НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Система керування роботизованою платформою інспекції інфраструктурних об'єктів»

Виконала:
студентка IV курсу, групи ІК-13
Ємцева Валерія Вікторівна _____

Керівник:
Асистент кафедри ІСТ,
Драган Михайло Сергійович _____

Рецензент:
доцент кафедри ІПІ, к.т.н., доцент,
Ліщук Катерина Ігорівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студентці
Ємцевій Валерії Вікторівні

1. Тема проєкту «Система керування роботизованою платформою інспекції інфраструктурних об’єктів», керівник проєкту Драган Михайло Сергійович, затверджені наказом по університету від «23» травня 2025 р. № 1705-с

2. Термін подання студентом проєкту: 09.06.2025

3. Вихідні дані до проєкту:

4. Зміст пояснювальної записки: аналіз предметної області, огляд існуючих рішень, визначення вимог та проектування архітектури системи, вибір технологій розробки, розроблення системи, загальні висновки.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо) діаграма розгортання, діаграма послідовності, діаграма компонентів, діаграма прецедентів.

6. Дата видачі завдання 05.03.2025

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Огляд завдання на дипломний проект	07.04.2025	
2	Аналіз предметної області та існуючих рішень	14.04.2025	
3	Вибір засобів та технологій розробки	21.04.2025	
4	Проектування архітектури системи	30.04.2025	
5	Розроблення та тестування програмного застосунку	19.05.2025	
6	Створення звіту дипломного проекту	30.05.2025	
7	Подання дипломного проекту на основний захист	02.06.2025	
8	Подання дипломного проекту рецензенту	09.06.2025	
9			

Студентка Валерія ЄМЦЕВА

Керівник Михайло ДРАГАН

АНОТАЦІЯ

Система керування роботизованою платформою інспекції інфраструктурних об’єктів.

Проєкт містить 63 с. тексту, 17 рисунків, 3 таблиці, посилання на 17 літературних джерел, додаток А та 4 конструкторських документів.

ІНСПЕКЦІЯ ІНФРАСТРУКТУРНИХ ОБ’ЄКТІВ, КЕРУВАННЯ РОБОТАМИ, УЛЬТРАЗВУКОВІ СЕНСОРИ, КОМП’ЮТЕРНИЙ ЗІР, МАШИННЕ НАВЧАННЯ, ОБРОБКА ЗОБРАЖЕНЬ, LIDAR, МОБІЛЬНІ РОБОТИ, НЕЙРОННІ МЕРЕЖІ.

Об’єктом розробки є система керування роботизованою платформою інспекції інфраструктурних об’єктів.

Мета розробки – розроблення єдиної веб-орієнтованої інформаційно-аналітичної системи моніторингу та управління роботизованою платформою, яка спрощує інспекцію інфраструктурних об’єктів.

У дипломному проєкті розроблено фрагменти системи керування інспекцією інфраструктурних об'єктів, а саме: верхній рівень інформаційної системи моніторингу та аналітики, а також підсистема обробки даних, що надходять від роботизованих платформ. Було реалізовано вебінтерфейс для відображення поточного стану об'єктів, інтерактивної карти розташування та аналітичних звітів про стан конструкцій..

Проведено ретельний аналіз способів збору, зберігання та обробки даних з сенсорів, а також побудовано модель машинного навчання для оцінки рівня пошкоджень. Значну увагу приділено зв'язку між апаратною частиною (роботами, сенсорами) та серверною частиною системи, а також вибору інфраструктури для безперервного моніторингу та прогнозування технічного стану об’єктів.

Отримані результати можуть бути корисними для впровадження автоматизованих систем інспекції на підприємствах, що займаються обслуговуванням мостів, будівель, пам’ятників та інших критично важливих споруд, а також для створення подібних систем у суміжних галузях.

SUMMARY

Control system of robotic platform of infrastructure facilities inspection.

The project contains 63 pages. text, 17 figures, 3 tables, links to 17 literary sources, annexes A and 4 design documents.

Keywords: infrastructure inspection, robot control, autonomous navigation, ultrasonic sensors, computer vision, machine learning, image processing, LIDAR, mobile robots, neural networks.

The object of development is a control system for a robotic platform for the inspection of infrastructure facilities.

The purpose of the development is to create a single web-based information and analytical system for monitoring and managing a robotic platform that simplifies the inspection of infrastructure facilities.

The diploma project developed fragments of the infrastructure inspection management system, namely: the upper level of the monitoring and analytics information system, as well as the subsystem for processing data coming from robotic platforms. A web interface was implemented to display the current status of objects, an interactive location map, and analytical reports on the condition of structures.

A thorough analysis of the methods of collecting, storing, and processing data from sensors was conducted, and a machine learning model was built to assess the level of damage. Considerable attention was paid to the connection between the hardware part (robots, sensors) and the server part of the system, as well as the choice of infrastructure for continuous monitoring and forecasting the technical condition of objects.

The results obtained may be useful for the implementation of automated inspection systems at enterprises engaged in the maintenance of bridges, buildings, monuments, and other critically important structures, as well as for the creation of similar systems in related industries.

.

№ рядка	Формат	Позначення	Найменування	Кіл. аркушів	№ екз.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІК13.080БАК.006 ПЗ	Система керування роботизованою	63		
6			платформою інспекції			
7			інфраструктурних об’єктів.			
8			Пояснювальна записка			
9	А3	ІК13.080БАК.006 Д1	Система керування роботизованою	1		
10			платформою інспекції			
11			інфраструктурних об’єктів.			
12			Діаграма розгортання			
13	А3	ІК13.080БАК.006 Д2	Система керування роботизованою	1		
14			платформою інспекції			
15			інфраструктурних об’єктів.			
16			Діаграма прецедентів			
17	А3	ІК13.080БАК.006 Д3	Система керування роботизованою	1		
18			платформою інспекції			
19			інфраструктурних об’єктів.			
20			Діаграма послідовності			
21	А3	ІК13.080БАК.006 Д4	Система керування роботизованою	1		
22			платформою інспекції			
23			інфраструктурних об’єктів.			
24			Схема алгоритму реєстрації та			
25			авторизації користувача			
26						
27						
28						

Зм.	Лист	№ докум.	Підпис		ІК13.080БАК.006 ТП		
Розробив		Ємцева			Система керування роботизованою платформою інспекції інфраструктурних об’єктів. Відомість дипломного проєкту	Літ.	Арк. 1 / Аркушів 1
Перевірив		Драган				Т	
						КПІ ім. Ігоря Сікорського Група ІК-13	
Затв.							

Пояснювальна записка
до дипломного проєкту
на тему: «Система керування роботизованою платформою інспекції інфраструктурних об’єктів»

Київ – 2025 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

API – Application Programming Interface

HTTP – Hypertext Transfer Protocol

IoT – Internet of Things

JSON – JavaScript Object Notation

JWT – JSON Web Token

REST – Representational State Transfer

UX – User Experience

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та робототехнічних систем якість і безпека інженерних споруд стають критично важливими. Особливо це стосується мостів, доріг, залізничних шляхів, будівель та інших інфраструктурних об’єктів, які піддаються значним навантаженням і зношенню. В Україні питання своєчасного виявлення дефектів загострилося через руйнування, спричинені військовим конфліктом, що створює додаткові виклики для підтримки життєздатності критично важливих споруд. Існуючі методи інспекції, засновані переважно на візуальному огляді або стаціонарних системах моніторингу, не завжди забезпечують достатню точність і оперативність в екстремальних або віддалених умовах.

Аналіз літературних джерел (монографії вітчизняних і закордонних дослідників з питань безпеки інфраструктури, звіти українських дорожніх компаній та міжнародні стандарти ASTM, ISO) свідчить про зростаючий інтерес до використання мобільних роботизованих платформ для автоматизованої інспекції. Досвід провідних фірм (Boston Dynamics, DJI, Trackunit) показує, що поєднання автономного навігаційного керування, сенсорного збору даних (ультразвукових, лазерних, оптичних) та алгоритмів машинного навчання дозволяє виявляти тріщини, корозію та інші дефекти на ранніх стадіях. Проте в Україні немає єдиного комплексного рішення, адаптованого до вітчизняних умов експлуатації й фінансових обмежень.

Таким чином, обґрунтована необхідність розробки та впровадження інноваційної системи керування роботизованою платформою для автоматизованої інспекції інфраструктурних об’єктів в українських реаліях.

Об’єктом дослідження є процес автоматизованої інспекції інфраструктурних об’єктів за допомогою роботизованої платформи.

Предметом дослідження є методи та технології керування роботизованою платформою для моніторингу стану інфраструктури, включаючи апаратні та програмні засоби, алгоритми автономного пересування і збору даних.

Мета дослідження полягає у розробці системи керування роботизованою платформою, яка буде застосовуватись для інспекції інфраструктурних об’єктів, з можливістю автономної або дистанційної роботи та збору даних про стан конструкцій.

Завданнями дослідження є:

– проведення аналізу існуючих мобільних інспекційних систем і визначення їхніх переваг та недоліків;

– формування функціональних та нефункціональних вимог до ПЗ;

– розроблення загальної архітектури системи керування та обробки даних;

– реалізація алгоритмів попередньої обробки та класифікації дефектів за допомогою методів машинного навчання;

– проведення тестування працездатності системи;

– внесення коректив для покращення функціональності та надійності системи.

Розроблена система може бути використана:

– державними службами інфраструктурного моніторингу (Укравтодор);

– будівельними та дорожніми компаніями для планових та аварійних оглядів;

– приватними інспекційними фірмами, що надають послуги з контролю безпеки об’єктів;

– науково-дослідними установами для збирання великих даних і подальшого прогнозування зношення конструкцій.

Дипломний проект складається з таких розділів:

– вступ – обгрунтування тематики, мета, завдання, сфера застосування, структура роботи;

– огляд існуючих рішень – зміст основних рішень, аналіз зарубіжного та вітчизнянного досвіду;

– постановка задачі – визначення функціональних та нефункціональних вимог, опис архітектури та вибір програмних модулів;

– вибір технологій розробки – визначення найкращих методів реалізації програмного забезпечення та обґрунтування вибору;

– розроблення системи – опис розроблення ПЗ, алгоритмів аналізу даних, обробку на серверній частині та вигляд клієнтської частини системи.

Обґрунтування основних проєктних рішень у розробці системи керування роботизованою платформою інспекції інфраструктурних об’єктів базується на необхідності забезпечити надійність, гнучкість та ефективність у реальних умовах експлуатації. З точки зору архітектури системи було реалізовано клієнт-серверну модель, яка передбачає веб-інтерфейс для віддаленого контролю, а також локальну обробку даних безпосередньо на платформі за допомогою вбудованого одноплатного комп’ютера. Це рішення дозволяє зменшити залежність від постійного інтернет-з’єднання та забезпечити швидку реакцію системи на події в реальному часі.

Графічна частина містить 4 креслення у форматі А3. Загальний обсяг роботи становить 63 сторінки тексту, 17 позицій списку використаних джерел та 1 додаток.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

У сучасних умовах війни та швидкої деградації інфраструктури в Україні особливо гостро постає питання регулярного та глибокого моніторингу технічного стану об’єктів. Пошкодження мостів, шляхопроводів, водогонів, енергетичних мереж і будівельних споруд мають як безпосередній економічний – відновлення потребує мільйонних вкладень – так і соціальний вимір: від порушення логістичних ланцюгів до ризику людських життів. Тому впровадження інтелектуальних систем інспекції здатне не лише знизити витрати, а й забезпечити безперервність роботи критично важливих об’єктів.

По-перше, ручний огляд пошкоджень у зонах активних бойових дій чи на важкодоступних спорудах часто неможливий через мінну небезпеку, руйнування під’їзних шляхів або нестабільну обстановку. Навіть у мирний час такі операції вимагають значних людських ресурсів і часу, тому їхня періодичність може бути недостатньою для своєчасної діагностики. Натомість мобільні роботизовані платформи, обладнані камерами високої роздільності, ультразвуковими сенсорами й LIDARом, можуть здійснювати детальне сканування поверхні споруди, виявляючи тріщини шириною до 0,1 мм та глибокі ушкодження матеріалу.

По-друге, інтеграція комп’ютерного зору та глибинного навчання відкриває можливості автоматизованої класифікації дефектів за їх характером (корозійні плями, осередки руйнування бетону, деформаційні зазори). Платформи на основі Convolutional Neural Networks здатні не лише фіксувати наявність дефекту, а й оцінювати його потенційну динаміку росту. Це дає змогу перейти від реактивного – проведення ремонту після виявлення аварії – до проактивного обслуговування: ремонтні бригади отримують прогнозовані карти пошкоджень із термінами, коли дефект може стати критичним.

По-третє, технології Інтернету речей (IoT) та хмарні сервіси забезпечують централізоване збирання, агрегацію та візуалізацію великих обсягів даних у режимі реального часу. Після кожної інспекції робот надсилає на сервер пакет інформації:

ширину та довготу частини ділянки, що перевірялась, температуру, дані корозії металу, ширину тріщин та інші. Спеціалізована веб-панель відображає зміни у вигляді діаграм, мап руху роботів та окремі звіти моніторингу об’єктів.

Таким чином, розроблення універсальної системи керування роботизованою платформою для інспекції інфраструктурних об’єктів, що поєднає автономну навігацію, адаптивні алгоритми аналізу дефектів і хмарні технології обробки даних, є не просто технічним завданням, а ключовим інструментом підвищення безпеки й стійкості національної економіки в умовах підвищеного ризику та обмежених ресурсів.

1.2 Постановка задачі

Метою даного дипломного проєкту є розроблення системи керування робототехнічною платформою для дослідження інфраструктурних об'єктів, яка дозволить автоматизувати процеси збору та аналізу інформації про стан об’єктів, підвищити точність моніторингу та зменшити людську участь у небезпечних процесах. Для досягнення цієї мети необхідно вирішити ряд конкретних завдань:

– аналіз існуючих рішень. Провести детальний огляд та аналіз існуючих систем для досліджень інфраструктурних об’єктів, включаючи механічні та автоматизовані рішення. Визначити переваги та недоліки кожного з рішень, а також оцінити їх функціональні можливості;

– визначення вимог до системи. Визначити функціональні та нефункціональні вимоги до системи керування роботизованою платформою. Врахувати потреби кінцевих операторів та аналітиків у зручному, безпечному та зрозумілому інструменті для виконання досліджень;

– розроблення архітектури системи. Розробити архітектуру системи, що включає всі необхідні компоненти для забезпечення ефективного керування роботизованою платформою та збором даних. Визначити методи післяобробки отриманих результатів з датчиків;

– розроблення панелі результатів. Створити зручний інтерфейс панелі результатів досліджень, що дозволить легко переглядати та самостійно робити висновки із зібраних даних;

– тестування та оцінка продуктивності. Провести тестування розробленої системи для оцінки її продуктивності, надійності та зручності використання.

