Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

Створення PaaS системи управління розумним будинком

Текстова частина до курсової роботи
за спеціальністю „Інженерія Програмного Забезпечення”

Керівник курсової роботи
ст. викладач, кандидат технічних наук,
Шабінська М.О.

_____
(підпис)
“____” _____ 2020 р.

Виконав студент
Каруна Д. Г.
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики,
кандидат фізико-математичних наук, доцент
_____ С. С. Гороховський
(підпис)
„10” жовтня 2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Каруні Даниїлу Геннадійовичу факультету інформатики 1 курсу

ТЕМА Створення PaaS системи управління розумним будинком

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Вступ
3. Огляд концепції розумного будинку, існуючих платформ та архітектурних рішень
4. Створення власної платформи
5. Тенденції розвитку IoT та смарт-будинків
6. Висновки
7. Глосарій
8. Список використаної літератури

Дата видачі „10” жовтня 2019 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Тема: Створення PaaS системи управління розумним будинком

Календарний план виконання роботи:

№	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1	Отримання завдання на курсову роботу.	10.10.2019	
2	Огляд існуючих платформ, сервісів та інфраструктур для управління розумним будинком.	25.11.2019	
3	Огляд можливих архітектур та моделей проектування платформи та SDK для пристроїв та сервісів.	02.12.2020	
4	Написання першого розділу.	25.01.2020	
5	Розробка proof of concept.	18.04.2020	
6	Написання останніх двох розділів.	30.04.2020	
7	Створення презентації для доповіді.	05.05.2020	
8	Корегування роботи.	10.05.2020	

Студент: Каруна Д. Г.

Керівник: Шабінська М.О.

“10” жовтня 2019 року

Зміст

Анотація

В цій роботі розглядається сьогоднішній стан IoT платформ та платформ для розумних будинків, таких як Apple HomeKit, Google Home та інші. Розглядається оптимальна архітектура на прикладі створення власного сервісу та прикладів API.

Вступ

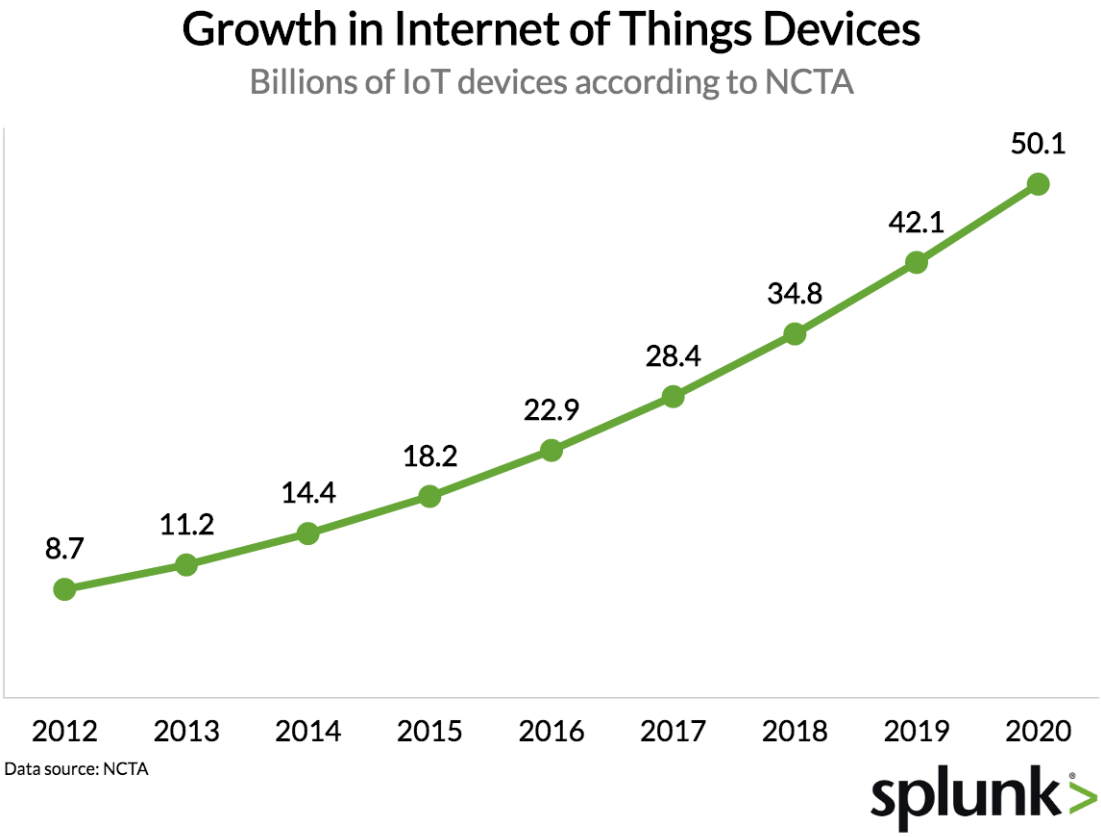
За останні 8 років кількість пристроїв інтернету речей (IoT) зросло майже у шість разів. Основні виробники персональних та побутових пристроїв розробили або розробляють власні екосистеми розумних будинків.

Робота складається з трьох розділів.

В першому розділі розглядаються існуючі сервіси та платформи для розумних будинків. Також розглядаються архітектурні рішення та бізнес моделі для створення власної платформи для розумного будинку.

В другому розділі розглядається процес створення власної платформи, компонентна структура та проблеми, які виникали під час розробки.

В третьому розділі розглядаються майбутні тенденції, напрямок розвитку IoT та платформ для розумних будинків.



1. ОГЛЯД КОНЦЕПЦІЇ РОЗУМНОГО БУДИНКУ, ІСНУЮЧИХ ПЛАТФОРМ ТА АРХІТЕКТУРНИХ РІШЕНЬ

1.1. Історія автоматизації будинків та Інтернету Речей

Створення приладів для автоматизації процесів в будинках має давню історію. Ще на початку XX століття люди почали створювати пилососи, холодильники, праски тощо. Але до першого «смарт» пристрою залишалося ще 50 років.

В 1966 році був створений перший розумний пристрій під назвою ECHO IV. Це був перший пристрій, що міг обраховувати список того, що потрібно купити, контролювати температуру дома та вмикати чи вимикати прилади.

Рисунок 1.1 – ECHO IV

Але такі прилади були непопулярними та майже не продавались. Більша частина таких пристроїв залишилась у стані прототипів.

Концепт інтернету речей, що є фундаментом для сьогоднішньої автоматизації будинків, з’явився у 1982 році. В університеті Карнегі-Мелон був модифікований автомат з продажу Кока-Коли, що став першим побутовим приладом, що був підключений до інтернету. Він міг видавати інформацію про наявність напоїв та про їх температуру.

В 90-ті роки було написано багато статей про повсюдний комп’ютинг. В 1994, Реза Раджи описав концепт як «передача маленьких пакетів даних між великою кількістю вузлів, щоб інтегрувати та автоматизувати все від побутових приладів до цілих заводів».

Сам термін «Інтернет речей» був створений Кевіном Ештоном в Procter & Gamble для опису системи, де можна було об’єднувати фізичні об’єкти з давачами і мережею Інтернет.

Для об’єднання пристроїв інтернету речей в систему автоматизації будинку створюються платформи та хаби.

1.2. Сучасні сервіси.

1.2.1. Сервіси, що орієнтовані на користувачів.

