МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: **Захищене управління розумним будинком з використанням Java**

Здобувача вищої освіти другого (магістерського) рівня,
II курсу, групи КМІ-24-2, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»

(підпис) Лук'яненко Сергія Васильовича
(прізвище, ім'я, по батькові)

Науковий керівник (підпис) Погорєловська Ірина Дем'янівна, канд. екон. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми (підпис) Омельчук Антон Анатолійович, канд. техн. наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «02» січня 2026 № 10

Завідувач кафедри (підпис) Філоненко Михайло Миколайович, канд. фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь - 2026

Реєстрація № 21 21.01.2026 _____
(дата) (підпис відповідальної особи)

Результати перевірки до захисту 02.01.2026 _____
(до захисту, на доопрацювання) (дата) (підпис)

Результат захисту 5 (відмінно) 05.02.2026 _____
(оцінка) (дата) (підпис)

Голова комісії: _____ О.С. Дюбанов
(підпис)

Члени комісії: _____ А.Ю. Горбовий
(підпис)

_____ М.М. Філоненко
(підпис)

_____ І.Д. Погорєловська
(підпис)

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки вбалах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		05.02.2026 Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	9		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	47		

Оцінювання кваліфікаційної роботи науковим керівником – 48 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100- бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
95	A	5	05.02.2026	Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	F X	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«11» грудня 2024

Завдання на кваліфікаційну роботу
Лук'яненко Сергію Васильовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Захищене управління розумним будинком з використанням Java
Затверджена наказом ректора від «10» грудня 2024 № 1852
2. Строк здачі закінченої роботи «02» січня 2026
3. Мета роботи Розробка та реалізація архітектури, а також ключових модулів захищеної інформаційної системи управління пристроями розумного будинку на базі мови програмування Java.

Об'єкт дослідження Процеси керування, моніторингу та інформаційного обміну в системі "Розумний будинок".

Предмет дослідження Методи, протоколи та програмні засоби кібербезпеки, реалізовані за допомогою.

Орієнтовна база дослідження IT - інфраструктура та інформаційні масиви, що симулюють пристрої "розумного будинку".
4. Вихідні дані до роботи Нормативні документи та стандарти безпеки, документація Java, технічні специфікації протоколів MQTTS/TLS, статистичні дані щодо загроз у сфері IoT.
5. Перелік графічного матеріалу Схема архітектури захищеного розумного будинку. Схема взаємодії компонентів з використанням TLS/MQTTS. Скріншоти розробленого програмного забезпечення.
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом): Розділ 1. Теоретико-методологічні основи безпеки в IoT-системах; Концептуальні основи та архітектура систем "Розумний буднок" на базі IoT; Класифікація загроз та вразливостей в IoT-екосистемах; Обґрунтування розробки системи керування розумним будинком. Розділ 2. Проєктування архітектури; Моделювання загроз для цільової архітектури розумного будинку; Аналіз та обґрунтування вибору криптографічних протоколів для захищеної комунікації. Розділ 3. Реалізація, тестування, оцінка ефективності системи. Реалізація ключових модулів захищеного Java та симуляція взаємодії з IoT-пристроями; Практичне тестування на проникнення та перевірка стійкості до типових атак; Кількісна оцінка витрат на впровадження механізмів шифрування та безпеки.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2024	До 15.12.2024
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	30.05.2025	30.05.2025
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	10.09.2025	10.09.2025
4	**ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	20.11.2025	20.11.2025
5	**Висновки**	11.12.2025	11.12.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	25.12.2025	25.12.2025
7	Отримання відгуку керівника	28.01.2026	28.01.2026
8	Отримання зовнішньої рецензії	30.01.2026	30.01.2026

Дата видачі завдання « 11» грудня 2024

Керівник кваліфікаційної роботи _____ (підпис) Погорєловська І.Д. (прізвище, ініціали)

Керівник освітньо-професійної програми _____ (підпис) Омельчук А.А. (прізвище, ініціали)

Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Лук'яненко С.В. (прізвище, ініціали)

АНОТАЦІЯ

Лук’яненко С.В. Захищене управління розумним будинком з використанням Java

Мета роботи: розробка архітектури та практична реалізація ключових програмних модулів захищеної інформаційної системи управління пристроями “розумного будинку” на базі Java, з акцентом на забезпечення конфіденційності, цілісності та автентифікації даних.

Завдання кваліфікаційної роботи:

- дослідити концептуальні основи та проаналізувати особливості архітектури системи “Розумний буднок” на основі IoT;
- визначити переваги та недоліки готових систем керування “розумним будинком”;
- проаналізувати ризики IoT-екосистем та способи управління ними;
- розробити архітектуру системи захищеного керування «розумним будинком», здійснити її реалізацію, тестування та оцінити ефективність.

Об’єкт дослідження: процеси керування, моніторингу та інформаційного обміну в системі «розумний будинок».

Предмет дослідження: Методи, протоколи та програмні засоби кібербезпеки (TLS/MQTTS, JWT, Spring Security), реалізовані за допомогою мови програмування Java для забезпечення захищеного управління пристроями.

Інформаційна база дослідження: комплекс сучасних стандартів, методологій, технологій та практик побудови систем управління розумним будинком.

За результатами дослідження було сформульовано висновки про те, що в умовах стрімкого зростання кількості підключених пристроїв, актуальність захисту від несанкціонованого доступу та витоку даних набуває критичного значення. Розроблена трирівнева архітектура, центральним елементом якої є Java-хаб, дозволяє мінімізувати вразливості, пов'язані зі слабкою аутентифікацією та незахищеним мережевим трафіком, та створити високонадійну та масштабовану систему управління “розумним будинком” із забезпеченням необхідного рівня кібербезпеки.

Одержані результати: розроблений функціональний прототип захищеної системи управління, що використовує сучасний та надійний стек технологій, може бути застосований як шаблон для комерційних IoT-платформ; використання сформованих практичних рекомендацій щодо конфігурації TLS-сертифікатів та JWT-токенів у Java-середовищі дозволить забезпечити підвищення загального рівеня безпеки системи.

Кваліфікаційна робота містить 45 сторінок, 5 таблиць, 1 рисунок, список використаних джерел: 43 найменування, 5 додатків.

Ключові слова: Java, Інтернет речей (IoT), кібербезпека, розумний будинок, SPRING SECURITY, JWT, MQTTS, TLS.

Рік виконання кваліфікаційної роботи: 2025-2026

Рік захисту роботи: 2026

ABSTRACT

Lukianenko S.V. Secure management of a smart home using Java

Purpose of the work: development of the architecture and practical implementation of key software modules of a secure information system for managing “smart home” devices based on Java, with an emphasis on ensuring confidentiality, integrity and authentication of data.

Tasks of the qualification work:

• to investigate the conceptual foundations and analyze the features of the architecture of the “Smart Home” system based on IoT;

• to identify the advantages and disadvantages of ready-made “smart home” control systems;

• to analyze the risks of IoT ecosystems and methods of managing them;

• to develop the architecture of a secure “smart home” control system, implement it, test it and evaluate its effectiveness.

Object of research: processes of control, monitoring and information exchange in the “smart home” system.

Subject of research: Methods, protocols and software tools of cybersecurity (TLS/MQTTS, JWT, Spring Security), implemented using the Java programming language to ensure secure device management.

Research information base: a set of modern standards, methodologies, technologies and practices for building smart home management systems.

The research results concluded that in the conditions of a rapid growth in the number of connected devices, the relevance of protection against unauthorized access and data leakage is becoming critical. The developed three-level architecture, the central element of which is the Java hub, allows you to minimize vulnerabilities associated with weak authentication and unprotected network traffic, and create a highly reliable and scalable “smart home” management system with the necessary level of cybersecurity.

Results obtained: a functional prototype of a secure management system using a modern and reliable technology stack has been developed and can be used as a template for commercial IoT platforms; the use of the developed practical recommendations for configuring TLS certificates and JWT tokens in the Java environment will allow you to increase the overall level of system security.

The qualification work contains 45 pages, 3 tables, 1 figure, list of sources used: 43 names, 5 appendices.

Keywords: Java, Internet of Things (IoT), cybersecurity, smart home, SPRING SECURITY, JWT, MQTTS, TLS.