1.3 Аналіз предметної області

Зношення матеріалів будівель та інфраструктурних об’єктів – це природний процес, який відбувається під впливом різних факторів, таких як механічні навантаження, погодні умови, хімічний вплив та біологічне руйнування (рисунок 1.1).

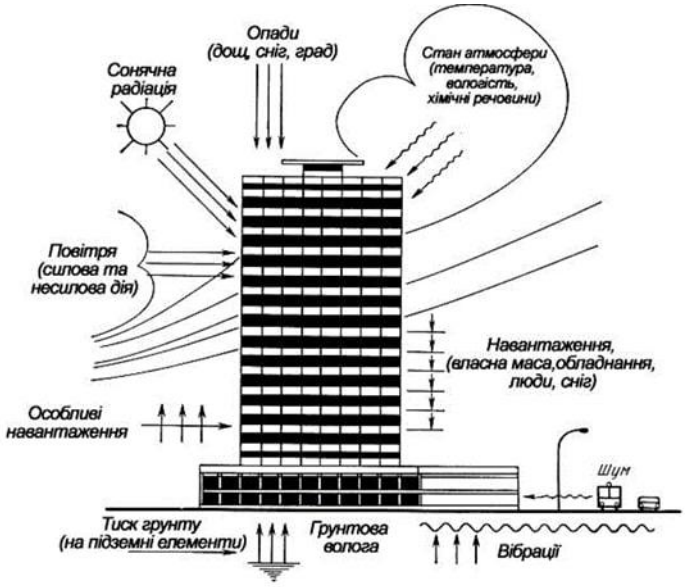

Рисунок 1.1 – Зовнішні впливи на інфраструктурний об’єкт [1]

З основних типів зношень можна виділити наступні:

– механічне зношення. Воно виникає унаслідок динамічних навантажень, ударів або транспортних ударів. Наприклад, мости постійно піддаються механічним навантаженням, що провокує утворення тріщин;

– корозія металів та лужна корозія. Корозія доволі поширена в наших регіонах, адже ми маємо великий вплив вологи на усі інфраструктурні об’єкти протягом року

(дощі, сніги). Також одною з причин її виникнення може бути вплив солей та брудного повітря, наприклад, в промислових районах. Прикладом такого впливу є корозія сталевих елементів мостів від постійного контакту з водою;

– тріщини в будівельних конструкціях. Найочевиднішою причиною їх виникнень є сейсмічна активність. Частими її осередками є Одеська область, Кримський півострів, Закарпаття та Прикарпаття. Також з причин слід зазначити температурні перепади, перевантаження та усадку будівель;

– біологічне руйнування. Одними з прикладів таких явищ є цвіль, грибки та різні бактерії, що виникають внаслідок високої вологості та слабкої або повної відсутності вентиляції. Найпоширенішими проявами є пошкодження дерев'яних конструкцій грибками та комахами-шкідниками;

– катастрофічне руйнування. Такі пошкодження є раптовими та, зазвичай, критичними для інфраструктурних об'єктів. Вони виникають через військові конфлікти, несучи за собою механічні пошкодження, термічне руйнування та пошкодження несучих конструкцій.

Роботизовані платформи підвищують безпеку людей, адже руйнування завжди створюють небезпечні умови для інспекційних груп у вигляді ймовірних обвалів конструкцій або ж викиду небезпечних речовин. У деякі ділянки об'єктів неможливо дістатись без ризику для життя чи за без допомоги складних будівельних конструкцій.

Основними технологіями, що використовуються в таких системах, є дрони, наземні автономні роботи, датчики IoT, тепловізори та системи обробки даних на базі штучного інтелекту.

Одним з ключових елементів сучасних систем моніторингу є застосування алгоритмів комп'ютерного зору та методів машинного навчання. Вони дозволяють аналізувати великі масиви зображень, отриманих з камер та сенсорів, і виявляти дефекти, тріщини, корозію або інші пошкодження інфраструктури. Крім того, методи SLAM (Simultaneous Localization and Mapping) дозволяють роботам самостійно орієнтуватися в просторі, будувати карти місцевості та уникати перешкод.

Висновки до розділу 1

У ході опису предметної області було розглянуто актуальність проблеми моніторингу інфраструктурних об’єктів та доцільність застосування роботизованих платформ для їх інспекції. Було виявлено, що значна частина критичної інфраструктури в Україні зазнала пошкоджень, і традиційні методи огляду не завжди дозволяють оперативно оцінити їхній стан. Роботизовані системи можуть забезпечити швидке та ефективне виявлення дефектів, мінімізуючи ризики для персоналу.

Було визначено основні задачі дослідження, які включають аналіз існуючих рішень, розробку архітектури системи, інтеграцію датчиків збору інформації, створення алгоритмів автономного переміщення та віддаленого керування, а також тестування системи.

Роботизовані платформи не замінюють людей у процесі дослідження пошкоджених об’єктів, але значно розширюють їхні можливості. Проведений аналіз предметної області та постановка задачі показали, що вони гарантують безпечність, швидкість та точність збору інформації. Впровадження такої системи дозволить застосовувати передові методи аналізу для глибшого розуміння стану конструкцій.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

2.1 Програмне рішення «Spot»

«Spot» – це автономний чотириногий робот (Рисунок 2.1), розроблений компанією Boston Dynamics, який використовується для інспекції промислових об'єктів, мостів, тунелів та будівель. Завдяки високій мобільності та підтримці різних сенсорів (LiDAR, тепловізори, камери високої роздільної здатності), він дозволяє збирати дані в складних умовах, куди людині потрапити складно або небезпечно.

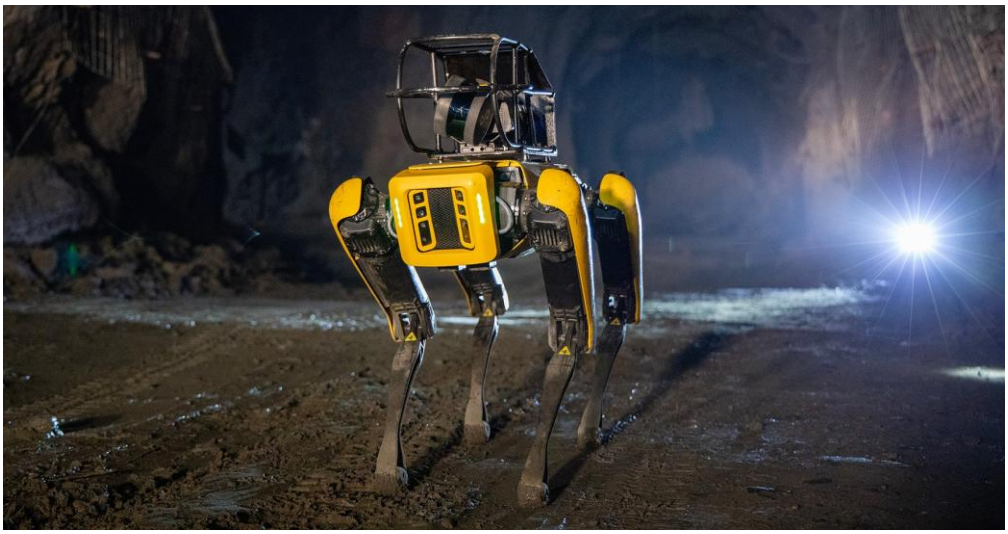

Рисунок 2.1 – Візуальне зображення робота «Spot» [2]

Він має наступні переваги:

– автономність та мобільність. Spot здатен автономно пересуватись нерівною місцевістю, наприклад, сходами, або ж у вузьких та небезпечних для людини просторах;

– гнучка модульність. Spot дозволяє під’єднувати різні модулі: камери високої роздільності, тепловізори, LIDAR, мікрофони та багато інших, що дозволяє адаптувати його під специфіку конкретних задач;

– безпека. Завдяки використанню Spot, інспектори не змушені працювати в небезпечних умовах (висота, радіація, завали);

– інтеграція зі сторонніми ПЗ. Spot має відкритий SDK, що дозволяє інтегрувати його з платформами для обробки зображень, системами штучного інтелекту та цифрового моніторингу об’єктів;

Окрім цього має такі недоліки:

– висока вартість. Spot є дорогим продуктом не тільки для придбання, а й для обслуговування, що робить його недоступним для багатьох комунальних служб;

– потреба в навченому персоналі. Для роботи зі Spot потрібні спеціалісти, що вміють налаштовувати маршрути, обробляти отримані дані та підтримувати ПЗ з апаратною частиною;

– обмежений час роботи. Батарея Spot обмежена в часі роботи приблизно на 90 хвилин активного використання, що вимагає планування завдань або запасних акумуляторів;

– чутливість до погодних умов. Робот не завжди ефективний у складних погодніх періодах, таких як сніг, сильний вітер та дощ.

У висновку Spot від Boston Dynamics є високотехнологічною мобільною платформою, що дозволяє ефективно досліджувати стан інфраструктурних об’єктів у складних або небезбечних умовах. Незважаючи на високу вартість та певні технічні обмеження, його переваги у вигляді автономної навігації, збору даних у реальному часі та модульності роблять Spot потужним інструментом для інженерного моніторингу та технічного обслуговування об’єктів критичної інфраструктури.

2.2 EverSense

Система EverSense – це комплексне рішення для безперервного моніторингу стану конструкцій, що поєднує роботу різних типів сенсорів, каналів зв’язку, алгоритмів аналізу та хмарних технологій. Вона використовується для нагляду за мостами, будівлями, дамбами, тунелями, вітровими турбінами та навіть ядерною інфраструктурою. Основна функція полягає в інтеграції та обробці різних типів автоматизованих вимірювань для надання релевантних індикаторів та інформаційних

панелей для різних зацікавлених сторін. Загальний вигляд системи збору інформації представлений на рисунку 2.2

Рисунок 2.2 – Встановлення системи «EverSense» [3]

Система має наступні переваги:

– комплексний моніторинг. EverSense використовує різноманітні сенсори для збору даних про деформації, вібрації, температурні зміни та інші параметри, що забезпечує повний огляд об’єкту;

– безперервний збір даних. Система працює в режимі реального часу, надаючи актуальну інформацію про стан конструкцій та дозволяє оперативно реагувати на виявлені відхилення;

– точність. Завдяки використанню сучасних технологій та високочутливих сенсорів, EverSense забезпечує точні вимірювання та мінімізує ймовірність хибних спрацювань;

– інтеграція з іншими системами. EverSense може бути легко інтегрована з існуючими платформами управління інфраструктурою, що може спрощувати процес її впровадження.

А також такі недоліки:

– вартість. Інсталяція та налаштування системи можуть вимагати значних фінансових вкладень;

– складність обслуговування. Для коректної роботи системи необхідно регулярне технічне обслуговування та калібрування сенсорів, що потребує залучення кваліфікованого персоналу;

– залежність від електроживлення. Система потребує стабільного електропостачання та надійного зв’язку для передачі даних, що може бути проблематичним у віддалених або важкодоступних районах;

– неавтономність. EverSense є стаціонарною платформою, що фіксується на самій конструкції. Вона не замінює дронів чи роботів для огляду нових або складнодоступних ділянок.

У висновку, система від є потужним інструментом для моніторингу стану інфраструктурних об’єктів, забезпечуючи комплексний та безперервний збір даних з високою точністю. Незважаючи на вартість впровадження та необхідність регулярного обслуговування, система добре підходить для об’єктів, де важливе своєчасне виявлення дефектів, прогнозування аварійних ситуацій та збереження безпеки на довготривалу перспективу.

2.3 PipeWalker

PipeWalker (також відома як PureRobotics) – це високотехнологічна модульна роботизована платформа–повзун, призначена для неруйнівної діагностики та оцінки технічного стану напірних і безнапірних трубопроводів водо- та водовідведення. Платформа здатна долати до 2,9–5 км від точки входу (залежно від конфігурації) і працювати як у заповнених водою, так і в зневоднених трубах. PipeWalker транспортує набір датчиків і інструментів, серед яких HD-камери з панорамуванням і збільшенням, ультразвукові сенсори, LiDAR-сканери та електромагнітні модулі для виявлення корозії, тріщин і дефектів матеріалу в реальному часі. Побачити систему можна на рисунку 2.3.

Рисунок 2.3 – Зображення роботи PipeWalker [4]

Переваги:

– надійність. PipeWalker від Xylem створений для роботи у важких умовах трубопровідних систем, забезпечуючи стабільну роботи навіть у зношених чи складних ділянках;

– точність діагностики. Завдяки високочутливим сенсорам робот забеспечує детальне виявлення пошкоджень, витоків чи ознак зносу труб;

– гнучкість конструкції. Модульність та керованість конструкції PipeWalker покращує здатність адаптації до труб різного діаметру та складності маршруту;

– економія часу. Застосування робота значно скорочує час, необхідний для перевірки довгих ділянок трубопроводів у порівнянні з ручними методами.

Недоліки:

– неавтономність. PipeWalker потребує безпосереднього керування оператором, що обмежує рівень автоматизації та вимагає присутності фахівця під час інспекції;

– вартість. Висока ціна обладнання та витрати на його обслуговування можуть бути суттєвими;

– потреба в навчанні персоналу. Для ефективного використання необхідна спеціальна підготовка операторів, що вимагатиме додаткового часу та ресурсів.

PipeWalker є високоточним і надійним інструментом для інспекції трубопроводів, що значно підвищує ефективність та безпеку перевірок. Незважаючи на відсутність автономності та початкові витрати, його здатність виявляти потенційні проблеми на ранніх стадіях робить його корисним рішенням для компаній, що обслуговують об’єкти критичної інфраструктури.

Висновки до розділу 2

Розглянуті рішення - Spot від Boston Dynamics, EverSense та PipeWalker від Xylem – демонструють потужні можливості у сферах мобільної робототехніки, інспекції інфраструктури та моніторингу навколишнього середовища. Кожне з них має свої сильні сторони: автономність і мобільність у Spot, безперервний збір даних та компактність EverSense, а також точність та адаптивність PipeWalker. В той же час, ці рішення мають низку обмежень, таких як складність налаштування, обмежена масштабованість чи залежність від фахівців.

На відміну від розглянутих рішень, розроблена система буде більш гнучкою та легше підтримуваною. У роботі буде спроба покращити існуючі недоліки та розширити деякий функціонал, аби запропонувати більш універсальне та масштабоване рішення, орієнтоване на реальні потреби користувачів.

