ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«Система управління ройовим комплексом.
Реконфігурація структури комплексу на основі мультиагентної технології»

(тема дипломної роботи українською мовою)

«Swarm complex control system.
Complex structure reconfiguration based on multi-agent technology»

(тема дипломної роботи англійською мовою)

Виконав: здобувач ВО денної форми навчання
спеціальності ___123 – Комп'ютерна інженерія_____
(код, назва спеціальності)
Освітня програма _____Комп'ютерна інженерія_____
(назва)
Куликов Вадим Володимирович
(прізвище, ім'я, по-батькові здобувача)
Керівник _к. ф-м. н., доц. Шпінарева І. М._____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)
Рецензент ___д.т.н., проф. Малахов Є. В._____
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ ___ від ____.____. 2025 р.

Завідувач кафедри
_____ Євгеній МАЛАХОВ
(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.2025 р.

Оцінка_____/_____/_____
(за національною шкалою/шкалою ECTS/ бали)
Голова ЕК
_____ Світлана АНТОЩУК
(підпис) (прізвище, ім'я)

Одеса 2025

АНОТАЦІЯ

Метою даної роботи є розробка системи управління ройовим комплексом на основі мультиагентної технології, що забезпечує ефективне розв’язання оперативних задач і координацію між агентами (дронами та операторами) через стандартизовану взаємодію мікросервісів. Для досягнення поставленої мети проведено аналіз предметної області, визначено ключові задачі системи, зокрема автентифікацію користувачів, обробку відеофайлів, та моніторинг стану дронів. Розроблено мікросервісну архітектуру, яка включає сервіси AuthService, CommandManagerService, VideoProcessorService, AIService та HealthService, із стандартизованою взаємодією через OpenAPI. Спроектовано розподілену базу даних із використанням MongoDB для автентифікаційних даних і PostgreSQL для операційних даних, що забезпечує ізольоване зберігання та швидкий доступ до інформації.

Особливістю системи є мультиагентний підхід, який підтримує координацію між агентами (дронами та операторами) через чітко визначені API, дозволяючи обробляти оперативні задачі в реальному часі. Основними функціями системи є авторизація користувачів, обробка відеофайлів, аналіз даних ШІ, а також моніторинг стану й історії дронів. Реалізація виконана з використанням мови програмування JavaScript (Node.js) для серверної логіки, бібліотеки Express для маршрутизації, MongoDB і PostgreSQL для баз даних, а також JSON Web Tokens для автентифікації. Архітектура системи побудована на принципах мікросервісів із стандартизацією через OpenAPI.

Результатом роботи є система управління ройовим комплексом, яка забезпечує ефективну координацію між агентами, зручний доступ до даних і функціонал для оперативного управління завданнями рою.

ABSTRACT

The purpose of this qualification work is to develop a swarm complex management system based on multi-agent technology, ensuring efficient resolution of operational tasks and coordination among agents (drones and operators) through standardized microservice interactions. To achieve this goal, an analysis of the subject area was conducted, identifying key system tasks, including user authentication, video file processing, AI-driven data analysis, and drone status monitoring. A distributed database was designed using MongoDB for authentication data and PostgreSQL for operational data, providing isolated storage and fast access to information. A microservice architecture was developed, incorporating services such as AuthService, CommandManagerService, VideoProcessorService, AIService, and HealthService, with standardized interaction via OpenAPI.

A distinctive feature of the system is the multi-agent approach, which facilitates coordination among agents (drones and operators) through well-defined APIs, enabling real-time handling of operational tasks. The main functionalities of the system include user authorization, video file processing, AI data analysis, and monitoring of drone status and history. The implementation was carried out using the JavaScript programming language (Node.js) for server-side logic, the Express library for routing, MongoDB and PostgreSQL for databases, and JSON Web Tokens for authentication. The system architecture is based on microservice principles with standardization through OpenAPI.

The result of the work is a swarm complex management system that ensures effective coordination among agents, convenient access to data, and functionality for operational task management.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface – набір засобів для взаємодії між програмними компонентами

JWT – JSON Web Token – відкритий стандарт для безпечної передачі інформації між сторонами у вигляді JSON-об’єкта

OpenAPI Generator – інструмент, що автоматично генерує клієнтські бібліотеки, серверні шаблони, документацію та конфігураційні файли з опису API у форматі OpenAPI (раніше Swagger)

REST API – Representational State Transfer Application Programming Interface – інтерфейс прикладного програмування, що ґрунтується на принципах REST і дозволяє взаємодію між клієнтом і сервером за допомогою HTTP-запитів

YAML – YAML Ain’t Markup Language – формат серіалізації даних, зручний для читання людиною; часто використовується для конфігураційних файлів

НС – надзвичайні ситуації

ВСТУП

Надзвичайні ситуації, такі як стихійні лиха чи техногенні аварії, створюють загрози для життя і вимагають оперативного реагування. Зростання їхньої складності підвищує навантаження на рятувальні служби, що робить традиційні методи пошуку неефективними в зонах з обмеженим доступом. В мультиагентних технологіях рішення виходить автоматично в результаті взаємодії безлічі самостійних цілеспрямованих програмних модулів – агентів. Рої дронів забезпечують сканування території та координацію дій, де запропоновано багатошарову модель для аварійних зон. Однак одинокі дрони мають обмежені ресурси, що вирішується ройовими комплексами з координацією через оператора [1]. Для ефективного управління роєм потрібна система з обробкою даних і моніторингом у реальному часі. Мікросервісна архітектура дозволяє стандартизувати взаємодію, обробку даних і моніторинг стану дронів [2, 3].

Мікросервісна архітектура є оптимальним рішенням для такої системи, оскільки дозволяє розбити функціонал на автономні компоненти, кожен із яких відповідає за окрему задачу: автентифікацію, обробку мультимедійних даних, аналіз, координацію чи моніторинг. Такий підхід забезпечує гнучкість, стійкість до збоїв завдяки локалізації проблем і можливість масштабування. На відміну від монолітної архітектури, яка обмежує адаптивність і ускладнює розширення, мікросервіси дозволяють розвивати окремі модулі незалежно один від одного. Кожен мікросервіс може використовувати окрему базу даних для зберігання специфічних даних, що підвищує відмовостійкість і спрощує інтеграцію нових функцій, адаптуючи систему до різноманітних сценаріїв надзвичайних ситуацій.

Безпека є критично важливим аспектом системи, оскільки вона обробляє чутливі дані, такі як мультимедійні файли чи стани дронів, які можуть бути об’єктом кібератак. Для захисту передбачається автентифікація користувачів за допомогою JWT-токенів і розмежування доступу на основі ролей, що забезпечує контроль над діями операторів і захист інформації.

Актуальність роботи зумовлена необхідністю підвищення ефективності реагування на НС шляхом автоматизації управління роєм дронів.

Метою даної роботи є розробка системи управління ройовим комплексом на основі мультиагентної технології, що забезпечує ефективне розв’язання оперативних задач і координацію між агентами (дронами та операторами) через стандартизовану взаємодію мікросервісів.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1) проаналізувати предметну область, зокрема особливості управління роєм дронів у надзвичайній ситуації;

2) обрати та обґрунтувати мікросервісну архітектуру для системи управління ройовим комплексом;

3) спроєктувати мікросервіси, включаючи їхні API та бази даних;

4) розробити та реалізувати мікросервіси з використанням Node.js, Express.js і OpenAPI Generator;

5) забезпечити стандартизовану взаємодію між мікросервісами через REST API;

6) реалізувати механізми автентифікації та розмежування доступу для захисту системи;

7) протестувати взаємодію мікросервісів у сценаріях НС, таких як моніторинг стану дронів чи обробка мультимедійних даних.

1 ПОСТАНОВКА ЗАДАЧІ СИСТЕМА УПРАВЛІННЯ РОЄМ ДРОНІВ НА ОСНОВІ МУЛЬТИАГЕНТНОЇ ТЕХНОЛОГІЇ

1.1 Введення у предметну область мультиагентних технологій

Суть мультиагентних технологій полягає в принципово новому методі вирішення завдань. На відміну від класичного способу, коли проводиться пошук деякого чітко визначеного (детермінованого) алгоритму, що дозволяє знайти найкраще вирішення проблеми, в мультиагентних технологіях рішення виходить автоматично в результаті взаємодії безлічі самостійних цілеспрямованих програмних модулів – так званих агентів. Мультиагентні технології включають як розробку та використання мультиагентних систем, так і мультиагентне керування. Система, у якій декілька агентів можуть спілкуватися, передавати одне одному деяку інформацію, взаємодіяти між собою і розв’язувати поставлене завдання, називається мультиагентною системою [4].

Агент – це самостійна програмна система, котра:

– має можливість приймати вплив із зовнішнього світу;

– визначає свою реакцію на цей вплив і формує відповідну дію;

– змінює свою поведінку з плином часу в залежності від накопиченої інформації і отриманих з неї знань;

– володіє мотивацією і здатна після делегування повноважень користувачем поставити себе на його місце і прийняти рішення, відповідне ситуації.

Агент повинен мати наступні властивості:

– автономність – здатність функціонувати без втручання з боку свого власника і здійснювати контроль внутрішнього стану і своїх дій;

– коллаборативного – агент може взаємодіяти з іншими агентами декількома способами, граючи різні ролі;

– здатність до міркувань – агенти можуть володіти частковими знаннями

або механізмами висновків, а також спеціалізуватися на конкретну предметну область;

– мобільність – здатність передачі коду агента з одного сервера на інший;

– соціальну поведінку – можливість взаємодії і комунікації з іншими агентами;

– комунікативність – агенти можуть спілкуватися з іншими агентами;

– реактивність – адекватне сприйняття середовища і відповідні реакції на її зміни [4].

Ройовий інтелект – це мультиагентна система, яка характеризується самоорганізованою поведінкою, яка виявляє деяку розумну поведінку. Кожен агент у рої діє автономно, спираючись на правила взаємодії, але завдяки колективній комунікації система демонструє ефективність у вирішенні завдань.

Для ефективного функціонування рою дронів у НС, таких як стихійні лиха, техногенні аварії чи пошуково-рятувальні операції, необхідно створити розподілену інформаційну систему, яка забезпечує координацію, обробку даних і адаптацію до змін у реальному часі. Використання мікросервісної архітектури дозволить розбити систему на окремі компоненти з чітко визначеною функцією та забезпечить стійкість до збоїв, можливість масштабування та стандартизований спосіб обміну інформацією між компонентами.

1.2 Огляд систем для підтримки ройових комплексів

Аналіз існуючих рішень для управління роєм дронів є важливим для оцінки ефективності пропонованої мікросервісної інформаційної системи. Порівняння з аналогами дозволяє визначити їхні функціональні можливості, обмеження та обґрунтувати переваги розроблюваної системи для координації рою, обробки даних і забезпечення безпеки в надзвичайних ситуаціях.

У цьому підрозділі розглянуто три популярні платформи для управління дронами та роями: FlytBase, DJI FlightHub і DroneDeploy, проаналізовано їхні особливості та проведено порівняння з пропонованою системою.

1.2.1 FlytBase

FlytBase – хмарна платформа для автоматизації та управління роями дронів [5], орієнтована на комерційні та промислові сценарії, такі як логістика чи інспекції (рис. 1.1). Вона забезпечує централізоване керування флотом через вебпанель, дозволяючи відстежувати місцезнаходження дронів у реальному часі. Платформа підтримує автономні польоти поза візуальною видимістю (BVLOS), автоматизоване планування маршрутів із урахуванням зон обмеження та потокове відео з камер дронів, що корисно для моніторингу. FlytBase пропонує API для інтеграції з корпоративними системами, що забезпечує сумісність із різними моделями дронів. Хмарна архітектура сприяє віддаленому доступу та масштабуванню, що робить платформу гнучкою для великих операцій. Однак FlytBase більше зосереджена на комерційних задачах, ніж на специфічних вимогах НС, таких як розпізнавання об’єктів у реальному часі. Відсутність вбудованих модулів для аналізу мультимедійних даних потребує додаткових інтеграцій, а залежність від стабільного інтернет-з’єднання може ускладнити роботу в віддалених зонах НС.

Рисунок 1.1 – Хмарна платформа FlytBase

1.2.2 DJI FlightHub

DJI FlightHub розроблена компанією DJI, є хмарною платформою для управління флотом дронів, переважно сумісних із продукцією DJI (рис. 1.2). Вона пропонує інтерактивну карту для моніторингу місцезнаходження дронів, планування маршрутів із урахуванням зон обмеження та потокове відео з камер, що забезпечує оперативне реагування. Платформа зберігає історію польотів, дозволяючи аналізувати дані для оптимізації майбутніх місій. Хмарна архітектура забезпечує доступ до даних із будь-якого місця з інтернет-з'єднанням, а централізований інтерфейс спрощує керування великими флотами. Проте обмежена сумісність із дронами інших виробників знижує гнучкість для гетерогенних роїв. Відсутність інструментів для розпізнавання об'єктів чи аналізу даних потребує додаткових програмних рішень, що може бути витратним. Безкоштовна версія платформи має суттєві обмеження, наприклад, до п'яти польотів, а професійна версія є платною, що може бути недоцільним для бюджетних проєктів [6].

Рисунок 1.2 – Хмарна платформа DJI FlightHub

1.2.3 DroneDeploy

DroneDeploy – хмарна платформа, орієнтована на картографування та аналіз даних, із застосуванням у будівництві, сільському господарстві та енергетиці (рис. 1.3). Вона підтримує автоматизоване планування польотів, обробку мультимедійних даних у хмарі та створення 2D-карт, 3D-моделей і ортофотопланів. Платформа забезпечує потокове відео та моніторинг у реальному часі, а також інтеграцію з корпоративними системами, такими як Autodesk чи Procore, що спрощує обмін даними. DroneDeploy сумісна з дронами DJI та деякими іншими моделями, що надає певну гнучкість. Хмарна обробка даних зменшує потребу в локальному обладнанні, що зручно для великих обсягів інформації. Однак платформа зосереджена на картографуванні, а не на координації рою чи управлінні в реальному часі, що обмежує її застосування в НС. Відсутність спеціалізованих модулів для розпізнавання об’єктів і обмежена підтримка гетерогенних роїв знижують її ефективність для складних сценаріїв [7].

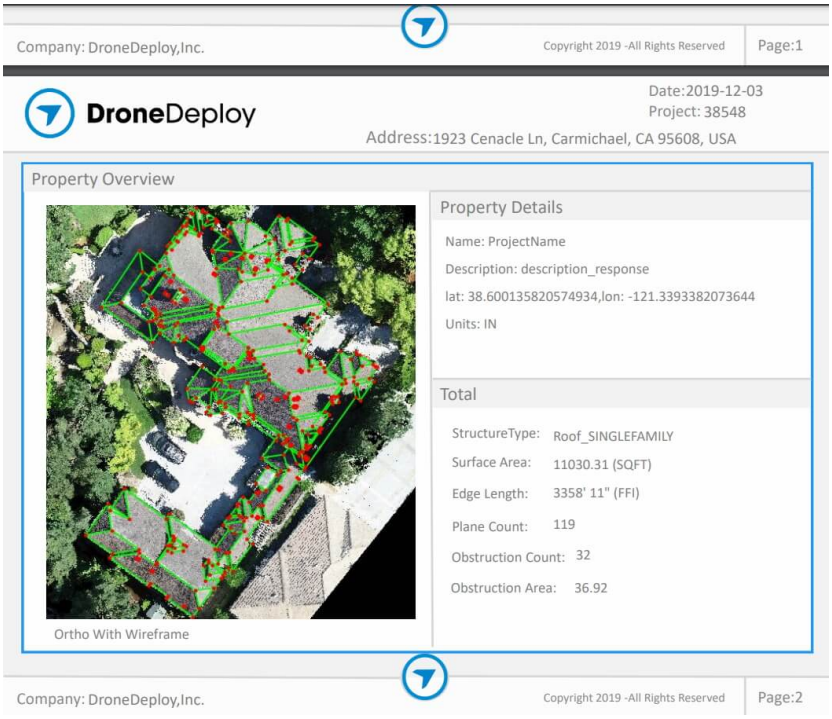


Рисунок 1.3 – Хмарна платформа DroneDeploy

1.2.4 Огляд дронів

Дрони в системі управління роєм представляють основний виконавчий елемент, який виконує завдання, такі як моніторинг, доставка чи збір даних. Кожен дрон ідентифікується унікальним ідентифікатором, який дозволяє відстежувати його діяльність у межах рою. Для ефективного управління дронами система покладається на дані, отримані від вбудованих датчиків, які відображають технічний стан пристрою. Зокрема, датчики вимірюють рівень заряду батареї (у відсотках від 0 до 100), стан камери (ввімкнена чи вимкнена) та статус зв’язку (активний чи втрачений). Ці дані регулярно оновлюються з фіксацією часу останнього оновлення, що забезпечує актуальність інформації для координації рою.

Аналіз стану дронів здійснюється на основі показників, зібраних датчиками, які передають інформацію про ключові параметри, такі як енергетичний рівень, функціональність камери та стабільність зв’язку. Такий підхід дозволяє системі виявляти потенційні збої, наприклад, низький заряд батареї чи втрату зв’язку, а також оптимізувати використання дронів у реальному часі, що є критично важливим для надійного функціонування ройової системи. Для наочного представлення характеристик дрона, які відстежуються датчиками, наведено схему (рис. 1.4).

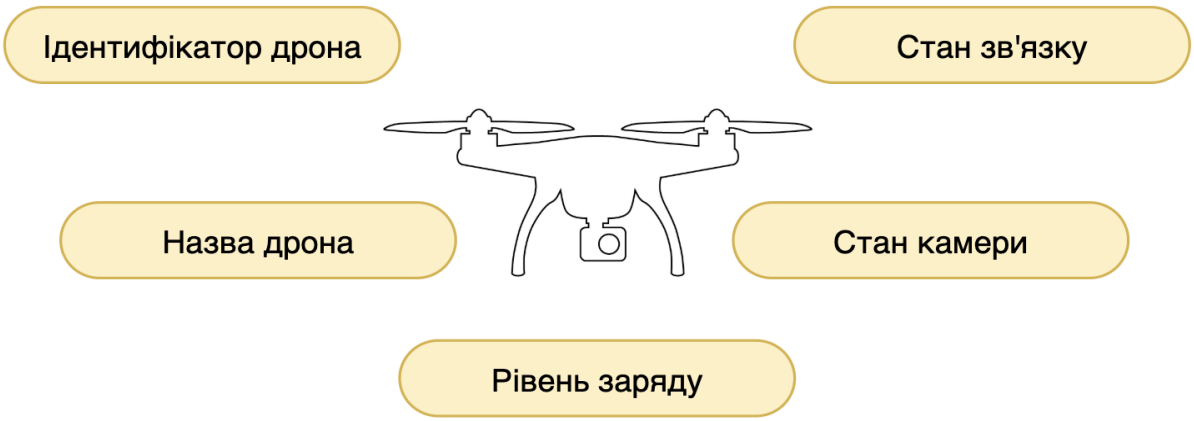


Рисунок 1.4 – Схематичне зображення дрону та його показників

Для реалізації апаратної частини дронів обрано платформу Raspberry Pi Zero 2W [8], яка забезпечує компактність (65×30 мм) і достатню обчислювальну потужність завдяки чотириядерному процесору Broadcom BCM2710A1 (Cortex-A53, 1 ГГц) та 512 МБ оперативної пам’яті LPDDR2 SDRAM. Вбудований модуль Wi-Fi (2.4 ГГц, IEEE 802.11b/g/n) підтримує бездротову взаємодію з іншими дронами та наземними сервісами, що відповідає вимогам децентралізованого обміну даними. Енергоефективність платформи (0,3–1 Вт) дозволяє тривалу автономну роботу за умови використання акумулятора.

Для відеозйомки інтегровано камеру OV5647 [8], підключену через CSI-інтерфейс, яка забезпечує роздільну здатність 5 Мп і запис у форматі 1080p при 30 кадрах на секунду. Ця камера є ключовим елементом для передачі відеопотоків, які обробляються мікросервісом VideoProcessor. Інтеграцію платформи Raspberry Pi Zero 2 W із камерою OV5647 у конструкцію дрона проілюстровано на рисунку 1.5.

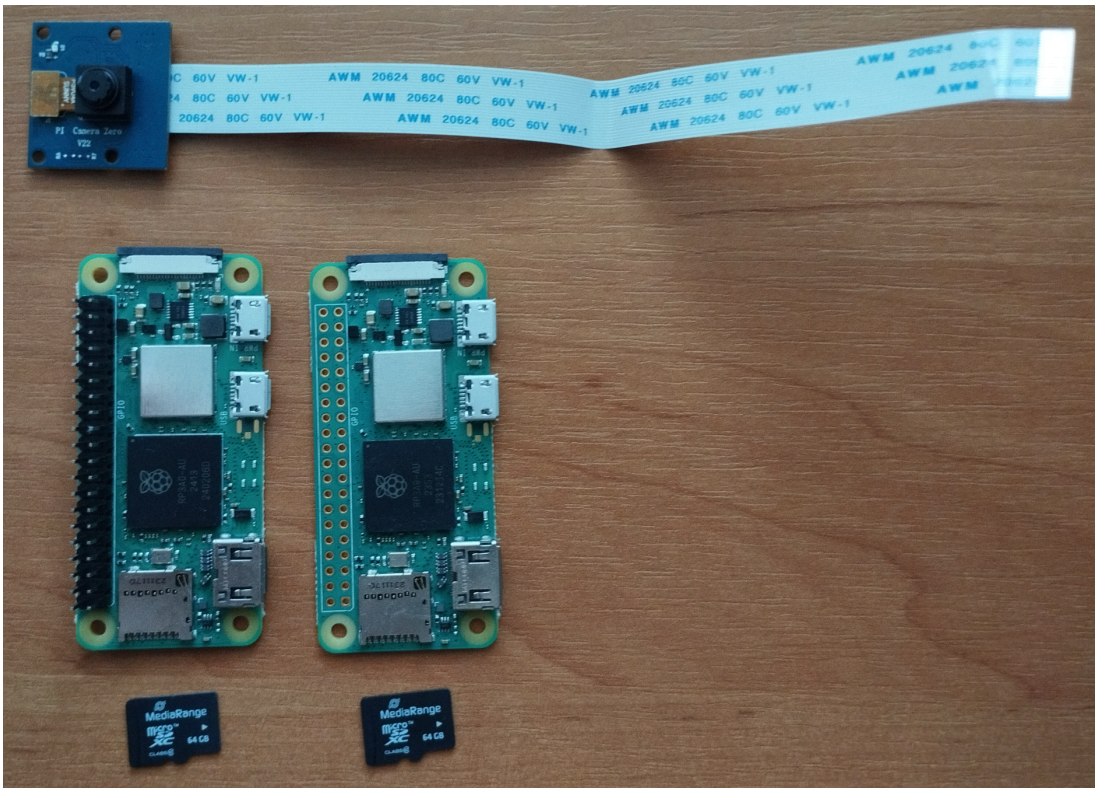


Рисунок 1.5 – Інтеграція Raspberry Pi Zero 2 W і камери OV5647 у дрон

Для оцінки доцільності вибору Raspberry Pi Zero 2 W порівняно з іншими платформами проведено аналіз за ключовими критеріями, важливими для дронів у ройових системах. Результати представлено на рисунку 1.6, де кожна платформа оцінена за п’ятьма параметрами: компактність, ціна енергоефективність, продуктивність, підтримка камери за шкалою від 1 до 5.

