НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ І.Р. Пархомей
(підпис)
“___”_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»

на тему: Системи збору даних та прийняття рішень з пристроїв моніторингу в системі Smart City.

_____
_____

Виконав: студент другого курсу, групи ІК-61м
(шифр групи)

Свічинський Владислав Валерійович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н., доцент Тимошин Ю. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Рівень вищої освіти – другий (магістерський)

Спеціальність 151 «Автоматизація та комп'ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ І.Р. Пархомей
(підпис)
«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Свічинському Владиславу Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Системи збору даних та прийняття рішень з пристроїв моніторингу в системі Smart City»_____ ,
науковий керівник дисертації ___доцент, к.т.н., доцент Тимошин Ю. А.____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «12» березня 2018 р. №885-с

2. Термін подання студентом дисертації _____

3. Об'єкт дослідження – базовий компонент (контролер) розумного дому.

4. Предмет дослідження – компоненти програмного та апаратного забезпечення, які, інтегруючись, виконують функції моніторингу в системі Smart City.

5. Перелік завдань, які потрібно розробити – Аналіз та вибір апаратного забезпечення для запуску ПЗ; Побудова архітектури. Побудова програмного забезпечення; Інтеграція компонентів у єдину систему.

6. Орієнтовний перелік ілюстративного матеріалу – шість плакатів

7. Орієнтовний перелік публікацій – публікація на тему «Як захистити свій IoT продукт від хакерів»

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз предметної області	13.03.2018 р.	
2	Постановка задачі	15.03.2018 р.	
3	Аналіз інформаційного забезпечення	20.03.2018 р.	
4	Аналіз алгоритмічного забезпечення	25.03.2018 р.	
5	Розробка алгоритмічного забезпечення	15.04.2018 р.	
6	Розробка програмного забезпечення	01.05.2018 р.	
7	Маркетинговий аналіз стартап-проекту	10.05.2018 р.	
8	Висновки	15.05.2018 р.	

Студент _____ Свічинський В. В.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Тимошин Ю. А.
(підпис) (ініціали, прізвище)

Анотація

Магістерська дисертація обсягом 73 сторінки, містить в собі 11 ілюстрацій, 20 таблицю, 19 джерел.

Акутуальність магістерської диссертації полягає у тому, що на сьогодні не існує універсальної та повної системи моніторингу за містом для системи Smart City на ринку. У кожного міста є свої унікальні вимоги, і ніхто не може запропонувати універсальний продукт, який відповідатиме потребам кожного міста.

Метою магістерської дисертації є розробка універсальної, гнучкої та розширюваної системи моніторингу для Smart City.

Об'єктом дослідження є архітектура системи моніторингу.

Предметом — технології та засоби які можна використовувати для побудови системи моніторингу в Smart City.

За метод дослідження було обрано *аналітичний,* так як отримати вибірку даних наданих стороннім сервісом без використання API можна тільки завдяки аналізу.

Відсутність універсальної та повної системи моніторингу пояснює *наукову новизну* роботи.

Ключові слова: *архітектура*, *система моніторингу, IoT, Smart City.*

Annotation

Master dissertation in volume of 73 pages, contains 11 illustrations, 20 table, 19 sources.

The actuality of the master's thesis is that today there is no universal and complete system for monitoring the city for the Smart City system in the market. Each city has its own unique requirements, and nobody can offer a universal product that meets the needs of each city.

The purpose of the master's thesis is to develop a universal, flexible and extensible monitoring system for Smart City.

The object of research is the architecture of the monitoring system.

The subject is technologies and tools that can be used to build a monitoring system in Smart City.

Analytical method was chosen for the research method, as it is possible to obtain a sample of data provided by a third-party service without using the API only through analysis.

The lack of a universal and complete monitoring system explains the scientific novelty of work.

Keywords: *architecture, monitoring system, IoT, Smart City.*

ЗМІСТ

ВСТУП 9

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10

1.1. IoT системи 10

1.2. Система Smart City 13

1.3 Прогресивні веб додатки 19

1.4 Реактивне програмування 21

1.5. Системи монітиронгу в Smart City 22

1.5.1. Моніторинг якості повітря 24

1.5.2. Моніторинг навколишнього рівня шуму 24

1.5.3. Моніторинг води та стічних вод 25

1.5.4 Регулювання освітлення 25

1.5.5 Оптимізація стоянки для автомобілістів 25

1.5.6 Рух транспорту 25

Висновки до розділу 1 26

РОЗДІЛ 2. РОЗРОБКА АРХІТЕКТУРИ РОЗТАШУВАННЯ ПРИСТРОЇВ 27

2.1. Розробка архітектури системи моніторингу в системі Smart City 27

2.2 Хмарна платформа 30

2.2.1 Cloud IoT Core 30

2.2.2 IBM Watson 31

Висновки до розділу 2 36

РОЗДІЛ 3. Розробка програмного забезпечення для системи моніторингу в Smart City 37

3.1 Розробка архітектури програмного забезпечення 37

3.2 Обрані технології 38

3.2.1 Angular 38

3.2.2 NodeJS та Express 39
3.2.3 RxJs 39
3.2.4 Couch DB 41
3.2.5 JSON 43
3.2.6 Electron 43
3.3 Додаток Core 44
3.4 Додаток Client 47
3.5 Додаток Admin 52
3.6 Технічні вимоги для додатків 52
Висновки до розділу 3 54
РОЗДІЛ 4. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ 55
4.1. Опис ідеї проекту 55
4.2. Технологічний аудит ідеї проекту 57
4.3. Аналіз ринкових можливостей запуску statup проекту 57
4.4. Розроблення ринкової стратегії проекту 61
4.5. Розроблення маркетингової програми statup проекту 63
Висновок до розділу 4 66
ВИСНОВКИ 67
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 68
ДОДАТКИ 70

ПЕРЕЛІК СКОРОЧЕНЬ

IoT – інтернет речей

ICT – інформаційні та комунікаційні технології

CSEMS – системи моніторингу емісій безперервного кола

COMS – системи моніторингу безперервної непрозорості

IAQM – сиситема моніторингу якості повітря

CAN – мережа контролерів

CANopen - відкритий мережевий протокол верхнього рівня для підключення пристроїв, що вбудовуються в бортових транспортних і промислових мережах

PnP(Plug and Play) – технологія, призначена для швидкого визначення та конфігурації пристроїв в комп'ютері і інших технічних пристроях

MVC – архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення (англ. Model-view-controller)

CRUD(Create, Read, Update, Delete) – базові операції для роботи з базою

Node.js– це JavaScript–оточення побудоване на JavaScript та Chrome V8

Angular – JavaScript фрейворк для створення клиентських додатків

RxJs – JavaScript бібліотека для реактивного асинхронного програмування

ВСТУП

Сьогодні не існує універсальної та повної системи Smart City на ринку. У кожного міста є свої унікальні вимоги, і ніхто не може запропонувати універсальний продукт, який відповідатиме потребам кожного міста. Міста - це живі організми, які постійно зростають і потреби яких завжди змінюються. Через це постійно розробляється та оновлюється програмне забезпечення для моніторингу та управління. Майбутні міста в кінцевому підсумку використовуватимуть ще нерозроблені продукти та стандарти, які повинні бути сумісними та мати можливість обмінюватися інформацією один з одним. Спільні дані включатимуть світлофор, ліхтарі, системи відеоспостереження, забруднення та шумові датчики тощо.

Справжньою проблемою буде забезпечення ефективного програмного забезпечення для управління всіма інноваційними системами, які будуть знайдені в Smart City в майбутньому. Це тому, що для кожного пристрою потрібна власна установка та конфігурація програмного забезпечення. Відключення світлофору або зламана система відеоспостереження є потенційно серйозною загрозою для міста. Ось чому цей процес вже розпочався, вживши необхідних заходів для створення постійної системи моніторингу для всіх елементів Smart City. Проте в даний час більшість міст окремо контролюють свої різні системи та елементи. Для того, щоб досягти максимальної надійності та ефективності, при максимально коротких прогалах, необхідно буде впровадити системи, які можуть обмінюватися інформацією між собою та аналізувати дані з різних джерел.

Інфраструктуру Smart City слід контролювати цілодобово. Якщо виникає будь-яка аномалія, вона повинна бути швидко розпізнана та зафіксована, щоб уникнути можливих простоїв. Хоча ми пройшли довгий шлях у розробці потужних і все більш автоматизованих та інтелектуальних технологій, нам все одно доводиться пам'ятати, що чим більша відповідальність ми покладемо на технології, тим ретельніше ми повинні її спостерігати.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. IoT системи

Інтернет речей (IoT) - це мережа фізичних пристроїв, транспортних засобів, побутової техніки та інших предметів, вбудованих в електроніку, програмне забезпечення, датчики, виконавчі пристрої та з'єднання, що дозволяє цим об'єктам підключати та обмінюватися даними. Кожна річ унікально ідентифікується через свою вбудовану обчислювальну систему, але вона здатна взаємодіяти в рамках існуючої інфраструктури Інтернету.

IoT дає змогу дистанційно відстежувати чи управляти об'єктами через існуючу мережеву інфраструктуру, створюючи можливості для більш безпосередньої інтеграції фізичного світу в комп'ютерні системи, що призводить до підвищення ефективності, точності та економічної вигоди, а також до зменшення втручання людини. Коли IoT доповнюється сенсорами та виконавчими пристроями, технологія стає прикладом більш загального класу кібер-фізичних систем, що також охоплює такі технології, як інтелектуальні мережі, віртуальні електростанції, розумні будинки, інтелектуальні транспортні засоби та розумні міста..

"Речі" в системі IoT можуть відноситись до найрізноманітніших пристроїв, таких як імпланти моніторингу серця, біочипи-ретранслятори для сільськогосподарських тварин, камери, автомобілі з вбудованими датчиками, Пристрої для аналізу ДНК для моніторингу навколишнього середовища / їжі / патогену або пристроїв експлуатації на місцях, які допомагають пожежникам в пошуково-рятувальних операціях. Юридичні вчені пропонують розглядати "речі" як "нерозривну суміш устаткування, програмного забезпечення, даних та сервісу". [1]

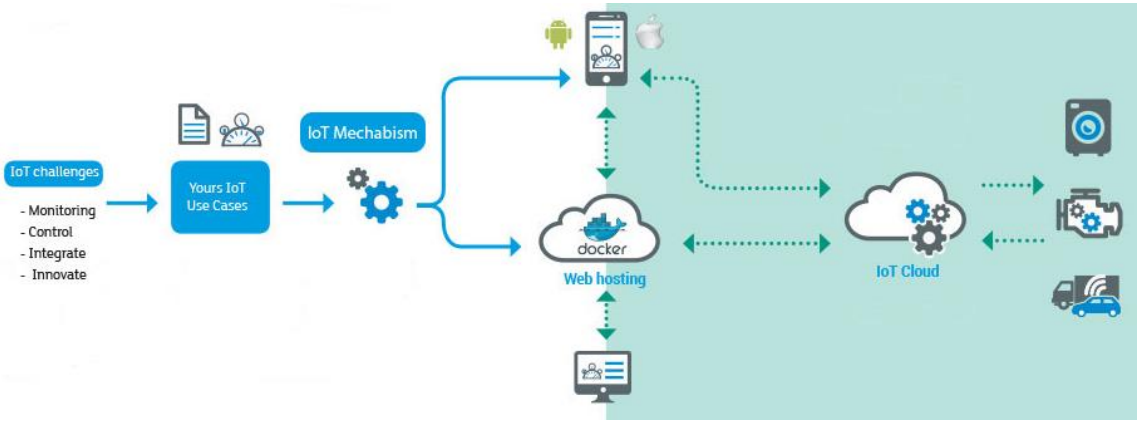


Рис. 1.1. Схема простої IoT системи

Повна система IoT об'єднує чотири різних компоненти: датчики / пристрої, підключення, обробка даних та користувальницький інтерфейс. Нижче я коротко поясню кожен компонент і те, що він робить.

Сенсори та пристрої

По-перше, датчики або пристрої збирають дані зі свого оточення. Це може бути настільки ж простим, як температура, або настільки ж складним, як повне відеовміст.

Я використовую "датчики / пристрої", оскільки декілька датчиків можуть бути об'єднані в комплект, або датчики можуть бути частиною пристрою, який робить більше, ніж просто чутно. Наприклад, ваш телефон - це пристрій з декількома датчиками (камера, акселерометр, GPS тощо), але ваш телефон - це не просто датчик.

Однак, незалежно від того, чи це окремий сенсор або повноцінний пристрій, в цьому першому кроці дані збираються з середовища чимось.

Підключення

Далі ці дані надсилаються в хмарую Але потрібен спосіб потрапити туди.

Датчики / пристрої можуть бути підключені до хмари за допомогою різноманітних методів, таких як: стільниковий, супутниковий, WiFi, Bluetooth, малопотужні широкосмугові мережі (LPWAN) або підключення безпосередньо до Інтернету через мережу Ethernet.

Кожен варіант має компроміс між енергоспоживанням, діапазоном та пропускною здатністю. Вибір того, який спосіб підключення найкраще підходить, залежить від конкретної програми IoT, однак всі вони виконують одне і те ж завдання: отримувати дані в хмарі.

Обробка даних

Після того, як дані потрапляють у хмару, програмне забезпечення виконує певну обробку на ньому.

Це може бути дуже простим, наприклад, перевірка того, що показник температури знаходиться в межах прийнятного діапазону. Або це також може бути дуже складним, наприклад, використання комп'ютерного бачення на відео для ідентифікації об'єктів (наприклад, вторгнень у вашому домі).

Але що трапляється, коли температура занадто висока, або у вашому домі є вторгнення? Саме тут входить користувач.

Користувацький інтерфейс

Далі інформація певним чином стане корисною для кінцевого користувача. Це може бути за допомогою сповіщення користувача (електронна пошта, текст, повідомлення тощо). Наприклад, текстовий сповіщення, коли температура занадто висока в холодному сховищі компанії.

Крім того, користувач може мати інтерфейс, який дозволяє їм активно входити в систему. Наприклад, користувач, можливо, захоче перевірити канали відео в їхньому домі за допомогою програми телефону або веб-браузера.

Проте це не завжди односторонній зв’язок. Залежно від програми IoT, користувач також може виконувати дії та впливати на систему. Наприклад, користувач може дистанційно регулювати температуру в холодному сховищі через додаток на своєму телефоні.

І деякі дії виконуються автоматично. Замість того, щоб чекати, коли ви налаштовуєте температуру, система могла б зробити це автоматично за допомогою попередньо визначених правил. І замість того, щоб просто

зателефонувати вам, щоб сповістити вас про вторгнення, система IoT може також автоматично повідомляти відповідні органи влади.[2]

1.2. Система Smart City

Що таке розумне місто? Відповідь залежить від того, про що ви запитуєте. Провайдери рішень скажуть вам, що це розумна парковка, розумне освітлення або щось інше, пов'язане з технологією. Міські службовці можуть сказати, що це про ведення бізнесу в мережі міста, наприклад, пошук документів або подання заявки на отримання дозволів. Мешканці міста можуть сказати вам, що це легко пересуватися або зменшити рівень злочинності. Кожен має рацію.

Розумне місто, побудоване належним чином, забезпечить різну цінність для різних зацікавлених сторін. Вони не можуть думати про своє місто як про "розумне" місто. Вони знають це лише як місце, в якому вони хочуть жити, працювати і бути частиною. Щоб побудувати цей тип міста, вам спочатку потрібно побудувати екосистему розумного міста.

