Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра ____Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____
____Адаптивна система керування ресурсами в small-office та home системах__
_____
(тема)

Виконав:
студент _2__ курсу, групи __СКСм-19-2_____
____Сєріков А.І._____
(прізвище, ініціали)

Спеціальність _____
_____123 Комп'ютерна інженерія_____
(код і повна назва спеціальності)
Тип програми __освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____
___Спеціалізовані комп'ютерні системи_____
(повна назва освітньої програми)

Керівник __проф Немченко В.П._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ _____
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Сєрікову Антону Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Адаптивна система керування ресурсами в small-office та home системах
затверджена наказом університету від 26 березня 2021р. № 385
2. Термін подання студентом роботи до екзаменаційної комісії _____ 20__ р.
3. Вихідні дані до роботи
Сенсори освітлення, температури, вологи
Розумна лампа
Пакет збірок .NET Core IoT
Платформа .NET Core
Пакет збірок ML.NET

4. Перелік питань, що потрібно опрацювати в роботі
Розглянути фізичні властивості кліматичних ресурсів і їх вимірювання
Розглянути підходи прогнозування параметрів користування кліматичними ресурсами
Спроектувати архітектуру системи адаптивного керування ресурсами в home системах
Розробити систему збору та збереження логів о стані середовища
Розробити модель прогнозування параметрів користування кліматичними ресурсами
Розробити систему адаптивного керування кліматичними ресурсами в home системах
Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
17 слайдів_____

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	01.02.2021 -20.02.2021	
2	Аналіз предметної області	20.02.2021 -10.03.2021	
3	Аналіз джерел з проблемної галузі	10.03.2021 -31.03.2021	
4	Збір та обробка даних	01.04.2021 -15.04.2021	
5	Розробка програми	15.04.2021 - 01.05.2021	
6	Оформлення пояснювальної записки	01.05.2021 - 10.05.2021	
7	Оформлення графічного матеріалу	10.05.2021-20.05.2021	
8	Перевірка виконаного проекту керівником	20.05.2021-25.05.2021	
9	Захист проекту	25.05.2021-30.05.2021	

Дата видачі завдання 01 02 2021р.

Студент _____
(підпис)

Керівник роботи _____ проф. Немченко В.П.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Атестаційна робота містить 78 сторінок, 22 рисунки, 6 таблиць, 20 джерел посилання, 6 додатків.

Об'єктом дослідження є адаптивна система керування ресурсами в small-office та home системах.

В результаті виконання атестаціїної роботи були проаналізовані технології для проектування систем керування ресурсами в small-office та home системах. Проаналізовані фізичні властивості кліматичних ресурсів в home системах, вимоги та принципи вимірювання цих ресурсів. Розроблено приклад архітектури системи керування ресурсами в small-office та home системах. Проаналізовані принципи та вимоги до збору та збереження логів о стані середовища в системі. Дослідженні алгоритми та стратегії для прогнозування параметрів використання кліматичних ресурсів в системі. Розроблена система для адаптивного керування кліматичними ресурсами в small-office та home системах.

IoT СИСТЕМА, РОЗУМНИЙ ДІМ, ВІДСТЕЖЕННЯ СТАНУ СЕРЕДОВИЩА, АДАПТИВНІСТЬ, МАШИННЕ НАВЧАННЯ, АДАПТИВНЕ КЕРУВАННЯ КЛІМАТИЧНИМИ РЕСУРСАМИ.

ABSTRACT

Certification work has 78 pages, 22 images, 6 tables, 20 sources, 6 addition. The object of the research is an adaptive resource management system in small-office and home systems. As a result of the certification work, technologies for designing resource management systems in small-office and home systems have been analyzed. The physical properties of climatic resources in home systems, requirements and principles of measuring these resources have been analyzed. An example of resource management system architecture in small-office and home systems has been developed. The principles and requirements of collecting and storing logs on the state of the environment in the system have been analyzed. Algorithms and strategies for prediction the parameters of the use of climatic resources in the system have been investigated. The system for adaptive management of climate resources in small-office and home systems has been developed.

IoT SYSTEM, SMART HOME, TRACK ON THE STATE OF THE ENVIRONMENT, ADAPTIVE SYSTEM, MACHINE LEARING, ADAPTIVE CONTROL, CLIMATIC RESOURCES.

ЗМІСТ

ВСТУП

Автоматизація в домашніх системах інтенсивним темпом розвивається з початку 2010-х років. Напрям є достатньо популярним і має перші досягнення ще з колишнього століття.

Основні тенденції розвитку домашній автоматизації останніх десятирічь:

– Зниження вартості.

Доступна вартість до кінцевого користувача в автоматизованих домашніх системах є одним із найважливіших критеріїв. І це було основною проблемою, яку мала автоматизація в домашніх системах з самого початку розвитку - це є достатньо висока вартість впровадження цієї системи в будинок чи офіс для широкого кругу користувачів. За останнє десятиріччя середня вартість на основні компоненти системи (датчики, хаб, актуатори) зменшилась у декілька разів[1]. Основні причини тому – це загальний спад цін на електроніку, появлення на ринку крупних та конкурентних компаній з Кореї та Китаю;

– Еволюційне покращення інтерфейсу керування системою.

Ще з перших років розвитку напрям мав багато форм інтерфейсу. До цього, це був інтерфейс на певному пристрої. Впродовж, останніх десяти років багато гігантів у сфері впроваджували розумних помічників до системи. Найчастіше, розумні помічники мали форму голосового асистента. Наприклад, Alexa - в розумних колонках Amazon Alexa або Alica від компанії Яндекс;

– Безпека.

Безпека була однією із головною проблемою в домашній автоматизації ще на перших кроках розвитку домашній автоматизації. Поточний тренд-ідея - це використання технології блокчейну для досягнення більш вищої безпеки в Інтернеті речей. На сьогоднішній день, безпека в IoT є однієї із самих інвестованих напрямів. Очікується, що витрати на безпеку в IoT досягнуть 3,1 млрд доларів у 2021 році[19];

1 ПОСТАНОВКА ЗАДАЧІ

Мета роботи - це розробка системи та моделі для адаптивного та автоматизованого керування кліматичними ресурсами в домашніх та офісних системах.

Задачі роботи:

– Спроектувати архітектуру системи адаптивного керування ресурсами в home системах;

– Дослідити підходи прогнозування параметрів користування кліматичними ресурсами;

– Розробити сервіс для збору та збереження логів стану середовища;

– Розробити модель прогнозування параметрів користування кліматичними ресурсами;

– Розробити сервіс для адаптивного та автоматизованого керування кліматичними ресурсами;

2 ПРОБЛЕМАТИКА ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Домашні системи вже мають розроблений механізм для зміни кліматичних умов в приміщеннях. Так для зміни освітлення ми використовуємо ліхтарик, для зміни температури кондиціонер, для зміни вологості - зволожувач повітря. Також вже існують розумні та цифрові інтерфейси для автоматизованого контролю вказаними пристроями. Найчастіше прикладом такого керування є програмний додаток чи голосовий асистент. Проблематика дослідження даної роботи є в першу чергу не автоматизоване керування, а адаптивне. Адаптивність системи полягає в зміні поведінки системи внаслідок накопичення досвіду від взаємодії з користувачем.

Будь яке приміщення, яке використовує людина, потребує в контролі кліматичних умов. Можна вказати наступні найпоширеніші кліматичні умови (див. табл. 2.1):

Таблиця 2.1 - Кліматичні умови та їх одиниці виміру

Кліматична умова	Одиниці вимірювання (SI)
Освітлення	Люкс
Температура	Градус Цельсія
Абсолютна вологість повітря	кг/м³
Тиск	Па (Паскаль)

2.1 Освітлення

2.1.1 Властивості фізичного процесу

Будь-який об'єкт випромінює і поглинає електромагнітне випромінювання. Електромагнітне випромінювання несе енергію і деякий спектр цього випромінювання може сприйматися людським оком. Такий спектр, називають **видимим світлом.** Видимий спектр починається з

400 нм(фіолетовий) до 700 нм(червоний) (див. рис. 2.1) [2].

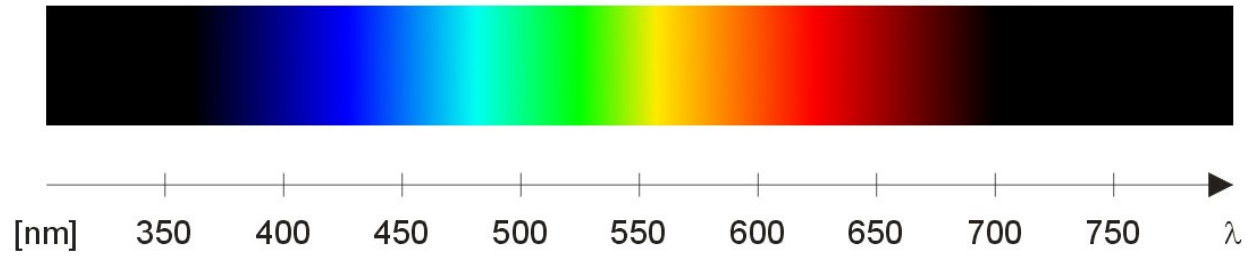

Рисунок 2.1 - Видимий спектр

Електромагнітне випромінювання має дві взаємопов'язані фізичні величини:

– довжина хвилі. Позначення - λ. Одиниця вимірювання в системі SI - м (найчастіше, вказують в нанометрах);

– частота хвилі. Позначення - f. Одиниця вимірювання в системі SI - Гц (англійський варіант - Hz);

Швидкість світла є стала і дорівнює приблизно 3×10^8, та вираховується як множення між довжиною та частотою.

По своєї природі світло має наступні властивості:

1) прямолінійне поширення;

2) відбиття;

3) заломлення;

Джерелом світла є тіло, що випромінює видимий спектр електромагнітного випромінювання. Можуть бути наступні джерела світла:

– природні. Наприклад: сонце, полярне сяйво, зірки та комети, біолюмінесценція;

– штучні. Наприклад: ліхтарі, свічка, світлодіод;

Основною властивістю джерела світла є світловий потік.

Світловий потік (Luminous Flux) - це фізична величина, що характеризує енергію світлового випромінювання через деяку поверхню за певну одиницю часу. Одиниця вимірювання - Люмен (Лм). Ця величина вказується в технічних характеристиках штучних джерелах світла.

Так наприклад (див. табл. 2.2):

Таблиця 2.2 - Приклади величин світлового потоку для певних штучних джерел світла

Штучне джерело світла	Світловий потік, Люмен
Гасова лампа	100
26-ватна газорозрядна лампа	1600
100-ватна лампа розжарення	1750
Світлодіодна розумна лампа Yeelight LED Filament Light Bulb	700

Просторовий кут (Ω) - це частина простору, яка об'єднує усі промені що виходять з деякої точки. Дорівнює відношенню площі сферичної поверхню S (A) до квадрата радіуса сфери R(див. рис. 2.2).

$$\Omega = \frac{S}{R^2} \tag{2.1}$$

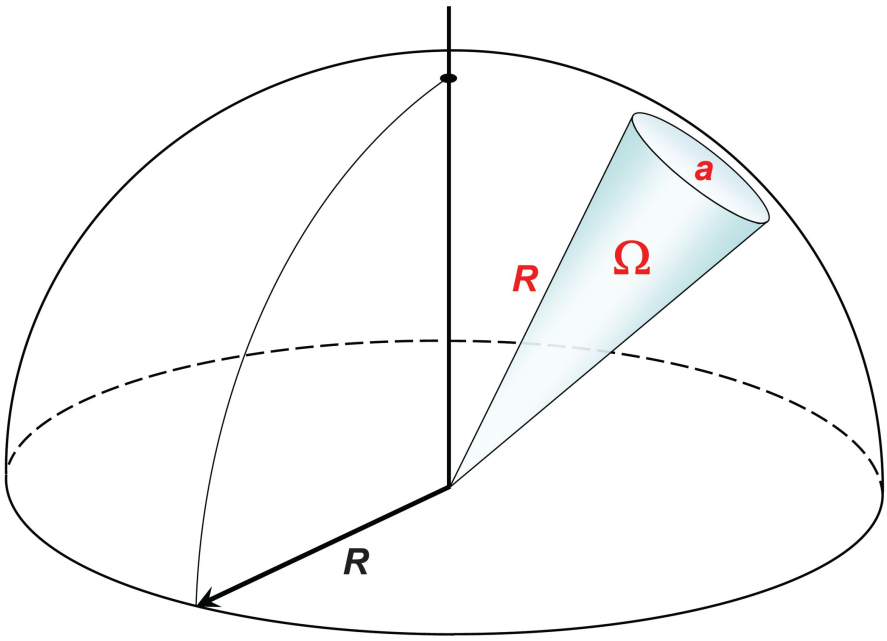

Рисунок 2.2 - Просторовий кут

Інтенсивність світлового потоку (Luminous Intensity) - це відношення світлового потоку до тілесного кута(див. рис. 2.3). Вимірюється в канделах (Кд).

$$I = \frac{F}{\Omega} \tag{2.2}$$

де F – сила світлового потоку, Ω – просторовий кут;

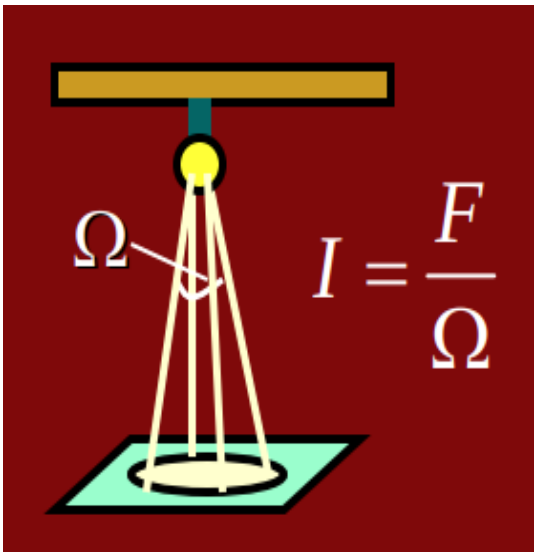

Рисунок 2.3- Сила світу

Освітлення (Illuminance) є відношення світлового потоку на площу (див. рис. 2.4). Вимірюється в люксах (люмен/$м^2$).

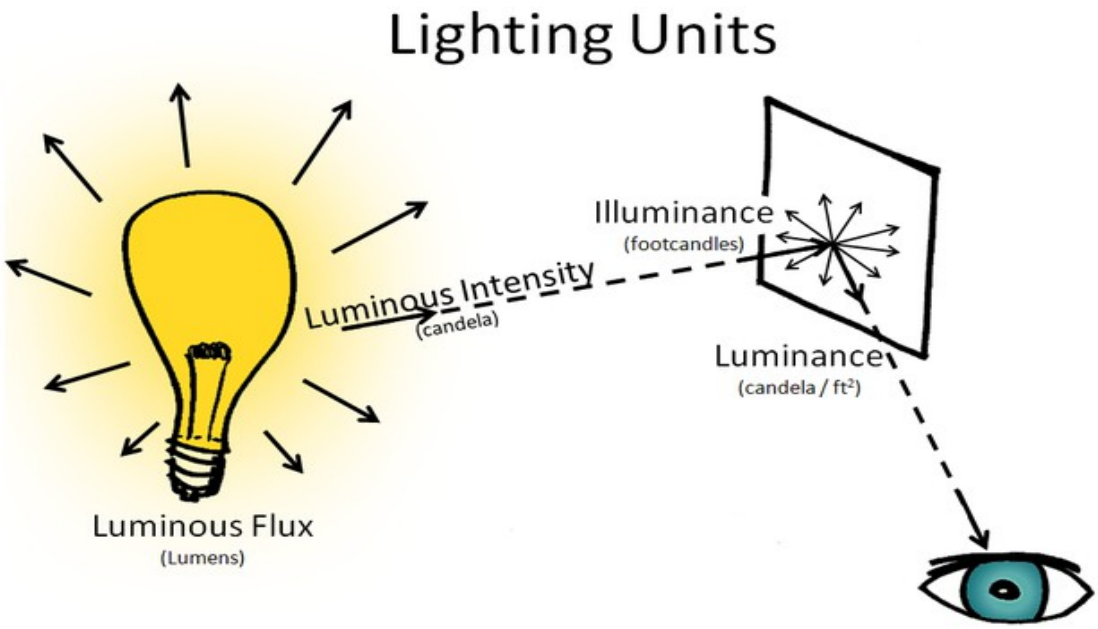

Рисунок 2.4 - Одиниці вимірювання світла[4]

2.1.2 Вимоги до рівня освітлення на робочому місці

В формулюванні нормативів забезпечення світла використовують величину освітлення. Освітлення є одним із найважливіших кліматичних ресурсів, які використовуються в вимогах до робочого місця. Рівень освітленості впливає на рівень комфорту, ефективності та здоров'я людини. Недостатній рівень освітленості негативно впливає на психофізіологічний стан та може створювати ризик травматизму на підприємстві.

Треба зауважити, що освітлення впливає на стан людину через два фактори: візуальні (зорові) та невізуальні потреби. Візуальні потреби не потребують детального розгляду. Багато процесів в житті людини тісно пов’язані з потребою в деталізованому зоровому контакті (наприклад, робота с документами, дисплеями або робочим станком). Але також освітлення впливає на психофізіологічний стан людини через циркадний ритм, тобто через щоденний період ритмів (пробудження, концентрація, сон і т.д). Тобто у протязі дня людина потребує різні рівні освітленості. Візуальні потреби відстежується через нормативні показники у вигляді мінімального та максимального рівнів освітлення. Невізуальна потреба має бути задовільна через динаміку освітленості згідно з зміною часу в добі.

Освітлення відрізняють за наступні функціональними потребами:

1) робоче;
2) аварійне;
3) евакуаційне;
4) охоронне;
5) чергове;

Також в залежності, від спеціальності фахівця може відрізнятися і необхідний рівень освітлення.

Загалом існують наступні рекомендації щодо освітлення робочого місця оператора ПК згідно з нормовані показники освітлення основних приміщень громадських, житлових, допоміжних будинків (див. табл. 2.3):

Таблиця 2.3 – Рекомендації щодо освітлення робочого місця

Тип роботи	Мінімальний рівень в люксах

В зоні робіт категорії високої зорової точності (наприклад, клавіатури та документів)	300 - 500
На екрані	200

Найчастіше, нормативні показники вказують на мінімальний та максимальний рівень для певного місця не зважаючи на потребу в динаміці для підтримки циркадних ритмів людини. Варто зауважити, що не доза кількості світла, а структура (час / динаміка) штучного світла є критично важливою вплив на циркадну фазу людини. Так наприклад, загально рекомендований рівень освітленості робочого місця - 500 люкс, але необхідний рівень освітленості для циркадного ритму активної людини є 2000 люкс[7].

