Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ТЕСТОВИХ ПРОГРАМ ДЛЯ КЕРУВАННЯ БАГАТОАГЕНТНОЮ СИСТЕМОЮ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-2
Дірявченко Є.О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Кузьомін О.Я.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дірявченку Єгору Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка тестових програм для керування багатоагентною системою

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 05 червня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, бібліотека з відкритим кодом OpenCV.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз принципів побудови багатоагентних систем ті їх застосування.
2. Дослідження можливостей LLM для інтерактивного керування агентами.
3. Розробка вебзастосунку для симуляції керування дронами в реальному часі.
4. Інтеграція інтелектуального чат-інтерфейсу та візуалізації агентів на мапі.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)
Актуальність задачі, постановка задачі, рисунки багатоагентної тестової системи у виді симуляції польоту дронів.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Вибір технологічного стеку	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Кузьомін О.Я.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 75 с., 1 табл., 14 рис., 4 дод., 31 джерело.

СИМУЛЯЦІЯ КЕРУВАННЯ ДРОНІВ, ВЕБЗАСТОСУНОК, ВЕБРОЗРОБКА, PYTHON, JAVASCRIPT, LLM, REACT, GOOGLE MAP API.

Об’єктом роботи є вебзастосунок для керування дронами через LLM модель та можливостями керування вручну.

Метою роботи є розробка інтелектуальної системи керування агентами у вигляді дронів у real-time системі. Де за допомогою чату з AI – моделлю здійснюється керування агентами для виповнення певних цілей.

Для вирішення проблем даної тематики роботи був проведений аналіз великих мовних моделей, та аналіз існуючих готових рішень на дану тематику для повного розуміння поставленої задачі та виявлення хороших та поганих рішень. Це допомогло у розробці вебзастосунку для якого були використані найкращі рішення.

Результат цієї роботи це створення вебзастосунку для керування багатоагентною системою керування дронів.

SIMULATION OF KERUVANNYA DRONIV, WEBZASTOSUNOK, WEBROZOBKA, PYTHON, JAVASCRIPT, LLM, REACT, GOOGLE MAP API.

The object of the robot is a web-based device for caravanning by drones through the LLM model and the possibility of caravanning manually.

The method of work is the development of an intelligent system for monitoring agents in the form of drones in a real-time system. In addition to the chat with the AI model, agents are used for advanced purposes.

To solve the problems of this topic, an analysis of large language models was conducted, and an analysis of existing ready-made solutions on this topic was conducted to fully understand the task and identify good and bad solutions. This helped in the development of a web application for which the best solutions were used.

The result of this work is the creation of a web application for managing a multi-agent drone control system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМОВЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

PC – Personal Computer (персональний комп’ютер)

JSX – JavaScript XML

HTML – HyperText Markup Language (гіпертекст розмітки мови)

LLM – Large Language Model (велика мовна модель)

ШІ – штучний інтелект

NP – Neurolinguistic Programming (нейролінгвістичне програмування)

ID – персональний номер

XML – EXtensible Markup Language (розширювана мова розмітки)

ВСТУП

На сьогоднішній день сучасні технології дуже стрімко розвиваються. Одна з таких технологій є штучним інтелектом, де він надає великий спектр послуг для звичайних людей. Починаючи від чатів спілкування з ШІ моделями і до керування будь якими об’єктами.

Першу публічну відому модель штучного інтелекту випустили 30 листопада 2022 року і це був Chat GPT розроблений компанією OpenAI. Після цього у короткі терміни з’явилося безліч нових застосунків з різними можливостями, включно з відкритим кодом, а це дало можливість встановити ШІ модель на свій PC. Але чат це не всі можливості ШІ, також їх інтегрують у реальні системи керування, автоматизацією процесів, або управління фізичними об’єктами, такими як дрони, роботи або транспортні засоби.

Кваліфікаційна робота з розробки тестових програм для керування багатоагентної системи буде мати вигляд симуляції керування дронів за допомогою ШІ, де кожен агент має вигляд дрону. Це буде непоганим прикладом того як штучний інтелект може бути інтегрований в сучасне життя людей. Завдяки цьому проєкту з’являться можливості для створення автономних дронів – кур’єрів для доставки вантажів, або моніторинг територій для пошукових операцій чи картографування місцевості, а також розважальні шоу з роєм дронів.

1 ОГЛЯД LLM МОДЕЛЕЙ

Великі мовні моделі – це моделі штучного інтелекту які навчаються за допомогою глибокого навчання на величезних об’ємах інформації. Основа їх структури є трансформер – це спеціальна архітектура нейронних мереж що складається з енкодерів та декодерів. Ці компоненти беруть інформацію з послідовного текста та розуміють співвідношення слів та фраз у тексті.

На відміну від рекурентних нейронних мереж, які обробляють інформацію послідовно, великі мовні моделі здатні аналізувати всі слова в реченні одночасно, паралельно. Такий підхід забезпечується завдяки трансформерній архітектурі, що дозволяє значно пришвидшити аналіз тексту. Саме ця особливість зробила можливим ефективне використання графічних процесорів (GPU) для навчання та виконання моделей. Великі мовні моделі дуже глибокі, одна модель може вирішувати безліч різних задач, наприклад: відповідати на запитання, узагальнювати документи або взагалі керувати фізичним об’єктом.

Також мовні моделі мають декілька переваг, їх первинна здібність це генерувати текст який дуже схожий на людський, вони можуть генерувати код, будь якої мови програмування якщо вона є в базі даних моделі, а також виконувати якість інструкції. Це і робить мовні моделі такими універсальними та корисними у повсякденному житті.

А якщо говорити про нейронні мережі загалом то це, методи, які навчають РС обробляти інформацію так як, це робить наш мозок. Цей процес називають глибоким навчанням, який використовує взаємозв’язані вузли або нейрони що нагадує людський мозок.

Основним вузлом цієї архітектури є self-attention mechanism (механізм само уваги). Це дає змогу моделі робити фокусування на певних частинах тексту

де б ця частина тексту не знаходилася у послідовності. Тому трасформери і працюють з великими текстами не втрачаючи суть слів або речень.

Великі мовні моделі також мають великий потенціал у сфері систем керування, в цілому це прийняття рішень на якихось факторах, як приклад це аналіз стану системи чи оцінка сценарію з прийняттям оптимального рішення.

1.1 Багатоагентні системи (MAS)

Сьогодні багатоагентні системи (MAS) дедалі частіше розглядаються як ефективний інструмент для аналізу, моделювання та проєктування різноманітних систем [1]. Цей напрям досліджень тісно пов’язаний зі створенням інтелектуального програмного забезпечення, що ґрунтується на концепції агентів автономних елементів, здатних діяти самостійно й взаємодіяти між собою. Хоча MAS вже продемонстрували свою ефективність у багатьох прикладних сферах, чимало досліджень досі залишаються лише на папері як документація. Через це науковцям нерідко доводиться створювати власні моделі й симуляції з нуля, що потребує значної витрати часу на пошук інформації.

MAS [2] можна також розглядати як нову парадигму програмування, особливо корисну при розробці розподілених програмних систем. Такі системи складно або неможливо централізовано контролювати, і тут MAS стають у пригоді. Вони є цікавою альтернативою традиційному моделюванню, яке зазвичай базується на математичних рівняннях, – особливо тоді, коли потрібно враховувати взаємодії між численними компонентами. Агенти можуть уособлювати окремі елементи соціальної структури великих систем, і завдяки цьому MAS дають змогу досліджувати, як із локальних взаємодій виникає глобальна поведінка.

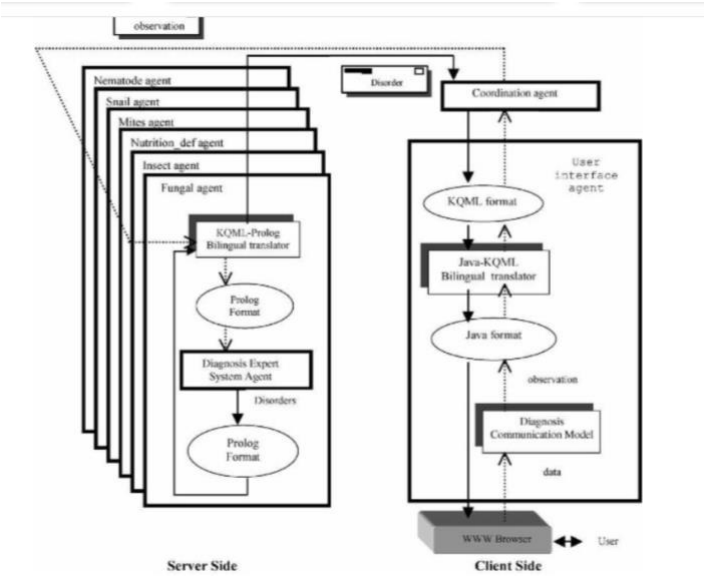

Рисунок 1.1 – Схема роботи декодера

У деяких випадках моделювання таких систем може передбачати участь мільйонів агентів, що потребує потужних ресурсів операційних систем. Особливого значення будуть обчислення на графічних процесорах (GPU), які дають необхідну продуктивність та масштабованість. Таким чином, виникає необхідність розробки нових підходів до моделювання MAS, які б дозволили ефективно описувати складні структури взаємодії між великою кількістю агентів, здатних змінювати свою поведінку у процесі взаємодії.

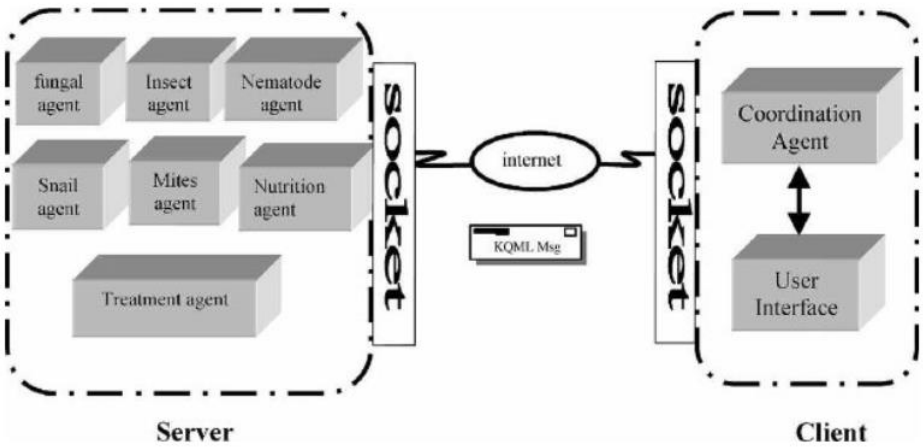

Рисунок 1.2 – Приклад спілкування агентів

Якщо об’єднати MAS+AI [3] то це відкриє інший рівень автономності систем. Агенти які будуть під’єднані до штучного інтелекту будуть мати можливість не тільки аналізувати навколишнє середовище а й приймати рішення на основі цих даних даючи змогу операторові займатись іншими справами, а не постійно спостерігати за станом системи.

Хоча MAS і не має великої підтримки та стрімкого розвитку, вона застосується в дуже різноманітних технологіях, починаючи від логістики де в компанії Amazon MAS застосовують для керування роботами у складських приміщеннях. А в іграх технологія агентів керує NPC (неігрові персонажі), що покращує реалістичність в іграх.

1.2 Обґрунтування вибору тематики вебзастосунку для керування дронами за допомогою ШІ

У процесі вибору тематики необхідно знайти приклад практичного застосування багатоагентної системи, який би був перспективним та сучасним, а також мав би цінність у майбутньому. Сфера керування дронами якнайкраще відповідає цим критеріям. Безпілотні літальні апарати вже стали частиною багатьох галузей – від сільського господарства до безпеки для рятувальних операцій.

Сьогодні особливої уваги потребує координація декількох дронів одночасно, що і є найкращим прикладом багатоагентної системи. Також включає задачі розподілу ролей між агентами. Крім того у розробці можна використовувати елементи штучного інтелекту які дозволять зробити систему більш адаптивною.

Чому саме вибір пав на створення вебзастосунку. Саме розробка вебзастосунку як засіб керування багатоагентною системою має кілька важливих переваг.

Вебзастосунки є кросплатформеними, тобто вони не залежать від операційної системи користувача що дає змогу керувати системою з будь-якого пристрою. Це спрощує керування агентами у вигляді дронів як у тестовому середовищі так і в реальних умовах.

Також вебтехнології забезпечують зручний інтерфейс для візуалізації стану агентів. Через браузер можна в реальному часі слідкувати за діями дронів, віддавати команди або отримувати зворотній зв’язок.

Сучасні вебтехнології дозволяють зробити з’єднання з серверною частиною через API або WebSocket що дозволяє легко інтегрувати систему з іншими сервісами для розширення функціоналу у майбутньому.

1.3 Потенціальна користь від реалізації проєкту

Вебзастосунок розроблений для керування дронами багатоагентної системи з штучним інтелектом, дає змогу підвищити ефективність управління декількома об’єктами, за допомогою MAS кожен дрон це автономна система, але яка має зв’язок з іншими системами для координації дій, а це дасть поштовх для розвитку пошукових операцій.

Оскільки у MAS буде інтегрований штучний інтелект, це дасть системам можливість адаптації для неочікуваних ситуацій, а це дасть змогу користувачеві краще опанувати цю систему, та не допускати помилок.

Якщо дивитися на цю систему з технічного боку то завдяки цьому проєкту можна буде протестувати новітні технології у сфері штучного інтелекту, протестувати його адаптацію, його прийняття рішень та з якою швидкістю від розвивається. Таким чином це добрий фундамент для подальшого вдосконалення цієї системи, та проводження досліджень на цю тему.

Але якщо мати корисні цілі на цей продукт то його можна застосовувати в комерції для автоматизованого моніторингу або охорони, чи навіть у доставці товару.

1.4 Проблеми інтеграції MAS із ШІ

Приєднання штучного інтелекту до багатоагентної системи [4], це чудова ідея, яка відкриває безліч можливостей у майбутньому розвитку такої технології, але як і все нові технології потребують багато тестів, обчислень та мають дуже багато обмежень, котрі потрібно враховувати при розробці майбутньої системи.

На сам перед великі мовні моделі не розроблялися як механізм керування багатоагентною системою в режимі реального часу, а координація між агентами може призводити до затримок у відгуках між ними або навіть неправильної роботи та взаємодії [5].

