ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття рівня вищої освіти «магістр»
(рівень вищої освіти)

тема: Дослідження і розробка систем керування
IoT мережею Розумного об'єкта на платформі хмарного провайдера

Research and development of IoT management systems for the Smart Object network on the platform of a cloud provider

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія .
(шифр і назва напряму підготовки, спеціальності)
Освітня програма «Комп'ютерна інженерія»
(назва освітньої програми)
Сомов Дмитро Сергійович
(прізвище, ім'я, по-батькові)

Керівник к.т.н, доц. Волощук Л.А.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент к.ф.-м.н., доц. Антоненко О.С.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент к.т.н., доц. Рудніченко М.Д.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____ 2024 р.

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

Захищено на засіданні ЕК № ___
протокол № __ від «_»__ ____ 2024 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ Алла КОБОЗЄВА
(підпис) (ім'я, прізвище)

Одеса – 2024

АНОТАЦІЯ

У кваліфікаційній роботі розробляється тема «Дослідження і розробка систем керування IoT мережею Розумного об’єкта на платформі хмарного провайдера».

Мета даної дипломної роботи - забезпечення економії електроенергії, комфортності освітленості шляхом розробки хмарної IoT системи моніторингу, аналітики та управління IoT мережею розумного об'єкта.

В результаті роботи проведено огляд і аналіз сучасних iot хмарних платформ та обрана для реалізації системи платформа хмарного провайдера amazon web services. Спроектовано архітектуру системи керування IoT мережею Розумного об’єкта. Розглянуті та обрані сервіси AWS IoT Core, AWS Quicksight для проектування та розробки хмарної частини системи керування IoT мережі розумного об’єкту.

Локальна частина iot системи спроектована в середовищі Cisco Packet Tracer та проемульована в середовищі Visual Studio Code. Розроблений сценарій керування iot мережею розумного об’єкту контролює включення світла по сигналу з датчика руху, та враховує дані прогнозу автоматичного освітлення за розкладом приходу людини.

ABSTRACT

The qualification work is developing the topic "Research and development of IoT control systems for the Smart Object network on the cloud provider platform".

The purpose of this thesis is to ensure energy savings, comfortable lighting by developing a cloud IoT system for monitoring, analytics and controlling the IoT network of the smart object.

As a result of the work, a review and analysis of modern IoT cloud platforms was conducted and the Amazon Web Services cloud provider platform was selected for the implementation of the system. The architecture of the IoT control system for the Smart Object network was designed. The AWS IoT Core and AWS Quicksight services were considered and selected for the design and development of the cloud part of the IoT control system for the smart object network.

The local part of the IoT system was designed in the Cisco Packet Tracer environment and emulated in the Visual Studio Code environment. The developed IoT control scenario for the smart object network controls the switching on of the light by the signal from the motion sensor, and takes into account the forecast data for automatic lighting according to the schedule of a person's arrival.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

AWS – Amazon Web Services.

GCP – Google Cloud Platform.

IDE – Integrated Development Environment.

IoT – Internet of Things.

MQTT – message queuing telemetry transport.

ВСТУП

Розвиток Інтернету речей (IoT) та хмарних технологій створює нові можливості для автоматизації, оптимізації ресурсів та підвищення комфорту в різних сферах життя, зокрема у створенні «розумних» будинків. Застосування хмарних платформ у системах IoT дозволяє ефективно збирати, обробляти та аналізувати дані, що робить можливим розробку інтелектуальних сценаріїв управління пристроями.

Тема моєї магістерської роботи Дослідження та розробка систем управління IoT мережею Розумного об'єкта на платформі хмарного провайдера. Актуальність роботи обумовлена необхідністю впровадження сучасних технологій для забезпечення підвищення комфорту.

Об'єкт дослідження: хмарні платформи та технології побудови систем IoT.

Предмет дослідження: хмарні платформи та технології розробки систем керування мережею розумного об'єкта на основі хмарних сервісів.

Мета дослідження: забезпечення економії електроенергії, комфортності освітленості шляхом розробки хмарної IoT системи моніторингу, аналітики та управління IoT мережею розумного об'єкта

Робота спрямована на розв’язання важливої задачі автоматизації систем управління розумним будинком на основі сучасних технологій IoT та хмарних сервісів.

Для досягнення мети необхідно вирішувати такі завдання:

1) Провести дослідження та аналіз сучасних тенденцій розвитку iot систем розумних об'єктів.
2) Провести дослідження можливостей служб та сервісів IoT платформ провідних хмарних провайдерів.
3) Сформувати постановку завдання розробки iot системи та мережі для обраного розумного об'єкта.

4) Провести проектування багаторівневої архітектори IoT системи управління розумним об'єктом.
5) Виконати проектування хмарної частини IoT системи із використанням сервісів хмарного провайдера.
6) Провести проектування алгоритму сценарію локальної частини IoT системи в середовищі Visual Studio Code .
7) Провести розробку IoT мережі розумного об'єкту.

1 ОГЛЯД І АНАЛІЗ ХМАРНИХ ПЛАТФОРМ ДЛЯ КЕРУВАННЯ IOT МЕРЕЖЕЮ

1.1 Введення в Інтернет речей (IoT)

Термін IoT, або Інтернет речей, відноситься до колективної мережі підключених пристроїв та технології, що полегшує зв'язок між пристроями та хмарою, а також між самими пристроями [1]. Завдяки появі недорогих комп'ютерних чіпів та високошвидкісної телекомунікації, тепер ми маємо мільярди пристроїв, підключених до Інтернету. Це означає, що повсякденні пристрої, такі як зубні щітки, пилососи, автомобілі та машини можуть використовувати датчики для збору даних та розумного реагування на дії користувачів.

Розвиток IoT робить можливим створення розумних будинків, розумних міст, промислових IoT-мереж і навіть автономних транспортних засобів.

Концепція домашньої автоматизації існує з кінця 1970-х років. Проте зі швидким розвитком технологій та появою інтелектуальних послуг наші очікування радикально змінилися. Сьогодні інтелектуальний будинок охоплює більше, ніж просто зручність — він втілює в собі всебічне розуміння того, яким має бути будинок та як мають надаватись та виходити його послуги. Системи домашньої автоматизації дозволяють кінцевим користувачам безперешкодно керувати та контролювати електроприлади, сприяючи ефективності, зручності та безпеці у своїх будинках [2].

Центр управління розумним будинком служить як сполучна ланка між усіма компонентами системи. Він дозволяє власнику будинку налаштувати сценарії освітлення, створити розклад увімкнення та вимикання світла, а також контролювати освітлення за допомогою голосових команд або мобільного додатка [3].

Розумні будинки набули широкого поширення, і сьогодні їх можна зустріти у більшості країн світу, включаючи й Україну. Сучасні рішення та

обладнання для розумних будинків дозволяють керувати зі свого телефону різними системами:

Система Розумний Дім дозволяє контролювати роботу освітлювальних приладів, встановлених як усередині будинку, так і за його межами [4].

Економне витрачання електроенергії. Можливе налаштування системи на автоматичне вимкнення всього світла за відсутності господарів. Це дозволяє заощадити на рахунках електроенергію тим, хто часто забуває відключити освітлення, залишаючи будинок;

Автоматизація. Увімкнення освітлення здійснюватиметься автоматично при вході господарів до будинку, а вимкнення під час їхнього догляду. Якщо кімната залишається порожньою протягом певного часу, система вимкне в ній світло.

1.2 Огляд хмарних платформ

Хмарні технології є важливим елементом екосистеми IoT [5]. Вони забезпечують надійне зберігання даних, обчислювальні потужності для аналізу даних і просту інтеграцію з інтерфейсами користувача. Без хмари керування великою кількістю IoT-пристроїв стає вкрай складним, оскільки обробка та аналіз даних з кожного пристрою потребують значних обчислювальних ресурсів.

Хмарні технології (або ж cloud технології) — це інфраструктура та послуги, що дозволяють обробляти та керувати даними через інтернет, використовуючи для цього віддалені сервери.

Сучасні хмарні технології забезпечують зручний та швидкий доступ до комп’ютерних ресурсів, таких як обчислювальні потужності чи сховища даних без необхідності мати фізичні сервери на місцях. Вони є справді гнучкими та масштабованими рішеннями як для звичайних користувачів, так і для бізнесу.

Хмарна платформа може обробляти величезний обсяг даних пристроїв, клієнтів, додатків, веб-сайтів та вживати заходів, щоб відповісти в режимі реального часу.

Вибір кращої платформи залежить від вимог компанії до обладнання, доступу в режимі реального часу, звітів користувача, бюджету. Розглянемо кілька найпопулярніших технологій.