3 ПОСТАНОВКА ЗАДАЧІ

3.1 Вимоги до системи

Функціональні та нефункціональні вимоги – це два взаємодоповнювальні класи характеристик, за допомогою яких формалізують очікування від будь-якої інформаційної або керованої системи. У широкому розумінні, вимоги описують «що» має робити система та «яким чином» вона повинна це робити, дозволяючи замовнику, розробникам і тестувальникам чітко зрозуміти масштаб робіт і критерії успіху.

Функціональні вимоги відповідають за опис конкретних функцій і сервісів, які система обов’язково повинна реалізувати. Іншими словами, це деталізована специфікація тих дій, які користувачі або інші підсистеми зможуть виконувати через інтерфейси системи. Наприклад, авторизація користувача, обробка вхідних сигналів, збереження даних чи генерація звітів – усе це є функціональними вимогами. Вони служать основою для формування технічного завдання та розробки архітектури системи, бо дозволяють сформувати чіткий перелік можливостей, які необхідно закодувати, протестувати й задокументувати.

Нефункціональні вимоги задають якість виконання цих функцій, тобто визначають умови та межі, в яких система має ефективно працювати. Це вимоги до продуктивності (наприклад, час відгуку, пропускна здатність), надійності (відсоток безвідмовної роботи, механізми відновлення), безпеки (шифрування, контроль доступу), масштабованості, зручності використання й утримання (легкість адміністрування, підтримуваність коду). Нефункціональні вимоги допомагають забезпечити, щоб задоволені всі бізнес і технічні обмеження: наприклад, щоб система стала доступною цілодобово, змогла обробити великий потік даних або була захищена від несанкціонованого доступу.

Розділення вимог на ці дві категорії дозволяє учасникам проєкту чітко розуміти, які саме можливості потрібно реалізувати (функціональні), і які характеристики системи повинні бути гарантовані незалежно від навантаження та зовнішніх

впливів (нефункціональні). Такий підхід мінімізує ризики забуття важливих пунктів у технічному завданні, оптимізує процес розробки та пришвидшує тестування і валідацію готового рішення перед його впровадженням.

3.1.1 Функціональні вимоги

Функціональні вимоги визначають конкретні завдання та функції, які повинна виконувати система для задоволення потреб користувачів та адміністраторів. Система повинна гарантувати надійну автентифікацію кожного користувача перед наданням доступу до функціоналу, тому реалізується багаторівневий механізм перевірки особи: входити до особистого кабінету можна тільки після введення облікових даних або пред’явлення цифрового сертифікату. Такий підхід забезпечує захищеність даних і унеможливлює несанкціоноване використання системи. Після успішного входу користувач отримує інтуїтивно зрозумілий інтерфейс, в якому в режимі реального часу відображаються актуальні показники стану об’єктів інфраструктури. Дані надходять безпосередньо з сенсорів, встановлених на роботизованих платформах: температура, вологість, рівень корозії, ширина тріщин та інтенсивність вібрацій надходять на сервер обробки із заданою періодичністю, що дозволяє своєчасно виявляти позаштатні ситуації та реагувати на потенційні ризики.

Однією з ключових складових є картографічне відображення результатів інспекцій: кожен об’єкт позначається відповідним маркером на інтерактивній мапі, при наведенні курсору відображається коротка довідка з поточним рівнем пошкоджень. Це дозволяє користувачеві миттєво оцінити розподіл проблемних зон по всій території та вибудувати оптимальний маршрут для подальших оглядів або ремонтних робіт.

Аналітична складова базується на машинному навчанні: зібрані дані проходять попередню обробку та нормалізацію, після чого алгоритм прогнозує рівень пошкоджень та категоризує його за чіткими параметрами. Завдяки цьому створюється можливість не лише фіксувати поточний стан, але й прогнозувати розвиток

дефектів у майбутньому, що суттєво підвищує ефективність планування обслуговування й економить ресурси. Отримані результати агрегуються в підсумкові звіти, де кожен розділ містить детальні графіки, діаграми та рекомендації щодо усунення виявлених проблем.

Для довгострокового аналізу всі дані упорядковуються та зберігаються в єдиній базі – історія показників об’єктів доступна для порівняння поточних і минулих значень. Користувач може застосовувати фільтри за часовими періодами, щоб сконцентруватися саме на тих аспектах, що його цікавлять.

Заключним етапом роботи з системою є генерація офіційних звітів у форматі PDF, які включають огляд усіх зафіксованих дефектів та часові графіки розвитку пошкоджень. Такий документ відповідає корпоративним стандартам і готовий до передачі керівництву або підрядним організаціям, що виконуватимуть ремонтні роботи. Таким чином, впроваджена система об’єднує в собі високий рівень безпеки, оперативний моніторинг, потужну аналітику та гнучкі можливості звітування, що забезпечує всебічний контроль за станом критичних об’єктів інфраструктури.

Ці функціональні вимоги визначають основні можливості системи та забезпечують виконання основних задач, пов'язаних з управлінням користувачами у системі керування. Важливо врахувати всі ці вимоги на етапі проектування та розробки, щоб забезпечити надійну та зручну у використанні систему.

3.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики системи, які не пов'язані з конкретними функціями, але є важливими для загальної ефективності, надійності та зручності використання системи [5]. Вони охоплюють аспекти продуктивності, безпеки, масштабованості, зручності використання та підтримки. Основні нефункціональні вимоги включають:

– надійність – система повинна працювати стабільно при надходженні великої кількості даних (від кількох роботів одночасно).

– масштабованість – система повинна мати архітектуру, що дозволяє масштабування на більшу кількість об’єктів та користувачів.

– безпека – система повинна мати автентифікацію та авторизацію, шифрування збережених і переданих даних по мережі.

– зручність використання – інтерфейс користувача повинен бути інтуїтивно зрозумілим та простим у використанні для забезпечення швидкого навчання користувачів;

– продуктивність – час відгуку інтерфейсу не повинен перевищувати 2 секунд при переході між сторінками.

Ці нефункціональні вимоги є ключовими для забезпечення ефективної, надійної та безпечної роботи системи керування роботизованою платформою дослідження інфраструктурних об’єктів. Вони допомагають визначити загальні характеристики та параметри системи, які мають бути враховані на етапі проектування та розробки для досягнення поставлених цілей.

3.2 Проектування системи

Архітектура системи моніторингу інфраструктурних об’єктів за допомогою роботизованої платформи відіграє ключову роль у забезпеченні надійності збору, обробки та аналізу даних, що надходять від сенсорів. Вона забезпечує масштабованість, модульність і гнучкість у впровадженні нових функцій. Основними компонентами системи є серверна частина з API, база даних, веб-інтерфейс адміністратора, агент для аналізу пошкоджень на основі ML-моделі, а також сторонні пристрої (роботизовані платформи), які надсилають дані.

Серверна частина реалізована у вигляді Flask-застосунку, який приймає запити від веб-клієнтів, здійснює обробку даних, взаємодіє з базою даних та викликає функціональність ML-агента. Веб-інтерфейс дозволяє адміністраторам переглядати інформацію про інфраструктурні об’єкти, стан пошкоджень, отримувати рекомендації щодо ремонту, а також керувати даними користувачів, об’єктів та сповіщеннями.

Користувачі (інженери, аналітики, представники комунальних служб) взаємодіють із системою через веб-інтерфейс. Для доступу до захищених функцій системи реалізовано механізм автентифікації за логіном і паролем.

Компоненти системи та їх взаємодія представлені на діаграмі розгортання. Метою побудови діаграми розгортання є не стільки опис бізнес-логіки програми, скільки документування та планування інфраструктури. По-перше, вона допомагає чітко уявити, скільки й яких саме обчислювальних ресурсів потрібно для запуску всіх компонентів, і де їх найкраще розмістити. Розробники та DevOps-інженери можуть переглянути, які служби будуть співіснувати на одному сервері, а які слід винести на окремі вузли задля підвищення продуктивності або забезпечення відмовостійкості.

По-друге, наявність такої діаграми значно спрощує проєктування безпеки. Візуалізувавши межі кожного вузла й канали зв'язку між ними, команда може визначити, де слід застосувати шифрування, якими мають бути політики доступу та які вузли потребують додаткового захисту від несанкціонованих підключень.

Для розробника діаграма розгортання є цінним інструментом уже на стадії тестування та впровадження. Якщо під час оновлення виникають питання, які саме артефакти потрібно переносити на тестовий або продукційний сервер, – достатньо поглянути на діаграму. Це мінімізує ризик помилок конфігурації або некоректного розміщення компонентів. У подальшому ж, коли система масштабуватиметься і в неї додаватимуть нові сервіси, діаграма розгортання дозволить швидко побачити, де слід розгорнути додаткові вузли або як перерозподілити навантаження між існуючими серверами.

Висвітлення фізичної інфраструктури в такому форматі є також важливою частиною технічної документації, оскільки створює єдине джерело істини для всіх учасників проєкту: розробників, адміністраторів, DevOps-інженерів і навіть менеджерів. На його підставі простіше планувати бюджети на закупівлю обладнання, оцінювати терміни впровадження і узгоджувати процеси CI/CD, адже кожен розуміє, куди і які артефакти треба доставити та запустити. Таким чином, діаграма ро-

згортання стає невід'ємним елементом комплексного підходу до проєктування, розробки та підтримки будь-якої сучасної інформаційної системи. Розроблена діаграма для моєї системи зображена на кресленику ІК13.080БАК.006 Д1. На діаграмі представлені чотири основні компоненти: Frontend-сервер, Backend-сервер, Сервер бази даних та Роботизовані пристрої. Компоненти та зв'язки:

– Frontend-сервер – інтерфейс для кінцевого користувача, реалізований як HTML-сторінки, згенеровані за допомогою Flask + Jinja2. Комунікує з сервером через HTTP-запити;

– Backend-сервер – основний сервер, що оброблює запити, відповідає за бізнес логіку, взаємодіє з базою даних, файлом CSV та ML-моделлю для визначення масштабності пошкоджень. Оброблює запити фронтенду на порту 5000;

– Сервер бази даних – зберігає дані про об'єкти, користувачів, сенсорні показники, результати аналізу та рекомендації. Сервер взаємодіє з компонентом безпосередньо за допомогою порту 543;

– Роботизовані пристрої – джерело даних. Вони здійснюють іспекції об'єктів та передають дані у форматі CSV в систему.

3.2.1 Проектування бази даних

База даних має бути реляційною, так як ми будемо зберігати структуровані дані. Структуровані дані – це тип даних, які мають фіксовану схему, чітку організацію і легко піддаються зберіганню та обробці в базах даних. Вони зазвичай представлені у вигляді таблиць з рядками і стовпцями, де кожен рядок відповідає запису, а кожен стовпець – полю, що зберігає конкретний тип інформації. SQL (Structured Query Language) – це спеціалізована мова програмування, яка використовується для управління і маніпулювання структурованими даними в реляційних базах даних. SQL дозволяє виконувати різні операції з даними, такі як вставка, вибірка, оновлення та видалення даних, а також створення і управління базами даних та їх структурами [6].

У базі даних зберігається інформація про користувачів, інфраструктурні об’єкти та звіти від сенсорів роботизованих платформ, що надходять під час інспекції.

Виділимо основні сутності:

– користувач (User): містить дані про зареєстрованих користувачів системи, описаний у таблиці 3.1;

– об’єкт (Object): містить інформацію про інфраструктурні об’єкти (будівлі, мости, памятки тощо), які підлягають інспекції, описаний у таблиці 3.2;

– звіт сенсорів (SensorReport): містить дані кожного звіту, отриманого від роботизованої платформи щодо певного об'єкту та описані у таблиці 3.3.

Таблиця 3.1 – Сутність користувач

PK	ID	int not null
	email	varchar not null
	password	varchar

Таблиця 3.2 – Сутність об’єкт

PK	ID	int not null
	name	varchar not null
	address	varchar not null
	photo_url	varchar not null
	last_inspection_date	date not null

Таблиця 3.3 – Сутність звіт сенсорів

PK	ID	int not null
FK	object_id	int not null
	robot	varchar not null
	timestamp	datetime not null
	latitude	float
	longitude	float

	crack_width	float
	corrosion_level	float
	temperature	Float
	vibration	Float
	scan_progress	float
	predicted severity	varchar

На рисунку 3.2 представлено фізичну схему бази даних.

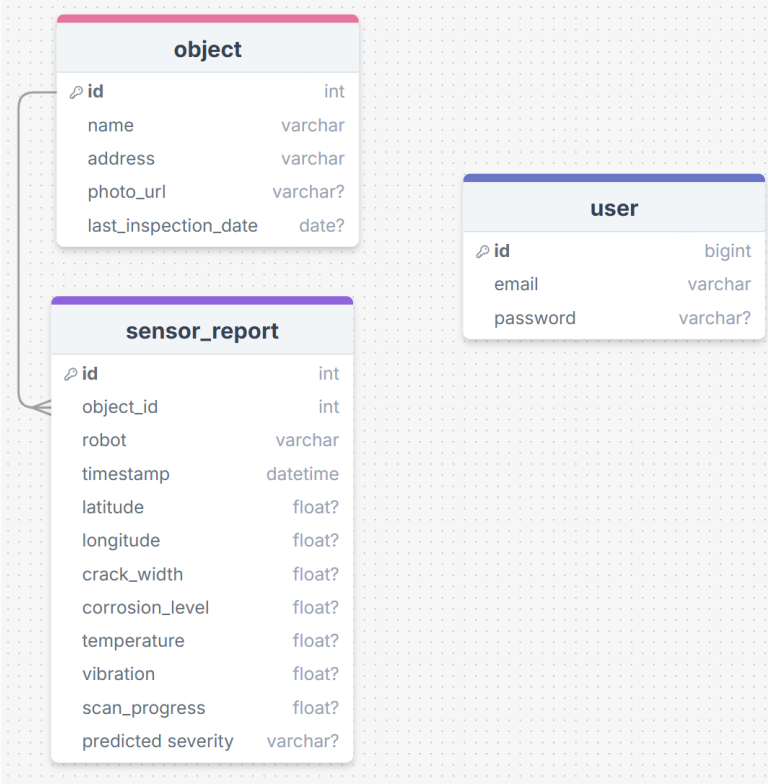

Рисунок 3.2 – ER-діаграма бази даних

Висновки до розділу 3

У результаті цього розділу було сформовано цілісне уявлення про вимоги та архітектуру системи моніторингу інфраструктурних об’єктів за допомогою робо-тизованих платформ.

Функціональні вимоги зосереджені на забезпеченні можливості керування користувачами, об'єктами інфраструктури та сенсорними звітами – включаючи додавання, редагування, перегляд детальної інформації та взаємодію з аналітичними модулями. Нефункціональні вимоги охоплюють аспекти надійності, безпеки, масштабованості, продуктивності та зручності інтерфейсу, що є критичними для стабільної роботи системи в реальному часі та ефективного прийняття рішень на основі зібраних даних.

