Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Факультет комп'ютерних наук

Рекомендовано до захисту:
протокол засідання кафедри
№ ___ від ____.____.2024 р.
В.о завідувача кафедри БІСТ
_____ Мелкозьорова О.М.

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему «Моделі та методи розробки програмного забезпечення для систем «Розумного будинку» з урахуванням вимог безпеки»

Оцінка _____ /
_____
Голова ЕК
_____ Лемешко О.В.

Виконав:
студент 4 курсу, групи КБ- 41
Спеціальності:
125 – Кібербезпека_____
Мазніцин М.І.

Керівник доц. каф МсіТ, д.ф. з комп'ютерних наук
Товстокоренко О.Ю
Рецензент Колтиков О.В.

Харків – 2024

АНОТАЦІЯ

Звіт про виконання кваліфікаційної роботи: 52 сторінки, 4 розділи, 20 рисунків, 5 таблиць, 24 використаних джерел посилання, 1 додаток.

Метою кваліфікаційної роботи є визначення та вдосконалення методів захисту та методів розробки програмного забезпечення у сфері розробки на базі Internet of Things (IoT), систем управління «Розумний будинок». Проаналізувати можливі варіанти реалізації програмних рішень.

Методами дослідження є розробка декількох варіантів програмних рішень, шляхом яких буде впроваджено нові функціональні можливості для систем «Розумний будинок» з використанням вбудованих інструментальних засобів та зовнішніх програмних компонентів (Docker, Spring Boot).

Для виконання поставленої мети буде досягнуто наступні цілі: аналіз сучасних методів захисту різних рівнів архітектури IoT / «Розумний будинок»; аналіз методів життєвого циклу в контексті розробки програмного забезпечення (ПЗ) управління для систем «Розумний будинок»; формування нової метрики оцінки захисту SHS (на основі існуючих); аналіз та вибір інструментальних засобів для реалізації запропонованого програмного рішення; розробка вимоги до програмного прототипу; дослідження отриманих результатів рівня захищенності реалізованих програмних компонентів.

Ключові слова: РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧННЯ, КІБЕРЗАХИСТ, АРХІТЕКТУРА, ВАРІАНТИ АТАК, ЖИТТЄВИЙ ЦИКЛ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ЗАХОДИ ЗАХИСТУ, АВТОМАТИЗОВАНА ПРОЦЕДУРА.

ABSTRACT

The report on the performance of the qualification work: 52 pages, 4 chapters, 20 drawings, 5 tables, 24 sources, used 1 application.

The purpose of the qualification work is to define and improve protection methods and software development methods in the field of development based on the Internet of Things (IoT), "Smart House" control systems. Analyze possible options for implementing software solutions.

Research methods include the development of several options for software solutions, through which new functionality will be implemented for Smart Home systems using built-in tools and external software components (Docker, Spring Boot).

To fulfill the set goal, the following goals will be achieved: analysis of modern methods of protection of various levels of the IoT / "Smart House" architecture; analysis of life cycle methods in the context of development of management software (software) for Smart Home systems; formation of a new SHS protection assessment metric (based on existing ones); analysis and selection of tools for implementing the proposed software solution; development of a requirement for a software prototype; study of the obtained results of the security level of implemented software components.

Keywords: SOFTWARE DEVELOPMENT, CYBERSECURITY, ARCHITECTURE, ATTACK OPTIONS, SOFTWARE LIFE CYCLE, PROTECTION MEASURES, AUTOMATED PROCEDURE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

НА – програмне забезпечення Home Assistant для автоматизації будинку,

ОС – операційна система,

ПЗ – програмне забезпечення,

IoT (Internet of Things) – інтернет речей,

SDLC (Software Development Life-Cycle) – життєвий цикл розробки програмного забезпечення,

IT (Information technology) – інформаційні технології,

SHS (Smart Home Systems) – система автоматизації та керування «Розумний будинок»

ЖЦ – життєвий цикл,

SDM (System Development Methods) – методи розробки систем,

ДМ – доменна модель,

ЛПП – лінійка програмних продуктів,

ВСТУП

У сучасному світі відбувається стрімкий розвиток технологій, що спрямований на поліпшення якості життя людей та оптимізацію повсякденного середовища. Серед ключових тенденцій цього розвитку особливе місце займає концепція «Розумного будинку», яка передбачає використання сучасних інформаційно-комунікаційних технологій для автоматизації та управління різними аспектами побутового життя.

Програмне забезпечення є ключовим компонентом будь-якої системи «Розумного будинку», від якого залежить не лише її функціональність, а й рівень безпеки. Оскільки взаємодія з цими системами стає все більш інтенсивною, проблема забезпечення їхньої безпеки набуває все більшого значення.

Система "Розумного будинку" або Smart Home System(SHS) - це мережа підключених до Інтернету пристроїв будинку, які можуть комунікувати між собою та виконувати різноманітні функції для полегшення життя та оптимізації ресурсів [8].

Наразі також активно починають розвиватися IT-компанії, які надають комерційні послуги з розробки ПЗ для різного роду SHS. Розробка програмного забезпечення для систем «Розумного будинку» вимагає використання специфічних моделей та методів, орієнтованих на складність інтеграції різноманітних пристроїв, забезпечення високої надійності та безпеки функціонування, а також максимальної зручності для користувачів. На даний момент часу існує вже багато систем з відкритим вихідним кодом, які може використовувати кожен охочий для виконання власних цілей. Але для того щоб надалі розвивати та робити свій будинок все більш розумним і захищеним потрібно розробити або використати вже наявне додаткове ПЗ, яке можна викачати наприклад з Github. Можна зробити припущення, що в майбутньому кожна людина буде користуватися функціональними можливостями розумного будинку, як наприклад було з появою смартфонів. Чим більше буде користувачів в системі та чим більше буде даних, тим більше буде і зловмисників, які захочуть несанкціоновано проникнути, отримати дозвіл шляхом шахрайських

маніпуляцій. Якщо втрата або викрадення даних з особистого комп'ютеру або смартфону є жахливою подією, а що вже казати про втрату над доступом до власного будинку? Тому розробка власних інтеграцій та розширень з урахуванням вимог безпеки для SHS – є актуальною задачею [9].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1.1 Основні поняття та визначення

«Інтернет речей — це мережа фізичних об'єктів, які мають вбудовані технології, що дозволяють здійснювати взаємодію з зовнішнім середовищем, передавати відомості про свій стан і приймати дані ззовні» [7]. Найпростійший приклад інтернету речей – це те, як можна за допомогою декількох натискань на своєму смартфоні можна дізнатись про стан безпровідних навушників та відслідкувати місцеположення розумного годинника(smart-watch).

Кожна складна структура має містити під собою певний перелік технологій, за допомогою яких ця структура функціонує. IoT не став виключенням.

- Технологія індентифікації об’єкту.
- Сенсори, які будуть відслідковувати зміни у стані елементу чи оточуючого середовища.
- Обробка та накопичення даних з пристроїв повинна виконуватись за допомогою вбудованого комп’ютеру(найчастіше Raspberry-Pi), або з використанням хмарних обчислень(cloud).
- Обмін інформацією між пристроями можна використати технології бездротових мереж(ZigBee, Bluetooth)

На даний момент IoT стала однією з ключових технологій у таких галузях, як промисловість, здоров'я, транспорт, сільське господарство, і багато інших. Вона сприяє впровадженню концепцій "розумних міст", "розумних будинків", "розумного виробництва" та інших ініціатив, спрямованих на покращення якості життя та ефективності виробництва [7]. Сьогодні інтернет речей продовжує швидко розвиватися, впливаючи на всі сфери нашого життя і стаючи неодмінною складовою сучасного цифрового світу. Є багато прикладів, коли користувачі

знаходили якусь прогалину у безпеці в коді і або самі виправляли цю вразливість, або повідомляли головним розробникам продукту.

На рисунку 1.1 можна побачити концептуальну доменну модель, яку було взято з AIOTI(Alliance for Internet of Things Innovation).

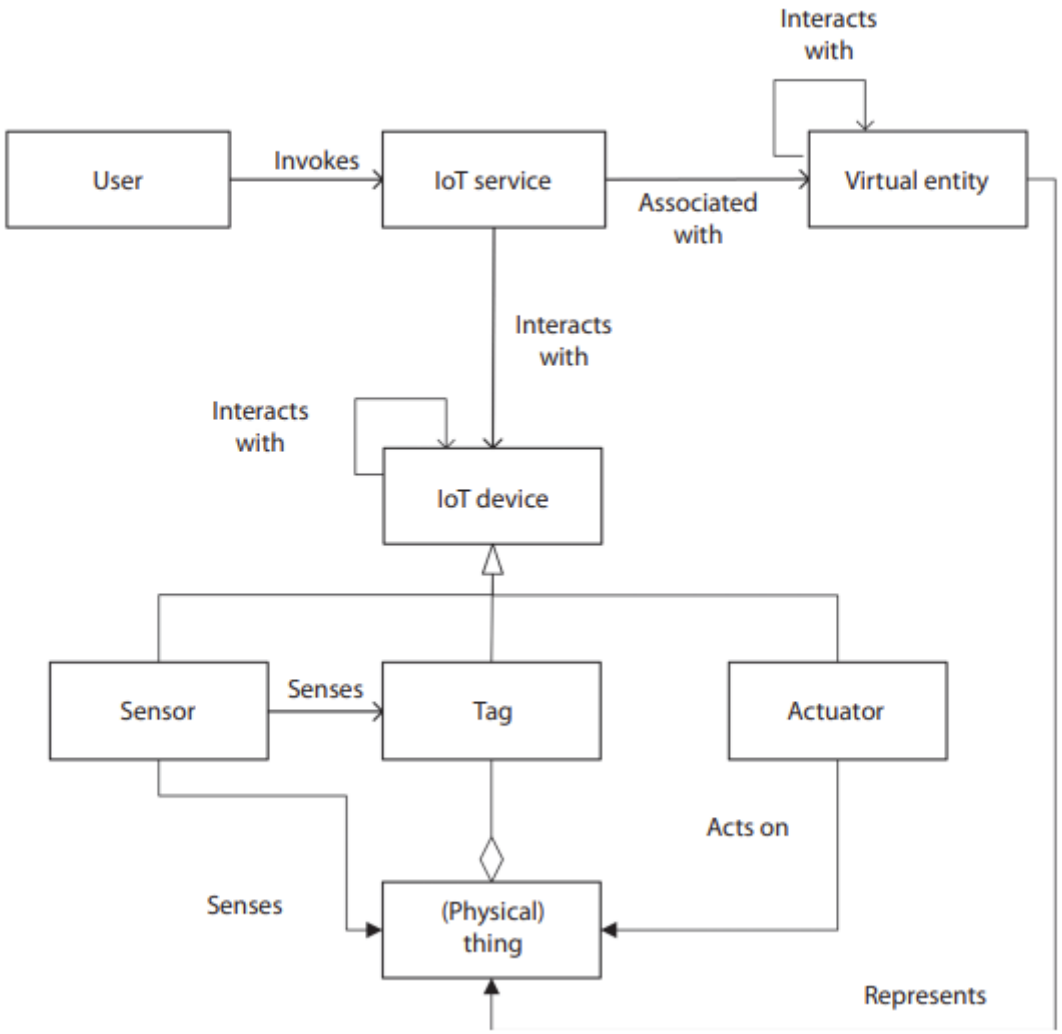

Рис. 1.1– Модель предметної області представляє основні поняття і відносини складових на найвищому рівні [11].

У контексті розумних будинків пристрої IoT — це низка підключених до Інтернету пристроїв, гаджетів і пристроїв, якими можна дистанційно керувати за допомогою смартфона або інших пристроїв з підтримкою Інтернету. Це дозволяє всім пристроям спілкуватися один з одним і з хмарою. Це означає, що розумний термостат тепер може взаємодіяти з розумною системою освітлення, а розумний замок може взаємодіяти з системою безпеки. Все може бути підключено, щоб

створити безперебійний досвід для користувача. У звіті Statista стверджується, що до 2025 року приблизно 75,44 мільярда підключених пристроїв IoT працюватимуть у всьому світі. Це означає, що розумні будинки тут, щоб змінити спосіб нашого життя [8].

«Розумний дім» — це житлове розширення автоматизації будівель і передбачає контроль і автоматизацію всіх вбудованих технологій. Він визначає житло, яке має побутову техніку, освітлення, опалення, кондиціонування повітря, системи безпеки та камери. Ці системи складаються з комутаторів і датчиків, підключених до центрального концентратора, яким керує мешканець будинку за допомогою настінного терміналу або мобільного пристрою, підключеного до хмарних Інтернет-сервісів [9].

1.2 Актуальність проблеми вибору архітектурного підходу до розробки програмного рішення для систем «Розумний будинок»

На даний момент часу у галузі програмної інженерії все більш стрімко розвиваються варіанти та можливості для розробки ПЗ, яке полегшує та пришвидшує роботу розробника, наприклад майже кожного року виходить все більше фреймворків(frameworks), які мають незначні відмінності по своїй суті, наприклад різниця у синтаксисі, реалізації певних методів або використання структур даних. Основною метою цього є спрощення і пришвидшення розробки та супроводу програмної системи.

Відповідно до цього у мережі інтернет можна знайти велику кількість вже готових реалізованих програмних рішень, велику кількість документації та посібників(guides) для кращого використання цього рішення у своїй системі.

Огляд науково-практичних розробок у галузі створення SHS-систем дозволяє зробити висновок, що наразі існує досить значний прогрес у створенні високотехнологічних апаратно-програмних рішень, зокрема, з використання методів і технологій з інструментарію побудови IoT – систем, а також у розробці нових інтелектуальних моделей та алгоритмів керування ресурсами цих систем та розв’язання проблемних ситуацій, що виникають в процесі їх функціонування [1]. Одним з найменш проаналізованих питань у зазначеному домені є питання

підвищення ефективності процесів розробки та супроводу компонентів ПЗ системах «Розумний дім», шляхом забезпечення їх варіабельності та можливостей повторного використання компонентів та конфігурацій системах «Розумний дім», з використанням відповідних кількісних оцінок (метрик) [1]. Важливим питанням також є можливість отримання кількісних оцінок результатів застосування варіабельних проектних рішень з точки зору їх впливу на трудомісткість процесів супроводу ПЗ систем «Розумний будинок» [1].

Існує певна і досить обмежена кількість публікацій, переважно закордонних, у яких розглядаються питання застосування понять: доменного моделювання (ДМ), моделей варіабельності (МВ) програмного забезпечення (ПЗ) та технології побудови лінійок програмних продуктів (ЛПП) саме для розробки SHS-систем. За даними до 2021 року до IoT було підключено приблизно 50 мільярдів технічних пристроїв різного призначення і при цьому вирішальним фактором корисності таких КФС буде повнота їх функціоналу та якість відповідного ПЗ. Різні традиційні підходи до вимірювання ефективність ПЗ потребують вдосконалення та адаптації до характеристик IoT [1].