2.1.3 Вимоги до вимірювання освітленості . Специфіка процесу

Значення освітленості можна виміряти за допомогою фотометрів. Такий пристрій може містити фотодіод, обладнаний відповідним оптичним фільтром.

На рівень освітленості в першу чергу впливають два фактори:

– дистанція від сенсора;

– кут падіння світлового променю на датчик;

У випадку робочого місця сенсор повинен для більш коректного значення освітленості виконувати наступні вимоги:

– сенсор повинен знаходитися близько до центру робочої зони;

– сенсор повинен позиціонуватися паралельно площині робочої зони;

Сенсори світла вимірюють освітленість, за допомогою якої можна виміряти більше, ніж силу світлового потоку (яскравість) джерела світла. Датчик світла може використовуватися для вимірювання відносної відстані від джерела[15].

2.2 Температура та вологість. Вимоги та принципи вимірювання

Температура та вологість пов’язані параметри, через те, що температура може впливати на вологість повітря.

Температура може бути представлена в багатьох системах вимірювання:

1) градус Цельсія;
2) градус Фаренгейта;
3) градус Ньютона;
4) градус Ранкіна;
5) Кельвін;

Температура вимірюється за допомогою терморезисторів. Температура зіставляється з опором діода. Чим холодніше температура, тим менше буде опір, і навпаки. Опір на діоді вимірюється і перетворюється на зчитуванні одиниці температури[17].

Існує два типи вологості:

– абсолютна;

Кількість водяної пари (у грамах), що міститься в 1 м³ повітря[16]

– відносна;

Відношення масової частки водяної пари в повітрі до максимально можливої частки при даній температурі. Вимірюється в %. Якщо повітря насичене водяною парою до максимально можливого рівня, відносна вологість такого повітря становить 100%[16].

У випадку ємнісного датчика вологості чутливим елементом є конденсатор. Зміна вологості повітря прямо пропорційна зміні ємності. У резистивному датчику є керамічний або провідний полімер, що поглинає вологу, що потім впливає на його питомий опір. Потім відносна вологість визначається на основі зміни сили струму.

Принципи вимірювання температури та вологості:

– сенсор не повинен знаходитися під прямими сонячними променями;

– сенсор повинен знаходитися поблизу до робочого місця, але не біля радіатору чи системи кондиціонування’

– сенсор не повинен знаходитися біля побутових приборів, що можуть споживати велику кількість електроенергії (духовка, холодильник, мультиварка, системний блок комп’ютера);

– сенсор повинно тримати подалі від вікон та вентиляційних систем;

– сенсор не повинен знаходитися біля побутових приборів, що можуть викидати велику кількість пару (духовка, кавоварка, мультиварка);

3 СТРУКТУРА СИСТЕМИ

3.1 Компоненти та рівні системи

Система IoT (як і система керування ресурсами) є набір компонентів та шарів, для забезпечення IoT рішення.

В описі архітектури системи необхідно описати:

1) рівні системи;
2) компоненти системи;
3) інтерфейси взаємодії;

Кожна компонента виконує певні функції та комунікує завдяки певному інтерфейсу з компонентами на навколишніх рівнів. Кількість рівнів та компонентів в системі залежить в першу чергу від мети та заявлених задач та проблем. Так наприклад, якщо система не потребує інтеграції зі сторонніми системами або контролю за допомогою підключення до глобальної мережі (наприклад, Інтернет, національний або муніципальний рівень), тобто взаємодія можлива тільки з довіреними пристроями, то і немає потреби необхідності у компоненті безпеки у вигляді шифрування/дешифрування між пристроями локальної мережі, достатньо заборонити підключення з усіх інших пристроїв окрім довірених.

Також треба зауважити, що існують два типи взаємодії:

1) машина к машині (M2M);
2) машина до людині (M2P);

Для того, щоб виділити необхідні компоненти необхідно зробити детальний аналіз прикладу архітектури IoT-системи та можливих компонентів, і рівнів в цьому прикладі[8]. За довгий період розвитку IoT систем, було розроблено багато патернів та прикладів архітектур з детальним розподілом відповідальності між компонентами.

Стандартний приклад архітектури IoT системи має наступний вигляд (див. рис. 3.1):

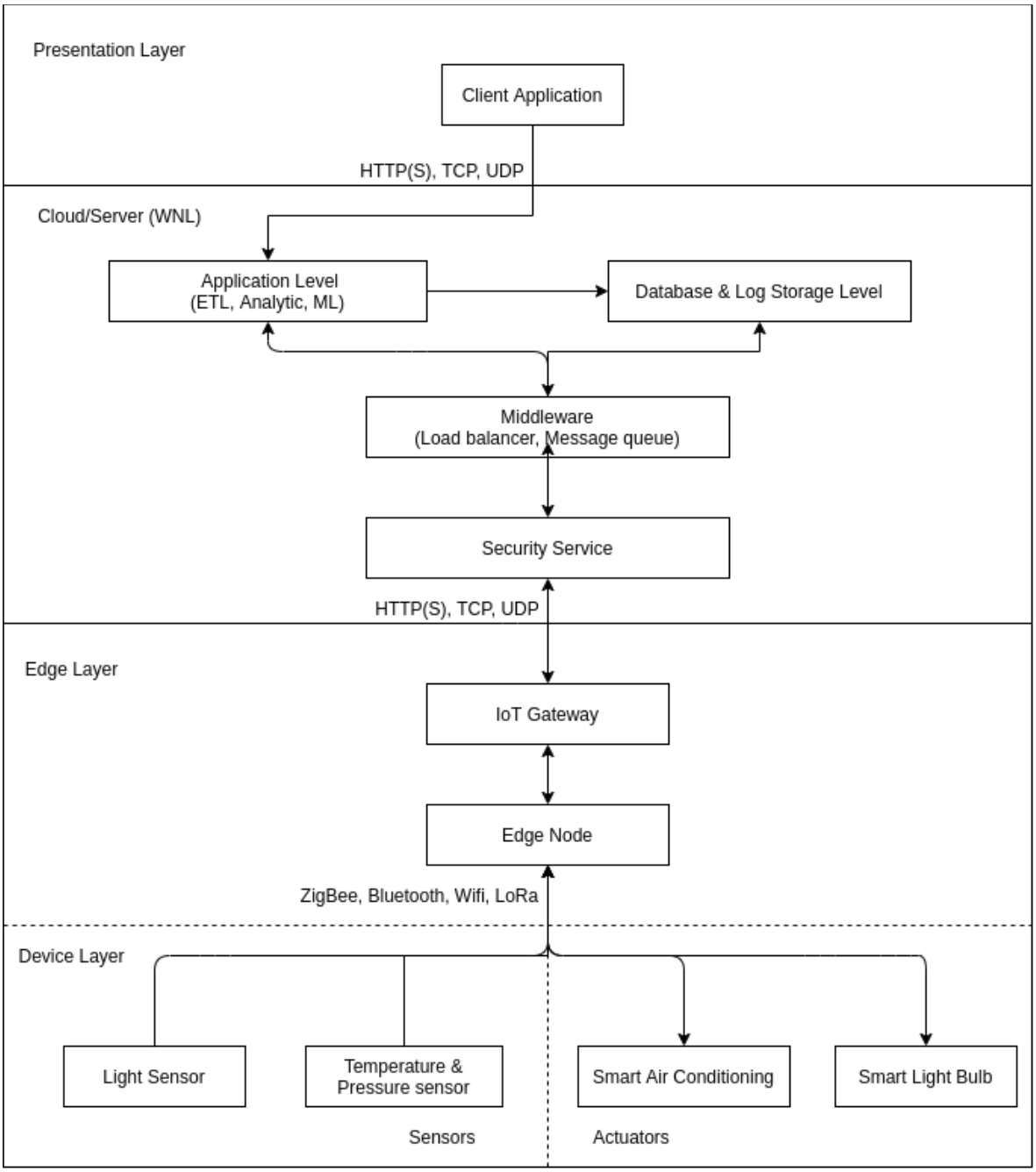

Рисунок 3.1 - Архітектура IoT системи

Згідно з архітектурним прикладом на рисунку 3.1 можна виділити щонайменше 4 рівня:

1) Device Layer (Рівень пристроїв);
2) Edge Layer (Рівень периферійних обчислень);
3) Cloud/Server Layer (Рівень хмари або серверу);
4) Presentation Layer (Рівень презентації або користувача);

Зробимо короткий аналіз вказаних компонентів (див. табл. 3.1):

Таблиця 3.1- Можливі компоненти та рівні в системі IoT

№	Рівень	Компонент	Тип об’єкту	Відповідальність
1	Пристроїв	Сенсори	Датчик, мікрофон	Збір інформації
2	Пристроїв	Актуатори	Реле, розумні лампи, тощо	Виконання певних фізичних явищ
3	Периферійний	Edge node	SoC модулі	Отримання, перетворення, збереження інформації на локальному рівні
4	Периферійний	IoT Gateway	SoC модулі, Mini PC	Комунікація с Cloud, локальне управління без зв’язку з Cloud. Обробка та фільтрація даних до передачі до Cloud
5	Рівень хмари або серверу	Security Service	Програмний компонент	Проведення аутентифікації, авторизації та обліку. Шифрування та дешифрування даних при взаємодії в глобальній мережі.
6	Рівень хмари або серверу	Middleware	Програмний компонент	Балансування навантаження та побудова черги повідомлень між серверами
7	Рівень хмари або серверу	Application	Сервер/мережа серверів	Може містити різноманітні сервіси: - аналітики; - AI/ML – блок;

				- Big Data; - Контроллер;
8	Рівень хмари або серверу	Database & Log	Сервер/мережа серверів	Реляційні або NoSQL бази даних.
9	Рівень представлення	Client Application	Клієнтський пристрій з додатком	Відображення статистики, активація певних пристроїв в системі. Механізми для керування та конфігурації системи.

Компоненти взаємодіють за допомогою наступних інтерфейсів:

1) Фізичний-рівень з IoT-шлюзом чи Edge Node;
    1) Безпровідні (ZigBee, LoRa, Wifi, Bluetooth);
    2) Провідні (I2C, SPI, Ethernet);
2) IoT gateway є пристроєм, що бере відповідальність за комунікацію з хмарою чи сервером. Найчастіше, використовується протоколи з транспортного та прикладного рівня стеку TCP/IP (TCP, UDP, HTTP/HTTPS);
3) Взаємодія між серверами на рівні хмари та з рівнем представлення також виконується за допомогою транспортних та протоколів. Так наприклад:
    1) Сервер (програмний компонент) взаємодії з сервером бази даних за допомогою TCP або UDP протоколу;
    2) Прикладні інтерфейси (наприклад, між сервером та клієнтським додатком) спираються на протокол HTTP(S);

3.2 Розробка та аналіз архітектури адаптивної системи керування кліматичними ресурсами

Спираючись на проаналізовані вище рівні, компоненти та інтерфейси взаємодії розробимо приклад архітектури адаптивної системи для керування кліматичними ресурсами в small-офіс та home системах. Треба зауважити, що в рамках роботи будуть розглянуті необхідні блоки і сама система, як об’єкт дослідження. А також те що, використані компоненти, пристрої та протоколи розглядаються лише як складова прототипу, який не претендує по задачі на комерційний успіх.

В першу чергу, потрібно вказати необхідні компоненти без яких система не може функціонувати:

1) Рівень пристроїв;

Саме наявність сенсорів для збору інформації о ресурсах та актуаторів для взаємодії з середовищем;

1) Сенсори:
   i) датчик освітлення;
   ii) датчик температури;
   iii) датчик вологості;
   iv) датчик тиску;
2) Актуатори;
   i) розумна лампа;
   ii) розумна система кондиціонування;

2) Рівень периферійного обчислення;

Датчики можуть генерувати багато даних в різному форматі. Що найменше є потреба в перетворенні та агрегації даних. В рамках одного середовища (кімнати або робочого місця) є потреба в вузлу для периферійних обчислень;

3) Рівень хмари або серверу;
   1) Сервер бази даних;
      i) збереження даних з сенсорів. (величина, одиниця виміру, дата та час). Дані необхідні для використання в алгоритмах машинного навчання та аналітики;

ii) збереження інформації о пристроях;

iii) збереження інформації о активації актуаторів (розумної лампи і системи кондиціонування);

2) Хмара або сервер;

i) компонент прогнозування, що відповідає за прогнозування використання певних ресурсів в певний час;

ii) компонент контролеру - є веб-служба, що дозволяє відстежувати та змінювати стан в системі (наприклад, периферійних пристроїв);

4 РІВЕНЬ ПРИСТРОЇВ

Адаптивна система базуються на ідеї зміни поведінки в наслідок накопичення досвіду. Інформація о кліматичному стані середовища, наявність споживача в системі, стан пристроїв для керування кліматичним станом, час взаємодії зі споживачем є системно важливі для забезпечення роботи компоненту прогнозування.

При відстеженні стану будь-якого параметру середовища або пристрою нас в першу чергу цікавить динаміка змінення цього параметра. Отримання необхідної інформації та динаміки можливо за двома моделями:

– модель постійного моніторингу. В залежності від передбачуваного/розрахункового часового інтервалу виконується з читання поточного стану та порівняння з минулим;

– модель оповіщення. В цьому випадку, пристрій самостійно сповіщає о зміні стану. Можлива в пристроях в яких стан змінюється від деякої логіки або взаємодії з користувачем (наприклад, розумна лампа), або в пристрої, де вже існує постійний моніторинг стану;

4.1 Відстеження кліматичного стану середовища

Як вже було зазначено розділі “Проблематика та аналіз предметної області”, необхідно виділити щонайменше три складові в кліматичному середовищі:

1) освітлення;
2) температура;
3) вологість повітря;

При розгляді можливих датчиків освітлення треба враховувати наступні фактори, які і формують різноманіття продуктів на ринку:

1) діапазон можливих значень;

Це основний параметр, який впливає на розрядність цифрових компонентів датчику. Наприклад, АЦП, регістри, тощо. Найчастіше є можливість зустріти 16 бітні датчики, що дають можливість працювати у діапазоні до 65535. Наприклад, у випадку сенсора освітлення цього діапазону достатньо для роботи датчику в office та home системах, але треба враховувати, що під прямими променями сонця рівень освітленості може вимірюватися в діапазоні від 30 до 130 тисяч люкс. Сенсори, які підтримують великий діапазон значень, очікувано будуть вище за ціною;

2) точність результату вимірювання;

Точність результату можна визначити як величину похибки вимірювання щодо абсолютного значення. Наприклад, точність результату вимірювання вологості за допомогою сенсору вологості BME280 дорівнює ± 3%. Необхідний рівень похибки залежить від вимог системи, де буде використовуватися сенсор, та вимірювального параметру;

3) чутливість;

Чутливість є мінімальний поріг фізичного параметра, який призведе до генерації вихідного сигналу з сенсора. Похибка чутливості є відхилення від очікуваного значення мінімального порогу;

4) час відгуку;

Час необхідний для зміни вихідного сигналу датчику з попереднього стану на остаточне встановлене нове значення в допустимому діапазону;

5) можливі режими роботи сенсора;

Найчастіше, виділяють три режими роботи:

1) режим сну;

Не виконує жодних вимірювань. Споживання енергії мінімальне;

2) режим одного вимірювання;

Виконує одну операцію вимірювання та повертається до режиму сна;

3) звичайний режим;

Режим включає автоматизовану безперервну циклічність між активним (періодом вимірювання) і неактивним (період очікування) станами;

6) роздільна здатність;

Найменша зміна вимірюваної величини, яку можна виявити в кількості, яку вимірює сенсор;

4.1.1 Сенсор освітлення

Розглянемо загальні принципи роботи датчику освітлення, комунікації та вимоги до його роботи. В розробці тестового прикладу системи використовується цифровий датчик освітлення BH1750FVI від компанії ROHM Semiconductor.

Цей пристрій має наступні характеристики[9]:

1) діапазон можливих значень: 1 - 65535 Люкс (Розрядність датчику - 16 біт);
2) точність результату вимірювання: 0.96-1.44%;
3) чутливість;

Варіюються від 0.45 до 3.68 Люкс. За замовчуванням, має чутливість 1 Люкс;

4) час відгуку;

Варіюється в залежності від режиму роботу сенсора (звичайний або режим одного вимірювання) та режиму роздільної здатності (найвищий, високий, низький) - від 16 мс до 120 мс;

5) режими роботи :
    1) звичайний;
    2) одного вимірювання;
6) роздільна здатність:
    1) Низький - 4 Люкс;
    2) Високий - 1 Люкс;
    3) Найвищий - 0.5 Люкс;

Розглянемо структуру сенсора (див. рис. 4.1):

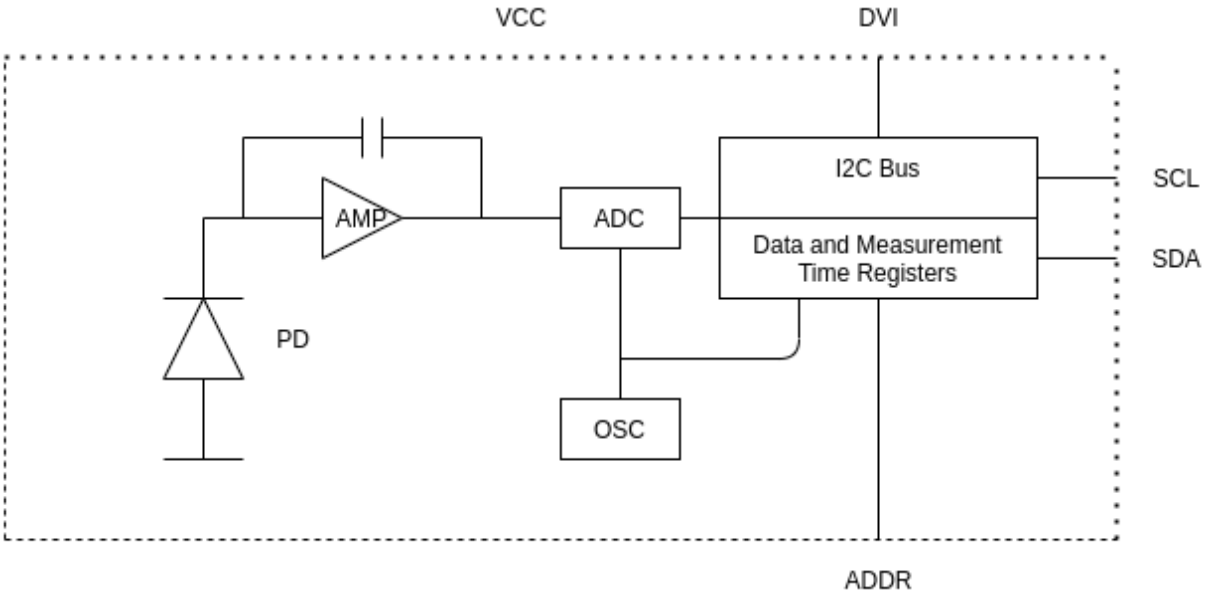

Рисунок 4.1 - Структура датчику освітлення BH1750FVI[9]