Для комфортного досвіду користувача створюються спеціальні платформи, хаби та застосунки, що збирають пристрої інтернету речей в єдине місце та дозволяють контролювати їх, створювати певні сценарії, програмувати реакції на події.

Наприклад, Apple HomeKit надає спеціальний протокол для виробників IoT пристроїв. Після запровадження його підтримки у пристрій, цей пристрій можна контролювати через ноутбук, телефон, годинник або смарт колонку Apple за допомогою вже встановленого застосунку “Home”, що є на кожному з приладів Apple.

Частіш за все, системи розумного будинку потребують централізований вузол, що буде облаштований спеціальним програмним забезпеченням та буде постійно доступним. В ролі такого вузла може виступати будь-який пристрій,

але частіш за все виробники створюють окремі пристрої, що часто поєднані з аудіосистемою та голосовим асистентом (наприклад, Apple HomePod, Google Home, Amazon Echo тощо).

Поруч з комерційними варіантами цих систем також існують некомерційні з відкритим вихідним кодом, наприклад OpenHAB чи Domoticz. Ці системи дозволяють розмістити сервер на будь-якому пристрої, як на домашньому комп'ютері, так і на пристроях, що складаються з одної плати, за типом Raspberry Pi чи PINE A64.

Кожен сучасний сервіс дозволяє користувачам створювати певні сценарії для комплексного запуску послідовності команд у групі пристроїв, а також можливість їх автоматизувати. Автоматизація сценаріїв дозволяє їм виконуватися при певних умовах (тригерах), наприклад: певний час, таймер, показник певного давача, сигнал з іншого смарт-пристрою тощо.

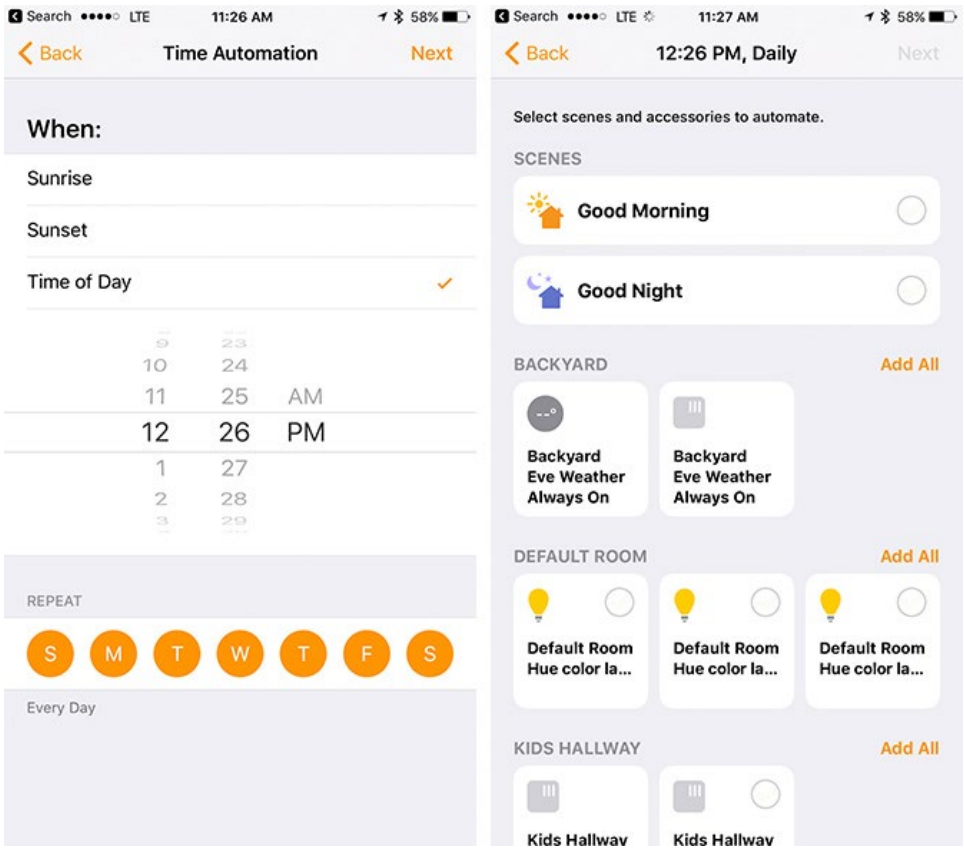


Рисунок 1.2. Створення автоматизації у додатку Apple Home. (MakeUseOf.com)

1.2.2. Сервіси, що орієнтовані на розробників.

Для створення додатків та сервісів є доцільним використання вже створених платформ, адже керування власними силами інфраструктурою, серверами та програмним забезпеченням низького рівню часто потребує багато ресурсів.

На сьогодні існує багато платформ для створення IoT сервісів та додатків. З основних можна назвати: Google Cloud IoT, AWS IoT, Microsoft Azure IoT Suite, IBM Watson IoT. Кожна платформа надає схожий набір послуг, основні з яких описані у підрозділі 1.4.

Всі платформи беруть на себе відповідальність за підтримку серверів, віртуалізацію, підтримку операційної системи та стеку програмного забезпечення, безпеку даних тощо. Користувачам частіш за все надається менеджмент таких сутностей високого рівня, як: реєстри пристроїв, набори правил, права доступу та інші. Менеджмент є доступним через веб-інтерфейс та через API, оскільки більшість розробників використовують саме API для створення програмного забезпечення на таких платформах.

Платформи великих провайдерів веб-сервісів, таких як Google Cloud чи Amazon Web Services є тісно пов’язаними з іншими сервісами цих провайдерів. Наприклад, AWS IoT дає можливість направляти потоки даних з розумних пристроїв за допомогою правил у інші сервіси, такі як Lambda, DynamoDB, SNS, CloudWatch та інші. Google Cloud має аналогічний набір сервісів, що взаємодіють з їхньою платформою розумних пристроїв.

Наявність такої взаємодії суттєво впливає на цінність таких IoT платформ для розробників, адже велика кількість компаній використовує інші сервіси цих провайдерів для своїх потреб, наприклад для розміщення веб-сайту, зберігання даних, запису журналів користування та потоків даних.

