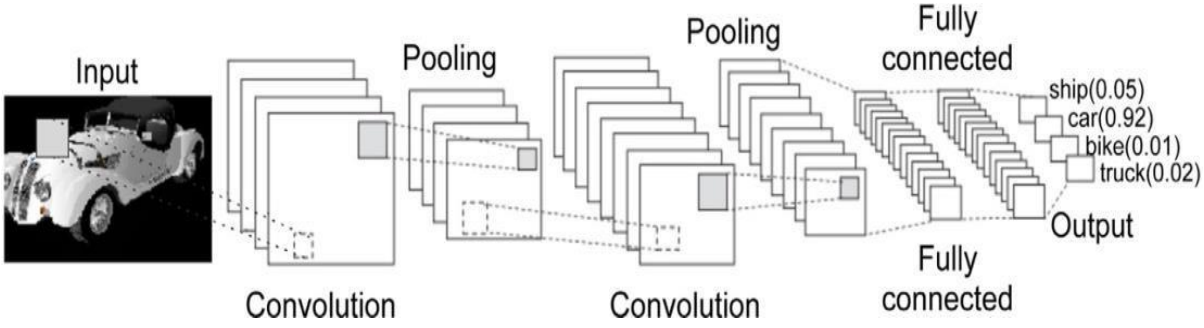

Рисунок 2.3 − Графічний опис глибокої архітектури згорткових нейронних мереж

Згорткові нейронні мережі використовують багатоканальні зображення, такі як автомобілі та кораблі, як вхідні дані для навчання. Ці мережі складаються з кількох шарів згортки, які використовують нелінійні функції активації для спрощення складності вхідних даних (зображень) та отримання вихідних результатів, таких як ймовірність приналежності кожного зображення до певного класу або категорії.

У згорткових нейронних мережах кожна зона введення пов'язана з нейроном на виході, що відомо як локальне зв'язування. Кожен шар використовує різні фільтри для розпізнавання абстрактних концепцій, наприклад, контурів

транспортного засобу. Під час роботи на більш глибоких шарах згорткова нейронна мережа може вивчати вищорівневі функції, такі як різні компоненти автомобіля.

Ці мережі не заздалегідь визначають фільтри; вони автоматично вивчають значення кожного фільтра під час навчання. Крім того, згорткові нейронні мережі використовують рівень зведення для пониження розміру. На вихідному шарі застосовується класифікатор для використання вищих функцій у завданні класифікації.

2.4 Рекурентні нейронні мережі

Рекурентні нейронні мережі (РНМ), відомі також як RNN, є категорією штучних нейронних мереж, спеціалізованих на аналізі послідовних даних [9]. Зазвичай вони використовуються для роботи з послідовностями значень $x1$,$x2$,..., xt, на відміну від згорткових нейронних мереж, призначених для роботи з даними у вигляді сітки, такими як зображення. Важливо зауважити, що більшість РНМ [10] можуть обробляти послідовності різної довжини.

Основна концепція рекурентних мереж та інших методів машинного навчання полягає в тому, щоб спільно використовувати параметри на різних рівнях моделі для того, щоб застосовувати модель до різних форм даних. Це особливо важливо, коли певний елемент даних може з'являтися в різних позиціях у послідовності. Цей підхід оптимізації, який використовує спільні параметри, часто значно економить пам'ять у моделях машинного навчання.

Рекурентні мережі можуть бути також використані для обробки двовимірних просторових даних, наприклад, зображень. Одним із ключових переваг використання рекурентних мереж є їх здатність обробляти послідовні дані так, що кожен зразок можна розглядати у залежності від попередніх.

Рекурентні нейронні мережі спеціалізовані на моделюванні послідовностей [11], де між зразками існує висока послідовна взаємодія. На кожному кроці часу РНМ використовує вхідні дані та інформацію, яка стосується попередніх станів, для генерації вихідних даних. Ця інформація передається через зворотні зв'язки між блоками, як показано на рисунку 2.4.

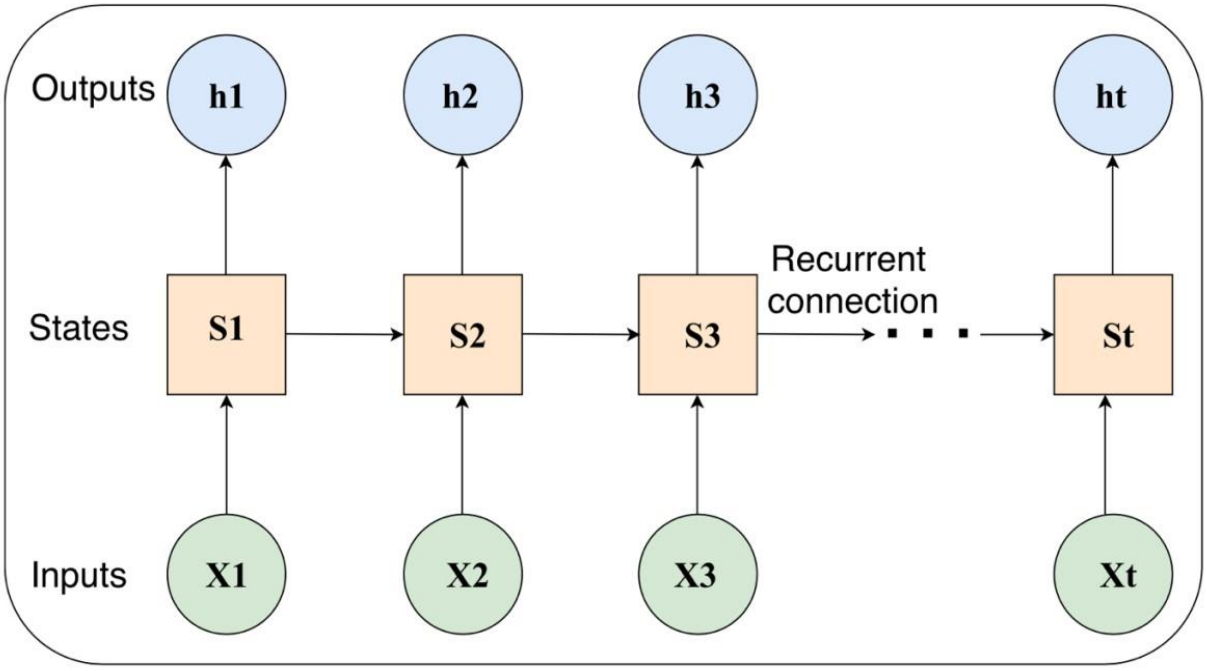

Рисунок 2.4 − Рекурента нейронна мережа

Припустимо, що ми маємо послідовність вхідних елементів **x** = ($x1$,$x2$,...,xt). За цього параметра РНМ проводить обчислення зображені в формулі 2.2 та формулі 2.3.

$$St = \sigma s(Wxxt + Wsst{-}1 + bs), \qquad (2.2)$$

$$\mathrm{h}t = \sigma \mathrm{h}(W\mathrm{h}st + b\mathrm{h}), \qquad (2.3)$$

На етапі часу t, St визначає стан РНМ, що виступає як пам'ять для цієї мережі. Щоб вирахувати значення St, потрібно врахувати функцію вхідного значення xt в момент часу t, а також попередній стан РНМ, а саме $st{-}1$. Крім того, Wx і $W\mathrm{h}$ - це ваги, які підлягають навчанню під час тренування мережі, тоді як bs і $b\mathrm{h}$ - це параметри зміщення. Для оновлення ваг та тренування мережі використовується алгоритм зворотного поширення через час у РНМ.

2.5 Моделі довгої короткочасної пам’яті (LSTM мережа)

Рекурентні нейронні мережі (РНМ) можуть використовувати цикли для збереження градієнта, який описує останні події введення на протязі тривалого часу. Це базова концепція моделі довгострокової пам'яті (LSTM). Ця функція є ключовою для різноманітних сфер застосування, включаючи розпізнавання мовлення, рукописний ввід, машинний переклад, створення рукописного тексту, обробку зображень та синтаксичний аналіз. LSTM було розроблено для вирішення двох основних проблем, що існували в попередніх техніках: проблеми зникнення та вибуху градієнта. Використання звичайних методів навчання на основі градієнта, таких як BPTT і RTRL, може призводити до зменшення або зростання сигналів помилок під час їх зворотнього поширення через мережу. Мережа LSTM пропонує рішення цих проблем за допомогою впровадження механізму шлюзів. LSTM успішно використовується в таких областях, як розпізнавання мовлення та класифікація тексту [12]. Структура LSTM ілюструється на рисунку 2.5.

У цій структурі, "forget gate" визначає, яку інформацію зі стану клітинки слід забути, оскільки вона може бути непотрібною. Фактично, "forget gate" приймає це рішення через сигмоїдний шар. Операція "forget gate" виконується відповідно до формули 2.4.

$$ft = \sigma(WxfXt + W\text{h}fHt{-}1 + Wcf \odot Ct{-}1 + bf), \qquad (2.4)$$

У цьому виразі операція "⊙" позначає Адамара або поелементний добуток, де Ct відображає вихідні дані стану комірки, а Ht позначає приховані стани. Механізм forget gate спрямований на пом'якшення проблем зникнення або зростання градієнту, значно підвищуючи ефективність LSTM в порівнянні з RNN.

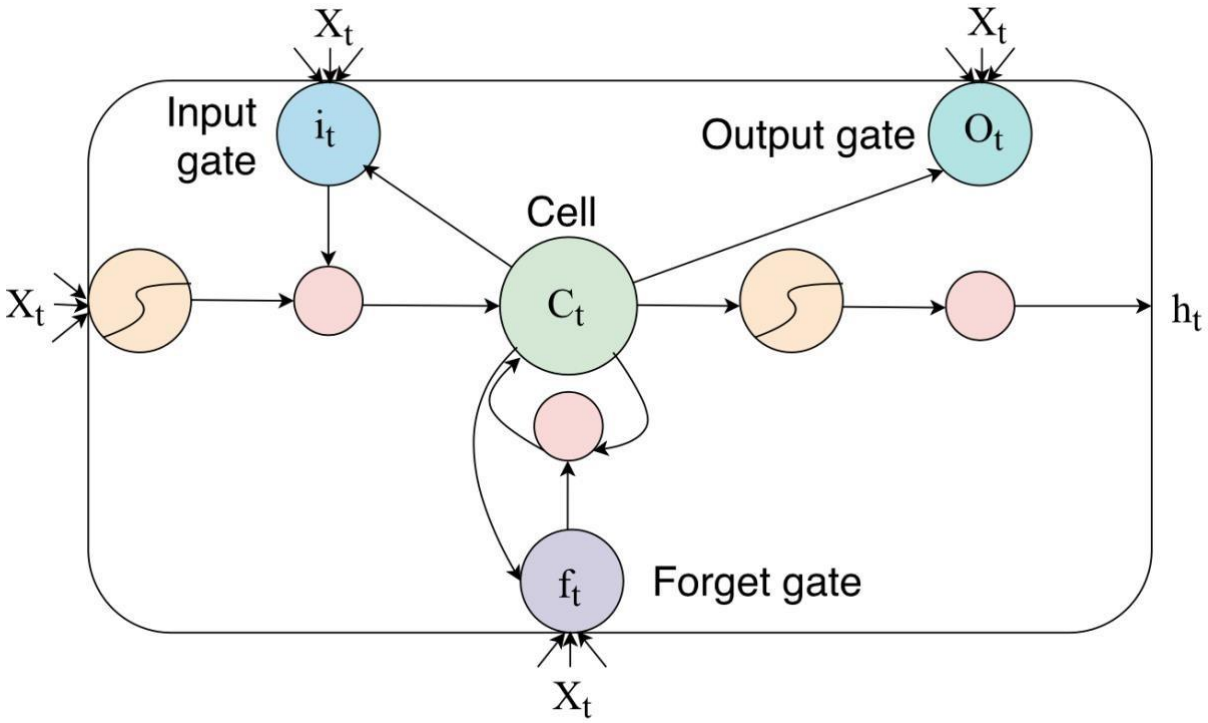

Рисунок 2.5 – Внутрішня структура LSTM

Ще однією ключовою функцією LSTM є вирішення, яку нову інформацію слід зберігати в стані клітинки [12]. З цією метою вхідні ворота it (2.5) та (2.6) визначають, яка інформація буде оновлена, і ці дані допомагають в актуалізації попереднього стану клітинки (Ct−1).

$$it = \sigma(WxiXt + W\text{hi}Ht-1 + Wci \odot Ct-1 + bf), \quad (2.5)$$

$$Ct = ft \odot Ct-1 + it \odot tan\text{h}(WxcXt + W\text{hc}Ht-1 + bc), \quad (2.6)$$

Останнім етапом для LSTM є визначення того, що слід вивести, на основі стану клітинки. Це можна здійснити за допомогою вихідних вентилів (2.7) та (2.8) (тобто ot), які вирішують, яка інформація про стан комірки буде передана на виход. Стан комірки також проходить через функцію tanh, після чого множиться на вихідні вентилі.

$$ot = \sigma(WxoXt + W\text{ho}Ht-1 + Wco \odot Ct + bo), \quad (2.7)$$

$$Ht = ot \odot tan\text{h}(Ct). \quad (2.8)$$

2.6 Глибоке навчання для моніторингу та аналізу трафіку в мережі

Методи машинного навчання, зокрема алгоритми Глибокого Навчання (ГН), наразі стали одними з найбільш популярних засобів для аналізу та обробки даних мережевого трафіку. Це може бути пояснено через властивості сучасних комунікаційних систем та мереж, таких як Інтернет речей та мобільні мережі, що ідеально відповідають принципам алгоритмів ГН. Ці особливості включають великі обсяги даних, складність, мультимодальні дані, масштабність та зростаючу кількість протоколів в таких мережах. Традиційні методи моніторингу та аналізу мережевого трафіку мають свої обмеження, такі як недостатня точність або сильна залежність від експертів. Навпаки, методи, що ґрунтуються на ГН, мають деякі переваги [13], які можуть бути використані у застосунках Методів Аналізу Мережевого Трафіку (МАМТ).

Моделі, які базуються на ГН, не потребують великих зусиль людини та не залежать від вибору конкретних характеристик. Вони можуть використовувати різні репрезентативні шари та ефективні алгоритми для вилучення прихованих знань з великого обсягу даних про трафік, без необхідності виокремлення конкретних характеристик. Ця перевага моделей ГН є дуже корисною для методів МАМТ, оскільки більшість даних управління мережею не має явних міток або є напівмаркованими. Моделі ГН можуть ефективно працювати з часово-просторовими даними [13], виявляючи відповідні залежності. Це особливо важливо для аналізу даних управління мережею, які зазвичай представлені у вигляді наборів часових рядів.

Застосування точних та ефективних методів для різних додатків МАМТ має велике значення. Наприклад, точне передбачення мобільного трафіку є критичним для оптимізації розподілу ресурсів (наприклад, розподіл на вимогу), економії енергії та аналізу мобільності користувачів у стільникових мережах.

Реалізація моделей ГН за допомогою нових парадигм машинного навчання дозволяє навчати їх на кожній окремій машині [13]. Це є великою перевагою, оскільки методи МАМТ повинні збирати дані керування мережею з різних машин для подальшого аналізу. Використання розподіленого машинного навчання дозволяє моделям ГН навчатися на кожній окремій машині, що зменшує навантаження на мережу та мінімізує ризик порушень безпеки та конфіденційності.

2.7 Проектування системи та інструменти для її розробки

Ми плануємо розробити систему, яка буде працювати відповідно до сучасної архітектури клієнт-сервер. Архітектура клієнт-сервер полягає у способі організації комп'ютерної мережі, де численні клієнти (віддалені процесори) звертаються за послугами до централізованого сервера (хост-комп'ютера). Клієнтські комп'ютери створюють інтерфейс, що дозволяє користувачам клієнта надсилати запити до сервера та відображати результати, повернені сервером. Сервери очікують запитів від клієнтів і відповідають на них. Ідеально, сервер надає клієнтам стандартний та прозорий інтерфейс, щоб клієнти не повинні були розуміти особливості системи (наприклад, апаратне чи програмне забезпечення), що надає послуги. Клієнти часто розташовані на робочих станціях або особистих комп'ютерах, в той час як сервери знаходяться в інших частинах мережі [14], як правило, на більш потужних машинах. Ця модель обчислень особливо ефективна, коли клієнти та сервер мають виконувати різні завдання, які вони регулярно виконують. Наприклад, під час обробки медичних даних на клієнтському комп'ютері може працювати додаткова програма для введення інформації про пацієнта, тоді як на серверному комп'ютері запущена програма для керування базою даних, де постійно зберігається інформація. Багато клієнтів можуть одночасно отримувати доступ до інформації на сервері [15], а також, клієнтські комп'ютери можуть виконувати інші завдання, наприклад, відправляти електронну пошту. Оскільки як клієнтські, так і серверні комп'ютери вважаються незалежними пристроями, модель клієнт-сервер відрізняється від старої моделі мейнфрейму [16], де централізований мейнфрейм виконував всі завдання для пов'язаних з ним "німих" терміналів, які лише

спілкувалися з центральним мейнфреймом. Приклад клієнт-серверної архітектури зображений на рисунку 2.6.

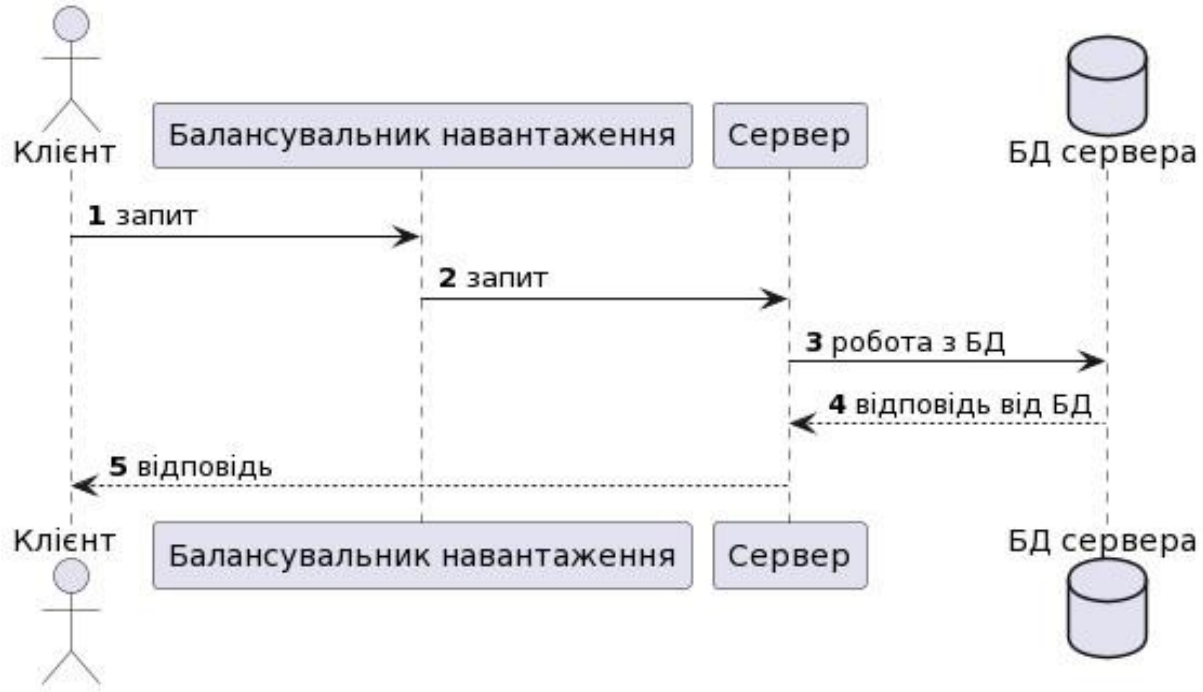

Рисунок 2.6 – Клієнт-серверна архітектура

У сучасних системах клієнт-сервера широко використовується балансувальник навантаження. Цей пристрій, що виступає як зворотний проксі-сервер [17], розподіляє мережевий або програмний трафік між різними серверами. Балансувальники навантаження застосовуються для збільшення обсягу (кількості одночасних користувачів) та надійності програм. Вони сприяють покращенню загальної ефективності додатків, розподіляючи навантаження на сервери, що відповідають за управління та підтримку сеансів програм та мережі, а також виконання завдань, що стосуються програми.

Зазвичай балансувальники навантаження поділяють на дві основні категорії [18]: рівень 4 і рівень 7. Балансувальники рівня 4 працюють на основі даних, отриманих з протоколів мережевого та транспортного рівнів (IP, TCP, FTP, UDP). А балансувальники рівня 7 розподіляють запити на основі інформації, що міститься в протоколах прикладного рівня, таких як HTTP.

Щоб обробити дані, спочатку їх потрібно записати. З цією метою ми розширимо схему з рисунку 2.5, додавши додаток для фіксації запитів. Удосконалену архітектуру можна побачити на рисунку 2.7.

