ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття рівня вищої освіти «бакалавр»

(рівень вищої освіти)

Хмарна інформаційна система підтримки ройового комплексу
Cloud information system for supporting a swarm complex

Виконав: студент денної форми навчання
спеціальності 126 – Інформаційні системи та технології

(шифр і назва напряму підготовки, спеціальності)

Освітня програма «Інформаційні системи та технології»

(назва освітньої програми)

Швець Юлія Олександрівна

(прізвище, ім'я, по-батькові)

Керівник д.т.н., проф. Малахов Є. В.

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент к.т.н., доц Пенко В. Г.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№___ від «___»_____ 2024 р.

Завідувач кафедри

_____ _Євгеній МАЛАХОВ_

(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ___

протокол № __ від «_ _» ____ 2024 р.

Оцінка _____/_____/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _Володимир ВИЧУЖАНІН_

(підпис) (ім'я, прізвище)

Одеса - 2024

АНОТАЦІЯ

Метою даної роботи є розробка хмарної системи керування роєм, яка забезпечить швидкий доступ рою та оператора, який є частиною рою, до різнорідної інформації, необхідної для розв'язання оперативних задач та зберігання даних.

Для досягнення визначеної мети проведено детальний аналіз предметної області та визначено задачі, виконання яких має забезпечувати система для підтримки ройових комплексів. Спроектовано базу даних, що містить необхідну інформацію для ефективного управління роями, з урахуванням вимог щодо зберігання та обробки даних. Також розроблено застосунок, який інтегрує дані з бази даних та надає операторам зручний інтерфейс для налаштування параметрів рою та нодів, зокрема.

Важливим елементом системи є розміщення бази даних у хмарі, що забезпечує високу доступність даних, дозволяючи операторам та системам отримувати до них доступ у реальному часі незалежно від їхнього місцезнаходження. Одними з головних задач застосунку, є візуальне відслідковування роботи рою та аналіз результатів.

Реалізація виконана з використанням мови програмування C# та інтерфейсу програмування додатків WPF, що є частиною платформи Microsoft .NET, СУБД PostgreSQL та EntityFramework для забезпечення взаємодії з базою даних. Архітектура системи відповідає шаблону MVVM.

Результатом кваліфікаційної роботи є хмарна інформаційна система підтримки ройового комплексу, що забезпечує зручний та інтуїтивно зрозумілий інтерфейс для оператора, а також надійну та ефективну взаємодію рою з оператором та системою у цілому.

ABSTRACT

The aim of this qualification work is to develop a cloud-based swarm management system that should provide swarms and operators, who are also part of the swarm, with quick access to heterogeneous information required to solve operational tasks and store data.

To achieve this goal, a detailed analysis of the subject area was carried out and the tasks to be performed by the system to support swarm complexes were identified. A database containing the necessary information for effective swarm management was designed, taking into account the requirements for data storage and processing. An application has also been developed that integrates data from the database and provides operators with a user-friendly interface for configuring swarm and node parameters, in particular.

An important element of the system is the placement of the database in the cloud, which ensures high data availability, allowing operators and systems to access it in real time regardless of their location. One of the main tasks of the application is to visually track the swarm and analyse the results.

An important element of the system is the placement of the database in the cloud, which ensures high data availability, allowing operators and systems to access it in real time regardless of their location. One of the main tasks of the application is to visually track the swarm and analyse the results.

The implementation was carried out using the C# programming language and the WPF application programming interface, which is part of the Microsoft .NET platform, the PostgreSQL database and the EntityFramework to ensure interaction with the database. The system architecture follows the MVVM pattern.

The result of the qualification work is a cloud-based information system for supporting the swarm complex, which provides a convenient and intuitive interface for the operator, as well as reliable and efficient interaction between the swarm and the operator and the system as a whole.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

НС – надзвичайні ситуації.

Debezium – платформа для захоплення змін даних (CDC, Change Data Capture), яка дозволяє відстежувати зміни в базах даних в реальному часі та передавати ці зміни в системи для потокової передачі даних, наприклад, Kafka.

MVVM – Model – View – ViewModel (паттерн проєктування).

ВСТУП

Надзвичайні ситуації (НС) можуть виникнути в будь-який момент, а це, у свою чергу, створює загрозу для життя людей, що потрапили у них. Швидка та ефективна реакція рятувальних служб є надзвичайно важливою за таких обставин. Враховуючи зміну кліматичних умов та вплив людського фактора, частота виникнення подібних ситуацій постійно зростає [1].

Це, в свою чергу, підвищує навантаження на рятувальні служби та їхні ресурси, що вимагає пошуку більш ефективних та інноваційних методів реагування на подібні ситуації.

У випадках, коли доступ до місця події ускладнений або небезпечний, використання традиційних методів пошуку та порятунку може бути надзвичайно складним і ризикованим для рятувальників. Тож, за такої ситуації, гарним рішенням є залучення до пошуку автоматизованих систем, зокрема дронів. Їх здатність проникати в ускладнені та небезпечні місця – дозволяє їм досягати та розпізнавати об'єкти, недосяжні для людини. Це особливо корисно в ситуаціях, коли традиційні методи пошуку та порятунку недостатньо ефективні або небезпечні для життя рятувальників.

Автоматизовані системи мають забезпечувати розпізнавання об'єктів, що потребують допомоги. Проте, розпізнавання об’єктів вимагає залучення вагомої кількості ресурсів, а саме обчислювальної потужності процесору та пам’яті, якими один дрон володіти не може через власні обмежені технічні характеристики. Також поодиноке використання дронів не забезпечить виконання таких задач, як сканування цілого простору одночасно, застосовуючи при цьому різні методи та сенсори.

Застосування ройових комплексів у повній мірі вирішують дані проблеми. Залучення цілих ройових комплексів надає можливість пришвидшити виконання завдання та покращити показники розпізнавання

об’єктів, разом із вивченням території. Ройові комплекси складаються з нодів, які мають свою конкретну роль для вирішення заздалегідь поставленої задачі. Ноди, якими, в даному випадку, виступають дрони є різними за можливостями та потужністю, тож має бути забезпечена можливість отримання даних про кожен дрон рою окремо: які сенсори він може витримати та які методи або ж функції для розпізнавання є можливість завантажити. В даному випадку, виникає потреба у залученні до роботи рою оператора, як окремої ланки рою, яка зможе забезпечити виконання завдання. Необхідною є і хмарна інформаційна система [2], яка забезпечить постійний, безперебійний доступ до певного функціоналу, бази даних та забезпечить, в певній мірі, можливість автоматизованого конфігурування рою та його окремих нодів: додавання дрону, налаштування сенсорів з обмеженнями по їх нанесенню та методам, які можуть містити ноди. Також є необхідним забезпечити можливість роботи кількох роїв для виконання більшої кількості задач. Такий підхід може посприяти підвищенню продуктивності та ефективності використання ройових комплексів.

Хмарна система керування ройовими комплексами може включати в себе наступний функціонал: визначення нового завдання з додаванням в нього контрольних точок для побудови приблизного шляху, додавання у завдання нодів з їх конфігураціями, які також кожен раз визначаються оператором і можуть відрізнятися; зберігання інформації від нодів про їх стан, власне розміщення, ідентифіковані об’єкти та стан середовища загалом. Також вона передбачає можливість аналізу роботи рою та окремих нодів з їх конфігурацією, за кількістю якісно розпізнаних об’єктів.

Забезпечення безпеки та конфіденційності інформації є важливим аспектом роботи системи, оскільки вона здатна надавати можливість обміну чутливою інформацією, яка може стати об’єктом кібератак або порушень приватності. Для захисту на рівні застосунку може бути реалізована

авторизація, а на рівні бази даних – розмежування доступу з використанням ролей та привілеїв.

Необхідність вирішення вищезазначеної проблеми забезпечує актуальність даної роботи. Тож, метою є розробка хмарної системи керування роєм, яка має забезпечити швидкий та безперебійний доступ рою та оператора, який також є частиною рою, до різнорідної інформації, необхідної для розв’язання оперативних задач та зберігання даних. Впровадження такої системи може значно підвищити продуктивність та ефективність використання ройових комплексів у ситуаціях надзвичайних обставин.

Дана робота є комплексною роботою на рівні бакалаврської та здобувача освіти третього рівня. В рамках цієї роботи виконується прикладна робота, яка є частиною комплексного дослідження, а саме створення хмарної інформаційної системи.

Отже, для досягнення поставленої мети необхідно вирішити наступні завдання:

1) проаналізувати предметну область, тобто роботу роїв;
2) обрати архітектуру та шаблон проєктування для створення ІС;
3) спроєктувати базу даних;
4) розробити та реалізувати десктопний застосунок для інтерактивної взаємодії користувачів зі створеною ІС;
5) розробити та реалізувати API для взаємодії дронів з ІС;
6) забезпечити цілісність та безпеку даних як на рівні БД, так і на рівні застосунку;
7) забезпечити захист системи від несанкціонованого доступу і розмежування повноважень з боку різних категорій користувачів;
8) протестувати та налагодити роботу застосунку.

1 ПОСТАНОВКА ЗАДАЧІ СТВОРЕННЯ ХМАРНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПІДТРИМКИ РОЙОВОГО КОМПЛЕКСУ

1.1 Введення у предметну область

Для ефективного функціонування рою необхідно створити багаторівневу архітектуру, яка забезпечить оптимальне керування процесом розпізнавання об'єктів та координації дій дронів. Ця архітектура повинна включати рівні, які взаємодіють між собою для досягнення спільної мети [2].

Хмарна інформаційна система має на меті забезпечити виконання таких завдань:

1) забезпечення безперебійного доступу до інформації за надзвичайних ситуацій;

2) виділення ресурсів (дронів, сенсорів, методів) для забезпечення максимальної якості виконання поставлених задач;

3) створення задач, які мають бути виконані у разі надзвичайних ситуацій;

4) збереження та обробка інформації, що надходить від рою, оператора та інших користувачів системи;

5) збереження історії завдань для їх оцінки та можливого аналізу конфігурацій рою та окремих нодів.

У рамках даної роботи особливо важливою є розробка хмарної інформаційної системи. Дана система буде забезпечувати зв'язок та керування між роєм дронів та оператором за допомогою хмарних технологій. Вона буде відповідальна за збір, обробку та аналіз даних, які надходять від дронів, а також за надання відповідної інформації оператору для прийняття рішень [3].

Якщо розглядати оператора та рій, як окремі елементи єдиного організму, то почнемо з того, які задачі виконує оператор:

1) конфігурація нодів рою: визначення сенсорів, якими кожна нода має бути наділена, та які методи має використовувати для аналізу території та розпізнавання;

2) визначення контрольних точок, для побудови приблизного шляху рою та визначення території, яка має бути проаналізована роєм;

3) редагування стратегії дій рою у разі виявлення нових об’єктів або небезпек, шляхом зміни розташування рою або ж нодів у просторі в окремий момент часу;

4) участь у процесі збору та аналізу даних: однією з задач оператора є аналіз усієї мультимедіа інформації, що надходить від рою, шляхом перегляду файлів, які надіслано;

5) аналіз польоту: визначення найкращих конфігурацій нодів та комплектації рою для подальших завдань;

6) аналіз розпізнавання об’єктів, шляхом перевірки мультимедіа файлів, отриманих від нодів;

7) реконфігурація рою.

Ноди рою мають виконувати наступні задачі:

1) аналіз території та надсилання усієї отриманої інформації оператору для прийняття подальших рішень;

2) розпізнавання об’єктів з використанням методів, отриманих від оператора, та їх передача оператору;

3) слідування маршруту, заданому оператором з максимальним покриттям території, шляхом зміни положення нодів одна відносно одної;

4) побудова власного маршруту на базі контрольних точок, заданих оператором перед початком польоту;

5) оцінка власної дієздатності: якщо нода втрачає можливість роботи, то вона має надати інформацію про свій стан оператору та повернутися на базу;

6) спілкування з іншими нодами для обміну інформацією та оцінки покриття території.

Також, для забезпечення роботи рою мають бути присутніми AI-спеціалісти. Головною їх задачею є забезпечення оператора та нодів методами для розпізнавання об’єктів з повним їх описом та визначеними обмеженнями обчислювальної потужності.

Отже, згідно з переліченими задачами, що виконуються ройовими комплексами, система обов’язково має надавати наступні можливості:

1) формування складу рою з усіма необхідними налаштуваннями, тобто вибором сенсорів для кожної ноди, з наданням їй методів, базуючись на визначених (для методів) обмеженнях обчислювальних потужностей;

2) побудова контрольного маршруту рою;

3) відслідковування роботи рою, аналізуючи кількість інформації, що від них надходить та положення рою відносно кінця контрольного шляху;

4) керуванням польотом рою, шляхом додавання нових точок для окремих нодів або рою у цілому;

5) аналіз роботи рою з певними складом та налаштуваннями, шляхом визначення кількості об’єктів, що їх розпізнано нодами;

6) оцінка розпізнавання роями об’єктів, шляхом перевірки;

7) аналіз мультимедіа файлів отриманих від рою та пошук об’єктів безпосередньо оператором.

Додатковим завданням є моделювання процесу роботи рою, для того, аби мати можливість перевірити працездатність інформаційної системи.

1.2 Огляд систем для підтримки ройових комплексів

Для того, щоб оцінити користь від створюваної інформаційної системи треба, перш за все, проаналізувати ринок з існуючими пропозиціями, порівняти знайдені рішення з власною розробкою й визначити, як вона може виконати поставлені задачі краще за існуючі аналоги. Розглянемо кожен з них детальніше.

1.2.1 Drone Assist

Drone Assist [4] розроблено компанією Altitude Angel. Даний застосунок надає інтерактивну карту неба і використовується у всьому світі, проте він є особливо популярним у Великобританії. Даний застосунок призначено для забезпечення безпеки польотів дронів.

Однією з головних особливостей Drone Assist є інтерактивна карта неба, яка надає користувачам можливість бачити в режимі реального часу небезпечні зони, місця з обмеженнями для польотів та інші перешкоди. Це дозволяє операторам дронів уникати потенційно небезпечних ділянок та планувати свої маршрути більш ефективно.

Drone Assist також дозволяє попередньо планувати маршрути польотів, враховуючи обмеження та ризики, що сприяє підвищенню ефективності та безпеки польотів, особливо в умовах обмеженого повітряного простору. Функція «Fly Now» дозволяє операторам дронів ділитися своїм місцезнаходженням з іншими користувачами застосунку та ширшою спільнотою операторів дронів. Це сприяє кращій координації польотів та уникненню колізій у повітряному просторі.

Окрім того, Drone Assist надає інформацію про наземні об'єкти, які можуть представляти ризики для польотів, такі як будівлі, лінії електропередач, аеропорти та інші структури. Це допомагає уникнути зіткнень та інших інцидентів під час польоту.

Drone Assist має свої переваги та недоліки, що варто враховувати при його використанні для роботи рою та розпізнавання об'єктів. Серед переваг застосунку – забезпечення безпеки польоту через інтерактивну карту неба, можливість попереднього планування маршрутів, що забезпечує ефективність та безпеку польотів, та функція «Fly Now», яка дозволяє ділитися місцезнаходженням польоту з іншими користувачами, що сприяє уникненню колізій та спілкуванню з іншими операторами дронів.

Недоліками є обмежена географія: він може бути найбільш ефективним лише в областях, де є підтримка та оновлення геодезичних даних. Крім того, додаток не має спеціалізованого функціоналу для роботи рою дронів, що обмежує його ефективність у цій сфері. Застосунок також не передбачений для дрону з завданням розпізнавання об'єктів, що є критично важливим.

1.2.2 DJI FlightHub

DJI FlightHub [5] – є інструментом для управління роєм, який розроблено компанією DJI. Він забезпечує централізовану платформу для моніторингу та управління певною кількістю дронів в реальному часі, що робить його ідеальним для комерційних та індустріальних застосувань. Цей інструмент надає користувачам можливість відслідковувати місцезнаходження дронів, планувати та координувати місії, а також отримувати доступ до даних, зібраних дронами, з єдиного інтерфейсу.

Основними особливостями DJI FlightHub є можливість моніторингу в реальному часі, що дозволяє відслідковувати місцезнаходження всіх дронів на інтерактивній карті. Це дає змогу операторам ефективно координувати свої дії та забезпечувати безпеку польотів. Інша важлива функція – планування та координація місій, що дозволяє задавати маршрути польотів та встановлювати інші параметри місій.

Окрім цього, DJI FlightHub забезпечує дистанційний доступ до даних, зібраних дронами, з будь-якого місця, де є інтернет-з’єднання, за рахунок того, що дана система є хмарною, а функція живого відео стрімінгу дозволяє операторам бачити все, що бачать дрони, в реальному часі, що є критично важливим за певних ситуацій.

Історія польотів, яка зберігається на платформі, дозволяє аналізувати дані та покращувати ефективність майбутніх місій. Завдяки можливості централізованого управління всіма дронами з одного інтерфейсу, процеси управління значно спрощуються, особливо для великих роїв.

Однак, DJI FlightHub хоча і має безкоштовну версію, проте, вона дуже обмежена і після виконання п'яти польотів, вимагає придбання професійної версії, 12-річний план користування якою коштує, на даний момент, 999$. Також, FlightHub розроблений переважно для роботи з дронами DJI, що обмежує, в певній мірі, його використання з дронами інших виробників. DJI FlightHub не має вбудованих інструментів для розпізнавання об'єктів, проте це завдання може бути виконане за допомогою додаткових програмних рішень або модулів, що інтегруються з системою, що, скоріш за все, буде мати достатньо високу вартість та займе певний час, що є не рентабельним.

1.2.3 AirMap

AirMap є досить популярним застосунком для безпечного управління польотами дронів, розроблений компанією AirMap, Inc. Даний застосунок надає користувачам інтерактивну карту, яка показує обмежені зони для польотів, аеропорти, контрольовані повітряні простори та інші потенційні перешкоди в реальному часі. Це дозволяє операторам дронів планувати свої польоти з урахуванням актуальних умов і забезпечувати безпеку польотів [6].

Також однією з доступних функцій є планування маршрутів польотів, враховуючи всі можливі обмеження та ризики. Завдяки цьому функціоналу оператори дронів можуть заздалегідь розробити ефективний та безпечний план польоту.

Основними перевагами AirMap є його широке охоплення та забезпечення високого рівня безпеки польотів. Інтерактивна карта з відображенням обмежень та перешкод сприяє безпечному плануванню та виконанню польотів.

Проте, додаток має і деякі недоліки. Він не підтримує розпізнавання об'єктів або машинне навчання для автоматизації завдань, що може бути необхідно для більш складних операцій. Також AirMap не призначений для роботи з роями дронів, що обмежує його використання.

AirMap є потужним інструментом для управління польотами дронів, забезпечуючи високий рівень безпеки та ефективності. Однак, для більш специфічних завдань, таких як робота роїв або розпізнавання об'єктів, можуть знадобитися додаткові або спеціалізовані рішення, побудова яких вимагає великих часов втрат, так як інтеграція нових модулів з системою може бути складним, тривалим та вартісним процесом.

1.2.4 Аналіз аналогів: висновок

Огляд аналогів програмних рішень показав, що існують достатньо потужні інструменти для керування дроном або ж ройовоим комплексом, проте обрати серед них підходящий продукт є складною задачею.

Нажаль, запропоновані приклади аналогів надають хоча й потужне, але дуже обмежене, у рамках даної роботи, рішення задачі хмарної підтримки ройового комплексу.

Загальні характеристики розглянутих систем підтримки дронів або ж ройових комплексів наведено у таблиці 1.1. Відповідність поставлених задач до програмних рішень, що надають можливість їх вирішення, вказано у таблиці 1.2.

Так як існуючі розробки не задовольняють потреби ройових комплексів та не надають функціоналу, який забезпечить можливість розпізнавання об’єктів, маємо за необхідне реалізувати хмарну систему підтримки ройових комплексів та інтегрувати її у запропоновану у [2] архітектуру.

1.3 Постановка задачі

Предметною областю даної роботи є ройовий комплекс, який виконує певний пакет задач, в які входить аналіз території, пошук та розпізнавання об’єктів, моніторинг місцевості.

Таблиця 1.1 – Загальні характеристики аналогів

Характеристика		DroneAssist	DJI FlighHub	AirMap	Власний продукт
Загальні характеристики					
1	Частота оновлення платформи	Рідко	Часто	Рідко	В залежності від потреб
2	Сучасність інтерфейсу	Сучасний інтерфейс основною складовою якого є карта	Сучасний інтерфейс з можливістю перегляду 2.5 D карти	Сучасний інтерфейс основною складовою якого є карта	Сучасний інтерфейс
3	Ціна ($)	Безкоштовно	999 $	Невідомо	Безкоштовно
4	Можливість інтеграції	Глибока інтеграція з усіма дронами та контролерами DJI.	Надає API, для отримування доступ до даних про обмеження польотів та інтегрувати цю інформацію у свої системи управління польотами.	Надає високий рівень інтеграції з іншими системами	Надає можливості інтеграції нових модулів для розширення функціоналу
5	Гнучкість функціоналу	Надається повний набір функцій	Надається повний набір функцій за підпискою	Надається повний набір функцій	Надається повний набір функцій

Таблиця 1.2 – Вирішення поставленої проблеми

Характеристика		DroneAssist	DJI FlighHub	AirMap	Власний продукт
Поставлені задачі					
1	Підтримка ройових комплексів	Ні	Так	Ні	Так
2	Можливість конфігурації нодів рою	Ні	Ні	Ні	Так
3	Надання можливостей для розпізнавання об’єктів	Ні	Ні	Ні	Так
4	Побудова критичного шляху	Так	Так	Так	Так
5	Відслідковування шляху рою	Так	Так	Так	Так
6	Надання інформації про розпізнані об’єкти	Ні	Ні	Ні	Так

При аналізі предметної області сформульовано основні завдання, які має вирішувати дана інформаційна система:

1) конфігурація рою та окремих нодів: вибір дронів, що будуть брати участь у виконанні завдання, вибір сенсорів для кожної ноди та підбір методів;

2) керування польотом: визначення контрольних точок, зміна розташування окремих нодів;

3) аналіз даних отриманих від нодів рою: розпізнані об’єкти, мультимедіа файли, розпізнані об’єкти;

4) аналіз історії польотів, тобто аналіз роботи кожної ноди за певної конфігурації;

5) створення інтерактивного інтерфейсу для взаємодії користувача із системою;

6) створення API для моделювання процесу виконання завдання роєм.

У результаті аналізу та детального огляду поставлених задач, виявлено 4 основні групи користувачів:

1) оператор – має право на перегляд усієї необхідної для створення завдання інформації: дрони, сенсори, методи. Також переглядає та редагує, за потреби, контрольні точки польоту, переглядає рух рою та може змінювати їх шлях;

2) дрони – мають право отримувати інформацію про методи та сенсори, якими вони володіють, а також надсилати у відповідь інформацію про об’єкти, територію та власне положення;

3) аі-спеціаліст – має право на внесення нових методів у систему;

4) адміністратор ІС – має право на додавання нових користувачів та надання їм прав, згідно з їх ролю, а також на виконання всіх необхідних операцій, щоб надати оператору дані для виконання його роботи.

Усі задачі користувачів перелічено у додатку А.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.4 Архітектура застосунку

Для реалізації проекту використано трирівневу архітектуру, що відображено на рисунку 2.1. Трирівнева архітектура складається з трьох основних блоків, що взаємодіють між собою. Ці блоки включають в себе хмару, застосунок та рій.

Хмара використовується для централізованого зберігання даних (наприклад, алгоритмів для розпізнавання об’єктів) у базі даних, що генеруються іншими рівнями. Також на даному рівні виконуються операції над даними, такі як агрегація даних, фільтрація та інше. Хмара забезпечує віддалене зберігання даних.

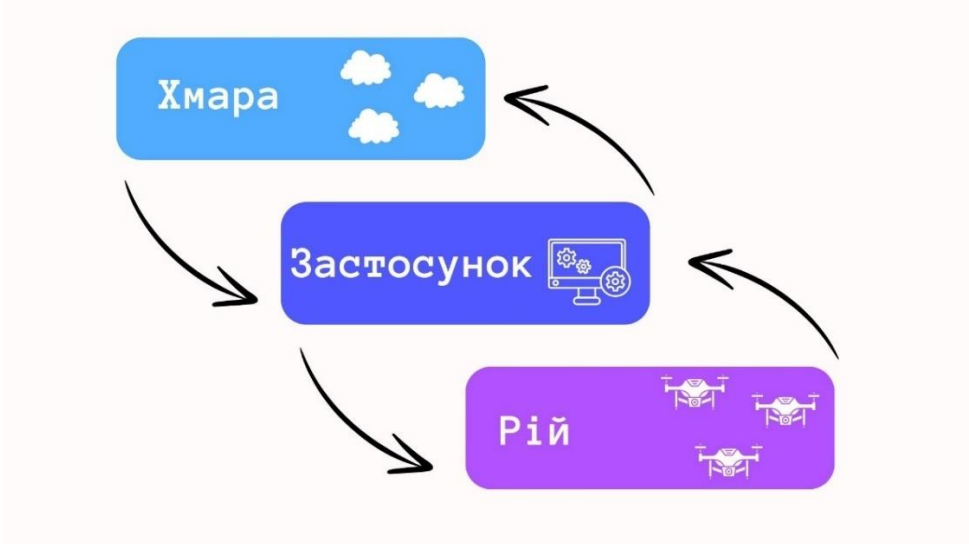


Рисунок 2.1 – Архітектура системи підтримки ройових комплексів

Застосунок виконує кілька ключових функцій, включаючи обробку даних, певний аналіз інформації, що надходить з хмари або ж рою, а також забезпечення користувацького інтерфейсу для взаємодії користувачів із системою через десктопний застосунок. Однією з важливих функцій є

спілкування оператора із системою: застосунок дозволяє оператору, який є частиною рою, вводити команди, контролювати роботу нодів рою та отримувати від них зворотний зв'язок. Крім того, застосунок забезпечує контроль і моніторинг роботи системи в режимі реального часу, що є важливим для операторів та інших користувачів.

Рій забезпечує виконання поставлених задач. У даний рівень входять оператор та, відповідно, ноди рою. Ноди є одиницями, які відповідають за сканування місцевості та розпізнавання об’єктів. Оператор ставить завдання, контролює процес виконання та вносить зміни у конфігурацію рою.

Для організації взаємодії між компонентами на рівні застосунку використано паттерн MVVM (рис. 2.2). Даний архітектурний шаблон призначено для того, аби розділити бізнес-логіку додатку (model), від представлення (view) та логіку взаємодії з користувачем (ViewModel), що, у свою чергу, спрощує тестування та розширення, а також забезпечує ефективну взаємодію між різними компонентами програми.