1.2.1 Google Cloud Platform

Google Cloud Platform (GCP) — це сучасна хмарна платформа, яка надає компаніям інструменти і технології для вирішення найрізноманітніших бізнес-завдань [6]. Вона охоплює такі ключові області, як зберігання даних, машинне навчання, аналіз великих даних, контейнеризація додатків, а також управління безпекою та інфраструктурою.

Google надає багаторівневу безпечну інфраструктуру , яка допомагає у підвищенні операційної ефективності . Вона має інтегровані інтелектуальні механізми для ретельного обслуговування обладнання , різні готові рішення для структур, а також відстеження активів у процесі роботи.

Основні переваги Google полягають у постійній бізнес-аналітиці , у можливості безперервного навчання для будь-яких потреб . Це хмара також забезпечує підтримку широкого спектру вбудованих операційних систем та розвідує розташування програми. Організація, управління та обмін документами за допомогою цього ресурсу дуже прості [7]. Хмарна система Google працює зі усіма операційними системами. Загалом вона забезпечує гарні функціональні можливості та простоту використання. Однак , вартість даної операційної хмари досить висока .

1.2.2 Microsoft Azure IoT

Інтернет речей Azure (IoT) - це колекція керованих корпорацією Майкрософт хмарних служб, прикордонних компонентів і пакетів SDK, які дозволяють підключати, відстежувати та контролювати ресурси Інтернету речей у масштабі [8]. Простіше кажучи, рішення Інтернету речей складається з пристроїв Інтернету речей , що взаємодіють з хмарними службами.

Хмарна система від Microsoft призначена для різних потреб бізнесу, для створення як величезних додатків, так і невеликих стартапів . IoT Suite надає рішення для віддаленого моніторингу, прогнозного обслуговування, інтелектуальних просторів та підключених продуктів. Надає безкоштовний посібник зі створення додатків. Пропонує велику кількість функцій та можливостей, а також легко масштабується. Має наступні особливості:

Хмарна система має відкриту платформу для створення надійного додатку.

Вона може бути використана як новачками, так і експертами.

Є два рішення, з яких можна почати: SaaS або шаблони з відкритим вихідним кодом.

1.2.3 Amazon AWS IoT

Amazon Web Services (AWS) – це хмарна платформа від компанії Amazon, яка надає користувачам можливість використовувати різні віртуальні ресурси через Інтернет [9]. Зокрема, забезпечуючи доступ до віртуальних машин, зберігання даних та різних веб-сервісів, AWS є інструментом, що широко використовується розробниками, компаніями та урядовими органами для реалізації їх технологічних потреб.

AWS IoT Services - це хмарна платформа, яка працює з тисячами підключених пристроїв і здатна обробляти трильйони запитів одночасно.

AWS IoT Core суттєво полегшує підключення хмари до пристрою, робить цей процес зрозумілим. Це керований хмарний сервіс. AWS IoT Core не тільки дозволяє пристроям підключатися до хмари, але також взаємодіяти з іншими пристроями та хмарними додатками. Він забезпечує підтримку HTTP, полегшеного протоколу зв'язку та MQTT.

Програма забезпечує безпечний доступ до будь-яких пристроям, а також безкоштовну пробну версію на певний період роботи. Навчальні посібники надаються всім бажаючим як навчальний матеріал. AWS IoT надає гарні можливості інтеграції з іншими сервісами.

1.3 Сервіси AWS IoT

AWS IoT Core: Це основна послуга, яка дозволяє підключати пристрої до хмари. Вона підтримує протоколи MQTT, HTTP та WebSockets, забезпечуючи надійний та безпечний зв'язок між пристроями та хмарою.

AWS IoT Core може обробляти величезну кількість повідомлень. надійна та безпечна платформа для маршрутизації повідомлень на кінцеві точки AWS та інші пристрої . додатки до AWS IoT можуть взаємодіяти, навіть якщо вони не підключені до Інтернету Можна використовувати інші сервіси AWS, такі як AWS Lambda , Amazon Kinesis , Amazon QuickSight .

AWS Lambda: середа розробки програмного забезпечення , у якій розробники можуть писати та редагувати код, об'єднувати проекти з інших сервісів AWS (включаючи IoT) і запускати написаний код. Це платформа для безперервного розгортання - розробники можуть випускати код для служби по одному, уникаючи технічного боргу та безладдя помилок.

QuickSight AWS: сервіс створює аналітику з даних IoT. Він відповідає за збирання , обробку, зберігання, аналіз машинного навчання в реальному часу та створення звітів на основі коду.

Шлюз пристроїв: всі пристрої в Amazon Web Services для IoT підключені до шлюзу. Служба відповідає за підтримку з'єднання між пристроями та

сервером навіть в умовах низька затримка. Шлюз пристроїв – це шлюз для використання платформи AWS IoT .

Брокер повідомлень : ця послуга дозволяє підключеним пристроям обмінюватися повідомленнями один з одним та з сервером додатків. Цей інструмент відповідає за зв'язність - він може обробляти, зберігати та систематизувати тисячі повідомлень одночасно .

Механізм правил: цей інструмент встановлює обмеження та забезпечує дотримання правил використання даних . Правило визначає , як пристрої обробляють дані . Наприклад , ви можете вказати поріг і встановити значення за промовчанням для значень вище порога. Правила AWS IoT ініціюють виконання певних функцій AWS Lambda , пов'язуючи оновлення обладнання з реакціями програмного забезпечення [10].

1.4 Постановка завдання

В результаті проведеного дослідження можуть бути сформовані наступні задачі:

1) Проектування розподіленої архітектури iot системи розумного об’єкту.
2) Проектування алгоритму сценарію автоматичного освітлення у iot мережі розумного об’єкту.
3) Побудова програмної моделі локальної мережи та підсистеми iot системи розумного об’єкту у середовищі Visual Studio Code.
4) Використання компонента Quicksight хмарної платформи aws для створення сервісу прогнозу часу включення світла на розумному об’єкті відповідно сценарію.
5) Передача прогнозу в локальну мережу розумного об’єкта з IoT платформи Amazon.
6) Дослідження можливостей реалізації iot системи розумного об’єкту на платформі aws.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА СЦЕНАРІЮ IoT СИСТЕМИ

2.1 Архітектура IoT системи

Архітектура iot системи наведена на рисунку 2.1

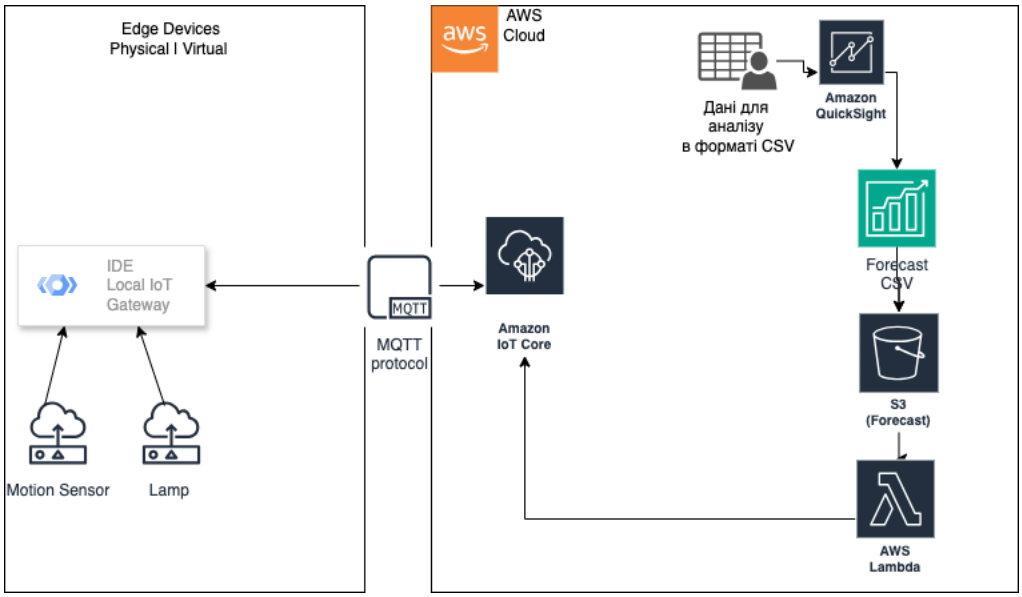

Рисунок 2.1 – Схема архітектури IoT системи розумного об’єкту.

1) Physical / Virtual Layer - фізичний рівень

Цей рівень представляє два типи операцій - збір інформації (Датчики) та здійснення механічної роботи (Виконавчі механізми)

Датчик світла та Датчик руху

2) Local Network Layer – рівень периферійної комунікації.

