НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
(назва кафедри)
Голуб Б.Л.
(підпис) (ПІБ)
"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Програмне забезпечення системи автоматичного моніторингу диму на природних територіях

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ Вайганг Г.О.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
д.е.н., професор _____ Руденський Р.А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Карась Павло Петрович
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)

_____ к.т.н., доц. Голуб Б.Л.
(підпис) (ПІБ)

"___" _____ 2025 р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Карась Павло Петрович
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи **Програмне забезпечення системи автоматичного моніторингу диму на природних територіях**

затверджена наказом ректора НУБіП України від "16" 12 2024 р. No 2248 "С"

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи
опис програмного з. абезпечення

Перелік питань, які потрібно розробити:

1) Системний наліз предметної області.
2) Інформаційне забезпечення.
3) Програмне забезпечення.
4) Висновки

Дата видачі завдання "_____"_____20___ р.

Керівник бакалаврської кваліфікаційної роботи
д.е.н., проф. _____ Руденський Р.А
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Карась П.П.
(підпис) (ПІБ студента)

АНОТАЦІЯ

Дана бакалаврська кваліфікаційна робота присвячена розробці програмного забезпечення системи автоматичного моніторингу диму на природних територіях. Предметною областю є процеси збору, аналізу візуальної інформації та оперативного реагування на виявлені ознаки задимлення з метою запобігання лісовим пожежам. Основною проблемою, що вирішується, є підвищення ефективності та швидкості детектування потенційних загроз шляхом автоматизації моніторингу.

У ході роботи було проведено аналіз предметної області, спроєктовано архітектуру системи та розроблено ключові програмні модулі. Реалізовано функціонал для управління об’єктами моніторингу, користувачами з різними ролями, а також механізм завантаження зображень, їх аналізу на наявність диму та сповіщення відповідальних осіб.

Результатом роботи є функціональне програмного забезпечення, що дозволяє автоматизувати процес моніторингу диму, сприяючи своєчасному виявленню та реагуванню на потенційні пожежі на природних територіях.

ANNOTATION

This bachelor's qualification thesis is dedicated to the development of software for an automated smoke monitoring system in natural territories. The subject area covers the processes of collecting, analyzing visual information, and promptly responding to detected signs of smoke to prevent forest fires. The main problem addressed is enhancing the efficiency and speed of detecting potential threats through monitoring automation.

During the work, an analysis of the subject area was conducted, the system architecture was designed, and key software modules were developed. Functionality for managing monitoring objects, users with different roles, as well as a mechanism for uploading images, analyzing them for smoke presence, and notifying responsible personnel was implemented.

The result of the work is a functional software prototype that automates the smoke monitoring process, contributing to the timely detection and response to potential fires in natural territories. The developed system has the potential for further development and implementation into the practical activities of relevant services.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- UML – Unified Modeling Language
- LoRaWAN – low-power, wide-area networking protocol
- Mesh-архітектура - це топологія мережі, де кожен вузол може зв'язуватися з будь-яким іншим вузлом без центрального контролера
- GIS – Geographic Information System
- AI – Artificial intelligence
- ER-діаграма – Entity-Relationship діаграма
- DB – Database
- API – Application Programming Interface
- ACID – atomicity, consistency, isolation, durability
- СУБД – система управління базами даних

ВСТУП

Сучасні екологічні виклики, зокрема зростання частоти та масштабів лісових пожеж, зумовлюють нагальну потребу в розробці та впровадженні ефективних рішень виявлення задимлення. Лісові пожежі завдають значних економічних збитків, руйнують екосистеми, становлять загрозу для людського життя та здоров’я, а також погіршують якість повітря. Актуальність завдання, що вирішується у даній кваліфікаційній роботі, полягає у створенні програмного інструментарію, здатного автоматизувати процес моніторингу природних територій та забезпечити оперативне виявлення ознак задимлення, що є первинним індикатором потенційної пожежі. Своєчасне виявлення задимлення дозволяє значно скоротити час реагування відповідних служб, мінімізувати наслідки та запобігти розповсюдженню вогню на великі площі.

Метою розробки програмного додатку є створення програмного забезпечення системи автоматичного моніторингу диму на природних територіях. З огляду на актуальність проблеми, розроблювана система спрямована на підвищення ефективності раннього виявлення ознак задимлення шляхом аналізу візуальних даних, забезпечення оперативного сповіщення відповідальних осіб та надання зручних інструментів для управління процесом моніторингу та реагування. Це дозволить покращити координацію дій пожежних служб та сприятиме збереженню природних ресурсів.

При розробці програмного додатку було використано сучасний стек методів та технологій. Для серверної частини застосовано середовище виконання Bun з мовою програмування JavaScript, що забезпечує високу продуктивність та статичну типізацію. В якості системи управління базами даних обрано PostgreSQL завдяки її надійності та підтримці складних типів даних. Клієнтський інтерфейс реалізовано

з використанням бібліотеки React для серверного рендерингу HTML-шаблонів, що сприяє швидкодії та оптимізації. Інтерактивність забезпечується бібліотекою HTMX, яка дозволяє оновлювати частини сторінки без повного перезавантаження, мінімізуючи використання JavaScript на клієнті. Стилізація інтерфейсу виконана за допомогою CSS-фреймворку Tailwind CSS. Для контейнеризації та спрощення розгортання використовується Docker.

Структура пояснювальної записки відображає логіку дослідження та етапи розробки програмного продукту. Робота складається зі вступу, чотирьох основних розділів, висновків, списку використаних джерел 6 та 2 додатків. Загальний обсяг записки становить 58 сторінок.

У першому розділі проведено системний аналіз предметної області, визначено актуальність теми, сформульовано вимоги до програмної системи, здійснено огляд існуючих аналогічних рішень та поставлено завдання на розробку.

Другий розділ присвячений проєктуванню інформаційного та програмного забезпечення. Тут представлено логічну модель даних у вигляді ER-діаграми, діаграму класів та кооперацій, а також діаграми пакетів та компонентів, що відображають архітектуру системи.

У третьому розділі описано процес розробки інформаційної бази та прикладного програмного забезпечення, обґрунтовано вибір інструментарію, а також наведено приклади алгоритмізації та програмування ключових програмних модулів.

Четвертий розділ містить рекомендації щодо впровадження та експлуатації розробленої системи, включаючи опис процесу тестування, вимоги до апаратного та програмного забезпечення, а також склад інсталяційного пакету.

У висновках підсумовано результати виконаної роботи та окреслено перспективи подальшого розвитку проекту.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметною областю даної кваліфікаційної роботи є процес автоматизованого моніторингу та раннього виявлення ознак задимлення на природних територіях. До таких територій належать лісові масиви, національні парки, заповідники та інші екологічно значущі об'єкти. Актуальність розробки програмного забезпечення для таких систем зумовлена зростаючою загрозою лісових пожеж. Ці пожежі призводять до значних екологічних та економічних збитків, а також становлять небезпеку для життя та здоров'я населення.

Ключовими аспектами предметної області є:

1. Збір візуальних даних: Це отримання зображень з контрольованих територій за допомогою стаціонарних або мобільних систем відеоспостереження, зокрема камер. Цей аспект включає питання розміщення обладнання, його технічних характеристик, таких як роздільна здатність, кут огляду та чутливість, а також каналів передачі даних.
2. Обробка та аналіз зображень: Передбачається застосування алгоритмів комп'ютерного зору та, потенційно, методів машинного навчання для ідентифікації характерних ознак диму на зображеннях. Це включає виділення текстур, кольорових характеристик, динаміки розповсюдження та інших патернів, що вказують на наявність диму.
3. Прийняття рішень та сповіщення: На основі результатів аналізу формується висновок про наявність або відсутність задимлення. У разі виявлення ознак

диму, система повинна генерувати оперативні сповіщення для відповідального персоналу, такого як оператори, диспетчери та пожежні служби. Сповіщення мають містити інформацію про локацію та, можливо, інтенсивність задимлення.

4. Управління системою та даними: Необхідно забезпечити функціонал для конфігурації системи та управління обліковими записами користувачів, серед яких адміністратори, оператори та пожежники. Також система має вести журнал подій, архівувати візуальні дані та результати аналізу для подальшого розслідування інцидентів або навчання моделей.
5. Взаємодія користувачів: Важливим є визначення ролей користувачів, наприклад, адміністратора системи, оператора моніторингового центру чи керівника пожежного підрозділу. Також визначається їхня взаємодія з системою через відповідні інтерфейси для отримання інформації, управління та реагування.

Програмне забезпечення, що розробляється, має на меті автоматизувати та оптимізувати ці процеси. Його впровадження підвищить швидкість та точність виявлення задимлень, тим самим сприяючи ефективнішому запобіганню та ліквідації пожеж на природних територіях. Система повинна інтегрувати збір даних, їх аналіз, механізми сповіщення та інструменти управління в єдине програмне рішення.

1.2 Аналіз вимог до програмної системи

Аналіз вимог до програмного забезпечення системи автоматичного моніторингу диму на природних територіях є критичним етапом, що визначає функціональність, характеристики та обмеження майбутнього продукту. Вимоги формуються на основі потреб потенційних користувачів та завдань, які система повинна вирішувати. Їх можна класифікувати на функціональні, нефункціональні та вимоги до інтерфейсу користувача.

1.2.1 Функціональні вимоги

Функціональні вимоги описують конкретні дії та операції, які система повинна виконувати:

1. Реєстрація та автентифікація користувачів:
   - Система повинна забезпечувати можливість реєстрації нових користувачів з різними ролями: адміністратор, оператор, пожежник.
   - Користувачі повинні мати можливість безпечного входу в систему за допомогою унікальних облікових даних (логін та пароль).

2. Управління об'єктами моніторингу:
   - Система повинна відображати список усіх об'єктів моніторингу.
   - Адміністратор повинен мати можливість додавати, редагувати та видаляти об'єкти моніторингу, вказуючи їх назву, географічне положення та розмір.

3. Управління персоналом:

- Адміністратор повинен мати можливість додавати, редагувати та видаляти облікові записи операторів та пожежників, для операторів, закріплювати за ними конкретні об'єкти моніторингу.

4. Завантаження та аналіз зображень:

- Авторизований оператор повинен мати можливість завантажувати зображення з закріпленого за ним об'єкта моніторингу для аналізу на наявність диму.
- Система повинна автоматично обробляти завантажене зображення за допомогою інтегрованого алгоритму виявлення диму.
- Результат аналізу повинен відображатися оператору.