Загалом більшість датчиків освітлення мають схожу структуру:

1) фотодіод (PD - Photodiode);

Відповідає за перетворення світлової енергії в електричну;

2) операційний підсилювач (AMP - AMPLIFIER);

Підсилює аналоговий сигнал з фотодіода;

3) аналого-цифровий перетворювач (ADC);

Перетворює аналоговий сигнал в цифровий;

4) генератор тактової частоти (OSC - Oscillator - CLK);

Виробляє періодичний електронний сингал. ;

5) блок регістрів та обчислювального блоку;

Необхідні для збереження та обчислення рівня освітлення. Містить наступні регістри:

1) Регістр даних. Зберігає рівень освітлення в люксах;

2) Регістр для зберігання часу вимірювання рівня освітлення;

6) інтерфейс взаємодії;

У випадку, BHV1750FVI - I2C (див. рис. 4.2);

Сенсор освітлення BH1750FVI має наступний граф переходів стану (див. рис 4.2):

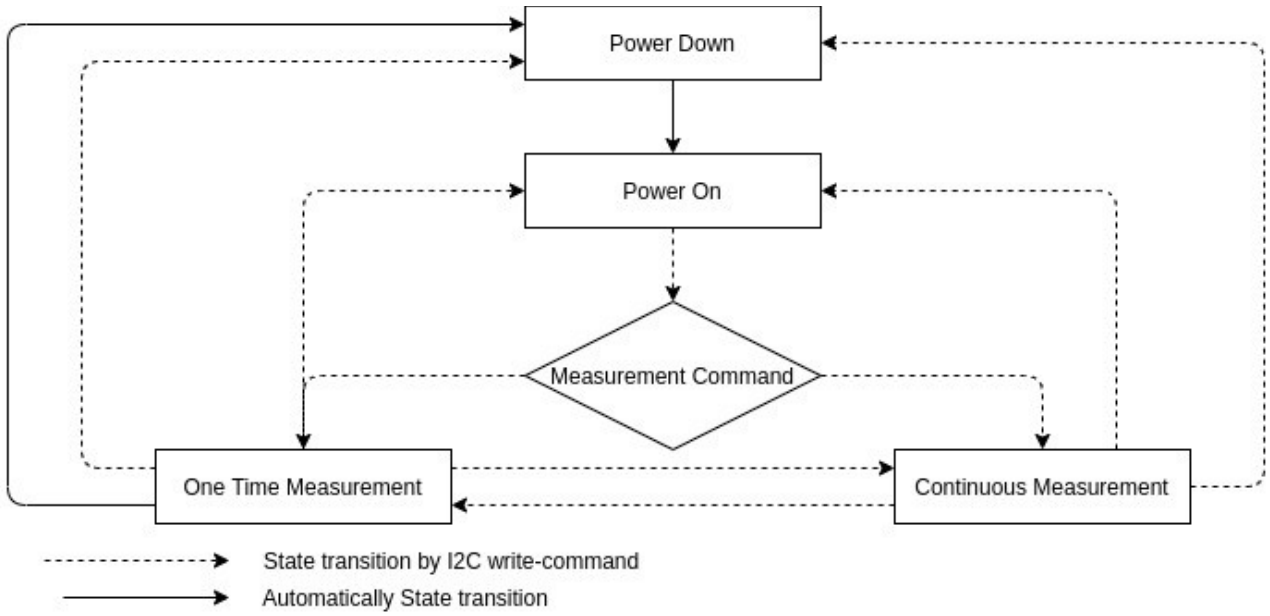

Рисунок 4.2 - Граф переходів стану сенсору BH1750FVI

Стани пристрою:

1) вимкнений (Power Down);
2) включений (Power On);
3) одиночне вимірювання (One Time Measurement);
4) постійне вимірювання (Continuous Measurement);

Зміна стану пристрою відбувається або автоматичними переходами (наприклад, автоматичний перехід з стану вимкнений в увімкнутий), або за допомогою команд, які посилаються за допомогою інтерфейсу I2C (наприклад, ініціація постійного вимірювання).

Приведемо список існуючих команд (див. табл. 4.1):

Таблиця 4.1 - Команди взаємодії з сенсором

Команда	Код операції	Результат
Вимкнення	0000 0000	Вимкнення сенсору
Увімкнення	0000 0001	Очікування операції вимірювання
Скидання	0000 0111	Скидання регістру значення. Втрата попереднього результату вимірювання
Постійне вимірювання в вищому режиму роздільної здатності	0001 0000	Постійне вимірювання з роздільної здатністю в 1 Люкс

Постійне вимірювання в найвищому режиму роздільної здатності	0001 0001	Постійне вимірювання з роздільної здатністю в 0.5 Люкс
Постійне вимірювання в нижчому режиму роздільної здатності	0001 0011	Постійне вимірювання з роздільної здатністю в 4 Люкс
Одиночне вимірювання в вищому режиму роздільної здатності	0010 0000	Одиночне вимірювання з роздільної здатністю в 1 Люкс
Одиночне вимірювання в найвищому режиму роздільної здатності	0010 0001	Одиночне вимірювання з роздільної здатністю в 0.5 Люкс
Одиночне вимірювання в нижчому режиму роздільної здатності	0010 0011	Одиночне вимірювання з роздільної здатністю в 4 Люкс

Алгоритм вимірювання освітлення:

1) надіслати код операції;

Наприклад, постійне вимірювання в вищому режиму роздільної здатності - “0001 0000”;

2) очікування виконання операції;

Максимальний час вимірювання - 180 мс;

3) зчитування результату в буфер;

Як було зазначено в характеристиках, результат має 32 розрядність. Наприклад, перший октет - “0000 0001”, другий октет - “0010 0000”;

4) форматування результату;

Трансформація результату в бінарного типу до десяткового. Поділ на коефіцієнт 1.2. Наприклад, “0000 0001 0010 0000” - $2^8 + 2^5$ = (256 + 32)/1.2 = 240 Люкс;

4.1.2 Сенсор температури, вологості та тиску

В розробці тестового прикладу системи є розрахунок на використання цифровий сенсор температури, вологості та тиску BME280 від компанії Bosch. Цей сенсор є одним із кращих по показникам точності, роздільної здатності, за можливостю конфігурування серед сегменту сенсорів низької вартості.

Має наступні характеристики[13]:

1) діапазон можливих значень;
    1) температура: від -40 ºC до 85 ºC (Розрядність 20 біт);
    2) тиск: від 300 до 1100 гектопаскаль (Розрядність 20 біт);
    3) вологість: від 0 до 100 % RH (Розрядність 16 біт);
2) точність результату вимірювання;
    a. температура: ±1-1.5 ºC ;
    b. тиск: ±1-1.7 гектопаскаль;
    c. вологість: ±3% RH;
3) час відгуку;

Варіюється в залежності від режиму роботи сенсору, вимірювального параметру та конфігурації операції вимірювання (наприклад, використання коефіцієнту рекурсивного фільтру). Загалом, час відгуку може варіюватися від 11,5 мс до 800 мс;

4) режими роботи;
    1) звичайний режим;
    2) режим одного вимірювання;
    3) режим сну;
5) роздільна здатність;
    1) температура: 0.001 ºC;
    2) тиск: 0.008%;
    3) тиск: 0.18 паскаль;

Розглянемо структуру цього датчику (див. рис. 4.3):

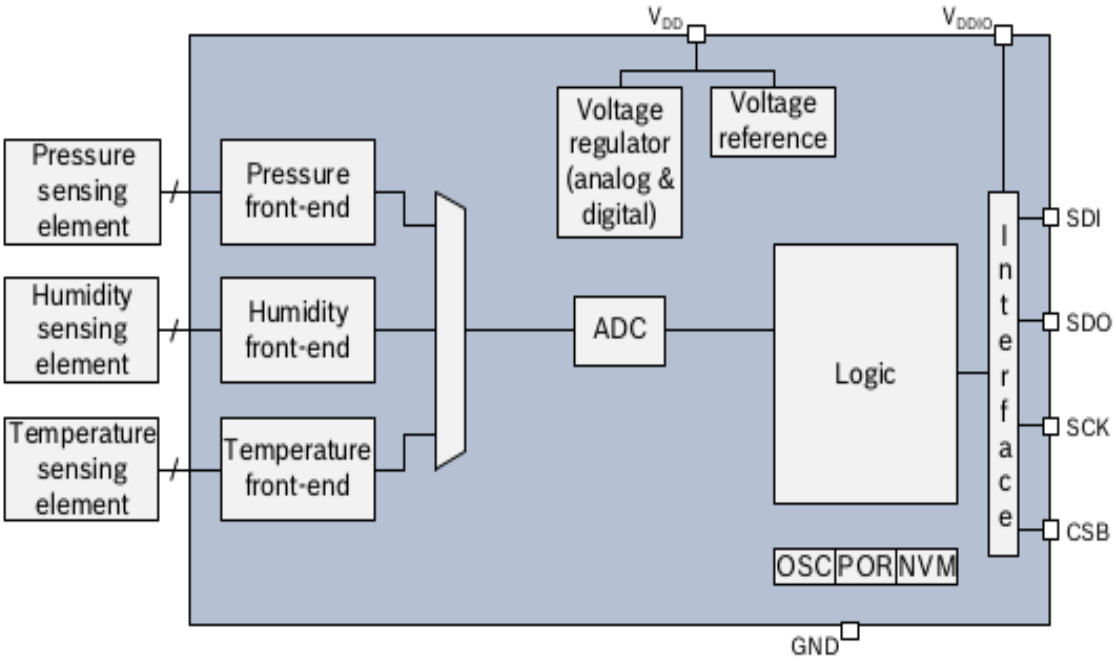

Рисунок 4.3 - Структура сенсора BME280

– pressure sensing element - датчик тиску;
– humidity sensing element - датчик вологості;
– temperature sensing element - датчик температури;
– ADC (Аналого-цифровий перетворювач);

Перетворює аналоговий сигнал в цифровий;

– logic - блок регістрів та обчислювального блоку;

Блоки необхідні для збереження та обчислення значень температури, тиску, вологості;

– OSC - генератор тактової частоти;
– POR (Power on Reset);

Вбудований генератор скидання живлення. Він скидає логічну частину та значення регістру після того, як входи V(DD) та V(DDIO) досягають мінімальних рівнів;

– NVM (Non-Volatile memory) - Енергонезалежна пам’ять;
– I2C та SPI інтерфейси;
– V(DD) – живлення для усіх аналогових та цифрових компонентів;
– V(DDIO) – живлення для цифрового інтерфейсу;

4.2 Розумна лампа

Розумна лампа - це лампа, що дозволяє здійснювати віддалене конфігурування, керування за використанням комп’ютерної мережі. Найчастіше для з'єднання з розумною лампою використовують безпровідні технології - такі як Wifi, Bluetooth, ZigBee. В ринкових домашніх системах, найчастіше використовують технологію ZigBee, яка більш підходить через то, що має невисоку вартість, низьке енергоспоживання, високу відмовостійкість. З точки зору розробки кінцевого продукту системи для керування ресурсами, використання розумної лампи з можливістю підключення через ZigBee є більш коректним вибором. Але в рамках дослідження лампа обиралась з точки зору, наявності зручного та відкритого інтерфейсу комунікації. З цих критерії найбільш доступним вибором є розумна лампа Yeelight LED Filament. Даний продукт підтримує віддалене керування за допомогою Wifi, та має наступні характеристики[3]:

- світловий потік - 35 - 700 Люмен;
- колірна температура - 2700K;
- стандарт бездротового з’єднання - WLAN IEEE 802.11 b/g/n 2.4 GHz;
- максимальна кількість TCP з'єднань – 4;

Розумна лампа підтримує наступні види взаємодії:

1) відправка команд до розумної лампи;

Команди отримання поточного стану лампи та його зміни;

2) багатоадресне оповіщення від розумної лампи (див.рис.4.4);

Лампа сповіщає інших учасників в мережі після приєднання до мережі. Адреса для багатоадресної розсилки - "239.255.255.250:1982". Дозволяє користувачам розумної лампи отримати сповіщення та підключитися до пристрою відразу при активації;

3) оповіщення о зміні стану;

Лампа сповіщає усі підключені пристрої о зміні стану (див. рис. 4.4);

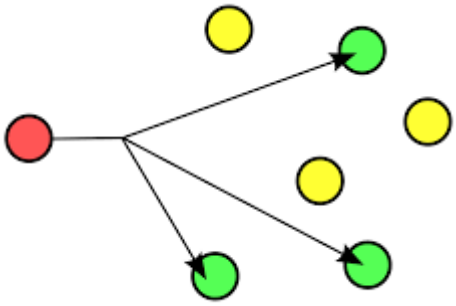

Рисунок 4.4 - Багатоадресна передача даних

При взаємодії використовуються наступні типи повідомлень:

1) команда;
2) результат виконання команди;
3) оповіщення;

Усі повідомлення надсилаються у форматі JSON за допомогою TCP з’єднання.

5 РІВЕНЬ ПЕРИФЕРІЙНИХ ОБЧИСЛЕНЬ

5.1 Стратегія збору та збереження логів

Home система може містити велику кількість пристроїв : сенсорів, актуаторів, які можуть генерувати велику кількість даних, які необхідно форматувати, також є потреба в відстеженні стану цих пристроїв. Виходячи з цього є потреба в фоновому процесі (background worker), який має наступні обов’язки:

1) формування потоків даних;
    1) запит даних з сенсорів (модель постійного моніторингу з певним інтервалом);
    2) підписка на оповіщення про зміну стану пристроїв;
2) агрегація потоків даних;
3) буферизація агрегованого потоку даних;
4) збереження даних з результуючого потоку даних;

Для вирішення наступних задач більш усього підійде парадигма реактивного програмування. Реактивність полягає в застосуванні певної дії у відповідь на певну подію. Основними складовими парадигми є асинхронний потік подій (який може містити певний результат), а також передплатник- який містить дію, як реакцію на подію. На сьогоднішній день, реактивне програмування дуже поширено використовується в клієнтським додатках (для взаємодії з користувачем), в системах з push-моделью (розсилка листів, збереження логів, тощо). В парадигмі об’єктно-орієнтованого програмування push-модель реалізується за допомогою патерну “Спостерігача” (або видавець-передплатник), класичний варіант якого був опублікований в відомій книзі “банди чотирьох”[10]. На сьогоднішній день вже розроблено багато бібліотек під різні платформи з реалізацію патерна “Спостерігача” і набором операцій для маніпулювання асинхронними потоками (Наприклад, RxJava, RxJs, RxNET, RxSwift).

Зробимо розгляд загальної структури на прикладі RxNET (див. рис. 5.1):

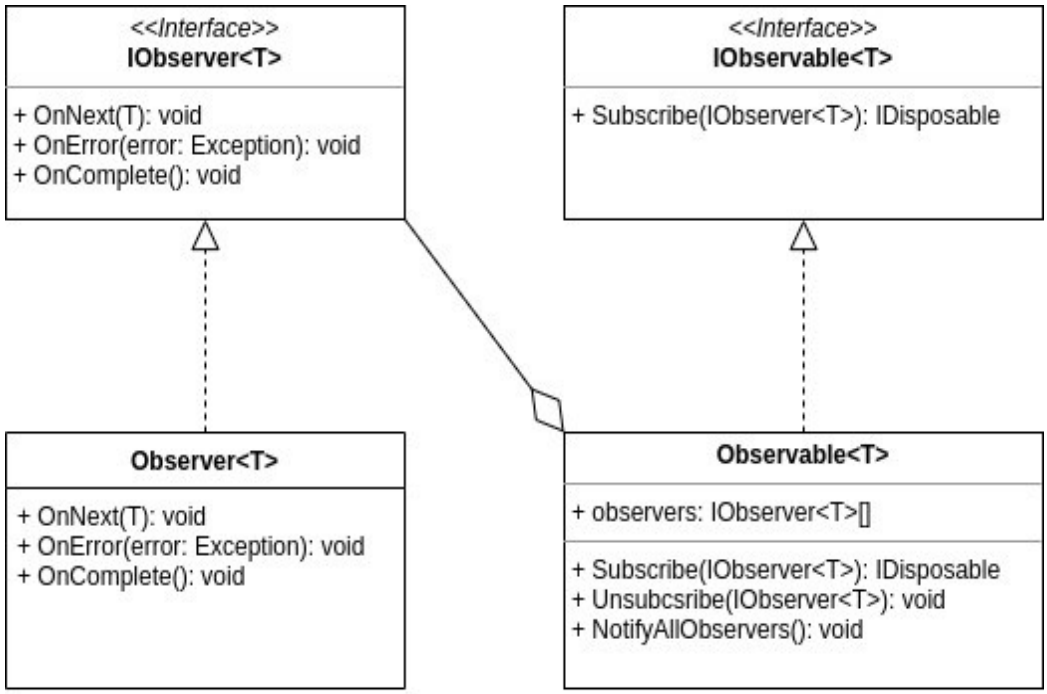

Рисунок 5.1 - Діаграма патерну “Спостерігач”

Учасники:

1) IObserver - інтерфейс спостерігача. Декларує наступні обробники;

1) OnNext - обробка успішної події;

2) OnError - обробка помилці, що не дозволила згенерувати подію;

3) OnComplete - обробка завершення черги подій;

2) IObservable - інтерфейс видавця подій. Декларує метод для підписки на потік подій. Повертає підписку, що реалізує інтерфейс IDisposable, який дозволяє спостерігачу відписатися від потоку подій;

3) Observer - імплементація спостерігача (IObserver інтерфейсу);

4) Observable - базова імплементація видавця. В базову варіанті містить список спостерігачів. Компонент, який змінює стан або обробляє повідомлення від дії користувача, ініціює метод NotifyAllObservers, який в циклі викликає обробники IObserver інтерфейсу;

Даний патерн дозволяє створювати слабо зв’язаний дизайн, де видавець події не знає деталей спостерігачів, інший компонент, що має потребу в обробці

події, також не знає деталей джерела подій. Завдяки слабо зв’язаності видавець та спостерігачі можуть знаходитися на різних рівнях абстракції. Наприклад, рівень обробки та зберігання логів системи не знає деталей рівня пристроїв - яким чином певний лог був отриманий. При використанні даного патерну треба уважно слідкувати за існуючими підписками, та відписуватися одразу, якщо потреби в обробці події від видавця вже немає. Справа в тому, що існуючі підписки продовжують посилатися на захоплені об’єкти, і ці підписки продовжують зберігатися в списку в компоненті Observer, що може привести до витіку пам’яті, тобто пам’ять не буде звільнятися з часом, хоча вже не потрібна, в деяких випадках це може призвести до того, що система не дозволить створення нових процесів або/та аварійно завершить процес з високим рівнем використання пам’яті.