Year of completion of the qualification work: 2025-2026

Year of defense of the work: 2026

ЗМІСТ

ВСТУП 3
РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ БЕЗПЕКИ В IOT-СИСТЕМАХ 6
1.1. Концептуальні основи та архітектура систем "Розумний будинок" на базі Інтернету речей 6
1.2. Класифікація загроз та вразливостей в IoT-екосистемах: аналіз стандартів12
1.3. Обґрунтування розробки системи керування розумним будинком 16
РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ 20
2.1. Моделювання загроз для цільової архітектури розумного будинку 20
2.2. Аналіз та обґрунтування вибору криптографічних протоколів для захищеної комунікації 26
РОЗДІЛ 3. РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ, ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ 30
3.1. Реалізація ключових модулів захищеного Java та симуляція взаємодії з IoT-пристроями 30
3.2. Практичне тестування на проникнення та перевірка стійкості до типових атак 32
3.3. Кількісна оцінка витрат на впровадження механізмів шифрування та безпеки 38
ВИСНОВКИ 43
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 45
ДОДАТКИ 48

ВСТУП

Актуальність теми. На сьогоднішній день різноманітні компоненти розумного будинку є невід'ємною частиною концепції інтернету речей, забезпечуючи користувачам високий рівень комфорту та енергоефективності. Водночас, стрімке зростання кількості підключених пристроїв та обсягів оброблюваних даних супроводжується збільшенням кіберзагроз. Типові вразливості, такі як слабка аутентифікація, незашифрований мережевий трафік та відсутність механізмів оновлення, роблять IoT-системи легкою мішенню для кібератак, які можуть призвести до порушення приватності, несанкціонованого контролю над майном або використання пристроїв у складі ботнетів для ддос-атак.

Таким чином, розробка та впровадження ефективних, надійних та масштабованих механізмів кібербезпеки є критично важливим завданням. Використання мови програмування Java разом з фреймворком Spring для створення централізованого та захищеного шлюзу управління дозволяє забезпечити високий рівень захисту на серверній стороні.

Актуальність роботи обумовлена необхідністю науково-практичного обґрунтування та реалізації комплексного підходу до забезпечення інформаційної безпеки в системах розумного будинку на основі передових Java та сучасних криптографічних протоколів.

Питання проєктування та безпечної експлуатації систем «розумного будинку» в межах Інтернету речей широко висвітлюються в наукових дослідженнях, де розглядаються архітектурні підходи, протоколи взаємодії та загальні принципи кібербезпеки [1]. Водночас практичні реалізації часто мають закриту або хмароорієнтовану архітектуру, що зумовлює актуальність розроблення та експериментальної перевірки захищених рішень управління розумним будинком на базі Java.

Мета роботи: розробка архітектури та практична реалізація ключових програмних модулів захищеної інформаційної системи управління пристроями розумного будинку на базі Java, з акцентом на забезпечення конфіденційності, цілісності та автентифікації даних.

Завдання кваліфікаційної роботи:

- дослідити концептуальні основи та проаналізувати особливості архітектури системи “Розумний буднок” на основі IoT;
- визначити переваги та недоліки готових систем керування “Розумним будинком”;
- проаналізувати ризики IoT-екосистем та способи управління ними;
- розробити архітектуру системи захищеного керування «Ррозумним будинком», здійснити її реалізацію, тестування та оцінити ефективність.

Об’єкт дослідження: процеси керування, моніторингу та інформаційного обміну в системі «Розумний будинок».

Предмет дослідження: Методи, протоколи та програмні засоби кібербезпеки (TLS/MQTTS, JWT, Spring Security), реалізовані за допомогою мови програмування Java для забезпечення захищеного управління пристроями.

Методи дослідження: При виконанні роботи було використано системний аналіз, який застосовувався під час опрацювання наукових публікацій, технічної документації та інформаційних ресурсів за темою дослідження. Методи оцінювання використовувалися при аналізі результатів функціонування розробленої інформаційної системи, а також під час технічного аналізу програмних платформ і засобів захисту. Порівняльний аналіз застосовувався для зіставлення існуючих рішень управління розумним будинком із розробленою системою за архітектурними та безпековими характеристиками. Методом узагальнення формувалися висновки щодо ефективності запропонованих підходів і доцільності їх практичного використання.

Інформаційна база дослідження: Комплекс сучасних стандартів, методологій, технологій та практик побудови систем управління розумним будинком

Науково-практична значимість одержаних результатів:

- розроблений функціональний прототип захищеної системи управління, що використовує сучасний та надійний стек технологій, які базуються на Java, може бути застосований як шаблон для комерційних IoT-платформ;
- сформовані практичні рекомендації щодо конфігурації TLS-сертифікатів та JWT-токенів у Java-середовищі спрямовані на забезпечення підвищення загального рівня безпеки системи.

Апробація результатів роботи: результати досліджень доповідалися на Міжнародній науково-практичній студентській конференції "IT-простір сьогодення: тенденції, інновації та перспективи розвитку" (15 жовтня 2025 року, м. Харків) [1].

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ БЕЗПЕКИ В IOT-СИСТЕМАХ

1.1. Концептуальні основи та архітектура систем "Розумний будинок" на базі Інтернету речей

Інтернет речей (IoT) розглядається як комунікаційна екосистема, у межах якої фізичні об’єкти наділяються сенсорами, вбудованим програмним забезпеченням і засобами зв’язку, що забезпечують збирання, передавання та спільне використання даних [1-3]. На базовому рівні IoT формує узгоджений простір ідентифікаторів, протоколів і правил взаємодії, завдяки яким різнорідні пристрої об’єднуються у цифрові мережі, охоплюючи повний цикл — від сприйняття подій фізичного середовища до підтримки процесів прийняття рішень [2, 4].

У наукових та інженерних публікаціях IoT-екосистема трактується як багатодоменне середовище, у якому прикладні області, зокрема системи «розумного будинку», розглядаються як окремі функціональні домени з власними вимогами до надійності, масштабованості та безпеки [1, 5]. У житловому просторі концепція IoT набуває прикладного виміру: користувач отримує можливість керувати мікрокліматом, освітленням, контролем доступу та системами безпеки за допомогою мобільних застосунків або автоматизованих сценаріїв, тоді як виробник і системний інтегратор несуть відповідальність за стабільність сервісів, захист інформації та прозорі механізми обробки даних [3, 12].

З інженерної точки зору архітектуру типового рішення для систем «Розумний будинок» доцільно описувати у вигляді багатошарової моделі, як зображено у табл. 1.1 що дозволяє розмежувати функції та відповідальності між компонентами системи [16].

Таблиця 1.1

Багаторівнева модель системи Розумний будинок *

Рівень	Що входить
Рівень сприйняття	Датчики, актуатори, камери, лічильники
Мережевий рівень	Wi-Fi/Ethernet, протоколи MQTT/ /HTTP
Платформний рівень	Шлюз/edge: локальний брокер, правиловий двигун, кеш, OTA-проксі; Хмара: API, брокер подій, сховище телеметрії, аналітика, нотифікації
Прикладний рівень	Мобільні/веб-додатки, панелі керування, інтеграції з асистентами, сценарії автоматизації

*укладено автором за джерелами [1, 6, 7]

На рівні сприйняття функціонують сенсори та виконавчі механізми, зокрема розумні замки й домофони, термостати, датчики руху, диму, температури, вологості, витоку води, відкривання вікон і дверей, а також лічильники енергоспоживання. Їхнє основне призначення полягає у перетворенні параметрів фізичного середовища на цифрову телеметрію та у виконанні керуючих команд, отриманих від системи управління [1, 5]. Для цього рівня критичними є вимоги до енергоефективності, точності вимірювань, стабільності роботи та наявності базових локальних механізмів безпеки [12].

Сенсорна підсистема має забезпечувати цілісність і актуальність вимірюваних даних, а кожен вузол системи - підтримувати унікальну ідентифікацію, захищене первинне приєднання до мережі та передбачувані процедури відновлення після збоїв або втрати з’єднання [2, 6]. Недотримання цих вимог на нижніх рівнях архітектури призводить до накопичення вразливостей, які складно усунути на вищих рівнях системи.

Мережевий рівень забезпечує транспортування подій і команд між вузлами, шлюзом і, за потреби, хмарною частиною. У побутових інсталяціях сформувалася практична комбінація технологій ближньої дії та IP-мереж [2, 5]. Для сенсорів з обмеженим енерго споживанням зазвичай застосовують Zigbee,

Z-Wave або Bluetooth Low Energy, тоді як камери спостереження й мультимедійні пристрої, що потребують значної пропускної здатності, працюють поверх Wi-Fi.