5. Сповіщення про інциденти:

- У разі виявлення диму система повинна генерувати сповіщення.
- Сповіщення мають надсилатися пожежникам, відповідальним за задимлену територію
- Сповіщення повинно містити інформацію про місцезнаходження інциденту та час виявлення.

6. Ведення журналу подій:

- Система повинна автоматично реєструвати всі ключові події: вхід/вихід користувачів, завантаження зображень, результати аналізу, генерація сповіщень, зміни в налаштуваннях.
- Журнал подій повинен бути доступний для перегляду адміністраторам.

1.2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи:

1. Безпека:
- Паролі користувачів повинні зберігатися у зашифрованому вигляді.
- Доступ до різних функцій системи повинен обмежуватися відповідно до ролей користувачів.

2. Простота використання:
- Інтерфейс користувача повинен бути інтуїтивно зрозумілим та легким для освоєння користувачами з різним рівнем технічної підготовки.
- Навігація по системі має бути логічною та послідовною.

3. Підтримуваність:
- Код системи повинен бути добре структурованим та легким для модифікації та розширення.

1.3 Моделювання предметної області

Моделювання предметної області є фундаментальним етапом у процесі розробки програмного забезпечення, що передує безпосередньому проєктуванню та реалізації системи. Метою цього етапу є створення чіткого, структурованого та формалізованого опису ключових концепцій, процесів, сутностей та їх взаємозв'язків у рамках розроблюваної системи. Побудова моделей дозволяє глибше зрозуміти специфіку функціонування майбутньої системи, виявити потенційні проблеми та неузгодженості на ранніх стадіях, а також забезпечити єдине бачення проекту для всіх учасників розробки.

У даному розділі будуть представлені моделі предметної області, розроблені з використанням UML. Ці моделі візуалізують різні аспекти системи:

- Діаграма прецедентів визначить основні функціональні можливості системи та взаємодію з нею різних типів користувачів (акторів).
- Діаграми послідовності деталізують динаміку взаємодії між об'єктами системи під час виконання ключових сценаріїв використання.
- Діаграми активності опишуть логіку складних робочих процесів та потоків управління в системі.

На рис. 1 представлена діаграма прецедентів, що візуалізує основні функціональні можливості програмного забезпечення системи автоматичного моніторингу диму на природних територіях та взаємодію з нею ключових зовнішніх акторів. Діаграма відображає системні вимоги з точки зору користувача, окреслюючи основні сценарії використання розроблюваної системи.

Система взаємодіє з такими основними акторами:

- Працівник пожежної служби: відповідає за безпосереднє реагування на інциденти задимлення
- Оператор: моніторинг об'єктів та отримує первинні сповіщення
- Керівник департаменту: відповідає за аналіз даних про інциденти та за загальне управління системою

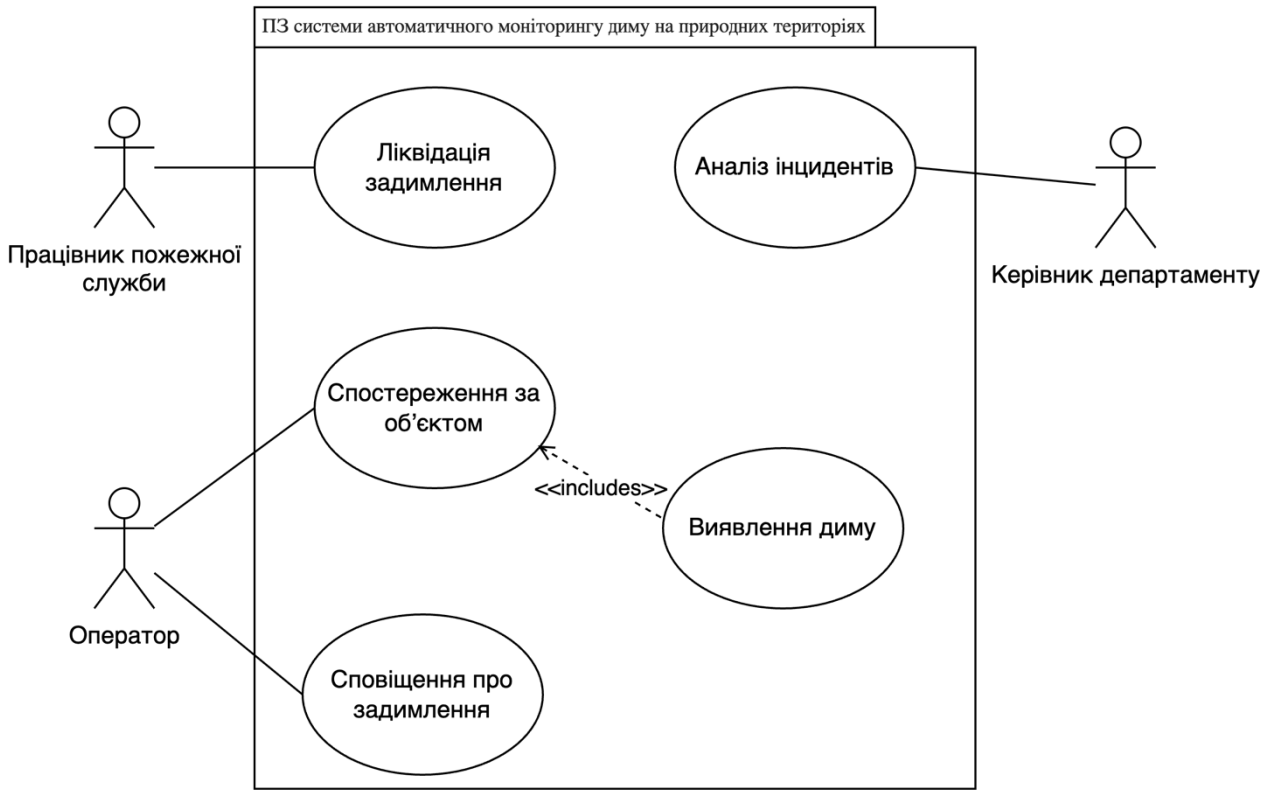

Рис.1 Діаграма прецедентів програмної системи

На рис. 2 представлена діаграма послідовності, яка ілюструє часову взаємодію між акторами системи. Діаграма демонструє потік повідомлень та послідовність дій, що виконуються різними ролями в системі.

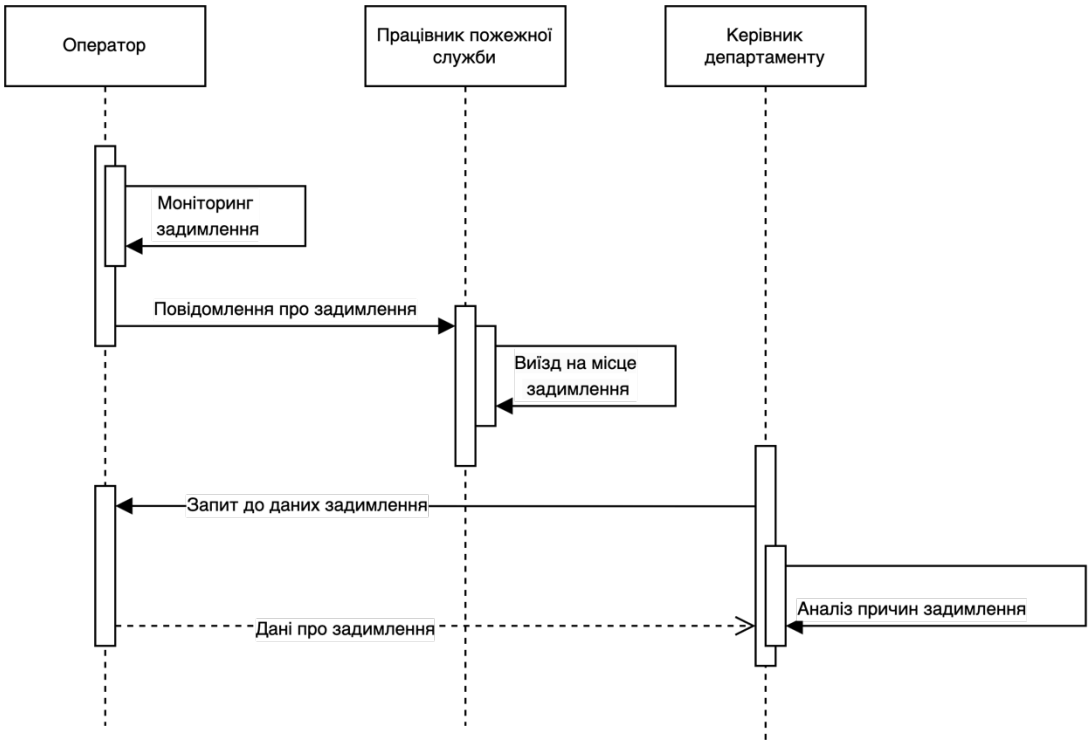

Рис. 2 Діаграма послідовності програмної системи

На рис. 3 представлена діаграма активності, що моделює основний робочий процес системи автоматичного моніторингу диму на природних територіях. Діаграма візуалізує послідовність дій, умовні переходи та ключові етапи обробки інформації від моменту отримання вказівки на спостереження до потенційного усунення задимлення.