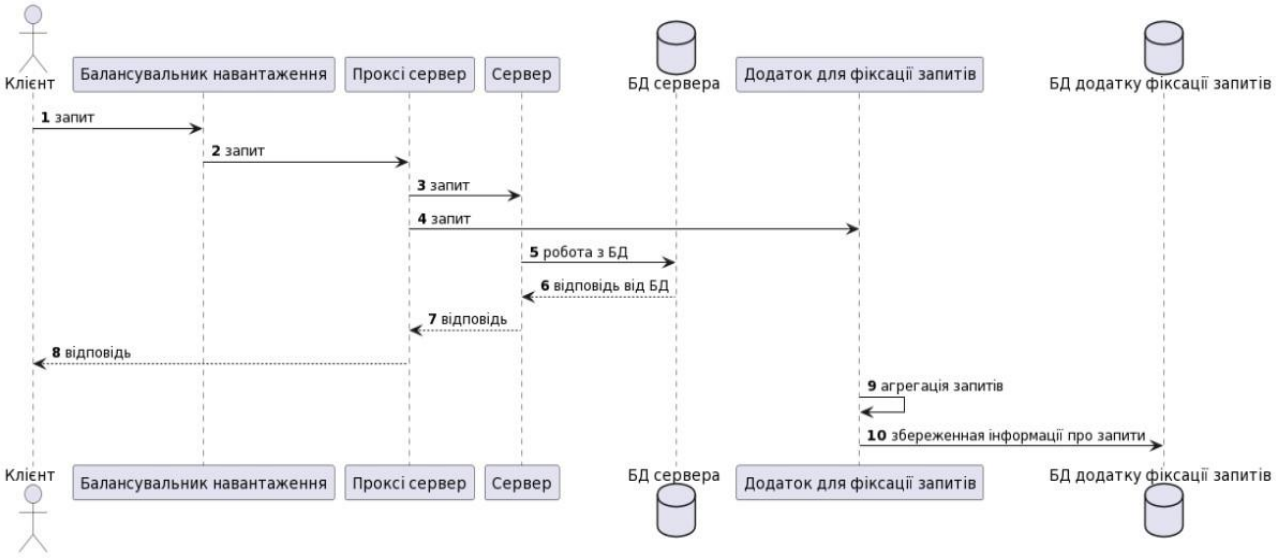

Рисунок 2.7 – Модернізована клієнт-серверна архітектура

У модернізованій структурі, запит від балансувальника навантаження буде надсилатися до проксі-сервера, який, у свою чергу, буде перенаправляти запити як до основного сервера нашого додатку, так і до сервера для фіксації запитів. Такий підхід допоможе уникнути зайвого навантаження на основному сервері. Для цієї ролі можна використовувати Nginx, який функціонує як HTTP-сервер та зворотний проксі-сервер [19], а також як TCP/UDP проксі-сервер загального призначення, здатний працювати при великому навантаженні.

У створенні додатку для фіксації запитів краще використовувати мову програмування Go. Go – це статично типізована, компільована мова, розроблена у Google Робертом Гріземером, Робом Пайком і Кеном Томпсоном. Вона відмінно підходить для обробки великого обсягу мережевого трафіку, завдяки своїм механізмам паралелізму, що дозволяють писати програми, оптимізовані під багатоядерні та мережеві системи [19]. Крім того, її нова система типів забезпечує гнучку та модульну конструкцію програм. Go швидко компілюється в машинний код та може бути ефективною альтернативою як статично типізованим, так і динамічно типізованим мовам.

Наш додаток буде зберігати інформацію про запити в базі даних. Для цих цілей ми плануємо використовувати СУБД PostgreSQL. PostgreSQL – це передова реляційна база даних з відкритим кодом корпоративного класу [20], яка підтримує запити SQL (реляційні) та JSON (нереляційні). Ця система управління базами даних володіє високою стійкістю та цілісністю, завдяки більш ніж 20-річному розвитку громади. PostgreSQL застосовується як основне сховище даних або частина сховища для різних веб-сайтів, мобільних додатків, систем геолокації та аналітичних програм. Остання версія PostgreSQL – 14.

Підтримка різноманітних даних PostgreSQL [20] і підтримка JSON роблять її зручною для взаємодії з іншими типами сховищ даних, включаючи системи NoSQL, і використовувати її як центральний збірник даних. Оскільки очікується велика кількість запитів, отримані дані будуть агреговані в додатку та зберігатимуться в базі даних. Схему бази даних можна побачити на рисунку 2.8.

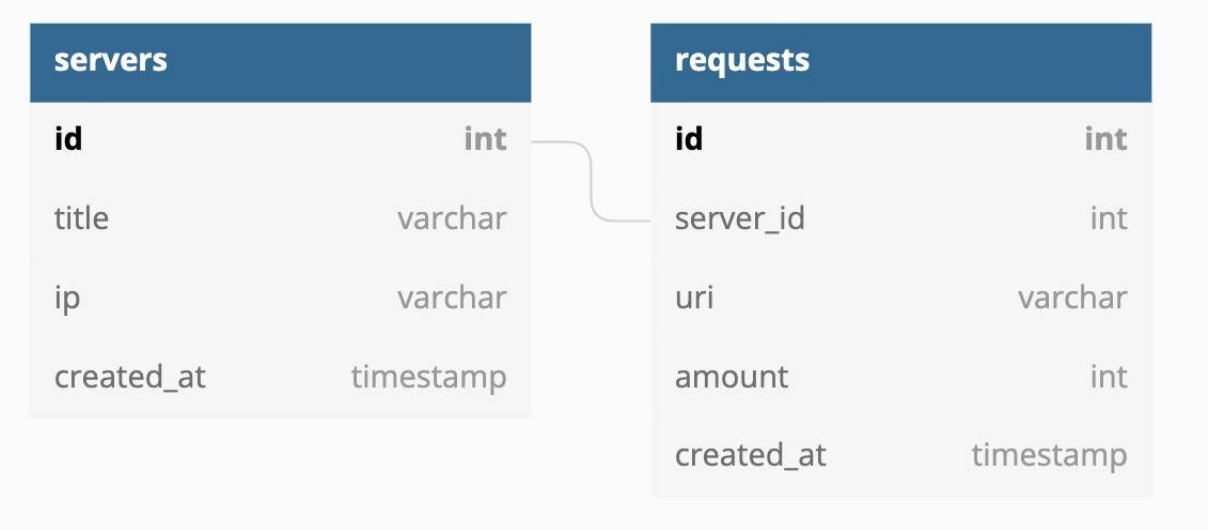

Рисунок 2.8 – Схема БД

У таблиці "servers" буде зберігатися наступна інформація про сервери:

1) Id – унікальний ідентифікатор сервера;
2) Title – псевдонім або назва, що використовується для сервера;
3) Ip – IP-адреса сервера;
4) Created_at – дата, коли запис був доданий до бази даних.

У таблиці "requests" буде міститися інформація про запити до серверів:

1) Id – унікальний ідентифікатор запису;

2) Server_id – зовнішній ключ, який посилається на таблицю "servers" та відображає зв'язок один-до-багатьох;

3) URI – уніфікований ідентифікатор ресурсу;

4) Amount – кількість запитів до ресурсу з останньої фіксації;

5) Created_at – дата, коли запис був доданий до бази даних.

Наступним кроком в архітектурі буде розгорнутий додаток для аналізу та передбачення навантаження на сервер. Його подальше розширення можна побачити на рисунку 2.9, яке продовжує схему зображену на рисунку 2.7.

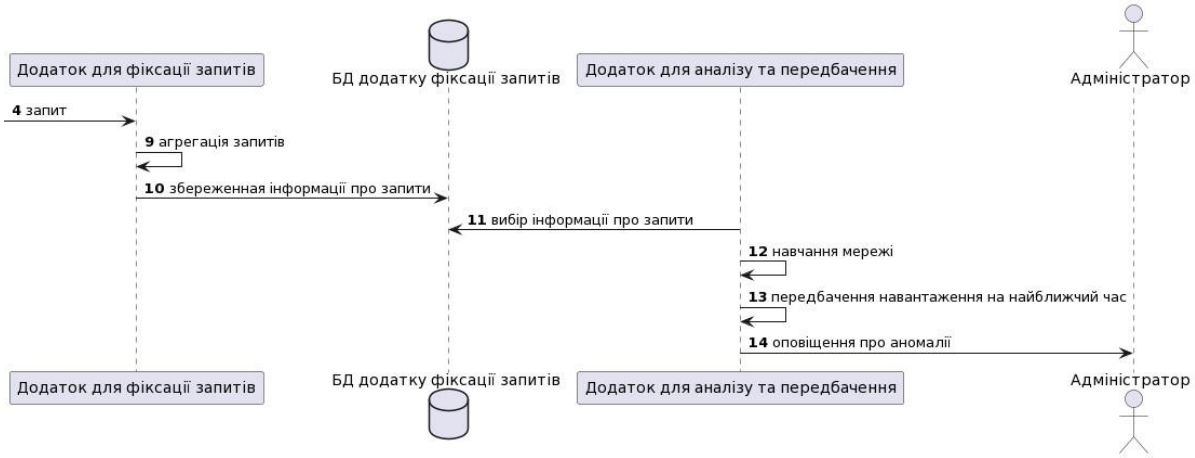

Рисунок 2.9 – Архітектура додатку для аналізу та прогнозування навантаження

Для створення програми для аналізу та прогнозування навантаження на сервери було вибрано мову програмування Python. Python - це високорівнева мова загального призначення з динамічною строгою типізацією та автоматичним управлінням пам'яттю. Ця мова спрямована на полегшення розробки, зрозумілість коду, його якість і переносимість написаних програм. Вона повністю об'єктно-орієнтована, де все представлено у вигляді об'єктів. Відмінністю Python є використання пробільних відступів для виділення блоків коду, що робить синтаксис мінімалістичним і майже не потребує звертання до документації. Мова є інтерпретованою та використовується для написання скриптів. Втім, має нижчу

швидкість роботи та вище споживання пам'яті у порівнянні з компільованими мовами, такими як C або C++.

Python є мультипарадигмальною мовою програмування, яка підтримує імперативне, процедурне, структурне, об'єктно-орієнтоване, метапрограмування та функціональне програмування. Аспектно-орієнтоване програмування підтримується через декоратори, а повну підтримку можна забезпечити за допомогою додаткових фреймворків та бібліотек. Важливі характеристики включають динамічну типізацію, автоматичне управління пам'яттю, повну інтроспекцію, механізм обробки виключень, підтримку багатопоточних обчислень з глобальним блокуванням інтерпретатора (GIL), високорівневі структури даних, розбиття програм на модулі та їх об'єднання в пакети.

Для створення сприятливого середовища проекту та для швидкого розширення як самого проекту, так і команди розробників, буде використовуватися Docker.

Docker - це відкрита платформа для розробки, доставки та запуску програм. Вона дозволяє ізолювати програми від інфраструктури для швидкої доставки програмного забезпечення. Docker забезпечує можливість керувати інфраструктурою такими ж способами, якими керуються програми. Використовуючи Docker для тестування та швидкого розгортання коду, можна значно зменшити затримки між написанням коду та його виконанням на виробництві. Контейнери Docker дозволяють упаковувати та запускати програми в ізольованому середовищі, званим контейнером, що дозволяє запускати багато контейнерів на заданому хості та зменшує навантаження гіпервізора.

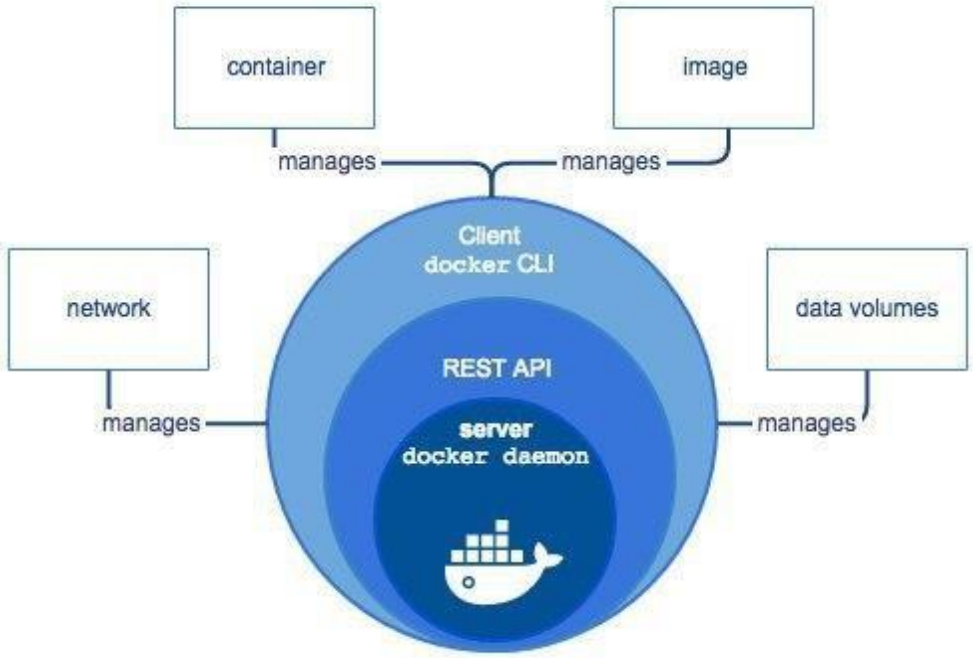

Рисунок 2.10 – Структура докеру

Для зберігання та управління різними версіями коду ми плануємо використовувати Git. Git є розповсюдженою системою управління версіями з відкритим кодом, що надає можливість ефективного та оперативного керування різними проектами — від невеликих до дуже об'ємних.

3. РЕАЛІЗАЦІЯ АЛГОРИТМІВ ТА ПОРІВНЯЛЬНИЙ АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

3.1 Підготовка тестових даних одиничного запиту

Після того, як дані будуть зібрані за допомогою нашого додатку для фіксації запитів, ми отримаємо тестові дані для одного з API ендпоінтів, зразок якого показано на рисунку 3.1. Фіксація зібраних даних проводиться кожну 1 хвилину.

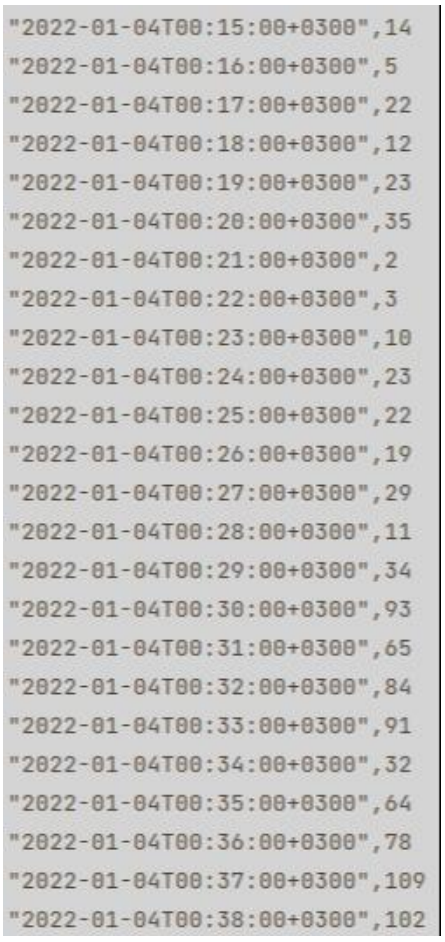

Рисунок 3.1 – Приклад даних

Ми використовуємо просту програму на Python для візуалізації тестових даних. Приклад коду програми наведено на рисунку 3.2.

```python
import pandas
import matplotlib.pyplot as plt

if __name__ == '__main__':
    dataset = pandas.read_csv('/test/path/requests.csv', usecols=[1], engine='python')
    plt.plot(dataset)
    plt.show()
```

Рисунок 3.2 – Код для візуалізації тестових даних

Результатом виконання програми зображеної на рисунку 3.2 є рисунок 3.3, на якому по осі X відображено час у хвилинах, а по осі Y – кількість запитів, отриманих сервером за кожну хвилину.

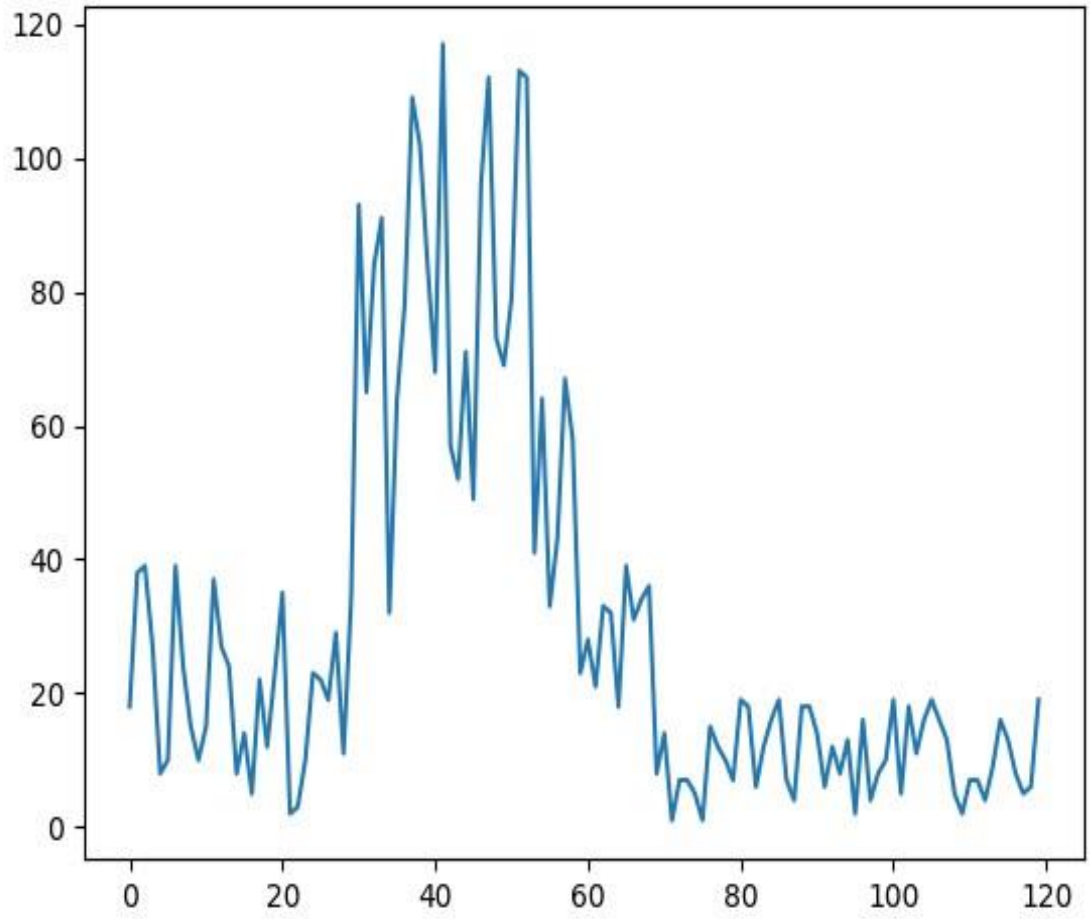

Рисунок 3.3 – Візуалізація тестових даних

3.2 Реалізація LSTM мережі для одиничного запиту

Мережі LSTM чутливі до масштабу вхідних даних, особливо, коли застосовуються сигмоподібні (за замовчуванням) або tanh функції активації. Доброю практикою може стати масштабування даних до діапазону від 0 до 1, відоме як нормалізація. Нормалізацію можна легко здійснити за допомогою класу попередньої обробки MinMaxScaler з бібліотеки scikit-learn. Приклад відповідного коду наведено на малюнку 3.4.

```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

Рисунок 3.4 – Нормалізація даних

Після того, як ми зробимо моделювання наших даних та оцінимо ефективність нашої моделі на навчальному наборі даних, ми потребуємо оцінити навички моделі на нових, невидимих даних. Для стандартної задачі класифікації або регресії можна було б використовувати перехресну перевірку. Однак для часових рядів важлива їхня послідовність. Простий метод, який можна використати, - це розбити впорядкований набір даних на навчальний та тестовий набори. У наведеному нижче коді на малюнку 3.5 обчислюється індекс точки розділення, розбиваючи дані на навчальний набір (67% спостережень, які можна використовувати для навчання моделі) і залишаючи решту 33% для тестування моделі.

```
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]
```

Рисунок 3.5 – Розділення даних на тестову та тренувальну вибірку

Тепер ми можемо визначити функцію для створення нового набору даних. Варіант реалізації цієї функції можна побачити на рисунку 3.6. Функція приймає

два аргументи: набір даних, який представлений у вигляді масиву NumPy та який ми хочемо перетворити у новий набір даних, і також look_back, що визначає кількість попередніх кроків часу для використання як вхідних змінних для прогнозування наступного періоду часу - за замовчуванням це значення 1.

За замовчуванням ця функція створює набір даних, де X представляє кількість запитів у певний момент часу (t), а Y - кількість запитів у наступний момент часу (t + 1).