Всі моделі які є можливість встановити на персональний комп'ютер, мають обмежену кількість токенів які можна передати за один запит. Це особливо затримує роботу з великою кількістю агентів, особливо коли потрібно передавати їх історію взаємодії чи стан всіх агентів системи.

Для використання LLM в управлінні агентами охоплює запити до моделі, яка частіше всього знаходиться на віддаленому сервері або їй потрібні великі обчислювальні ресурси. Ця проблема створює додаткову затримку між LLM моделлю та агентом, яка може бути критичною коли рішення повинне бути прийнято миттєво.

Отже якщо система буде масштабною, де в дії буде велика кількість агентів то запит до LLM моделі може бути технічно складним. Чим більше запитів до моделі то тим більше навантаження на обчислювальні ресурси яких може бути недостатньо для аналізу даної задачі що призведе до критичної помилки або сильно затримує на час відповіді системи, а це на сам перед впливає на стабільність застосунку.

Інша не менш важлива ситуація коли при відправленні однакового запиту LLM модель може відповідати по різному, тобто при однакових вхідних умовах, відповідь буде відрізнятися. Це призведе до критичних помилок в задачах де потрібна стабільна поведінка системи.

Оскільки кожен агент це окрема система яка може приймати рішення на основі відповідей LLM моделі, то є ризик не правильної розуміння вхідних даних, що призведе до аномального рішення яке агент виконає та це може призвести критичних наслідків.

1.5 Аналіз існуючих рішень у сфері керування дронами

У нашому часі існує велика кількість рішень для керування дронами, як апаратно так і програмно. Такі системи активно використовуються у багатьох сферах: від військових операцій та до сільського господарства чи екологічного моніторингу або навіть логістиці і рятувальних місій.

Найбільш відомий виробник це компанія DJI, вона пропонує високо якісні дрони та зручне програмне забезпечення. Забезпечення DJI дають змогу ручного керування, а також дозволяють задавати маршрут польоту або отримувати дані з камер та сенсорів які вбудовані у дрон. Але їх операційне забезпечення дозволяє керувати лише одним дроном та потребує постійну участь оператора.

Також існує компанія Amazon Prime Air, вона займається доставкою малогабаритного вантажу за допомогою дронів [6]. Операційна система Amazon Air використовує вже автоматичне керування, але система побудована інакше та не використовує багатоагентний підхід, всі дрони керуються через єдиний сервер.

Ще є компанії з відкритими платформами такі як PX4 та ArduPilot, вони дають змогу автоматичного керування, а також мають функцію автономного польоту. Але поведінка дронів в цих системах діє згідно з написаним сценарієм та на мають динамічної взаємодії одного з одним.

Якщо розглядати ШІ для керування дронами, то зараз такі рішення тільки починають розвиватися. Деякі компанії поки що лише експериментують з штучним інтелектом для розпізнавання об’єктів або планування маршруту.

Отже аналіз існуючих систем демонструє, що хоч технології у цій сфері і досягли високого рівня, але все ще не вистачає систем які поєднують LLM та MAS. Саме тому таку тематику я вибрав для написання даної кваліфікаційної роботи.

1.6 Постановка задачі

Підсумовуючи, розробка вебзастосунку для керування дронами є актуальним рішенням. Тому є потреба в створенні сервісу на цю тему, який би облегшував користувачу керування агентами в реальному часі за допомогою LLM.

Об’єктом роботи є вебзастосунок для керування дронами через LLM модель та можливостями керування вручну.

Метою роботи є розробка інтелектуальної системи керування агентами у вигляді дронів у real-time системі. Де за допомогою чату з AI–моделлю здійснюється керування агентами для виповнення певних цілей.

Для вирішення проблем даної тематики роботи був проведений аналіз великих мовних моделей, та аналіз існуючих готових рішень на дану тематику для повного розуміння поставленої задачі та виявлення хороших та поганих рішень.

Для досягнення мети необхідно вирішити такі завдання:

– аналіз існуючих LLM для локального застосування;
– вибір технологічного стеку;
– проєктування схеми взаємодії компонентів;
– налаштування локальної LLM моделі;
– розробка промтів для точного парсингу команд;
– обробка помилок якщо ШІ повертає невалідний JSON;
– розробка алгоритму групового руху;
– розробка динамічної маршрутизації;

– розробка візуалізації дронів на карті;
– чат-інтерфейс для текстових команд;
– навантажувальні тести;
– оптимізація продуктивності для скорочення часу відгуку;
– розробка інструкції для локального запуску.

2 ОБРАННЯ ПРОГРАМНИХ РІШЕНЬ ДЛЯ ЛОКАЛЬНОГО ЗАСТОСУВАННЯ

2.1 Аналіз існуючих LLM для локального застосування

Для аналізу були взяті 5 LLM для локального застосування. Розберемо кожну модель окремо.

Модель Ministral 7B [7] – продемонструвала чудову продуктивність у різних тестах, перевершуючи навіть моделі у яких кількість параметрів більша. Вона перевершує в таких галузях, як математика, генерація коду та міркування.

Переваги:

– оптимізована для CPU/GPU (працює на MacBook Air M1);

– висока швидкість генерації;

– підтримка формату JSON для структурованих команд.

Недоліки:

– вимагае 6-8 ГБ оперативної пам’яті.

Llama3 [8] – Найбільші моделі Llama 3 були навчені на двох спеціально створених кластерах на 24,000 2 графічних процесорів з використанням комбінації методів розпаралелювання даних, розпаралелювання моделей та конвеєрного розпаралелювання. Удосконалений стек навчання Meta дозволяє автоматично виявляти, обробляти та обслуговувати помилки, максимально збільшуючи час безвідмовної роботи графічного процесора та підвищуючи ефективність навчання приблизно втричі.

Переваги:

– покращена якість генерації;

– краща підтримка багатозадачності;

– контекст до 8k/128k токенів.

Недоліки:

– 7B – версія вимагає великі потужності у GPU;

– 8B – версія гірша за якістю ніж Ministral 7B.

Gemma(2B/7B) [9] – розроблені для швидкої роботи безпосередньо на пристроях - від телефонів і ноутбуків до робочих станцій - допомагаючи розробникам створювати програми, де б вони не були потрібні людям. Gemma поставляється в різних розмірах, що дозволяє вам вибрати найкращу модель для ваших конкретних потреб в обладнанні та продуктивності.

Переваги:

– модель 2 дуже легка;

– оптимізована для кодування;

– інтеграція з TensorFlow.

Недоліки:

– менш ефективна для NPL – завдань;

– обмежена документація.

Phi-3 [10] – є найбільш ефективними та економічними моделями малої мови (SLM), доступними на ринку, що перевершують моделі того ж і наступного розміру в різних мовник, міркуючих, кодуючих та математичних тестах. Цей випуск розширює вибір високоякісних моделей для клієнтів, пропонуючи більш практичний вибір при складанні та створенні додатків.

Переваги:

– найкраща ефективність серед малих моделей;

– працює на 4ГБ оперативної пам’яті.

Недоліки:

– працює лише на 4k токенів.

Falcon [11] - це сімейство високопродуктивних великих мовних моделей, створених Інститутом технологічних інновацій (TII), дослідницьким центром, що входить до складу Ради з передових технологічних досліджень уряду Абу-Дабі, що займається технологічними дослідженнями.

Переваги:

– відкрита ліцензія;

– добра якість для англійської мови.

Недоліки:

– гірше розуміє інші мови;

– вимагає багато ресурсів.

Мій вибір LLM моделі пав на Ministral 7B, вона повністю відповідає потребам які вимагає мій проєкт. Усі інші LLM моделі в тому чи іншому сенсі не задовольняли усіх потреб, вони були чи сильно слабкіші, чи потребували доволі багато ресурсів операційної системи.

2.2 Вибір мов для Back-end

Вибір мови програмування для Back-end є однією із головних частин розробки вебзастосунка. Не правильним вибором можна зіпсувати весь проєкт та не отримати потрібну швидкодію та надійність яка повинна бути у застосунку для гарної його працездатності.

Якщо роздивлятись це більш глибоко то нам потрібні такі інструменти які забезпечать наш проєкт усім необхідним на кращому рівні в критеріях таких як функціональність, безпека та масштабованість.

Вибір серверної частини повинен забезпечити усі необхідні потрібності по функціональності. Вона повинна відповідати за такі функції як: обробку інформації, взаємодії з БД та бізнес логіку застосунка. Тобто потрібно обрати інструменти, які будуть задовольняти потреби та ефективно їх виконувати.

Отже правильний вибір програмних рішень впливає на всю систему. А саме на ефективність, продуктивність, швидкість відповіді та обробки даних. Також потрібно враховувати такий важливий аспект як масштабованість, який дозволить у стислі терміни зробити необхідні модифікації вебзастосунку.

Безпека є одним із найважливіших аспектів на сьогоднішній день. Забезпечити конфіденційність та цілісність даних важливо для будь якого

застосунку. Отже дуже важливо обрати ті інструменти які забезпечать надійність та засоби аутентифікації, шифрування даних або авторизації.

Масштабованість системи також є одним із головних ключем, який необхідно враховувати при виборі необхідної мови програмування для серверної частини. Якщо правильно та наполегливо підійти до цієї проблеми то можна знайти інструменти які будуть допомагати у розробці такі як контейнеризація яка спростить масштабування та розгортання системи.

Таким чином при виборі програмних рішень для серверної частини проєкту є відповідальним та важливим етапом який вплине на весь розвиток у майбутньому.

2.2.1 Мікросервісна архітектура

Це архітектура яка розбиває застосунок на незалежні мікросервіси, де кожен мікросервіс виступає як незалежна частина коду та виконує певні функції [12]. Таким чином це дає швидкодію та незалежність від інших компонентів що можуть сповільняти їх роботу. Тому вони можуть бути розгорнуті або оновлені незалежно від інших компонентів.

Чому мікросервісна архітектура так важлива для вебзастосунків та чому вона потрібна і в інших проєктах.

Вона дає змогу швидко розвивати проєкт та вводити нові функції. Це дає змогу працювати розробнику з окремими частинами застосунку, не звертати увагу на інші частини поки вони не знадобляться.

Отже другий аспект мікросервісної архітектури це надійність системи. Якщо один з аспектів застосунку перестав працювати, або знаходиться у відновленні то інші аспекти продовжать працювати у нормальному стані. Таке рішення дозволить підтримувати робото стійкість навіть коли один із сервісів має проблеми.

2.2.2 Python

Python – це потужна мова програмування, яка проста у вивченні [13]. Вона має простий синтаксис але структури даних високого рівня, та ефективний підхід до об'єктно-орієнтованого програмування. Простий синтаксис та динамічна типізація роблять його найкращою мовою для створення сценаріїв та швидкої розробки систем у багатьох сферах.

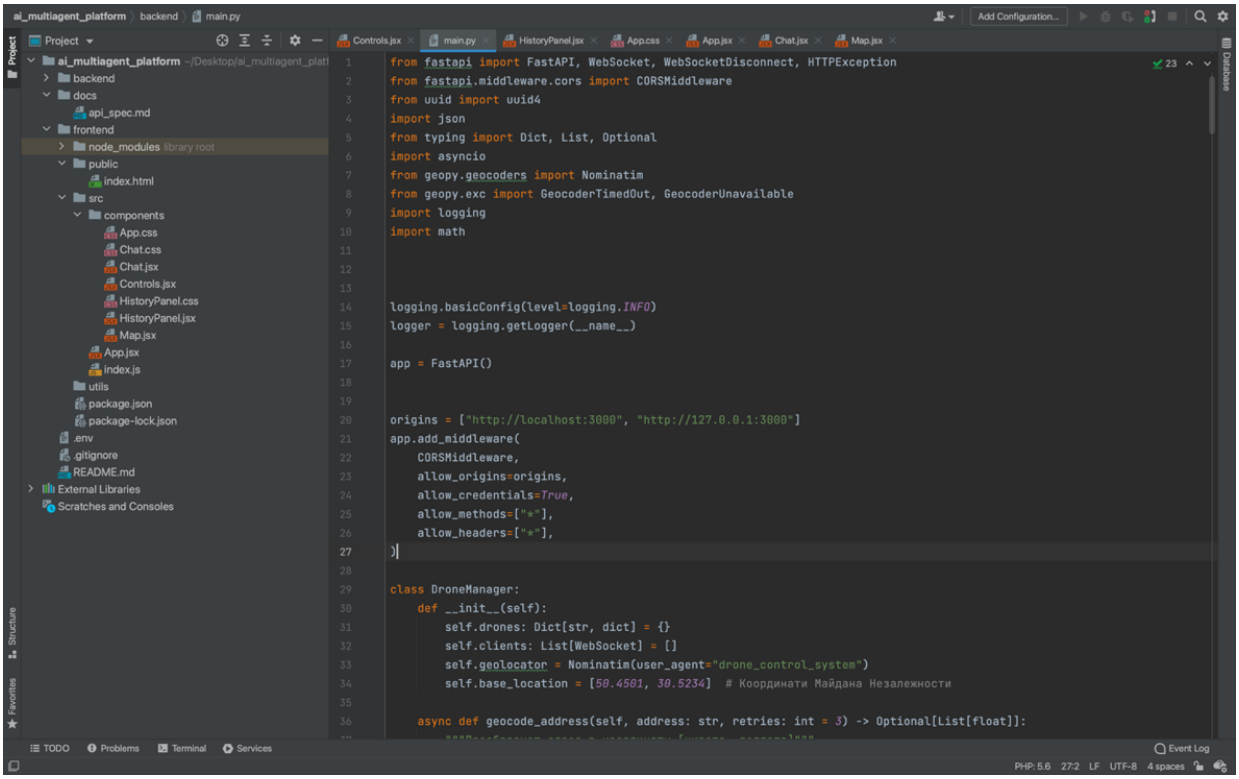

Рисунок 2.1 – Приклад Python коду

Python простий у виконанні, але це справжня мова програмування, яка пропонує набагато більше структури та підтримки для великих програм. Але з іншого боку потрібно перевіряти більше помилок ніж до прикладу мова C. Також Python дозволяє ділити програму на модулі, які можна повторно використовувати у проєкті.