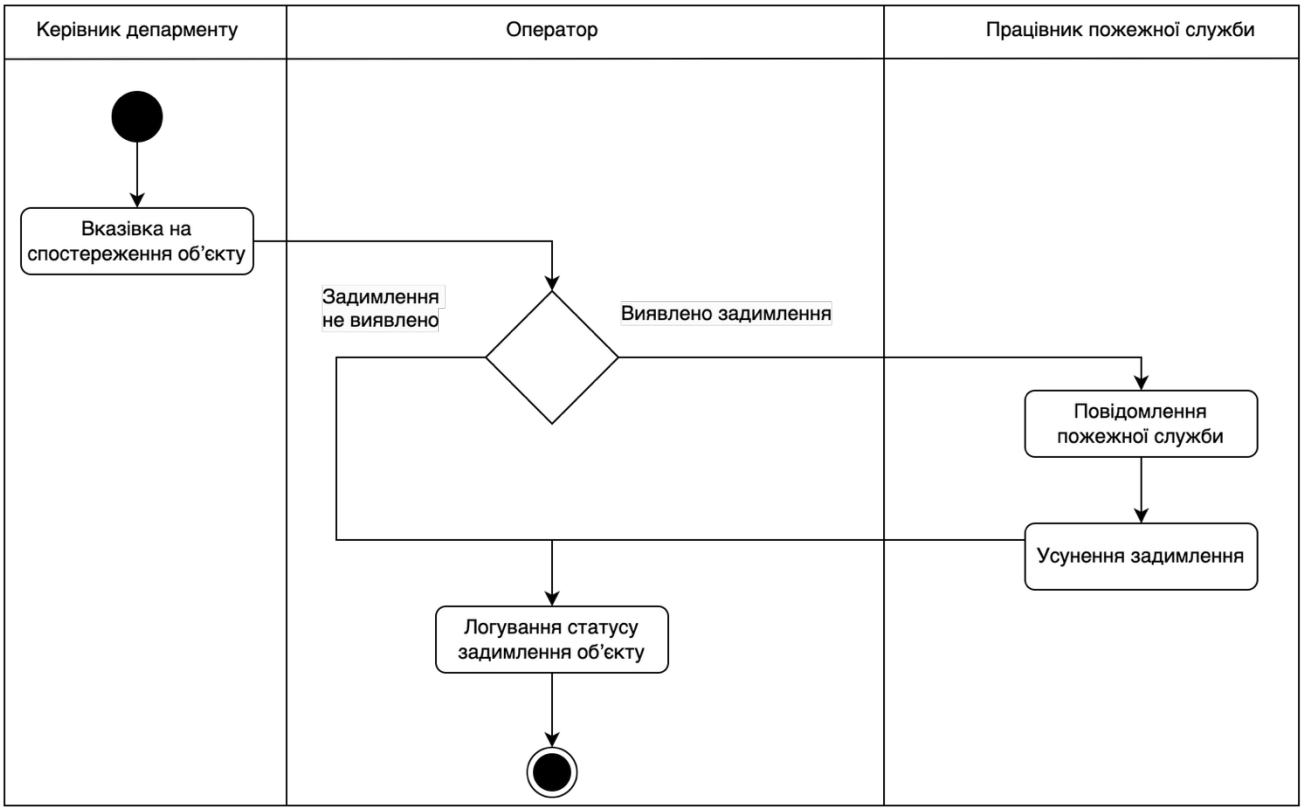

Рис.3 Діаграма активності програмної системи

Застосування цих моделей забезпечить системний підхід до аналізу предметної області та створить міцну основу для подальших етапів розробки програмного забезпечення.

1.4 Огляд інформаційних джерел та існуючих рішень

Невід'ємним етапом розробки нової програмної системи є ретельний аналіз вже існуючих рішень у відповідній або суміжній предметній області. Такий огляд дозволяє не тільки ознайомитися з актуальними технологіями та підходами, що застосовуються для вирішення подібних завдань, але й виявити усталені практики, потенційні переваги та недоліки наявних продуктів. Це, дає змогу уникнути повторення вже відомих помилок, визначити унікальні аспекти та конкурентні переваги розроблюваної системи, а також обґрунтувати доцільність її створення.

Dryad Silvanet – це інноваційна система ультра-раннього виявлення лісових пожеж, яка використовує мережу бездротових газових сенсорів, розміщених безпосередньо в лісових масивах. Сенсори працюють на сонячній енергії та виявляють зміни у складі повітря, характерні для початкових стадій горіння, ще до появи відкритого полум'я чи видимого диму. Дані з сенсорів передаються через LoRaWAN протокол до хмарної платформи для аналізу за допомогою штучного інтелекту.
Переваги:

- Ультра-раннє виявлення: Здатність виявляти пожежу на стадії тління, задовго до появи візуальних ознак, що дає значно більше часу для реагування.
- Енергоефективність та автономність: Використання сонячних панелей та технології LoRaWAN забезпечує тривалу автономну роботу сенсорів без частої заміни батарей
- Масштабованість та покриття: Mesh-архітектура мережі дозволяє покривати великі території, включаючи важкодоступні ділянки лісу

- Екологічність: Мінімальний вплив на довкілля завдяки сонячній енергії та відсутності потреби у прокладанні кабелів

Недоліки:

- Залежність від щільності сенсорної мережі: Ефективність виявлення прямо залежить від кількості та розташування сенсорів. Для великих територій може знадобитися значна кількість пристроїв, що впливає на вартість.
- Чутливість до напрямку вітру: Газові сенсори можуть не зафіксувати початкове загоряння, якщо продукти горіння не досягнуть їх через напрямок вітру.
- Відсутність візуального підтвердження: Система базується на аналізі газів, тому для підтвердження інциденту може знадобитися додаткова візуальна верифікація.
- Вразливість сенсорів: Фізичні сенсори в лісі можуть бути пошкоджені дикими тваринами або погодними умовами.

EVS ForestWatch – це система моніторингу лісових пожеж, що базується на використанні оптичних та тепловізійних камер, встановлених на висотних щоглах або існуючих вежах. Система використовує алгоритми штучного інтелекту для аналізу відеопотоку в режимі реального часу, автоматично виявляючи стовпи диму або теплові аномалії. ForestWatch забезпечує панорамний огляд, можливість керування камерами для детального огляду та інтеграцію з GIS.

Переваги:

- Візуальне підтвердження: Система надає безпосереднє візуальне підтвердження наявності диму або вогню, що дозволяє операторам швидко оцінити ситуацію.
- Великий радіус дії: Сучасні камери можуть моніторити значні території з однієї точки.
- Точне визначення місцезнаходження: Інтеграція з GIS та можливість точного наведення камери дозволяють досить точно визначати координати осередку задимлення.
- Цілодобовий моніторинг: Використання тепловізійних камер забезпечує ефективне виявлення вночі та за умов обмеженої видимості.

Недоліки:

- Залежність від видимості: Ефективність оптичних камер знижується за поганих погодних умов або за наявності перешкод.
- Виявлення на більш пізній стадії: Система реагує на вже видимий дим або значне теплове випромінювання, що може бути пізніше, ніж виявлення газовими сенсорами.
- Висока вартість інфраструктури
- Енергоспоживання: Камери споживають значно більше енергії порівняно з автономними сенсорами.
- Потенціал хибних спрацювань: Незважаючи на використання AI, система може реагувати на об'єкти, схожі на дим

1.5 Постановка завдання

На основі проведеного аналізу предметної області та огляду існуючих рішень, метою розробки програмного забезпечення є системи моніторингу диму на природних територіях. Ключовим завданням є створення програмного комплексу, здатного ефективно обробляти візуальну інформацію для виявлення ознак задимлення та оперативного інформування працівників пожежної служби. Система повинна забезпечити централізоване управління процесом моніторингу, включаючи адміністрування користувачів з різними рівнями доступу, такими як адміністратори, оператори та пожежники, а також управління об'єктами, що підлягають моніторингу.

Необхідно реалізувати функціонал для завантаження зображень операторами з закріплених за ними об'єктів та їх подальшого аналізу виявлення задимлення. У випадку виявлення задимлення, система має мати механізм генерації та надсилання сповіщень пожежному департаменту для уникнення задимлення.

Програмне забезпечення повинно включати ведення журналу значущих подій та дій користувачів для забезпечення можливості аудиту та аналізу інцидентів. Також передбачається розробка інтуїтивно зрозумілих веб-інтерфейсів для кожної ролі користувача, що забезпечують зручний доступ до функціоналу системи. Кінцевим результатом має стати програмний продукт, готовий до розгортання та експлуатації, що сприятиме підвищенню ефективності виявлення задимлень та запобіганню лісовим пожежам.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних є ключовим етапом проєктування інформаційного забезпечення системи. Вона візуалізує структуру даних, основні сутності предметної області, їх атрибути та взаємозв'язки між ними, незалежно від конкретної системи управління базами даних, яка буде використана для фізичної реалізації. Для представлення логічної моделі даних системи автоматичного моніторингу диму на природних територіях використовується нотація ER-діаграми.

Розроблена ER-діаграма (рис. 4) відображає основні інформаційні об'єкти, необхідні для функціонування системи, та логічні зв'язки, що існують між ними. Ця модель слугує основою для подальшої розробки фізичної схеми бази даних та забезпечує цілісність і несуперечливість даних у системі.

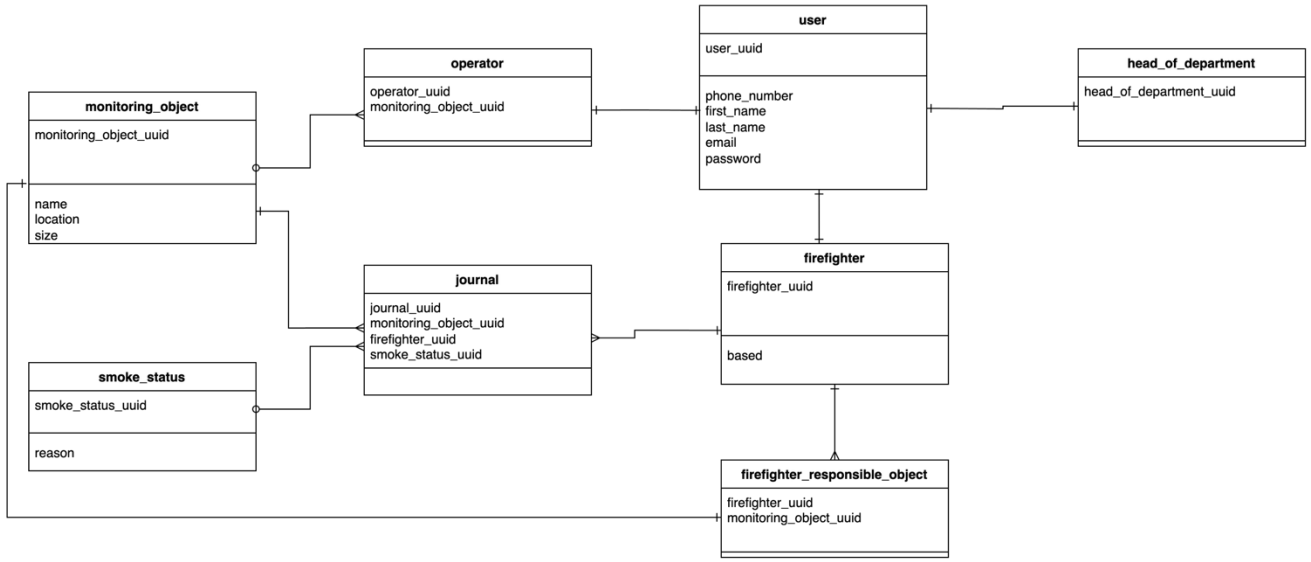

Рис. 4 Логічна модель даних

На представленій ER-діаграмі визначено наступні ключові сутності та їх взаємозв'язки:

- user: Базова сутність, що представляє будь-якого користувача системи. Ця сутність містить загальні атрибути, такі як ідентифікатор, логін, хешований пароль, адресу електронної пошти, телефон та роль. Ця сутність є узагальненням для більш специфічних ролей.
- operator: Представляє оператора, відповідального за моніторинг об’єкту.
- firefighter: Представляє працівника пожежної служби. Має додатковий атрибут для визначення локації працівника пожежної служби.
- head_of_department: Представляє керівника, відповідального за загальне управління та аналіз задимлень.