```
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

Рисунок 3.6 – Функцію для створення нового набору даних

Тепер ми використаємо цю функцію для підготовки навчальних та тестових наборів даних для моделі. Приклад відповідного коду наведено на рисунку 3.7.

```
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

Рисунок 3.7 – Підготовка навчальної та тестової вибірки

Модель LSTM має очікування щодо структури вхідних даних (X), де очікується формат масиву [зразки, кроки часу, ознаки]. У нас наразі дані мають формат [зразки, кроки часу], тому ми представляємо проблему як один часовий крок для кожного зразка. Щоб змінити підготовлені навчальні та тестові вхідні дані до очікуваної структури, ми можемо використати метод numpy.reshape(), який показано на рисунку 3.8.

```
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

Рисунок 3.8 – Підготовка навчальної та тестової вибірки

Тепер ми можемо розробити та налаштувати нашу мережу LSTM для вирішення цієї задачі. Структура мережі включає один видимий шар з 1 вхідним вузлом, прихований шар з 4 блоками або нейронами LSTM та вихідний шар, що здійснює передбачення одного значення. Для блоків LSTM використовується стандартна функція активації у формі сигмоїди. Мережа навчається протягом 100 епох, використовуючи пакетний розмір 1. Приклад коду наведено на рисунку 3.9.

```
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

Рисунок 3.9 – Реалізація моделі

Після завершення навчання моделі ми зможемо провести оцінку її ефективності на тренувальному та тестовому наборах даних. Це дозволить нам мати точку порівняння для нових моделей.

Важливо відзначити, що перед оцінкою помилок передбачень ми здійснюємо інверсію, щоб переконатися, що метрики вимірюються в тих самих одиницях, що й вихідні дані (кількість запитів за хвилину). Приклад коду представлений на рисунку 3.10.

```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
print('Test Score: %.2f RMSE' % (testScore))
```

Рисунок 3.10 – Перевірка точності моделі

Тепер ми можемо використовувати модель для прогнозування як на навчальному, так і на тестовому наборі даних для візуального оцінювання її ефективності. Оскільки дані були підготовлені, необхідно зсунути прогнози, щоб вони коректно відображалися вздовж осі x порівняно з вихідними даними. Після цього можна відобразити дані на графіку: вихідний набір даних відображається синім кольором, прогнози для навчального набору - помаранчевим, а прогнози для тестового набору - зеленим. Код візуалізації результатів представлений на рисунку 3.11, а отримані графіки демонструються на рисунку 3.12. Інтерпретація результатів показує, що модель працює добре як на навчальній, так і на тестовій вибірці даних. Середня похибка моделі складає близько 20 запитів у навчальному наборі та приблизно 9 запитів у тестовому наборі.

```
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict) + look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict) + (look_back * 2) + 1:len(dataset) - 1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Рисунок 3.11 – Код візуалізації результатів

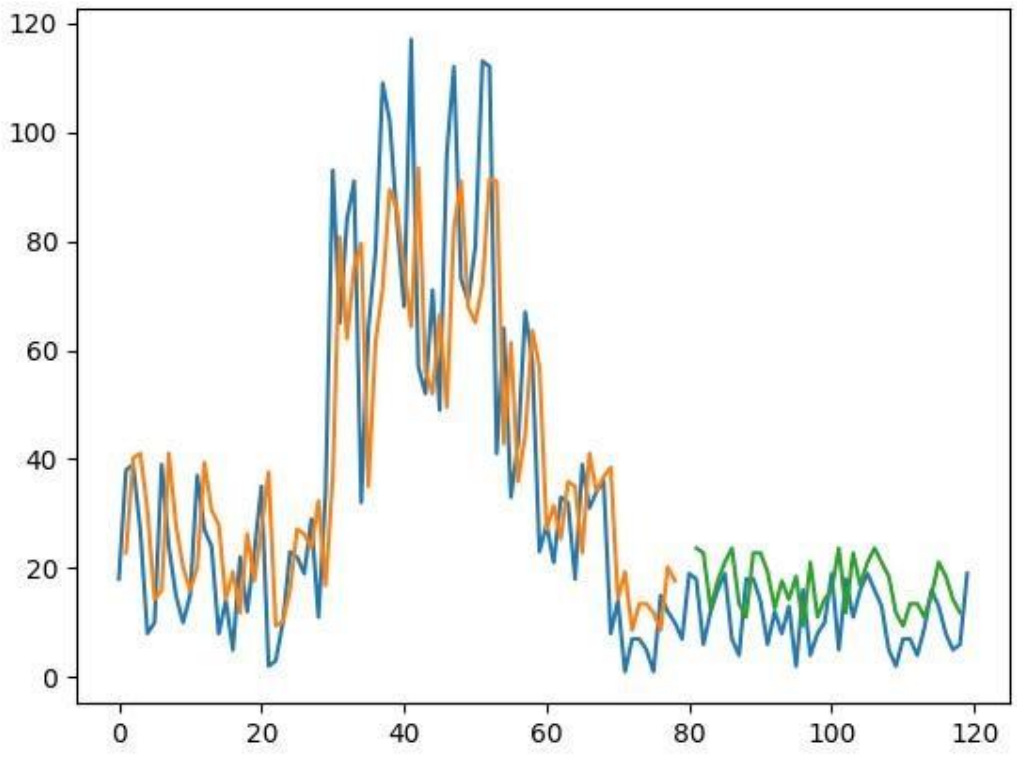

Рисунок 3.12 – Результат візуалізації

Ми також можемо сформулювати задачу таким чином, щоб використовувати не лише один, а декілька останніх часових кроків для прогнозування майбутнього. Це концепція вікна, і розмір цього вікна є параметром, який можна налаштувати для кожної конкретної задачі. Наприклад, якщо ми бажаємо передбачити значення наступного часу (t+1) на основі поточного часу (t), можемо використати також два попередні моменти (t-1 і t-2) як вхідні змінні.

Якщо це формулювати як задачу регресії, то вхідні змінні - t-2, t-1, t, а вихідна - t+1. Функція create_dataset(), яку ми раніше створили, дозволяє нам сформулювати таку проблему з часовими рядами, збільшивши значення аргументу look_back з 1 до 3.

У такій конфігурації можна спостерігати, що модель має середню похибку близько 20 запитів у навчальному наборі даних і приблизно 6 запитів у тестовому наборі. Графічне відображення роботи моделі можна побачити на рисунку 3.13.

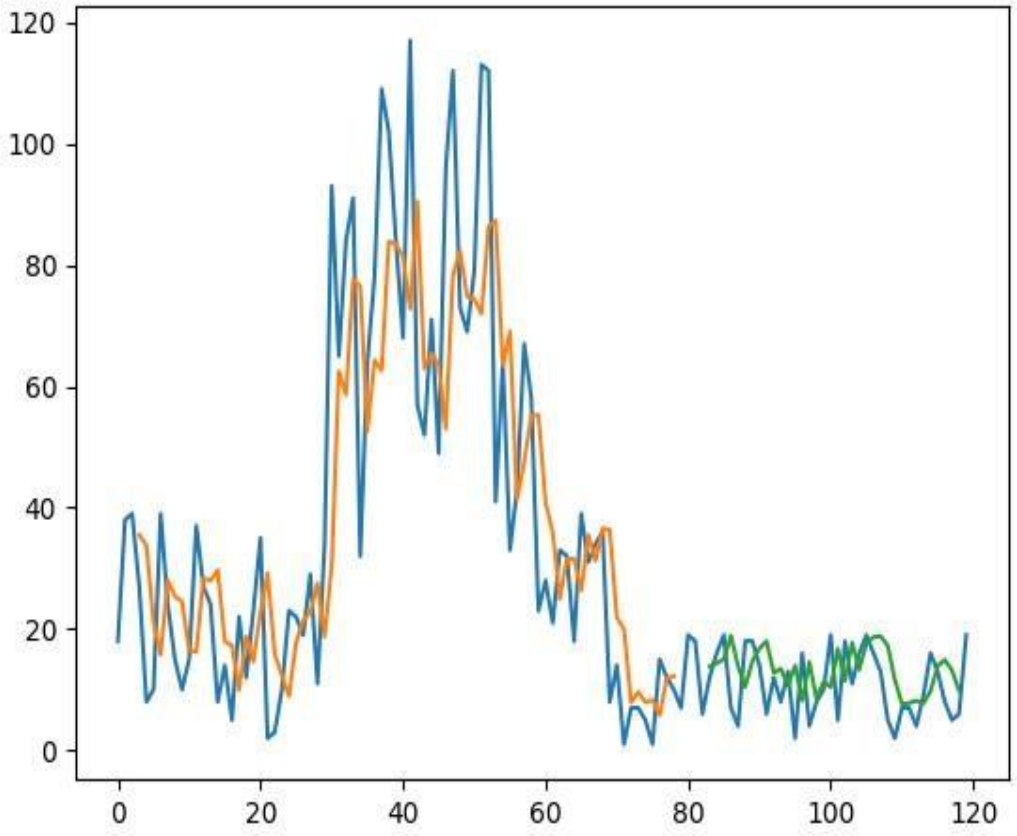

Рисунок 3.13 – Результат візуалізації

Мережа LSTM має властивість пам'яті, що дозволяє їй зберігати інформацію про довгі послідовності. Зазвичай, стан мережі очищується після кожного навчального пакету під час встановлення моделі та під час кожного виклику model.predict() або model.evaluate().

У Keras можна краще керувати очищенням внутрішнього стану мережі LSTM, створюючи шар LSTM зі збереженням стану. Це дозволяє створювати та зберігати стан протягом всієї навчальної послідовності, якщо це необхідно для прогнозування.

Для цього важливо, щоб навчальні дані не перемішувалися при встановленні мережі. Також потрібно явно очищати стан мережі після кожної епохи навчання через model.reset_states(). Це означає, що треба створити власний зовнішній цикл епох та всередині кожної епохи викликати model.fit() і model.reset_states(). Приклад такого коду зображено на рисунку 3.14.

```
for i in range(100):
    model.fit(trainX, trainY, epochs=1, batch_size=batch_size, verbose=2, shuffle=False)
    model.reset_states()
```

Рисунок 3.14 – Модифіковане навчання моделі

При налаштуванні шару LSTM необхідно встановити параметр stateful на значення True. Замість того, щоб явно вказувати вхідні розміри, необхідно жорстко прописати кількість зразків у пакеті, кількість часових кроків у вибірці і кількість функцій за кожен часовий крок, встановивши параметр batch_input_shape. Такий самий розмір має бути використаний під час прогнозування. У цій конфігурації модель показує середню похибку приблизно 15 запитів у навчальному наборі даних та близько 8 запитів у тестовому наборі. Візуалізацію роботи моделі можна побачити на рисунку 3.15.

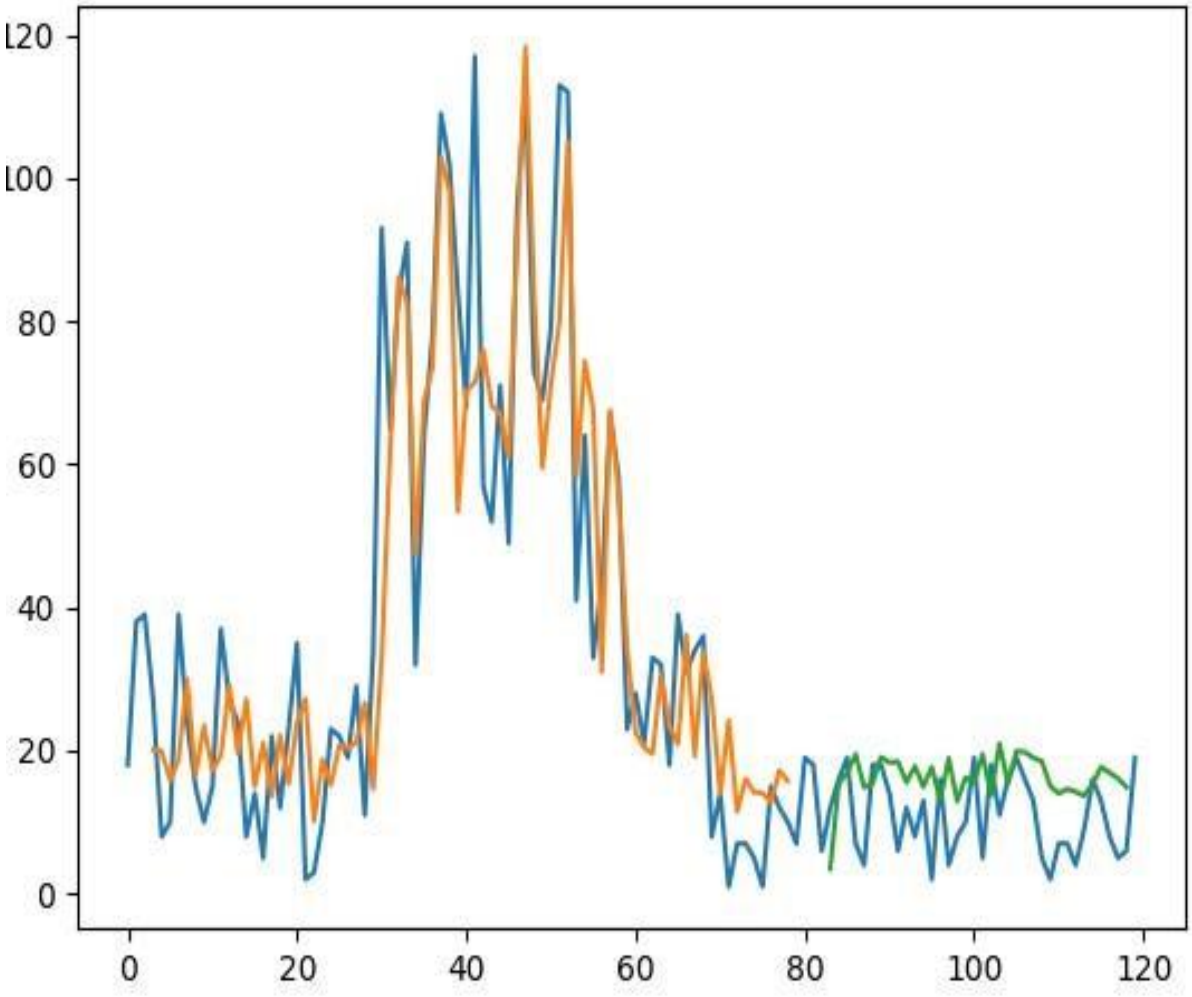

Рисунок 3.15 – Результат візуалізації

3.3 Підготовка тестових даних групи запитів

У цьому розділі ми дослідимо групу запитів, які були зареєстровані. Інформація про запити збирається кожну хвилину, і ми маємо п'ять різних типів запитів, що були зафіксовані. На рисунку 3.16 наведено приклад тестових даних, які будуть використовуватися для моделювання. А візуалізація цих даних представлена на рисунку 3.17.

```
"2022-01-04T02:02:00+0300",82,3,49,22,8
"2022-01-04T02:03:00+0300",78,5,48,20,2
"2022-01-04T02:04:00+0300",68,4,54,23,1
"2022-01-04T02:05:00+0300",68,6,43,24,7
"2022-01-04T02:06:00+0300",80,10,41,15,2
"2022-01-04T02:07:00+0300",70,18,36,18,7
"2022-01-04T02:08:00+0300",65,7,36,19,4
"2022-01-04T02:09:00+0300",76,7,48,17,3
"2022-01-04T02:10:00+0300",70,18,40,16,6
"2022-01-04T02:11:00+0300",65,25,42,15,3
"2022-01-04T02:12:00+0300",64,2,33,24,3
"2022-01-04T02:13:00+0300",75,5,47,16,5
```

Рисунок 3.16 – Приклад даних для групи запитів

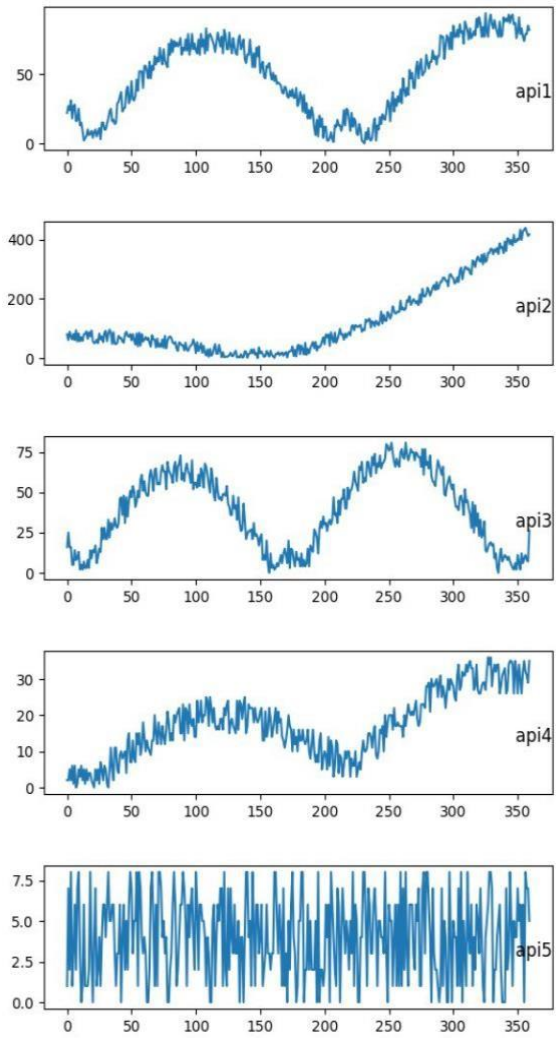

Рисунок 3.17 – Візуалізація групи запитів

3.4 Реалізація LSTM мережі для групи запитів

Ми почнемо реалізацію LSTM мережі для групи запитів, підготувавши дані: нормалізувавши їх, адаптувавши часові ряди для використання у мережі та розділивши вибірку на навчальну та тестову частини. Код, що відповідає описаному вище фрагменту, наведено на рисунку 3.18.

```
values = dataset.values
values = values.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
n_h = 5
n_f = 2
reframed = series_to_supervised(scaled, n_h, 1)
print(reframed.shape)

values = reframed.values
n_train_hours = 300
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
n_obs = n_h * n_f
train_X, train_y = train[:, :n_obs], train[:, -n_f]
test_X, test_y = test[:, :n_obs], test[:, -n_f]
print(train_X.shape, len(train_X), train_y.shape)
train_X = train_X.reshape((train_X.shape[0], n_h, n_f))
test_X = test_X.reshape((test_X.shape[0], n_h, n_f))
```

Рисунок 3.18 – Приклад коду

Поданий на рисунку 3.18 приклад розділяє набір даних на навчальні та тестові вибірки, а також розбиває їх на вхідні та вихідні змінні. На завершення, вхідні дані (X) перетворюються у формат 3D, що очікується LSTM, а саме [зразки, часові кроки, ознаки]. Тепер ми готові створити та навчити нашу модель LSTM. Ми задамо LSTM з 20 нейронами на кожному з трьох прихованих шарів та 1 нейроном у вихідному шарі для передбачення навантаження. Навчання моделі триватиме 150 епох з розміром пакета 72. Ми також відстежуватимемо помилку навчання за допомогою параметру validation_data у функції fit(). Реалізація моделі показана на рисунку 3.19.

```
# design network
model = Sequential()
model.add(LSTM(20, input_shape=(train_X.shape[1], train_X.shape[2]), return_sequences=True))
model.add(LSTM(20, input_shape=(train_X.shape[1], train_X.shape[2]), return_sequences=True))
model.add(LSTM(20, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='rmsprop')
# fit network
history = model.fit(train_X, train_y, epochs=150, batch_size=72,
                    validation_data=(test_X, test_y), verbose=2,
                    shuffle=False)
```

Рисунок 3.19 – Приклад коду реалізації моделі

Після завершення навчання моделі ми виконуємо прогноз для всього набору тестових даних. Потім об'єднуємо ці прогнози з тестовим набором та відновлюємо початковий масштаб даних. Ми також відновлюємо масштаб тестового набору з очікуваними значеннями. Візуалізація результатів роботи моделі показана на рисунку 3.20. З використанням прогнозів та фактичних значень у початковій шкалі ми обчислюємо оцінку помилки для моделі. У даному випадку наша модель має середньоквадратичну помилку (RMSE) на рівні 8.8 запитів для наших тестових даних групи запитів.