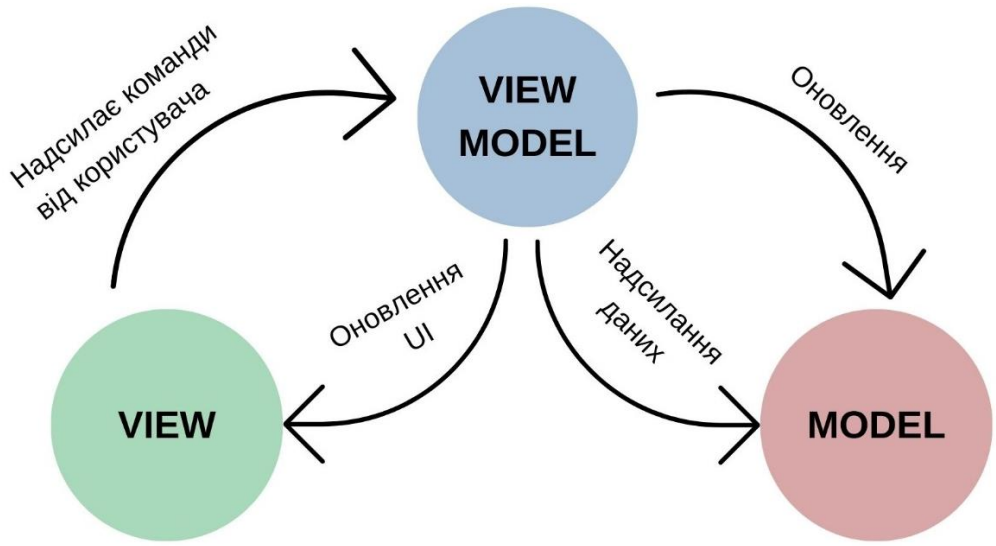


Рисунок 2.2 – Паттерн MVVM

Отже, використання трирівневої архітектури та паттерна MVVM дозволило створити достатньо структурований застосунок з можливістю

легкої підтримки. Розділення бізнес-логіки й інтерфейсу користувача сприяє підтримці та розширенню кодової бази застосунка, оскільки це дозволяє змінювати або додавати новий функціонал, не впливаючи на інші компоненти. Хмарні сервіси дозволяють легко

масштабувати обчислювальні ресурси відповідно до навантаження, що забезпечує високу продуктивність системи. Також використання хмарних сервісів забезпечує високу доступність та надійність завдяки резервуванню даних та автоматичному оновленню після збоїв.

Використання трирівневої архітектури, а паттерн MVVM допомагає керувати даними та їх відображенням на клієнтському рівні. Такий підхід забезпечує гнучкість, простоту розробки та ефективну підтримку застосунку, що є важливими аспектами у роботі з ройовими комплексами.

1.5 Інформаційне моделювання предметної області

Сутності та їх атрибути з описом обмежень, що потрібні для розв’язання поставлених задач, наведено у додатку Б.

Для реалізації задач користувачів інформаційної системи для підтримки ройового комплексу, є необхідним створити певний пакет сутностей, що забезпечуватимуть оператора всією необхідною інформацією для налаштування завдань, відслідковування та аналізу роботи ройових комплексів.

У базі даних хмарної інформаційної системи підтримки ройового комплекса існують 3 види зв’язку між сутностями, а саме:

- один-до-одного умовний;
- один-до-багатьох;
- багато-до-багатьох.

Нижче наведено опис сутностей, що вони відображають та їхні зв’язки.

Група таблиць, які відповідають за зберігання інформації про методи та їх версії:

1) таблиця «method_identification» (методи ідентифікації) призначена для збереження інформації стосовно методів ідентифікації;

2) таблиця «method_version» (версія методу) призначена для зберігання версій методів, кожна версія має зв’язок з методом ідентифікації (one-to-many), також, кожна версія має зв’язок з користувачем (one-to-many), який його додав;

3) таблиця «method_permission» (дозволи методу) призначена для зберігання інформації про критерії використання методів, кожен метод може мати певну кількість параметрів (one-to-many) з мінімальними для використання значеннями.

Група таблиць, які відповідають за зберігання інформації про мапи та їх версії:

1) таблиця «map» (мапа) відповідає за збереження загальної інформації про карту;

2) таблиця «map_version» (версія мапи) призначена для збереження версій мапи, що додано, кожна версія має зв’язок з мапою (one-to-many) й кожна версія має зв’язок з користувачем (one-to-many), який його додав.

Група таблиць, що відповідають за створення та підтримку завдань або ж польотів:

1) таблиця «flight_history» (історія польотів) призначена для збереження усіх польотів, що вже звершено та тих, які знаходяться на етапі виконання;

2) таблиця «drone_on_the_flight» (дрони на завданні) містить у собі інформацію про дрони, що приймають участь у конкретному завданні, кожен дрон з довідника пов’язаний з дроном на завданні (one-to-many), кожен політ пов’язаний з певною кількістю дронів на завданні, але конкретний дрон на завданні належить лише одному польоту (one-to-many);

3) таблиця «flight_points» (точки польоту) зберігає дані про контрольні точки польоту, дана сутність посилається сама на себе (one-to-one) для побудови зв’язаного списку точок;

4) таблиця «drone_with_sensors» (дрон із сенсорами) зберігає інформацію про те, які сенсори має дрон у певному завданні, кожен запис у цій таблиці є сполучною ланкою між дронами на завданні та сенсорами, що він використовує (many-to-many);

5) таблиця «drone_with_method» (дрон з методом) зберігає інформацію про те, які методи містить у собі дрон для виконання конкретного завдання, дана таблиця пов’язує дрон на завданні з методами (many-to-many), що можуть бути використані нодою під час виконання завдання;

6) таблиця «drone_location» (місцезнаходження дрону) зберігає інформацію про місцезнаходження конкретного дрону на завданні у певний момент часу, місцезнаходження дрону пов’язане з дронами на завданні (one-to-many) та зі списком точок (one-to-many);

7) таблиця «exclusion_drone» (дрон для виключення) поєднує між собою дрон на завданні з причинами для виключення за допомогою зовнішніх ключів (many-to-many);

8) таблиця «identified_object» (розпізнані об’єкти) містить у собі тип розпізнаного об’єкта та час, коли це сталося, тип розпізнаного об’єкта пов’язаний з таблицею типи об’єктів (one-to-many);

9) таблиця «recognition_file» є допоміжною для зв’язування таблиць розпізнані об’єкти та мультимедіа файлів (many-to-many) шляхом використання зовнішніх ключів;

10) таблиця «multimedia file» (мультимедіа файл) призначена для зберігання повної інформації про файли, включно із посиланнями на них, які надіслано нодами під час виконання завдання, дана сутність пов’язана з користувачами (one-to-many);

11) таблиця «object_parameter» (параметр об’єкта) призначена для зберігання інформації про параметри об’єкту задля покращення результатів роботи рою, кожен рядок даної таблиці пов’язаний з історією польотів, типами об’єктів та параметрами (one-to-many (n = 3)).

Група таблиць, які є допоміжними й використовуються для забезпечення можливості нормальної роботи рою:

1) таблиця «user» (користувач) зберігає інформацію про користувачів системи, один користувач може мати лише один профіль;

2) таблиця «exclusion_reason» виступає у ролі довідника, у якому зберігається інформація про можливі причини для виключення;

3) таблиця «points» (точки) зберігає інформацію про точки на карті, а також точки від дронів, які додано в процесі виконання завдань або за інших причин; для побудови шляху, кожна точка, окрім початкової, має посилатися на попередню точку польоту (one – to – one – or - none);

4) таблиця «drone» (дрон) зберігає коротку інформацію про дрони, які можуть бути задіяні оператором для виконання завдання; дана сутність пов’язана з типами дронів (one-to-many);

5) таблиця «type_of_drone» виступає у ролі довідника, який забезпечує користувачів інформацією про можливі типи дронів;

6) таблиця sensor (сенсор) містить в собі коротку інформацію про сенсори, що є придатними для використання оператором при налаштуванні рою. Кожен сенсор містить інформацію про власний тип, тож дана сутність пов’язана із сутністю тип сенсорів (one-to-many);

7) таблиця «type_of_sensors» (тип сенсорів) представляє собою довідник, у якому зберігається інформація про можливі (існуючі) типи сенсорів;

8) таблиця «parameters» (параметри) є довідником, у якому зберігається інформація про параметри, що стосуються дронів, сенсорів й обмежень методів;

9) таблиця «sensor_parameter» (параметр сенсору) є мостом для зв’язування сенсору з параметром (many-to-many), а також для надання певного значення відповідному параметру;

10) таблиця «drone_parameter» (параметр дрону) є допоміжною таблицею для зв’язування дрона із параметром, а також для надання певного значення відповідному параметру;

11) таблиця «type_of_objects» (типи об’єктів) постає у вигляді довідника з можливими для розпізнавання об’єктами, якими можуть виступати: коробка, людина, тварина, пожежа і таке інше.

Група таблиць, що відповідають за забезпечення роботи платформи Debezium:

1) таблиця «debezium_offset_storage» використовується для зберігання зсувів, які вказують на останнє оброблене повідомлення з БД, зсуви дозволяють Debezium відновити роботу з місця на якому він зупинився у разі перезапуску або збою;

2) таблиця «debezium_database_history» використовується для зберігання історії змін схеми БД, Debezium використовує цю інформацію для відстеження змін у структурі таблиць, таких як додавання або видалення стовпців, що дозволяє коректно обробляти дані навіть при зміні схеми.

ER-діаграма, отримана у результаті формалізації та нормалізації зв’язків хмарної ІС для підтримки ройового комплексу, наведена на рисунку 2.3. Для покращення сприйняття виділено окремі блоки. Для побудови діаграми використано програмний застосунок Navicat 17 за використання нотації «вороняча лапка».

1.6 Вибір програмного забезпечення для створення інформаційної системи

Хмарна інформаційна система підтримки ройового комплекса розроблена у форматі десктопного додатку, тобто користувач має можливість взаємодіяти із системою тільки після її успішного встановлення на пристрій.

Система реалізована із залученням технологій описаних далі.

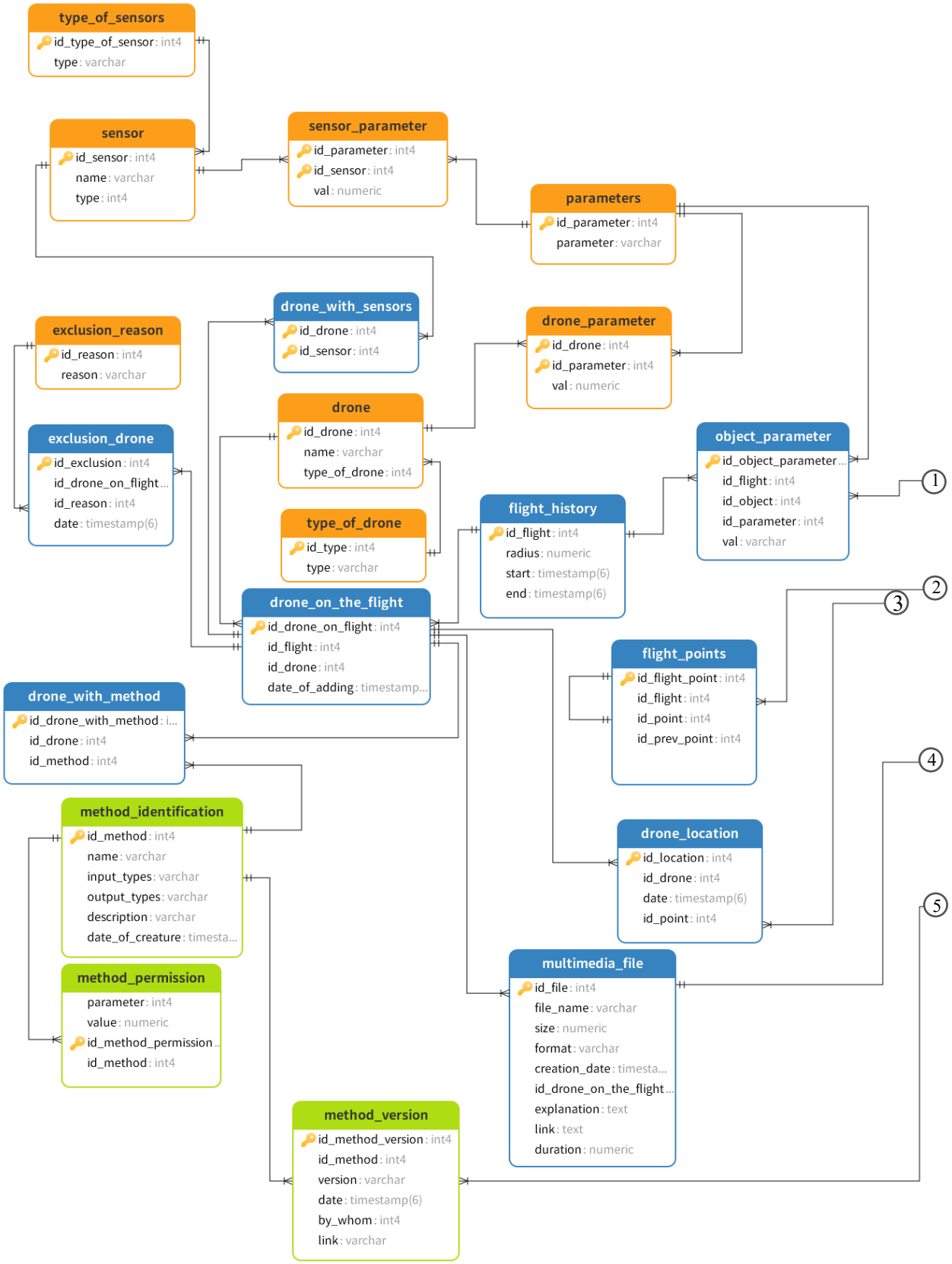


Рисунок 2.3 – ER-діаграма БД хмарної ІС підтримки ройового комплексу

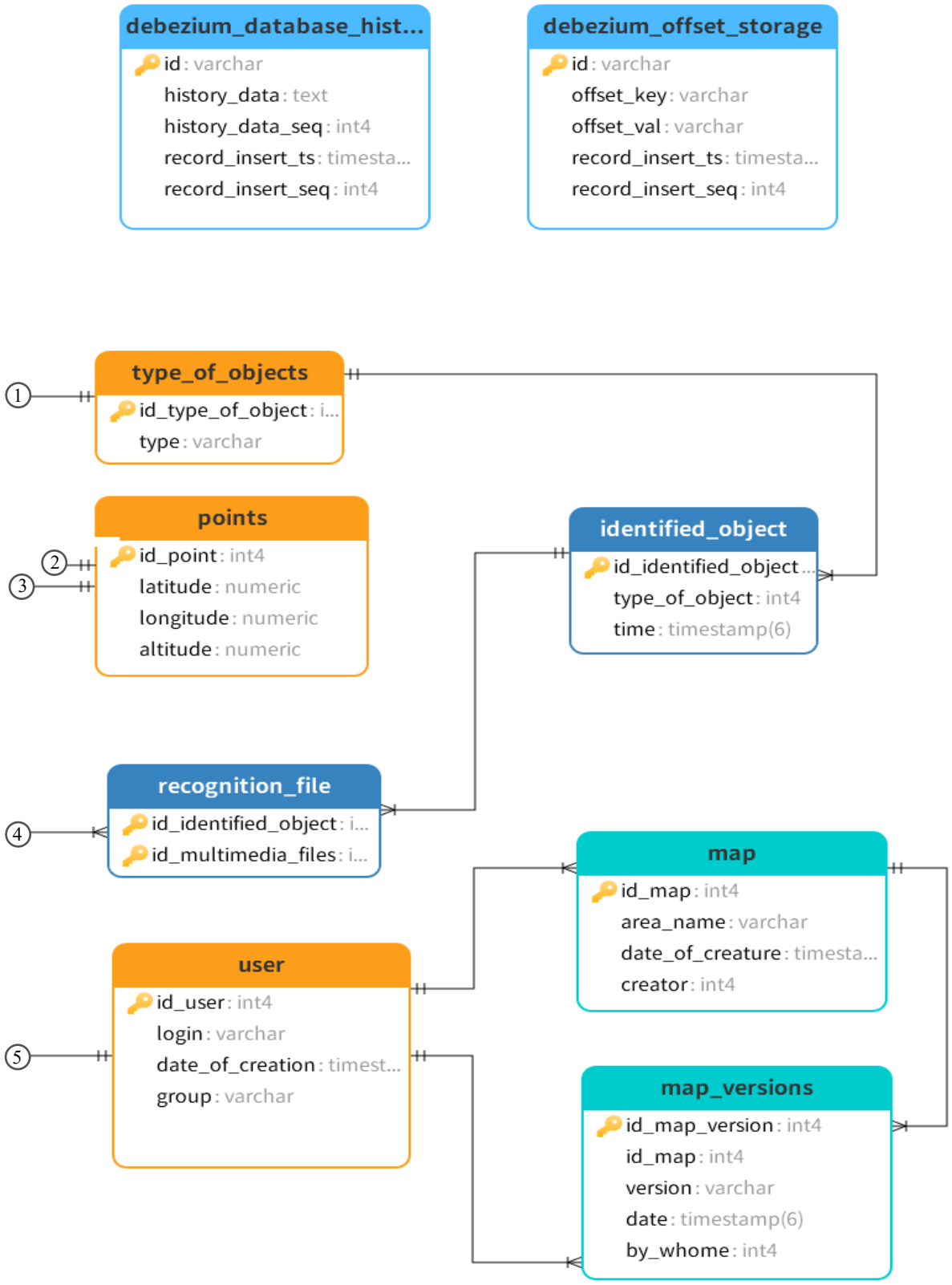


Рисунок 2.3, аркуш 2

Для взаємодії із базою даних обрано СУБД PostgreSQL v.16.2. Причини вибору даної СУБД наступні: система ролей та привілеїв дозволяє створювати різні рівні доступу для різних груп користувачів, що, у свою чергу, забезпечує надійність та безпеку даних.

Для реалізації Front-End обрано технологію WPF так як дана технологія використовує XAML для опису інтерфейсу користувача, що дозволяє чітко розділити презентаційний шар (UI) від логіки додатку. Важливим є й те, що вона підтримує MVVM, забезпечуючи кращу організацію коду та знижуючи рівень складності реалізації багатофункціонального застосунку. WPF забезпечує гнучкою системою змін стилів та шаблонів, а також надає можливість зв’язування даних з елементами інтерфейсу.

Для реалізації Back-End використано технологію компанії Microsoft - .NET Framework. .NET Framework надає розробнику широкі можливості для створення функціонального застосунку. Дана технологія має розширену платформну підтримку, тому у вільному доступі можна знайти велику кількість документації. Також .NET Framework підтримує таку мову програмування, як C# й надає бібліотеки для більш продуктивного написання коду, що, у свою чергу, пришвидшує розробку.

Для взаємозв’язку з БД обрано технолігую Entity Framework Core. Дана технологія представляє собою об’єктно-реляційний засіб для відображення (ORM) даних. EF Core підтримує різні платформи (Windows, macOS, Linux) та різні СУБД (SQL Server, PostgreSQL, MySQL, SQLite), забезпечуючи гнучкість у виборі середовища. Технологія підтримує підхід "код-по-першому" (Code First), міграції баз даних, LINQ для зручного написання запитів, асинхронні операції, автоматичне відстеження змін (Change Tracking) та різні механізми завантаження пов'язаних даних. Це робить розробку з EF Core інтуїтивно зрозумілою, продуктивною і легкою для підтримки.

Google Cloud обрано як хмарну платформу для серверного рівня системи. Хмарні сервіси Google Cloud забезпечують високу масштабованість, надійність та безпеку, що є критично важливими для сучасних інформаційних

систем. Основні переваги використання Google Cloud включають: легка масштабованість обчислювальних ресурсів відповідно до потреб системи, що забезпечує високу продуктивність за будь-яких навантажень; наявність резервування та автоматичного відновлення даних, що гарантує високу доступність та стійкість до збоїв, а це є критичним для поставленої задачі.

Для розробки додатку обрано середу інтегрованої розробки Visual Studio 2022 завдяки її численним перевагам, які роблять процес розробки ефективнішим і зручнішим. Visual Studio 2022 пропонує потужні інструменти для кодування, налагодження та тестування, що значно підвищує продуктивність розробників. Вона забезпечує відмінну інтеграцію з платформою .NET і технологіями, такими як Entity Framework Core, що дозволяє легко створювати та управляти проєктами, які використовують сучасні підходи до роботи з базами даних.

2.4 Програмна модель застосунку

Модель взаємодії бізнес-логіки у додатку реалізована через різноманітні ViewModel, які виконують роль посередників між моделлю даних та представленням (UI). ViewModel отримують запити від інтерфейсу користувача, передають дані у модель для обробки і повертають результати у вигляді оновленого інтерфейсу. Архітектура додатку складається з двох основних шарів: сервісів та репозиторіїв.

Сервіси виступають як безпосередні моделі, обробляючи дані, які надходять від ViewModel. Вони підготовлюють необхідну інформацію і, у разі потреби доступу до бази даних, надсилають запити до репозиторіїв. Репозиторії, в свою чергу, відповідають за взаємодію з базою даних через контекст даних. Вони займаються підключенням до бази, виконанням запитів та виконанням усіх необхідних операцій з даними. Варто зазначити, що взаємодія з сервісами та репозиторіями відбувається через інтерфейси.

Така структура забезпечує чітке розділення обов'язків у додатку, підвищуючи його гнучкість, масштабованість та підтримуваність. ViewModel дозволяють легко керувати станом і поведінкою представлення, сервіси обробляють бізнес-логіку, а репозиторії забезпечують надійний і ефективний доступ до даних.

Взаємодія усіх компонентів на прикладі авторизації зображено на рисунку 2.4.

Рівень представлення являє собою певний простір сторінок та вікон. В залежності від того під якою роллю авторизованого користувача, він отримує специфічні для конкретної ролі або групи сторінок. Наприклад, оператор має доступ до вікон, що стосуються безпосередньо завдання: склад рою, параметри шуканих об'єктів, контрольні точки польоту та місцезнаходження рою, що не має бути доступним для користувачів з іншими ролями. В той самий час, оператор не може змінювати методи, не має доступу до коду та обирає методи лише за описом, що надається AI спеціалістом, що забезпечує безпеку їх даних.

Ієрархію форм десктопного застосунку наведено на рисунках 2.5 – 2.11.

