МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____
"_____"_____202_ р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: «Використання технологій Інтернету речей для керування обслуговуванням пасажирів на території аеропортів»

Виконавець: Маркулич Роман Олександрович

Керівник: Зубок Віталій Юрійович

Консультанти з окремих розділів пояснювальної записки:

Нормоконтролер:

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні системи та технології»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«_____»_____202 __ р.

ЗАВДАННЯ
на виконання дипломної роботи (проєкту)
Маркулича Романа Олександровича

1. Тема дипломної роботи (проєкту): «Використання технологій Інтернету речей для керування обслуговуванням пасажирів на території аеропортів»,
затверджена наказом ректора від 15.02.2022 р. №251/ст.
2. Термін виконання роботи (проєкту): з 16.05.2022 по 19.06.2022
3. Вихідні дані до роботи (проєкту): програмний комплекс, що забезпечує інтеграцію технологій інтернету речей у прототип системи керування аеропортом.
4. Зміст пояснювальної записки:
    1. Аналіз предметної області
    2. Специфікація вимог до програмного забезпечення для оптимізації обслуговування пасажирів аеропорту із використанням технологій інтернету речей
    3. Розроблення програмних модулів для програмно-технологічного рішення
    4. Прототипування. Тестування прототипу
5. Перелік обов'язкового графічного (ілюстративного) матеріалу:
    - діаграма потоків даних
    - діаграма архітектури інформаційної системи
    - мультимедійна презентація із демонстрацією роботи програми

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Ознайомитись з постановкою задачі дипломного проектування	9.05.2022	
2	Вивчити спеціальну літературу і технічну документацію	14.05.2022	
3	Проаналізувати предметну область та існуючі рішення	16.05.2022	
4	Написати перший розділ	18.05.2022	
5	Написати другий розділ	22.05.2022	
6	Написати третій розділ	26.05.2022	
7	Завершити розробку програмного комплексу	27.05.2022	
8	Написати четвертий розділ	28.05.2022	
9	Виконати оформлення пояснювальної записки	29.05.2022	
10	Підготувати графічний матеріал	01.06.2022	

7. Консультанти з окремих розділів

Розділ	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв

8. Дата видачі завдання: “__" ___ 2022р.

Керівник дипломної роботи (проєкту) _____ Зубок Віталій Юрійович
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Маркулич Роман Олександрович
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи на тему «Використання технологій Інтернету речей для керування обслуговуванням пасажирів на території аеропортів»: 46с., 14 рис., 1 табл., 7 інформаційних джерел.

Об’єкт дослідженння: впровадження технологій Інтернету речей у системи обслуговування клієнтів аеропортів.

Мета роботи: створення програмного комплексу, що моделює інформаційну структуру підприємства (аеропорту) із інтеграцією технологій Інтернету речей.

ІНФОРМАЦІЙНІ СИСТЕМИ, ОБСЛУГОВУВАННЯ КЛІЄНТІВ, СИСТЕМИ КЕРУВАННЯ, СЕРВЕРНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ІНТЕРНЕТ РЕЧЕЙ, БАЗИ ДАНИХ, ВЕБ-ДОДАТКИ, ВІДКРИТИЙ ПРОГРАМНИЙ КОД

ЗМІСТ

ВСТУП

Раніше будь-яка галузь обходилась без залучення у неї складних технологій, а поставлені перед підприємством завдання для впровадження бізнес-процесів вирішувались найбільш доступними способами. Вочевидь, з плином часом усе розвивається, розширяються можливості, та підприємства, які ведуть свої справи із використанням новітніх технологій, будуть на крок попереду від інших. В таких умовах здорова конкуренція в контексті сучасних реалій породжує необхідність як у постачанні та впровадженні існуючих рішень, так і в інноваційних підходах до вирішення завдань. Наведені дані свідчать про те, що використання технологій Інтернету речей для керування обслуговуванням пасажирів є актуальною науково-практичною проблемою.

Метою дипломної роботи є вдосконалення процесу інтеграції технологіями Інтернету речей в систему управління аеропортом шляхом розробки програмного комплексу, який моделює таку систему та її інтеграцію з пристроями Інтернету речей. Технології інтеграції являтимуться предметом дослідження даної роботи.

Методами дослідження у роботі є методи системного аналізу, теорії управління, теорії мереж, теорії алгоритмів.

Для досягнення мети було поставлено та вирішено наступні наукові та практичні завдання:

– огляд систем керування та способів використання технологій IoT в комп'ютеризованих системах управління;

– аналіз існуючих програмно-технологічних рішень, що використовуються в керуванні аеропортами;

– аналіз проблем застосування технологій IoT для керування складними та розподіленими системами;

– формулювання вимог до інформаційної системи та її прототипування.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Теорія керування в обслуговуванні клієнтів

З тих пір, як для спілкування використовувалась лише електронна пошта, обслуговування клієнтів зазнало суттєвих змін. У сучасному світі компаніям для отримання можливості конкурувати за клієнтів доводиться працювати з ними на безлічі різних платформ, щоб охопити якнайбільші аудиторії. Однак, і цього вже недостатньо: у наш час компаніям слід мати на увазі і багато інших аспектів клієнтського обслуговування. Технології мають неабияке значення у факторах успіху цієї задачі.

Одним із основних факторів є оперативність – те, як швидко сервіси підприємства реагують на звернення клієнтів. Ніхто не любить очікувати в черзі – життя для цього занадто коротке. А якщо клієнту не сподобається повільне клієнтське обслуговування, та черги на відповідь чи поставку послуг будуть куди більшими, ніж у ваших конкурентів – це стане однією із ключових причин скористатись їхніми послугами чи продуктами замість ваших. Зайвим буде сказати, що це не в інтересах підприємства, яке має на меті ріст і максимізацію прибутку.

Саме тут в нагоді стають технології, які готові допомогти підприємствам вирішити проблеми, які перешкоджають оптимальному потоку обслуговування – про що далі і буде іти мова у роботі.

Варто згадати і про мотивацію персоналу, оскільки це також впливає і на оперативність роботи сервісів. Мотивовані фахівці клієнтського обслуговування щасливіші, більш зацікавлені та ефективніші в роботі — а це ті якості, до наявності

яких у своєму персоналі повинен прагнути будь-який роботодавець. Звісно, тут грають роль такі фактори, як заробітна плата, страхування, внутрішні настрої співробітників та багато інших. Проте, залишаючись в межах контексту, зосередимо увагу на умовах праці. Оскільки це саме те, що знову ж таки можна покращити впровадженням технологій. Людина, яка працює із використанням новітнього обладнання та/або користується комунікаційними системами, які щільно вплетені у робочий процес, матиме високу продуктивність та навіть почуття власної значимості. Всі ці деталі вкупі формують бачення, що компанія цінує своїх співробітників, забезпечуючи їх зручними умовами та звільняючи від рутинних дій, які несуть в собі мале корисне навантаження. Це дуже позитивно впливає на сприйняття людьми свого робочого процесу, та згідно з моєю точкою зору, яку зазначено на початку абзацу, задоволений співробітник – задоволений клієнт.

Ще один аспект, який варто згадати у впровадженні рішень для керування бізнес-процесами – доступність. Якщо ваша компанія працює не в безперервному режимі, клієнти знову ж таки можуть обрати конкурента. Операторів каналів зв’язку варто забезпечити надійними і зручними програмними продуктами, а якщо підприємство працює не цілодобово – можна навіть подбати про клієнтське самообслуговування хоча б з частковим функціоналом.

Окрім поняття “Uptime” (безвідмовний час роботи), термін «доступність» також означає і “Usability” (зручність використання). Користувацькі інтерфейси для взаємодії із системами, які рухають бізнес-процеси, повинні бути ретельно продуманими на стадії проектування, щоб досягнути тандему функціоналу та зручності використання. Взаємодія клієнта із сервісами повинна бути простою, аби охопити більшу аудиторію, а взаємодія співробітника – максимально продуктивною.

Використання технологій інтернету речей може позитивно вплинути на швидкість роботи, викликати захоплення у людей як відносно новітня технологія, а взаємодія із ними або мінімальна, або повністю непомітна.

1.2 Загальні відомості про Інтернет речей

За останні роки Інтернет речей став однією з найважливіших технологій. Ми можемо підключати повсякденні предмети – автомобілі, кухонну техніку, датчики – до Інтернету за допомогою вбудованих пристроїв, що робить можливим безперебійне «спілкування» між людьми та машинами. За допомогою хмарних обчислень фізичні пристрої можуть обмінюватися та збирати дані з мінімальним втручанням людини. У цьому тісно зв’язаному світі цифрові системи можуть записувати, контролювати та коригувати кожну взаємодію між підключеними речами.

Ідея Інтернету речей існувала вже давно – але повністю придатною для практичного застосування її зробили саме досягнення у інших областях інформаційних технологій, а саме:

- Малопотужні та недорогі сенсорні пристрої – доступні датчики становлять основу поширення практичного використання IoT;
- Підключення – мережеві протоколи для Інтернету дозволяють легко підключати датчики до хмари та інших «речей» для ефективної передачі даних;
- Платформи хмарних обчислень – збільшення доступності хмарних платформ дає можливість споживачам отримати доступ до інфраструктури, яка їм потрібна для розширення та простого керування;
- Машинне навчання та аналітика – завдяки досягненням у сфері машинного навчання та аналітики, компанії можуть швидше та легше отримувати статистику. Поява цих суміжних технологій продовжила розширювати межі Інтернету речей;
- Розмовний штучний інтелект – розвиток нейронних мереж зробив можливою обробку NLP (Natural Language Processing) на пристроях IoT,

що дало життя цифровим персональним помічникам та багатьом іншим застосункам.

Найшвидше Інтернет речей проник саме у промислові галузі та викликав там справжню революцію. Для досягнення бездротової автоматизації та керування промисловими підприємствами почали використовувати хмарні технології вкупі із суміжними технологіями, такими як аналітика та машинне навчання. Галузі тепер можуть досягти нового рівня автоматизації, а разом з ним створити нові підходи та бізнес-моделі. Виробники можуть отримати конкурентну перевагу, використовуючи моніторинг виробничої лінії, щоб забезпечити активне технічне обслуговування обладнання, коли датчики виявляють збій. За допомогою сенсорних оповіщень виробники можуть швидко перевірити обладнання на точність або зняти його з експлуатації для ремонту. Це дозволяє компаніям знизити експлуатаційні витрати, збільшити час безвідмовної роботи та покращити управління і продуктивність.

Інтернет речей проник і в індустрію автомобілів - датчики можуть виявляти несправність обладнання в транспортних засобах, які вже знаходяться на дорозі, і можуть сповіщати водія за допомогою деталей та рекомендацій. Завдяки інформації, зібраної за допомогою додатків, виробники та постачальники автомобілів можуть дізнатися більше про те, як підтримувати автомобілі в працездатності та інформувати власників автомобілів.

Від розвитку Інтернету речей у виграші не лише виробництва. Для прикладу, широкого використання IoT набуло відразу і в логістиці. Вище уже було згадано про оснащення автомобілів різними датчиками, проте і самі вантажні контейнери можуть бути оснащені датчиками температури та інших вимірювальних приладів для контролю дотримання умов транспортування, що важливо у харчовій, фармацевтичній та навіть квітковій галузях. Для прикладу, у цьому випадку програми моніторингу з використанням IoT надсилають сповіщення, коли температура підвищується або падає до рівня, який загрожує продукту.