На локальному рівні знаходиться кінцевий пристрій мікроконтролер, який, збирає з пристроїв інформацію і виконує дії сценарію, вимикати та вмикати лампу вручну, а також при появі руху, і якщо протягом певного часу нового руху не було зафіксовано лампу вимкнутись.

3) Gateway Layer - рівень шлюзу

Підключення мікроконтролеру локальної мережі до глобальної мережі інтернет до хмарних серверів AWS . Рівень спілкування хмарної платформи та периферії.

4) Middleware Layer - рівень усередині серверного зв'язку

Рівень спілкування хмарних компонентів aws між собою

5) Big Data and Analytic Layer – рівень аналітики

Рівень використання аналітики на хмарній платформі AWS . Відбувається обробка зібраної інформації На рівні аналітики обробляється зібрана інформація про активність датчика руху та часу роботи освітлювального пристрою. Результатом аналізу є прогноз часу приходу людини до будинку. І визначається дані про необхідність включення освітлювальних приладів у певний час.

Для реалізації IoT мережі відповідно 1,2,3 рівнів архітектури необхідні наступні компоненти:

- Датчик руху (PIR Motion Sensor):
  Виявлення присутності людини в зоні дії.
  Використовується для активації освітлення або інших автоматизованих сценаріїв.
- Розумні лампи (Smart Lights):
  Регулювання яскравості та кольору освітлення.
  Вмикання/вимикання за командами з IoT платформи.
- Мікроконтролери (ESP32, Raspberry Pi):
  Відповідають за локальне збирання даних з датчиків, виконання автоматизації, а також передачу даних до хмари.
  Підтримують MQTT для зв’язку з хмарними сервісами.
- Wi-Fi/Bluetooth модулі:
  Для підключення до локальної мережі чи прямого спілкування між пристроями.

Працездатність локальної частини iot системи було промодельовано в середовищі Cisco Packet Tracer. Модель iot мережі наведена на рисунку 2.2.

В iot мережі мікроконтролер збирає інформацію с датчику руху для виявлення активності та з лампи для показання роботи лампи та її рівня яскравості. Якщо мікроконтролер отримає з датчику руху сигнал рух, то він ввімкне лампу.

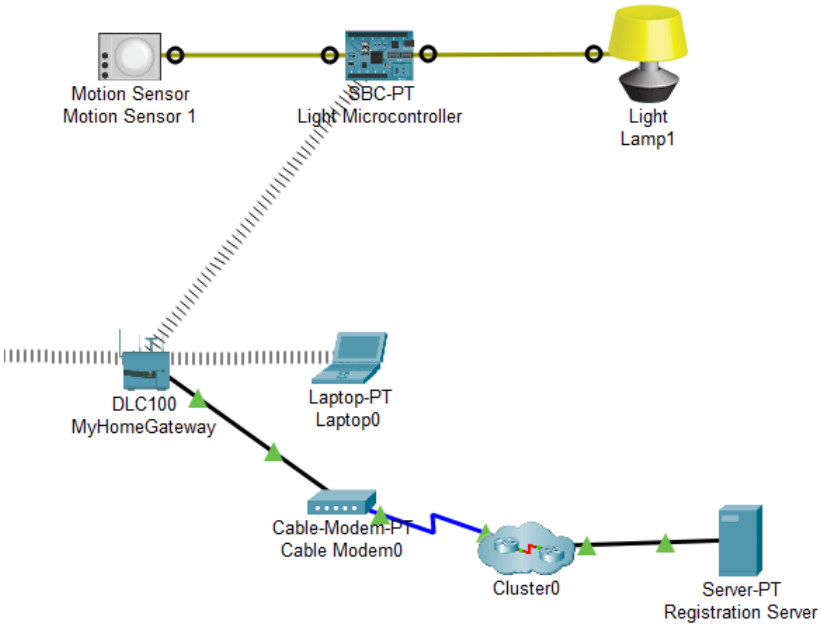

Рисунок 2.2 – Схема моделювання iot мережі

Для реалізації системи можуть бути наступними рівнями:

1) Шар сприйняття (також називається фізичним чи віртуальним рівнем). Це основа IoT , що складається з віртуальних компонентів, таких як датчик руху та лампа. На цьому рівні в моїй системі відбувається збір даних із датчиків IDE системою.

2) Рівень обробки відповідає за зберігання, аналіз та перетворення даних.

Локальна одиниця обробки даних, що отримує інформацію від датчиків руху, порівнює з розкладом та виводить відповідні команди на контролери освітлення. Рівень обробки також дозволяє системі обробляти дані та реагувати в режимі реального часу

3) Рівень шлюзу.

Спілкування між локальною IDE платформою за протоколом MQTT і хмарною системою через хмарний компонент AWS IoT Core . А також передавання прогнозу через компонент AWS Lambda .

4) Рівень спілкування між компонентами AWS. Компонент AWS S3 .
5) Рівень інтерфейсу користувача.

Інтерфейс через AWS компонент Amplify для перегляду історії даних і управління освітленням вручну.

6) Рівень аналітики на платформі AWS

Це програмний компонент Quicksight , що відповідає за збір та обробку історичних даних про час приходу та активність датчиків.

2.2 Проектування алгоритму сценарію автоматичного освітлення за розкладом приходу людини

Проведемо проектування сценарію. Система розумного освітлення керує освітленням на основі розкладу приходу людини та даних від датчиків руху. У певний час, коли людина зазвичай повертається додому (наприклад після роботи), система активує освітлення на потрібних ділянках, навіть якщо датчики руху ще не зафіксували активність. Якщо протягом певного часу руху не було виявлено, система вимикає світло для економії енергії. Дані про час спрацьовування датчиків та включення світла використовуються для аналізу та покращення точності автоматизації.

Блок схема алгоритму сценарію наведена на рисунку 2.2.

Реалізація:

1) Збір даних:
   - Датчики руху фіксують, коли людина приходить додому та переміщається по кімнатах.
   - Час приходу фіксується у розкладі, наприклад, з 18:00 до 19:00 – типовий час повернення людини з роботи.

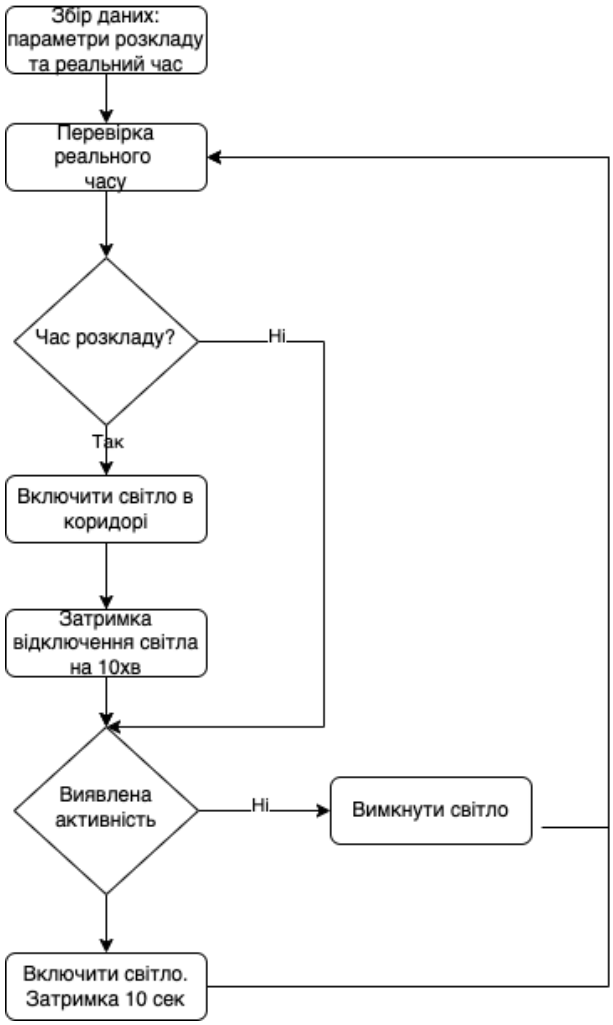

Рисунок 2.2 – Блок схема алгоритму сценарію

2) Алгоритм сценарію:
   - У передбачуваний час повернення людини (наприклад, о 18:00), система включає світло у коридорі, навіть якщо датчик руху поки не спрацював. Це дозволяє створити комфортне освітлення під час входу.
   - Якщо датчик руху фіксує активність у передпокої, освітлення продовжує працювати.
   - Якщо датчик руху не фіксує активності протягом 15 хвилин (наприклад, людина затрималася і не прийшла додому вчасно), світло автоматично вимикається для економії енергії.