5.2 Визначення структур моделей логів

Зробимо розгляд можливих потоків даних (видавців подій) в home системі:

1) потоки даних сенсорів (з рівним інтервалом, таймер)
    1) освітлення;
    2) температура;
    3) вологість;
2) потоки даних через підписку на оповіщення
    1) зміна стану розумної лампи;
    2) зміна стану розумного кондиціонування;
    3) поява людини в середовищі;

На даний момент, в системі є потреба в спостерігачі, який відповідатиме за збереження даних в певне сховище або термінал користувача. Цей спостерігач буде відстежувати зміни за усіма потоками даних від сенсорів та актуаторів.

Кожний видавець генерує лог певної структури о стані сенсора або актуаторі. Усі логи мають три загальних атрибута - час, ідентифікатор

логу, ідентифікатор пристрою. Їх структура виглядає наступним чином (див. рис. 5.2):

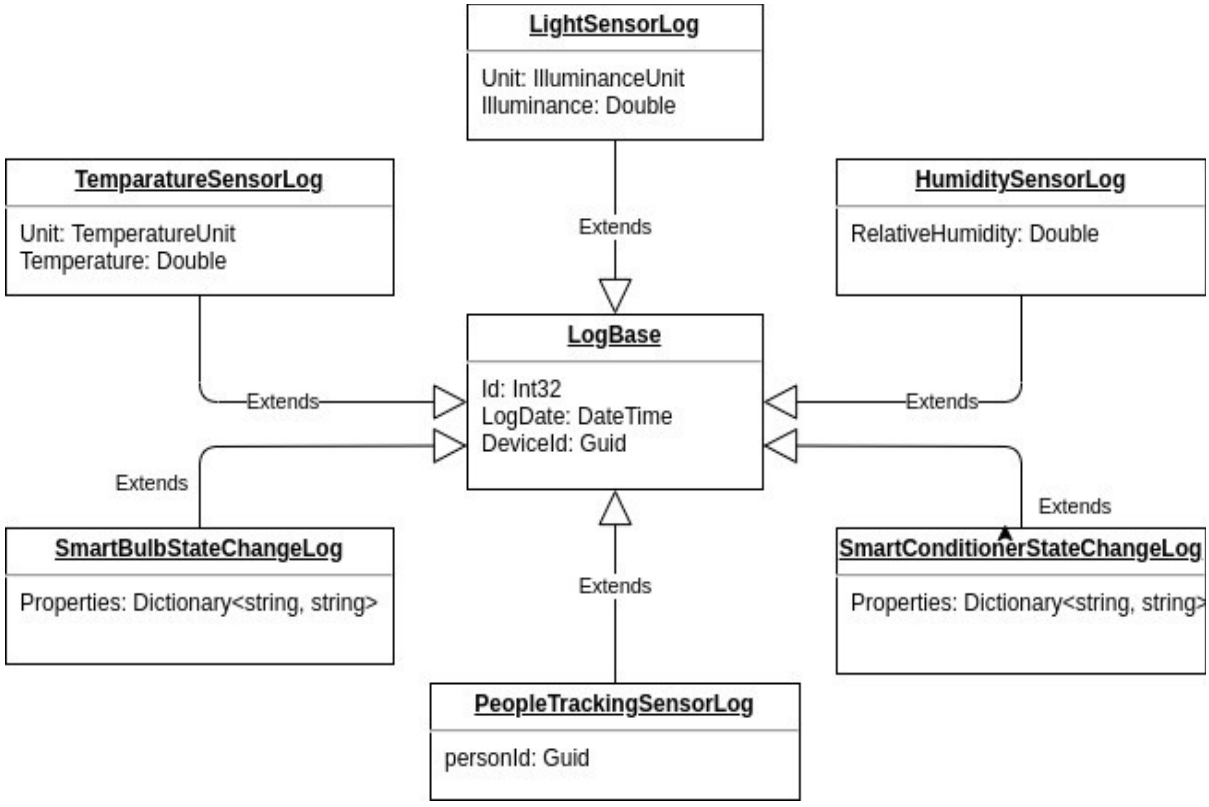

Рисунок 5.2 - Структура моделей логів

1) LogBase - базова структура від якої унаслідуються всі інші моделі;

1) Id - ідентифікатор запису певного стану. Може представлятися як цілочисельний ідентифікатор, який буде генеруватися за допомогою об’єкту Sequence бази даних. Також альтернативним типом для ідентифікатору є GUID (або UUID) - статистичний унікальний 128-бітний ідентифікатор. Цей тип дозволяє запобігти перевірку на обмеження унікальності від генератора (типу Sequence) і в цілому через структуру має більш унікальну складову, але в той же час він більше за розміром (128 біт проти 32), та має більш складний алгоритм генерування ніж генератор для цілочисельного ідентифікатору. З точки зору кількості логів та частоти їх генерування - цілочисельний ідентифікатор є більш оптимальним вибором, ніж GUID;

2) LogDate - дата та час, коли був отримано стан певного пристрою або кліматичної величини з сенсору;

3) DeviceId - ідентифікатор певного пристрою в системі. Має такий же тип і альтернативний тип, що і ідентифікатор запису (логу) певного стану. Але треба зауважити, що зазвичай з точки зору зберігання кількість сенсорів не є велика, також пристрої не добавляються в систему занадто часто. Тому в цьому випадку GUID є більш вдалим вибором типом даних для цього атрибуту;

2) LightSensorLog - структура записи логу від сенсора освітлення;

1) IlluminanceUnit - одиниця виміру освітлення (кілолюкс, люкс, мегалюкс, мілілюкс);

2) Illuminance - значення у вигляді дійсного числа;

3) TemperatureSensorLog - структура запису логу від сенсора температури;

1) TemperatureUnit - одиниця виміру температури (градус Цельсія, градус Фаренгейта, Кельвін, тощо);

2) Temperature - значення у вигляді дійсного числа;

4) SmartBulbStateChangeLog - структура запису логу о зміні стану розумної лампи;

1) Properties - Dictionary<string, string>, структура “ключ-значення”, де ключ - змінна характеристика, значення - величина на яку була виконана зміна; Наприклад, при увімкненні розумної лампи ключ – це “power”, значення – “on”.

5) SmartConditionerStateChangeLog - структура запису логу о зміні стану розумної системи кондиціонування;

1) Properties - Dictionary<string, string> , структура “ключ-значення”, де ключ - змінна характеристика, значення - величина на яку була виконана зміна;

6) PeopleTrackingSensorLog - структура запису логу о появі/втраті людини з середовища;

1) PersonId – GUID;

5.3 Визначення компонентів фонового процесу

Загальна UML діаграма компонентів фонового процесу буде мати наступний вигляд (див. рис. 5.3):

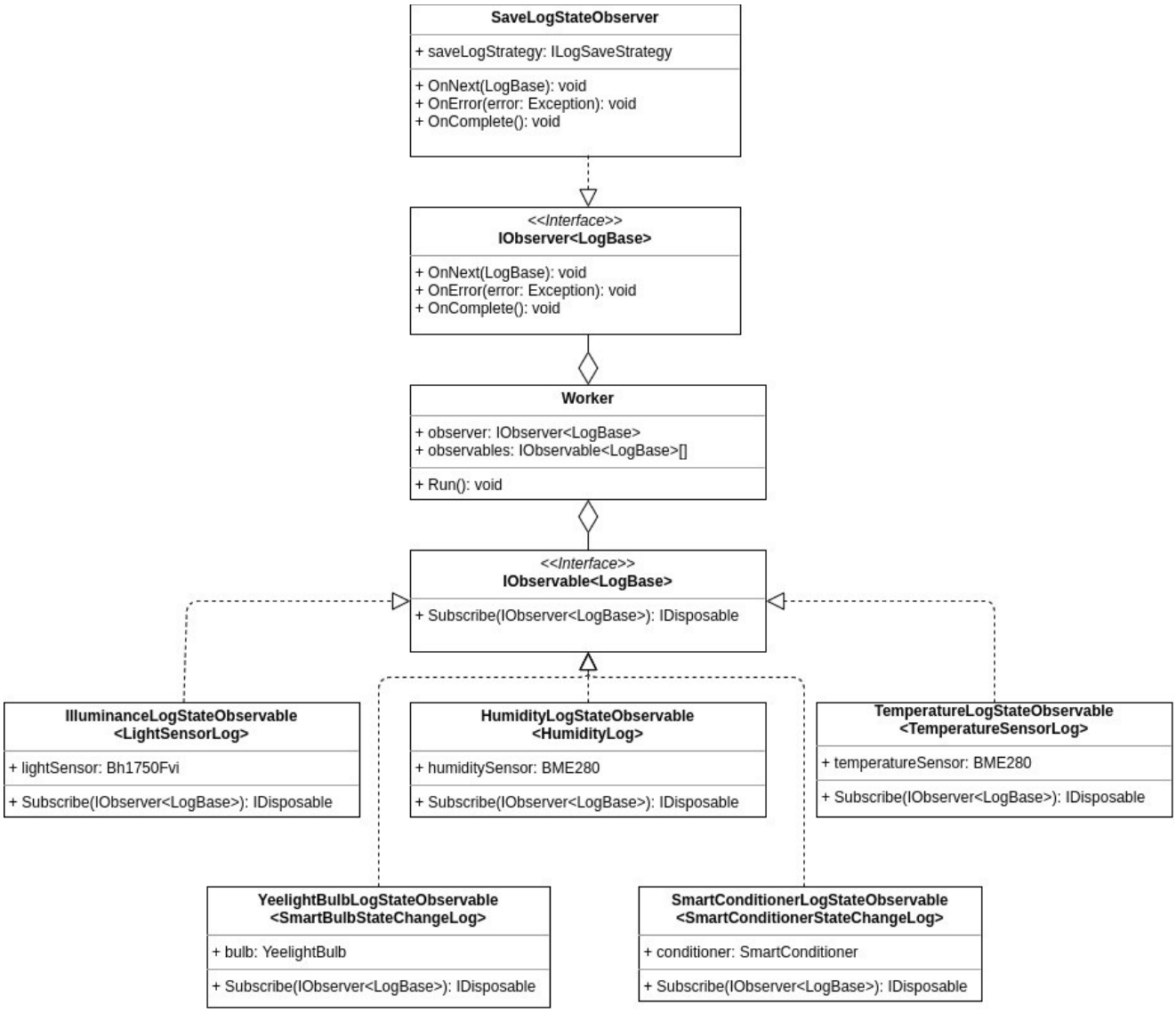

Рисунок 5.3 - UML діаграма компонентів фонового процесу моніторингу

Зробимо опис вказаних компонентів-імплементацій:

– Worker компонент.

Виконує функцію контролеру. Запускає та зупиняє фоновий процес моніторингу стану середовища. Відповідає за створення контексту виконання фонового процесу. Контекст виконання містить активні сенсори та актуатори. Формує колекцію видавців, що ініціюють подію отримання лога о стані сенсора або актуатора, та одного спостерігача, що зберігає логи в певне сховище або

термінал користувача. Процес підписки та відписки спостерігача, що зберігає логи, відбувається також в компоненті Worker. Відписка відбувається разом з знищенням контексту виконання, і можлива при двох обставинах: зупинці процесу або викиданні програмного виключення. Не виконання знищення контексту і відписці призведе до витіку ресурсів. У випадку контексту – це витік системних ресурсів таких, як сокет, файл, I2C або SPI з'єднання, що насправді може призвести до серйозних проблем (наприклад, відмова в виділенні ресурсів новим процесам) не тільки в системі, де працює фонова програма, но і з серверами, з якими існують активні з'єднання. Наприклад, база даних має встановлений ліміт відкритих з'єднань; після його перевищення сервер бази даних буде відповідати відмовою саме новим запитам на з'єднання, у більшості випадків ця ситуація потребує втручання системного адміністратора, для того, щоб явно закрити активні з'єднання, якщо існує робоче з'єднання або є можливість з'єднатися з сервером, або в іншому випадку повністю перезавантажити процес;

– IlluminanceLogStateObservable - видавець подій о стані сенсора освітлення;

– TemperatureLogStateObservable - видавець подій о стані сенсора температури;

– HumidityLogStateObservable - видавець подій о стані сенсора вологості;

– YeelightBulbLogStateObservable - видавець подій о стані розумної лампи;

– SmartConditionerLogStateObservaable - видавець подій о стані розумного кондиціонування;

– PeopleTrackingLogStateObservable - видавець подій о появі/втраті людини з середовища;

– SaveLogStateObserver - спостерігач, що відповідає за зберігання логів від вище перерахованих видавців в певне сховище або термінал користувача для діагностичних цілей;

5.4 Формування потоку даних для операції збереження

Результуючий потік формується за наступною послідовністю операторів (див. рис 5.4):

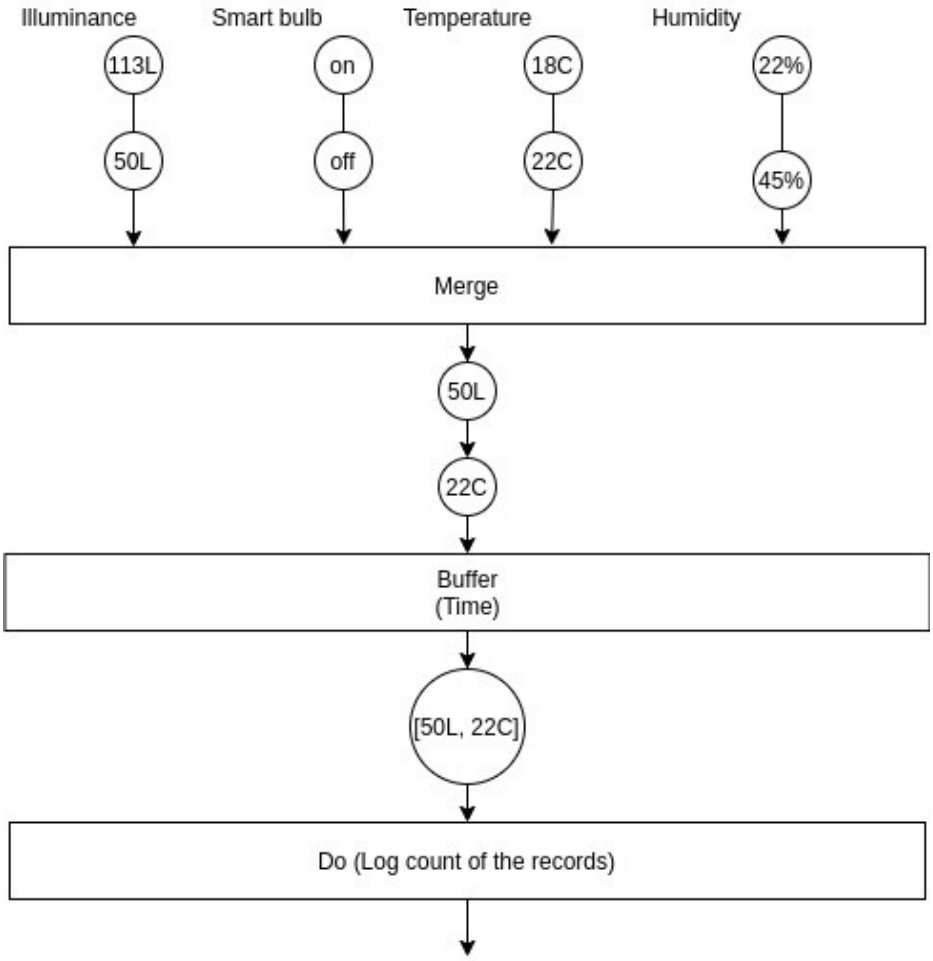

Рисунок 5.4 - Формування потоку даних для операції збереження

На діаграмі 5.4 вказані декілька операторів, які поширені в використанні парадигми реактивного програмування:

1) Оператор об'єднання потоків - Merge (комбінуючий оператор);

Цей оператор об'єднує декілька видавців в один, об'єднавши їх оповіщення о події. Необхідність цього оператора здається очевидної - для забезпечення одного видавця для одного спостерігача;

2) Оператор буферизації - Buffer (трансформуючий оператор);

Цей оператор періодично збирає події з видавця в певні пачки і видає ці пачки, і не видає елементи по одному. Періодичність залежить від певної події

або певного часового інтервалу, або кількості елементів в одній пачці. Цей оператор може бути використаний для запобігання постійного навантаження на сервер бази даних або файл. Сервер бази даних, також як і файл використовують певні системні ресурси. В нашому випадку, періодичність буде залежить від певного часового інтервалу;

3) Оператор дії – Do;

Цей оператор реєструє певну дію щодо події перед підпискою;

У цьому випадку реалізація компоненту Worker матиме наступний вигляд(див. лістинг. 5.1):

Лістинг 5.1 - Формування та підписка на потік даних для збереження логів

```
DeviceEnvironment deviceEnvironment = await CreateDeviceEnvironmentAsync();

IEnumerable<IObservable<LogBase>>            temperatureObservables            =
GetTemperatureSensorObservables(deviceEnvironment);

IEnumerable<IObservable<LogBase>>              humidityObservables             =
GetHumiditySensorObservables(deviceEnvironment);

IEnumerable<IObservable<LogBase>>            lightSensorObservables            =
GetLightSensorObservables(deviceEnvironment);

IEnumerable<IObservable<LogBase>>               bulbObservables                =
GetYeelightBulbObservables(deviceEnvironment);

IEnumerable<IObservable<LogBase>>            conditionerObservable             =
GetConditionerSensorObservables(deviceEnvironment);

IEnumerable<IObservable<LogBase>>       peopleTrackingLogStateObservable       =
GetPeopleTrackingLogStateObservable(deviceEnvironment);

IObserver<IList<LogBase>>            saveLogObserver            =            new
LogStateIntoDbObserver(_serviceProvider);


IEnumerable<IObservable<LogBase>> observables = temperatureObservables

      .Concat(humidityObservable)

      .Concat(lightSensorObservables)

      .Concat(bulbObservables)

      .Concat(conditionerObservable)

      .Concat(peopleTrackingLogStateObservable);
```

```
observables
      .Buffer(TimeSpan.FromMinutes(_saveLogTimeIntervalInMinutes))
      .Do(logs => _logger.LogInformation($"{logs.Count} have been collected!"))
      .Subscribe(observer);
```

В цій реалізації DeviceEnvironment - класс, який інкапсулює набір доступних пристроїв в системі. Далі формуються колекції потоків даних пристроїв. IEnumerable - інтерфейс колекції, що містить метод GetEnumerator(), який забезпечує доступ до ітератору. GetTemperatureSensorObservables - колекція потоків даних від сенсорів температури, GetHumiditySensorObservables - від сенсорів вологості, GetLightSensorObservables - від сенсорів освітлення, GetYeelightBulbObservables - від розумних ламп, GetConditionerSensorObservables - від розумної системи кондиціонування. Далі усі колекції об’єднуються в одну колекцію потоків даних за допомогою LINQ оператора Concat. Наступним кроком усі потоки даних об’єднуються в один за допомогою оператора Merge. Результуючий потік проходить буферизацію за часовим інтервалом, тобто буферизований потік видасть пачку логів після певного часового інтервалу, який задається в конфігурації процесу.