Інтернет речей може допомогти оптимізувати бізнес, навіть коли товар уже на полицях. Чудовим прикладом стануть, власне, «смарт-полиці», які мають вбудовані

ваги та можуть надсилати сповіщення, коли навантаження падає нижче необхідного – товари на полицях будуть вчасно поповнені і продажі не припинятимуться навіть на мить.

Технології інтернету речей можуть допомагати і у державному секторі – комунальні служби можуть відстежувати перебої у постачанні послуг громадянам, локалізувати проблеми, сповістити користувачів та формувати звіти.

Окрім відстеження фізичних активів, Інтернет речей можна використовувати для підвищення безпеки працівників. Наприклад, працівники у небезпечних середовищах, таких як шахти, нафтові й газові родовища, хімічні та електростанції, повинні знати про виникнення небезпечної події, яка може вплинути на них. Коли вони підключені до додатків на основі датчиків IoT, їх можна сповіщати про аварії або рятувати від них якомога швидше. Додатки IoT також використовуються для носіїв, які можуть контролювати здоров'я людей та умови навколишнього середовища. Ці типи програм не тільки допомагають людям краще зрозуміти власне здоров'я, вони також дозволяють лікарям дистанційно контролювати пацієнтів.

Як ми бачимо, Інтернет речей є дуже широкою областю застосування і може стати в нагоді як для організації елементів критичної інфраструктури, так і для дрібниць, які просто роблять наше життя зручнішим. Або навіть може організувати систему глобального стеження за громадянами із використанням біометричних даних, із залученням усіх можливих реєстрів і державних служб, як це зроблено при диктатурі у одній із великих держав Азії. Одночасно з тим, наскільки це лячно і антиутопічно, це демонструє безмежний потенціал Інтернету речей, і прямо зараз мільярди людей відчувають вплив цих технологій на їхнє життя.

1.3 Існуючі програмно-технологічні рішення

Як уже згадано вище, Інтернет речей активно використовується у різноманітних сферах. Оскільки ця технологія охоплює будь-яку взаємодію якихось матеріальних конкретних прикладних речей із будь-якою інформаційною системою, ми зосередимо увагу на рішеннях, які втілені в сучасних аеропортах світу для керування ним, оптимізації або ж в цілях маркетингу.

Аеропорт Гельсінкі-Вантаа (HEL). Цей аеропорт використовує пропрієтарні технології Walkbase, які допомогають розвивати комерційний сектор аеропорту та надають дані для оптимізації пасажиропотоку. Якщо бути конкретним, там використовуються Wi-Fi датчики, які сканують пристрої поблизу із увімкненим Wi-Fi. На основі цього єдина система із таких датчиків може співставляти дані із кожного з них, та формувати звіти, де ж пасажири затримуються більше всього, або якими шляхами проходять. Деталі реалізації невідомі, але можна припустити, що аби не запрошувати у клієнтів додаткові мережеві пакети, не вимагати від них підключення до локальної мережі та збирати максимальну кількість даних, на цих пристроях повинні бути встановлені Wi-Fi модулі, які підтримують роботу у так званому режимі моніторингу (ориг. monitor mode). Wi-Fi модуль при цьому перестає працювати із протоколом TCP/IP та вищими рівнями, і починає вловлювати усі дані на усіх видимих каналах, виступаючи своєрідним «сніффером» (програмним забезпеченням, яке призначене для перехоплення даних) для незашифрованих пакетів. Таким чином можна збирати дані і про людей, які не підключені ні до якої мережі – оскільки просто увімкнений Wi-Fi на пристрої час від часу надсилає «зонди» (ориг. probe frames) для пошуку точок доступу поблизу. Апаратне забезпечення із підтримкою режиму моніторингу частот Wi-Fi може збирати про них свідчення, такі як MAC-адреса, що дозволяє ідентифікувати джерело сигналу, та звідси, вирахувати кількість унікальних пристроїв в певній зоні. Кластер таких пристроїв дозволяє і встановити місцезнаходження методами триангуляції в локальних масштабах, оскільки можна зібрати дані і про силу сигналу від джерела.

Аеропорт Маямі в США (MIA). В аеропорту мегаполісу на південному сході США до мережі підключили 500 Bluetooth-маячків. Ці пристрої Bluetooth відправляють на мобільні пристрої пасажирів дані про вихід на посадку, інформацію

про різні торгові точки та заклади. Сама технологія не така складна, як у вищеописаному прикладі – сервіс доступний відвідувачам, які завантажили та встановили мобільний додаток аеропорту, що вже створює потребу взаємодії з клієнтської сторони і в певній мірі зменшує частку потенційних користувачів. Власне, реалізовано це достатньо просто – із використанням програмних інтерфейсів операційної системи мобільного пристрою застосунок встановлює Bluetooth-з'єднання з найближчим пристроєм і розпочинає обмін даними. З його допомогою можна сканувати посадкові талони, будувати маршрути різними секторами аеропорту та виконувати інші дії.

Лондонський аеропорт Heathrow (LHR). У терміналах аеропорту Heathrow планують встановити сканери розпізнавання осіб пасажирів. Завдяки цьому для реєстрації на рейс клієнтом в майбутньому більше не знадобляться паспорти та посадкові талони. Щоб полегшити процедуру реєстрації, потрібно буде завантажити паспортні дані та фото у спеціальний додаток, і така система вже тестується останні кілька років. Основною заслугою, що зробила це можливим, є стрімкий розвиток технологій штучного інтелекту, обговорення деталей реалізації якого вже знаходиться поза межами контексту.

У власному проекті також змодельовано підключення пристроїв розпізнавання особистості до веб-сервера, який є центром інформаційних процесів підприємства, що дозволяє втілити будь-які сценарії використання даних, отриманих з Інтернету речей.

1.4 Проблематика області

Технології Інтернету речей своєю силою і розповсюдженістю проникли у всі сфери життя та наростили неабиякий вплив як на індустрію, так і на соціум. Однак, як відносно нова технологія, вона має в собі певні недосконалості, і потрібно бути

обізнаним, щоб технології, які створені для покращення нашого життя, не принесли нікому шкоди.

Перелік найпоширеніших проблем систем із використанням IoT зі стислим описом:

Неправильний контроль доступу. Послуги, що пропонуються пристроєм IoT, повинні бути доступні лише власнику та людям із їхнього найближчого оточення, яким вони довіряють. Однак це часто недостатньо забезпечується системою безпеки пристрою. Пристрої Інтернету речей можуть довіряти локальній мережі до такого рівня, що не вимагають подальшої аутентифікації або авторизації. Будь-який інший пристрій, підключений до тієї ж мережі, також вважається довіреним. Це особливо гостра проблема, коли пристрій підключено до Інтернету: тепер кожен у світі потенційно може отримати доступ до функцій, які пропонує пристрій. Тому мережа пристроїв IoT повинна бути як мінімум ізольована від зовнішніх мереж, або знаходитись за ретельно налаштованим фаєрволом.

Вкрай велика поверхня атаки. Кожне з'єднання, яке можна встановити з системою, надає зловмисникові нові можливості для виявлення та використання вразливостей. Чим більше послуг пристрій пропонує через Інтернет, тим більше сервісів можна атакувати. Це відомо як поверхня атаки. Зменшення поверхні атаки є одним із перших кроків у процесі захисту системи. Пристрій може мати відкриті порти з запущеними службами, які не є строго необхідними для роботи. Атаку на такий непотрібний сервіс можна легко запобігти, не розкриваючи сервіс до зовнішньої мережі (Telnet, SSH або інший інтерфейс налагодження можуть відігравати важливу роль під час розробки, але рідко потрібні у роботі).

Застаріле програмне забезпечення. Оскільки вразливості програмного забезпечення виявляються та усуваються, важливо розповсюджувати оновлену версію для захисту від уразливості. Це означає, що пристрої IoT повинні поставлятися з сучасним програмним забезпеченням без будь-яких відомих вразливостей, і що вони повинні мати функціональні можливості оновлення, щоб виправити будь-які вразливості, які стали відомі після розгортання пристрою.

Відсутність шифрування. Коли пристрій передає інформацію у вигляді простого тексту (ориг. plain text), вся інформація, якою обмінюється з клієнтським пристроєм або серверною службою, може бути отримана за допомогою MitM-атаки (Man in the middle). Кожен, хто здатний отримати позицію на мережевому шляху між пристроєм і його кінцевою точкою, може перевірити мережевий трафік і потенційно отримати конфіденційні дані, такі як облікові дані для входу.

Відсутність симптом вторгнення. Коли пристрій скомпрометовано, він часто продовжує нормально функціонувати з точки зору користувача. Будь-яка додаткова пропускна здатність або споживання енергії зазвичай не виявляється. Більшість пристроїв не мають функції ведення журналу або оповіщення, щоб повідомляти користувача про будь-які проблеми з безпекою. Якщо вони є, їх можна перезаписати або вимкнути, коли пристрій зламано. В результаті, користувачі рідко виявляють, що їхній пристрій атаковано, відповідно ніякі заходи по відновленню безпеки не вживаються.

Висновок

У першому розділі було в абстрактному розумінні розглянуто теорію успішності керування системою обслуговування клієнтів, виділено критерії, на які потрібно звернути увагу при оптимізації процесу обслуговування із використанням інформаційних технологій, та методи їх покращення за допомогою впровадження технологій Інтернету речей. Розглянуто використання IoT-технологій в контексті предметної області на прикладі інформаційних систем деяких аеропортів світу.

Надано короткі відомості про технологію Інтернету речей в цілому як базис до реалізації власної системи із використанням цих технологій та окреслено проблематику цих технологій з точки зору інформаційної безпеки.

РОЗДІЛ 2. СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОПТИМІЗАЦІЇ ОБСЛУГОВУВАННЯ ПАСАЖИРІВ АЕРОПОРТУ ІЗ ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ ІНТЕРНЕТУ РЕЧЕЙ

2.1 Нефункціональні вимоги та загальний опис продукту

Головне завдання програмного комплексу – змоделювати взаємодію пристроїв Інтернету речей із основною інформаційною інфраструктурою підприємства. В даному випадку буде представлено прототип системи, що має налагоджену взаємодію із пристроями ідентифікації користувачів (із камерами, що розпізнають людей та їх переміщення по зонам аеропорту), базу даних із клієнтами та інформацією про них, та веб-сервер, який за запитами може обробляти цю інформацію в реальному часі та скеровувати інформаційні потоки в інші підсистеми.

Участь клієнта в роботі IoT в даній системі лишається максимально безшовною та не вимагає ніяких дій від нього. Проте, для повного функціоналу користувачеві потрібно бути зареєстрованим у системі клієнтів аеропорту, вказавши персональні дані та додатково підв’язавши сертифікати про вакцинацію.

Результуючий продукт являтиме собою збірку з бібліотеки класів, які містять в собі сервіси для взаємодії веб-протоколами, для взаємодії із пристроями IoT, набір класів для взаємодії з базою даних та власне модель SQL бази даних. Таким чином, це програмне рішення являє собою базис для безмежного нарощування функціоналу, де ідеально підходить ітеративна модель розробки. Детальніше це можна буде побачити у наступному розділі роботи, де розглянеться архітектура.

На рисунку 2.1 далі зображено діаграму інформаційних потоків (DFD – data flow diagram), що в загальних рисах характеризує робочий процес системи.