Архітектура системи побудована на основі взаємодії між клієнтським інтерфейсом (UI), серверною частиною (Backend API), модулем машинного навчання для аналізу пошкоджень та базою даних SQLite. Проєктування включає створення компонентної та розгортальної діаграм, що демонструють логічну структуру системи та її технічну реалізацію. Проєктування бази даних передбачає створення таблиць для зберігання інформації про користувачів, об’єкти та звіти сенсорів, що забезпечує цілісне та ефективне управління даними в системі.

Таким чином, визначені вимоги й архітектурні рішення створюють надійну основу для подальшої розробки та впровадження системи, яка відповідатиме сучасним потребам у сфері автоматизованого моніторингу стану інфраструктури.

4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

4.1 Python

Мова програмування Python відіграє ключову роль у реалізації серверної частини та бізнес-логіки розробленої системи керування роботизованою платформою. По-перше, Python є інтерпретованою, об’єктно-орієнтованою високорівневою мовою з динамічною семантикою, що дозволяє розробникам швидко переходити від ідеї до робочого коду без додаткового етапу компіляції. Завдяки простому, лаконічному та інтуїтивно зрозумілому синтаксису, розроблення й підтримка системи суттєво прискорюються, а ризик помилок у коді зменшується.

Вбудовані структури даних – списки, кортежі, словники та множини – у поєднанні із динамічною типізацією забезпечують гнучкість при роботі з різними форматами вхідних даних та дозволяють без зайвих накладних витрат реалізовувати складні алгоритми обробки сенсорних показників. Інтерпретатор Python та його стандартна бібліотека доступні безкоштовно на переважній більшості операційних систем, що забезпечує крос-платформеність розгортання: від Linux-серверів у хмарі до вбудованих одноплатних комп’ютерів на борту роботизованої платформи.

Однією з ключових переваг Python є його багата екосистема сторонніх бібліотек і фреймворків. Для роботи з базами даних можна використати SQLAlchemy, для побудови REST-інтерфейсів – Flask або FastAPI, для черг повідомлень – Celery, для асинхронних операцій – asyncio, а для інтеграції машинного навчання – бібліотеки TensorFlow, PyTorch та scikit-learn. Така різноманітність дозволяє створювати всі компоненти системи в одній мові, що спрощує підтримку та розвиток проєкту, а також знижує бар’єр входу для нових членів команди.

Крім того, величезна та активна спільнота користувачів Python гарантує наявність докладної документації, безліч прикладів застосування у різних галузях та оперативну підтримку у разі виникнення питань. Це особливо важливо для проєктів із високими вимогами до надійності та швидкого реагування на зміни, адже розробники можуть швидко знайти готові рішення для типових задач і зосередитися на унікальних аспектах системи. Завдяки цим характеристикам Python залишається

одним із найпопулярніших та ефективних інструментів для розробки серверних та аналітичних компонентів у складних програмних комплексах.

4.2 Flask

Для розробки серверної частини та реалізації основної бізнес-логіки у виконаному проєкті обрана мікрофреймворк Flask. Flask– це легковаговий Python-фреймворк для створення веб-застосунків, що поєднує мінімалізм «з коробки» із можливістю нарощування функціональності через готові розширення. Він будується на WSGI-інтерфейсі Werkzeug, який відповідає за обробку HTTP-запитів і маршрутизацію, та за замовчуванням використовує потужний шаблонізатор Jinja2 для формування HTML-сторінок.

Водночас у Flask немає накинутих шарів абстракції або складної конфігурації: створення нового застосунку зводиться до імпорту самого фреймворку, визначення обробників маршрутів (route handlers) та запуску вбудованого сервера для локальної розробки. Завдяки цьому розробник може зосередитися на реалізації конкретних RESTful-ендпоїнтів, написанні логіки зчитування сенсорних даних та алгоритмів обробки без відволікання на налаштування «важких» каркасових рішень.

Важливою перевагою Flask є його екосистема розширень (extensions). Для роботи з базою даних часто використовують Flask-SQLAlchemy, який інтегрує потужний ORM SQLAlchemy; для керування міграціями – Flask-Migrate; для організації аутентифікації й авторизації – Flask-Login та Flask-Principal; для обробки запитів і відповіді у форматі JSON – Flask-RESTful або Flask-RESTX. Кожне розширення підключається простою установкою через pip та додаванням до конфігурації застосунку, і при цьому не зобов’язує використовувати його всюди, зберігаючи легкість початкової інсталяції.

Архітектурно Flask підтримує патерн «application factory», коли створення об’єкта застосунку виноситься в окрему функцію. Це дозволяє гнучко керувати конфігурацією (development, testing, production), підключати розширення лише пі-

сля ініціалізації застосунку та організовувати код у вигляді модулів і «синіх принтів» (blueprints). Використання blueprints спрощує розділення різних частин проєкту – наприклад, окремий модуль для API інспекційної платформи, окремий – для веб-інтерфейсу управління, окремий – для внутрішніх служб обробки даних.

Flask також пропонує готовий CLI (командний рядок) для керування застосунком: запуск сервера, виконання міграцій, запуск юніт-тестів. Це робить інтеграцію із CI/CD-процесами (Jenkins, GitLab CI) чи з контейнеризацією в Docker максимально простою: у Dockerfile або у скрипті складання можна викликати flask run або gunicorn як WSGI-сервер для продакшн-оточення.

Ще одна сильна сторона Flask – простота інтеграції машинного навчання. Перетворення натренованої моделі в RESTful-сервіс зводиться до імпорту модуля з вагою моделі (наприклад, pickle-файл), створення відповідного маршруту, де за POST-запитом модель отримує вхідні дані й повертає передбачення у JSON-форматі. Такий підхід дозволяє швидко прототипувати аналітичні та діагностичні функції платформи без додаткових надбудов.

Використання кешування (Flask-Caching) або черг повідомлень (Flask-Celery) для довготривалих завдань гарантує, що важкі обчислення з обробки сенсорних даних не блокуватимуть основний потік обробки HTTP-запитів.

Таким чином, Flask поєднує простоту початкового налаштування та потужний інструментарій для побудови надійних, масштабованих і розширюваних серверних компонентів. Саме ці властивості роблять його ідеальним вибором для розробленої системи керування роботизованою платформою інспекції інфраструктурних об’єктів.

4.3 SQLite

SQL – це стандартна мова для зберігання, маніпулювання та отримання даних в базах даних. SQLite – це широко вживана вбудована СУБД, відзначена своєю простотою та зручністю експлуатації. Головна її перевага полягає в тому, що вона не

потребує налаштування окремого серверного процесу: всі дані зберігаються в звичайних файлах, що дозволяє швидко та без зайвих витрат інтегрувати базу даних у будь-який додаток.

У процесі розробки системи керування роботизованою платформою дослідження інфраструктурних об’єктів використання ORM-бібліотеки SQLAlchemy у поєднанні з базою даних SQLite є технічно доцільним та ефективним рішенням. Воно забезпечує гнучке, масштабоване та стабільне управління даними, що надходять від роботизованих пристроїв.

SQLAlchemy дозволяє працювати з базою даних на високому рівні абстракції – шляхом опису структури даних у вигляді Python-класів, які відображаються на таблиці бази даних. Це особливо зручно у проєкті, де обробляється велика кількість типізованої інформації від сенсорів: координати, температура, вібрації, рівень корозії, тріщини тощо. ORM-підхід спрощує збереження, оновлення та вибірку таких даних, підвищуючи швидкість розробки та зменшуючи кількість помилок.

У системі, де необхідно регулярно зберігати дані від роботизованих платформ, важливою є стабільність бази даних, зокрема підтримка ACID-транзакцій. SQLAlchemy забезпечує контроль транзакцій через сесії, що гарантує цілісність та надійність інформації навіть у разі збою або несподіваного завершення роботи програми.

Додатковою перевагою є можливість легкої міграції між різними СУБД. У рамках цього проєкту SQLite є зручним рішенням для локального зберігання даних, що надходять у реальному часі, оскільки не потребує окремого сервера. У разі масштабування системи – наприклад, для централізованого зберігання даних з кількох платформ – SQLAlchemy дозволяє перейти на потужніші СУБД (наприклад, PostgreSQL) без суттєвих змін у логіці програми.

Підсумовуючи, використання SQLAlchemy у проєкті дозволяє:

– ефективно працювати з великими обсягами сенсорних даних;
– підтримувати високу цілісність і надійність збереження інформації;
– швидко масштабувати систему та адаптуватися до нових вимог;
– зменшити складність розгортання і підтримки бази даних.

4.4 Jinja2

У процесі розробки вебінтерфейсу системи керування роботизованою платформою дослідження інфраструктурних об’єктів важливо забезпечити зручне, динамічне та гнучке відображення даних. Для цього у проєкті використовується шаблонізатор Jinja2, який є невід’ємною частиною вебфреймворку Flask.

Jinja2 забезпечує розділення логіки програми та представлення даних, що сприяє більш структурованому, зрозумілому та масштабованому коду. У контексті даної системи це дозволяє ефективно формувати HTML-сторінки, які відображають результати моніторингу, аналітики та діагностики інфраструктурних об’єктів на основі даних, отриманих від роботизованої платформи.

Однією з ключових переваг Jinja2 є підтримка динамічного вмісту. Зокрема, шаблони дозволяють:

– генерувати таблиці даних з сенсорів у режимі реального часу;

– відображати аналітичні блоки, що формуються на основі машинного навчання;

– виводити карти або фотографії інспектованих об’єктів;

– формувати дашборди зі змінним вмістом залежно від рівня пошкоджень чи типу об’єкта.

Вбудовані фільтри, цикли та умовні конструкції Jinja2 дають змогу гнучко керувати тим, як і що саме буде виводитися на сторінці. Наприклад, при візуалізації звітів можна легко підсвічувати об’єкти, що потребують ремонту, або сортувати інформацію за рівнем ризику.

Ще одна важлива перевага – підтримка спадкування шаблонів, що дозволяє створювати єдину базову структуру (наприклад, навігаційне меню, футер, стилі), яку можуть наслідувати всі інші сторінки. Це суттєво спрощує підтримку проєкту, підвищує уніфікованість дизайну та зменшує дублювання коду.

Оскільки система може розгортатися на різних середовищах (наприклад, на польовому сервері або локальній машині інженера), використання Jinja2 дозволяє

швидко формувати HTML-звіти без залежності від сторонніх клієнтських бібліотек. Це зменшує вимоги до інтернет-з'єднання та забезпечує стабільну роботу навіть у віддалених умовах.

4.5 Scikit-learn

У рамках розробки системи керування роботизованою платформою дослідження інфраструктурних об’єктів важливим компонентом є автоматизований аналіз отриманих сенсорних даних, зокрема для виявлення пошкоджень, прогнозування технічного стану об’єкта та підтримки прийняття рішень щодо подальших дій. Для реалізації інтелектуального аналізу даних у проєкті була обрана бібліотека Scikit-learn, яка є однією з найбільш поширених у Python-екосистемі для вирішення задач машинного навчання.

Бібліотека Scikit-learn була обрана завдяки своїй простоті інтеграції, широкому функціоналу та надійності при роботі з даними невеликого та середнього обсягу, що відповідає потребам проєкту. Система обробляє структуровані дані з роботизованих платформ (наприклад, інформацію про рівень вібрацій, температуру, рівень корозії або тріщини), після чого виконує оцінку стану інфраструктурного об’єкта та визначає ймовірність необхідності ремонту.

Scikit-learn надає великий вибір алгоритмів, серед яких алгоритми класифікації, регресії, кластеризації, а також засоби попередньої обробки даних, зменшення розмірності та моделювання. У рамках даної системи було протестовано декілька моделей класифікації (наприклад, логістична регресія, дерева рішень, випадковий ліс), що дозволило порівняти їхню ефективність за допомогою вбудованих інструментів для розбиття даних на тренувальні та тестові вибірки, а також крос-валідації.

До переваг Scikit-learn у контексті даної системи можна віднести наступне:

– Зрозумілий та логічний API, що дозволяє швидко реалізовувати повний цикл машинного навчання: від завантаження та підготовки даних до побудови та оцінки моделей.

– Повна інтеграція з іншими інструментами екосистеми Python, зокрема Pandas і NumPy, що забезпечує зручну обробку сенсорних даних.

– Можливість швидкого тестування гіпотез і моделей, що є критично важливим у проєктах, де активно проводиться експериментальний підбір параметрів моделей.

– Продуктивність та ефективність при роботі з обмеженим обсягом даних, що відповідає реальним умовам, у яких роботизована платформа проводить періодичні інспекції окремих об’єктів і накопичує дані поступово.

У порівнянні з альтернативними фреймворками, такими як TensorFlow або PyTorch, які більше орієнтовані на глибоке навчання та великі обсяги даних, Scikit-learn є більш відповідним вибором для задач класичного машинного навчання. Він дозволяє досягти високої якості прогнозів без надмірного ускладнення архітектури системи та не потребує додаткових обчислювальних ресурсів, таких як GPU.

Крім того, бібліотека активно підтримується спільнотою, має вичерпну документацію та є стабільною, що робить її зручним інструментом як для розробки, так і для подальшого супроводу системи. Завдяки Scikit-learn у системі реалізовано інтелектуальний механізм оцінки стану інфраструктурних об’єктів, який адаптується до вхідних даних, забезпечує високу точність класифікації та підвищує загальну ефективність роботи платформи.

Висновки до розділу 4

У цьому розділі розглянуто вибір технологій, що були використані для розробки системи керування роботизованою платформою дослідження інфраструктурних об’єктів. Кожен з обраних інструментів відповідає вимогам до функціональності, продуктивності, зручності розробки та розгортання програмного забезпечення.

Python було обрано як основну мову програмування для реалізації серверної частини системи. Завдяки своїй читабельності, лаконічному синтаксису та багатій екосистемі бібліотек Python дозволив швидко та ефективно реалізувати логіку об-

робки даних, управління користувачами, а також інтеграцію з модулями машинного навчання. Його популярність у сфері наукових та інженерних обчислень також сприяла вибору саме цієї мови для реалізації інтелектуального компонента системи.

Flask – мікрофреймворк для Python – був обраний як основа для створення веб-сервера. Його гнучкість, мінімалістичний підхід та велика кількість розширень роблять його зручним для побудови веб-додатків середнього масштабу. У рамках проєкту Flask забезпечує маршрутизацію запитів, взаємодію з базою даних, а також інтеграцію з фронтендом.

Jinja2 – шаблонізатор, який є складовою частиною Flask – використовується для генерації HTML-сторінок на основі динамічних даних. Він дозволяє відокремити логіку представлення від логіки обробки, що підвищує структурованість коду та полегшує підтримку інтерфейсу користувача. За допомогою Jinja реалізовано основні сторінки моніторингу, аналітики та налаштувань, які динамічно відображають отримані з платформ дані.