Сканування різноманітних розумних визначень міста визнало, що технологія є загальним елементом. Наприклад, TechTarget визначає розумне місто як "муніципалітет, який використовує інформаційні та комунікаційні технології для підвищення операційної ефективності, обміну інформацією з громадськістю та покращення якості державних послуг та добробуту громадян".

Інститут інженерів з електротехніки та електроніки (IEEE) передбачає розумне місто, яке об'єднує технології, уряд та суспільство, щоб забезпечити такі характеристики: інтелектуальна економіка, розумна мобільність, інтелектуальне середовище, розумні люди, розумне життя, інтелектуальне управління .

Але що дійсно робить розумне місто? Наша сканування спритна міського проекту по всьому світу показала, що ініціативи потрапляють в одне чи декілька "результатів" розумного міста (рис. 1.1).

Рис. 1.1 Складові розумного міста

В якості відправної точки, ми визначаємо розумне місто, яке широко використовує технології для досягнення ключових результатів для своїх зацікавлених сторін, включаючи мешканців, підприємств, муніципальних організацій та відвідувачів.

Другий малюнок показує нашу основу для розумної екосистеми міста. Яскраве та стале місто - це екосистема, що складається з людей, організацій та підприємств, політики, законів та процесів, об'єднаних разом для створення бажаних результатів, показаних на першому малюнку. Це місто є адаптивним, чуйним і завжди актуальним для всіх, хто живе, працює та відвідує місто. Спритне місто інтегрує технологію для прискорення, полегшення та перетворення цієї екосистеми.

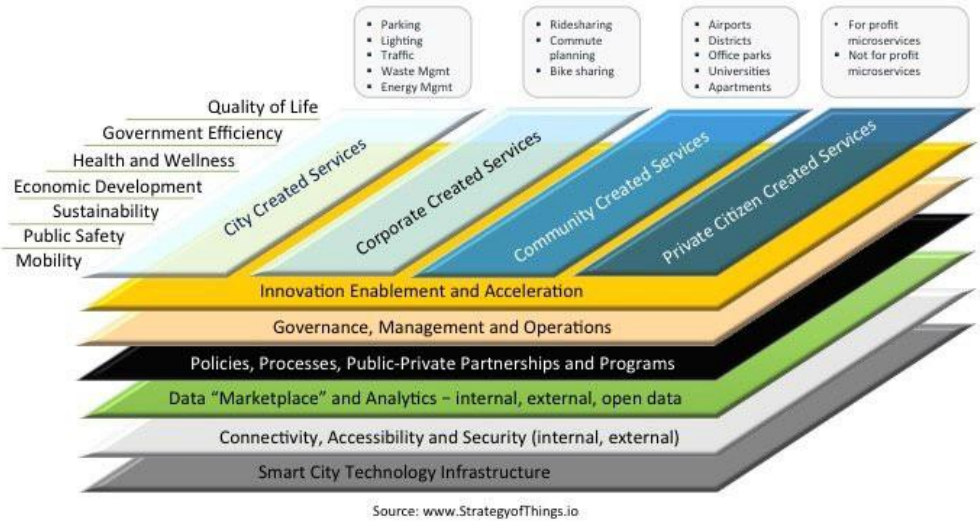


Рис. 1.2 Екосистема розумного міста

Існує чотири типи значущих творців в екосистемі "розумного міста". Вони створюють та споживають цінність навколо одного з результатів, наведених на першому малюнку.

Коли люди думають про розумне місто, вони автоматично думають про послуги, надані муніципальними та квазіурядовими агентствами, такими як розумна парковка, розумне управління водними ресурсами, розумне освітлення тощо. Фактично існує три інші постачальники та користувачі, які співіснують у розумному місті: підприємства та організації, громади та мешканці.

Підприємства та організації можуть створювати послуги, які використовують та створюють інформацію для створення результатів для своїх зацікавлених сторін. Деякі приклади "розумного" бізнесу включають "Убер" та "Лифт" для особистої мобільності, NextDoor для обміну інформацією та Waze / Google для планування трафіку та комунікації.

Общини - мініатюрні, розумні міста, але з дуже локальними потребами. Деякими прикладами потенційних розумних громад є університетські містечка, офісні парки, аеропорти, вантажні порти, багатоповерхові будинки (МДУ) або житлові комплекси, житлові будинки / квартали, ділові райони та навіть окремі "розумні" будівлі. Вони потребують інтелектуальних послуг, які можуть бути спеціально розроблені для своїх зацікавлених сторін.

Мешканці або окремі громадяни також є інтелектуальними постачальниками послуг у розумному місті. Мешканець, що мешкає поблизу небезпечної вулиці, може вказувати на камеру на перехресті та потік цієї інформації жити в планувальників руху та поліції. Реактори розміщують вимірювальні датчики якості повітря за своїми властивостями для моніторингу забруднення та рівня пилку в певні години року та надають цю інформацію доступним іншим учасникам громади. Резиденти можуть вибрати, щоб зробити ці розумні послуги тимчасовими або постійними, а також безкоштовними або платними.

Розумне місто - це екосистема, що складається з декількох "шарів можливостей". Хоча технологія є критично важливим фактором, це лише одне з багатьох основних можливостей, яке повинно мати кожне розумне місто. Жодна здібність важливіша, ніж решта. Кожна з можливостей відіграє іншу роль у розумному місті, і вони повинні інтегрувати та координувати один з одним для виконання своєї місії. [3]

Значення шару Це найяскравіший шар для жителів міста, підприємств, відвідувачів, працівників, студентів, туристів та інших. Цей рівень є каталогом інтелектуальних міських послуг або "випадків використання", орієнтованих на результати (Рис 1.1), і пропонованих цінностями творців та споживаних міськими зацікавленими сторонами.

- Інноваційний шар. Щоб залишатись актуальними, цінніші творці в розумному місті повинні постійно інноваційно оновлювати та оновлювати свої послуги для своїх зацікавлених сторін. Розумні міста активно сприяють цьому через різноманітні інноваційні програми, включаючи лабораторії, інноваційні зони, тренінги, семінари з ідей, розвиток навичок та партнерські відносини з університетами та бізнесом.
- Управління, управління та операційний рівень. Спритне місто створює збій і призводить до цифрового перетворення існуючих процесів та послуг. Інтелектуальні моделі управління містом

повинні інтегрувати нову екосистему ціннісних творців та новаторів. Вони повинні планувати, підтримувати та монетизувати нові бізнес-моделі, процеси та послуги. Вони повинні вдосконалити існуючу інфраструктуру та процеси управління, щоб підтримувати "розумні" послуги. Нарешті, вони повинні оцінити ефективність роботи міста за допомогою нового набору показників.

- Політика, процеси, державно-приватні партнерства та рівень фінансування. Спритне місто не просто чарівно з'являється в один прекрасний день. Зовсім новий набір моделей взаємодії, правила, джерела фінансування та партнери вимагають побудови, експлуатації та підтримки інтелектуального міста. Міста повинні розробити новий набір "розумних" компетенцій для того, щоб отримати та залишитися в "розумній міській грі".
- Інформація та рівень даних. Жива сила розумного міста - це інформація. Спритне місто має сприяти цьому по-різному, включаючи ініціативи з відкритими даними, ринки даних, служби аналітики та політику монетизації. Не менш важливо, вони повинні мати програми, які заохочують обмін даними та політику конфіденційності для захисту того, що і як збирають дані.
- Зв'язок, доступність та рівень безпеки. Люди, речі та системи взаємопов'язані в розумному місті. Можливість безперешкодно з'єднувати всі три, керувати і перевіряти, хто і що пов'язано та спільно використовується, тимчасом як захищати інформацію та користувачів є надзвичайно важливим. Найважливішими пріоритетами для "розумних" міст є забезпечення бездоганного шару надійних з'єднань.
- Розумна міська технологічна інфраструктура. Більшість людей автоматично думають про технології, коли говорять про розумні міста. Інтелектуальна інфраструктура міських технологій повинна

поширюватися за межі традиційних муніципальних користувачів та підтримувати новий клас творців цінностей та зацікавлених сторін міста / користувачів.

Розумне місто - це складна екосистема людей, процесів, політики, технологій та інших засобів, що працюють разом для досягнення набору результатів. Спритне місто не "належить" виключно містом. Також беруть участь інші творці, котрі іноді співпрацюють, а іноді самі по собі. Успішні та стабільні "розумні міста" використовують програмний підхід для залучення своїх зацікавлених сторін до всієї екосистеми.

Наші дослідження виявили, що багато міст не використовують екосистемний підхід до проектів розумного міста. Це пояснюється, зокрема, тим, що розумні міські проекти управляються організацією інформаційних технологій (IT), де їх статут стосується розробки та розгортання систем. На відміну від більш досвідчених, розумних міст управляють своїми розумними міськими програмами через внутрішні крос-функціональні організації "Трансформація" або "Інновації".

Незалежно від того, де міста знаходяться в розумному міському проїзді, вони повинні випереджати "криву" з розумними міськими проектами. Вони починають з того, що думають про будівництво ширшої екосистеми, щоб створити стійке та масштабоване розумне місто. Ключовими наступними кроками є:

1. Зрозумійте інтелектуальну систему екосистеми міста та адаптуйте її до реалій свого конкретного міста. Внесіть цю модель у розвиток їхнього розумного бачення, стратегії та планів розвитку.
2. Відносно інтелектуальної системи екосистем в місті, визначити існуючі можливості та прогалини у різних рівнях. Зрозумійте, що потрібно для підтримки чотирьох типів цільових творців.
3. Оцінювати існуючі та нові проекти та ініціативи в області інтелектуального розвитку міста, що стосуються екосистем.

Використовуйте цю схему для визначення того, чого не вистачає з планів проекту, і що потрібно для успішного виконання проектів.

4. Визначте та розвивайте компетенції різних екосистемних шарів. Спритне місто вимагає нових навичок і компетенцій. Розширювати існуючі можливості за допомогою стратегічних партнерських відносин та укладання контрактів з постачальниками послуг у міру необхідності. [4]

1.3 Прогресивні веб додатки

Це веб-додатки, які є звичайними веб-сторінками або веб-сайтами, але можуть з'являтися користувачеві, як традиційні додатки або нативні мобільні додатки. Тип програми намагається об'єднати функції, пропоновані більшістю сучасних веб-переглядачів, з перевагами мобільного інтерфейсу.

Маніфест веб-програми - це специфікація W3C, яка визначає маніфест на основі JSON, щоб надати розробникам централізоване місце для розміщення метаданих, пов'язаних із веб-додатком, включаючи:

- Назва веб-програми
- Посилання на значки веб-програми або об'єкти зображення
- Рекомендована URL-адреса для запуску або відкриття веб-програми
- Дані конфігурації веб-програми для ряду характеристик
- Декларація за умовчанням веб-програми
- Дозволяє встановити режим відображення, наприклад, повноекранний
- Завдяки налаштуванню та обробці метаданих для файлу веб-маніфеста розробники дозволяють користувальницьким агентам створити бездоганний нативний мобільний досвід через прогресивну веб-програму.

Внутрішні мобільні додатки забезпечують багатий досвід і високу продуктивність, придбані за рахунок місця для зберігання, відсутність оновлень у режимі реального часу та низький рівень видимості пошукової

системи. Традиційні веб-програми страждають від зворотного набору факторів: відсутність власного компілятора виконуваного файлу, а також залежність від ненадійного та потенційно повільного веб-з'єднання. Службовці використовуються в спробі забезпечити прогресивні веб-додатки кращим з обох цих світів. [5]

Технічно працівники служби надають в веб-браузері проксі-сервер мережею для сценаріїв для керування запитами веб / HTTP програмно. Обслуговуючі працівники розташовуються між мережею та пристроєм для забезпечення контенту. Вони здатні ефективно використовувати механізми кешування і дозволяють працювати без помилок під час автономних періодів. Властивості службовців:

- Тригер і підтримка відношень до подій, а не до документів
- Загальний за своєю природою
- Події керовані контекстами сценаріїв за часом та працюють у початковій точці
- З природними кінцевими точками для широкого кола служб виконання
- Має свій стан з URL-адресою сценарію, що містить реєстрацію
- Виділений ідентифікатор або UUID
- З життєвими подіями
- Має карту ресурсів скриптів
- Можна пропустити, чекаючи тригеру

Переваги веб-службовців:

- Можливість простого обробки push-повідомлення
- Синхронізуйте дані у фоновому режимі
- Можливість відповідати на запити ресурсів до оригінального ресурсу в будь якому місці
- Отримання централізованого оновлення [6]

1.4 Реактивне програмування

Ми хочемо зробити наше застосування більш чутливим. Ми хочемо забезпечити користувачам безперебійну роботу наших користувачів без зависання основного потоку, сповільнюючи їх, і ми не хочемо надавати нашим користувачам високоякісну продуктивність.

Щоб зберегти основний потік, нам потрібно заробити багато важкої роботи, яку ми хочемо зробити у фоновому режимі. Ми також хочемо провести важку роботу та складні розрахунки на наших серверах, оскільки мобільні пристрої не дуже потужні для важкого підйому. Тому ми потребуємо асинхронної роботи для мережевих операцій.

Давайте подивимося, що нам потрібно від бібліотеки, яка обробляє всю асинхронну роботу. Ви можете уявити нижче 4 бали як матрицю оцінки для асинхронної бібліотеки.

Явне виконання: якщо ми розпочнемо виконання купу робіт над новою гілкою, ми повинні мати можливість контролювати це. Якщо ви збираєтеся виконати певне фонове завдання, ви збираєте інформацію та готуєте їх. Як тільки ви будете готові, ви можете почати фонове завдання.

- Легке управління потоками: в асинхронній роботі управління нитками є ключовим. Нам часто доводиться оновлювати інтерфейс користувача в основному потоці з фонової нитки посередині завдання або в кінці завдання. Для цього ми повинні передати нашу роботу з одного потоку (фоновий потік) в іншу гілку (тут основна тема). Таким чином, ви повинні мати можливість легко перемикати потоку і передавати роботу іншому потоку, коли це необхідно.
- Легко компостувати: в ідеалі, було б чудово, якщо ми можемо створити асинхронну роботу, і коли ми починаємо прямування фонової нитки, це просто робота без будь-якої іншої гілки (особливо в потоці користувальницького інтерфейсу) і

залишається незалежною від іншого потоку, поки не закінчиться його робота. Але в реальному житті нам потрібно оновити інтерфейс користувача, внести зміни в базу даних і ще багато речей, які роблять взаємозалежність потоків. Таким чином, асинхронна бібліотека повинна бути легко складною і забезпечити менше місця для помилки.

- Мінімум побічних ефектів: під час роботи з декількома потоками, інший потік повинен мати мінімальні побічні ефекти з іншого потоку. Це робить ваш код легко читабельним і зрозумілим для нової людини, а також робить помилку легко відстежуваною.[7][8]

1.5. Системи монітиронгу в Smart City

Smart City Monitor - новаторська та потужна міська інформаційна технологія, що забезпечує цифрові перетворення та цифрову програму. Він ефективно сприяє реалізації концепцій інтелектуального міського майбутнього та відповідних відкритих екосистем для експериментів, створення прототипів, навчання та поетапного прогресу з низькими витратами та мінімальними ризиками.

Завдяки цілісному баченню комплексних процесів у міських районах як Системи систем на загальних мобільних телефонах в режимі реального часу, столичні зацікавлені сторони можуть ефективно поліпшити стабільність місцевого і регіонального врядування та якості життя, просувати та оцінювати нові бізнес-моделі, оперативну ефективність.

Технологічне рішення Smart City забезпечує високий рівень індивідуалізації та інтеграції інформації з декількох окремих міських систем, що перетворюють кілька великих потоків даних у багатий набір індивідуальних послуг для зацікавлених сторін. Вона включає в себе командування, контроль та аналітика для управління громадами, містобудівниками та комунальними підприємствами, послуги для громадян, туристів та місцевого бізнесу.