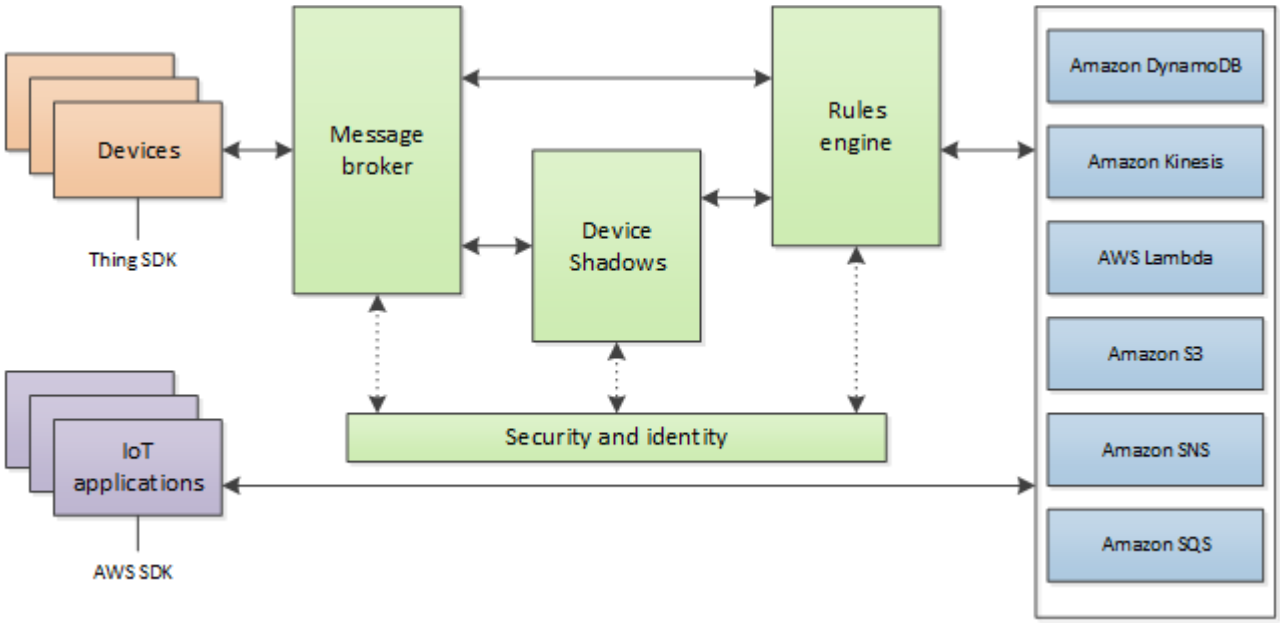


Рисунок 1.3. Структура комунікації між сервісами Amazon Web Services.

Недоліком таких платформ є відсутність концентрації на деталях, що присутні інтернету речей, а також прив'язка до екосистеми цих провайдерів. Такі платформи, як ThingsBoard, вирішують це питання. ThingsBoard є відкритим програмним забезпеченням, що надає можливість модифікувати його, розміщувати на своїх серверах та використовувати без обмежень. В той же час, ThingsBoard надає більш специфічну аналітику та візуалізацію даних, більш гнучку та дружелюбну до користувача систему правил, велику кількість сутностей, як користувачі, клієнти, майно та інші.

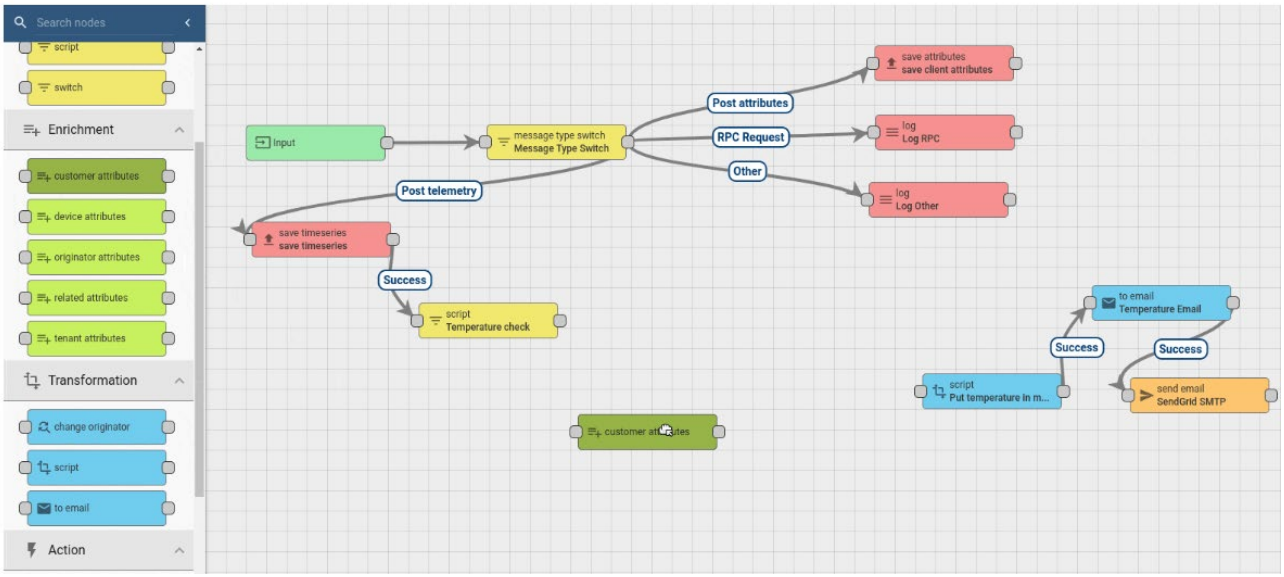


Рисунок 1.4. Візуальний редактор правил ThingsBoard.

1.3. Організація мереж розумних пристроїв.

Для реалізації комунікації між розумними пристроями використовують як існуючі протоколи, так і спеціально розроблені з врахуванням особливостей (низьке споживання енергії, невеликі об’єми даних, можливість сітчастої топології). На ринку розумних пристроїв та розумних будинків поширено чотири основних протоколи: Z-Wave, Zigbee, Wi-Fi та BLE (Bluetooth Low Energy).

1.3.1. Z-Wave.

Z-Wave – бездротовий власницький протокол, що був розроблений спеціально для автоматизації розумних будинків. Він використовує топологію сітчастої мережі для комунікації між пристроями за допомогою радіохвиль з низькою енергією. Для керування пристроями потрібен контролер/шлюз, що підтримує Z-Wave. За необхідності контролю через комп’ютер, мобільний телефон, через Інтернет тощо, шлюз можна підключити до звичайного інтернет маршрутизатору та забезпечити цим доступ ззовні.

Цей протокол був розроблений для надійної передачі невеликих пакетів даних с низькою затримкою. На сучасних чіпах швидкість передачі даних сягає 40 кілобіт на секунду, а відстань між двома вузлами мережі може сягати в середньому 30-40 метрів (до 90 метрів на відкритому просторі). Сітчаста мережа дозволяє пристроям використовувати інші пристрої для отримання інформації (до чотирьох посередників), що дозволяє більш вільно розташовувати їх по будинку. Оптимізація споживання енергії дозволяє певним пристроям роками працювати від акумулятору. Протокол дозволяє максимум 232 пристроїв в одній мережі. Z-Wave працює на частоті 800-900 МГц, що допомагає уникнути проблем з передачею даних, так як ця частота не конфліктує з більшістю побутових частот для передачі даних (таких, як 2.4 ГГц та 5 ГГц).

Патенти та закритість Z-Wave ускладнює дослідження безпечності цього протоколу, а також його імплементацію іншими виробниками. Було знайдено

декілька вразливостей, але жодна з них була проблемою не специфікації протоколу, а його імплементації виробником. У 2016 році Z-Wave Alliance оновили стандарти безпеки, впровадили нові процедури підключення пристроїв до мережі з унікальними PIN-кодами чи QR-кодами, а також опублікували специфікації протоколу для підвищення доступності протоколу для розробників. За допомогою уніфікованих стандартів Z-Wave забезпечує кращу сумісність між різними пристроями та різними версіями протоколу, аніж Zigbee.

1.3.2. Zigbee.

Zigbee – специфікація для низки протоколів високого рівня, що базується на стандарті IEEE 802.15.4 (стандарт, що визначає роботу безпровідних мереж з низьким трафіком). Він, як і Z-Wave, був розроблений як більш проста та дешева альтернатива Bluetooth чи Wi-Fi та використовується для пристроїв, яким потрібно передавати невелику кількість даних на невелику відстань. Мережевий шар підтримує такі топології, як сітчаста мережа, зірка, дерево. ZigBee пристрої можуть самостійно знаходити один одного, будувати мережу, вони також намагаються встановлювати нові шляхи в разі виникнення проблем з вузлами мережі.