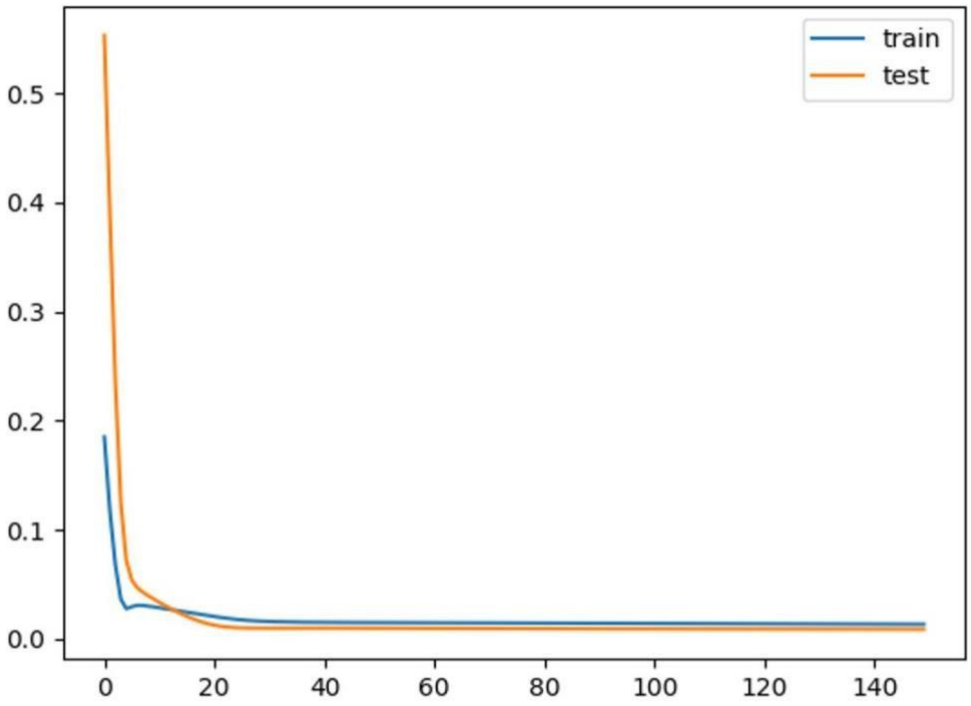

Рисунок 3.20 – Візуалізація результатів

3.5 Аналіз результатів дослідження

У процесі цього дослідження було реалізовано кілька варіантів LSTM мереж для передбачення запитів:

1) LSTM мережа для окремого запиту;

2) LSTM мережа з використанням методу вікна;

3) LSTM мережа з використанням пам'яті;

4) LSTM мережа для групи запитів.

У базовій реалізації мережі для одного запиту середньоквадратична помилка склала 9 запитів. У випадку методу вікна для одного запиту - середньоквадратична помилка становила 6 запитів. Для варіанту з використанням пам'яті для одного запиту помилка склала 8 запитів. У випадку мережі для групи запитів середньоквадратична помилка також склала 8 запитів.

Оцінюючи отримані результати, можна сказати, що розроблений метод є досить працездатним та ефективним. Однак, через те, що він розглядає лише інформацію про один запит, він не враховує потенційні зв'язки між іншими запитами, що може призводити до неточних прогнозів. Використання методу вікна показало кращі результати, але все ще не враховує прихованих залежностей між запитами. Використання всіх переваг методів з окремим запитом дозволяє визначити останню реалізацію мережі як найбільш підходящу, оскільки вона дозволяє враховувати приховані зв'язки. Збір більшої кількості даних може покращити точність результатів.

Висновок з аналізу показує, що найкращим варіантом для передбачення навантаження мережі є використання LSTM з методом вікна для групи запитів.

Підготовка до роботи

Порядок завантаження даних і програм

Для початку роботи з програмним забезпеченням QGIS, слід дотриматися цих кроків.

Для завантаження інсталятора, необхідно перейти на офіційний веб-сайт QGIS, доступний українською мовою, та завантажити файл інсталятора, який має приблизний розмір 1,1 Гб (в залежності від операційної системи). Після завершення завантаження файлу інсталятора, слід виконати інструкції з встановлення програми на комп'ютері.

Після успішної установки QGIS для запуску програми необхідно двічі клацнути на значок, який з'явиться на робочому столі комп'ютера (див. рис. 4.2).

Рисунок 4.2 – Іконка запуску програми QGIS

Після відкриття основного вікна QGIS (див. рис. 4.3), ви можете вибрати вкладку "Останні проекти" або створити новий порожній проект та двічі клацнути на ньому.

Якщо серед доступних файлів відсутній той, який вам потрібен, тоді спробуйте перейти до вкладки "Проект" => "Відкрити". Після цього перейдіть у папку, де знаходиться необхідний файл.

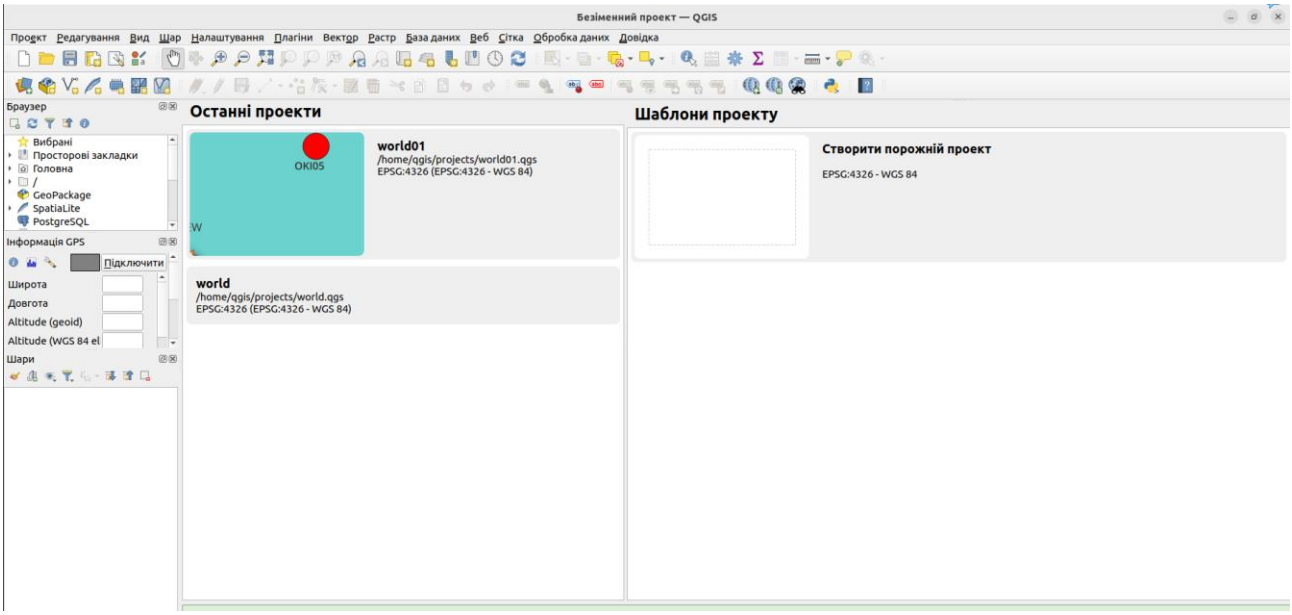

Рисунок 4.3 – Головне вікно програми QGIS

Після цього, відкриється вікно "Редактор полів", де можна змінювати значення атрибутів об'єктів векторного шару. Вибравши шар, можна відредагувати поле, натиснувши кнопку "Відредагувати" на панелі інструментів або у контекстному меню шару (див. рис. 4.4).

Для встановлення додаткових модулів слід перейти до вкладки «Плагіни» => «Управління та встановлення плагінів…».

Завершення роботи з програмою відбувається шляхом закриття основного вікна програмного середовища QGIS або через меню «Проект» => «Вийти з QGIS».

Розпізнавання об'єктів та прогнозування їх дій базується на інформації, отриманій від сенсорного комплексу і перед використанням обробленої у модулі збору інформації.

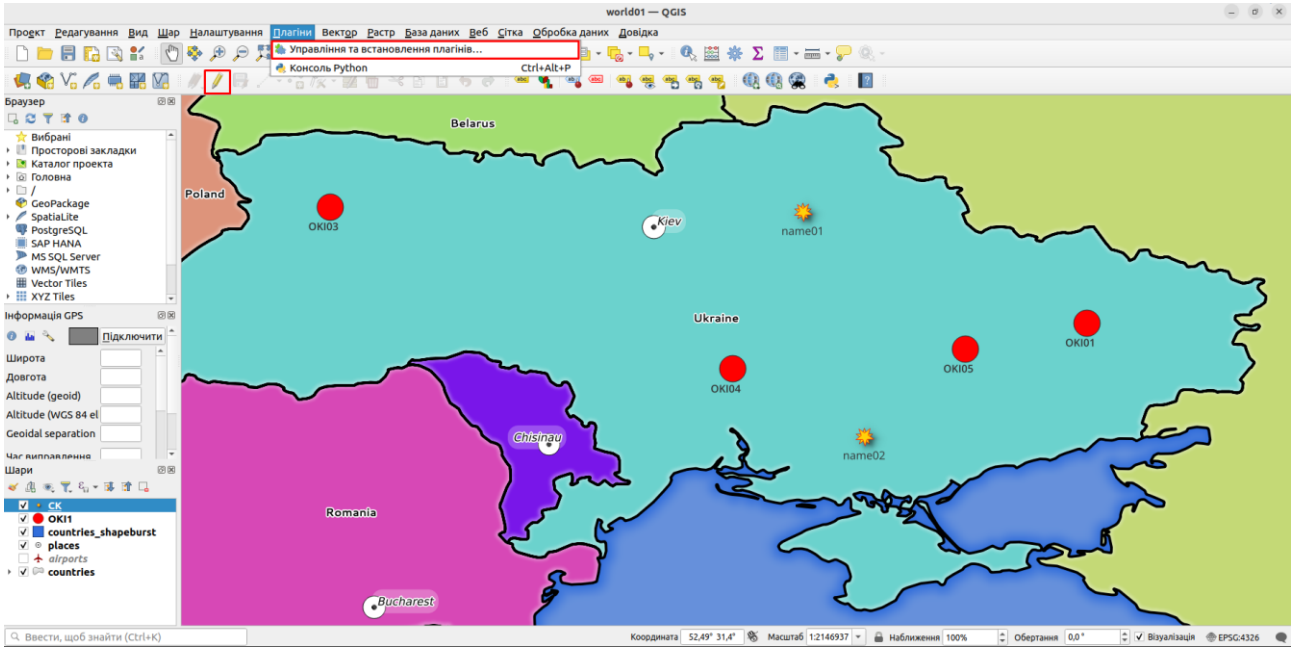

Рисунок 4.4 – Вікно редактора текстів програм у середовищі QGIS

Модуль геоінформаційних додатків (ГІС) QGIS приймає такі дані:

- Географічні координати об'єкту.

- Тип об'єкту.

- Назва об'єкту.

- Тип позначки.

- Колір позначки.

- Тип картографічного об'єкту (наприклад, точковий, лінійний, площинний, растровий).

- Інші атрибутивні дані, специфічні для об'єкту (наприклад, потужність генерації для ТЕЦ).

Перевірка працездатності QGIS включає такі кроки:

1. Завантаження тестових даних та їх обробка.

2. Створення різних типів карт і використання різних методів візуалізації.

3. Перевірка взаємодії з іншими програмними засобами.

4. Оцінка швидкодії та виконання обчислень.

QGIS має різноманітні функції та модулі, серед яких:

- Редагування та відображення векторних об'єктів.
- Обробка растрових даних.
- Відображення геоданих в різних форматах.
- Робота з системами координат та проекціями.
- Взаємодія з базами даних, включаючи PostgreSQL та SpatiaLite.
- Створення карт та звітів через графічний інтерфейс.

Також відбувається розробка та використання додаткових модулів і плагінів для розширення функціоналу QGIS.

Серед комплексів задач QGIS:

- Пошук найкоротшого маршруту між точками на карті.
- Обчислення площі та об'єму геоданих.
- Аналіз зон ризику.
- Робота з GPS-даними та іншими геодезичними параметрами.
- Використання геометричних алгоритмів для розрахунку геодезичних параметрів та навігації за допомогою глобальних навігаційних супутникових систем.

2.3. Інструкція користувача

Головне меню включає наступні розділи:

1. **Моніторинг:** Відображення повідомлень про невдачі у вході для адміністратора безпеки. Карта району спостереження через ГІС, де відображаються границі району та об'єкти за певними умовами.

2. **Ідентифікація:** Меню для уточнення умов відбору об'єктів на карті, включаючи час, регіон, категорію критичності, сенсори, об'єкти, та маршрути.

3. **Прогнозування:** Вибір часового інтервалу для відображення об'єктів загроз.

4. **Налаштування:** Функція автоматичного оновлення на карті кожні 5 секунд.

5. **Допомога:** Підтримка та інформація про систему.

Відмінності в кольорах позначають основні групи об'єктів, в яких вони виявлені. Меню "Моніторинг" також дозволяє фільтрувати об'єкти за часом, регіоном, категорією критичності, сенсорами, об'єктами, та маршрутами, щоб спростити спостереження. Також є можливість вибору періоду часу для відображення об'єктів загроз на карті. Категорії критичності від I до IV вказують рівень важливості об'єктів.

Меню "Регіон" дозволяє виділяти конкретні області на карті, спрощуючи аналіз даних. "Об'єкти" дозволяють категоризувати об'єкти за їхнім типом, а

"Маршрути" дають змогу переглядати та вмикати/вимикати актуальні маршрути на карті разом з інформацією про них.

Ця функція дозволяє користувачам спростити та налаштувати аналіз даних відповідно до конкретного регіону, що є важливим для ефективного управління ризиками та безпекою.

Ця дія здійснюється за допомогою випадаючого меню. Системою буде відфільтровано об'єкти критичної інфраструктури та залишаться видимими тільки ті які відповідають умовам вибору. Категорії I – IV: включає в себе чотири категорії, позначені як I, II, III та IV.

Кожна категорія відповідає різному рівню важливості, де категорія I вказує на найвищий рівень важливості, а категорія IV - на найнижчий. Користувач має можливість вибирати категорію в залежності від конкретної ситуації.

За допомогою підменю «Об’єкти» в розділі Моніторинг можна категоризувати об’єкти відповідно до їхнього типу:

В підменю «Маршрути» надається можливість включати/виключати актуальні маршрути на карті, які збережені в системі. Після переключення підменю «Маршрути» ви зможете переглядати докладну інформацію про той чи інший маршрут заданий в системі.

Розділ "Ідентифікація" надає можливість користувачам скористатися різноманітними інструментами та функціями для розпізнавання та виявлення об'єктів. У таблиці містяться дані, що стосуються моніторингу в різних регіонах протягом певного часу.

У верхній правій частині таблиці розміщено перемикач "Автоматичне оновлення", який дозволяє ввімкнути або вимкнути автоматичне оновлення інформації в таблиці з інтервалом у 5 секунд.

Стовпці у таблиці включають такі поля:

- "Час": дані представлені у форматі дати та часу.

- "Регіон": вказує географічний регіон, область, в якій проводилися вимірювання.

- "Широта": географічні координати північної або південної широти.

- "Довгота": географічні координати східної або західної довготи.

- "Імовірність": числове значення, що вказує на точність ідентифікації об'єкта. Значення 0 означає повну невпевненість у його ідентифікації, а 1 - повну впевненість.

- "Дії": містить дві кнопки. Натискання кнопки з дискетою виводить екранну форму з інформацією про успішне збереження сигналу в базі сигналів системи (див. Рис. 4.5).

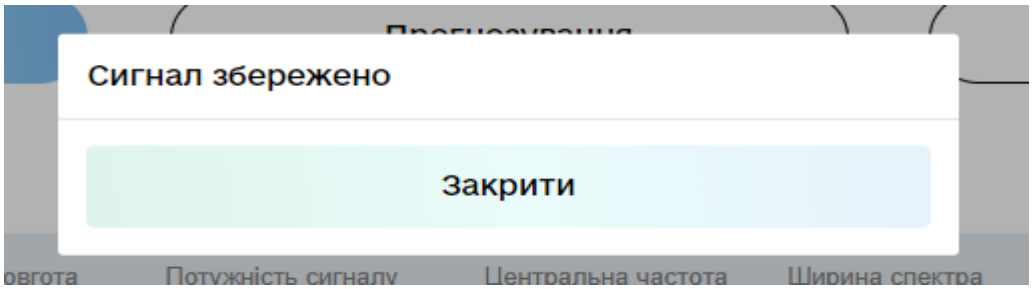

Рисунок 4.5 - Форма збереження сигналу в таблиці «Ідентифікація»

Розділ «Прогнозування» виводить на екран графічну складову (мапу) яка надає можливість візуально аналізувати всі об’єкти та можливі напрямки руху (Рис. 4.6.).

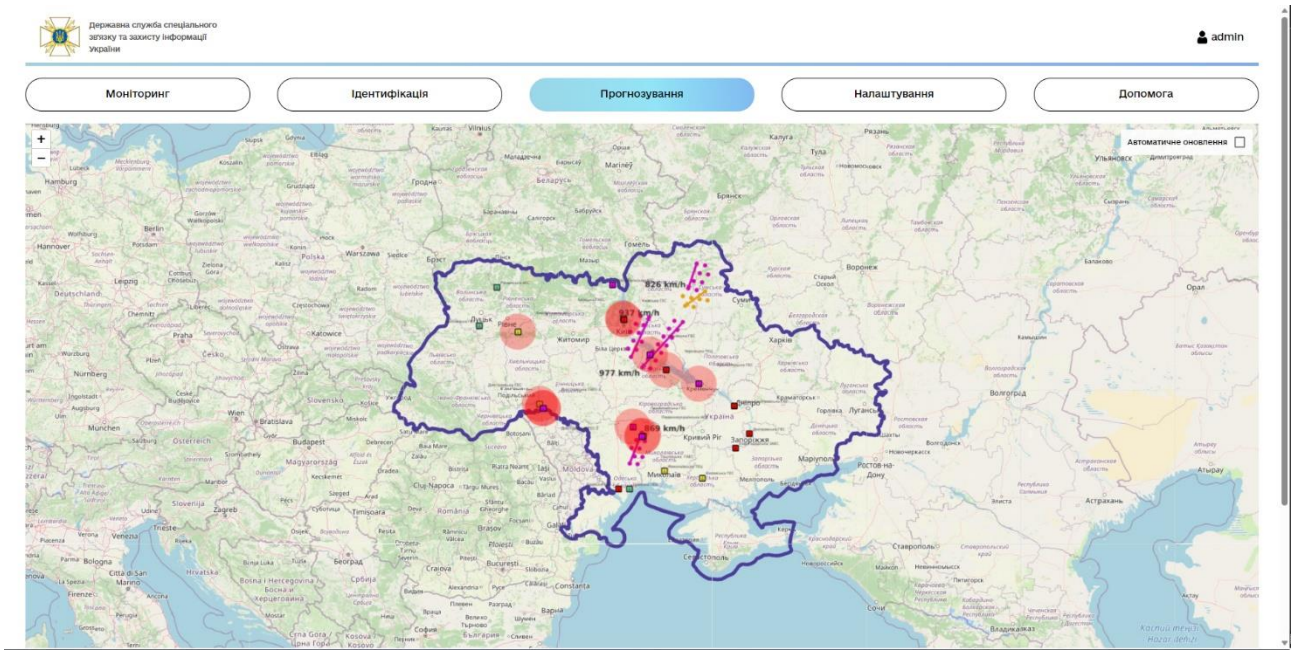

Рисунок 4.6 - Розділ «Прогнозування» мапа

На карті відображено критичну інфраструктуру, траєкторії руху об'єктів та їхню швидкість. Також присутня кнопка "Автоматичне оновлення", яка може увімкнути або вимкнути автоматичне оновлення інформації на карті кожні 20 секунд.

Розділ "Налаштування" складається з панелі випадаючих списків та панелі керування (див. Рис. 4.7).