- monitoring_object: Представляє природну територію або конкретну локацію, що підлягає моніторингу. Атрибути можуть включати унікальний ідентифікатор, назву, географічні координати, розмір території та опис об’єкту.
- smoke_status: Сутність, що визначає можливі стани задимлення.
- journal: Сутність для фіксації подій моніторингу та інцидентів. Кожен запис у журналі пов'язаний з конкретним monitoring_object. Запис пов'язаний з firefighter, який відреагував на інцидент, та зі smoke_status для фіксації статусу задимлення.
- firefighter_responsible_object: Ця сутність визначає за які об’єкти відповідає працівник пожежної служби.

Типи зв’язків:

- Зв'язки між user та operator, firefighter, head_of_department є зв'язками *один до одного*, що реалізують спеціалізацію/успадкування. Кожен оператор, пожежник чи керівник є користувачем, але не кожен користувач є, наприклад, оператором.
- Зв'язок між operator та monitoring_object є *один до одного*, один оператор може відповідає за один об’єкт.
- Зв'язок між journal та monitoring_object: *багато до одного*, на одному об’єкті може відбуватись багато інцидентів.

- Зв'язок між journal та firefighter: *багато до одного*, один пожежник може вирішувати багато інцидентів.
- Зв'язок між journal та smoke_status: зв’язок *багато до одного*.
- Зв'язки між firefighter_responsible_object та firefighter: зв’язок *багато до одного*.

Дана логічна модель забезпечує необхідну структуру для зберігання інформації про користувачів системи, об'єкти моніторингу, події задимлення та пов'язані з ними статуси, що є основою для ефективного функціонування програмного забезпечення.

2.2 Діаграма класів та кооперацій

Діаграма класів є статичним представленням системи, що описує набір класів, їх атрибути, методи та відносини між ними, такі як асоціації, агрегації, композиції та успадкування. Вона слугує для деталізації структури програмних компонентів та є основою для подальшої реалізації коду.

Діаграма класів для системи автоматичного моніторингу диму на природних територіях (рис. 5) розроблена на основі сутностей, ідентифікованих у логічній моделі даних, та відображає їх програмне представлення.

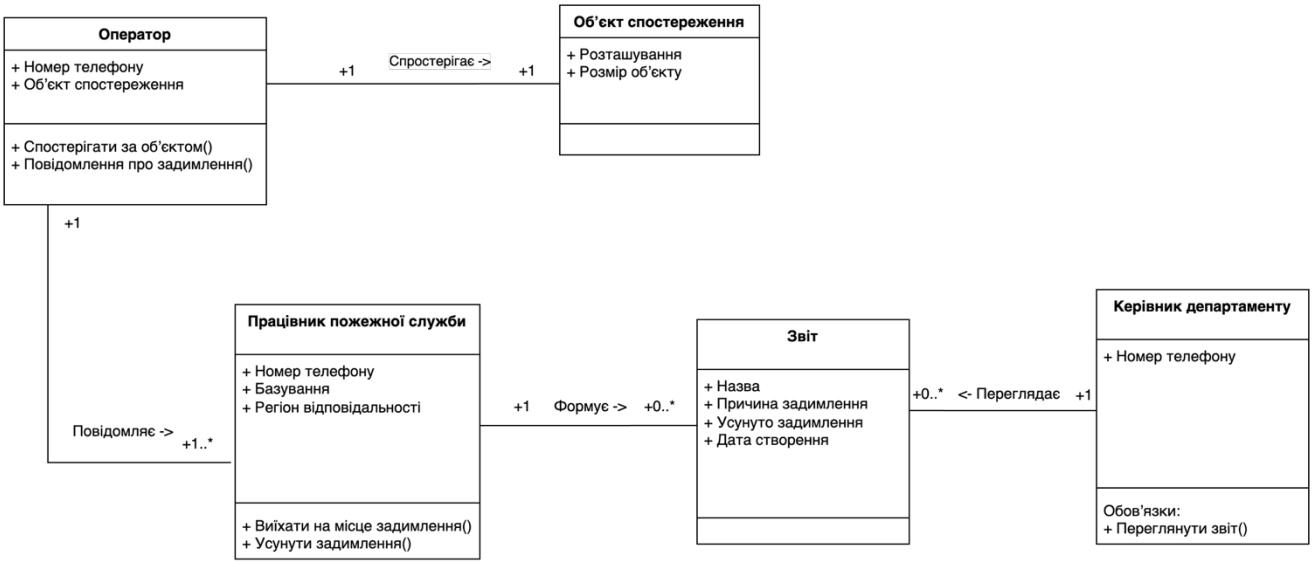

Рис. 5 Діаграма класів програмної системи

На представленій діаграмі класів маємо наступні класи:

- Оператор: Цей клас представляє користувача системи, відповідального за безпосередній моніторинг стану об'єктів. Його основна відповідальність полягає у спостереженні за Об'єктом спостереження, виявленні потенційного задимлення задимлення та ініціюванні відповідних дій, таких як сповіщення Працівника пожежної служби.
- Об'єкт спостереження: Цей клас моделює природну територію або локацію, яка підлягає моніторингу. Він містить інформацію, необхідну для ідентифікації та характеристики об'єкта, наприклад, його назву, географічне положення та поточний статус.
- Працівник пожежної служби: Цей клас представляє особу, відповідальну за реагування на інциденти задимлення. Він отримує інформацію про виявлені загрози від Оператора та виконує дії з ліквідації задимлення.

- Керівник департаменту: Цей клас представляє користувача з вищим рівнем доступу, відповідального за аналіз ефективності системи моніторингу, прийняття стратегічних рішень та контроль за роботою персоналу. Він взаємодіє з класом Звіт для отримання узагальненої інформації про інциденти та стан системи.
- Звіт: Цей клас моделює інформаційний артефакт, що містить деталізовані дані про інциденти задимлення.

Дана діаграма класів деталізує статичну структуру програмних елементів системи, визначаючи їх атрибути, потенційні операції та взаємозв'язки. Вона слугуватиме основою для написання коду та подальшого проєктування кооперацій між об'єктами цих класів.

На рис. 6 представлена діаграма кооперації, що ілюструє взаємодію між об'єктами системи під час виконання сценарію *Спостереження за об’єктом*. Діаграми кооперації фокусуються на структурних зв'язках між об'єктами, що беруть участь у взаємодії, та повідомленнях, якими вони обмінюються.

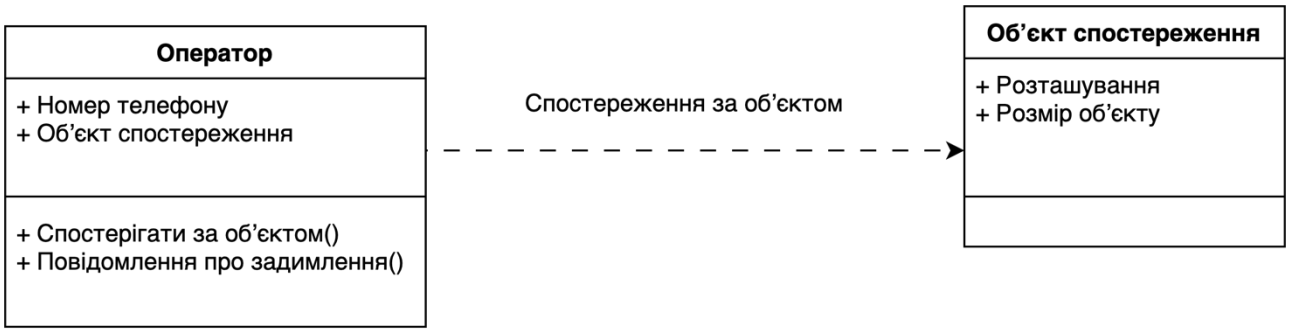

Рис. 6 Діаграма кооперації: Спостереження за об'єктом

У даній кооперації беруть участь наступні об'єкти:

- Оператор: Представляє конкретного оператора системи, який виконує функцію моніторингу об’єкта
- Об’єкт спостереження.

Ця діаграма кооперації надає спрощене уявлення про взаємодію двох ключових об'єктів у процесі моніторингу. Вона підкреслює роль оператора як активного учасника, що здійснює спостереження за конкретним об'єктом. Для більш повного розуміння процесу можуть бути розроблені додаткові діаграми кооперації, що включатимуть інші об'єкти та деталізуватимуть обмін повідомленнями під час виявлення задимлення, сповіщення та інших пов'язаних дій.

2.3 Діаграма пакетів

Діаграма пакетів є важливим інструментом для візуалізації високорівневої архітектури програмної системи. Вона дозволяє згрупувати пов’язані елементи системи в пакетами, та показати залежності між ними. Такий підхід сприяє кращому розумінню структури системи, її модульності та взаємодії між основними функціональними блоками.

На рис. 7 представлена діаграма пакетів для серверної частини системи автоматичного моніторингу диму. Ця діаграма ілюструє основні програмні модулі сервера та їх взаємозв'язки.