- Система збирає дані про фактичний час приходу та активність датчиків руху, щоб на основі аналізу коригувати розклад освітлення та зменшувати енерговитрати.

3) Аналіз даних:

- Система аналізує, наскільки точно передбачено час повернення людини і як часто світло вмикається в порожньому будинку, якщо датчики руху не фіксують активність. Наприклад, якщо протягом тижня світло вмикалося в очікуванні, але людина приходила пізніше на 30 хвилин, система запропонує скоригувати розклад.

Для реалізації керування IoT мережею Розумного об’єкта використовуються наступні служби та сервіси iot платформи AWS:

AWS IoT Core:

Всі пристрої, такі як датчики руху, розумні лампи та контролери освітлення, підключені до AWS IoT Core для взаємодії та передачі даних. AWS IoT Core забезпечує зв'язок пристроїв із хмарою та надсилання команд на пристрої для увімкнення/вимкнення світла.

Amazon QuickSight:

QuickSight використовується для візуалізації зібраних даних і допомагає користувачеві побачити, як часто вмикалося світло без необхідності, та запропонувати оптимальні зміни у розкладі для більш ефективного використання ресурсів.

Приклад аналізу:

Аналіз розкладу приходу людини: За тиждень система фіксувала, що людина зазвичай повертається додому з 18.15 до 18.30, хоча розклад був налаштований на 18.00. Світло включалося раніше, ніж це було необхідно, що призвело до непотрібного споживання енергії.

Аналіз даних датчиків руху: Протягом перших 15 хвилин після включення світла в передпокої рух фіксувалося в 80% випадків, але в 20% випадків світло працювало вхолосту, тому що людина не приходила додому вчасно.

Оптимізація:

Коригування розкладу: На основі даних за тиждень система пропонує змінити час увімкнення світла з 17:50 на 18:10, що скоротить непотрібне споживання енергії, зберігаючи комфорт для користувача.

Розумне очікування: Якщо датчик руху не спрацьовує протягом перших 5 хвилин після ввімкнення світла, система запропонує вмикати світло на 50% яскравості, щоб заощаджувати енергію, доки рух не буде зафіксовано.

3 РОЗРОБКА IoT СИСТЕМИ КЕРУВАННЯ IoT МЕРЕЖЕЮ РОЗУМНОГО ОБ'ЄКТУ

3.1 Розробка підсистеми IoT мережі розумного об'єкту в середовищі Visual Studio Code

Реалізація проекту локальної частини виконано віртуально в середовищі Visual Studio Code, де виконання функцій мікроконтролеру емульується класом AutomationSystem, а робота датчику руху та лампа класами MotionSensor і SmartLight (рисунок 3.1).

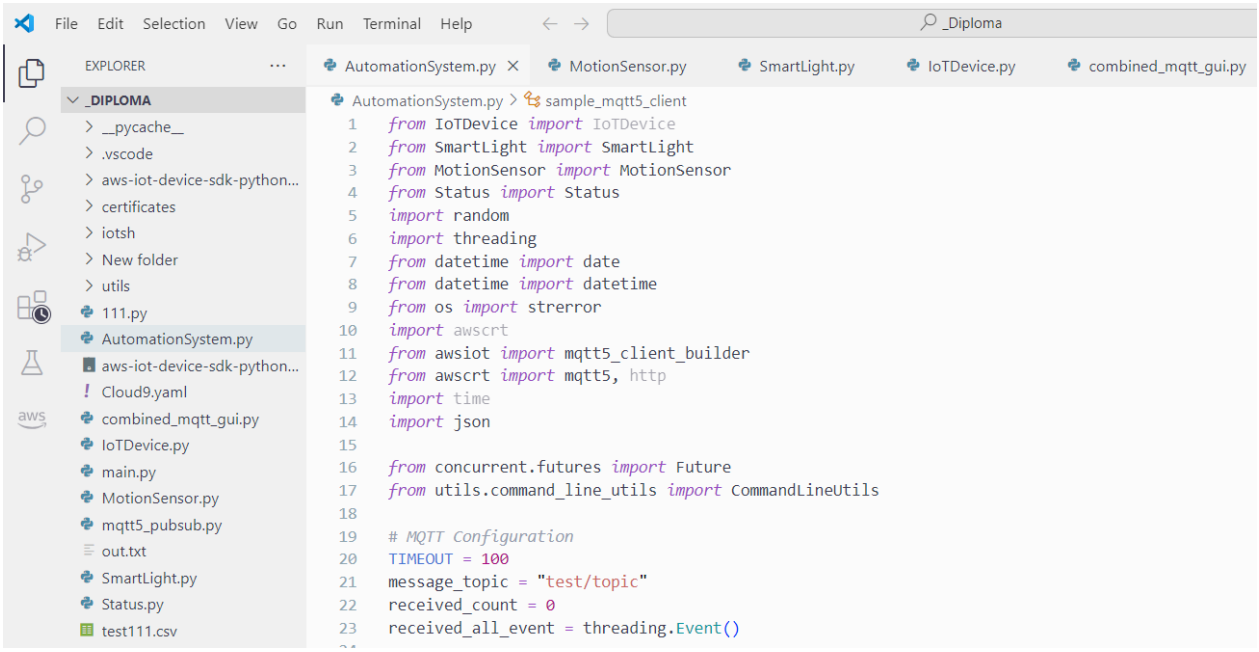

Рисунок 3.1 – Емуляція підсистеми середовищі Visual Studio Code.

В моделі iot підсистемі створено два пристрої: датчик руху та лампа.

Реалізація пристрою лампи здійснюється через створений клас SmartLight . Код класу наведено в додатку А. У даному класі Розумна лампа (SmartLight):

– Реалізує керування яскравістю світла та його станом (ввімкнено/вимкнено) через клас.
– Підтримує функції встановлення яскравості, увімкнення/вимкнення світла, а також поступового зменшення яскравості (dimming).

Реалізація пристрою датчика руху здійснюється через створений клас MotionSensor. Код класу наведено в додатку Б. У даному класі Датчик Руху (MotionSensor):

- Датчик руху включає свій статус (Status.On), якщо виявлено рух через метод detect_motion .
- Якщо пристрій перебуває в стані Status.Off , він не передаватиме інформацію про рух, навіть якщо рух зафіксований.
- Використовується для автоматичного керування іншими пристроями (наприклад, увімкнення світла при виявленні руху).

В підсистемі також реалізовані наступні класи.

Клас Status (Стани пристроїв):

Використовується перелік Status для представлення двох станів:

- On (включено)
- Off (вимкнено).

Повний код класу в додатку В.

Клас IoTDevice - це базовий клас для всіх пристроїв в IoT -системі, який визначає основні властивості та поведінку. Він є основою для інших класів, таких як SmartLight та MotionSensor . Повний код класу в додатку Г.

Клас AutomationSystem (Автоматизація) Повний код класу в додатку Д.:

Керує списком пристроїв (наприклад, світильниками та датчиками руху) та виконує автоматичні завдання (AutomationSystem):

- Увімкнення світла для виявлення руху.
- Автоматичне вимикання світла через заданий час без активності.
- Збирає дані з датчиків та зберігає їх для аналізу.
- Імітатор датчиків та пристроїв

Компонент зв'язку через MQTT:

- Система інтегрована з протоколом MQTT для управління пристроями та отримання даних про стан (AutomationSystem) через Mqtt Client, який передає інформацію про стан лампи через topic передачи даних, та отримає дані розкладу для автоматизації сценарію через другий topic прийняття даних.
- Підтримується підписка на тему управління лампою та обробка команд, що надходять із зовнішніх джерел.

Основні функції системи:

- Автоматизація увімкнення/вимкнення світла на основі даних датчиків руху.
- Дистанційне керування через MQTT: надсилання та отримання повідомлень про стан пристроїв.
- Збір та зберігання даних з пристроїв (яскравість світла, дані датчиків руху) для подальшого аналізу.
- Підтримка функцій енергозбереження (наприклад, автоматичне вимкнення світла після заданого часу).

3.2 Використання аналітичного сервісу на AWS Quicksight для створення прогнозу включення світла.

3.2.1 Огляд можливостей сервісу Amazon QuickSight

Amazon QuickSight – це комплексний сервіс бізнес-аналітики, за допомогою якого кожному користувачеві організації буде простіше створювати візуалізації, виконувати спонтанний аналіз та швидко отримувати аналітичні результати на основі своїх даних у будь-який час та з будь-якого пристрою. За допомогою QuickSight всі користувачі можуть виконувати різні

аналітичні завдання, використовуючи те саме джерело інформації, за допомогою сучасних інтерактивних інформаційних панелей.