SQLite використовується як вбудована база даних для збереження структурованої інформації про об’єкти, інспекції та сенсорні показники. Це легка, портативна база даних, що не потребує розгортання окремого сервера, що особливо зручно для системи, яка може бути розгорнута як локально, так і на хмарних платформах з обмеженими ресурсами. SQLite забезпечує достатній рівень продуктивності для задач зберігання та обробки невеликих обсягів даних, які надходять від роботизованих платформ.

Для реалізації блоку машинного навчання було використано бібліотеку Scikit-learn. Вона надає широкий спектр алгоритмів для класифікації, регресії, кластеризації, а також інструменти для підготовки та нормалізації даних. Scikit-learn дозволяє швидко створити прототип моделі, провести її оцінку, а також інтегрувати у виробничу систему з мінімальними витратами. У межах проєкту ця бібліотека використовується для класифікації стану об’єктів на основі показників сенсорів, що дозволяє визначати ступінь пошкодження й необхідність обслуговування.

Фронтенд системи реалізовано з використанням HTML, CSS та JavaScript, що дозволяє створити зручний та адаптивний інтерфейс для користувача. Інтерфейс включає динамічне оновлення інформації, візуалізацію результатів аналізу, відображення карт об'єктів, а також сторінки з аналітикою стану інфраструктури.

Отже, комбінація таких технологій, як Python, Flask, Jinja2, SQLite, Scikit-learn та стандартні засоби веб-розробки, дозволила створити гнучку, надійну та розширювану систему керування роботизованою платформою. Кожна технологія була обрана з урахуванням специфіки завдань системи, що забезпечує ефективну обробку даних, зручність у розробці та підтримці, а також можливість масштабування у майбутньому.

5 РОЗРОБЛЕННЯ СИСТЕМИ

Для розробки системи було створено діаграму прецедентів для наочного подання майбутніх функціональних можливостей користувача. Діаграма прекедентів(Use Case Diagram) у рамках UML-моделювання покликана наочно відобразити функціональні можливості системи з точки зору її зовнішніх користувачів – акторів. На такій діаграмі кожен актор (наприклад, оператор платформи, системний адміністратор або зовнішня аналітична служба) пов’язується з набором «сценаріїв використання» (use cases), які він може ініціювати або в яких брати участь. Ці сценарії описують ключові бізнес-процеси або сервіси, що реалізуються системою, – від запуску інспекційної місії до перегляду аналітичних звітів і налаштування параметрів сповіщень.

Основна мета діаграми прецедентів полягає в тому, щоб на ранніх етапах розробки формалізувати та узгодити з усіма зацікавленими сторонами фундаментальні функції системи. Завдяки графічному поданні стає зрозумілим, хто саме взаємодіє з якою частиною програмного забезпечення і які послуги йому необхідні. Це допомагає проектній команді зібрати повний перелік вимог – як базових, так і додаткових – і попередньо оцінити складність кожного сценарію.

Для розробника системи діаграма прецедентів є незамінним інструментом на кількох рівнях. По-перше, вона служить початковою картою функціональних точок, з якої починається моделювання детальнішої поведінки кожного компонента (класів, сервісів, API-методів). По-друге, чітко вказує межі відповідальності кожного модуля: які дії повинні виконуватись у беку, а які можна делегувати фронтенду чи зовнішнім сервісам. По-третє, діаграма сприяє узгодженню термінів із тестувальниками – вони можуть використовувати кожен сценарій use case для складання тест-кейсів і перевірки, що всі бізнес-логіки реалізовані правильно.

Крім того, на етапі планування релізів і спринтів діаграма прецедентів допомагає пріоритизувати роботи: чітко видно, які випадки використання є критичними для запуску MVP і які можна реалізувати в пізніших ітераціях. Наявність такої діаграми також полегшує комунікацію з кінцевими замовниками: їм легше оцінити,

що саме вони отримають у складі готової системи, побачивши знайомі бізнес-сценарії у вигляді простих блоків із зрозумілими зв’язками.

У результаті, діаграма прецедентів виконує роль містка між бізнес-цілями та технічним виконанням: вона формує загальне уявлення про систему, закладає основу для детальної реалізації і тестування, а також сприяє прозорому обговоренню вимог усіма учасниками проєкту. Завдяки цьому команда отримує чітку «дорожню карту» функціональності, яка полегшує прийняття архітектурних рішень, розподіл задач і контроль якості кінцевого продукту. Моя діаграма представлена в документі ІК13.080БАК.006 Д2 та включає наступні дії:

– адміністратор є головним користувачем, що може зареєструватись різними методами;

– адміністратор може додавати та видаляти нові об’єкти для дослідження;

– адміністратор може переглядати аналітику лише після створення нового об’єкту у системі;

– адміністратор може переглядати звіти моніторингу об’єкту.

Діаграма послідовності (Sequence Diagram) у UML призначена для детального відображення динаміки взаємодії об’єктів або компонентів системи протягом конкретного сценарію використання. Вона зосереджується на порядку обміну повідомленнями між учасниками (акторами й об’єктами доменної моделі) та дозволяє побачити, як саме реалізуються окремі функціональні вимоги «зсередини».

У центрі діаграми послідовності лежать життєві лінії (lifelines) – вертикальні лінії, що показують часову вісь існування компонентів. Горизонтальні стрілки між цими лініями позначають повідомлення або виклики методів, які один об’єкт надсилає іншому. Час рухається зверху вниз, тому легко відстежити, яка операція запускає яку, і у якій послідовності відбувається обмін даними. Для позначення виконання коду на життєвій лінії використовують активні прямокутники (activation bars), а умови чи альтернативні гілки логіки – фрагменти “alt” або “opt”.

По-перше, діаграма послідовності слугує детальною «інструкцією» для написання коду: за її допомогою можна з точністю відтворити порядок викликів методів, сформувати інтерфейси між модулями та узгодити, хто відповідає за яку дію.

По-друге, вона спрощує написання юніт-тестів і інтеграційних тестів: тестувальник або розробник чітко бачить серію кроків, які потрібно змоделювати, та очікуваний результат кожного з них. По-третє, діаграма полегшує командну роботу – фронтендер може зрозуміти, коли і за якою ознакою на бекенді з'явиться дані для відображення, DevOps-інженер – які служби слід підняти та як вони повинні спілкуватися.

Нарешті, у процесі підтримки та розвитку системи діаграми послідовності допомагають швидко оцінити, який вплив матиме зміна у одному компоненті на весь сценарій: якщо потрібно замінити сервіс обробки даних або додати новий крок, достатньо подивитися на послідовності повідомлень і відповідно оновити модель. Завдяки цьому діаграма послідовності стає невід'ємним інструментом для забезпечення узгодженості, прозорості та контролю якості динамічної поведінки прикладних систем.

Тож була розроблена діаграма послідовності, де зображено процес доступу та отримання даних про моніторинг об'єкту та його аналітику. Переглянути її можна у файлі ІК13.080БАК.006 Д3.

5.1 Розроблення серверної частини системи

Перед початком розроблення слід звернутись до діаграми класів. Діаграма потоку (flow diagram) є одним із ключових артефактів при проєктуванні користувацьких сценаріїв і описі бізнес-процесів у програмних системах. Вона наочно демонструє послідовність дій, умовні переходи та рішення, що приймаються під час виконання конкретної операції – у нашому випадку реєстрації та авторизації користувача. Завдяки такому графічному поданню, розробники та аналітики одразу розуміють, як саме «тече» інформація між формами, сервісами та базою даних, і де саме потрібно реалізувати перевірки та обробку помилок.

Насамперед, діаграма потоку слугує дорожньою картою для програмістів, які впроваджують логіку реєстрації та входу. Перед тим як писати код обробників маршрутів у Flask, вони можуть звернутися до діаграми і чітко побачити:

– які кроки необхідно виконати у формі реєстрації, перш ніж створити новий запис у базі,

– в якому місці застосувати валідацію полів (унікальність email, складність пароля),

– коли саме генерувати та зберігати хеш пароля, а коли – перенаправляти користувача на Google OAuth-сторінку,

– як реалізувати логіку обробки помилок і повторних спроб входу.

По-друге, діаграма потоку є незамінною при плануванні тестування. Зі зрозумілою схемою переходів тестувальники формують сценарії: перевіряють валідацію даних, коректність генерації JWT-токена, обробку ситуацій з недійсним паролем чи токеном Google. Кожен вузол і кожен гілковий перехід на діаграмі перетворюється на тест-кейс, що гарантує повне покриття критичних шляхів.

Третій аспект – узгодження з бізнес-аналітиками та замовником. Діаграма потоку дозволяє неспеціалістам побачити “усю картину” процесу реєстрації та входу: звідки система отримає дані, які рішення прийме при різних умовах, куди перенаправить користувача після успіху або помилки. Це прискорює погодження технічного завдання, бо виключає двозначності та непорозуміння.

У подальшому, під час підтримки та розвитку, діаграма потоку допомагає швидко вбудовувати нові сценарії, наприклад, додавання двофакторної автентифікації або інтеграції з іншим провайдером OAuth. Розробник бачить, де розширити логіку, і як уникнути порушення існуючого флоу.

Нарешті, діаграма потоку є невід’ємною частиною технічної документації. Вона дає чіткий алгоритм для DevOps-інженерів, які налаштовують середовище та моніторинг: за якими логами шукати помилки авторизації, які сервіси залучені й коли. Таким чином, діаграма потоку створює єдине «джерело істини» про поведінку системи у критичному для безпеки та зручності користувачів процесі – реєстрації та входу.

Діаграма потоку реєстрації та авторизації користувача моєї системи представлена у файлі ІК13.080БАК.006 Д4

У процесі розробки серверної частини системи керування роботизованою платформою дослідження інфраструктурних об’єктів було прийнято рішення використовувати мову програмування Python і мікрофреймворк Flask, оскільки вони забезпечують надзвичайну гнучкість та швидкість реалізації веб-додатків середнього масштабу. Саме у Flask реалізовано основні REST-ендпоінти, які приймають запити від клієнтської частини, виконують бізнес-логіку, взаємодіють з базою даних через SQLAlchemy, а також викликають компоненти машинного навчання і сервіс генерації звітів.

При переході на вхід у систему користувач спочатку надсилає логін і пароль на спеціальний маршрут /login. На рисунку 5.1 зображено частину системного коду для створення користувача. Пароль на сервері захищається хешуванням (за допомогою бібліотеки werkzeug.security) перед тим, як зберігти у таблиці User, тому навіть у разі компрометації доступ до сирих паролів неможливий. Після успішної перевірки хешу сервер створює JWT-токен (чи сесію, залежно від конфігурації), яким клієнт надалі захищає всі наступні запити.

```
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=True)

    1 usage
    def set_password(self, password):
        self.password = generate_password_hash(password)

    1 usage (1 dynamic)
    def check_password(self, password):
        return check_password_hash(self.password, password)
```

Рисунок 5.1 – Метод створення користувача

Крім класичної аутентифікації, система підтримує вхід через Google OAuth 2.0, інтеграція представлена на рисунку 5.2. При виборі цієї опції користувач перенаправляється на сторінку авторизації Google, де вводить свої облікові дані. Після успішного входу Google повертає на розроблений сервер спеціальний токен, який

перевіряється через офіційний API. Якщо все гаразд, ми отримуємо електронну адресу користувача, створюємо (або витягуємо) відповідний запис у таблиці User і знову ж таки повертаємо JWT-токен клієнту. Таким чином, інтеграція зі стороннім сервісом аутентифікації забезпечує зручність для користувачів і додатковий рівень безпеки.

Однією з важливих переваг інтеграції Google OAuth 2.0 у систему аутентифікації є спрощення життя як для кінцевого користувача, так і для команди розробки. По-перше, із боку користувача це означає можливість безпечного входу в додаток без необхідності запам’ятовувати ще один логін і пароль. Використання облікового запису Google – з уже налаштованими політиками складності паролів і додатковими шарами захисту (SMS-коди, push-сповіщення через Google Prompt, U2F-ключі тощо) – автоматично підвищує рівень безпеки вашої платформи без додаткових зусиль з боку розробників чи користувачів.

По-друге, із точки зору розробника та DevOps-команди впровадження OAuth 2.0 означає значне зниження навантаження на відповідальні за безпеку компоненти: зберігання та обробку паролів, організацію політик блокування облікових записів, реалізацію відновлення доступу через email-листи чи CAPTCHA-перевірки. Усе це бере на себе Google, а ваші сервіси отримують лише підтверджену інформацію – електронну адресу, ім’я користувача та рішення про успішність аутентифікації.

Ще однією важливою перевагою є можливість швидкої реалізації єдиного входу (SSO). Оскільки багато корпоративних клієнтів і приватних користувачів вже мають Google Workspace (раніше G Suite), їм достатньо одного кліку, щоб перейти від внутрішнього порталу чи поштового сервісу до вашого додатку. Це підвищує зручність, скорочує час на навчання та адаптацію і суттєво підвищує конверсію реєстрацій і входів.

```python
@app.route('/authorize/google')
def google_authorize():
    token = google.authorize_access_token()
    userinfo_endpoint = google.server_metadata['userinfo_endpoint']
    resp = google.get(userinfo_endpoint)
    user_info = resp.json()
    email = user_info['email']
    user = User.query.filter_by(email=email).first()
    if not user:
        user = User(email=email)
        db.session.add(user)
        db.session.commit()
    login_user(user)
    session['email'] = email
    session['oauth_token'] = token

    return redirect(url_for('home'))
```

Рисунок 5.2 – Інтеграція для авторизації за допомогою Google

5.1.1 Функціонал перегляду об’єктів

Як тільки користувач авторизований, він може переходити до роботи з інфраструктурними об’єктами. Усі об’єкти – це сутності Object, які мають назву, адресу, посилання на фото та дату останньої інспекції. Створення нового об’єкта виконується через POST-запит на /monitoring, куди надсилаються дані форми разом із файлом фотографії. Представлення зображено на рисунку 5.3. Сервер зберігає завантажену картинку у папці static/images, зберігаючи відносний шлях у полі photo_url об’єкта. Цей же механізм дозволяє міняти адресу чи фото об’єкта через PUT-запит, а видаляти його – через DELETE.