5.5 Реалізація спостерігача - компонента збереження логів

У випадку імплементації спостерігача треба врахувати, що система може мати необхідність зберігати дані не тільки в одне сховище та мати можливість його замінити. Так на старті роботи системі може використовуватися збереження в файл, то з розширенням системи може з’явитися потреба на перехід до серверу бази даних, або замінити певну СУБД на іншу (наприклад, MS SQL Server на Postgres). Задачу інкапсуляції алгоритму можна вирішити за допомогою патерну “Стратегія” (див. рис. 5.5):

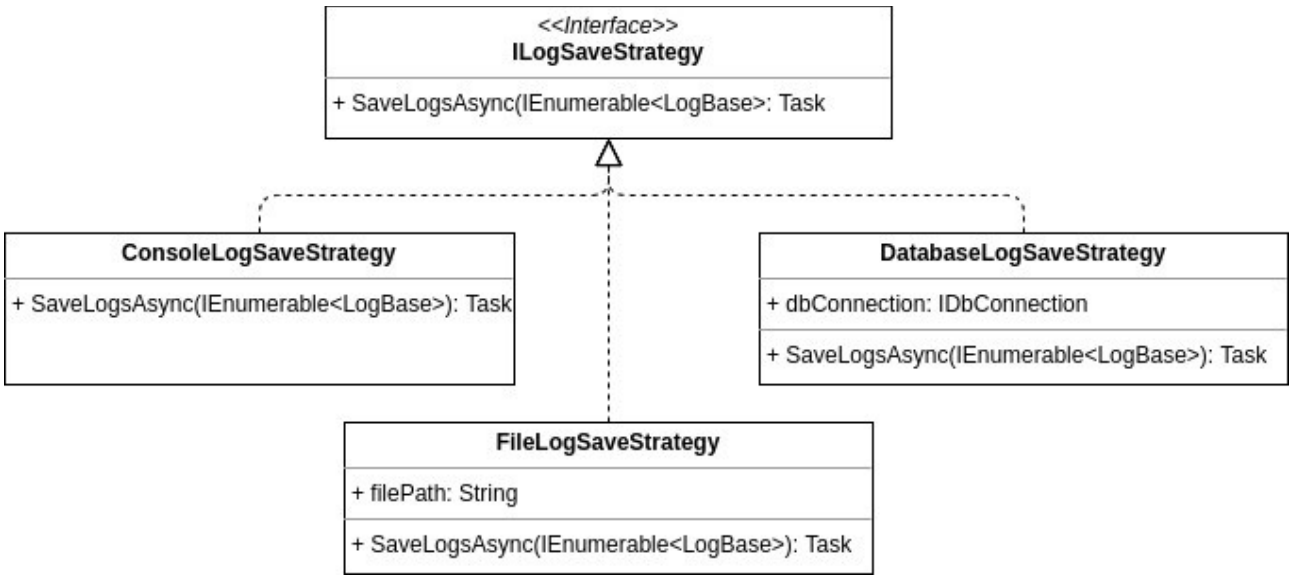

Рисунок 5.5 - Стратегії збереження логів

Інтерфейс ILogSaveStrategy декларує асинхронний метод збереження логів, що приймає колекцію, що реалізує патерн “Ітератор”, та повертає Task, тип що репрезентує асинхронну операцію.

Загалом, використовують наступні імплементації стратегії збереження логів:

– ConsoleLogSaveStrategy - стратегія виконує запис логів в термінал діагностичного процесу. Використовується при діагностиці фонового процесу моніторингу;

– FileLogSaveStrategy - стратегія виконує запис логів в файл. Компонент приймає путь до файлу;

– DatabaseLogSaveStrategy - стратегія зберігає записи в базу даних. Приймає та спирається для збереження на з’єднання, що реалізує інтерфейс IDbConnection, представляє підключення до джерела даних і реалізується постачальниками даних .NET (наприклад, Npgsql), які отримують доступ до реляційних баз даних;

Використання асинхронній операції у випадку бази даних чи файлу дозволяє запобігти блокування потоку. Мається на увазі, що у випадку синхронії операції потік виконання буде очікувати результату під час запису інформації (у більшості випадків на диск). Асинхронна операція дозволяє

відпустити потік виконання на переключення до іншого контексту для виконання активних розрахунків, а очікуванням переривання з системи вводу-виведення займається інший спеціальний потік (IO thread), який витрачає значно менше процесорних ресурсів, ніж звичайний потік (CPU thread).

Конкретна реалізація підключення до джерела даних забезпечуються за допомогою патерну “Абстрактна Фабрика” [11], що представляє собою стратегію створення сімей взаємопов'язаних або родинних об’єктів (див. рис. 5.6).

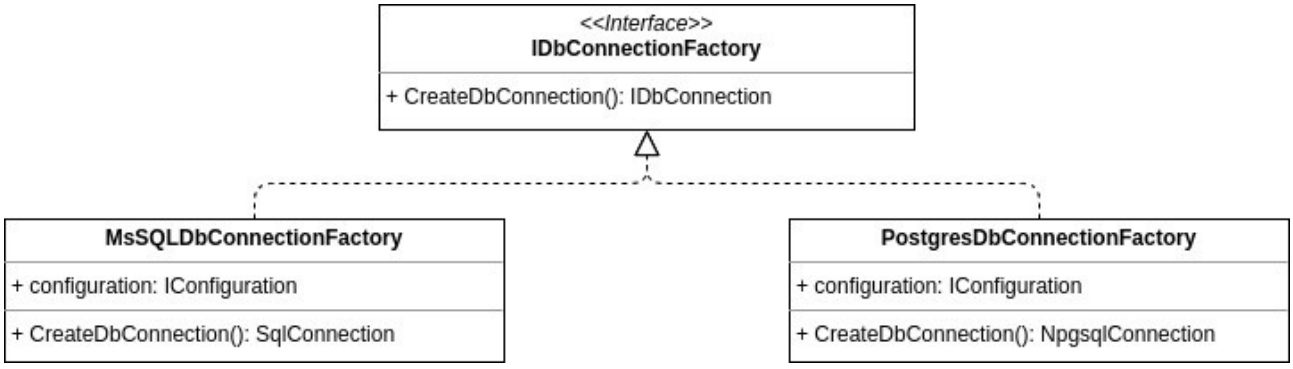

Рисунок 5.6 - Фабрики створення з’єднання до бази даних

Учасники:

1) IDbConnectionFactory - інтерфейс, що декларує метод створення абстрактного з’єднання, що реалізує інтерфейс IDbConnection;

2) PostgresDbConnectionFactory - реалізація інтерфейсу IDbConnectionFactory для забезпечення доступу до бази даних Postgres за допомогою постачальника даних Npgsql. Створює з’єднання типу NpgsqlConnection;

3) MsSqlDbConnectionFactory - реалізація інтерфейсу IDbConnectionFactory для забезпечення доступу до бази даних MS SQL Server за допомогою постачальника даних System.Data.SqlClient. Створює з’єднання типу SqlConnection;

Імплементації IDbConnectionFactory - створюються з передачею інформації о конфігурації проекту - IConfiguration. Цей об’єкт містить набір конфігурацій для підключення - “ConnectionString”. “ConnectionString” - містить адрес чи хост серверу бази даних, порт процесу бази даних, ім’я бази даних, дані для автентифікації та авторизації (наприклад, логін та пароль).

В реалізаціях ILogSaveStrategy - ConsoleLogSaveStrategy, FileLogSaveStrategy, DbLogSaveStrategy є проблема з тим, як зберегти логи, якщо тип логів, які отримує стратегія, є ILogBase. Для збереження логу в базу даних нам необхідно знати тип (LightSensorLog, HumiditySensorLog, TemperatureSensorLog, SmartConditionerStateChangeLog, SmartBulbStateChangeLog) тому що:

– у випадку бази даних є потреба сформувати запит, де необхідно вказати схему, ім’я таблиці логу, та колонки. Специфіка типу очевидна;

– у випадку файлу та консолі також є схожа проблема, кожний тип описується в різному форматі;

У випадку файлу та консолі ідея добавити логіку форматування моделі в символьний рядок в кожний тип через перевизначення методу ToString ще може бути допустимою, хоча форматування моделі в символьний рядок в файлі та консолі може відрізнятися, і в цьому випадку додавання методів ToConsoleString та ToFileString в моделі вже порушує принцип єдиного обов’язку компоненту [11]. Цей принцип гласить, що «У модуля чи класу повинна існувати тільки одна проблема для зміни» [11]. Використання таких функція, як ToConsoleString та ToFileString в класі моделі логу вже призводить до того, що існує і буду існувати декілька причин для зміни. Наприклад, зміна типу певного атрибута, зміна формату виводу до консолі користувача, зміна формату виводу до файлу.

Аналогічна проблема у випадку бази даних додавання методів для отримання скрипту або інформації для його генерування – використання такого методу як ToSqlScript або GetSqlCommand виглядає ще навіть більшим порушенням принципів проектування.

Для вирішення цієї проблеми в парадигмі об’єктно-орієнтованого програмування існує патерн “Відвідувач” (див. рис. 5.7):

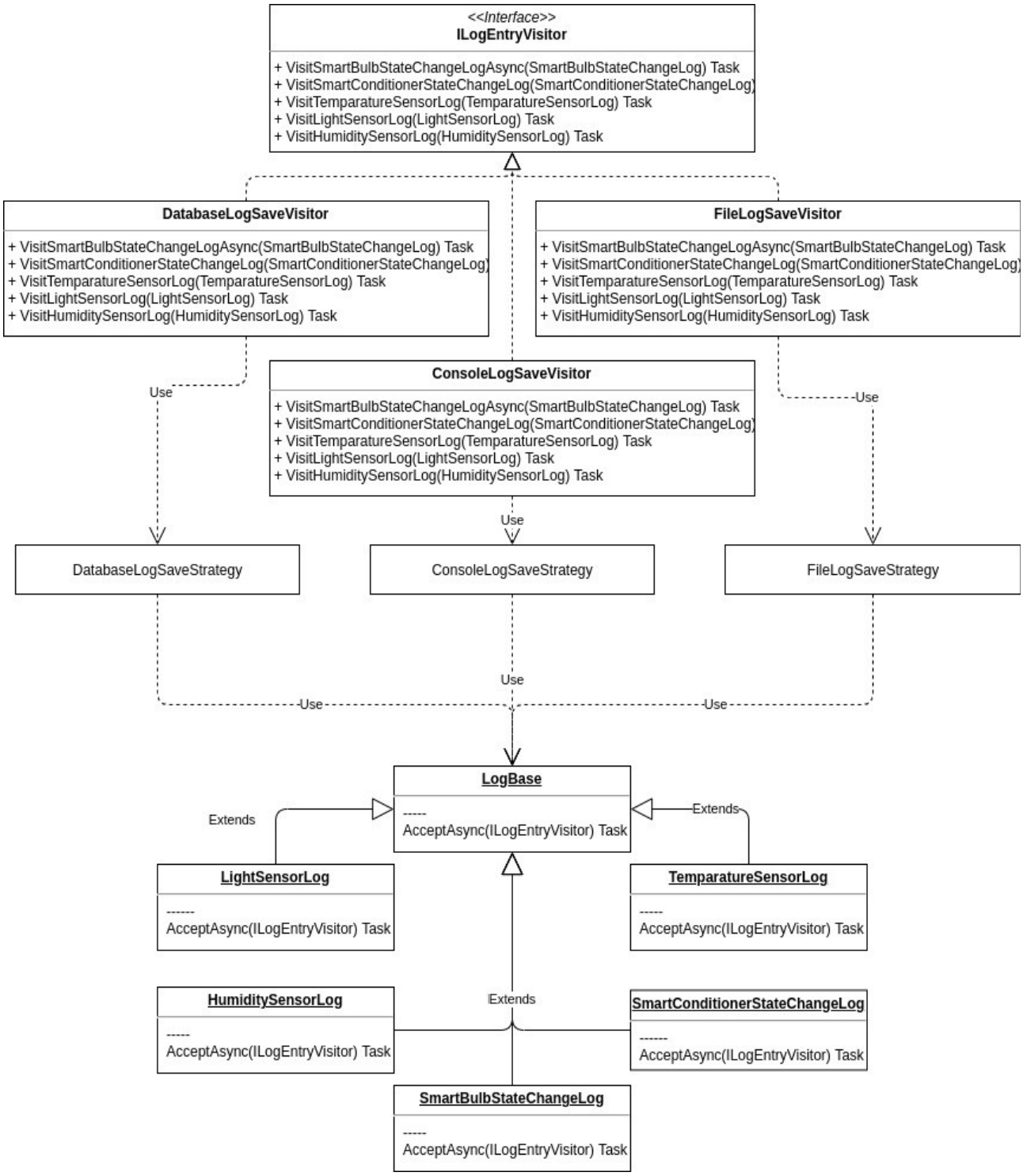

Рисунок 5.7 - Відвідувачі збереження логів

Учасники:

1) ILogEntryVisitor - інтерфейс, що декларує методи для відвідування кожної моделі логу;

2) Релізації ILogEntryVisitor інтерфейсу;

1) DatabaseLogSaveVisitor - збереження одного логу в базу даних;

2) ConsoleLogSaveVisitor - збереження одного логу в термінал діагностики;

3) FileLogSaveVisitor - збереження одного логу в файл;

3) Моделі логів. (LogBase, LightSensorLog, HumiditySensorLog та інші). Базовий клас (LogBase) повинен містити операцію AcceptAsync, який приймає об’єкт, що реалізує інтерфейс ILogEntryVisitor з викликом операцій VisitAsync у цього об’єкту;

4) Клієнти:

1) DatabaseLogSaveStrategy.

Відкриває з’єднання до бази даних, створює об’єкт типу DatabaseLogSaveVisitor, і в циклі виконує операцію AcceptAsync у створеного об’єкта;

2) ConsoleLogSaveStrategy.

Створює об’єкт типу ConsleLogSaveVisitor, і в циклі виконує операцію AcceptAsync у створеного об’єкта;

3) FileLogSaveStrategy.

Відкриває та блокує файл, створює об’єкт типу FileLogSaveVisitor, та в циклі виконує операцію AcceptAsync у створеного об’єкта;

Наприклад, операція VisitAsync у DatabaseLogSaveVisitor класу для типу LightSensorLog матиме наступну реалізацію (див.лістинг 5.2).

Лістинг 5.2 - Операція VisitAsync у DatabaseLogSaveVisitor класу для типу LightSensorLog

```
public Task VisitAsync(SmartHub.LogModels.LightSensorLog lightSensorLog)
{
    LightSensorLog entity = new LightSensorLog()
    {
        Unit = lightSensorLog.Unit.ToString(),
        Value = lightSensorLog.Value.Value,
```

```
        LogDate = lightSensorLog.LogDate
    };

    return _lightSensorLogCommand.InsertAsync(entity);
}
```

Виклик цієї операції з моделі логу (див.лістинг 5.3):

Лістинг 5.3 - Виклик операції VisitAsync в методі AcceptAsync для типу LightSensorLog

```
public class LightSensorLog : LogBase
{
    public IlluminanceUnit Unit { get; set; }
    public Illuminance Value { get; set; }

    public override Task AcceptAsync(ILogEntryVisitor visitor)
    {
        return visitor.VisitAsync(this);
    }
}
```

Модель перевизначає метод AcceptAsync, що задекларований як абстрактний (тобто не містить реалізації) в базовому класі LogBase, таким чином що передає себе в виклик методу VisitAsync.

Клієнтом даного методу (AcceptAsync) є метод збереження колекції логів в певне сховище в кожній стратегії. Наприклад, у випадку збереження логу до бази даних це буде мати наступний вигляд (див.лістинг 5.4).

Лістинг 5.4 - Операція збереження колекції логів в DatabaseLogSaveStrategy стратегії

```
public async Task SaveLogsAsync(System.Collections.Generic.IEnumerable<SmartHub.LogModels.LogBase> logList)
{
    using (IServiceScope scope = _serviceProvider.CreateScope())
    {
            IDbConnection dbConnection = scope.ServiceProvider.GetRequiredService<IDbConnection>();
```

```
            dbConnection.Open();

            DatabaseLogSaveVisitor saveLogVisitory = new DatabaseLogSaveVisitor(
            scope.ServiceProvider.GetRequiredService<ICommandAsync<LightSensorLo
            g>>()
            scope.ServiceProvider.GetRequiredService<ICommandAsync<BulbChangeSta
            teLog>>(),
                scope.ServiceProvider.GetRequiredService<ICommandAsync<StateChan
            geProperty>>()
            );

        foreach (SmartHub.LogModels.LogBase log in logList)
        {
            await log.AcceptAsync(saveLogVisitory);
        }
    }
}
```

Збереження логів до базу даних має наступну послідовність дій: створюємо окрему область для використання з'єднання («Scope»), створюємо з'єднання до бази даних використовуючи абстракцію IDbConnection (за якою створюється та конфігурується з'єднання до певного постачальника даних), виконується відкриття з'єднання, створення відвідувача та виконання команд.