Файли даних у форматах XLSX, CSV, TSV, CLF, XLF можна завантажувати безпосередньо на сайті QuickSight .

Виконання аналізу. QuickSight ефективно виконує пошук даних у найпоширеніших репозиторіях даних AWS на обліковому записі AWS. Вказуємо сервісу QuickSight які з виявлених джерел даних слід використовувати. Потім вибираємо таблицю та приступаємо до аналізу даних. Щоб створити візуалізацію, спочатку вибираємо поля даних, які необхідно проаналізувати, або переміщуємо поля безпосередньо в область візуалізації. Також можна використовувати поєднання обох цих способів. Сервіс QuickSight автоматично вибере найбільш підходящу візуалізацію на основі вказаних даних [11].

3.2.2 Створення прогнозу включення світла на AWS Quicksight

Створення прогнозу для сценарію автоматичного освітлення за розкладом приходу людини з використанням датчиків руху виконується наступними етапами:

1) Вибираємо dataset для аналізу, файл CSV з часом роботи iot пристроїв в локальній мережі (рис. 3.2).

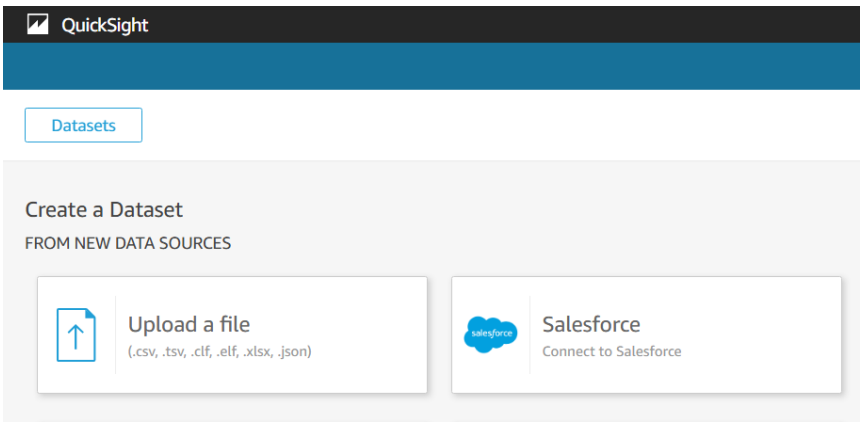

Рисунок 3.2 – Додання datasource

2) Створюємо аналіз на основі даних. Для цього заповнюємо деякі поля даних, які використаємо і далі вибираємо візуалізацію (рисунок 3.3).

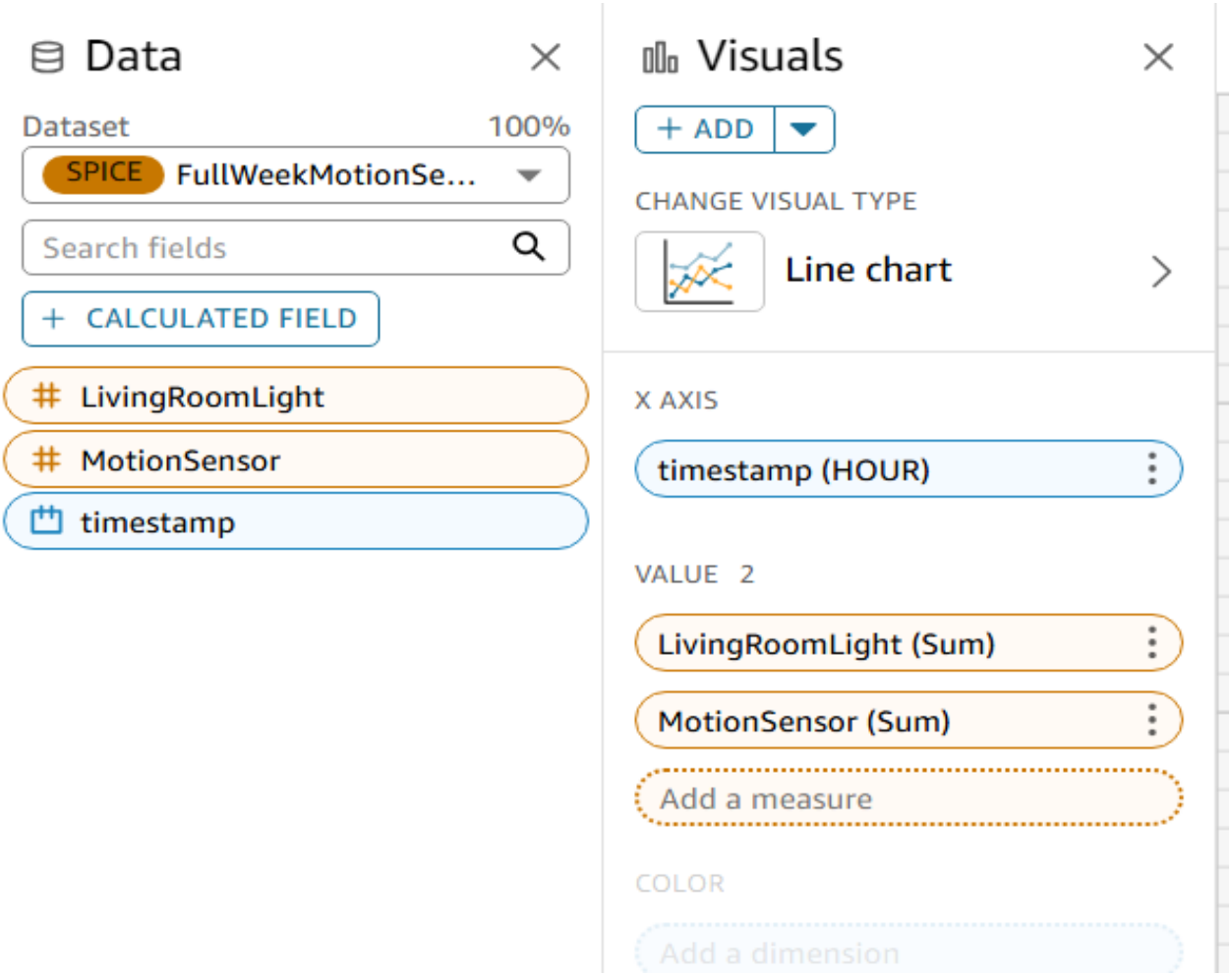

Рисунок 3.3 – Налаштування параметрів прогнозу для сценарію автоматизації включення світла

3) Відповідно обраним даним запускаємо процес аналізу в сервісі aws Quicksight. В результаті одержуємо передбачені дати з можливих часів для запуску дії сценарію автоматизації включення світла (рисунок 3.4).

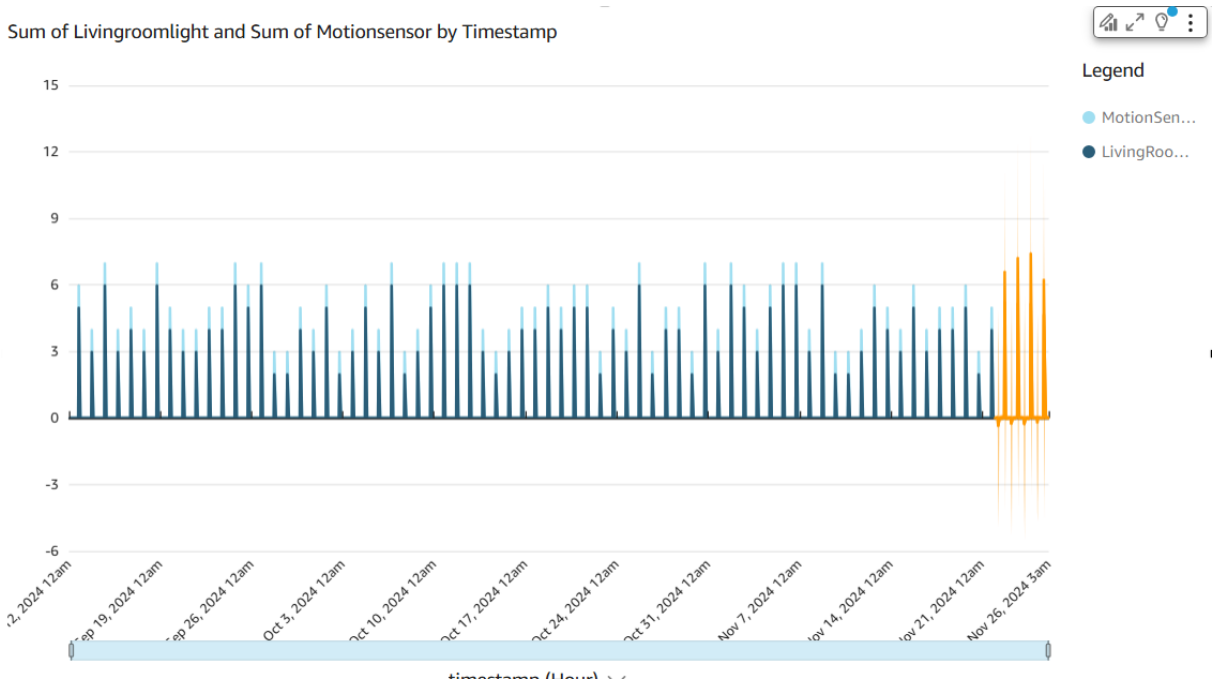

Рисунок 3.4 – Результат аналізу даних в сервісі aws Quicksight

3.3 Дослідження можливостей реалізації iot системи розумного об’єкту на платформі aws.