Мережа ZigBee складається з трьох типів пристроїв:

- Координатор – формує мережу, зберігає паролі, може підключатися до інших мереж. У мережі має обов’язково бути хоча б один координатор.
- Маршрутизатор – може виконувати функцію додатку, а також працювати як посередник для передачі даних іншим пристроям.
- Кінцевий пристрій – має тільки можливість передавати дані координатору або маршрутизатору, не працює як посередник.

Кожен розумний пристрій може виступати у будь-якій з цих ролей, а також їх комбінувати. Це дозволяє ізольованим пристроям, що працюють від акумулятора виступати в ролі тільки кінцевих пристроїв та зберігати енергію роками, а такі пристрої, як смарт-лампочка, можуть паралельно виступати

координатором та маршрутизатором за рахунок доступу до постійного джерела електричного струму.

ZigBee має більш високу швидкість передачі даних, ніж Z-Wave. Вона може варіюватися від 20 кілобіт/с (для 868 МГц) до 250 кілобіт/с (для 2.4 ГГц). Максимальна відстань між пристроями може бути від 10 до 20 метрів в залежності від умов, що є нижчим показником за Z-Wave. Оскільки більшість домашніх пристроїв ZigBee працює на частоті 2.4ГГц, це може створювати проблеми зі скупченням сигналу, тому що ця частота може конфліктувати з частотою Wi-Fi.

1.3.3. Wi-Fi.

Wi-Fi – популярний стандарт для передачі даних по радіоканалах. Незважаючи на те, що він не розроблений спеціально для IoT, він є досить поширеним (див. рисунок 1.2).

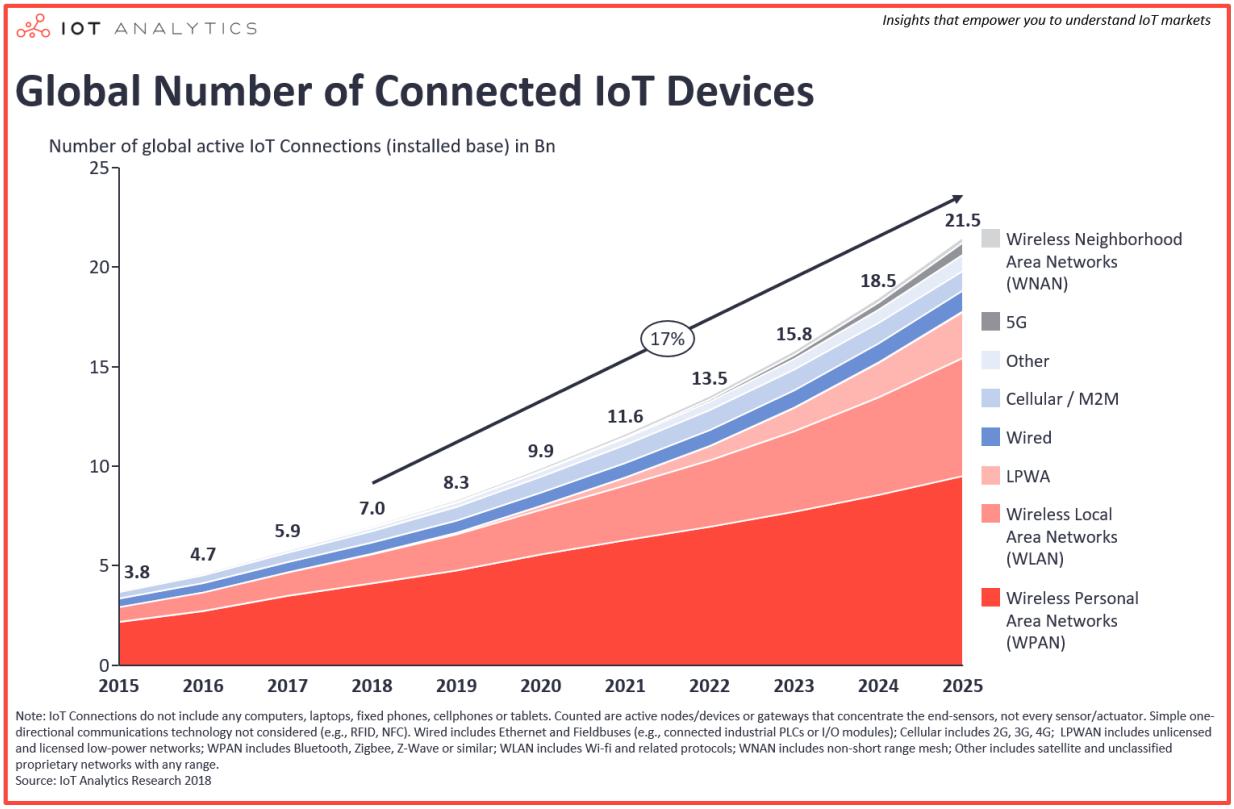


Рисунок 1.5. Кількість IoT пристроїв за протоколами. (IoT Analytics)

Легкість підключення та встановлення пристроїв, а також відсутність необхідності придбання та налаштування контролерів є одною з переваг цього

стандарту. Також, за допомогою Wi-Fi можна швидко передавати великі об’єми даних, що робить його єдиним поширеним стандартом для камер відеоспостереження та супутніх пристроїв, що є великою частиною ринку розумних пристроїв.

В той же час, Wi-Fi має певні суттєві недоліки, що можуть спонукати до мінімізації його використання:

- Можлива перевантаженість спектру за рахунок великої кількості підключених пристроїв, що може заважати користуванню персональних пристроїв (комп’ютерів, телефонів, тощо). Також є можливість перешкод сигналу для протоколу ZigBee.
- Відсутність підтримки сітчастої топології та необхідність додавати маршрутизатори при недостатній силі сигналу.
- Велике споживання енергії, що вимагає від пристроїв підключення до джерела струму.
- Проблеми безпеки. На відміну від Z-Wave та Zigbee пристрої підключені напряму до маршрутизатору, що дає їм вихід в Інтернет, а також потенційний доступ зловмисників з Інтернету до них. Це може призводити до викрадення даних, зловживання, підключення пристроїв до мережі ботів.
- Відсутність універсального стандарту на рівні протоколу, що часто призводить до створення окремих додатків для пристроїв різних виробників.

Незважаючи на це, Wi-Fi є основним протоколом підключення смарт-хабів до маршрутизаторів. Цей протокол залишається досить популярним серед користувачів, оскільки багато виробників сучасної техніки вбудовують саме Wi-Fi у свої пристрої та не розглядають підтримку таких спеціалізованих протоколів, як Z-Wave та ZigBee.

1.4. Архітектура сучасних IoT PaaS.

Усі сучасні IoT платформи надають схожий набір компонентів. З основних можна виділити: шлюз (gateway), брокер повідомлень, сервіси авторизації та аутентифікації, реєстр пристроїв, тіньові пристрої, система створення правил. Розглянемо кожен з них окремо.

1.4.1. Gateway.

Шлюз – компонент системи, що відкритий до доступу зовні, який слугує вхідною точкою для пристроїв при підключенні до платформи. Шлюз керує усіма активними підключеннями пристроїв та запроваджує семантику для певних протоколів, щоб пристрої мали змогу безпечно спілкуватися з платформою.