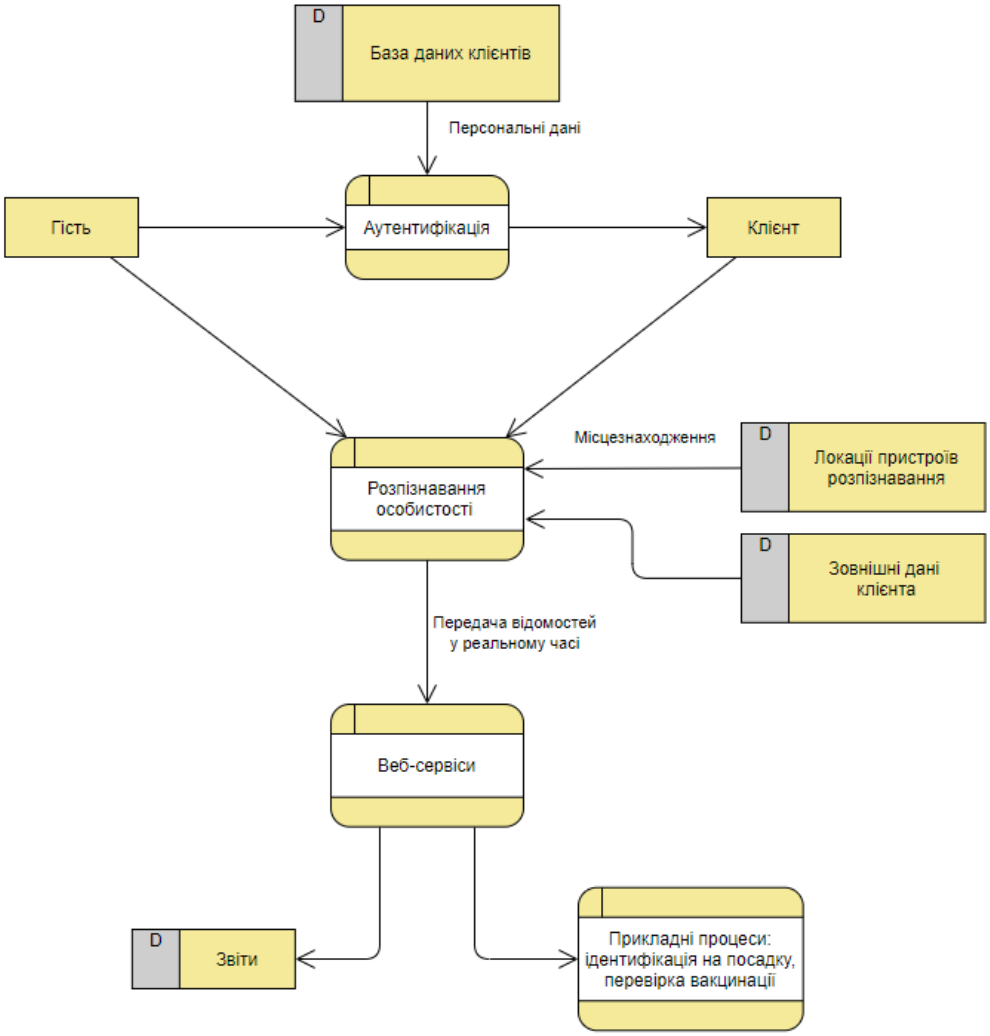

Рис. 2.1 – Data Flow Diagram

2.2 Характеристика протоколів взаємодії з IoT

Розпочнемо із дослідження самого суб'єкту роботи – на чому будується взаємодія із пристроями IoT. Власне те, що перетворює фізичний пристрій у пристрій Інтернету речей – наявність можливості взаємодіяти з ним за допомогою

мережевих протоколів: налагоджувати потоки даних та/або впливати на його внутрішню роботу ззовні. Тому перше питання, яке у нас породжується необхідністю налаштувати комунікацію сервера із пристроями, – які мережеві протоколи підтримуватимуться як пристроями IoT, так і сервером інформаційної системи, та який із них буде оптимальним вибором для розробки і використанні у роботі.

В поточний час у світі IoT пропонуються для використання наступні мережеві протоколи:

- MQQT
- CoAP
- AMQP
- DDS
- XMPP
- WebSocket

Дамо їм стислу характеристику:

MQTT (Message Queuing Telemetry Transport) – мережевий протокол, що працює поверх стеку TCP/IP, створений для передачі даних за моделлю “pub-sub” (дослівно «видавець-абонент»). Протокол особливий тим, що не створює великого навантаження на мережу, простий у користуванні, передбачує нестабільності мережевого підключення (що корисно в умовах бездротового зв’язку), та відносно просто вбудовується в будь-які системи. Модель “pub-sub” дозволяє пристроям виходити на зв’язок будь-коли і публікувати повідомлення без прив’язки до певного формату передачі даних.

CoAP (Constrained Application Protocol) – це спеціальний мережевий протокол для пристроїв та мереж з певними обмеженнями (низької потужності чи з нестабільним підключенням) за моделлю точка-точка, які в контексті протоколу називаються вузлами (node). Ідейно CoAP унаслідований від HTTP, що і створює

можливість трансляції на HTTP та інтеграції у більш широкі мережі. Протокол працює поверх UDP. Підтримує групову адресацію (multicast), що є особливо важливим у міжмашинній комунікації.

AMQP (Advanced Message Queuing Protocol) – відкритий протокол прикладного рівня, який вводить свою модель обміну повідомленнями. Вона складається з точок обміну (exchange) та черг, у які точки обміну маршрутизують повідомлення, котрі зберігаються там, поки їх не забере клієнт. Співвідношення точок обміну, черг та клієнтів може бути будь-яким, а самі точки обміну можуть бути різних типів, що реалізують різну поведінку, що дозволяє будувати системи обміну повідомленнями найскладніших ієрархій. Повідомлення ніяк не інтерпретується сервером (що звільняє від формалізації переданої інформації).

DDS (Data Distribution Service) – протокол IoT, який забезпечує високоякісну комунікацію в системах Інтернету речей. Подібно до MQTT, DDS також працює за моделлю “pub-sub”. Являється стандартом обміну даних в міжмашинній взаємодії в реальному часі. Потребує проміжного програмного забезпечення, яке реалізує інтерфейси DDS. DDS керує процесом передачі: маршрутизацією повідомлень, десереалізацією даних, транспортуванням і керуванням потоками, QoS (повторними спробами). Будь-який вузол може бути як видавцем чи абонентом, так і обома одночасно. Фактично, це вважається першим відкритим міжнародним стандартом IoT для проміжного програмного забезпечення.

XMPP (eXtensible Messaging and Presence Protocol) – це протокол на основі XML. Завдяки своїй відкритості та простоті використання він широко використовується у програмах миттєвого зв'язку в Інтернеті. Оскільки режим зв'язку пристроїв Інтернету речей дуже схожий на режим обміну миттєвими повідомленнями в веб-додатках, протоколи обміну миттєвими повідомленнями, які зазвичай використовуються в Інтернеті спробували перекочувати у світ IoT, і найяскравішим представником такого протоколу є XMPP. У порівнянні з HTTP, XMPP більше підходить для системи Інтернету речей з точки зору

комунікаційного бізнес-процесу. Проте безпека протоколу HTTP та недоліки у споживанні обчислювальних ресурсів практично не усунуті.

WebSocket – протокол зв'язку, який з'явився нещодавно з появою HTML5. WebSocket працює поверх TCP як оновлення до стандартного з’єднання HTTP. Це забезпечує повнодуплексний зв’язок між клієнтом і сервером на основі повідомлень за допомогою одного сокета, який відкривається програмним інтерфейсом на обладнанні. Цей протокол є вдосконаленням протоколу HTTP для спілкування в реальному часі, керованого подіями в межах кількох пристроїв. Для встановлення серверу потрібна бібліотека WebSocket, а клієнт WebSocket і веб-браузер встановлені на клієнті або пристрої, який підтримує WebSocket. Проте, протокол унаслідував і проблеми свого попередника і не дуже добре себе показує в мережах, де існують обмеження потужності.

Як у випадку HTTP із веб-сокетами або XMPP, дизайн яких заснований на архітектурах Інтернет-застосунків, загальна проблема цих протоколів полягає в тому, що вони взагалі не можуть бути застосовані до різноманітності пристроїв IoT та не можуть бути пристосовані до вимог низького енергоспоживання. Хоча багато виробників застосовують їх у системах IoT, вони неминуче будуть неприйнятними по мірі витіснення їх більш оптимальними протоколами.

Коротка порівняльна характеристика

Таблиця 2.1

Протокол	MQTT	CoAP	XMPP	AMQP	DDS	WS
Транспортний стек	TCP	UDP	TCP	TCP	TCP/UDP	TCP
Реалізація публікацій	Так	Так	Так	Так	Так	Ні
Реалізація запитів	Ні	Так	Так	Ні	Ні	Так

Безпека	SSL	DTLS	SSL	SSL	SSL/DTLS	SSL
Повторне надсилання	Так	Так	Ні	Так	Так	Ні
Пристосованість до обмежених систем	Так	Відмінна	Ні	Так	Ні	Ні
Двійкове кодування	Так	Так	Так	Так	Так	Ні

DDS хоч і пропонує надзвичайно точний обмін даними у реальному часі, ці мілісекунди не гратимуть великої ролі у інформуванні про переміщення клієнтів і не вартуватимуть складного архітектурного підходу з огляду на специфіку даного протоколу, та складності його реалізації. Вибір коливається між AMQP та MQTT, проте в проекті немає великої кількості клієнтів (наразі – лише один, це хост із веб-сервером) та складної ієрархії, тому, на мій погляд, варто надати перевагу зручності інтеграції та розробки, яку пропонує MQTT.

2.3 Функціональні вимоги до серверного програмного забезпечення

У пункті 2.1 було описано нефункціональні вимоги – основні задачі, які повинні виконуватись продуктом, яким програмний комплекс повинен бути. У цьому пункті буде розроблено функціональні вимоги до програмного забезпечення, що буде виконувати ці завдання, а саме опис внутрішньої роботи системи та її процеси: зберігання даних, обробка і передача даних, модулі взаємодії із пристроями IoT та модулі взаємодії із іншими сервісами.

Отже, на даному етапі нам потрібен MQQT-брокер, MQQT-клієнт, база даних та веб-сервер.

Модель протоколу MQTT вимагає наявності сервера-брокера – центрального вузла мережі, який відповідає за отримання всіх повідомлень, їх фільтрацію, визначення отримувачів та доставку повідомлень. Сам брокер не залежить від платформ абонентів та видавців та працює тільки в межах MQTT протоколу, тому існує безліч його реалізацій, на різних мовах та для різних платформ. Більше того, можна зібрати його самому з нуля, або із частин існуючих модулів у вільному доступі. Оскільки специфікація протоколу стандартизована, для використання в проекті згодиться будь-яка реалізація, а сам хост повинен бути в мережі із використанням стеку TCP/IP.

Серверне програмне забезпечення за проєктом повинно приймати дані від пристроїв IoT, і оскільки ми обрали для цього протокол MQTT, обговорили необхідність брокера повідомлень, залишається їх отримати. Для цього нам потрібна сутність у виді MQTT-клієнта. Саме це поняття являється абстрактним, адже що завгодно, що читає опубліковані повідомлення, буде клієнтом. Звідси, основна вимога до клієнта – підтримка стека TCP/IP.

Вимоги до бази даних закріплені за високими стандартами практично на протязі всього часу існування реляційних баз даних та їх систем управління. Для забезпечення функціонування економічного об’єкта та виконання його завдань, база даних повинна відповідати комплексу вимог. Поточний проєкт – не виключення, тому зазначимо ці вимоги, щоб враховувати їх при виборі БД/СУБД для використання у прототипі.