5.6 Взаємодія з сенсорами та актуаторами

Відстеження стану пристроїв, відправка команд на зміну стану або виконання операцій вимірювання здійснюється за допомогою окремих бібліотек. В рамках розробки та дослідження був розрахунок на наступні сенсори та актуатори:

- Сенсор освітлення - BH1750FVI. З'єднання - I2C;
- Сенсор температури, вологості та тиску - BME280. З'єднання - I2C;
- Розумна лампа - Yeelight LED Filament. З'єднання - Сокет Берклі;

Для роботи з сенсорами маємо можливість використовувати пакет - .NET Core IoT від компанії Microsoft, який містить бібліотеку Iot.Device.Bindings, яка містить набір інтерфейсів для комунікації з різними IoT пристроями,

в тому числі і BH1750FVI та BME280. У випадку Yeelight LED Filemanent була створена окрема бібліотека з інтерфейсом для комунікації використовуючи програмний інтерфейс – Сокет Берклі. UML- діаграма цих компонентів виглядає наступним чином (див. рис. 5.8):

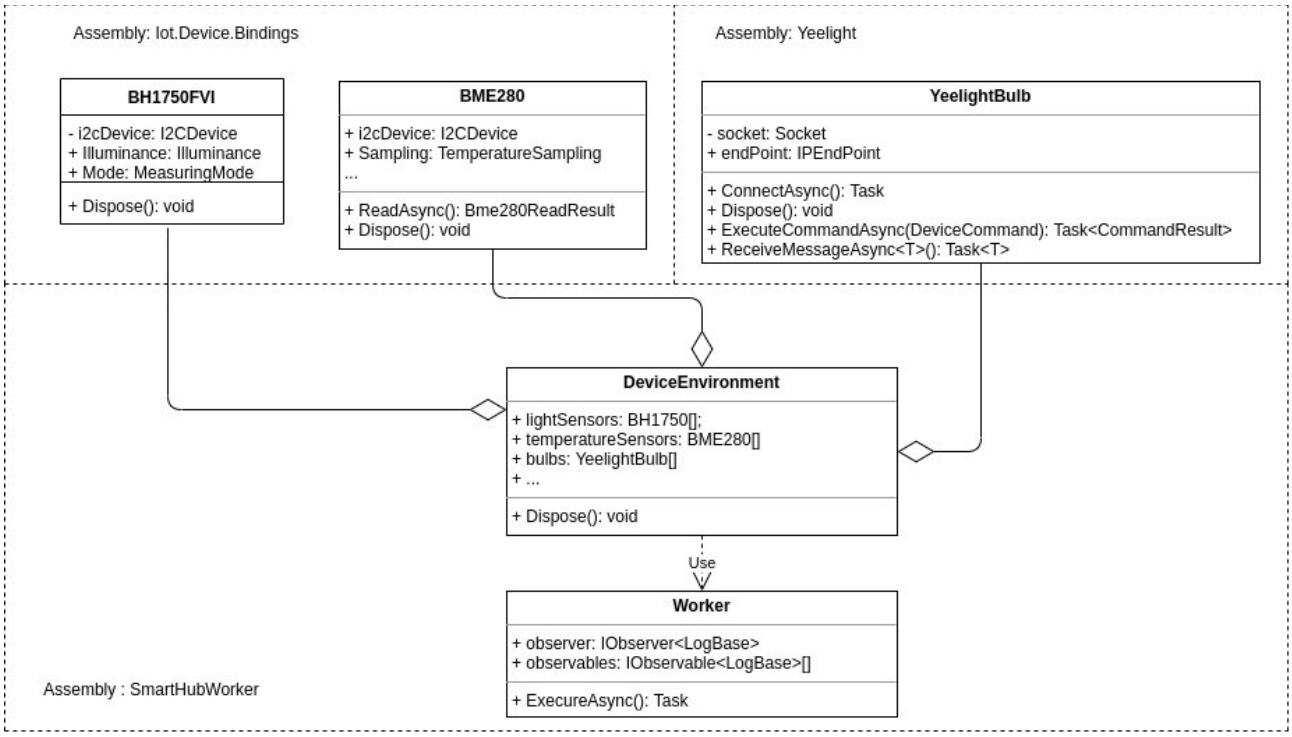

Рисунок 5.8 - Класи пристроїв (актуаторів та сенсорів)

1) Iot.Device.Bindings збірка;

1) BH1750FVI - сенсор освітлення;

I2C з'єднання к сенсору здійснюється в процесі створення об'єкту. Для відключення I2C з'єднання необхідно явно здійснити операцію Dispose. Точність результату виміру задається при ініціалізації об'єкту (найвищий, високий та низький) та режим (одиночного та постійного вимірювання). Вимірювання відбувається при зверненні к властивістю Illuminance;

2) BME280 - сенсор температури, тиску, вологості;

Також як, і в BH1750FVI, I2C з'єднання к сенсору здійснюється в процесі створення об'єкту, а відключення при виклику Dispose. Одразу всі характеристики (температура, тиск, вологість) можна виміряти та отримати за допомогою асинхронного методу ReadAsync(). Також є можливість вимірювати певні характеристики за допомогою синхронних

методів - TryReadHumidity (вологості), TryReadPressure(тиск), TryReadTemperature(температура). Також є метод, який дозволяє змінити режим сенсору (одиночне вимірювання та постійне вимірювання, режим сну) – SetPowerMode;

2) Yeelight збірка

1) YeelightBulb - розумна лампа;

Компонент використовує сокет берклі з TCP з'єднанням. Для створення об'єкту необхідно передати IPv4 адресу та порт пристрою. Для з'єднання необхідно явно викликати асинхронну операцію ConnectAsync, для роз'єднання - Dispose. Для комунікації використовує два методи: ExecuteCommandAsync для відправки повідомлень до розумної лампи, ReceiveMessageAsync для отримання повідомлення від розумної лампи. Операція ExecuteCommandAsync повертає вже повідомлення від розумної лампи після обробки команди. За замовчуванням, повідомлення очікується протягом 100 мс, потім викидається програмний виняток;

Усі об'єкти для керування пристроями (сенсорами та актуаторами) створюється при старті процесу, створюється об'єкт DeviceEnvironment, який інкапсулює в собі всі пристрої. Об'єкт DeviceEnvironment вже використовується для завдань моніторингу, збереження логів та активації актуаторів внаслідок ініціації від компоненту прогнозування. Усі неконтрольовані середою .NET ресурси (I2C з'єднання та сокети) знищується під час завершення процесу.

6 КОМПОНЕНТ ПРОГНОЗУВАННЯ

6.1 Аналіз проблеми прогнозування

Компонент прогнозування використовується в системі для наступних задач:

1) прогнозування найбільш комфортного рівня кліматичних ресурсів;
2) прогнозування параметрів та час для активації актуаторів в системі для задоволення комфортного рівня кліматичних ресурсів;

Прогнозовані одиниці кліматичних ресурсів:

1) Рівень освітлення на робочому місці (Люкс);
2) Рівень температури повітря (Градус Цельсія);
3) Рівень відносної вологості повітря (Відношення, процент);

Прогнозовані параметри для активації актуаторів:

1) Розумна лампа;
    1) Рівень світлового потоку (Люмен);
    2) Рівень яскравості;
2) Система розумного кондиціонування;
    1) Рівень температури (Градус Цельсія);
    2) Рівень вологості повітря (Відношення або процент);

Треба зауважити, що компонент прогнозування відповідає безпосередньо за прогнозування певних параметрів використовуючи вхідну модель та дані логів попередніх станів системи, але не відповідає за ініціацію вмикання актуаторів, та не має доступу до інтерфейсу взаємодії з рівнем пристроїв. За ініціацію вмикання актуаторів відповідає інший компонент (див. розділ 7), який є клієнтом компоненту прогнозування.

Компонент прогнозування необхідно розробити, як окремий компонент розгортання, для того щоби існувала можливість змінити логіку, модель чи алгоритм прогнозування без змін та необхідності збору інших компонентів.

6.2 Прогнозування параметрів вмикання штучного освітлення

6.2.1 Розробка стратегії вмикання штучного освітлення

Прогнозування комфортного рівня освітлення спирається на історію користування розумною лампою та даних з сенсора освітлення. Головною ознакою того, що рівень є не комфортним є подія о вмиканні штучного освітлення в середовищі. Тому є можливість висунути наступні визначення:

– некомфортний рівень освітлення - значення освітлення в певному середовищі, яке передує вмикання штучного освітлення.

– комфортний рівень освітлення - значення освітлення в певному середовищі, яка встановлюється після вмикання штучного освітлення.

Враховуючи, що штучне освітлення може вмикатися випадково, треба брати середнє значення у проміжок усієї історії користування штучним освітленням.

$$x = \frac{\sum_{i=1}^{n} x_i}{n} \tag{6.1}$$

де x_i – значення одного елемента, n – кількість елементів, x – середнє арифметичне;

Наприклад, історія за деякі дні в період 18 квітня до 4 травня (див. табл. 6.1)

Таблиця 6.1 - Значення комфортного та некомфортного рівнів освітлення за деякі дні в період 18 квітня до 4 травня

Час отримання некомфортного рівня освітлення	Час отримання комфортного рівня освітлення	Час вмикання розумної лампи	Некомфортний рівень освітлення, Люкс	Комфортний рівень освітлення, Люкс
2021-04-16 14:02:04	2021-04-16 14:04:04	2021-04-16 14:02:44	16.67	95
2021-04-18 17:21:51	2021-04-18 17:23:51	2021-04-18 17:23:16	17.5	77.5

2021-04-22 23:00:39	2021-04-22 23:02:39	2021-04-22 23:01:58	10	83.33
2021-04-23 17:04:18	2021-04-23 17:06:18	2021-04-23 17:04:21	5.83	97.5
2021-04-24 15:38:18	2021-04-24 15:40:18	2021-04-24 15:39:25	10	40
2021-05-01 19:15:04	2021-05-01 19:17:04	2021-05-01 19:15:17	7.5	174.1
2021-05-02 18:04:39	2021-05-02 18:06:39	2021-05-02 18:05:05	8.33	203.3
2021-05-03 11:17:57	2021-05-03 11:19:57	2021-05-03 11:18:36	15	17.5
2021-05-03 18:27:59	2021-05-03 18:29:59	2021-05-03 18:29:11	33.33	193.3
2021-05-03 21:36:00	2021-05-03 21:38:00	2021-05-03 21:37:13	9.17	110.8
2021-05-04 18:21:27	2021-05-04 18:23:27	2021-05-04 18:23:27	8.33	124.17

Згідно з формулою 6.1 середнє значення некомфортного рівня освітлення є близько – 12.87 Люкс, а комфортний рівень освітлення - 110.59 Люкс.

В такому випадку, це дозволяє сформулювати одну нескладну стратегію вмикання розумної лампи:

Якщо в певний час в середовищі присутня людина і рівень освітлення в цей момент є менш за визначений рівень освітлення, тоді ініціюється подія про вмикання розумної лампи до комфортного рівня освітлення. Ця стратегія передбачає перерахунок параметрів: некомфортного та комфортного рівнів освітлення після кожного вмикання лампи.

Переваги цієї стратегії:

– ввідносно швидкий та простий підхід для отримання комфортного рівня освітлення;

– достатньо активний в ініціації події вмикання, через наявності тільки двох факторів (присутності людини в середовищі та рівнів освітлення в певний момент);

Недоліки:

– не враховує залежність від часу;

Проблема стратегії очевидна - час може бути важливим фактором в прогнозуванні, навіть коли людина знаходиться на робочому місці. Час природним чином має важливий вплив на рівень освітлення від природних джерел світла (випромінювання від сонця та луни), якщо вони мають вплив на середовище. В залежності від часу може змінюватися графік праці, а від типу праці може змінюватися і комфортний та некомфортний рівні освітлення середовища. Наприклад, робота за комп'ютером потребує менший рівень освітлення, ніж прі роботі з документами чи читані книзі. Відсутність залежності від часу навпаки може призвести до вмиканні штучного освітлення в період часу, коли людина може існувати в середовищі, але не бути активною. Наприклад, у світанок.

Але при всьому тому час не має прямого зв'язку з комфортним рівнем освітленості через те, що:

– природні джерела світла (випромінювання від сонця та луни) також залежать від інших багатьох факторів. Наприклад, погоди або доступності промінів до середовища;

– графік праці може змінюватися з часом;

Час вмикання розумної лампи повинен враховуватися як один із факторів прогнозування, але тільки разом з іншими факторами прогнозування.

В ідеальній ситуації, яка фізично не можлива, якщо система вбачає повний збіг стану середовища в попередньому досвіді з певним станом за всіма параметрами крім самого кліматичного стану (наприклад, час, дата, наявність споживача в системі) вона повідомляє або/та ініціює змін стану пристроїв для забезпечення комфортного рівня освітлення.

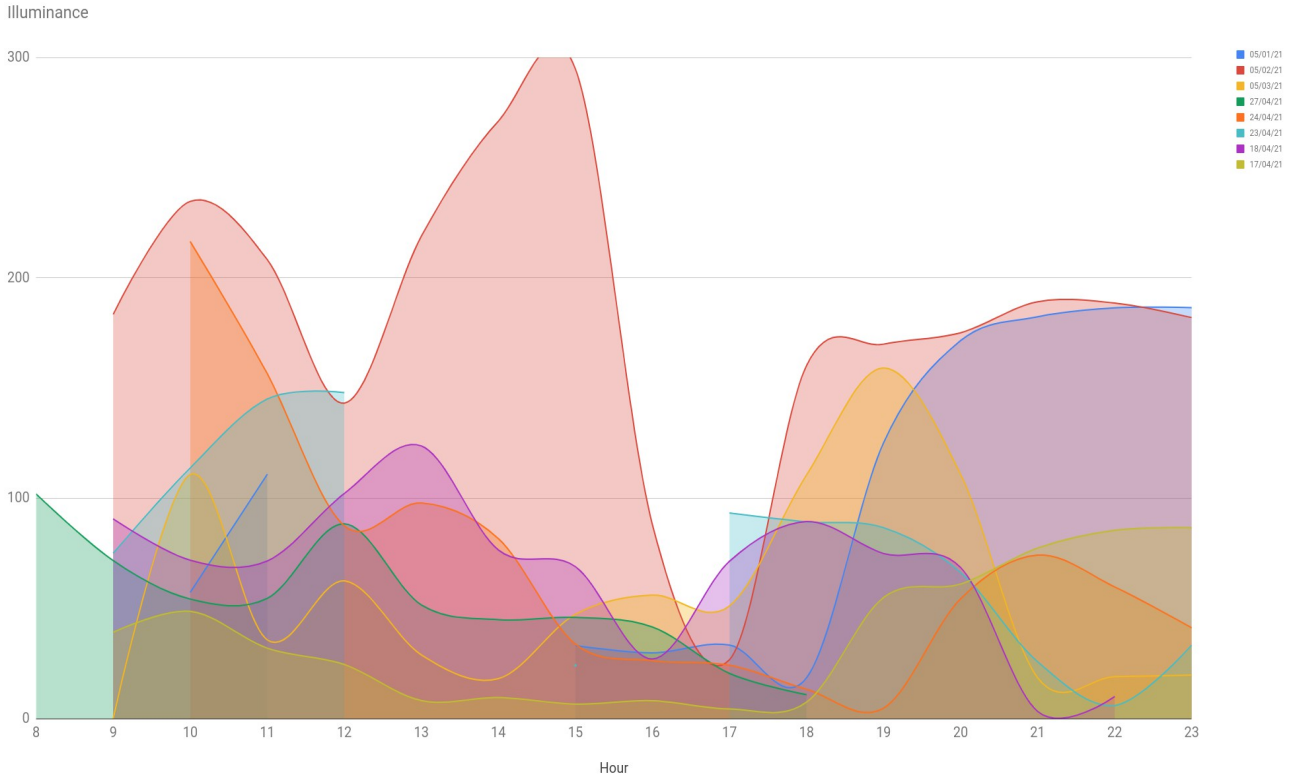

Рисунок 6.1 - Динаміка зміни освітлення за фактором часу

На рисунку 6.1 зображена динаміка зміни освітлення за фактором часу на зібраних даних в рамках дослідження, на графіку є можливість знайти закономірність в тому, що:

– зранку (8-13 годин) рівень освітлення знаходиться на піці;

– в день (13-17 годин) рівень освітлення спадає до некомфортного рівня;

– увечері (17 - 23 годин) рівень освітлення знов підвищується до комфортного рівня;

Причина в високому рівні освітлення до 13 годин та спаданні до некомфортного рівня у проміжок від 13 до 17 годин є цикл сонця. Найчастіше, зранку до полудня сонце знаходиться на піці активності в цей період року. Вже в інтервалі після полудня до вечора активність спадає. Ввечері рівень підвищується до комфортного рівня через вмикання штучного освітлення.

Даний тренд підтверджується графіком вмикання штучного освітлення (див. рисунок 6.2).

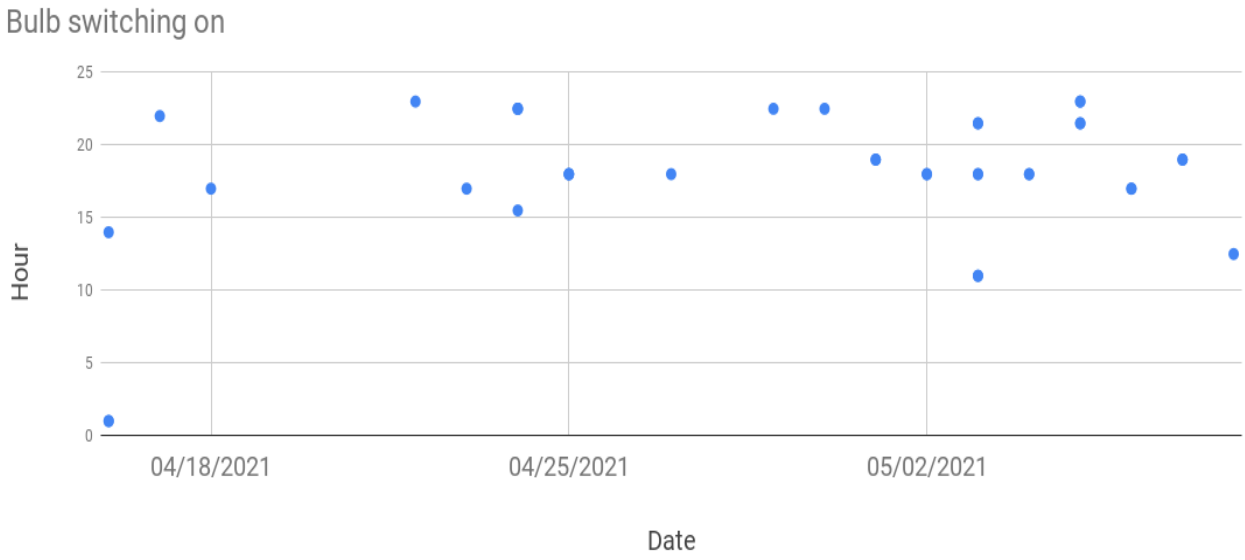

Рисунок 6.2 - Графік вмикання штучного освітлення з 16 квітня до 8 травня

Згідно з цим трендом є можливість використовувати тренд вмикання штучного освітлення разом с стратегією щодо прогнозування некомфортного та комфортного рівнів використання освітлення. У цьому випадку маємо наступний алгоритм щодо прогнозування вмикання освітлення на певний рівень:

1) Вхідні дані;
    1) Комфортний рівень використання освітлення;
    2) Некомфортний рівень використання освітлення;
    3) Функція прогнозування часу;
    4) Поточний рівень освітлення;
    5) Поточний час та дата;
    6) Індикатор наявності людини в середовищі;
2) Умова вмикання штучного освітлення;
    1) Поточний рівень освітлення знаходиться менше ніж некомфортний рівень освітлення;
    2) Час відповідає функції прогнозування;
    3) Індикатор наявності людини в середовищі має значення “істина” (true);

6.2.2 Розробка моделі прогнозування графіку вмикання штучного освітлення за часом

Для прогнозування часу вмикання використаємо підхід лінійної регресії.