Python це інтерпретована мова яка дозволяє економити час на розробку оскільки не потрібен час на компіляцію та зв'язування, це дозволяє легко

експериментувати з особливостями цієї мови програмування. Завдяки відсутності етапу компіляції цикл редагування, тестування та налагодження відбувається набагато швидше, ніж у багатьох інших мовах.

Налагоджувати програми на Python доволі зручно: помилки чи неправильні введення не призводять до критичних збоїв, таких як сегментаційні помилки. Якщо скрипт не обробляється програмою, Python автоматично виводить детальне трасування стека, що значно спрощує пошук і виправлення помилок.

2.2.3 Flask

Flask – це легкий мікро – фреймворк для мови Python розроблений спеціально для вебсервісів [14]. Flask дуже простий що робить його найкращим вибором для новачків, достатньо знати основні конструкції мови Python щоб почати їм користуватися. Також цей фремворк має багато інструментів для веброзробки. Його можна легко інтегрувати з іншими бібліотеками і не мати ніяких помилок по встановленню. Flask – це функції Python які обробляють та повертають відповіді. Як правило відповідь Flask нагадує HTML сторінку, но також за бажанням це може бути текст у Json форматі.

Ще одним із плюсів використання Flask є його підтримка майже всіх існуючих баз даних, а підключення робиться трьома строками коду.

2.2.4 Архітектура керування ройовим інтелектом

Існує три різні види архітектури систем керування [15]: централізована, розподілена та децентралізована.

У розподіленій архітектурі кожен агент має власний контролер, це дозволяє робити обчислення на місцевості від інформації на основі сусідніх агентів. Такий підхід дає високий рівень автономності та надійності, тому що

система не залежить від одного центру контролю. А якщо використовувати 5g мережу то це забезпечить ефективність реагування у динамічному середовищі.

В децентралізованій архітектурі функції розподіляються між незалежними підсистемами. Такий підхід збільшує гнучкість та спрощує масштабування, але має недолік у виді викликів координації між підсистемами.

Централізована архітектура працює як один контролер в якому знаходяться всі функції для керування. Такий підхід дає змогу детального аналізу, та дозволяє оперативно керувати поведінкою кожного дрона.

Таблиця 1 – Порівняння характеристик архітектур управління

Критерій	**Централізована архітектура**	**Розподілена архітектура**	**Децентралізована архітектура**
Затримка в передачі інформації	Низька, якщо вузли розташовані біля контролера	Залежить від місцевих з'єднань та швидкості оброблення	Залежить від з'єднань між підсистемами
Масштабованість	Обмежена через залежність від одного контрольного центру	Висока, агенти можуть додаватися без утручання центру	Висока, але може виникати складність у координації
Надійність	Залежить від єдиної точки збою (центральний контролер)	Висока, збій одного агента мало впливає на інших	Висока, але потребує ефективного взаємозв'язку між підсистемами
Стійкість до помилок	Низька, помилка в центрі впливає на всю систему	Висока, система може продовжувати роботу в разі збоїв окремих агентів	Висока, але вимагає додаткових механізмів синхронізації та відновлення

2.3 Вибір програмних рішень для Front-end

Клієнтська частина вебзастосунку також має дуже велике значення. Від цієї частини сайту залежить візуальній досвід користувача, який базується на відчутті, зручності та легкості освоювання. Також як і від серверної частини вебзастосунку, Front-end частина повинна бути ефективною, та масштабованою

для майбутнього розширення. Є декілька ключових особливостей які вплинуть на успіх проєкту.

Як і завжди, спочатку потрібно визначитися зі стеком технологій які ми будемо використовувати. Існує багато інструменті, фреймворків та бібліотек які допоможуть зробити цікавий та гарний інтерфейс, щоб користувачу подобалося ним користуватись. Фреймворки надають великі можливості для розробки такі як React, Vue або Angular допоможуть розробити багатофункціональний інтерфейс, а бібліотеки такі як Redax, MobX допоможуть в управлінні застосунком.

Щоб вибрати певні фреймворки та бібліотеки також потрібно враховувати потреби проєкту для реалізації певних функцій. Та оскільки нам потрібно динамічний візуальний інтерфейс то React [16] буде кращим вибором для цього із за його швидкості та простоті управління.

Також потрібно враховувати екосистему та спільноту навколо обраних рішень. Наявність великої спільноти розробників дуже допоможе у розробці цього застосунку, а саме забезпечивши доступ до шаблонів, плагінів та навіть документації [17].

Не менш важливо враховувати гнучкість та масштабованість проєкту. Потрібно вибирати технології які будуть підтримувати модульність та масштабованість що в майбутньому спростить розвиток проєкту.

2.3.1 JavaScript

Це найпопулярніша мова програмування для розробки вебзастосунків. Він використовується для створення денамічних сторінок оскільки має широкі можливості для реалізації таких рішень.

Вибір JavaScript є обумовленим та логічним оскільки він дуже поширений і актуальний, саме це дуже спрощує розробку вебзастосунку.

Також JavaScript це мова яка напряму підтримується браузерами, без встановлення додаткових інструментів, що буде мати особливу перевагу перед іншими мовами програмування.

Але якщо вибирати певну технологію то використовувати будемо JSX – це дуже потужний інструмент для створення вебзастосунку, який дозволяє використовувати мову розмітки HTML разом з JavaScript у одному файлі. А це зробить скрипт більш читабельним та легким в розумінні, особливо для розробників які в основному користуються мовами розмітки.

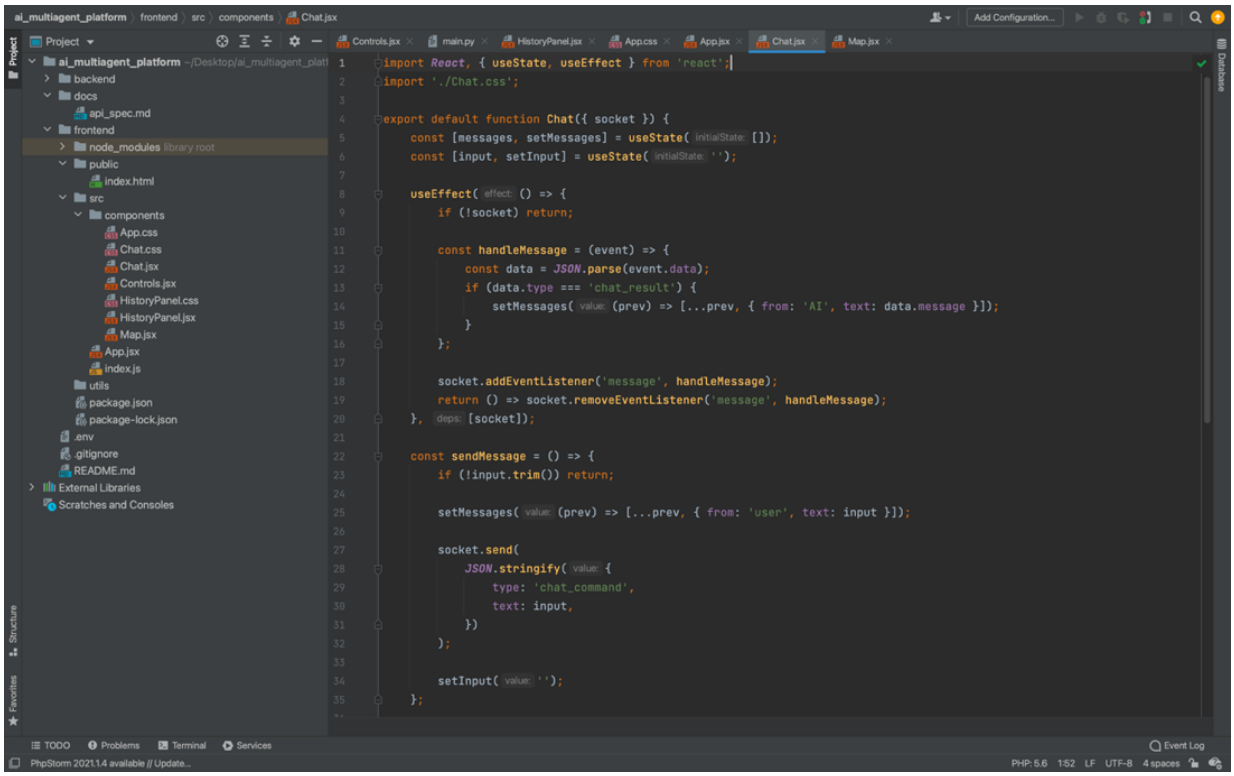

Рисунок 2.2 – Приклад JSX коду

JSX не тільки робить написаний код зручним, а й ще дозволяє використовувати сучасні інструменти розробки таких як рефакторінг коду. Характер технології JSX дозволяє створювати динамічні інтерфейси використовуючи менше зусиль ніж з іншими технологіями.

2.3.2 React

React – це одна з бібліотек JavaScript, яку використовують для створення користувацьких інтерфейсів. Вона здобула широку популярність серед програмістів завдяки своїй зручності у використанні. В розробці онлайн-симуляцій React виступає найкращим рішенням по деяким причинам.

А саме React базується на компонентній архітектурі, це дозволяє робити інтерфейс у вигляді окремих незалежних блоків. Такий підхід полегшує читаємість, створення структури коду і значно спрощує підтримку та масштабування проєкту.

Також, бібліотека дає змогу створювати динамічні інтерфейси, які реагують в реальному часі на зміни даних і взаємодію з користувачем – без необхідності перезавантаження сторінки. Це забезпечує високий рівень інтерактивності, що особливо важливо для симуляцій та візуалізацій.

Саме із за цих особливостей React є кращим вибором для створення сучасних вебзастосунків, які потребують продуктивності, масштабованості та зручного інтерфейсу. Тому варто детально розглянути можливості його застосування у межах даного проєкту.

В проєкті, який передбачає створення системи для управління багатоагентною симуляцією, React відкриває широкі можливості. Його архітектура, дозволяє реалізовувати динамічні візуалізації, зокрема відображення станів агентів у реальному часі, інтерактивне керування симуляцією, а також зручну навігацію між різними рівнями даних.

React легко зв’язується з серверною частиною через REST API або WebSocket, що дає змогу швидко обмінюватися даними між клієнтом та backend-сервером, де відбувається обробка логіки MAS. Це дозволяє взаємодіяти із симуляцією в реальному часі – змінювати параметри агентів, запускати чи зупиняти процеси, а також дивитися на поведінку системи через зручний інтерфейс.

2.3.3 Google Maps API

Google Maps API – це інтерактивна мапа світу яку розробила компанія Google для відображення її у браузері тому вона підтримує взаємодію з JavaScript, що забезпечує динамічну зміну контенту без перезавантаження сторінки. Це дозволяє оновляти координати, маршрути та статусу об'єктів у режимі реального часу без перезавантаження сторінки в браузері.

Також перевагою Google Maps є детальне покриття картографічними даними, включаючи як міську інфраструктуру, так і природні території які вони сканували власноруч за допомогою супутників або спеціальних машин з датчиками. Це розширює можливості симуляції та візуалізації маршрутів дронів у різних умовах.

Завдяки масштабованості та гнучкості Google Maps API добре інтегрується з іншими вебсервісами навіть на безкоштовній основі та дозволяє створити зручний інтерфейс для користувачів або використовувати стандартний що створила компанія Google, це значно покращує загальне враження від роботи з вебзастосунком.

2.4 Вибір бази даних для вебзастосунку

Вибір бази даних є також важливим етапом розробки вебзастосунка, від вибору залежить надійність, швидкість, та продуктивність цілого застосунка.

Насамперед потрібно враховувати який тип даних потрібно буде зберігати та передавати. Для вебзастосунку симуляції є важливим зберігати дані дронів, їх маршрути, та прийняті рішення LLM моделлю, а також данні користувача. Тому важливо вибрати базу даних яка зможе швидко опрацьовувати та надсилати або приймати таку інформацію.

Не мало важливим буде взяти на увагу масштабність проєкту. Наприклад якщо у застосунка буде багато активних користувачів то потрібно обрати БД яка

зможе забезпечити швидкий доступ до збережених даних. При виборі БД також потрібно звернути увагу на легке розширення застосунку та самої бази даних щоб мати можливість легко масштабувати весь проєкт, якщо користувачів різко стане більше.

Ну і на останок це безпека, безпека також є одним із важливіших аспектів розробки. БД яку ми візьмемо повинна забезпечувати високий рівень захисту та блокувати несанкціонований доступ для зловживання даними що будуть зберігатися у базі даних.

Серед популярних баз даних є MongoDB на основі NoSQL [18]. Вона відома своєю продуктивністю та гнучкість. У БД дані будуть зберігатися у форматі Json, який буде генерувати LLM модель, що дуже спростить та прискорить відправку та аналіз цих даних. Такий підхід дозволить зберігати складну та вкладену інформацію у закодованій формі, що спростить роботу та поліпшить безпеку цих даних.

MongoDB має одну з важливіших ознак і це її масштабованість, а оскільки на протязі всієї кваліфікаційної роботи ми згадуємо про це, то можна вважати що це один із важливіших елементів для вебзастосунка.

Бази даних підтримують масштабування у горизонтальному вигляді що дає змогу зберігати інформацію з декількох PC та розподіляти інформацію на декілька серверів. Це важливий аспект для великих вебзастосунків, яким потрібно обробляти великі обсяги даних не втрачаючи високу продуктивність.

Інша важлива ознака MongoDB це можливість додавання нових полей у базу без її повного оновлення, що робить процес розширення та адаптування більш легким та швидким. А саме, дозволяє зменшити час оновлення нових вимог до застосунку.

2.5 Вибір загальних рішень для вебзастосунку

Правильні підходи та принципи для написання коду є фундаментальними, це основна складова для успіху будь-якого проєкту.

В основі це забезпечити розширюваність коду, читабельність і підтримка всього проєкту. Якщо дотримуватися цих принципів то це гарантує зробити меншу кількість помилок та прискорить створення проєкту в цілому. Чим більше ми будемо опиратися на досвід інших поколінь розробників, та використовувати їх застереження або поради тим більше шанс успіху всього проєкту.