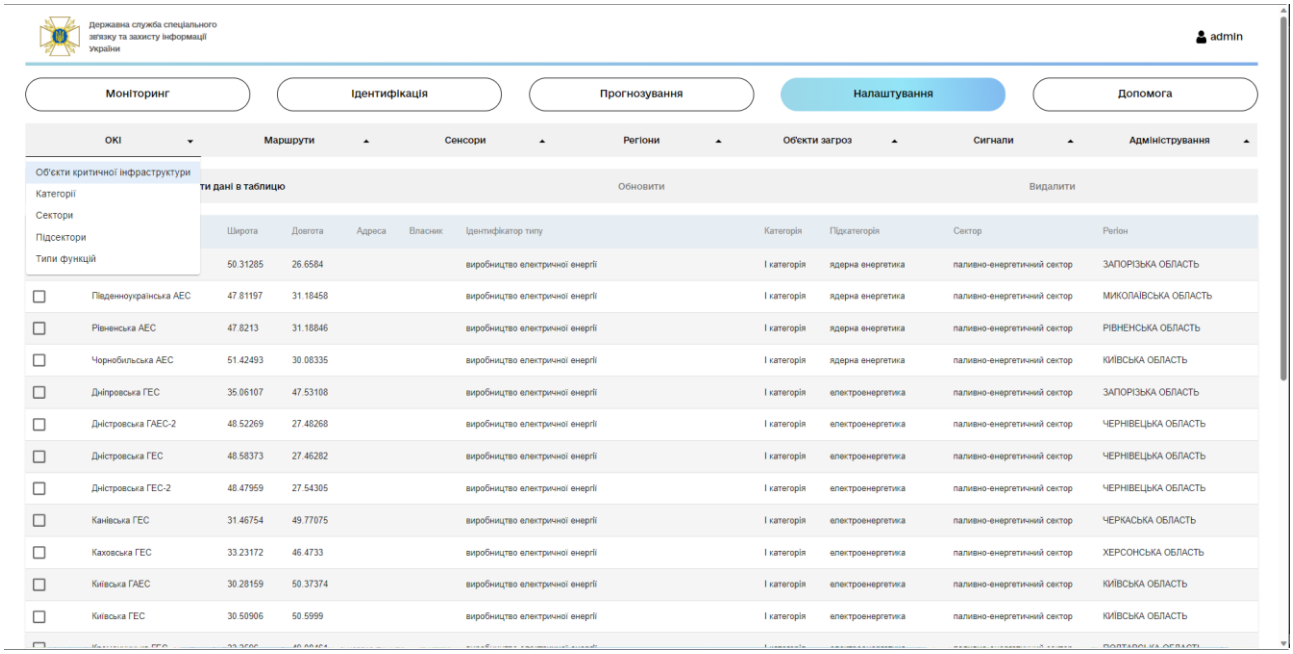

Рисунок 4.7 - Розділ «Налаштування» підменю «Об’єкти критичної інфраструктури»

Після обрання відповідного пункту меню будуть доступні кнопки редагування "Оновити" та "Видалити" для Об'єктів Критичної Інфраструктури (ОКІ). Таблиця містить наступні стовпчики:

- "Вибрати" - для встановлення прапорця і вибору об'єкта.
- "Код" - унікальний ідентифікатор об'єкта.
- "Найменування" - назва об'єкта критичної інфраструктури.
- "Широта" і "Довгота" - точне місцезнаходження.
- "Адреса" - конкретне місцезнаходження об'єкта.

- "Власник" - інформація про відповідальну особу за об'єкт.

- "Ідентифікатор типу" - вказує на сектор інфраструктури об'єкта.

- "Категорія" - класифікація за важливістю та критичністю, поділення на чотири категорії: I, II, III та IV, де I - найбільш важлива, а IV - найменш критична.

- "Підкатегорія" - деталізована класифікація сфери діяльності об'єкта.

- "Сектор" - загальна класифікація галузі, до якої належить об'єкт.

- "Регіон" - територіальна одиниця, що показує конкретну область, де розташований об'єкт.

Меню створення об'єкта критичної інфраструктури представлене на Рис. 4.8.

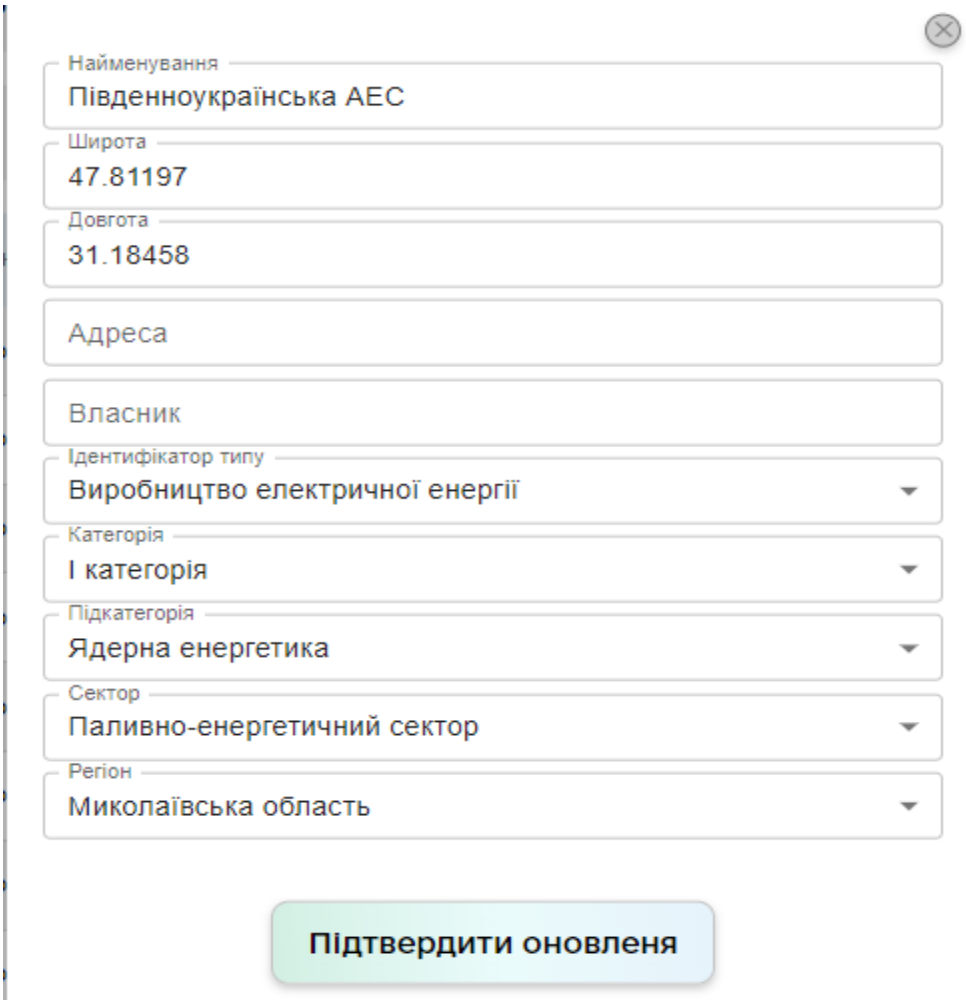

Рисунок 4.8 - Меню створення об’єкта критичної інфраструктури, заповненні поля

Меню створення об’єкта критичної інфраструктури складається з полів:

- Найменування

- Широти
- Довготи
- Адреса
- Власник
- Ідентифікатор типу
- Категорія
- Підкатегорія
- Сектор
- Регіон

Якщо поля заповнено неправильно чи залишити деякі пустими то система підсвітить ці поля червоним (Рис. 4.9).

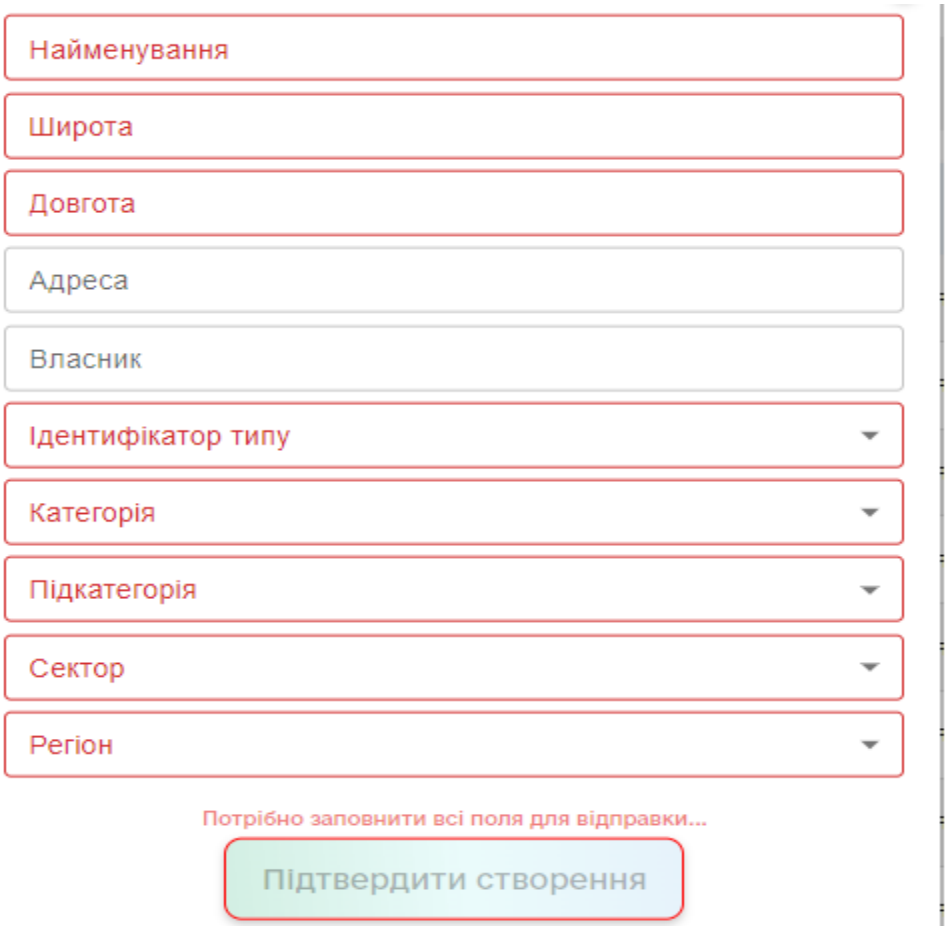

Рисунок 4.9 – Меню створення об’єкта критичної інфраструктури з не заповненими полями

Для успішного створення об'єкту необхідно заповнити всі поля, за винятком "адреси" та "власника", які є необов'язковими. Після введення інформації кнопка "Підтвердити створення" стане активною.

Підтвердження внесення даних передбачає підпис користувача (електронний підпис) та відмітку часу. В системі використовується Удосконалений Електронний Підпис (УЄП) для підписування внесених змін у дані.

Після натискання кнопки "Підтвердити створення" система запитає підтвердження внесення даних за допомогою електронного підпису. Відкриється вікно вибору носія ключа (див. Рис. 4.10). Якщо на носії ключів доступні кілька сертифікатів, потрібно буде вибрати один з них для підписування даних.

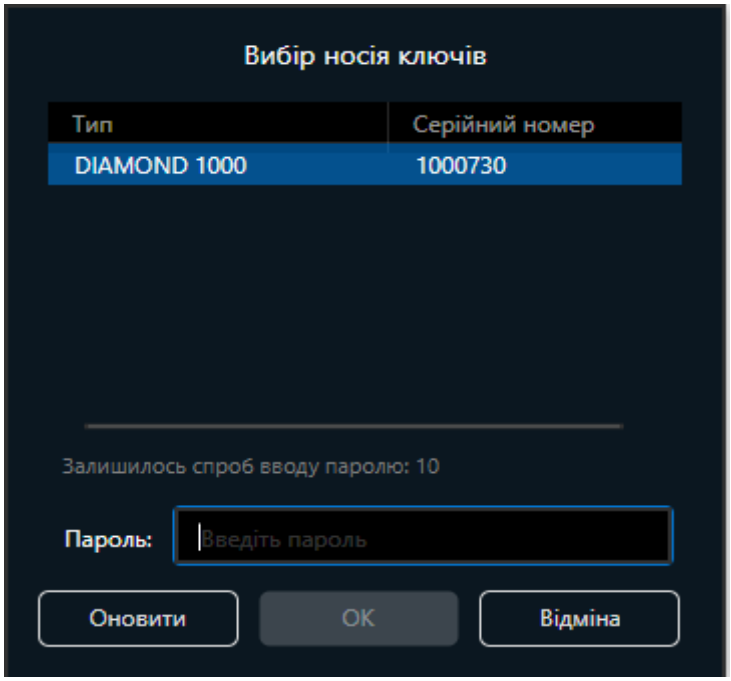

Рисунок 4.10 - Вікно вибору носія ключа

Після вказівки пароля до сертифіката, система проведе перевірку правильності пароля. Якщо буде введено неправильний пароль, з'являтиметься відповідне повідомлення (Рис. 4.11).

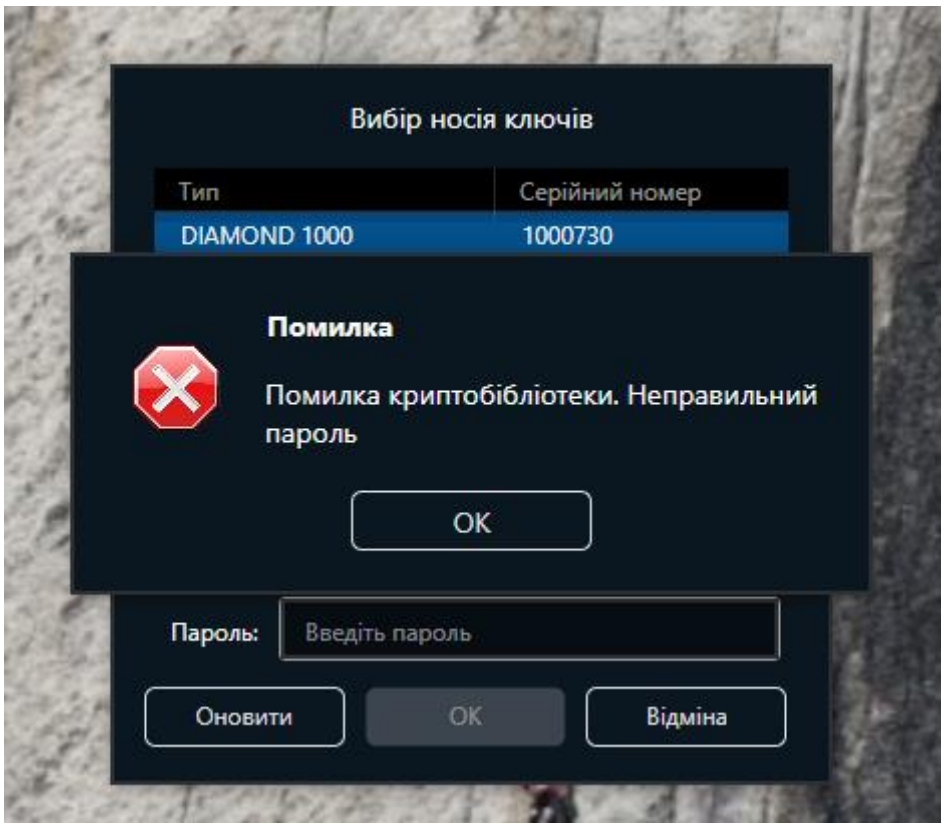

Рисунок 4.11 - Вікно повідомлення, що введено неправильний пароль

У разі позитивного результату дані будуть збережені до бази даних (Рис. 4.12).

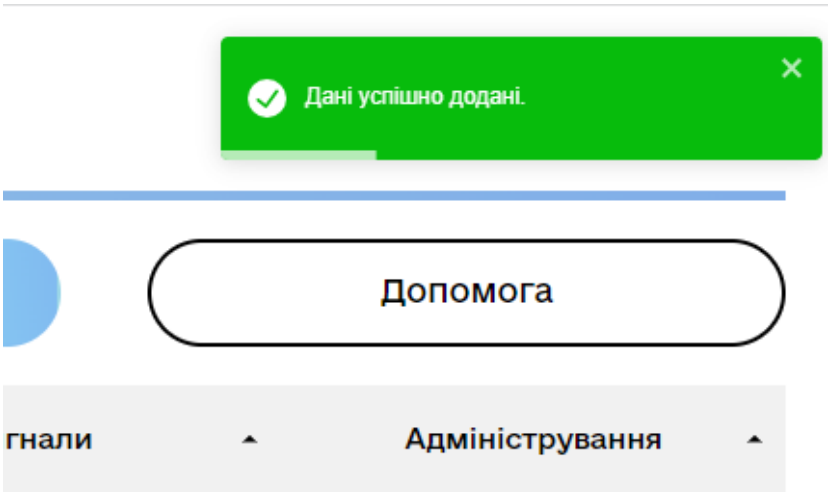

Рисунок 4.12 – Повідомлення, що дані збережені до бази даних

У разі негативного результату дані не будуть збережені до бази даних (Рис.4.13).

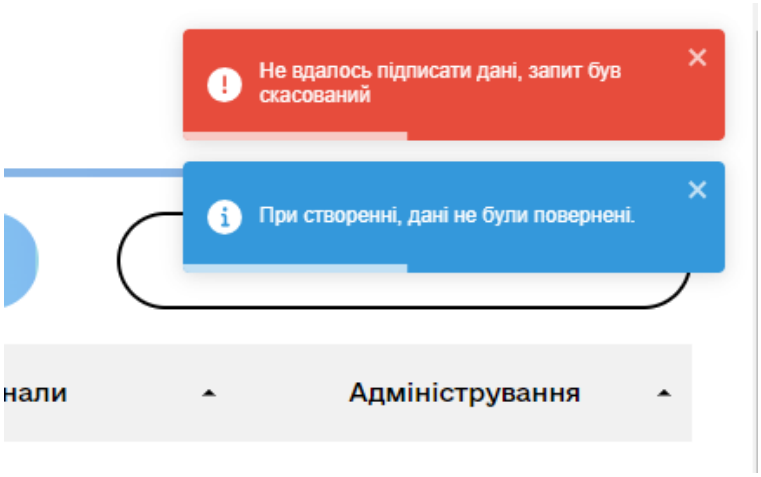

Рисунок 4.13 – Повідомлення, що дані не будуть збережені до бази даних

У розділі "Об'єкти критичної інфраструктури" в підменю "Категорії" користувач має можливість створювати, оновлювати та видаляти елементи структури (див. Рис.4.14). У таблиці присутні такі стовпці:

• "Вибрати" - після активації цієї опції стає доступним оновлення та видалення даних.

• "Код категорії" - внутрішній унікальний ідентифікатор для категорій.

• "Найменування" - номер та назва категорії.

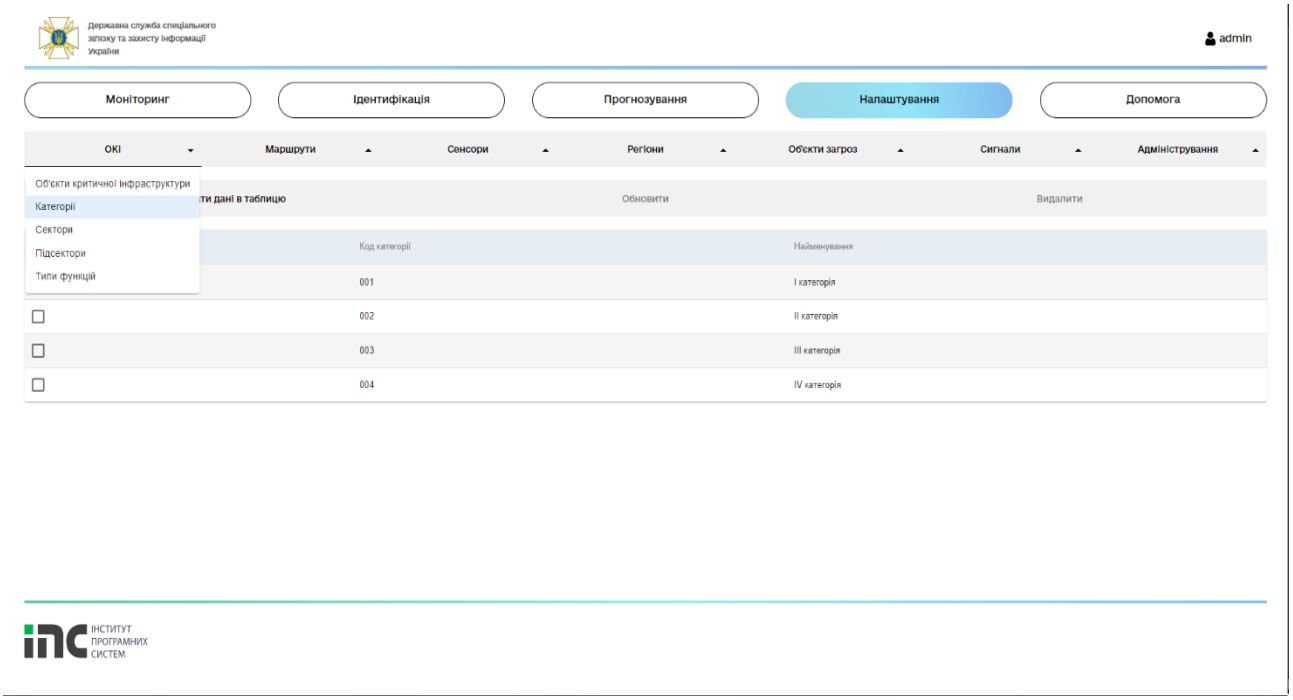

Рисунок 4.14 – Меню «ОКІ» підменю «Категорії»

Після натискання кнопки "Додати дані в таблицю", відкриється вікно з червоними полями "Код категорії" та "Найменування", які необхідно обов'язково заповнити (див. Рис. 4.15). Після введення цих даних стане доступним поле "Підтвердити створення", яке дозволить створити нову категорію у таблиці (див. Рис. 4.16).