Частіш за все шлюз підтримує такі протоколи, як MQTT, WebSockets та HTTP. В залежності від протоколу, шлюз підтримує різні види підключень. Для MQTT та WebSockets потрібно створювати двосторонні довготривалі підключення, а для HTTP потрібно підтримувати класичну структуру веб-сервера.

У більшості сервісів шлюз надає можливість створення правил, що допускають певні протоколи, порти, адреси, домени тощо. Це дозволяє не створювати власні проксі-сервери. Також важливим є питання автоматичного масштабування, адже кількість пристроїв може досягати мільйонів, а керування масштабуванням потребує окремих інфраструктурних рішень. Більшість платформ надає таку можливість без втручання користувача.

1.4.2. Брокер повідомлень.

Брокер повідомлень – модуль, що працює за моделлю publish-subscribe (публікація-підписка).

1.4.2.1. Модель publish-subscribe.

Модель publish-subscribe реалізує передачу повідомлень, де відправники (publishers) публікують повідомлення, при цьому вони не вказують отримувачів та не знають про їх наявність. Отримувачі (subscribers) отримують

повідомлення, при цьому вони не знають про відправників. Для фільтрації повідомлень існує два основних виду фільтрації: за темою та за змістом.

При системі, що базується на темі повідомлення (topic-based system), повідомлення публікуються в «теми» (іменовані логічні каналі). Отримувачі підписуються на певні теми та отримують тільки ті повідомлення, на теми яких вони підписані. Відправник відповідає за визначення тем, на які можуть підписатися отримувачі.

При системі, що базується на змісті повідомлення (content-based system), отримувач отримує повідомлення тільки за умови задоволення визначених отримувачем обмежень. Отримувач відповідає за класифікацію повідомлень, що робить це менш популярним варіантом системи через підвищене навантаження на кінцевих отримувачів.

1.4.2.2. Задача IoT брокера повідомлень.

Брокер повідомлень – основний компонент IoT платформи. Він надає можливість відправляти та отримувати повідомлення до та від будь-якої кількості пристроїв. Моделі повідомлень можуть бути як один-до-одного, так і один-до-багатьох чи багато-до-багатьох. Розділення за темами та контроль прав доступу дає можливість керувати тим, що отримують та надсилають пристрої чи застосунки. Архітектура publish-subscribe вирішує проблему масштабування до певних розмірів, тому в більшості випадків автоматичне масштабування можливе.

1.4.3. Сервіси авторизації та аутентифікації.

Сервіси авторизації та аутентифікації відповідають за контролем доступу пристроїв до брокеру. Існує багато методів авторизації, такі як сесії чи JWT (JSON Web Token). Сучасні IoT-пристрої дозволяють використовувати будь-які протоколи, оскільки логіка, протоколи комунікації та авторизації підлягають програмуванню та мало залежить від архітектури плат, програмного забезпечення низького рівню тощо.

1.4.4. Реєстри пристроїв.

Реєстри пристроїв ідентифікують пристрої та зберігають такі метадані, як атрибути пристроїв, їх можливості та інше. Вони дозволяють розділити пристрої в різні реєстри, контролювати доступ, створення ключів, реєстрацію пристроїв.

1.4.5. Тіньові пристрої.

Тіньові пристрої – віртуальна версія пристрою, що має останній збережений стан пристрою, що дозволяє іншим пристроям чи застосункам зчитувати повідомлення чи працювати з пристроєм. Тіньовий пристрій зберігає останній стан навіть якщо сам пристрій відключений за будь-яких причин. Це дозволяє встановлювати майбутній стан пристрою незалежно від його доступності та потім синхронізувати його при наявності підключення.

1.4.6. Система правил.

Система правил надає можливість приймати певні рішення в залежності від повідомлень, певних показників пристроїв чи інших подій. Користувачеві надається можливість програмувати правила або кодом, або спеціальним візуальним редактором. Також є можливість створювати, видаляти та редагувати правила через API, що дає змогу розробникам створювати власні редактори правил для користувачів.

2. СТВОРЕННЯ ВЛАСНОЇ ПЛАТФОРМИ

2.1. Огляд архітектури платформи.

Платформа складається з окремих модулів: сервіс брокеру, сервіс авторизації, сервіс API та платформи, веб-інтерфейс користувача, база даних.

Сервіс брокеру використовує сервіс авторизації для підтвердження доступу пристроїв та керує повідомленнями, що надходять з пристроїв. Він може бути розміщений на окремих віртуальних серверах та підтримує горизонтальне масштабування за допомогою “bridging” – передачі повідомлень між окремими екземплярами брокерів.

Сервіс авторизації перевіряє правильність JSON Web Token, що надсилають як пристрої, так і користувачі. Цей сервіс надає інформацію про сутності та їх права доступу.

Сервіс API та платформи є клієнтом брокеру, що підписаний на певні повідомлення, теми яких виведені зі збережених правил. Він також використовує сервіс авторизації та надає API для веб-інтерфейсу та прямого користування. Сервіс авторизації та сервіс платформи мають підключення до бази даних.

Веб-інтерфейс користувача є статично скомпільованим додатком, тому він не потребує підключення до бази даних та серверу. Тому він може бути розміщений на статичних серверах за типом Amazon S3 та для нього може бути налаштована CDN дистрибуція за допомогою такого сервісу, як Amazon CloudFront. Масштабування дистрибуції відбувається автоматично за допомогою розміщення файлів на більшій кількості серверів.

База даних може бути розміщена на такому сервісі, як Amazon RDS, що дозволяє уникнути питань резервного копіювання, безпеки серверів, масштабування сховища даних. Також цей сервіс дозволяє створювати Read Replicas – реплікацію на зчитування інформації, що забезпечує швидкість та надійність виконання запитів вибірки даних.

Доступ до всіх сервісів може бути обмежений за допомогою такого сервісу, як Amazon API Gateway. В той же час створення окремих екземплярів платформи може використовувати Amazon Virtual Private Cloud для ізоляції всієї архітектури та забезпечення окремого шару безпеки.

2.2. Процес реєстрації та аутентифікації пристроїв.

Для захисту комунікації між пристроями та платформою було обрано підхід PKI (Public Key Infrastructure). Інфраструктура публічних ключів дозволяє безпечно та однозначно ідентифікувати кожен пристрій без необхідності передавати дані незахищеними каналами. Для кожного пристрою за алгоритмом RSA генерується безпечний приватний та публічний ключ. Публічний ключ дозволяє перевірити, що повідомлення зашифроване приватним ключем, тим самим надає безпеку комунікації між пристроями та сервісами.

За рахунок унікального ключу, у випадку компрометації виключається можливість створення Botnet та отримання доступу до інших пристроїв. Незважаючи на це, рекомендується регулярно оновлювати ключі для зниження шансів зламу.

2.2.1. Створення пари ключів.

Для створення пари, що складається з приватного та публічного ключів, може використовуватися декілька підходів:

- Використання TPM (Trusted Platform Module) для генерації пари на пристрої.
- Використання окремого пристрою для генерації пари та передача приватного ключу на кінцевий пристрій.