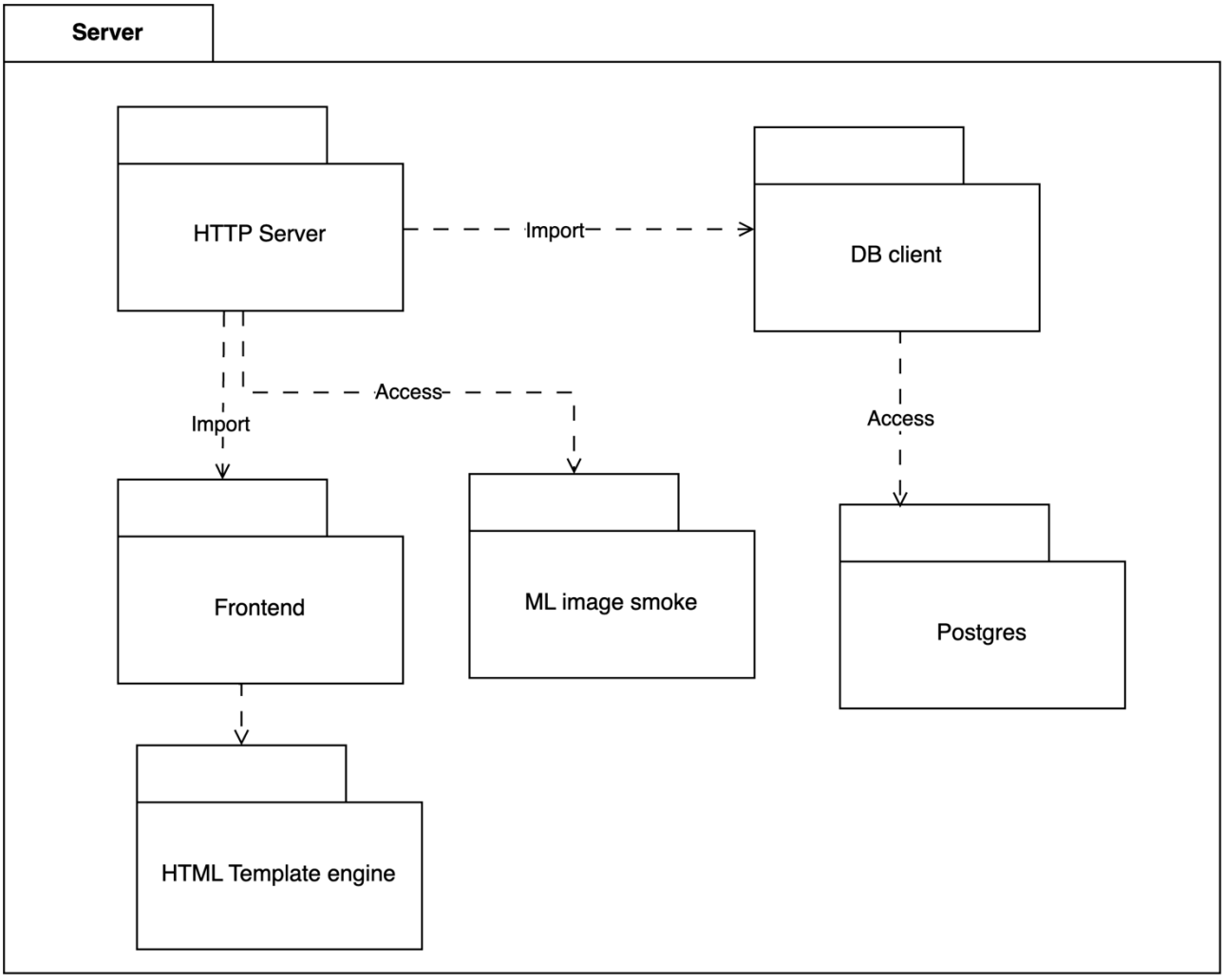

Рис. 7 Діаграма пакетів серверної частини системи

Представлена діаграма пакетів для серверної частини системи. Вона ілюструє ключові модулі, відповідальні за обробку запитів, бізнес-логіку, взаємодію з базою даних та інші серверні функції.

- Server: Це кореневий пакет, що представляє всю серверну частину програмного забезпечення системи моніторингу диму.
- HTTP Server: Цей пакет відповідає за обробку HTTP-запитів від клієнтів, маршрутизацію запитів до відповідних обробників та формування HTTP-

відповідей. Він є основною точкою входу для взаємодії з системою через веб-інтерфейс та API.

- DB client: Пакет, що інкапсулює логіку взаємодії з системою управління базами даних. Він надає абстрактний шар для виконання операцій читання та запису даних, незалежно від конкретної реалізації бази даних.
- Postgres: Цей пакет має доступ до Postgres для виконання операцій з даними.
- Frontend: Цей пакет містить логіку, пов'язану з генерацією та представленням користувацького інтерфейсу. Він відповідає за рендеринг HTML-сторінок, які надсилаються клієнту.
- HTML Template engine: Пакет, що надає інструменти для генерації HTML-коду на основі шаблонів та даних.
- ML image smoke: Пакет, що реалізує функціональність аналізу зображень для виявлення ознак диму.

Дана діаграма пакетів чітко структурує серверну частину системи, виділяючи основні модулі та їхні взаємозв’язки, що сприяє кращому розумінню архітектури та полегшує подальшу розробку та підтримку програмного забезпечення.

2.4 Діаграма компонентів

Діаграма компонентів є важливим інструментом моделювання фізичної архітектури програмної системи. Вона візуалізує організацію та залежності між програмними компонентами, які є модульними, замінними частинами системи, що надають певну функціональність через чітко визначені інтерфейси. Ця діаграма допомагає зрозуміти, як система розбита на фізичні блоки, як ці блоки взаємодіють між собою, та як вони реалізують логічні елементи, визначені на попередніх етапах проєктування, наприклад, класи.

Для системи автоматичного моніторингу диму на природних територіях розроблено діаграму компонентів на рис. 8, що відображає основні програмні компоненти та їх стереотипи.

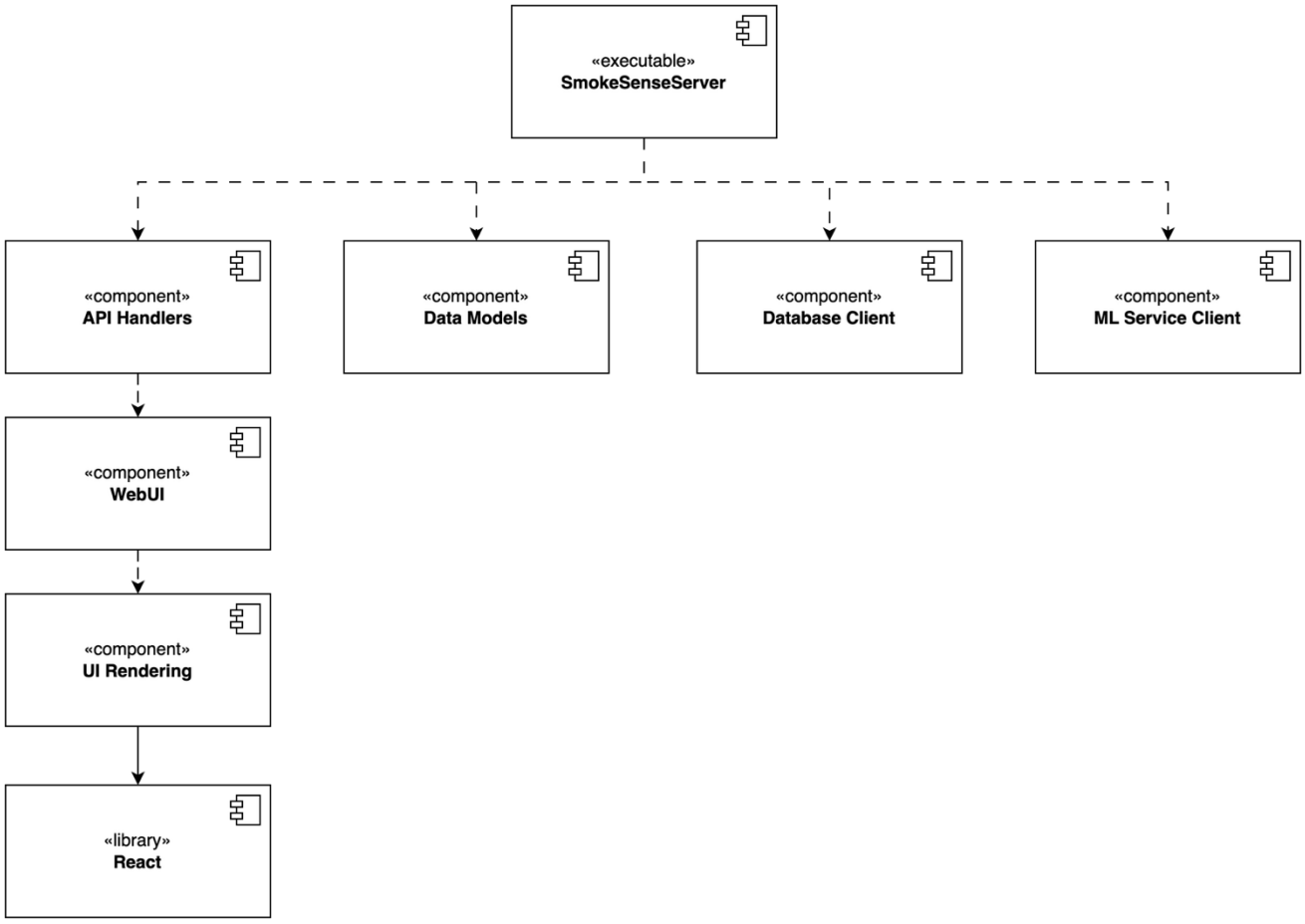

Рис. 8 Діаграма компонентів програмної системи

На представленій діаграмі компонентів визначено наступні ключові програмні компоненти, що складають архітектуру системи:

- SmokeSenseServer: Це головний виконуваний компонент системи, що представляє серверний додаток. Він відповідає за координацію роботи всіх інших серверних компонентів, обробку запитів, управління бізнес-логікою та надання основного функціоналу системи.
- API Handlers: Цей компонент інкапсулює логіку обробки запитів до API. Він відповідає за прийом запитів від клієнтської частини, їх валідацію, виклик відповідних сервісів для обробки та формування відповідей.

- Data Models: Компонент, що представляє моделі даних системи. Він містить визначення структур даних, що використовуються в системі, та логіку для взаємодії з цими даними, забезпечуючи абстракцію над рівнем зберігання даних.
- Database Client: Цей компонент відповідає за взаємодію з системою управління базами даних.
- ML Service Client: Призначений для взаємодії із зовнішнім сервісом, який відповідає за аналіз зображень на предмет виявлення диму.
- WebUI: Цей компонент представляє сукупність елементів користувацького веб-інтерфейсу. Він включає сторінки, форми, візуальні елементи та логіку, необхідну для взаємодії користувача з системою через браузер.
- UI Rendering: Компонент, відповідальний за процес рендерингу користувацького інтерфейсу. Він може включати механізми шаблонізації, генерації HTML на сервері та логіку динамічного оновлення інтерфейсу на клієнті.
- React: Бібліотека яка використовується для побудови користувацького інтерфейсу.