Вимоги до бази даних:

1. Цілісність;
2. Масштабованість;
3. Можливість обробки інформації за запитами;
4. Багатократне використання даних;

5. Відмовостійкість і захищеність;

Деякі із них звучать неоднозначно, тому коротко розкриємо суть понять.

Цілісність – коректність даних та їх несуперечність. Для забезпечення коректності даних система управління базою даних повинна представляти механізми, що дозволять проектування моделей таким чином, щоб звіряти внесені дані на відповідність встановленим вимогам. Найпоширенішим прикладом буде встановлення обмеження на поле сутності, яке відповідає за посилання на батьківську сутність, аби воно обов'язково не було пустим. Таким чином вже запобігається ситуація, при якій може порушитись цілісність зв'язку. Або ж банально обмеження на поле, що слугує ідентифікатором, щоб уникнути ситуації неоднозначності записів – в такому разі доступ до запису було б втрачено, або запити на отримання інформації втратили б ідемпотентність.

Масштабованість – повинна існувати можливість розширити місткість, розподілити базу в разі зростання навантаження, або навіть відредагувати модель.

Обробка інформації за запитами – повинна існувати можливість створювати записи, редагувати їх, видаляти, та шукати й видавати інформацію за вказаними критеріями.

Багатократне використання даних – запити на використання інформації не повинні знищувати чи модифікувати дані, якщо це не був прямий запит на редагування інформації.

Відмовостійкість і захищеність – сукупність запобіжних заходів, що підтримуватимуть роботу бази даних у непередбачуваних сценаріях та захищатимуть від несанкціонованого доступу.

Практично всі нині існуючі системи управління базами даних задовольняють ці вимоги за замовчуванням чи з мінімальним втручанням в конфігурацію, основним завданням лишиться проектування самих моделей зберігання інформації.

Переважна більшість баз даних будується за реляційною моделлю, проте існують сучасні рішення, які не поступаються функціоналом та надійністю,

будуючись при цьому на якихось інших принципах зберігання і структуризації інформацію. Їх усіх відносять до категорії NoSQL баз даних, вони мають право на життя і активно розвиваються, набуваючи широкої підтримки особливо в спільнотах розробників. Підприємства, щоправда, особливо фінансовий сектор, свого майбутнього без реляційних баз не бачать, оскільки з 70-их років ці рішення закріпились як найнадійніші для таких масштабних сфер, де є місце точності і аналітиці великих об’ємів інформації, та й системи управління реляційними базами даних розвинули свій інструментарій в достатній мірі, щоб ніхто з постійних користувачів найближчим часом не задумувався про відмову від продукту в користь інших, принципіально нових підходів.

Оскільки інформаційна система для даної предметої моделює бізнес-процеси, традиційний реляційний підхід зарекомендував себе як надійний варіант реалізації БД. Для створення прототипу буде використано РСУБД, яка працює з SQL.

Оскільки всі існуючі реляційні системи управління базами даних ділять між собою один і той же концепт, будь-яке рішення задовольнить вимоги проекту. Щоправда, між ними є певні відмінності. Дамо коротку характеристику найпопулярнішим продуктам – Oracle DB, Microsoft SQL Server та MySQL, та зробимо висновки, який підійде для використання у проєкті.

Microsoft SQL Server.

Microsoft SQL Server має інтерфейси взаємодії із іншими продуктами Microsoft, але при цьому може запускатись і на операційних системах Linux. Має безкоштовну ліцензію для розробників, яка включає в себе широкий інструментарій. Має відносно простий механізм створення резервних копій та відновлення. Розрахований на розгортання як на машинах, так і на хмарних сервісах. Ліцензія для комерційного користування із додатковими функціями, щоправда, має високу вартість, а розширена скриптова мова запитів Transact SQL може здаватися складною для розробників, які звикли до інших програмних рішень.

Oracle DB

Oracle DB стала першою реляційною системою управління базами даних, яка випустила свій продукт для комерційного використання, із набором «розумних» інструментів, широкою користувацькою підтримкою і стала найпопулярнішою РСУБД у світі, набувши широкого застосування у великих корпораціях. Та це не лише маркетингові заслуги – продукт постійно розвивається за плановими графіками, а компінія дуже прискіпливо ставиться до інформаційної безпеки. Продукт постійно стає кращим і кращим у швидкодії, відмовостійкості та масштабованості, доступний до розгортання на практично всіх платформах, в тому числі і на хмарних сервісах. Щоправда, компанія заслужено користується своїм успіхом, і додатковий функціонал з ліцензій для комерційного застосування має свою вартість. Варто згадки і те, що розгортання і налаштування самої системи керування – не самий інтуїтивний процес та вимагає чималого обсягу знань специфіки власне цієї платформи.

MySql

MySql — це реляційна система управління базами даних з відкритим вихідним кодом , що значно відрізняється від Oracle або Microsoft SQL Server. Оскільки Інтернет став основним інструментом для ведення бізнесу, абсолютна необхідність створення веб-сайтів компаній призвела до того, що MySQL став одним із популярних рішень для баз даних, підключених до веб-сайтів. Безумовними плюсами є те, що продукт поширюється за вільною ліцензією, підтримується практично на кожній платформі, та має широку спільноту користувачів. Проте, для крупних компаній попередні рішення можуть бути більш привабливими через ширший інструментарій, особливо для складних бізнес-процесів. Також, MySql може демонструвати зниження швидкодії при великій кількості підключень, що також можна віднести до критичних недостатків.

Оскільки у MS SQL Server є досить зручний інструментарій для розробки на операційній системі Windows, це буде оптимальним вибором для прототипування.

Сформулюємо вимоги до веб-сервера. Це повинно бути програмне забезпечення, яке може формувати відповіді на запити клієнта протоколом

HTTP/HTTPS, яке підтримує налаштування маршрутизації із можливістю скерувати запити або пропонує середовище для виконання web-додатків. Із цією метою впорається будь-який популярний веб-сервер у вільному доступі. Принципових переваг один над одним вони не мають – лише специфіку розгортання і налаштування, що не вартує порівняльної характеристики. Для зручності розробки і відлагодження було обрано веб-сервер IIS-Express для локального хостингу.

2.4 Вимоги до операційного середовища

Для розгортання програмного забезпечення потрібне середовище, яке відповідає вимогам цього ПЗ.

Microsoft SQL Server вимагає:

- Мінімум 6 гігабайтів доступного простору на диску;
- Наявність мережевого підключення для виконання мережевих функцій;
- 1 гігабайт ОЗУ та більше по мірі розростання бази даних для забезпечення швидкодії;
- Процесор архітектури x64 з тактовою частотою не менше 1.4 ГГц;
- ОС Windows, Linux або Windows Server.

Для розгортання веб-додатку потрібне хоча б мінімальне середовище виконання, без набору інструментів для розробки – ASP.NET Core Runtime.

Вимоги для встановлення і функціонування **ASP.NET Core Runtime Environment**:

- ОС Windows 7 або пізніші, Windows Server 2008 або пізніші, Ubuntu 14 та пізніші, Red Hat Enterprise, або MAC OS 10.*;
- Процесор архітектури x86 або x64 із тактовою частотою від 1 ГГц;

- Мінімум 512 мегабайт ОЗУ;
- Встановлення модулів може зайняти до 3 гігабайт вільного простору на диску.

IIS Express 8.0 вимагає ОС Windows 8 або Windows Server 2012. IIS Express не конкретизує вимог до системи, оскільки припускається, що якщо система здатна запустити ОС Windows, то всі потреби для подальшої роботи IIS уже виконані.

Висновок

У цьому розділі було визначено функціональні та нефункціональні вимоги до розроблюваної системи, охарактеризовано робочий процес системи й побудовано діаграму потоків даних. Сформовано мінімальні необхідні вимоги до середовища, на якому буде розгортатись прототип системи.

Також було розглянуто найпоширеніші в IoT мережеві протоколи із короткою характеристикою для визначення найбільш оптимального для використання в контексті застосунку в користь протоколу MQTT.

Згідно зі сформованою специфікацією продукту планується подальша розробка програмних модулів для майбутнього прототипу системи.

РОЗДІЛ 3. РОЗРОБЛЕННЯ ПРОГРАМНИХ МОДУЛІВ ДЛЯ ПРОГРАМНО-ТЕХНОЛОГІЧНОГО РІШЕННЯ

3.1 Опис інструментарію для розробки

В процесі розробки програмних модулів були використані наступні інструменти:

.NET 6 – найновіша реалізація безкоштовної платформи розробки для створення практично всіх можливих видів додатків від дочірньої компанії Microsoft – .NET Foundation. Як платформу її виділяє спільне середовище виконання (CLR – Common Language Runtime, аналог JVM у Java) для коду мовами, що підтримуються .NET, вкрай розвинута модульність та широка спільнота. Підтримує JIT-компіляцію кодів CLR у машинні коди (компіляцію на ходу) або ж повну передчасну компіляцію у машинний код. Останні релізи стрімко рухались в сторону кросплатформеності та відкритого вихідного коду, що вкупі із функціональними перевагами робить платформу для розробки самим технологічним рішенням в даний момент.

C# та компілятор **Roslyn**. C# – об’єктно-орієнтована (в останніх версіях – мультипарадигменна) C-подібна мова програмування, розроблена компанією .NET Foundation для програмування на платформі .NET. C# характеризується статичною строгою типізацією, що вкупі з об’єктно-орієнтованим підходом гарантує максимальну безпеку типів, але мова пропонує і динамічні типи для спеціальних ситуацій використання. Ідейно наслідувалась від Java та C++ та пристосована для використання таким чином, щоб уникнути проблематичних моделей програмного забезпечення, ставши новим кроком у Enterprise-секторі. Roslyn – кодова назва набору компіляторів та аналізаторів, що переводять початковий код у IL-інструкції.

IL (intermediate language) – специфікація кодів, які машина CLR транслює у машинні інструкції і приводить їх до виконання у власному наборі потоків. За таким стисло описаним принципом програми мовою C# приводяться до виконання.

SQL Server Express – компактна реалізація Microsoft SQL Server для розробників, яку безкоштовно можна використовувати. Express-версія містить лише ядро СУБД. Для процесу розробки буде використано спеціальну версію Express – LocalDB, яка запускається в режимі користувача та інтегрується в середовище розробки Microsoft Visual Studio.

Entity Framework Core – реалізація технології доступу до даних O/RM (object/relational mapping). Цей інструмент дозволяє ефективно взаємодіяти із записами у БД за допомогою допоміжних об'єктів .NET. Принцип технологій OR/M полягає у представленні наборів даних із БД у вигляді сутностей, які зберігають свої властивості та зв'язки, пропонуючи програмні інтерфейси для вільної маніпуляції ними, та конвертації цих сутностей назад у набори скалярних величин, які записуються до бази даних чи вносять зміни в уже існуючі постійні об'єкти. У контексті мови C# для роботи із цими об'єктами спеціально розроблена інтегрована мова запитів LINQ (Language Integrated Query).

ASP.NET Core 6 – вільний кросплатформенний фреймворк для розробки веб-застосунків. Пропонує широку варіативність модулів для створення WebAPI, технологій для розробки як back-end, так і front-end, для побудови веб-додатків, які запускають C#-код на клієнті, що дозволяє уніфікувати платформу для написання логіки та неймовірну кількість інших новітніх інструментів. Пропонує оптимальні архітектурні рішення у шаблонах за замовчуванням, багату документацію, та легку й високошвидкісну модульну конвеєризацію HTTP-запитів.