Прогнозована величина:

– кількість годин та хвилин в годинах (Наприклад, 13:45 - 13.75 годин);

Набір аргументів (“features”) для прогнозування часу вмикання штучного:

1) Номер дня по рахунку в тижні. Наприклад, 8 травня є субота, 6 день з початку тижня;

2) Номер місяця;

Використовується для прогнозування підвищення або зниження тривалості дня згідно з літнім сонцестоянням. Як відомо, самий довгий день - 21 червня, самий короткий день - 21 грудня. Згідно з цим, треба зробити нормалізацію за наступною логікою: якщо дата припадає на інтервал з 21 грудня до 21 червня використовується місяць вказаної дати, інакше використовується значення різниці між 12 (грудень) та місяцем вказаної дати. Наприклад, у випадку якщо це місяць травень – то результатом нормалізації буде його номер по рахунку в році, тобто 5. У випадку, якщо це листопад (номер по рахунку в році – 11), то результат нормалізації буде 1.

Для прогнозування використаємо пакет Microsoft.ML для роботи з вже розробленими механізмами машинного навчання від компанії Microsoft. Попередній випуск ML.NET включав трансформації для розробки особливостей, таких як створення n-грамів, та особливостей, які навчаються обробляти бінарну класифікацію, багатокласову класифікацію та завдання регресії (лінійна, логістична). Також ML.NET продемонстрував, що він здатний навчати моделям аналізу настроїв, використовуючи великі набори даних, одночасно досягаючи високої точності. Його результати показали 95% точність на набору даних огляду Amazon розміром 9 ГБ[20].

Прогнозування здійснюється за наступними кроками (див. рис. 6.3):

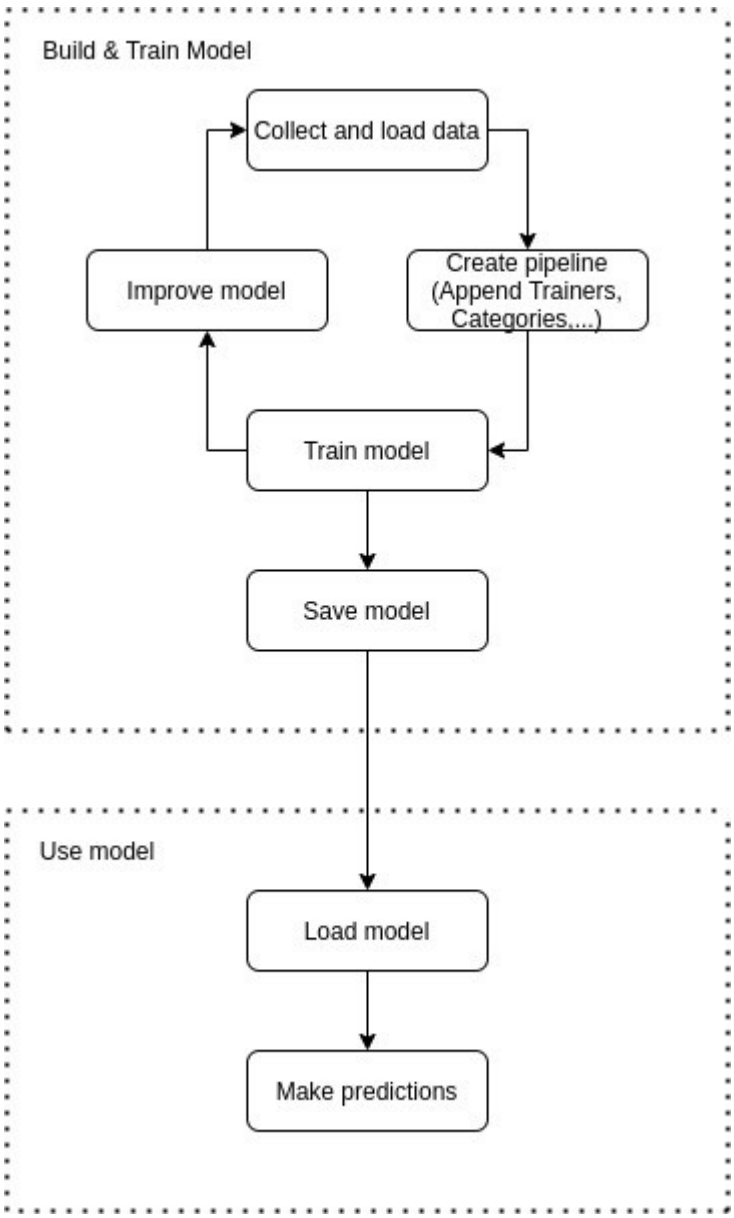

Рисунок 6.3 - Етапи побудови та навчання моделі машинного навчання, прогнозування результату

Етапи побудови та навчання модель машинного навчання:

1) Збір та завантаження даних для навчання;

2) Створення конвеєру (pipeline) для застосування функцій нормалізації параметрів (features), алгоритму машинного навчання;

3) Навчання моделі;

4) Оцінювання моделі та покращення її після оцінки;

5) Збереження моделі в сховище для перевикористання;

6) Загрузка моделі з сховища;

7) Прогнозування результату;

Збір даних описаний в розділі “Рівень периферійних обчислень” здійснюється за допомогою групою сенсорів та відстеженням станів групи актуаторів, дані зберігаються та завантажуються з бази даних.

Вхідна (див. лістинг 6.1) та вихідна моделі (див. лістинг 6.2) для навчання мають наступну структуру.

Лістинг 6.1 - Вхідна модель логу для навчання:

```
public class BulbChangeStateLogInputModel
{
      public float NormalizedMonthAccordingToSoltice { get; set; }
      public float DayInWeek { get; set; }
      [ColumnName("Label")]
      public float Hour { get; set; }
}
```

Атрибут NormalizedMonthAccordingToSoltice - нормалізований місяць згідно з літнім сонцестоянням, DayInWeek - номер дня по рахунку в тижні, Hour - час вмикання штучного освітлення.

Лістинг 6.2 - Вихідна модель логу для навчання

```
public class BulbChangeStateLogOutputModel
{
    [ColumnName("Score")]
    public float Hour { get; set; }
}
```

Результатом прогнозування є ймовірний час вмикання штучного освітлення. Побудова моделі та навчання моделі робиться наступним чином (див.лістинг 6.3):

Лістинг 6.3 - Побудова та навчання моделі

```
private ITransformer BuildAndTrainModel(IDataView splitTrainSet)
{
      var estimator = _mlContext.Transforms
        .Concatenate("Features", nameof(BulbChangeStateLogInputModel.DayInWeek),
nameof(BulbChangeStateLogInputModel.NormalizedMonthAccordingToSoltice))
```

```
        .Append(_mlContext.Regression.Trainers.Sdca(labelColumnName:    "Label",
featureColumnName: "Features"));

      Debug.WriteLine("===============    Create    and    Train    the    Model
===============");

      var model = estimator.Fit(splitTrainSet);

      Debug.WriteLine("=============== End of training ===============");

      var weights = model.LastTransformer.Model.Weights;

      Debug.WriteLine($"Bias - {model.LastTransformer.Model.Bias}");

      Debug.WriteLine($"Weights - {String.Join(",", weights)}");

      return model;

}
```

Будуємо модель навчання наступним чином: за допомогою операції Concatenate визначаємо атрибут Features (список параметрів для прогнозування) з атрибутів вхідної моделі (DayInWeek та NormalizedMonthAccordingToSoltice), визначаємо алгоритм навчання - SdcaRegressionTrainer (Стохастичний подвійний координатний підйом). Метод SDCA поєднує в собі кілька найкращих властивостей, таких як можливість навчання потокового навчання (без поміщення всього набору даних у пам'ять), досягаючи розумного результату за допомогою декількох сканувань цілого набору даних, і не витрачаючи обчислень на нулі в розріджених наборах даних. Виконуємо процес навчання, визначаємо лінійну функцію за допомогою операції Fit. Атрибут Bias - є базовим коефіцієнтом в функції (зміщенням), атрибут Weights - коефіцієнти до кожного параметру функції

У результаті навчання за допомогою даних отриманих за квітень та травень місяці на робочому місці маємо наступні аргументи:

Bias - 5.2557445; Weights - 0.34525156, 2.366395

І наступну функцію:

$$y = 5.255 + x_1 * 0.345 + x_2 * \ 2.366 \qquad (6.2)$$

де x_1 – номер дня по рахунку в тижні, x_2 – нормалізований місяць згідно з літнім сонцестоянням.

Отримана функція (6.2) відображається разом з даними наступним чином:

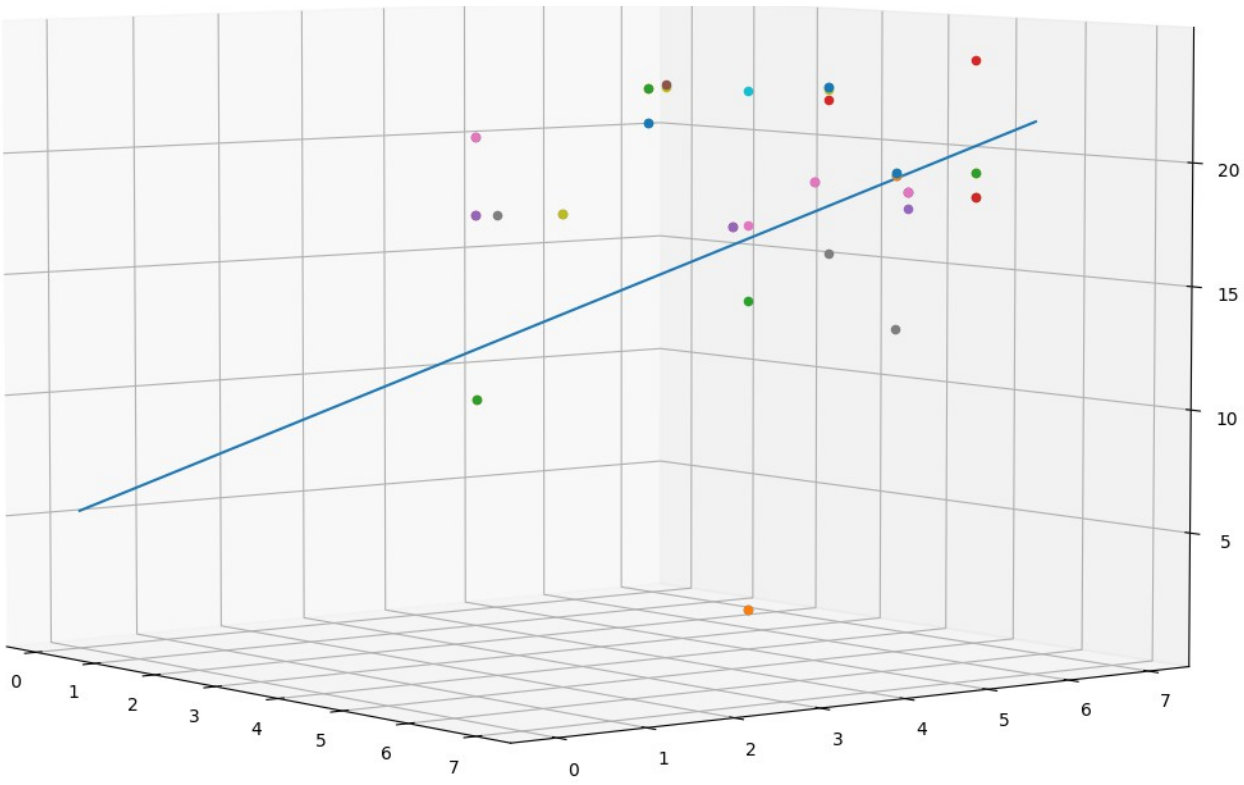

Рисунок 6.4 - Функція прогнозування часу ввімкнення штучного освітлення

За цією функцію є можливість зробити наступний висновок: функція підтверджує тренд підвищення прогнозованого часу ввімкнення штучного освітлення внаслідок зміни підвищення тривалості дня. Також регресія оцінила низьким коефіцієнтом вплив параметра номер дня по рахунку в тижні. Це очікувані висновки. Ця функція більш менш є працездатною на наступні місяці після досліджених місяців (червень, липень), але треба зауважити, що вплив параметра місяця (коефіцієнт - 2.36) занадто великий, очікуване значення від 0.8 до 1. В той же час вплив базового коефіцієнта є недостатнім - 5.25. Наприклад, якщо використовувати цю функцію для прогнозування часу в січні місяці в понеділок, то є можливість отримати час, який приблизно дорівнює 8 годинам.

Виходячи з цього необхідно зробити наступний висновок - сам підхід є коректним, але потребує більше даних, ніж за два місяці, щонайменше один цикл з зимового до літнього сонцестояння - приблизно 6 місяців.

Використання функції прогнозування результату відбувається наступним чином (див. лістинг. 6.4):

Лістинг 6.4 - Використання функції прогнозування

```
      public BulbChangeStateLogOutputModel Predict(BulbChangeStateLogInputModel
input)

      {

            PredictionEngine<BulbChangeStateLogInputModel,BulbChangeStateLogOutp
            utModel>                    predictionFunction                    =
            _mlContext.Model.CreatePredictionEngine<BulbChangeStateLogInputModel
            , BulbChangeStateLogOutputModel>(_transformer);

          return predictionFunction.Predict(input);

      }
```

У якості параметра приймається вхідна модель логу типу BulbChangeStateLogInputModel (див. лістинг 6.1), створюється об’єкт класу PredictEngine, що інкапсулює в собі логіку прогнозування результату за моделлю, яка вже пройшла навчання. За допомогою виклику функції Predict отримується вихідна модель логу типу BulbChangeStateLogOutputModel (див.лістинг 6.2)

7 СТРАТЕГІЇ АДАПТИВНОГО КЕРУВАННЯ КЛІМАТИЧНИМИ РЕСУРСАМИ

7.1 Розробка стратегії вмикання штучного освітлення

Вмикання штучного освітлення - це також реакція на певну подію або ланцюжка події. Як вже було зазначено, штучне освітлення необхідно ввімкнути у випадку коли всі наступні складові мають місто бути:

1) Поточний рівень освітлення знаходиться менше ніж некомфортний рівень освітлення;

2) Поточний час дорівнює або більше ніж прогнозований час від функції прогнозування (див. розділ 6.2.2);

3) Індикатор наявності людини в середовищі має значення “істина” (true);

Аналізуючи ці умови є можливість зробити висновок, що зазначені складові також розкладаються як сукупність подій:

1) Зміна поточного рівня освітлення;

Подія генерування сенсором значення освітлення. Має сенс зберігати значення в буфері та оновлювати при зміні;

2) Некомфортний та комфортних рівнів освітлення;

Від самого початку генерується згідно зі станом колекції логів з бази даних при запуску програми, але необхідно перераховувати значення після ланцюжка з трьох подій: подія генерування значення освітлення - подія ввімкнення штучного освітлення - подія генерування значення освітлення;

3) Прогнозований час ввімкнення;

Також від самого початку генерується згідно зі станом колекції логів з бази даних, і також перераховується після події ввімкнення штучного освітлення;

4) Індикатор наявності людини;

Генерується компонентом відстеження присутності людини в середовищі під час появи чи зникнення людини в середовищі;

5) Подія зміни часу;

Переперевірка умови після проходження певного часового інтервалу (Наприклад, дві хвилини);

Розглянемо алгоритм побудови кожного видавця подій використовуючи існуючи механізми Rx.NET. Наприклад, побудова видавця зміни поточного рівня освітлення матиме натупаний вигляд (див. лістинг. 7.1):

Лістинг 7.1 - Побудова видавця зміни рівня освітлення

```
private IObservable<Double>
GetIlluminanceChangeObservable(DeviceStateObservableContext observableContext)
{
      return Observable.Merge(observableContext.LightSensorObservables)
            .Select(l => l.Value.Value)
            .StartWith(0)
            .Buffer(2)
            .Where(pair => pair[0] != pair[1])
            .Select(pair => pair[1]);
}
```

LightSensorObservables - перемінна, що зберігає колекції подій від усіх сенсорів освітлення. Виконуємо операцію Merge для об'єднання усіх подій в одну подію. Далі трансформуємо потік виносячи з нього значення освітлення в Люксах, та імітуємо нульове значення як стартове значення. Після цього робимо буферизацію потіку на дві події (попередня та поточна), грубо кажучи створюємо пару. В деяких бібліотеках (наприклад, RxJS) існує більш лаконічна альтернатива – pairwase, але не в RxNET. Дали робимо фільтрацію, що попереднє значення відрізняється від минулого, та трансформуємо потік виносячи з пари поточне значення.

У випадку розрахунку не комфортного та комфортного рівнів освітлення задачу треба розподілити на три складові:

– Формування вхідних потоків подій зі наступними складовими;

1) Потік існуючих даних у вигляді колекції пар, де перше значення освітлення до ввімкнення штучного освітлення, друге - значення освітлення після ввімкнення;

2) Потік подій генерування нових даних з сенсора освітлення;

3) Потік подій отримання повідомлень з розумної лампи о включенні;

– Формування ланцюга з вказаних вхідних потоків з новими даними у наступному порядку;

1) Подія генерування значення освітлення (некомфортний рівень освітлення);

2) Подія ввімкнення штучного освітлення та його настроювання;

3) Подія генерування значення освітлення (комфортний рівень освітлення);

– Формування кінцевого потоку подій;

1) Об’єднання потоку існуючих даних з новими, акумулювання даних;

2) Розрахунок комфортного та некомфортного рівнів за формулою 6.1;

Лістинг 7.2 - Формування вхідних потоків даних для розрахунку комфортного та некомфортного рівнів освітлення

```
List<(LightSensorLog beforeActivateBulbLightLog, LightSensorLog afterActivateBulbLightLog)> existingLogs =
    (await _logService.GetBeforeActivateBulbStateLightLogsAsync())
    .ToList();
IObservable<LightSensorLog> lightSensorObservable = Observable.Merge(context.LightSensorObservables);
IObservable<LogBase> switchOnBulbObservable = Observable.Merge(context.BulbStateObservables)
    .Where(l =>
      l.ChangeProperties.TryGetValue("power", out string powerStatus)
      && String.Equals(powerStatus, "on", StringComparison.Ordinal));
```

Функція GetBeforeActivateBulbStateLightLogsAsync повертає існуючі дані з бази даних. Змінна зберігає потік даних з усіх сенсорів освітлення, змінна switchOnBulbObservable є потік повідомлень з розумної лампи о включенні.