Платформа	Компактність	Енергоефективність	Продуктивність	Ціна	Підтримка камери	Загальна оцінка
Raspberry Pi Zero 2 W	5	5	4	5	5	**24**
Arduino Nano	4	4	2	4	1	**15**
BeagleBone Black	2	2	3	3	3	**13**
NVIDIA Jetson Nano	3	1	5	2	4	**15**

Рисунок 1.6 – Оцінка апаратних платформ для ройових комплексів

Raspberry Pi Zero 2W отримує найвищу оцінку завдяки компактним розмірам, низькому енергоспоживанню, достатній продуктивності для реальних завдань, доступній ціні та повній підтримці камери через CSI-інтерфейс. Це робить її оптимальним вибором для дронів у рої.

Arduino Nano вирізняється компактністю та енергоефективністю, але низька продуктивність і відсутність підтримки камери обмежують його використання. BeagleBone Black має середню продуктивність, але більші розміри та вищі енергетичні витрати знижують його придатність для дронів. NVIDIA Jetson Nano демонструє високу обчислювальну потужність, але високе енергоспоживання та ціна роблять його менш практичним для масового використання в ройових системах.

Отже, апаратна платформа Raspberry Pi Zero 2W із камерою OV5647 забезпечує необхідний баланс характеристик для реалізації дронів у ройовому комплексі, відповідаючи вимогам до автономності, легкості та ефективності роботи.

1.2.5 Аналіз аналогів: висновок

Проведений огляд свідчить, що FlytBase, DJI FlightHub і DroneDeploy пропонують потужні інструменти для управління дронами, але мають обмеження для задач управління роєм у НС.

FlytBase підтримує автономні операції та інтеграцію, але не забезпечує аналізу даних і залежить від інтернет-з'єднання. DJI FlightHub зручна для моніторингу та потокового відео, але обмежена сумісністю з дронами DJI і не підтримує розпізнавання об'єктів. DroneDeploy ефективна для картографування, але не орієнтована на координацію рою чи обробку даних у реальному часі.

На основі вище розглянутого можна провести порівняння можливостей систем FlytBase, DJI FlightHub і DroneDeploy та визначити функціонал для системи, що розробляється. Таблиця 1.1 показує, які задачі підтримуються аналогами та чи забезпечують вони аналогічний функціонал.

Таблиця 1.1 – Порівняння задач системи з можливостями аналогів

№	Задача системи	FlytBase	DJI FlightHub	DroneDeploy
1	Автентифікація та авторизація користувачів	Так	Частково	Частково
2	Керування обробкою мультимедійних файлів	Частково	Частково	Так
3	Аналіз мультимедійних даних для розпізнавання об'єктів	Ні	Ні	Ні
4	Координація дій рою дронів	Так	Частково	Ні
5	Моніторинг технічного стану дронів	Частково	Так	Ні
6	Створення REST API для взаємодії	Так	Так	Так

FlytBase підтримує автентифікацію через API, координацію рою та REST API, однак її можливості з обробки мультимедійних файлів обмежені, а розпізнавання об’єктів відсутнє. Моніторинг технічного стану зосереджений переважно на відстеженні місцезнаходження. DJI FlightHub забезпечує базову автентифікацію, моніторинг стану дронів і REST API, але обробка мультимедійних файлів обмежується потоковим відео, а координація рою є частковою через сумісність лише з DJI-дронами; функція розпізнавання об’єктів не підтримується. DroneDeploy ефективно обробляє мультимедійні дані для картографування та підтримує REST API, але не забезпечує координацію роїв, моніторинг технічного стану чи розпізнавання об’єктів; автентифікація реалізована частково через корпоративні інтеграції.

Порівняння свідчить, що жоден із аналогів не забезпечує повного спектру задач, необхідних для управління роєм дронів у НС. Пропонована система, завдяки мікросервісній архітектурі повинна інтегрувати всі перелічені функції, зокрема розпізнавання об’єктів і моніторинг стану дронів, що робить її унікальним рішенням для оперативного реагування в надзвичайних ситуаціях.

1.3 Постановка задачі

Предметною областю даної роботи є управління роєм дронів у НС, таких як стихійні лиха, техногенні аварії чи пошуково-рятувальні операції. Ройовий комплекс виконує ряд задач, що включають координацію польотів, обробку мультимедійних даних, розпізнавання об’єктів (наприклад, постраждалих чи перешкод), моніторинг технічного стану дронів і забезпечення безпеки доступу до системи. Система буде побудована на мікросервісній архітектурі, яка повинна забезпечити ефективну взаємодію між дронами, оператором і серверними компонентами в реальному часі. Усі мікросервіси відносяться до типу агентів мікросервісів, які пов'язані з певними програмними сутностями у

системі. Важливою характеристикою цього виду агентів є можливість масштабування числа екземплярів зі збільшенням навантаження.

У рамках даної роботи основна увага приділяється розробці мікросервісної системи управління роєм дронів, яка забезпечує стандартизовану взаємодію між компонентами для координації дій рою в НС.

На основі аналізу предметної області сформульовано основні завдання, які має вирішувати система.

1) Забезпечення автентифікації та авторизації користувачів для безпечного доступу до системи, зокрема генерація та перевірка JWT-токенів.

2) Керування обробкою мультимедійних файлів, отриманих від дронів, включаючи їхнє завантаження, зміну статусів і передачу для аналізу.

3) Обробка мультимедійних даних для імітації розпізнавання об’єктів і формування структурованих результатів для оператора.

4) Імітація координації дій рою дронів шляхом надсилання команд у базу даних і отримання попередньо підготовлених результатів обробки даних.

5) Моніторинг технічного стану дронів, виявлення несправностей (наприклад, відмова камери, низький заряд батареї, збій зв’язку) і передача даних для реагування.

6) Створення REST API для стандартизованої взаємодії між мікросервісами та дронами.

Система призначена для взаємодії з оператором, який керує роєм, переглядає мультимедійні дані, результати аналізу та статуси дронів, а також із дронами, які надсилають інформацію про своє положення, технічний стан і мультимедійні файли, виконуючи отримані команди.

Якщо розглядати оператора, рій дронів і систему як єдиний комплекс, можна виділити ключові задачі кожного елемента. Оператор виконує такі функції:

1) підтвердження особи через сервіс управління доступом для входу в систему управління роєвим комплексом;

2) моніторинг стану дронів за допомогою сервісу діагностики;

3) імітація обробки результатів операцій, отриманих від модуля аналізу та сервісу обробки відеоданих, для прийняття рішень;

4) надсилання команд через сервіс координації для управління роєм, наприклад, зміни маршрутів.

Рій дронів відповідає за:

1) збір і передачу мультимедійних даних до сервісу обробку відеоданих;

2) повідомлення про технічний стан до сервісу діагностики, наприклад, про несправність камери;

3) адаптацію до змін у формуванні рою на основі команд від сервісу управління роєвим комплексом.

Отже, система має надавати такі можливості:

1) безпечний доступ користувачів до функцій через автентифікацію та розмежування ролей;

2) обробку мультимедійних даних із підтримкою пріоритетів для ефективного аналізу;

3) моніторинг технічного стану дронів із реакцією на несправності;

4) збереження історії операцій для оцінки ефективності;

5) стандартизовану взаємодію між мікросервісами через API.

2 ПРОЕКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ РОЙОВИМ КОМПЛЕКСОМ

2.1 Архітектура системи

Для виконання всіх функцій, система буде включати п’ять агентів мікросервісів: AuthService для автентифікації, авторизації користувачів та верифікації запитів в обміні інформацією між всіма сервісами, VideoProcessorService для обробки та збереження мультимедійних даних, AIService для аналізу даних, CommandManagerService для координації рою та HealthService для моніторингу технічного стану дронів. Кожен мікросервіс використовує окрему базу даних для зберігання даних, таких як журнали операцій, мультимедійні файли чи стани дронів (рис. 2.1).

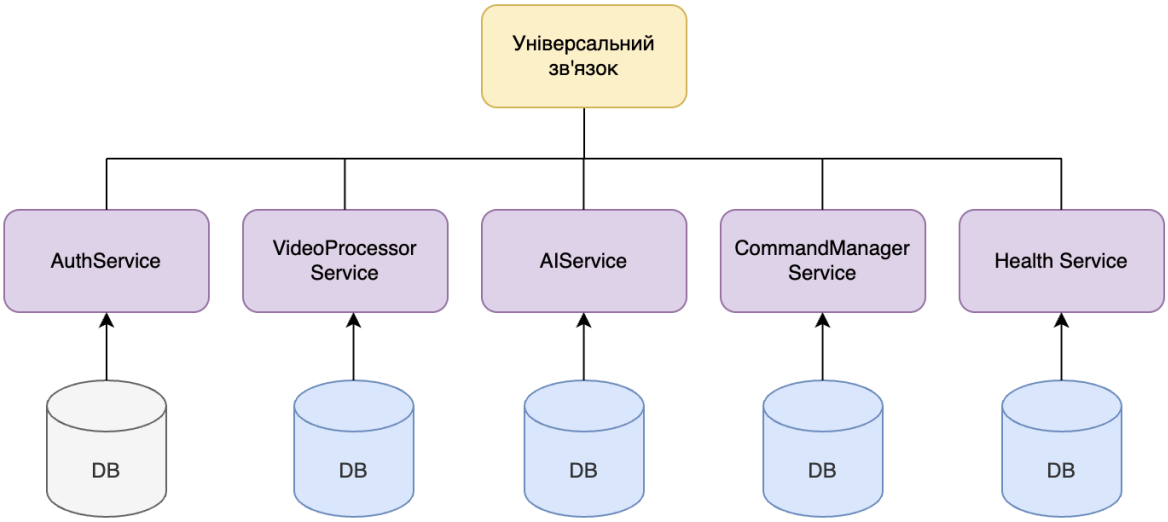


Рисунок 2.1 – Загальна структура мікросервісів

AuthService – це важлива частина системи, яка займається тим, щоб усі користувачі могли безпечно увійти до системи та отримати доступ до її функцій. Його основна робота – перевіряти, хто хоче скористатися системою, і давати дозвіл лише тим, хто має право. Цей сервіс дозволяє новим

користувачам зареєструватися, надаючи їхню електронну адресу, пароль і роль (наприклад, оператор чи адміністратор). Після реєстрації він зберігає ці дані в окремому сховищі.

Коли користувач намагається входити, AuthService буде перевіряти, чи правильні його електронна адреса та пароль, і якщо все гаразд, створює спеціальний ключ (токен), який слугує перепусткою для доступу до інших частин системи. Цей ключ діє протягом певного часу і видається користувачу, щоб він міг працювати без повторного введення даних. Також сервіс може перевірити, чи дійсний цей ключ, коли хтось намагається щось зробити в системі, і сказати, чи має користувач право на певні дії залежно від його ролі.

Цей сервіс допомагає тримати систему в безпеці, стежить за тим, щоб усі дії були дозволені легітимному користувачу, і передає інформацію про користувачів (наприклад, їхні ідентифікатори та ролі) іншим сервісам, щоб вони знали, з ким працюють.

Нижче наведено таблицю 2.1, яка показує, які дії виконує AuthService, що йому потрібно для роботи (входи) і що повертає (виходи).

Таблиця 2.1 – Задачі мікросервісу AuthService

Функція	Вхідні дані	Вихідні дані
Реєстрація	Електронна адреса, пароль, роль	ідентифікатор, електронна адреса, роль
Авторизація	Електронна адреса, пароль	Ключ доступу (токен), ідентифікатор, електронна адреса, роль
Верифікація	Ключ доступу (токен)	Інформація про валідність ключа, ідентифікатор, електронна адреса, роль

Для AuthService потрібна окрема база даних (табл. 2.2), щоб надійно зберігати інформацію про користувачів і забезпечувати безпеку системи.

Унікальний ідентифікатор дозволяє швидко знаходити профіль користувача, а електронна адреса слугує основним ключем для авторизації. Пароль зберігається в зашифрованому вигляді, щоб захистити його від несанкціонованого доступу, а поле ролі визначає права користувача, що важливо для контролю доступу до різних функцій системи. Така структура дозволяє ефективно перевіряти автентифікаційні дані, видавати токени та передавати інформацію про користувачів іншим сервісам, зберігаючи їхню ізоляцію та безпеку.

Таблиця 2.2 – Структура таблиці «User»

User	
Поле	Опис
Ідентифікатор	Унікальний ідентифікатор користувача для розпізнавання в системі
Електронна адреса	Унікальна електронна адреса користувача, що використовується для входу
Пароль	Зберігається в зашифрованому вигляді для забезпечення безпеки
Роль	Визначає рівень доступу (оператор, адміністратор)

VideoProcessorService забезпечує управління відеофайлами, що надходять від дронів. Основне завдання – приймати файли, відстежувати їхній стан і передавати на подальший аналіз у потрібний момент. Сервіс дозволяє переглядати список усіх відеофайлів у системі та визначати, які з них ще очікують на обробку. Функціонал також передбачає позначення файлу як «обробляється» під час початку роботи оператором і зміну стану на «завершено» після закінчення обробки. Після завершення обробки сервіс надає детальну інформацію про файл, наприклад, до якого дрона належить цей файл і в якому стані перебуває. Така інформація допомагає іншим частинам системи зрозуміти поточний стан файлу та передати його для аналізу, якщо все

готово. Усі дії фіксуються в спеціальному журналі, що дозволяє відстежувати, які зміни відбувалися з кожним файлом і коли саме. Дані про файли, їхні стани та журнали зберігаються в окремому сховищі.

Сервіс відіграє ключову роль у правильній організації відеоматеріалів і підготовці їх до аналізу, забезпечуючи безперебійну роботу системи в реальному часі. Нижче наведено таблицю 2.3, яка показує, які дії виконує VideoProcessorService, що йому потрібно для роботи і що повертає.

Таблиця 2.3 – Задачі мікросервісу VideoProcessorService

Функція	Вхідні дані	Вихідні дані
Показати всі відеофайли	-	Список усіх відеофайлів із їхніми деталями
Показати файли на обробку	-	Список файлів, які чекають на обробку
Почати обробку файлу	Ідентифікатор файлу	Повідомлення, що файл позначено як обробляється
Завершити обробку файлу	Ідентифікатор файлу	Повідомлення, що файл позначено як завершено
Отримати інформація про файл	Ідентифікатор файлу	Дані про файл

Для VideoProcessorService потрібна окрема база даних, щоб ефективно керувати відеофайлами та їхнім станом. В таблиці «Файли» (табл. 2.4) зберігаються метадані про кожен файл, включаючи походження (дрон), тип, географічні координати та стан обробки, що забезпечує швидкий доступ до інформації для планування аналізу. Поле пріоритету допомагає визначати порядок обробки, а стан відображає поточний етап роботи з файлом.

Таблиця «Журнали обробки файлів» (табл. 2.5) слугує для відстеження історії змін, фіксуючи дії та час їх виконання, що важливо для моніторингу та діагностики. Така структура гарантує ізоляцію даних, полегшує їхню організацію та підтримує безперебійну роботу сервісу в реальному часі.

Таблиця 2.4 – Структура таблиці «Files»

Files	
Поле	Опис
Ідентифікатор	Унікальний ідентифікатор кожного файлу для розпізнавання
Ідентифікатор дрона	Посилання на дрон, від якого надійшов файл
Назва файлу	Ім’я файлу для ідентифікації в системі
Тип файлу	Вказівка, чи це відео, чи фото
Дата створення	Час, коли файл був доданий до системи
Широта	Географічна координата широти
Довгота	Географічна координата довготи
Час захоплення	Час, коли файл був створений дроном
Стан	Поточний стан файлу

Таблиця 2.5 – Структура таблиці «File_logs»

File_logs	
Поле	Опис
Ідентифікатор	Унікальний ідентифікатор запису в журналі
Ідентифікатор файлу	Посилання на файл, до якого відноситься запис
Дія	Опис дії, виконаної з файлом
Час виконання	Час, коли дія була виконана
Опис	Додаткові деталі про дію

AIService займається перевіркою відеофайлів, отриманих від дронів, щоб знаходити важливі деталі, такі як люди, які потребують допомоги, або перешкоди на шляху.

Основна роль сервісу – отримувати файли, аналізувати їхній вміст і передавати результати тим, хто керує дронами або роєм. Цей сервіс дозволяє починати перевірку файлу, вказавши, звідки він узятий і до якого дрона належить, а потім зберігає інформацію про те, що виявлено.

Сервіс також дає змогу переглядати результати аналізу в будь-який момент, показуючи, що знайдено в файлі, чи завершена перевірка, і чи виникли якісь проблеми. Усі ці дані зберігаються в окремому сховищі, де фіксуються деталі про кожен аналіз, включаючи час початку та оновлення. Перед початком роботи сервіс перевіряє, чи файл підходить для аналізу, і переконується, що до нього є доступ через спеціальний дозвіл.

Цей сервіс відіграє важливу роль у допомозі операторам і рою дронів швидко реагувати на ситуацію, надаючи чіткі висновки на основі відеоматеріалів і зберігаючи всю інформацію для подальшого використання.

Нижче наведено таблицю 2.6, що показує, які дії виконує AIService, які дані потрібні для роботи і які результати повертаються.

Таблиця 2.6 – Задачі мікросервісу AIService

Функція	Вхідні дані	Вихідні дані
Почати аналіз файлу	Ідентифікатор файлу, ідентифікатор дрона	Повідомлення про початок аналізу, ідентифікатор результату
Перегляд результатів аналізу	Ідентифікатор результату	Інформація про результат (файл, дрон, висновки, стан)

Для AIService потрібна окрема база даних з таблицею (табл. 2.7), щоб зберігати результати аналізу відеофайлів і забезпечувати швидкий доступ до них. Ідентифікатор результату дозволяє відстежувати кожен аналіз, а посилання на файл і дрон допомагають пов’язати дані з їхнім джерелом. Поле «Результати аналізу» у гнучкому форматі дозволяє зберігати різноманітні висновки (наприклад, виявлені об’єкти чи перешкоди), адаптуючись до різних сценаріїв. Стан аналізу відображає його прогрес, а поле «Оповіщення про помилку» допомагає діагностувати проблеми. Дата створення та оновлення забезпечують хронологію, що важливо для моніторингу та подальшого використання результатів іншими сервісами. Така структура гарантує ізоляцію даних і підтримує ефективну обробку інформації в реальному часі.

Таблиця 2.7 – Структура таблиці «Results»

Results	
Поле	Опис
Ідентифікатор	Унікальний ідентифікатор результату для розпізнавання
Ідентифікатор файлу	Посилання на файл, який аналізується
Ідентифікатор дрона	Посилання на дрон, звідки походить файл
Результати аналізу	Зберігає деталі аналізу у гнучкому форматі
Дата створення	Час, коли аналіз розпочався
Стан	Поточний стан аналізу
Повідомлення про помилку	Текст із описом, якщо аналіз не вдалося завершити
Дата оновлення	Час останнього оновлення запису

CommandManagerService відповідає за організацію роботи рою дронів, видаючи вказівки, такі як зміна маршрутів або запуск завдань, і обмінюється інформацією з іншими частинами системи. Цей сервіс дозволяє переглядати список усіх відеофайлів, отриманих від дронів, визначати, які файли ще потребують обробки, і починати цю обробку, вказуючи, до якого дрона належить файл.

Після завершення обробки сервіс завершує процес і передає файли для аналізу, а також дає змогу дізнатися результати цього аналізу.

Сервіс стежить за станом дронів, отримуючи дані про їхній технічний стан, наприклад, рівень заряду батареї чи стан камери, і отримує історію цих даних для аналізу. На основі отриманих даних створюються команди, наприклад, повернення дрона на базу, якщо виявляються проблеми, як відключена камера чи низький заряд. Усі дії, включаючи створення команд, їх відправлення та оновлення стану, фіксуються в базі даних.

Цей сервіс також оновлює статус команд, позначаючи їх як виконані чи невиконані, і записує час завершення, якщо команда успішно завершена.

CommandManagerService допомагає координувати рій дронів, об’єднуючи інформацію від інших сервісів і видаючи чіткі інструкції для ефективної роботи в реальному часі. Нижче наведено таблицю 2.8, що показує, які дії виконує CommandManagerService, які дані потрібні для роботи і які результати повертаються.