У результаті зацікавлені сторони отримують нову міську екосистему для столичних районів та островів, що забезпечує високу прозорість, участь і громадські ініціативи.

- Основні компоненти платформи включають в себе наступне:
- Розумний монітор міського монітора працює локально або в хмарі
- Веб-клієнти та мобільні додатки Smart City Monitor для доступу до інформації про міські об'єкти, процеси та карти
- Інтерактивні програмні інструменти для розробки місцевої міської моделі та його зв'язування з різними міськими системами, датчиками та відкритими джерелами даних з використанням локальних параметрів зв'язку
- Міська модель за ISO 37120 повинна бути пов'язана з місцевими відкритими джерелами даних і використовуватися як шаблон для запуску та розробки детальної міської моделі

Платформа надається як сервіс під моделлю SaaS, яка включає безперервну підтримку, покращення функціональних можливостей, продуктивність, інтеграцію мобільних додатків та налаштування. Двигуни з місцевими міськими моделями управляються місцевою командою Smart City. Прогрес моделі може бути здійснений на місцевому рівні або переданий стороннім спеціалістам з питань міського планування на регіональному або національному рівнях або постачальник послуг Smart City Monitor. Місцева спільнота має авторські права на власну модель міста, яка поступово розвивається з часом.

Реалізація рішення в рамках пілотного проекту забезпечує місцеву адміністрацію та зацікавлені сторони з економічно вигідним та низьким ступенем доступу до інноваційних технологій, знань та сприяє створенню місцевої громадської екосистеми. На етапах проекту - планування, навчання, монтаж, розробка міських моделей, інтеграція з місцевими джерелами даних, створення нових бізнес-моделей та інформаційних послуг для міських районів

та міст маленьких міст, оцінка та презентації на відповідних публічних заходах. [9]

1.5.1. Моніторинг якості повітря

Моніторинг та підтримка якості повітря є одним з найважливіших аспектів управління містом. Значна частина населення живе у містах, де показник якості повітря перевищив межі деяких забруднювачів повітря, таких як: частинки (ПМ), озон, діоксид азоту. Це забруднення створює серйозні ризики для здоров'я та навколишнього середовища. Система управління якістю повітря в режимі реального часу є важливою складовою будь-якого розумного міста.

Системи моніторингу емісій безперервного кола (CSEMS) та Системи моніторингу безперервної непрозорості (COMS) передбачають встановлення обладнання для моніторингу, яке накопичує дані за заздалегідь встановленим графіком роботи. Регуляторні установи або плати з контролю за забрудненням спираються на методи безперервного моніторингу, за якими джерела викидів повинні самостійно звітувати, коли перевищено дозволенний ліміт.

1.5.2. Моніторинг навколишнього рівня шуму

Підвищений рівень шуму впливає як на здоров'я, так і на поведінку. Звук стає небажаним, коли він або перешкоджає нормальній діяльності, такій як сплячий режим, розмова або порушення або зниження якості життя. Небажаний звук (шум) має шкідливий вплив на психологічне здоров'я. Забруднення шуму може спричинити гіпертонію, підвищений рівень стресу, шум у вухах, втрату слуху, порушення сну та інші шкідливі наслідки.

Завдяки моніторингу шуму в режимі реального часу, можна буде відповісти на певні проблеми. Він надає дані для підтримки містобудівників рішення про відмову або введення умов контролю нових проектів, які мають потенціал для створення високого рівня шуму.

1.5.3. Моніторинг води та стічних вод

Одним з найважливіших об'єктів міста є його водойми та питна вода. З наростанням у містах неминуче стрімко зросте стічні води та забруднення водних джерел. Споживання води в містах зростає, і в більшості міст важко задовольнити зростаючий попит на воду. Моніторинг води буде контролювати та управляти водою та стічними водами. Це буде гарантія того що вода не забруднена та не витрачається даремно. Це дасть можливість отримувати дані в реальному часі та дані про якість води, рівні забруднювачів, використання води та витоків, що дає можливість ефективно керувати та зберігати цей цінний ресурс.

1.5.4 Регулювання освітлення

Міста можуть контролювати витрати на обслуговування та світлову продуктивність, щоб заощадити час та гроші, а оптимізувати умови освітлення на основі потреб кожного блоку. Також за допомогою сенсорів руху можна буде контролювати коли потрібно включати та виключати освітлення щоб не було “холостого” освітелння.

1.5.5 Оптимізація стоянки для автомобілістів

Автомобілісти можуть отримувати інформацію в режимі реального часу про місцезнаходження наявних місць для паркування.

1.5.6 Рух транспорту

Затори можуть бути зменшені завдяки маршрутизації трафіку в режимі реального часу залежно від погоди, часу будівництва та інтервалу зміни кольору. Це дасть можливість заощадити витрати часу на подорож від пункту А до пункту Б.

Висновки до розділу 1

Визначивши що таке IoT системи, Smart City, та які можливості нам можуть дати системи моніторингу, були поставленні задачі дипломного проекту, а саме:

- Архітектура кожної підсистеми моніторингу
- Вибір хмарної платформи
- Побудова архітектури моніторингу для міста
- Побудова архітектури IoT системи
- Створення додатку для користувача для аналізу даних
- Вибір технологій

РОЗДІЛ 2. РОЗРОБКА АРХІТЕКТУРИ РОЗТАШУВАННЯ ПРИСТРОЇВ

2.1. Розробка архітектури системи моніторингу в системі Smart City

Для ідеального моніторингу слід розмістити датчики на кожному стовбуру ліхтаря або світлофорі, відстань між ліхтарями приблизно 10 м, отже в нас буде приблизно 100 пристроїв на кілометр, це велика кількість інформації і це дуже велике навантаження на сервер. Організація мережі буде дротовою, тому що бездротове вимагає батарей. Так що система у нас реального часу, то термін життя батареї буде дуже маленьким. Крім того, негативні температури сильно зменшують ємність живильних елементів. Да і заглушити бездротовий зв'язок відносно легко, а це вже дуже істотним недоліком для системи безпеки. ГОСТи забороняють проводити по огорожі 220В, тому живитись наші пристрої будуть від постійного струму. Для передачі інформації між датчиком та сервером будемо використовувати шлюз. Він містить 2 шини до яких будуть підключатися датчики та інші пристрої. Також шлюз займається управлінням датчиків, контролем параметрів живлення та навколишньої середи, тобто система розбивається на модулі, кожен модуль містить станцію — сукупність датчиків, сенсорів та якихось ще приладів та IoT шлюз. Таким чином з сервера можна зняти дуже велике навантаження.

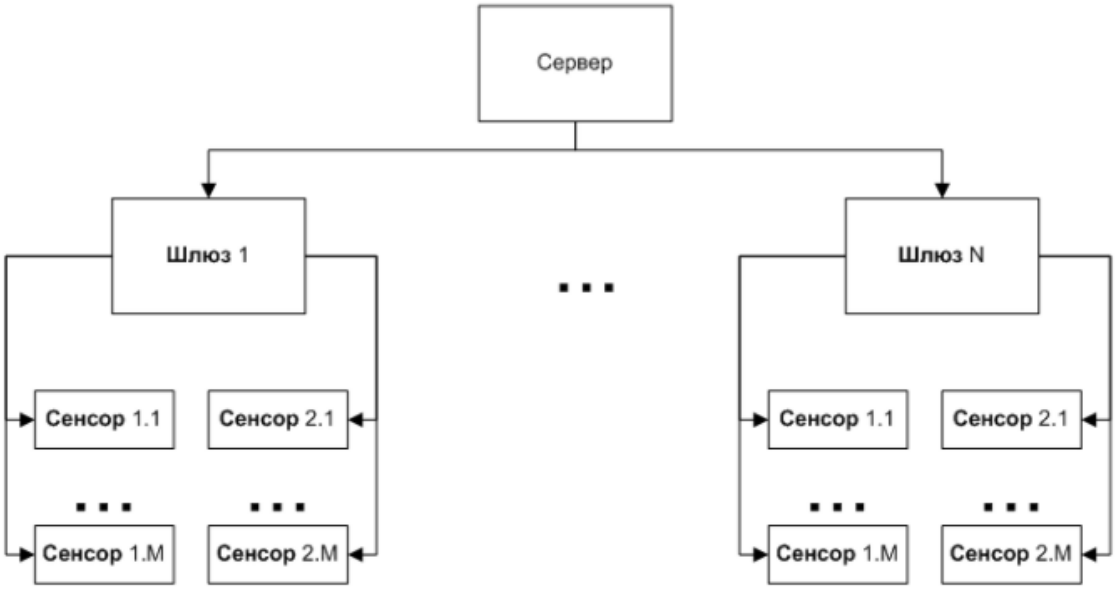


Рис. 2.1 Архітектура мережі системи

Для більш швидкого результату будемо використовувати асинхронне надходження подій від датчика до сервера. У ролі шини ми будемо використовувати CAN, що дозволить нам підключити велику кількість приладів на велику відстань, оптимально буде вибрати 500м, щоб не було втрати швидкості та спотворення данних. В шлюзі будемо використовувати 2 драйвери CAN, що дозволить об’єднати датчики з 2х сторін і ми зможемо покрити 2*500м = 1км покриття, тобто в нас вийде 1 модуль на 1 км. Також CAN дає можливість використовувати протокол CANopen, хоча б нам і хотілось зробити всю систему по принципу PnP, CANopen дає деякі обмеження що до цього, але це не дуже суттєво.

Шлюз має вихід Ethernet. Це найбільш універсальна технологія передачі даних. Мережа може бути організована як завгодно: оптичними каналами, бездротовими каналами, звичайної кручений парою - при цьому ми завжди зможемо підключитися до цієї мережі, використовуючи оптичні конвертери і точки доступу. Це дозволяє проектувати інфраструктуру мережі будь-якої складності і протяжності. Передача організуємо за допомогою Сокетів Берклі на базі TCP / IP. Таке рішення дозволяє серверу гарантовано отримувати інформацію від будь-якого датчика і не залежати від програмних платформ.

Поверх TCP / IP будемо використовувати байтовий протокол, для оптимізації роботи на стороні мікроконтролера. У байтових протоколів є великий мінус: складність з подальшою модифікацією. Однак текстовий протокол для мікроконтролерного пристрою занадто надмірний.

Для максимальної гнучкості системи слід розбити наше Smart City на райони Smart Area, у кожного району буде свій сервер, який буде надсилати дані до загальної системи збору та обробки інформації, отже архітектура нашої системи буде виглядати наступним чином.

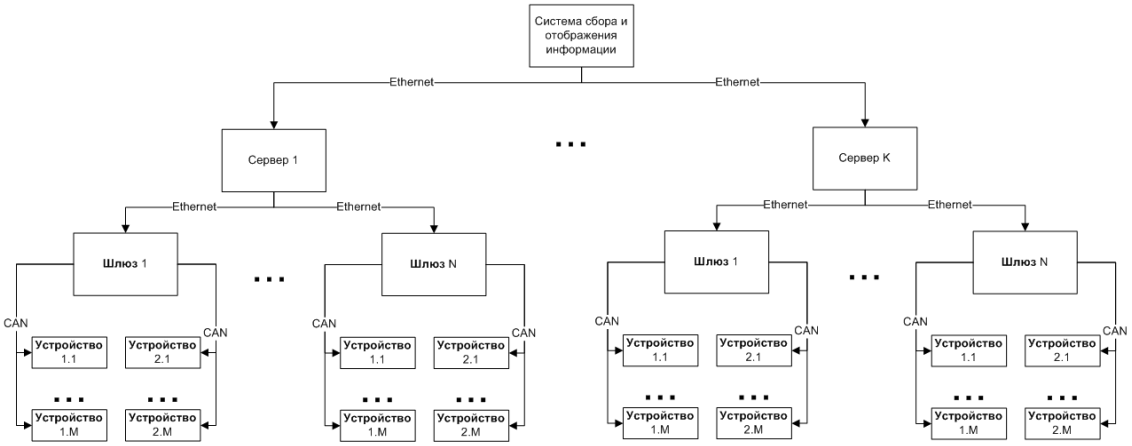


Рис. 2.2 Архітектура мережі приладів для системи Smart City

З безпекою системи все досить просто. Всі мережі, які знаходяться на об'єктах що охороняються, самі по собі є охоронними об'єктами. Таким чином всі мережі, з якими працює система, стають "довіреними".

Крім того, "ціна" злому інформаційної системи охорони набагато вище, ніж інші способи подолання. Іншими словами, досвідчений порушник знайде більш простий спосіб подолати загородження, а менш досвідчений просто не зможе зламати систему. Тому ніякими особливими способами захисту інформації користуватися не будемо, обмежившись лише базовими принципами.

2.2 Хмарна платформа

2.2.1 Cloud IoT Core

Cloud IoT Core надає підприємствам повнофункціональне рішення для безпечного з'єднання та моніторингу / керування цими пристроями у глобальному масштабі. Це означає, що пристрої можуть підключатися до хмари через IoT Core, а не оброблятись шлюзом / іншою платформою та передачею даних.

Окрім усунення іншого шару процесу розгортання, IoT Core автоматично додає безпеку за замовчуванням і допомагає перемістити величезну кількість датчиків у свій набір продуктів обробки та аналізу даних. Google Cloud IoT Core також автоматично забезпечує балансування навантаження та горизонтальне масштабування як частину роботи в їх серверній інфраструктурі. І, нарешті, якщо ви вже перебуваєте на платформі Android Things, інтеграція з IoT Core допоможе вимкнути прошивку та пристрій без проблем.

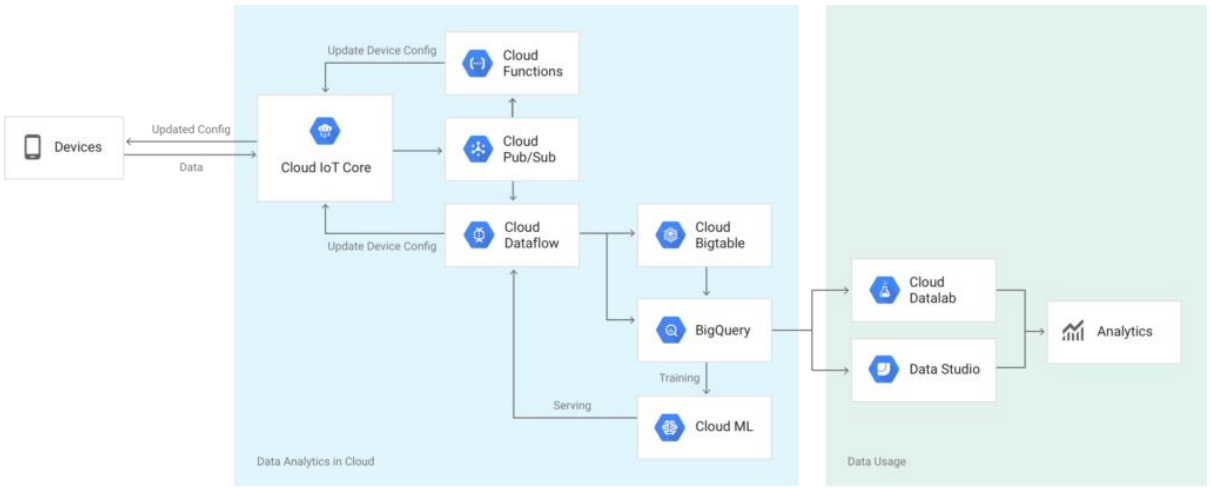


Рис. 2.3 Cloud IoT Core архітектура [10]

Блог Google описує архітектуру зразка для збору даних з пристроїв, аналітики в хмарі та оновлення конфігурації на пристрої. Простий приклад - включити смарт-вентилятор, коли пристрій виявить, що кімната стає надто спекотною.