Формування ланцюга з вказаних вхідних потоків з новими даними у наступному порядку має наступну реалізацію (див. лістинг. 7.3):

Лістинг 7.3 - Потік подій генерування нових логів освітлення у вигляді колекції пар логів

```
IObservable<(LightSensorLog beforeActivateBulbLightLog, LightSensorLog afterActivateBulbLightLog)[]> beforeAfterLightSensorLogObservable =

    lightSensorObservable

    .Select(

        beforeActivateBulbLightLog => switchOnBulbObservable.

            Select(bulbState => beforeActivateBulbLightLog))

    .Switch()

    .Select(beforeActivateBulbLightLog =>

        lightSensorObservable

            .FirstAsync()

                                    .Select(afterActivateBulbLightLog => new { beforeActivateBulbLightLog, afterActivateBulbLightLog }))

    .Switch()

                .Select(p => new (LightSensorLog, LightSensorLog)[] { (p.beforeActivateBulbLightLog, p.afterActivateBulbLightLog) });
```

Послідовне об'єднання одного потоку подій з іншим потоком подій має певні оператори в деяких бібліотеках - mergeMap, switchMap. В Rx.NET цих операцій нема, але є операція Switch. Цей оператор підписується на видавця (потік), що видає іншого видавця. Кожний раз коли початковий видавець ініціює подію, він відписується від попередньої підписки.

На прикладі з потоком нових даних, оператор Switch застосовується на об’єднанні потоку отримання поточного стану освітлення з повідомленням о включенні розумної лампи. Кінцевий видавець, результат оператору Switch, видаватиме подію, коли відбудуться обидві події в послідовності: спочатку подія отримання поточного стану освітлення, потім подія включення розумної лампи. Коли відбудеться подія включення розумної лампи, видавець відпишеться від попередньої підписки на очікування повідомлення о включенні розумної лампи, і підпишіться знову. Тобто, в випадку коли відбудеться подія включення розумної лампи, ми отримуємо останнє значення освітлення.

Після отримання видавця подій останнього значення освітлення перед включенням розумної лампи, таким же чином об'єднуємо цього видавця з наступною першою подією о значенні освітлення, тобто першим значенням освітлення після включення розумної лампи.

Далі трансформуємо видавця для того щоби повернути пару - де перше значення освітлення до включення штучного освітлення, друге - значення освітлення після включення.

Формування результуючого потоку даних має наступну реалізацію (див. лістинг. 7.4):

Лістинг 7.4 - Потік подій генерування комфортного та некомфортного рівнів освітлення

```
return beforeAfterLightSensorLogObservable

                                        .StartWith(Array.Empty<(LightSensorLog
beforeActivateBulbLightLog, LightSensorLog afterActivateBulbLightLog)>())

          .Do(newLogs => existingLogs.AddRange(newLogs))

          .Do(newLogs => _logger.LogInformation($"Comfortable level calulation.
Count of logs - {existingLogs.Count})"))

          .Select(newLogs => (

                                        min:     existingLogs.Average(l     =>
l.beforeActivateBulbLightLog.Value.Value),

                                        max:     existingLogs.Average(l     =>
l.afterActivateBulbLightLog.Value.Value)));
```

Використання оператора StartWith дозволяє почати генерацію комфортного та не комфортного рівнів з існуючих даних до того, як відбудеться перерахунок через включення розумної лампи. При використанні першого оператора Do виконуємо акумуляцію логів в колекцію existingLogs, за допомогою другого оператора Do логуємо поточну кількість об’єктів в колекції для діагностики, в операції Select трансформуємо потік з розрахунком комфортного та не комфортного рівнів освітлення.

Для прогнозування часу включення розумної лампи також необхідно мати певний список вхідних аргументів, які також змінюються від певних подій (наприклад, зміна поточної години, місяця, доби). Метод для отримання видавця, який сповіщає о прогнозованому часі включення має наступний вигляд (див. лістинг. 7.5):

Лістинг 7.5 - Створення видавця для оповіщення прогнозованого часу

```
public IObservable<float> CreateAsync(DeviceStateObservableContext context)
{
    return Observable.Interval(TimeSpan.FromDays(1))
        .Select(l => _dateTimeProvider.Now)
        .StartWith(_dateTimeProvider.Now)
                                                    .Select(date       =>
_predictService.Predict(_bulbChangeStateLogInputService.Create(date)).Hour);
}
```

Оператор Interval генерує послідовність цілих чисел. Кожне число генерується після закінчення періоду часу, передбаченого для параметра періоду[14]. Після кожної доби функція прогнозування перераховує час включення розумної лампи.

Перемінна _predictService вказує на сервіс прогнозування, який зберігає модель, яка повинна пройти навчання та аналіз перед застосуванням. При запуску сервісу навчання буде здійснюватися за допомогою даних з бази даних.

Але крім цього система повинна бути адаптивною, тобто функція прогнозування часу включення розумної лампи має автоматично переховуватися. По поточній моделі функція базується тільки на даних сповіщень о включенні розумної лампи.

В такому випадку метод для створення видавця, який сповіщає о визначенні або зміні вхідних даних до функції прогнозуванні, має наступний вигляд (див. лістинг. 7.6):

Лістинг 7.6 - Метод для створення видавця, який сповіщає о визначенні або зміні вхідних даних до функції прогнозуванні

```
public async Task<IObservable<BulbChangeStateLogInputModel[]>> CreateAsync(DeviceStateObservableContext context)
{
    List<BulbChangeStateLogInputModel> existingInputModels = (await _logService.GetPowerOnBulbLogs())
        .Select(l => _bulbChangeStateLogInputService.Create(l.LogDate))
        .ToList();

    IObservable<BulbChangeStateLogInputModel> switchOnBulbObservable = Observable.Merge(context.BulbStateObservables)
        .Where(l =>
            l.ChangeProperties.TryGetValue("power", out string powerStatus)
            && String.Equals(powerStatus, "on", StringComparison.Ordinal))
        .Select(l => _bulbChangeStateLogInputService.Create(l.LogDate));

    return switchOnBulbObservable
        .Select(log => new BulbChangeStateLogInputModel[] { log })
        .StartWith(Array.Empty<BulbChangeStateLogInputModel>())
        .Do(newInputLogs => existingInputModels.AddRange(newInputLogs))
        .Select(newInputLogs => existingInputModels.ToArray());
}
```

Аналогічним чином до потоку подій генерування комфортного та некомфортного рівнів освітлення, функція в першу чергу ініціалізує колекцію

використовуючи існуючи логи з бази даних, а потім оновлює колекцію даними, які приходять з оповіщеннями о включенні від розумної лампи.

Використовуючи цього видавця зробимо підписку на проведення тренування та оцінювання моделі прогнозування(див. лістинг. 7.7).

Лістинг 7.7 - Виклик тренування та оцінювання моделі прогнозування

```
        private async Task<IDisposable>
SubscribeToRetrainBulbSwitchingModel(DeviceStateObservableContext context)
        {
            return (await
_retrainPredictionModelObservableProvider.CreateAsync(context))
                .Subscribe(inputs => TrainBulbSwitchingModel(inputs));
        }

        private void TrainBulbSwitchingModel(BulbChangeStateLogInputModel[]
inputs)
        {
            _predictService.BuildAndTrainModel(inputs);
            _predictService.Evaluate();
            _logger.LogInformation("Model has been trained and evaluated");
            _communicator.BulbSwitchingOnModelTrained.OnNext(true);
        }
```

Базуючись на визначених видавців подій можемо визначити видавця, що ініціюватиме включення розумної лампи. Для цього необхідно в першу чергу згрупувати видавців в один так чином, щоби після того як будь-який з видавців випускає подію, після того як кожен із видавців випромінює принаймні одну подію. Для цього підійде оператор CombineLatest (групуючий оператор). Після випромінювання події необхідно перевірити необхідність включення розумної лампи. Повна послідовність операторів має наступний вигляд (див. рис 7.1), де вхідні видавці:

– PersonInSystem - видавець подій появи чи покидання споживача в системі;

– RecheckState - видавець подій для повторної перевірки умови після певного часового інтервалу;

– CurrentLevel – це видавець події отримання поточного рівня освітлення;

– ComfortableLevels – це видавець події ініціалізації та зміни некомфортного та комфортного рівнів освітлення;

– BulbHour – це видавець події ініціалізації та зміни прогнозованого часу вмикання штучного освітлення;

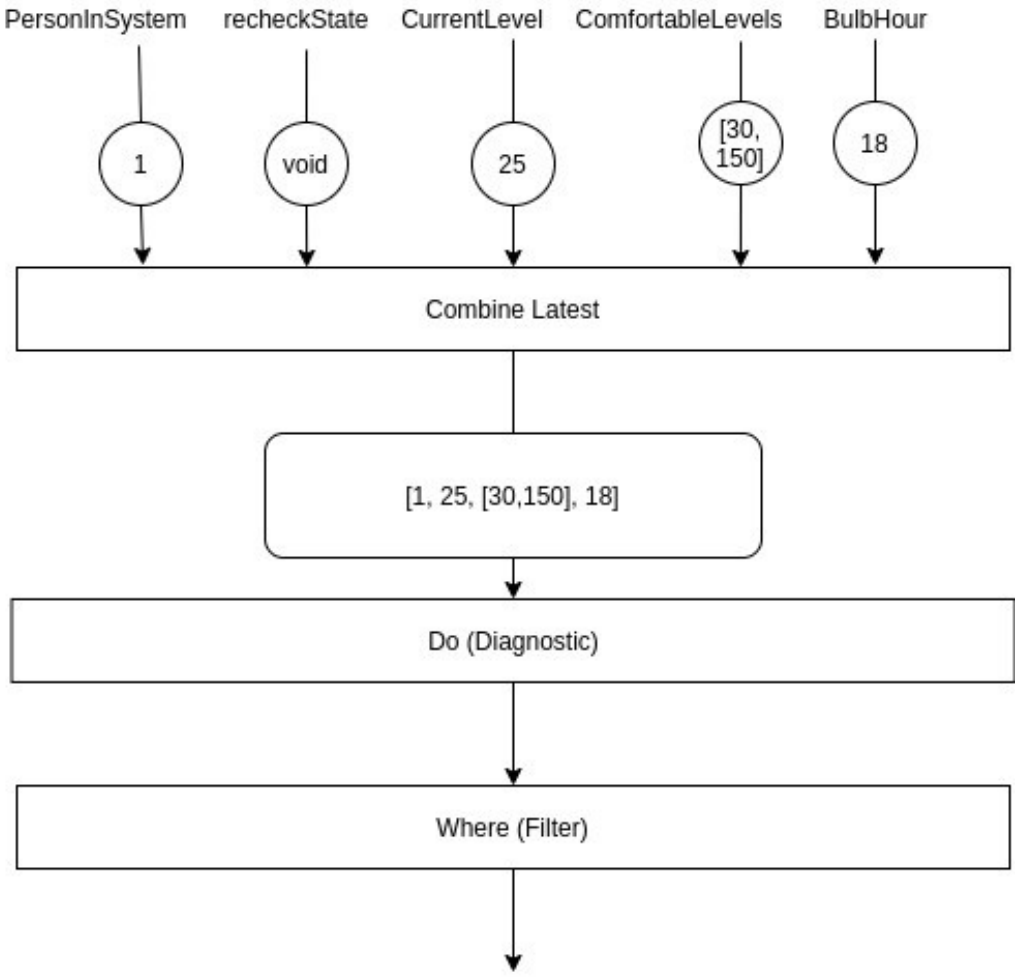

Рисунок 7.1 - Формування потоку подій включення розумної лампи

Після групування видавців в одного, необхідно залогувати стан генерованих значень з діагностичною метою відстеження контексту прийняття рішення щодо включення розумної лампи. Цю задачу виконує оператор Do. Після цього виконується фільтрація, за допомогою оператору Where - функція

приймає контекст з згенерованими значеннями від видавців, та повертає булівський флаг - чи існує необхідність включити лампу (див. лістинг 7.8).

Лістинг 7.8 - Формування та підписка на потік подій включення розумної лампи

```
      IObservable<CheckConditionToActivateSmartBulbContext>
checkCondtionToActivateSmartBulbObservable = Observable.CombineLatest(

          personInSystemObservable,

          recheckStateObservable,

          currentIlluminanceLevelObservable,

          minComforableIlluminanceLevelObservable,

          predictedBulbSwitchOnHourObservable,

                              (personInSystem,      _,      currentIlluminanceLevel,
minComfortableIlluminanceLevel, bulbSwitchOnHour)

                  => new CheckConditionToActivateSmartBulbContext(personInSystem,
bulbSwitchOnHour, currentIlluminanceLevel, minComfortableIlluminanceLevel)

      );


      ActivateSmartBulbObserver          smartBulbObserver          =          new
ActivateSmartBulbObserver(deviceEnvironment.SmartBulbs.First(),
_bulbDeviceCommandFactory);


      return checkCondtionToActivateSmartBulbObservable

           .Do(p => _logger.LogInformation($"Check condition to active the smart
bulb{Environment.NewLine}{p.ToString()}"))

           .Where(CheckConditionToActivateSmartBulb)

                                                        .Subscribe(p            =>
smartBulbObserver.OnNext(p.ComfortableIlluminanceLevels.max));
```

CheckConditionToActivateSmartBulbContext перемінна, що зберігає посилання на всі генеровані значення необхідних для визначення події та включення розумної лампи на комфортний рівень. Фільтрація подій застосовується за допомогою наступного предикату (див. лістинг. 7.9):

Лістинг 7.9 - Предикат фільтрації подій для включення розумної лампи

```
return context.PersonInEnvironment

    && context.CurrentlluminanceLevel <=
context.ComfortableIlluminanceLevels.min
```

```
    && context.BulbSwitchingOnHour <= _dateTimeProvider.Now.Hour +
_dateTimeProvider.Now.Minute / 60;
```

Логістичне висловлювання містить три складові, що об’єднується через логічний оператор “І”:

1) Наявності людини в системі;

2) Поточний рівень освітлення знаходиться менше ніж некомфортний рівень освітлення;

3) Поточний час дорівнює або більше ніж прогнозований час від функції прогнозування;

Після фільтрації подій робиться підписка на активацію розумної лампи з переданням комфортного рівня освітлення. ActivateSmartBulbObserver перемінна зберігає посилання на спостерігача події активації розумної лампи. Безпосередньо на рівні спостерігача вже може здійснюватися калібрування яскравості штучного освітлення до наявного комфортного рівня освітлення.

ВИСНОВКИ

Головним завданням атестаційної роботи було розробити систему та модель для адаптивного та автоматизованого керування кліматичними ресурсами в домашніх та офісних системах.

Були досліджені наступні розділи:

– Актуальність проблеми та необхідність в адаптивному керуванні кліматичними ресурсами;

– Фізичні властивості кліматичних ресурсів;

– Архітектура системи керування кліматичними ресурсами в домашніх та офісних системах;

– Вимоги та принципи вибору сенсорів та актуаторів для використання в системі;

– Загальні принципи роботи та використання сенсорів та актуаторів в системі;

– Парадигми, патерни та механізми для реалізації збереження логів о стані середовища в системі;

– Алгоритми та стратегії для прогнозування параметрів використання кліматичних ресурсів в системі;

– Парадигми, патерни та механізми для реалізації адаптивної активації актуаторів в системі;

Було спроектовано та розроблено:

– Сервіс для зчитування та запису логів о стані середовища;

– Модель прогнозування параметрів користування кліматичними ресурсами;

– Сервіс для адаптивного та автоматизованого керування кліматичними ресурсами;

Список використаних джерел

1. Manufacturing Trends Report. [Електронний ресурс] / Microsoft. 2019 – Режим доступу : https://info.microsoft.com/rs/157-GQE-382/images/EN-US-CNTNT-Report-2019-Manufacturing-Trends.pdf
2. Paul Tippens. Honor Physics - Light and Illumination. / Paul Tippens. // Lecture 33. Southern Polytechnic State University, 2007.
3. Yeelight Smart LED Filament Bulb. Data Sheet. [Електронний ресурс] / Yeelight Germany GmbH. 2020 – Режим доступу : https://www.yeelight.de/pub/media/downloads/yeelight_smart_led_filament_lampe_datenblatt_EN.pdf
4. Image credit: J.C. Walker, Light Sources - Technology and Applications / J.C.Walker // Attribution-ShareAlike 3.0
5. Громадський Ю.С. Природне і штучне освітлення // Державні будівельні норми України. Київ. 2006. Додаток К. Таблиця К.І - Нормовані показники освітлення основних приміщень громадських, житлових, допоміжних будинків С. 68.
6. GomesLighting in the Workplace: Recommended Illuminance (lux) at Workplace Environs. / Sandra Preto, Cristina Caramelo // Faculty of Architecture, University of Lisbon, Lisbon, Portugal
7. Human Centric Lighting. / Tralau, B.: // 2014 - Режим доступу : http://www.ibe-biv.be/media/pdf/Studiedag_2014/06_ Tralau.pdf
8. IoT архитектура . [Електронний ресурс]. / Дов Нимцар. // Habrahabr. 2019. - Режим доступу: https://habr.com/ru/post/455377/
9. Digital 16bit Serial Output Type Ambient Light Sensor IC. BH1750FVI . [Електронний ресурс] / ROHM Co., Ltd, 2010 – Режим доступу: https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf
10. Паттерны объектно-ориентированного проектирования. Паттерны проектирования. / Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес // СПб.: Питер, 2020. - 320с

11. Сергей Тепляков. Паттерны проектирования на платформе .NET / Сергей Тепляков //СПб.: Питер, 2019.

12. Tamir Dresher. Rx.NET in Action / Tamir Dresher // Manning Publications Co, 2017.

13. BME280. Combined humidity and pressure sensor. Data Sheet [Електронний ресурс] / Bosch Sensortec GmbH, 2020 – Режим доступу : https://www.mouser.com/datasheet/2/783/BST-BME280-DS002-1509607.pdf

14. Observable.Interval [Електронний ресурс] / MSDN – Режим доступу: https://docs.microsoft.com/en-us/previous-versions/dotnet/reactive-extensions/hh228911(v=vs.103)

15. Light Sensors: Units, Uses, and How They Work [Електронний ресурс] – Режим доступу : https://blog.endaq.com/how-light-sensors-work

16. Фізика зволоження [Електронний ресурс] – Режим доступу : https://venta.com.ua/ukr/physics.html

17. Humidity Sensor – Types and Working Principle [Електронний ресурс] – Режим доступу : https://www.electronicshub.org/humidity-sensor-types-working-principle/

18. Over 100 million IoT attacks [Електронний ресурс] – Режим доступу : https://www.infosecurity-magazine.com/news/over-100-million-iot-attacks/

19. The cost of making IoT more secure will reach $3.1bn by 2021 [Електронний ресурс] – Режим доступу : https://www.telecomtv.com/content/iot/the-cost-of-making-iot-more-secure-will-reach-3-1bn-by-2021-16528/

20. Machine Learning at Microsoft with ML.NET [Електронний ресурс] / Zeeshan Ahmed // ACM – Режим доступу: https://dl.acm.org/doi/10.1145/3292500.3330667