Проаналізувавши свій проект, можу зробити подальші припущення про перспективу мого проекту в інших сценаріях.

Перспективи використання AWS IoT для розумного об’єкту виходять за межі базових функцій. За допомогою аналітичних можливостей AWS можна не лише налаштовувати сценарії на основі зібраних даних, а й прогнозувати потреби користувача, адаптувати роботу систем під його звички та навіть пропонувати додаткові автоматизації на основі аналізу даних.

Як подальша перспектива використання результатів магістерської роботи може бути розвиток та поліпшення функціональних можливостей сценаріїв моніторингу та управління освітленістю, розширення сфер застосування в різних розумних об'єктах, у тому числі підприємствах, організаціях, різних місцях колективного відвідування.

Після проведеної роботи можна проаналізувати яка реалізація системи краща та зручніша, локальна або хмарна.

Якщо поставити запитання яка реалізація iot системи локально або на хмарі краще можна провести аналіз головних аспектів, а саме питання вартості, швидкості реагування та простоти реалізації.

– Вартість.

Наявність готової інфраструктури хмари знижує витрати на локальне апаратне забезпечення. Але Хмарні платформи мають модель оплати "pay-as-you-go", де витрати залежать від обсягу даних, обробки та зберігання. AWS IoT Core, наприклад, стягує плату за кожне повідомлення MQTT, а також за зберігання в Amazon S3 та обробку в AWS Lambda [12].

Витрати локальну реалізацію обмежуються обладнанням (ESP32, Raspberry Pi, сенсори) і є разовими. Відсутність постійних витрат на обчислення та зберігання в хмарі робить її дешевшою для малих проектів

– Швидкість.

Локальна система реагує миттєво, оскільки дані обробляються безпосередньо на мікроконтролері. Типові затримки в ESP32 складають близько 1-10 мс, оскільки це лише внутрішній процесорний час [13].

На хмарі процес включає кілька етапів: передача даних до хмари, обробка в хмарі та передача команди назад. Загальний час залежить від швидкості мережі (типово 50-500 мс) і часу обробки в хмарі (10-100 мс) [14].

– Установка та масштабування.

При установці на хмарі вже э готові сервіси (наприклад, AWS IoT Core) які надають прості в налаштуванні API для підключення пристроїв, обробки даних і керування. Для користувача інтерфейс може бути простішим. Додавання нових пристроїв вимагає лише реєстрації в хмарі [15].

При локальній реалізації для кожного нового пристрою потрібно доопрацьовувати локальне програмне забезпечення. Налаштування складніше через необхідність інтеграції вручну [16].

Основуючись на розглянутих аргументах можна сказати, що в питанні ціни хмарна реалізація є дорогим рішенням, у питанні швидкості є різниця, але вона не така велика для користувача та в питанні встановлення та

масштабування хмарні рішення простіші в масштабуванні та інтеграції але локальна система легша у початковій установці для невеликих проектів. Тому все ж таки локальна реалізація є більш дешевшим, але складнішим рішенням.

Отже найкращим варіантом буде змішана форма iot системи, як в моїй роботі.

ВИСНОВКИ

В результаті роботи було проведено дослідження можливостей служб та сервісів iot платформ хмарних провайдерів. Було проведено проектування багаторівневої архітектури iot системи управління розумним об’єктом. Також проведено проектування алгоритму сценарію локальної частини iot системи в середовищі visual studio code, та було спроектовано та промодельовано iot мережу розумного об’єкту. Спроектовано хмарну частину iot системи із використанням хмарних сервісів та за допомогою одного з хмарних сервісів отримано результат аналізу у вигляді файла з прогнозом для автоматизації сценарію автоматизації включення світла в локальній iot мережі розумного об’єкту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is IoT [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is/iot/?nc1=h_ls
2. The Art of Smart Living [Електронний ресурс] – Режим доступу: https://medium.com/@kavisha653/the-art-of-smart-living-python-and-iot-for-seamless-home-automation-194dec6ee70f
3. Система управління освітленням в Розумному домі [Електронний ресурс] – Режим доступу https://smartsys.team/ru/blog/light-smart-home/
4. Особливості освітлення в системах «розумний дім» [Електронний ресурс] – Режим доступу: https://evrodom.kh.ua/blog/osobennosti-osveshcheniya-v-sistemah-umnyy-dom
5. Що таке хмарні технології [Електронний ресурс] – Режим доступу: https://wezom.com.ua/ua/blog/scho-take-hmarni-tehnologiyi-viznachennya-vazhlivist-ta-zastosuvannya
6. Google Cloud Platform [Електронний ресурс] – Режим доступу: https://cloud.google.com/
7. Що таке Google Cloud? [Електронний ресурс] – Режим доступу: https://wiseit.com.ua/shho-take-google-cloud-platform/
8. What is Azure Internet of Things (IoT)? [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/azure/iot/iot-introduction
9. What is AWS [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is-aws/?nc1=h_ls
10. AWS IoT [Електронний ресурс] – Режим доступу: https://flexa.cloud/en/o-que-e-aws-iot/
11. What is Amazon QuickSight? [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/quicksight/latest/user/welcome.html
12. AWS IoT Core Pricing [Електронний ресурс] – Режим доступу: https://aws.amazon.com/iot-core/pricing/?nc1=h_ls

13. Espressif Technical Documents [Електронний ресурс] – Режим доступу: https://www.espressif.com/en/support/documents/technical-documents
14. AWS IoT Core [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/iot/latest/developerguide/
15. AWS IoT Core Documentation [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/iot/
16. MQTT Guide [Електронний ресурс] – Режим доступу: https://mqtt.org/

ДОДАТОК А КОД КЛАСУ SMARTLIGHT

```
from IoTDevice import IoTDevice
from Status import Status
import time
class SmartLight(IoTDevice):
    def __init__(self, id, brightness):
        super().__init__(id)
        self.__brightness = brightness

    def set_status(self, status):
        if status == Status.On and self.__brightness == 0:
            self.__brightness = 0
        elif status == Status.Off and self.__brightness > 0:
            self.__brightness = 0
        self._status = status

    def get_brightness(self):
        return self.__brightness

    def set_brightness(self, brightness):
        if self._status == Status.Off and brightness > 0:
            self._status = Status.On
        elif self._status == Status.On and brightness == 0:
            self._status = Status.Off

        self.__brightness = brightness

    def gradual_dimming(self, steps, duration, delay, step_size):
        for _ in range(steps):
            new_brightness = self.__brightness - step_size
            if(new_brightness >= 0):
                self.__brightness = new_brightness
            time.sleep(delay)
        self.__brightness = 0
        self._status = Status.Off
```

ДОДАТОК Б КОД КЛАСУ MOTIONSENSOR

```
from IoTDevice import IoTDevice
from Status import Status
import random

class MotionSensor(IoTDevice):
    def __init__(self, id, security_status):
        super().__init__(id)
        self.__security_status = security_status
        self.__motion = False

    def get_security_status(self):
        return self.__security_status

    def set_security_status(self, security_status):
        self.__security_status = security_status

    def get_motion(self):
        if self._status == Status.Off:
            return False

        return self.__motion

    def detect_motion(self, motion):
        if self._status == Status.Off:
            self._status = Status.On

        self.__motion = motion
```

ДОДАТОК В КОД КЛАСУ STATUS

```
import enum

class Status(enum.Enum):
    On = 0
    Off = 1
```