Причини захисту власного інстансу «Розумного будинку» — крім очевидного ризику заморозити ваш будинок або відкрити двері, доступ до інформації, ймовірно, є більш цінним аспектом. Якщо ви не дотримуєтесь етики надійного складання паролів, то отримання доступу до паролів/секретів у вашій SHS може надати зловмисникам доступ важливих облікових записів. Крім того, з повним доступом до мережі розгортання програм-вимагачів буде легким, і ваша інформація може бути заблокована, доки ви не заплатите за її розблокування. Для того, щоб знайти інстанс будь-якого іншого «розумного будинку» не потрібно рахувати якісь складні алгоритми та аналізувати великий потік даних. Для цих цілей існує дуже багато сервісів, які від самого початку були створені для дослідницьких цілей, але не всі люди є дослідниками саме вашої SHS. Одним за популярніших сервісів є Shodan [6]. Після того, як я ввів запит “Home Assistant” на цьому сервісі, мені видало декілька тисяч результатів

серверів інших користувачів цієї платформи. На рисунку 1.2 зображено результати пошуку серверів «Розумного будинку» за допомогою сервісу Shodan.

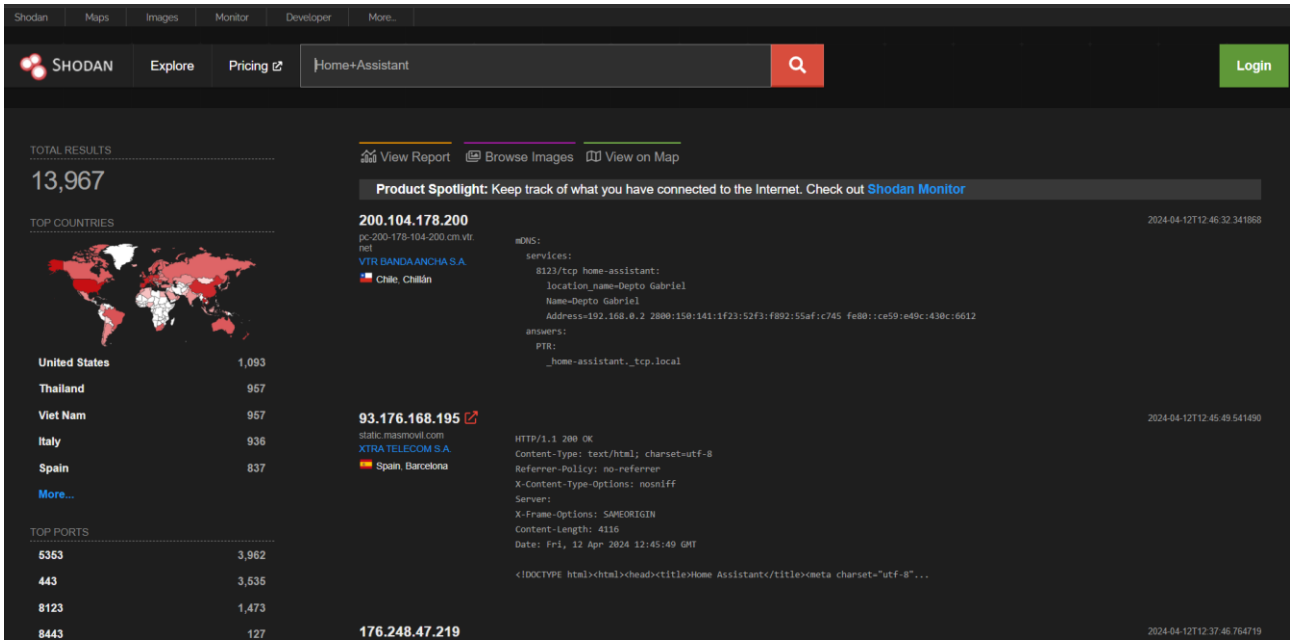

Рис. 1.2 – Результат пошуку з використанням сервісу Shodan

Безпека комлексної системи вимагає багаторівневих інструментів і методологій захисту, щоб запобігти порушенням безпеки. Кожний спосіб/метод безпеки може мати певні вразливості, але коли декілька методів скомпоновані один між одним в один «великий шар» захисту, то на перших етапах потенційну загрозу може бути зупинено і далі в системі є ще рівні для запобігання.

Підсумовуючи, можна сказати, що вивчення та застосування різних методологій та моделей до розробки ПЗ в рамках SHS є актуальним завданням, а доповнення цього арсеналу основними засобами щодо безпеки власних даних, пристроїв та у цілому власного будинку є критичним для забезпечення захисту та підвищення рівня власного комфорту.

1.3 Постановка задачі дипломної роботи

Ця дипломна робота присвячена аналізу, розробці та дослідженню моделей та методів розробки програмного забезпечення для систем «Розумного будинку». Особлива увага зосереджується на вивченні принципів безпеки, що входять до складу цих систем, з метою створення надійних та захищених рішень.

Об'єктом дослідження є моделі та методи розробки для систем розумного будинку з врахуванням вимог та заходів безпеки.

Предметом дослідження є оцінка способів та можливостей щодо розробки власних програмних компонентів.

Мета даної роботи полягає в розробці нових методів, які дозволять ефективно впроваджувати та підтримувати системи «Розумного будинку» з урахуванням вимог безпеки. Аналіз і порівняння існуючих підходів до розробки програмного забезпечення для таких систем, а також виявлення як захищати дані в цих системах, є основними завданнями дослідження.

Для досягнення поставленої мети потрібно виконати наступні цілі:

a) Проаналізувати архітектури SHS.
b) Розглянути варіанти та способи розробки ПЗ для систем «Розумного будинку».
c) Сформувати чіткий перелік для застосування методів захисту структур IoT.
d) Реалізувати власне ПЗ для використання у системі «Розумного будинку»
e) Протестувати реалізоване програмне рішення
f) Оцінити переваги та недоліки імплементації ПЗ з врахуванням вимог безпеки з огляду на архітектуру системи.

Методами дослідження є моделі розробки власного програмного компоненту, який буде впроваджено у систему «Розумного будинку» з врахуванням архітектури та критеріїв безпеки.

Проведений консолідований аналіз та розглянуто SDLC для програмних компонентів «Розумного будинку».

Запропоновано способи та варіанти для імплементації власного програмного рішення для систем керування «Розумний будинок», з врахуванням вимог безпеки.

Використання запропонованих рішень для реалізації програмних компонентів SHS дозволить на етапі проектування та формування вимог оцінити основні витрати по часу, відшукати прогалини у безпеці серед програмних

компонентів, а також забезпечити більшу гнучкість у виборі ПЗ у рамках системи «Розумного будинку».

2 АНАЛІЗ МОДЕЛЕЙ ТА МЕТОДІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМ «РОЗУМНОГО БУДИНКУ». ПРИВАТНІСТЬ ТА БЕЗПЕКА

2.1 Архітектура та програмні компоненти

В останні десятиліття розумні будинки перетворилися на житлові приміщення, які дозволяють пристроям, підключеним до Інтернету, контролювати домашнє обладнання та системи. Інтернет речей (IoT) відноситься до цих гаджетів, які оснащені датчиками, приводами, програмним забезпеченням і можливостями обробки даних [5]. Розумні домашні гаджети спочатку використовувалися як тип дистанційного вимикача, але згодом перетворилися на пристрої, які можуть керувати нашими будинками на основі встановлених шаблонів і сценаріїв або уподобань користувача. Архітектура IoT Smart Home розділена на багато рівнів, таких як сприйняття(perception), передача(transport), мережа(network) та ПЗ [10]. Ці рівні гармонійно доповнюють один одного та дозволяють користувачу контролювати пристрої IoT з будь-якого місця та у будь-який час. На рисунку 2.1 зображено приклад простої рівневої архітектури IoT-системи.

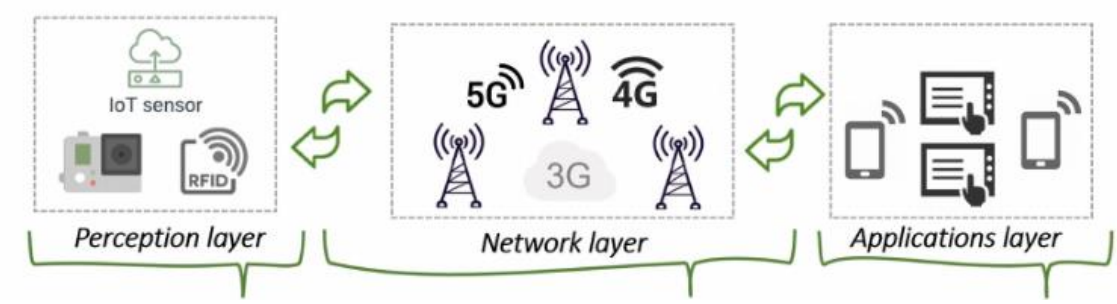

Рис. 2.1– Приклад простої рівневої архітектури IoT-системи [10]

Рівень сприйняття середовища розумного дому фокусується на фізичному компоненті, який містить апаратні пристрої. Датчик є одним із фізичних ресурсів, який може контролювати навколишнє середовище, сприймаючи рух, світло, двері та температуру, отже, збираючи інформацію та передаючи її до пристроїв через мережу [10].

Дані бездротовим способом передаються від кінцевих пристроїв до хмарної платформи через LPWAN. MQTT і HTTP — це два протоколи, які Cloud IoTCore підтримує для ефективної передачі даних [5]. Використовуючи або міст MQTT, або міст HTTP, пристрої підключаються до Cloud IoT Core. Крім того, дані, зібрані з бездротової мережі датчиків, будуть завантажені в хмарне сховище через шлюзи [10]. Після отримання дані зберігаються в хмарі та будуть доступні для користувачів там. Внутрішня програма, пристрої зберігання даних і зовнішні програми формують хмарне рішення.

Рівень програмного забезпечення відповідальний за відображення та керування пристроями на інформаційній панелі або для віддаленого моніторингу, наприклад за допомогою мобільного застосунку.

2.2 Моделі та методи для розробки ПЗ

Життєвий цикл розробки програмного забезпечення (SDLC) — це процес або методологія, яку дотримуються команди розробників програмного забезпечення для проектування, розробки, тестування, впровадження та підтримки програмних систем [1].

SDLC також включає в себе моделі і методології, які команди розробників використовують для розробки програмних систем, при цьому методології формують основу для планування і контролю всього процесу розробки. В даний час існує дві категорії методологій SDLC: традиційна розробка програмного забезпечення (наприклад, водоспад, Rational Unified Process) і AGILE розробка ПЗ (наприклад, SCRUM, Kanban, SAFe) [1].

Розробка технологічних систем є доволі складною роботою, особливо якщо ця система окрім ПЗ включає в себе апаратне забезпечення та комунікаційні компоненти. Щоб зменшити та мати контроль над цією складністю – було запропоновано багато різних методів розробки систем(SDM). Основна мета SDM в IoT – це скеровувати команду проекту в розробці та поєднанні складних компонентів, щоб мати змогу задовільнити потреби користувачів.

Методи розробки систем надають багато переваг для розробки якісного продукту, такі як [3]:

a) SDM надає інженерам набір інструкцій для розробки деяких артефактів і їх перевірку на відповідність вимогам, визначеним у постановці задачі.
b) Оскільки SDM формалізують певні процедури проектування та розширюють дизайнерське мислення, вони допомагають уникнути виникнення непомічених проблем у процесі розробки та мають тенденцію до розширення пошуку відповідних рішень, заохочуючи і дозволяючи інженеру думати не тільки про перше рішення, яке спадає на думку.
c) Вони допомагають забезпечити логічну узгодженість між різними процесами та фазами розробки. Це особливо важливо для розробки комплексних систем.
d) SDM допомагає зменшити можливі помилки в процесі розробки та забезпечує правила оцінки проектних рішень.

Розробниками у сфері IoT технологій було запропоновано декілька варіантів методологій, основні з них про які я хочу далі розповісти - це Ignite та IoT-Meth [23].

Ignite призначений для різних зацікавлених сторін IoT, включаючи менеджерів продуктів, менеджерів проектів і архітекторів рішень. Методологія включає дві основні групи дій: реалізація стратегії і надання рішень IoT [23].

Реалізація стратегії IoT стосується бізнес-перспектив і передбачає виявлення можливості, розробку бізнес-моделі та прийняття рішень про те, як керувати цими можливостями (наприклад, внутрішній проект, зовнішнє придбання тощо).

Надання рішення концептуально визначається на етапі реалізації стратегії IoT і має життєвий цикл, що складається з планування, створення та експлуатації системи. Планування починається з ініціації проекту, під час якого надається

початковий дизайн рішення та організаційна схема проекту. Крім того, слід провести аналіз зацікавлених сторін, середовища, вимог, ризиків і ресурсів.

Після ініціації завдання виконуються в рамках семи потоків [23]:

- управління проектами (охоплює дії з ініціювання, планування, виконання, моніторингу, контролю та закриття проекту.);
- наскрізні завдання(забезпечення безпеки, тестування);
- інфраструктура рішень і операції(установка та керування апаратним та програмним забезпеченням);
- серверні служби(служби, які розміщуються у хмарному середовищі та взаємодіють з пристроями IoT);
- комунікаційні служби(встановлення та управління комунікаційною структурою);
- компоненти активів;
- підготовка активів(виробництво або закупівля компонентів для функціонування системи).

IoT-Meth не визначає чітко визначені ролі з описами та обов’язками. Він стосується лише деяких ролей, таких як кінцевий користувач, дизайнер, виконавець і керівник проекту, без будь-яких деталей. На рисунку 2.2 зображено діаграму технологічного процесу методології Ignite.