2.5.1 WebSocket

WebSocket – це протокол який забезпечує з’єднання між сервером та клієнтською частиною системи. Він забезпечує постійне з’єднання між ними в обидві сторони в реальному часі [19]. Також існує REST API, але в цій системі кожен запит робиться виключно клієнтом, а закінчується після отримання відповіді. WebSocket же пропонує встановити одне з’єднання через яке дві сторони будуть надсилати запити або данні в будь який момент.

В контексті кваліфікаційної роботи WebSocket служить для відправки даних між сервером та клієнтом. Після проходження авторизації система створює з’єднання в якому створюється токен, та цей токен буде в першому повідомленні WebSocket. Сервер отримує цей токен перевіряє його на дійсність і якщо все в порядку встановлює з’єднання та присвоює користувачеві унікальний ID. Далі весь обмін інформації буде відбуватися постійно без потреби авторизації що і зменшує затримки між відповідями.

2.5.2 Чистий код (Clean Code)

Це підхід при написанні коду який має на увазі щоб він був зрозумілий іншим розробникам, легким для тестування та підтримки [20]. Такий код повинен бути простим, а кожна функція повинна виконувати лише одну дію, а назви функцій повинні бути зрозумілими. Написання такого коду включає в себе видалення застарілого або непотрібного коду, також уникнення дублювання та організування в логічні модулі. Такий підхід полегшує роботу з оновленням та підтримкою існуючої системи. Якщо підтримувати підхід написання Чистого коду то можна покращити кінцевий продукт роблячи його більш охайним та стабільним.

2.5.3 Асинхронний метод

Асинхронність дає змогу виконувати декілька дій одночасно, не блокуючи інші потоки виконання. Такий метод важливий в проєктах де необхідно обробляти багато запитів від користувачів або працювати з БД чи при взаємодією з іншими сервісами. Це забезпечує продуктивність всієї системи та ефективніше використовує ресурси.

Асинхронність у Front-end зазвичай робиться через конструкції такі як flech чи axioc, або навіть за допомогою бібліотек призначених для запитів на сервер. Зазвичай такі операції працюють у фоновому режимі та не призводять до блоку інших операцій. Js в чистому виді не підтримує загальні прийняті принципи асинхронності, тому у цієї мови програмування є свій метод. Цей метод працює таким чином що програма буде чекати поки запит їй не повернеться, та тільки після цього вона буде виконувати наступні дії.

3 РОЗРОБКА ВЕБ ЗАСТОСУНКУ

3.1 Вирішення програмних рішень для Back-end

Для написання Back-end частини проєкту було використане програмне середовище PhpStorm з мовою програмування Python. Для початку треба зайти на сайт виробника текстового редактора, встановити його та можна починати налаштовувати редактор для роботи з мовою Python. Так як стандартна версія не має підтримки мови Python то необхідно встановити необхідні плагіни та розширення, щоб редактор адекватно реагував на цю мову програмування. Щоб це зробити, необхідно відкрити PhpStorm далі перейти у розділ File – Settings, у розділі Project додати Python Interpreter та натиснути кнопку ADD. Або відкрити термінал у застосунку та ввести команду «pip install pytest».

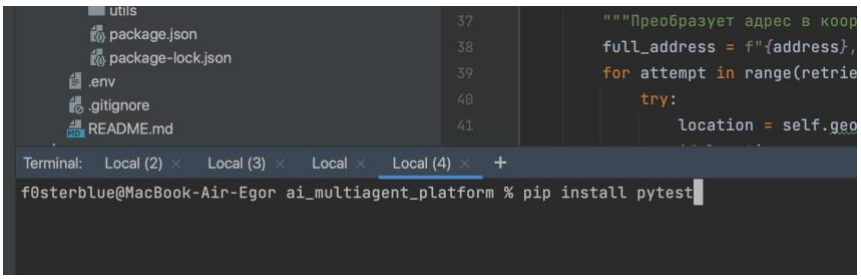

Рисунок 3.1 – Команда для встановлення Python у застосунок PhpStorm

Після встановлення всіх необхідних додатків, можна буде приступати до створення серверної частини проєкту.

3.1.1 Flask у розробці API

Для Flask додатку по перше було створено базову структуру. Об’єкти роблять ряд дій за допомогою flask(_name_). Використовується CORs механізм який забезпечує безпечний зв’язок між front-end та beck-end частинами. База данних підключається за допомогою драйвера pymongo.

Така реалізація проєкту має особливість режиму налагодження, яка дозволяє перезавантажувати сервер автоматично після зміни коду в ньому, та одразу отримувати повідомлення про помилки якщо вони є. Це дуже прискорює та спрощує розробку системи.

Щоб отримати дронів були реалізовані GET-endpoint, вони виконують запит до бд за допомогою методу find(), та одразу перетворює отримані данні у json формат, потім повертає відповідь що має масив даних та статус операції, а також повідомляє про можливі помилки, повертаючи відповідний код.

Для додавання або створення нового дрона створено окремий endpoint. Який отримує Json дані із запиту request.get_json(), потім робить валідацію даних по обов'язковим полям таких як dron_id або user_id, робить перевірку на унікальність ідентифікатора дрона, після чого зберігає дані у MongoDB і повертає інформацію про виконання операції якщо вона успішна і помилку у разі виникнення невідповідності вимогам.

Велику увагу приділено помилковим ситуаціям, для запобігання непередбачуваним помилкам всі операції були обгорнуті у блоки try-except.

А для виявлення помилок використовуються стандартні HTTP-коди такі як:

- 200(OK) – для успішних операцій;
- 400(Bad Request) – некоректні вхідні дані;
- 500(Internal Server Error) – для визначення серверних помилок.

Тому усі повідомлення містять деталізовану структуру для облегшення налагодження проєкту.

3.1.2 FastAPI

Це фреймворк на базі Python, який збудований поверх асинхронного сервера Starlette, та використовує Pydemic для валідації отриманих даних.

Головна перевага, це швидкість із за підтримки асинхронного програмування, що дає змогу обробляти велику кількість одночасних запитів.

У системі FastAPI виконує функції серверної частини, яка відповідає за всю логіку управління системою дронів. Через FastAPI інтерфейс користувача взаємодіє з модулями серверу, створення маршрутів, запуску дронів, також обробка команд користувача, та передача даних у реальному часі.

FastAPI є центральним вузлом, який координує зв’язок між клієнтом, штучним інтелектом, та API від Google Maps. Він приймає запит на обчислення маршруту, викликає ці обчислення з урахуванням небезпечних явищ, та надсилає готовий маршрут назад до клієнта або на front-end, також передає клієнту стан дронів і візуалізацію погодних явищ на карті. Тому завдяки асинхронному методу у FastAPI це дозволяє мати стабільну та інтерактивну систему, що так важливо в реальному часі при керуванні агентами у виді дронів.

3.2 Мікросервісна архітектура проєкту

Проєкт керування багатоагентною системою дронів створений на основі мікросервісів, це дозволяє розподіляти функціональність між компонентами які не залежать один від одного. Кожен окремий макросервіс виконує свою визначену роль у системі таким чином забезпечуючи модульність та легкість підтримки проєкту.

Мікросервіс для керування дронами є ядром усієї системи, він відповідає за всі операції які пов’язані з дронами. Обробляє запити на створення дрону, оновлює їх статус, отримує інформацію про стан та маршрут дрону.

LLM відігдає одну з ключових ролей у системі, вона робить інтелектуальне планування маршрутів, а також оптимізує маршрути. Модель отримує запит, який містить кількість дронів, початкову та кінцеву точку координат, а також координати погодних зон, які потрібно оминати.

Мікросервіси зв’язуються між собою через API-інтерфейси, це дозволяє масштабувати систему або легко додавати новий функціонал. Зв’язок між ними відбувається через http запити які передаються у виді json формату, таким чином забезпечуючи легкість інтеграції. Наприклад, мікросервіс керування дронами отримує запит на оптимізацію нового маршруту, то він передає його на мікросервіс відповідаючий за LLM модель, а потім після отримання результату оновлює дані у БД.

Архітектура проєкту розроблена так що кожен мікросервіс можна запустити на різних серверах, це підвищує стабільність системи. Кожен мікросервіс має розділену схему даних, а це мінімізує залежність між ними, тобто кожен мікросервіс може функціонувати самостійно, навіть якщо частина інших перестала працювати із за помилки. Це дуже важливо у системі керування дронів, де швидкість відповіді є критичним параметром.

Для комунікації між мікросервісами використовується токени доступу та шифрування даних. Кожен запит має в собі токен авторизації, який перевіряється перед виконанням запиту. Якщо у роботі якого небудь мікросервісу виникає помилка, то система одразу повертає повідомлення, це дозволяє швидко виявити та виправити проблему, не зупиняючи роботу усієї системи.

3.2.1 Створення БД

У процесі розробки було вирішено використовувати БД як основне сховище, тому що MongoDB є документально орієнтованою базою даних, це дає змогу працювати із складними структурами без потреби жорстко визначеній схемі. А це в свою чергу ідеально підходить для проєкту, у якому структура може змінюватися, або доповнюватися новими полями у процесі розвитку.

Кожен дрон представлений окремим документом, цей документ зберігає інформацію таку як, ID дрона, його поточне положення, його статус, маршрут із списку координат, та дані про час запуску. Це дозволяє швидко отримувати

повноцінну картину того щодо кожного дрона, навіть якщо система обслуговує десятки або й сотні агентів одночасно.

Дані у MongoDB можуть не лише забезпечувати надійність, а й реалізувати аналіз. До прикладу можна реалізувати відслідковування завершення маршрутів, або робити аналіз часу доставки. Також структура БД підтримує збереження погодних зон, та містить у собі координати центра, радіусу та погодне явище, це дозволяє системі швидко оновлювати мапу, та на основі даних змінювати маршрут.

MongoDB передбачає такі операції як, створення, оновлення або видалення даних. Як кращий приклад, при створенні нового маршруту, буде виконуватися завдання, а у базі з’явиться відповідний запис, та якщо в процесі польоту зміниться маршрут, то відповідне поле оновлюється, це дозволяє системі передавати актуальну інформацію в реальному часі.

Наявність такої бази дозволяє масштабувати систему оскільки всі її вузли звертаються до одного джерела. Також MongoDB дає реалізовувати багатокористувацький режим, де користувач взаємодіє лише з тими дронами до яких у нього є доступ, а це дає змогу в подальшому реалізовувати авторизацію на основі ролей.

Далі потрібно створити БД. У MongoDB дані організовуються у бази даних та містять у собі документи. Через MongoDB Atlas створюється БД для Вебсимуляції дронів, для прикладу візьмемо базу мікросервісів де в ній буде декілька основних колекцій наприклад Drones. Також необхідно визначитися з структурою для кожної колекції.

Отже колекція users буде мати такі поля:

– Id – унікальний код що присвоєний користувачу;
– Email - пошта користувача;
– Password – пароль.

А колекція drones має:

– Id - унікальний код що присвоєний користувачу;

– drone_id – унікальний код присвоєний дрону;
– start – стартові координати;
– end – кінцеві координати;
– addressStart – назва вулиці старту;
– addressEnd – назва вулиці кінцевої точки;
– created_at – час створення;
– status – статус операції.

3.2.2 Сервіс для авторизації користувача

Мікросервіс для авторизації має дуже важливе значення у вебзостосунку, це дає змогу для розробки функціоналу для реєстрації користувача. Це невід'ємна частина будь-якого вебсервісу, вона забезпечує безпеку даних та доступ користувачів до системи. Сервіс для авторизації, потрібен зокрема для зберігання даних, перевірки даних, управління сесіями та захисту даних.

Такі мікросервіси потребують використовування безпечних методів, наприклад хешування даних, за допомогою алгоритмів. Такий спосіб дає гарантію того що навіть якщо зловмисники зможуть отримати доступ до БД то паролі будуть захешовані, що в свою чергу не дасть їм змогу легко отримати паролі користувачів.

Коли користувач реєструється то він вводить свої дані, які проходять етап валідації та зберігаються у БД. При створені логіну та паролю користувачеві додається унікальний токен, який дає змогу відстежувати сесію та зберігати авторизацію, таким чином забезпечуючи зручність та безпеку.

При процесі авторизації, коли користувач ввів свої данні то вони звіряються з базою на коректність та наявність. Після успішної перевірки користувачеві надається токен на час сесії, який використовується для автендифікації подальших запитів. Таким чином користувачу не потрібно

проходити авторизацію для кожної дії на сайті, це підвищує зручність вебзастосунку.

Реєстрація це процес у мікросервісі, який створює нові облікові записи користувачів, даючи змогу новим користувачам мати доступ до вебзастосунку. Після вводу даних, користувач надсилає їх на етап валідації де вони переходять перевірку на повноту та правильність, дані можуть бути такі як логін і пароль, але також це може бути email або інші необхідні дані. Після чого якщо валідація пройдена успішно, то пароль користувача хешується за допомогою спеціального алгоритму, щоб запобігти витоку інформації. Далі дані надсилаються у БД де і створюється новий обліковий запис. Але для підвищення безпеки користувачеві на пошту відправляється підтвердження, що забезпечує впевненість у правильності введених даних. Такий спосіб не лише підвищує безпеку, а й робить персоналізований підхід для кожного користувача що буде зацікавлений у цьому вебзастосунку.

3.2.3 Створення безпеки

Для бази даних безпека є особливо важливою, насамперед для вебзастосунків в яких часто фігурує інформація користувача. MongoDB дає кілька рівнів безпеки що гарантує конфіденційність та цілісність даних.

MongoDB має такі механізми безпеки:

– авторизація;
– аутифікація;
– моніторинг;
– аудит;
– шифрування.

Спочатку аутифікація, вона відбувається за допомогою пароля. Авторизація це вже наступний пункт, де дає визначення які функції може використовувати аутенфікований користувач. Role-Based Access Control – це

модель у MongoDB, де кожен користувач має певну роль, що і показують його права доступу. База даних сама надає шифрування даних які вона містить, тому це дуже спрощує роботу з нею. Усі дані шифруються завдяки технології WiredTiger, що дає змогу використовувати власні ключі для шифрування даних. При передачі застосовується TSL/SSL що забезпечує захищений канал спілкування між клієнтом та сервером. В сховище також є механізм аудиту, ця технологія дає можливість слідкувати які операції робив користувач з базою даних. А це в свою чергу дозволяє відстежувати підозрілі дії користувачів у базі. Цей механізм має аудит-лог в якому зберігається успішні та невдалі спроби аутендицікації, тобто записує невдалі спроби входу в обліковий запис, або CRUD функції чи інші дії.