Кожен з підходів має певні переваги та недоліки. У випадку використання TPM існують прецеденти, де стійкість ключів була недостатньо високою, в особливості випадок з Infineon у 2017 році, де уразливість дозволяла виводити приватний RSA ключ з публічного. Але TPM, в свою чергу, дозволяє постійно

оновлювати ключі без необхідності втручання ззовні. Також TPM може бути важко вбудувати у дешеві та невеликі пристрої через те, що це окремий модуль.

У випадку використання окремого пристрою (наприклад, генерація ключів на комп'ютері та запис на пристрій під час виготовлення) є проблема безпеки передачі даних на пристрій, а також безпеки пристрою, де генерується ключ. Але це дозволяє не додавати TPM модуль до пристрою, тим самим спростити розробку та зробити продукт дешевше. Такі протоколи, як Z-Wave додають можливості шифрування у власні модулі, що також може оптимізувати витрати на створення пристроїв.

Після створення ключів приватний ключ зберігається на пристрої, а публічний зберігається на платформі. Для їх передачі використовується підключення через TLS, що має виключити можливість перехоплення ключів третьою особу (за умови відсутності фізичного доступу до пристроїв та відсутності зловмисного програмного забезпечення на пристроях).

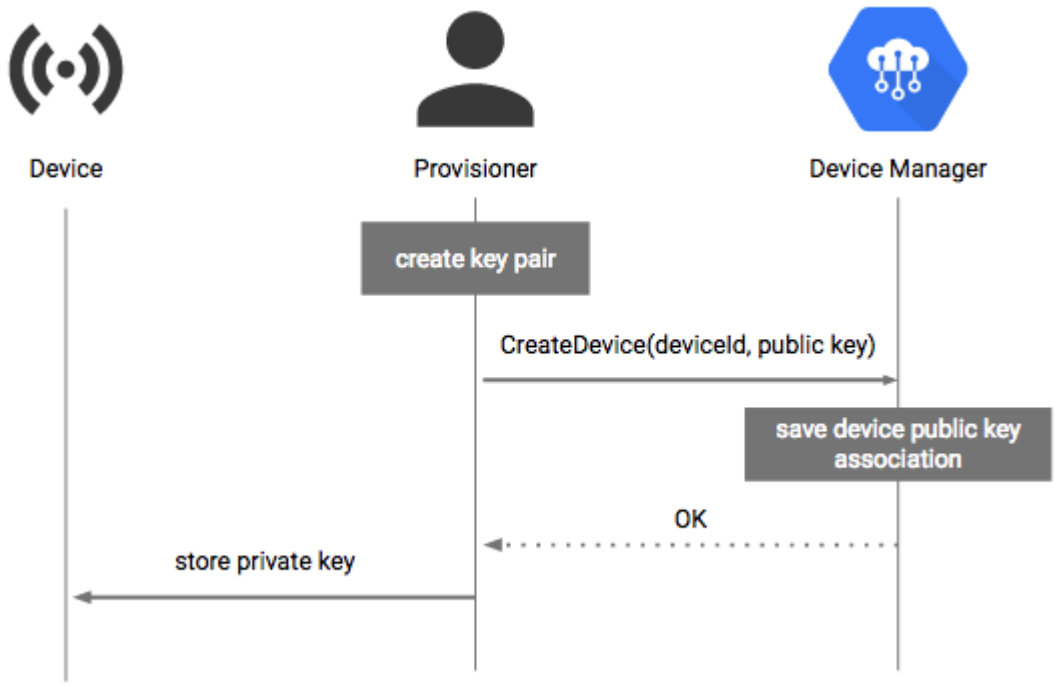


Рисунок 2.1. Процес збереження ключів (Google Cloud IoT)

2.2.2. Використання JWT.

Після створення ключів пристрій отримує змогу ідентифікувати себе на платформі за рахунок підпису інформації про себе приватним ключем. Для цього використовується поширений формат JWT (JSON Web Token). При аутентифікації брокер:

- Декодує JWT.
- Отримує з нього інформацію про пристрій.
- Знаходить в реєстрі публічний ключ пристрою.
- Підтверджує, що JWT був підписаний приватним ключем цього самого пристрою.

У випадку успішної перевірки брокер дозволяє передавати повідомлення.

Відкликання доступу пристрою до платформи можливо за рахунок зміни публічного ключа, зміни статусу пристрою на неактивний, або видалення пристрою з реєстру.

2.3. Комунікація пристроїв та брокер.

Для комунікації пристроїв з платформою використовується протокол MQTT (Message Queuing Telemetry Transport), що є стандартом у індустрії IoT. На відміну від інших протоколів, він спеціалізований для надійної передачі невеликої кількості даних та використанні мінімуму ресурсів IoT пристроїв. Також на платформі можлива підтримка HTTP та WebSockets у якості альтернативних протоколів.

В ролі сервера для брокеру виступає платформа Node.js та бібліотека Mosca. Неблокуюча концепція Node.js дозволяє обробляти велику кількість повідомлень з невеликим розміром, а вбудована підтримка JSON дозволяє зручно працювати з даними.

2.3.1. Вимоги до пристроїв.

Платформа розроблена з мінімальними обмеженнями для розробників пристроїв та встановлює мінімальні обмеження для створення підключення,

передачі та отримання даних тощо. Для того, щоб пристрій міг відправляти дані до платформи, мають бути виконані дві вимоги:

- Повідомлення CONNECT при підключенні до MQTT брокеру має мати наступний формат: user name має бути «JWT», а password – JWT токен, підписаний встановленим приватним ключем. JWT токен має містити поле “id”, за яким пристрій буде ідентифіковано.
- Пристрій повинен підтримувати TLS, тому що всі повідомлення MQTT будуть передаватися через цей протокол.

2.3.2. Маршрутизація повідомлень

MQTT брокер направляє усі повідомлення, що надійшли з пристрою до системи правил, що визначає подальші дії над отриманими даними. З боку платформи також можуть надходити повідомлення, що можуть зчитуватися пристроями.

Теми повідомлень, що надходять з пристроїв, мають дотримуватися формату `device/ідентифікатор_пристрою/назва_повідомлення`. Тіло повідомлення може мати будь-який зміст, але воно повинно бути коректною стрічковою репрезентацією JSON-об’єкта.

Кожен розробник має відокремленого брокера, що мінімізує можливість перехоплення повідомлень інших систем. Навіть у випадку цього, усі повідомлення зашифровані приватним ключем пристрою, що робить дешифрування фактично неможливим. Відокремленість брокерів також дає можливість уникнути штучного перевантаження трафіку, що може тимчасово блокувати передачу повідомлень інших систем та навіть відключити сервер.

2.4. Робота з даними

Існуючі правила дозволяють зберігати поточний стан пристрою або записувати в журнал стан пристрою з певним інтервалом. В ролі ORM виступає Sequelize, що підтримує драйвери MySQL, PostgreSQL, SQLite та інших СУБД.

Для масштабування, безпечності та доступності даних планується використання сервісу Amazon RDS та СУБД MySQL, оскільки ця СУБД показує більш ефективну архітектуру для запису та реплікації даних.

Основні сутності платформи:

- Device (пристрій) – зберігає інформацію про пристрій, метадані, публічний ключ.
- Registry (реєстр) – зберігає список пристроїв, ізолює групи пристроїв від інших реєстрів, має метадані.
- Home (дім) – сутність, яка зберігає та пов’язує пристрої та правила для цих пристроїв.
- Rule (правило) – описує подальшу реакцію платформи на певне повідомлення від певного пристрою.