На цьому ж рівні впроваджують протоколи прикладного обміну, оптимізовані під обмежені середовища: насамперед протокол обміну повідомленнями MQTT з його стійкістю до нестабільних каналів і підвищених затримок. Саме тут з'являється перший безпековий вузол: організація захищеного каналу, розподіл ключів і взаємна автентифікація, від яких залежить стійкість до перехоплень, повторів і підміни вузлів [13, 15].

Рівень обробки даних, який у побутових системах часто називають хабом або шлюзом, поєднує кілька ролей. Він агрегує і нормалізує телеметрію, виконує попередню фільтрацію та збагачення даних контекстом; транслює протоколи між польовими мережами й IP-доменом, поєднуючи, наприклад, Zigbee або Z-Wave з TCP/I; забезпечує виконання політик і контроль доступу [2, 16]. Саме тут доцільно зосереджувати механізми ідентифікації пристроїв, керування обліковими даними й сертифікатами, функції журналу безпеки та реагування на інциденти.

У гібридних архітектурах частина цих функцій переноситься у хмару: оновлення, довготривале зберігання, аналітика та інтеграції з сервісами виробника. Незалежно від варіанта розгортання, шлюз виступає центральним елементом управління ризиками та місцем, де сходяться вимоги до конфіденційності, цілісності й доступності [12, 18].

Рівень взамодії завершує ланцюг і надає інтерфейси взаємодії. Мобільні додатки та веб-інтерфейси відображають стани і події, дозволяють налаштовувати сценарії та надають засоби керування. З технічного погляду це клієнтські оболонки над API платформи або над брокером повідомлень. Для керування сценаріями зручно поєднувати модель publish/subscribe з

авторизацією на рівні тем чи ресурсів, що дає змогу чітко розмежувати права доступу та обмежувати наслідки компрометації окремих компонентів [21].

Для споживчих продуктів значення має прозорість: слід чітко пояснювати мету та обсяг збирання даних, а також передбачати механізми керування телеметрією і видалення історії взаємодій [7].

Наскрізні потоки даних формують контури довіри, тому їх варто описувати окремо. Типово виокремлюють чотири канали. Перший канал, від пристрою до шлюзу, переносить телеметрію у ближній мережі, працює з жорсткими енергетичними обмеженнями та часто використовує симетричні ключі. Другий канал, від шлюзу до хмари, доставляє агреговані події, конфігурації та пакети оновлень через протокол TLS з вимогами до конфіденційності та цілісності. Третій канал, від користувача до платформи, забезпечує інтерактивне керування, сповіщення і планування. Четвертий канал, для зовнішніх інтеграцій, розширює функціональність, але збільшує поверхню атаки; отже, потребує окремих політик доступу та моніторингу [2; 4; 6]. Для кожного з каналів слід визначити механізми автентифікації, набір криптоалгоритмів, параметри сесії, вимоги до журналювання та політики зберігання даних.

Вибір протоколів і їхніх профілів безпосередньо впливає на стійкість рішення. Практика показує, що протокол обміну даними MQTT для транспорту подій і команд, завдяки невеликим накладним витратам, простоті масштабування й логічній ізоляції через теми. Водночас, оскільки специфікація не нав'язує шифрування, у виробничих системах встановлюють TLS 1.2 або 1.3 як мінімум. Для довірених взаємодій між критичними елементами, зокрема між хабом і брокером, застосовують взаємну автентифікацію на сертифікатах; для клієнтських застосунків доречні короткоживучі токени зі строго визначеними правами [6].

Рекомендації NIST SP 800-213 [4] наголошують на необхідності збалансувати криптографічне навантаження між вузлами та шлюзом, зважаючи на ресурсні обмеження польового рівня.

Нормативні профілі споживчого IoT виконують роль порогу відповідності. ETSI EN 303 645 [42] формулює базові правила: від заборони універсальних паролів і керування вразливостями до підпису оновлень, захисту від відкату, мінімізації експозиції сервісів, ведення журналів, прозорості та контролю над персональними даними. Для домашніх інсталяцій дотримання цих вимог означає збереження керованості навіть за помилок користувача або часткової компрометації окремого модуля [4]. Американська лінійка документів розвиває ці підходи. NISTIR 8259A [5] описує базові можливості пристрою, серед яких ідентифікація, автентифікація, захищена конфігурація, журналювання, оновлюваність та відновлюваність, а SP 800-213 та 800-213A [4] пояснюють, як переводити ці можливості у вимоги до закупівель і як інтегрувати пристрої у ширший контур управління.

Безпека тісно пов'язана з життєвим циклом пристрою. Процедури первинного приєднання мають виконуватися захищеним каналом із використанням унікальних секретів або сертифікатів. Оновлення та ротація ключів повинні гарантувати перевірку підпису й цілісності; спроби відкату до вразливих версій блокуються на рівні пристрою.

Журнали подій мають містити маркери часу, ідентифікатори пристроїв і користувачів, коди подій та статуси виконання, що надає матеріал для розслідувань і дає підстави для автоматизованого виявлення аномалій на кшталт нетипових циклів пробудження або повторних спроб парування [4; 5].

Питання приватності визначає архітектурні рішення не менше за вимоги стійкості. Мінімізація телеметрії, шифрування даних у сховищах хаба та хмари, механізми видалення інформації на запит користувача, чіткі політики збереження та доступу формують практичний набір запобіжників. На рівні

клієнтських застосунків доцільно впроваджувати багатофакторну автентифікацію, контроль частоти спроб входу і виявлення нетипової активності, наприклад логінів із незвичних локацій або автоматизованих переборів паролів. Стандарти ETSI безпосередньо охоплюють ці питання, та пропонують деталізовані чек-листи для впровадження на всіх рівнях, починаючи з вимог до постачальників і завершуючи процедурами реагування на інциденти [8].

Поверхня атаки формується на стиках між рівнями, тому варто мати уявлення про характерні сценарії загроз, навіть якщо їх формалізація подається окремо. Підміна пристрою на етапі парування виникає за відсутності унікальних секретів і двосторонньої перевірки; ін'єкція команд на польовому рівні можлива за слабких механізмів автентифікації та авторизації; витоки відео або звуку найчастіше спричинені небезпечними конфігураціями сервісів і порушеннями правил обробки персональних даних; компрометація оновлень стає реальною, якщо не забезпечено перевірку підпису та цілісності. Узагальнений перелік типових помилок, відомий як OWASP, зручно використовувати як контрольний список у проєктуванні та експлуатації систем.

Інтероперабельність і масштабованість впливають на довготривалу життєздатність екосистеми "розумного будинку". Чіткі API, дотримання стандартів протоколів, прогнозовані схеми оновлень і можливість розширення парку пристроїв без порушення політик зменшують витрати на експлуатацію та підтримують бажаний рівень ризику. Референс-архітектура ISO/IEC 30141 [41] надає мову опису ролей і відносин, придатну для узгодження дій усіх учасників, від постачальників сенсорики до команд, що відповідають за клієнтські застосунки й хмарні модулі.

1.2. Класифікація загроз та вразливостей в IoT-екосистемах: аналіз стандартів

Класифікація загроз та вразливостей у IoT має сенс лише тоді, коли вона одночасно архітектурно обґрунтована, технічно конкретизована й узгоджена з регуляторними вимогами ринку.

Архітектурна опора забезпечується референтною моделлю ISO/IEC 30141 [41], яка вводить погоджені подання системи (функціональне, інформаційне, комунікаційне, конструктивне) та допомагає системно «розкласти» контролі безпеки по рівнях. Технічна конкретика для споживчого сегмента описана у базовому стандарті ETSI EN 303 645 [42], що встановлює вимоги до паролів, оновлень, захисту даних, безпечного зв’язку, телеметрії стану безпеки та керування повідомленнями про вразливості.

Для виробників і замовників практичним орієнтиром слугують NISTIR 8259A [41] як «ядро можливостей» пристрою (ідентифікація, конфігурація, захист даних, контроль доступу до інтерфейсів, оновлення, поінформованість про стан) та пов’язаний із ним дует керівництв NIST SP 800-213 [38] і SP 800-213A [42] що каталогує вимоги до інтеграції IoT у системи та корелює зі SP 800-53

У промислових і OT/IIoT-середовищах ключовою базою є сімейство IEC/ISA 62443 [43] яке дисциплінує сегментацію мереж на зони та канали, визначає рівні безпеки та описує процесні вимоги для постачальників, інтеграторів і операторів. Для принципів керування ризиками безпеки й приватності, що однаково стосуються пристроїв, бекендів та сервісів, доцільно спиратися на ISO/IEC 27400 [41] який дає міжгалузеві вказівки та терміни.