Хоча цей приклад добре, він не забезпечує кінцевої архітектури розгортання IoT, включаючи компонент інтерфейсу користувача.

Ось деякі речі, які стосуються Cloud IoT Core:

Генерація ключів RS256 / ES256 дуже проста

- Деякі основні функції все ще розробляються. Наразі створення та видалення партії пристроїв не підтримується. Оскільки реєстри не можна видалити, поки всі пов'язані пристрої не буде видалено, опція для пакетних дій буде корисною для швидкого створення прототипів.
- Інтеграція з Pub / Sub є приємною і бездоганною, але може бути кращою. Наразі вам потрібно надати обліковому запису Cloud IoT Core публікувати доступ до Pub / Sub перед налаштуванням реєстрів. Автоматично створюючи цей доступ, коли активує Cloud IoT Core або дає програмний спосіб дозволити обліковий запис служби Compute Engine, цей доступ призведе до вилучення іншої точки тертя.
- Існує квота на 1 конфігурацію / оновлення штату на пристрій за сек. Це може стати на шляху до демонстрації в залежності від того, як планується використовувати Cloud IoT Core. Існує також 10 000 пристроїв за запитом на список, тому буде потрібен перелік сторінок, якщо прикладному обладнанні потрібний більший набір пристроїв. [10]

2.2.2 IBM Watson

У якості хмарної платформи IoT можна використовувати один з готових рішень: IBM Watson, Google Cloud, AZURE Cloud. Всі вони надають потужні засоби для роботи з IoT, обробку інформації та інформаційну безпеку. На мою думку IBM Watson найкращий варіант.

Платформа Watson IoT розпізнає два класи пристроїв: керовані пристрої та некеровані пристрої.

Керовані пристрої визначаються як пристрої, що містять агент керування пристроєм. Агент керування пристроєм - це набір логіки, який дає змогу пристрою взаємодіяти з Watson IoT Platform Device Management Device за допомогою протоколу керування пристроєм. Керовані пристрої можуть виконувати операції керування пристроями, включаючи оновлення місцезнаходження, завантаження та оновлення прошивки, оновлення та перезавантаження заводських налаштувань.

Протокол керування пристроєм визначає набір підтримуваних операцій. Агент керування пристроєм може підтримувати підмножина операцій, проте операції керування та невдалого керування повинні підтримуватися. Пристрій, який підтримує операції із вбудованим програмним забезпеченням, повинен також підтримувати спостереження. API, визначені тут, використовуються для взаємодії з керованими пристроями за допомогою протоколу керування пристроєм (Таблиця 2.1). [11]

Таблиця 2.1 Управління пристроями

Тип запиту	Запит	Дія
DELETE	/device/types/{typeId}/devices/{deviceId}/diag/errorCodes	Очистити коди помилок
GET	/device/types/{typeId}/devices/{deviceId}/diag/errorCodes	Отримати коди помилок діагностики
POST	/device/types/{typeId}/devices/{deviceId}/diag/errorCodes	Додати код помилки діагностики для пристрою
DELETE	/device/types/{typeId}/devices/{deviceId}/diag/logs	Очистити журнал діагностики

GET	/device/types/{typeId}/devices/{deviceId}/diag/logs	Отримати всі діагностичні журнали пристроїв
POST	/device/types/{typeId}/devices/{deviceId}/diag/logs	Додати інформацію про журнал діагностики пристрою
GET	/device/types/{typeId}/devices/{deviceId}/diag/logs/{logId}	Отримати діагностичний журнал пристрою
GET	/mgmt/custom/bundle	Отримати список зареєстрованих розширень керування пристроєм
POST	/mgmt/custom/bundle	Додати розширення керування пристроєм
DELETE	/mgmt/custom/bundle/{bundleId}	Видалити розширення керування пристроєм
GET	/mgmt/custom/bundle/{bundleId}	Отримайте спеціальне зареєстроване розширення

		керування пристроєм
PUT	/mgmt/custom/bundle/{bundleId}	Оновити існуюче розширення керування пристроєм
GET	/mgmt/requests	Перелік запитів на керування пристроєм
POST	/mgmt/requests	Ініціюйте запит на керування пристроєм
GET	/mgmt/requests/{requestId}	Очистити стан запиту керування пристроєм
POST	/mgmt/requests/{requestId}/cancel	Скасувати запит на керування пристроєм

API управління інформацією можна використовувати для доступу до даних про події пристрою, а також отримання та оновлення місцезнаходження пристрою та отримання метеорологічної інформації для цього місця розташування. (Таблиця 2.2)

Таблиця 2.2 Організація управління інформацією REST API

Тип запиту	Запит	Дія

GET	/device/types/{typeId}/devices/{deviceId}/location	Отримати інформацію про місцезнаходження пристрою
PUT	/device/types/{typeId}/devices/{deviceId}/location	Оновіть інформацію про місцезнаходження пристрою
GET	/devices/types/{typeId}/devices/{deviceId}/exts/twc/ops/geocode	Отримати поточні данні спостереження за місцезнаходженням, пов'язаним із вашим пристроєм
GET	/config/lec	Отримати конфігурацію кешу останньої події для вашої організації
GET	/config/lec	Оновити конфігурацію кешу останньої події для вашої організації

Також IBM Watson надає велику кількість API для створення користувацьких інтерфейсів. [12]

Висновки до розділу 2

На даному етапі було створено оптимальну архітектуру для розташування пристроїв для системи моніторингу в Smart City. Також була обрана хмарна платформа IBM Watson, яка буде збирати данні с пристроїв, обробляти їх та віддавати нам у сприятливому для людини вигляді. Система була обрана дротова, та було прийняте рішення розбити все розумне місто на розумні райони, які будуть мати свої сервера, та збирати данні дані для певного райони і надсилати їх на головний сервер.

Використовуючи API, яке надає нам IMB Watson, можна перейти для створення програмного забезпечення.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СИСТЕМИ МОНІТОРИНГУ В SMART CITY

3.1 Розробка архітектури програмного забезпечення

IBM Watson надає велику кількість API для управління пристроями та отримання даних за допомогою REST API, тому слід розбити системи на основну, клієнтську та адмінську. Наш додаток буде складатися с 3х піддодатків: core, client, admin. Це потрібно для того щоб уникнути дублювання функціоналу додатків. Основна логіка роботи буде знаходитись у додатку CORE, який буде втягувати інформацію з платформи та за допомогою API, який він буде надавати іншим 2м додаткам, показувати її користувачу, або адміністратору.

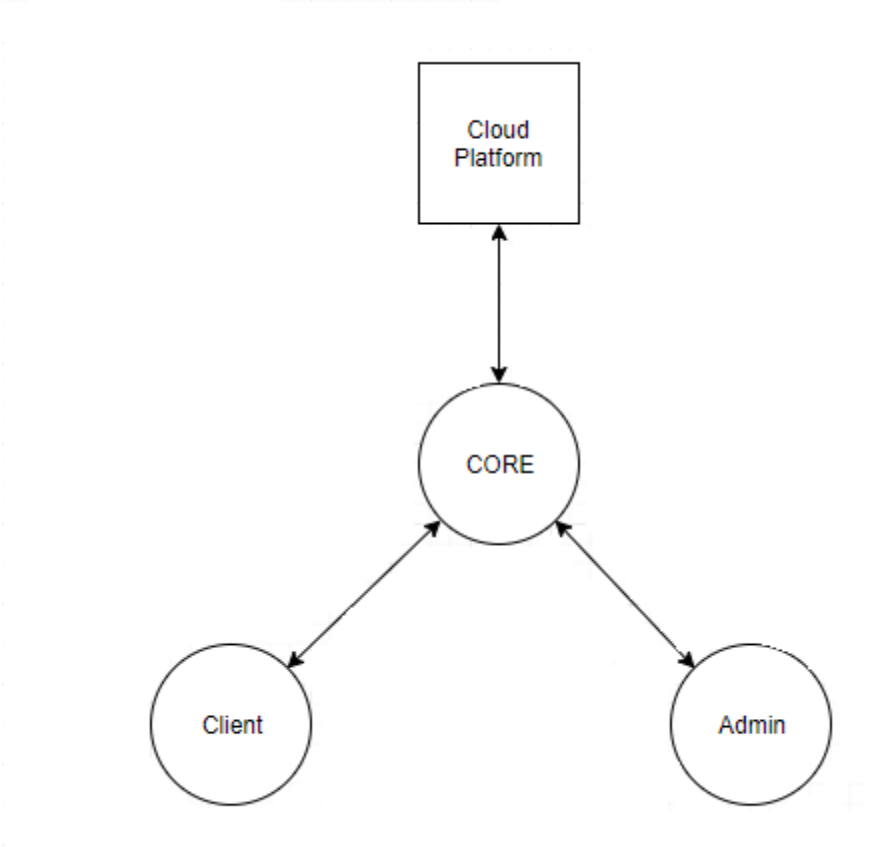


Рис. 3.1 Архітектура взаємодії додатків

3.2 Обрані технології

3.2.1 Angular

Angular - це платформа, яка полегшує створення додатків з Інтернетом. Це комбінат декларативних шаблонів, ін'єкцій залежності, кінцевих інструментів і інтегрованих передових методів вирішення проблем розвитку. Angular дозволяє розробникам створювати програми, які живуть в Інтернеті, на мобільних пристроях або на робочому столі.

Робота з Angular передбачає, що ви вже знайомі з JavaScript та деякими інструментами з найсучасніших стандартів, таких як класи та модулі. Зразки коду записуються за допомогою TypeScript. Більшість Angular кодів можна записати лише за допомогою останнього JavaScript, використовуючи типи для ін'єкцій залежності та використання декораторів для метаданих.

Angular дозволяє розділити бизнесс-логіку, рендер-логіку, та відображення, що робить розробку веб додатків більше зручною та читабельною.

Прогресивні веб-додатки генерують велику популярність в ці дні. Розуміючи цю тенденцію, компанія Angular Team підкреслила важливість спрощення процесу створення PWA. Не тільки в тому, що з Angular 5.0 можна отримати функції власних мобільних додатків із мобільних веб-додатків, таких як push-сповіщення та офлайнові можливості. Це стало можливим, оскільки Angular може самостійно створювати код та конфігурувати за допомогою Angular-CLI.

Інше важливе оновлення в Angular 5.0 полягає в тому, що компоненти Material Design тепер сумісні з рендерингом на стороні сервера. Однак ця функція поки що не випущена.

Angular 5.0 поставляється з Angular Universal State Transfer API та підтримкою DOM для спільного кодування між версіями програми на сервері та на стороні клієнта. Кутовий Універсал робить програму на стороні сервера. Це збільшує сприйману продуктивність програми. [13]

3.2.2 NodeJS та Express

Node.js — це JavaScript–оточення побудоване на JavaScript–рушієві Chrome V8. Node.js використовує подієву, неблокуючу I/O модель, що робить його легким та ефективним. Пакетна екосистема Node.js, npm, є найбільшою у світі екосистемою бібліотек з відкритим кодом.

На високому рівні Node.js поєднує в собі движок Google V8 JavaScript, однопотоковий неблокуючий цикл подій і низькорівневий API введення / виводу.

Express - це мінімалістичний і гнучкий веб-фреймворк для додатків Node.js, що надає великий набір функцій для мобільних і веб-додатків.

Маючи в своєму розпорядженні безліч службових методів HTTP і проміжних оброблювачів, створити надійний API можна швидко і легко.

Express надає тонкий шар фундаментальних функцій веб-додатків, які не заважають вам працювати з функціями Node.js. [14]

3.2.3 RxJs

Реактивні розширення для JavaScript (RxJS) - це бібліотека реактивних потоків, яка дозволяє працювати з асинхронними потоками даних. RxJS може використовуватися як у браузері, так і на серверній стороні за допомогою Node.js.

Асинхронний, в JavaScript означає, що ми можемо викликати функцію та зареєструвати зворотний виклик, щоб отримувати сповіщення, коли результати доступні, тому ми можемо продовжувати виконання та уникати того, що веб-сторінка не відповідає. Це використовується для викликів ajax, DOM-подій, промісів, WebWorker та WebSockets.

Дані, необроблена інформація у вигляді типів даних JavaScript: Число, Строка, Об'єкти (Масиви, Набори, Мапи).

Потоки, послідовності даних, доступних з часом. Як приклад, протилежність масиву вам не потрібна вся інформація, яка буде присутня, щоб почати використовувати їх.

У RxJS ви представляєте асинхронні потоки даних із використанням спостережуваних послідовностей або навіть просто називають observables. Спостережувані елементи дуже гнучкі і можуть бути використані за допомогою push або pull шаблону.

Використовуючи push шаблон, ми підписуємось на вихідний потік і реагуємо на нові дані, як тільки буде доступний.

Використовуючи pull схему, ми використовуємо однакові операції, але синхронно. Це трапляється при використанні масивів, генераторів або ітерацій.

RxJS робить багато чого дійсно чудово. Це дозволяє створювати власні потоки подій, які можуть мати декілька слухачів на виході. Моделі спостережуваних та перетворення ланцюгів дуже корисні і допомагають зберігати суміжні частини коду разом. Спостережувані створюють потік, який можна поєднати з іншими потоками. Він забирає глобальні події та зберігає їх у локалізованому місці, де ваш код потребує реагування на події. Якщо ви використовували мутаційні спостерігачі, пов'язані з передачею даних або магістральні події в JS або KVO або NSNotificationCenter в Objective-C, RxJS подібний цьому, але набагато більш потужним. RxJS підвищує потік даних, щоб бути першокласним архітектурним шаблоном на рівні MVC.

Отож озирнувшись і розмовляючи з друзями, я подивився на Go Lang. Останнім часом з'являється велика кількість друкованих видань для гарної паралелізму з потоками і без неї (з вагомих причин). Вони використовують схему під назвою CSP. Ви створюєте об'єкт каналу, який дозволяє передавати дані по ньому. Це дуже схоже на FRP. Потім, коли ви хочете, щоб передавати дані по каналу або приймати дані, ви просто сказати йому, щоб встановити дані змінної і «парк», поки ці дані не встановлені. Так код, який виглядає як нормальний синхронний і йде через підрядник acutally парків і чекає, поки щось йде через канал перед переходом до наступного рядка. JavaScript буде отримувати це з `очікування 'в ES2016.

Go lang і CSP дійсно здорово, чи не так? Робота Async виглядає як робота синхронізації, і весь код разом дозволяє легко раціоналізувати. Але я не можу використовувати це в моєму веб-додатку. Саме тут з'являється ClojureScript. Їх спільнота відома тим, що використовує чудові ідеї з інших мов програмування. Core.async ClojureScript є неймовірним. Вони в основному вкрали найкращі частини Go і FRP - це дуже простий API. Core.async відчуває себе як програмування 5 років у майбутньому. Ядро асинхронне мають канал Go, що «парк», щоб зробити асинхронний код виглядати як код синхронізації і на вершині, що вони додали перетворювач. Перетворювачі - це ці чудові речі, щоб взяти купу абстрактних функцій, які виконують незначні завдання і швидко складають їх разом і прикріплюють їх до каналу. [8] [15]

3.2.4 Couch DB

Apache CouchDB - це програмне забезпечення з відкритим вихідним кодом, яке фокусується на простоті використання та масштабованій архітектурі. Вона має документально-орієнтовану архітектуру бази даних NoSQL і реалізована в мультивалютному середовищі Erlang; він використовує JSON для зберігання даних, JavaScript як мову запиту, використовуючи MapReduce, і HTTP для API.