ДОДАТОК Г КОД КЛАСУ IOTDEVICES

```
from Status import Status

class IoTDevice:
    def __init__(self, id):
        self._id = id
        self._status = Status.Off

    def get_status(self):
        return self._status

    def set_status(self, status):
        self._status = status

    def get_id(self):
        return self._id
```

ДОДАТОК Д КОД КЛАСУ AUTOMATIONSYSTEM

```
from IoTDevice import IoTDevice
from SmartLight import SmartLight
from MotionSensor import MotionSensor
from Status import Status
import random
import threading
from datetime import date
from datetime import datetime
from os import strerror
import awscrt
from awsiot import mqtt5_client_builder
from awscrt import mqtt5, http
import time
import json

from concurrent.futures import Future
from utils.command_line_utils import CommandLineUtils

# MQTT Configuration
TIMEOUT = 100
message_topic = "test/topic"
received_count = 0
received_all_event = threading.Event()

control_topic = "control/lamp"

# cmdData is the arguments/input from the command line placed
into a single struct for
# use in this sample. This handles all of the command line
parsing, validating, etc.
# See the Utils/CommandLineUtils for more information.
cmdData = CommandLineUtils.parse_sample_input_mqtt5_pubsub()
proxy_options = None
future_connection_success = Future()
```

```
future_stopped = Future()

from datetime import datetime, timedelta

class sample_mqtt5_client:
    client: mqtt5.Client
    name: str
    count: int
    future_stopped: Future
    future_connection_success: Future

    # Creates a MQTT5 client using direct MQTT5 via mTLS with
the passed input data.
    def __init__(
            self,
            input_endpoint,
            input_cert,
            input_key,
            input_ca,
            input_client_id,
            input_client_name,
            automation_system) -> None:
        try:
            self.name = input_client_name
            self.future_stopped = Future()
            self.future_connection_success = Future()
            self.client = mqtt5_client_builder.mtls_from_path(
                endpoint=input_endpoint,
                cert_filepath=input_cert,
                pri_key_filepath=input_key,
                client_id=input_client_id,
                ca_filepath=input_ca,
                on_publish_received=self.on_publish_received,
                on_lifecycle_stopped=self.on_lifecycle_stopped,
```

```
on_lifecycle_connection_success=self.on_lifecycle_connection_suc
cess,

on_lifecycle_connection_failure=self.on_lifecycle_connection_fai
lure,

on_lifecycle_disconnection=self.on_lifecycle_disconnection,
            )
            self.automation_system = automation_system
        except Exception as ex:
            print(f"Client creation failed with exception:
{ex}")
            raise ex

    # Callback when any publish is received
    def on_publish_received(self, publish_packet_data):
        print(f"[{self.name}] Received a publish")

        publish_packet = publish_packet_data.publish_packet
        assert isinstance(publish_packet, mqtt5.PublishPacket)
        print(f"\tPublish received message on topic:
{publish_packet.topic}")
        print(f"\tMessage: {publish_packet.payload}")

        # if (publish_packet.user_properties is not None):
        #     if (publish_packet.user_properties.count > 0):
        #         for i in range(0,
publish_packet.user_properties.count):
        #             user_property =
publish_packet.user_properties[i]
        #             print(f"\t\twith UserProperty
({user_property.name}, {user_property.value})")
        # условие топиков
        if publish_packet.topic == control_topic:
            payloadDic = json.loads(publish_packet.payload)
```

```
            print(payloadDic)
            statusString = payloadDic['Status']
            brightnessString = payloadDic['Brightness']
            brightness = int(brightnessString)
            dateturnString = payloadDic['Timestamp']
            # dateturn = str(dateturnString)
            if dateturnString == '2024-11-25 18:00:00' and
statusString == '1':
                self.automation_system.set_status(Status.On)

self.automation_system.set_brightness(brightness)
            else:
                self.automation_system.set_status(Status.Off)
                self.automation_system.set_brightness(0)

            print("Status '{}'".format(statusString))
        # elif publish_packet.topic == message_topic:

    # Callback for the lifecycle event Stopped
    def on_lifecycle_stopped(self, lifecycle_stopped_data:
mqtt5.LifecycleStoppedData):
        print(f"[{self.name}]: Lifecycle Stopped")
        self.future_stopped.set_result(lifecycle_stopped_data)

    # Callback for the lifecycle event Connection Success
    def on_lifecycle_connection_success(self,
lifecycle_connect_success_data:
mqtt5.LifecycleConnectSuccessData):
        print(f"{self.name}]: Lifecycle Connection Success")

self.future_connection_success.set_result(lifecycle_connect_succ
ess_data)

    # Callback for the lifecycle event Connection Failure
```

```
    def on_lifecycle_connection_failure(self,
lifecycle_connection_failure:
mqtt5.LifecycleConnectFailureData):
        print(f"{self.name}]: Lifecycle Connection Failure")
        print(f"{self.name}]: Connection failed with
exception:{lifecycle_connection_failure.exception}")

    # Callback for the lifecycle event Disconnection
    def on_lifecycle_disconnection(self, disconnect_data:
mqtt5.LifecycleDisconnectData):
        print(f"{self.name}]: Lifecycle Disconnected")

        if (disconnect_data.disconnect_packet is not None):
            print(f"\tDisconnection packet code:
{disconnect_data.disconnect_packet.reason_code}")
            print(f"\tDisconnection packet reason:
{disconnect_data.disconnect_packet.reason_string}")
            if (disconnect_data.disconnect_packet.reason_code ==

mqtt5.DisconnectReasonCode.SHARED_SUBSCRIPTIONS_NOT_SUPPORTED):
                # Stop the client, which will interrupt the
subscription and stop the sample
                 self.client.stop()

class AutomationSystem:
    def __init__(self):
        self.__devices = []
        self.__sensor_data = []
        self.current_time = datetime.now() + timedelta(days = 2)
        self.is_first_entry = True
        self.time_amount = timedelta(seconds=20)  # 5 минут
    # # Integration of MQTT functionalities with GUI (if needed)
    #     self.mqtt_client = None
    #     self.setup_mqtt()

                # Create MQTT5 client
```

```
        self.client = mqtt5_client_builder.mtls_from_path(
                endpoint=cmdData.input_endpoint,
                port=cmdData.input_port,
                cert_filepath=cmdData.input_cert,
                pri_key_filepath=cmdData.input_key,
                ca_filepath=cmdData.input_ca,
                http_proxy_options=proxy_options,
                on_publish_received=on_publish_received,
                on_lifecycle_stopped=on_lifecycle_stopped,

on_lifecycle_connection_success=on_lifecycle_connection_success,

on_lifecycle_connection_failure=on_lifecycle_connection_failure,
                client_id=cmdData.input_clientId)
        print("MQTT5 Client Created")

        subscriber_one = sample_mqtt5_client(
            cmdData.input_endpoint, cmdData.input_cert,
cmdData.input_key, cmdData.input_ca,
            cmdData.input_clientId + "2", "Subscriber One",
self)
        subscriber_one.client.start()
        subscriber_one.future_connection_success.result(60)
        print(f"[{subscriber_one.name}]: Connected")



        if not cmdData.input_is_ci:
                print(f"Connecting to {cmdData.input_endpoint}
with client ID '{cmdData.input_clientId}'...")
        else:
                print("Connecting to endpoint with client ID")

        self.client.start()
        lifecycle_connect_success_data =
future_connection_success.result(TIMEOUT)
```

```
        connack_packet =
lifecycle_connect_success_data.connack_packet
        negotiated_settings =
lifecycle_connect_success_data.negotiated_settings
        if not cmdData.input_is_ci:
                print(
                    f"Connected to
endpoint:'{cmdData.input_endpoint}' with Client
ID:'{cmdData.input_clientId}' with
reason_code:{repr(connack_packet.reason_code)}")
            # client.start()

        # # Subscribe
        # print("Subscribing to topic
'{}'...".format(message_topic))
        # subscribe_future =
self.client.subscribe(subscribe_packet=mqtt5.SubscribePacket(
        #     subscriptions=[mqtt5.Subscription(
        #         topic_filter=message_topic,
        #         qos=mqtt5.QoS.AT_LEAST_ONCE,
        #         callback=on_message_received)]
        # ))

        # suback = subscribe_future.result(TIMEOUT)
        # print("Subscribed with
{}".format(suback.reason_codes))

          # Subscribe to the shared topic on both subscribers
        subscribe_packet = mqtt5.SubscribePacket(
            subscriptions=[mqtt5.Subscription(
                topic_filter=control_topic,