MQTTnet – вільна високошвидкісна бібліотека класів .NET для комунікації із використанням протоколу MQTT. Містить в собі як компоненти для створення MQTT-клієнтів, так і для створення власних MQTT-брокерів. Широко використовує асинхронність, що дозволяє писати швидкий, оптимальний код. Підтримує стандарти безпеки TLS. Не має важких надбудов, мінімальна абстракція від

низькорівневої взаємодії MQTT, а представлений програмний інтерфейс уніфікований для всіх версій протоколу.

Mosquitto – MQTT-брокер з відкритим кодом, який підтримує версії протоколу 3.1 та 5.0, оптимізований для використання як на слабких пристроях з низьким енергоспоживанням, так і для роботи на повномасштабних серверних машинах. Розробник пропонує безкоштовний сервіс – хмарний брокер Eclipse Mosquitto, яким може скористатись будь-хто для тестування своїх MQTT взаємодій.

Microsoft Visual Studio – інтегроване середовище розробки від Microsoft, яке включає в себе дуже широкий ряд інструментів. Дозволяє вести розробку під множину платформ, в тому числі і .NET-застосунків. Із інструментів під час розробки активно використовувався SQL Server Object Explorer – менеджер баз даних з обмеженим функціоналом, аналог окремої утиліти від Microsoft “SQL Server Management Studio”. Для зручності і ефективності написання коду програми використовувались технології IntelliSense від Microsoft та ReSharper від JetBrains за учбовою ліцензією.

3.2 Архітектура системи

Для прототипування системи необхідні екземпляр бази даних, сервер-брокер повідомлень MQTT та веб-сервер із розгорнутим веб-застосунком.

В умовах розробки БД буде розміщено на тому ж хості, що і веб-сервер – локально. При подальшому розгортанні інформаційної системи, адреса сервера із системою управління базою даних вказується у файлі конфігурації проекта.

```
{
  "ConnectionStrings": {
    "AirportContext":
      "Server=(localdb)\\MSSQLLocalDB;Database=AirportDB;Trusted_Connection=True;MultipleActiveResultSets=true"
  },

```

Рис. 3.1 – Рядок підключення в файлі конфігурації appsettings.json

Вбудований постачальник конфігурацій ASP.NET Core зчитає дані з файлів конфігурації при збірці, та передасть ці дані у екземпляр класу "*ConfigurationManager*", який створюється при збірці проекту. Звернувшись до менеджера конфігурацій, можна отримати потрібне значення – в даному випадку виконується пошук за ключем "*AirportContext*" для отримання рядку підключення. Це лише один із способів ін'єкції параметрів, всі вони існують для того, щоб при розгортанні додатку конфігурація проводилась максимально поза доменом самого додатку і лише один раз.

Сервером у ролі брокера для повідомлень MQTT виступить публічно доступний хост Eclipse Mosquitto від компанії Cedalo. Існує також опція запустити брокер повідомлень на локальному хості, але обрано було попередній варіант з наступних причин. Зовнішній брокер краще моделює розподілену систему. На локальному хості моделювання обміну даними доводилось би робити за допомогою імітації пакетів на програмному забезпеченні MQTT-брокера, а якщо й дійсно надсилати пакети з локальних клієнтів-видавців, налаштувавши на хості loopback-з'єднання (маршрутизацію хоста до самого себе) – маршрут слідування даних буде максимально коротким, не повністю відображатиме модель системи.

Архітектуру змодельованої системи можна побачити на наступному рисунку 3.2. Весь функціонал будується навколо веб-додатку ASP.NET Core, який в даному випадку являє собою центр інформаційної інфраструктури. Веб-сервером виступає IIS, який розміщує на собі веб-додаток. Існує два підходи для розміщення веб-додатків на IIS: модель всередині процесу та модель ззовні процесу. Їх схематично зображено на рисунку 3.3:

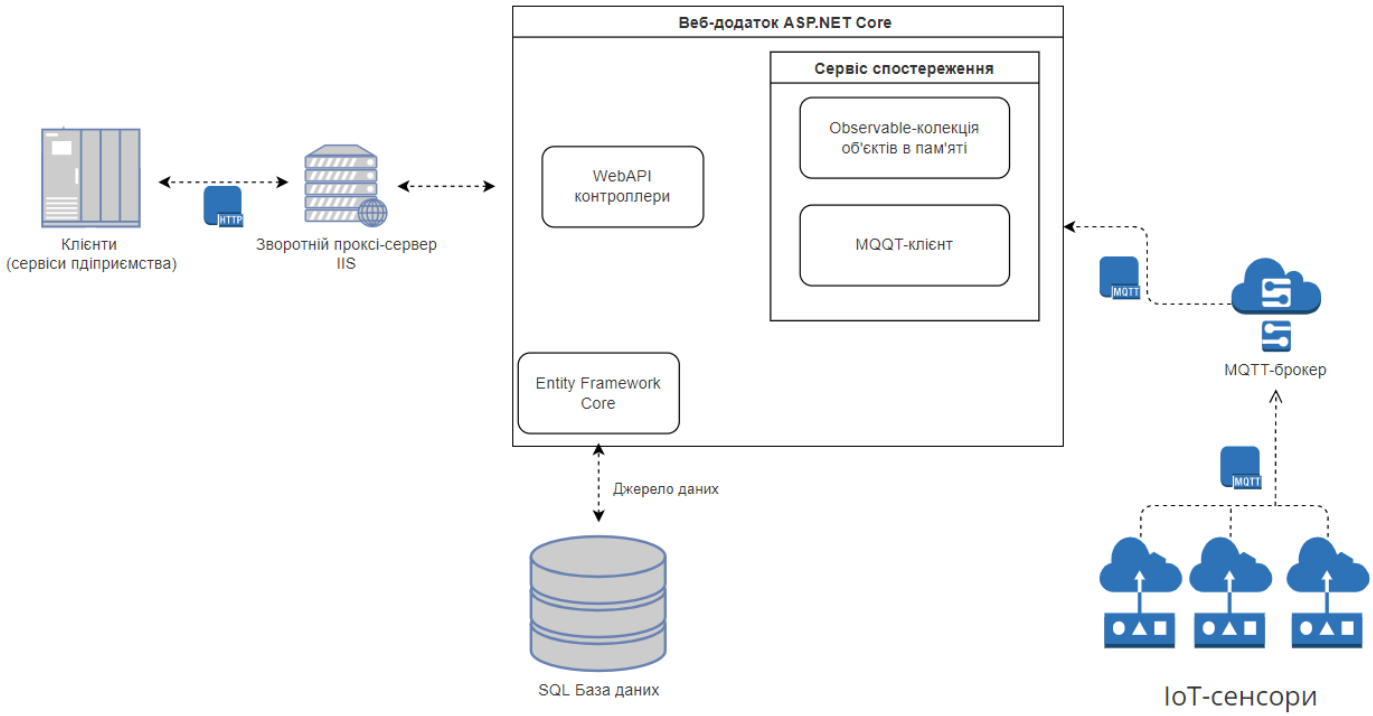

Рис. 3.2 – Архітектура розробленого прототипу інформаційної системи

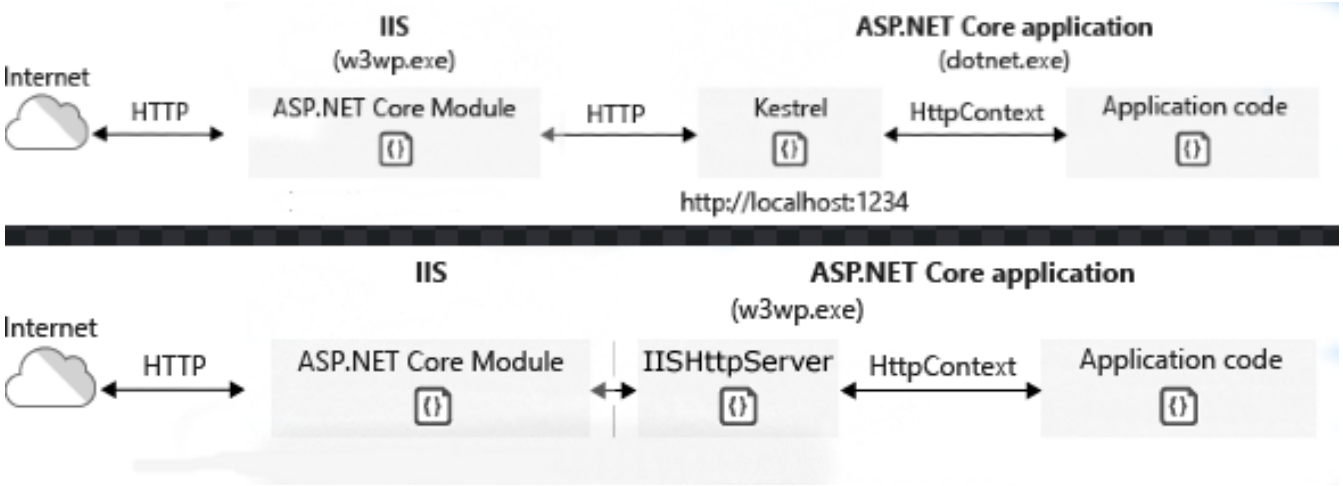

Рис. 3.3 – Моделі розміщення .NET додатків на веб-сервері. Перша схема (зверху) – out-of-process модель, друга – in-process модель.

При розміщенні поза процесом веб-сервера, веб-запити, що надходять до хоста, обробляються HTTP-драйвером системи, скеровуються до IIS (або іншого веб-сервера). В свою чергу, завантажений до IIS модуль перенаправляє запит до

вбудованого .NET Core веб-сервера Kestrel, який є початковою точкою конвеєру запитів у домені додатку. Він перетворює HTTP-запит у об’єкт класу *HttpContext*, що несе всі інформацію про запит та передає його у конвеєр для подальшої обробки додатком.

При розміщенні ж додатку у просторі процесу веб-сервера, HTTP-запит після скерування HTTP-драйвером також потрапляє до сервера IIS, але не перенаправляється до сервера Kestrel – він в цьому випадку не бере участі в конвеєризації запитів і не розгортається зовсім. Натомість, разом із ASP.NET Core модулем у простір процесу IIS завантажується модуль, який перетворює нативні HTTP-запити у керовані (машиною CLR) об’єкти *HttpContext*, відразу стаючи початком конвеєру обробки запитів і передає їх для обробки логікою веб-додатку.

За замовчуванням, для IIS налаштовано розгортання за моделлю in-process, оскільки це не вимагає налаштування loopback-адаптера для перенаправлення запитів, яке сильно впливає на потенційну пропускну здібність HTTP-запитів. В разі використання іншого веб-серверу, окрім IIS, за замовчуванням буде виконуватись розміщення поза процесом веб-сервера.

В подальшому прототипі буде використано out-of-process модель розміщення для уникнення залежності від конкретної платформи в майбутньому.

У ролі MQTT-клієнтів, які публікують повідомлення, виступатиме програмний mock-up (макет), який містить в собі «будівельника» MQTT-повідомлень, що надсилатиме дані, які взаємодіятимуть із бізнес-логікою веб-додатка та базою даних. Клієнт-видавець реалізовано у вигляді desktop-додатку із інтерфейсом командного рядка. Він використовує бібліотеку MQTTnet, та публікує повідомлення на тестовий брокер Mosquitto.