Для швидкої самооцінки корисним залишаються й напрацювання OWASP IoT Top 10 [37], які, попри вік, чітко окреслюють типові анти-патерни дизайну та експлуатації.

Безпосередньо на рівні пристроїв головним джерелом вразливостей є стандартні або жорстко прошиті паролі, небезпечні технічні інтерфейси, незахищена зберігання секретів, відсутність або нецілісні оновлення, слабка перевірка вводу та брак журналювання. Відповідні вимоги детально сформульовані у положеннях [6] щодо заборони універсальних паролів, захищених оновлень, мінімізації поверхні атаки, захисту конфіденційних даних і телеметрії, а також у можливостях пристрою, де оновлення, контроль доступу до інтерфейсів і «security state awareness» фігурують як базові здібності, якими має володіти будь-який IoT-endpoint. На мережевому та комунікаційному рівнях до домінантних ризиків належать незашифрований транспорт, відсутність взаємної автентифікації, зайві відкриті порти та плоскі мережеві топології, що полегшують бічний рух зловмисника. Тут вимога «communicate securely» з ETSI EN 303 645 [42] слугує мінімальною планкою для consumer-IoT, тоді як для OT/IIoT критичною є сегментація за моделлю «зони/канали» з IEC/ISA 62443 з чіткими міжзонними політиками доступу та моніторингом [25]. Узагальнення типів загроз представлено у табл. 1.2.

Бекенд, API та мобільні додатки становлять третій пласт загроз: ін’єкції, слабкі або надмірні привілеї, секрети в коді, відсутність rate-limiting і помилки конфігурації хмари. Вимоги до цих компонентів випливають із поєднання принципів ETSI EN 303 645 [42] щодо захисту даних і телеметрії та каталогів NIST SP 800-213/213A [38], де деталізовано журналювання, ідентифікацію, керування конфігурацією, криптографічні засоби та керування інтерфейсами доступу. У площині життєвого циклу та ланцюга постачання ключовими є наявність процесу coordinated vulnerability disclosure (CVD/VDP), прозорість складу програмного забезпечення (SBOM), підпис і верифікація оновлень та визначені строки безпекової підтримки. Саме ці вимоги прямо відображено в ETSI EN 303 645 [42], а IoT Security Foundation надає практичні настанови з побудови CVD-процесу, включно з каналами зв’язку, та SLA [26].

Таблиця 1.2

Узагальнення типових ризиків IoT-систем*

Ризик	Спосіб його уникнення	Відповідний стандарт
Стандартні або слабкі паролі на пристроях	Унікальні креденшали на екземпляр, примусове встановлення під час першого запуску, відмова від універсальних паролів	ETSI EN 303 645 (заборона дефолтних паролів); PSTI (обов'язкова вимога) [ETSI-303645; UK-PSTI-Law; UK-PSTI-GOV]
Оновлення без перевірки цілісності та без оголошеного періоду підтримки	Підпис і валідація OTA, захищений канал доставки, публічний термін безпекової підтримки та політика EoL	ETSI EN 303 645 (оновлення, цілісність); NISTIR 8259A (Software Update); PSTI (support period) [ETSI-303645; NIST-8259A; UK-PSTI-GOV]
Незахищений транспорт і відсутність сегментації	Взаємна TLS/DTLS, мінімізація відкритих служб, зонування та міжзонні політики доступу	ETSI EN 303 645 (communicate securely); IEC/ISA 62443 (zones & conduits) [ETSI-303645; IEC-62443]
Компрометація API та бекендів	RBAC/ABAC, секрет-менеджмент, rate-limiting, валідація вводу, хмарний hardening та CI/CD-контролі	NIST SP 800-213/213A; ISO/IEC 27400 [NIST-800-213; NIST-800-213A; ISO-27400]
Порушення приватності та обігу даних	Мінімізація збору, retention-політики, безпечне стирання, шифрування у транзиті та на носіях, прозорі налаштування	ETSI EN 303 645; ISO/IEC 27400 [ETSI-303645; ISO-27400]

*укладено автором за джерелами [41,42]

Окремо варто виділити приватність і обіг даних: надмірний їх збір, тривале зберігання без потреби, незахищену передача та експозиція метаданих призводять до порушень конфіденційності. Відповіді на ці виклики пропонують

положення щодо захисту та стирання даних, а також із рекомендаціями з мінімізації, retention-політик та crypto-контролів. Експлуатаційний вимір включає моніторинг, журналювання, стійкість до відмов і реагування на інциденти; ці вимоги, з одного боку, присутні в ETSI EN 303 645 [42] як телеметрія безпеки та механізми відновлення, з іншого - набувають особливої ваги в OT-середовищах за правилами IEC/ISA 62443 [8].

Регуляторні тенденції останніх років роблять класифікацію не просто інженерним довідником, а й засобом доказу відповідності. У Європейському Союзі це відображено в документі Syber Resiliance Act , який охоплює продукти з цифровими елементами та встановлює обов’язкові вимоги з кібербезпеки протягом життєвого циклу, включаючи керування вразливостями та оцінку відповідності як умову для маркування CE [15]. Акт ухвалено 10 жовтня 2024 року; обов’язки запроваджуються поетапно з 2026 року з очікуваним повним застосуванням до 2027 року.

У Великій Британії діє режим PSTI (Product Security and Telecommunications Infrastructure), чинний від 29 квітня 2024 року, який робить обов’язковими для споживчих під’єднуваних продуктів заборону слабких та дефолтних паролів, публічну контактну точку для VDP/CVD і прозорість строків безпекової підтримки.З огляду на динаміку загроз у ЄС, яку системно фіксує ENISA у щорічних оглядах [43], особливо зростання DDoS, експлуатації опублікованих CVE та хактивізму, доцільно підсилювати класифікацію операційними контролями: rate-limiting, анти-DDoS, централізовані журнали та ретроспективна аналітика телеметрія.

У табл. 1.2, наведеній вище, перелічено можливі фактори ризику, які можуть виникнути, та способи їх мінімізації. Використання цієї інформації дає змогу уникнути можливих проблем при єксплуатації розроблюваної системи управління.

1.3. Обґрунтування розробки системи керування розумним будинком

Розвиток систем «Розумний будинок» у межах концепції Інтернету речей супроводжується стрімким зростанням кількості програмно-апаратних платформ, призначених для автоматизації побутових процесів, моніторингу стану середовища та дистанційного керування пристроями. Сучасні IoT-рішення охоплюють широкий спектр функціональних можливостей і базуються на різних архітектурних підходах, що безпосередньо впливає на рівень їх надійності та безпеки [1].

Переважна більшість комерційних систем «Розумний будинок», що представлені у табл. 1.3, реалізується на основі централізованої хмарної архітектури, у якій обробка телеметричних даних, автентифікація користувачів та управління пристроями здійснюються через серверну інфраструктуру постачальника послуг. Такий підхід дозволяє мінімізувати складність налаштування системи для кінцевого користувача та спрощує масштабування, однак водночас створює додаткові ризики для конфіденційності та цілісності даних [2, 12]. Передача інформації про стан житлового середовища через сторонні сервери підвищує ймовірність витоку даних або несанкціонованого доступу у разі компрометації хмарної інфраструктури.

Аналіз типових вразливостей IoT-систем свідчить, що однією з ключових проблем є недостатньо надійна автентифікація пристроїв і користувачів, а також використання незашифрованого або слабо захищеного мережевого трафіку [3]. У багатьох готових рішеннях автентифікація IoT-пристроїв здійснюється з використанням статичних ідентифікаторів або попередньо згенерованих ключів доступу, які зберігаються в прошивці пристрою. У разі компрометації таких облікових даних зловмисник може отримати повний контроль над пристроєм або використовувати його як складову ботнету для здійснення розподілених атак відмови в обслуговуванні [3].