Для кращої візуалізації та полегшення розуміння видалення користувача представлено у вигляді кошику для сміття, що створить знайоме відчуття розуміння процесу. Іконка кошику зображена посередині картки об’єкту та загоряється при наведені на неї, після натискання виникає спливаюче сповіщення, чи справді користувач хоче видалити об’єкт, що вбереже від випадкових дій на сторінці.

```python
@app.route( rule: '/monitoring', methods=['GET', 'POST'])
def monitoring():
    if request.method == 'POST':
        name = request.form['name']
        address = request.form['address']
        date_str = request.form['last_inspection_date']
        last_inspection = datetime.strptime(date_str, __format: '%Y-%m-%d').date()
        photo_file = request.files.get('photo')
        filename = ''
        if photo_file and photo_file.filename:
            filename = secure_filename(photo_file.filename)
            timestamp = datetime.now().strftime("%Y%m%d%H%M%S_")
            filename = timestamp + filename
            save_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            photo_file.save(save_path)

        new_obj = Object(
            name=name,
            address=address,
            photo_url=filename,
            last_inspection_date=last_inspection
        )
        db.session.add(new_obj)
        db.session.commit()

        return redirect(url_for('monitoring'))
```

Рисунок 5.3 – Метод створення нового об’єкта досліджень

Коли користувач потрапляє на сторінку моніторингу, фронтенд виконує GET-запит до /objects, а сервер повертає список усіх об’єктів із бази даних у вигляді JSON. Кожен елемент містить базову інформацію – ідентифікатор, назву, адресу, шлях до фото та дату останньої інспекції. На клієнті ці дані відображаються у вигляді карток із фото та коротким описом. Натискання на картку ініціює AJAX-запит GET /api/object/<id>, і сервер повертає вже повний об’єкт разом із масивом звітів SensorReport. Приклад коду наведений на рисунку 5.4. Кожен запис звіту містить часову мітку, дані сенсорів (температура, вібрація, рівень корозії, ширина тріщини тощо) та передбачений рівень пошкодження, обчислений раніше або в режимі реального часу.

Саме такий підхід дозволяє швидко та легко отримувати інформацію: у найближчому доступі користувач має не лише представлення об’єкту дослідження, а й усі дані, що надсилає роботизована платформа, поки безпосередньо проводить його інспекцію у реальному часі.

```python
@app.route('/api/object/<int:object_id>')
def get_object_details(object_id):
    obj = Object.query.get_or_404(object_id)
    reports = SensorReport.query.filter_by(object_id=object_id).order_by(SensorReport.timestamp).all()

    return jsonify({
        'id': obj.id,
        'name': obj.name,
        'address': obj.address,
        'last_inspection': obj.last_inspection_date.strftime('%Y-%m-%d') if obj.last_inspection_date else None,
        'photo_url': f'/static/images/{obj.photo_url}' if obj.photo_url else '/static/images/default.jpg',
        'reports': [
            {
                'robot': r.robot,
                'timestamp': r.timestamp.strftime('%Y-%m-%d %H:%M'),
                'latitude': r.latitude,
                'longitude': r.longitude,
                'crack_width': r.crack_width,
                'corrosion_level': r.corrosion_level,
                'temperature': r.temperature,
                'vibration': r.vibration,
                'scan_progress': r.scan_progress,
                'predicted_severity': r.predicted_severity
            } for r in reports
        ]
    })
```

Рисунок 5.4 – Метод перегляду детальної інформації об’єкту

5.1.2 Перегляд детальної аналітики

Перегляд сторінки аналітики в системі керування роботизованою платформою інспекції інфраструктурних об’єктів дає користувачам змогу перетворити сукупність «сирих» даних із сенсорів на зрозумілі візуальні метрики й індикатори, які відразу ж підсвічують ключові проблемні зони. Замість перегляду десятків показників вручну оператор отримує узагальнену інформацію про рівень зносу, темпи розвитку тріщин або корозії, що дозволяє оперативно виявляти «гарячі точки» та пріоритизувати огляд чи ремонт. Це особливо корисно для інженерів, які повинні прогнозувати, в якій ділянці мосту, газопроводу або дороги дефект може перерости в аварійну ситуацію. Наявність трендових графіків із відмітками попередніх інспекцій допомагає аргументувати необхідність виділення коштів на ремонт перед керівництвом чи замовником.

Аналітична панель також дає змогу зберігати й експортувати звіти у вигляді PDF або таблиць для подальшого обговорення та архівування. Це спрощує підготовку документів для державних або корпоративних аудитів, а також дозволяє структуровано доводити виконані роботи та отримані результати. Вбудовані шаблони звітів економлять час інженерів, адже не потрібно формувати документи «з нуля» після кожного виїзду на об’єкт.

Забезпечення можливості завантаження PDF-звітів є надзвичайно корисним і зручним інструментом для користувачів системи керування роботизованою платформою інспекції інфраструктурних об’єктів. PDF-формат є універсальним стандартом, який коректно відображається на будь-яких пристроях – від десктопів до мобільних ґаджетів – без залежності від версій програмного забезпечення чи операційних систем. Це означає, що інженери, менеджери або керівники, які перебувають у польових умовах чи офісі, завжди матимуть змогу миттєво відкрити звіт, навіть якщо інтернет-зв’язок обмежений.

Можливість відокремленого завантаження окремих розділів або всього звіту в PDF прискорює процес комунікації з підрядниками і замовниками. Інспекційні компанії, дорожні чи будівельні фірми можуть оперативно надати повний звіт на тендері або технічній нараді, не вимагаючи від інших учасників доступу до самої системи. Це полегшує обговорення результатів, оскільки всі учасники бачать однакові дані у зрозумілому вигляді, з чіткими графіками та фотографіями дефектів.

З точки зору аудиту та контролю якості, PDF-звіт забезпечує незаперечну фіксацію даних на момент проведення інспекції. У разі виникнення непорозумінь щодо правильності діагностики або термінів виконання робіт, цей документ слугуватиме доказовою базою: він містить дату й час формування звіту, підпис оператора. Таким чином, можливість завантаження PDF-звітів підвищує прозорість і відповідальність усіх учасників процесу, підтверджуючи, що рішення щодо ремонту чи реконструкції приймалися на основі об’єктивних даних.

Нарешті, доступ до аналітики сприяє більш зваженому та проактивному підходу до експлуатації інфраструктури. Керівники зможуть більш точно планувати бюджети, визначати оптимальні терміни технічного обслуговування та ризик-менеджмент, а технічні фахівці – комбінувати результати обстежень із зовнішніми факторами (погодні умови, трафік, час доби), щоб побудувати найбільш ефективний графік роботи роботизованої платформи. У сукупності це підвищує безпеку, економить ресурси та продовжує термін служби об’єктів, які контролюються системою.

Аналітичний функціонал винесено у окремий маршрут POST /analytics. Клієнт передає у тілі запиту ідентифікатор об’єкта та, за бажанням, часовий проміжок (дата початку та кінця). Сервер звертається до таблиці SensorReport, отримує всі записи для зазначеного об’єкта в заданому діапазоні, перетворює їх у pandas.DataFrame і передає дані в компонент машинного навчання на базі scikit-learn. Метод зображений на рисунку 5.5.

```
rows = []
for r in reports:
    rows.append({
        'timestamp': r.timestamp,
        'robot': r.robot,
        'crack_width': r.crack_width,
        'corrosion_level': r.corrosion_level,
        'temperature': r.temperature,
        'vibration': r.vibration,
        'scan_progress': r.scan_progress
    })
df = pd.DataFrame(rows)

model = joblib.load('damage_severity_model.pkl')
label_encoder = joblib.load('label_encoder.pkl')
feature_cols = ['crack_width', 'corrosion_level', 'temperature', 'vibration', 'scan_progress']
X = df[feature_cols].values
codes = model.predict(X)
labels = label_encoder.inverse_transform(codes)
df['predicted_severity'] = labels
summary_counts = df['predicted_severity'].value_counts().to_dict()
avg_severity = round(pd.Series(codes).mean(), 2)
```

Рисунок 5.5 – Метод з використанням машинного навчання

Також, після надіслання DataFrame, в якому кожен рядок відповідає одній інспекції: містить часову мітку (timestamp), виміряні значення параметрів (temperature, vibration, corrosion_level, crack_width, scan_progress) та прогнозований рівень пошкодження (predicted_severity) після навчання моделлю штучного інтелекту – візуалізується на сторінці аналітики відповідно. Для візуалізації цих даних генеруються два основні типи графіків:

– лінійний графік зміни рівня пошкодження в часі.

На горизонтальній осі відкладається timestamp, а на вертикальній – числове уявлення прогнозованого ступеня пошкодження (наприклад, Low=1, Medium=2, High=3). Кожна точка відповідає одній інспекції, а сполучні лінії допомагають візуально відстежити динаміку погіршення чи покращення стану об’єкта.

– гістограма розподілу рівнів пошкодження.

Цей графік показує, скільки разів під час досліджень на різних періодах для вибраного об’єкта модель класифікувала стан як “Low”, “Medium” чи “High”. Кожен стовпець відповідає одній категорії, його висота пропорційна кількості спостережень. Така візуалізація дає швидке уявлення про те, чи найчастіше об’єкт перебував у безпечному стані чи потребував негайного втручання.

Кожен із цих графіків підписаний відповідними заголовками, підписами осей і легендою, щоб користувач відразу розумів, які саме сенсорні параметри та прогнозовані категорії показані на діаграмі. Після створення їх зберігають у статичній папці (static/) у форматі PNG, а потім передають у фронтенд для відображення або включають у кінцевий PDF-звіт. Такий підхід забезпечує наочну, інформативну та технічно обґрунтовану аналітику стану інфраструктурних об’єктів.

5.1.3 Візуалізація руху роботів

Сторінка візуалізації руху роботів відіграє ключову роль як для оперативного контролю, так і для подальшого аналізу ефективності інспекційних місій. У реальному часі вона дозволяє оператору бачити поточне розташування кожного робота на карті об’єкта, що дає змогу миттєво оцінити, чи рухається він за заданим маршрутом, чи натрапив на перешкоду. Якщо траєкторія збивається – наприклад, через незаплановану перешкоду або збій сенсорів – оператор відразу це помітить і матиме можливість скоригувати рух вручну або змінити параметри автономного контролю.

Візуалізація траєкторій після завершення місії дає змогу оцінити оптимальність прокладених шляхів. На підставі пройдених маршрутів можна виявити зони, де робот занадто часто змінював курс або рухався надто близько до поганої поверхні, – саме в цих ділянках варто провести додаткову діагностику з використанням інших сенсорів або навіть із залученням дрібнішого дрона для огляду важкодоступних місць. Аналіз густоти покриття (скільки разів робот «проїжджав» по тій самій

території) допомагає уникнути дублювання інспекцій та скоротити загальний час роботи.

Крім того, візуалізація руху є важливим інструментом для звітності перед замовником чи керівництвом. Графічне відображення пройдених маршрутів у супроводі ключових подій (автоматичних зупинок, тривожних сигналів, моментів запуску аналізу) створює прозорий та наочний звіт про виконану роботу. Це підвищує довіру до системи, оскільки клієнт бачить не лише результати діагностики, а й сам процес збору даних.

У рамках реалізації сторінки «Карта руху роботів» було розроблено компонент клієнтської частини, який забезпечує безперервне оновлення й відображення положення кожної роботизованої платформи в реальному часі із застосуванням бібліотеки Leaflet.js та REST-інтерфейсу. У цьому розділі наведено детальний опис роботи відповідного JavaScript-модуля та інтеграції з бекендом.

Після завантаження HTML-документа браузером розпочинається ініціалізація інтерактивної карти за допомогою бібліотеки Leaflet. У тілі HTML-файлу викликається функція L.map('map').setView([0, 0], 2), яка створює новий екземпляр об’єкта карти з ідентифікатором 'map'. Ця карта одразу позиціонується на глобальних координатах [0, 0], що відповідає точці перетину екватора і нульового меридіана, а початковий масштаб задається як 2. Такий масштаб дозволяє охопити більшість поверхні Землі на одному екрані, що є зручним для подальшого додавання геопросторових елементів на різних широтах і довготах.

Після створення карти до неї додається базовий фоновий шар у вигляді картографічних тайлів. Це здійснюється через виклик L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {...}).addTo(map). Тут використовується стандартне URL-шаблонування для підключення картографічних плиток з відкритого сервісу OpenStreetMap, який надає безкоштовні й деталізовані мапи світу. Тайли автоматично підвантажуються відповідно до поточного масштабу і координат відображення. Передача об'єкта конфігурації (наприклад, з атрибуцією або обмеженням масштабу) дозволяє налаштувати вигляд та поведінку карти згідно з потребами застосунку.

Цей фоновий шар служить основою для подальшої роботи з картою — на ньому можна розміщувати маркери, геометричні фігури, шари даних тощо. Завдяки інтеграції з географічною проекцією, кожен елемент, який додається на карту, автоматично співвідноситься з реальними координатами, що критично важливо для точного візуального представлення просторової інформації.

Метод представлено на рисунку 5.6.

```
const map = L.map('map').setView([0, 0], 2);
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '© OpenStreetMap'
}).addTo(map);

const robotMarkers = {};

1+ usages
async function fetchRobots() {
    const response = await fetch('/api/robots');
    const robots = await response.json();

    robots.forEach(robot => {
        if (robotMarkers[robot.name]) {
            robotMarkers[robot.name].setLatLng([robot.lat, robot.lng]);
            robotMarkers[robot.name].setPopupContent(`Робот: ${robot.name}<br>Прогрес сканування: ${robot.scan_progress}
        } else {
            const marker = L.marker([robot.lat, robot.lng]).addTo(map)
                .bindPopup(`Робот: ${robot.name}<br>Прогрес сканування: ${robot.scan_progress}%`);
            robotMarkers[robot.name] = marker;
        }
    });
}
```

Рисунок 5.6 – Метод відображення карти на сторінці

Для зберігання та керування екземплярами маркерів використовується об’єкт robotMarkers = {}. Ключами цього асоціативного масиву є унікальні ідентифікатори роботів (значення поля robot.name із відповіді API), а значеннями – об’єкти маркерів Leaflet. Така структура дозволяє швидко перевіряти наявність маркера для конкретного пристрою і виконувати або створення, або оновлення відповідного елемента на карті.

Функція fetchRobots() реалізує логіку отримання оновлених координат. Вона асинхронно виконує HTTP-запит на REST-ендпоінт /api/robots за допомогою стандартного API fetch. Після отримання відповіді у форматі JSON відбувається ітерація масиву об’єктів, кожен із яких має поля:

– name – рядковий ідентифікатор робота;

– lat, lng – числові значення географічних координат у десятковому поданні;
– scan_progress – показник поточного прогресу сканування у відсотках.