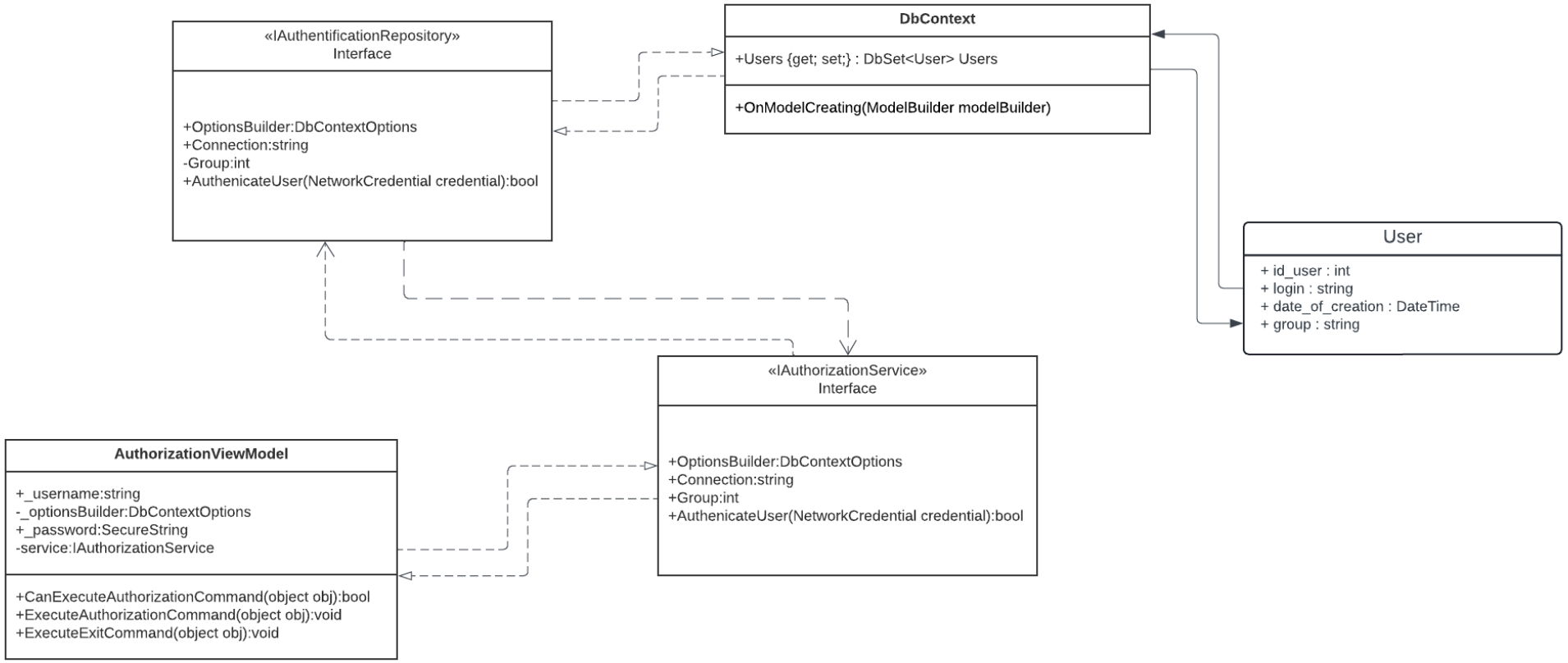


Рисунок 2.4 – UML-діаграма взаємодії класів

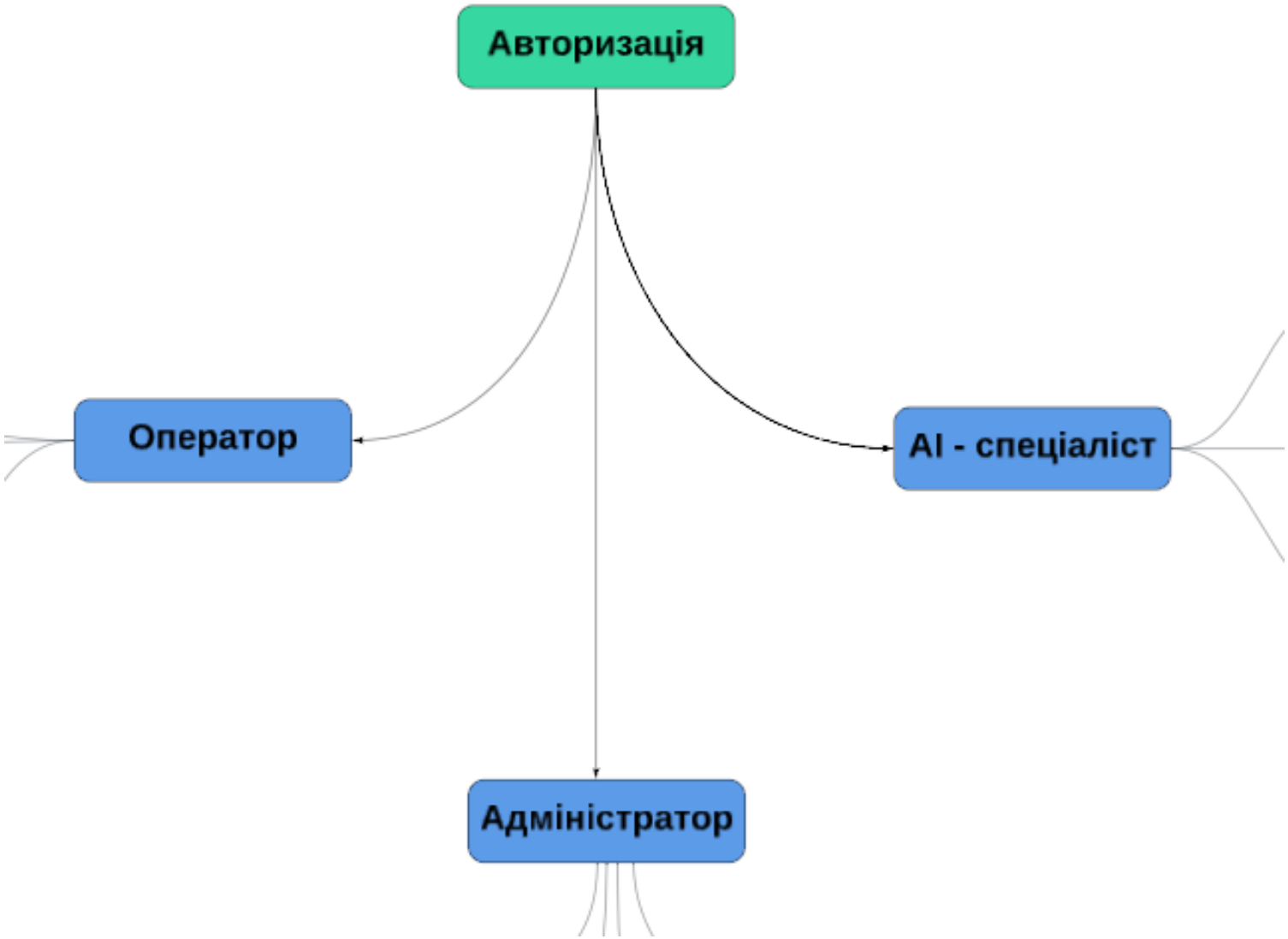


Рисунок 2.5 – Ролі доступу у застосунку

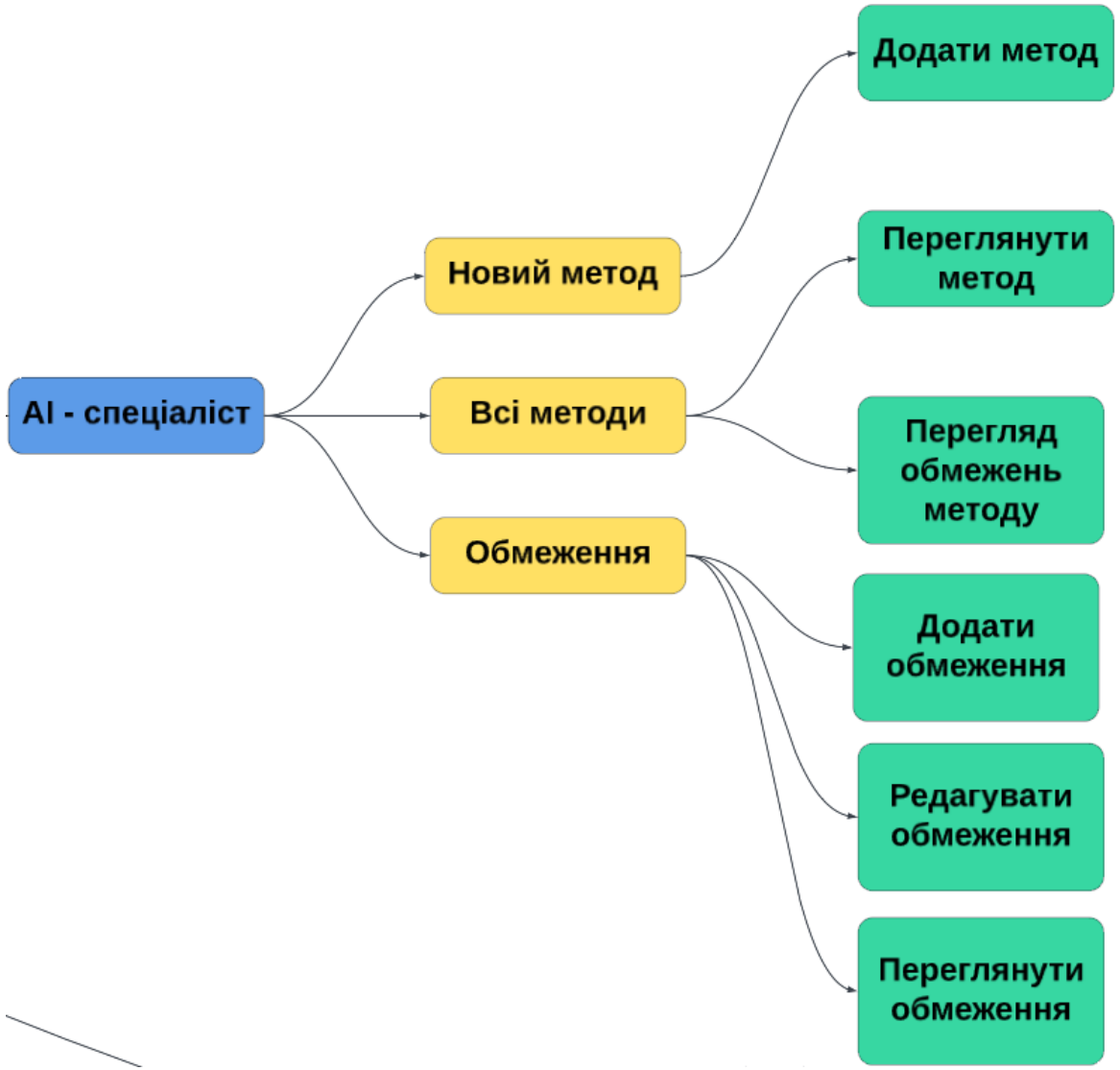


Рисунок 2.6 – Сторінки AI-спеціаліста

Рисунок 2.7 – Сторінки Адміністратора ч.1

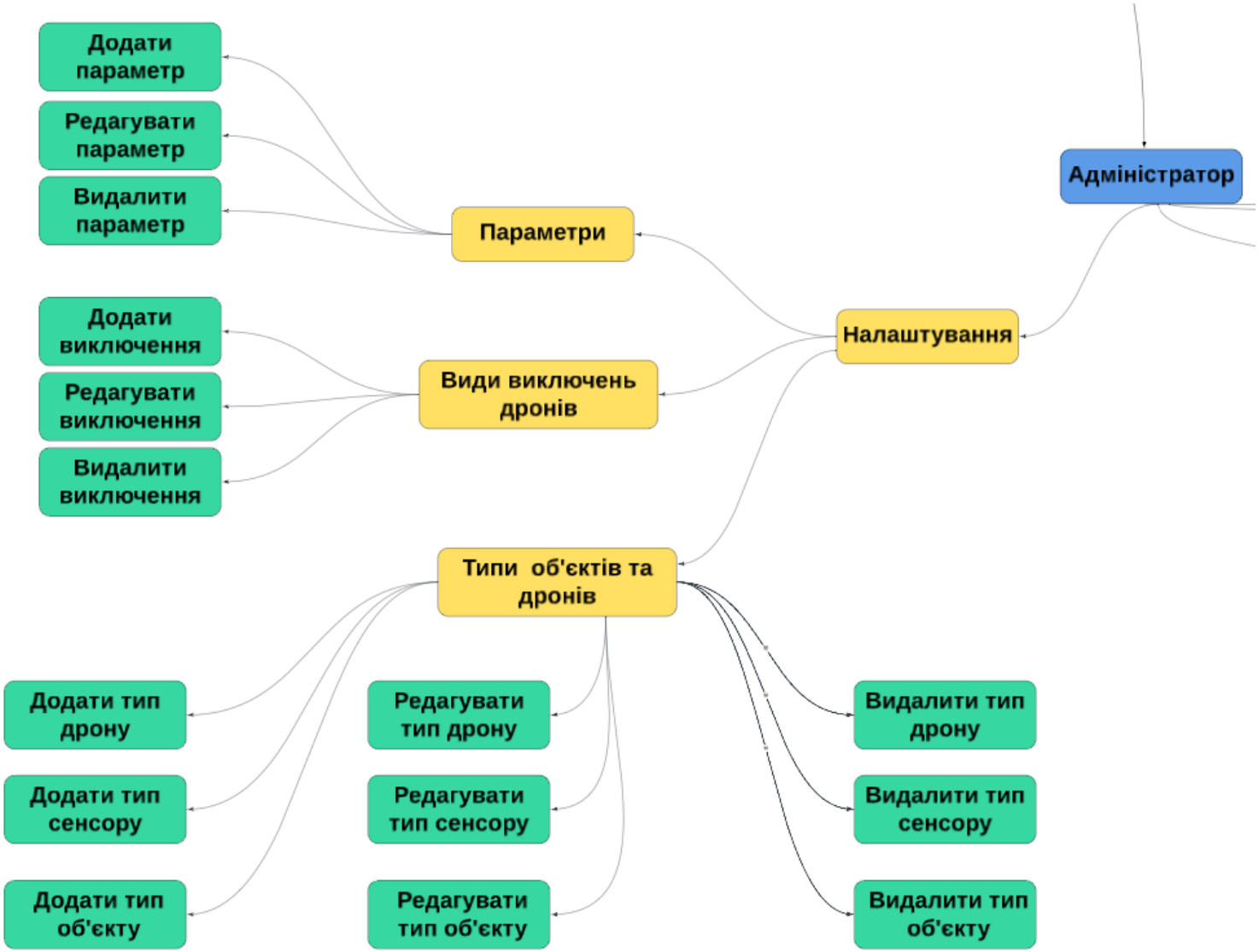


Рисунок 2.8 – Сторінки Адміністратора з налаштування

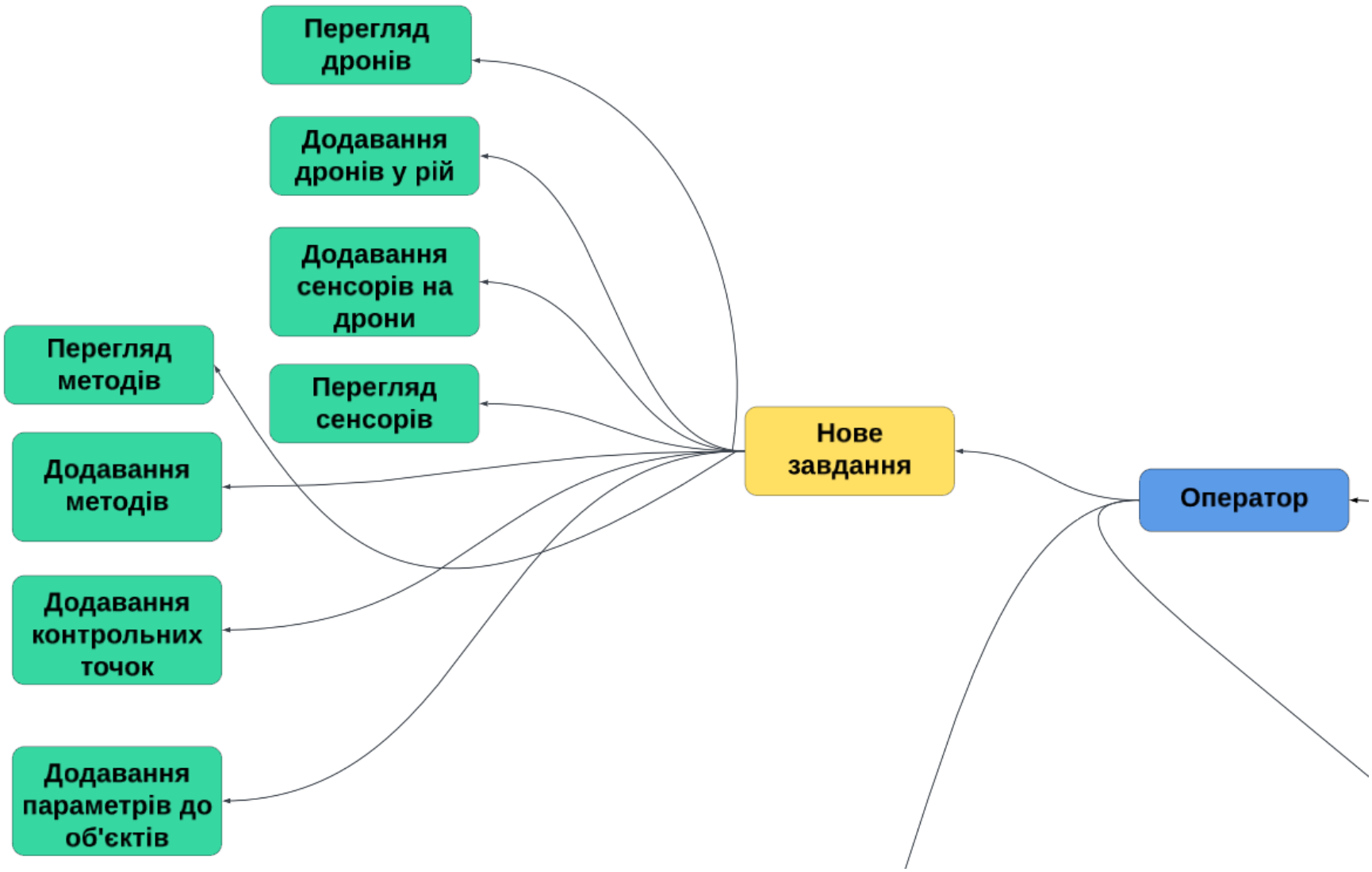


Рисунок 2.9 – Сторінки створення завдання. Сторінки Оператора

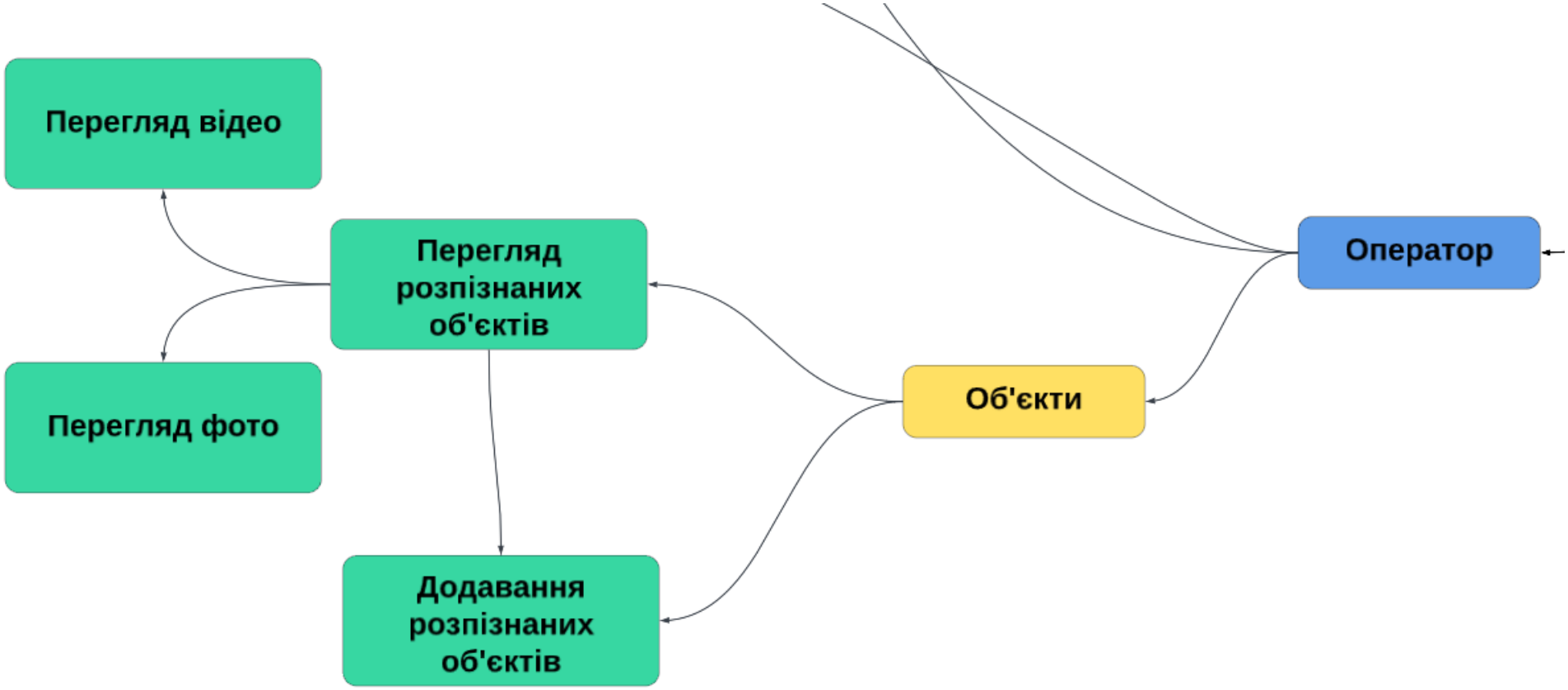


Рисунок 2.10 – Перегляд інформації по об’єктам. Сторінки Оператора

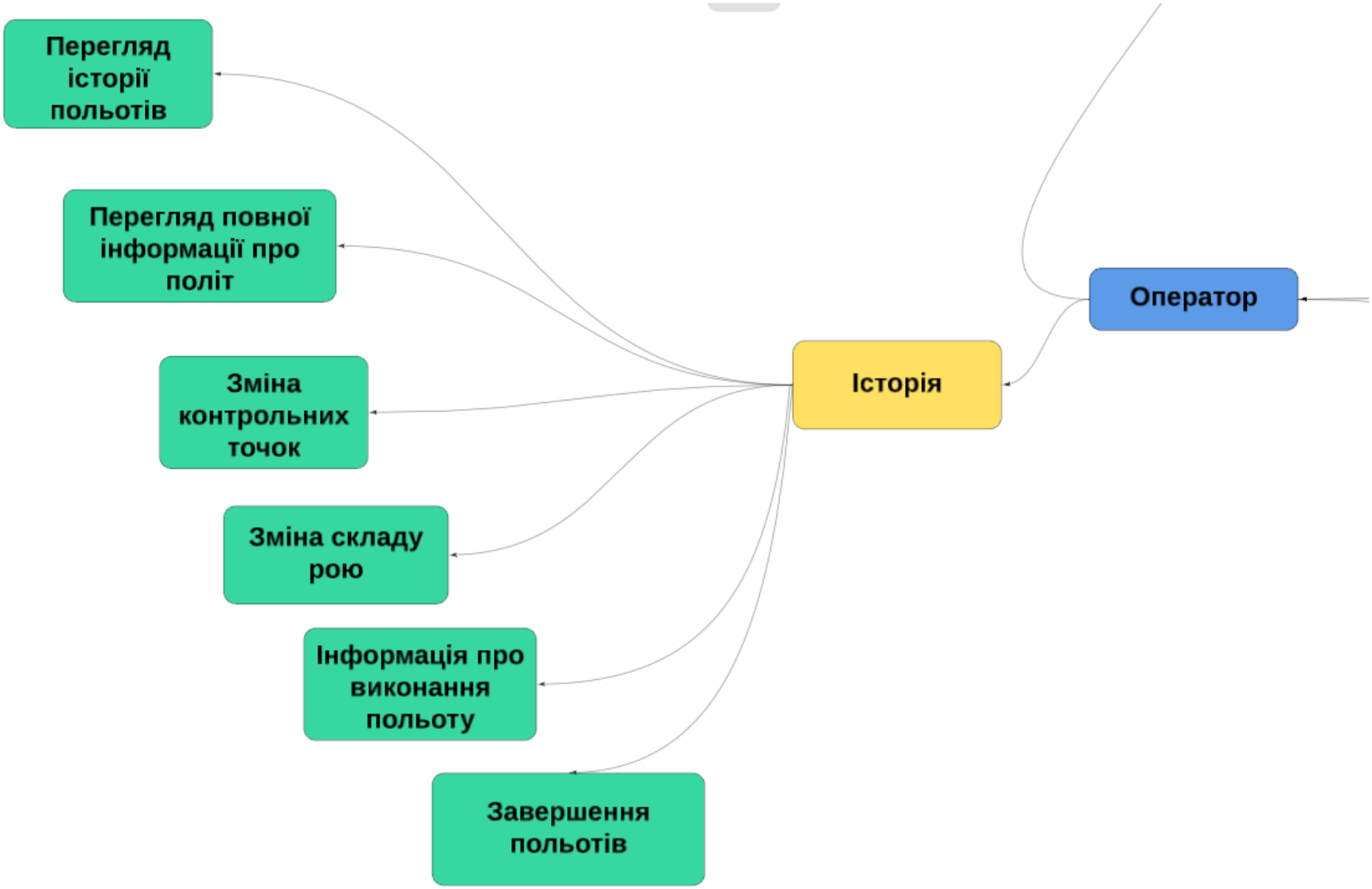


Рисунок 2.11 – Історія польотів. Сторінки Оператора

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розв’язання задач користувачів за допомогою SQL-запитів

Для розв’язання задач користувачів хмарної інформаційної системи підтримки ройових комплексів є необхідним певний набір SQL-запитів, функцій, тригерів та збережених процедур.

SQL-запити для створення таблиць інформаційної системи підтримки ройових комплексів надано у додатку В.

SQL-запити, що використовуються для створення допоміжних тригерів для забезпечення цілісності БД інформаційної системи надано у додатку Г.

Певна кількість SQL-запитів, що отримує база даних є сгенерованими ORM Entity Framework, яка надає можливість роботи з базою даних, використовуючи об’єктно орієнтований підхід у тандемі з LINQ.

Далі наведено приклад операції, що дозволяє отримати інформацію про дрони, які знаходяться на конкретному завданні. З використанням Entity Framework та мови запитів LINQ це можна зробити наступним чином:

```
var result = _context.Drones.Join( _context.DroneOnFlight,
             drone => drone.id_drone,
             dof => dof.id_drone, (drone, dof) => new { Drone =
drone, DroneOnFlight = dof }).Where(
joined => joined.DroneOnFlight.id_flight == flight)
                      .Select(joined => joined.Drone).ToList();
```

Лістинг 1.1 – Приклад запиту для отримання інформації про дрони

Вище можна побачити, що даний запит містить у собі звичні для SQL-запитів до БД оператори JOIN, WHERE, SELECT. Немає жодних обмежень у використанні даного фреймворку, тобто є можливість для виконання поєднання (UNION), виключення (EXCEPT), сумування (SUM) та інших.

Вирішення задач користувачів відбувається шляхом виконання наступних SQL-запитів:

1) для виконання задач А4, А8, А12, А20, А21, О1, О2, О4, О6, О16, О17, АІ1, АІ4, Д1 виконується запит з підстановкою `SELECT список_стовпців FROM таблиця [WHERE умова]` з підстановкою відповідних стовпців. Представлення (додаток Д) викликаються за допомогою запиту `SELECT список_стовпців FROM представлення [WHERE умова];`

2) задачі А1, А5, А9, А13, А15, А18, А23, А24, О8 - О12, О18, АІ2, АІ3, Д2 – Д5 вирішуються за допомогою виконання запиту типу `INSERT INTO таблиця(список_стовпців) VALUES (список_стовпців)` з підстановкою відповідних імен таблиць та списку стовпців;

3) задачі А2, А10, А16, А19, А22, А25, АІ5, АІ6 вирішуються за допомогою виконання запиту типу `UPDATE таблиця SET стовпець_1 = значення_1 […, стовпець_k = значення_k] [WHERE умова]` з підстановкою відповідних імен таблиць та необхідних стовпців;

4) задачі А3, А6, А11, А14, А17, О7 вирішуються за допомогою виконання запиту `DELETE FROM таблиця[WHERE умова]` з підстановкою відповідних імен таблиць.

Вищезазначені види SQL-запитів націлені на відображення загальних CRUD операцій. Певні запити є дуже простими, проте присутні й такі, процес обробки виявляється більш складним. Наприклад, при виконанні запиту для отримання переліку дронів, які, на даний момент, не призначені для жодного польоту застосовується операція вибірки з таблиці `drone` всіх рядків, крім тих, що зустрічаються в результаті об'єднання з таблицями `drone_on_the_flight` та `flight_history`, де час завершення польоту (`end`) у таблиці `flight_history` є нульовим або невизначеним (лістинг 1.2).

Процес виклику всіх SQL-запитів відбувається в межах репозиторіїв, які через контекст даних отримують доступ до відповідних ресурсів, або ж, інакше кажучи, таблиць. Дані операції виконуються безпосередньо на рівні бази даних, що дозволяє пришвидшити обробку даних.

```
SELECT * FROM drone EXCEPT SELECT d.* FROM drone d
JOIN drone_on_the_flight dof ON d.id_drone = dof.id_drone
JOIN flight_history fh ON dof.id_flight = fh.id_flight
WHERE fh."end" IS NULL;
```

Лістинг 1.2 – Приклад SQL-запиту

Для вирішення деяких задач використовуються тригери. Наприклад для реалізації задачі А17 є необхідним застосування тригеру (додаток Д), який при спробі видалення або ж списання сенсору спочатку перевірить чи не є він задіяним у завданні і тільки тоді дозволяє або ж не дозволяє виконати зазначену дію.

У системі присутні задачі, що вирішуються за допомогою функцій, для виклику яких виконується запит: `SELECT список_стовпців FROM назва_функції`. Функції у більшості своїй призначені для того, щоб виконати декілька дій для досягнення поставленої задачі, зменшивши кількість запитів до БД, що може значно підвищити ефективність роботи системи. (додаток Е)

Також, для вирішення певної кількості задач використовуються процедури (додаток Е). Так, наприклад, для оновлення даних користувача у системі (А2) є процедура, яка перевіряє коректність даних, які треба оновити, а потім вносить зміни у права доступу користувача за ситуації зміни ролі.