На відміну від реляційної бази даних, база даних CouchDB не зберігає дані та відносини в таблицях. Натомість кожна база даних являє собою сукупність незалежних документів. Кожен документ зберігає власні дані та самостійну схему. Програма може мати доступ до декількох баз даних, таких як один, що зберігається на мобільному телефоні користувача, а інший - на сервері. Метадані документів містять інформацію про перегляд, що дає змогу об'єднати будь-які розбіжності, які могли мати місце під час відключення баз даних.

CouchDB реалізує форму керування мультиверсійним паралелізмом (MVCC), тому він не блокує файл бази даних під час запису. Конфлікти залишаються у заявці для вирішення. Вирішення конфлікту зазвичай

передбачає спочатку об'єднання даних в один з документів, а потім видалення застарілого.

Інші функції включають семантику ACID на рівні документа з можливістю послідовності, (додаткової) MapReduce та (інкрементальної) реплікації. Одним з відмінних функцій CouchDB є реплікація декількох майстрів, що дозволяє масштабувати всі машини для створення високопродуктивних систем. Вбудована веб-програма під назвою Fauxton (раніше Futon) допомагає з адмініструванням.

CouchDB є одним з тих, що багато хто називає рішення NoSQL. Зокрема, CouchDB є орієнтованою на документи базою даних, і в кожному документі поля зберігаються як карти основних значень. Поля можуть бути або простою парою ключ / значення, списком або картою.

У кожному документі, який зберігається в базі даних, надається унікальний ідентифікатор (_id) на рівні документа, а також номер редакції (_rev) для кожної зміни, яка складається та зберігається в базі даних.

Бази даних NoSQL являють собою перехід від традиційних реляційних баз даних і можуть запропонувати багато переваг (і власних проблем). CouchDB пропонує нам такі можливості:

- Легка реплікація бази даних через кілька серверних примірників
- Швидке індексування та вилучення
- REST-подібний інтерфейс для вставки, оновлення, вилучення та видалення документів
- Формат документа на основі JSON (легко перекладається на різних мовах)
- Кілька бібліотек для вибраної мови (показати деякі популярні мовні варіанти)
- Підписувані дані оновлюються в каналі _changes [16][17]

3.2.5 JSON

В нашому додатку всі данні приходять у форматі JSON(JavaScript Object Notation). У той самий час JSON базується на тексті, і може бути з легкістю прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). JSON виступає як заміна XML під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти. JSON будується на двох структурах:

1. Набір пар ім'я/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативним масивом.
2. Впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список, або послідовність.

При використанні JSON формату даних для їх обробки можуть використовуватися одні і ті ж інструменти серіалізації/десеріалізації. [18]

3.2.6 Electron

Електрон (раніше відомий як Atom Shell) - це система з відкритим кодом, створена та підтримувана GitHub. Це дозволяє розробляти настільні графічні програми з використанням передніх і задніх компонентів, які були розроблені для веб-додатків: час виконання Node.js для бекенда та Chromium для зовнішнього інтерфейсу. Electron є основним графічним інтерфейсом для декількох відомих проектів з відкритим кодом, включаючи GitHub's Atom і редакторів початкового коду Microsoft Visual Studio Code, настільне додаток служби Tidal streaming services та LightNet IDE, крім безкоштовний настільний клієнт для служби чату Discord. [19]

3.3 Додаток Core

Додаток Core – це проміжний додаток між платформою та користувачем(middleware). Як було сказано в розділі вище, додаток Core буде надавати API для користування за допомогою REST API. Цей додаток буде фільтрувати запити за допомогою інтерсепторів, потім цей додаток буде відправляти запит на IOT платформ, результат приводити у читабельний для людини вигляд та відправляти далі. Кількість користувачів у нас ціле місто, отже має сенс кешувати результат для певного проміжку часу, щоб уникнути надання застарілої інформації слід ревалідувати кеш через певні проміжки часу, час буде залежати від типу інформації, наприклад, для погоди раз в годину, для стану на дорогах раз в 15 хвилин, стан у водоймі раз в 24 години.

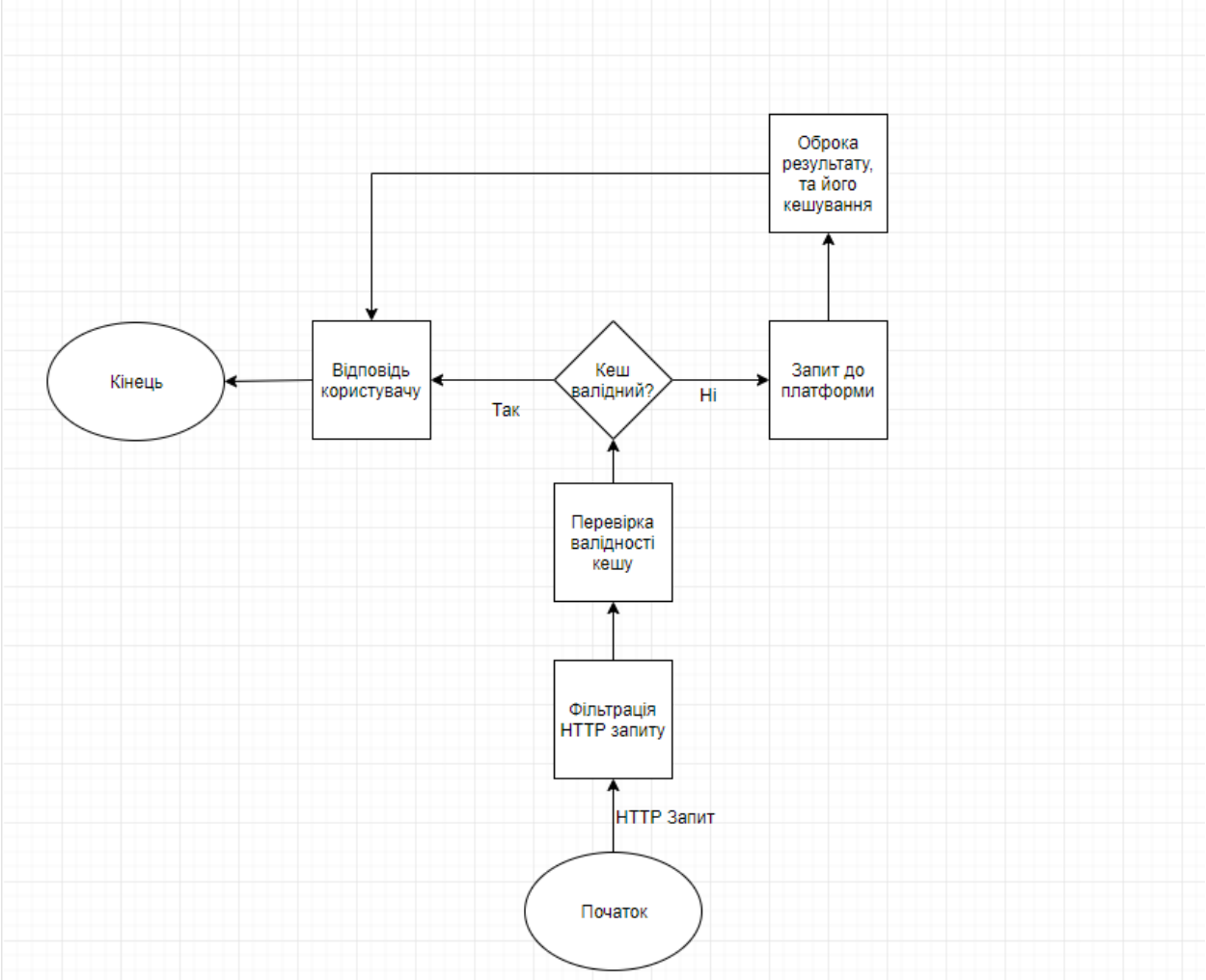


Рис. 3.2 Система обробки запитів в додатку Core від додатку Client

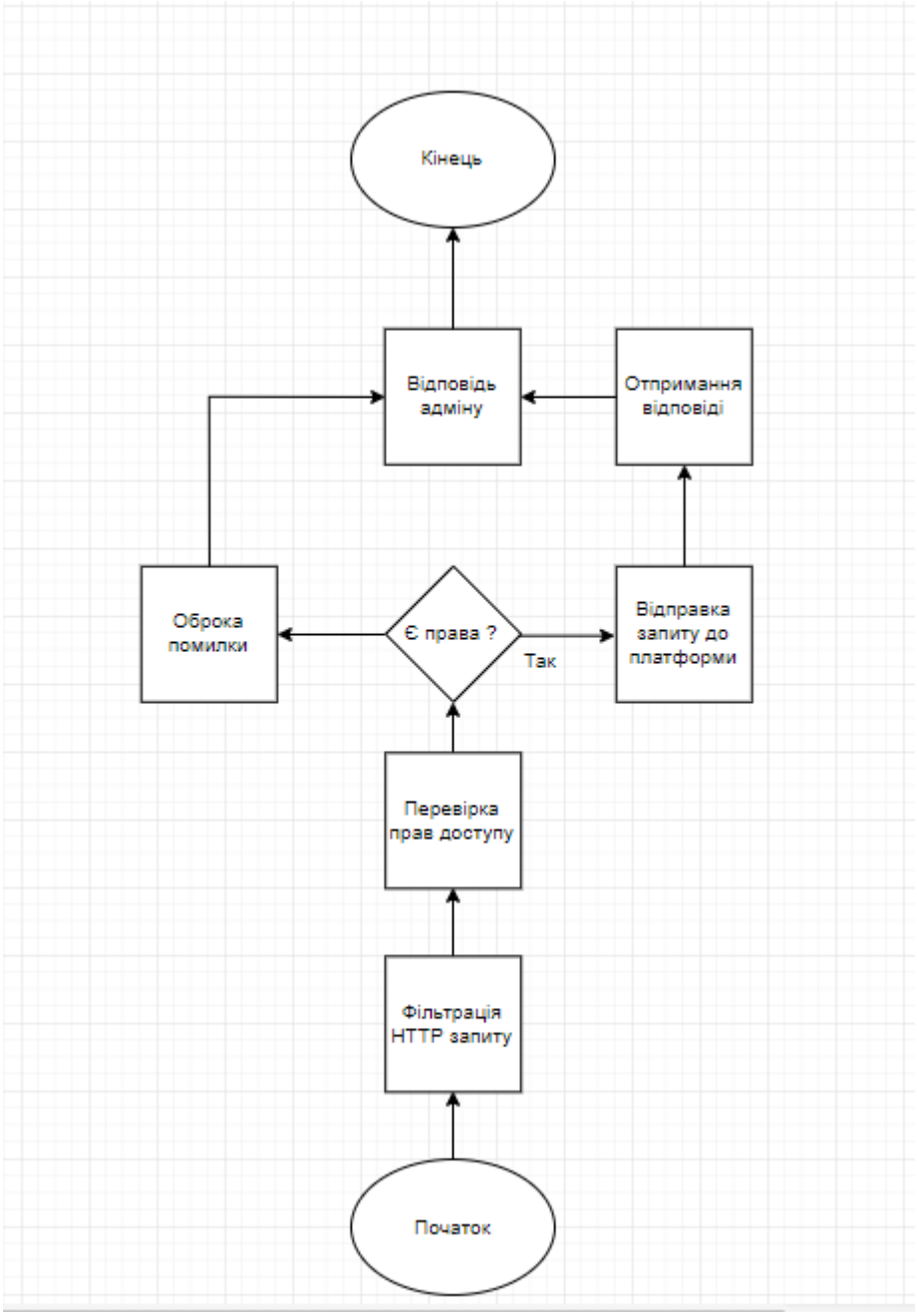


Рис. 3.3 Система обробки запитів в додатку Core від додатку Admin

Також через додаток Core будуть проходити всі повідомлення від користувача до адміністратора, або навпаки.

Для створення даного проміжного додатку будемо використовувати node.js та фреймворк express, який не потребує багато зусиль для створення інтерсепторів та маршрутизаторів та роботи з базою даних.

Буде створюватися власний кабінет для кожного користувача, отже потрібно десь зберігати данні про аккаунти, а так як наша платформа

знаходиться на IBM Watson, то для бази даних будемо використовувати IBM Cloud, яка використовує не реляційну базу даних Couch DB, для того щоб працювати за даною базою треба буде використовувати node-couchdb, який дає API, для під'єднання до БД, та роботи CRUD операції.

Приклад інтерсептору для запиту від client до core на погоду:

```
const express = require('express');
const interceptor = require('express-interceptor');
const rx = require('Rx');
const NodeCache = require("node-cache");
const myCache = rx.Observable.of(new NodeCache({stdTTL: 100, checkperiod:
360000}));

const app = express();

const finalInterceptor = interceptor(function (req, res) {
    return {
a→      if(req.type === 'GET' && req.link.include('weather') &&
req.headers.filter("'weather':'true'")){
b→       const response = myCache.switchMap((cache) => chache.checkStatus() ?
cache.getData().weather :
            rx.observable.ajax('/watson/collectedData/weather#latest/'));

c→        response.subscribe((data) => {
            const convertedData = JSON.parse(data);

d→          const response = {
                [convertedData.time]: {
                    'temperature': convertedData.temperature,
                    'windSpeed': convertedData.windSpeed,
                    'windDirection': convertedData.windDirection,
                    'preasure': convertedData.preasure,
                    'clearly': convertedData.clearly
                }

                send(response);
e→              myCache.addData(data);
        })else {
f→              next(req)
      }
    }
};

// Add the interceptor middleware
app.use(finalInterceptor);

app.use(express.static(__dirname + '/public/'));

app.listen(3000);
```

a) Умова фільтрування запиту
b) Перевірка валідності кешу, якщо кеш валідний, то ми повертаємо результат із кешу, якщо ні, то мі робимо запит до хмарної платформи

c) Так я ми використовуємо RxJS для реактивного програмування, то на потік в який прийде результат потрібно підписатися, щоб відстежувати зміни і як тільки туди прийде результат, то ми зможемо його обробляти
d) Результат приходить в форматі строки, тому його потрібно перевести в об’єкт, витягуючи данні які нам потрібно передати користувачу, та надсилаємо це користувачу
e) Кешуємо новий результат
f) Якщо умова не виконалась, то пропускаємо запит далі, де його може перехопити інший інтерсептор

3.4 Додаток Client

Client – це додаток для жителя міста в якому він зможе знаходити певну інформацію яка його цікавить. Для максимальної зручності треба дати можливість користувачу використовувати додаток в режимі офлайн, на основі останньої отриманої інформації та її кешу, тому розробка піде в сторону прогресивних веб додатків.

Прогресивні веб-додатки мають бути швидкими та інстальованими, що означає, що вони працюють в режимі онлайн, в автономному режимі та на періодичних і повільних з'єднаннях. Для цього ми повинні кешувати нашу оболонку додатків за допомогою service-worker, щоб вона завжди була доступною швидко і надійно. Першим кроком до роботи додатка в автономному режимі є реєстрація службовця, скрипт, що дозволяє працювати у фоновому режимі без необхідності відкритої веб-сторінки або взаємодії з користувачем. Для цього треба правильно вибрати стратегію кешування, стратегія буде залежати від типу даних, наприклад інформація про стан на дорогах або погоді має бути максимально свіжою. Стратегія cache-first-then-network ідеально підходить для нашого додатка. Він отримує дані на екрані якомога швидше, після чого оновлюється, коли мережа повертає останні дані.