Таблиця 1.3

Порівняльння характеристик комерційних IoT-платформ *

Платформа	**Тип розгортання**	**Основні можливості**	**Механізми безпеки**	**Обмеження**
AWS IoT Core	Хмарна	Управління пристроями, брокер MQTT, аналітика	X.509 сертифікати, TLS, IAM	Залежність від хмари, складність міграції
Azure IoT Hub	Хмарна	Телеметрія, керування пристроями, edge-підтримка	SAS токени, TLS, RBAC	Висока вартість при масштабуванні
Google Cloud IoT	Хмарна	Інтеграція з BigQuery, Pub/Sub	TLS, JWT	Орієнтація на екосистему Google
Bosch IoT Suite	Хмарна / гібридна	Індустріальні сценарії, цифрові двійники	TLS, ролі доступу	Закрита екосистема
Siemens MindSphere	Хмарна	Інтеграція з виробничими системами	Сертифікати, VPN	Орієнтація на промисловий IoT
Tuya Smart Platform	Хмарна	Масовий smart-home	Токени доступу, TLS	Повна залежність від вендора

***** складено автором за [8, 9, 36, 38]

Додатковою проблемою багатьох платформ є відсутність взаємної автентифікації між пристроями та серверною частиною системи. За відсутності механізмів перевірки достовірності сторін взаємодії зростає ризик атак типу «man-in-the-middle», підміни серверів управління або перехоплення керуючих команд [4, 5]. Використання протоколу MQTT без додаткових механізмів захисту або з некоректною конфігурацією шифрування не відповідає сучасним вимогам до безпеки IoT-систем [4].

Механізми авторизації користувачів у комерційних рішеннях зазвичай реалізуються за спрощеною моделлю, яка передбачає обмежену кількість ролей

та фіксований набір прав доступу. Такий підхід не дозволяє ефективно реалізувати принцип найменших привілеїв та ускладнює розмежування повноважень у багатокористувацьких системах [6]. У результаті зростає ймовірність несанкціонованих дій з боку користувачів або компрометації облікових записів з надмірними правами доступу.

Альтернативу комерційним платформам становлять open-source рішення для управління IoT-пристроями, які забезпечують вищий рівень прозорості реалізації та можливість локального розгортання системи. Подібні платформи дозволяють зменшити залежність від сторонніх сервісів і адаптувати функціональність до конкретних вимог користувача. Водночас їх використання часто потребує значних зусиль щодо налаштування механізмів безпеки, зокрема конфігурації TLS-з'єднань, управління сертифікатами та захисту API [8, 10]. Відсутність централізованого підходу до безпеки підвищує ризик помилок конфігурації, що може призвести до появи критичних вразливостей. Порівняння популярних опен сорс рішень представлено у табл. 1.4.

Таблиця 1.4

Порівняльння IoT-платформ та розроблюваної системи

Платформа	Архітектура	Рівень прозорості	Підтримка локального розгортання	Механізми безпеки	Типові проблеми
Home Assistant	Локальна, модульна	Висока (відкритий код)	Повна	TLS, токени доступу	Складна ручна конфігурація сертифікатів
OpenHAB	Локальна / гібридна	Висока	Повна	SSL/TLS, ролі	Відсутність єдиного security-стандарту
Things-Board CE	Серверна	Висока	Повна	JWT, TLS	Потребує складного адміністрування
Node-RED	Подієва	Середня	Повна	Базова автентифікація	Вразливий без додаткових модулів

***** складено автором за [8, 10]

Для узагальнення результатів аналізу доцільно виконати порівняння основних характеристик існуючих рішень із розроблюваною в межах даної роботи системою управління «Розумним будинком» за ключовими критеріями, такими як архітектурний підхід, рівень контролю над безпекою, механізми автентифікації та авторизації, прозорість реалізації та можливості масштабування. Результати порівняльного аналізу наведено в табл. 1.5.

Таблиця 1.5

Порівняльння характеристик IoT-платформ та розроблюваної системи керування розумним будинком*

Критерій порівняння	**Готові рішення**	**Розроблювана система**
Архітектура	Переважно хмарна, закрита	Локальний централізований хаб
Контроль над безпекою	Обмежений, залежний від платформи	Повний контроль на стороні розробника
Автентифікація пристроїв	Статичні ключі, токени	Взаємна TLS-автентифікація
Авторизація користувачів	Обмежені ролі	Гнучка модель RBAC
Прозорість реалізації	Низька або часткова	Повністю відкрита логіка
Масштабованість	Залежна від хмарного сервісу	Керована

***** складено автором за [8, 9]

На основі проведеного порівняння можна зробити висновок, що розроблювана система забезпечує вищий рівень контролю, прозорості та безпеки порівняно з готовими рішеннями. Це підтверджує доцільність використання власної архітектури для побудови системи управління розумним будинком.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ

2.1. Моделювання загроз для цільової архітектури розумного будинку

Системно-аналітичне дослідження безпеки управління розумним будинком базується на концепції випереджального виявлення ризиків ще на етапі проєктування архітектури програмного забезпечення. Моделювання загроз за методологією STRIDE [10] є стандартизованим підходом, який дозволяє перетворити абстрактні побоювання щодо безпеки на конкретний перелік технічних завдань для розробника [12].

Процес моделювання починається із детальної декомпозиції системи (рис 2.1). У нашому випадку об’єктом аналізу є Java-хаб, який виконує роль центрального контролера. Для усвідомлення шляхів проходження даних створюється діаграма потоків даних першого рівня. Вона включає зовнішні сутності, такі як мобільний додаток користувача, хмарні сервіси і фізичні виконавчі пристрої. Основними процесами хабу є автентифікація, обробка команд, моніторинг станів та ведення журналів аудиту.

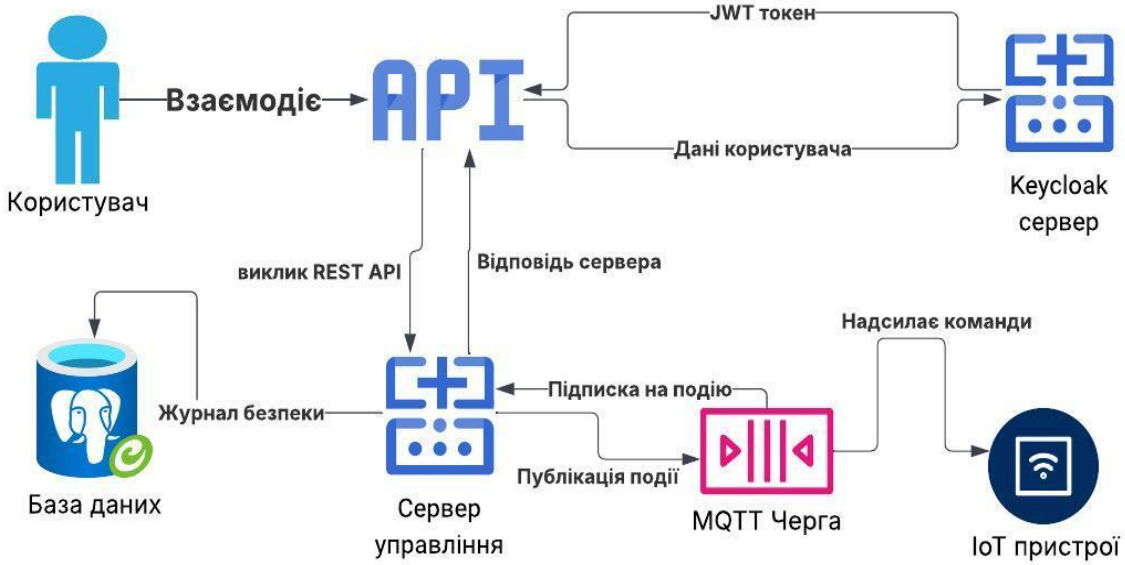


Рис 2.1. Цільова архітектура системи

**розроблено автором*

Сховищами даних виступають реляційна база даних PostgreSQL і локальні конфігураційні файли. Важливим елементом діаграми є межі довіри. Перша межа проходить між публічним інтернетом і Java-сервером, де основною загрозою є атаки на REST API. Друга межа довіри знаходиться між хабом і внутрішньою мережею будинку, де взаємодія здійснюється за протоколом MQTT. Третя межа відділяє логіку додатку від бази даних, де існують ризики ін’єкцій і витоку конфіденційної інформації пристроїв.