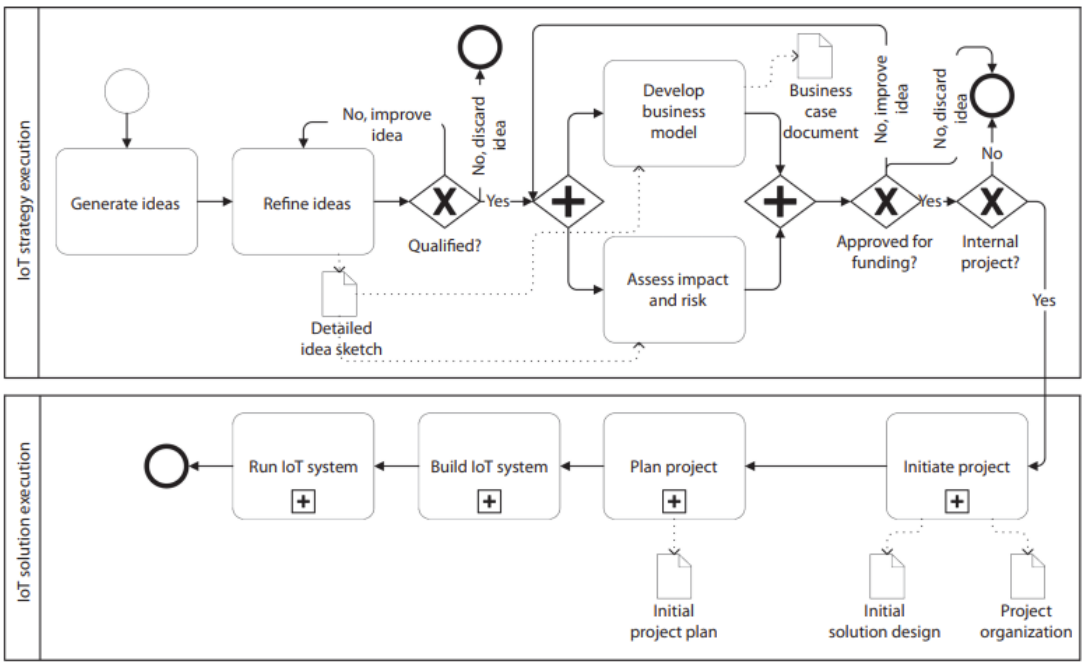

Рис. 2.2– Діаграма технологічного процесу Ignite [23]

IoT-Meth - це загальний легкий метод, побудований на ітераційному прототипуванні [23]. Він передбачає наступні ітераційні кроки:

a) Генерація ідеї (спільне створення): цей крок передбачає визначення проблемних областей шляхом спілкування з зацікавленими сторонами, особливо з кінцевими користувачами. Мета полягає в тому, щоб генерувати ідеї щодо потенційних проблем з точки зору бізнесу. Деякі з цих ідей розроблено разом із їхніми варіантами використання, які потрібно вдосконалити на наступному кроці.

b) Уточнення ідей: деякі з ідей, виявлених на попередньому кроці, доопрацьовуються далі, щоб передати їх керівникам проектів, дизайнерам і виконавцям.

c) Концептуалізація проекту (запитання та відповіді): цей крок передбачає подальший аналіз уточнених ідей, щоб усунути розрив між ідеєю та реалізацією. Вимоги аналізуються та підтверджуються [23].

d) Розробка архітектури (IoT OSI): на цьому кроці вимоги зіставляються з архітектурою та інфраструктурою.

e) Підтвердження концепції (прототип): прототип — це ранній зразок, модель або випуск продукту, створений для перевірки концепції або

процесу. Цей крок включає створення прототипу та ітерацію до мінімально життєздатних систем IoT [23].

f) Розгортання системи IoT (розгортання): останній крок стосується розгортання системи та закриття циклу зворотного зв'язку. Як правило, система постійно вдосконалюється за відгуками.

На рисунку 2.3 зображено діаграму технологічного процесу методології IoT-Meth.

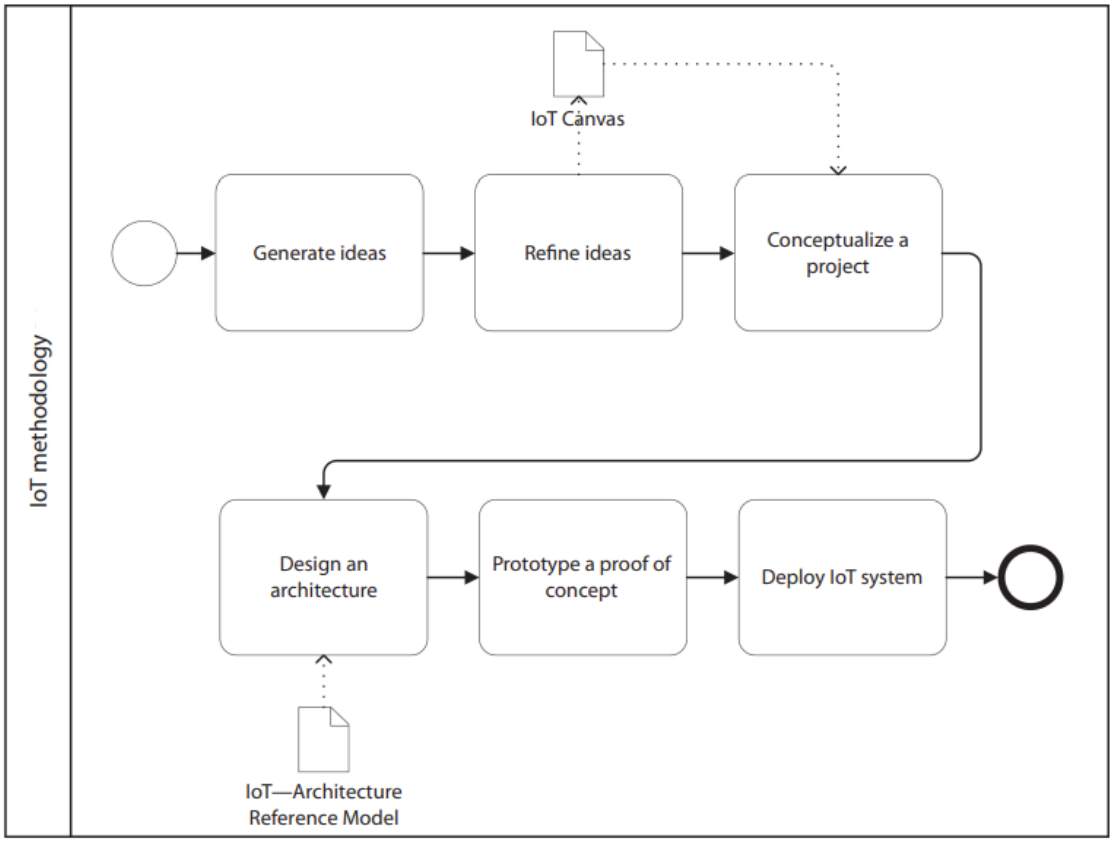

Рис. 2.3 – Діаграма технологічного процесу IoT-Meth [23]

У висновку можна виділити основні етапи SDLC для IoT:

- Збір і аналіз вимог: цей етап передбачає визначення потреб користувачів і зацікавлених сторін системи IoT. Розробники повинні розуміти призначення системи, типи даних, які збиратимуться й оброблятимуться, а також вимоги до безпеки, масштабованості та надійності.
- Проектування системи: після того, як вимоги будуть зібрані та проаналізовані, розробники можуть приступати до проектування

системи IoT. Це включає в себе проектування архітектури системи, апаратних і програмних компонентів, а також протоколів зв'язку [23].

- Розробка ПЗ: ця фаза передбачає в собі розробку програмного забезпечення для пристроїв IoT. Це включає в собі розробку користувацького інтерфейсу(UI), проміжного програмного забезпечення, хмарних платформ тощо.
- Тестування та реліз: після того як ПЗ було розроблено, його необхідно ретельно перевірити, щоб переконатися, що воно не містить помилок та недоліків. Після тестування його можна розгортати в системі IoT.
- Технічне обслуговування та підтримка: система IoT потребує обслуговування та підтримки після розгортання. Це включає виправлення помилок, додавання нових функцій і надання підтримки користувачам.

2.3 Кіберзахист IoT пристроїв

Технологічні та інноваційні тенденції спонукали людей адаптуватися до комфортного способу життя, який надають розумні домашні пристрої. Незважаючи на те, що ці пристрої дуже зручні у використанні та мають певні переваги щодо захисту та безпеки, все ж таки вони схильні до ризиків кібербезпеки. Головною причиною взлому будь-яких систем є сам користувач. Через це для користуванням SHS потрібно розробити чіткі інструкції та засоби безпеки.

2.3.1 Види атак та їх запобіжні заходи для систем «Розумного будинку»

Незахищений зв’язок із витоком приватної інформації про будинок і його мешканців, а також уразливості пристроїв, які можуть бути використані зловмисниками для шпигунства чи іншого віддаленого втручання в життя мешканців, є однією з ключових проблем безпеки SHS [15]. Як вже було розглянуто SHS складається з чотирьох рівнів: мережевий, програмний, транспортний, сприйняття. Так як ці тісно взаємодіють один з одним необхідно врахувати заходи безпеки для кожного з них, адже якщо один рівень буде зламаний, то отримати доступ до всіх інших компонентів не складе труднощів.

Рівень передачі: MQTT розроблено для передачі повідомлень IoT та M2M; дані передаються через TCP. MQTT використовує IP/TCP і використовує захист транспортного рівня для забезпечення шифрування всього з'єднання. MQTT важливий для даних на основі подій або потокових даних. Багато додатків IoT використовують MQTT, оскільки його можна використовувати в обмежених додатках і надсилати корисне навантаження. Оскільки HTTP використовує TLS над TCP, CoAP використовує Datagram TLS над UDP. На рисунку 2.4 зображено основні види атак, які можна використати на транспортному рівні [10]:

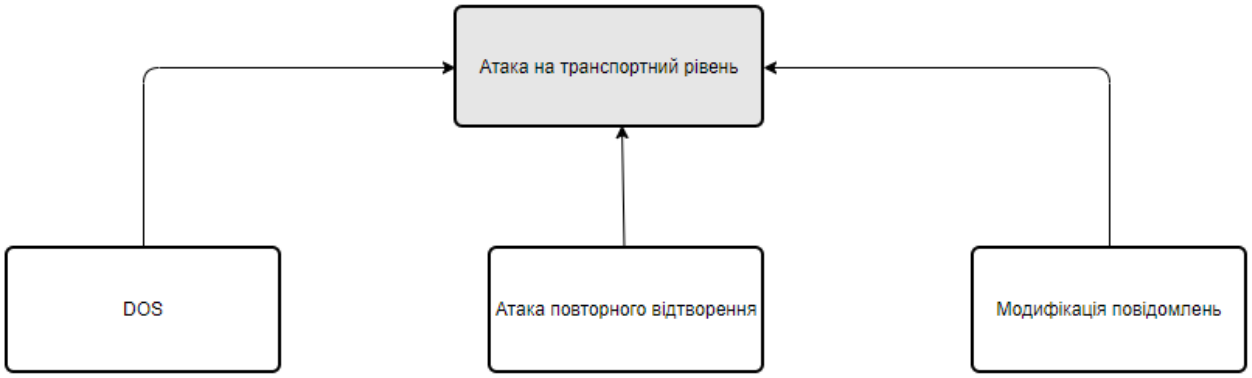

Рис. 2.4 – Діаграма видів атак на мережевий рівень SHS

Рівень сприйняття: його часто називають сенсорним шаром. Він відповідає за ідентифікацію об'єктів і вилучення з них інформації. Кілька типів датчиків для збору інформації включають RFID, 2-D штрих-код і датчики [10]. Датчики вибираються на основі вимог додатків. MEMS є одним із таких датчиків, які виявляють зміни в оточенні та відстежують фізичні чи навколишні умови в домівках. Крім того, обертальний рух вимірюється за допомогою гіроскопів. Таким чином, він може визначити, коли двері чи вікно відкриваються та закриваються. Спостерігаючи за їх рухливістю, датчик положення може визначити, чи є люди або інші об'єкти в певному просторі. Завдяки цим датчикам власник може відстежувати двері та вікна кімнат і прилади з будь-якого місця для безпеки будинку. Інформація, зібрана цими датчиками, може стосуватися положення, якості повітря, оточення, руху, вібрації тощо. Однак вони є основною ціллю зловмисників, які намагаються підробити ці датчики або

заглушити їх [10]. Отже, більшість загроз на цьому рівні стосуються цих датчиків.

Програмний рівень: протоколи прикладного рівня є вирішальними в неоднорідному середовищі IoT. Вони визначають зв'язок між пристроями IoT і мережею. Ці пристрої можна відстежувати та контролювати за допомогою кількох програм у середовищі SHS. Слабкість системи на рівні додатків може бути використана для компрометації безпеки всієї мережі IoT і доступу до відповідних персональних і приватних даних.

Мережевий рівень: пристрої розумного дому використовують протоколи мережевого рівня, такі як Bluetooth, IrDA, WiFi, ZigBee, RFID, NUWB, NFC, Wireless Hart та інші комунікаційні технології для підключення пристроїв, мереж і серверів [10]. Зловмисники, швидше за все, націлені на бездротові мережі. Основні ризики під час мережевого зв'язку пов'язані з поганими налаштуваннями конфіденційності та автентифікації. Більше того, зловмисники використовують незахищені мережі для здійснення атак мережевого підслуховування, також відомих як атаки мережевого перехоплення або мережевого спостереження. На рисунку 2.5 зображено основні види атак на цей рівень.

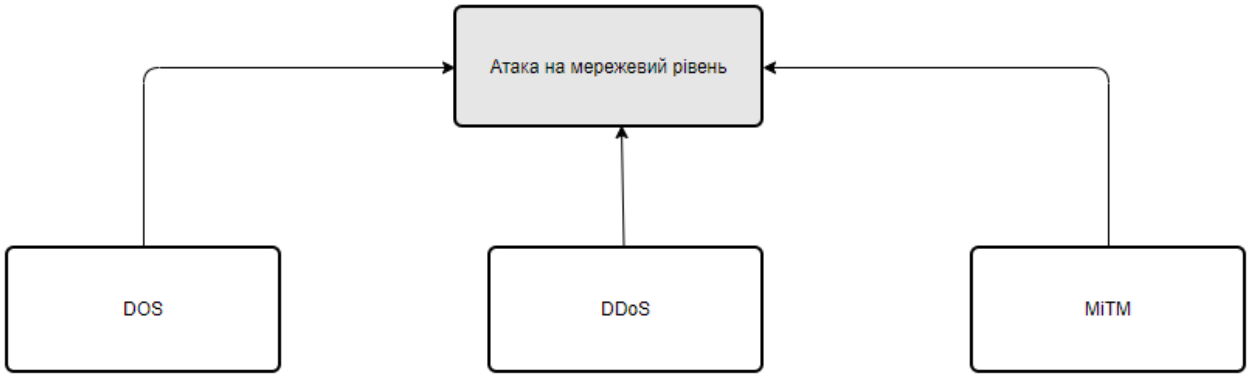

Рис 2.5 – Види атак на мережевий рівень SHS

У таблиці 2.1 перелічено рівні архітектури «Розумного будинку», види атак кожного рівня, та на які програмні/апаратні компоненти ці атаки впливають.