Таблиця 2.8 – Задачі мікросервісу CommandManagerService

Функція	Вхідні дані	Вихідні дані
Перегляд усіх відеофайлів	-	Список усіх відеофайлів із деталями
Визначення файлів на обробку	-	Список файлів, що потребують обробки
Початок обробки файлу	Ідентифікатор файлу, ідентифікатор дрона	Повідомлення про початок обробки
Завершення обробки файлу	Ідентифікатор файлу	Повідомлення про завершення обробки
Передача файлу на аналіз	Ідентифікатор файлу, ідентифікатор дрона	Повідомлення про початок аналізу, ідентифікатор результату
Отримання результатів аналізу	Ідентифікатор результату	Інформація про результати аналізу
Отримання стану дрона	Ідентифікатор дрона	Дані про поточний стан дрона
Отримання історії стану дрона	Ідентифікатор дрона	Історія стану дрона
Перевірка стану дрона та створення команд	Дані про стан дрона (ідентифікатор, назва, заряд, камера, зв’язок)	Список створених команд із деталями
Оновлення стану команди	Ідентифікатор команди, новий стан	Оновлена інформація про команду

Для CommandManagerService потрібна окрема база даних, щоб координувати дії рою дронів і відстежувати виконання команд. Таблиця «Команди» (табл. 2.9) зберігає деталі кожної інструкції, включаючи дрон-

одержувача, текст команди та її стан, що дозволяє ефективно управляти процесом і перевіряти прогрес. Поле «Дата виконання» допомагає аналізувати час реакції дронів.

Таблиця «Журнали команд» (табл. 2.10) фіксує історію дій із командами, забезпечуючи можливість відстеження змін і діагностики проблем. Така структура забезпечує ізоляцію даних, підтримує реальну координацію та полегшує моніторинг роботи рою.

Таблиця 2.9 – Структура таблиці «Commands»

Commands	
Поле	Опис
Ідентифікатор	Унікальний ідентифікатор кожної команди
Ідентифікатор дрона	Посилання на дрон, для якого призначена команда
Текст команди	Опис дії, яку потрібно виконати
Стан	Поточний стан команди
Дата створення	Час, коли команда була створена
Дата виконання	Час, коли команда була виконана

HealthService відповідає за стеження за технічним станом дронів, виявляючи можливі проблеми, такі як відмова камери, низький рівень заряду батареї чи втрата зв’язку.

Основна роль сервісу – збирати дані про стан кожного дрона і передавати цю інформацію іншим частинам системи для швидкого реагування. Сервіс дозволяє дізнатися поточний стан дрона, включаючи рівень заряду, стан камери та зв’язок, а також переглянути історію змін цього стану.

Усі зібрані дані про дрони, їхні стани та минулі події зберігаються в окремому сховищі. Ця інформація допомагає відстежувати, що відбувалося з дроном у минулому, наприклад, коли виникала проблема чи коли змінювався стан. Сервіс забезпечує доступ до актуальних даних, що дозволяє іншим частинам системи координувати дії рою на основі поточного стану техніки.

HealthService відіграє важливу роль у підтримці працездатності дронів, надаючи точну інформацію про їхній стан і допомагаючи уникати збоїв під час роботи в реальному часі. Нижче наведено таблицю 2.11, що показує, які дії виконує HealthService, які дані потрібні для роботи і які результати повертаються.

Таблиця 2.11 – Задачі мікросервісу HealthService

Функція	Вхідні дані	Вихідні дані
Перегляд поточного стану дрона	Ідентифікатор дрона	Дані про стан дрона (заряд, камера, зв’язок)
Перегляд історії стану дрона	Ідентифікатор дрона	Список подій із історії стану дрона

Для HealthService потрібна окрема база даних, щоб ефективно відстежувати технічний стан дронів і забезпечувати швидке реагування на проблеми.

Таблиця «Дрони» (табл. 2.12) зберігає актуальні дані про стан кожного дрона, включаючи рівень заряду батареї, працездатність камери та статус зв’язку. Така деталізація дає змогу оперативно виявляти збої, такі як низький заряд батареї, відмова камери чи втрата зв’язку з дроном. Поле «час останнього оновлення» відіграє ключову роль, допомагаючи визначити, наскільки свіжими є ці дані, що критично для своєчасного реагування в реальному часі.

Таблиця «Історія стану дронів» (табл. 2.13) фіксує всі зміни стану дронів протягом усього періоду їхньої експлуатації. Це забезпечує можливість детального аналізу їхньої роботи, діагностики потенційних проблем і планування подальших дій, таких як профілактичне обслуговування чи заміна обладнання.

Така структура бази даних не лише гарантує ізоляцію даних HealthService від інших сервісів, а й підтримує безперервний моніторинг, підвищуючи загальну надійність системи.

Таблиця 2.12 – Структура таблиці «Drones»

Drones	
Поле	Опис
Ідентифікатор дрона	Унікальний ідентифікатор дрона для розпізнавання
Назва дрона	Ім’я дрона для ідентифікації в системі
Рівень заряду	Поточний рівень заряду батареї (від 0 до 100)
Стан камери	Показує, чи працює камера (ввімкнена/вимкнена)
Стан зв’язку	Показує, чи є зв’язок із дроном (активний/втрачений)
Час останнього оновлення	Час, коли дані про дрон були оновлені

Таблиця 2.13 – Структура таблиці «Drone_history»

Drone_history	
Поле	Опис
Ідентифікатор запису	Унікальний ідентифікатор запису в історії
Ідентифікатор дрона	Посилання на дрон, до якого відноситься запис
Тип події	Опис типу події
Опис події	Додаткові деталі про подію
Час події	Час, коли подія відбулася

2.2 Взаємодія матки рою та мікросервісної архітектури

Взаємодія матки рою з дронами та мікросервісною архітектурою забезпечує координацію роботи ройової системи. Матка рою, виступаючи як посередник, отримує від дронів телеметрію (координати, швидкість, стан батарей) і відеопотоки через протокол GStreamer [10] із кодеком H.264, а також передає ці дані до відповідних мікросервісів. VideoProcessorService приймає відеопотоки для попередньої обробки (наприклад, обрізання чи підготовки), HealthService – дані стану дронів для моніторингу, а AIService – телеметрію та

оброблені відеодані для аналізу. Обмін телеметрією та командами здійснюється через Fast DDS [11] із серіалізацією у форматі Protocol Buffers (Protobuf), що забезпечує швидкість і структуру даних. AIService обробляє дані, отримані від VideoProcessorService, які містять інформацію про об’єкти чи події (наприклад, перешкоди чи об’єкти, що потребують уваги), і передає ці дані до CommandManagerService для подальшої обробки.. Останній, базуючись на цих даних, формує команди (наприклад, зміну маршруту чи перепризначення ролей) і надсилає їх матці рою. Матка розподіляє ці команди серед дронів, використовуючи асинхронний обмін за моделлю publish-subscribe через Fast DDS.

Для критичних команд, таких як екстрене повернення, застосовуються синхронні запити (request-reply) із підтвердженням. HealthService повертає матці дані про стан дронів для подальшого моніторингу, а VideoProcessorService – оброблені відеодані. Під час перебоїв зв’язку матка забезпечує буферизацію даних і повторну передачу через Fast DDS, а також динамічне виявлення дронів у рої. Автентифікація між маткою, дронами та мікросервісами здійснюється через токени, а пріоритетність даних (телеметрія, команди, відео) регулюється QoS-профілями Fast DDS.

Процес взаємодії виглядає так: дрони надсилають телеметрію і відеопотоки матці, яка передає відеопотоки до VideoProcessorService для обробки, телеметрію – до HealthService і AIService, а також дані стану – до HealthService. AIService аналізує оброблені відеодані, розпізнаючи об’єкти, і передає результати до CommandManagerService. Сервіс формує команди на основі аналізу та надсилає їх матці рою. Матка розподіляє ці команди серед дронів, а HealthService і VideoProcessorService повертають матці оновлені дані стану та оброблені відеопотоки для подальшого використання.

Схема (рис. 2.2) ілюструє взаємодію між маткою рою, дронами та мікросервісами, відображаючи потоки даних, обробку та координацію. Ця взаємодія забезпечує адаптивність рою до змін умов і складу, інтегруючи апаратну частину (дрони, матка) із програмною (мікросервіси).

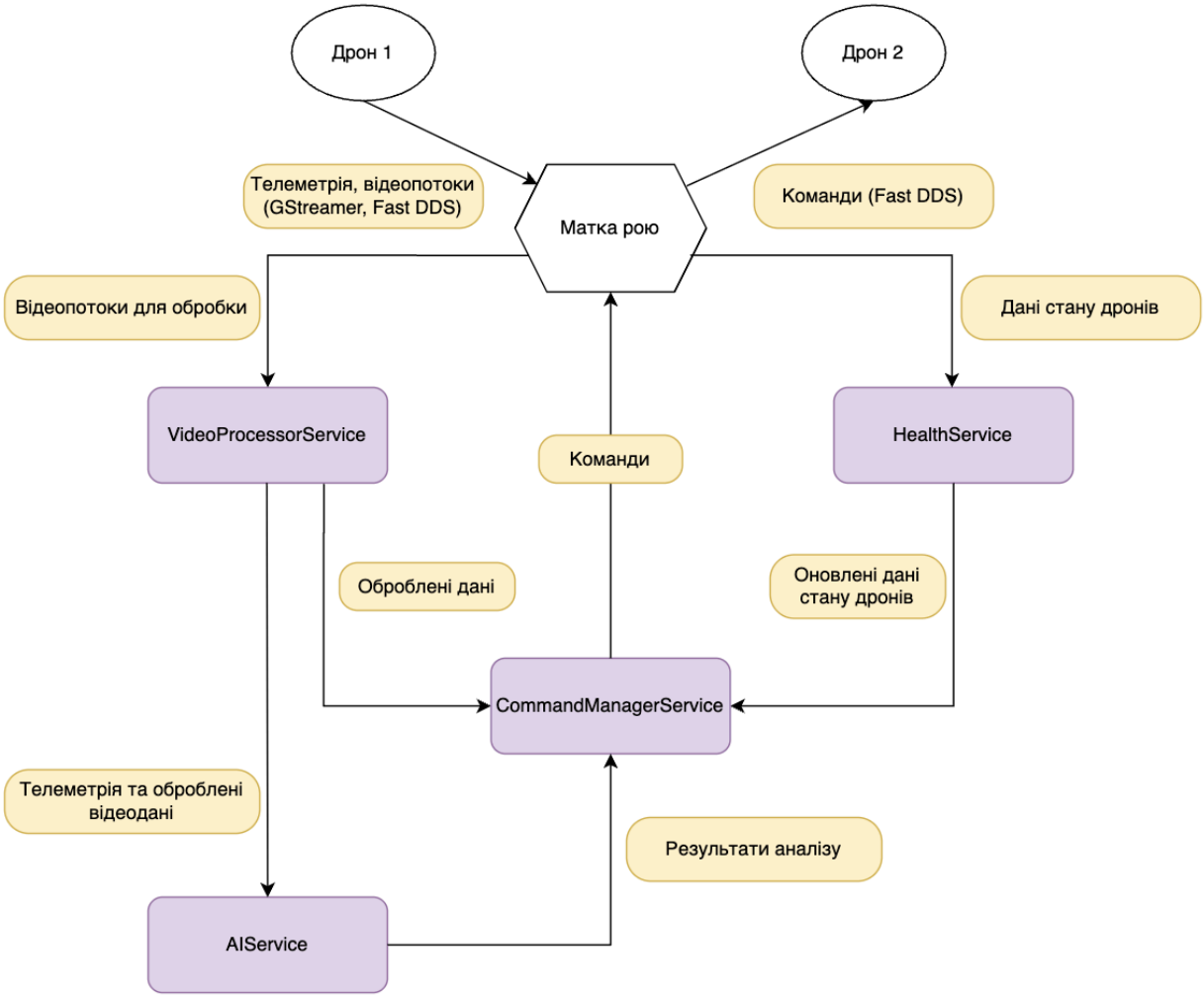


Рисунок 2.2 – Схема взаємодії матки рою з дронами та мікросервісами

2.3 Вибір засобів реалізації для створення системи

Технологічний стек системи для керування роєм дронів (рис. 2.3) сформовано з урахуванням потреб високої продуктивності, масштабованості та забезпечення стабільної взаємодії між компонентами. Для розробки мікросервісів обрано Node.js у поєднанні з Express.js [12], для зберігання даних – PostgreSQL і MongoDB, а для стандартизації API – OpenAPI Generator. Кожен із цих інструментів має обґрунтоване застосування, яке враховує специфіку роботи системи в реальному часі та необхідність обробки великих обсягів даних. Ця схема показує, як Node.js і Express.js формують основу для

створення мікросервісів, які використовують PostgreSQL і MongoDB для зберігання даних, а OpenAPI Generator забезпечує стандартизовану взаємодію.

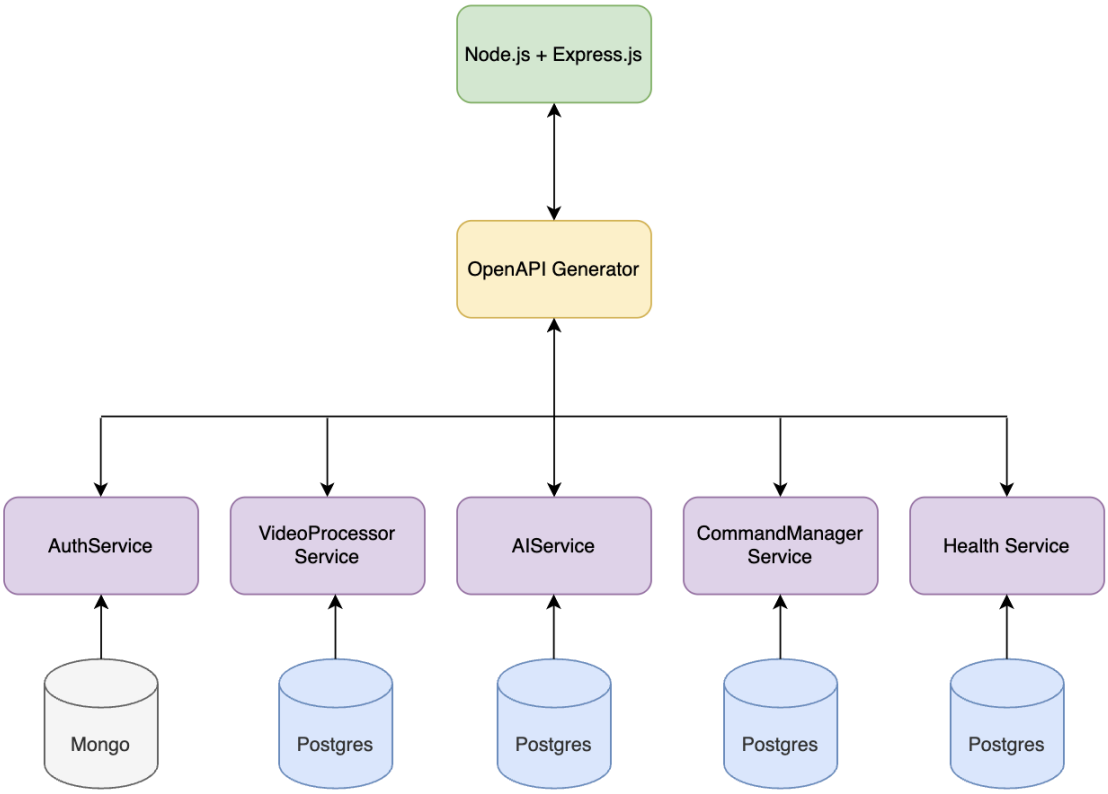


Рисунок 2.3 – Схема технологічного стеку

Node.js разом із Express.js обрано для створення мікросервісів через здатність ефективно обробляти асинхронні операції, що необхідно для швидкої координації рою дронів, коли запити надходять одночасно від багатьох джерел. Такий підхід дозволяє системі миттєво реагувати на зміни, наприклад, на нові команди чи оновлення стану дронів, що критично важливо для роботи в надзвичайних ситуаціях. Express.js забезпечує чітку структуру для маршрутизації запитів, що зменшує час на розробку складних взаємодій між сервісами [13], таких як передача відеофайлів для аналізу чи генерація команд на основі стану дронів. Цей вибір також підтримує масштабування, адже додавання нових сервісів чи розширення функціоналу не потребує значних змін у архітектурі.

Для зберігання даних використовується PostgreSQL із розподілом на окремі бази для кожного мікросервісу [14], що дозволяє ізолювати інформацію та забезпечити стабільність (рис. 2.4). Наприклад, дані про стан дронів, журнали команд і результати аналізу зберігаються окремо, що зменшує ризик збоїв, якщо одна база даних зазнає перевантаження. PostgreSQL обрано через підтримку транзакцій і складних запитів, що необхідно для забезпечення послідовності даних, наприклад, при одночасному оновленні стану кількох дронів чи обробці історії команд.

Для AuthService додатково застосовується MongoDB, оскільки документна модель цієї бази даних дозволяє гнучко адаптувати структуру даних до змін, таких як додавання нових ролей користувачів чи розширення їхніх характеристик [15].

MongoDB забезпечує швидкий доступ до даних автентифікації, що важливо для перевірки доступу в реальному часі, коли кожен запит до системи потребує миттєвої верифікації. Використання MongoDB для AuthService також обґрунтовується його високою продуктивністю при роботі з ієрархічними даними, адже профілі користувачів можуть включати вкладені структури, наприклад, списки дозволів чи історію входів, які зручніше зберігати у форматі документів.

Крім того, MongoDB підтримує горизонтальне масштабування, що дозволяє легко розширювати систему при зростанні кількості користувачів без значних змін у структурі бази даних.

Таким чином, комбінація PostgreSQL і MongoDB забезпечує баланс між стабільністю, гнучкістю та швидкістю обробки даних для різних потреб системи. OpenAPI Generator використовується для створення стандартизованих API-контрактів, що забезпечує чітку взаємодію між мікросервісами [16]. Цей інструмент дозволяє автоматично генерувати клієнтський код, що зменшує ризик помилок під час інтеграції, наприклад, коли CommandManagerService надсилає запити до AIService для аналізу відео. Стандартизація API гарантує однаковий формат обміну даними, що необхідно

для стабільної роботи системи, де затримки чи невідповідності можуть призвести до збоїв у координації дронів. Використання OpenAPI також полегшує тестування та подальше розширення системи, адже нові функції можна додавати без порушення існуючих зв'язків.

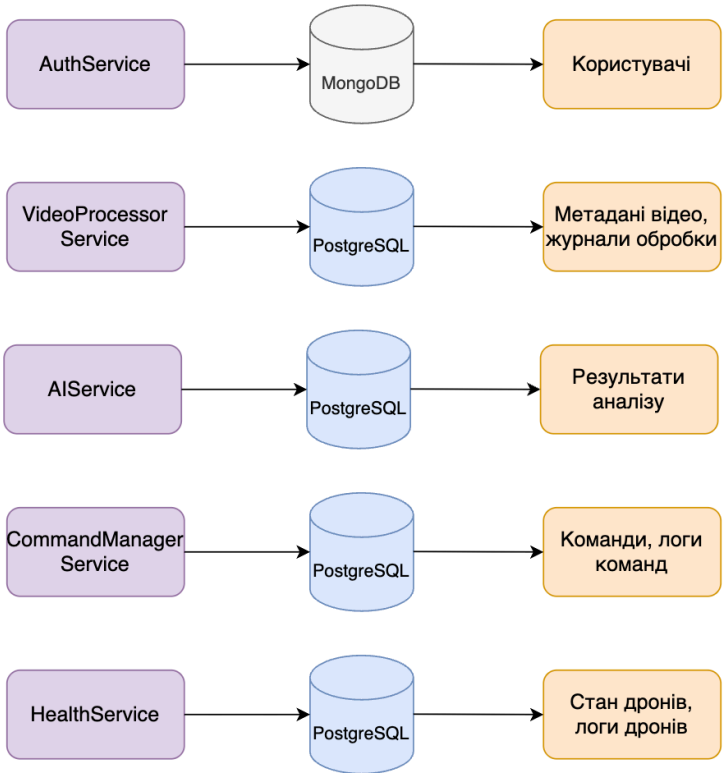


Рисунок 2.4 – Схема розподілу баз даних

Вибір цього стеку зумовлений необхідністю забезпечити високу швидкість обробки даних, стабільність і можливість масштабування системи. Node.js і Express.js дозволяють обробляти велику кількість одночасних запитів, що відповідає вимогам роботи рою дронів у реальному часі. PostgreSQL гарантує надійність і цілісність даних, а MongoDB додає гнучкості для роботи з користувацькими даними, що оптимізує автентифікацію. OpenAPI Generator забезпечує стандартизовану взаємодію, що критично для системи з кількома мікросервісами, де кожен залежить від іншого. Такий підхід дозволяє системі обробляти складні завдання, як-от аналіз відео чи координація дронів, без втрати продуктивності чи безпеки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

Взаємодія між мікросервісами системи для керування роєм дронів здійснюється через REST API, що забезпечує чіткий і структурований обмін даними. Для стандартизації API застосовується OpenAPI Generator, який дозволяє створювати специфікації, генерувати клієнтський код і підтримувати документацію, що значно полегшує інтеграцію між компонентами. Усі API-контракти зберігаються в централізованому репозиторії, звідки кожен мікросервіс може отримувати актуальні специфікації, забезпечуючи однаковість взаємодії та можливість повторного використання коду. Такий підхід дозволяє уникнути помилок при обміні даними та забезпечує стабільну роботу системи, де кожен сервіс чітко знає, які запити надсилати та які відповіді очікувати.

3.1 Автентифікація та верифікація доступу

Розглянемо детально процес автентифікації та верифікації доступу, що є основою безпечної взаємодії між мікросервісами. Наприклад, CommandManagerService перед виконанням будь-якої операції, такої як надсилання команди дронам, повинен перевірити права доступу користувача через AuthService. Цей процес включає кілька етапів: визначення API через OpenAPI специфікацію, генерацію клієнтського коду для взаємодії, використання цього коду в CommandManagerService і обробку запитів у самому AuthService. Усі ці етапи забезпечують безперебійну та безпечну роботу системи.