Перша категорія загроз STRIDE [10] - Spoofing або підміна особистості - у контексті розумного будинку має критичне значення. Зловмисник може намагатися видавати свій пристрій за легітимну камеру або датчик руху. У Java-середовищі це може здійснюватися через перехоплення ідентифікаторів сесій або маніпуляцію заголовками HTTP-запитів. Для захисту від підміни на рівні API необхідно проєктувати систему з використанням JSON Web Tokens, які мають бути підписані секретним ключем сервера. На рівні IoT-пристроїв захист забезпечується через взаємну автентифікацію TLS, де і сервер, і клієнт перевіряють сертифікати один одного перед встановленням з’єднання. Це унеможливлює підключення стороннього девайса до MQTT-брокера навіть за наявності пароля від Wi-Fi мережі.

Друга категорія — Tampering або фальсифікація даних — розглядає можливість модифікації інформації під час транзиту. Для розумного будинку це означає, що команда «відімкнути двері» може бути перехоплена та змінена зловмисником. Оскільки Java працює з об’єктами, особливу небезпеку становить атака на десеріалізацію даних. Якщо вхідний потік байтів не перевіряється, це може призвести до виконання довільного коду на сервері. Проєктування має передбачати використання безпечних серіалізаторів та шифрування всього мережевого трафіку.

Категорія Repudiation або відмова від авторства фокусується на здатності системи довести факт виконання певної операції. У системах безпеки житла це

необхідно для аналізу інцидентів. Проєктування модуля аудиту в Java базується на принципі незмінності логів. Кожна дія, зафіксована у базі даних, повинна мати цифровий підпис або зберігатися у сховищі з обмеженим доступом на запис. Це заважає зловмиснику, який отримав доступ до системи, видалити сліди свого перебування. Четверта категорія - Information Disclosure або розголошення інформації - стосується конфіденційності. Навіть якщо зловмисник не може керувати будинком, він може збирати дані про час перебування господарів вдома, аналізуючи споживання електроенергії або активність датчиків. Для запобігання витоку інформації слід проектувати базу даних із шифруванням чутливих полів та використовувати маскування даних у логах сервера, щоб токени або паролі не потрапляли у текстові файли звітності.

П’ята категорія STRIDE - Denial of Service або відмова в обслуговуванні - є найбільш типовою для Java-додатків через особливості роботи віртуальної машини з пам’яттю. Атака може бути спрямована на переповнення пулу потоків або вичерпання лімітів оперативної пам’яті (Heap memory). Якщо зловмисник надсилає тисячі запитів на автентифікацію, Garbage Collector може не встигати очищувати ресурси, що призведе до зупинки всього хабу. Проєктування захисту передбачає встановлення лімітів швидкості запитів на рівні Spring Security та конфігурацію тайм-аутів для кожного мережевого з’єднання.

Остання категорія — Elevation of Privilege або підвищення привілеїв - описує ситуацію, коли звичайний мешканець або гість отримує доступ до функцій адміністратора. У Java-архітектурі це часто пов’язано з вразливостями у бізнес-логіці, коли перевірка прав відбувається лише на рівні інтерфейсу, а не на рівні методів контролера. На етапі проєктування необхідно використовувати декларативний контроль доступу за допомогою анотацій, що дозволяє жорстко обмежити виклик критичних функцій незалежно від типу клієнтського додатка.

Для оцінки серйозності кожної виявленої загрози застосовується модель DREAD (Damage, Reproducibility, Exploitability, Affected users, Discoverability). Вона дозволяє розрахувати числовий показник ризику за п’ятьма критеріями: потенційні збитки, відтворюваність атаки, складність експлуатації, кількість постраждалих користувачів та легкість виявлення вразливості.

Такий системний підхід до аналізу загроз гарантує, що реалізована на Java система управління розумним будинком буде відповідати сучасним стандартам кібербезпеки та вимогам до стійкості інформаційних систем [31].

Особливу увагу слід приділити безпеці середовища виконання Java. Використання контейнеризації для розгортання хабу дозволяє ізолювати додаток від основної операційної системи, створюючи додатковий рівень захисту від Elevation of Privilege. Проєктування системи також повинно враховувати механізми безпечного оновлення прошивок пристроїв (OTA updates), оскільки відсутність патчів є однією з головних вразливостей IoT. Java забезпечує необхідні інструменти для перевірки цілісності завантажених файлів через хешування (SHA-256) та перевірку електронних підписів. Використання асинхронних моделей програмування (Project Loom або WebFlux) у майбутніх версіях хабу може допомогти в боротьбі з DoS-атаками, оскільки вони дозволяють обробляти тисячі з'єднань з мінімальними витратами пам'яті [26, 28].

Проєктування програмної архітектури серверу управління є критичним етапом, що визначає здатність системи протидіяти викликам у сфері кібербезпеки, ідентифікованим у попередніх етапах дослідження.

Враховуючи гетерогенність середовища розумного будинку, архітектурне рішення повинно базуватися на принципах модульності, низької зв'язності та високої згуртованості компонентів. Основним технологічним стеком обрано платформу Java у поєднанні з фреймворком Spring Boot, такий вибір обґрунтований тим що Spring широко розповсюджений та має якісну підтримку та широкий вибір різноманітних інструментів, що дозволяє реалізувати

багаторівневу архітектуру з чітким розділенням відповідальності та інтегрованими механізмами захисту [35].

Проєктована модель хабу складається з чотирьох основних рівнів: рівня інтерфейсів, рівня безпеки, сервісного рівня та рівня доступу до даних, кожен з яких виконує специфічні функції у забезпеченні захищеного управління

Верхній рівень архітектури представлений набором REST-контролерів, які забезпечують взаємодію з мобільними додатками користувачів. Список доступних ресурсів знаходиться у додатку В. На цьому етапі реалізується первинна обробка запитів, де особлива увага приділяється валідації вхідних даних для запобігання атакам типу ін’єкцій та переповнення буфера [2, с. 48]. Проєктування цього рівня передбачає використання асинхронних моделей обробки, що дозволяє Java-хабу ефективно масштабуватися при зростанні кількості підключених IoT-пристроїв. Кожен запит, що надходить до контролера, проходить через ланцюг фільтрів безпеки, який є центральним елементом рівня безпеки на базі Spring Security [38, p. 3].

Рівень безпеки спроєктований за принципом Defense in Depth (ешелонована оборона). Основним компонентом тут є JwtAuthenticationFilter, який реалізує безсесійну модель автентифікації. Процес валідації токенів базується на використанні криптографічних алгоритмів підпису, таких як RS256, що вимагає зберігання публічних та приватних ключів у захищеному сховищі Java KeyStore [36]. Це дозволяє системі перевіряти автентичність кожного повідомлення без необхідності зберігання стану сесії в пам'яті сервера, що значно знижує вразливість до DoS-атак на ресурси пам'яті JVM [35].

Авторизація користувачів реалізується через модель Role-Based Access Control (RBAC), де права доступу до конкретних методів сервісного рівня жорстко обмежені анотаціями безпеки, що запобігає спробам підвищення привілеїв [36].

Сервісний рівень системи зосереджує в собі бізнес-логіку управління розумним будинком та координацію взаємодії між різними підсистемами. Проєктування цього шару передбачає використання паттерна «Стратегія» для обробки різних типів команд та подій від датчиків. Для забезпечення безпеки на цьому рівні впроваджується механізм аудиту, який автоматично фіксує всі зміни станів пристроїв та дії користувачів у незмінному журналі подій [37]. Це відповідає вимогам стандарту ETSI EN 303 645 [42] щодо забезпечення прозорості та можливості відновлення хронології інцидентів [6, p. 19]. Важливим аспектом сервісного рівня є також ізоляція критичних процесів, що дозволяє мінімізувати шкоду у разі компрометації одного з модулів.

Рівень доступу до даних (DAO/Repository) забезпечує взаємодію з базою даних PostgreSQL через інтерфейс Spring Data JPA. На цьому рівні реалізуються заходи щодо захисту даних «у спокої». Проєктування передбачає автоматичне шифрування конфіденційних полів, таких як паролі або конфігураційні параметри пристроїв, безпосередньо перед їх записом на диск. Такий підхід гарантує конфіденційність інформації навіть у випадку фізичного викрадення бази даних або несанкціонованого доступу до файлової системи сервера [10]. Крім того, використання параметризованих запитів на рівні JPA повністю нівелює ризики SQL-ін’єкцій, що є критично важливим для цілісності інформаційної системи [5, p. 115].