Таблиця 2.1 – Перелік атак на SHS

Рівень	Компонент	Вид атаки
Сприйняття	Фізичні об’єкти, датчики, актуатори	Фізична шкода, підслуховування, захоплення вузла, повторна атака, тимчасова атака
Мережа/Передача	Маршрутизатор, шлюзи, 3G, 4G	Повний контроль, Прослуховування, Аналіз трафіку, DDoS-атака, Людина посередині(Man in the middle), DDoS
Програма	Програмні компоненти	Міжсайтовий сценарій(Cross-Site Scripting), Атака шкідливим кодом

Основними заходами безпеки, щоб захистити свій дім від такого широкого спектру атак є:

а) Належне налаштування маршрутизатора. На мережевому рівні домашні мережі створюють неоднорідне середовище, підключене до Інтернету через домашній шлюз, що має обмежені ресурси. В SHS маршрутизатор Wi-Fi і є шлюзом. З різноманітністю підключених пристроїв у домашньому шлюзі та проблемами трафіку, спричиненими недоліками безпеки в домашній мережі, TR-069 є протоколом управління CPE WAN (CWMP), який полегшує моніторинг усіх онлайн-пристроїв і забезпечує високу якість обслуговування [10]. Саме маршрутизатор є таким собі «клеєм», який утримує всі гаджети IoT і є ключем до їх корисності. Не вказуйте назву маршрутизатора за замовчуванням, краще використовуйте ім’я, яке не відповідає моделі чи виробнику маршрутизатора. До розумної домашньої мережі можна легко отримати доступ, якщо інші знають марку

та модель розумного пристрою. Необхідно шифрувати дані, нарешті, обрати найнадійніше шифрування, яким є WPA3 (захищений доступ Wi-Fi). Він, як правило, замінює WPA2 і додає більше покращень безпеки. Приймаючи новий WPA3, можна захистити бездротову безпеку в середовищі IoT від злому, прослуховування та вторгнення. Індивідуальне шифрування підтримується WPA3, що унеможливлює доступ пристроїв до даних один одного, навіть якщо вони підключені до однієї мережі [15] Зловмисники зосереджуються на домашніх маршрутизаторах як головній цілі в Інтернеті речей [16]. Це означає, що розумний дім із безпечним маршрутизатором є набагато захищеним.

b) Політика надійних паролів. Надійний пароль важливий не лише для мережі Wi-Fi; це також важливо для облікового запису електронної пошти та інших облікових записів онлайн. Кожен обліковий запис і програма кожного пристрою IoT повинні мати власний унікальний набір облікових даних.

c) Додаткова Wi-FI мережа лише SHS. Функція гостьової (або додаткової) мережі доступна на великій кількості сучасних маршрутизаторів. Це дозволяє гостям, членам родини та друзям отримувати доступ до мережі, яка не підключена до пристроїв інтернету речей. У результаті лише користувач і його родина мають доступ до розумної домашньої мережі. Пристрої IoT не можуть отримати доступ до ноутбука чи смартфона, якщо вони встановлені в окремій мережі, тому зловмисники не зможуть проникнути до системи керування «Розумного будинку».

d) Постійне оновлення для ПЗ та прошивки пристроїв. Хоча оновлення для програмних компонентів повинно виконуватись автоматично, для гаджетів, це потрібно буде робити самостійно, бажано робити такі аудити кожні 2-3 місяці.

e) Не використовувати сторонні служби для збору та аналізу даних. Суть цього заходу в тому, що уникнути витоку «чутливих» даних, які сторонні сервіси можуть збирати про ваше помешкання. Якщо не має можливості

припинити користуватися сторонніми сервісами, то можна налаштувати дозволи обмеження часу. Для більшої конфіденційності обмеження роздільної здатності може забороняти доступ до роздільних здатностей менше 15-хвилинного агрегування. Як необов'язкове обмеження воно незначно впливає на зручність використання та підвищує конфіденційність користувача.

f) Використання мультифакторної автентифікації. Багатофакторна автентифікація надає додатковий рівень захищенності системі та запобігає деяким видами атак, наприклад brute-force. Основна концепція полягає в тому, щоб, у користувача, який запитує доступ до системи, отримувати не тільки логін та пароль, але й ще додаткові параметри своєї особистості.

g) Використання брандмауеру наступного покоління (NGFW). Можливо, брандмауер, включений у маршрутизатор, не справляється зі своїми завданнями. Захист від зловмисного програмного забезпечення, фільтрація вмісту, перехоплення SSL/SSH, керування QoS і перехоплення віртуальної приватної мережі (VPN) відсутні в стандартних брандмауерах (VPN) [10]. Завдяки NGFW можна отримати брандмауер і всі інші функції безпеки в типовому брандмауері, все в одній інтегрованій мережевій платформі. На додаток до стандартних функцій брандмауера, NGFW може виявляти та захищати від кібератак. Вартість брандмауерів наступного покоління висока, але вони забезпечують значне підвищення безпеки для розумних будинків.

3 РОЗРОБКА ПРОЦЕДУРИ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ВИБОРУ ПРОГРАМНИХ РІШЕНЬ «РОЗУМНИЙ ДІМ» З УРАХУВАННЯМ ВИМОГ БЕЗПЕКИ

3.1 Схема методології Scrum з урахуванням необхідності ВИБОРУ архітектури для програмних рішень «РОЗУМНИЙ ДІМ» включаючи вимоги безпеки

Методологія Scrum - це гнучкий та ітеративний підхід до розробки програмного забезпечення, який дозволяє командам ефективно працювати над проектами зі збільшенням продуктивності та якості. Як прийнято вважати Scrum зазвичай використовується у проектах, де розробляється тільки програмне забезпечення, і можна припустити що для розробки IoT-системи, такої як «Розумний будинок», ця методологія не зовсім підійде, але тут є певні хитрощі та й не дарма цю методологією називають «гнучкою».

Застосування широко поширених процесів SDLC для проектування, створення, тестування, розгортання і підтримки систем IoT стикається з проблемами, не схожих на ті, які виникають при розробці традиційних програмних систем, через властивих екосистемі IoT особливостей. В першу чергу, з технологічної точки зору, існує значна кількість прогалин в знаннях в області програмної інженерії, що стосуються IoT, включаючи проектування, розробку і тестування. По-друге, з точки зору робочої команди, розробникам необхідно мати ширшу базу знань, від надвеликих систем і закінчуючи програмуванням вбудованих систем [1].

Класичний Scrum-project складається з 5 фаз [4]:

a) Ініціація – цей етап включає процеси, пов’язані з ініціюванням проекту.
b) Планування та оцінка. Ця фаза складається з процесів, пов’язаних із завданнями планування та оцінювання, які включають створення історій користувачів, затвердження, оцінку та фіксацію історій користувачів, створення завдань, оцінку завдань і створення резерву спринту.

c) Реалізація – ця фаза пов’язана з виконанням завдань і дій для створення продукту проекту. Ці види діяльності включають створення різноманітних результатів, проведення щоденних «мітингів».
d) Огляд та ретроспектива. Ця фаза пов’язана з переглядом результатів і виконаної роботи та визначенням шляхів покращення практик і методів, що використовуються для роботи над проектом.
e) Реліз — на цьому етапі наголошується на доставці прийнятих результатів замовнику, а також на ідентифікації, документуванні й узагальненні уроків, отриманих під час проекту.

Ці фази цілком підходять і для розробки SHS-системи, проте деякі процеси будуть відрізнятися від «класичної» методології. Також необхідно врахувати та поставити на перше місце фактор безпеки, адже у такій комплексній системі, де є багато пристроїв та потенційних прогалин у безпеці, не врахувати хоча б один аспект, то увесь кінцевий продукт можна вважати не ефективним і найближчим часом цим продуктом і усіма даними буде володіти зловмисник.

При розробці програмних рішень для SHS потрібно враховувати особливості архітектури та програмних реалізацій, які можуть використовуватись в рамках тієї або іншої платформи. Існує багато різних платформ, які можуть бути використані для автоматизації різних аспектів життя в будинку такі як: Home Assistant, Open Hub, Google Assistant та інші.

Під час розробки нового функціоналу для SHS, перш за все потрібно визначитися з тим чи потрібно нам збільшувати функціональні можливості системи, чи можливо задовільнити потреби користувача на основі вже існуючих інструментів створення ПЗ в рамках SHS. В рамках цього питання ми будемо розглядати декілька варіантів програмних рішень: інтеграції, розширення та компоненти візуалізації, основні з них зображені на рисунку 3.1.

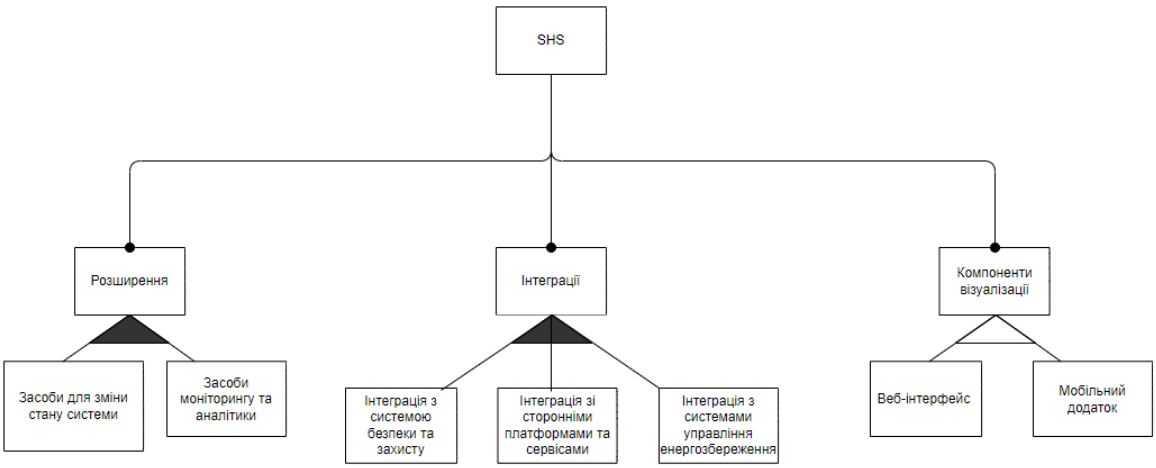

Рис. 3.1 – Фіча діаграма програмних рішень для інтеграції зовнішніх систем SHS

В залежності від поставлених вимог можна зробити висновок, щодо вибору платформи, який спосіб реалізації функціоналу обрати, чи можливо обійтись вже існуючими рішеннями та багато іншого.

На рисунку 3.2 наведено схему методології Scrum для програмних рішень «Розумний дім» включаючи вимоги безпеки:

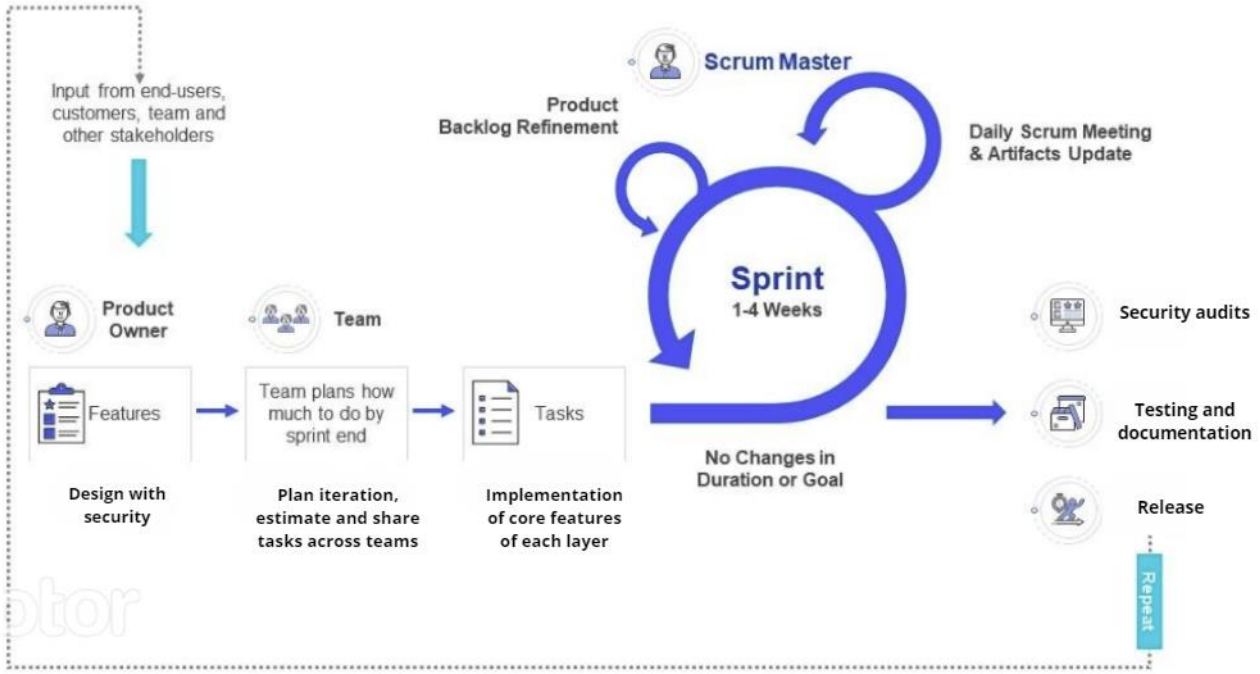

Рис. 3.2 – Схема методології Scrum для SHS з вимогами безпеки [24]

3.2 Вимоги безпеки, що необхідно враховувати під час процесу вибору архітектури для програмних рішень «РОЗУМНИЙ ДІМ»

Розробляючи системи розумного будинку з урахуванням безпеки з самого початку, можна створити системи, які є більш безпечними та стійкими до кіберзагроз.

Це особливо важливо в контексті розумних будинків, де вразливі місця можуть мати серйозні наслідки як для особистої конфіденційності, так і для фізичної безпеки. Однією з основних проблем безпеки IoT є той факт, що безпека традиційно не враховується при проектуванні та виробництві підключених пристроїв і об’єктів. Тому, оскільки IoT продовжує масово розширюватися, а в нашому житті стає все більше підключених пристроїв, важливо, щоб була система, яка гарантувала б більш сувору безпеку.

На даний момент часу, на жаль, пристрої IoT, системи «розумного будинку» створюється не для того, щоб бути захищеними і надійними, вони створюються для того, щоб був ширший, більш гнучкий функціонал з точку зору досвіду користувача. Через це потім у системі керування розумним будинком, у кожному програмному компоненті буде по невеличкій прогалині у безпеці, що у майбутньому створить великий ризик несанкціонованого доступу в систему та отримання повного керування системою. В даному розділі буде перелічено основні вимоги безпеки, та методи захисту для програмних рішень «Розумний дім».