3.2 Створення та налаштування хмарного сховища

Для безперебійного доступу до даних є необхідним помістити її у хмарне сховище. Провайдером хмарних послуг, у даному випадку є Google Cloud Platform (GCP). Перейдемо до налаштування хмарного сховища [7].

3.2.1 Налаштування GCP

GCP надає зручний та інтуїтивно зрозумілий для роботи інтерфейс, що дозволяє швидко та без проблем виконати усі налаштування. І так, на рисунку

3.1 можна побачити конфігурацію хмарної бази PostgreSQL. Розглянемо детальніше деякі з параметрів.

Регіон (region) визначає фізичне розташування бази даних у дата центрах Google Cloud. До вибору регіону, треба підходити серйозно, бо близькість розташування визначає кількість затримок доступу до даних.

Віртуальні центральні процесори або ж vCPU визначають обчислювальну потужність БД. Кількість vCPU впливає на швидкість обробки запитів та загальну продуктивність системи. У межах даної роботи виділено 2-ядерний центральних процесор, що дозволяє одночасно обробляти більшу кількість запитів, та, відповідно, виконувати більше процесів в один момент. За потреби, кількість ядер може бути збільшена.

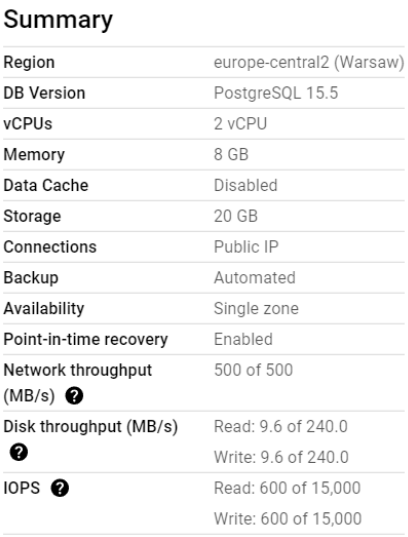
Оперативна пам’ять (RAM) у обсязі 8 GB також дозволяє обробляти достатні для даної роботи обсяги даних, повністю покриваючи потреби. У разі нестачі, кількість пам’яті може бути збільшена лише за допомогою декількох кліків на сервері.

Спосіб підключення «Public IP» дозволяє швидко та без особливих проблем підключитися до БД через Інтернет, не вимагаючи жодних додаткових налаштувань конфігурації.

Point-in-time recovery є однією з найважливіших функцій, яка, у разі збою, помилках або некоректних операціях з даними, дозволяє відновити базу даних з будь-якого визначеного часу у минулому.

Backup (бекап) – функція призначена для створення копіювання, у випадку даної роботи, автоматичного. Вона регулярно створює копії даних, що забезпечує відновлення у разі збоїв або втрати даних. За неможливості відновлення можна скористатися функцією вище.

Таким чином забезпечуються прийнятні для нормальної роботи інформаційної системи умови, що дозволяє працювати, не хвилюючись за можливі труднощі. У разі збоїв надані можливості бекапу й, для перестрахування, можливість відновлення будь-якої з минулих версії БД.



Summary

Region	europe-central2 (Warsaw)
DB Version	PostgreSQL 15.5
vCPUs	2 vCPU
Memory	8 GB
Data Cache	Disabled
Storage	20 GB
Connections	Public IP
Backup	Automated
Availability	Single zone
Point-in-time recovery	Enabled
Network throughput (MB/s)	500 of 500
Disk throughput (MB/s)	Read: 9.6 of 240.0 Write: 9.6 of 240.0
IOPS	Read: 600 of 15,000 Write: 600 of 15,000

Рисунок 3.1 – Конфігурація хмарної БД.

3.2.3 Логування та аналіз роботи системи

Google Cloud надає достатньо широкий функціонал для відслідковування роботи з хмарною БД (рис. 3.2). Логування у хмарному сховищі дозволяє відслідковувати активність користувачів та події у хмарній інфраструктурі й, у разі виникнення підозрілих дій одразу застосувати певні дій для уникнення проблем, які можуть виникнути у подальшому.

Також наявний інструмент, що надає можливості глибокого аналізу (рис. 3.3, 3.4), а також інформацію про стан, продуктивність та поведінку системних компонентів. Наприклад, аналіз ключових показників продуктивності (використання процесора, пам’яті, мережевої пропускної зданості, тощо) дозволяє оцінити якість конфігурації та за потреби прийняти рішення щодо внесення змін задля покращення. Аналіз роботи системи як і логування дозволяє оптимізувати роботу БД, підвищити продуктивність шляхом оптимізації використання ресурсів, швидко та якісно відслідковувати виникнення аномальних дій та вразливостей для швидкого усунення та можливість прогнозування майбутніх потреб у ресурсах у випадку масштабування системи.

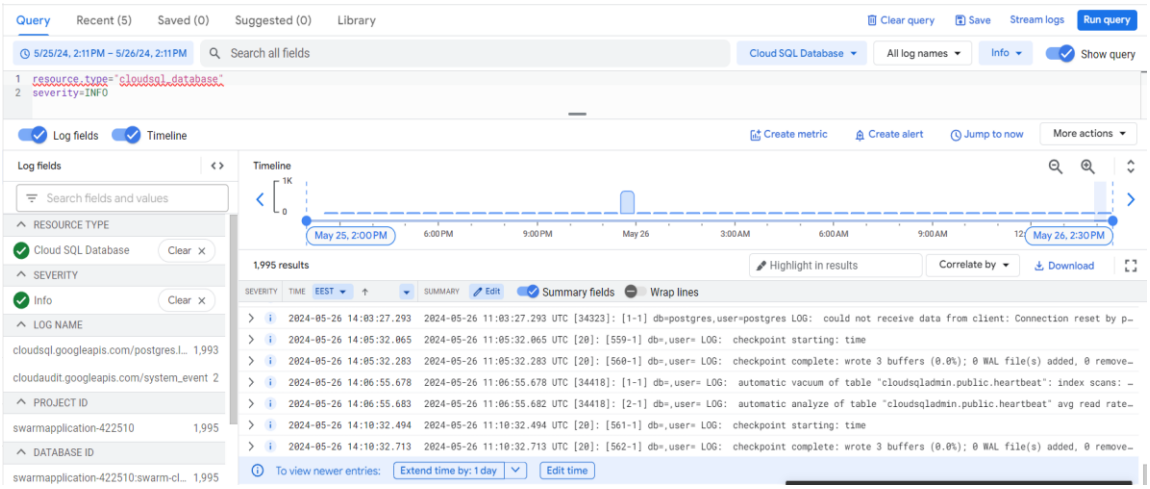


Рисунок 3.2 – Журнал активності та подій

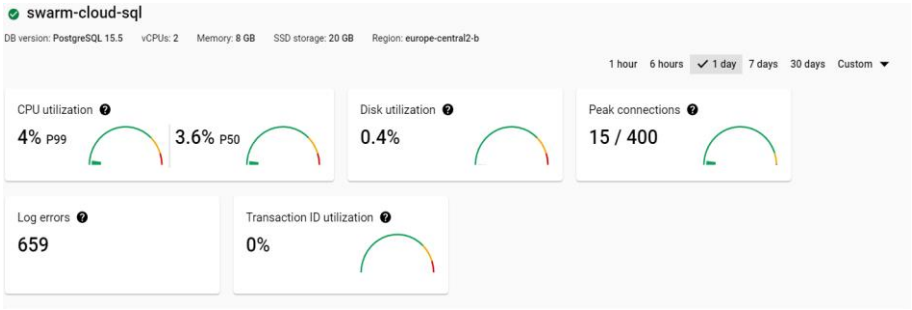


Рисунок 3.3 – Ключові показники продуктивності

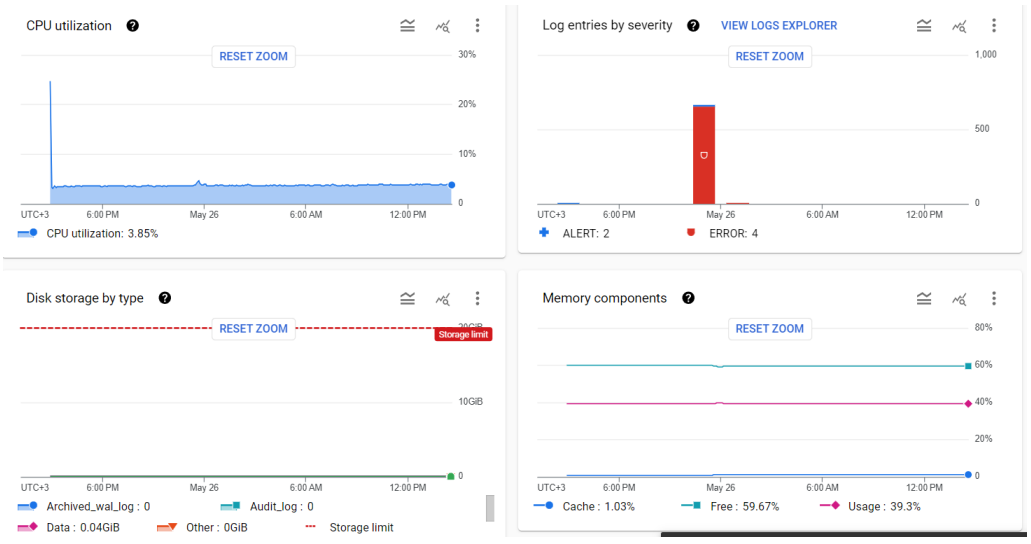


Рисунок 3.4 – Аналіз активності та продуктивності

3.3 Програмна реалізація інтерфейсу користувача

3.3.1 Реалізація фронтенд сторони застосунку

У WPF інтерфейс представляється у вигляді вікон та сторінок, з допомогою яких відбувається взаємодія користувача із системою. Для побудови інтерфейсу взаємодії використовується декларативний підхід, що базується на мові розмітки XAML та патерні MVVM.

Даний підхід полягає в тому, що розмітка або ж опис інтерфейсу користувача проводиться шляхом опису відображення компонентів і взаємозв'язків між ними без програмування кожної деталі. Тож такий підхід до створення інтерфейсу користувача дозволяє забезпечити простоту відображення складних інтерфейсів, покращити швидкість розробки й зменшити ймовірність появи помилок. Важливо відзначити, що декларативний підхід дозволяє швидко та без необхідності зміни програмного коду модифікувати інтерфейс.

Інтерфейс отримує дані від ViewModel з допомогою використання наступних механізмів:

1) Data Binding дозволяє автоматичну синхронізацію даних, що досить вагомо полегшує прив'язку властивостей ViewModel з візуальними елементами інтерфейсу користувача, наприклад, зміна положення дрону автоматично відображається у користувача шляхом зміни розміщення мітки;

2) Commands дозволяє зробити зв'язку між діями користувачами, такими як: натискання кнопок, вибір параметрів, тощо з методами у ViewModel й призводить до відокремлення логіки взаємодії від представлення;

3) INotifyPropertyChanged є інтерфейсом, який виконує роль сповіщувача й при будь-яких змінах зі сторони користувача або ж фронтенду надсилає повідомлення про внесені зміни прив'язаних з допомогою DataBinding даних.

У лістингу Ж.1 надано приклад, який представляє собою інтерфейс для авторизації користувача. Даний приклад надає простий й наочний опис того, як відбувається побудова інтерфейсу у WPF-застосунках. Використання XAML для опису візуальної частини інтерфейсу та MVVM для управління логікою дозволяє розділити представлення від бізнес-логіки, що полегшує розробку, тестування та підтримку програми.

3.3.2 Реалізація бекенд сторони застосунку

Моделі у MVVM десктопного WPF застосунку є комплексними елементами, які складаються з трьох зв’язаних між собою шарів: сервісів, репозиторіїв та моделей.

Моделі представляють собою ключовий компонент, що відповідає за управління даними та бізнес-логікою застосунку. Кожен клас моделі сутності представляє собою окрему таблицю або ж представлення у БД.

Моделі надають можливість зберігання структури даних. У моделі визначено, які дані необхідно зберігати, як вони взаємодіють між собою та які відносини існують між об’єктами. Також вони забезпечують валідацію даних, що забезпечує коректність та відповідність даних встановленим правилам та умовам. У лістингу Ж.2 наведено приклад моделі, яка представляє собою представлення для відображення інформації по методам.

Репозиторії відповідають за доступ до даних та їх зберігання. Вони забезпечують абстракцію над базою даних, дозволяючи використовувати методи для отримання, оновлення та видалення даних. Репозиторії виконують роль проміжного шару між моделлю та базою даних у патерні MVVM, слугуючи своєрідним мостом, який дозволяє працювати з даними без необхідності прямого втручання користувача у роботу бази даних. Приклад репозиторія наведено у лістингу Ж.3.

Сервіс є компонентом бізнес-логіки, який забезпечує взаємодію між репозиторієм та ViewModel. Даний елемент логіки виконує команди, які надходять від користувача через ViewModel з використанням репозиторіїв для

виконання власних функцій або ж методів. Сервіси можуть включати логіку для обробки складних операцій, інтеграції з іншими системами, виконання обчислень та іншої бізнес-логіки. Також сервіси можуть взаємодіяти із зовнішнім API для отримання необхідних даних або ж відправки інформації. У лістингу Ж.4 наведено частину сервісу, де відображено взаємодію з різними API для отримання інформації та її надсилання, а також взаємодію з репозиторіями з підготовкою даних до надсилання у ViewModel.

ViewModel виступають посередниками між моделлю та відображенням користувача. Основними функціями, що виконуються ViewModel є:

1) керування станом представлення. ViewModel зберігає стан інтерфейсу користувача, наприклад, обрані значення елементів, введені значення та інші параметри, що впливають на вигляд та поведінку інтерфейсу;

2) логіка взаємодії. ViewModel визначає, як інтерфейс користувача реагує на дії користувача, такі як натискання кнопок, вибір елементів у списку, введення тексту тощо. Реагування може включати в себе виклик методів з сервісу для обробки запиту або ж виконання навігації між сторінками та вікнами;

3) реалізація двостороннього зв’язку між даними та представленням. Інтерфейс INotifyPropertyChanged дозволяє автоматично оновлювати представлення при зміні даних у ViewModel.

У лістингу Ж.5 наведено приклад частини ViewModel, який надає можливість чіткого розуміння того, яка роль у застосунку відведена даному елементу.

3.4 Безпека інформаційної системи

3.4.1 Обмеження доступу на рівні застосунку

Після того, як користувач вводить свої облікові дані в вікно авторизації, ці дані передаються в ViewModel авторизації для подальшої обробки.

ViewModel перетворює логін та пароль на об'єкт NetworkCredentials, який потім передається у сервіс авторизації для подальшої обробки.

Сервіс авторизації отримує дані та виконує перевірку наявності користувача з такими обліковими даними у системі. Якщо користувач знайдений або виконано успішне підключення до БД, то визначається його роль. Роль користувача разом з контекстом даних передаються у репозиторій, який відповідає за обробку даних для цієї ролі.

Якщо користувача знайдено та підключення виконано успішно, то визначається роль користувача й разом з нею передається контекст даних. Саме цей контекст даних передається у репозиторій відповідного ролі користувача й використовується для надсилання даних у хмарну БД.

3.4.2 Обмеження доступу на рівні хмари

Забезпечення безпеки даних у хмарній БД з використанням хмарного провайдера GCP, відбувається шляхом обмеження доступу з визначенням діапазону IP-адрес, які визначено у якості довірених (рис. 3.5).

Також присутня опція авторизації з інших Google Cloud за допомогою внутрішньої автентифікації. В межах даної хмарної БД дана опція є деактивованою. Не дивлячись на те, що її виключення може викликати деякі труднощі, проте такий підхід підвищує безпеку, запобігаючи автоматичному доступу без явного дозволу.

Ще одним пунктом, для забезпечення доступу до БД є підключення з використанням шифрування SSL/TLS, що, у свою чергу, забезпечує захист даних при передачі.

3.4.3 Обмеження доступу на рівні бази даних

Безпека на рівні бази даних забезпечується шляхом створення облікових записів кожного користувача у вигляді ролей бази даних та видачі

їм власних привілеїв або певної рольової групи.

Так як користувачами системи є ролі у базі даних, то паролі при створенні кожної конкретної ролі автоматично хешуються алгоритмом `SHA-256` та зберігаються у таблиці системних каталогів `pg_authid`, доступ до якої мають лише суперкористувачі, тобто жоден користувач інформаційної системи не зможе потрапити до даної таблиці та переглянути хеші паролів.

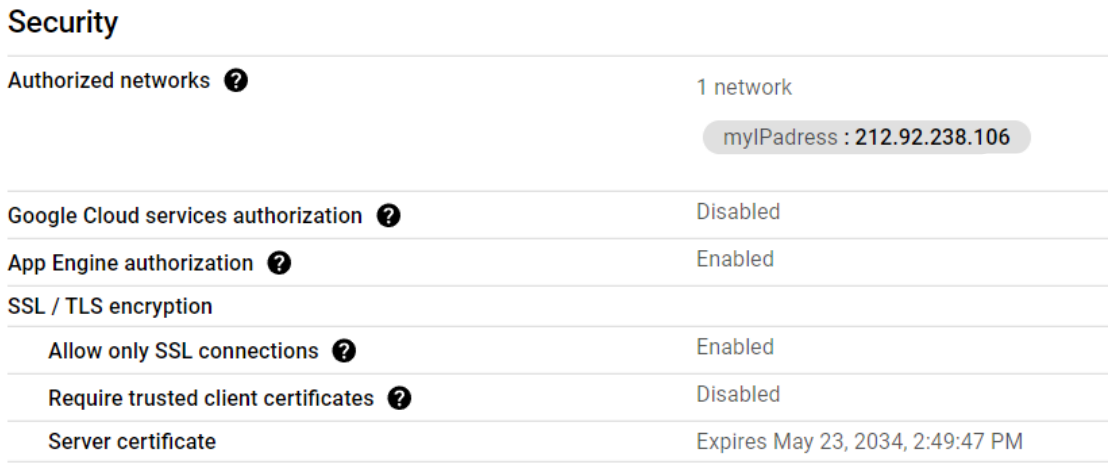


Рисунок 3.5 – Налаштування безпеки

Лише адміністратор має право додавати у систему нових користувачів та призначати їм певні ролі. Кожен новостворений користувач отримує свій власний обліковий запис у таблиці User й одразу ж отримує привілеї згідно з груповою роллю, яка надана йому адміністратором.

Список наявних у системі ролей користувачів:

1) адміністратор: admin;
2) оператор: operator;
3) AI спеціаліст: ai_specialist;
4) дрон: drone.

Для кожної ролі у БД встановлено привілеї на доступ до об’єктів бази даних. Ознайомитися зі списком доступу ролей до компонентів БД можна у додатку К.

3.5 Демонстрація функціонування ІС

При кожному запуску застосунку користувача, у першу чергу, зустрічає вікно авторизації (рис. 3.6). Після успішної авторизації користувачу відкривається вікно відповідно до його ролі у системі.

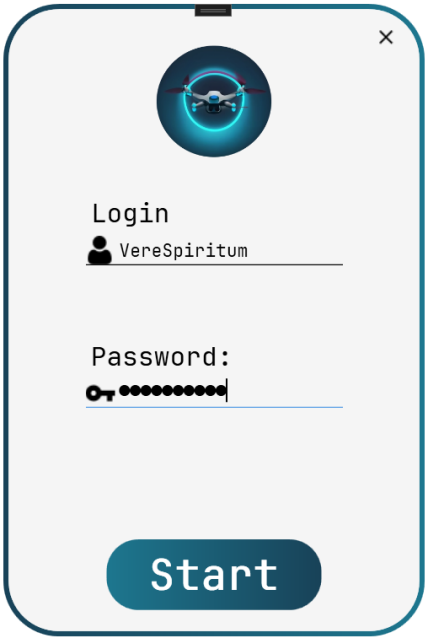


Рисунок 3.6 – Інтерфейс авторизації користувача

Після успішної авторизації користувач потрапляє на головну сторінку й може навігувати за допомогою пунктів меню, що знаходяться у верхній частині застосунку (рис. 3.7). Кожна групова роль користувачів суттєво відрізняється від іншої за завданнями, тож розглянемо інтерфейс кожної з ролей.

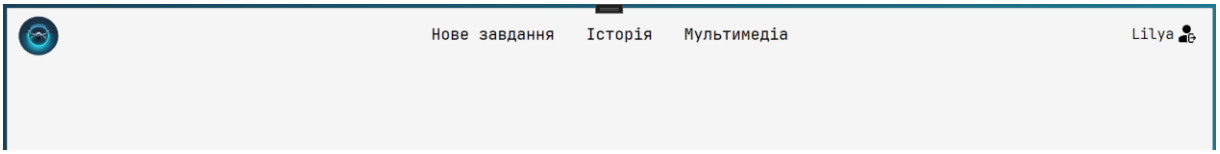


Рисунок 3.7 – Приклад навігації користувача

3.5.1 Інтерфейс адміністратора ІС

Адміністратор даної ІС забезпечує її нормальне функціонування, тобто йому надається доступ до інтерфейсу, користування яким, у свою чергу, забезпечує інших користувачів усією необхідною для роботи інформацією.

Адміністратор через навігаційну панель може перейти до перегляду списку користувачів, відповідно до їх ролей у системі. Також з цього інтерфейсу йому надається можливість перейти до створення та редагування користувачів через модальні вікна або ж їх видалення (рис. 3.8).

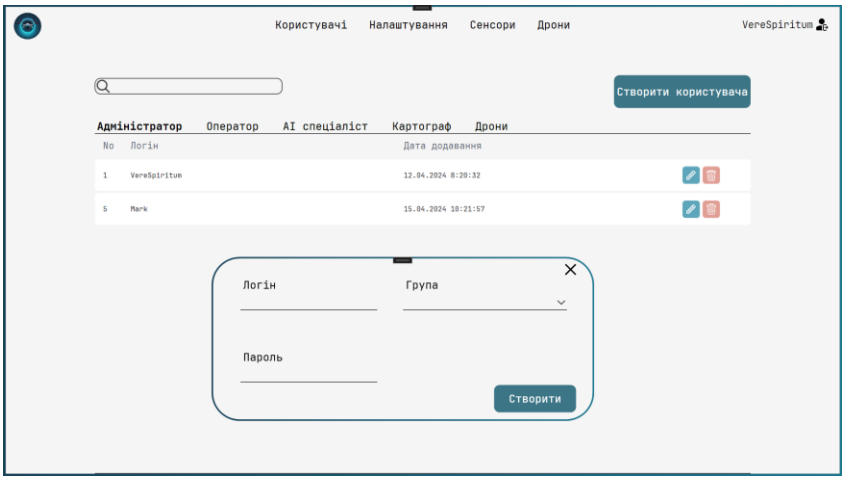


Рисунок 3.8 – Список користувачів системи

Наступним пунктом у меню є «Налаштування». Даний пункт містить у собі три підпункти, кожен з яких відповідає за певні базові налаштування, які є необхідними для інших користувачів (рис. 3.9).

Підпункт навігаційної панелі «Параметри» дозволяє користувачу переглянути або ж виконати певні дії з параметрами, які можуть бути застосовані для визначення можливостей дронів, сенсорів та об’єктів, а також для визначення параметричних обмежень методів для їх надання у дію конкретному методу. На рисунку 3.10 можна побачити інтерфейс перегляду параметрів, а їх додавання наведено на рисунку 3.11.

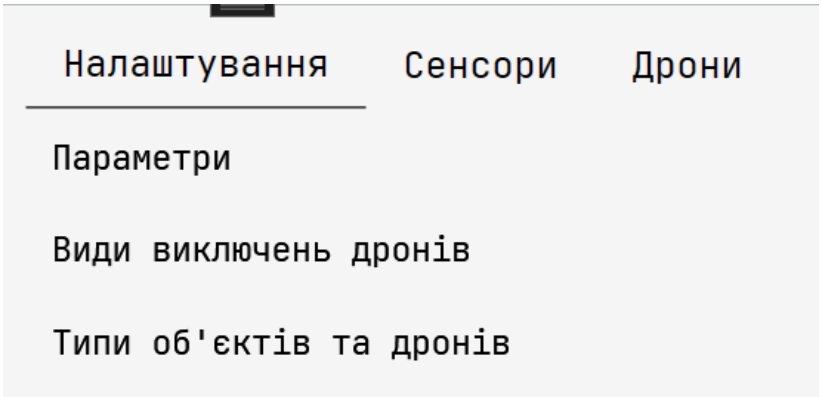


Рисунок 3.9 – Вибір базових налаштувань

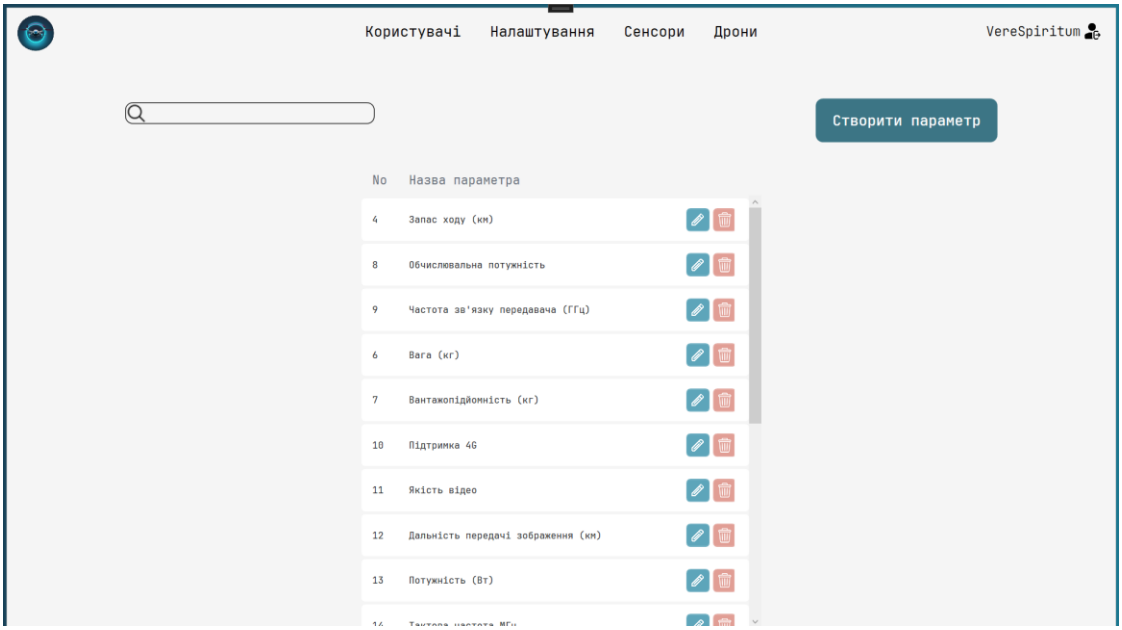


Рисунок 3.10 – Список параметрів

Додавання та редагування мають однаковий інтерфейс за відмінністю того, що при редагуванні ми бачимо попередній варіант запису. Редагування та видалення додані для того, аби у разі введення помилкових даних можливість все виправити чи перезаписати.