Першим кроком у реалізації взаємодії є створення специфікації API для AuthService. Специфікація оформлюється у форматі YAML за стандартом OpenAPI 3.0.0 і визначає, які запити може обробляти AuthService, які дані потрібні для цих запитів і які відповіді повертаються. У YAML-файлі для AuthService описано кілька ендпоінтів, зокрема для реєстрації, входу

користувача та верифікації токена. Приклад фрагмента специфікації для ендпоінта /auth/verify, який відповідає за перевірку JWT-токена наведено у лістингу 3.1.

```
openapi: 3.0.0 info:
  title: Auth Service API
  version: 1.0.0
servers:
  - url: http://localhost:3000
paths:
  /auth/verify:
    post:
      summary: Verify JWT token
      requestBody:
        required: true
        content:
          application/json:
            schema: type: object
              required: - token
              properties:
                token: type: string
      responses:
        '200':
          description: Token is valid
          content:
            application/json:
              schema: type: object
                properties:
                  valid: type: boolean
                  identity: type: object
                    properties:
                      id: type: string
                      email: type: string
                      role: type: string
        '400':
          description: Token missing
          content:
            application/json:
              schema: type: object
                properties:
                  error:
                    type: string
```

Лістинг 3.1 – Фрагмент YAML-контракту в AuthService

```
'401':
  description: Invalid or expired token
  content:
    application/json:
      schema: type: object
        properties:
          error:type: string
```

Лістинг 3.1, аркуш 2

Цей фрагмент YAML-файлу визначає, що ендпоінт /auth/verify приймає POST-запит із JSON-об’єктом, який містить поле token типу рядка. У разі успішної перевірки повертається відповідь із кодом 200, що містить об’єкт із полем valid, яке вказує на валідність токена, та об’єктом identity, що включає ідентифікатор, електронну адресу та роль користувача. Якщо токен відсутній, повертається код 400 із повідомленням про помилку, а якщо токен невалідний чи прострочений – код 401. Повний код YAML-специфікації для AuthService та інших сервісів наведено в додатку А.

Наступним етапом є генерація клієнтського коду для використання в інших мікросервісах, таких як CommandManagerService. Для цього застосовується OpenAPI Generator із набором команд, визначених у скриптах проєкту. Команда для генерації наведена у лістингу 3.2.

```
"sdk:generate": "openapi-generator-cli generate -i ./auth-
api.yaml -g javascript -o ./auth-sdk --additional-
properties=usePromises=true,projectName=auth_service_api,moduleN
ame=AuthApi,ensureUniqueParams=true",
"sdk:build": "npx babel ./auth-sdk/src --out-dir ./auth-sdk/lib
--presets=@babel/preset-env",
"sdk:package": "cd ./auth-sdk && npm pack",
"sdk:install": "npm run sdk:package && mv ./auth-
sdk/auth_service_api-1.0.0.tgz ../command-manager && cd
../command-manager && npm install auth_service_api-1.0.0.tgz",
"sdk:full": "npm run sdk:generate && npm run sdk:build && npm
run sdk:install"
```

Лістинг 3.2 – Команди для генерації клієнтського коду

Цей набір команд виконує кілька дій: спочатку генерується клієнтський код на основі YAML-файлу auth-api.yaml для JavaScript, з увімкненням використання промісів (usePromises=true) і унікальних параметрів (ensureUniqueParams=true). Результат зберігається в директорії ./auth-sdk. Потім код компілюється за допомогою Babel для забезпечення сумісності з різними середовищами, пакується в архів і встановлюється в інші мікросервіси, такі як CommandManagerService. Це дозволяє мікросервісам легко викликати методи AuthService, використовуючи згенерований SDK.

Згенерований код для ендпоінта /auth/verify міститься у файлі DefaultApi.js і включає методи для виклику цього ендпоінта. Приклад згенерованого методу для перевірки токена наведено у лістингу 3.3.

```
 /**
     * Verify JWT token
     * @param {module:model/AuthVerifyPostRequest}
authVerifyPostRequest
     * @return {Promise} a {@link
https://www.promisejs.org/|Promise}, with an object containing
data of type {@link module:model/AuthVerifyPost200Response} and
HTTP response */
authVerifyPostWithHttpInfo(authVerifyPostRequest) {
  var postBody = authVerifyPostRequest;
  // verify the required parameter 'authVerifyPostRequest' is
set
  if (authVerifyPostRequest === undefined ||
authVerifyPostRequest === null) {
    throw new Error("Missing the required parameter
'authVerifyPostRequest' when calling authVerifyPost");
  }
  var pathParams = {};
  var queryParams = {};
  var headerParams = {};
  var formParams = {};
  var authNames = [];
  var contentTypes = ['application/json'];
  var accepts = ['application/json'];
  var returnType = _AuthVerifyPost200Response["default"];
```

Лістинг 3.3 – Згенерований код для верифікації

```
return this.apiClient.callApi('/auth/verify', 'POST',
pathParams, queryParams, headerParams, formParams, postBody,
authNames, contentTypes, accepts, returnType, null);
},
/**
 * Verify JWT token
 * @param {module:model/AuthVerifyPostRequest}
authVerifyPostRequest
 * @return {Promise} a {@link
https://www.promisejs.org/|Promise}, with data of type {@link
module:model/AuthVerifyPost200Response}
 */
authVerifyPost(authVerifyPostRequest) {
  return
this.authVerifyPostWithHttpInfo(authVerifyPostRequest).then(func
tion (response_and_data) {
    return response_and_data.data;
  });
}
```

Лістинг 3.3, аркуш 2

Цей код визначає два методи: authVerifyPostWithHttpInfo повертає повну відповідь із HTTP-даними, включаючи деталі запиту, такі як код статусу, заголовки та тіло відповіді, що дозволяє детально аналізувати взаємодію між сервісами, а authVerifyPost – лише дані відповіді, що спрощує використання, якщо потрібен лише результат верифікації. Метод authVerifyPostWithHttpInfo очікує об’єкт authVerifyPostRequest, який містить токен як обов’язковий параметр, і надсилає POST-запит на ендпоінт /auth/verify. Перед відправленням запит перевіряється на наявність цього параметра, щоб уникнути помилок через відсутність даних.

Згенерований SDK забезпечує чіткий і стандартизований інтерфейс для виклику цього ендпоінта з інших мікросервісів, таких як CommandManager Service чи VideoProcessorService, зменшуючи ризик помилок під час інтеграції завдяки автоматично створеним методам і документації. Такий підхід також полегшує оновлення API, адже зміни в специфікації автоматично відображаються в згенерованому коді без необхідності ручної модифікації.

Далі CommandManagerService використовує згенерований SDK для перевірки токена перед виконанням будь-яких операцій, таких як надсилання команд рою дронів чи доступ до відеофайлів.

У middleware-файлі CommandManagerService визначено функцію для верифікації токена, яка викликає метод із SDK і виступає захисним бар'єром, забезпечуючи, що лише авторизовані користувачі можуть взаємодіяти з системою. Ця функція перевіряє наявність токена в кукі запиту, надсилає його на верифікацію через згенерований метод authVerifyPostWithHttpInfo і, якщо токен визнано валідним, додає дані про користувача (ідентифікатор, електронну адресу, роль) до об'єкта запиту для подальшої обробки іншими маршрутами. У разі відсутності токена чи його невалідності функція повертає відповідь із кодом 401 і детальним повідомленням про помилку, що дозволяє швидко виявити проблему.

Такий механізм інтегрується в усі маршрути CommandManagerService, забезпечуючи безперервну перевірку прав доступу й підвищуючи безпеку системи під час координації дронів у реальному часі. Приклад використання наведено у лістингу 3.4.

```
const AuthApi = require('auth_service_api');
const sdk = new AuthApi
    .DefaultApi();
sdk.basePath = `${process.env.AUTH_SERVICE_URL}/auth`;
module.exports = async (req, res, next) =>
{
    const token = req.cookies?.access_token;
    if (!token) {
        return res.status(401)
            .json({ error: 'No token provided' });
    }
    try {
        const response = await sdk
            .authVerifyPostWithHttpInfo(
                { token });
        const data = response.data;
```

Лістинг 3.4 – Використання методу з SDK в CommandManagerService

```
        if (!data || !data.valid) {
            return res.status(401).json({ error: 'Invalid token'
});
        }
        req.user = data.identity;
        next();
    } catch (error) {
        console.error('SDK  Error:',  error?.response?.body  ||
error.message);
        res.status(401).json({ error: 'Token verification failed'
});
    }
};
```

Лістинг 3.4, аркуш 2

Ця функція перевіряє наявність токена в кукі, надсилає його на верифікацію через згенерований метод authVerifyPostWithHttpInfo і, якщо токен валідний, додає дані користувача (ідентифікатор, електронну адресу, роль) до запиту для подальшої обробки.

Якщо токен відсутній або невалідний, користувач отримує відповідь із кодом 401 і повідомленням про помилку. Цей middleware застосовується до всіх маршрутів CommandManagerService, щоб гарантувати, що лише авторизовані користувачі можуть надсилати команди дронам.

Основний файл CommandManagerService index.js налаштовує сервер і застосовує middleware для всіх маршрутів. Його вміст наведено у лістингу 3.5.

```
require('dotenv').config();
const express = require('express');
const cookieParser = require('cookie-parser');
const commandRoutes = require('./routes/command');
const rootCommandRoutes = require('./routes/root-command');
const authMiddleware = require('./middleware/auth');
const app = express();
app.use(express.json());
app.use(cookieParser());
```

Лістинг 3.5 – Код серверу в CommandManagerService

```
app.use('/api/command', authMiddleware, commandRoutes);
const PORT = process.env.PORT || 3002;
app.listen(PORT, () => console.log(`Command Manager running on
port ${PORT}`));
```

Лістинг 3.5, аркуш 2

Цей файл налаштовує сервер на порту 3002, підключає middleware для обробки JSON-запитів і кук, а також застосовує authMiddleware до всіх маршрутів /api/command. Це означає, що кожен запит, наприклад, до ендпоінта для надсилання команд чи перегляду відео, спочатку проходить перевірку токена через виклик до AuthService. Якщо перевірка успішна, запит обробляється далі, якщо ні – користувач отримує повідомлення про помилку.

Для наочності розглянемо схему (див. рис. 3.1) взаємодії між AuthService і CommandManagerService:

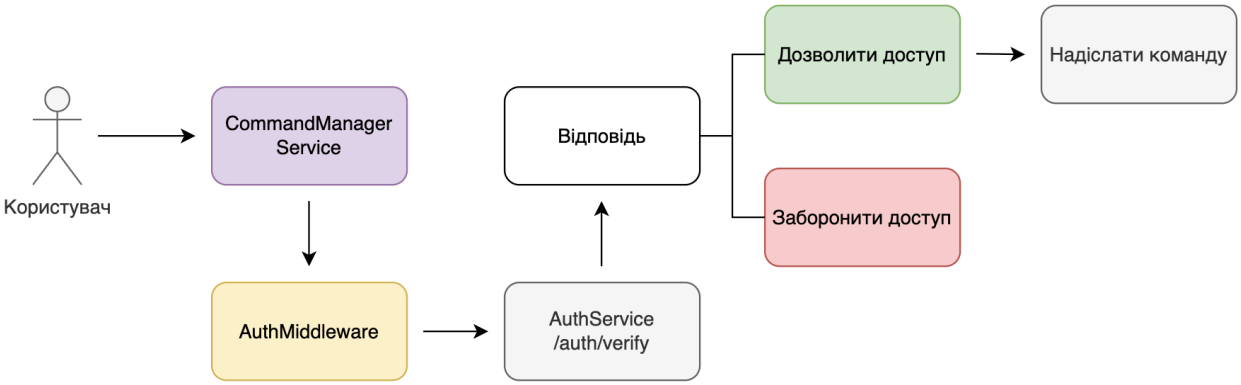


Рисунок 3.1 – Схема взаємодії мікросервісів

Ця схема показує, як запит від користувача надходить до CommandManagerService, проходить перевірку токена через виклик до AuthService, і залежно від результату або дозволяється виконання команди, або повертається помилка. Такий механізм забезпечує безпечну та ефективну координацію рою дронів, підтримуючи стандартизовану взаємодію через REST API та обробку даних у реальному часі.

3.2 Аналіз відеоданих

Розглянемо ще один приклад взаємодії між мікросервісами, який підкреслює важливість стандартизованих API та роботи з розподіленими базами даних у системі керування роєм дронів.

Цей сценарій демонструє, як CommandManagerService ініціює аналіз відеофайлу, звертаючись до AIService, який взаємодіє з VideoProcessorService для отримання метаданих, а отримані дані фіксуються в базі AIService для подальшого аналізу. Такий процес відображає гармонію між API-взаємодією та управлінням даними.

Процес розпочинається з CommandManagerService, який отримує запит на аналіз відеофайлу, ідентифікованого за fileId, та вказівку про дрон drone_id.

CommandManagerService використовує згенерований OpenAPI-клієнт для надсилання POST-запиту до AIService на ендпоінт /process-and-send. Для цього налаштовується клієнт із передачею токена доступу в заголовках, що забезпечує автентифікацію. Ключовий фрагмент коду наведено у лістингу 3.6.

```
const aiApiClient = new ApiClientAi();
aiApiClient.defaultHeaders =
       {Cookie: `access_token=${token}`,};
const aiApi = new AIModuleApi(aiApiClient);
aiApi.basePath = 'http://localhost:3003/ai';
const response = await aiApi.analyzePost
      ({ file_id: fileId, drone_id });
```

Лістинг 3.6 – Код взаємодії CommandManagerService та AIService

Код ініціалізує клієнт AIService, додає токен до заголовків для верифікації та викликає метод analyzePost, передаючи ідентифікатори файлу й дрона. Після отримання відповіді з ідентифікатором результату result_id, CommandManagerService записує лог у свою базу command (лістинг 3.7).

Запит фіксує початок аналізу, забезпечуючи відстеження операцій у власній базі CommandManagerService.

```
await client.query(
    `INSERT INTO command.logs (email, action, file_id, drone_id,
result_id, status)
     VALUES ($1, $2, $3, $4, $5, $6)`,
    [req.user?.email || 'unknown', 'process-and-send', fileId,
drone_id, response.result_id, 'success']
);
```

Лістинг 3.7 – Логування дій, в базу даних CommandManagerService

AIService, отримавши запит, також використовує OpenAPI-клієнт для звернення до VideoProcessorService. Налаштування клієнта наведено у лістингу 3.8.

```
const videoApiClient = new VideoApiClient();
videoApiClient.defaultHeaders = {
    Cookie: `access_token=${token}`,};
const videoApi = new VideoProcessorApi(videoApiClient);
videoApi.basePath = 'http://localhost:3001/api/video';
const fileResponse = await videoApi
    .fileInfoPost
      ({ fileId: file_id });
```

Лістинг 3.8 – Код взаємодії AIService та VideoProcessorService

Фрагмент створює клієнт VideoProcessorService, передає токен у заголовках для автентифікації та викликає метод fileInfoPost, щоб отримати метадані файлу. Після перевірки, що файл є відео, AIService додає запис у свою базу ai (див. лістинг 3.9).

```
const result = await client
    .query(
    `INSERT INTO ai.results
     (file_id, drone_id, status)
     VALUES
    ($1, $2, 'pending')
RETURNING id`,
     [file_id, drone_id]);
const resultId = result.rows[0].id;
```

Лістинг 3.9 – Додавання даних про файл, в базу даних AIService

Запит створює новий запис із початковим статусом pending і повертає унікальний resultId, який передається назад до CommandManagerService.

VideoProcessorService, отримавши запит від AIService, звертається до своєї бази video, щоб витягти метадані файлу (див. лістинг 3.10).

```
const result = await client.query(
    `SELECT id AS file_id, drone_id, file_type, status
     FROM video_processor_db.files
     WHERE id = $1`, [fileId]
);
```

Лістинг 3.10 – Отримання даних про файл, з бази даних VideoProccessorService

Запит витягує дані з таблиці files, перевіряючи, чи файл відповідає критеріям (є відео і має статус processed), і повертає їх AIService для подальшого аналізу.

Цей сценарій підкреслює злагоджену роботу розподілених баз даних, де кожен сервіс використовує свою базу – command для логів CommandManager Service, ai для результатів AIService і video для метаданих VideoProcessorService. Взаємодія між сервісами забезпечується через згенеровані OpenAPI-клієнти, які передають токен у заголовках для верифікації на кожному етапі, гарантуючи доступ лише авторизованим користувачам.

Для наочності розглянемо схему (рис. 3.2) взаємодії між Command ManagerService, AIService і VideoProcessorService під час аналізу відеофайлу, яка допомагає краще зрозуміти складний процес координації в системі керування роєм дронів. Ця схема відображає, як дані передаються між сервісами та базами даних, забезпечуючи безперервну обробку запиту від користувача до отримання результату. Кожен етап взаємодії підкреслює важливість стандартизованих API та ізольованого зберігання даних, що є основою надійної роботи системи в реальному часі.

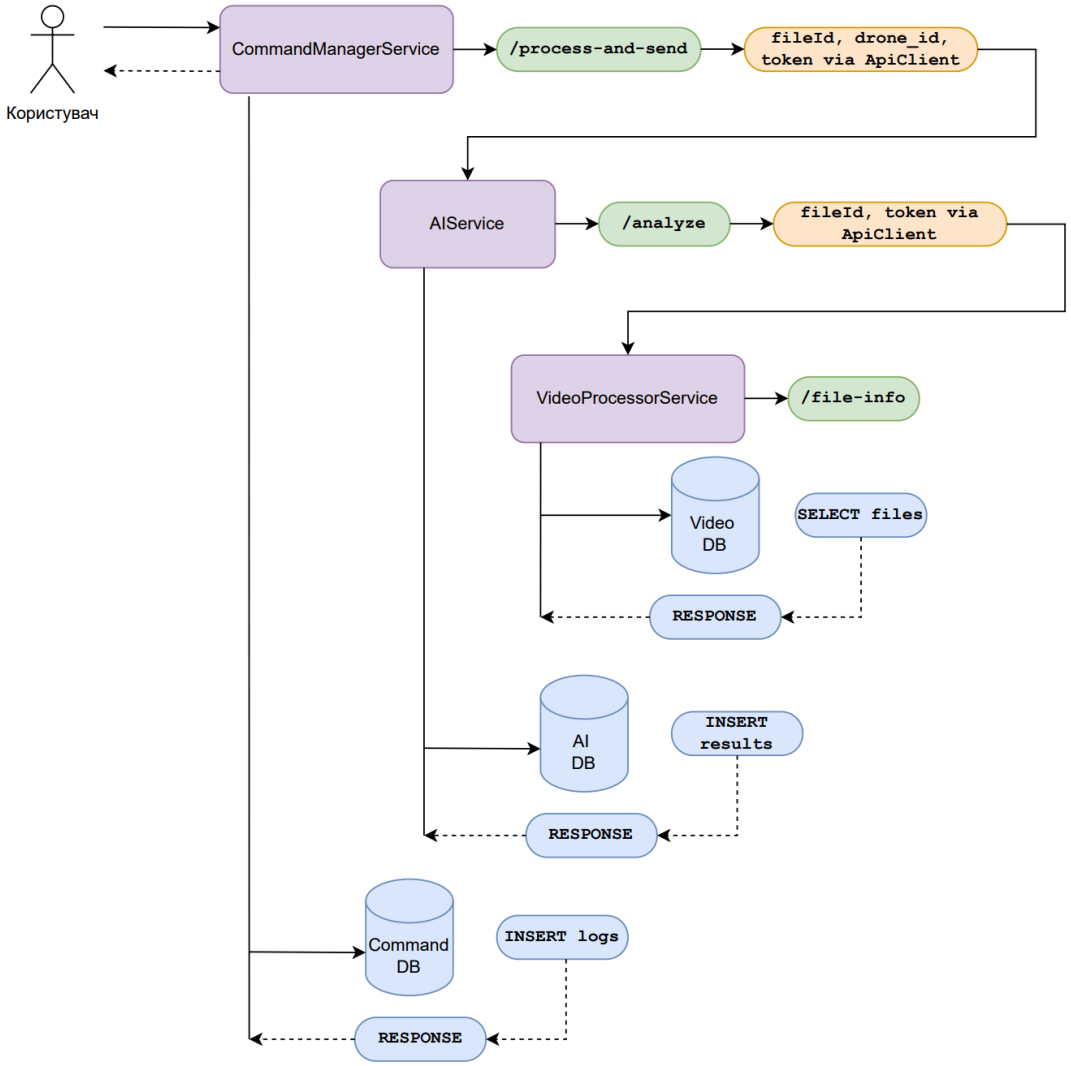


Рисунок 3.2 – Схема взаємодії мікросервісів

Такий підхід до взаємодії між мікросервісами забезпечує кілька ключових переваг. Модульність досягається завдяки розподілу функцій між сервісами: AuthService відповідає за автентифікацію, а CommandManagerService – за координацію дронів. Стандартизація через OpenAPI гарантує однаковість інтерфейсів, що полегшує інтеграцію та тестування. Відмовостійкість забезпечується ізоляцією сервісів: якщо AIService тимчасово недоступний, CommandManagerService може обробити цю помилку і повідомити користувача. Розподіленість даних через окремі бази (MongoDB для AuthService і PostgreSQL для CommandManagerService, AIService та VideoProcessorService) зменшує залежності між сервісами, що

підвищує загальну стабільність системи. Ця архітектура дозволяє системі гнучко адаптуватися до навантажень і забезпечувати безперебійну координацію рою дронів навіть у складних умовах.

3.3 Реєстрація та авторизація користувача

Процеси автентифікації в системі забезпечуються сервісом AuthService, який відповідає за реєстрацію та авторизацію користувачів. Нижче наведено опис цих задач із відповідними схемами для кращого розуміння.