Основними вимогами щодо безпеки інформації є:

a) Моніторинг активності. Необхідно встановити системи моніторингу, які відслідковують активність у SHS. Це може виявити незвичайну або підозрілу діяльність, яка може бути результатом атаки.
b) Захист мережі. Необхідно забезпечити захист мережі Wi-Fi, яка з'єднує всі пристрої розумного будинку. Необхідно використовувати надійні паролі, шифрувати з'єднання і встановлювати мережеві заходи безпеки, такі як брандмауери.

c) Контроль доступу. У великих системах, де користувачі повинні мати різні рівні доступу до функціональних компонентів системи, повинно бути розмежування з використанням привілей. Зазвичай це «адміністратор» та група користувачів.

d) Шифрування даних. Відсутність будь-якої видозміни даних для неавторизованих користувачів може призвести до витоку чутливої інформації. Щоб уникнути цього потрібно використовувати надійні алгоритми шифрування; використання E2EE методу(шифрує дані на пристрої та дешифрує у місці призначення). Зазвичай, в рамках розумного будинку шифрування застосовується для інтеграцій з різними сервісами, таким як “Google” наприклад.

e) Дотримання стандартизації. Коли у вас є кілька пристроїв IoT від різних постачальників. У цих випадках в екосистемі IoT може не існувати універсальних вказівок або протоколів. Це може спричинити проблеми сумісності та труднощі з інтеграцією пристроїв у SHS.

f) Оновлення ПЗ до останньої версії та резервне копіювання даних.

3.3 Процедура для підтримки процесу вибору програмних рішень «РОЗУМНИЙ ДІМ» з урахуванням вимог безпеки

Процес вирішення вибору програмних рішень ґрунтується на п’яти основних етапах:

- визначення функціональних вимог;
- встановлення вимог безпеки;
- вибір спеціалізованої платформи, яка покриє вимоги безпеки та зможе забезпечити гнучкість, масштабованість та вирішення способу реалізації для програмного компоненту;
- процедура оцінювання реалізованого функціоналу з визначеними вимогами ПЗ та виявлення потенційно «слабких» місць або вразливостей в системі.

Даний процес може застосовуватися в будь-якої зручної фазі життєвого циклу для кожного компонента ПЗ.

Метою початкової стадії є встановлення вимог в розумінні функціональних характеристик. Вимоги висловлюють потреби зовнішнього оточення для даної програмної реалізації й повинні бути визначені до початку розробки.

Після визначення функціональних вимог необхідно встановити ключові вимоги безпеки, які зможуть зробити кінцевий продукт більш безпечним та надійним.

Так як існує дуже багато різних платформ та варіантів реалізації компонентів з врахуванням програмної архітектури цієї платформи, третій етап полягає у аналізі та вибору спеціалізованої платформи SHS, А також вибір спеціального способу(підходу) до реалізації ПЗ, які будуть надійно захищені, матимуть широкий функціонал та великі можливості до масштабованості.

Кінцевим результатом є рішення експертів, оціночний компонент може бути відправлений на доопрацювання та повторну оцінку або ж задокументований та затверджений на новий випуск.

На рисунку 3.3 відображена модель процесу основних етапів, необхідних для вибору програмного рішення з урахуванням вимог безпеки для систем «Розумний будинок». Для побудови цієї моделі було використано методологія функціонального моделювання графічного опису процесів – Function Modeling (IDEF0). Особливістю її використання є акцентування на ієрархічне представлення об’єктів, яке дозволяє краще зрозуміти предметну область системи, що розробляється.

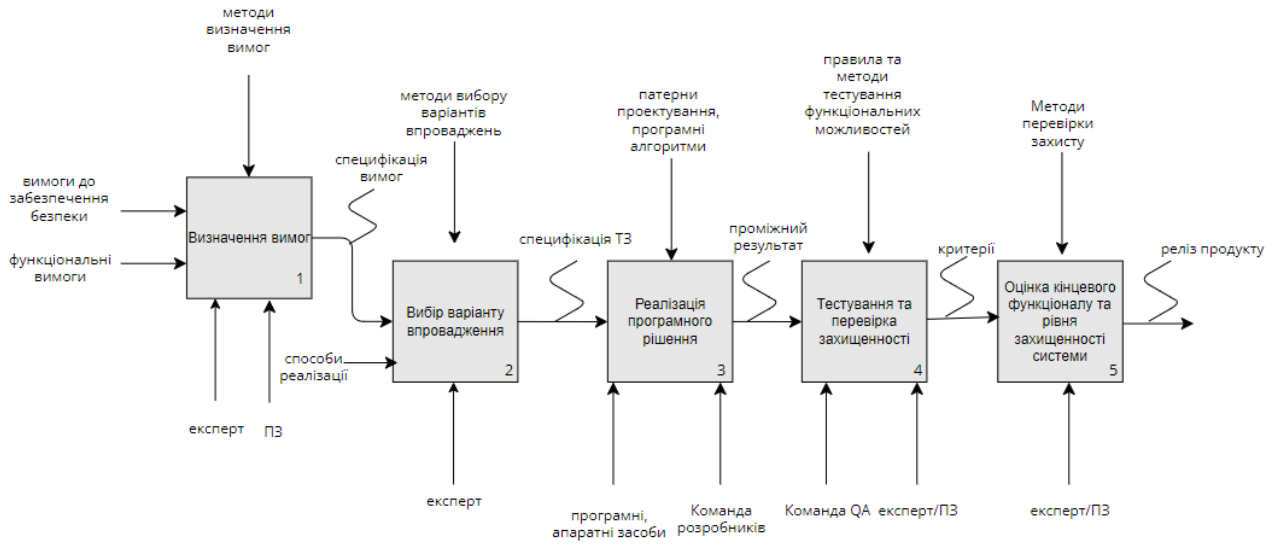

Рис. 3.3 – Модель процесу вибору програмного рішення

4 РОЗРОБКА ПРОТОТИПУ ПІДТРИМКИ ПРОЦЕСУ ВИБОРУ АРХІТЕКТУРИ ДЛЯ ПРОГРАМНИХ РІШЕНЬ «РОЗУМНИЙ ДІМ» З УРАХУВАННЯМ ВИМОГ БЕЗПЕКИ

4.1 Формування тестового набору функціональних вимог та вимог безпеки

4.1.1 Функціональні вимоги та вимоги безпеки для власного розширення

Мета розробки вимог до ПЗ полягає в тому, щоб виявити та задокументувати артефакти вимог для конкретної програми і в той же час повторно використати, наскільки це можливо, артефакти вимог домену [1].

Перед тим як почати створювати програмний компонент потрібно визначити перелік загальних умов щодо вимог кінцевого функціоналу, яке ПЗ повинно забезпечити та врахувати як можна захистити програмний продукт, використовуючи існуючі методи захисту.

Для початку необхідно сформвати загальну ідею ПЗ, яке потрібно імплементувати в рамках системи керування «Розумним будинком». На даний момент часу, є дуже важливим правило щодо дотримання правил безпеки під час повітряних тривог в Україні. Потрібно, щоб система відслідковувала наявність повітряної тривоги по поточному місцеположенню користувача та активувала певні захисні дії, наприклад закриття вікон, щоб убезпечити себе від пошкодження уламків скла.

Під час реалізації даного рішення повинні бути враховані такі заходи безпеки:

- Конфіденційність даних – поточне місцеположення, дані про пристрої, які задіяні у захисному механізмі. Програмне рішення не повинно розповсюджувати ці «чутливі дані» до зовнішніх сервісів. Бажано, щоб всі ці дані були в одній системі, до якої буде встановлено надійний пароль та ця система має змогу застосовувати мультифакторну автентифікацію.
- Захист від несанкціонованого доступу до програмного рішення в рамках SHS. Потрібно встановити певні привілеї для користувачів системи, та

зробити так, щоб лише адміністратор міг змінювати параметри ПЗ, вимикати/вмикати програмний компонент.

- Повинно бути реалізовано автоматичне оновлення програмного рішення, щоб усунути вразливості безпеки, для обходження якої хакери можуть розробляти різного роду експлоїти.
- Чутливі дані програмного рішення повинні бути приховані від перегляду сторонніми персонами або надійно зашифровані.

Звісно ж перелік вимог повинен починатися з врахування заходів захисту, а вже потім реалізація основного функціоналу. Кінцевий вигляд вимог до функціонування програмного компоненту зображений на табл. 4.1.

Таблиця 4.1 - набір функціональних вимог та вимог безпеки

№	Прецендент	Сценарій для прецендету	Додатковий інструментарій
1	Захист від несанкціонованого доступу	Система «Розумний будинок» повинна: надавати змогу для автентифікації поточного користувача; реалізовувати шифрування збережених даних та передачі даних через мережу; надавати певні привілеї користувачам для користування певних функцій	

Продовження таблиці 4.1

2	Аналіз помилок та підозрілих дій в системі	Програмний компонент повинен логувати всі виконані операцій в межах SHS	У системі повинно бути впроваджено функціонал журналювання та аналізу повідомлень. У програмному компоненті використовутись бібліотеки логування, такі як SLF4J
3	Автоматичне оновлення	SHS повинно автоматично оновлювати програмний компонент для виправлення виявлених вразливостей безпеки	
4	Виявлення тривоги та виконання захисних механізмів	Розроблене розширення повинно постійно моніторити наявність повітряних тривог за поточним місцеположенням користувача. У разі настання повітряної тривоги – у системі повинні бути виконані захисні дії(закриття вікон)	API повітряних тривог в Україні

4.1.2 Вимоги щодо реалізації власної інтеграції з метою захисту домашнього серверу

Одним із класичних видів атак спрямованих на методи автентифікації – це brute-force. Класичний метод грубої сили, який перебирає велику кількість комбінацій, що у висновку призведе до злому системи. Основними вимогами є блокування айпі-адреси та встановлення ліміту на кількість спроб. Після трьох невдалих спроб автентифікації SHS повинна заблокувати айпі-адрес зловмисника.

4.2 Стек технологій та інструментарій для реалізації ПЗ

На даний момент часу існує дуже багато різноманітних платформ, які дозволяють автоматизувати різні аспекти життя в будинку. Одними з найбільш популярних платформи, які надають певною мірою широкий функціонал є: Home Assistant та Open Hub, один з яких буде використано у подальшій реалізації та тестуванні. Інші платформи від популярних IT-корпорацій такі як Google Assistant, Apple HomeKit, Amazon Alexa мають менші можливості до автоматизації, працюють лише з певним набором пристроїв, користувачі не мають змоги значно покращити та розширити функціонал свої системи.

Програмна платформа Home Assistant є клієнт-серверною службою, яка призначена для керування, моніторингу та контролю всього інтелектуального обладнання в житловій мережі, навіть у випадку, якщо це обладнання постачається різними виробниками. Платформа HA є безкоштовною глобальною програмою з відкритим вихідним кодом, до якої долучилися досвідчені користувачі та захоплені розробники автоматизації з усього світу з метою об'єднання всього інтелектуального обладнання в одній екосистемі [23]. До переваг цієї платформи можна віднести наступні аспекти:

- має приємний користувацький інтерфейс та багато способів конфігурації;
- відкритий вихідний код та велика спільнота користувачів, що полегшує процес виявлення недоліків, розробці нових функціональних можливостей та підтримки платформи;

- високий рівень безпеки. Використання різних методів захисту, які запобігають різним типам атак, таких як: SQL-ін’єкції, несанкціонований доступ до конфігураційних параметрів, перехоплення сеансів.
- велика кількість підтримуваних пристроїв та можливість додавати власні (розширення, інтеграції).

Для реалізації поставлених вимог доцільно розробити додаток, який зможе виконувати запити до зовнішнього API, обробляти їх, і в залежності від результатів обробки відправляти запити до ОС Home Assistant за допомогою внутрішнього проксі та/або вебсокетів. Для цих цілей доцільно використовувати кросплатформну, швидку та багатопотокову мову програмування з підтримкою великої кількості інструментів для відправлення та обробки веб-запитів. В області домашньої автоматизації та розробки серверної частини(backend) веб-додатку найбільш популярними та зручними є JavaScript(Node.js) та Java(Spring boot).

Spring Boot – це фреймворк Java з відкритим вихідним кодом, який використовується для надання серверних веб-сервісів клієнтським або серверним технічним програмам з можливістю впровадження великої кількості модулів з екосистеми Spring [20].

Node.js - це крос-платформне середовище виконання JavaScript з відкритим кодом, яке може працювати в Windows, Linux, Unix, macOS тощо. Node.js працює на механізмі JavaScript V8 і виконує код JavaScript поза веб-браузером [21].

Аналіз ключових аспектів фреймворків Spring Boot та Node.js зображений у табл. 4.2.

Таблиця 4.2 – важливі аспекти серверної частини та функціональні можливості між двома фреймворками.

Аспект	**Spring Boot**	**Node.js**
Архітектура	Трьохрівнева архітектура, яка складається з рівня презентації, сервісу та доступу до даних. Рівні взаємодіють один з одним в ієрархічному порядку, щоб забезпечити підтримку розробки серверної частини та мікросервісів.	Неблокуюча архітектура виведення, яка використовує цикл подій і асинхронні операції для обробки операцій введення та виведення.
Продуктивність	Дотримується тонкої матриці продуктивності і найкраще працює для програм корпоративного рівня, які вимагають надійної масштабованості та зручності обслуговування.	Працює з високопродуктивною матрицею та ефективною швидкістю для програм реального часу.
Мультизадачність	Оскільки Java є багатопоточною мовою програмування, Spring boot забезпечує потужну підтримку складних завдань і багатопроцесорних вимог.	Працює в одному процесі без створення нового потоку для кожного запиту. Node.js надає набір асинхронних примітивів вводу/виводу у своїй стандартній бібліотеці, які запобігають блокуванню коду [21].

Продовження таблиці 4.2

Масштабованість	Дозволяє значно розширювати програмний продукт з використанням мікросервісної архітектури та великої кількості програмних компонентів в екосистемі Spring.	Дозволяє одночасно надсилати велику кількість запитів без блокування сервера.
Безпека	Має потужну і автентичну систему безпеки та контролю доступу, яка вже визначена та налаштована в його 3-рівневій архітектурі.	Node.js є безпечним, але оскільки використовуються сторонні бібліотеки з відкритим кодом через свою систему керування пакетами (npm), вона вразлива до кібератак

Як можна побачити з таблиці 4.2, що обидва фреймоворки є багатофункціональними та надійними для розробки ПЗ для IoT-системи, проте кожен з них має свої особливості, на які варто звертати увагу під час вибору програмно-технічних засобів. На табл. 4.3 можна побачити плюси та мінуси використання фреймворку для вирішення поставленої задачі.

Таблиця 4.3 – плюси та мінуси Spring Boot та Node.js в контексті реалізації поставлених вимог

Аспект	**Spring Boot**	**Node.js**
Продуктивність	+/-	+
Мультизадачність	+	+
Масштабованість	+	-
Безпека	+	-

Отже, для виконання поставлених вимог, забезпечення високого рівня безпеки та подальшої можливості до масштабування більш доцільно використовувати Spring Boot.