Платформа розрахована на те, що користувач платформи буде мати реквізити (ключ авторизації) та використовувати публічний API (HTTP/WebSocket) та/або додавати свої веб-хуки (спеціальні URL-адреси, до яких платформа буде відправляти запити, щоб проактивно повідомляти про оновлення даних).

У випадку проблем з передачею даних має бути реалізована спроба повторно відправити старі повідомлення. Ця функція також має бути необов’язковою та конфігуруватися для кожного пристрою, тому що логіка пристрою може вимагати чи не вимагати запис старих та можливо неактуальних даних.

2.5. Створення правил

Правила описують подальший шлях даних від пристроїв після обробки їх брокером. Їх створення можливе через користувацький інтерфейс, але також існує можливість реалізації доступу через API, оскільки динамічне створення правил є одною з важливих частин платформи. Відсутність правил у системі означає, що всі повідомлення з пристроїв буде проігноровано.

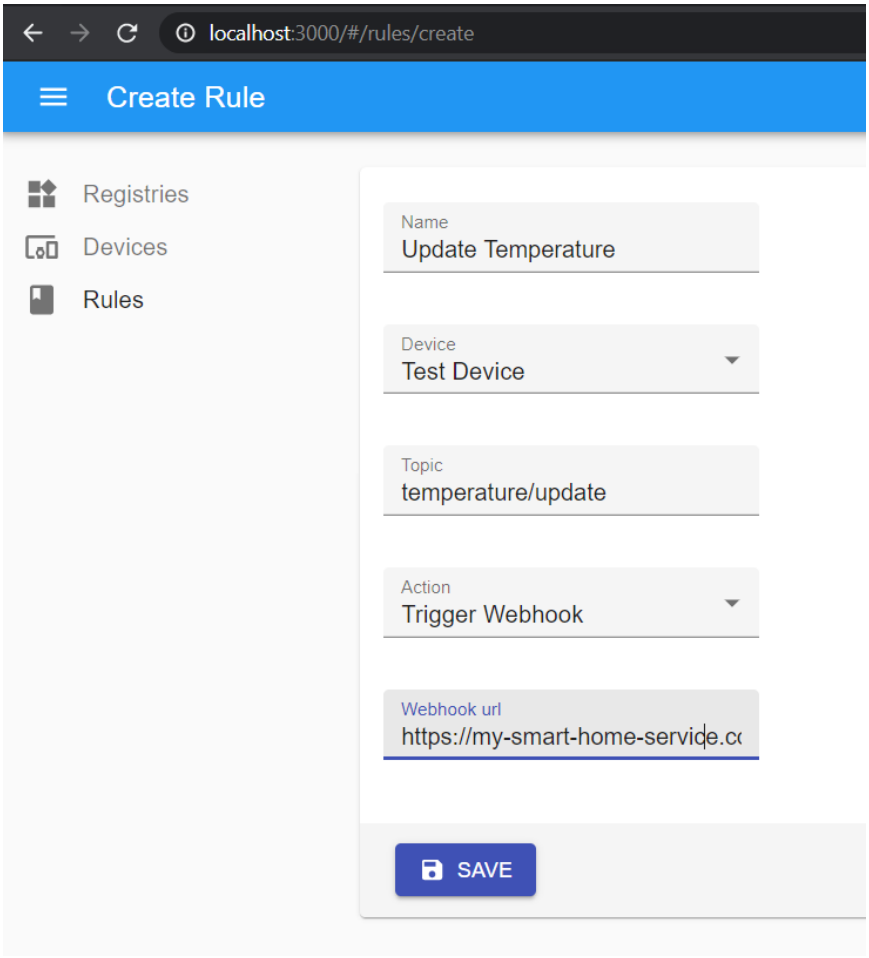


Рисунок 2.2. Створення правила

Для кожного пристрою можна створити багато правил, кожне з яких буде спрацьовувати при співпадінні тем (topic) повідомлень. Формат теми може бути довільним, але для коректної роботи він має співпадати з форматом теми повідомлень, що надсилає пристрій. Платформа видаляє першу частину теми (`device/ідентифікатор_пристрою/`) та залишає тільки те, що йде за цією частиною. Після чого платформа проводить порівняння значень та викликає дію правила, якщо таке правило існує.

Платформа дозволяє викликати веб-хуки, а також зберігати стани пристроїв в базі даних. Потенційно можливі будь-які комбінації правил, що можуть включати в себе код розробника, пряму відправку даних на інші пристрої тощо.

2.6. Сторінка адміністрації.

Сторінка адміністрації була створена за допомогою React та бібліотеки react-admin. Вона дозволяє додавати пристрої, встановлювати їм публічний ключ, створювати реєстри пристроїв та правила для пристроїв.

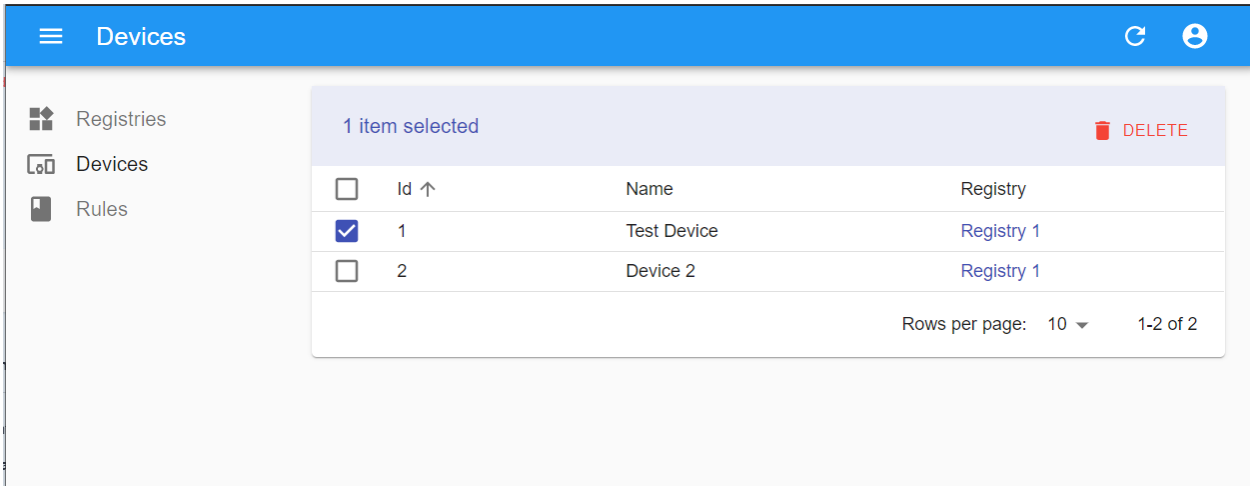


Рисунок 2.3. Вигляд сторінки адміністрації.

Додавання пристроїв через користувацький інтерфейс не є основним методом, оскільки встановлення публічного ключу таким чином є небезпечним, а ручне створення пристроїв займає дуже багато часу. Основний метод – додавання через API.

Створення реєстрів також можливе як через користувацький інтерфейс, так і через API. Кожен реєстр зберігає в собі інформацію про пристрої та метадані.

Оскільки API сервер було створено у стилі REST (Representational State Transfer), то більшість функціоналу можливо використовувати напряму без користувацького інтерфейсу, через API. Це дозволяє розробникам використовувати сам веб-інтерфейс тільки для пошуку проблемних місць та ручного виправлення певних випадків. При цьому більша частина керування усіма елементами платформи проходить через програмні виклики інтерфейсу.