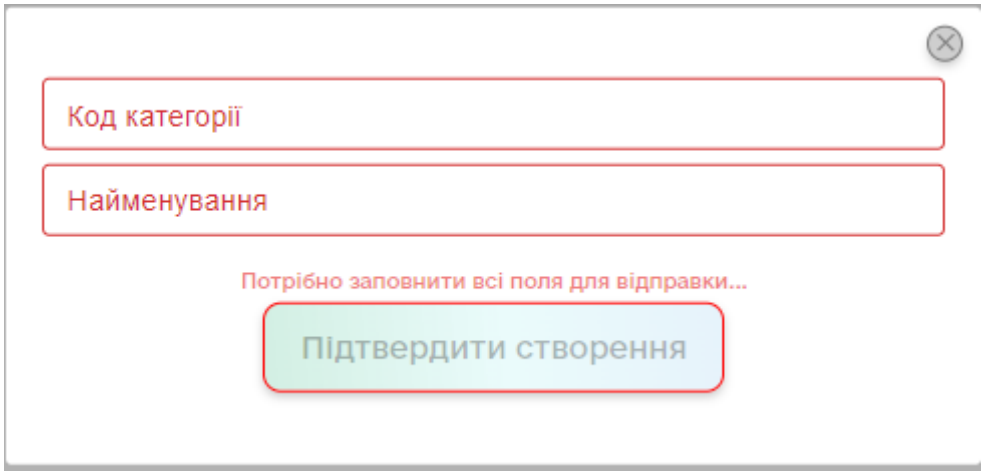

Рисунок 4.15 – Вікно створення категорії з незаповненими полями

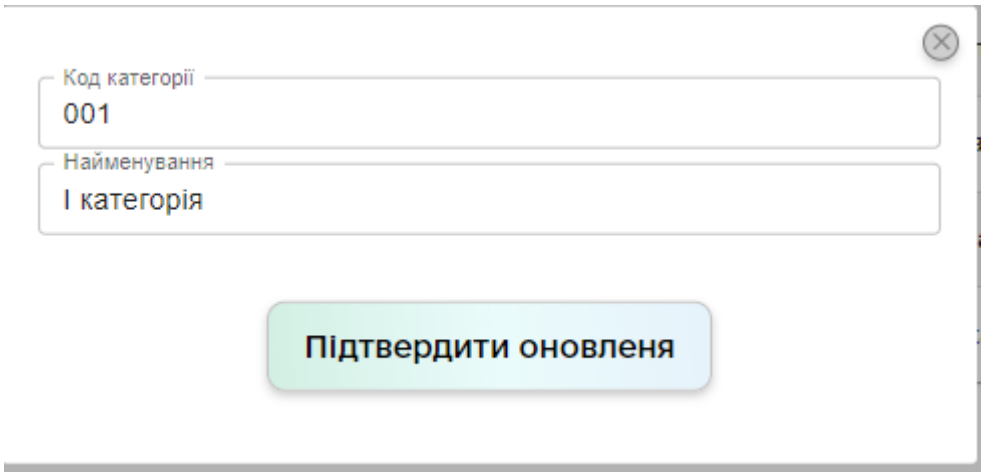

Рисунок 4.16 – Приклад успішно заповнених даних вікна «Категорії»

При виборі опції "Сектори" з розкривного списку пункту "ОКІ", у таблиці відображаються всі існуючі в системі сектори об'єктів критичної інфраструктури (див. Рис. 4.17). Коли натискається кнопка "Додати дані в таблицю", з'являється

вікно для створення нового сектору.

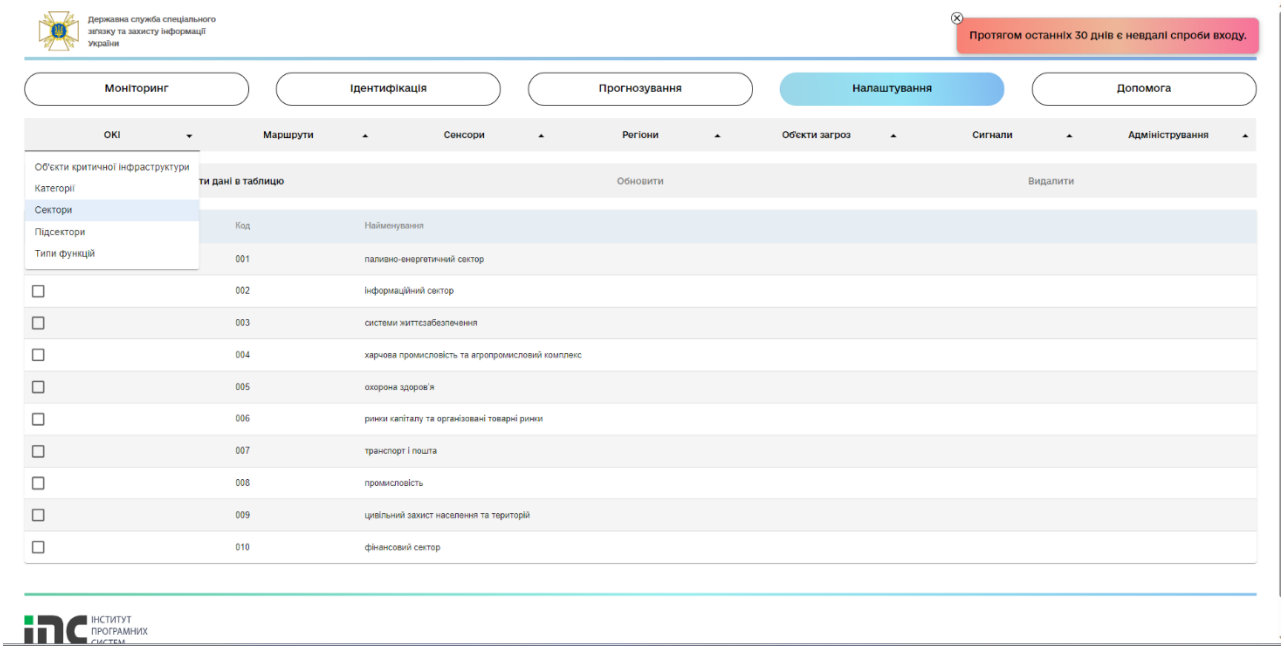

Рисунок 4.17 – Меню «ОКІ» підменю «Сектори»

Червоним відмічені поля для обов'язкового заповнення а саме (Рис.4.18):

- Код – кожному сектору призначений спеціальний код внутрішній код.
- Найменування – назва сектору інфраструктури.

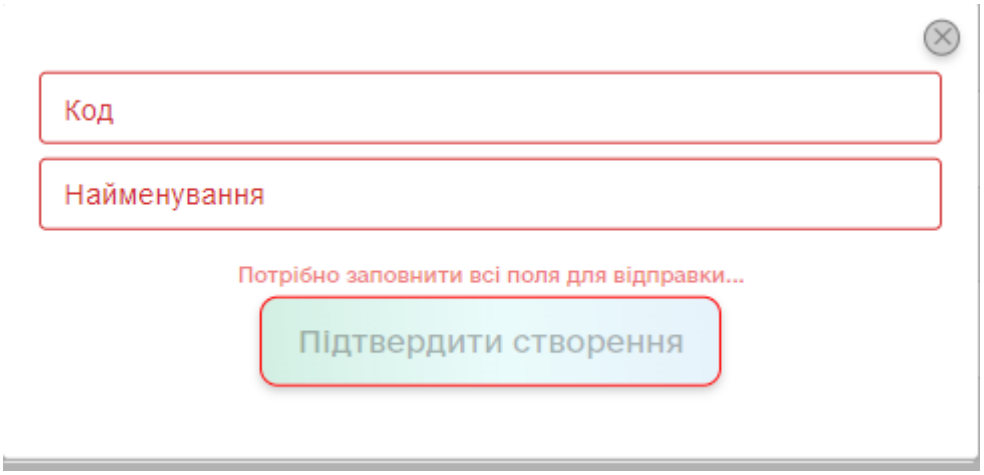

Рисунок 4.18 – Вікно створення сектору з незаповненими полями

Після успішного заповнення полів, кнопка «Підтвердити оновлення» стає доступною і дає можливість додати данні про сектор у таблицю (Рис.4.19).

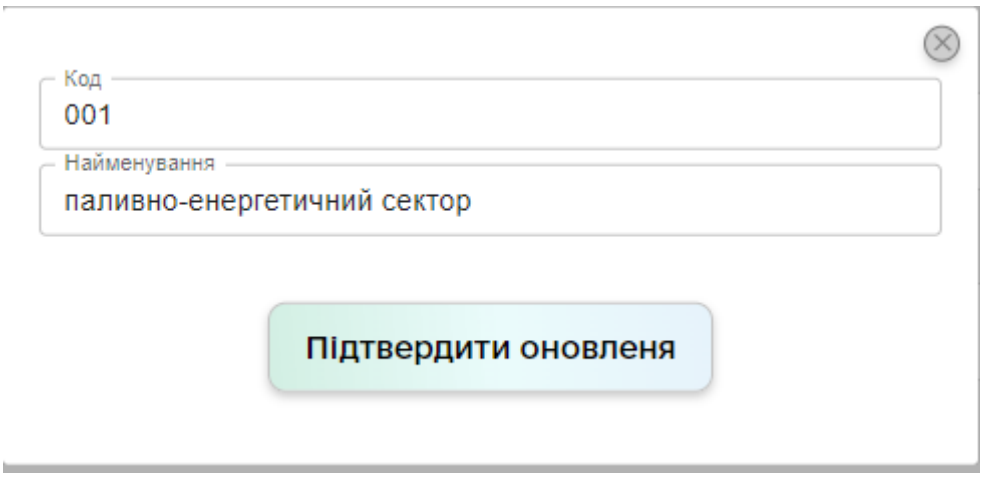

Рисунок 4.19 – Приклад успішного заповнення даних вікна «Сектори»

Після вибору опції "Підсектори" зі списку, який відображений як "ОКІ" (див. Рис. 4.20), у таблиці з'являються всі наявні сектори, що вже присутні у системі. Для додавання даних в систему необхідно натиснути кнопку "Додати дані в таблицю", після чого відкриється вікно для створення нового сектору.

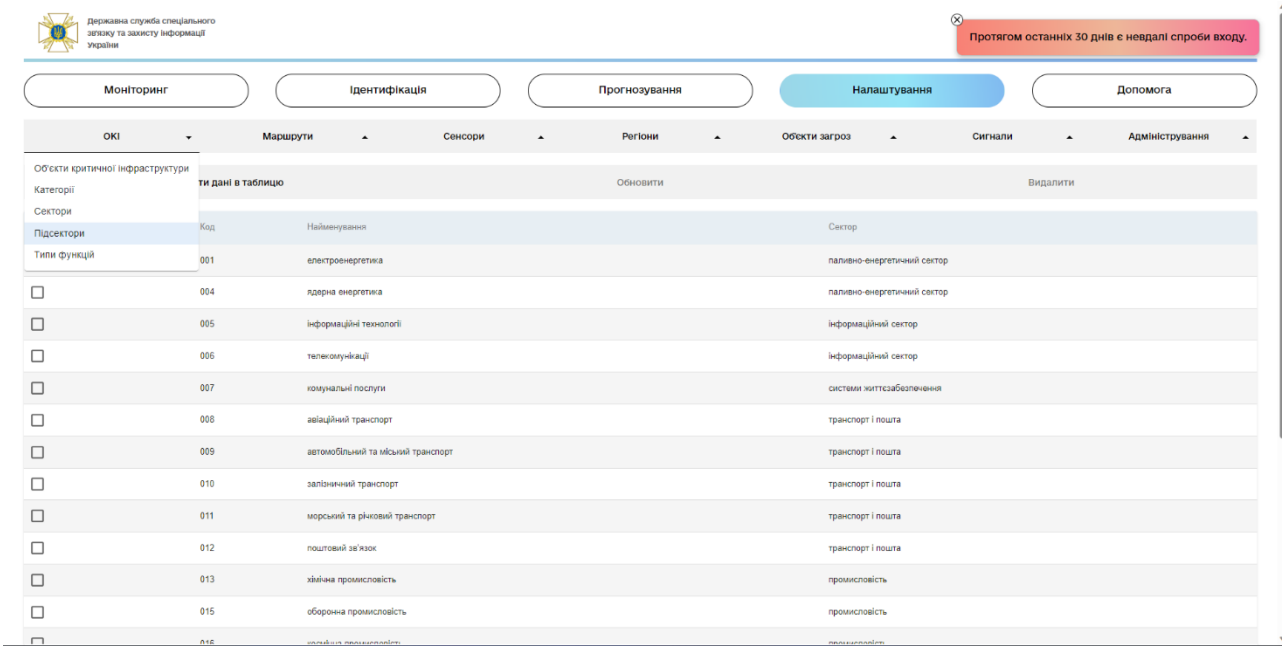

Рисунок 4.20 – Меню «ОКІ» підменю «Підсектор»

Червоним відмічені поля для обов’язкового заповнення, а саме (Рис.4.21):

- Код - кожному підсектору призначений спеціальний код.
- Найменування – назва підсектору інфраструктури.
- Сектор – найменування типу до якого відноситься підсектор.

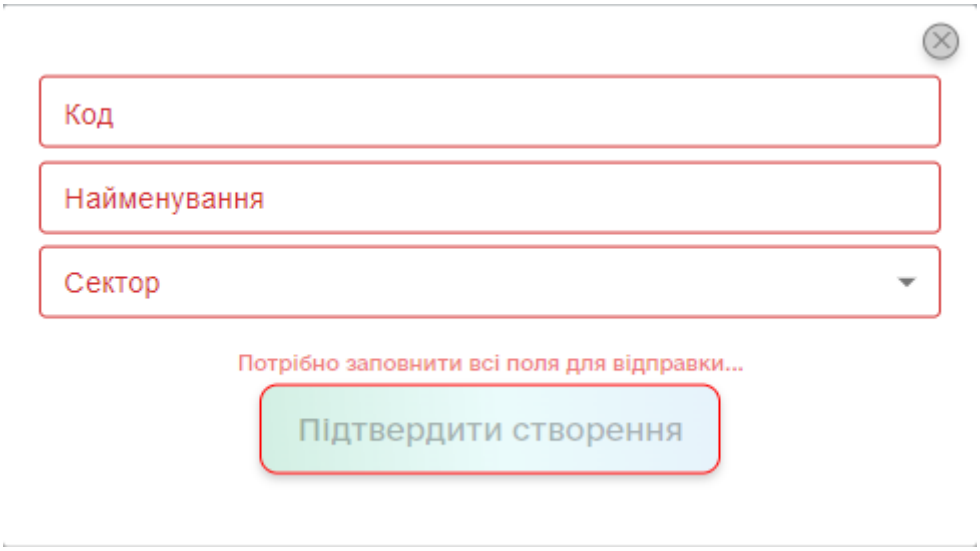

Рисунок 4.21 – Вікно створення підсектору з незаповненими полями

Після успішного заповнення полів, кнопка «Підтвердити оновлення» стає доступною і надається можливість додати данні про підсектору у таблицю (Рис.4.22) .

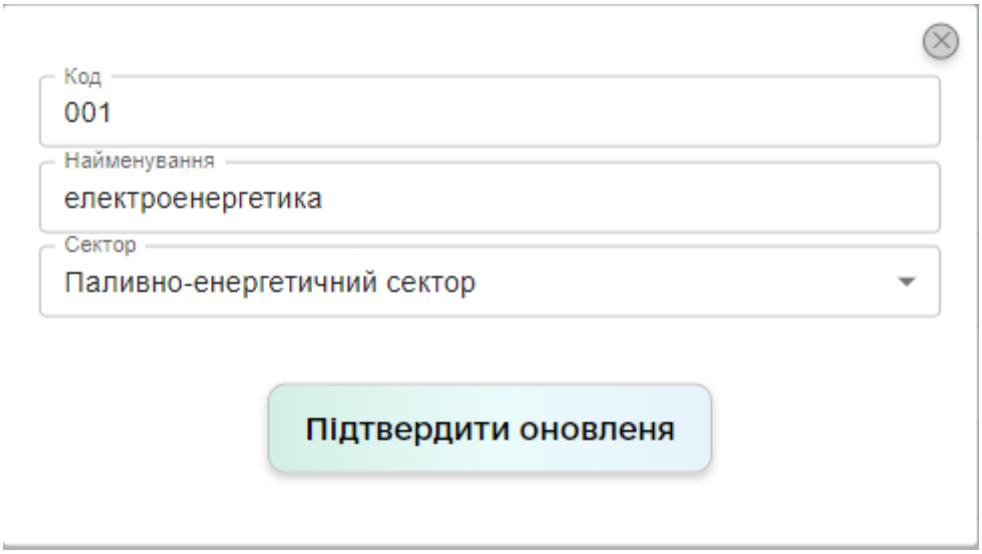

Рисунок 4.22 – Приклад успішного заповнення даних вікна «Підсектори»

При виборі «ОКІ» зі списку обираємо «Типи функцій» (Рис.4.23). В таблиці відображаються всі існуючі функції в системі. При натисканні на кнопку «Додати данні в таблицю» з’являється вікно створення нового об’єкту.

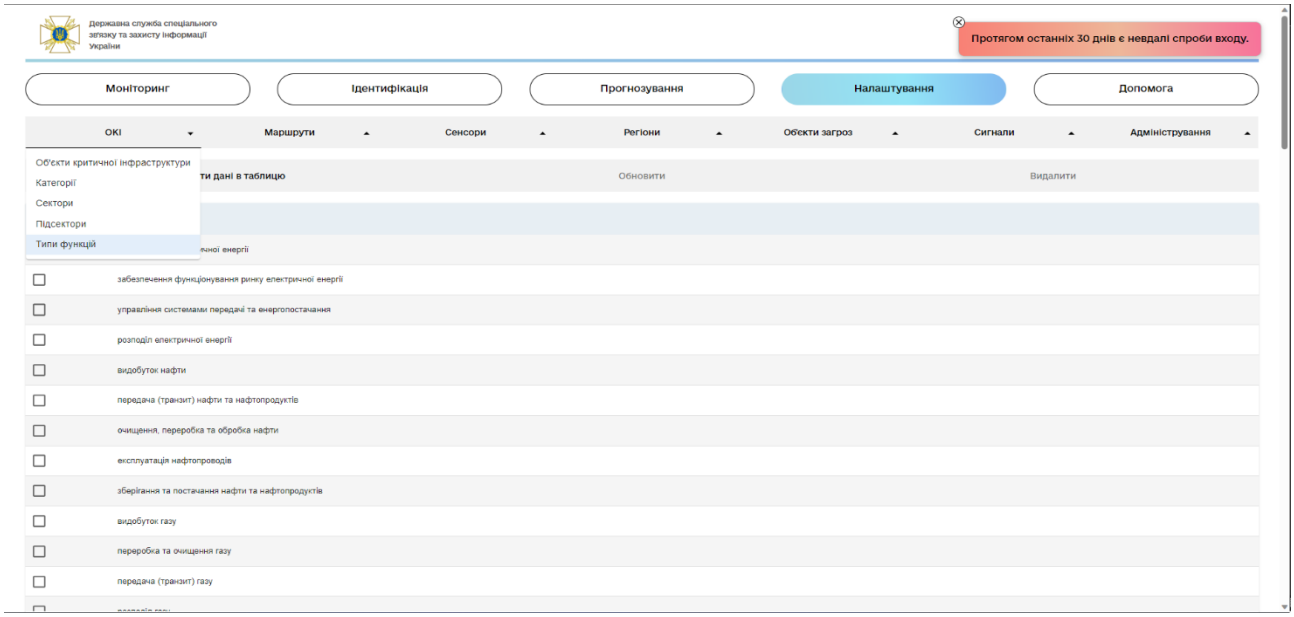

Рисунок 4.23 – Меню «ОКІ» підменю «Типи функцій»

Червоним відмічені поля для обов'язкового заповнення, а саме (Рис.4.24):

- Найменування – назва функції об'єкта критичної інфраструктури

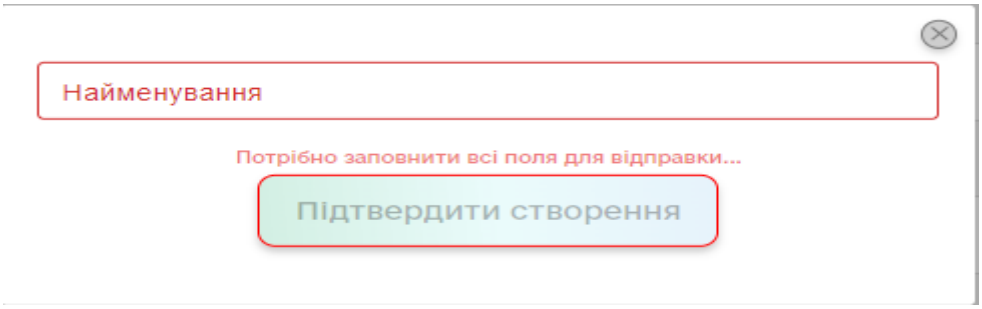

Рисунок 4.24 – Вікно створення типи функцій з незаповненими полями

Після успішного заповнення полів, «Підтвердити оновлення» стає доступною і додає данні «типи Функцій» у таблицю (Рис.4.25).