Інтерфейс перегляду видів виключення та типів різноманітних дронів мають подібний інтерфейс до параметрів. Їх простий інтерфейс виправданий

простотою задачі, яка полягає у звичайному наповнені системи критично важливою інформацією.

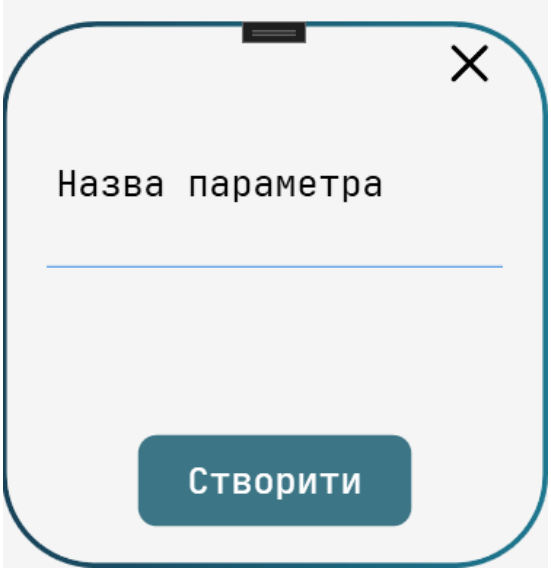


Рисунок 3.11 – Додавання параметра

Перегляд сенсорів наведено на рисунку 3.12, де відображено відповідний інтерфейс. Для того, аби не перенаповнити інтерфейс, параметри сенсорів можна переглянути, натиснувши на віподвідну кнопку, після чого відкриється модальне вікно з додатковою інформацією (рис. 3.13). Зміна сенсорів, є більш комплексною, як і створення. На відміну від додавання можливих для конфігурації параметрів або ж причин для виключення, треба внести базову інформацію про сенсор – назва, тип, а також визначити необхідні параметри. При редагуванні усі необхідні пункти вже заповнено інформацією й за потреби її можна змінити (рис. 3.14).

Додавання додаткових параметрів можливе з відкриття ще одного додаткового віконця, де обирається один з можливих параметрів та вводиться його значення. Додавання, перегляд та інші можливі дії з інформацією по дронам, які можуть бути використані оператором для створення рою та

налаштування завдання є ідентичними за інтерфейсом до роботи з сенсорами, так як вони мають один і той самий принцип опису роботи.

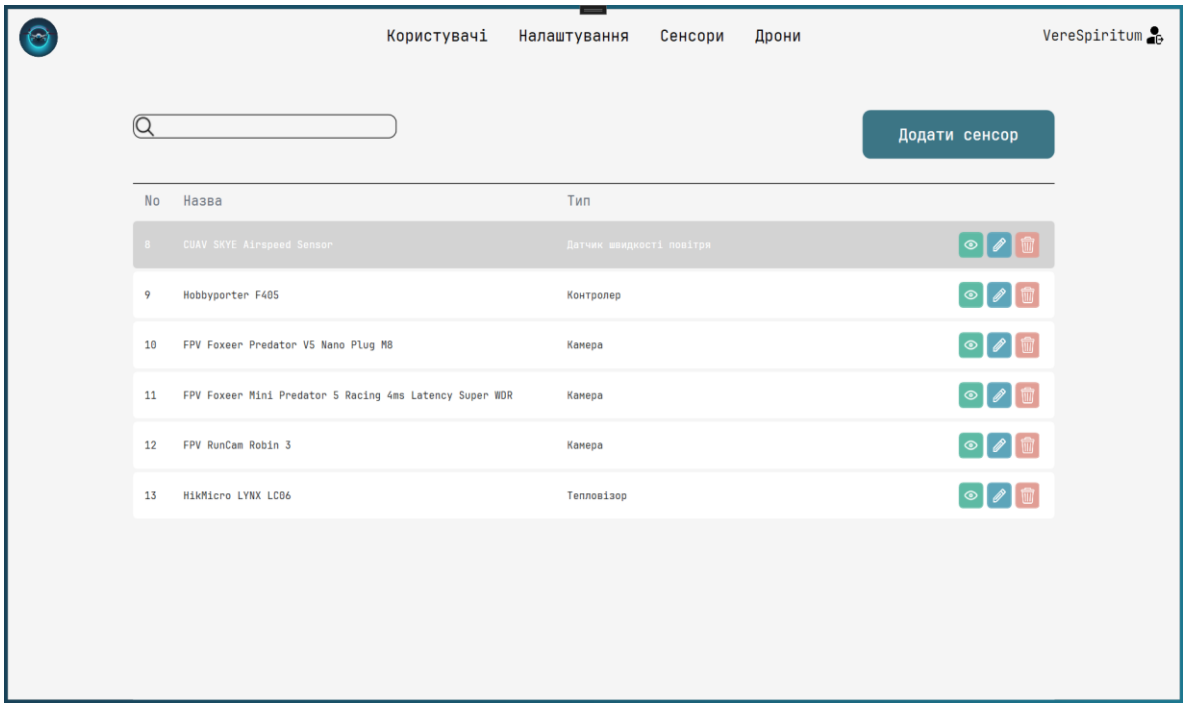


Рисунок 3.12 – Список сенсорів

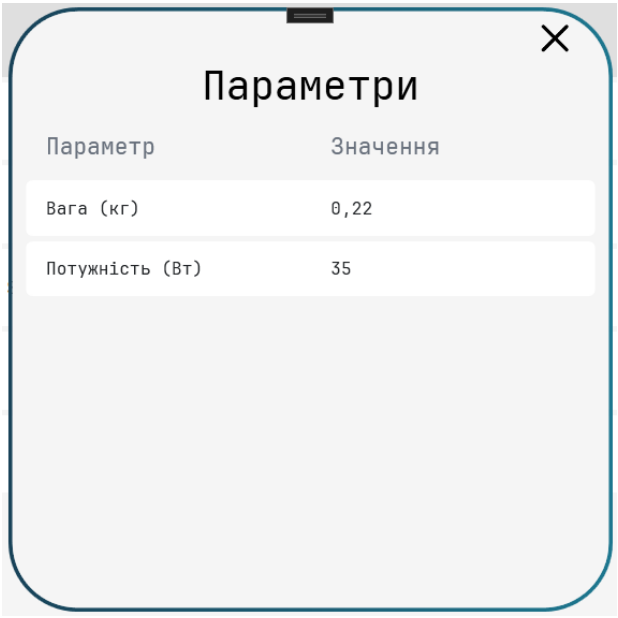


Рисунок 3.13 – Параметри сенсору

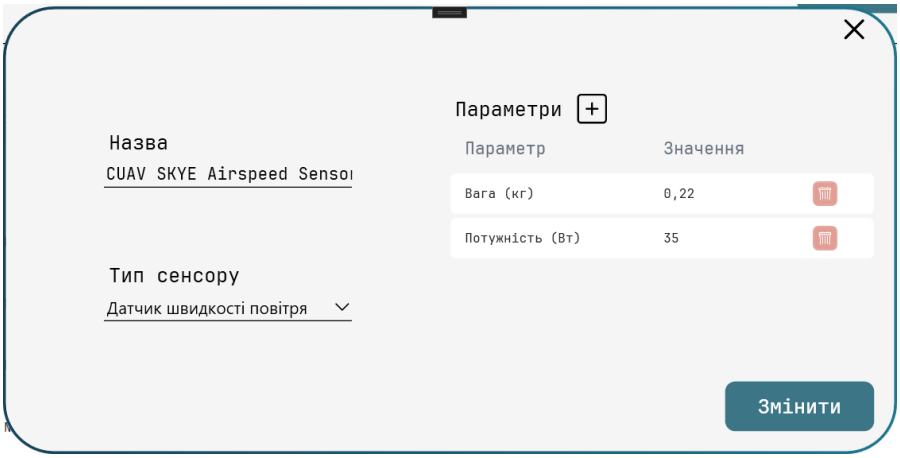


Рисунок 3.14 – Зміна сенсору

3.5.2 Інтерфейс оператора рою

Навігаційна панель оператора є досить компактною, проте кожен з трьох пунктів є комплексним й містить у собі достатньо велику кількість вікон та сторінок для забезпечення оператора усіма необхідними інструментами.

Почнемо зі навігаційного пункту «Нове завдання». Даний пункт дозволяє користувачу перейти на сторінку створення завдання (рис. 3.15). Для того, аби створити завдання оператору необхідно пройти кожен з пунктів та додати дрони у рій, встановити початкові контрольні точки та додати параметри для об’єктів, які необхідно знайти.

Вибір дрону відбувається шляхом звичайного вибору зі списку. Після цього дрон «додається у рій», тобто відображається у списку дронів, які обрано (рис. 3.16). Для кожного з них існує інтерфейс вибору сенсорів (рис. 3.17) та, відповідно до них, методів (рис. 3.18), які дозволено на додавання дрону з конфігурацією, яку обрано. Варто зазначити, що остаточно усі налаштування нодів рою входять у силу після натискання відповідної кнопки «Додати дрони».

Для дронів, сенсорів та методів є спеціальний інтерфейс, який є ідентичним зображеному на рис. 3.11, який відображає їх параметри, що допомагає оператору приймати рішення, щодо подальшого конфігурування.

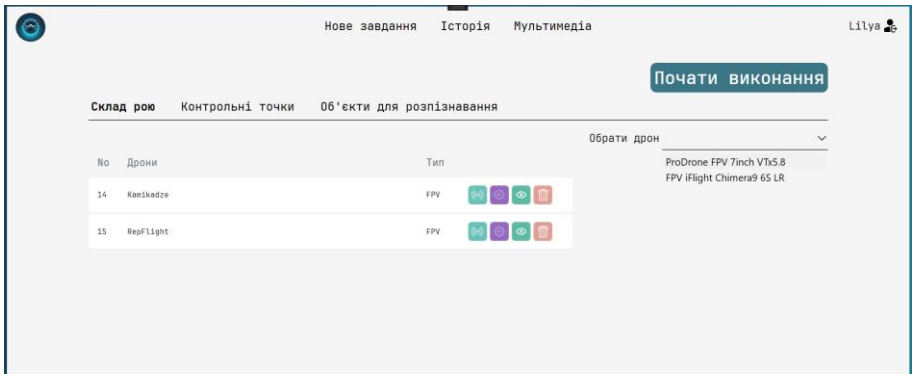


Рисунок 3.16 – Список дронів на завданні

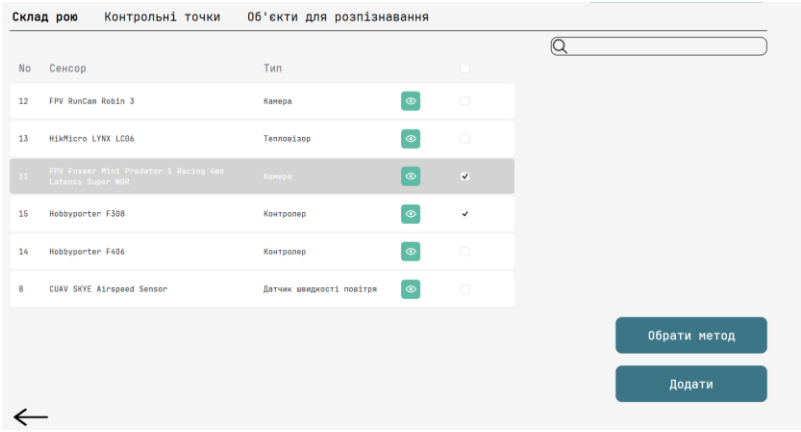


Рисунок 3.17 – Список «вільних» сенсорів

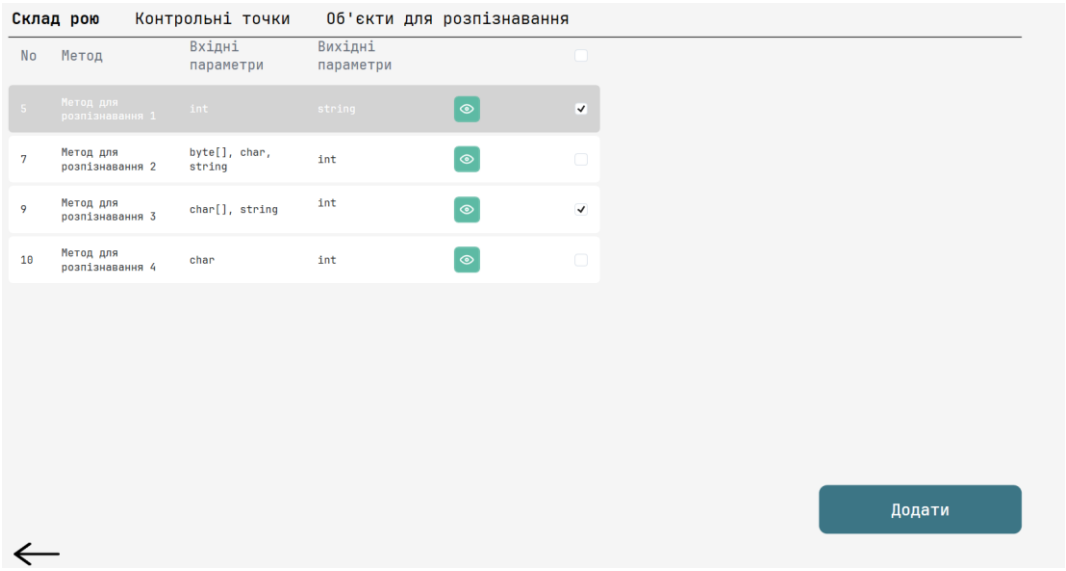


Рисунок 3.18 – Список методів для розпізнавання

Далі оператор має можливість визначити контрольні точки польоту (рис. 3.19) та, за потреби, переглянути розташування кожної з точок (рис. 3.20).

Рисунок 3.19 – Визначення контрольних точок

Також оператор має можливість перейти з допомогою додаткового навігаційного меню на сторінку «Об’єкти для розпізнавання», де відбувається додавання параметрів для об’єктів, які має шукати рій (рис. 3.21).

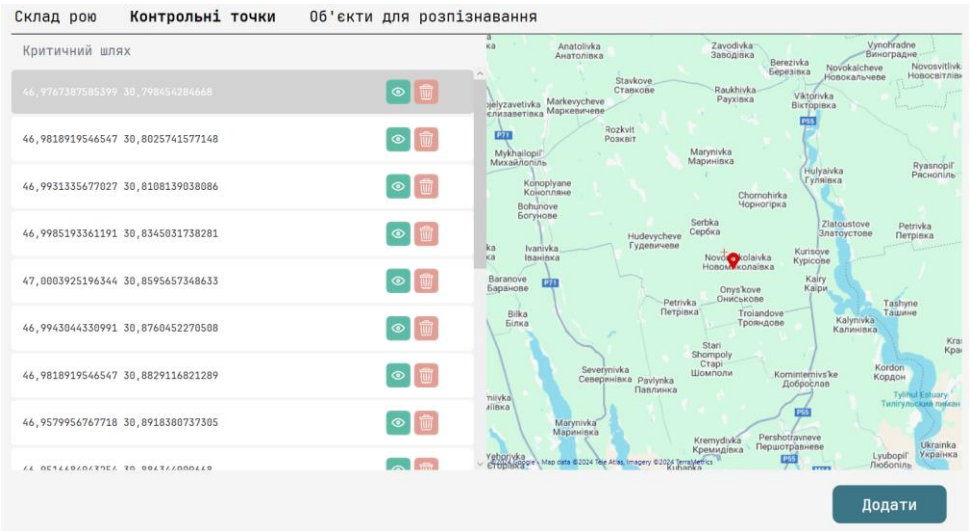


Рисунок 3.20 – Список контрольних точок

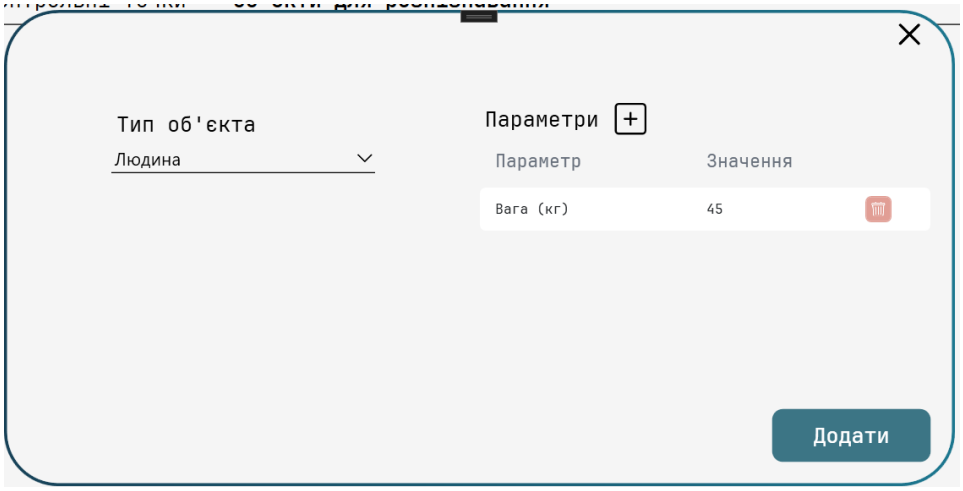


Рисунок 3.21 – Додавання об’єктів для пошуку

Розглянемо наступну частину головно меню оператора і це «Історія». Даний пункт відкриває для оператора список польотів або ж завдань які виконувалися роєм з певними відмітками про стан їх виконання (рис. 3.22).

З цієї сторінки оператор має можливість завершити політ або ж завдання, переглянути налаштування завдання та змінити їх за потреби, відкриваючи додаткові модальні вікна (рис. 3.23). Також існує можливість перегляду стану виконання польота через виклик контекстного меню, звідки існує перехід на місцезнаходження кожного дрону з рою (рис. 3.24, 3.25).

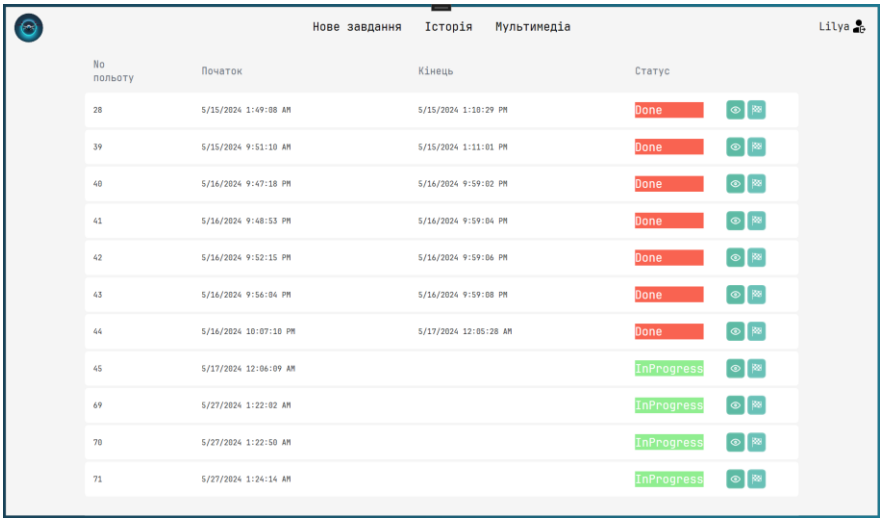


Рисунок 3.22 – Список польотів

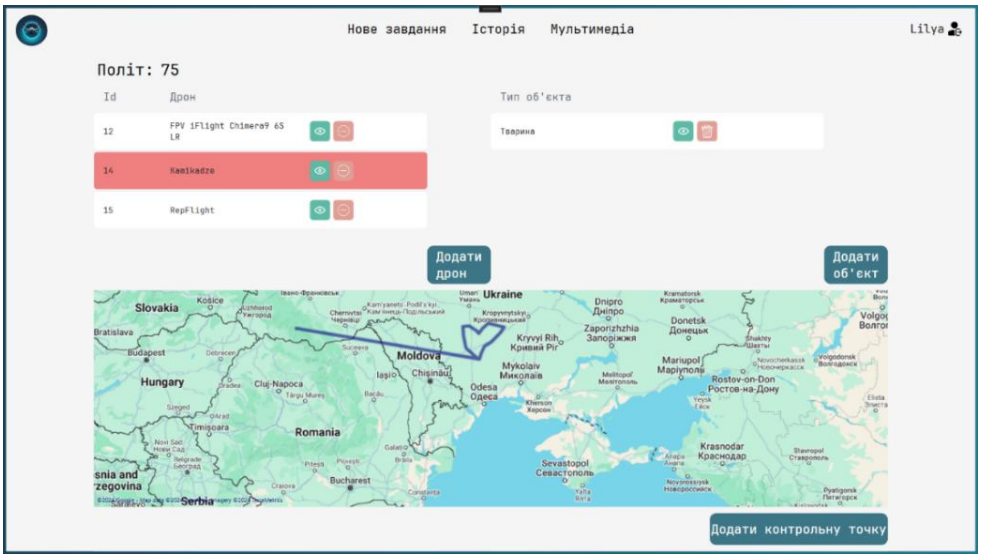


Рисунок 3.23 – Налаштування завдання

Перегляд мультимедіа файлів отриманих від рою виконується шляхом переходу за відповідним пунктом головного навігаційного меню. З даної сторінки є можливість переглянути фото або ж відео отримані від нодів рою, а також додавання розпізнаних об’єктів, що прив’язуються до конкретного файлу.

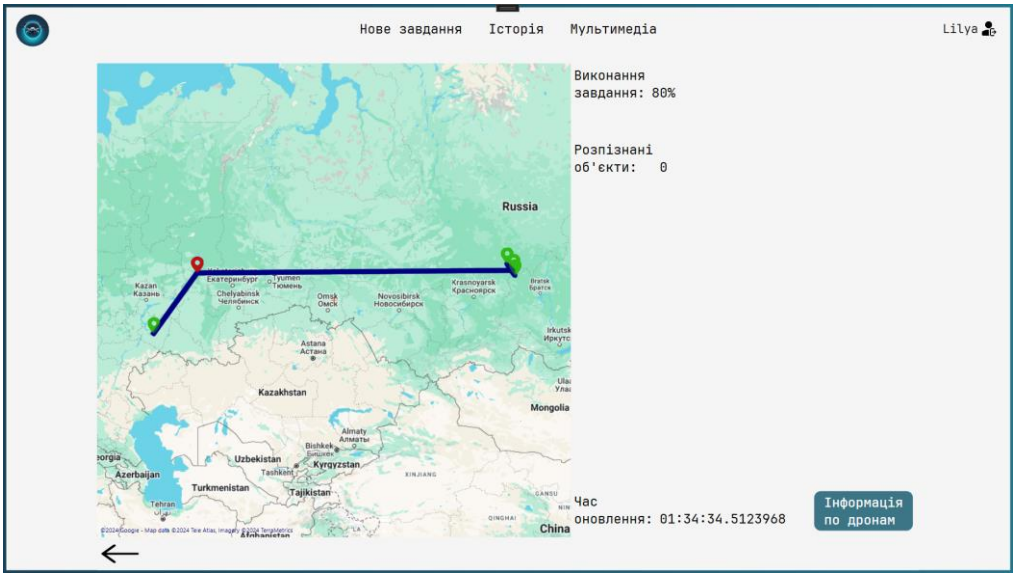


Рисунок 3.24 – Стан виконання завдання

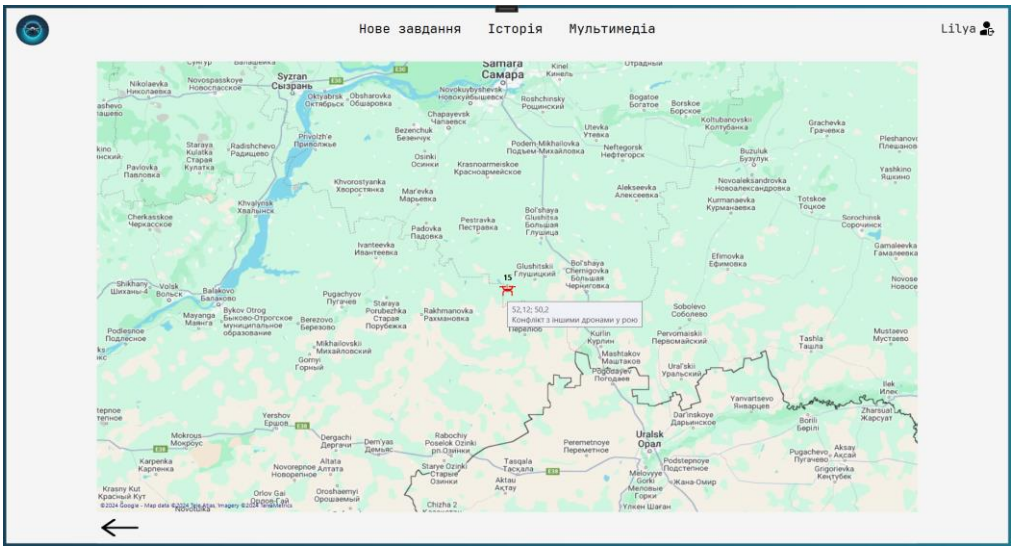


Рисунок 3.25 – Відслідковування роботи рою

3.5.3 Інтерфейс AI-спеціаліста

AI–спеціаліст, як і інші користувачі системи, має власну навігаційну панель, розміщену у верхній частині застосунку. Розглянемо кожен з пунктів.

«Новий метод» (рис. 3.26) є пунктом меню, котрий відповідає за додавання у систему нового методу. Файл завантажується з користувацького пристрою та після збереження методу завантажується у хмару.