3.2.4 CRUD сервіс

CRUD – це мікросервіс без якого не обходиться жоден вебзастосунок. Він дозволяє розробнику керувати сайтом, вмістом сторінок чи даними. Цей сервіс має функціонал як для розробника так і для користувача. Розробник має доступ до керування будь-яким вмістом сайту, коли користувач має лише обмежений функціонал доступу до застосунку, який передбачає вже сам розробник. Адміністраторам також функціонал CRUD дає контроль над всім вебзастосунком.

Але користувач вже має обмеження, та не має доступ до всього функціоналу, тому користувач може лише зареєструватися та робити певні операції з дронами чи картою.

3.2.5 Ollama

Ollama це інструмент для локального запуску LLM моделей з відкритим кодом, вона дає змогу взаємодіяти та керувати великими мовними моделями

локально на персональному комп’ютері. Це дозволяє використовувати ШІ моделі локально, тобто усі обчислювальні процеси будуть здійснюватися на персональному комп’ютері, без потреби підключення до хмарних сервісів, даючи змогу взаємодіяти з LLM моделями без підключення до мережі інтернету.

Ollama дуже легко встановити на персональний комп’ютер тому що вона використовує контейнеризацію на шталт Docker. Також вона дуже гнучка бо підтримує безліч відкритих LLM моделей, іншою її перевагаю є конфіденційність. Але є й недоліки такі як вимоги до великих обчислювальних процесів, а також багато моделей слабкіші за відомий GPT-4.

Після завантаження Ollama ми можемо встановити модель Ministral 7b яка підходить нам по параметрам персонального комп’ютера. Таким чином в нас з’являється локальна LLM модель, з якою ми можемо проводити необхідні операції.

Для звернення до LLM моделі ми можемо використовувати промпт. Це інструкція яка вказує моделі які відповіді нам потрібні якщо в нас специфічне завдання. У випадку кваліфікаційної роботи промпт це запит для побудови маршруту дрона. В свою чергу це дає можливість обмежити відповідь LLM моделі даючи змогу отримувати лише ті данні які нам потрібні, а це також зменшить навантаження на систему, та скоротить час відповіді LLM моделі.

```
def optimize_waypoints_with_context(origin, destination, avoid_zones, raw_waypoints):
    prompt = f"""
    Ти навігаційний агент для дронів.

    Побудуй оптимальний маршрут від точки {origin} до {destination}, уникаючи погодні зони:
    {avoid_zones}.

    Поточний маршрут (Google): {raw_waypoints}

    Поверни лише список координат у форматі:
    [{{"lat": ..., "lng": ...}}, ...]

    ЖОДНА точка маршруту не повинна потрапляти всередину жодної з зон avoid_zones:
    [{ "lat": ..., "lng": ..., "radius": ... }].

    Облітай зони, дотримуючись оптимальності маршруту, але з абсолютним униканням.

    """
```

Рисунок 3.1.1 – Приклад промпту для оптимізації маршруту дрона

Важливим елементом системи стала інтеграція з великою мовною моделлю, вона реалізована локальним запуском Ministral через платформу Ollama. Мета цієї інтеграції полягає перекинути частину логіки прийняття рішень на штучний інтелект, що дозволить реалізувати гнучкі маршрути в умовах змінного середовища. В цій системі було досягнуто того, що велика мовна модель не просто реагує на текстові команди користувача, а й безпосередньо бере участь в аналізі та побудови маршрутів, враховуючи погодні явища, та пропонуючи оптимальні маршрути.

Комунікація з LLM працює шляхом формування текстового запиту – промпту, який містить необхідну контекстну інформацію для виконання запиту. Модель отримує ці данні як завдання, та повертає список нових координат які будуть оптимізованим маршрутом. Завдяки промпту у відповіді повертається лише список координат без зайвого тексту, що дозволяє інтерпретувати результат програмно.

Велику увагу було привернуто то обробки відповідей LLM. Оскільки модель може повертати дані у довільному форматі, або з зайвими символами. Тому було реалізовано попередню очистку відповіді, вилучаючи зайві символи, коментарі або маркери. Подалі відповідь парситься як Json-об’єкт, з якого вилучають координати. Якщо модель повернула некоректні дані то вона не підлягає декодуванню і система переходить в резервний режим застосовуючи маршрут який спочатку запропонував Google Maps API. Це дозволяє підтримувати стабільність системи та уникати помилок у роботі дронів через помилки моделі.

Окрім оптимізації та побудови маршрутів, велика мовна модель використовується для обробки текстових команд від користувача, введених через чат у інтерфейсі. До прикладу команда «Відправ дрон до київського зоопарку» передається як структура керування, та містить тип дії, ID дрону, назву об’єкта, та список зон які необхідно уникати. Так ШІ працює посередником між клієнтом та сервером що перетворює природну мову

користувача у формалізовані команди для системи. Це дуже спрощує керування системою, підвищує швидкість навчання та зручність, дозволяючи керувати системою інтуїтивно.

Важливим та найкритичнішим етапом було створення комунікації між користувачем та LLM моделлю. Потрібно було створити промти, які б давали гарантію, що буде правильний синтаксичний результат у форматі Json, а це є критично важливим для обробки команд.

Велику увагу було приділено структурі промтів, вони повинні формуватись так, щоб модель розуміла необхідність строгої генерації. Саме це допомогло суттєво знизити кількість синтаксичних помило, хоча проблема невалідного json і не була усунена повністю, для цього було реалізовано механізм перевірки коректності отриманих даних, а це допомагає виявити помилку на ранніх етапах і забезпечити повну генерацію команди.

Особлива роль у системі це підтримка динамічної маршрутизації, система повинна адаптуватися до змін в оточені, наприклад поява нових погодних зон чи зміну координат. Для виконання цієї задачі був розроблений механізм повторного запиту до LLM моделі з оновленими параметрами. Модель повинна генерувати новий маршрут, який негайно інтегрується в систему і передається відповідному дрону. Такий підхід дає змогу оперативно реагувати на зовнішні зміни, але із за обмеження у обчислювальних можливостей персонального комп’ютера таку поведінку не вдалося реалізувати у повному обсязі.

3.2.6 WebSocket

Дуже важливу роль також відіграє механізм постійного зв’язку між клієнтською частиною та серверною, цей механізм реалізований за допомогою WebSocket технології. У класичному HTTP запиті працює лише односторонній канал зв’язку, на відміну від WebSocket який забезпечує двосторонній режим передачі інформації у реальному часі, а це дуже важлива річь якщо говорити про

динамічне керування дроном, які перебувають у русі та реагують на навколишнє середовище.

Як тільки з’являється підключення клієнтської частини до серверної то одразу вмикається постійне підключення між ними яке завершить свою роботу тільки після виходу з застосунку. Саме таке підключення дає змогу відмалювати рух дрона на мапі в реальному часі. Це дає нам змогу не тільки слідкувати за поточним місцем знаходження дрона, а й миттєво реагувати на появу небезпек або різку зміну погодних умов. З цього можна зробити висновок що WebSocket є каналом синхронізації між системою серверу з візуальним представленням клієнтської частини на front-end.

Окрім передачі координат цією технологією можна користуватися для надсилання оновлених маршрутів, надсилання змін погодних явищ чи сигналів про завершення поставлених завдань, а також надсилання повідомлень надісланих користувачем через чат з ШІ. Ця технологія зменшує навантаження на сервер, оскільки не має потреби постійного звернення до API, дані будуть надсилатися лише за потреби, при виконанні певних подій.

Особливість даної реалізації полягаю у тому що серверна частина написана на FastAPI та підтримує асинхронну обробку запитів, а це в свою чергу дозволяє одночасно працювати з великою кількістю юзерів без втрати продуктивності. Таким чином кожне підключення обробляється окремо, тому відправлення та оновлення відбувається одночасно всім клієнтам, наприклад якщо одна погодна зона поганої погоди стосується декілька дронів різних юзерів.

Саме тому така архітектура системи створює не лише стабільну і масштабовану систему, а й передумови для розвитку у майбутньому, наприклад, додавання голосових сповіщень, реакцій на аварійні ситуації, або підказки у реальному часі від ШІ. WebSocket став ключовою технологією у цьому проєкті забезпечуючи безперервне управління автономними агентами.

3.2.7 Google Maps API

Побудова маршрутів базується на системі сервісів, де є додаткова логіка яка враховує зовнішні фактори, такі як погодні явища. На першому етапі побудови система отримує початкову та кінцеву точку маршруту. Ці данні надсилаються до Google Maps API [21], а повертається розрахований маршрут на основі вуличної інфраструктури. Цей маршрут завжди є оптимальним за відстанню та часу польоту, але він не розраховує специфіку польоту, яка потребує врахування уникнення певних зон, які можуть становити загрозу.

Саме тому був доданий модуль обробки небезпечних та погодних зон, який отримує дані, якщо на маршруті є перешкоди. Якщо на маршруті виявляється перешкода то, ШІ перебудовує маршрут для уникнення цих зон. У стандартному варіанті перебудова маршруту реалізована як додавання додаткових точок координат, які обведуть перешкоду по дузі певного радіусу перешкоди. Такий підхід дає змогу швидко уникнути зони, але має недолік такий як ефективність відстані.

Для уникнення цього недоліку, була створена логіка на основі ШІ, яка повністю перебудовує початковий маршрут з нуля. Забрані координати початкової та кінцевої точки, а також координати погодних або небезпечних зон передаються до великої мовної моделі, вона вже в свою чергу генерує повністю новий послідовний маршрут, який враховує загальну довжину, складність траєкторії, та навіть зігзаги, які можуть утворитися при геометричному обході.

Тому модуль маршрутизації у проєкті має вигляд багаторівневої системи, де спочатку маршрут відбудовується за допомогою Google Maps API, далі йде перевірка на перетин небезпечних зон, та наприкінці додавання додаткових точок, чи створення повністю нового маршруту за допомогою штучного інтелекту. Такий спосіб дає змогу мати баланс між швидкістю побудови шляху та його гнучкістю.

3.2.8 Алгоритм групового руху

Ще одним важливим етапом було створення алгоритму для групового руху дронів. У багатоагентних сценаріїв взаємодії це є надзвичайно важливим, для того щоб дрони не діяли автономно не враховуючи один одного, а мали спільну логіку поведінки. Такий алгоритм вимагав впровадження логіки, яка забезпечує утримання формації, та дотримання певної дистанції між агентами, а також реакцію на зміну маршруту головного дрону [24]. Алгоритм був реалізований на принципі поведінки кожного агента залежно від стану його сусіда, а координування відбувалося завдяки рішень LLM або за допомогою головного дрона який виступав координатором.

У такому підході роль лідера призначається для окремого агента. Тобто тільки лідеру зазначається маршрут місії, а іншим агентам потрібно утримуватися в бажаній формації рухаючись за ним (рис 3.1.2). Є два типу стратегії «лідер – послідовник»: Де послідовники підтримують формацію за лідером, або кожен агент тримається за наступним агентом тримаючись формації.

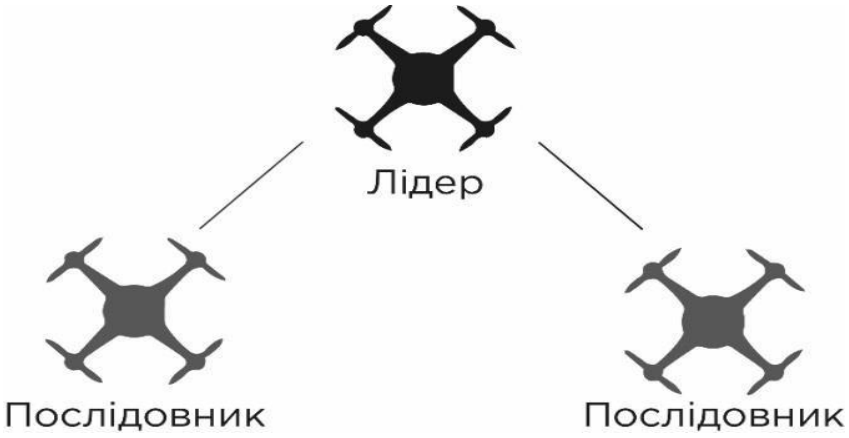

Рисунок 3.1.2 – Приклад системи «лідер – послідовник»

Такий підхід є найпоширенішим, де лідер тримається заданого маршруту, а інші утримують задану відстань та орієнтацію щодо дрона лідера. [24].

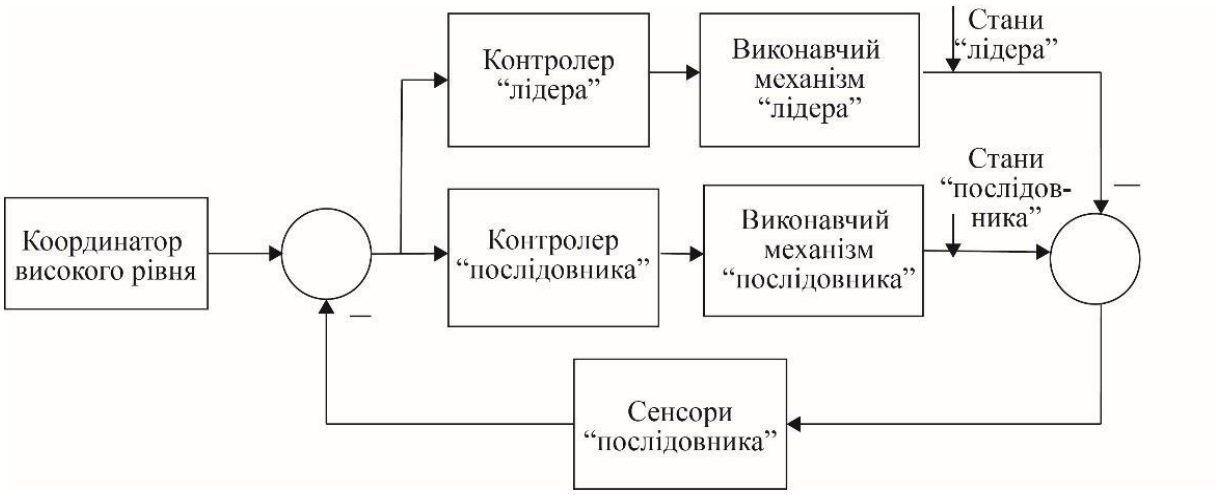

Рисунок 3.1.3 – Схема управління «лідер – послідовник»

3.2.9 Особливості моделі Ministral 7b

При розробці тестових програм для керування багатоагентною системою було обрано модель Ministral 7B [1], як основу ядра кожного агента. Вибір цієї моделі був із за її компактності та продуктивності, а також з урахуванням обчислювальних можливостей персонального комп'ютера. Ministral 7B – це велика мовна модель, розроблена на архітектурі трансформерного типу, вона здатна генерувати відповіді у режимі автокомпілювання. Вона має відносно невеликий розмір – 7 мільярдів параметрів – але демонструє кращу продуктивність ніж моделі більші від неї.