                qos=mqtt5.QoS.AT_LEAST_ONCE)]
        )

        subscribe_one_future =
subscriber_one.client.subscribe(subscribe_packet)
```

```
        suback_one = subscribe_one_future.result(60)
        # print(f"[{subscriber_one.name}]: Subscribed to topic
'{message_topic}' in shared subscription group
'{message_topic}'.")
        # print(f"[{subscriber_one.name}]: Full subscribed topic
is: '{message_topic}' with SubAck code:
{suback_one.reason_codes}")

        print(f"[{subscriber_one.name}]: Subscribed to topic
'{control_topic}'.")
        print(f"[{subscriber_one.name}]: Full subscribed topic
is: '{control_topic}' with SubAck code:
{suback_one.reason_codes}")



    def get_devices(self):
        return self.__devices
    def add_devices(self):
        sl = SmartLight(0, 0)
        ms = MotionSensor(0, 0)

        self.__devices.append(sl)
        self.__devices.append(ms)

    def set_brightness(self, brightness):
        self.__devices[0].set_brightness(brightness)

    def set_status(self, status):
        self.__devices[0].set_status(status)

#логика
    def exec_automation_tasks(self):
        if self.__devices[1].get_motion():
            self.__devices[0].set_status(Status.On)
            if self.__devices[0].get_brightness() == 0:
```

```
                self.__devices[0].set_brightness(100)

        else:
            print(' threadID = ',threading.get_ident())
            print(datetime.now(),' is_first_entry =',
self.is_first_entry)
            if self.is_first_entry:
                self.current_time = datetime.now()
                self.is_first_entry = False
                print(datetime.now(),' CurrentTime =',
self.current_time)
            else:
                entry_period = datetime.now() -
self.current_time
                print(datetime.now(),' entry_period =',
entry_period)
                print(datetime.now(),' is_first_entry =',
self.is_first_entry)
                if entry_period > self.time_amount:
                    self.__devices[0].set_brightness(0)
                    self.__devices[0].set_status(Status.Off)
                    self.is_first_entry = True


    def randomize(self):
        self.__devices[0].set_brightness(random.randint(0,100))

        self.__devices[1].detect_motion(True if
random.randint(0,1) else False)

    def randomize_detect_motion(self):
        self.__devices[1].detect_motion(True if
random.randint(0,1) else False)
        return self.__sensor_data

    def get_sensor_data(self):
```

```
        return self.__sensor_data

    def gather_sensor_data(self):
        current_time = datetime.now().strftime("%H:%M:%S")
        self.__sensor_data.append("[" + str(date.today()) + " "
+ current_time + "] " + "Living room Light brightness: " +
str(self.__devices[0].get_brightness()) + "%" + "\n")
        self.__sensor_data.append("[" + str(date.today())  + " "
+ current_time + "] " + "Motion Sensor: " +
str(self.__devices[1].get_motion()) + "\n\n")

    def store_sensor_data(self):
        # return
        try:
            fo = open('out.txt', "wt")
            for line in self.__sensor_data:
                fo.write(line)
            fo.close()
        except IOError as e:
            print("Error: ", strerror(e.errno))

        #         # Create MQTT5 client
        # client = mqtt5_client_builder.mtls_from_path(
        #         endpoint=cmdData.input_endpoint,
        #         port=cmdData.input_port,
        #         cert_filepath=cmdData.input_cert,
        #         pri_key_filepath=cmdData.input_key,
        #         ca_filepath=cmdData.input_ca,
        #         http_proxy_options=proxy_options,
        #         on_publish_received=on_publish_received,
        #         on_lifecycle_stopped=on_lifecycle_stopped,
        #
on_lifecycle_connection_success=on_lifecycle_connection_success,
        #
on_lifecycle_connection_failure=on_lifecycle_connection_failure,
        #         client_id=cmdData.input_clientId)
```

```
        # print("MQTT5 Client Created")

        # if not cmdData.input_is_ci:
        #         print(f"Connecting to {cmdData.input_endpoint}
with client ID '{cmdData.input_clientId}'...")
        # else:
        #         print("Connecting to endpoint with client ID")

        # client.start()
        # lifecycle_connect_success_data =
future_connection_success.result(TIMEOUT)
        # connack_packet =
lifecycle_connect_success_data.connack_packet
        # negotiated_settings =
lifecycle_connect_success_data.negotiated_settings
        # if not cmdData.input_is_ci:
        #         print(
        #             f"Connected to
endpoint:'{cmdData.input_endpoint}' with Client
ID:'{cmdData.input_clientId}' with
reason_code:{repr(connack_packet.reason_code)}")

    #print(f"brightness - {self.devices[0].get_brightness()}%")

        # message_string = "brightness"


        # !!!!!
        # message = "{} [{}]".format(message_string,
self.__devices[0].get_brightness()) #145
        # publish_future =
self.client.publish(mqtt5.PublishPacket(
        #         topic=message_topic,
        #         payload=json.dumps(message),
        #         qos=mqtt5.QoS.AT_LEAST_ONCE
        #     ))
```

```
        # publish_completion_data =
publish_future.result(TIMEOUT)
        # print(publish_completion_data)


# Callback when any publish is received
def on_publish_received(publish_packet_data):
    publish_packet = publish_packet_data.publish_packet
    assert isinstance(publish_packet, mqtt5.PublishPacket)
    print("Received message from
topic'{}':{}".format(publish_packet.topic,
publish_packet.payload))
    global received_count
    received_count += 1
    # if received_count == cmdData.input_count:
    #     received_all_event.set()


        #  automation_system.__devices[0].set_status(Status.On)

# Callback for the lifecycle event Stopped
def on_lifecycle_stopped(lifecycle_stopped_data:
mqtt5.LifecycleStoppedData):
    print("Lifecycle Stopped")
    global future_stopped
    future_stopped.set_result(lifecycle_stopped_data)

# Callback for the lifecycle event Connection Success
def
on_lifecycle_connection_success(lifecycle_connect_success_data:
mqtt5.LifecycleConnectSuccessData):
    print("Lifecycle Connection Success")
    global future_connection_success

future_connection_success.set_result(lifecycle_connect_success_d
ata)
```

```
# Callback for the lifecycle event Connection Failure
def
on_lifecycle_connection_failure(lifecycle_connection_failure:
mqtt5.LifecycleConnectFailureData):
    print("Lifecycle Connection Failure")
    print("Connection failed with
exception:{}".format(lifecycle_connection_failure.exception))


# Define MQTT methods from mqtt5_pubsub.py
def on_message_received(topic, payload, dup, qos, retain,
**kwargs):
    global received_count
    print("Received message from topic '{}'".format(topic))
    received_count += 1
    received_all_event.set()

def mqtt_subscribe_and_publish(client, message_topic,
message_string, message_count):
    # Start the MQTT client
    print("Starting MQTT client...")
    # client.start()



    # # Subscribe
    # print("Subscribing to topic
'{}'...".format(message_topic))
    # subscribe_future =
client.subscribe(subscribe_packet=mqtt5.SubscribePacket(
    #     subscriptions=[mqtt5.Subscription(
    #         topic_filter=message_topic,
    #         qos=mqtt5.QoS.AT_LEAST_ONCE)]
    # ))
    # suback = subscribe_future.result(TIMEOUT)
```

```
    # print("Subscribed with {}".format(suback.reason_codes))

    # # Publish
    # if message_string:
    #     if message_count == 0:
    #         print("Sending messages until program killed")
    #     else:
    #         print("Sending {}
message(s)".format(message_count))

    #     publish_count = 1
    #     while (publish_count <= message_count) or
(message_count == 0):
    #         message = "{} [{}]".format(message_string,
publish_count)
    #         print("Publishing message to topic '{}':
{}".format(message_topic, message))
    #         publish_future =
client.publish(mqtt5.PublishPacket(
    #             topic=message_topic,
    #             payload=json.dumps(message_string),
    #             qos=mqtt5.QoS.AT_LEAST_ONCE
    #         ))
    #         publish_completion_data =
publish_future.result(TIMEOUT)
    #         print("PubAck received with
{}".format(repr(publish_completion_data.puback.reason_code)))
    #         time.sleep(1)
    #         publish_count += 1

    received_all_event.wait(TIMEOUT)
    print("{} message(s) received.".format(received_count))

    # Unsubscribe
    print("Unsubscribing from topic '{}'".format(message_topic))
```

```
    unsubscribe_future =
client.unsubscribe(unsubscribe_packet=mqtt5.UnsubscribePacket(
        topic_filters=[message_topic]))
    unsuback = unsubscribe_future.result(TIMEOUT)
    print("Unsubscribed with {}".format(unsuback.reason_codes))

    print("Stopping Client")
    client.stop()
    future_stopped = client.stopped()
    future_stopped.result(TIMEOUT)
    print("Client Stopped!")
```