3. ТЕНДЕНЦІЇ РОЗВИТКУ ІОТ ТА РОЗУМНИХ БУДИНКІВ.

3.1. Питання захисту та безпеки IoT пристроїв.

За даними Cisco, на сьогодні кожну секунду до інтернету приєднується 127 нових IoT пристроїв. У 2025 році очікується приблизно 75 мільярдів IoT пристроїв. Кожен з них має загрозу бути атакованим чи зламаним та приєднаним до мережі ботів, або встановленим на збір конфіденційної інформації.

У роботі розглядався TPM як модуль, який дозволяє знизити можливість несанкціонованого доступу до даних, що передаються. Але дуже багато пристроїв використовують більш вразливі рішення, такі як статичний пароль або короткий пароль, що може бути підібраним за час активності вікна авторизації. Перехід до біометричної перевірки, hardware tokens, TPM тощо є наступним кроком до підвищення безпеки пристроїв з боку самих пристроїв.

З боку контролю за мережею також є актуальні розробки, такі як Tempered Networks, що замінюють IP-адреси на криптографічну ідентифікацію, що проходить на захищеному апаратному забезпеченні. Таким чином немає можливості отримати доступ до пристрою без правильного ключа, який майже неможливо підібрати за резонний час.

IoT платформи також працюють над забезпеченням безпеки комунікації пристроїв. Cisco Edge Intelligence забезпечує знаходження аномалій за допомогою штучного інтелекту та аналізу трафіку. Такі платформи, як Azure IoT, Google Cloud IoT, AWS IoT забезпечують менеджмент сертифікатів та ключів, що дає можливість уникнути менш безпечних методів аутентифікації. Azure Sphere – апаратна платформа, що дозволяє створювати пристрої зі спеціальними безпечними мікроконтролерами від Qualcomm та NXP.

3.2. Розвиток платформ для розумних будників.

Ще 5 років тому ринок пристроїв для розумних будинків був помітно сегментований через наявність невеликої кількості пристроїв та великі затрати на розробку під кожну платформу.

Сьогодні такі власники екосистем, як Amazon та Apple, значно полегшили розробку та покращили свої SDK та документацію. Наприклад, раніше для підтримки HomeKit Apple вимагали встановлення патентованого чіпу, що відповідав за шифрування, аутентифікацію та комунікацію. Зараз цю вимогу прибрали, тим самим було прискорено процес вироблення IoT пристроїв, а також суттєво збільшено кількість пристроїв, що підтримують HomeKit.

Окремі виробники пристроїв для розумного будинку, такі як Xiaomi, реалізують універсальну підтримку своїх пристроїв як власним додатком, так і прозорим режимом до HomeKit та Google Home.

У поєднанні з загальним здешевленням вартості виготовлення апаратного забезпечення та покращенням процесів розробки очікується активний зріст ринку Smart Home та Internet of Things.

Висновки

Створення платформи для розумних будинків є комплексною задачею, що вимагає проектування правильної архітектури, такої, що зможе надати автоматичне масштабування, стійкість до аварій та безпечний доступ. Також необхідно забезпечити максимальну гнучкість для користувачів платформи, адже функціонал розробленого на цій платформі програмного забезпечення може сильно варіюватися.

Актуальність та доцільність даної розробки полягає в тому, що питання контролю розумними пристроями стає необхідністю в повсякденному житті, а надання зручних інструментів для розробки та стабільної підтримки мільйонів пристроїв є основою екосистеми розумних будинків.

Результатом виконання роботи є прототип платформи, що дозволяє додавати, перевіряти доступ, та редагувати пристрої. До пристроїв додаються правила, що виконуються при відправленні пристроями повідомлень. В перспективі ця платформа може надати розробникам можливість сфокусуватися на програмному забезпеченні та не витрачати ресурси на менеджмент пристроїв та інфраструктуру повідомлень.

Глосарій

SDK (англ. Software Development Kit) — набір інструментів для розробки програмного забезпечення для певної платформи чи технології.

TPM (англ. Trusted Platform Module) – міжнародний стандарт для безпечного криптопроцесору, мікроконтролеру, що створений для забезпечення безпеки пристроїв за допомогою криптографічних ключів.

TLS (англ. Transport Layer Security — захист на транспортному рівні) – протокол, що надає можливість безпечної передачі даних за рахунок використання асиметричного шифрування та сертифікатів X.509.

MQTT (англ. Message Queue Telemetry Transport) – протокол, що працює за рахунок TCP/IP. Використовується для обміну повідомленнями між пристроями за принципом publisher-subscriber.

API (англ. Application Programming Interface) – прикладний програмний інтерфейс, набір засобів для створення програмного забезпечення.

JSON (англ. JavaScript Object Notation) – формат обміну даними, що прямо підтримується інтерпретатором мови програмування JavaScript.

Список використаної літератури

1. "Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3". hivemq.com. – https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/.
2. AWS IoT – https://aws.amazon.com/iot.
3. Google Cloud IoT Core – https://cloud.google.com/iot/docs/concepts/device-security.
4. PKI: Solving the IoT Authentication Problem by Jeff Chandler – https://www.digicert.com/blog/pki-solving-the-iot-authentication-problem/
5. Auth0: Authenticating Devices Using MQTT – https://auth0.com/docs/integrations/authenticating-devices-using-mqtt\
6. Why Uber Engineering Switched from Postgres to MySQL – https://eng.uber.com/postgres-to-mysql-migration/

Додаток 1

(обов'язковий)

Код компоненту авторизації брокеру.

```
const mosca = require("mosca");
const axios = require("axios");
const jwt = require("jsonwebtoken");

const settings = {
  port: 9999,
};

const server = new mosca.Server(settings);

server.authenticate = (client, username, password, callback) => {
  if (username === "PLATFORM" && password.toString() ===
"PLATFORM_SECRET") {
    client.isPlatform = true;
    return callback(null, true);
  }
  if (username !== "JWT") {
    return callback("Invalid credentials", false);
  }

  const { id: deviceId } = jwt.decode(password);
  axios
    .get(`http://localhost:3001/private/devicekey/${deviceId}`)
    .then((res) => res.data)
    .then((publicKey) => {
      jwt.verify(password.toString(), publicKey, function (err,
profile) {
        console.log(err);
        if (err) {
          return callback("Error decoding token", false);
        }
        console.log("Authenticated device " + profile.id);
```

```
        console.log(profile.topics);
        client.deviceProfile = profile;
        return callback(null, true);
      });
    })
    .catch(() => {
      return callback("Error requesting public key");
    });
};

server.authorizePublish = function (client, topic, payload,
callback) {
  callback(
    null,
    client.isPlatform ||
      (client.deviceProfile &&
        client.deviceProfile.topics &&
        client.deviceProfile.topics.indexOf(topic) > -1)
  );
};

server.authorizeSubscribe = function (client, topic, callback) {
  callback(
    null,
    client.isPlatform ||
      (client.deviceProfile &&
        client.deviceProfile.topics &&
        client.deviceProfile.topics.indexOf(topic) > -1)
  );
};

server.on("ready", setup);

// Fired when the mqtt server is ready
function setup() {
```

```
  console.log("Mosca server is up and running");
}

server.on("clientConnected", function (client) {
  console.log("New connection: ", client.id);
});
```