Для реалізації багатоагентної системи ця модель використовується локально завдяки платформі Ollama, таким чином це дозволяє інтегрувати її у кожний агент без потреби звернення до хмари. Такий підхід дає змогу отримати високу швидкість взаємодії між агентами. Так кожен агент отримує можливість самостійно аналізувати ситуацію та повідомлення від інших агентів.

Ministral 7B технологічно функціонує наступним чином. Вхідне повідомлення перетворюється у токени – це числові представлення тексту. Потім ці токени передаються у 32 шари трасформера кожен з якого складається з механізмів глибинної нейронної обробки. Внутрішньо модель зважує контекст, шукає залежності та на цій основі формує відповідь, у вигляді певної послідовності токенів, яка вже в свою чергу перетворюється на текст.

За допомогою інструкційного навчання модель здатна розвиватися та адаптуватися до чітко визначених завдань. Такий підхід дозволяє реалізувати оптимальну поведінку агентів.

Тому модель Ministral 7B стала ключовим компонентом, надаючи кожному агенту можливості генерації та аналізу у рамках децентралізованого керування.

3.3 Front-end рішення

В першу чергу це розробка візуального контенту для користувача. Тобто спочатку потрібно визначитись з головною сторінкою сайту, її кольорами, вмістом, шрифтом та іншим, це все повинно привертати увагу користувача, повинно бути простим і зручним у користуванні, тому це відіграє ключову роль у популярності застосунку та відповідає за кількість користувачів на сайті. При переході з посилання на головну сторінку сайту, користувача одразу зустрічає головна сторінка з авторизацією користувача (рис. 3.2).

Головна сторінка сайту з авторизацією була виконана у темних тонах з вибором тенденції темних тем у всіх застосунках в нинішні часи. Авторизація обов’язкова для роботи з агентами через LLM модель тому це і було винесено на перший план вебзастосунку. А також показує користувачу що вебзастосунок має безпеку і що користувач може не хвилюватися що з його даними щось скоїться.

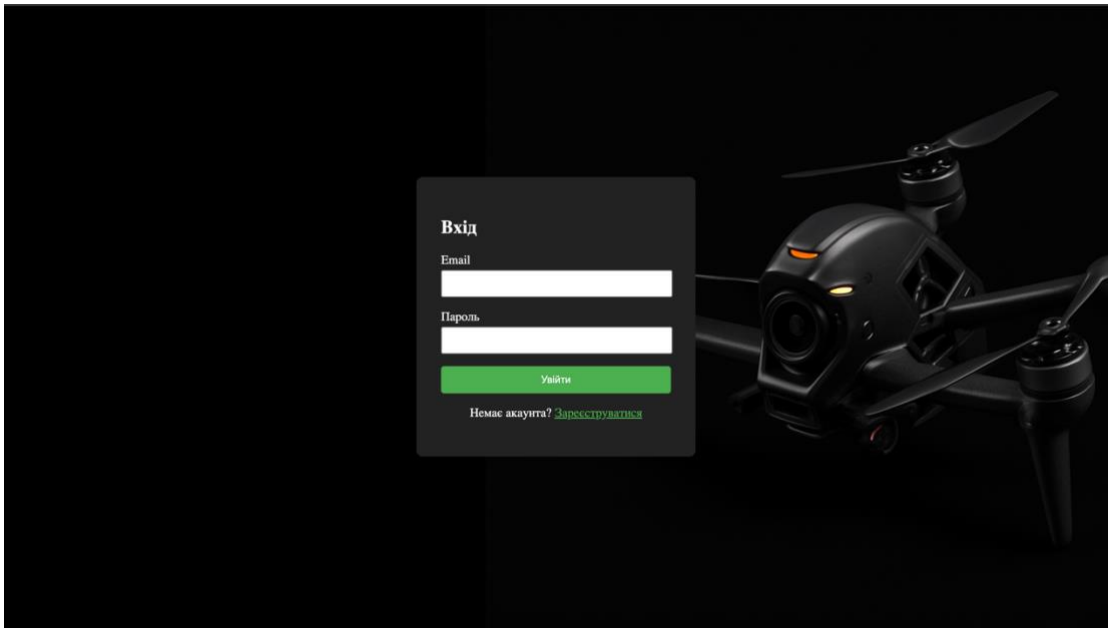

Рисунок 3.2 – Головна сторінка сайту

Оскільки користувач перший раз заходить на посилання цього вебзастосунку то йому одразу необідно зареєструватися, тому зеленим кольором виділена кнопка реєстрації (рис. 3.3), яка переведе користувача на форму реєстрації. Де є два поля Email та Password, цих двох значень достатньо для реєстрації щоб забезпечити безпеку даних користувача.

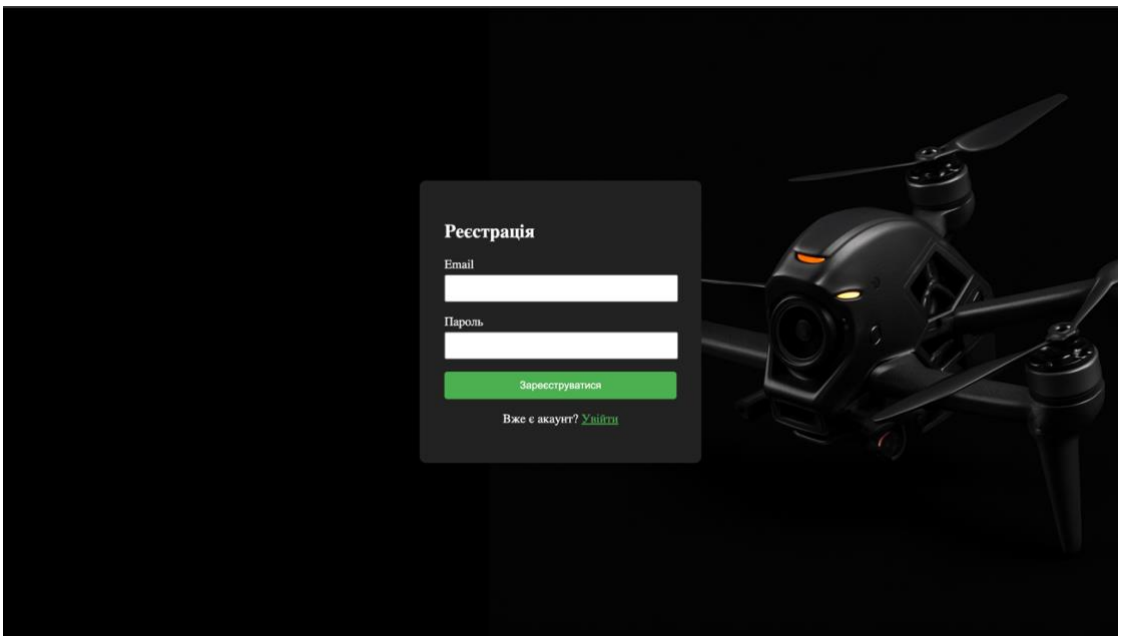

Рисунок 3.3 – Форма реєстрації у вебзастосунку

Після створення акаунту користувачу відкриється весь доступний функціонал застосунку для симуляції керування дронами (рис. 3.4). Одразу користувача зустріне на весь екран мапа Києва з вибором простої мапи або мапи з супутникових знімків (рис. 3.5).

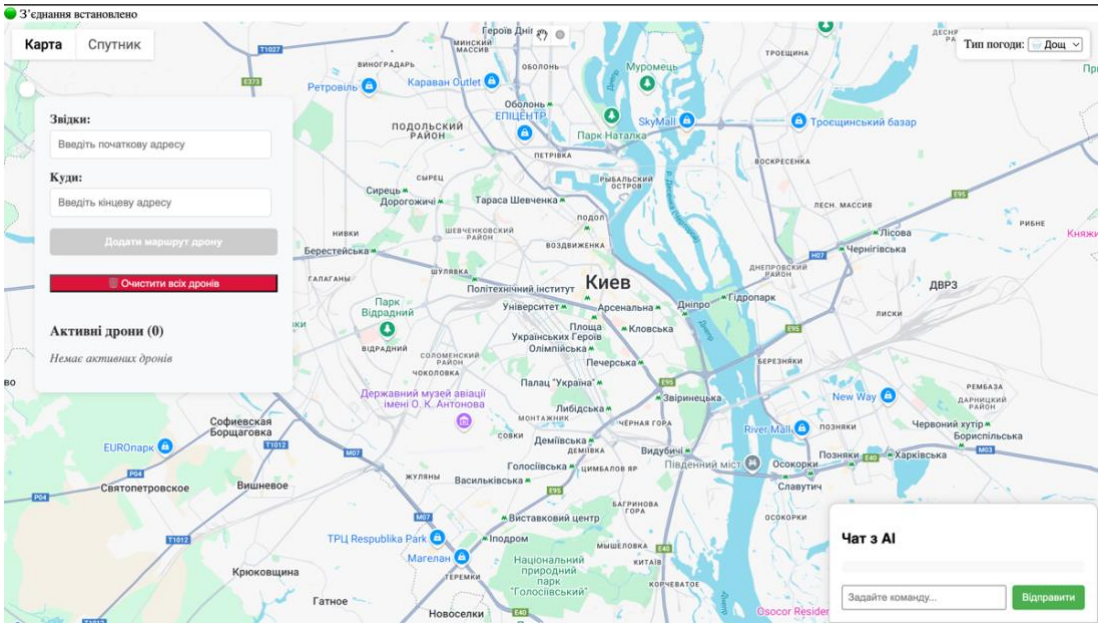

Рисунок 3.4 – Приклад функціоналу доступного користувачеві

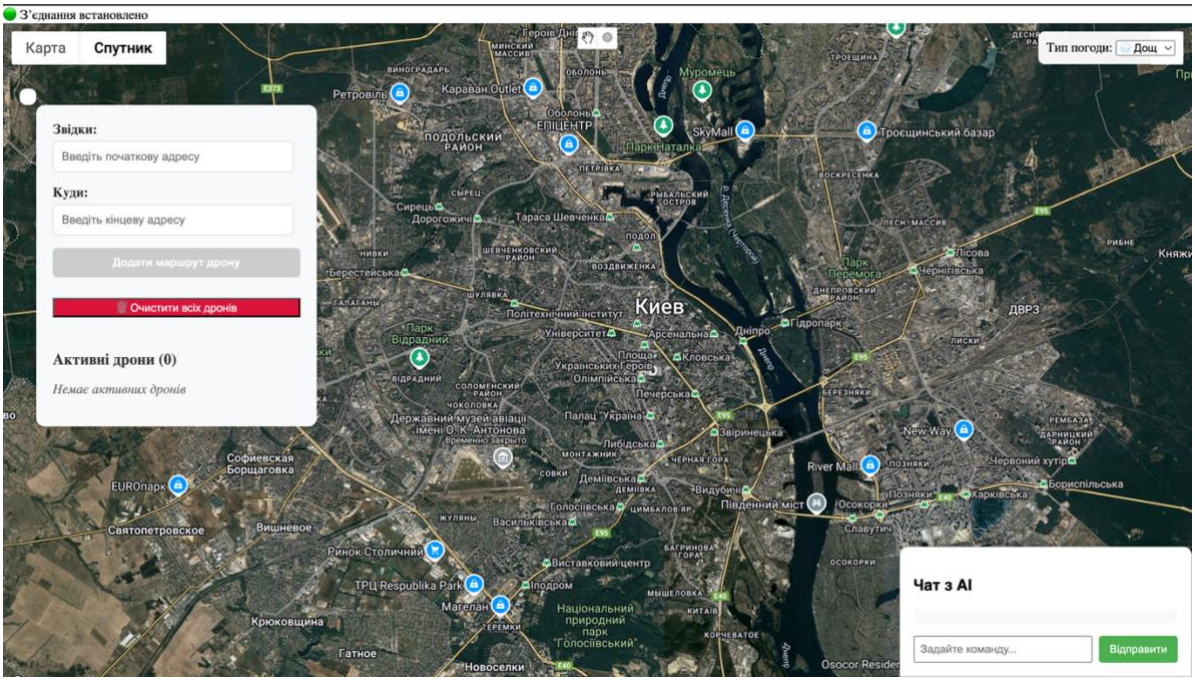

Рисунок 3.5 – Мапа зі супутникових знімків

Користувач побачить що додавання дронів можливе декількома способами такими як введення початкової та кінцевої адреси, або через спілкування з штучним інтелектом. Також з верху екрана користувач може побачити чи є з’єднання з сервером чи сервер на технічній перерві. Одразу після цього користувачеві доступне ручне додавання погодних умов (рис. 3.6) для взаємодії з симуляцією погодних явищ на мапі. Також є варіанти вибору погодних явищ таких як дощ, град або сильний вітер. Карта добавлена за допомогою API ключа від GOOGLE MAPS API, тому це точна карта місцевості всієї планети, що дозволяє відправляти дрон у будь-яку точку планети, не знаючи координат, достатньо ввести назву вулиці, міста, чи якоїсь певної точки, та маршрут відразу буде прорахований технологіями Google. Для розширення функціоналу мапи, можна придбати платну підписку від Google, це дуже сильно розширить функціонал та можливості проєкту, але на даний час функціоналу пробного періоду від Google достатньо.