Для реєстрації користувач надсилає електронну адресу, пароль та обирає роль. Дані надсилаються до AuthService, який перевіряє їх коректність, зберігає в базі даних MongoDB і генерує унікальний ідентифікатор користувача. Після успішної реєстрації повертаються ідентифікатор, електронна адреса та призначена роль, що забезпечує можливість подальшого використання системи.

Схема відображає взаємодію між користувачем, AuthService і базою даних під час реєстрації, демонструючи послідовність обробки даних і збереження інформації, представлена на рисунку 3.3.

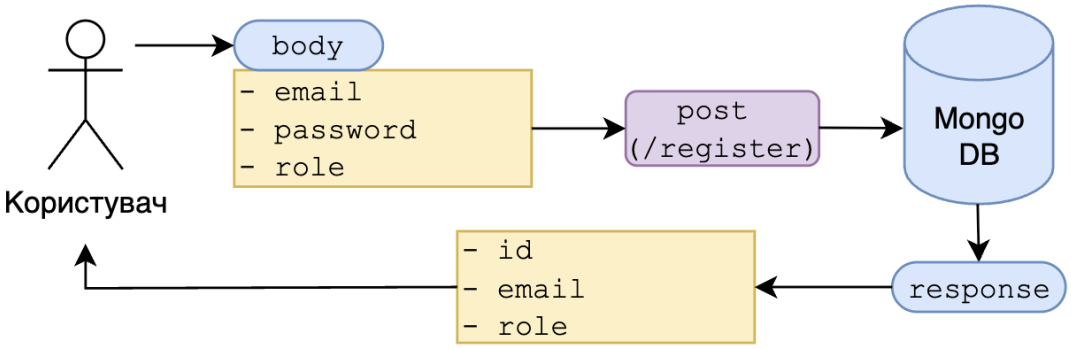


Рисунок 3.3 – Схема взаємодії при реєстрації користувача

Для авторизації користувач надсилає електронну адресу та пароль через POST-запит на ендпоінт /login. AuthService виконує перевірку даних у базі

MongoDB, порівнюючи введений пароль із захешованим значенням. У разі успішної авторизації генерується JWT-токен, який включає ідентифікатор користувача, електронну адресу, роль і термін дії, а також зберігається в cookie під назвою access_token. Користувачу повертаються токен, ідентифікатор, електронна адреса та роль для доступу до захищених ресурсів.

Схема, яка ілюструє процес авторизації, показуючи етапи перевірки даних, генерації токена та його передачі клієнту, представлена на рисунку 3.4.

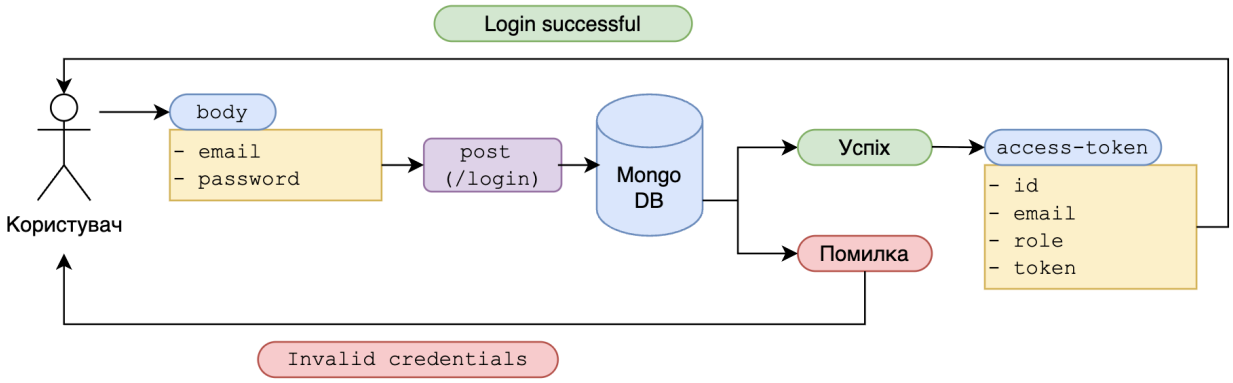


Рисунок 3.4 – Схема взаємодії при авторизації користувача

3.4 Перегляд відеофайлів

Процес перегляду усіх відеофайлів у системі реалізується через CommandManagerService, який координує доступ до даних. Користувач надсилає запит через інтерфейс до ендпоінта /videos у CommandManagerService. Сервіс перевіряє наявність токена авторизації в кукі, що забезпечує автентифікацію користувача. Якщо токен відсутній, повертається помилка. При успішній перевірці CommandManagerService використовує API клієнта для звернення до VideoProcessorService, передаючи токен у заголовку. VideoProcessorService виконує запит до бази даних PostgreSQL, де відбирає всі записи таблиці files із типом video, сортує їх за часом створення у спадному порядку та повертає список із деталями.

CommandManagerService отримує ці дані й передає їх користувачу разом із повідомленням про успішне виконання.

Схема, яка відображає взаємодію між CommandManagerService, VideoProcessorService і базою даних PostgreSQL під час перегляду відеофайлів, підкреслюючи послідовність обробки запиту та передачі даних, представлена на рисунку 3.5.

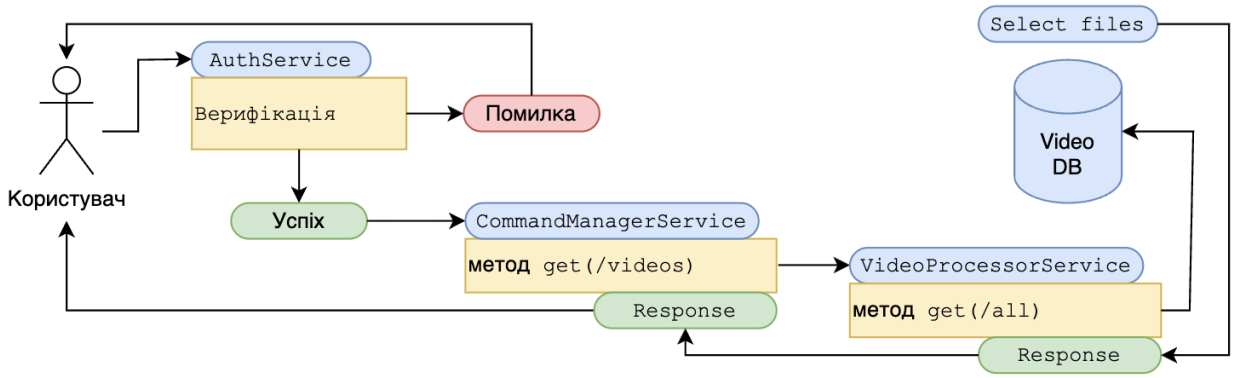


Рисунок 3.5 – Схема перегляду відеофайлів

Такий підхід забезпечує централізоване управління доступом через CommandManagerService і ефективну інтеграцію з VideoProcessorService, що полегшує доступ до відеофайлів у системі.

3.5 Обробка файлу

Процес початку обробки файлу в системі реалізується через взаємодію CommandManagerService та VideoProcessorService. Користувач надсилає запит до CommandManagerService через ендпоінт /start-processing, передаючи ідентифікатор файлу та дрона. CommandManagerService перевіряє наявність токена авторизації в кукі, а також коректність переданих даних. Після цього створюється команда в базі даних PostgreSQL із статусом pending, а дія логуються в таблиці command_logs як створення команди. Далі CommandManagerService викликає VideoProcessorService, передаючи токен і

ідентифікатор файлу. VideoProcessorService оновлює статус файлу в таблиці files на processing, додає запис у таблицю file_processing_logs про початок обробки та повертає підтвердження. CommandManagerService після цього оновлює статус команди на sent, фіксує час виконання та додає ще один запис у «command_logs» про відправлення команди. Уся операція виконується в рамках транзакції для забезпечення цілісності даних, а користувачу повертається повідомлення про успішний початок обробки.

Схема ілюструє взаємодію між CommandManagerService, VideoProcessorService і базою даних PostgreSQL під час початку обробки файлу, відображаючи послідовність дій, логування та оновлення статусів, , представлена на рисунку 3.6.

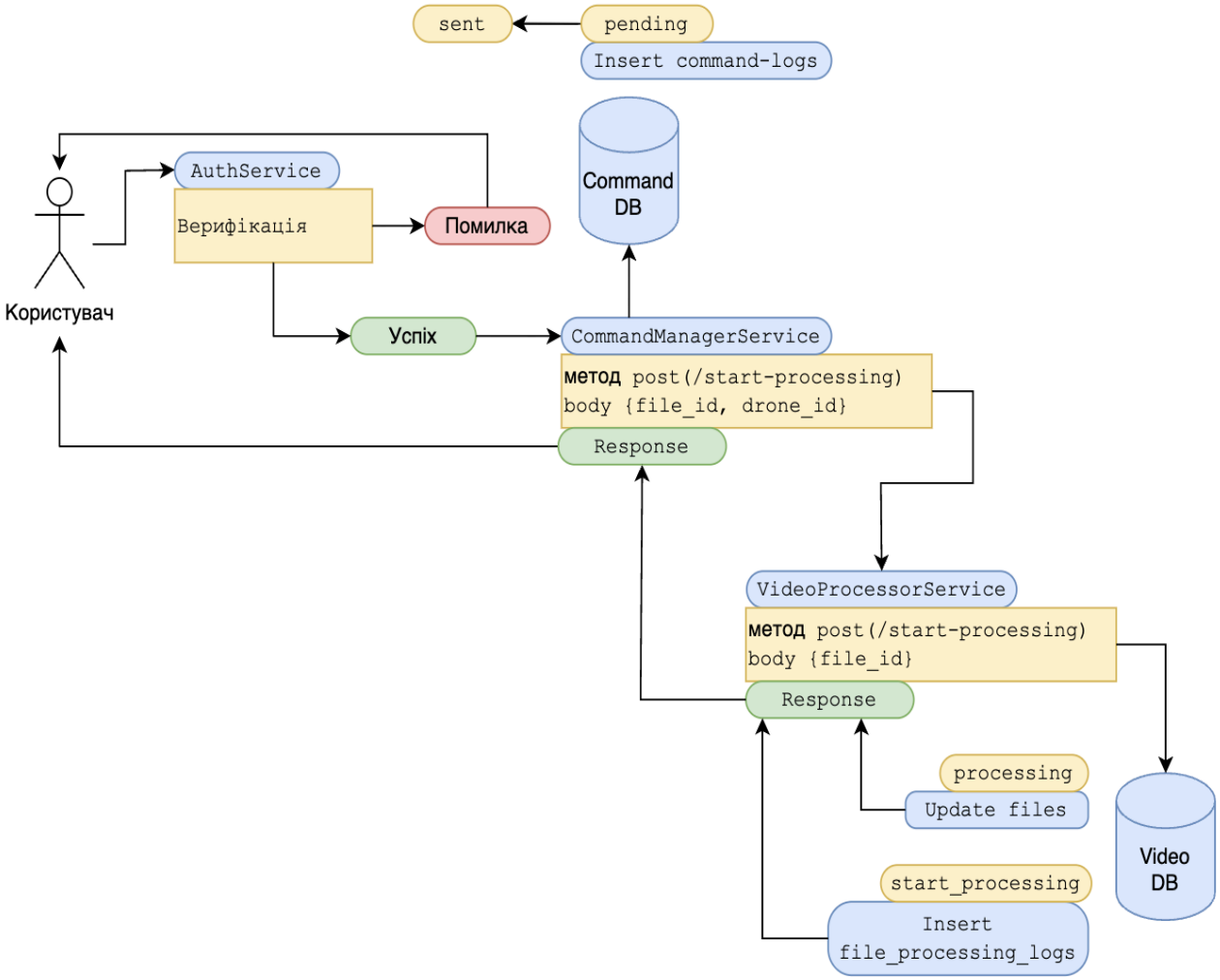


Рисунок 3.6 – Схема початку обробки файлу

3.6 Результати аналізу

Процес отримання результатів аналізу в системі реалізується через взаємодію CommandManagerService та AIService. Користувач надсилає запит до CommandManagerService через ендпоінт /ai-results/:resultId, вказуючи ідентифікатор результату. CommandManagerService перевіряє валідність ідентифікатора та наявність токена авторизації в кукі. При успішній перевірці сервіс звертається до AIService через API, передаючи токен і ідентифікатор результату. AIService виконує запит до бази даних PostgreSQL, відбираючи дані з таблиці results за вказаним ідентифікатором, включаючи деталі файлу, дрона, результат аналізу, статус та час оновлення. Отримані дані повертаються до CommandManagerService, який логує дію в таблиці logs із зазначенням електронної адреси, дії get-ai-results, ідентифікаторів файлу, дрона, результату та статусу. Уся операція виконується в транзакції для забезпечення цілісності даних. Після цього CommandManagerService передає результат користувачу разом із підтвердженням успішного виконання.

Схема ілюструє взаємодію між CommandManagerService, AIService і базою даних PostgreSQL під час отримання результатів аналізу, відображаючи послідовність запитів, логування та повернення даних, представлена на рисунку 3.7.

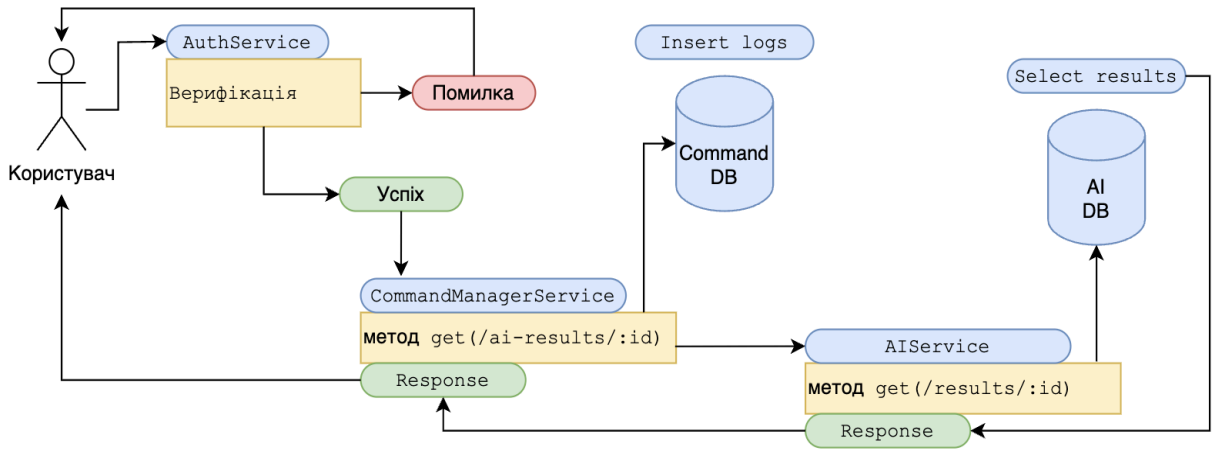


Рисунок 3.7 – Схема отримання результатів аналізу

Такий підхід забезпечує надійну координацію між сервісами, дозволяючи відстежувати дії через логування та гарантувати цілісність даних завдяки транзакціям.

3.7 Перегляд стану дрона

Процес перегляду стану дрона в системі реалізується через взаємодію CommandManagerService та HealthService. Користувач надсилає запит до CommandManagerService через ендпоінт /drone-status/:drone_id, вказуючи ідентифікатор дрона. CommandManagerService перевіряє наявність токена авторизації в кукі, що забезпечує захист доступу. У разі успішної перевірки сервіс звертається до HealthService через API, передаючи токен і ідентифікатор дрона. HealthService виконує запит до бази даних PostgreSQL, відбираючи з таблиці drones дані про стан дрона, зокрема його ідентифікатор, назву, рівень заряду батареї, статус камери, статус зв’язку та час останнього оновлення. Отримані дані повертаються до CommandManagerService, який передає їх користувачу разом із повідомленням про успішне виконання. Якщо дрона з вказаним ідентифікатором не знайдено, повертається відповідна помилка.

Схема ілюструє взаємодію між CommandManagerService, HealthService і базою даних PostgreSQL під час перегляду стану дрона, відображаючи послідовність запитів і передачу даних, представлена на рисунку 3.8.

Такий підхід забезпечує швидкий і безпечний доступ до даних про стан дрона, дозволяючи ефективно контролювати його технічні параметри.

3.8 Генерування команд на основі стану дрона

Процес генерування команд на основі стану дрона реалізується через CommandManagerService. Користувач надсилає POST-запит до ендпоінта /health-status із даними про стан дрона (ідентифікатор, рівень заряду батареї, статус камери, зв’язку та час оновлення). Сервіс перевіряє коректність даних і,

якщо рівень заряду < 30%, камера відключена чи зв'язок втрачено, генерує команди, які заносяться в таблицю commands (статус sent) і логуються в command_logs. Без проблем повертається повідомлення про відсутність команд, інакше – результат із командами передається користувачу.

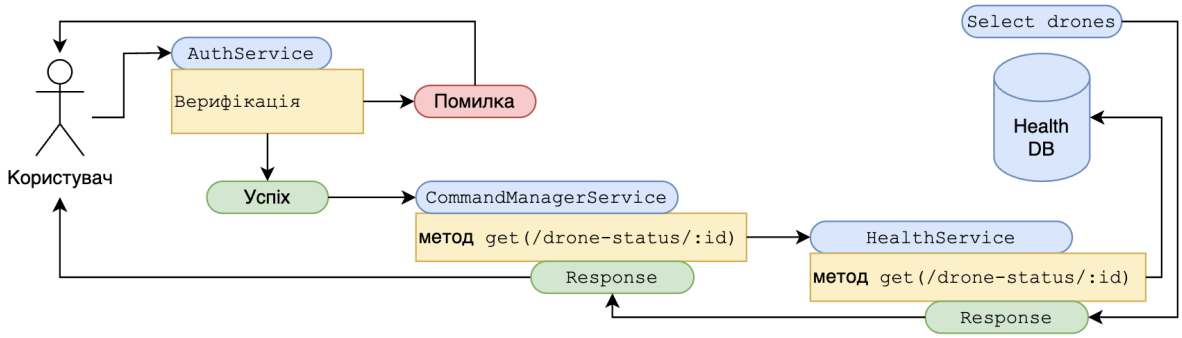


Рисунок 3.8 – Схема перегляду стану дрона

Схема ілюструє взаємодію між користувачем і CommandManagerService під час обробки стану дрона, відображаючи послідовність аналізу даних, генерування команд і оновлення бази даних PostgreSQL, представлена на рисунку 3.9. Такий підхід дозволяє автоматично реагувати на стан дрона, формуючи необхідні команди для його керування.

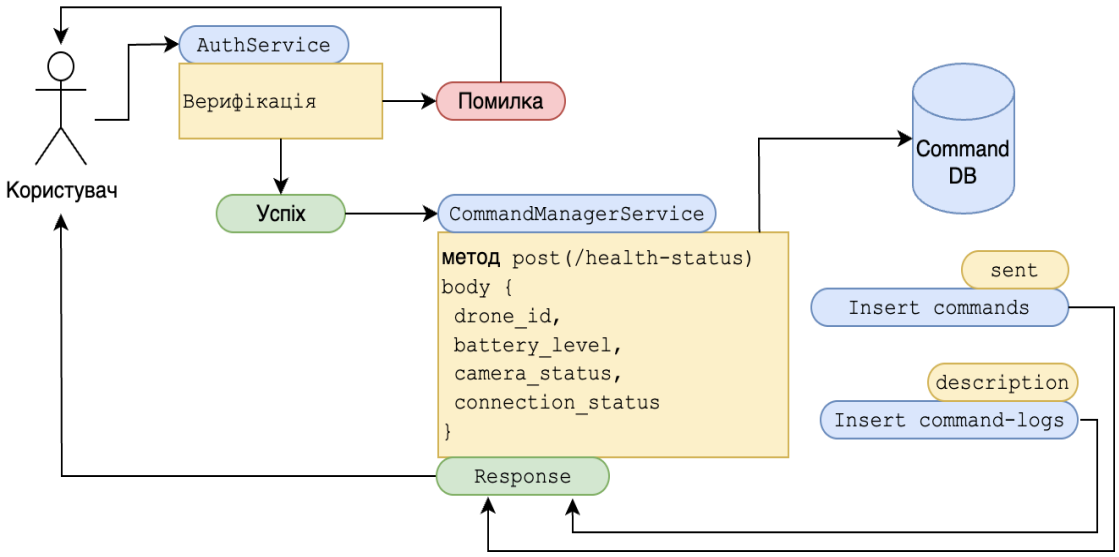


Рисунок 3.9 – Схема обробки стану дрона та генерування команд

3.9 Перегляд історії дрона

Процес перегляду історії дрона в системі реалізується через взаємодію CommandManagerService та HealthService. Користувач надсилає запит до CommandManagerService через ендпоінт /drone-status/history/:id, вказуючи ідентифікатор дрона. CommandManagerService перевіряє наявність токена авторизації в кукі, забезпечуючи безпечний доступ. При успішній перевірці сервіс звертається до HealthService через API, передаючи токен і ідентифікатор дрона. HealthService виконує запит до бази даних PostgreSQL, відбираючи з таблиці drone_status_history записи про історію подій дрона, включаючи ідентифікатор історії, тип події, опис та час її виникнення, відсортовані за спаданням часу. Отримані дані повертаються до CommandManagerService, який передає їх користувачу разом із повідомленням про успішне виконання.

Схема ілюструє взаємодію між CommandManagerService, HealthService і базою даних PostgreSQL під час перегляду історії дрона, відображаючи послідовність запитів і передачу даних, представлена на рисунку 3.10.