4.3 Застосування процедури для підтримки процесу вибору архітектури для програмних рішень «Розумний дім» з урахуванням вимог безпеки на тестовому прикладі

Для реалізації програмного прототипу було розроблено системну архітектуру, яка виглядає наступним чином:

- Веб-додаток для керування програмними рішеннями «Розумного будинку» Home Assistant;
- ОС Home Assistant встановлена на одноплатний комп'ютер Raspberry PI 3 для керування пристроями IoT;
- SwitchBot Curtain Rod 2 інтегрований пристрій у HA, який дозволяє керувати станом закриття вікон.

На рисунку 4.1 зображена Use Case діаграми можливостей використанням системи для автенти та непривілейованого користувача.

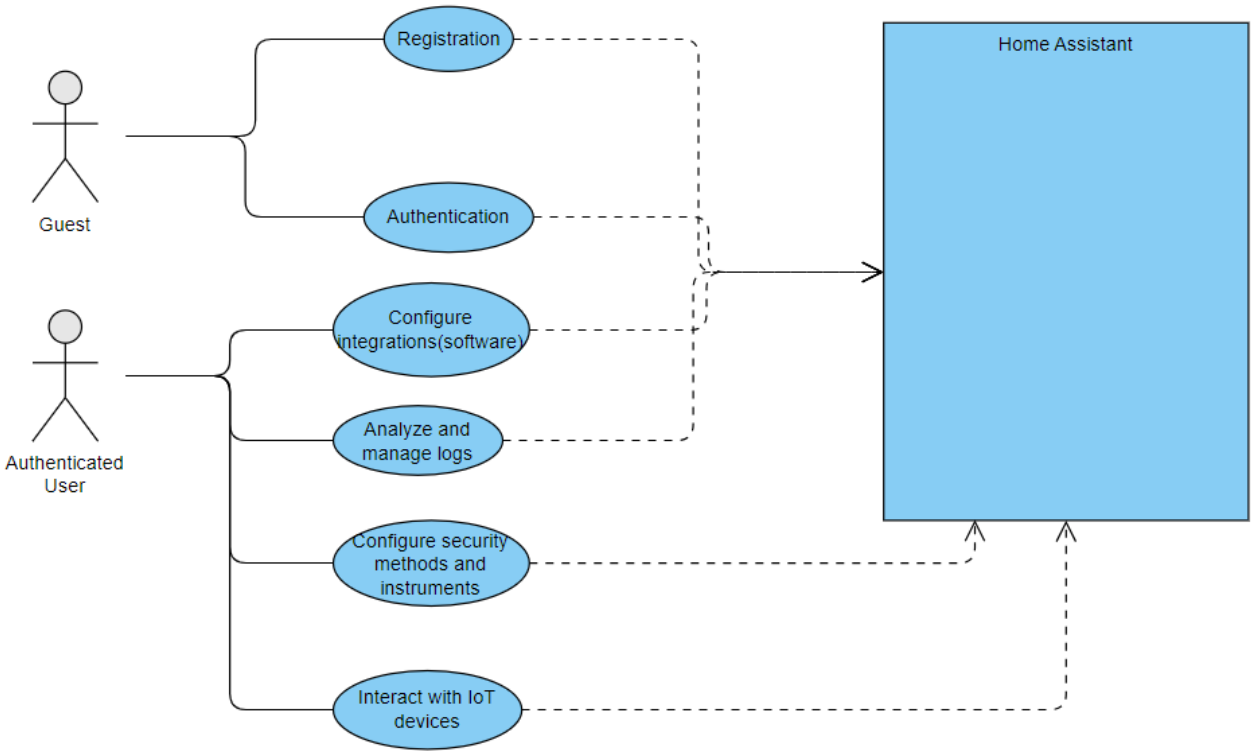

Рис. 4.1 – Діаграма можливостей використання системи Home Assistant

В НА можливо створювати(а також публічно поширювати для інших користувачів) три види програмних компонентів, це – розширення, інтеграції та компоненти візуалізації.

a) **Розширення (Add-ons):** Home Assistant має систему розширень, які дозволяють додавати додаткові функції та сервіси безпосередньо до системи. Наприклад, ви можете додати розширення для резервного копіювання, моніторингу мережі, віддаленого доступу тощо.
b) **Інтеграції (Integrations):** Home Assistant має широкий спектр інтеграцій з різними пристроями та сервісами. Ці інтеграції дозволяють підключати до системи різноманітні смарт-пристрої, такі як розумне освітлення, термостати, камери спостереження, медіа-програвачі та багато іншого.
c) **Компоненти візуалізації (Visualization Components):** Home Assistant має різноманітні компоненти візуалізації, які дозволяють створювати графіки, діаграми та інші візуальні елементи для відображення даних. Наприклад, використовувати компоненти Lovelace для створення інтерактивних інтерфейсів з власними картами, кнопками, сенсорами тощо.

На рисунку 4.2 зображено діаграму компонентів, які надають можливість розширення функціональних можливостей SHS.

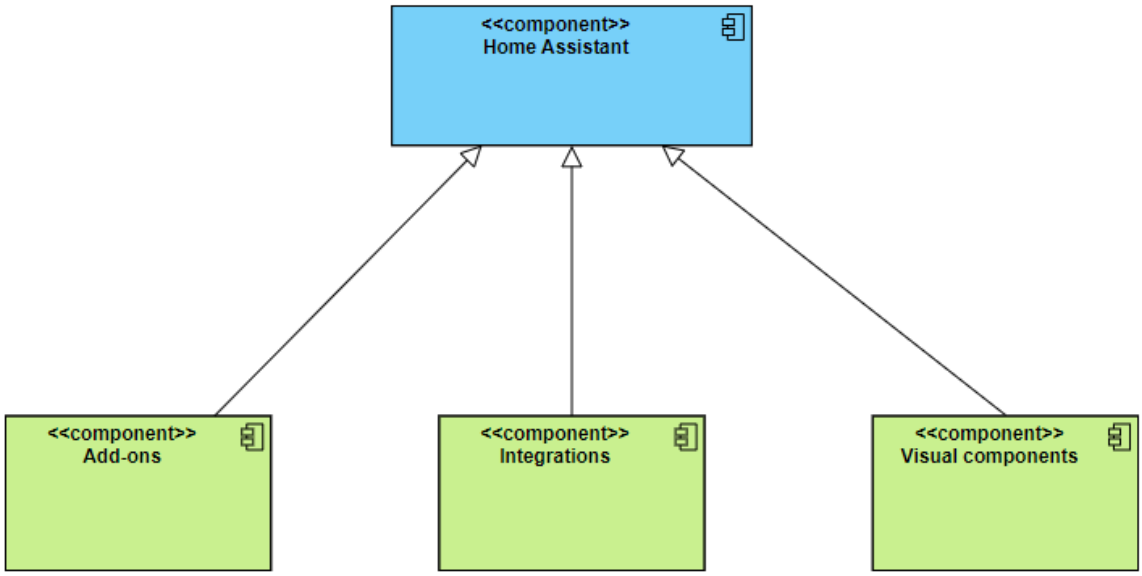

Рис. 4.2 – Програмні компоненти, які надають можливість розширення SHS

Розширення – це окреме ПЗ, яке необхідно «додати» у файлову систему HA. Інтеграція вже підтримується ядром HA, слід тільки підтримуватись певних правил(обов'язкові файлі, текова структура) і можна реалізувати свою інтеграцію. Компоненти візуалізації розробляються та додаються напряму в HA на основній сторінці користувацького інтерфейсу.

Так як архітектура HA побудована з використанням Docker [2], розширення це як окремий Docker-контейнер, який можна інтегрувати до загального ПЗ. Розширення можуть бути реалізовані за допомогою будь-якої мови програмування будь то Java, Ruby, C++ не має різниці. В той же час, для створення інтеграції можливо використовувати лише мову Python, та й саму логіку інтеграції не так сильно можливо кудись застосувати, окрім якого специфічного пристрою.

Реалізація доповнення - отже, для вирішення задачі з виконанням захисних дій під час повітряної тривоги потрібно реалізувати власне розширення, запустити та моніторити її активність.

Щоб розпочати розробку доповнень, нам спочатку потрібен доступ до місця, де Home Assistant шукає локальні додатки. Для цього слід використовувати доповнення Samba або SSH.

Щойно запустите Samba, екземпляр Home Assistant з’явиться на вкладці локальної мережі та надасть спільний доступ до папки під назвою «addons». Це папка для зберігання ваших власних додатків.

Власно кажучи, розробляти розширення можна у будь-якій зручній середовищі розробки, в моєму випадку буду використовувати середу IntellijIdea Premium Edition, можливістю користуванням якої надав університет.

На рисунку 4.3 зображена файлова структура розробленого розширення.

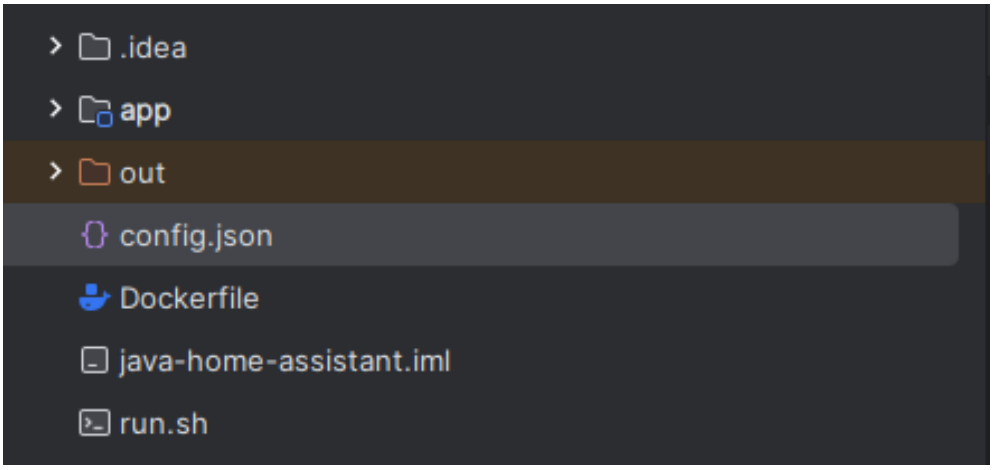

Рис. 4.3 – Файлова структура розширення в середовищі розробки

У власному розширенні обов’язково повинно бути три типи файлів: Dockerfile це зображення(image), яке буде використано для створення доповнення; config.json - це конфігурація розширення, яка вказує Супервайзеру, що робити та як представити власний додаток; run.sh - це сценарій, який запускатиметься під час запуску розширення. Ключовим компонентом серед цих трьох є якраз Dockerfile. За допомогою нього можна вказувати всі необхідні залежності, бібліотеки, визначати де буде точка входу ПЗ, виконувати збірку проекту та запускати інтеграційні тести та багато іншого. На рисунку 4.4 представлений код конфігурації.

```
ARG BUILD_FROM=homeassistant/amd64-base:latest
FROM $BUILD_FROM

ENV LANG C.UTF-8

RUN apk add maven
RUN apk add openjdk17

WORKDIR /app
COPY . /app
COPY /app/pom.xml .
COPY /app/src/ ./src/
RUN mvn dependency:go-offline
RUN mvn clean package -DskipTests=true
COPY /app/target/app-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["sh", "/app/run.sh"]

LABEL io.hass.version="VERSION" io.hass.type="addon" io.hass.arch="armhf|aarch64|i386|amd64"
```

Рис. 4.4 – Вміст Dockerfile розширення

Після того як всі базові та обов'язкові файли були додані, тепер необхідно безпосередньо реалізувати ПЗ з належним до нього функціоналом. В якості мови програмування я обрав Java, допоміжним фреймворком скористався Spring Boot, система збірки – Maven.

Алгоритм роботи даного програмного компоненту полягає в тому, що відбувається з певною періодичністю моніторинг наявності повітряної тривоги в заданій області(за допомогою зовнішнього API). Якщо повітряної тривоги немає то застосунок далі продовжує моніторити її наявність, якщо ж навпаки застосунок виконає запит до серверу HA, за допомогою спеціального токену достучиться до спеціального пристрою та виконає дію захисту вікон. Вся дія застосунку супроводжується логуванням. На рисунку 4.5 зображено створене розширення після інсталяції в систему Home Assistant.

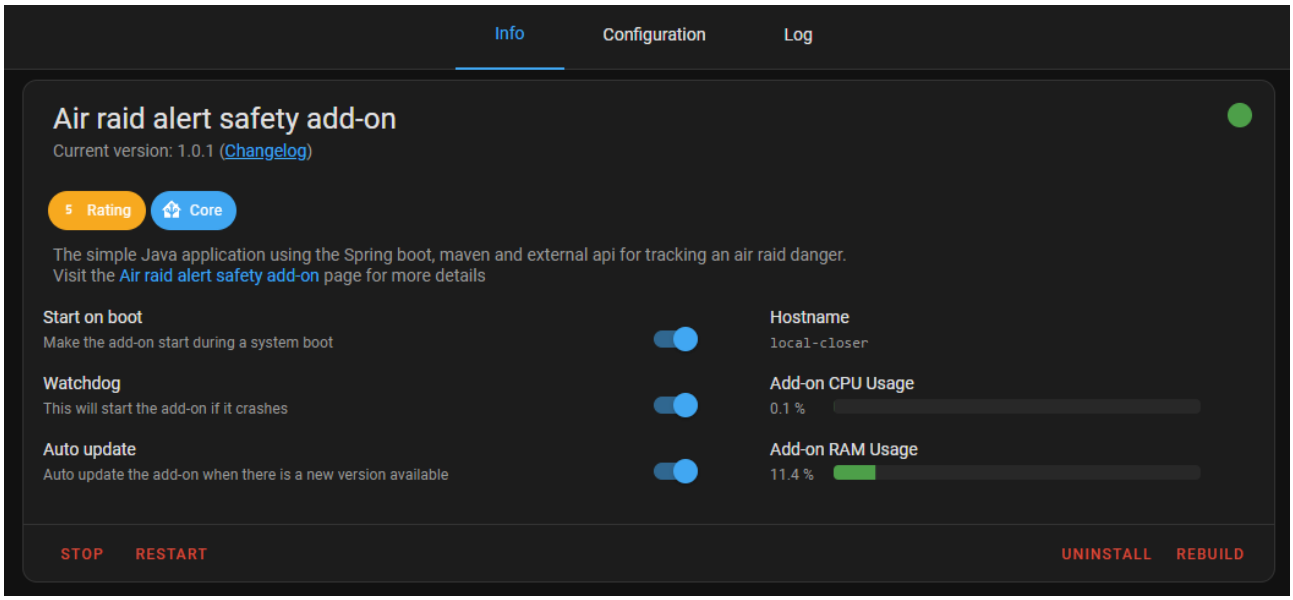

Рис. 4.5 – Вигляд розробленого розширення в системі HA

Реалізація інтеграції - навідміну від розширень інтеграції розробляються всередині власного інстансу «Розумного будинку» і вони майже не потребують зусиль в конфігурації. Єдине, що потрібно це встановити редактор коду з доступних розширень та почати писати код. Я встановив розширення “File editor” для створення та керуваннями файлами коду HA. На рисунку 4.6 зображено інтерфейс платформи Home Assistant з встановленим розширенням «File editor».