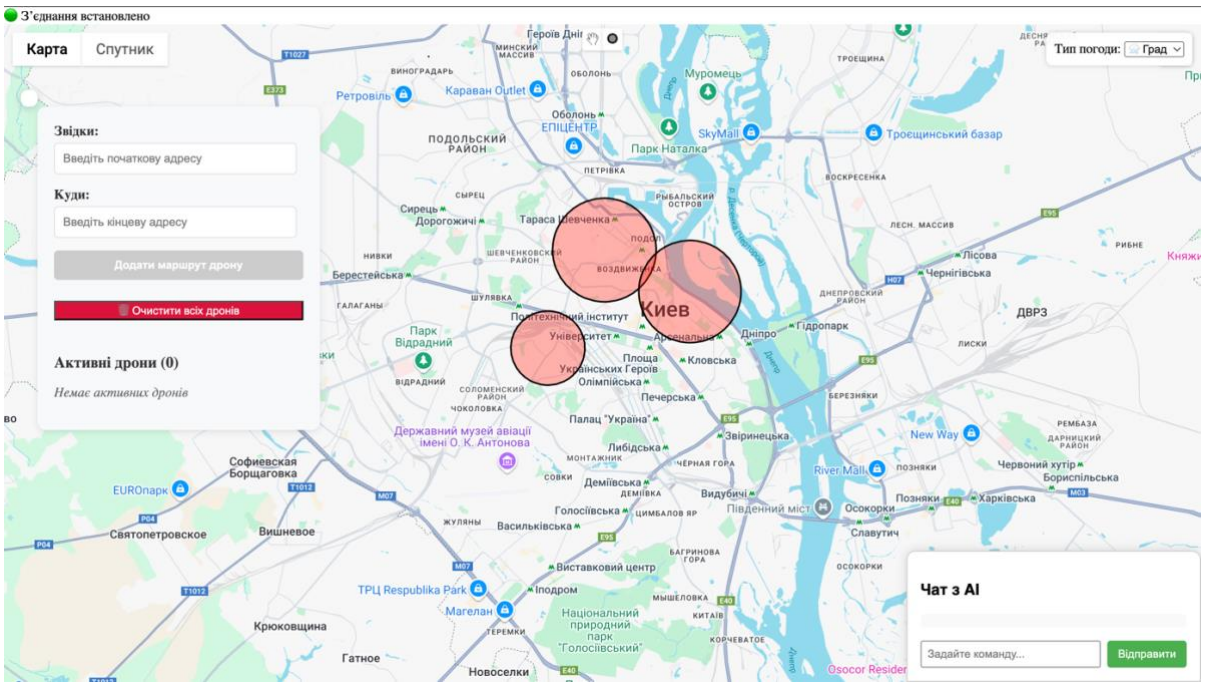

Рисунок 3.6 – Приклад зображення погодних зон на мапі

Користувач на даний час має два вибору створення симуляції з дроном, перший варіант передбачає введення адрес з підказками від Google, після чого для дрона створиться маршрут через Google Maps з урахуванням вулиць, але у цьому випадку штучний інтелект не буде приймати участь. З лівої сторони сайту є два поля з початковою точкою та кінцевою. Через ці дві адреси створиться маршрут та відобразиться візуально на карті, також на сервері буде створено дрон, який буде занесено до бази даних та приписано до користувача який його запустив, тобто в базі даних, буде зберігатися індикаційний код дрона, його стартова точка та кінцева точка маршруту. Це зроблено для того щоб розробник міг додавати новий функціонал на основі запитів користувачів.

Після введення адрес, на боковій панелі застосунку з’являться дрони (рис. 3.7) де користувачеві буде видно індикаційний код дрону, його статус, початкову та кінцеву адресу, та кнопки видалення дрону.

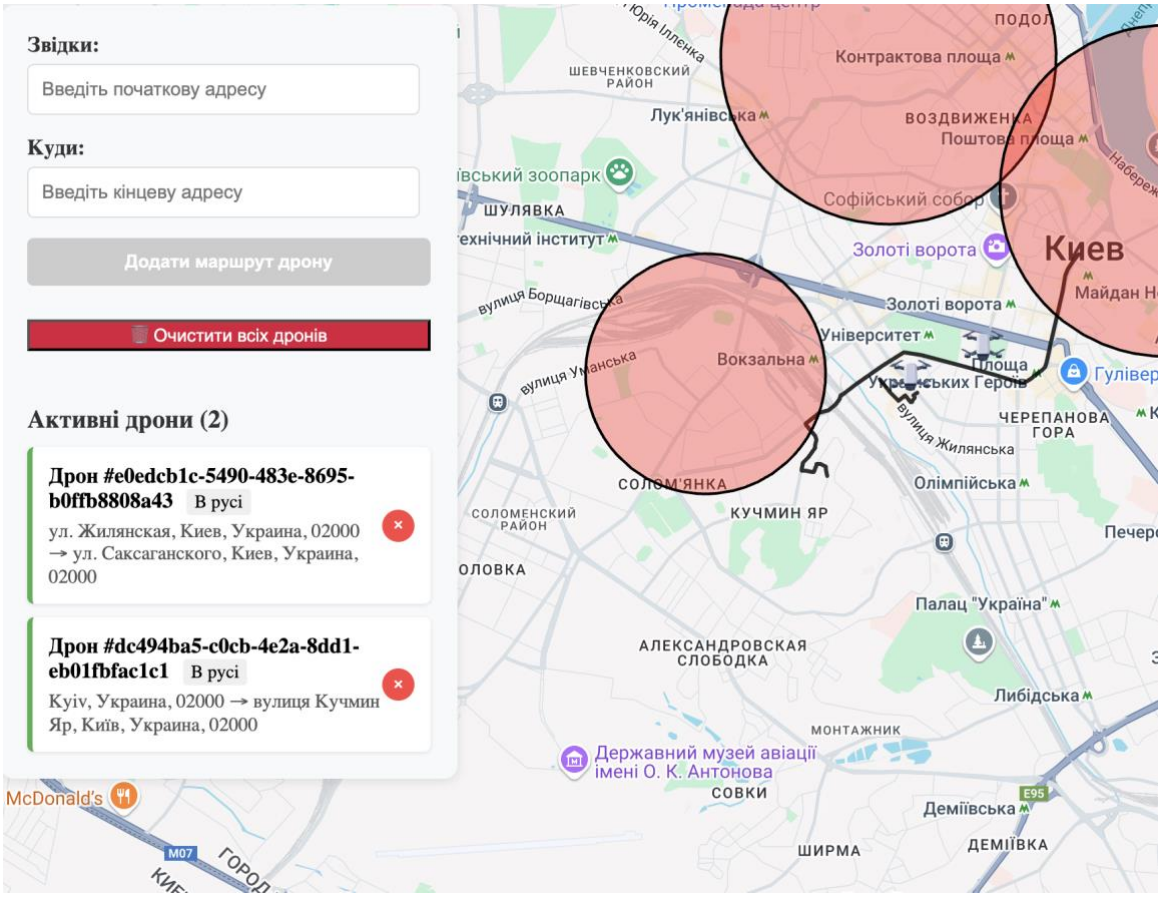

Рисунок 3.7 – Приклад візуальних можливостей застосунку

Далі якщо дивитися у нижню праву сторону екрану, то користувач помітить чат з ШІ (рис. 3.8), за допомогою якого користувач не буде обмежений лише графічним елементом інтерфейсу. Тому він має можливість надсилати звичайні текстові команди, які будуть розпізнаватися штучним інтелектом.

Чат побудований на принципі живої взаємодії, де користувач вводить команду, а система штучного інтелекту розпізнає завдання, та перетворює його на структурований Json об’єкт, який далі передається на сервер. Такий підхід перетворює природну мову у формат, який буде зрозумілий системі. В результаті чого користувачу не потрібно знати точні координати, або параметри маршруту, йому буде достатньо вказати місце призначення та тип дії, а вся логіка виконається автоматично.

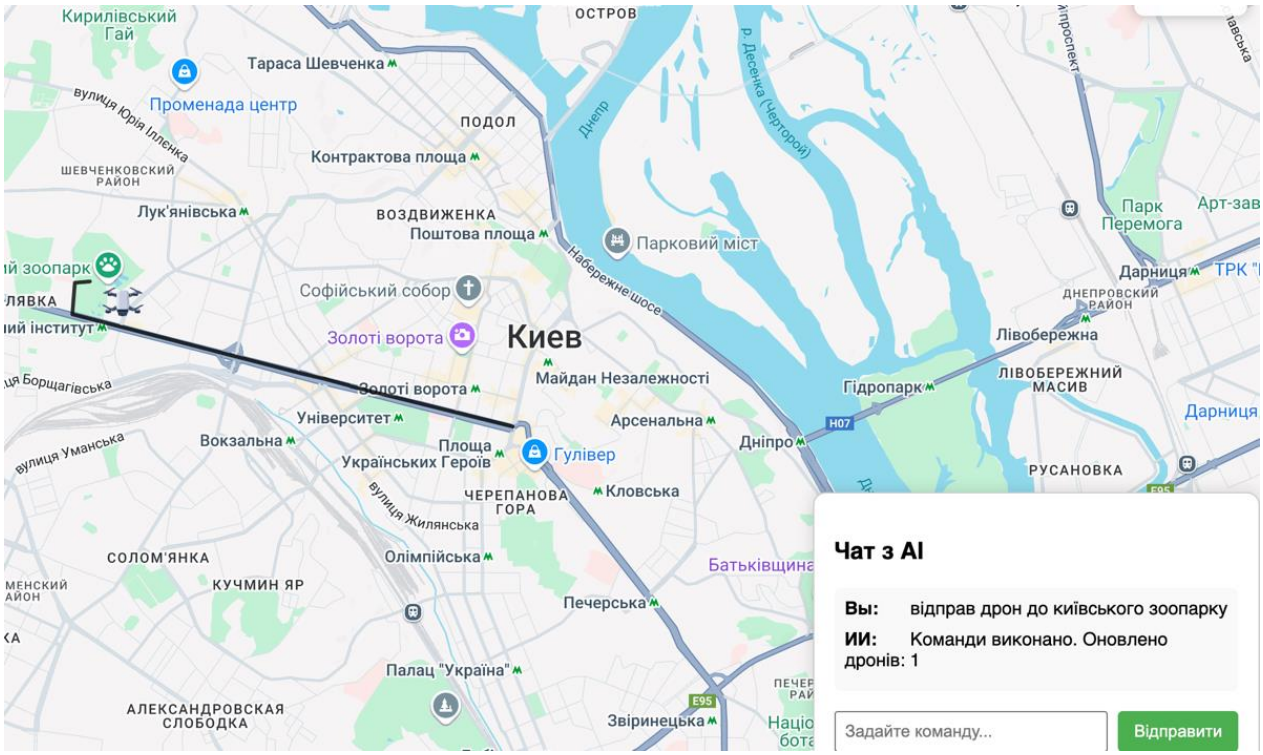

Рисунок 3.8 – Приклад взаємодії з ШІ у текстовому чаті

Особливістю чату є те що, він тісно взаємодіє з системою, де після команди користувача, відразу запускається обробник маршруту, перевіряються погодні умови і за потреби підключається штучний інтелект для побудови оптимального

шляху. Користувач же отримає повідомлення про успіх завдання, або помилку якщо вона могла виникнути. Такий спосіб створює відчуття взаємодії з розумним помічником, а не просто взаємодії з інтерфейсом.

Крім того у майбутньому, цей чат можна розширяти та модифікувати для підтримки більш складних запитів. У майбутньому це буде потребувати більш складної логіки розбору та аналізу, однак за допомогою великої мовної моделі така функціональність вже частково присутня або легко реалізується.

3.3.1 Обмеження локальної системи

Незалежно від функціональності розробленого застосунку, необхідно розглянути важливі обмеження які присутні при локальному встановленні застосунку, та які система буде мати в реальних умовах застосування. Ці обмеження виникають не тільки з боку апаратних ресурсів, а й з боку технічних рішень обраних під час розробки застосунку. Розгляд цих факторів є край необхідним для подальшого розвитку та масштабування застосунку.

Одним із головних обмежень є ефективність великої мовної моделі, незважаючи на те що модель обирається під характеристики операційної системи. Обрана модель Ministral 7B є досить оптимізованою та швидкою, але все ще вимагає багато ресурсів. Для її запуску на локальній машині необхідно не менше 8 ГБ оперативної пам’яті, а також стабільне та потужне CPU або GPU. Якщо у системі буде недостатньо оперативної пам’яті, або буде застаріле обладнання то час відповіді моделі значно збільшиться, що напряму вплине на продуктивність системи. Крім того локальна модель має обмеження на об’єм інформації яку вона може обробити. А це в свою чергу ускладнює роботу з великою кількістю агентів, оскільки обробка усієї інформації про них може перевищити ліміт токенів яких може обробити модель.

Ще одним обмеженням є WebSocket, хоча він і забезпечує двоканальний режим роботи в реальному часі, але теж має свої обмеження у продуктивності.

Тобто при зростанні кількості підключень або при одночасній взаємодії великої кількості дронів сервер може не впоратись через велике навантаження, що може призвести до розриву з’єднання, або затримці у передачі даних.

Якщо дивитись з архітектурного боку, система працює у двовимірному просторі. Положення маршруту та погодні явища представлені на площині, та це є допустимим у тестовій симуляції, але якщо переводити це до реального тривимірного простору, то цього не достатньо. Поточна реалізація не враховує висоту або повітряні перешкоди. З цього виходить що це також є одним із функціональних обмежень системи, якщо розглядати її використання з реальними дронами.

Таким чином ми маємо високий рівень реалізації та інтеграції між компонентами, але система має обмеження з продуктивністю. Але усі ці обмеження є типовими для тестових систем, та з годом у наступних версіях можуть бути вирішені, за допомогою розширення функціональності, або при введені нових модулів.

3.3.2 Ідеї для майбутнього розвитку системи

Подальше вдосконалення розробленої системи, природньо прямує за межі стимуляційного простору, та відкриває велику кількість перспективних напрямів для удосконалення. В першу чергу постає завдання у інтеграції вже з реальними безпілотними апаратами за допомогою офіційних виробників. Поєднання реальних безпілотників з системою, дає змогу передавати сформовані маршрути прямо на бортове устаткування дрона, а також у зворотному напряму отримувати телеметрію, або інші данні з датчиків дрону для відображення їх у клієнтському інтерфейсі. Але реальне трьох вимірне середовище потребує точних обчислень, тому при переході з двовимірного середовища у трьох вимірного, нам не уникнути обчислень висоти польоту, швидкості вітру та

географічні обмеження що можуть виникати на певних ділянках простору, тому всі ці обчислення повинні бути на етапі розрахунку маршруту.