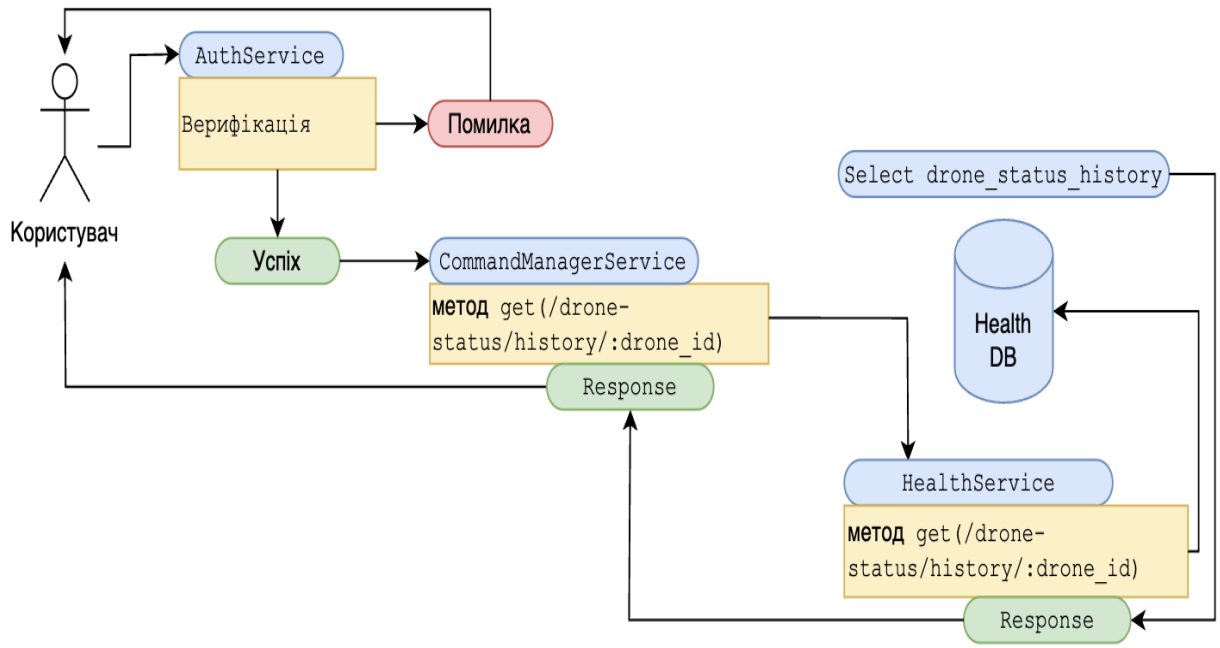


Рисунок 3.10 – Схема перегляду історії дрона

3.10 Оновлення статусу команди

Процес оновлення статусу команди в системі реалізується через CommandManagerService. Користувач надсилає PATCH-запит до ендпоінта /command/:command_id/status, вказуючи ідентифікатор команди та новий статус у тілі запиту. CommandManagerService перевіряє коректність вхідних даних, зокрема чи належить статус до допустимих значень (pending, sent, executed, failed). У разі успішної перевірки сервіс оновлює статус команди в таблиці commands у базі даних PostgreSQL, а також встановлює час виконання, якщо статус змінено на executed. Якщо команда з вказаним ідентифікатором не знайдена, повертається помилка. Після оновлення статусу дія логується в таблиці command_logs із позначкою STATUS_UPDATED та описом зміни. Користувачу повертаються оновлені дані команди, включаючи ідентифікатор, ідентифікатор дрона, текст команди, статус, час створення та виконання, разом із повідомленням про успішне оновлення.

Схема ілюструє взаємодію між користувачем і CommandManagerService під час оновлення статусу команди, відображаючи послідовність перевірки, оновлення бази даних і логування, представлена на рисунку 3.11. Такий підхід забезпечує контроль і відстеження змін статусу команд, сприяючи ефективному управлінню операціями в системі.

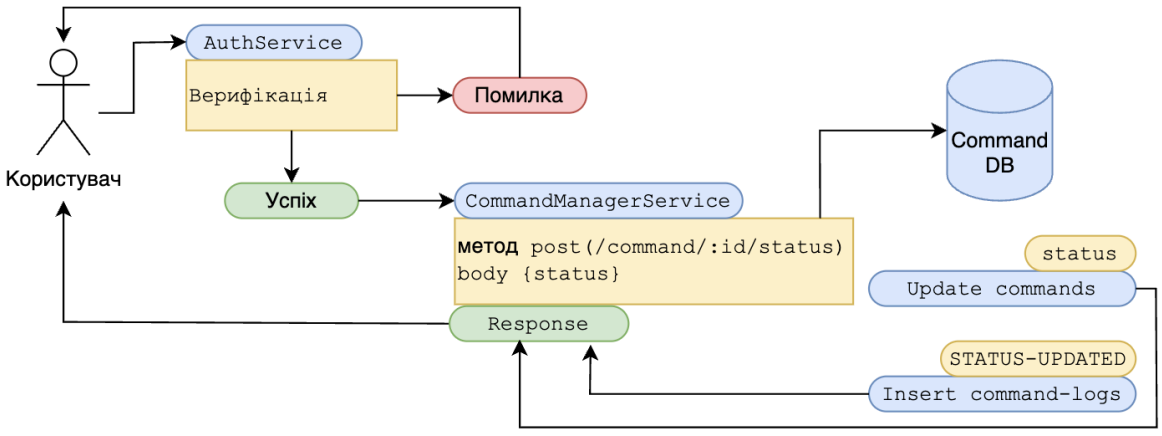


Рисунок 3.11 – Схема оновлення статусу команди

3.11 Безпека інформаційної системи

Користувач вводить свої облікові дані (електронну адресу та пароль) у форму авторизації, після чого клієнтський додаток надсилає POST-запит на ендпоінт /login. У тілі запиту передаються електронна адреса, пароль і, за потреби, параметр для повернення токена у форматі JSON.

Сервіс авторизації отримує ці дані та перевіряє їх. Спочатку виконується запит до бази даних MongoDB для пошуку користувача за електронною адресою. Якщо користувача не знайдено, повертається помилка. Далі порівнюється введений пароль із захешованим паролем, збереженим у базі за допомогою методу comparePassword, який використовує bcrypt (лістинг 3.11). У разі невідповідності пароля також повертається помилка авторизації.

```
userSchema.methods.comparePassword = async function
(candidatePassword) {
    return await bcrypt.compare(candidatePassword,
this.password);
};
```

Лістинг 3.11 – Метод для порівняння паролів

При успішній перевірці генерується JWT-токен, який включає ідентифікатор користувача, електронну адресу, роль (Admin, Operator або Analyst), а також час створення та термін дії токена. Токен підписується секретним ключом за допомогою бібліотеки jsonwebtoken. Після цього токен зберігається в cookie під назвою access_token з налаштуваннями безпеки: httpOnly, sameSite: strict і терміном дії 1 година. У відповідь клієнту повертається повідомлення про успішну авторизацію, ідентифікатор, електронна адреса та роль користувача.

Для подальшої взаємодії з системою клієнт додає отриманий токен у заголовок Authorization у форматі Bearer <token> кожного запиту до захищених ендпоінтів. Кожен сервіс (VideoProcessorService, AIService, CommandManagerService, HealthService) використовує middleware для

верифікації токена та визначення ролі користувача. Middleware перевіряє валідність токена, витягаючи з нього дані, і визначає, чи має користувач право виконати запит на основі його ролі. Якщо доступ дозволено, запит передається на обробку; у противному разі повертається помилка 403 Forbidden.

Дані, отримані з токена, передаються до відповідного сервісу для обробки запиту та взаємодії з базою даних MongoDB. Кожен новий користувач отримує свій обліковий запис у таблиці User бази даних із відповідною роллю, яка визначає його привілеї в системі.

Для забезпечення належного розмежування доступу та виконання завдань у системі визначено три ролі, кожна з яких відповідає за певний набір функцій. Роль Admin призначена для повного управління системою, включаючи адміністрування користувачів і контроль усіх процесів. Роль Operator створена для спеціалістів, які займаються операційною діяльністю, такою як обробка файлів і управління дронами, забезпечуючи їхній технічний супровід. Роль Analyst розроблена для тих, хто зосереджений на аналізі даних ШІ, що дозволяє ефективно працювати з результатами обробки файлів. Детальний розподіл функцій між цими ролями наведено у таблиці 3.1.

Таблиця 3.1 – Розподіл функцій між ролями

Функція	Admin	Operator	Analyst
Реєстрація	Так	Ні	Ні
Авторизація	Так	Так	Так
Верифікація токену	Так	Так	Так
Перегляд усіх відеофайлів	Так	Так	Так
Почати обробку файлу	Ні	Так	Ні
Завершити обробку файлу	Ні	Так	Ні
Почати аналіз файлу	Ні	Ні	Так
Перегляд результатів аналізу	Так	Ні	Так
Отримати стан дрона	Ні	Так	Ні
Отримати історію дрона	Так	Так	Ні
Перевірка стану дрона та створення команд	Так	Так	Ні

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

У цьому розділі описується послідовність дій для роботи з системою управління ройовим комплексом. Буде розглянуто авторизацію користувача, перегляд доступних відеофайлів, обробку відео, аналіз даних за допомогою ШІ, моніторинг стану та історії дрона, а також генерацію команд для дрона та оновлення їхнього статусу. Для демонстрації використовується інструмент Postman для відправлення запитів до системи [16].

4.1 Авторизація користувача

Процес авторизації користувача здійснюється через AuthService для отримання доступу до функціоналу системи. Здійснюється POST-запит за ендпоінтом http://localhost:3000/auth/login із зазначенням електронної адреси та пароля у тілі запиту (рис. 4.1).

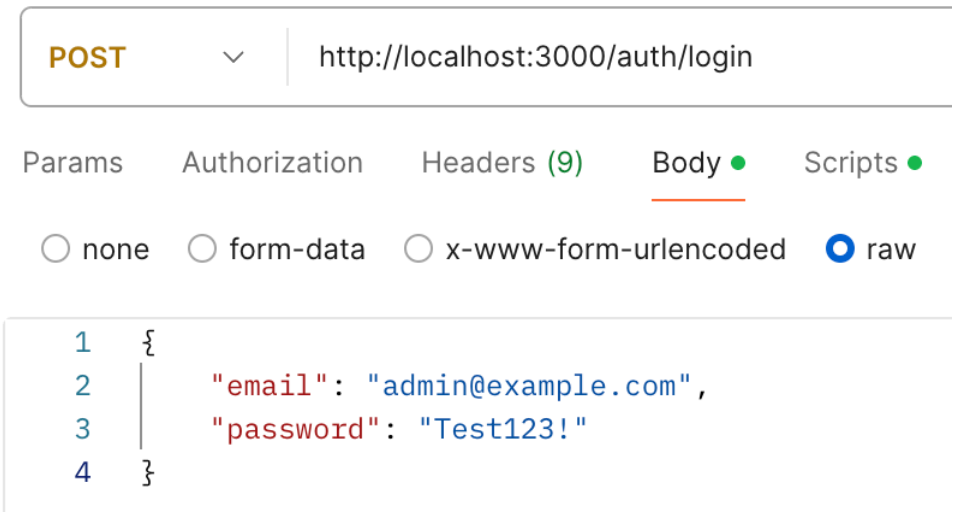


Рисунок 4.1 – POST-запит для авторизації користувача

У разі успішної авторизації повертається JWT-токен, який застосовується для подальших запитів. Відповідь включає ідентифікатор користувача, електронну адресу, роль і токен (рис. 4.2).

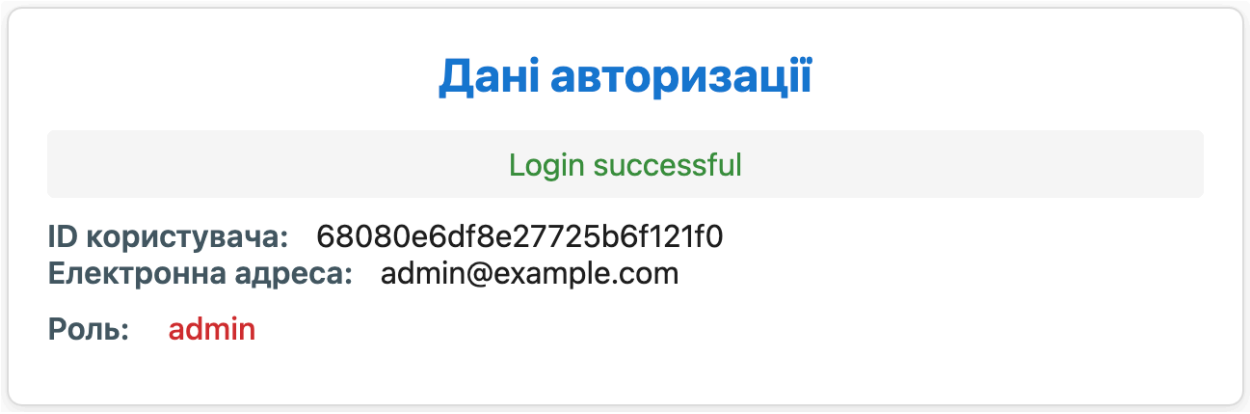


Рисунок 4.2 – Відповідь після авторизації

4.2 Перегляд доступних відеофайлів через CommandManagerService

Для перегляду доступних відеофайлів, що зберігаються в базі даних VideoProcessorService, у заголовок Cookie додається JWT-токен, отриманий під час авторизації (рис. 4.3). Далі з CommandManagerService надсилається GET-запит (рис. 4.4) до ендпоінту http://localhost:3002/api/command/videos. У відповідь VideoProcessorService повертає список відеофайлів із їхніми ідентифікаторами, назвами та статусами (рис. 4.5).

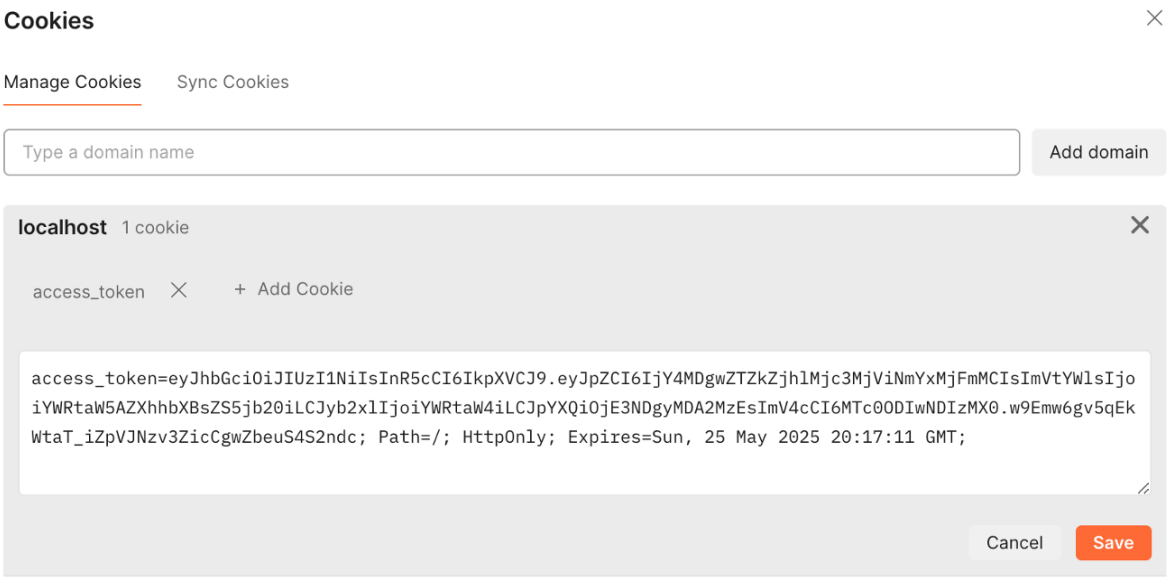


Рисунок 4.3 – Передача JWT-токена у заголовку Cookie

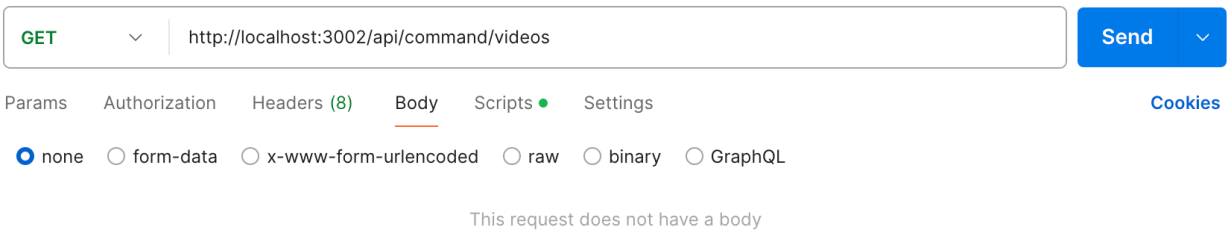
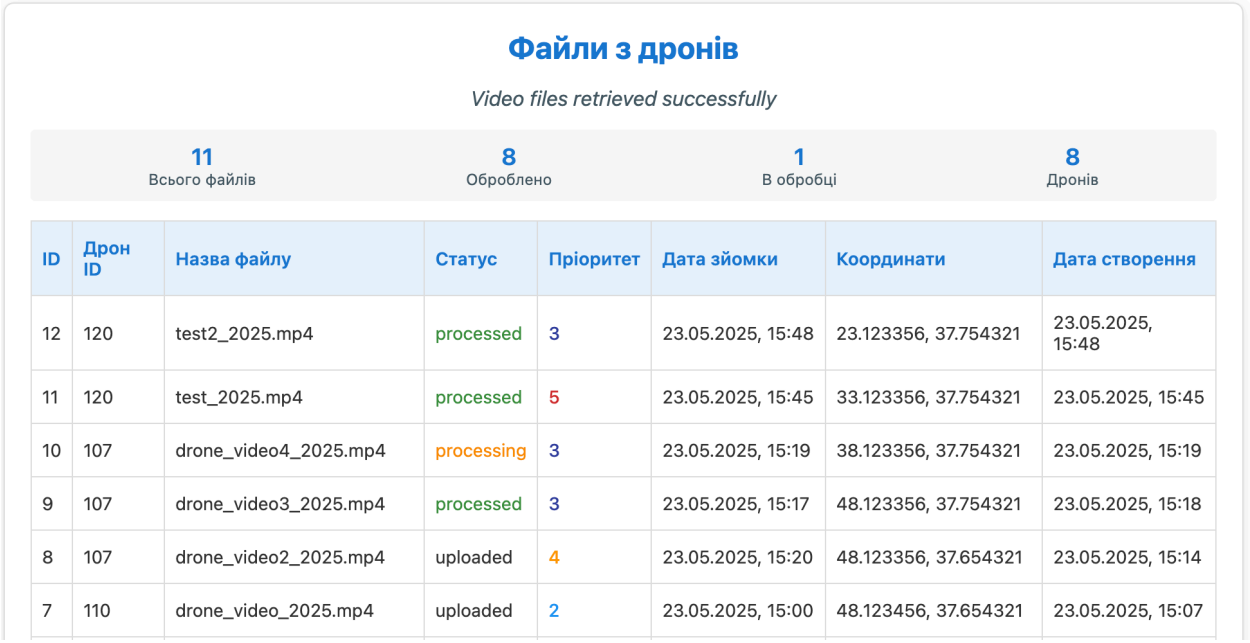


Рисунок 4.4 – GET-запит для перегляду доступних відеофайлів



ID	Дрон ID	Назва файлу	Статус	Пріоритет	Дата зйомки	Координати	Дата створення
12	120	test2_2025.mp4	processed	3	23.05.2025, 15:48	23.123356, 37.754321	23.05.2025, 15:48
11	120	test_2025.mp4	processed	5	23.05.2025, 15:45	33.123356, 37.754321	23.05.2025, 15:45
10	107	drone_video4_2025.mp4	processing	3	23.05.2025, 15:19	38.123356, 37.754321	23.05.2025, 15:19
9	107	drone_video3_2025.mp4	processed	3	23.05.2025, 15:17	48.123356, 37.754321	23.05.2025, 15:18
8	107	drone_video2_2025.mp4	uploaded	4	23.05.2025, 15:20	48.123356, 37.654321	23.05.2025, 15:14
7	110	drone_video_2025.mp4	uploaded	2	23.05.2025, 15:00	48.123456, 37.654321	23.05.2025, 15:07

Рисунок 4.5 – Відповідь із переліком відеофайлів

4.3 Обробка відеофайлу через VideoProcessorService

Для обробки обраного відеофайлу застосовується POST-запит за ендпоінтом http://localhost:3002/api/command/start-processing, який надсилається через CommandManagerService. У тілі запиту вказується ідентифікатор вибраного відео та відповідного дрона (рис. 4.6). Після цього CommandManagerService переспрямовує запит до VideoProcessorService, який розпочинає процес обробки відеофайлу: виконує необхідні дії згідно з логікою сервісу, оновлючи відповідний запис у базі даних, змінюючи статус файлу. У відповідь VideoProcessorService повертає

підтвердження успішного запуску обробки, а також оновлену інформацію про відеофайл, включаючи його поточний статус (рис. 4.7).