При запуску нашого додатку нам потрібно буде запустити зразу 2 асинхронні запити, один в кеш пам’ять, а один в мережу, також треба

налаштувати service-worker так, щоб спочатку він закешував відповідь, а потім уже оновив данні на екрані. За звичайних умов кешовані дані буде повернуто майже відразу, надаючи додатку останні дані, які він може використовувати. Потім, коли мережевий запит повертається, додаток буде оновлюватися, використовуючи останні дані мережі. Для того щоб наша технологія працювала правильно треба налаштувати інтерсептори, які будуть перенаправляти запит у кеш пам’ять.

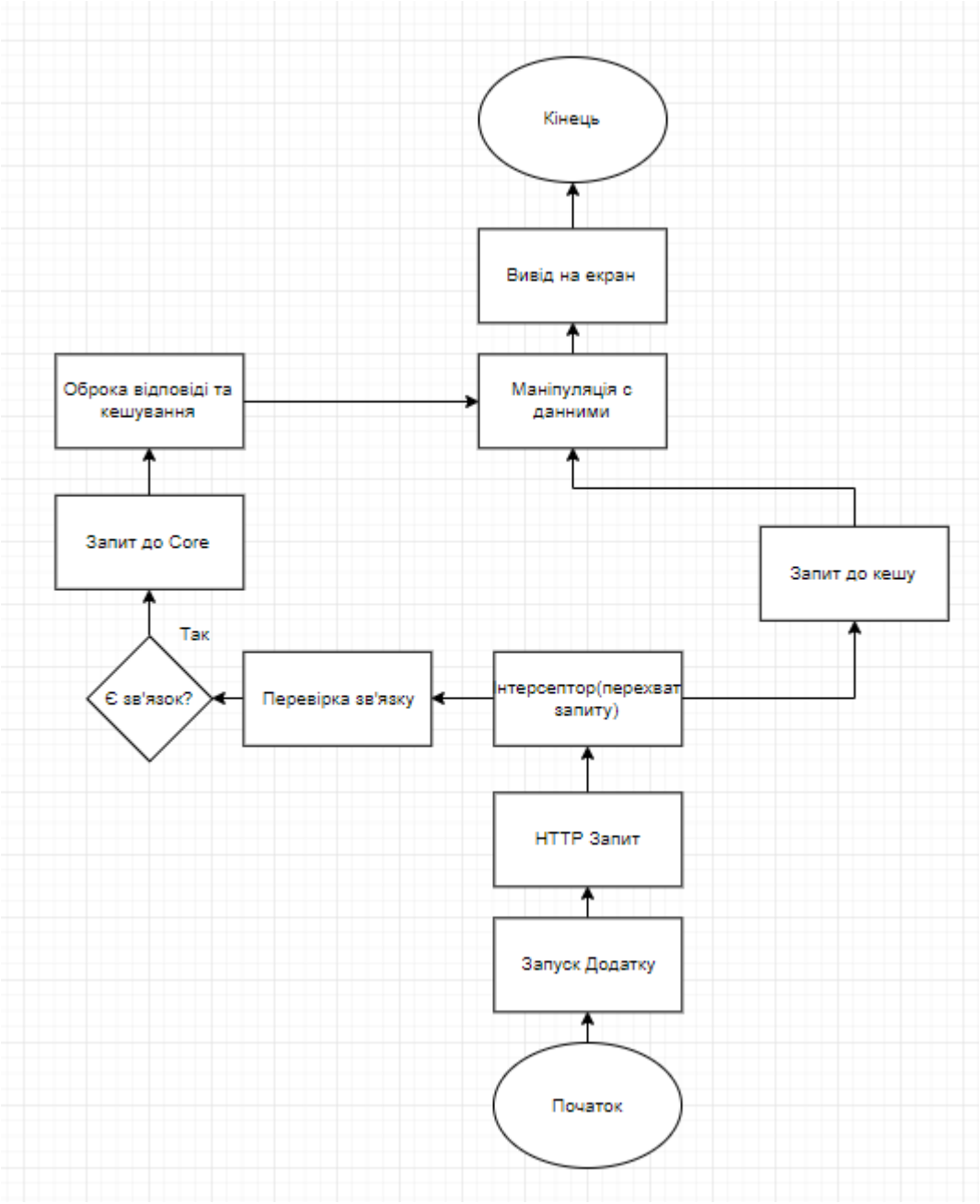


Рис 3.4 Принцип роботи додатка Client

На додатку на клієнтській частині буде лише декілька вкладок, одна на загальну інформацію, типу погоди, стану повітря і тому подібне, вкладка зі станом на дорогах, пробки, ремонти, вільні парковки та стан ближчих водойм. Для відображень результату треба інтегрувати API, якихось карт.

Застосунок для відображення обирався серед додатків Яндекс.Карты, 2ГИС та Google Maps. З недоліків Яндекс.Карт можна відзначити відсутність деталізації на рівні будівель, відсутність відображення у режимі 3D та слабке покриття міст і регіонів по всьому світу. 2ГИС карти мають гарну деталізацію об’єктів але слабке покриття у всьому світі. Google Maps покривають весь світ на високому рівні деталізації. Так як 2ГИС і Яндекс.Карты поступаються Google Maps в більшості порівняльних характеристик (покриття світу, деталізація відображення), а універсальність і точна робота методу націлена на будь-яку територію, то було обрано Google Maps API як технологію відображення картографічних даних.

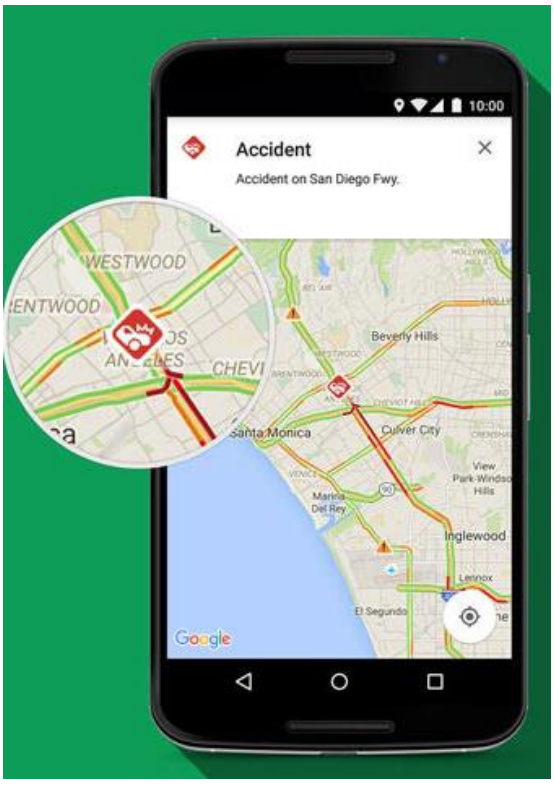


Рис. 3.3 Приклад роботи клієнтського додатку.

Для роботи із статистичними даними на клієнтській стороні було обрано формат JSON, так як цей формат найбільше підходить для збереження даних у картографічному вигляді. Виходячи з даних умов, для попередньої перевірки даних на стороні кліента було обрано технологію для валідації JSON Schema. JSON Schema це потужний інструмент для перевірки структури даних JSON.

Для створення клієнтського додатку будемо використовувати Angular, Service-worker та RxJs.

Angular побудований на MVC схемі. У моделі MVC можна створити кілька представлень для моделі. Дублювання коду в MVC дуже обмежене, оскільки відокремлює дані та бізнес-логіку від дисплея. це нам дуже підходить оскільки логіка для всіх вкладок з інформацією буде однакова.

RxJs дає нам змогу асинхронно створювати запити та моментально реагувати на відповідь від запиту.

Для максимального ефекту, до імплементації кешування service-worker'у додамо RxJs.

Отже для початка зареєструемо наш service-worker

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker
        .register('./service-worker.js')
        .then(function() { console.log('Service Worker Registered'); });
}
```

Коли service-worker запускається, він повинен відкрити об'єкт кеш-пам'яті та занести його до ресурсів, необхідних для завантаження оболонки програми. Створемо файл із назвою service-worker.js у кореневій папці програми. Цей файл повинен жити в кореневому застосуванні, оскільки можливості працівників служби визначаються директорією, в якій знаходиться файл, обернемо його в Angular Service, який буде запускатися при запуску додатку.

```
@Injectable()
export class ServiceWorker {
    constructor(){
        const cacheName = 'monitoring data';
        const filesToCache = [];

        window.addEventListener('install', function(e) {
            console.log('[ServiceWorker] Install');
```

```
            e.waitUntil(
                caches.open(cacheName).then(function(cache) {
                    console.log('[ServiceWorker] Caching app shell');
                    return cache.addAll(filesToCache);
                })
            );
        });
    }
}
```

Пізніше до масиву filesToCache додамо файли які потрібно буде кешувати, по перве це будуть файли оболочки(зобрження, меню і тому подібно) і json файл в якому будуть зберігатися наші данні.

Код нижче гарантує, що наш service-worker оновить кеш-пам'ять кожного разу, коли зміниться якийсь із файлів оболонки додатка. Для того, щоб це працювало, потрібно збільшити змінну cacheName у верхній частині файлу службовця.

```
window.addEventListener('activate', function(e) {
  console.log('[ServiceWorker] Activate');
  e.waitUntil(
    caches.keys().then(function(keyList) {
      return Promise.all(keyList.map(function(key) {
        if (key !== cacheName) {
          console.log('[ServiceWorker] Removing old cache', key);
          return caches.delete(key);
        }
      }));
    })
  );
  return window.clients.claim();
});
```

Як було сказано вище, на початку потрібно дати 2 асинхронні запити, один для кешу інший до додатку core, для цього мы використаємо Interceptor, але для початку нам треба зробити ін’єкцію сервісу який ініціалізує кеш.

```
@Injectable()
export class FetchingData implements HttpInterceptor {
    constructor(private serviceWorker: ServiceWorker) {}
    intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
        const newRequest = request.clone({
            headers: request.headers.set('settings',
JSON.stringify(this.userSettings))
        })

        return
next.handle(newRequest).combineLatest(this.serviceWorker.caches.claim());
```

```
    }
}
```

Отже тепер у нас є імплементація перехвату запиту, який робить два запити, один до кешу, один до додатку core.

3.5 Додаток Admin

На відмінну від додатку Client додаток Admin створений для слідкування не тільки за станом міста, але й за станом приладів, отримувати швидкі сигнали тривоги, попередження певних служб і тому подібне, отже данні в цьому додатку не має сенсу кешувати, бо нам потрібні максимально свіжі данні і вони будуть дуже часто оновлюватись. За допомогою додатка Admin, можна отримувати інформацію про місце знаходження приладів, керувати ними, відключати їх від мережі та отримувати порівняльну статистику та графіки враховуючи попередні данні.

Для зручності додаток admin має бути декстопнтим, тому для його створення ми оберемо наступні технології: Angular, RxJs, Electron, але щоб застовувати Electron та Angular одночасно, треба зробити декілька махінацій.

По перше, в головному файлі(index.html) підключити уже перекомпільований Angular додаток.

По друге в скриптах для компиляції скрипта додати спочатку компіляцію Angular додатку, а потім Electron додатку.