Дана діаграма компонентів дає уявлення про фізичну структуру програмного забезпечення, його основні модулі та технології, що використовуються для їх реалізації. Вона є важливою для розуміння того, як система збирається з окремих частин та як ці частини взаємодіють для забезпечення загальної функціональності.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Ключовим елементом інформаційного забезпечення будь-якої програмної системи є СУБД, яка відповідає за надійне зберігання, організацію, доступ та маніпулювання даними. Для даного проєкту було обрано об'єктно-реляційну систему управління базами даних PostgreSQL. Цей вибір зумовлений низкою технічних характеристик та функціональних можливостей, що оптимально відповідають вимогам розроблюваної системи.

PostgreSQL відома своєю надійністю, відповідністю стандартам ACID, що гарантує транзакційну цілісність даних, що є критично важливим для системи, яка обробляє інформацію про потенційні загрози та інциденти. Розширена підтримка різноманітних типів даних, включаючи вбудовані геометричні типи для зберігання локаційних даних об'єктів моніторингу, та можливість використання типу UUID для генерації унікальних ідентифікаторів, забезпечує гнучкість у моделюванні предметної області. Крім того, PostgreSQL пропонує потужні механізми для забезпечення цілісності даних через обмеження, тригери та збережені процедури, а також розвинені засоби для індексації, що сприяють високій продуктивності запитів.

Взаємодія серверної частини додатку з базою даних PostgreSQL реалізованої за допомогою Bun, що забезпечують ефективне та безпечне з'єднання. Bun, дозволяє легко інтегруватися з PostgreSQL, використовуючи асинхронні операції для неблокуючої взаємодії з базою даних, що є важливим для підтримки високої

продуктивності серверного додатку. Обрана СУБД надає стабільну та масштабовану платформу для зберігання всієї інформації, необхідної для функціонування системи моніторингу, включаючи дані про користувачів, об'єкти спостереження, журнал подій та статуси задимлення.

3.2 Розробка інформаційної бази

Розробка інформаційної бази є практичною реалізацією логічної моделі даних, описаної в попередніх розділах, та передбачає створення фізичної структури бази даних. Цей етап включає створення таблиць, їх атрибутів з відповідними типами даних, первинних та зовнішніх ключів для забезпечення зв'язків між таблицями, а також інших обмежень цілісності.

Фізична структура бази даних для системи автоматичного моніторингу диму на природних територіях була розроблена з урахуванням принципів нормалізації, зокрема, з метою досягнення третьої нормальної форми. Такий підхід спрямований на мінімізацію надлишковості даних, усунення потенційних аномалій при їх оновленні, видаленні чи додаванні, та забезпечення логічної узгодженості інформації, що зберігається.

Нижче наведено опис створених таблиць, що визначають структуру основних таблиць інформаційної бази:

- Таблиця *monitoring_object* призначена для зберігання інформації про об'єкти, що підлягають моніторингу. Вона включає унікальний ідентифікатор об'єкта, його назву, географічне положення у вигляді точки та розмір.
- Таблиця *users* слугує для зберігання загальної інформації про всіх користувачів системи. Вона містить унікальний ідентифікатор користувача, його роль у системі, контактні дані, такі як номер телефону, ім’я, прізвище, адресу електронної пошти, а також хешований пароль для автентифікації.
- Для реалізації специфічних ролей користувачів створено окремі таблиці, що посилаються на базову таблицю *users* через зовнішній ключ, реалізуючи таким чином успадкування:

  - Таблиця *firefighter* зберігає додаткову інформацію про працівників пожежної служби, зокрема місце їх базування.
  - Таблиця *operator* містить інформацію про операторів системи та може включати посилання на об'єкт моніторингу, за який відповідає оператор.
  - Таблиця *head_of_department* представляє керівників департаменту.

- Таблиця *firefighter_responsible_objects* є асоціативною таблицею, що реалізує зв'язок "багато до багатьох" між пожежниками та об'єктами моніторингу, вказуючи, за які об'єкти відповідає конкретний пожежник.
- Таблиця *user_tokens* призначена для зберігання токенів користувачів, забезпечуючи механізм сесій та безпечного доступу.
- Таблиця *smoke_status* є довідником можливих статусів задимлення, що використовуються для класифікації інцидентів.
- Таблиця *journal* слугує для ведення журналу подій та інцидентів. Кожен запис пов'язаний з об'єктом моніторингу, може містити посилання на пожежника, який реагував, та статус задимлення.

Застосування первинних та зовнішніх ключів, а також обмежень *not null* та *UNIQUE,* забезпечує підтримку консистентності даних в інформаційній базі. Використання типу *uuid* для первинних ключів сприяє унікальності ідентифікаторів у розподіленому середовищі, а тип *point* дозволяє ефективно зберігати та обробляти географічні координати.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Вибір інструментарію є критично важливим аспектом розробки будь-якого програмного продукту, оскільки він безпосередньо впливає на ефективність процесу розробки, продуктивність, масштабованість та підтримуваність кінцевої системи. Для створення прикладного програмного забезпечення системи автоматичного моніторингу диму на природних територіях було обрано сучасний та гнучкий стек технологій, що дозволяє реалізувати поставлені функціональні та нефункціональні вимоги.

3.3.1 Серверна частина (Backend)

Основою серверної частини є середовище виконання Bun. Цей вибір зумовлений його високою продуктивністю, швидким часом запуску та вбудованою підтримкою TypeScript, що значно спрощує процес розробки та забезпечує переваги статичної типізації. Bun також пропонує оптимізовані API для роботи з файловою системою, мережею та іншими системними ресурсами, що є важливим для серверних додатків. Мовою програмування для серверної логіки обрано JavaScript. Для взаємодії з базою даних PostgreSQL використовуються відповідні драйвери та бібліотеки, сумісні з середовищем Bun, що забезпечують ефективне виконання SQL-запитів.

3.3.2 Клієнтська частина (Frontend) та взаємодія з сервером

Для побудови користувацького інтерфейсу та забезпечення його динамічності було прийнято рішення використовувати комбінацію React для серверного рендерингу HTML-шаблонів та бібліотеку HTMX для організації клієнт-серверної взаємодії.

Використання React на сервері дозволяє створювати складні та компонентно-орієнтовані інтерфейси, генеруючи статичний HTML, що передається клієнту. Такий підхід сприяє швидшому першому завантаженню сторінки. Ключовою перевагою такого підходу є централізація стану програми на сервері, що спрощує логіку клієнтської частини.

HTMX обрано для реалізації інтерактивності без необхідності завантаження великих JavaScript-фреймворків на клієнт. HTMX дозволяє оновлювати частини веб-сторінки, надсилаючи HTTP-запити до сервера. Це ідеально доповнює SSR-підхід, оскільки сервер продовжує відповідати за генерацію HTML, а HTMX забезпечує динаміку з мінімальним використанням JavaScript на клієнті. Такий підхід зменшує складність фронтенду та покращує загальну продуктивність.

Для стилізації інтерфейсу використовується CSS-фреймворк Tailwind CSS. Його підхід *utility-first* дозволяє швидко створювати унікальні дизайни без написання власного CSS, застосовуючи утилітарні класи безпосередньо в HTML-розмітці. Це прискорює розробку, забезпечує консистентність стилів та мінімізує розмір кінцевих CSS-файлів завдяки механізму очищення невикористаних стилів.

Обраний набір інструментів забезпечує збалансоване поєднання продуктивності, гнучкості розробки, надійності та можливостей для подальшого розвитку та підтримки програмного забезпечення системи моніторингу диму.

3.4 Алгоритмізація та програмування програмних модулів

Етап алгоритмізації та програмування є центральним у процесі створення програмного забезпечення. Він передбачає розробку детальних алгоритмів для ключових функцій системи та їх подальшу реалізацію у вигляді програмних модулів мовою програмування, обраною на попередніх етапах. У даному підрозділі розглянуто алгоритм доступу до захищеної сторінки та продемонстровано фрагменти коду, що ілюструють реалізацію серверної логіки та взаємодії з базою даних.

Для забезпечення безпеки та розмежування доступу до різних частин системи, реалізовано механізм перевірки авторизації користувача перед наданням доступу до захищених ресурсів, наприклад, адміністративної панелі. Алгоритм цього процесу представлений на рис. 9.

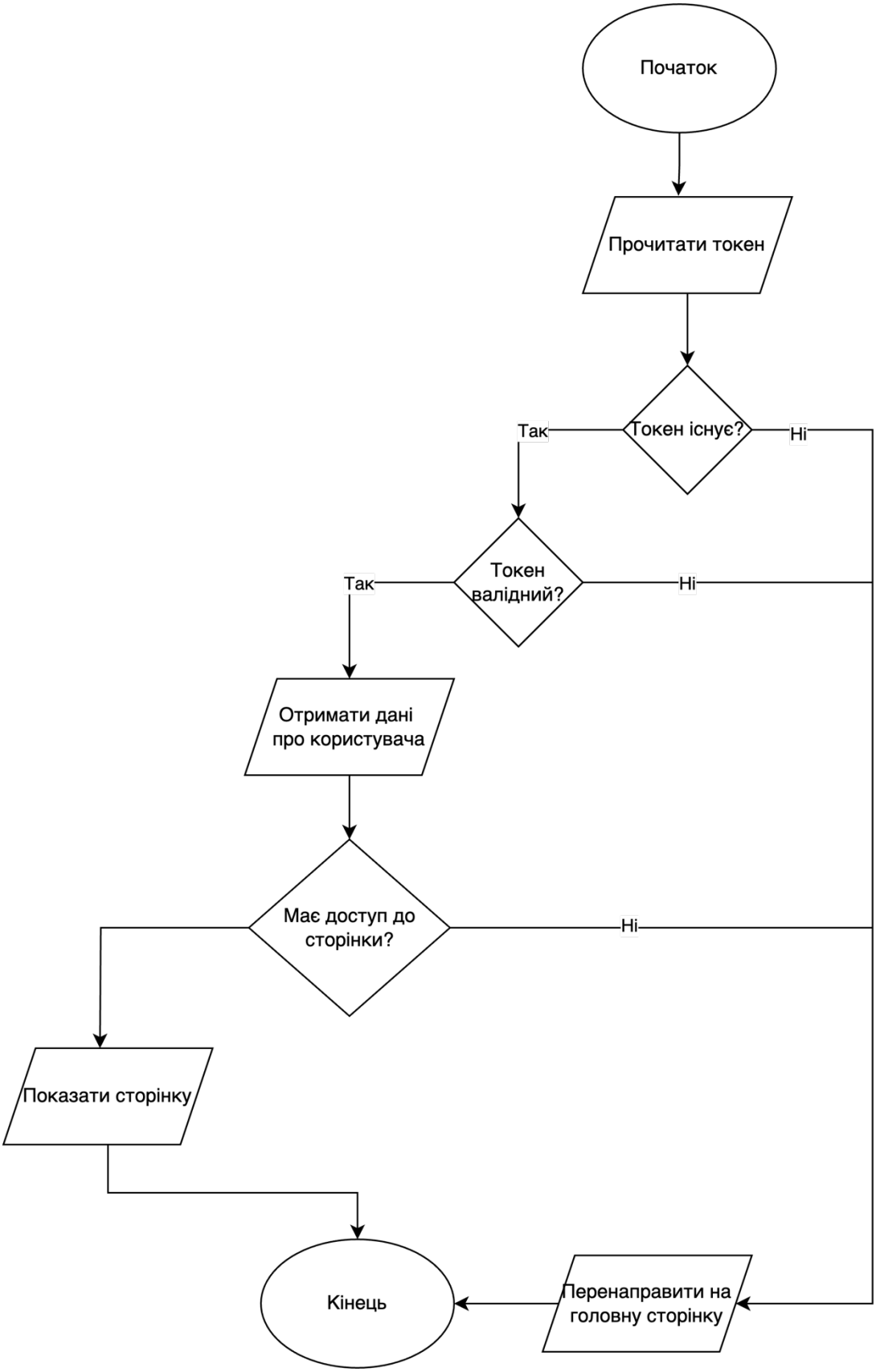
Початок
Прочитати токен
Токен існує?
Так
Ні
Токен валідний?
Так
Ні
Отримати дані про користувача
Має доступ до сторінки?
Ні
Показати сторінку
Кінець
Перенаправити на головну сторінку