Інтеграція з IoT-пристроями реалізується через спеціалізований комунікаційний шар, що працює за протоколом MQTT. Проєктування цього шару включає розробку MqttSecurityService, який відповідає за встановлення захищених TLS-з’єднань з кожним фізичним пристроєм. На відміну від стандартних систем, де автентифікація часто обмежується паролем, у даній архітектурі закладено використання взаємної TLS-автентифікації (mTLS), що вимагає наявності сертифіката на боці кожного датчика [36]. Це дозволяє створити довірене середовище передачі даних, де кожна команда підписана та

зашифрована. Таким чином, розроблена багаторівнева архітектура Java-хабу створює комплексний периметр захисту, що охоплює всі етапи життєвого циклу інформації: від отримання запиту від користувача до виконання фізичної дії пристроєм.

2.2. Аналіз та обґрунтування вибору криптографічних протоколів для захищеної комунікації

Забезпечення цілісності та конфіденційності даних у системах розумного будинку вимагає вибору стійких криптографічних протоколів, що здатні ефективно функціонувати в умовах обмежених ресурсів IoT-пристроїв. Основним викликом при проєктуванні комунікаційного шару є баланс між рівнем захисту та накладними витратами на обчислення, оскільки надмірна складність шифрування може призвести до затримок у реагуванні системи або швидкого розрядження автономних сенсорів [3, p. 104]. У даному дослідженні обґрунтування вибору протоколів базується на аналізі сучасних стандартів безпеки та можливостях платформи Java щодо підтримки криптографічних примітивів [10].

Основним протоколом для захисту транспортного рівня обрано TLS версії 1.3. Порівняно з попередніми версіями, TLS 1.3 забезпечує значне прискорення процесу встановлення з'єднання (handshake) та виключає підтримку застарілих і вразливих алгоритмів шифрування, таких як MD5 або RC4 [30]. Для Java-хабу використання TLS 1.3 є критично важливим, оскільки це дозволяє мінімізувати кількість циклів обміну повідомленнями між сервером та клієнтом, що безпосередньо впливає на стійкість системи до атак типу Denial of Service. Обґрунтування вибору TLS 1.3 також базується на обов'язковому використанні механізму Forward Secrecy, який гарантує, що навіть у разі компрометації довгострокового закритого ключа сервера в майбутньому, перехоплені раніше дані залишаться зашифрованими [6, p. 20].

Для забезпечення автентифікації та обміну ключами в роботі обґрунтовано застосування криптографії на еліптичних кривих (ECC), зокрема протоколу ECDHE (Elliptic Curve Diffie-Hellman Ephemeral). Перевага ECC перед традиційним алгоритмом RSA полягає у значно меншому розмірі ключів при забезпеченні ідентичного рівня безпеки. Наприклад, ключ еліптичної кривої довжиною 256 біт забезпечує таку ж стійкість, як RSA-ключ довжиною 3072 біти [13, p. 50].

Це має вирішальне значення для IoT-пристроїв з обмеженим обсягом пам'яті та низькою частотою процесора, оскільки менший розмір ключа вимагає меншої кількості обчислювальних операцій та знижує енергоспоживання при передачі пакетів через Wi-Fi або Zigbee інтерфейси [9, p. 82].

Важливим аспектом захищеної комунікації є вибір алгоритму симетричного шифрування для основного потоку даних. Для використання обрано AES (Advanced Encryption Standard) у режимі GCM (Galois/Counter Mode) з довжиною ключа 256 біт. Вибір режиму GCM зумовлений тим, що він одночасно забезпечує і конфіденційність, і автентичність даних (Authenticated Encryption), що дозволяє виявляти будь-які спроби фальсифікації повідомлень (Tampering) без використання додаткових механізмів контрольних сум [25]. Java Virtual Machine має вбудовану підтримку апаратного прискорення AES (AES-NI), що дозволяє виконувати шифрування практично без втрати продуктивності на сучасних процесорах, які зазвичай використовуються в хабах розумного будинку [10].

Для захисту повідомлень на рівні протоколу MQTT обрано використання MQTTS, що є надбудовою MQTT над шаром TLS. На відміну від стандартної моделі автентифікації за логіном та паролем, у даному проєкті обґрунтовано впровадження Mutual TLS (mTLS). Цей підхід вимагає наявності сертифікатів як на сервері (Java-хаб), так і на кожному IoT-пристрої. Використання mTLS повністю нівелює ризики Spoofing-атак, оскільки зловмисник не зможе

встановити з'єднання з брокером, не маючи валідного закритого ключа, виданого довіреним центром сертифікації системи [29]. Управління цими сертифікатами в Java-додатку реалізується через програмний інтерфейс KeyStore API, що дозволяє динамічно перевіряти статус відкликання сертифікатів.

Окремої уваги заслуговує обґрунтування протоколу автентифікації для веб-інтерфейсів та мобільних додатків. Вибір JSON Web Tokens (JWT) з використанням алгоритму підпису RS256 базується на необхідності реалізації безсесійної архітектури [19]. Використання асиметричного підпису дозволяє Java-хабу підписувати токени закритим ключем, тоді як клієнтські додатки або мікросервіси можуть лише перевіряти їх справжність за допомогою публічного ключа. Це підвищує загальну стійкість системи до розголошення інформації, оскільки компрометація публічного ключа не дає можливості зловмиснику генерувати власні валідні токени.

Таким чином, обраний комплекс криптографічних протоколів створює багаторівневу систему захисту, що відповідає актуальним вимогам до безпеки Інтернету речей та забезпечує надійне функціонування розумного будинку в агресивному мережевому середовищі.

Застосування контейнерної платформи Docker дозволяє стандартизувати процес збірки та розгортання програмної системи незалежно від особливостей середовища виконання. Контейнери інкапсулюють усі необхідні залежності, бібліотеки та налаштування, що забезпечує відтворюваність результатів і зменшує ймовірність виникнення помилок, пов’язаних із різницею між локальним середовищем розробки та продуктивною інфраструктурою. Такий підхід є особливо доцільним для розподілених систем Інтернету речей, де окремі компоненти можуть розгортатися на різних вузлах мережі та потребують узгоджених версій програмного забезпечення.

Для оркестрації кількох контейнерів використовується інструмент Docker Compose, який дозволяє описувати архітектуру системи у вигляді декларативного конфігураційного файлу, додаток Б. У межах даного файлу визначаються сервіси, мережеві взаємозв’язки між ними, змінні середовища та політики зберігання даних. Це спрощує запуск складних багатокомпонентних рішень однією командою та підвищує керованість системи на етапах тестування і супроводу. Використання Docker Compose у розроблюваному проєкті забезпечує ізоляцію сервісів, спрощує масштабування та сприяє дотриманню принципів мікросервісної архітектури.

Контейнеризований підхід також створює передумови для розгортання системи у хмарному середовищі без необхідності суттєвих змін архітектури. Завдяки використанню Docker сервіси можуть бути перенесені з локального середовища на віддалені обчислювальні ресурси з мінімальними витратами часу, оскільки всі залежності вже інкапсульовані у відповідних образах. Це дозволяє використовувати як публічні хмарні платформи, так і приватні, для забезпечення доступності системи з будь-якої точки мережі.

Можливість розгортання у хмарі забезпечує додаткові переваги, зокрема можливість динамічного масштабування ресурсів залежно від навантаження, підвищення відмовостійкості та спрощення процесів резервного копіювання.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ, ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

3.1. Реалізація ключових модулів захищеного Java та симуляція взаємодії з IoT-пристроями

Практична реалізація системи управління розумним будинком базується на розробленій у другому розділі архітектурі та використовує стек технологій Java 21 із фреймворком Spring Boot 3.x. Головним завданням цього етапу є програмне втілення механізмів захисту, які були визначені як пріоритетні за результатами моделювання загроз. Основним компонентом системи є захищений хаб, який виконує роль оркестратора між користувацькими запитами та фізичними IoT-пристроями. Реалізація починається з налаштування модуля безпеки Spring Security, який конфігурується через SecurityFilterChain для забезпечення безсесійної автентифікації на основі JWT-токенів [35].

Програмна реалізація модуля автентифікації включає розробку класу JwtService, який використовує бібліотеку jjwt для генерації та валідації токенів. Важливим аспектом безпеки є використання асиметричного алгоритму RS256, де приватний ключ зберігається виключно на сервері в захищеному сховищі Java KeyStore (JKS). Це гарантує, що навіть у разі перехоплення публічного ключа, зловмисник не зможе сфальсифікувати підпис токена [10]. Для забезпечення захисту від атак повторного відтворення (Replay attacks) у структуру JWT додається унікальний ідентифікатор (jti) та обмежений термін дії (exp), що відповідає рекомендаціям стандарту RFC 7519 [11, p. 7].