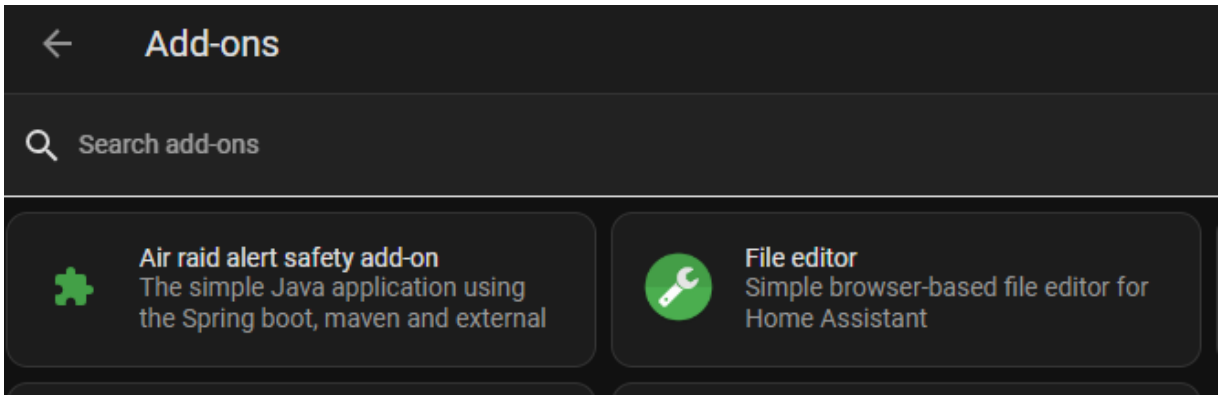

Рис. 4.6 – Скачане доповнення «File editor» в HA

Далі необхідно обрати файл, в який будемо вносити зміни та реалізувати власну інтеграцію. На рисунку 4.7 зображено як виглядає редактор коду на інтерфейсі користувача.

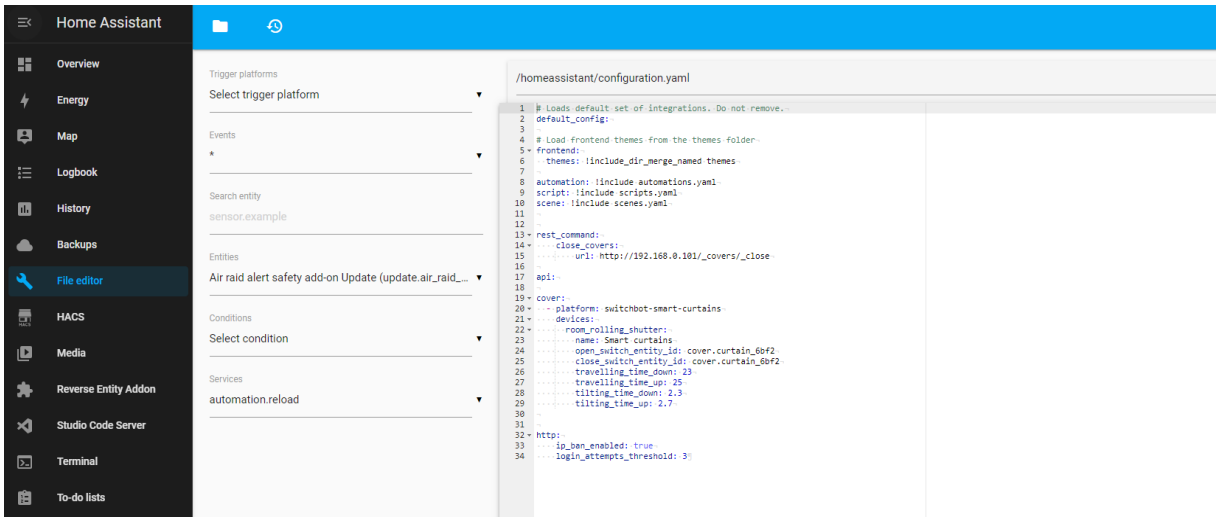

Рис. 4.7 – Вікно “File editor” на інтерфейсі

Для того, щоб вказати «домашнього помічнику», щоб необхідно було блокувати айпі-адресу після n-кількості невдалих спроб необхідно прописати наступні параметри:

http:

ip_ban_enabled: true

login_attempts_threshold: 3

Дозволяємо блокування айпі-адреси після невдалої кількості спроб, та зазначили кількість спроб, яка дорівнює трьом.

На рисунках 4.8 та 4.9 зображено інтерфейс сторінки авторизації Home Assistant з блокуванням айпі-адреси після декількох невдалих спроб.

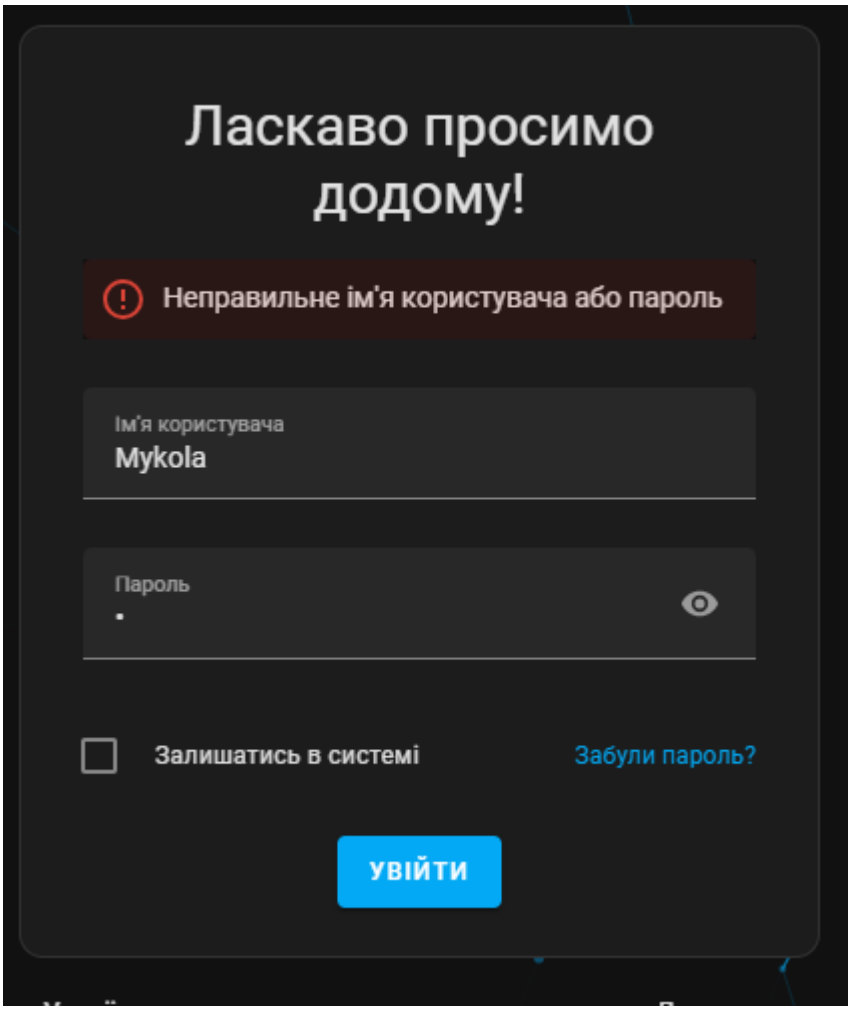

Рис. 4.8 – Панель авторизації користувача Home Assistant

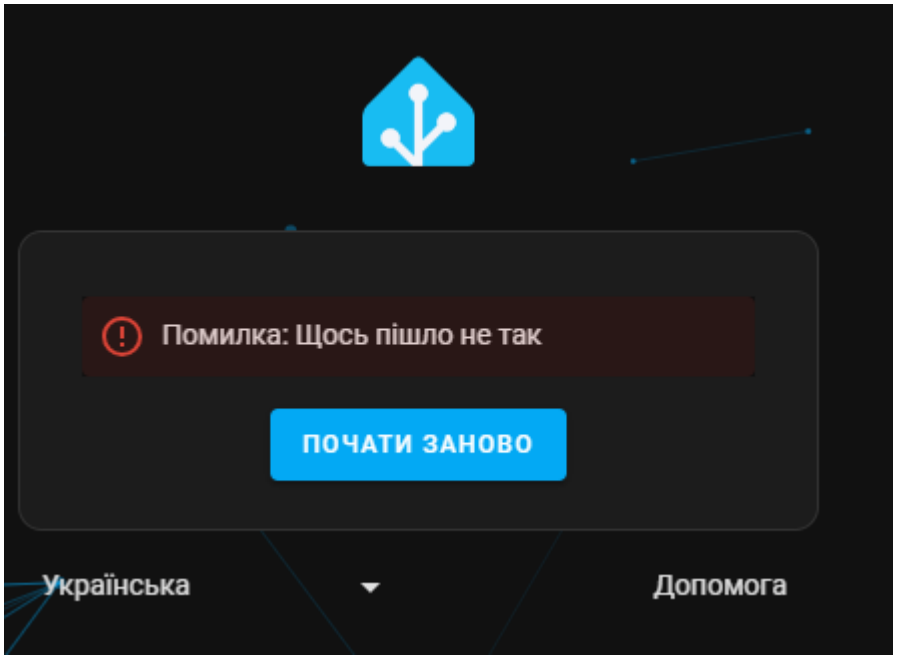

Рис. 4.9 – Результат блокування після невдалих спроб

Після трьох невдалих спроб авторизуватись в системі НА мою айпі адресу до списку заблокованих і всі наступні спроби відкрити дану платформу повертаються зі статусом “403 Fobidden”. Щоб знову отримати доступ до системи необхідно прибрати айпі-адресу з файлу ip_bans.yaml.

4.4 Оцінка ефективності застосування запропонованої процедури для підтримки процесу вибору програмних компонентів для системи «Розумний дім»

Для того, щоб оцінити ефективність застосованої процедури необхідно її порівняти на практичному прикладі з іншими варіантами розробки програмних компонентів для SHS. Необхідно також оцінити варіанти розробки з точки зору безпеки. Нижче запропоновано власні математичні моделі для визначення рівня безпеки для кожного варіанту програмного рішення.

Формула розрахунку кількісної оцінки захищенності програмного рішення для SHS (S_{SHS}) (формула 4.1):

$$S_{SHS} = \frac{\sum_{i=0}^{n} p_i * k_i}{n} \rightarrow max \quad (4.1)$$

де p – параметр безпеки,

k – коефіцієнт важливості відповідного параметру,

n – кількість параметрів

Якщо у кінцевому результаті оцінка буде менше ніж у два бали, то необхідно передивитися реалізацію програмного рішення, так як воно може зробити всю систему «Розумного будинку» більш вразливою до потенційних атак.

На основі проведених досліджень[22], для розширень було запропоновано власні значення параметрів безпеки з їх коефіцієнтами:

- дозвіл користуванням серверної частини НА – auth_api: true. Значення – 2, коефіцієнт – 4.
- обмежений дозвіл керуванням системою – full_access: false. Значення – 3, коефіцієнт – 5.
- увімкнення модулю захисту AppArmor – apparmor – true. Значення – 1, коефіцієнт – 2.

Для інтеграцій запропоновано параметри [25] та визначені власні значення та коефіцієнти:

- обробка прострочених облікових даних. Значення – 4, коефіцієнт – 5.
- чітко визначені класи пристроїв. Значення – 2, коефіцієнт – 2.
- наявність унікального айді в системі. Значення – 2, коефіцієнт – 3.

В таблиці 4.4 можна побачити вагу кожного параметру та кінцевий результат оцінки захищенності програмного компоненту.

Таблиця 4.4 – результати оцінювання безпеки програмних компонентів

Програмне рішення	**Оцінка захищенності**
Розширення	$S_{SHS} = \frac{(2*4+3*5+1*2)}{3} = 8.33$
Інтеграція	$S_{SHS} = \frac{(4*5+2*2+2*3)}{3} = 10$

На рисунку 4.10 зображена порівняльна діаграма результатів захищенності розроблених програмних компонентів для SHS.

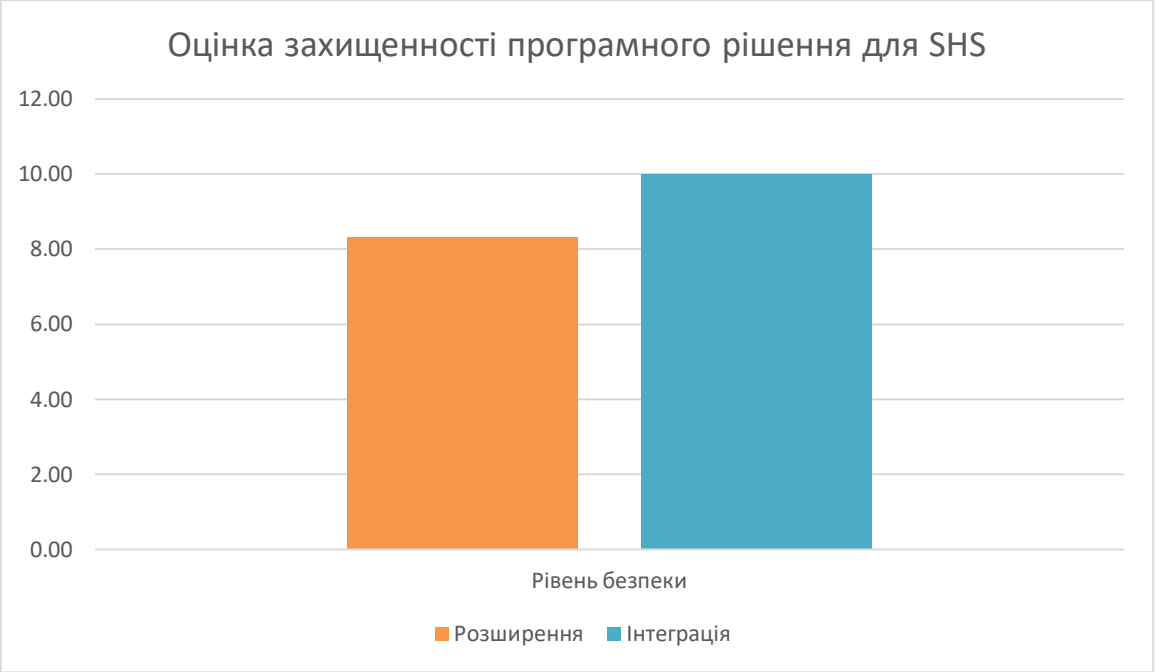

Рис 4.10 – Порівняння захищенності двох програмних рішень для SHS

Отже, підсумовуючи даний розділ, що розробка розширень потенційно може внести певні вразливості у загальний функціонал SHS, за ними потрібен постійний аудит та моніторинг активностей, але вони предоставляють ширший функціонал, який може виходити за рамки класичного функціоналу пристроїв IoT та впроваджувати більш складну логіку, наприклад як інтеграція по API зі сторонніми сервісами. В той же час Інтеграції є більш вузьконапрямленим компонентом, яким повністю керує ядро HA, через це воно є захищеним та має невелику кількість факторів, які можуть зашкодити безпеці SHS.