Рис. 9 Блок-схема алгоритму доступу до захищеної сторінки

Процес починається з отримання HTTP-запиту від клієнта на доступ до певної сторінки. Система намагається вилучити автентифікаційний токен з HTTP-cookie запиту. Якщо токен відсутній, користувач вважається неавторизованим, і доступ до сторінки забороняється, шляхом перенаправлення на сторінку входу. Якщо токен наявний, система перевіряє його валідність, шукаючи відповідного користувача в базі даних. У разі успішної валідації токена та знаходження користувача, перевіряється його роль. Якщо роль користувача відповідає вимогам доступу до запитуваної сторінки – доступ надається. В іншому випадку, якщо роль не відповідає, доступ забороняється, і користувач перенаправляється на головну сторінку.

Весь програмний код алгоритму представлений у Додаток Б.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення є невід'ємним етапом життєвого циклу розробки, спрямованим на виявлення помилок, перевірку відповідності системи визначеним вимогам та забезпечення її якості. Процес тестування включає планування, проєктування тестових випадків, їх виконання та аналіз результатів. Для системи автоматичного моніторингу диму на природних територіях було проведено функціональне тестування для перевірки коректності роботи основних функцій системи з точки зору різних ролей користувачів.

Функціональне тестування для ролі керівника департаменту має на меті перевірку можливостей управління основними сутностями системи та доступу до адміністративних панелей. Процес тестування включав наступні кроки:

- Автентифікація: Перевірка можливості успішного входу в систему з обліковими даними адміністратора (рис. 10 – головна сторінка, рис. 11 – сторінка входу). Тестувалися як коректні дані, так і спроби входу з невірними логіном/паролем для перевірки обробки помилок.
- Доступ до адміністративної панелі: Після успішної автентифікації перевірявся доступ до головної адміністративної панелі на рис. 12, яка відображає узагальнену інформацію та надає доступ до розділів управління.

- Управління пожежниками: Тестувалася можливість перегляду списку пожежників, додавання нового пожежника, редагування інформації про існуючого пожежника та його видалення. Перевірялася коректність відображення даних у таблиці.
- Управління операторами: Аналогічно до управління пожежниками, тестувалися операції перегляду, додавання, редагування та видалення операторів, а також можливість закріплення за оператором конкретного об'єкта моніторингу.
- Управління об'єктами моніторингу: Перевірялася функціональність створення нових об'єктів моніторингу із зазначенням назви, локації та розміру, а також можливість редагування та видалення існуючих об'єктів.
- Перевірка відображення даних: На всіх етапах контролювалася коректність відображення введених та збережених даних у відповідних таблицях та формах адміністративної панелі.