Сам веб-додаток має монолітну архітектуру – в одній assembly (збірці) реалізовано і презентаційний функціонал (web-контроллери), і імплементовано сервіси, на які цей функціонал спирається. Збірка у .NET – основна одиниця розгортання, контролю версій і повторного використання, це колекція типів і ресурсів, які побудовані для спільної роботи, і є мінімальною інфраструктурною

одиницею, яка може бути завантажена до виконання у машину CLR. Водночас, збірка не має прямої залежності із структурою файлів початкового коду, з яких вона побудована. Це дозволяє навіть одній мінімальній інфраструктурній одиниці бути максимально модульною і доступною для роботи в команді та для можливого подальшого рефакторингу. Детальніше це буде розглянуто у пункті 3.3.

3.3 Структура проекту

Основні компоненти системи було розроблено у середовищі Microsoft Visual Studio та об’єднано у одне рішення з метою зручного відлагодження. Панель оглядача рішень представляє проекти і їх файлові ієрархії у легкому для сприйняття вигляді.

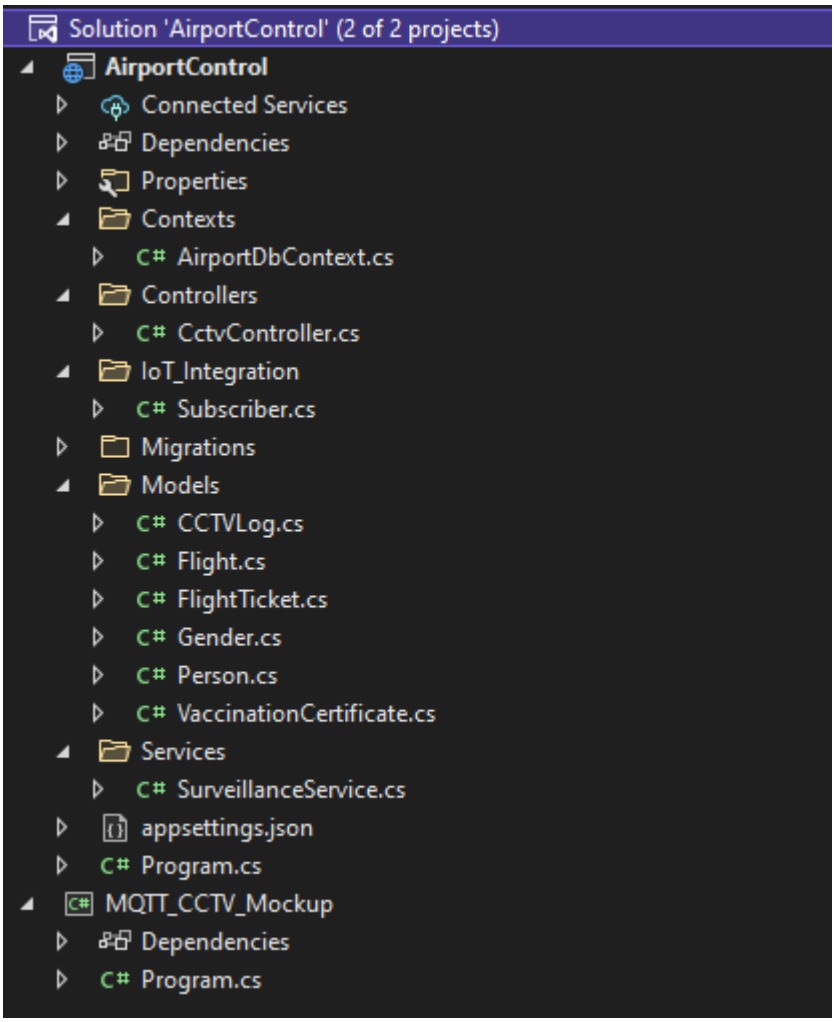

Рис. 3.4 – Ієрархічне дерево рішення

Конвенція мови C# рекомендує групувати елементи з однорідним функціоналом чи однією смисловою приналежністю у простори імен, а у файловій системі організовувати їх, як каталоги із вкладеними в них файлами початкового коду. Цю модель дотримувати не обов'язково – компілятор не враховує фізичне представлення початкового коду. Відхилення від цього підходу ніяких переваг при розробці не дає, тому файли організовані за конвенцією.

Простори імен існують для розбиття навіть монолітних проектів на логічні модульні частини, як для внутрішньої організації й розробки, так і для доступу інших програм до компонентів, які були їм надані для використання. Взаємодія з іншими збірками відбувається за допомогою директиви *using*. Вона включає вказану область типів до контексту поточного простору імен, а компонент при використанні цих типів стає залежним від компоненту, який ці типи надає для використання. В контексті додатку, ці компоненти стають зв'язними, тому що один компонент має залежності від іншого. Виокремлюють також типи зв'язності – сильна (тісна) та слабка (вільна). При явному використанні типів, описаних в іншій збірці, формується сильна зв'язність. Хорошим підходом для масштабування систем є уникання сильної зв'язності там, де це можливо – так з'явилось поняття слабкої зв'язності разом із механізмами визначення типів під час виконання програми, а не в час компіляції. Це відкрило можливість розробляти повністю незалежні один від одного компоненти системи, а головне – тестувати їх, адже якщо тип, з яким взаємодіє компонент, ще не готовий, а його поведінка вже передбачена за планами розробки – цей тип можна замінити макетом з аналогічною поведінкою, не редагуючи код, який підлягає тестуванню.

Проект MQQT_CCTV_Mockup має лише один простір імен для застосунку, оскільки його призначення виключно тестове, а функціонал згруповано в top-level-statements. Робота в ньому ведеться в глобальному просторі імен додатку, а інструкції в ньому інтерпретуються як точка входу в програму.

У проекті AirportControl, який формує ASP.NET Core додаток, на даному етапі виділено такі простори імен:

- AirportControl.Contexts – тут описані компоненти, що налагоджують OR/M систему;
- AirportControl.Controllers – веб-контроллери, що формують відповіді на запити;
- AirportControl.IoT_Integration – компоненти, що відповідають за налаштування асинхронного MQTT-клієнта, прийом повідомлень від MQTT-брокера та їх передачу у інші компоненти додатку;
- AirportControl.Migrations – допоміжні класи, згенеровані фреймворком OR/M для модифікації моделей та їх резервного копіювання за допомогою снапшотів (знімків поточного стану);
- AirportControl.Models – простір для опису класів, які моделюють сутності, що безпосередньо беруть участь у бізнес-процесах;
- AirportControl.Services – простір для типів, які містять логіку для завдань, що повинен виконувати додаток.

Файл Program.cs використовує підхід top-level-statements. Він знаходиться в глобальному просторі імен додатку, а інструкції в ньому інтерпретуються як точка входу в програму. Там відбувається налаштування допоміжних класів, реєстрація типів у dependency injection-контейнери та конфігурація HTTP-конвеєра.

Висновок

У цьому розділі було наведено короткі відомості про використані інструменти для розробки та окреслено архітектуру інформаційної системи. Розроблено схему архітектури, окреслено поняття логічних модулів, описано призначення кожного

модуля на прикладі змодельованого веб-додатку. Проаналізовано способи розгортання веб-додатку. Розуміння архітектури системи і її призначення – необхідна умова оптимального підходу розробки програмного продукту.

РОЗДІЛ 4. ПРОТОТИПУВАННЯ. ТЕСТУВАННЯ ПРОТОТИПУ

4.1 Прототипування

Прототипування – ключовий етап моделювання цифрового продукту, під час якого створюється демонстраційна версія системи із реалізацією базового функціоналу, що дозволяє побачити запропоновані шляхи імплементації та роботу компонентів системи в цілому. Це необхідно для подальшого аналізу оптимальності обраних шляхів реалізації і пошуку недоліків архітектури.

Прототип може не реалізовувати всю передбачену логіку в повній мірі, може допускати необроблені виключні ситуації, оскільки основне завдання прототипу – продемонструвати підходи до вирішення поставлених завдань та отримати зворотній зв’язок від замовника.

Із переваг такого підходу можна виділити участь замовника на доволі ранньому етапі життєвого циклу ПЗ, потенційна економія ресурсів за рахунок раннього виявлення дефектів чи потреб в зміні використовуваних технологій. Недоліком є те, що всі нюанси таким чином виявити неможливо через ймовірну розбіжність у програмній логіці та використовуваних технологіях.

Виокремлюється два види прототипування програмного забезпечення: швидке прототипування та еволюційне прототипування. Швидке прототипування означає створення макета, який дуже поверхнево відображає сутність продукту, та не буде використовуватись як його частина. Найчастіше використовується для макетів користувацьких інтерфейсів, оскільки вони вимагають мінімального написання

декларативного коду, або ж зовсім його не вимагають завдяки стороннім інструментам. Цей підхід явно буде нерентабельним у тому випадку, якщо для демонстрації функціоналу системи необхідно реалізувати багатокомпонентну систему та розробляти складну архітектуру. Тому у роботі було обрано інший вид прототипування – еволюційний. При еволюційному прототипуванні за мету ставиться розробка моделі системи, яка стане основою для майбутнього фінального продукту. Відповідно, обираються доцільні для production-середовища технології, а вкладення ресурсів у розробку архітектури системи не буде марним кроком. Із таким підходом доцільно застосовувати ітераційну модель розробки, оскільки модель, як правило, спроектована так, щоб на ній було доступно поступово нарощувати функціонал.

4.2 Тестування створеної моделі

Після розробки прототипу інформаційної системи із певним реалізованим функціоналом, у цьому пункті буде проведено тестування моделі з фокусом на основні поставлені для неї завдання: взаємодія сервісів із технологіями Інтернету речей (в даному випадку – із MQTT-брокером та моделями клієнтів-видавців), зберігання бізнес-сутностей у базі даних та взаємодія додатку із нею, та прикладні програмні інтерфейси для подальшої інтеграції у загальну інфраструктуру підприємства (в контексті предметної області – аеропорту). Для оптимального обсягу матеріалів роботи тестування CRUD (create-read-update-delete)-операцій над бізнес-сутностями не проводитиметься, оскільки це атомарні елементарні операції, реалізовані на рівні фреймворку OR/M, а увага зосереджена на логіці, що напряму пов’язана із предметом дослідження дипломного проєкту.

Тестування розпочинається із підготовки вхідних даних. У базу даних внесено дві тестові бізнес-сутності.

Id	FirstName	MiddleName	LastName	BirthDate	Gender
1	Roman	*NULL*	Markulych	13/06/2000 00:00:00	1
3	Arisa	*NULL*	Gleipnir	15/09/1998 00:00:00	2

Рис. 4.1 – Вміст таблиці dbo.Person

За спроектованою схемою БД, клієнт може підвантажити до свого профілю сертифікати про вакцинацію від вірусу COVID-19. Внесемо відповідні записи до таблиці, яка зберігає дані про сертифікати вакцинації:

ID	OwnerId	VaccineType	VaccinationDate	ValidUntil
1	1	Coronavac	15/08/2021 00:00:00	15/08/2022 00:00:00
2	3	Coronavac	11/01/2021 00:00:00	11/01/2022 00:00:00

Рис. 4.2 – Вміст таблиці dbo.VaccinationCertificates

Для створення контрольних сценаріїв тестових ситуацій, перший запис сертифікату на момент тестування ще залишається діючим, а другий запис внесено для моделювання випадку закінчення терміну дії раніше завантаженого клієнтом сертифікату. Діючий сертифікат належить клієнту із ідентифікатором 1, сертифікат із вичерпаним терміном дії належить клієнту з ідентифікатором 3.