У іншому векторі розвитку даної системи є впровадження голосового інтерфейсу. Сучасні технології розпізнавання мови, легко можна інтегрувати у цей застосунок. Перетворення голосових команд у текст за допомогою WebRTC технології, значно спростить взаємодію з великою кількістю агентів, що в свою чергу підвищить ергономіку управління системою в цілому. Самі великі мовні моделі вже пристосовані для обробки голосу тому додавання цієї технології, лише розширить функціонал застосунку, та не принесе великих змін в архітектуру проєкту.

Отже є багато напрямків для розвитку даної системи. Але кожне з цих покращень вимагає перенесення проєкту з локального застосування у хмарне середовище, де усі необхідні обчислення не будуть виконуватися на персональному комп'ютері з обмеженими обчислювальними здібностями, проте перенесення проєкту у хмарне середовище, дасть можливість ще більшого масштабування системи, з можливістю виконання більш складних завдань для великої мовної моделі.

3.3.3 Тестування застосунку

Під час розробки вебзастосунку, дуже важливе проведення тестування, що дозволить перевірити всі ключові аспекти застосунку такі як, стабільність, надійність або функціональність всіх компонентів [13]. Особливо потрібно перевірити взаємодію між мікросервісами, обробку команд моделі LLM та швидкість застосунку.

Тестування функціоналу охоплювало основні сценарії які міг зробити користувач: побудова маршруту, додавання дрона, відправка команди через чат, відбудова маршруту на мапі та обробка повідомлень. Всі ці аспекти були

протестовані вручну з різними даними. Система стабільно виконувала усі необхідні завдання, а некоректні запити оброблялися без аварійного завершення.

Найбільш важливим стало тестування WebSocket з'єднання, яке забезпечує постійний зв'язок. Для тестування було створено сценарій де застосунком одночасно користувалося декілька клієнтів. Навіть при швидкій зміні положення великої кількості дронів система працювала стабільно, а обмін повідомленнями залишався синхронним та надійним [17].

Навантажувальне тестування, показало продуктивність системи при збільшенні кількості активних агентів, та вимірювання часу відповіді сервера показали що система витримує навантаження, але зі збільшенням кількості агентів час відповіді LLM моделі поступово зростав, тому для подальшого розвитку цього застосунку потрібні більші потужності обчислювальної машини.

Оскільки система орієнтована на велику кількість агентів, було важливо провести оцінку, як вона поводиться при зростанні навантаження. В експериментальних умовах у симуляцію поступово додавали більшу кількість дронів, котрі паралельно отримували команди від LLM моделі. Отримані результати дали виявити точки навантаження, після яких система починає втрачати стабільність, що стало підґрунтям для подальшої оптимізації, хоча критична точка, це запуск системи на персональному комп'ютері із за невеликих обчислювальних потужностей.

3.3.4 Оптимізація продуктивності

Оптимізація стала необхідністю етапу доопрацювання, через те що час відгуку на пряму впливає на якість керування в реальному часі. Першим кроком була оптимізація розміру запитів, оскільки великий об'єм інформації викликає затримку. Завдяки цьому фактору було прийнято рішення зменшити кількість даних, що передаються до LLM моделі, залишивши лише ті що впливають на прийняття рішення. Також було оптимізовано асинхронну взаємодію між

компонентами завдяки впровадження черг повідомлень. Ці зміни дали змогу досягти зменшення затримки відповіді та виконання дії. Хоча і не вдалося досягти великих успіхів через обчислювальні потужності персонального комп’ютера.

3.4 Локальний запуск проєкту

Щоб запустити симуляцію багатоагентного середовища було реалізовано локальне рішення. Це дасть змогу проєкту працювати незалежно від зовнішніх серверів, що в свою чергу дозволить краще протестувати систему та дослідити її архітектуру.

В перші кроки потрібно зробити налаштування серверної частини. Так як back-end написаний на мові Python, то нам необхідно мати встановлене середовище Python 3.10, або вище. В корені проєкту є файл requirements.txt, в ньому зберігаються усі необхідні бібліотеки для роботи з проєктом. Щоб встановити всі необхідні залежності для проєкту, то потрібно запустити термінал та виконати команду для встановлення «pip install». Після чого усі залежності які знаходяться у файлі requirements.txt автоматично завантажаться.

Для запуску серверної частини застосовується FastAPI в режимі розробки. Саме цей режим дасть змогу бачити усі помилки у консолі. Для запуску сервера потрібно в термінал ввести команду «source backend/venv/bin/activate» для запуску віртуального середовища, а вже після команду «uvicorn backend.main:app –reload», де префікс «–reload» буде перезавантажувати сервер відразу після внесення змін до коду, що прискорить написання та редагування серверної частини. Сервер після запуску автоматично підключається до БД, яка локально розвернута на комп’ютері. У файлі .env зберігаються усі ключові підключення такі як, адреса БД, секретні токени, API ключи для Google Maps, а також необхідні підключення до ШІ.

Окрім серверної частини також потрібно запускати клієнтський інтерфейс або front-end, оскільки ця частина написана на React то для її запуску необхідно мати встановлений Node.js. Після встановлення для його запуску необхідно через термінал спочатку перейти у кореневу папку проєкту front-end командою «cd frontend», а потім запустити його командою «npm start», одразу після запуску у браузері відкриється сторінка за адресою http://localhost:3000 .

Однією з ключових складових проєкту є LLM модель, яка теж працює локально, через платформу Ollama. Після встановлення Ollama користувачу необхідно завантажити модель Ministral 7b також через команду в терміналі. Після чого для її запуску потрібно буде ввести команду «ollama serve». Модель запуститься у фоновому режимі, та буде очікувати поки до неї не звернуться. Також потрібно запускати локально MongoDB якщо він автоматично не запускається з включенням комп'ютера командою «mongosh».

Після запуску усіх необхідних компонентів, для перевірки WebSocket з'єднання достатньо лише відкрити сторінку у браузері де одразу після входу в аккаунт зверху буде надпис (рис. 3.9).

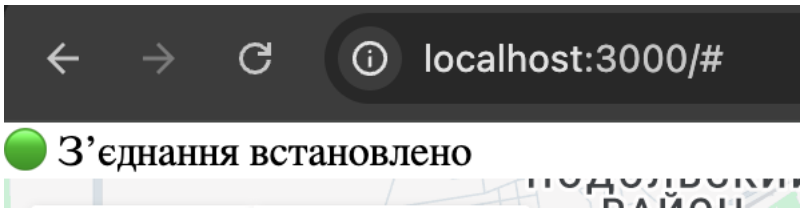

Рисунок 3.9 – Приклад перевірки WebSocket з'єднання

Якщо ж трапилася якась помилка то у терміналі вона одразу буде показана, а на сторінці в браузері буде горіти червона лампа з надписом «Немає з'єднання».

Тобто для повного запуску локального проєкту необхідно ввести 6 простих команд у терміналі, після чого можна буде приступати до роботи з застосунком.

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено і реалізовано тестову програму для керування багатоагентною системою яка має вигляд симуляції дронів з використанням інтелектуальної підтримки у вигляді великої мовної моделі. В процесі роботи було досліджено сучасні підходи з реалізацією систем які мають підтримку штучного інтелекту, було проведено глибокий аналіз багатьох мовних моделей які застосовуються локально, та обрано оптимальне рішення, яке забезпечує виконання всіх необхідних потреб застосунку. Саме через це інтеграція LLM через платформу Ollama дала можливість досягнути автономної роботи застосунку без потреби в зовнішніх ресурсах, що в свою чергу підвищило безпеку та стабільність системи.

Окрему увагу було привернуто побудові проєкту на основі мікросервісної архітектури, що дало змогу розподілити логіку між окремими незалежними сервісами, кожен з яких виконує певну чітку функцію. Це дало зручну масштабованість та легкість в обслуговуванні всієї системи в цілому.

Розроблений вебзастосунок дає можливість керувати агентами за допомогою ручного інтерфейсу а також за допомогою текстових команд через чат. Вебінтерфейс дає відображення стану дронів на карті в реальному часі, також відстеженням погодних зон, а також будування маршрутів з урахуванням цих самих зон.

Таким чином результати кваліфікаційної роботи демонструють не лише знання теоретичної частини, а й практичні навички що були здобуті у період навчання. Створений застосунок є реальним прикладом ефективного керування багатоагентною системою, та може стати основою для подальших досліджень у даній сфері. Ця кваліфікаційна робота дала змогу поглибити навички у напрямках таких як веброзробка, програмна інженерія, робота з базами даних, застосування штучного інтелекту, а також моделювання багатоагентних систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Харківський національний університет радіоелектроніки. (n.d.). Навчальний матеріал з відкритого архіву. URL : https://openarchive.nure.ua/server/api/core/bitstreams/656c4758-b054-45ec-99bf-5fe69892bd94/content (Дата звернення: 02.05.2025).

2. Kuzomin, O., & Lyashenko, V. (2022). Agent-Based Model as a Research Tool. URL: https://openarchive.nure.ua/bitstream/document/20459/1/KuzLyas05.pdf

3. Рідкокаша, А. А., & Голдер, К. К. (2002). Основи систем штучного інтелекту. Навчальний посібник. *Черкаси: Відлуння-Плюс*.

4. Technology Innovation Institute. (n.d.). Home. URL : https://www.tii.ae/ (Дата звернення: 02.05.2025).

5. Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

6. FlyEye. (n.d.). Drone Technology & Flight Control Systems. URL : https://www.flyeye.io/drone-technology-flight-control-systems/ (Дата звернення: 02.05.2025).

7. Prompting Guide. (n.d.). Mistral 7B. Prompting Guide. URL : https://www.promptingguide.ai/ua/models/mistral-7b (Дата звернення: 02.05.2025).

8. Mittal, A. (2024, July 31). Everything you need to know about Llama 3: Most powerful open-source model yet – Concepts to usage. Unite.AI. URL : https://www.unite.ai/everything-you-need-to-know-about-llama-3-most-powerful-open-source-model-yet-concepts-to-usage/ (Дата звернення: 02.05.2025)

9. Farabet, C., & Warkentin, T. (2025, March 12). Gemma 3: Google's новий Open Model заснований на Gemini 2.0. Google Blog. URL : https://blog.google/technology/developers/gemma-3/ (Дата звернення: 02.05.2025).

10. Bilenko, M. (2024, April 17). Introducing Phi-3: Redefining what's possible with SLMs. Microsoft Azure Blog. URL : https://azure.microsoft.com/en-

us/blog/introducing-phi-3-redefining-whats-possible-with-slms/ (Дата звернення: 02.05.2025).

11. Ollama. (n.d.). Falcon 7B. URL : https://ollama.com/library/falcon:7b (Дата звернення: 02.05.2025).

12. Romin, P. (2020). Unraveling Microservices: A study on microservices and its complexity.

13. Python Software Foundation. (n.d.). Перші кроки у Python (версія 3.13). URL : https://docs.python.org/uk/3.13/tutorial/appetite.html (Дата звернення: 02.05.2025).

14. Pallets Projects. (n.d.). Flask Documentation (Stable). URL : https://flask.palletsprojects.com/en/stable/ (Дата звернення: 02.05.2025).

15. Наукова періодика України. URL: https://journals.uran.ua/itssi/article/view/308821/300355 (Дата звернення: 02.05.2025)

16. Танянський, О., & Руденко, Д. (2018). Порівняльний аналіз популярних JavaScript-фреймворків та бібліотек для front-end розробки.

17. ХНУРЕ. (2023). Навчальний курс «Вебпрограмування» [внутрішній методичний матеріал]. URL : https://elearn.nure.ua/course/view.php?id=510 (Дата звернення: 02.05.2025)

18. Banker, K., Garrett, D., Bakkum, P., & Verch, S. (2016). *MongoDB in action: covers MongoDB version 3.0*. Simon and Schuster.

19. Mozilla Developer Network. (n.d.). WebSocket API. URL : https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (Дата звернення: 02.05.2025)

20. McConnell, S. (2004). Code Complete. Microsoft Press.

21. Google Developers. (n.d.). Maps documentation. URL : https://developers.google.com/maps/documentation (Дата звернення: 02.05.2025).

22. Technology Innovation Institute. (n.d.). Home. URL : https://www.tii.ae/ (Дата звернення: 02.05.2025).

23. Сухоруков, В. Є. (2019). Архітектура комп’ютерних систем. Харків: ХНУРЕ.

24. Бінько, І., & Шевель, В. (2024). Комплексний підхід до управління формуванням групи роботів. *Сучасний стан наукових досліджень та технологій в промисловості*, (2 (28)), 17-32.

25. Центральноукраїнський державний університет імені Володимира Винниченка. (n.d.). Тестування. Методичні вказівки. URL : https://eprints.cdu.edu.ua/1482/1/testyvan.pdf (Дата звернення: 02.05.2025).

26. Тютюник, О. О. (2023). Системи BI (business intelligence): робоча програма навчальної дисципліни для здобувачів вищої освіти спеціальності 126" Інформаційні системи та технології" освітньої програми" Інформаційні системи та технології" першого (бакалаврського) рівня.

27. Кузьомін, О. Я. (2022). Кейс-стаді з моделювання MAS систем: збірник прикладів. Харків: ХНУРЕ.

28. Van Rossum, G. (n.d.). The “Blurb” about Python. URL : https://www.python.org/doc/essays/blurb/ (Дата звернення: 02.05.2025).

29. Iryna, T., & Heorhii, M. (2021). TO THE QUESTION OF ANALYSIS OF EXISTING MECHANISMS OF WEB APPLICATION TESTING. Problems of modern science and practice, 1, 403.

30. Massachusetts Institute of Technology (MIT). (n.d.). Architectures of Multi-Agent Systems. URL : https://mas.mit.edu/projects/architectures/ (Дата звернення: 02.05.2025)

31. Кузьомін О.Я. Алгоритми та методи штучного інтелекту для інформаційних систем: посібник. – Харків. ХНУРЕ, 2023. – 396 с. (РЕКОМЕНДОВАНО Вченою радою університету Протокол № 1\17 від 31.01.2023)