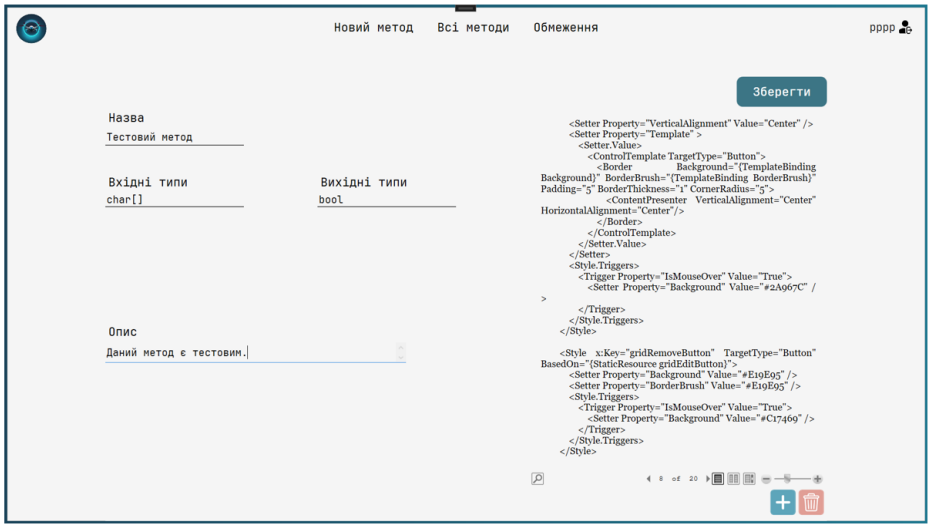
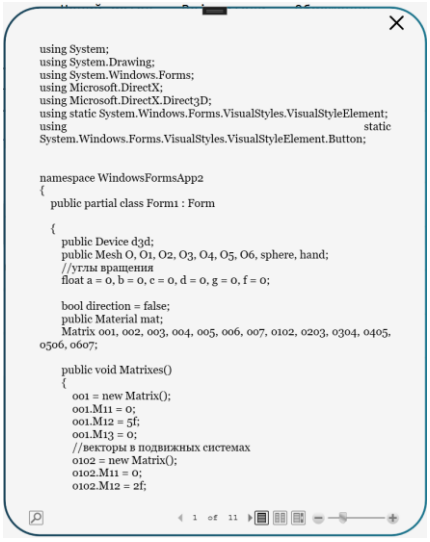


Рисунок 3.26 – Створення методу

Наступним пунктом меню є «Всі методи». Даний пункт відображає список методів, які є у системі, вказуючи на їх версію (рис. 3.27). З даної сторінки можна відкрити модальні вікна для перегляду опису методу та перегляду, відповідно, його коду (рис. 3.28), який завантажується з хмарного сховища. Сторінка редагування також викликається, шляхом кліку на відповідну кнопку у списку методів й інтерфейс є схожим на інтерфейс створення методу (рис. 3.29).

Рисунок 3.27 – Список методів



```
using System;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Button;

namespace WindowsFormsApp2
{
    public partial class Form1 : Form

    {
        public Device d3d;
        public Mesh O, O1, O2, O3, O4, O5, O6, sphere, hand;
        //углы вращения
        float a = 0, b = 0, c = 0, d = 0, g = 0, f = 0;

        bool direction = false;
        public Material mat;
        Matrix o01, o02, o03, o04, o05, o06, o07, o102, o203, o304, o405,
o506, o607;

        public void Matrixes()
        {
            o01 = new Matrix();
            o01.M11 = 0;
            o01.M12 = 5f;
            o01.M13 = 0;
            //векторы в подвижных системах
            o102 = new Matrix();
            o102.M11 = 0;
            o102.M12 = 2f;
```

Рисунок 3.28 – Перегляд коду методу

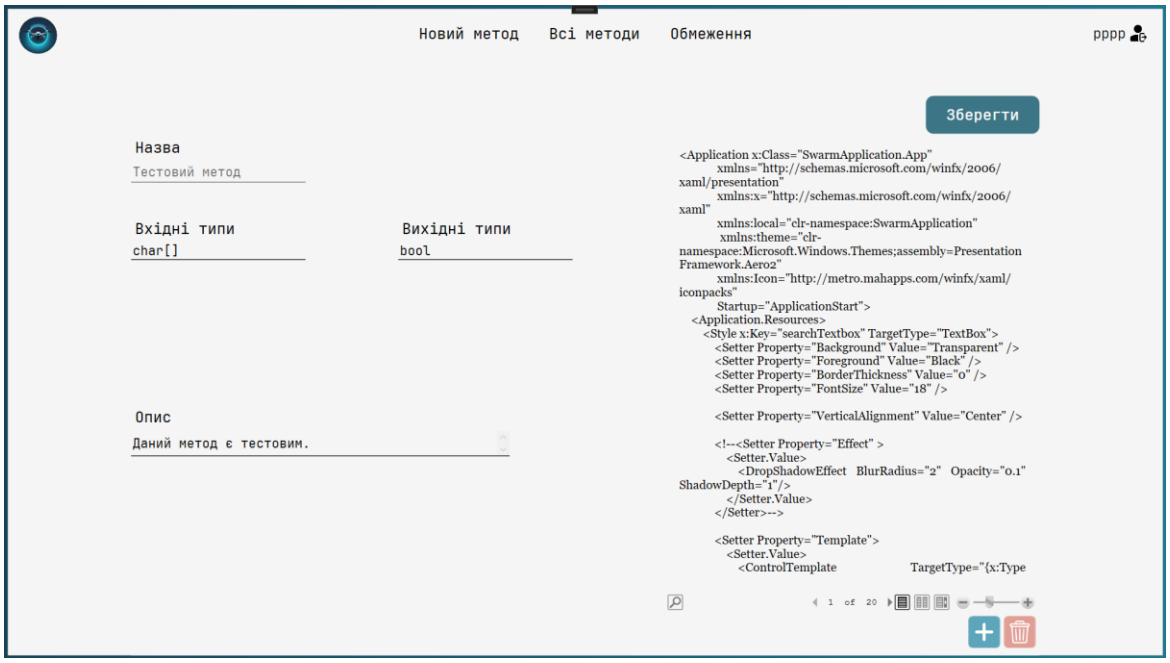


Рисунок 3.29 – Редагування методу

«Обмеження» – пункт меню, який перенаправляє користувача на сторінку, де є необхідний інтерфейс для додавання до методу необхідних обмежень для використання (рис. 3.30).

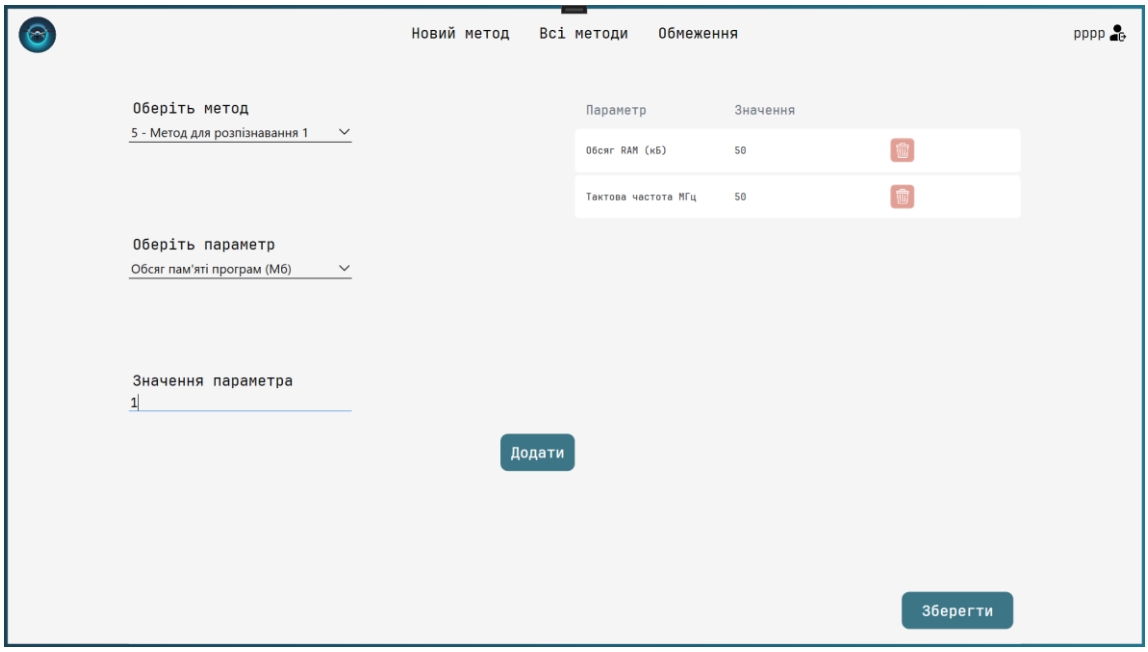


Рисунок 3.30 – Додавання обмежень для методів

ВИСНОВКИ

У результаті аналізу предметної області сформульована мета створення інформаційної системи підтримки ройових комплексів. Визначено основні особливості предметної області та сформульовано перелік ключових задач, необхідних для функціонування даної інформаційної системи. Окрім цього, визначено основні категорії користувачів: адміністратор, оператор, ноди та AI спеціаліст.

Для реалізації дипломного проєкта обрано трирівневу архітектуру й шаблон проєктування MVVM. Зберігання та взаємодія з даними відбувається з використанням СУБД PostgreSQL із задіянням фреймворку EntityFramework Core для спілкування з БД. Хмарний сервіс Google Cloud Platform забезпечує безперебійний обмін даними між застосунком та базою даних зі збереженням історії роботи. Технологія WPF забезпечує розробку зручного та інтерактивного користувацького інтерфейсу. Мова програмування – C#.

У ході розробки створено додаткові компоненти бази даних такі як функції, процедури та тригери, які відповідають за різноманітні аспекти роботи з даними і забезпечують цілісність даних або реалізують певні частини бізнес-логіки в базі даних. Додатково цілісність даних забезпечується з використанням механізму валідації на рівні застосунку.

Рольове розмежування даних забезпечує захист від несанкціонованого доступу до системи, що є важливим аспектом безпеки. Чітке визначення прав та привілеїв кожного користувача гарантує, що лише авторизовані особи мають доступ до певних функцій та даних.

Хмарний сервіс забезпечує можливість логування усіх подій, які відбуваються у БД, забезпечує безперебійний доступ до інформації для авторизованих користувачів за наявності Інтернету та надає ефективні засоби для відновлення бази даних у разі збоїв, що можуть виникнути в процесі роботи.

Створена інформаційна система успішно реалізує функціонал, що забезпечує виконання задач, визначених на етапі постановки. Вона надає інструменти для ефективного управління та обробки даних, а також забезпечує необхідний рівень безпеки та доступності.

Дана система має можливості розвитку та розширення:

1) інтеграція 3D – движка для відображення місцезнаходження дронів у тривимірному просторі;

2) функціонал для керування конкретними нодами рою без додавання контрольних точок для усього рою;

3) розширення інтерфейсу для можливості візуального моделювання розміщення сенсорів на дронах, що надає оператору більше можливостей для формування та конфігурування дронів й рою;

4) функціонал для слідкування за станом нодів й працездатності сенсорів.

Перспективи для розвитку даної інформаційної системи є досить широкими, оскільки вона надає базові можливості управління та моніторингу ройових комплексів, відповідно до визначених завдань. Інтеграція новітніх технологій та розширення функціональних можливостей підвищать привабливість та конкурентоспроможність системи на світовому ринку, забезпечуючи її стабільний розвиток та постійне вдосконалення.

Репозиторій з кодом програми розміщено на платформі GitHub [8].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тенденції виникнення надзвичайних ситуацій [Електронний ресурс]. Режим доступу: https://dsns.gov.ua/operational-information/nadzvicaini-situaciyi-v-ukrayini-2/tendeciyi-viniknennia-nadzvicainix-situacii
2. Tsariuk A. O., Malakhov E. V. The multilayer distributed intelligence system model for emergency area scanning. Herald of Advanced Information Technology. 2021; Vol. 4 No. 3: 268–277. DOI: https://doi.org/10.15276/hait.03.2021.6
3. Швець Ю. О. Хмарна система підтримки ройового комплексу / Швець Ю. О. Малахов Є. В., Козлов М. С. // Інформатика, інформаційні системи та технології: тези доповідей XX Всеукр. конференції студентів і молодих науковців. Одеса, 26 квітня 2024 р. – Одеса, 2024. – С. 134 – 136.
4. Drone Assist | Flight Planning & Approval App for Drone Operators [Електронний ресурс]. – Режим доступу: https://www.altitudeangel.com/solutions/drone-assist
5. DJI FlightHub 2 - Drone Software and Management Platform - DJI Enterprise [Електронний ресурс]. – Режим доступу: https://enterprise.dji.com/flighthub-2?site=enterprise&from=nav
6. AirMap [Електронний ресурс] – Режим доступу: https://appsource.microsoft. com/ru-by/product/web-apps/airmap-inc.airmap?
7. Google Cloud Documentation [Електронний ресурс] – Режим доступу: https:// cloud.google.com/docs
8. Репозиторій проекту [Електронний ресурс] – Режим доступу: https://github .com/VerreSpiritum/SwarmApplication

ДОДАТОК А

Задачі користувачів

Таблиця А.1 – Список задач користувачів хмарної ІС підтримки ройового комплексу

Номер	Задача	Вхідні дані	Вихідні дані
Адміністратор			
А1	Додавання нового користувача	Ідентифікатор користувача, логін, пароль, роль, дата створення	Новий користувач
А2	Редагування користувачів	Ідентифікатор користувача, логін, роль, пароль	Змінений користувач
А3	Видалення користувача	Ідентифікатор користувача	-
А4	Перегляд користувачів	Ідентифікатор користувача	Список: ідентифікатор користувача, логін, роль
А5	Додавання параметрів	Ідентифікатор параметра, назва	Новий параметр
А6	Видалення параметрів	Ідентифікатор параметра	-
А7	Редагування параметрів	Ідентифікатор параметра, назва	Змінений параметр
А8	Перегляд параметрів	Ідентифікатор параметра	Список: ідентифікатор параметра, назва

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
A9	Додавання причини вибуття дрону з рою	Назва	Нова причина вибуття
A10	Редагування причини вибуття дрону з рою	Ідентифікатор причини, назва	Змінена причина
A11	Видалення причини вибуття дрону з рою	Ідентифікатор причини	-
A12	Перегляд типів об’єктів/дронів/сенсорів	Ідентифікатор типу, чий тип	Список: ідентифікатор типу, значення
A13	Додавання типів об’єктів/дронів/сенсорів	Ідентифікатор типу, чий тип, значення	Новий тип
A14	Видалення типів об’єктів/дронів/сенсорів	Ідентифікатор типу	-
A15	Додавання сенсорів	Ідентифікатор сенсору, назва, тип	Новий сенсор
A16	Редагування сенсорів	Ідентифікатор сенсору, тип	Змінений сенсор
A17	Видалення сенсорів	Ідентифікатор сенсору	-
A18	Додавання параметрів до сенсорів	Ідентифікатор сенсору, параметр, значення	Новий параметр
A19	Редагування параметрів сенсорів	Ідентифікатор сенсору, ідентифікатор параметру, значення	Змінене значення

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
А20	Перегляд сенсорів	Ідентифікатор сенсору	Список: id сенсору, назва, тип, список: параметр, значення
А21	Перегляд дронів	Ідентифікатор дрону	Список: ідентифікатор, назва, тип, список: параметр, значення
А22	Редагування дронів	Ідентифікатор дрону, назва, тип	Оновлені дані
А23	Додавання нового дрона	Ідентифікатор дрону, назва, тип	Новий дрон
А24	Додавання параметрів для дрона	Ідентифікатор дрона, ідентифікатор параметра, значення	Новий параметр для дрону
А25	Редагування параметру для дрона	Ідентифікатор дрона, ідентифікатор параметра, значення	Оновлені дані
Оператор			
О1	Перегляд розміщення рою	Ідентифікатор польоту	Список: координати кожної ноди, час
О2	Перегляд інформації за завданням	Ідентифікатор польоту	Список: контрольні точки рою, дрони з сенсорами та методами, об’єкти з параметрами

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
О4	Перегляд польотів	-	Список: ідентифікатор польоту, дата початку, дата закінчення, статус
О5	Встановлення контрольних точок для рою	Широта, довгота, висота, ідентифікатор попередньої точки	Нова контрольно точка
О6	Перегляд контрольних точок рою	Ідентифікатор завдання	Список: широта, довгота, висота
О7	Видалення контрольної точки рою	Ідентифікатор польоту, ідентифікатор точки	-
О8	Додавання контрольної точки у завдання	Широта, довгота, висота	Ідентифікатор точки у завданні
О9	Додавання дрону у рій	Ідентифікатор дрону	Ідентифікатор дрону у завданні
О10	Додавання сенсорів на дрони для завдання	Ідентифікатор дрону на завданні, ідентифікатор сенсору	Дрон із сенсором

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
О11	Додавання методів на дрони для завдання	Ідентифікатор дрону на завданні, ідентифікатор методу	Дрон із методом
О12	Додавання параметрів до об’єктів для розпізнавання	Ідентифікатор типу об’єкту, ідентифікатор завдання, параметр, значення	Нові параметри
О13	Створення польоту	Ідентифікатор польоту, дата початку, радіус	Новий політ
О14	Завершення польоту	Ідентифікатор польоту, дата завершення	Зміна статусу польоту
О15	Виключення дрону з рою	Ідентифікатор дрону, ідентифікатор польоту, причина вибуття	-
О16	Перегляд виконання завдання	Ідентифікатор польоту	Відсоток виконання завдання, кількість розпізнаних об’єктів, статус дронів, розміщення дронів у певний момент часу

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
О17	Перегляд мультимедіа файлів	Ідентифікатор польоту	Список: ідентифікатор файлу, назва, опис, дата додавання, розмір, посилання
О18	Додавання розпізнаного об’єкта	Ідентифікатор файлу, тип об’єкту	Новий розпізнаний об’єкт
О19	Пошук мультимедіа файлів за розміром, форматом, датою створення, автором	Ідентифікатор, назва, розмір, формат, дата створення	Список мутльтимедіа файлів
AI - спеціаліст			
AI1	Перегляд методів	-	Список: ідентифікатор методу, назва, опис, вхідні типи, вихідні типи, посилання, дата додавання, версія
AI2	Додавання нових методів	Ідентифікатор методу, назва, опис, вхідні типи, вихідні типи, посилання, дата додавання, версія, ідентифікатор користувача	Новий метод

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
АІ3	Додавання обмежень	Ідентифікатор методу, параметр, значення	Нове обмеження
АІ4	Перегляд обмежень	Ідентифікатор метода	Список: параметр, значення
АІ6	Зміна обмежень	Ідентифікатор методу, ідентифікатор параметра, значення	-
АІ7	Видалення обмеження	Ідентифікатор методу, ідентифікатор параметра	-
Дрон			
Д1	Отримання інформації про контрольні точки польоту	Ідентифікатор польоту	Список: широта, довгота, висота
Д2	Надсилання власних координат	Ідентифікатор польоту, ідентифікатор дрону, довгота, широта, висота	Ідентифікатор польоту, ідентифікатор дрону, довгота, широта, висота

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
Д2	Надсилання власних координат	Ідентифікатор польоту, ідентифікатор дрону, довгота, широта, висота	Ідентифікатор польоту, ідентифікатор дрону, довгота, широта, висота
Д3	Надсилання інформації про розпізнаний об’єкт	Тип об’єкта, мультимедіа файл, ідентифікатор польоту, ідентифікатор дрону	Новий розпізнаний об’єкт
Д4	Надсилання мультимедіа файлів	Мультимедіа файл, ідентифікатор польоту, ідентифікатор дрону	Новий мультимедіа файлів
Д5	Виключення дрону	Ідентифікатор польоту, ідентифікатор дрону, причина вибуття	-

ДОДАТОК Б

Опис сутностей предметної області

Таблиця Б.1 – Опис сутностей ІС підтримки ройових комплексів

Ім’я атрибута	Призначення атрибута	Обмеження
User (користувач)		
id_user	Ідентифікатор користувача системи	первинний ключ
login	Логін користувача	Унікальне, не порожнє
date_of_creation	Дата створення користувача	Не порожнє
group	Група	Не порожнє, значення з множини: ‘operator’, ‘ai_specialist’, ‘drone’, ‘admin’, ‘cartographer’
Type_of_sensors (Типи сенсорів)		
id_type_of_sensor	Ідентифікатор типу сенсору	Первинний ключ
type	Тип	Не порожнє
Type_of_objects (Типи об’єктів)		
id_type_of_object	Ідентифікатор типу об’єкта	Первинний ключ
type	Тип	Не порожнє
Type_of_drone (Типи дронів)		
id_type	Ідентифікатор типу дрона	Первинний ключ
type	Тип	Не порожнє
Sensor_parameter (Параметри сенсорів)		

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
id_parameter	Ідентифікатор параметру	Зовнішній ключ для зв’язку із сутністю parameters, не порожнє
id_sensor	Ідентифікатор сенсору	Зовнішній ключ для зв’язку із сутністю sensor, не порожнє
val	Значення параметру	Не порожнє
		Первинний ключ (id_parameter, id_sensor)
Sensor (Сенсори)		
id_sensor	Ідентифікатор сенсору	Первинний ключ
name	Назва сенсору	Не порожнє
type	Тип сенсору	Зовнішній ключ для зв’язку з сутністю type_of_sensors, не порожнє
Recognition_file (Файли у який розпізнано об’єкти)		
id_identified_object	Ідентифікатор об’єкту	Зовнішній ключ для зв’язку з таблицею identified_object, не порожнє
id_multimedia_files	Ідентифікатор мультимедіа файлу	Зовнішній ключ для зв’язку з таблицею multimedia_files, не порожнє

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
		Первинний ключ (id_identofoed_object, id_multimedia_files)
Points (Точки)		
id_point	Ідентифікатор точки	Первинний ключ
latitude	Широта	Не порожнє
longitude	Довгота	Не порожнє
altitude	Висота	Не порожнє
Parameters (Параметри)		
id_parameter	Ідентифікатор параметра	Первинний ключ
parameter	Назва параметру	Не порожнє
Object_parameter (Параметри об’єкта)		
id_object_parameter	Ідентифікатор параметра об’єкта	Первинний ключ
id_flight	Ідентифікатор польоту	Зовнішній ключ для зв’язку з сутністю flight_history, не порожнє
id_object	Ідентифікатор об’єкта	Зовнішній ключ для зв’язку з сутністю type_of_objects, не порожнє
id_parameter	Ідентифікатор параметра	Зовнішній ключ для зв’язку з сутністю parameters, не порожнє
val	Значення параметра	Не порожнє
Multimedia_file (мультимедіа файли)		
id_file	Ідентифікатор польоту	Первинний ключ

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
file_name	Назва файлу	Не порожнє
size	Розмір файлу	Не порожнє
format	Формат файлу	Не порожнє
creation_date	Дата додавання файлу	Не порожнє
id_drone_on_the_flight	Ідентфикатор дрону на завданні, який додав файл	Зовнішній ключ для зв’язку з сутністю drone_on_the_flight, не порожнє
explanation	Короткий опис того, що знаходиться у файлі	Не порожнє
link	Посилання на файл у хмарі	Не порожнє
duration	Тривалість файлу	Не порожнє
Method_version (версія методу)		
id_method_version	Ідентифікатор версії методу	Первинний ключ
id_method	Ідентифікатор методу	Зовнішній ключ для зв’язку із сутністю method_identification, не порожнє
version	Версія методу	Не порожнє
date	Дата додавання версії	Не порожнє
by_whom	Ким додано	Зовнішній ключ для зв’язку із сутністю user, не порожнє

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
id_method_permission	Ідентифікатор дозволу для метода	Первинний ключ
id_method	Ідентифікатор методу	Зовнішній ключ для зв’язку із сутністю method_identification, не порожнє
parameter	Параметр за яким буде порівнюватися доступність методу	Зовнішній ключ для зв’язку з сутністю parameters, не порожнє
value	Значення параметра	Не порожнє
Method_identification (Методи для розпізнавання)		
id_method	Ідентифікатор методу	Первинний ключ
name	Назва методу	Не порожнє
input_types	Вхідні параметри методу	Не порожнє
output_types	Вихідні параметри метода	Не порожнє
description	Опис методу	Не порожнє
date_of_creation	Дата додавання методу	Не порожнє
Map_versions (Версії мап)		
Id_map_version	Ідентифікатор версії мапи	Первинний ключ
Id_map	Ідентфиікатор мапи	Зовнішній ключ для зв’язку із сутністю map, не порожнє
version	Версія мапи	Не порожнє
date	Дата додавання версії	Не порожнє
by_whom	Ким додано версію мапи	Зовнішній ключ для зв’язку із сутністю user, не порожнє

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
Map (Мапи)		
id_map	Ідентифікатор мапи	Первинний ключ
area_name	Назва території	Не порожнє
date_of_creature	Дата додавання мапи у систему	Не порожнє
creator	Ким додано	Зовнішній ключ для зв’язку із сутністю user, не порожнє
identified_object (Розпізнані об’єкти)		
id_identified_object	Ідентифікатор розпізнаного об’єкта	Первинний ключ
type_of_object	Тип об’єкта, який розпізнано	Зовнішній ключ для зв’язку із сутністю type_of_objects, не порожнє
time	Час, коли додано	Не порожнє
id_drone_on_the_flight	Ідентифікатор дрону, який є присутнім на завданні	Зовнішній ключ для зв’язку із сутністю drone_on_the_flight, не порожнє
Flight_points (Контрольні точки польоту)		
Id_flight_point	Ідентифікатор контрольної точки польоту	Первиннй ключ
Id_flight	Ідентифікатор польоту	Зовнішній ключ для зв’язку із сутністю Flight_history, не порожнє

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
id_point	Ідентифікатор точки	Зовнішній ключ для зв’язку із сутністю Points, не порожнє
id_prev_point	Ідентифікатор минулої точки польоту	Зовнішній ключ для зв’язку із сутністю Flight_points, може бути порожнім
Flight_history (Історія польотів)		
id_flight	Ідентифікатор польоту	Первинний ключ
radius	Радіус сканування	Не порожнє
start	Дата початку польоту	Не порожнє
end	Дата кінця польоту	Може бути порожнім
Exclusion_reason (Причини виключення дронів)		
id_reason	Ідентифікатор причини вибуття дрону з польоту	Первинний ключ
reason	Причина	Не порожнє
Exclusion_drone (Виключені дрони)		
id_exclusion	Ідентифікатор дрону з причиною його виключення	Первинний ключ
id_drone_on_the_flight	Ідентифікатор дрону, який перебуває на завданні	Зовнішній ключ для зв’язку із сутністю Drone_on_the_flight, не порожнє

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
id_reason	Ідентифікатор причини	Зовнішній ключ для зв’язку із сутністю Exclusion_reason, не порожнє
date	Дата виключення	Не порожнє
Drone_with_sensors (Дрони з сенсорами)		
id_drone	Ідентифікатор дрону, який перебуває на завданні	Зовнішній ключ для зв’язку із сутністю Drone_on_the_flight, не порожнє
id_sensor	Ідентифікатор сенсора	Зовнішній ключ для зв’язку із сутністю Sensors, не порожнє
		Первинний ключ (id_drone, id_sensor)
Drone_with_method (Дрони з методами)		
id_drone_with_method	Ідентифікатор методу, який належить до конкретного дрону на завданні	Первинний ключ
id_drone	Ідентифікатор дрону на завданні	Зовнішній ключ для зв’язку із сутністю Drone_on_the_flight, не порожнє