```
"electron": "ng build && electron .",
"electron-aot": "ng build --prod && electron ."
```

Побудова всього останнього не відрізняється від побудови додатку Client.

3.6 Технічні вимоги для додатків

1. Основні вимоги до устаткування:
    - комп’ютер на базі AMD - сумісного процесору, починаючи з процесора Phenom, з тактовою частотою 2,10 гігагерц,

рекомендовано —Phenom(tm) II N830 Triple-Core з тактовою частотою 4200 мегагерцта вище;

- об’єм оперативної пам’яті — 256 Мб, рекомендований об’єм — 1024 Мб та більше;
- мінімальний об’єм вільного дискового простору — 15 мегабайт, рекомендований об’єм — 100 мегабайт та більше;
- монітор із підтримкою мінімального дозволу екрану 800 на 600 пікселів;
- для зручної роботи рекомендується дозвіл 1280 на 1224 пікселів та більше.

2. Вимоги до програмного забезпечення:
    - Будь-яка операційна система
    - браузер з підтримкою Java Script (Google Chrome, Opera, Mozilla Firefox, Safari)
    - налаштоване з’єднання з мережею Інтернет.

Висновки до розділу 3

В даному розділі було розроблено архітектуру програмного забезпечення як для адмінської частини, так і для жителів міста. Було обрано ряд технологій, для розробки максимально зручного користування для обох сторін, які дозволять швидко та якісно розробити програмне забезпечення, було враховано платформи на яких будуть використовуватись данні додатки, як мобільні так і стаціонарні.

РОЗДІЛ 4. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

Стартап як форма малого ризикового (венчурного) підприємництва впродовж останнього десятиліття набула широкого розповсюдження у світі через зниження бар'єрів входу в ринок (із появою Інтернету як інструменту комунікацій та збуту стало простіше знаходити споживачів та інвесторів, займатись пошуком ресурсів, перетинати кордони між ринками різних країн), і вважається однією із наріжних складових інноваційної економіки, оскільки за рахунок мобільності, гнучкості та великої кількості стартап-проектів загальна маса інноваційних ідей зростає.

4.1. Опис ідеї проекту

Таблиця 4.1. Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для влади
Розробка архітектури розташування датчиків та мережі в місті Київ. Розробка додатку для жителів міста Розробка додатку керування для адміністрації.	Моніторинг за станом міста	Покращення рівня життя для жителів міста. Економія витрат енергії.

Додатки виконані у формі декстопного інтерфейсу, веб інтерфейсу та мобільного інтерфейсу.

Таблиця 4.3 Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	Розроблювальний проект	Компанія Kyiv Smart City	Компанія Smart city solutions	W	N	S
1	Ергономічний та	Сучасний веб інтерфейс	Сучасний інтерфейс	Застарілий інтерфейс		+	

	сучасний дизайн інтейфейсу		настільного додатку				
2	Дротова мережа пристроїв	є	відсутній	відсутній			+
3	Бездротова мережа пристроїв	відсутня	так	ні		+	
4	Розподілення прав доступу через проміжний додаток	є	відсутній	відсутній			+
5	Платформа на наймотужнішому компьютері IBM Watston	є	відсутня	відсутня			+
6	Використання своїх серверів для IoT плаформи	Немає	є	немає		+	
7	Кроссплатформеність ПЗ	так, за рахунок виконання в формі веб сервісу	MacOS, Windows, Linux	MacOS та Windows			+

Основними перевагами є те, що в додатках є розподілення прав доступу, додатки працюють асинхронно, мережа використовується дротова та працює за принципом PnP, що дає більшу захищеність системи, а саме головне, що платформа в нас знаходиться на IBM Watson. Слабкість проекту в тому що компанії вже деякий час створюють прототип системи яка буде працювати без перебоїв.

4.2. Технологічний аудит ідеї проекту

Таблиця 4.2. Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Система моніторингу в Smart City.	Моніторинг за станом навколишнього середовища	+	+
		Моніторинг за станом на дорогах.	+	+
		Сенсори для контролю трафіка та освітлення.	+	+

Технології, необхідні для створення продукту, доступні наразі для розробника. Але треба зауважити, що технологія для моніторингу системи Smart City потребує додаткових досліджень так як це нова та швидко розвиваюча сфера.

4.3. Аналіз ринкових можливостей запуску statup проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Таблиця 4.4. Характеристика потенційних клієнтів стартап-проекту

№	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*

1	Моніторинг всіх аспектів міста	Жителі міста	Кожен район, кожне місто потребує свого індивідуального підходу	Економічні та естетичні
2	Повний контроль над приладами	Міська адміністрація	Для цієї групи більш важливий контроль за приладами та їх станом	Економічні та естетичні

Таблиця 4.5. Фактори загроз

№	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1	Є аналоги які вже деякий час на ринку	Використання додатків аналогів	Відмова від продукту

Таблиця 4.6 Фактори можливостей

№	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1	Моніторинг за різними аспектами міста	Автоматизований процес збору даних пристроями моніторингу	Розширити аспекти моніторингу
2	Надання додатку в формі веб сервісу, мобільного додатку та декстопного додатку	Можливість використовувати те що подобається не залежно від платформи	Підвищення ефективність та швидкості виконання робіт
3	Візуалізація даних моніторингу	Отримання характеристик які не можна побачити не озброєним оком	Відображати більш детальну інформацію
4	Інтеграція з Google map для відображення стану на дорогах, водойм та парковках	Отримання даних у реальному часі, зручне відображення на карті	Побудова дорожніх карт за рахунок розташованих датчиків

Таблиця 4.7. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Тип конкуренції	монополістична	Виведення на ринок нової технології та її активний розвиток
Рівень конкурентної боротьби	міжнародний	Розширення ринку продажу та підтримка багатомовного інтерфейсу
За галузевою ознакою	галузева	Вдосконалення та розвиток технологій в залежності від появи схожих аналогів.
Конкуренція за видами товарів	товарно-родова	Аналіз ринкових технологій та відстеження найпопулярніших рішень на ринку.
За характером конкурентних переваг	нецінова	Удосконалення продукту, що спрямоване на підвищення базових переваг.
За інтенсивністю	не марочна інтенсивність	Забезпечення масштабованості стартапу в найближчій перспективі для створення стійкого сприйняття стартапу як окремої бізнес одиниці.

Таблиця 4.8. Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Навести перелік прямих конкурентів	Визначити бар'єри входження в ринок	Визначити фактори сили постачальників	Визначити фактори сили споживачів	Фактори загроз з боку замінників
Висновки	Компанія Smart city solutions яка надає майже ті ж самі послуги	Великі можливості входу на ринок. Потенційний конкурент – Smart city solutions	Слабкий вплив постачальників. Залежить від збоїв на серверах хостингової компанії	Клієнти частково диктують умови формування цінової політики та критерії якості продукту.	Більш якісні товари замінники можуть вплинути на лояльність споживачів і перетягнути їх на

					свою сторону

Таким чином проект має високі шанси виходу на ринок. Сильною стороною проекту є те ще що дротова система буде більш дешевшою та більш захищеною, також мається 2 додатку, для адміністрації та для жителя міста, що дає змогу фільтрувати потрібну інформацію. Всі додатки крос-платформені, чого немає у конкурентів.

Таблиця 4.9. Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Подання системи як веб додаток	Зручність у використанні, відпадає необхідність у додаткових установках
2	Збір можливостей з різних додатків в одну систему	Ефективніше використання програми та економія грошей та часу

Таблиця 4.10. Порівняльний аналіз сильних та слабких сторін

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з … (назва підприємства)						
			–3	–2	–1	0	+1	+2	+3
1	Подання системи як веб додаток	18							+
2	Збір можливостей з різних додатків в одну систему	16						+	

Таблиця 4.11. SWOT- аналіз стартап-проекту

Сильні сторони: унікальність проекту; подання системи як веб сервіс.	Слабкі сторони: мала розвиненість функціональності додатку в порівнянні з аналогами; не розрекламований продукт.

Можливості: зростання грошових доходів; застосування сучасних технологій; покращення якості товару.	Загрози: інфляційні процеси; поява аналогів від великих компаній.

Таблиця 4.12. Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Стратегія нейтралізації ринкових загроз сильними сторонами стартапу.	висока	1 рік
2	Стратегія підсилення сильних сторін за рахунок ринкових можливостей.	висока	6 місяців
3	Стратегія компенсації слабких сторін наявними ринковими можливостями.	середня	2 роки

4.4. Розроблення ринкової стратегії проекту

Визначимо стратегію охоплення ринку за рахунок опису цільових груп потенційних споживачів.

Таблиця 4.13. Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Приватні споживачі та радіостанції	висока	достатній	помірна	Невисокий бар’єр входження
2	Приватні споживачі та студії звукозапису	висока	достатній	помірна	Середній бар’єр входження

Таблиця 4.14. Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи
1	Стратегія диференціації	Диференційний маркетинг – один універсальний товар для декількох різних сегментів.	- відмінність від конкурентів тим що додаток подається у формі веб сервісу, що значно полегшує його використання - надання можливості визначати жанр музичного твору, чого нема ні в одного з конкурентів - збір можливостей з різних додатків аналогів в одну систему, що підвищує ефективність роботи та оптимізує використання часу

Реалізація стратегії диференціації вимагає, як правило, більш високих витрат. Проте успішна диференціація дозволяє компанії домогтись більшої рентабельності за рахунок того, що ринок готовий прийняти більш високу ціну (цінову премію бренду).

Найважливішими здібностями, які повинна мати компанія, що приймає цю стратегію, є з генерування маркетингових ноу-хау, здійснення продуктових новацій.

Таблиця 4.15. Визначення базової стратегії конкурентної поведінки

№	Чи є проект «першопроходьцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	ні	так	Так, нові затребуванні можливості по аналізу аудіо сигналу та частково інтерфейс користувача	Наслідування лідеру

Таблиця 4.16. Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто-спроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Легкість у використанні; можливість визначення важливих музичних ознак треку; проведення частотного та часового аналізу аудіо сигналу	Стратегія диференціації	Виконання додатку у формі веб сервісу; можливість визначення жанру; збір можливостей з різних додатків аналогів в одну систему	виконання аналізу музику онлайн без додаткових установок та на будь-якій системі; унікальна можливість визначати жанр музики якої нема в аналогах; одночасне поєднання можливостей аналізу частотних та часових характеристик музики й визначення таких характеристики як темп та жанр

4.5. Розроблення маркетингової програми statup проекту

Ефективний маркетинг починається з розглянутою, добре обізнаною маркетинговою стратегією. Хороша стратегія допоможе визначити бачення, місії та бізнес-цілі, а також описуються кроки, які необхідно зробити для досягнення цих цілей.

Маркетингова стратегія визначає загальний напрямок і цілі маркетингу, і, отже, відрізняється від плану маркетингу, в якому викладаються конкретні дії, які необхідно виконати для реалізації маркетингової стратегії.

Таблиця 4.17. Визначення ключових переваг концепції потенційного товару

№	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)*
1.	Максимальний обсяг моніторингу за	Покращення рівня життя жителям міста. Економія ресурсів	Більш захищена та більш дешевша мережа

	різними аспектами міста		
2.	Візуалізація даних моніторингу за містом.	Отримання характеристик які не можна побачити не озброєним оком	Додатки крос платформені та можуть бути у вигляді веб-додатку, десктопного додатку та мобільного додатку

Таблиця 4.18. Формування системи збуту

№	*Специфіка закупівельної поведінки цільових кліє- нтів*	*Функції збуту, які має виконувати постачаль-ник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1.	Звичайна	Збут власними силами	Збут через Інтерент	Реалізація торгівлі через Інтерент

Таблиця 4.19. Концепція маркетингових комунікацій

№	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуються цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*
1.	Вимогливість до дизайну інтерфейсу додатку та легкості у використання його функціональності	Інтернет-розсилки, соціальні мережі, розважальні сайти	Технології, програмна система виконання в формі веб додатку	Привернути увагу

Таким чином, отримано ринкову (маркетингову) програму, що включає в себе:

- Концепція товару орієнтована на максимальний моніторинг за різними аспектами міста, а також максимально зручне відображення даних, їх фільтрація користувачем і тому подібне

- Збут товару буде проводитись через мережу Інтерент.
- Ціноутворення продукту зробити мінімальним, в залежності від конкурентних товарів.
- Якісний та чіткий рекламний проект допоможе підняти попит у споживачів, а також приверне увагу людей, які шукали саме веб сервіс для проведення аналізу над музикальними сигналами.

Висновок до розділу 4

Таким чином, проект має у собі можливість ринкової комерціалізації, так як реалізує функціональність затребувану споживачами.

- Проект має певні переваги в порівнянні за конкурентами, а саме:
- Більш захищена та дешевша мережа розташування приладів
- Має 2 додатки, для адміністрації та жителя міста
- Додатки кросплатформені
- Унікальна система кешування робить додаток максимально швидким, а також дає можливість користуватися ним без підключення до мережі на основі останніх данних.

Але треба зауважити, що більшість конкурентів знаходяться дуже давно на ринку та мають закріплений бренд та додатки з багатою та розвинутою функціональністю, тому за стратегію розвитку було обрано *слідування лідеру.*

Зважуючи на вище зазначене, є доцільною подальша імплементація проекту та залучення інвестицій у розробку.

ВИСНОВКИ

Було проведено аналіз що таке система Smart City, Smart City моніторинг, а також що таке прогресивні веб додатки.

Вияснили що не існує максимального універсальної системи розумного міста, так як у кожного міста свої потреби. Отже за основу для створення системи було обрано місто Київ.

На основі цього було створено максимально гнучку архітектуру розташування приладів, яку з легкістю можна розширювати до нескінченності не боячись перенавантаження серверів та мережі. Було вибрано дротову систему, так як вона більше безпечна та більш дешевша, а також було вирішено питання з живленням.

За платформу для наших приладів ми вибрали IBM Watson Platform, так як це, на даний момент, найпотужніша платформа яка дає величезну кількість API, для роботи з приладами, їх діагностики та збору даних.

На основі даної системи та API платформи було створено додатки, для жителів міста, а також для адміністрації, так як вони мають різний рівень доступу до інформації.

Для клієнтського додатку була розроблена система кешування, яка дає змогу користувачу максимально ефективно використовувати додаток, а також дає можливість використовувати додаток в режимі офлайн, на основі останніх даних.

Для розробки додатків було використано передові технології, та були застосовані принципи реактивного програмування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Internet of things [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Internet_of_things.
2. How IOT work [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/.
3. Smart City [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Smart_city.
4. Smart City Model [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/iotforall/get-ahead-of-the-curve-with-this-model-for-planning-smart-cities-876b11f18f6e.
5. Прогресивні Веб Додатки [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Progressive_Web_Apps
6. Кешування в прогресивних веб додатках [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/.
7. Реактивне програмування [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Reactive_programming.
8. Реактивне розширення JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@kevalpatel2106/what-is-reactive-programming-da37c1611382.
9. Smart City Monitoring [Електронний ресурс] – Режим доступу до ресурсу: https://smartcity.pharosnavigator.com/static/content/en/677/Smart-City-Monitor.html.
10. Your Intro to Cloud IoT Core [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/iotforall/your-intro-to-cloud-iot-core-86e4741cc81f.
11. Watson IoT Platform [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/cloud/watson-iot-platform.

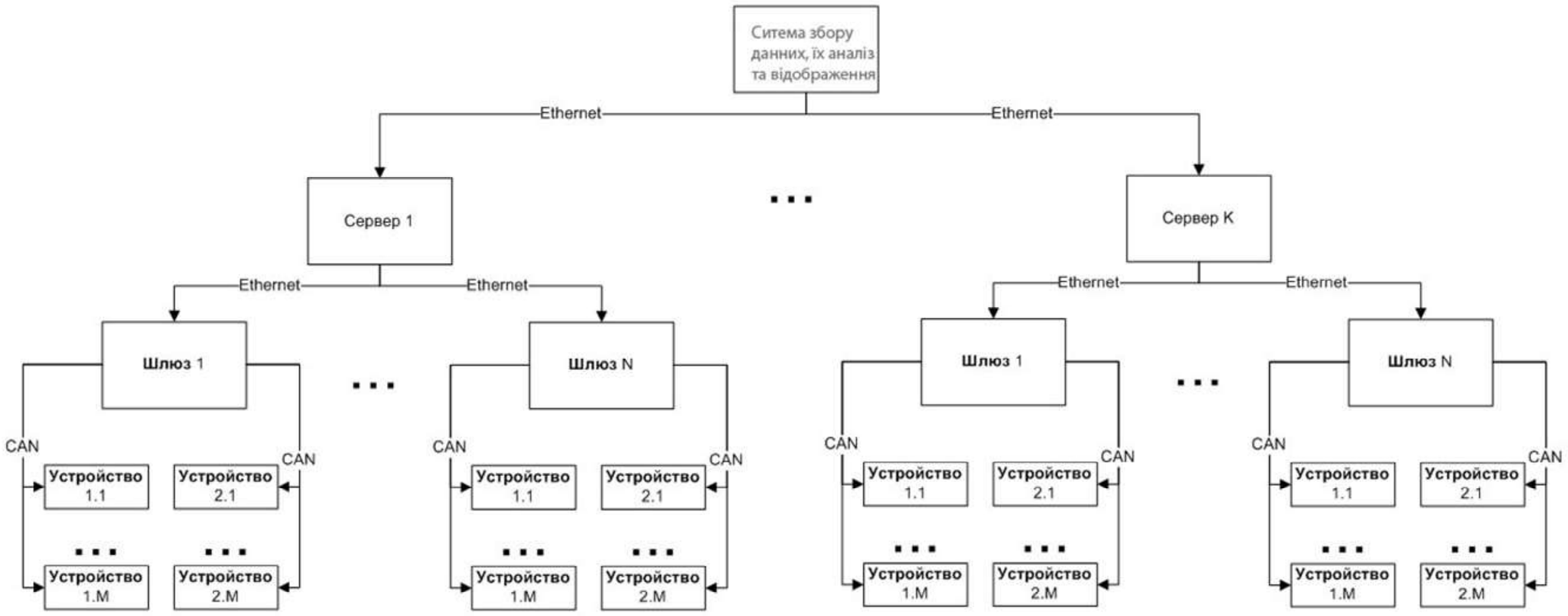
12. IBM IoT [Електронний ресурс] – Режим доступу до ресурсу: https://internetofthings.ibmcloud.com/?cm_mc_uid=46398370079315020939851&cm_mc_sid_50200000=83104801525878463513&cm_mc_sid_52640000=643822015258784635 21#/.
13. Angular Benefits [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/features.
14. NodeJS & Express [Електронний ресурс] – Режим доступу до ресурсу: http://expressjs.com/ru/.
15. RxJs [Електронний ресурс] – Режим доступу до ресурсу: http://reactivex.io/rxjs/.
16. CouchDb [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/CouchDB.
17. CouchDb documentation [Електронний ресурс] – Режим доступу до ресурсу: http://couchdb.apache.org/.
18. JSON: What It Is, How It Works, & How to Use It [Електронний ресурс] – Режим доступу до ресурсу: https://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/.
19. Electron benefits [Електронний ресурс] – Режим доступу до ресурсу: https://logicaladvantage.com/7-benefits-of-using-electron/.

ДОДАТКИ

ДОДАТОК А

Відомість дипломного проекту

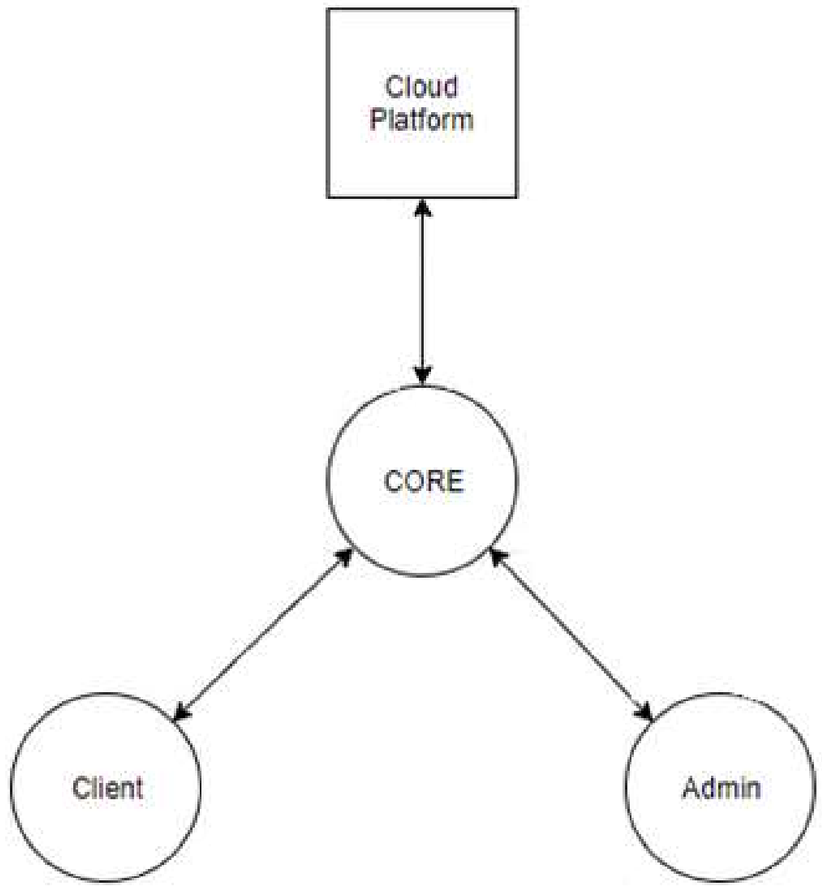
Архітектура мережі приладів



Демонстраційний плакат №1
до магістерської дисертації на тему
„Систестеми збору даних та прийняття рішень з пристроїв моніторингу в системі Smart City”

Розробив: Свічинський В.В.
Прийняв: Тимошин Ю.А.

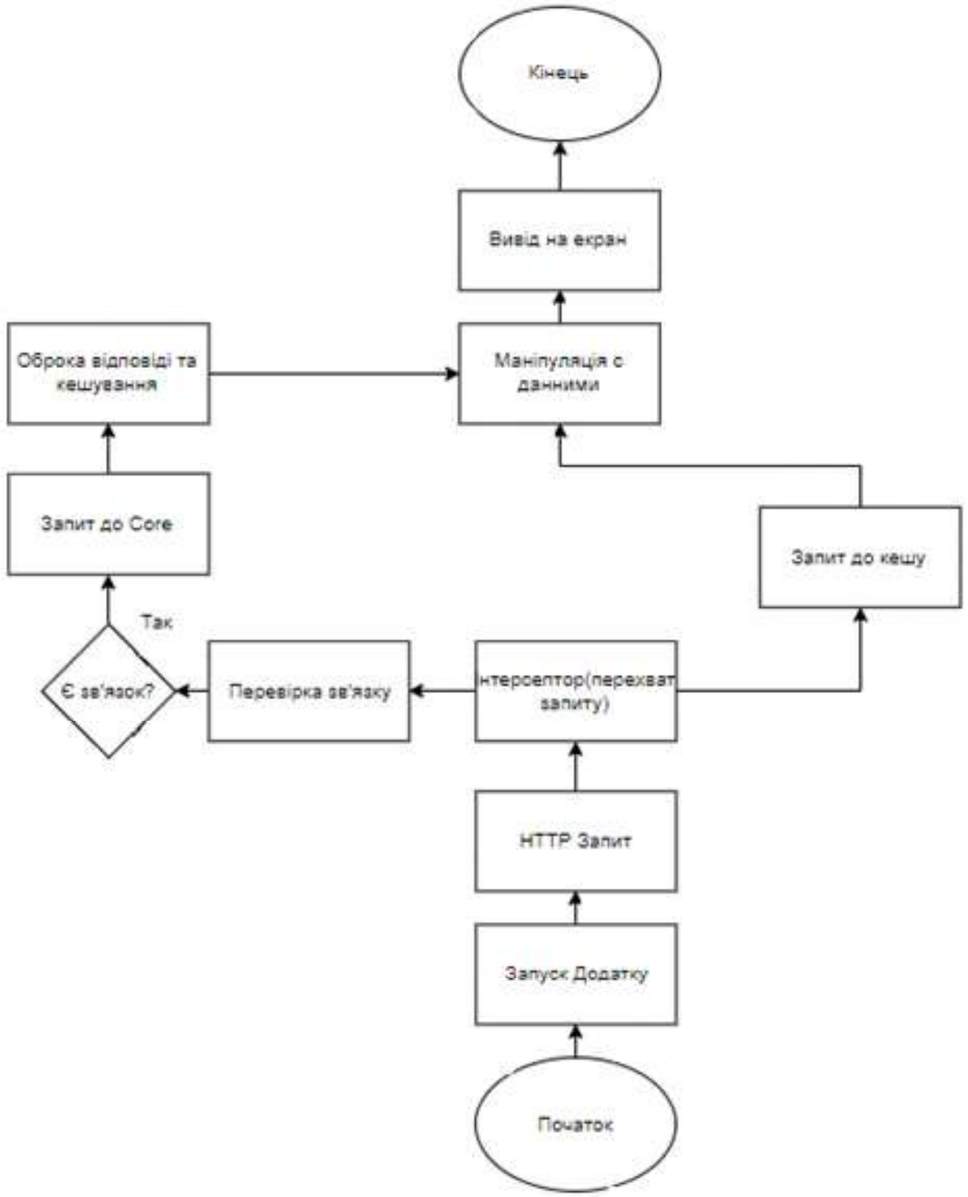
Архітектура взаємодії додатків



Демонстраційний плакат №2
до магістерської дисертації на тему
„Систестеми збору даних та прийняття рішень з пристроїв моніторингу в системі Smart City”

Розробив: Свічинський В.В.
Прийняв: Тимошин Ю.А.

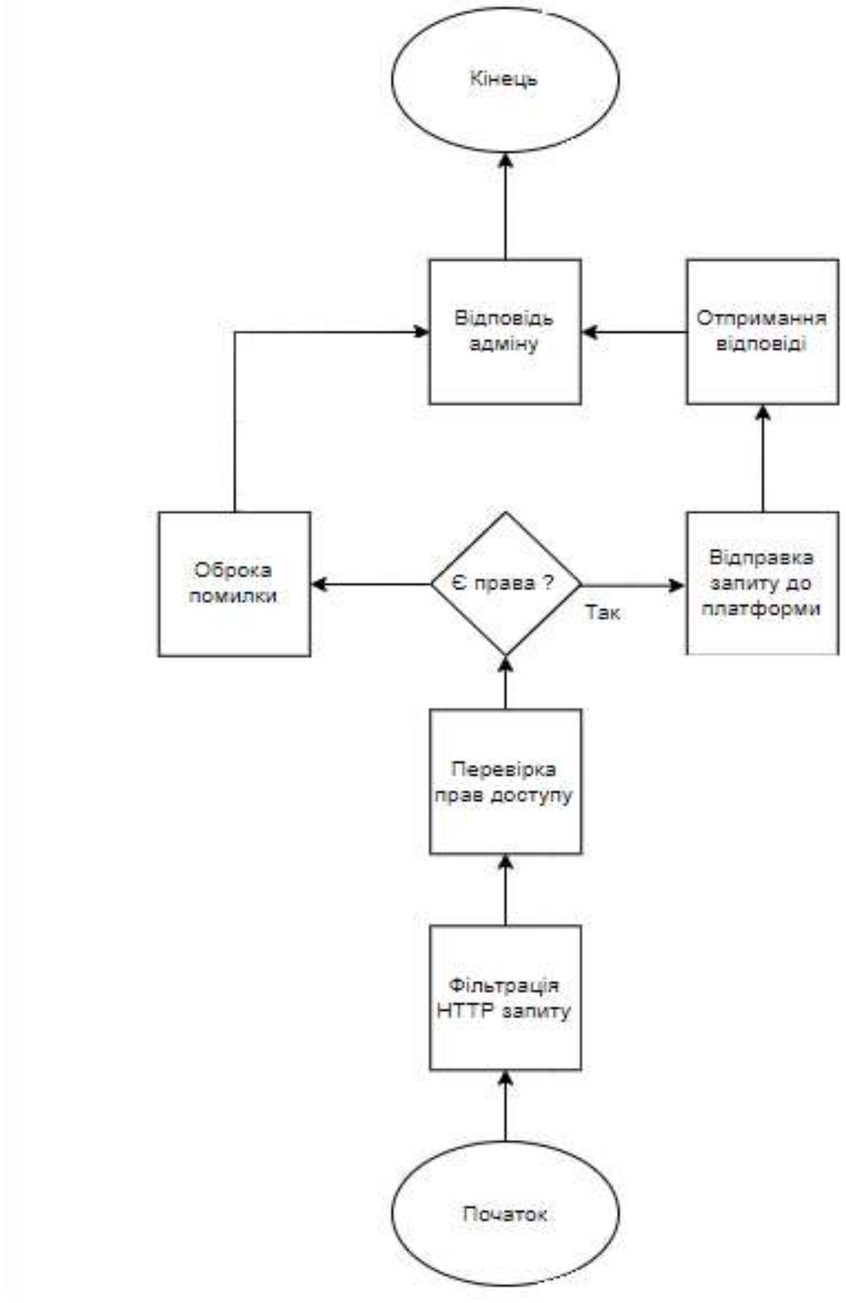
Архітектура додатку Client



Демонстраційний плакат №3
до магістерської дисертації на тему
„Систестеми збору даних та прийняття рішень з пристроїв моніторингу в системі Smart City”

Розробив: Свічинський В.В.
Прийняв: Тимошин Ю.А.

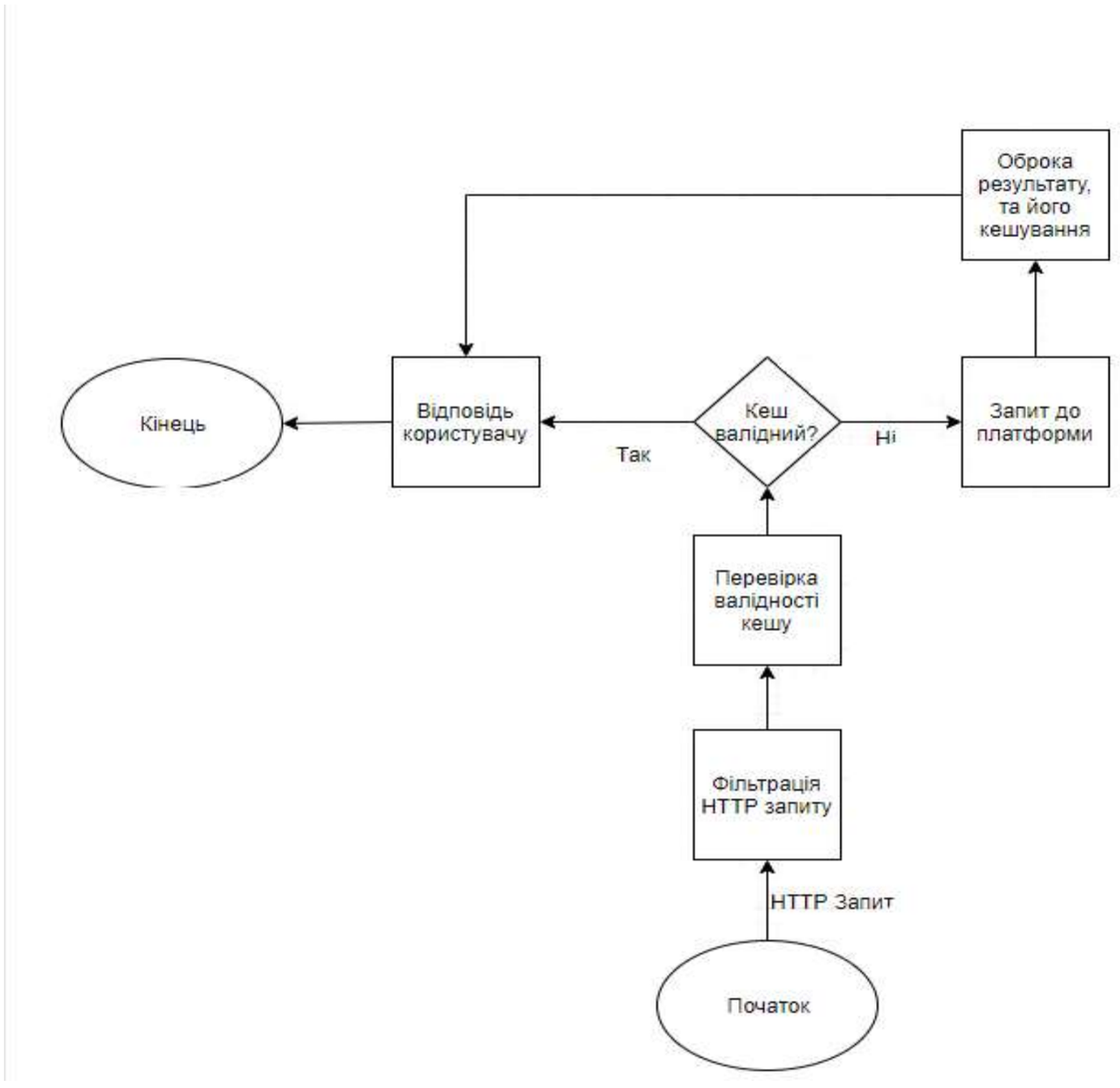
Обробка запиту від додатка Admin до додатка Core



Демонстраційний плакат №4
до магістерської дисертації на тему
„Систестеми збору даних та прийняття рішень з пристроїв моніторингу в системі Smart City”

Розробив: Свічинський В.В.
Прийняв: Тимошин Ю.А.

Обробка запиту від додатка Client до додатка Core



Демонстраційний плакат №5
до магістерської дисертації на тему
„Систестеми збору даних та прийняття рішень з пристроїв моніторингу в системі Smart City”

Розробив: Свічинський В.В.
Прийняв: Тимошин Ю.А.

Лістинг системи кешування для офлайн роботи з додатком

Ініціалізація Service Worker