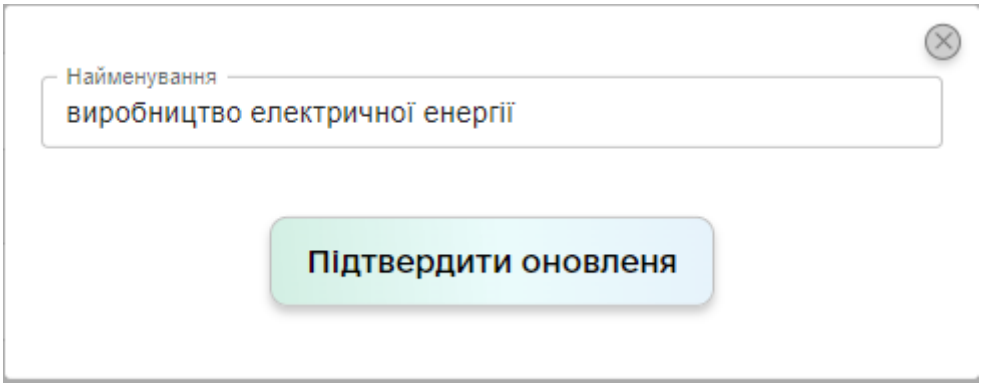

Рисунок 4.25 – Приклад успішного заповнення даних вікна «Типи Функцій»

У пункті меню "Маршрути" на екрані користувача доступна таблиця, яка містить інформацію про збережені в системі маршрути об'єктів (див. Рис. 4.26). Ця таблиця надає можливість перегляду даних щодо кодів об'єктів, їхніх найменувань, дати створення маршруту, типу та інформації про актуальність поточного маршруту.

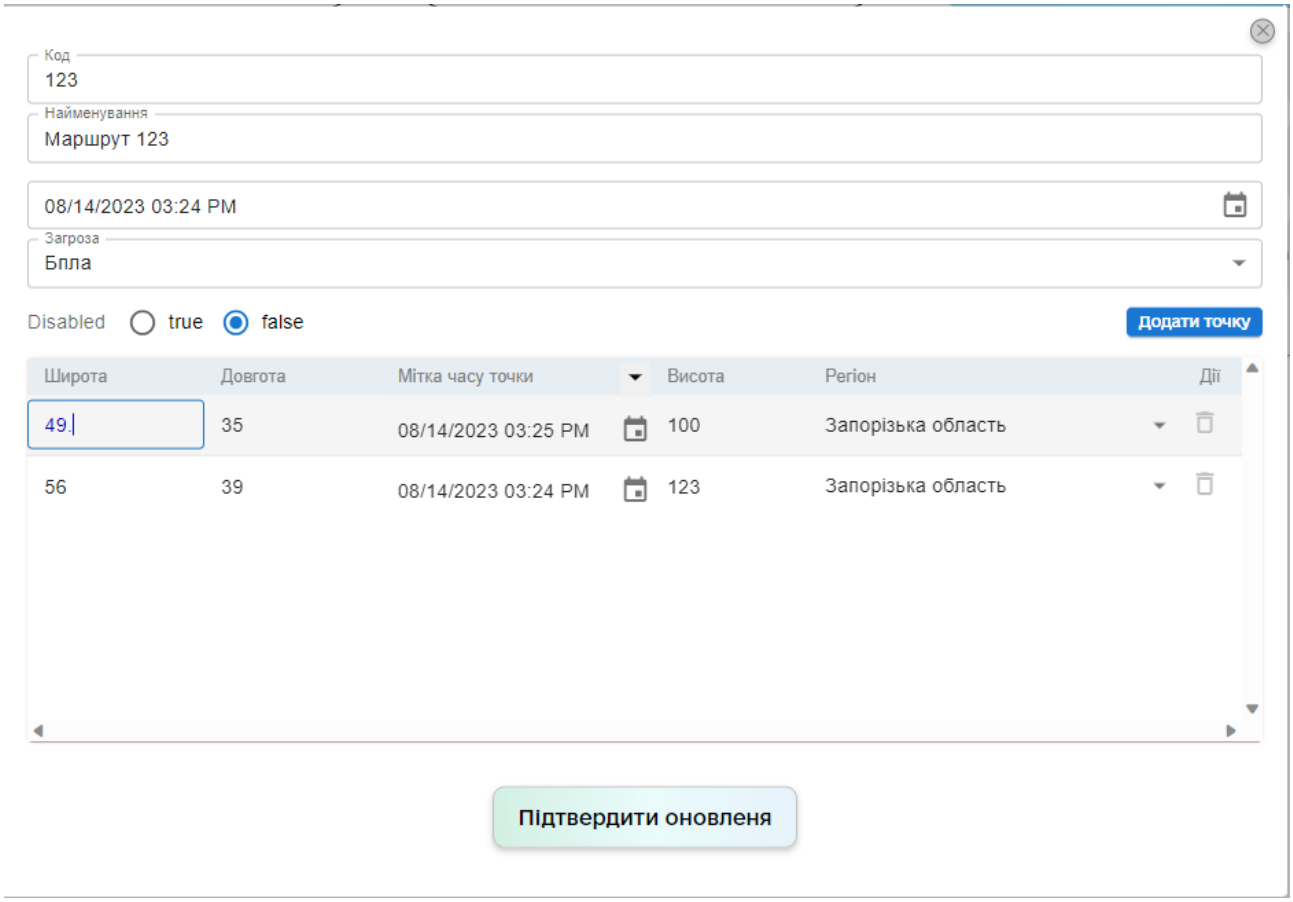

Рисунок 4.26 – Вікно створення «Маршруту»

Для створення нового маршруту потрібно натиснути кнопку "Додати дані в таблицю". Після цього відкриється вікно, де можна створити новий маршрут. Деякі поля позначені червоним кольором і потребують обов'язкового заповнення (див. Рис. 4.27):

- Код: кожному маршруту надається спеціальний ідентифікатор.
- Найменування: текстове найменування маршруту.

- Дата створення маршруту: відображається у форматі місяця, числа, року, години, хвилини та формату AM/PM.

- Активність маршруту (Disabled).

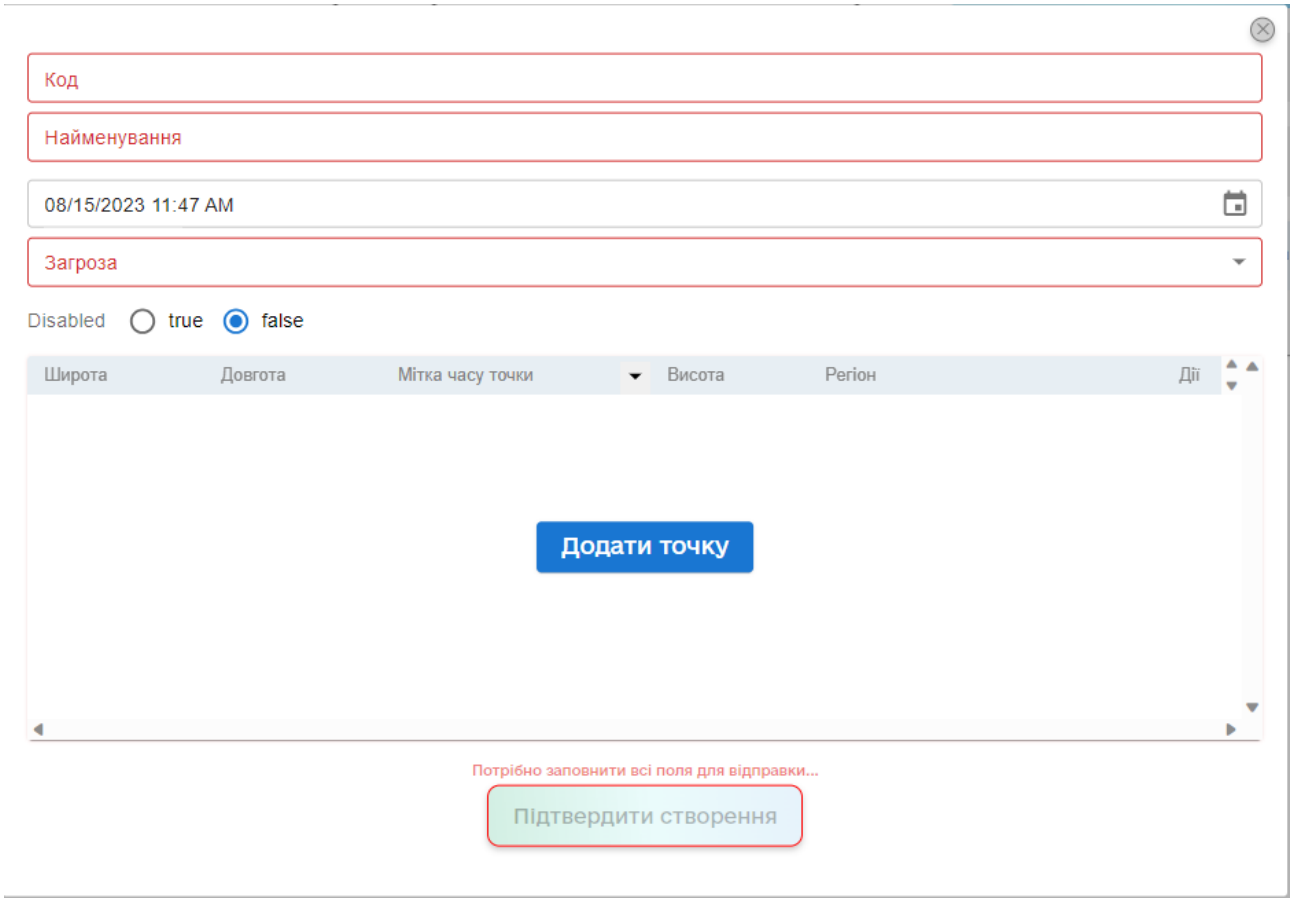

Рисунок 4.27 – Вікно створення маршруту з незаповненими полями

На екрані форми є кнопка "Додати точку", яка дозволяє вводити інформацію про точки маршруту, представлену такими параметрами:

- Широта: географічна широта точки.

- Довгота: географічна довгота точки.

- Мітка часу точки: відображає час у форматі місяця, числа, року, години, хвилини та AM/PM.

- Висота: висота точки над рівнем моря.

- Регіон: область, до якої відноситься точка маршруту.

Можливість додавання точок необмежена, що дозволяє створювати маршрути будь-якої довжини. Рекомендується вводити точки маршруту в порядку зростання часу мітки.

Всі поля у таблиці створення маршруту є обов'язковими. Дата та час автоматично встановлюються, але можуть бути змінені за допомогою іконки календаря в правій частині поля з датою та часом (див. Рис. 4.28).

Після успішного заповнення полів кнопка "Підтвердити оновлення" стає доступною і дозволяє додавати або оновлювати дані маршрутів у таблиці.

Натискання на іконку з кошиком в колонці "Дії" призведе до видалення точки маршруту з таблиці.

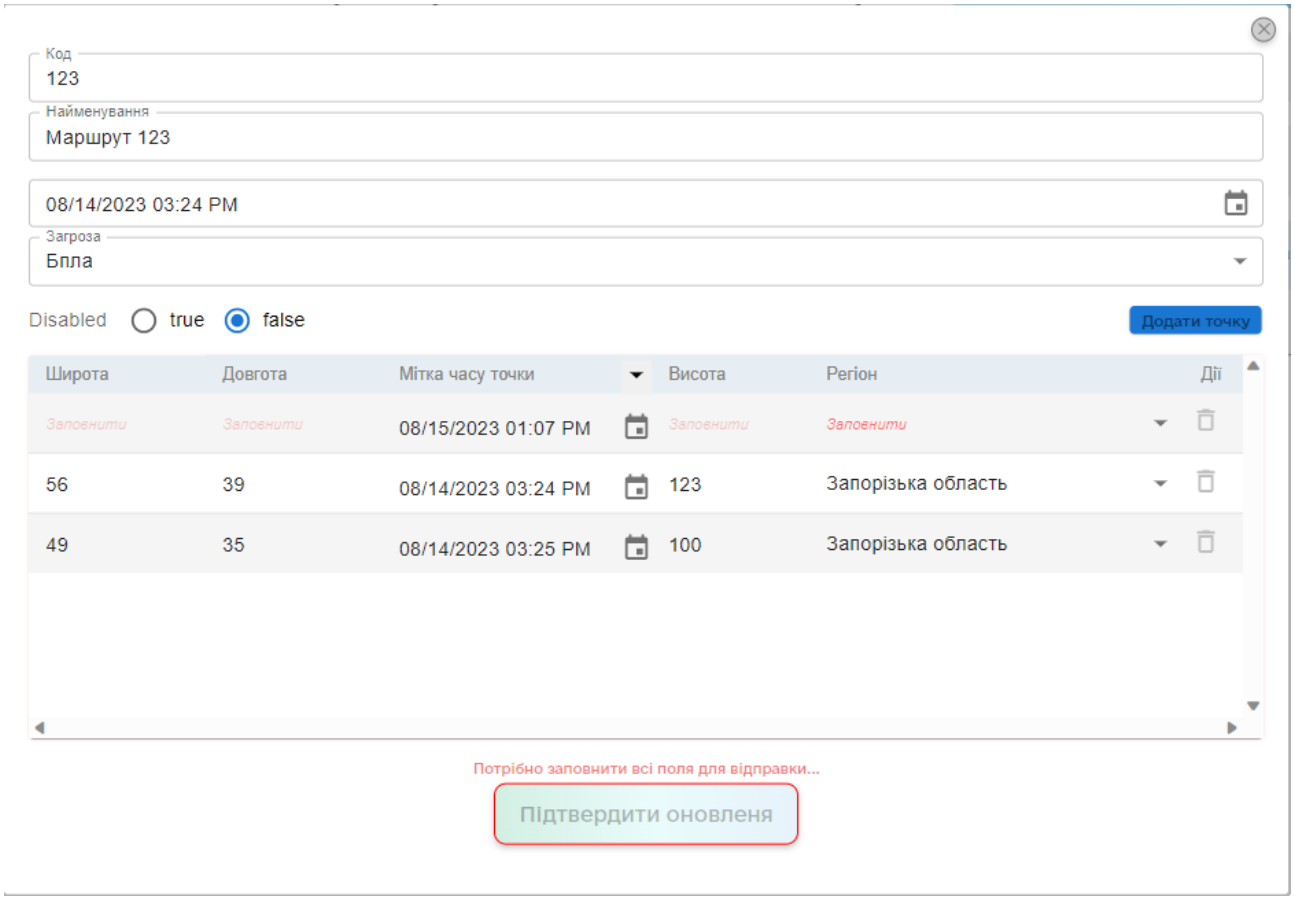

Рисунок 4.28 – Вікно створення «Маршруту» з незаповненими полями

При виборі "Регіони" у головному меню та підпункту "Класифікатор об’єктів адміністративно-територіального устрою" на екрані відображається таблиця, що містить усі області України (див. Рис. 4.29):

- Код: унікальний ідентифікатор.
- Найменування: назва регіону або області.

Слід зазначити, що можливості додавання, видалення чи оновлення відсутні.

При переході до меню "Адміністрування" та підпункту "Користувачі" на екрані з’являється таблиця, що містить усіх користувачів, які присутні у базі даних (див. Рис. 4.30). У цьому вікні доступна можливість додавання та оновлення ролей для користувачів (див. Рис. 4.31).

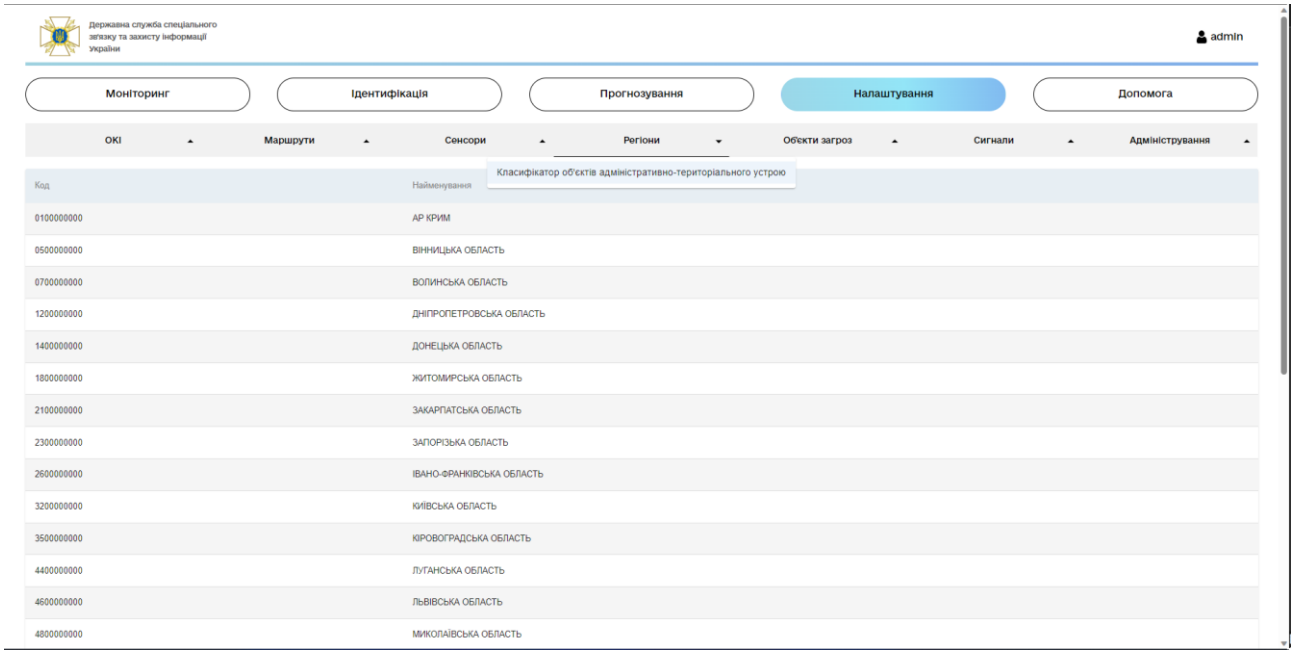

Рисунок 4.29 – Таблиця з регіонами в меню «Регіони»

Існують такі ролі як:

- Користувач (User)
- Адміністратор нейронної мережі (Neural network administrator) .
- Адміністратор обміну (Data exchange administrator).
- Системний адміністратор (System administrator.

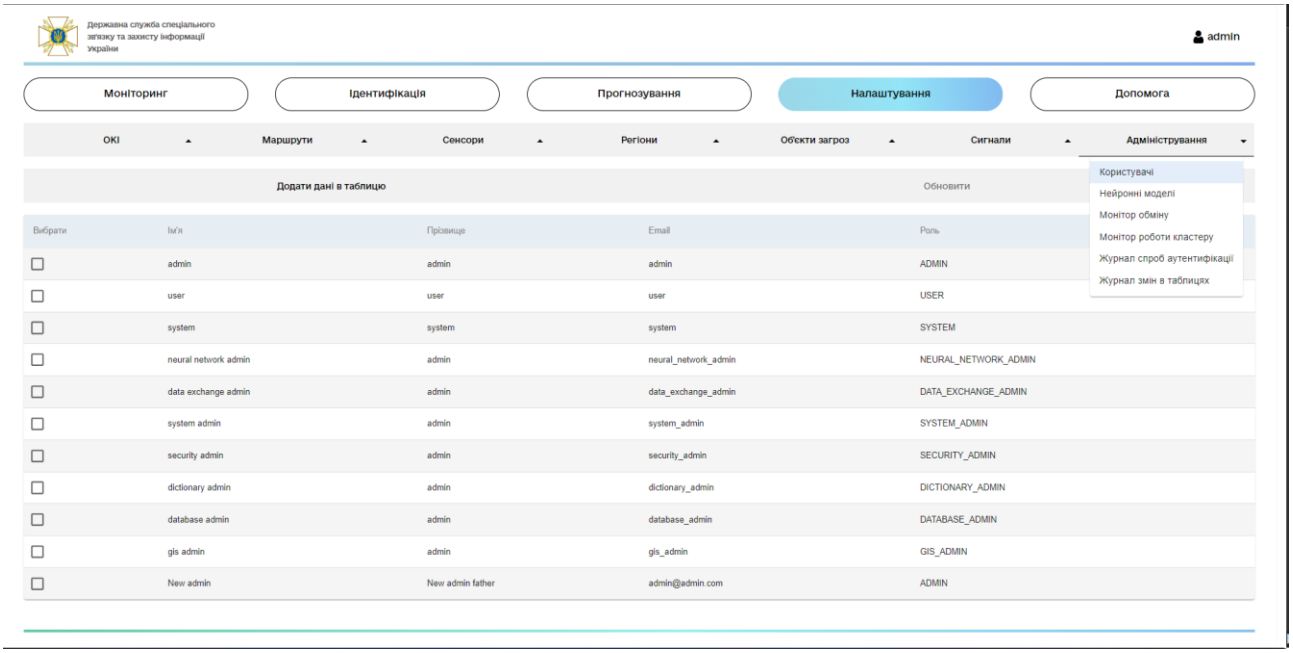

Рисунок 4.30 – Меню «Ідентифікація»

• Адміністратор безпеки (Security Administrator) - ідентифікація, яка не має доступу до збереження даних у таблиці. Немає можливості відкриття розділу сигналів. Не має доступу до адміністрування таких розділів та функцій, як нейронні моделі, монітор обміну та монітор роботи кластера. Також не має доступу до редагування інших довідників.