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
id_method	Ідентифікатор методу для розпізнавання	Зовнішній ключ для зв’язку із сутністю Method_identification, не порожнє
Drone_parameter (Параметри дрон)		
id_drone	Ідентифікатор дрона	Первинний ключ
id_parameter	Ідентифікатор параметра	Зовнішній ключ для зв’язку із сутністю parameters, не порожнє
val	Значення параметра для конкретного дрона	Не порожнє
Drone_on_the_flight (Дрон на завданні)		
id_drone_on_flight	Ідентифікатор дрона на завданні	Первинний ключ
id_flight	Ідентифікатор сформованого польота	Зовнішній ключ для зв’язку із сутністю Flight_history, не порожнє
id_drone	Ідентифікатор дрона	Зовнішній ключ для зв’язку із сутністю drone, не порожнє
date_of_adding	Дата додавання дрона у завданні	Не порожнє
Drone_location (Місцезнаходження дрона)		
id_location	Ідентифікатор локації дрона	Первинний ключ

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
id_drone	Ідентифікатор дрона	Зовнішній ключ для зв’язку із сутністю drone, не порожнє
date	Дата додавання інформації	Не порожнє
id_point	Ідентифікатор точки	Зовнішній ключ для зв’язку із сутністю Points, не порожнє
Drone (Дрон)		
id_drone	Ідентифікатор дрона	Первинний ключ
name	Назва дрона	Не порожнє
type_of_drone	Тип дрона	Зовнішній ключ для зв’язку із сутністю Type_of_drone, не порожнє
Debezium_offset_storage		
id	Ідентифікатор запису	Первинний ключ
offset_key	Ключ для зберігання зміщення, використовується Debezium для відстеження змін	Не порожнє
offset_val	Значення зміщення	Не порожнє
record_insert_ts	Мітка часу додавання запису	Може бути порожнім
record_insert_seq	Послідовність вставки	Може бути порожнім
Debezium_database_history		

Продовження таблиці Б.1

Ім’я атрибута	Призначення атрибута	Обмеження
id	Ідентифікатор запису	Первинний ключ
history_data	Інформація про схему БД	Не порожнє
history_data_seq	Послідовність даних історії	Не порожнє
record_insert_ts	Мітка вставки запису	Може бути порожнім
record_insert_seq	Послідовність вставки запису	Може бути порожнім

ДОДАТОК В

Запити на створення БД

```
create table debezium_database_history
(
    id                varchar not null,
    history_data      text    not null,
    history_data_seq  integer not null,
    record_insert_ts  timestamp with time zone,
    record_insert_seq integer,
    primary key (id)
);

create table debezium_offset_storage
(
    id                varchar not null,
    offset_key        varchar not null,
    offset_val        varchar not null,
    record_insert_ts  timestamp with time zone,
    record_insert_seq integer,
    primary key (id)
);

create table exclusion_reason
(
    id_reason serial,
    reason    varchar not null,
    primary key (id_reason)
);

create table flight_history
(
    id_flight serial,
    radius    numeric   not null,
    start     timestamp not null,
    "end"     timestamp,
    primary key (id_flight)
);

create table method_identification
(
    id_method        serial,
    name             varchar   not null,
```

```
    input_types      varchar   not null,
    output_types     varchar   not null,
    description      varchar   not null,
    date_of_creature timestamp not null,
    primary key (id_method)
);

create table parameters
(
    id_parameter serial,
    parameter    varchar not null,
    primary key (id_parameter)
);

create table method_permission
(
    parameter            integer not null,
    value                numeric not null,
    id_method_permission serial,
    id_method            integer not null,
    primary key (id_method_permission),
    constraint method_permission_pk
        unique (id_method_permission),
    foreign key (id_method) references method_identification,
    constraint method_permission_parameters_id_parameter_fk
        foreign key (parameter) references parameters
);

create table points
(
    id_point  serial,
    latitude  numeric not null,
    longitude numeric not null,
    altitude  numeric not null,
    primary key (id_point)
);

create table flight_points
(
    id_flight_point serial,
    id_flight       integer not null,
    id_point        integer not null,
    id_prev_point   integer,
    primary key (id_flight_point),
    foreign key (id_prev_point) references flight_points
        on update cascade on delete cascade,
```

```
    constraint flight_points_points_id_point_fk
        foreign key (id_point) references points
            on update cascade on delete cascade,
    constraint flight_points_flight_history_id_flight_fk
        foreign key (id_flight) references flight_history
            on update cascade on delete cascade
);

create table "user"
(
    id_user          serial,
    login            varchar    not null,
    date_of_creation timestamp  not null,
    "group"          group_name not null,
    primary key (id_user),
    unique (login)
);

create table map
(
    id_map           serial,
    area_name        varchar   not null,
    date_of_creature timestamp not null,
    creator          integer   not null,
    primary key (id_map),
    foreign key (creator) references "user"
        on update cascade on delete cascade
);

create table map_versions
(
    id_map_version serial,
    id_map         integer   not null,
    version        varchar   not null,
    date           timestamp not null,
    by_whom        integer   not null,
    primary key (id_map_version),
    constraint map_versions_by_whome_fkey
        foreign key (by_whom) references "user"
            on update cascade on delete cascade,
    foreign key (id_map) references map
        on update cascade on delete cascade
);

create table method_version
(
```

```
    id_method_version serial,
    id_method         integer   not null,
    version           varchar   not null,
    date              timestamp not null,
    by_whom           integer   not null,
    link              varchar   not null,
    primary key (id_method_version),
    foreign key (by_whom) references "user"
        on update cascade on delete cascade,
    foreign key (id_method) references method_identification
        on update cascade on delete cascade
);

create table type_of_drone
(
    id_type serial,
    type    varchar not null,
    primary key (id_type),
    unique (type)
);

create table drone
(
    id_drone      serial,
    name          varchar not null,
    type_of_drone integer not null,
    primary key (id_drone),
    foreign key (type_of_drone) references type_of_drone
        on update cascade on delete cascade
);

create table drone_on_the_flight
(
    id_drone_on_flight serial,
    id_flight          integer   not null,
    id_drone           integer   not null,
    date_of_adding     timestamp not null,
    primary key (id_drone_on_flight),
    unique (id_flight, id_drone),
    foreign key (id_drone) references drone
        on update cascade on delete cascade,
    foreign key (id_flight) references flight_history
        on update cascade on delete cascade
);

create table drone_location
```

```
(
    id_location serial,
    id_drone    integer   not null,
    date        timestamp not null,
    id_point    integer   not null,
    primary key (id_location),
    foreign key (id_drone) references drone_on_the_flight
        on update cascade on delete cascade,
    constraint drone_location_points_id_point_fk
        foreign key (id_point) references points
);

create table drone_parameter
(
    id_drone     integer not null,
    id_parameter integer not null,
    val          numeric not null,
    primary key (id_drone, id_parameter),
    foreign key (id_drone) references drone
        on update cascade on delete cascade,
    foreign key (id_parameter) references parameters
        on update cascade on delete cascade
);

create table drone_with_method
(
    id_drone_with_method serial,
    id_drone             integer not null,
    id_method            integer not null,
    primary key (id_drone_with_method),
    foreign key (id_drone) references drone_on_the_flight
        on update cascade on delete cascade,
    foreign key (id_method) references method_identification
        on update cascade on delete cascade
);

create table exclusion_drone
(
    id_exclusion       serial,
    id_drone_on_flight integer   not null,
    id_reason          integer   not null,
    date               timestamp not null,
    primary key (id_exclusion),
    foreign       key       (id_drone_on_flight)       references
drone_on_the_flight
        on update cascade on delete cascade,
```

```
    foreign key (id_reason) references exclusion_reason
        on update cascade on delete cascade
);

create table multimedia_file
(
    id_file                serial,
    file_name              varchar   not null,
    size                   numeric   not null,
    format                 varchar   not null,
    creation_date          timestamp not null,
    id_drone_on_the_flight integer   not null,
    explanation            text      not null,
    link                   text      not null,
    duration               numeric   not null,
    primary key (id_file),
    foreign      key      (id_drone_on_the_flight)      references
drone_on_the_flight
        on update cascade on delete cascade
);

create table type_of_objects
(
    id_type_of_object serial,
    type              varchar not null,
    primary key (id_type_of_object),
    unique (type)
);

create table identified_object
(
    id_identified_object serial,
    type_of_object       integer   not null,
    time                 timestamp not null,
    id_drone_on_flight   integer   not null,
    primary key (id_identified_object),
    constraint
identified_object_drone_on_the_flight_id_drone_on_flight_fk
        foreign      key      (id_drone_on_flight)      references
drone_on_the_flight
            on update cascade on delete cascade,
    foreign key (type_of_object) references type_of_objects
        on update cascade on delete cascade
);

create table object_parameter
```

```
(
    id_object_parameter serial,
    id_flight           integer not null,
    id_object           integer not null,
    id_parameter        integer not null,
    val                 varchar not null,
    primary key (id_object_parameter),
    foreign key (id_flight) references flight_history
        on update cascade on delete cascade,
    foreign key (id_object) references type_of_objects
        on update cascade on delete cascade,
    foreign key (id_parameter) references parameters
        on update cascade on delete cascade
);

create table recognition_file
(
    id_identified_object integer not null,
    id_multimedia_files  integer not null,
    primary key (id_multimedia_files, id_identified_object),
    foreign      key      (id_identified_object)      references
identified_object
        on update cascade on delete cascade,
    foreign key (id_multimedia_files) references multimedia_file
        on update cascade on delete cascade
);

create table type_of_sensors
(
    id_type_of_sensor serial,
    type              varchar not null,
    primary key (id_type_of_sensor),
    unique (type)
);

create table sensor
(
    id_sensor serial,
    name      varchar not null,
    type      integer not null,
    primary key (id_sensor),
    unique (name, type),
    constraint fk_for_sensor_type
        foreign key (type) references type_of_sensors
);
```

```
create table drone_with_sensors
(
    id_drone  integer not null,
    id_sensor integer not null,
    primary key (id_drone, id_sensor),
    foreign key (id_drone) references drone_on_the_flight
        on update cascade on delete cascade,
    foreign key (id_sensor) references sensor
        on update cascade on delete cascade
);

create table sensor_parameter
(
    id_parameter integer not null,
    id_sensor    integer not null,
    val          numeric not null,
    primary key (id_parameter, id_sensor),
    foreign key (id_parameter) references parameters
        on update cascade on delete cascade,
    foreign key (id_sensor) references sensor
        on update cascade on delete cascade
);
```

ДОДАТОК Г

Запити на створення тригерів збереження цілісності ІС

Тригери збереження цілісності:

```
--################################################
--[ТРИГЕР] Перевірка параметрів
--Опис: не можна додавати параметр, який вже існує у системі.
--################################################

create or replace function check_adding_parameters()
returns trigger as
$$
declare existing_row integer;
begin
    -- отримуємо кількість співпадінь параметрів
    select count(*) into existing_row
        from parameters
        where parameter = NEW.parameter;
    -- якщо є хоча б одне співпадіння - додавання є неможливим
    if existing_row > 0 then
    raise exception 'Value already exists in the table';
    end if;

    return new;
end;
$$
language plpgsql;

create trigger check_new_row_in_parameters
    before insert on parameters
    for each row
    execute function check_adding_parameters();
```

Лістинг Г.1 – Тригер для перевірки параметрів

```
--################################################
--[ТРИГЕР] Перевірка причин вибуття дронів з польоту
--Опис: не можна додати причину, яка вже є у системі.
--################################################

create function check_adding_reason() returns trigger
    language plpgsql
as
$$
    declare existing_row integer;
```

Лістинг Г.2 – Тригер для перевірки причин вибуття дронів з польоту

```
        begin
        select count(*) into existing_row -- не можна додати
причину, яка вже є у системі
            from exclusion_reason where reason = NEW.reason;
        if existing_row > 0 then
    raise exception 'Value already exists in the table';
    end if;
    return new;
    end; $$ language plpgsql;
create trigger check_new_row_in_exclusion_reason
    before insert on exclusion_reason
    for each row execute function check_adding_reason();
```

Лістинг Г.2, аркуш 2

```
--################################################
--[ТРИГЕР] Додавання дрону у завдання
--Опис: не можна додати дрон, який перебуває на завданні, навіть
якщо його виключено.
--################################################

create trigger check_new_row_in_drone_on_the_flight
    before insert on drone_on_the_flight
    for each row
    execute function check_adding_drone_on_the_flight();

create function check_adding_drone_on_the_flight() returns
trigger
    language plpgsql
as
$$
    declare existing_row integer;
        begin
        select count(*) into existing_row
        from drone_on_the_flight JOIN flight_history fh using
(id_flight)
        where id_drone = NEW.id_drone and fh."end" is null;
        if existing_row > 0 then
    raise exception 'Drone already busy';
    end if;
        return new;
    end;
    $$;
```

Лістинг Г.3 - Тригер для додавання дрону у завдання

```
--################################################
--[ТРИГЕР] Додавання мультимедіа файлів
--Опис: не можна додати файл, якщо він вже є у системі.
--################################################
```

Лістинг Г.4 - Тригер для додавання дрону у завдання

```
create function check_adding_multimedia() returns trigger
    language plpgsql
as $$
declare ex_row integer;
begin
    select count(*) into ex_row
    from multimedia_file where link = new.link;
    if ex_row > 0 then
        raise exception 'File already exists';
    end if; return new;
end; $$;
create trigger check_new_row_in_multimedia
    before insert on multimedia_file
    for each row execute function check_adding_multimedia();
```

Лістинг Г.4, аркуш 2

```
--################################################
--[ТРИГЕР] Додавання точок у карти
--Опис: не можна додати файл, якщо він вже є у системі.
--################################################

create function check_adding_points() returns trigger language
plpgsql as $$
declare ex_row integer;
begin
    select count(*) into ex_row
    from points where latitude = new.latitude and longitude =
new.longitude and altitude = new.altitude;
    if ex_row > 0 then
raise exception 'This point already exists';
    end if;
    return new;
end; $$;
create trigger check_new_row_in_points
    before insert on points
    for each row
    execute function check_adding_points();
```

Лістинг Г.5 – Тригер для додавання точок для карти

```
--################################################
--[ТРИГЕР] Видалення дрону
--Опис: не можна списати дрон, який перебуває на завданні.
--################################################

create or replace  function try_delete_drone()
returns trigger
as $$
declare existing_row integer; begin
```

Лістинг Г.6 – Тригер для видалення дрону

```
    select count(*) into existing_row
        from drone
        join drone_on_the_flight using (id_drone)
    join flight_history on drone_on_the_flight.id_flight =
flight_history.id_flight
    where "end" is null and drone.id_drone == old.id_drone;
    if existing_row > 0 then
    raise exception 'Drone is busy!';
    end if;

    return old;
end;
$$
language plpgsql;
CREATE TRIGGER before_delete_drone
BEFORE DELETE ON drone
FOR EACH ROW
EXECUTE FUNCTION try_delete_drone();
```

Лістинг Г.6, аркуш 2

```
--################################################
--[ТРИГЕР] Видалення сенсору
--Опис: не можна списати сенсор, який перебуває на завданні.
--################################################

create or replace  function try_delete_sensor()
returns trigger as $$
declare
existing_row integer;
begin
    select count(*) into existing_row from sensor
            join drone_with_sensors using (id_sensor)
        join drone_on_the_flight using (id_drone)
    join flight_history on drone_on_the_flight.id_flight =
flight_history.id_flight
    where "end" is null and sensor.id_sensor == old.id_sensor;
    if existing_row > 0 then
    raise exception 'Sensor is busy!';
    end if;

    return old;
end;
$$
language plpgsql;

CREATE TRIGGER before_delete_sensor
BEFORE DELETE ON sensor
FOR EACH ROW
EXECUTE FUNCTION try_delete_sensor();
```

Лістинг Г.7 – Тригер для видалення сенсору

```
--################################################
--[ТРИГЕР] Видалення методу
--Опис: не можна видалити метод, який використовується
--################################################

create or replace function try_delete_method()
returns trigger as
    $$
declare existing _ row integer;
begin
    select count ( * ) into existing_row
        from method_identification
            join drone_with_method using (id_method)
        join drone_on_the_flight using (id_drone)
    join flight_history on drone_on_the_flight.id_flight =
flight_history.id_flight
    where "end" is null
and method_identification.id_method ==
old.id_method;

    if existing_row > 0 then
    raise exception 'Method is using!';
    end if;
    return old;
end ;
$$
language plpgsql;
CREATE TRIGGER before_delete_method
BEFORE DELETE ON drone
FOR EACH ROW
EXECUTE FUNCTION try_delete_method( );
```

Лістинг Г.8 – Тригер для видалення методу

ДОДАТОК Д

Запити на створення представлень для вирішення задач користувачів

```
--################################################
--[ПРЕДСТАВЛЕННЯ]
--Опис: перегляд дронів з їх параметрами
--################################################

create or replace view show_drones_parameters(id_drone,
parameter, val) as
SELECT d.id_drone, parameters.parameter, drone_parameter.val
FROM drone d
         JOIN drone_parameter USING (id_drone)
         JOIN parameters USING (id_parameter);
```

Лістинг Д.1 – Представлення для перегляду дронів з їх параметрами

```
--################################################
--[ПРЕДСТАВЛЕННЯ]
--Опис: перегляд дозволів для методів
--################################################

create or replace view show_method_permissions(id_method,
parameter, value) as
SELECT mp.id_method,
       p.parameter,
       mp.value
FROM method_permission mp
         JOIN parameters p ON mp.parameter = p.id_parameter;
```

Лістинг Д.2 – Представлення для перегляду дозволів для методів

```
--################################################
--[ПРЕДСТАВЛЕННЯ]
--Опис: перегляд інформації по методам
--################################################

create or replace view show_methods_info
            (id_method, name, input_types, output_types,
by_whom, date_of_creature, version) as
SELECT mi.id_method,
       mi.name,
       mi.input_types,
```

Лістинг Д.3 – Представлення для перегляду інформації по методам

```
       mi.output_types,
       u.login AS by_whom,
       mi.date_of_creature,
       mv.version
FROM method_identification mi
         JOIN (SELECT method_version.id_method,
                      max(method_version.version::double
precision) AS max_version
               FROM method_version
               GROUP BY method_version.id_method) mv_max ON
mi.id_method = mv_max.id_method
         JOIN method_version mv ON mi.id_method = mv.id_method
AND mv.version::double precision = mv_max.max_version
         JOIN "user" u ON u.id_user = mv.by_whom;
```

Лістинг Д.3, аркуш 2

```
--################################################
--[ПРЕДСТАВЛЕННЯ]
--Опис: перегляд сенсорів з їх параметрами
--################################################

create or replace view show_sensors_parameters(id_sensor,
parameter, val) as
SELECT s.id_sensor,
       parameters.parameter,
       sensor_parameter.val
FROM sensor s
         JOIN sensor_parameter USING (id_sensor)
         JOIN parameters USING (id_parameter);
```

Лістинг Д.4 – Представлення для перегляду сенсорів з параметрами

ДОДАТОК Е

Запити на створення функцій та процедур для вирішення задач користувачів

Процедури для вирішення задач користувачів:

```
--################################################
--[ПРОЦЕДУРА]
--Опис: створення користувача з наданням йому відповідних
привілеїв та внесення його у таблицю «User»
--################################################

create or replace procedure create_user(IN n_user character
varying, IN password text, IN n_group character varying)
    language plpgsql
as $$
    declare check_user integer;
    begin
        SELECT COUNT(login) INTO check_user
    FROM "user"
        WHERE login = n_user;
    IF check_user > 0 THEN
        RAISE EXCEPTION 'Такий користувач вже є у системі';
    ELSE
        EXECUTE 'CREATE ROLE ' || quote_ident(n_user) || ' WITH
LOGIN PASSWORD ''' || $2 || ''' INHERIT;';
        EXECUTE 'GRANT ' || quote_ident(n_group) || ' TO ' ||
quote_ident(n_user);
        if n_group = 'admin' then
            EXECUTE 'ALTER ROLE "' || n_user ||'" CREATEROLE;';
        end if;
        INSERT INTO "user" (login, date_of_creation, "group")
        VALUES (n_user, current_timestamp, n_group);
    END IF;
    end;
$$;
```

Лістинг Е.1 – Процедура для створення користувача

```
--################################################
--[ПРОЦЕДУРА]
--Опис: оновлення інформації користувача з оновленням пароля
--################################################
```

Лістинг Е.2 – Процедура для оновлення даних про користувача з паролем

```
CREATE OR REPLACE PROCEDURE update_user(
    IN user_id INT,
    IN n_username VARCHAR,
    IN n_password TEXT,
    IN n_group varchar)
as
$$
    declare check_user integer;
    declare prev_username varchar;
    declare prev_group varchar;
    begin
        SELECT COUNT(login) INTO check_user
        FROM "user"
         WHERE login = n_username and id_user != user_id;
    IF check_user > 0
THEN
        RAISE EXCEPTION 'Такий користувач вже є у системі';
    else
        select login, "group" into prev_username, prev_group
        from "user"
        where user_id = id_user;
        if (select current_user = prev_username and prev_group
!= n_group and n_username != prev_username)
then
        raise exception 'Не можна змінити власну групу та
логін!';
        end if;
        if(user_id <= 0) then
            raise exception 'Немає користувача з такими даними';
            else
                execute 'ALTER ROLE "' || prev_username || '"
WITH PASSWORD ''' || $3 ||''' INHERIT;';
                if n_username != prev_username then
                    execute 'ALTER ROLE "' || prev_username ||
'" RENAME TO "' || n_username || '";';
                    end if;

            execute 'REVOKE ' || prev_group ||' FROM "' ||
n_username || '"; ';
            execute 'GRANT '|| n_group ||' TO "'|| n_username ||
'";';

            UPDATE "user" SET login = n_username, "group" =
n_group
            where id_user = user_id;
        end if;
    end if;
end;
$$
LANGUAGE plpgsql;
```

Лістинг Е.2, аркуш 2

```
--################################################
--[ПРОЦЕДУРА]
--Опис: оновлення інформації користувача без оновлення пароля
--################################################

CREATE OR REPLACE PROCEDURE update_user(
    IN user_id INT,
    IN n_username VARCHAR,
    IN n_group varchar)
as $$
    declare check_user integer;
    declare prev_username varchar;
    declare prev_group varchar;

    begin
        SELECT COUNT(login) INTO check_user
        FROM "user"
         WHERE login = n_username and id_user != user_id;

    IF check_user > 0 THEN
        RAISE EXCEPTION 'Такий користувач вже є у системі';
    else
        select login, "group" into prev_username, prev_group
        from "user"
        where user_id = id_user;
        if (select current_user = prev_username and prev_group
!= n_group and n_username != prev_username) then
        raise exception 'Не можна змінити власну групу та
логін!';
        end if;
        if(user_id <= 0) then
            raise exception 'Немає користувача з такими даними';
            else
                if n_username != prev_username then
                    execute 'ALTER ROLE "' || prev_username ||
'" RENAME TO "' || n_username || '";';
                    end if;

            execute 'REVOKE "' || prev_group ||'" FROM "' ||
n_username || '"; ';
            execute 'GRANT '|| n_group || ' TO "'|| n_username
|| '";';

            UPDATE "user" SET login = n_username, "group" =
n_group
            where id_user = user_id;
        end if;
    end if;
end; $$
LANGUAGE plpgsql;
```

Лістинг Е.3 – Процедура для оновлення даних про користувача без пароля

```
--################################################
--[ПРОЦЕДУРА]
--Опис: видалення користувача із системи з видаленням його ролі
та привілеїв
--################################################


CREATE OR REPLACE PROCEDURE delete_user(
    IN user_id  INT)
as $$
    declare check_user integer;
    declare username varchar;
    begin
        SELECT id_user INTO check_user
        FROM "user"
         WHERE id_user = user_id;
    IF check_user = 0 or check_user is null THEN
        RAISE EXCEPTION 'Немає користувача з такими даними';
    else
        select login into username
        from "user"
        where user_id = id_user;

        if (select current_user = username) then
        raise exception 'Не можна видалити самого себе!';
        else
        execute 'DROP ROLE "' || username || '";';
        delete from "user" where id_user = user_id;
        end if;
    end if;
end;
$$
LANGUAGE plpgsql;
```

Лістинг Е.4 – Процедура для видалення користувача із системи

```
--################################################
--[ПРОЦЕДУРА]
--Опис: додавання розміщення дронів
--################################################

create or replace procedure AddDroneLocation(drone_id integer,
flight_id integer, coordinates decimal[])
as $$
declare drone_flight_id integer; point_id integer;
    begin
    -- Initialize point_id to a default value
    point_id := -1;
-- Get the drone_flight_id
    SELECT id_drone_on_flight INTO drone_flight_id
```

Лістинг Е.5 – Процедура додавання інформації про розміщення дронів

```
    FROM drone_on_the_flight
    WHERE id_flight = flight_id AND id_drone = drone_id;
    -- Ensure drone_flight_id is found
    IF drone_flight_id IS NULL THEN
        RAISE EXCEPTION 'No matching drone_on_the_flight entry
found for flight_id % and drone_id %', flight_id, drone_id;
    END IF;
    -- Attempt to find an existing point
    SELECT id_point INTO point_id FROM points
    WHERE latitude = coordinates[1] AND longitude =
coordinates[2] AND altitude = coordinates[3];
    -- Insert the point if it does not exist
    IF NOT FOUND THEN
        INSERT INTO points (latitude, longitude, altitude)
        VALUES (coordinates[1], coordinates[2], coordinates[3])
        RETURNING id_point INTO point_id;
    END IF;
    -- Insert into drone_location
    INSERT INTO drone_location (id_drone, id_point, date)
    VALUES (drone_flight_id, point_id, CURRENT_DATE);
end; $$ LANGUAGE plpgsql;
```

Лістинг Е.5, аркуш 2

```
--################################################
--[ПРОЦЕДУРА]
--Опис: додавання дронів у завдання
--################################################

create or replace procedure AddDroneInTask(flight_id integer,
drone_id integer, sensors integer[], methods integer[])
    as $$
        declare droneOnFlight_id integer;
    begin
        insert into drone_on_the_flight (id_flight, id_drone,
date_of_adding) values (flight_id, drone_id, current_date)
            returning id_drone_on_flight into droneOnFlight_id;
        --insert sensors
        for i in 1..array_length(sensors, 1) loop
            insert into drone_with_sensors (id_drone, id_sensor)
values (droneOnFlight_id, sensors[i])
            ON CONFLICT DO NOTHING;
        end loop;
        --insert methods
        for i in 1..array_length(methods, 1) loop
            insert into drone_with_method (id_drone, id_method)
values (droneOnFlight_id, methods[i])
            ON CONFLICT DO NOTHING;
            end loop;
    end; $$ language plpgsql;
```