Рис. 10 Головна сторінка системи

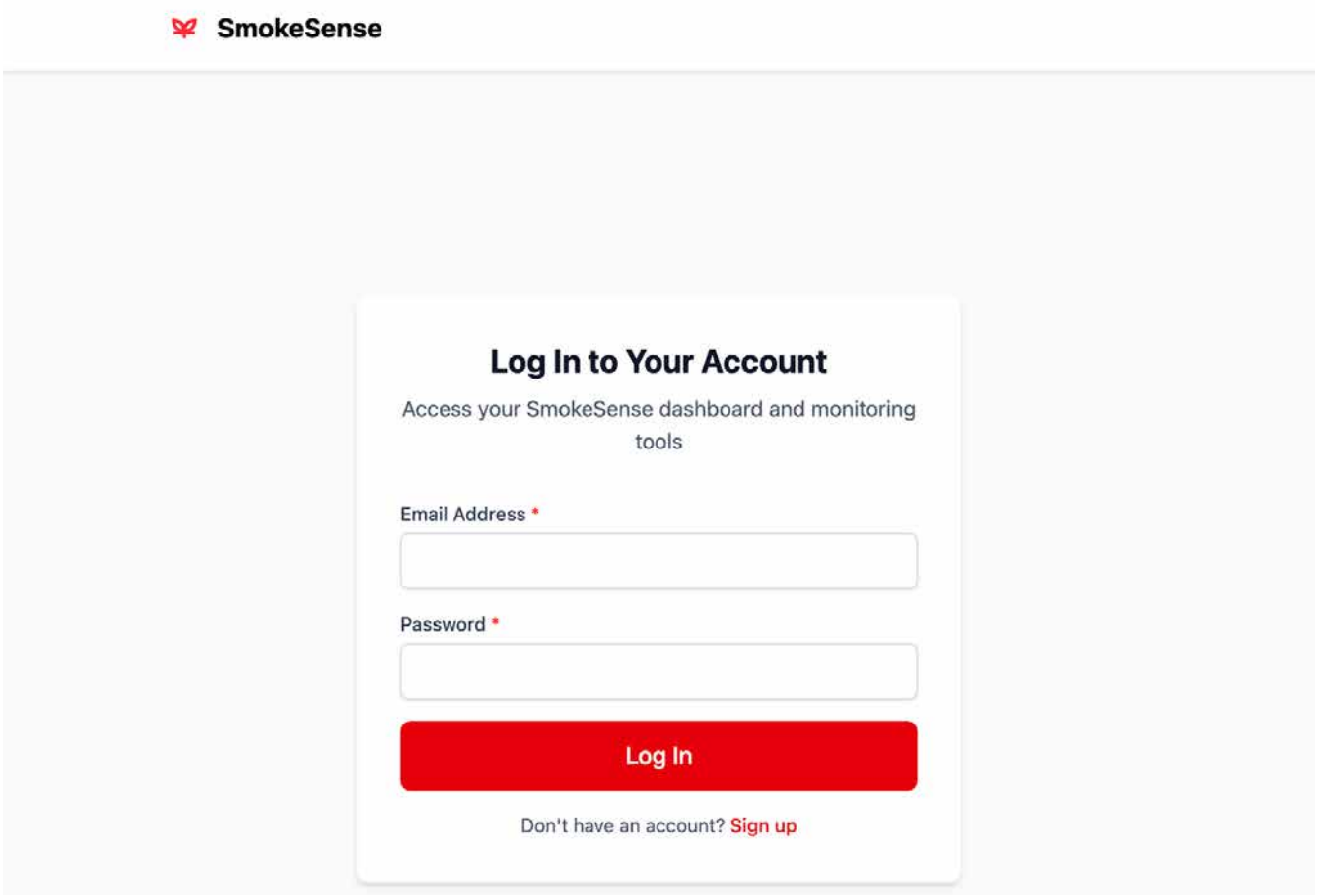

Рис. 11 Сторінка автентифікації користувача

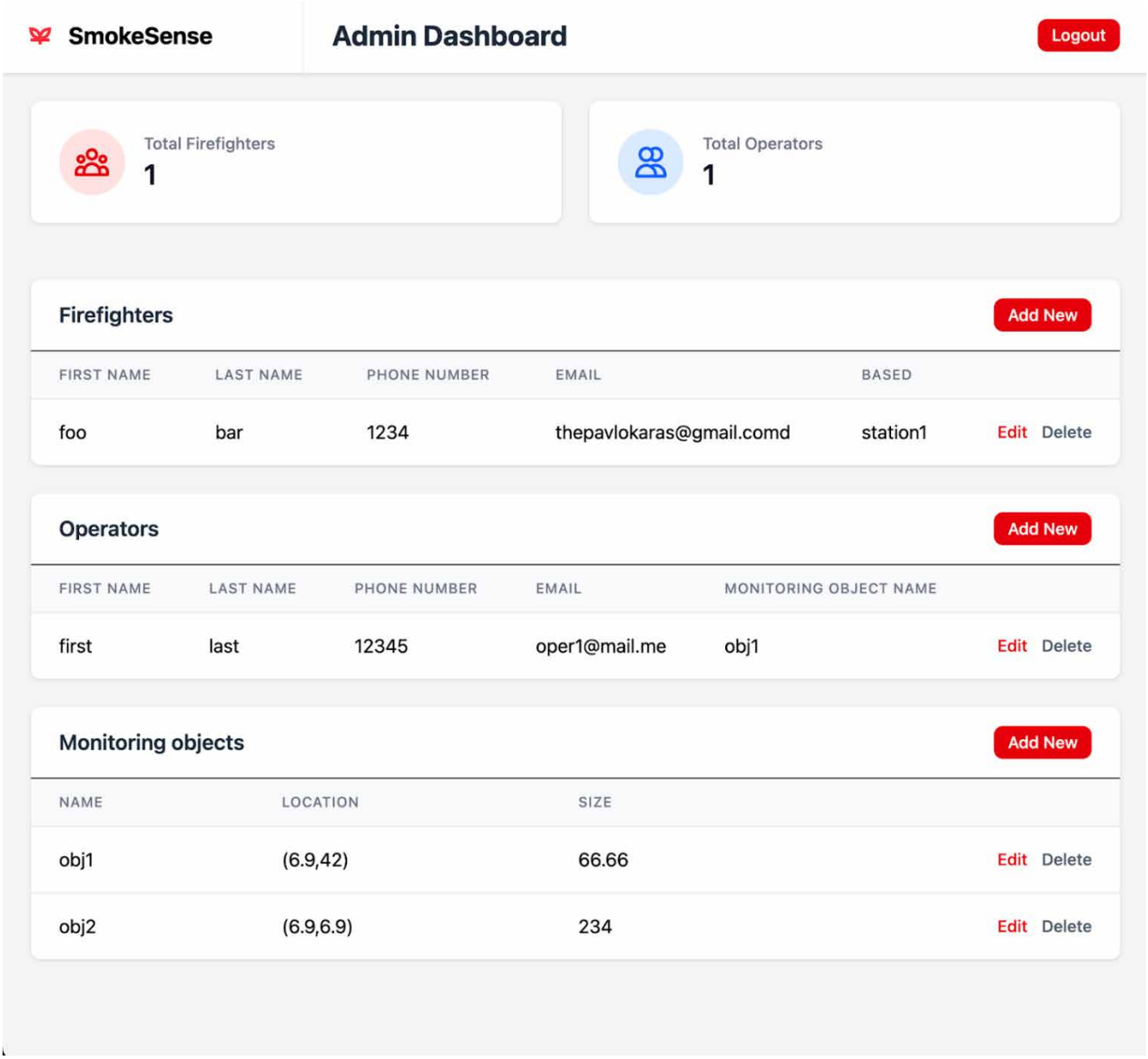

Рис. 12 Адміністративна панель Керівника департаменту

За результатами функціонального тестування для ролі адміністратора було підтверджено коректну реалізацію основних управлінських функцій, включаючи створення, редагування та видалення ключових сутностей системи, а також захищений доступ до адміністративної панелі.

Проведене функціональне тестування для різних ролей користувачів дозволило переконатися в працездатності основних модулів системи та їх

4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної та ефективної роботи програмного забезпечення необхідно дотримуватися певних вимог до апаратних ресурсів та програмного оточення як для серверної, так і для клієнтської частин системи. Ці вимоги базуються на архітектурі системи, використаних технологіях та очікуваному навантаженні.

4.2.1 Вимоги до серверної частини

Серверна частина відповідає за обробку всієї бізнес-логіки, взаємодію з базою даних, аналіз зображень та обслуговування запитів від клієнтів.

Апаратні вимоги:

- Процесор: 2-ядерний процесор з тактовою частотою не менше 2.5 ГГц
- Оперативна пам’ять: Мінімум 1 ГБ
- Дисковий простір: SSD накопичувач для забезпечення високої швидкості доступу до даних, мінімум 10 ГБ вільного простору для системи, бази даних та додатку
- Мережеве підключення: Стабільне високошвидкісне підключення до Інтернету

Програмні вимоги:

- Операційна система: Ubuntu Linux 20+
- Середовище виконання: Встановлена актуальна версія середовища виконання Bun, що використовується для запуску серверного додатку
- Система управління базами даних: Встановлений та налаштований PostgreSQL 17+

4.2.2 Вимоги до клієнтської частини

Клієнтська частина призначена для взаємодії користувачів із системою через веб-інтерфейс.

Апаратні вимоги:

- Процесор: Чотириядерний процесор з тактовою частотою 1.6 ГГц або вище.
- Оперативна пам'ять: 8 ГБ або більше для комфортної роботи з веб-інтерфейсом та іншими програмами
- Вільне місце на SSD: Мінімум 200 МБ для кешу браузера та тимчасових файлів
- Мережеве підключення: Стабільне підключення до Інтернету зі швидкістю, достатньою для завантаження веб-сторінок та потенційної передачі зображень.
- Дисплей

Програмні вимоги:

- Веб-браузер: Встановлений веб-браузер Google Chrome версії 130 або новішої, з увімкненою підтримкою JavaScript та Cookies.
- Мережевий доступ: Можливість доступу до веб-адреси системи через мережу Інтернет або внутрішню мережу.

Дотримання зазначених вимог забезпечить належну працездатність, продуктивність та стабільність розробленої системи як на серверному, так і в клієнтському середовищах.

4.3 Склад інсталяційного пакету

Враховуючи сучасні підходи до розробки та деплою програмного забезпечення, основним методом розгортання системи є використання технології контейнеризації Docker. Такий підхід гарантує узгодженість середовища виконання, ізоляцію додатку та спрощує процес інсталяції та оновлення.

Інсталяційний пакет, представлений набором файлів та інструкцій, необхідних для побудови та запуску Docker-образу системи. До його складу входять:

- Вихідний код програмного забезпечення: Усі файли проєкту, включаючи серверну логіку, клієнтські компоненти, файли конфігурації та інші необхідні ресурси, що містяться у відповідних директоріях проекту.

- Файл Dockerfile: Спеціальний конфігураційний файл, що містить покрокові інструкції для автоматичної побудови Docker-образу. Він визначає базовий образ, копіювання вихідного коду, встановлення всіх необхідних залежностей та команди для запуску серверного додатку всередині контейнера.
- Файли управління залежностями: Файли package.json та bun.lockb, що містять перелік усіх програмних залежностей проєкту та їхні точні версії, необхідні для коректної збірки та роботи додатку.
- Файли конфігурації середовища: .env файл необхідний для роботи системи для підключення до бази даних PostgreSQL, URL-адреси зовнішніх сервісів та інші конфігураційні налаштування.

Процес інсталяції системи полягає у виконанні інструкцій з Dockerfile для створення готового до запуску Docker-образу, який вже містить у собі скомпільований додаток та всі його залежності. Такий підхід значно спрощує розгортання, оскільки усуває необхідність ручного налаштування серверного оточення та встановлення окремих компонентів на цільовій машині.

ВИСНОВКИ

У рамках даної роботи було успішно розроблено програмне забезпечення системи автоматичного моніторингу диму на природних територіях, досягнувши поставленої мети створення програмного комплексу для виявлення задимлень. Проведено детальний аналіз предметної області, сформульовано вимоги до системи та здійснено моделювання за допомогою UML-діаграм, що забезпечило чітке розуміння функціональних можливостей та сценаріїв взаємодії.

На етапі проєктування було створено логічну модель даних у вигляді ER-діаграми та відповідну діаграму класів, що деталізували структуру інформаційного та програмного забезпечення. Архітектура системи, представлена діаграмами пакетів та компонентів, відобразила модульну організацію серверної та клієнтської частин. Для реалізації було обрано сучасний стек технологій, включаючи PostgreSQL, JavaScript, React, HTMX та Tailwind CSS, що дозволило ефективно реалізувати ключовий функціонал: управління користувачами, об’єктами та аналіз зображень.

Завершальний етап включав формулювання рекомендацій щодо впровадження системи, опис процесу тестування, визначення апаратних та програмних вимог, а також складу інсталяційного пакету на основі Docker. Розроблений програмний продукт є функціональною системою, що демонструє реалізацію основних вимог та підтверджує набуті навички в аналізі, проєктуванні та розробці програмних систем, маючи потенціал для подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Silvanet: Ultra-early Wildfire Detection by Dryad Networks. URL: https://www.dryad.net/silvanet (дата звернення: 15.05.2025).
2. ForestWatch: Wildfire Detection System by EVS. URL: https://www.evsusa.biz/forestwatch (дата звернення: 15.05.2025).
3. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/ (дата звернення: 18.05.2025).
4. Bun: Fast JavaScript all-in-one toolkit. URL: https://bun.sh/ (дата звернення: 23.05.2025).
5. htmx - high power tools for HTML. URL: https://htmx.org/ (дата звернення: 23.05.2025).
6. MDN Web Docs. URL: https://developer.mozilla.org/en-US/ (дата звернення: 23.05.2025).

ДОДАТКИ

Додаток А

Схема бази даних:

```
create table monitoring_object (
    monitoring_object_uuid uuid primary key default gen_random_uuid(),
    name                   varchar(255)     not null,
    location               point            not null,
    size                   double precision not null
);

create table users (
    user_uuid    uuid primary key default gen_random_uuid(),
    role         varchar(50)  not null, -- head_of_department, operator, firefighter

    phone_number varchar(255) not null unique,
    first_name   varchar(255) not null,
    last_name    varchar(255) not null,
    email        varchar(255) not null unique,
    password     varchar(255) not null
);

create table firefighter (
    firefighter_uuid uuid primary key references users (user_uuid),
```

```
    based           varchar(255) not null
);

create table firefighter_responsible_objects (
    firefighter_uuid       uuid references firefighter (firefighter_uuid),
monitoring_object_uuid  uuid  references  monitoring_object  (monitoring_object_uuid)
not null,
    primary key (firefighter_uuid, monitoring_object_uuid)
);

create table operator (
    operator_uuid          uuid primary key references users (user_uuid),
    monitoring_object_uuid uuid references monitoring_object (monitoring_object_uuid)
);

create table head_of_department (
    head_of_department_uuid uuid primary key references users (user_uuid)
);

create table user_tokens (
    token_uuid uuid primary key default gen_random_uuid(),
    token      bytea not null,
    user_uuid  uuid references users (user_uuid)
);

create table smoke_status (
    smoke_status_uuid uuid primary key default gen_random_uuid(),
    reason            varchar(255) not null
```

```
);

create table journal (
    journal_uuid            uuid primary key default gen_random_uuid(),
monitoring_object_uuid  uuid  references  monitoring_object  (monitoring_object_uuid)
not null,
    firefighter_uuid        uuid references firefighter (firefighter_uuid)          not null,
    smoke_status_uuid       uuid references smoke_status (smoke_status_uuid)
);
```

Додаток Б

Програмний код алгоритму доступу до захищеної сторінки:

```
static async getCurrentUser(e: Request) {
    const token = getCookie(e);
    if (!token) {
      return;
    }

    return await UserToken.getUserByToken(token);
  }

export const TOKEN = "token";

export function getCookie(req: Request, key = TOKEN) {
  const token = req.headers
    .get("cookie")
    ?.split("; ")
    .find((c) => {
      return c.startsWith(key);
    });

  if (!token) {
    return;
  }
```

```
  const [_k, v] = token.split("=");
  return v;
}


    if (!user || user.role !== "head_of_department") {
      return new Response("", {
        status: 302,
        headers: [["Location", "/"]],
      });
    }
```