Якщо в robotMarkers уже існує маркер із ключем robot.name, то поточний маркер оновлюється методом setLatLng([robot.lat, robot.lng]), що переміщує його на нову позицію без перезавантаження чи видалення елемента з DOM-дерева. Одночасно викликається setPopupContent(...), щоб відобразити актуальний прогрес сканування у спливаючому вікні. У разі, якщо маркера ще немає, створюється новий об’єкт L.marker([robot.lat, robot.lng]) із прив’язкою до карти через .addTo(map) та встановленням вмісту popup методом .bindPopup(...). Після цього екземпляр маркера зберігається у robotMarkers[robot.name] = marker, що гарантує його доступність при наступних оновленнях.

Щоб забезпечити безперервне оновлення без зайвого навантаження на мережу та ресурси браузера, виклик fetchRobots() спочатку відбувається одразу після ініціалізації карти, а потім періодично – за допомогою setInterval(fetchRobots, 3000). Інтервал у три секунди було обрано як компроміс між «живою» реакцією системи та помірним споживанням серверних ресурсів. Такий механізм дозволяє відображати рух роботів у режимі близькому до реального часу, оновлюючи лише ті маркери, які потребують переміщення, і створюючи нові маркери для тих пристроїв, які з’явилися у полі спостереження.

З огляду на характер завдань, інженерам та аналітикам важливо мати можливість швидко зорієнтуватися щодо поточного стану роботів та їх географічного розташування. Використаний підхід із диференційним оновленням – через перевірку наявності маркерів і виклик відповідних методів setLatLng та bindPopup – дозволяє мінімізувати кількість операцій над DOM і забезпечує плавний досвід користувача. При цьому методи Leaflet.js оптимізовані для роботи з великою кількістю маркерів, що дає змогу масштабувати систему на сотні чи тисячі роботів без значного погіршення продуктивності.

Таким чином, створена механіка поєднує асинхронний запит даних із клієнта, ефективне управління маркерами на карті та періодичне оновлення у фоновому ре-

жимі. Це забезпечує зручний та надійний інструмент для відстеження руху роботизованих платформ у режимі реального часу, що є критично важливим для оперативного прийняття рішень під час інспекцій інфраструктурних об’єктів.

5.2 Розроблення клієнтської частини системи

У рамках клієнтської частини системи керування роботизованою платформою для інспекції інфраструктурних об’єктів основною метою було створити інтуїтивно зрозумілий та одночасно функціонально насичений веб-інтерфейс, який дає змогу адміністраторам швидко орієнтуватися у великій кількості даних, що надходять від роботів. Інтерфейс виконано з використанням класичних HTML-шаблонів Jinja2 у поєднанні зі стилями CSS і ванільним JavaScript, що дозволило зберегти максимальну легкість і переносність рішення та забезпечити гнучку інтеграцію з Flask-бекендом.

Головною точкою входу є домашня сторінка з переліком наявних сторінок для перегляду та переходу. Також наявний короткий опис системи зі згадуванням скільки об’єктів вже було досліджено, коли було проведено останній огляд та коли було останнє оновлення системи. Представлення зображено на рисунку 5.7.

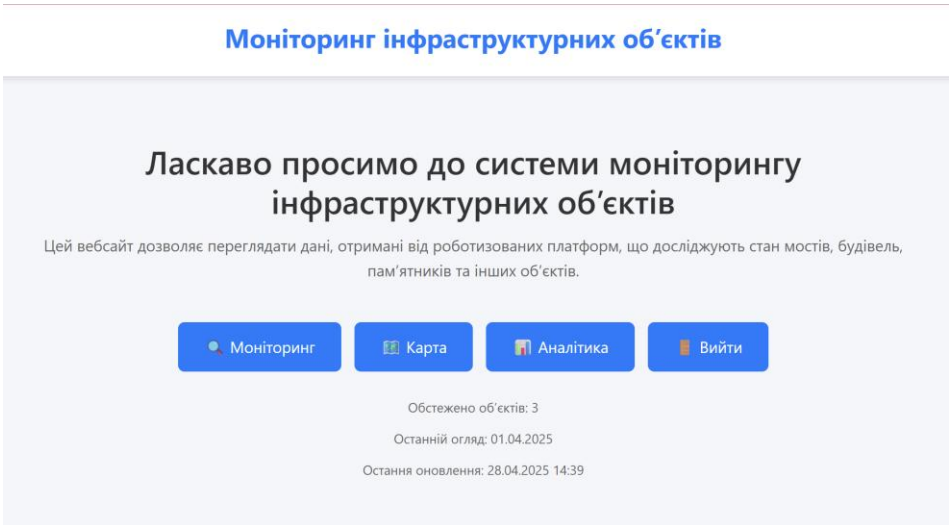

Рисунок 5.7 – Головна сторінка застосунку

Головною є сторінка моніторингу об’єктів, де кожен інфраструктурний об’єкт відображається у вигляді картки з фото, адресою та датою останньої інспекції. Дані підвантажуються з бекенду через AJAX-запит на /monitoring одразу після рендерингу шаблону, а подальша навігація між сторінками здійснюється класичними посиланнями та GET-параметрами. Для зручності перегляду картки оновлюються за принципом пагінації: по три об’єкти на сторінку, із можливістю переходу вперед-назад через кнопки та активні індикатори сторінок. Переглянути інтерфейс можна на рисунку 5.8

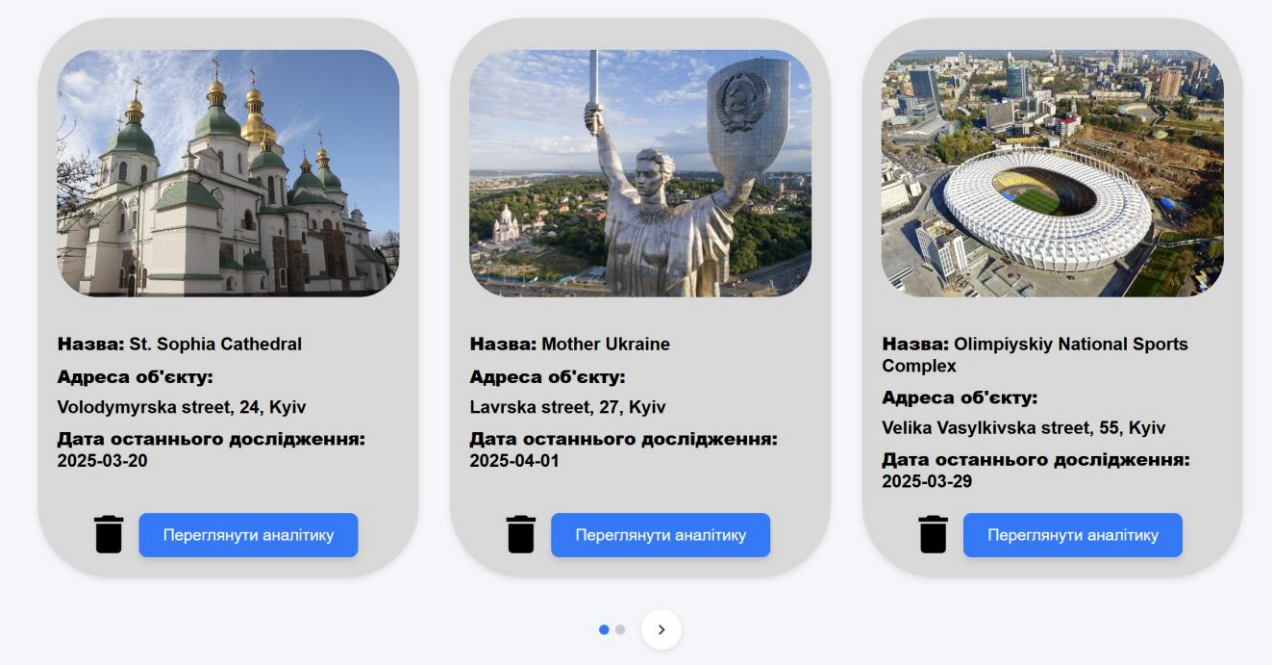

Рисунок 5.8 – Інтерфейс сторінки моніторингу

На сторінці моніторингу також відображається висувна панель (sidenav), яка з’являється після кліку на «+» і дозволяє адміністраторам створювати новий об’єкт для досліджень: вводити назву, адресу, дату інспекції та завантажити фото. Після підтвердження дані надсилаються на бекенд методом POST, зберігаються в базі та одразу відображаються серед карток. Для цієї форми реалізовано базову валідацію в HTML5 (обов’язкові поля, перевірка формату дати) та додаткову перевірку на стороні JavaScript, щоб запобігти порожнім запитам. Представлення можна переглянути на рисунку 5.9.

Мала висувна форма для створення об’єктів має низку переваг у порівнянні з повноекранними або окремими сторінками, адже це пришвидшує взаємодію, знижує когнітивне навантаження, так як користувач бачить лише ключові поля, необхідні для створення об’єкта; створює компактний інтерфейс, тому що не займає всю площину екрану чи окрему сторінку; мінімальний контекстний зсув грає немалу роль, адже користувач не покидає основну сторінку, що дозволяє швидко повернутись до роботи без втрати контексту.

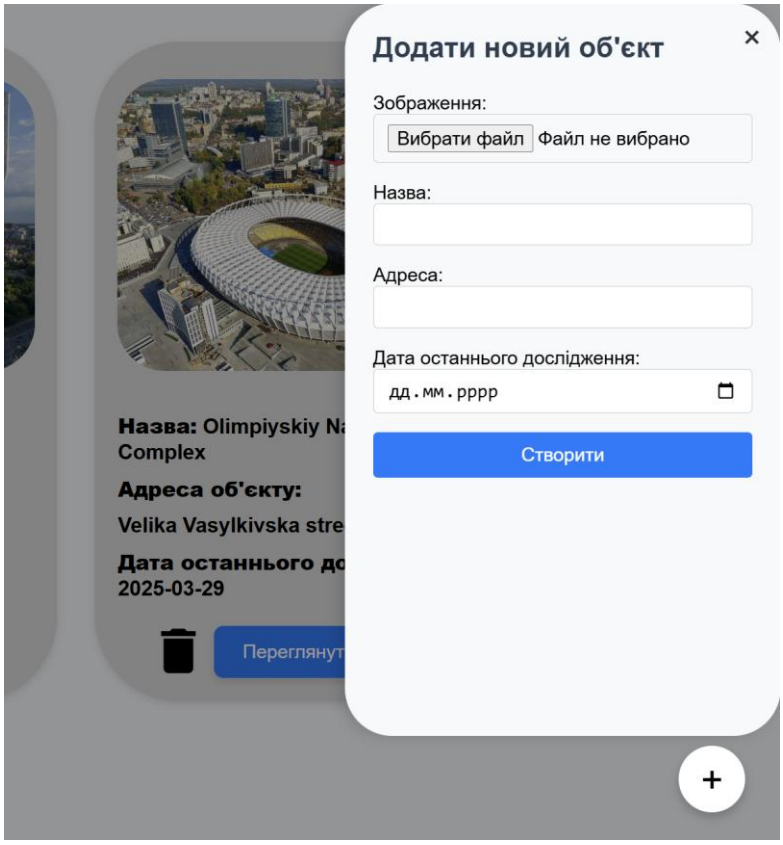

Рисунок 5.9 - Висувна панель для створення нових об’єктів

Перехід до деталей об’єкта відбувається при кліку на картку: виконується AJAX-запит на /api/object/<id>, і у спливаючому шарі (popup) відкривається «докладна картка» з усіма параметрами останнього звіту (координати, температура, вібрації, корозія, ширина тріщин, прогрес сканування об’єкту, оцінка ML-моделі). У цьому вікні також передбачена кнопка «Перейти до аналітики», що перенаправляє на /analytics?object_id=<id>, що можна помітити на рисунку 5.10.

Сторінка деталей об’єкту висвітлює основу інформацію як про об’єкт: може бути доповненою зображеннями надісланими від роботизованої платформи, адресою та датаю останнього дослідження, що допоможе краще орієнтуватись наскільки оновленою є інформація про об’єкт. Нижче представлена таблиця з результатами інспекцій з детальними позначеннями у який самі ділянці проводилось дослідження (широта і довгота) та результатами зібраними з датчиків, що допоможе швидко переглянути перші результати інспекцій.

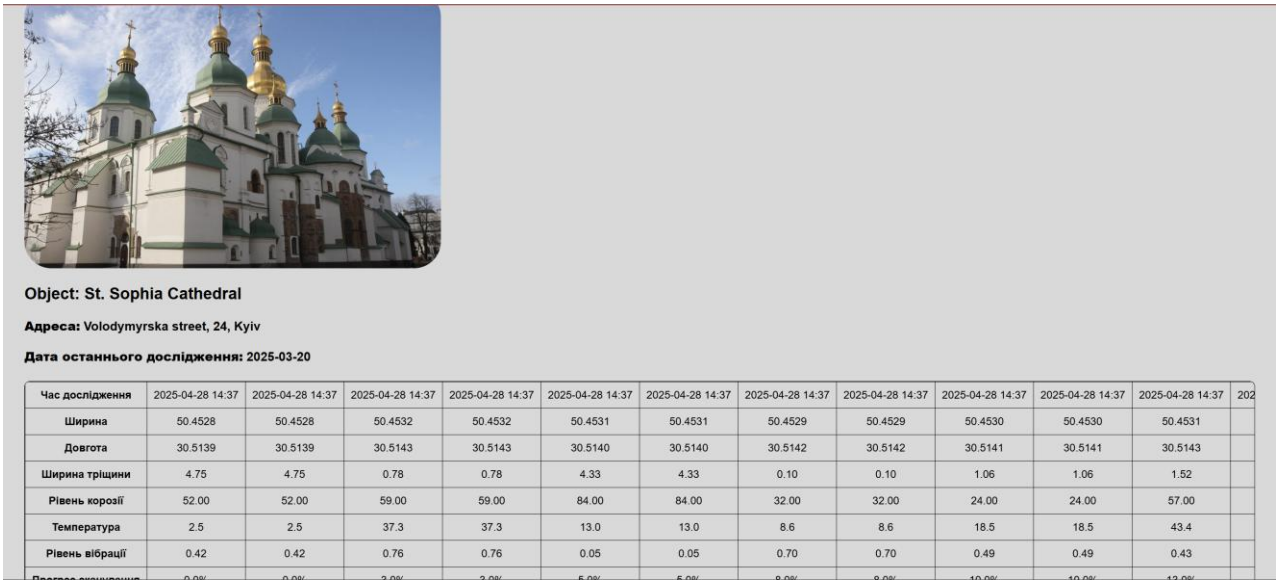

Рисунок 5.10 – Зображення картки детальної інформації об’єкту

Сторінка аналітики динамічно формує графіки та гістограми: після завантаження вона надсилає POST-запит із object_id та, за потреби, фільтрами за датою. Отримані URL зображень лінійного графіка зміни рівня пошкодження та гістограми розподілу вставляються в шаблон через тег <img>, а під ними виводяться табличні підсумки (summary) у вигляді HTML-списку. (Рисунок 5.11)

Перший графік "Severity Over Time" ілюструє динаміку рівня пошкоджень об’єкта в часовому розрізі. По горизонтальній осі (X) відображено часові мітки сканувань, по вертикальній осі (Y) – числові коди, що відповідають класифікації пошкодження. Графік демонструє постійну зміну рівня пошкоджень між значеннями 1 (середній) та 2 (високий), що свідчить про нестабільність у стані об’єкта та потенційну загрозу його подальшої деградації.