Набір сутностей у БД для мінімальної взаємодії готовий. Далі проводиться локальне розгортання системи. Виконується збірка і експорт готового до публікації проекту до вказаного каталогу та редагування файлів конфігурації (за потреби).

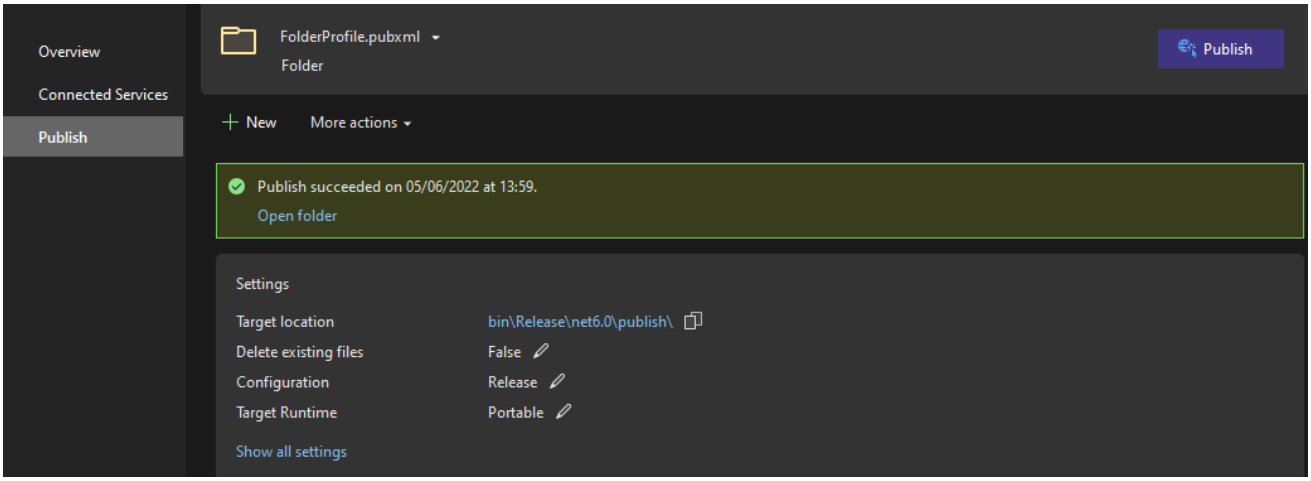

Рис. 4.3 – Графічний інтерфейс, аналогічний процедурі “dotnet publish” в CLI

В результаті отримано згенеровані бінарні файли програми разом із набором локалей, ресурсів, та збірок, від яких залежить створений додаток – усе готово до розгортання. Запуск і розміщення веб-хоста у ASP.NET Core 6.0 відбувається з виконуваного файлу з інтерфейсом командного рядка з відповідною до проекту назвою. У даному випадку це файл AirportControl.exe.

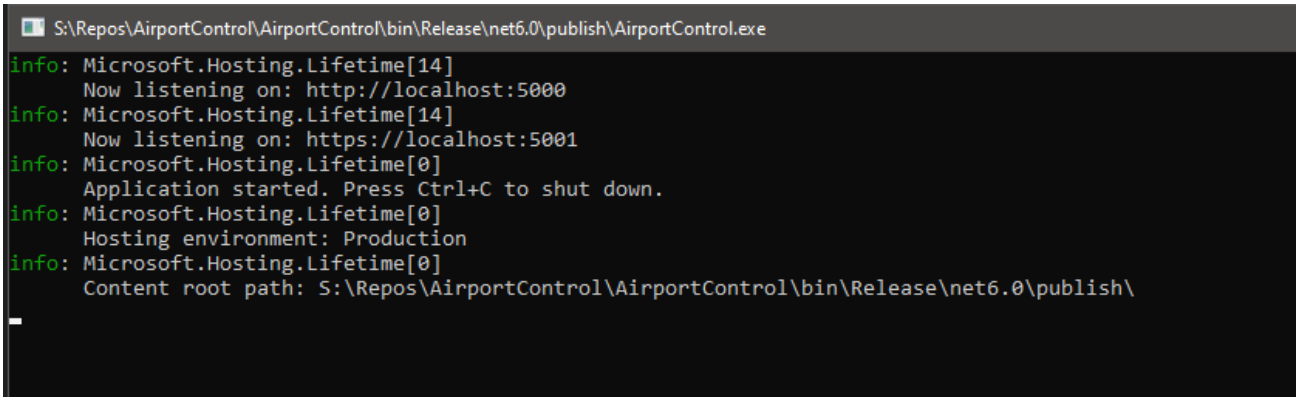

Рис. 4.4 – Запуск веб-сервера із розміщеним на ньому веб-додатком

У першу чергу потрібно перевірити доступність веб-сервера та наявність зареєстрованого контроллера у додатку. Посилаючись на відкритий веб-сервером порт для прослуховування HTTP-запитів, буде сформовано перший тестовий запит. Для надсилання елементарних HTTP-запитів GET можна скористатись адресним рядком будь-якого веб-браузера, але тестування веб-сервісів оптимальніше проводити за допомогою спеціалізованих інструментів. В цьому випадку тестування проводиться інструментом Postman.

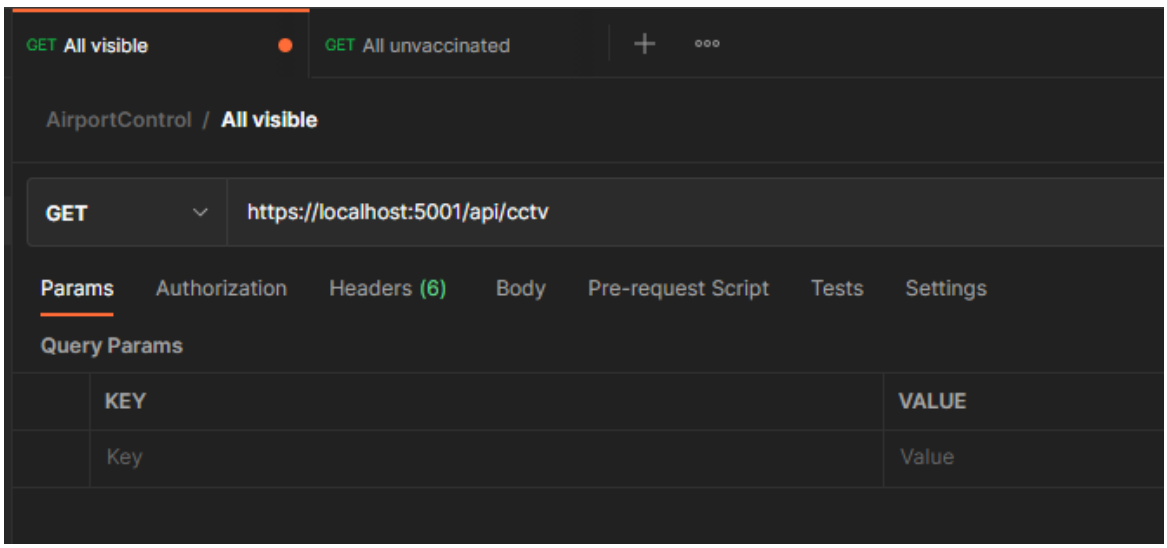

Рис. 4.5 – Формування запиту

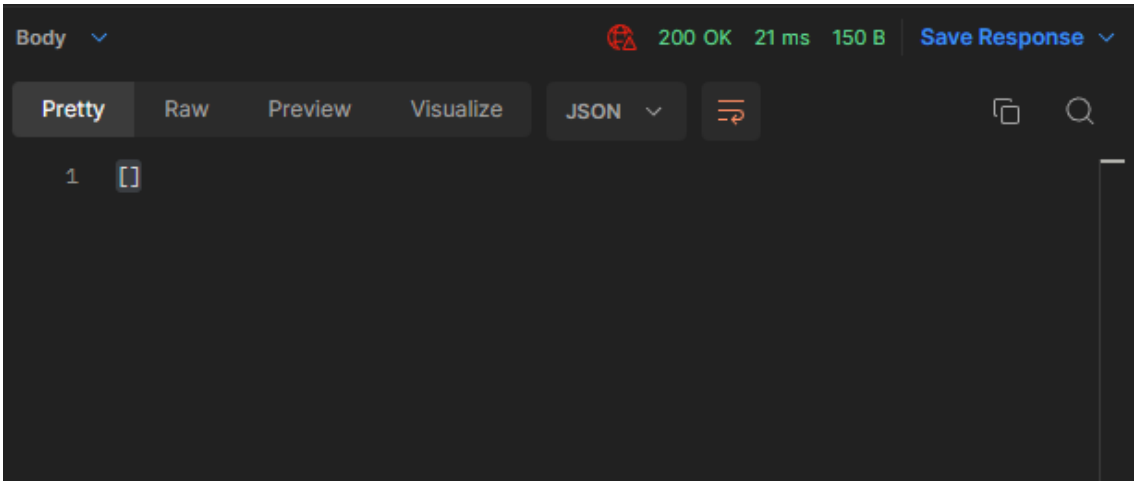

Рис. 4.6 – Відповідь сервера

На рисунку 4.5 було сформовано GET запит до кінцевої точки веб-сервісу за маршрутом */api/cctv*. На рисунку 4.6 зображено відповідь сервера зі статус-кодом 200 (Успіх) та символами “[]” у тілі відповіді. Ця поведінка є очікуваною, оскільки це – порожній json-об’єкт. У цьому форматі даний метод контролера повертає масив сутностей, які в момент запиту знаходяться у колекції *PresentPeople*, яка при чистому запуску додатку ініціалізується порожнім списком. Ця колекція відображає стан поточних об’єктів, що знаходяться в полях зору IoT-сенсорів (в контексті – високотехнологічних камер, що розпізнають особистості), взаємодія з нею відбувається через модулі інтеграції з IoT, що підтримують зв’язок із MQTT-брокером.

Для оновлення даних в цій колекції потрібно надіслати повідомлення MQTT-брокеру. Для моделювання взаємодії від MQTT-клієнта «видавця» буде надіслано повідомлення брокеру з інформацією, що розпізнано людину, яка в системі розпізнавання внесена з ідентифікатором 1 (із припущенням, що ідентифікатори людей в базах розпізнавання відповідають номеру ідентифікатора клієнта в БД бізнес-процесів, якщо ця людина є зареєстрованим клієнтом), яка ввійшла в один із секторів аеропорта із умовним позначенням «A2», та час, в який ця подія була зафіксована. Ці дані у форматі json кодуються у послідовність байтів, входять в сформоване повідомлення як корисне навантаження, та відправляються брокеру публікуватись у тему, яка прослуховується клієнтом, інтегрованим у веб-додаток.

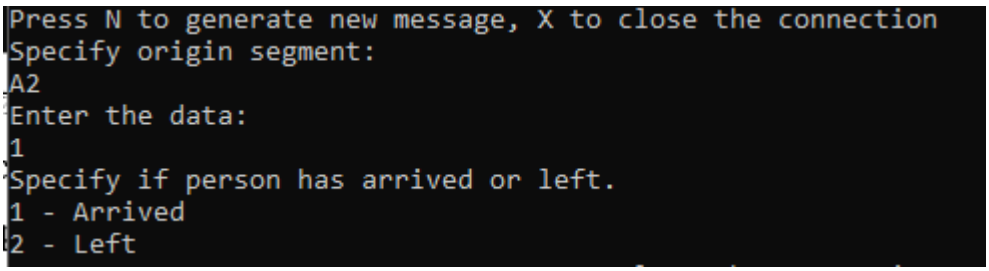

Рис. 4.7 – Інтерфейс побудови json для публікації повідомлень

Після того, як повідомлення було надіслано брокеру, список людей на веб-сервері повинен змінитись. Виконання повторного запиту дало наступний результат.