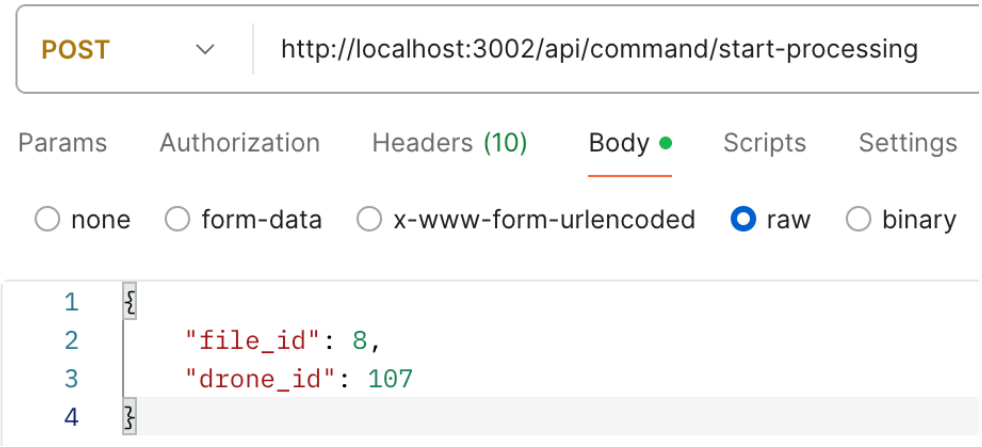


Рисунок 4.6 – POST-запит для обробки відеофайлу

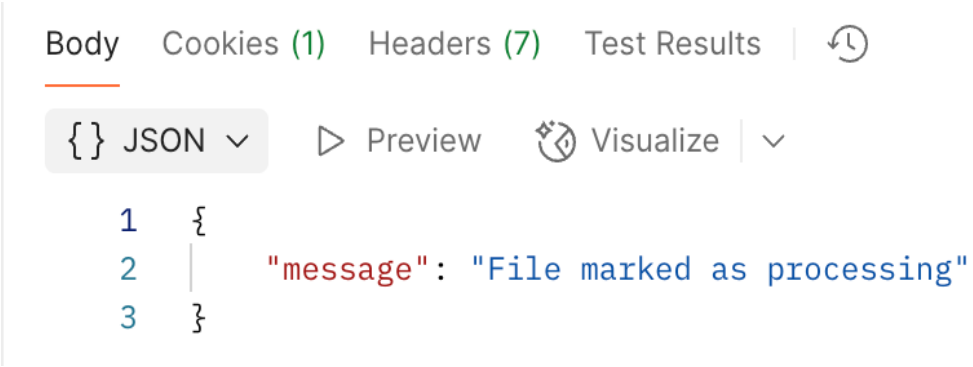


Рисунок 4.7 – Відповідь із результатом обробки відео

Після цього можна повторно виконати попередній метод, щоб перевірити, як змінився статус відеофайлу (рис. 4.8).

9	107	drone_video3_2025.mp4	processed	3	23.05.2025, 15:17	48.123356, 37.754321	23.05.2025, 15:18
8	107	drone_video2_2025.mp4	processing	4	23.05.2025, 15:20	48.123356, 37.654321	23.05.2025, 15:14
7	110	drone_video_2025.mp4	uploaded	2	23.05.2025, 15:00	48.123456, 37.654321	23.05.2025, 15:07

Рисунок 4.8 – Змінений статус файлу

Після завершення обробки відеофайлу для можливості подальшого його використання необхідно оновити статус файлу на processed. Для цього застосовується POST-запит до ендпоінту /api/command/complete-processing, який також надсилається з CommandManagerService. У тілі запиту передається лише ідентифікатор відеофайлу (file_id), що потребує зміни статусу (рис. 4.9).

CommandManagerService пересилає запит до VideoProcessorService, який оновлює статус відповідного запису у базі даних. У відповіді повертається підтвердження успішного завершення обробки та оновлена інформація про відеофайл із новим статусом «processed» (рис. 4.10).

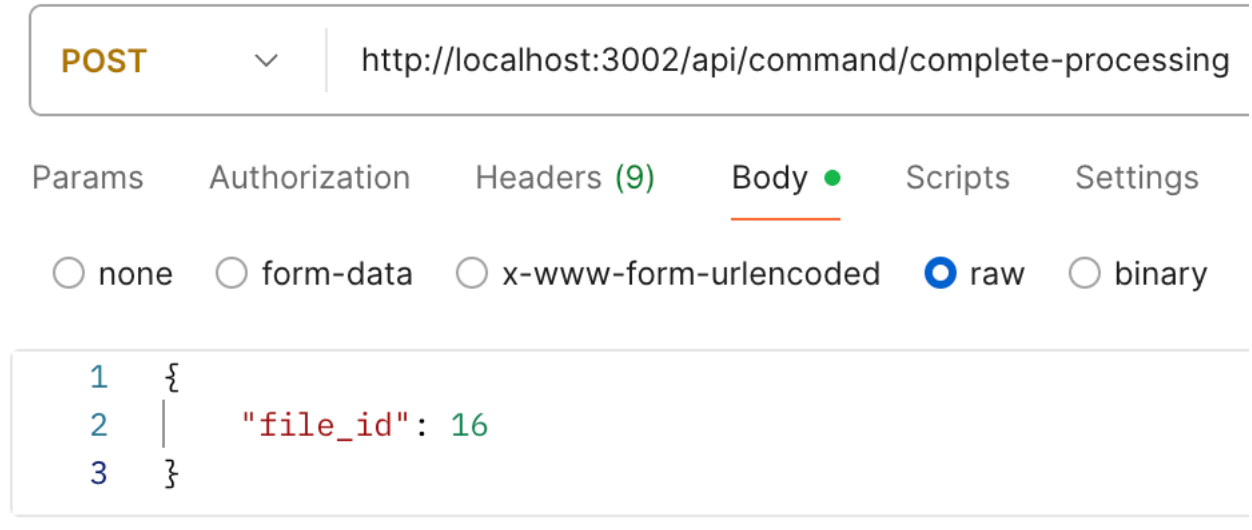


Рисунок 4.9 – POST-запит для завершення обробки відеофайлу

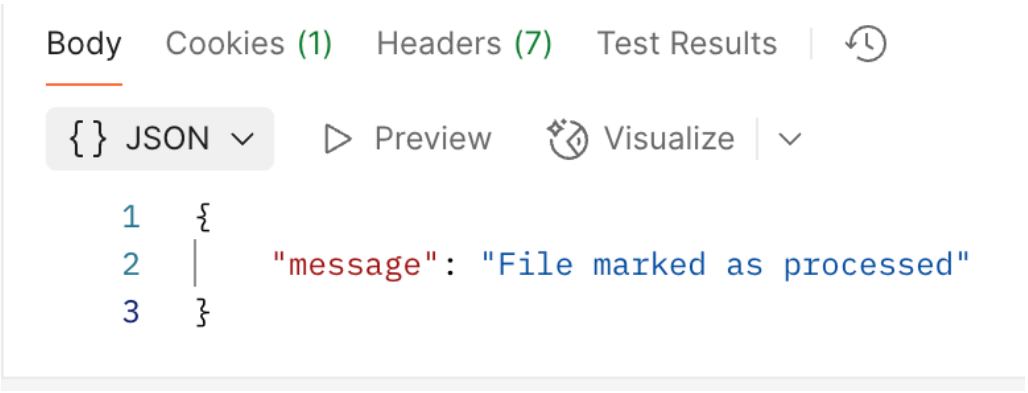


Рисунок 4.10 – Відповідь після зміни статусу на processed

За потреби можна повторно звернутися до методу отримання списку відеофайлів, щоб переконатися, що статус відповідного файлу було оновлено (рис. 4.11).

9	107	drone_video3_2025.mp4	processed	3	23.05.2025, 15:17	48.123356, 37.754321	23.05.2025, 15:18
8	107	drone_video2_2025.mp4	processed	4	23.05.2025, 15:20	48.123356, 37.654321	23.05.2025, 15:14
7	110	drone_video_2025.mp4	uploaded	2	23.05.2025, 15:00	48.123456, 37.654321	23.05.2025, 15:07

Рисунок 4.11 – Список відеофайлів після оновлення статусу

4.4 Аналіз відеофайлів через AIService

Для аналізу обробленого відео застосовується POST-запит через CommandManagerService до AIService за ендпоінтом http://localhost:3002/api/command/process-and-send. У тілі запиту передається ідентифікатор відеофайлу та відповідного дрона (рис. 4.12).

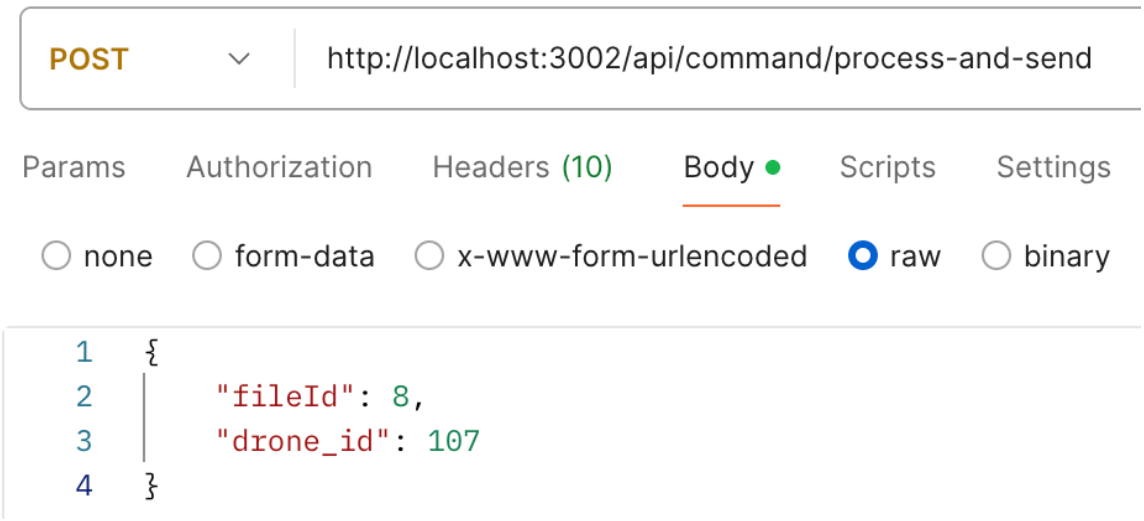


Рисунок 4.12 – POST-запит для аналізу відео через AIService

У відповідь AIService повертає повідомлення про успішне надсилання запиту на аналіз та ідентифікатор результату (рис. 4.13).

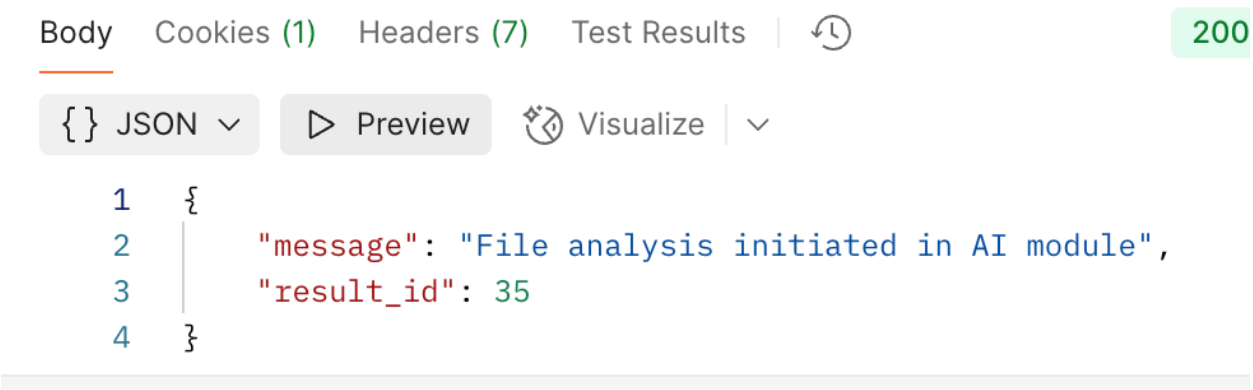


Рисунок 4.13 – Відповідь із ідентифікатором результату аналізу

Для перевірки результатів аналізу використовується GET-запит за ендпоінтом /api/command/ai-results/{result_id} (рис. 4.14).

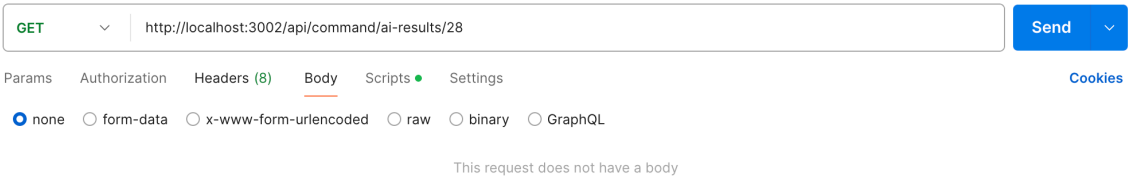


Рисунок 4.14 – GET-запит для перевірки результатів аналізу

У відповідь від AIService можливі кілька варіантів: якщо відеофайл ще перебуває в обробці, повертається повідомлення зі статусом «pending» (рис. 4.15).

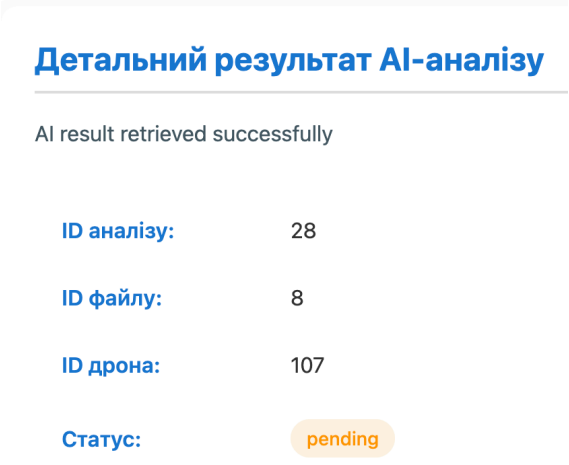


Рисунок 4.15 – Відповідь для відео в обробці

У разі успішного завершення аналізу – повідомлення зі статусом «completed» та відповідними результатами (рис. 4.16).

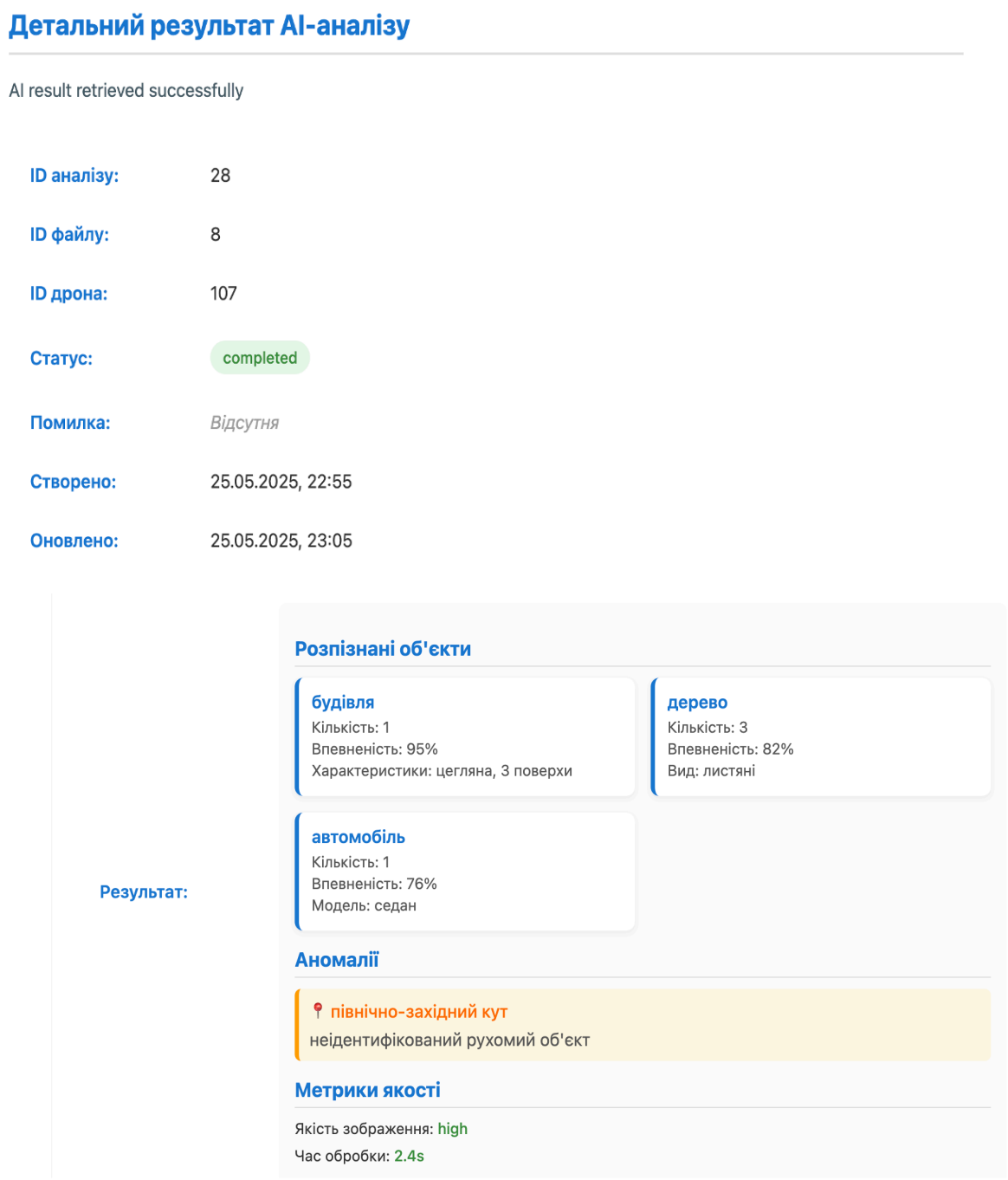


Рисунок 4.16 – Успішний результат аналізу відео

Якщо ж під час аналізу сталася помилка, наприклад, через некоректний формат відео, система повертає повідомлення зі статусом failed і вказаною причиною (рис. 4.17).

Рисунок 4.17 – Невдалий результат аналізу відео

4.5 Перегляд стану дрона через HealthService

Для перевірки поточного технічного стану дрона використовується GET-запит, який надсилається через CommandManagerService до ендпоінту /api/command/drone-status/{drone_id}, де {drone_id} – це унікальний ідентифікатор відповідного пристрою (рис. 4.18).

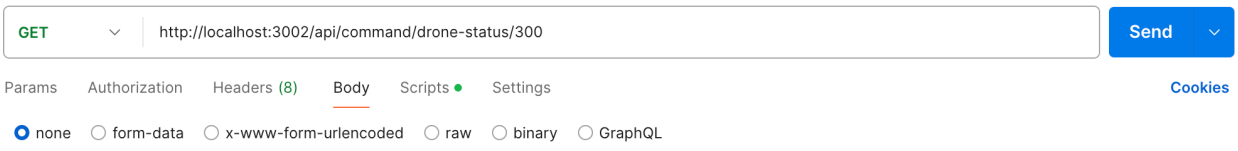


Рисунок 4.18 – GET-запит для перегляду стану дрона

Після отримання запиту CommandManagerService звертається до спеціалізованого сервісу моніторингу – HealthService, який здійснює перевірку зазначеного дрона та повертає у відповідь структуровані дані. Зокрема, у відповіді міститься ідентифікатор дрона, його умовна назва, поточний рівень заряду акумулятора у відсотках, статус вбудованої камери (активна або ні), наявність з’єднання з системою, а також мітка часу, що вказує момент останнього оновлення стану (рис. 4.19). Отримані дані дозволяють оперативно оцінити працездатність дрона перед виконанням завдань або у разі виникнення збоїв у роботі.

Рисунок 4.19 – Відповідь із поточним станом дрона

4.6 Генерація команд для дрона через CommandManagerService

На основі отриманої інформації про стан дрона формується перелік відповідних команд для подальших дій. Для цього виконується POST-запит через CommandManagerService до ендпоінту /api/root/health-status, у тілі якого передаються структуровані дані про поточний технічний стан дрона, отримані на попередньому етапі. Рисунок 4.20 демонструє приклад такого запиту.

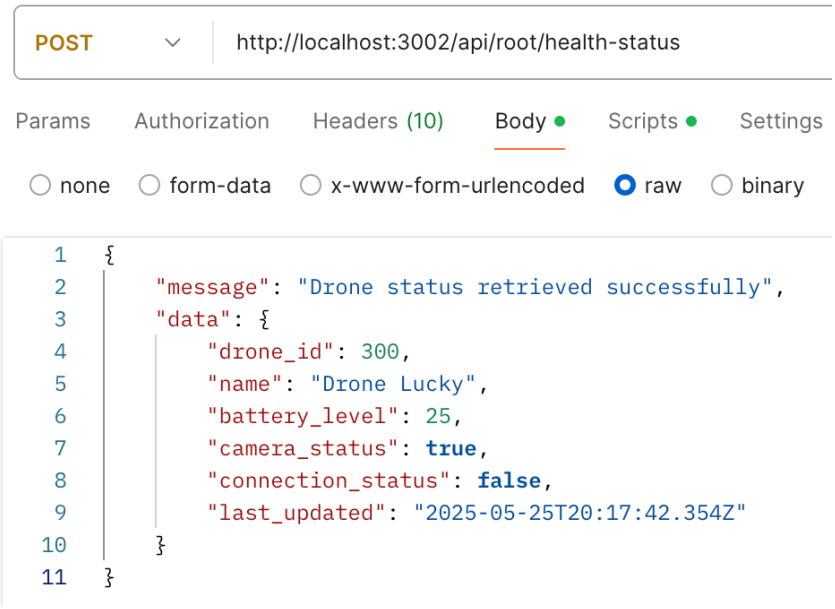


Рисунок 4.20 – POST-запит для генерації команд для дрона

Після обробки вхідної інформації CommandManagerService автоматично аналізує параметри (рівень заряду, наявність зв’язку, статус камери) й генерує команди. У відповіді повертається список сформованих інструкцій із зазначенням їхніх ідентифікаторів, текстових описів, передбачених дій (наприклад спроба відновлення зв’язку) та пояснень до кожної команди з урахуванням контексту (рис. 4.21). Цей механізм дозволяє забезпечити оперативне реагування на критичні стани дрона в автоматичному режимі.