```
@Injectable()
export class ServiceWorker {
    constructor(){
        const cacheName = 'monitoring data';
        const filesToCache = [];

        window.addEventListener('install', function(e) {
            console.log('[ServiceWorker] Install');
            e.waitUntil(
                caches.open(cacheName).then(function(cache) {
                    console.log('[ServiceWorker] Caching app shell');
                    return cache.addAll(filesToCache);
                })
            );
        });
    }
}
```

Метод для перезапису Кешу

```
window.addEventListener('activate', function(e) {
  console.log('[ServiceWorker] Activate');
  e.waitUntil(
    caches.keys().then(function(keyList) {
      return Promise.all(keyList.map(function(key) {
        if (key !== cacheName) {
          console.log('[ServiceWorker] Removing old cache', key);
          return caches.delete(key);
        }
      }));
    })
  );
  return window.clients.claim();
});
```

Подвійний ассинхронний запит до кешу та серверу

```
@Injectable()
export class FetchingData implements HttpInterceptor {
    constructor(private serviceWorker: ServiceWorker) {}
    intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        const newRequest = request.clone({
            headers: request.headers.set('settings', JSON.stringify(this.userSettings))
        })

        return next.handle(newRequest).combineLatest(this.serviceWorker.caches.claim());
    }
}
```

Демонстраційний плакат №6
до магістерської дисертації на тему
„Систестеми збору даних та прийняття рішень з пристроїв моніторингу в системі Smart City”

Розробив: Свічинський В.В.
Прийняв: Тимошин Ю.А.