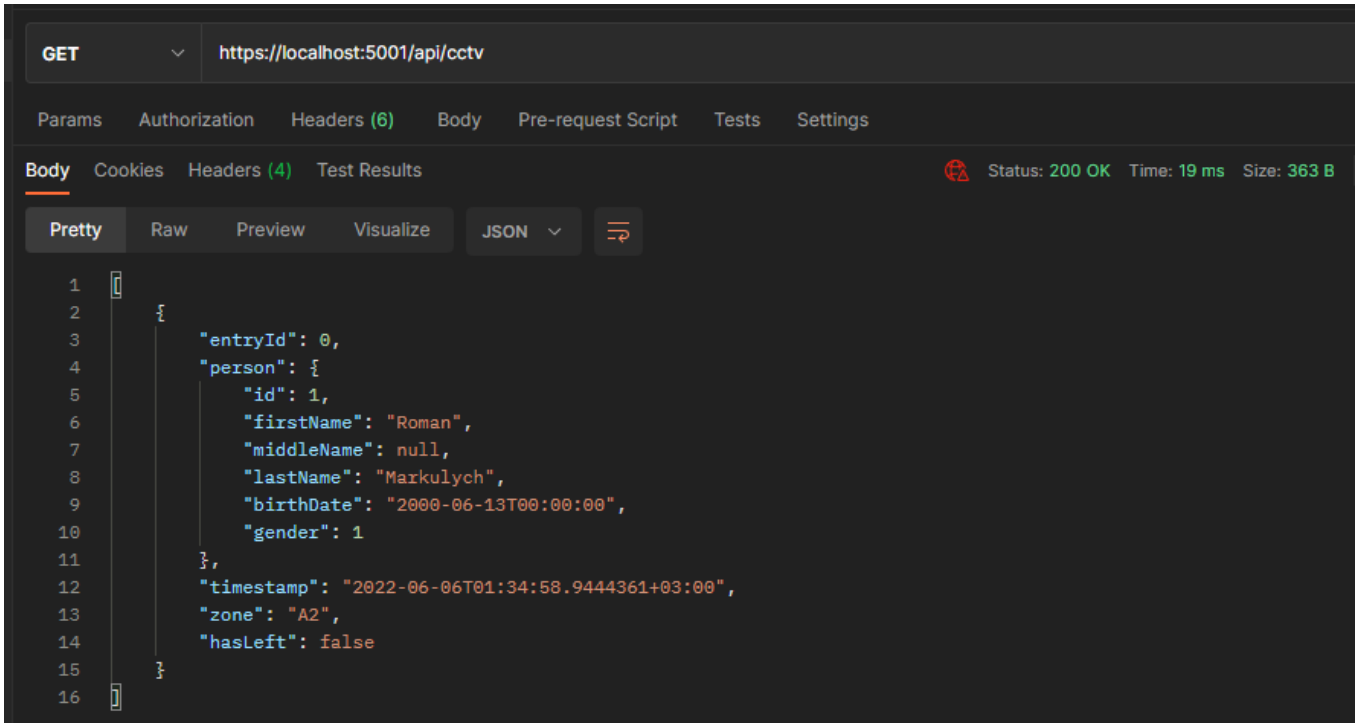

Рис. 4.8 – Відповідь на повторний GET-запит до контроллера *cctv*

В тілі відповіді – json-об’єкт, що містить один елемент. IoT-пристрій, який надсилав повідомлення, не має доступу до БД бізнес-інфраструктури, і здатний лише за збігом зовнішніх даних людини із власною базою видавати ідентифікатор людини, який не несе сам по собі ніякої корисної інформації. Проте видно, що в полях присутня детальна інформація про особу. Це робота сервісу веб-додатка – перед внесенням в колекцію поточно видимих об’єктів, він намагається знайти зареєстрованого у системі клієнта з таким ідентифікатором, що прийшов у

повідомленні, та обвішати його додатковою інформацією про поточну бізнес-сутність. Наприклад, можна дізнатися, чи є у полі зору люди без дійсного сертифікату вакцинації. Для цього було надіслано пакет із повідомленням, що в поле зору сенсорів попав клієнт із ідентифікатором 3. Після цього виконано запит GET до кінцевої точки за маршрутом /api/unvaxed:

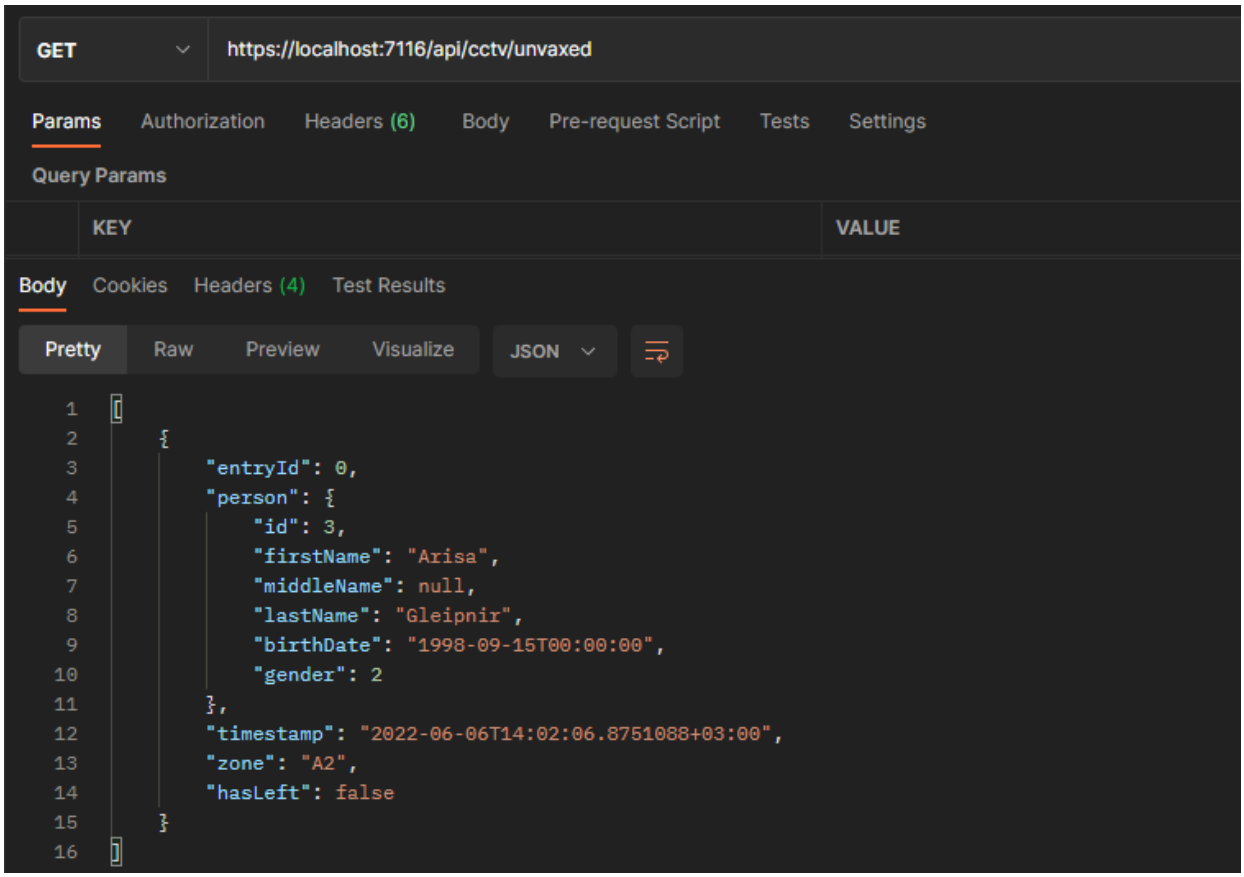

Рис. 4.9 – Відповідь на GET-запит невакцинованих людей серед присутніх

Цю логіку також обробляє сервіс, шукаючи для впізнаних людей закріплений за ними діючий на даний момент сертифікат. Сервіс також може оперувати незареєстрованими в системі людьми, якщо IoT-пристрої видаватимуть тимчасові ідентифікатори новим клієнтам. Ці сутності будуть розцінюватись як невакциновані.

Висновок

У останньому розділі роботи було створено прототип системи та проведено тестування його базового функціоналу. Прототип підпадає під критерії еволюційного типу, являючи собою основу для розвитку системи. В результаті було реалізовано основні функціональні та нефункціональні вимоги із використанням описаного у попередньому розділі інструментарію.

Проведено тестування за допомогою тестових даних та створених тестових ситуацій для демонстрації основного функціоналу. Його перебіг повністю успішний, оскільки всі результати були очікуваними для сформованих вхідних даних.

ВИСНОВКИ

Для досягнення поставленої мети роботи було проведено аналіз предметної області із оглядом теорії керування в обслуговуванні клієнтів, охарактеризовано технології Інтернету речей у реаліях сьогодення та їх роль у інформаційних інфраструктурах підприємств, та можливості систем, де ці технології інтегровані у процеси керування. Було наведено ідейні приклади існуючих систем керування аеропортів із інтеграцією в них IoT, проте всі вони імплементовані з залученням пропрієтарних технологій.

Метою була розробка власної системи, що не має пропрієтарних залежностей та дозволяє вільно вдосконалювати підприємства, що ведуть комерційну діяльність із обслуговуванням клієнтів.

Розроблений прототип відповідає визначеним функціональним та нефункціональним вимогам, а завдяки підходу еволюційного прототипування являє собою платформу для подальшого розвитку і нарощування функціоналу, та може класифікуватись як MVP (minimum viable product) – тобто працювати у production-середовищі. Додаток налагоджує зв’язок IoT-пристроїв з бізнес-інфраструктурою і відкриває можливості по розвитку нових бізнес-процесів та оптимізації вже існуючих.

Представлена робота дозволяє будь-кому використовувати цю систему у своїх власних проектах, або впроваджувати це рішенння у свої інформаційні інфраструктури підприємств та розвивати їх.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Designing the Internet of Things» / Adrian McEwen та Hakim Cassimally. ISBN-13 : 978-1118430620
2. «Нова ера розумних аеропортів» [Електронний ресурс] Режим доступу до ресурсу: https://www.kelltontech.com/kellton-tech-blog/iot-ushering-new-era-smart-airports
3. «Проблеми безпеки пристроїв IoT» [Електронний ресурс] Режим доступу до ресурсу: https://www.eurofins-cybersecurity.com/news/security-problems-iot-devices/
4. Ресурс технічних документацій продуктів Microsoft «MSDN» [Електронний ресурс] Режим доступу до ресурсу: https://docs.microsoft.com/en-us/
5. Головний ресурс із відомостями та документацією протоколу MQTT [Електронний ресурс] Режим доступу до ресурсу: https://mqtt.org/
6. «CLR via C#» / Jeffrey Richter. ISBN: 9780735621633
7. «Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications» / Adam Freeman. ISBN: 9781484279564