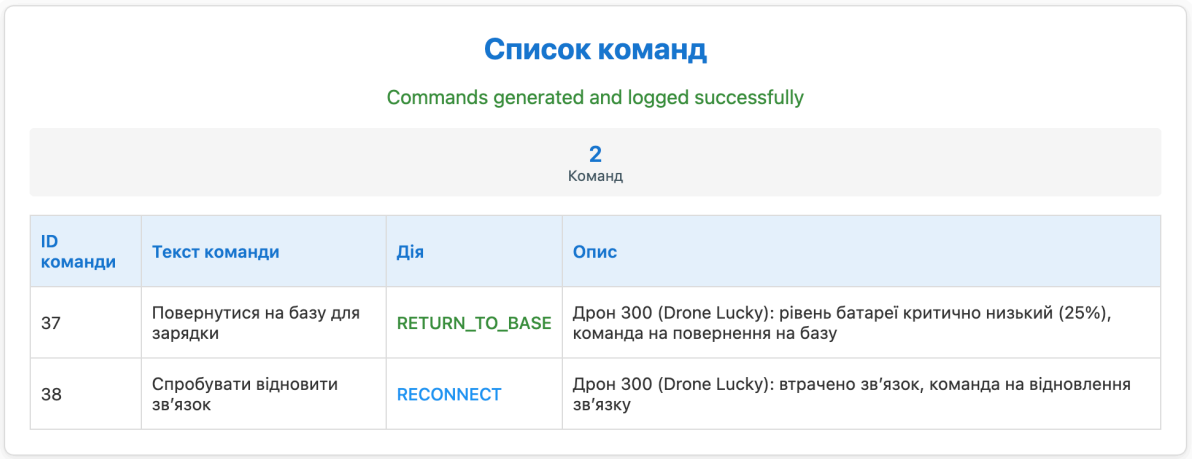


Рисунок 4.21 – Відповідь із переліком згенерованих команд

4.7 Оновлення статусу команди

Після виконання дронами відповідних інструкцій необхідно оновити статус команди, що була надіслана. Для цього через CommandManagerService виконується PATCH-запит до ендпоінту http://localhost:3002/api/root/command/{command_id}/status, де {command_id} – це ідентифікатор команди, яку потрібно оновити. У тілі запиту передається новий статус, що відображає результат виконання, наприклад «executed». Рисунок 4.22 ілюструє приклад такого запиту.

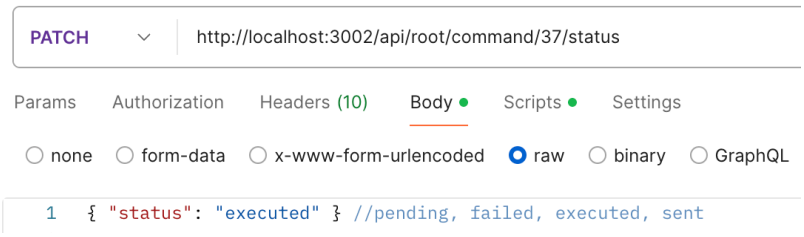


Рисунок 4.22 – PATCH-запит для оновлення статусу команди

У відповідь сервіс повертає підтвердження успішного оновлення разом із детальною інформацією про команду: її текст, пов'язаний дрон, поточний статус, а також часові мітки створення та фактичного виконання (рис. 4.23). Це дозволяє відслідковувати хід виконання завдань і фіксувати їхній результат у системі.

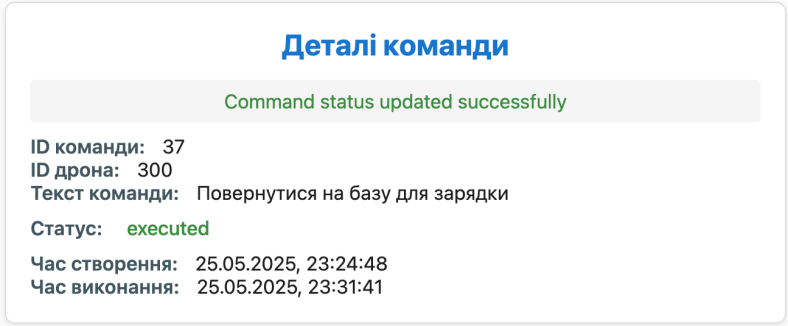


Рисунок 4.23 – Відповідь із оновленими даними команди

4.8 Перегляд історії дрона через HealthService

Для отримання повної картини технічного стану дрона в динаміці використовується GET-запит через CommandManagerService до ендпоінту /api/command/drone-status/history/{drone_id}, де {drone_id} – ідентифікатор дрона, для якого потрібно отримати історію подій. Рисунок 4.24 демонструє приклад такого запиту.

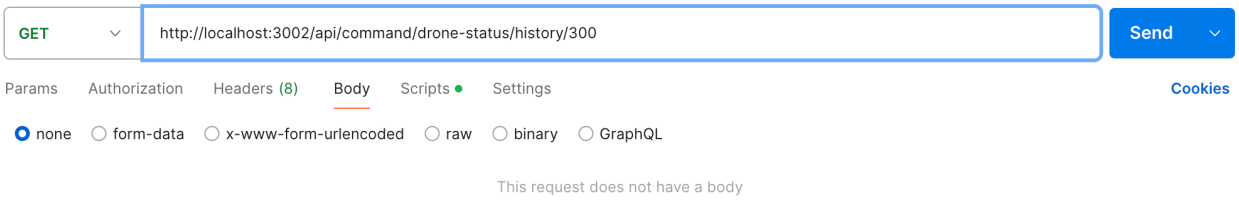


Рисунок 4.24 – GET-запит для перегляду історії дрона

У відповідь HealthService повертає відсортований за спаданням список подій, пов’язаних із дроном, кожна з яких містить тип події, її опис і точний час виникнення. Це дозволяє проаналізувати попередні стани дрона, зокрема виникнення технічних проблем, перебоїв у зв’язку або інші критичні ситуації, що можуть впливати на його працездатність (рис. 4.25).

Історія статусу дрона (ID: 300)

Drone status history retrieved successfully

ID події	Тип події	Опис	Час
41	BATTERY_LOW	Рівень батареї критично низький: 25%	25 трав. 2025 р., 23:17:42
42	CONNECTION_LOST	Втрачено зв’язок із дроном	25 трав. 2025 р., 23:17:42
43	CAMERA_ON	Камера активна	25 трав. 2025 р., 23:17:42

Рисунок 4.25 – Відповідь із історією подій дрона

Розглянуті дії демонструють комплексне функціонування системи, яка охоплює весь цикл взаємодії з дроном – від аутентифікації користувача до аналізу відео та прийняття рішень на основі отриманих даних. Система забезпечує авторизацію, обробку й аналіз відеофайлів, моніторинг технічного стану дронів у режимі реального часу, а також генерацію та виконання керівних команд залежно від ситуації.

Кожен з мікросервісів – AuthService, VideoProcessorService, AIService, HealthService і CommandManagerService – виконує чітко визначену роль у межах загальної архітектури, що дозволяє досягти високої модульності, гнучкості та масштабованості рішення. Завдяки злагодженій взаємодії між сервісами, оператор отримує надійний інструмент для ефективного управління роєм дронів, що може застосовуватись як у повсякденному моніторингу, так і в умовах НС або в складних польових умовах.

Важливо, що система також дозволяє виявляти критичні стани (наприклад, низький заряд батареї чи втрату зв’язку), автоматично реагувати на них за допомогою згенерованих команд, а також зберігати історію подій для подальшого аналізу та оптимізації стратегії керування. Таким чином, представлена архітектура є надійною основою для побудови інтелектуальної платформи керування автономними безпілотними апаратами в реальному середовищі.

ВИСНОВКИ

У результаті аналізу предметної області сформульовано мету створення системи управління ройовим комплексом на основі мультиагентної технології.

Визначено ключові особливості предметної області та сформульовано перелік задач, необхідних для функціонування системи, зокрема автентифікація користувачів, обробка мультимедійних даних, аналіз ШІ, моніторинг стану дронів і генерація команд. Окрім цього, визначено основні ролі користувачів: оператор, адміністратор і AI-спеціаліст.

Для реалізації проєкту обрано мікросервісну архітектуру, що забезпечує модульність, гнучкість і масштабування. Особливістю системи є мультиагентний підхід, який підтримує координацію між агентами (дронами та операторами) через чітко визначені API, дозволяючи обробляти оперативні задачі в реальному часі. Зберігання даних організовано з використанням СУБД MongoDB для автентифікаційних даних і PostgreSQL для операційних даних. Взаємодія між мікросервісами (AuthService, CommandManagerService, VideoProcessorService, AIService, HealthService) стандартизована через REST API з використанням OpenAPI. Для серверної логіки застосована мова програмування JavaScript (Node.js) із фреймворком Express, а автентифікація реалізована за допомогою JSON Web Tokens (JWT).

У ході розробки створено розподілену систему, яка забезпечує ефективну координацію між дронами та операторами через матку рою.Для обміну даними між дронами та мікросервісами використано Fast DDS із серіалізацією Protocol Buffers, а відеопотоки обробляються через GStreamer із кодеком H.264. Механізми QoS у Fast DDS дозволяють регулювати пріоритетність даних, а буферизація забезпечує стійкість до перебоїв зв’язку.

Безпека системи гарантується автентифікацією через JWT-токени та розмежуванням доступу на основі ролей, що захищає чутливі дані, такі як телеметрія дронів і відеопотоки, від несанкціонованого доступу. Чітке визначення прав користувачів забезпечує контроль над діями операторів і безпеку інформації.

Розроблена система успішно реалізує функціонал, визначений на етапі постановки задач. Вона забезпечує авторизацію, обробку та аналіз мультимедійних даних, моніторинг стану дронів, генерацію команд і координацію роботи рою в реальному часі, що дозволяє ефективно застосовувати її в надзвичайних ситуаціях.

Система має перспективи для розвитку та розширення:

1) інтеграція алгоритмів автоматичної реконфігурації рою для адаптації до змінних умов;

2) розширення AIService для аналізу складніших сценаріїв, наприклад, прогнозування аварій;

3) розробка інтерактивного інтерфейсу для операторів із візуалізацією телеметрії в реальному часі;

4) додавання функціоналу для автоматичного керування пріоритетами команд залежно від критичності подій.

Перспективи розвитку системи є широкими, оскільки вона створює основу для управління ройовими комплексами в умовах надзвичайних ситуацій. Подальша інтеграція новітніх технологій і розширення функціоналу сприятимуть підвищенню її ефективності та конкурентоспроможності.

За результатами кваліфікаційної роботи опубліковані тези доповіді на двадцять другій всеукраїнській конференції студентів і молодих науковців «Інформатика, інформаційні системі та технології» [3].

Репозиторій з кодом програми розміщено на платформі GitHub [18].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 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. – C. 268–277. [Електронний ресурс]. – DOI: https://doi.org/10.15276/hait.03.2021.6
2. Швець Ю. О., Малахов Є. В., Козлов М. С. Хмарна система підтримки ройового комплексу // Інформатика, інформаційні системи та технології: тези доповідей XX Всеукраїнської конференції студентів і молодих науковців. – Одеса, 26 квітня 2024 р. – Одеса, 2024. – С. 134–136.
3. Куликов В. В., Шпінарева І. М. Розробка мікросервісної архітектури для керування роєм дронів // Інформатика, інформаційні системи та технології: тези доповідей XXII Всеукраїнської конференції студентів і молодих науковців. – Одеса, 25 квітня 2025 р. – Одеса, 2025. – С. 201–202.
4. Касілов О.В. Мультиагентні системи та технології в ігрових додатках: довідник модуля. / О.В.Касілов. – Х.: «Друкарня Мадрид», 2018. - 82 с
5. FlytBase [Електронний ресурс]. – Режим доступу: https://www.flytbase.com/
6. DJI FlightHub 2 [Електронний ресурс]. – Режим доступу: https://enterprise.dji.com/flighthub-2
7. DroneDeploy [Електронний ресурс]. – Режим доступу: https://www.dronedeploy.com/
8. Evo Play. Теоретичні аспекти роботи Raspberry Pi Zero 2 W [Електронний ресурс]. – Режим доступу: https://evo.net.ua/ru/pod-kapotom-u-raspberry-pi-zero-2w/
9. Arducam. Огляд технічних характеристик камери OV5647 для Raspberry Pi [Електронний ресурс]. – Режим доступу: https://evo.net.ua/ru/kamera-arducam-ov5647-5-mp-1080p-mini-camera-for-raspberry-pi/
10. GStreamer: gst-launch tool documentation [Електронний ресурс]. – Режим доступу: https://gstreamer.freedesktop.org/documentation/tools/
11. eProsima Fast DDS Documentation [Електронний ресурс]. – Режим доступу: https://fast-dds.docs.eprosima.com/en/latest/

12. Node.js [Електронний ресурс]. – Режим доступу: https://nodejs.org/uk
13. Express.js [Електронний ресурс]. – Режим доступу: https://expressjs.com/
14. PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/
15. MongoDB [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/
16. OpenAPI Generator [Електронний ресурс]. – Режим доступу: https://openapi-generator.tech/
17. Postman. Документація та інструменти для тестування API [Електронний ресурс]. – Режим доступу: https://www.postman.com/
18. Репозиторій проєкту на GitHub [Електронний ресурс]. – Режим доступу: https://github.com/kulikovjob/drone_managment_system

ДОДАТОК А

YAML – специфікація всіх мікросервісів

Специфікація YAML для AuthService

```
openapi: 3.0.0
info:
  title: Auth Service API
  version: 1.0.0
servers:
  - url: http://localhost:3000
paths:
  /auth/register:
    post:
      summary: Register a new user
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required:
                - email
                - password
                - role
              properties:
                email:
                  type: string
                  format: email
                password:
                  type: string
                  format: password
                role:
                  type: string
      responses:
        '201':
          description: User registered
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                  user:
                    type: object
                    properties:
                      id:
```

Лістинг А.1 – YAML для AuthService

```
                    type: string
                  email:
                    type: string
                  role:
                    type: string
      '400':
        description: Bad request
        content:
          application/json:
            schema:
              type: object
              properties:
                error:
                  type: string
      '500':
        description: Server error

/auth/login:
  post:
    summary: Log in user and return token
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            required:
              - email
              - password
            properties:
              email:
                type: string
                format: email
              password:
                type: string
              return_json:
                type: boolean
    responses:
      '200':
        description: Login successful
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
                identity:
                  type: object
```

Лістинг А.1, аркуш 2

```
                properties:
                  id:
                    type: string
                  email:
                    type: string
                  role:
                    type: string
              token:
                type: string
              token_type:
                type: string
              expires_in:
                type: integer
      '400':
        description: Missing credentials
      '401':
        description: Invalid credentials

/auth/verify:
  post:
    summary: Verify JWT token
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            required:
              - token
            properties:
              token:
                type: string
    responses:
      '200':
        description: Token is valid
        content:
          application/json:
            schema:
              type: object
              properties:
                valid:
                  type: boolean
                identity:
                  type: object
                  properties:
                    id:
                      type: string
                    email:
                      type: string
```

Лістинг А.1, аркуш 3

```
                    role:
                      type: string
        '400':
          description: Token missing
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    type: string
        '401':
          description: Invalid or expired token
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    type: string

components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
```

Лістинг А.1, аркуш 4

Специфікація YAML для VideoProcessorService

```
openapi: 3.0.3
info:
  title: Video Processor API
  description: API for managing video file processing
  version: 1.0.0
  contact:
    name: Support
    email: support@example.com
servers:
  - url: http://localhost:3001/api/video
    description: Local development server
components:
  securitySchemes:
    cookieAuth:
      type: apiKey
      in: cookie
```

Лістинг А.2 – YAML для VideoProcessorService

```
      name: access_token
  schemas:
    File:
      type: object
      properties:
        id:
          type: integer
        file_type:
          type: string
        status:
          type: string
        priority:
          type: integer
        created_at:
          type: string
          format: date-time
    Error:
      type: object
      properties:
        error:
          type: string
security:
  - cookieAuth: []
paths:
  /all:
    get:
      summary: Retrieve all video files
      security:
        - cookieAuth: []
      responses:
        '200':
          description: List of video files
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                  data:
                    type: array
                    items:
                      $ref: '#/components/schemas/File'
        '500':
          description: Server error
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
  /to-process:
```

Лістинг А.2, аркуш 2

```
  get:
    summary: Retrieve files ready for processing
    security:
      - cookieAuth: []
    responses:
      '200':
        description: List of files to process
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
                data:
                  type: array
                  items:
                    $ref: '#/components/schemas/File'
      '500':
        description: Server error
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
/start-processing:
  post:
    summary: Start processing a file
    security:
      - cookieAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              file_id:
                type: integer
            required:
              - file_id
    responses:
      '200':
        description: File marked as processing
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
```

Лістинг А.2, аркуш 3

```
      '400':
        description: Bad request
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
      '500':
        description: Server error
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
/complete-processing:
  post:
    summary: Complete processing a file
    security:
      - cookieAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              file_id:
                type: integer
            required:
              - file_id
    responses:
      '200':
        description: File marked as processed
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
      '400':
        description: Bad request
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
      '500':
        description: Server error
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
```

Лістинг А.2, аркуш 4

```
/file-info:
  post:
    summary: Get file information
    security:
      - cookieAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              fileId:
                type: integer
            required:
              - fileId
    responses:
      '200':
        description: File information retrieved
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
                data:
                  type: object
                  properties:
                    id:
                      type: integer
                    file_type:
                      type: string
                    status:
                      type: string
                    priority:
                      type: integer
                    created_at:
                      type: string
                      format: date-time
      '400':
        description: Bad request
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Error'
      '404':
        description: File not found
        content:
          application/json:
```

Лістинг А.2, аркуш 5

```
            schema:
              $ref: '#/components/schemas/Error'
        '500':
          description: Server error
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
```

Лістинг А.2, аркуш 6

Специфікація YAML для AIService

```
openapi: 3.0.3
info:
  title: AI Module API
  description: API for AI analysis of video files
  version: 1.0.0
servers:
  - url: http://localhost:3003/ai
components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
security:
  - bearerAuth: []
paths:
  /analyze:
    post:
      summary: Initiate analysis of a video file
      security:
        - bearerAuth: []
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                file_id:
                  type: integer
                drone_id:
                  type: integer
              required: [file_id, drone_id]
      responses:
        '200':
```

Лістинг А.3 – YAML для AIService

```
        description: Analysis initiated
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
                result_id:
                  type: integer
      '400':
        description: Bad request
        content:
          application/json:
            schema:
              type: object
              properties:
                error:
                  type: string
      '500':
        description: Server error
        content:
          application/json:
            schema:
              type: object
              properties:
                error:
                  type: string
/results/{resultId}:
  get:
    summary: Get analysis result
    security:
      - bearerAuth: [ ]
    parameters:
      - name: resultId
        in: path
        required: true
        schema:
          type: integer
      - name: Content-Type
        in: header
        required: false
        schema:
          type: string
          default: application/json
    responses:
      '200':
        description: Result retrieved
        content:
          application/json:
```

Лістинг А.3, аркуш 2

```
            schema:
              type: object
              properties:
                message:
                  type: string
                data:
                  type: object
      '404':
        description: Result not found
        content:
          application/json:
            schema:
              type: object
              properties:
                error:
                  type: string
      '500':
        description: Server error
        content:
          application/json:
            schema:
              type: object
              properties:
                error:
                  type: string
```

Лістинг А.3, аркуш 3

Специфікація YAML для HealthService

```
openapi: 3.0.3
info:
  title: Health Service API
  description: API for monitoring drone health status in a swarm
system
  version: 1.0.0
servers:
  - url: http://localhost:3004
    description: Local Health Service
security:
  - cookieAuth: []
paths:
  /health/drone-status/{drone_id}: get:
      summary: Get current status of a drone
      description: Retrieves the current health status of a
specific drone by its ID.
      operationId: getDroneStatus parameters:
```

Лістинг А.4 – YAML для HealthService

```
        - name: drone_id
          in: path
          required: true
          schema:
            type: integer
            example: 103
      responses:
        '200':
          description: Successful response
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/DroneStatus'
        '401':
          description: Unauthorized
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
        '404':
          description: Drone not found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
        '500':
          description: Internal server error
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
  /health/drone-status/history/{drone_id}:
    get:
      summary: Get status history of a drone
      description: Retrieves the history of status events for a
specific drone by its ID.
      operationId: getDroneStatusHistory
      parameters:
        - name: drone_id
          in: path
          required: true
          schema:
            type: integer
            example: 103
      responses:
        '200':
          description: Successful response
          content:
            application/json:
              schema:
```

Лістинг А.4, аркуш 2

```
                type: array
                items:
                  $ref:
'#/components/schemas/DroneStatusHistory'
        '401':
          description: Unauthorized
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
        '500':
          description: Internal server error
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
components:
  securitySchemes:
    cookieAuth:
      type: apiKey
      in: cookie
      name: access_token
      description: JWT token passed in Cookie header as
`access_token=<token>`
  schemas:
    DroneStatus:
      type: object
      properties:
        drone_id:
          type: integer
          example: 103
        name:
          type: string
          example: Drone Alpha
        battery_level:
          type: integer
          minimum: 0
          maximum: 100
          example: 85
        camera_status:
          type: boolean
          example: true
        connection_status:
          type: boolean
          example: true
        last_updated:
          type: string
          format: date-time
          example: 2025-05-13T22:10:00+03:00
      required:
```

Лістинг А.4, аркуш 3

```
    - drone_id
    - name
    - battery_level
    - camera_status
    - connection_status
    - last_updated
DroneStatusHistory:
  type: object
  properties:
    history_id:
      type: integer
      example: 1
    drone_id:
      type: integer
      example: 103
    event_type:
      type: string
      example: BATTERY_NORMAL
    event_description:
      type: string
      nullable: true
    event_timestamp:
      type: string
      format: date-time
      example: 2025-05-13T22:10:00+03:00
  required:
    - history_id
    - drone_id
    - event_type
    - event_timestamp
Error:
  type: object
  properties:
    error:
      type: string
      example: Drone not found
  required:
    - error
```

Лістинг А.4, аркуш 4

Olga Roznovets
Digitally signed by Olga Roznovets
Date: 2025.06.12 14:30:10 +03'00'