Лістинг Е.6 – Процедура для додавання дронів у завдання

Функції для вирішення задач користувачів:

```
--################################################
--[ФУНКЦІЯ]
--Опис: отримання відсортованих точок контрольного шляху
--################################################

CREATE OR REPLACE FUNCTION GetSortedPoints(flightId INT)
RETURNS TABLE(id_point INT, latitude numeric, longitude numeric)
AS $$
BEGIN
    RETURN QUERY
    WITH RECURSIVE RecursiveCTE AS ( SELECT fp.id_flight_point,
            p.latitude, p.longitude,
            fp.id_prev_point, 0 AS SortOrder
        FROM flight_points fp JOIN points p USING (id_point)
        WHERE fp.id_flight = flightId AND fp.id_prev_point IS
NULL
        UNION ALL
        SELECT
            fp.id_flight_point,
            p.latitude,
            p.longitude,
            fp.id_prev_point,
            r.SortOrder + 1
        FROM points p  JOIN flight_points fp ON p.id_point =
fp.id_point
        JOIN RecursiveCTE r ON fp.id_prev_point =
r.id_flight_point)
    SELECT r.id_flight_point, r.latitude, r.longitude
    FROM RecursiveCTE r ORDER BY SortOrder;
END; $$ LANGUAGE plpgsql;
```

Лістинг Е.7 – Функція для отримання відсортованих точок контрольного шляху

```
--################################################
--[ФУНКЦІЯ]
--Опис: додавання сенсорів у систему
--################################################

create or replace function create_sensor(nameS varchar, typeS
varchar, parameterN varchar[], parameterVal numeric[])
returns void
as $$
    declare id_new_sensor integer; id_type integer; id_param
integer;
    begin
```

Лістинг Е.8 – Функція для додавання сенсорів у систему

```
        select id_type_of_sensor into id_type
            from type_of_sensors ts
        where ts.type = typeS;
        insert into sensor (name, type) values
                                              (nameS, id_type)
        returning id_sensor into id_new_sensor;
        for i in 1..array_length(parameterN, 1) loop
            select id_parameter into id_param
                from parameters
            where parameter = parameterN[i];
            insert into sensor_parameter values
                                                (id_param,
id_new_sensor, parameterVal[i]);
        end loop;
    end;
    $$
language plpgsql;
```

Лістинг Е.8, аркуш 2

```
--################################################
--[ФУНКЦІЯ]
--Опис: додавання нових дронів у систему
--################################################
create or replace function create_drone(nameD varchar, typeD
varchar, parameterN varchar[], parameterVal numeric[])
returns void
as $$
    declare id_new_drone integer; id_typeD integer; id_param
integer;
    begin
        select id_type into id_typeD
            from type_of_drone
        where type = typeD;
        insert into drone (name, type_of_drone) values
                                               (nameD, id_typeD)
        returning id_drone into id_new_drone;
        for i in 1..array_length(parameterN, 1) loop
            select id_parameter into id_param
                from parameters
            where parameter = parameterN[i];

            insert into drone_parameter values
                                                (id_new_drone,
id_param, parameterVal[i]);
        end loop;
    end;
    $$
language plpgsql;
```

Лістинг Е.9 – Функція для додавання нових дронів у систему

ДОДАТОК Ж

Приклади програмних компонентів інтерфейсу користувача

```
<Window x:Class="SwarmApplication.AuthorizationWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentatio
"
        xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:SwarmApplication"
        xmlns:viewModel="clr-
namespace:SwarmApplication.ViewModels"
        xmlns:customcontrols="clr-
namespace:SwarmApplication.CustomControls"
        mc:Ignorable="d"
        Height="600" Width="400" ResizeMode="NoResize"
        WindowStyle="None"
        WindowStartupLocation="CenterScreen"
        MouseDown="AuthorizationWindow_OnMouseDown"
        AllowsTransparency="True" Background="Transparent">
    <Window.Resources>
        <BooleanToVisibilityConverter
x:Key="BooleanToVisibility"/> </Window.Resources>
    <Window.Visibility>
        <Binding Path="IsViewVisible" Mode="TwoWay"
Converter="{StaticResource BooleanToVisibility}" />
    </Window.Visibility>
    <Border CornerRadius="50" Background="WhiteSmoke"
BorderThickness="5"> <Border.Style>
            <Style TargetType="Border">
                <Setter Property="BorderBrush">
                    <Setter.Value>
                        <LinearGradientBrush EndPoint="1,0.5"
StartPoint="0,0.5">
                            <GradientStop Color="#1D778F"
Offset="1"/>
                            <GradientStop Color="#184258"/>
                        </LinearGradientBrush>
                    </Setter.Value>
                </Setter>
            </Style>
        </Border.Style>
        <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"/>
```

Лістинг Ж.1 – Приклад інтерфейсу користувача. Сторінка авторизації.

```
<RowDefinition Height="auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
        </Grid.RowDefinitions>
        <Button Grid.Row="0"
                Command="{Binding ExitCommand}"
                Width="13" Height="13"
                BorderBrush="{x:Null}"
                HorizontalAlignment="Right"
                Margin="0 20 25 0"
                Cursor="Hand" VerticalAlignment="Top">
            <Button.Background>
                <ImageBrush
ImageSource="/Views/CloseLogin.png"/>
            </Button.Background>
            <Button.Style>
                <Style TargetType="Button">
                    <Setter Property="Background"
Value="Transparent"></Setter>
                    <Style.Triggers>
                       <Trigger Property="IsMouseOver"
Value="True">
                           <Setter Property="Background"
Value="Transparent"></Setter>
                       </Trigger> </Style.Triggers>
                </Style>
            </Button.Style>
            <Button.Template>
                <ControlTemplate TargetType="Button">
                    <Border Background="{TemplateBinding
Background}"/>
                </ControlTemplate>
            </Button.Template>
        </Button>
        <Image Grid.Row="1" Source="/Resources/Logo.png"
Width="109" Height="109"/>
        <Grid Grid.Row="2" Margin="0 30 0 0">
            <Grid.RowDefinitions>
                <RowDefinition Height="*"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>
            <StackPanel Grid.Row="0"
HorizontalAlignment="Center" MinWidth="244">
                <Label FontSize="25" >
                        <Label.Resources>
                            <Style TargetType="Label">
                                <Setter Property="FontFamily"
Value="pack://application:,,,/Fonts/JetBrains
Mono/JetBrainsMono-
```

Лістинг Ж.1, аркуш 2

```
Regular.ttf#JetBrains Mono" />
                            </Style>
                        </Label.Resources>
                        Login</Label>
                <TextBox x:Name="TxtUser"
                         HorizontalContentAlignment="Left"
                         Height="28"
                         FontSize="17"
                         VerticalContentAlignment="Center"
                         BorderThickness="0, 0, 0, 1"

                         BorderBrush="Black"
                         Foreground="Black"

                         Padding="30 0 0 0"
                    Text="{Binding Username,
UpdateSourceTrigger=PropertyChanged}" >
                    <TextBox.Background>
                        <ImageBrush
ImageSource="/Views/user.png"
                                    Stretch="Uniform"
TileMode="None" AlignmentX="Left"/>
                    </TextBox.Background>
                    <TextBox.Resources>
                        <Style TargetType="TextBox">
                            <Setter Property="FontFamily"
Value="pack://application:,,,/Fonts/JetBrains
Mono/JetBrainsMono-Regular.ttf#JetBrains Mono" />
                            </Style>
                    </TextBox.Resources>
                </TextBox>


            </StackPanel>
            <StackPanel Grid.Row="1"
HorizontalAlignment="Center" MinWidth="244">
                <Label FontSize="25" >
                        <Label.Resources>
                            <Style TargetType="Label">
                                <Setter Property="FontFamily"
Value="pack://application:,,,/Fonts/JetBrains
Mono/JetBrainsMono-Regular.ttf#JetBrains Mono" />
                            </Style>
                        </Label.Resources>
                        Password:</Label>
                    <customcontrols:BindablePasswordBox
Password="{Binding Password, Mode=TwoWay ,
UpdateSourceTrigger=PropertyChanged}"
Height="28">
<customcontrols:BindablePasswordBox.Background>
```

Лістинг Ж.1, аркуш 3

```
<ImageBrush
ImageSource="/CustomControls/key.png"
                                Stretch="Uniform"
TileMode="None" AlignmentX="Left"/>

</customcontrols:BindablePasswordBox.Background>
                    </customcontrols:BindablePasswordBox>

                </StackPanel>
        </Grid>
        <TextBlock Grid.Row="3" VerticalAlignment="Center"
                   HorizontalAlignment="Center"
                   FontSize="17"
                   FontFamily="./Fonts/JetBrains
Mono/JetBrainsMono-Regular.ttf"
                   Margin="0 0 0 40"
                   Foreground="Red"
                   Text="{Binding ErrorMessage}">

        </TextBlock>
        <Button Grid.Row="4" Command="{Binding LoginCommand}"
                MinWidth="204"
                MinHeight="67"
                Cursor="Hand"
                IsDefault="True"
                HorizontalAlignment="Center"
                HorizontalContentAlignment="Center"
                VerticalContentAlignment="Center"
                FontSize="40"

                FontWeight="Bold"
                Foreground="White"
                Margin="0 0 0 20">
            <Button.Resources>
                <Style TargetType="Border">
                    <Setter Property="CornerRadius" Value="40"/>

                    </Style>

                </Button.Resources>

            <Button.Style>
                <Style TargetType="Button">
                    <Setter Property="FontFamily"
Value="pack://application:,,,/Fonts/JetBrains
Mono/JetBrainsMono-Regular.ttf#JetBrains Mono" />
                    <Setter Property="Background">
                        <Setter.Value>
                            <LinearGradientBrush
EndPoint="0,0.5" StartPoint="1,0.5">
```

Лістинг Ж.1, аркуш 4

```
                        <GradientStop Color="#1D778F"
                    Offset="1"/>
                                <GradientStop Color="#184258"/>
                            </LinearGradientBrush>
                        </Setter.Value>
                    </Setter>
                    <Style.Triggers>
                        <Trigger Property="IsMouseOver"
Value="True">
                            <Setter Property="Background" >
                                <Setter.Value>
                                    <LinearGradientBrush
EndPoint="0,0.5" StartPoint="1,0.5">
                                        <GradientStop
Color="#31BFE4" Offset="1"/>
                                        <GradientStop
Color="#184258"/>
                                    </LinearGradientBrush>
                                </Setter.Value>
                            </Setter>
                        </Trigger>
                    </Style.Triggers>
                </Style> </Button.Style> <Button.Template>
                <ControlTemplate TargetType="Button">
                    <Border Width="204" Height="67"
Background="{TemplateBinding Background}" CornerRadius= "30">
                        <ContentPresenter
HorizontalAlignment="Center" VerticalAlignment="Center"/>
                    </Border>
                </ControlTemplate>
            </Button.Template>
            Start </Button>
    </Grid>
    </Border>
</Window>
```

Лістинг Ж.1, аркуш 5

```
public class Show_methods_info
{
```

Лістинг Ж.2 – Приклад моделі даних

```
    [Key]
    public int id_method { set; get; }
    public string name { set; get; }
    public string input_types { set; get; }
    public string output_types { set; get;}
    public string by_whom { set; get; }
    public DateTime date_of_creature { get; set; }
    public string version { set; get; }
}
```

Лістинг Ж.2, аркуш 2

```
public string DropUser(int id)
{
    string sql;
    var user = _userContext.Users.FirstOrDefault(u => u.id_user
== id);
    if(user != null)
    {
        try
        {
            sql = $"call delete_user({id})";
            _userContext.Database.ExecuteSqlRaw(sql);
            _userContext.SaveChanges();
        }
        catch (Exception ex)
        {
            CheckResults = false;
            return ex.Message;
        }
        CheckResults = true; return "Успішно видалено!";
    }
    CheckResults = false;
    return "Немає такого користувача";
}
public string UpdateData(int id, NetworkCredential credential,
string group)
{
    try
    {
        string sql;
        if (!string.IsNullOrEmpty(credential.Password))
        {
            sql = $"call update_user({id},
'{credential.UserName}', '{credential.Password}', '{group}')";


        }
        else
```

Лістинг Ж.3 – Частина репозиторія адміністратора

```
{
            sql = $"call update_user({id}, '{
credential.UserName}', '{group}')";
        }
        _userContext.Database.ExecuteSqlRaw(sql);
        _userContext.SaveChanges();
    }
    catch (Exception ex)
    {
        CheckResults = false;
        return ex.Message;
    }
    CheckResults = true;
    return "Дані оновлено!";
}
public string CreateUser(NetworkCredential credential, string
group)
{
    try
    { string sql = $”call create_user(‘{credential.UserName}’,
‘{credential.Password}’, ‘{group}’)”;
        _userContext.Database.ExecuteSqlRaw(sql);
        _userContext.SaveChanges();
    }
    catch (Exception ex)
    {
        CheckResults=false;
        return ex.Message;
    }
    CheckResults=true;
    return “Дані оновлено!”;
}
```

Лістинг Ж.3, аркуш 2

```
public string AddDroneInFlight(int id_flight, int id_drone,
List<int> id_sensors, List<int> id_methods)
{
```

Лістинг Ж.4 – Частина сервіса адміністратора

```
    string res = repository.AddDroneInFlight(id_flight,
id_drone, id_sensors, id_methods);
    checkResult = repository.CheckResult;
    return res;
}
public async Task<ObservableCollection<MultimediaFile>>
GetMultimediaFiles()
{
    ObservableCollection<MultimediaFile> temp = await
repository.GetMultimediaFiles();
    return temp;
}
public string GetFileDescription(int id_file)
{
    return repository.GetFileDescription(id_file);
}
public async Task<ObservableCollection<MultimediaFile>>
SearchMultimediaFile(string text)
{
    ObservableCollection<MultimediaFile> temp = await
repository.GetMultimediaFiles();
    List<MultimediaFile> data = temp.ToList();
    data = data.Where(
d => d.id_flight.ToString().Contains(text) ||
d.id_drone.ToString().Contains(text) ||
d.format.ToString().Contains(text)
    || d.creation_date.ToString().Contains(text)
|| d.name.ToString().Contains(text)).ToList();

    ObservableCollection<MultimediaFile> files = new
ObservableCollection<MultimediaFile>();
    foreach(MultimediaFile file in data)
        files.Add(file);
```

Лістинг Ж.4, аркуш 2

```
    return files;
}
private string GetFileIdFromUrl(string fileUrl)
{
    int startIndex = fileUrl.IndexOf("/d/") + 3;
    int endIndex = fileUrl.IndexOf("/view", startIndex);
    if (startIndex >= 0 && endIndex >= 0)
    {
        return fileUrl.Substring(startIndex, endIndex -
startIndex);
    }
    return null;}
public BitmapImage GetImageFromFlights(int id_file)
{
    List<multimedia_file> files =
repository.GetMultimediaFilesList();
    string filename =
GetFileIdFromUrl("https://drive.google.com/file/d/1VN4yDiqoG-
GZx_UgxTNOQzbIh_jhRjcy/view");
    try {
        var request = driveService.Files.Get(filename);
        var file = request.Execute();
        using (MemoryStream stream = new MemoryStream())
        {   request.Download(stream);
            BitmapImage bitmap = new BitmapImage();
            bitmap.BeginInit();
            bitmap.StreamSource = new
MemoryStream(stream.ToArray());
            bitmap.CacheOption = BitmapCacheOption.OnLoad;
            bitmap.EndInit();
            bitmap.Freeze();
            return bitmap;
        }
```

Лістинг Ж.4, аркуш 3

```
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Произошла ошибка при получении
файла: {ex.Message}");
        return null;
    }
}


private bool GetDriveService()
{
    try
{
        using (var stream = new
FileStream("../../Keys/credentials.json", FileMode.Open,
FileAccess.Read))
        {
            var credential = GoogleCredential.FromStream(stream)
                .CreateScoped(Scopes);
            driveService = new DriveService(new
BaseClientService.Initializer()
            {
                HttpClientInitializer = credential,
                ApplicationName = "SwarmApplication",
            });
        }
    }
    catch (Exception ex)
    {
        return false;
    }
    return true;
}
```

Лістинг Ж.4, аркуш 4

```
internal class AuthorizationViewModel : ViewModelBase
{
    private string _username = "Username";
    private DbContextOptions _optionsBuilder;
    private SecureString _password;
    private int _group;
```

Лістинг Ж.5 – Приклад ViewModel

```
    private string _errorMessage;
    private bool _isViewVisible = true;
    private IAuthorizationService service;

    //Properties
    public string Username
    {
        get => _username;
        set { _username = value;
OnPropertyChanged(nameof(Username)); }
    }
    public SecureString Password
    {
        get => _password;
        set { _password = value;
OnPropertyChanged(nameof(Password)); }
    }
    public int Group { get => _group; }
    public string ErrorMessage { get => _errorMessage; set {
_errorMessage = value; OnPropertyChanged(nameof(ErrorMessage));
} }
    public bool IsViewVisible {
get => _isViewVisible;
set { _isViewVisible = value;
OnPropertyChanged(nameof(IsViewVisible)); } }
    public DbContextOptions OptionsBuilder
    {
        get => _optionsBuilder;
        set => _optionsBuilder = value;
    }
    public string Connection
{ get; set; }
    //Commands
    public ICommand LoginCommand { get; }
    public ICommand ExitCommand { get; }
    public ICommand ShowPasswordCommand { get; }
    public object CanExecuteLoginCommand { get; }
    public AuthorizationViewModel(IAuthorizationService service)
    {
        this.service = service;
        ExitCommand = new ViewModelCommand(ExecuteExitCommand);
        LoginCommand = new
ViewModelCommand(ExecuteAuthorizationCommand,
CanExecuteAuthorizationCommand);

    }
    private void ExecuteExitCommand(object obj)
    {
        Application.Current.Shutdown();
    }
```

Лістинг Ж.5, аркуш 2

```
    private void ExecuteAuthorizationCommand(object obj)
    {
        var isValidUser = service.AuthenicateUser(new
NetworkCredential(Username, Password));
        _optionsBuilder = service.OptionsBuilder;
        Connection = service.Connection;

        if (isValidUser)
        {
            _group = service.Group;
            IsViewVisible = false;
        }
        else
        {
            ErrorMessage = "Incorrect information!";
        }
    }

    private bool CanExecuteAuthorizationCommand(object obj)
    {
        bool validData = false;
        if (string.IsNullOrWhiteSpace(Username) ||
Username.Length < 3 || Password == null || Password.Length < 1)
            validData = false;
        else
            validData = true;

        return validData;
    }

}
```

Лістинг Ж.5, аркуш 3

ДОДАТОК К

Список привілеїв ролей на об’єкти БД інформаційної системи

Таблиця К.1 – Список привілеїв ролей на об’єкти БД.

Об’єкти БД	Ролі			
	Адміністратор ІС (admin)	Оператор рою (operator)	AI спеціаліст (ai_specialist)	Дрон (drone)
Таблиці				
Drone	CRUD	R	-	-
Drone_location	-	R	-	CR
Drone_on_the_flight	-	CRUD	-	CR
Drone_parameter	CRUD	R	-	R
Drone_with_method	-	CRUD	-	R
Drone_with_sensors	-	CRUD	-	R
Exclusion_drone	-	CR	-	CR
Exclusion_reason	CRUD	R	-	R
Flight_history	-	CRUD	-	-

Продовження таблиці К.1

Об’єкти БД	Ролі			
	Адміністратор ІС (admin)	Оператор рою (operator)	AI спеціаліст (ai_specialist)	Дрон (drone)
Flight_points	-	CRUD	-	R
Identified_object	-	CRUD	-	C
map	-	-	-	-
Map_versions	-	-	-	-
Method_identification	-	R	CRUD	R
Method_permission	-	R	CRUD	R
Method_version	-	R	CRU	R
Multimedia_file	-	R	-	C
Object_parameter	-	CRUD	-	R
parameters	CRUD	R	-	R
points	-	CRUD	-	CR
Recognition_file	-	CRUD	-	C
sensor	CRUD	R	-	-

Продовження таблиці К.1

Об’єкти БД	Ролі			
	Адміністратор ІС (admin)	Оператор рою (operator)	AI спеціаліст (ai_specialist)	Дрон (drone)
Sensor_parameter	CRUD	R	-	-
Type_of_objects	CRUD	R	-	R
Type_of_drone	CRUD	R	-	-
Type_of_sensors	CRUD	R	-	-
user	CRUD	R	-	-
Debezium_database_history	-	-	-	-
Debezium_offset_storage	-	-	-	-
Представлення				
Show_drones_parameters	R	R	-	-
Show_method_permissions	-	-	-	R
Show_methods_info	-	-	R	-
Show_sensors_parameters	R	R	-	-
Функції				

Продовження таблиці К.1

Об’єкти БД	Ролі			
	Адміністратор ІС (admin)	Оператор рою (operator)	AI спеціаліст (ai_specialist)	Дрон (drone)
Create_sensor	E	-	-	-
Create_drone	E	-	-	-
GetControlPoints	-	-	-	E
GetAvailableMethod	-	E	-	-
CreateNewTask	-	E	-	-
GetSortedPoints	-	E	-	E
Процедури				
AddDroneLocation	-	-	-	E
AddDroneInTask	-	E	-	-
Create_user	E	-	-	-
Update_user	E	-	-	-
Update_user	E	-	-	-
Delete_user	E	-	-	-

ДОДАТОК Л

Створення ролей БД та призначення їм привілеїв

```
--###
-- №1 Адміністратор системи (admin)
--###

-- Процедури
grant select on "user" to "admin";
grant execute on procedure create_user(varchar, text, varchar)
to "admin";
grant execute on procedure update_user(int, varchar, text,
varchar) to "admin";
grant execute on procedure update_user(int, varchar, varchar),
delete_user(int) to "admin";

-- Таблиці
GRANT UPDATE, INSERT, DELETE ON TABLE "user" TO "admin";
grant select, update, insert, delete on parameters to "admin";
grant select, update, insert, delete on exclusion_reason to
"admin";
grant select, insert, update, delete on type_of_drone to
"admin";
grant select, insert, update, delete on type_of_objects to
"admin";
grant select, insert, update, delete on type_of_sensors to
"admin";
grant select, update, delete, insert on sensor to "admin";
grant select, insert, update, delete on sensor_parameter to
"admin";
grant select on show_sensors_parameters to "admin";
grant select on Show_drones_parameters to "admin";
grant select, insert, delete, update on drone_parameter, drone
to "admin";

-- Послідовності
grant usage on type_of_drone_id_type_seq to "admin";
grant  usage on type_of_objects_id_type_of_object_seq,
type_of_sensors_id_type_of_sensor_seq to "admin";
grant usage on sensor_id_sensor_seq to "admin";
grant usage on exclusion_reason_id_reason_seq to "admin";
grant usage on parameters_id_parameter_seq to "admin";
GRANT USAGE ON  user_id_user_seq to  "admin";

-- Функції
grant execute on function create_drone(varchar, varchar,
varchar[], numeric[]) to "admin";
```

Лістинг Л1 – Надання привілеїв для адміністратора системи

```
grant execute on function create_sensor(varchar, varchar,
varchar[], numeric[]) to "admin";
```

Лістинг Л1, аркуш 2

```
--###
-- №2 Оператор рою (operator)
--###

-- Таблиці
grant select on "user" to operator;
grant select on "drone" to operator;
grant select on "type_of_drone", show_sensors_parameters,
show_drones_parameters to operator;
grant select on sensor, sensor_parameter, parameters
to operator;
grant select on type_of_objects to operator;
grant select on type_of_sensors to operator;
grant select on drone_parameter, method_identification,
method_permission to operator;
grant select, insert, update on points,
drone_with_method, drone_with_sensors,
drone_on_the_flight, object_parameter,
    flight_history, flight_points to operator;
grant select on exclusion_reason to operator;
grant select, insert, delete on exclusion_drone to operator;
grant select on drone_location, identified_object to operator;
grant select, insert on identified_object to operator;
grant delete on flight_points, object_parameter to operator;
grant select on multimedia_file to operator;
grant select, insert on recognition_file to operator;
grant select on flight_points to operator;

-- Функції
grant execute on function GetAvailableMethod,
CreateNewTask(double precision[][])
to operator;
grant execute on function GetSortedPoints to operator;

-- Процедури
grant execute on procedure adddroneintask(integer, integer,
integer[], integer[]) to operator;

-- Послідовності
grant usage on drone_on_the_flight_id_drone_on_flight_seq,
drone_with_method_id_drone_with_method_seq to operator;
grant usage on flight_history_id_flight_seq to operator;
grant usage on flight_points_id_flight_point_seq to operator;
```

Лістинг Л2 – Надання привілеїв для оператора системи

```
grant usage on points_id_point_seq to operator;

grant usage on object_parameter_id_object_parameter_seq to
operator;
grant usage on exclusion_drone_id_exclusion_seq to operator;
grant usage on identified_object_id_identified_object_seq to
operator;
```

Лістинг Л2, аркуш 2

```
--###
-- №3 AI спеціаліст(ai_specialist)
--###
-- Таблиці
grant select, insert, delete, update on method_identification,
method_version to ai_specialist;
grant select on show_methods_info to ai_specialist;
grant select on parameters to ai_specialist;
grant select, insert, update, delete on method_permission to
ai_specialist;

-- Послідовності
grant usage on method_identification_id_method_seq,
method_version_id_method_version_seq to ai_specialist;
grant usage on method_permission_id_method_permission_seq to
ai_specialist;
```

Лістинг Л3 – Надання привілеїв для AI-спеціаліста

```
--###
-- №4 Дрони(drone)
--###


-- Таблиці
grant select on "user" to drone;
grant select on flight_points to drone;
grant select, insert on points to drone;
grant select on drone_on_the_flight to drone;
grant select, insert on drone_location to drone;
grant select on flight_history to drone;
grant select on drone to drone;
grant select on exclusion_reason to drone;
grant select, insert, delete on exclusion_drone to drone; grant
select on drone_with_method, method_identification,
```

Лістинг Л4 – Надання привілеїв для дронів

```
method_permission, type_of_objects, object_parameter, parameters
to drone;


-- Процедури
grant execute on procedure AddDroneLocation to drone;
grant execute on procedure adddronelocation to drone;

-- Функції
grant execute on function GetControlPoints to drone;
grant execute on function GetSortedPoints to drone;

-- Послідовності
grant usage on drone_location_id_location_seq to drone;
grant usage on points_id_point_seq to drone;
grant usage on exclusion_drone_id_exclusion_seq to drone;
```

Лістинг Л4, аркуш 2