ВИСНОВКИ

Способи розробки програмного забезпечення та основні методи та вимог безпеки – було розглянуто під час проектування та розробки процедури вирішення проблем даної задачі. У ході виконання кваліфікаційної роботи проведений аналіз та розглянуто заходи щодо забезпечення надійності безпеки систем «Розумний будинок». Розглянута архітектура враховуючи апартні та програмні властивості IoT-системи. Для досягнення мети проведено аналіз, проектування, програмування, а також зібрано отримані результати.

Дослідження в рамках даної дипломної роботи розкриває важливі аспекти розробки програмного забезпечення для систем «Розумного будинку», зокрема зосереджуючись на питаннях безпеки. На сьогоднішній день системи «Розумного будинку» стають все більш популярними, проте їх безпека залишається однією з найважливіших проблем.

Важливо відзначити, що безпека програмного забезпечення для систем «Розумного будинку» вимагає комплексного підходу. Це означає, що необхідно поєднувати різні методи та моделі розробки, включаючи вимоги безпеки на різних етапах розробки, від планування до тестування та впровадження.

У даній роботі реалізоване комплексне рішення щодо вирішення питань:

1. Проаналізовано та розглянуто архітектуру IoT систем
2. Запропоновано SDLC-моделі для розробки апаратного та програмного забезпечення для систем «Розумного будинку».
3. Визначено основні типи атак та заходи безпеки, щоб запобігти різного роду вразливостям IoT-пристроїв.
4. Розроблено декілька варіантів власного програмного забезпечення, яке взаємодіє з системою керування «Розумного будинку», враховуючи функціональні вимоги та вимоги безпеки.

Реалізовані програмні рішення дозволяють підвищити рівень безпеки власного інстансу «Розумного будинку, гнучко розширити функціонал та роботу системи, впровадити нові механізми захисту.

У подальшому планується провести консолідований аналіз в області безпеки апаратних засобів IoT, створенні власного програмного-апаратного забезпечення, яке спрямоване на кожний з рівнів архітектури систем «Розумний будинок», вивчити більш детально механізми захисту бездротових локальних мереж, розширити захист та використання системи «Розумного будинку» поза межами власної мережі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Товстокоренко О.Ю. «МОДЕЛІ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РОЗРОБКИ ТА СУПРОВОДУ ВАРІАБЕЛЬНИХ КОМПОНЕНТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМ «РОЗУМНИЙ БУДИНОК»: дис. доктора комп'ютерних наук – с. 29-40.
2. Integration Architecture | Home Assistant Developer Docs. Home Assistant Developer Docs | Home Assistant Developer Docs. URL: https://developers.home-assistant.io/docs/architecture_components (дата звернення: 11.03.2024).
3. Ismail S., Dawoud D. W. Software Development Models for IoT. 2022. URL: https://www.researchgate.net/publication/359035044_Software_Development_Models_for_IoT.
4. Top 8 Software Development Life Cycle (SDLC) Models used in Industry - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/top-8-software-development-models-used-in-industry/ (дата звернення: 14.03.2024).
5. The 5 Layers of the IoT Technology Stack - Daniel Elizalde. Daniel Elizalde. URL: https://danielelizalde.com/iot-primer/ (дата звернення: 17.03.2024).
6. Shodan (сайт) – Вікіпедія. Вікіпедія. URL: https://ru.wikipedia.org/wiki/Shodan (дата звернення: 16.03.2024).
7. IoT devices in Smart Home URL: https://parametric-architecture.com/smart-homes-and-iot-how-technology-is-revolutionizing-architecture/ (дата звернення: 17.03.2024).
8. Domb M. Smart Home Systems Based on Internet of Things. IoT and Smart Home Automation. 2019. URL: https://doi.org/10.5772/intechopen.84894 (дата звернення: 18.03.2024).
9. 9 Important Security Requirements to Consider for IoT Systems - IoTAC. IoTAC. URL: https://iotac.eu/9-important-security-requirements-to-consider-for-iot-systems/ (дата звернення: 20.03.2024).
10. Aaesha Aldahmani,Bassem Ouni,Thierry Lestable,mérouane Debbah. Cyber-Security of Embedded IoTs in Smart Homes: Challenges, Requirements,

Countermeasures, and Trends. 2023. URL: https://www.researchgate.net/publication/366869292_Cyber-security_of_embedded_IoTs_in_smart_homes_challenges_requirements_countermeasures_and_trends

11. Conceptual model for IoT 2023. URL: https://www.researchgate.net/figure/Conceptual-model-for-IoT_fig1_322271635
12. K. Ghirardello, C. Maple, D. Ng, and P. Kearney, “Cyber security of smart homes: Development of a reference architecturefor attack surface analysis,” in Living in the Internet of Things:Cybersecurity of the IoT-2018. IET, 2018, с. 1–10.
13. D. Mocrii, Y. Chen, and P. Musilek, “Iot-based smart homes:A review of system architecture, software, communications,privacy and security,” Internet of Things, с. 81–98, 2018.
14. P. Siddhanti, P. M. Asprion, and B. Schneider, “Cybersecurity by design for smart home environments.” in ICEIS (1), 2019, с. 587–595
15. P. Siddhanti, P. M. Asprion, and B. Schneider, “Cybersecurity E. Zeng, S. Mare, and F. Roesner, “End usersecurity and privacy concerns with smart homes,”in Thirteenth Symposium on Usable Privacy andSecurity (SOUPS 2017). Santa Clara, CA: USENIX Association, 2017, с. 65–80. URL: https://www.usenix.org/conference/soups2017/technicalsessions/presentation/zeng
16. H. Kim and J. Song, “Analysis of iot security in wi-fi 6,”Journal of the Institute of Convergence Signal Processing, с. 38–44, 2021
17. N. Vlajic and D. Zhou, “Iot as a land of opportunity for ddoshackers,” Computer, с. 26–34, 2018.
18. Securing Your Home Assistant Instance. Home Assistant Community. URL: https://community.home-assistant.io/t/securing-your-home-assistant-instance/336908 (дата звернення: 13.04.2024).
19. (2022). Residential security through the Home Assistant Platform. MATEC Web of Conferences. 354. 00008. 10.1051/matecconf/202235400008.
20. Spring Boot. URL: https://spring.io/projects/spring-boot (дата звернення: 30.04.2024).

21. Node.js – Introduction to Node.js. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (дата звернення: 30.04.2024).
22. Add-ons | Home Assistant Developer Docs. Home Assistant Developer Docs | Home Assistant Developer Docs. URL: https://developers.home-assistant.io/docs/add-ons/presentation (дата звернення: 03.05.2024).
23. IoT System Development Methods. Bilkent Repository :: Home. URL: https://repository.bilkent.edu.tr/server/api/core/bitstreams/1a579fe5-5200-4606-894a-9373b4c161be/content (дата звернення: 13.05.2024)
24. Scrum development steps involved in scrum model software development life cycle. PowerPoint Templates | PowerPoint Slides Templates| PPT Themes Presentation. URL: https://www.slideteam.net/scrum-development-steps-involved-in-scrum-model-software-development-life-cycle.html#images-1 (дата звернення: 13.05.2024).
25. Integration Quality Scale | Home Assistant Developer Docs. Home Assistant Developer Docs | Home Assistant Developer Docs. URL: https://developers.home-assistant.io/docs/integration_quality_scale_index (дата звернення: 15.05.2024).

ДОДАТОК А

Програмна реалізація компонентів системи

1. Вихідний код реалізованого розширення для забезпечення захисту SHS під час повітряної тривоги

```
#!/bin/bash

BASEDIR=$(dirname $0)
echo "Script location: ${BASEDIR}"
java -jar app.jar
```

Лістинг А.1 – Код shell-скрипту

```
{
  "name": "Air raid alert safety add-on",
  "description": "The simple Java application using the Spring boot, maven and external api for tracking an air raid danger",
  "version": "1.0.1",
  "slug": "closer",
  "url": "https://github.com/kmaznitsyn/HA-air-raid-alert-open-close-entity",
  "init": false,
  "arch": [
    "aarch64",
    "amd64",
    "armhf",
    "armv7",
    "i386"
  ],
  "startup": "application",
  "boot": "auto",
```

```
  "homeassistant_api": true
}
```

Лістинг А.2 – Файл конфігурації розширення

```
ARG BUILD_FROM=homeassistant/amd64-base:latest
FROM $BUILD_FROM

ENV LANG C.UTF-8

RUN apk add maven
RUN apk add openjdk17

WORKDIR /app
COPY . /app
COPY /app/pom.xml .
COPY /app/src/ ./src/
RUN mvn dependency:go-offline
RUN mvn clean package -DskipTests=true
COPY /app/target/app-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["sh", "/app/run.sh"]

LABEL io.hass.version="VERSION" io.hass.type="addon"
io.hass.arch="armhf|aarch64|i386|amd64"
```

Лістинг А.3 – Скрипт налаштування Dockerfile

```
#value of ukrainian region in external api
region.id=10

#######################################
# FILL BELOW TOKENS AND IDS FOR CORRECT USAGE
```

```
#####################################
#token for accessing air raid alert api
air-raid-alert.api.token=******************

#webhook id of your close automation trigger
ha.webhook.covers.close.id=********************

#webhook id of your open automation trigger
ha.webhook.covers.open.id=**************************
#url of your home assistant server
ha.url=http://192.168.0.101:8123


#boolean value for indicating strict mode for air raid alert in villages and regional centers
app.settings.useStrictMode=false


#bearer token for HA server authentication
ha.rest.bearer-token=************************
```

Лістинг А.4 – Конфігураційний-проперті файл серверної частини

```
package com.example.app;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
```

```
@SpringBootApplication
public class AppApplication {

    private static HAScheduler scheduler;

    @Autowired
    public AppApplication(HAScheduler scheduler) {
        AppApplication.scheduler = scheduler;
    }

    public static void main(String[] args) {
        SpringApplication.run(AppApplication.class, args);
        ScheduledExecutorService executor = Executors.newScheduledThreadPool(1);
        executor.scheduleAtFixedRate(scheduler, 0, 10, TimeUnit.SECONDS);
    }

}
```

Лістинг А.5 – Точка входу програми

```
package com.example.app;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/**
 * Home assistant scheduler that will run periodically and send requests via {@link HARestClient}
 */
@Slf4j
@Component
public class HAScheduler implements Runnable{

    @Value("#{new Boolean('${app.settings.useStrictMode}')}")
    private boolean useStrictMode;
```

```
    @Autowired
    private HARestClient haRestClient;


    @Override
    public void run() {
        String response = haRestClient.getAirRaidAlertState().getBody();
        if (response == null) {
            log.error("Something went wrong while getting response about air-raid alerts");
        } else {
            haRestClient.manageWebhookCall(isAlert(response));
        }
    }

    private boolean isAlert(String code) {
        return code.equals("A") || (useStrictMode && code.equals("P"));
    }
}
```

Лістинг А.6 – Шедулер, який виконує моніторинг з певною періодичністю

```
package com.example.app;

import com.example.app.dto.StateResponseDto;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.configurationprocessor.json.JSONObject;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.RequestEntity;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
```

```
import org.springframework.web.client.RestTemplate;

import java.net.URI;

@Slf4j
@Component
public class HARestClient {

    @Value("#{new Integer('${region.id}')}")
    private Long regionId;

    @Value(value = "${ha.url}")
    private String haUrl;

    @Value(value = "${ha.webhook.covers.open.id}")
    private String webhookOpenId;

    @Value(value = "${ha.webhook.covers.close.id}")
    private String webhookCloseId;

    @Value(value = "${air-raid-alert.api.token}")
    private String airRaidAlertTokenApi;

    @Value(value = "${ha.rest.bearer-token}")
    private String haBearerToken;

    private final RestTemplate restTemplate = new RestTemplate();

    /**
     * Method makes calls for rest api of air raid alert state.
```

```
     * @return response body with specific code (
     * A - повітряна тривога активна в усій області
     * P - часткова тривога в районах чи громадах
     * N - немає інформації про повітряну тривогу)
     * or else error response
     */
    public ResponseEntity<String> getAirRaidAlertState() {
        try {
            String response = restTemplate.getForObject(constructUrl4AirRaidAlertApi(),
String.class);
            log.info("Success got air raid alert state");
            return new ResponseEntity<>(response, HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(HttpStatus.BAD_GATEWAY);
        }
    }

    public void manageWebhookCall(boolean isAlert) {
        if (isAlert) {
            StateResponseDto responseDto = getStateOfEntityId("cover.curtain_6bf2");
            if (responseDto.getState() != null &&
responseDto.getState().equals("opened")) {

restTemplate.postForEntity(constructUrl4WebhookHAServer(HttpMethod.POST,
false), "null", String.class);
                log.info("Webhook call was executed with success");
            }
        }
    }
```

```
    @SneakyThrows
    public StateResponseDto getStateOfEntityId(String entityId) {
        HttpHeaders headers = new HttpHeaders();
        headers.setBearerAuth(haBearerToken);
        ResponseEntity<StateResponseDto> response = restTemplate.exchange(RequestEntity.get(new URI(haUrl + "/api/states/" + entityId)).headers(headers).build(), StateResponseDto.class);
        if (response.getStatusCode().is4xxClientError()) {
            log.error("HA server is not responding or wrong entityId is provided");
        }
        return response.getBody();
    }


    private String constructUrl4WebhookHAServer(HttpMethod method, boolean open) {
        return haUrl + "/api/webhook/" + (open ? webhookOpenId : webhookCloseId);
    }

    private String constructUrl4AirRaidAlertApi() {
        return "https://api.alerts.in.ua/v1/iot/active_air_raid_alerts/" + regionId + ".json?token=" + airRaidAlertTokenApi;
    }
}
```

Лістинг А.7 – Rest-клієнт, який виконує запити до зовнішнього АПІ повітряних тривог та Home Assistant