• Адміністратор довідників (Dictionary Administrator) - ідентифікація, яка не має доступу до спектрограми та збереження даних у таблиці. Не може відкривати розділ сигналів. Не має доступу до розділу адміністрування. Також не має можливості редагувати інші довідники.

• Адміністратор (Administrator) - загальний тип адміністратора.

• Адміністратор баз даних (Database Administrator) - адміністратор, який керує базами даних.

• Адміністратор ГІС (GIS Administrator) - адміністратор геоінформаційної системи

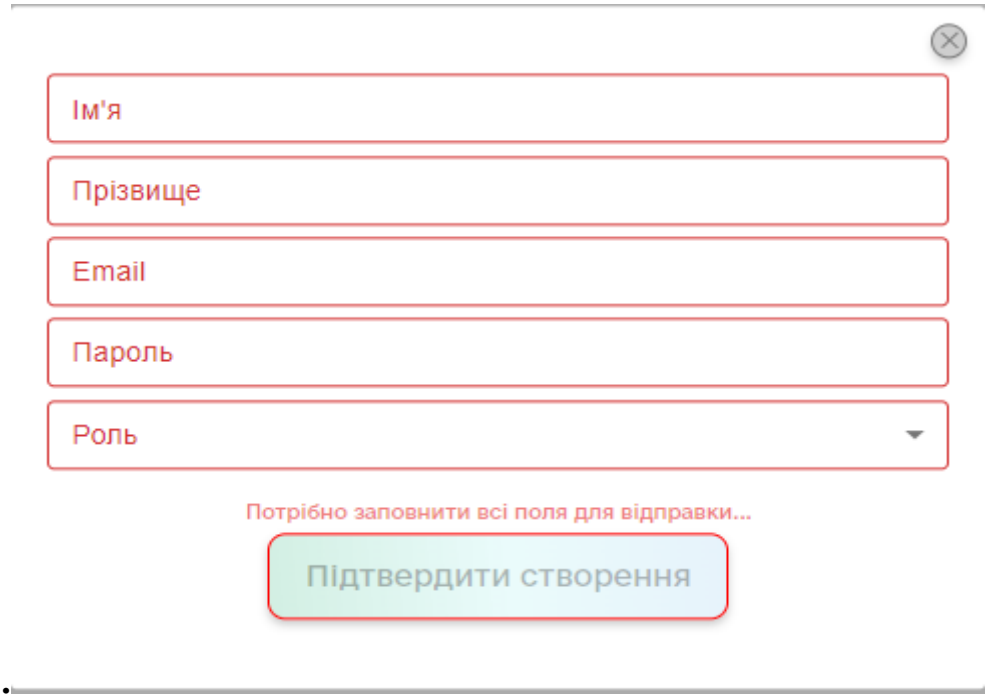

Рисунок 4.31 – Меню створення ролі з незаповненими полями

Для створення нової ролі у таблиці, слід використовувати кнопку "Додати дані в таблицю". Це спричинить появу вікна введення даних, де всі поля обов'язкові до заповнення (Рис. 4.32). Після внесення даних кнопка "Підтвердити створення" активується, і нова роль буде успішно створена.

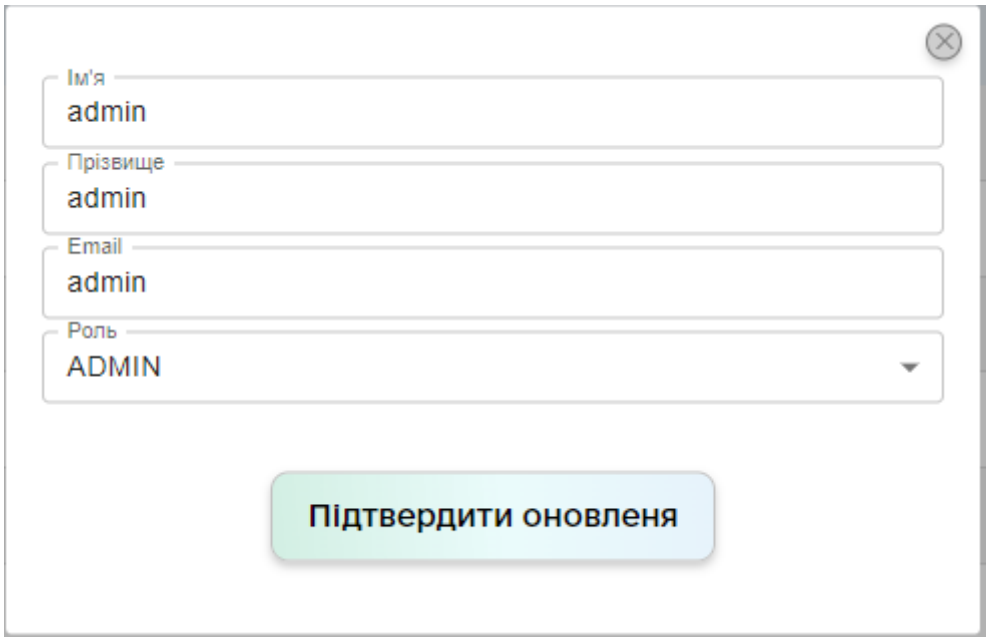

Рисунок 4.32 – Меню створення ролі з заповненими полями

Перейшовши у підменю нейронні моделі та пройшовши авторизацію користувач потрапить у вікно нейронних моделей (Рис.4.33).

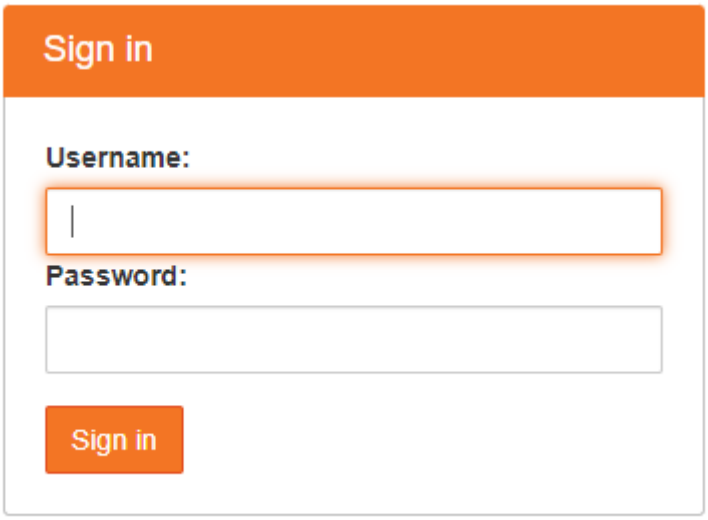

Рисунок 4.33 – *Авторизація вікна «Нейронні моделі»*

Відкривається додаток «Jupyter Notebook» (Рис.4.34). Меню з ліва висвітлює списки файлів. В основному вікні відображаються скрипти та нейронні моделі доступні для перегляду.

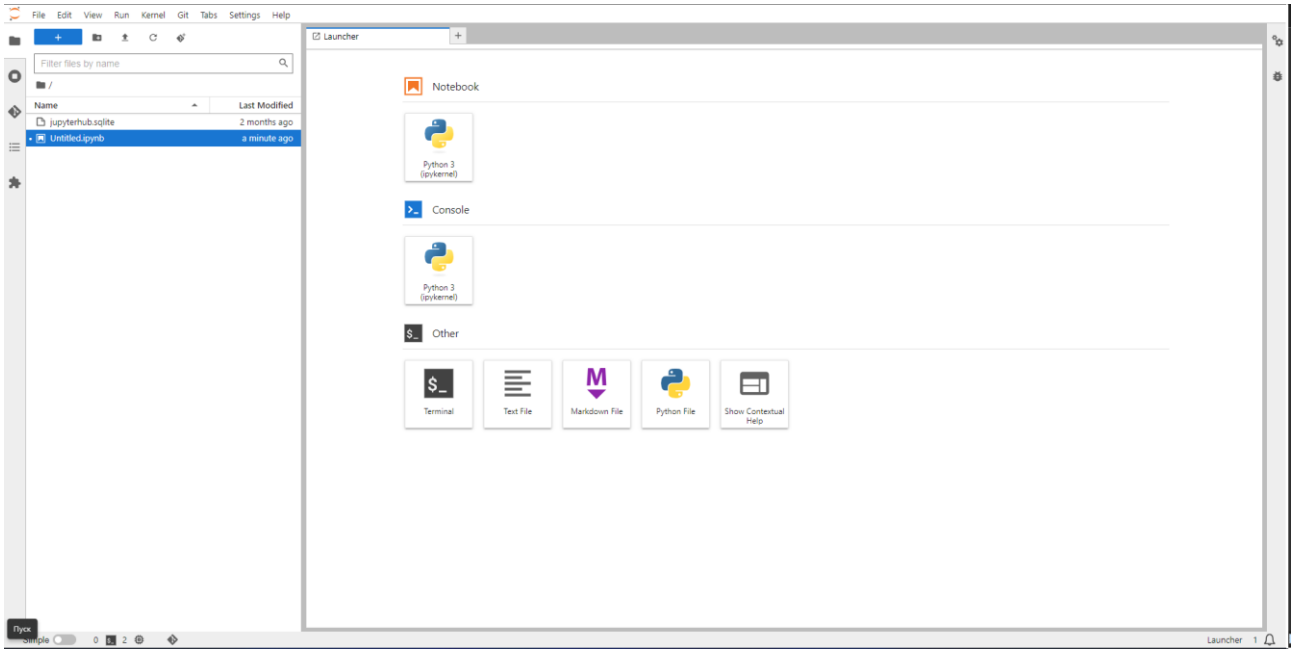

Рисунок 4.34 – Вікно «Jupyter Notebook»

Рисунок 4.35 – Підменю «Монітор обміну»

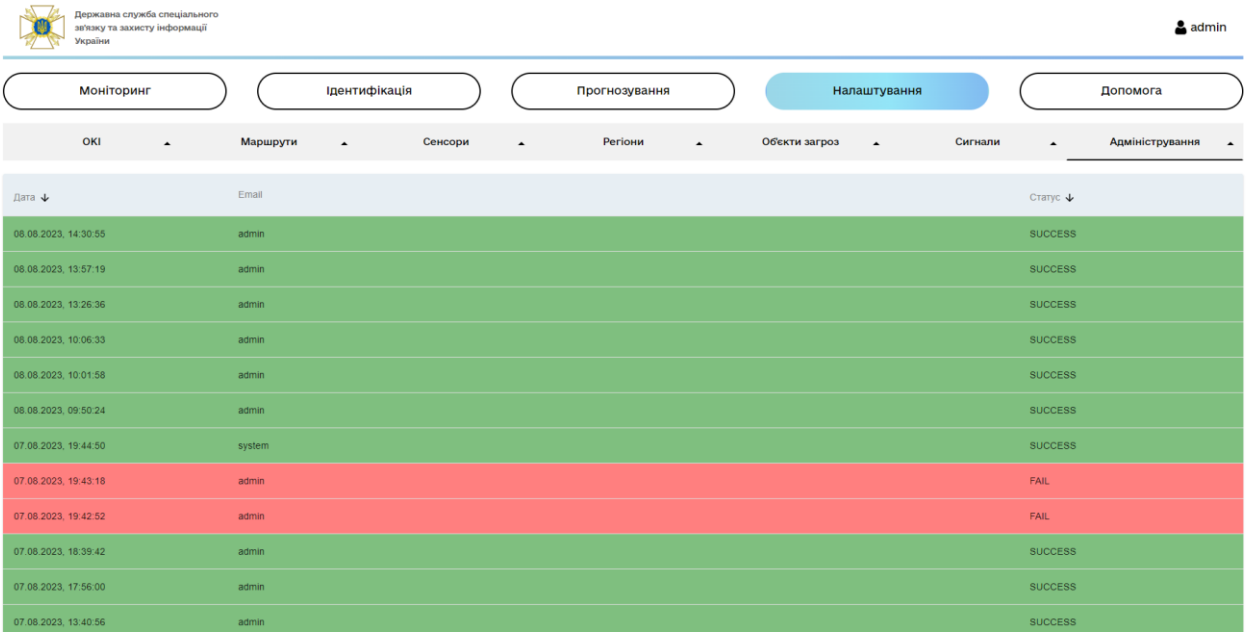

Рисунок 4.36 – Підменю «Журнал спроб аутентифікації»

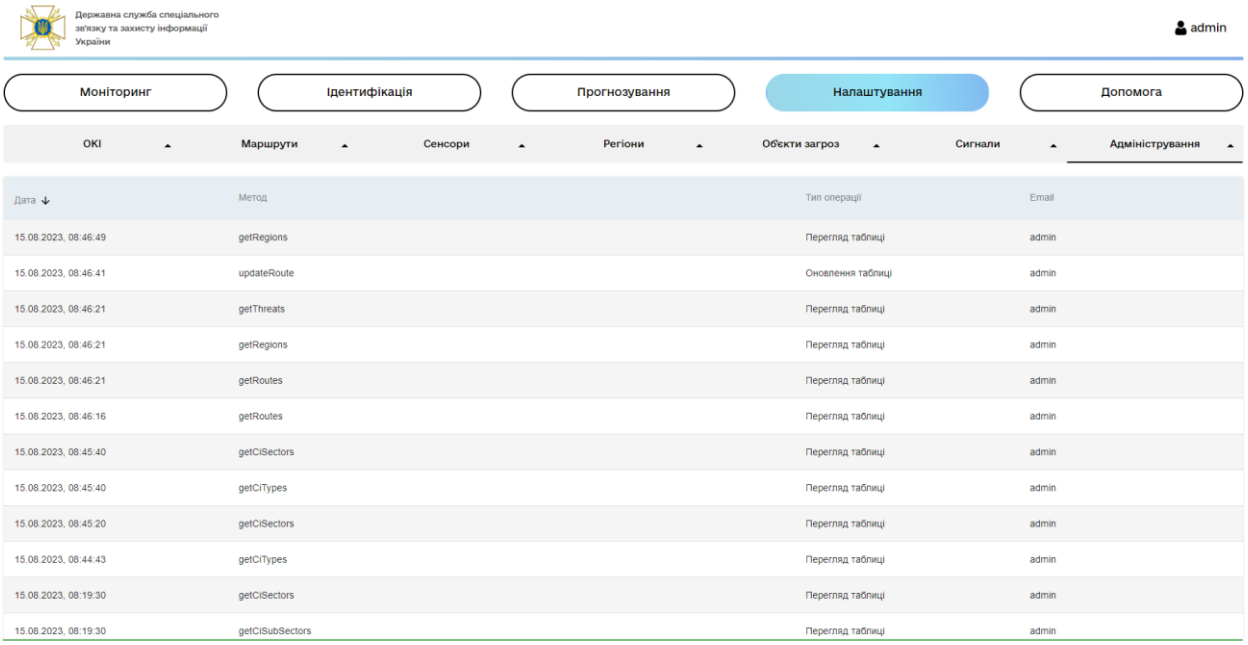

Рисунок 4.37 – Меню «Адміністрування» підменю «Журнал змін в таблицях»

У розділі "Допомога програмного інтерфейсу" забезпечується користувачам інформація та ресурси для отримання додаткової підтримки, відповідей на запитання та вирішення можливих проблем (Рис.4.38).

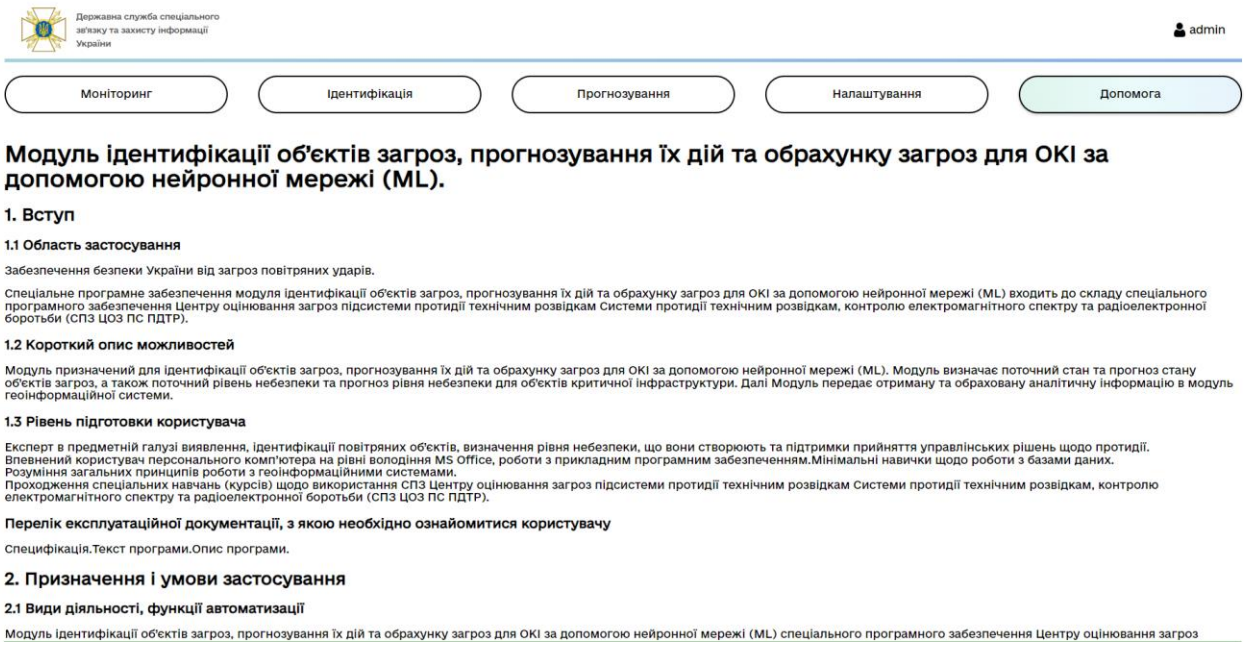

Рисунок 4.38 – Розділ «Допомога»

Цей розділ містить додаткову інформацію щодо таких аспектів:

• Модуль ідентифікації об'єктів та прогнозування за допомогою нейронної мережі (ML).

• Модуль геоінформаційних додатків (GIS) – опис функцій модуля, який спрямований на візуалізацію картографічної інформації.

• Реакція на аварійні ситуації – алгоритми дій у випадку збоїв чи інших критичних подій у системі.

ВИСНОВКИ

Результатом кваліфікаційної роботи є розроблена система для моніторингу та аналізу мережевого трафіку, заснована на методі аналізу та прогнозування кількісної інформації про мережевий рух, який був розроблений у рамках цієї роботи.

У ході дослідження були оцінені наявні рішення, визначені їхні основні недоліки, зокрема у складності роботи з часовими даними. Під час проектування системи була розроблена її архітектура, вибрана СУБД PostgreSql, описана структура бази даних, включаючи основні сутності та їх зв’язки. Були проведені порівняльні аналізи різних типів нейронних мереж, виявлені їхні сильні та слабкі сторони, і обрано найбільш придатний для розв'язання поставленої задачі тип мережі LSTM.

Архітектура системи описує взаємодію ключових вузлів: балансувальника, проксі-сервера, головного серверу додатку, бази даних головного додатку, додатку для агрегації запитів, бази даних для цього додатку та додатку для моніторингу та аналізу запитів.

Для розробки використовувалися різноманітні інструменти, такі як: Docker для налаштування середовища, Pip для керування залежностями, Git для зберігання кодової бази проекту та PyCharm для розробки та керування проектом.

Перспективним є розвиток додатку для агрегації запитів та створення моделі рекурентної нейронної мережі з довгою короткочасною пам’яттю для прогнозування навантаження веб-орієнтованих високонавантажених систем.

Додаток А. ДЕМОНСТРАЦІЙНИЙ МАТЕРІАЛ (ПРЕЗЕНТАЦІЯ)