Другий графік "Severity Distribution" відображає загальну кількість кожного рівня пошкодження серед усіх звітів. По осі X зазначено категорії ("Середній" і "Низький"), по осі Y – кількість відповідних випадків. Домінування середнього рівня пошкодження свідчить про наявність значущих дефектів у структурі об’єкта, які, хоча й не є критичними, потребують постійного моніторингу та можливої профілактики.

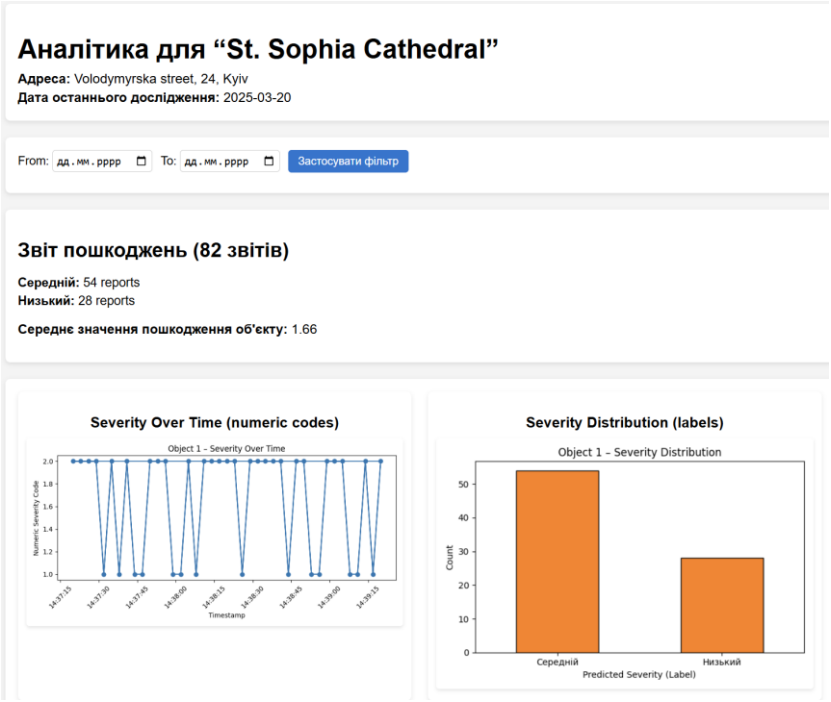

Рисунок 5.11 – Сторінка аналітики у вигляді графіків

Параметри, що впливають на оцінку пошкодження та використовуються в моделі, включають: ширину тріщин (crack_width), рівень корозії (corrosion_level), температуру (temperature), вібрації (vibration) та прогрес сканування (scan_progress). Ці ознаки подаються до навченої моделі машинного навчання, яка на основі них прогнозує рівень пошкодження. Згодом результати кодуються за допомогою label_encoder, що дозволяє інтерпретувати числові коди у вигляді текстових міток, зрозумілих для користувача.

Додатково реалізовано кнопку «Завантажити PDF-звіт», яка викликає /download_pdf і одразу повертає згенерований файл у діалозі завантаження. (Рисунок 5.12)

PDF звіт містить у собі усю інформацію зібрану на сторінці аналітики, але у компактному вигляді, та з можливими наявними фільтрами даних.

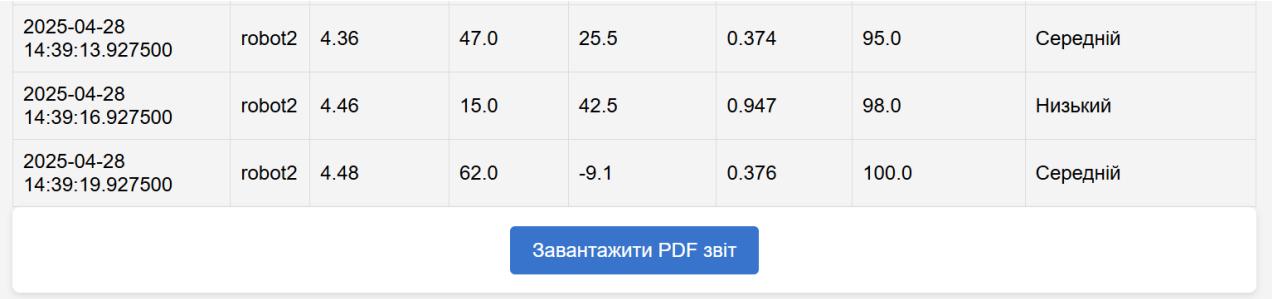

Рисунок 5.12 – Сторінка аналітики у вигляді таблиці та файлом звіту

Особливу увагу приділено інтерактивності: на кожній сторінці, де відображається статус (наприклад, прогрес сканування чи підключення роботів на карті), працює фоновий JavaScript-таймер, що викликає відповідний AJAX-запит з інтервалом у 3–10 секунд. Це дозволяє тримати інтерфейс «живим» без необхідності ручного оновлення. Навігація між різними секціями клієнтської частини реалізована за допомогою стандартних HTML-посилань, що направляють на інші маршрути Flask.

Загалом, клієнтська частина поєднала в собі простоту реалізації з високим рівнем інтерактивності: динамічне підвантаження даних, формування графіків та звітів, а також зручний CRUD-інтерфейс для всіх сутностей інфраструктурних об’єктів Завдяки використанню Jinja2 для сервісної генерації HTML та легковагого JavaScript для «живої» взаємодії, вдалося досягти балансу між зручністю розробки, продуктивністю та користувацьким досвідом.

Висновки до розділу 5

У цьому розділі описано реалізацію як серверної, так і клієнтської складових системи керування роботизованою платформою для інспекції інфраструктурних об’єктів. Серверна частина побудована на Python з використанням Flask та SQLAlchemy і відповідає за повний цикл CRUD-операцій над сутностями «Об’єкт» та «Звіт сенсорів», а також за інтеграцію з компонентом машинного навчання на

базі scikit-learn. Було реалізовано REST-ендпоінти для створення, редагування та видалення об’єктів, завантаження й зберігання фото, імпорту CSV-даних від роботів і виклику аналітичних сервісів, а також механізм аутентифікації користувачів із підтримкою як базового логін-пароль, так і Google OAuth 2.0.

Клієнтська частина, реалізована за допомогою Jinja2-шаблонів у Flask та чистого JavaScript, забезпечує інтуїтивний інтерфейс для адміністраторів. Основна сторінка моніторингу відображає картки об’єктів із фото, адресою та датою останньої інспекції, а сторінка «Карта руху роботів» динамічно показує положення роботів у реальному часі з плавним оновленням маркерів. Сторінка аналітики генерує та відображає графіки зміни рівня пошкоджень і гістограми розподілу, а також пропонує завантажити PDF-звіт із графіками та табличними даними. Валідація форм, пагінація, пошук та періодичне оновлення статусів (через AJAX-запити кожні кілька секунд) гарантують сучасний UX без необхідності перезавантаження сторінки.

Завдяки поєднанню легкої архітектури на Flask, адаптивного фронтенду з Jinja2 і JavaScript та ефективного аналізу даних за допомогою scikit-learn створено повнофункціональну систему моніторингу та аналітики, яка відповідає всім вимогам гнучкості, продуктивності та масштабованості.

ВИСНОВКИ

У процесі виконання дипломного проєкту «Система керування роботизованою платформою інспекції інфраструктурних об’єктів» було всебічно досліджено та реалізовано комплексне рішення, яке поєднує в собі сучасні підходи до мобільної робототехніки, веб-технологій та машинного навчання. На початку роботи було обґрунтовано актуальність теми: у зв’язку зі старінням інженерних споруд, складними кліматичними та військовими викликами традиційні методи візуальної інспекції виявились недостатньо ефективними для своєчасного виявлення та оцінки дефектів. Аналіз світових і вітчизняних напрацювань показав, що мобільні роботизовані платформи з різноманітними сенсорами (LiDAR, ультразвук, термокамера) та алгоритмами штучного інтелекту здатні забезпечувати високу точність діагностики, мінімізуючи ризики для персоналу та скорочуючи час проведення обстежень.

З урахуванням вимог до системи було чітко сформульовано функціональні завдання – маршрутизацію платформи, збір і синхронізацію даних із сенсорів, попередню обробку та класифікацію пошкоджень, а також автоматизовану генерацію звітів. Поряд з ними було визначено критично важливі нефункціональні характеристики: забезпечення низької затримки інтерфейсу, безперебійної доступності сервісу, захищеного зберігання та передачі даних, масштабованості під одночасне обслуговування декількох десятків роботів і можливості автономної роботи. Такий підхід дозволив сформувати деталізовану технічну специфікацію та уникнути розпорошення зусиль під час реалізації.

У результаті проєктування була спроектована модульна архітектура, яка охоплює клієнтську частину з інтерактивними картами та аналітичними дашбордами, серверну реалізацію на основі Flask із REST-інтерфейсами для авторизації, моніторингу, аналітики та управління платформою. В процесі роботи було розроблено UML-моделі: діаграми прецедентів, послідовності та розгортання, а також діаграма класів, що відображає сутності User, SensorData, DamageSeverityModel, ReportGenerator та інші ключові компоненти системи.

Реалізація серверної логіки з використанням Python і Flask дала змогу швидко розгорнути основні REST-ендпоінти для реєстрації, входу за допомогою Google OAuth 2.0, отримання координат та даних сенсорів, а також викликів модуля машинного навчання для оцінки рівня пошкоджень in situ. На стороні клієнта шаблони Jinja2 і JavaScript-карта забезпечують наочне відображення траєкторій руху роботів та маркерів дефектів. Завдяки використанню стандартних бібліотек та готових розширень було досягнуто балансу між простотою коду й продуктивністю системи.

Практичне значення розробленого рішення полягає у значній економії часу та ресурсів інженерів: автоматизована інспекція дозволяє оперативно виявляти критичні дефекти, пріоритизувати ремонтні роботи та зменшувати тривалість простою об’єктів. Гнучка REST-архітектура відкриває можливості для інтеграції з існуючими SCADA- та GIS-системами.

У ході проєкту автор здобув важливі професійні навички: поглиблене володіння Python, Flask, робота з ML-бібліотеками та побудова REST-сервісів; навички UML-моделювання й технічного документування. Усі ці компетенції створили міцну основу для подальшого розвитку в галузі «розумної» інфраструктури та автоматизації інженерного моніторингу.

В майбутньому система керування роботизованою платформою інспекції інфраструктурних об’єктів може еволюціонувати в напрямах як технічного вдосконалення, так і розширення спектра бізнес-послуг. По-перше, інтеграція додаткових видів сенсорів – зокрема акустичних датчиків для виявлення внутрішніх вад матеріалів, хімічних аналізаторів корозійної активності та радіочастотних модулів для діагностики підземних комунікацій – дозволить отримувати комплексніший портрет стану об’єктів без залучення зовнішніх виконавців. Це відкриє шлях до створення «мультисенсорних» репортажів, де дані з різних джерел автоматично обробляються єдиною платформою.

По-друге, у перспективі логічним кроком стане реалізація кооперативного управління кількома роботами в межах однієї місії. Впровадження алгоритмів роз-

поділеного планування траєкторій і балансування навантаження допоможе скоротити час повного обстеження великих інфраструктурних комплексів і зменшити енергоспоживання кожної платформи. Сценарії, коли одна машина виконує швидкий поверхневий огляд, а інша – детальне сканування окремих ділянок, можуть бути реалізовані через оркестрацію на сервері зі штучним інтелектом.

По-третє, удосконалення машинного навчання від простих моделей класифікації пошкоджень до глибинних нейронних мереж і методів часових рядів для прогнозування динаміки зношення дозволить перейти від реактивного реагування на виявлені дефекти до проактивного обслуговування. Можливість автоматичного оновлення моделей за допомогою MLOps-процесів – безперервного збору анонімізованих даних, тренування нових версій і безшовного деплою – забезпечить високу точність прогнозів та адаптивність до змін у середовищі.

На рівні інфраструктури варто опрацювати сценарії безсерверного (serverless) розгортання в хмарі та edge-комп’ютингу на борту роботів, щоб обчислювати важливі аналітики без постійного зв’язку із центральним сервером. Це підвищить стійкість системи у зонах із нестабільним інтернет-з’єднанням.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Будівельні конструкції : Навчальний посібник. Львів : Видавництво Львівської політехніки, 2016. – 200 с. URL: https://vseosvita.ua/library/dystsyplina-budivelni-konstruktsii-konspekt-lektsii-z-temy-tema-11-vymohy-do-budivelnykh-konstruktsii-789105.html

2. Spot. BostonDynamics. URL: https://bostondynamics.com/products/spot/

3. EverSense. Sixense group. URL: https://www.sixense-group.com/en/offer/monitoring/structural-health-monitoring-shm/eversense

4. PipeWalker. Pure Technologies / Xylem. URL: https://www.xylem.com/en-us/brands/pure-technologies/technologies/pipewalker-condition-assessment-platform/

5. Нефункціональні вимоги. Вікіпедія. URL: https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%96_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3%D0%B8

6. Python official Documentation. Python. URL: https://www.python.org/

7. SQLite documentation. SQLite. URL: https://www.sqlite.org/

8. Flask tutorial. Geeksforgeeks. URL: https://www.geeksforgeeks.org/python/flask-tutorial/

9. Flask Documnentation. Pallets. URL: https://flask.palletsprojects.com/en/stable/

10. Werkzeug Documentation. Pallets. URL: https://werkzeug.palletsprojects.com/en/stable/

11. Jinja Documentation. Pallets. URL: https://jinja.palletsprojects.com/en/stable/

12. Leaflet documentation. Leaflet. URL: https://leafletjs.com/

13. Getting started with Jinja Template. Geeksforgeeks. URL: https://www.geeksforgeeks.org/python/getting-started-with-jinja-template/

14. Beazley, Jones B. K. Python Cookbook. 3rd ed. O'Reilly Media, 2013. 706 p.

15. Stonebraker M., Hellerstein J. M. Readings in Database Systems. MIT Press, 2005. 878 p..

16. Miguel Grinberg. Flask Web Development: Developing Web Applications with Python. 2nd ed. O'Reilly Media, 2018. 312 p.

17. Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2019. 856 p.