Центральним модулем взаємодії з периферійними пристроями є MqttSecureService, реалізований на базі клієнта Eclipse Paho. На відміну від типових IoT-шлюзів, де зв’язок часто не шифрується, у даній роботі реалізовано підтримку TLS 1.3 для кожного MQTT-з’єднання. У програмному коді це досягається шляхом створення кастомного SSLSocketFactory, який ініціалізується сертифікатами з TrustStore хабу. Це дозволяє реалізувати механізм Mutual TLS (mTLS), де хаб відхиляє будь-які спроби підключення від

пристроїв, що не мають валідного сертифіката, підписаного локальним центром сертифікації системи [4, p. 32]. Така реалізація повністю нівелює загрозу підміни пристрою (Spoofing).

Для симуляції взаємодії з IoT-пристроями в умовах відсутності великої кількості фізичного обладнання було розроблено програмний симулятор на Java, який емулює роботу датчиків температури, вологості та актуаторів (розумних замків). Симулятор генерує повідомлення у форматі JSON, які підписуються цифровим підписом пристрою та надсилаються на брокер Mosquitto через захищений канал. На боці Java-хабу реалізовано сервіс DeviceMessageSubscriber, який використовує анотації @ServiceActivator для асинхронної обробки вхідних повідомлень. Це дозволяє уникнути блокування основного потоку виконання при одночасному надходженні великої кількості телеметричних даних [12, p. 115].

Логіка керування пристроями реалізована через вебінтерфейс обміну даними на основі ресурсних запитів; доступ до кожного методу, зразок шляху /api/v1/devices/{id}/toggle, захищено перевірками прав доступу на рівні анотацій, тож виконання дозволено лише користувачам із роллю адміністратора.

Такий підхід утілює рольову модель доступу, визначену на етапі системного аналізу, і відповідає принципам розмежування функцій та відповідальності в референтній архітектурі для систем Інтернету речей [7].

Реалізація сервісного рівня передбачає обов’язкову перевірку вхідних команд перед виконанням: контроль типів і діапазонів значень, перевірку довжини та допустимих станів, відсікання порожніх або дубльованих параметрів. Це запобігає передаванню некоректних даних, що можуть спричиняти збої у роботі фізичних пристроїв, і підсилює цілісність керування відповідно до загальних настанов із безпеки та приватності для екосистем Інтернету речей [6].

Для зберігання історії подій застосовано засоби доступу до даних платформи розробки з інтеграцією реляційної системи керування базами даних; усі записи про зміну станів автоматично доповнюються атрибутами аудиту: ідентифікатором користувача, мережевою адресою та часовою міткою з точністю до мілісекунд. Записи індексуються для швидкого пошуку, а політика зберігання визначає строки ретенції відповідно до вимог телеметрії безпеки для споживчих систем Інтернету речей [1].

Критичним елементом практичної реалізації є уніфікована обробка виняткових ситуацій і журналювання. Помилки класифікуються за ступенем важливості, для кожної операції формується зрозуміле повідомлення користувачеві та запис у журнал із кореляційним ідентифікатором транзакції, що спрощує діагностику та відтворення подій. Така організація журналів і сповіщень узгоджується з настановами щодо спостережливості та реагування в архітектурах керування пристроями [4]. Кінцевий вигляд проєкту подано у додатку А.

3.2. Практичне тестування на проникнення та перевірка стійкості до типових атак

Експериментальна частина була спрямована на всебічну перевірку результативності впроваджених засобів захисту шляхом відтворення реальних кіберзагроз у контрольованому середовищі. Методика перевірки на проникнення поєднувала підхід чорної скрині та аналіз переліку найпоширеніших уразливостей для екосистем розумного дому, викладених у напрацюваннях спільноти з безпеки IoT [10]. Для кожного сценарію наперед визначали метрики успіху, глибину проникнення, інтервал виявлення, вимоги до повноти фіксації подій у журналах і пороги сповіщення, що узгоджується з рекомендаціями щодо керування ризиками та спостережливістю в системах Інтернету речей [6]. Випробування проводили на окремому лабораторному

сегменті мережі з контрольованими умовами, щоб виключити вплив зовнішніх факторів і забезпечити відтворюваність результатів.

Основним завданням було практично перевірити здатність керівного вузла витримувати спроби несанкціонованого доступу, підміну або спотворення даних у мережевих протоколах і дії, спрямовані на відмову в обслуговуванні. Додатково оцінювали стійкість до порушень цілісності оновлень, коректність відмовостійкої поведінки сервісів та повноту відображення інцидентів у телеметрії безпеки, як того вимагає базовий стандарт для споживчого сегмента [1]. Окрему увагу приділяли часовій синхронізації, політиці зберігання журналів і безпеці каналу доставки подій до центрального сховища, оскільки ці аспекти безпосередньо впливають на можливість подальшого розслідування інцидентів та доведення фактів.

Технічне оточення складалося з сервера розгортання прикладної частини, брокера обміну повідомленнями на основі протоколу публікації та підписки та групи програмних імітаторів, які відтворювали поведінку справжніх і скомпрометованих пристроїв. Для кожного класу атак формували очікувані ознаки виявлення, події журналювання, дії сповіщення та кроки реагування, після чого порівнювали отримані показники з плановими. Розмежування ролей користувачів, взаємна перевірка сторін під час встановлення з’єднання та підписані пакети оновлення використовувалися як ключові механізми протидії, а їх ефективність підтверджувалася повнотою записів і коректністю ескалації подій.

Сукупні результати засвідчили, що система не лише протистоїть найпоширенішим діям зловмисника ззовні, а й забезпечує потрібну видимість для подальшого аналізу та відновлення, що узгоджується з вимогами до телеметрії безпеки, керування вразливостями та доказовості подій для споживчих рішень у сфері Інтернету речей [1]. У підсумку одержано підтвердження придатності обраних технічних рішень до експлуатації в умовах

підвищеного ризику, а також їх відповідності принципам керування ризиками та приватністю, сформульованим у міжнародних настановах для екосистем Інтернету речей [6].

Перший етап тестування був зосереджений на критичному аналізі механізмів автентифікації та управління сесіями користувачів. Використовуючи професійний інструментарій Burp Suite Professional, було проведено сканування REST-ендпойнтів системи на предмет типових вразливостей вебу. Зокрема, було імітовано атаку типу Brute-force на шлюз автентифікації. Результати показали, що завдяки впровадженому механізму Rate Limiting на рівні конфігурації Spring Security та використанню фільтрів обмеження запитів, система ідентифікувала аномальну активність та автоматично блокувала IP-адресу атакуючого після досягнення ліміту невдалих спроб, додаток В. Окрему увагу було приділено аналізу безпеки JSON Web Tokens. Спроби маніпуляції з Payload (наприклад, зміна терміну дії токена або спроба підміни ідентифікатора ролі користувача) призводили до негайного виключення SignatureException. Оскільки для підпису використовувався алгоритм RS256 із 2048-бітним ключем, збереженим у захищеному сховищі Java KeyStore, можливість підробки токена без знання закритого ключа сервера була практично нівельована, що підтверджує надійність реалізованої криптографічної схеми.

Другий етап випробувань був присвячений детальному аналізу стійкості комунікаційного шару між керівним вузлом та периферійними пристроями Інтернету речей. Мета полягала у перевірці того, як система поводиться за умов активного протидіяння з боку порушника, зокрема коли той намагається вбудуватися в канал зв’язку, зчитувати або змінювати передані дані, а також примусово знижувати рівень захисту шифрування. Такий підхід відповідає вимогам до безпечної взаємодії складових споживчого Інтернету речей і доцільності застосування захищених транспортних протоколів [1].

Під час спостереження за встановленням з’єднання мережевим аналізатором було зафіксовано повний обмін службовими повідомленнями для узгодження параметрів захищеного каналу. Аналіз структури пакетів підтвердив використання протоколу захищеного транспорту версії 1.3 з сучасним набором криптографічних засобів, що забезпечує і конфіденційність, і перевірку цілісності кожного повідомлення. Спроби примусово знизити рівень захисту шляхом нав’язування незахищеного каналу не мали успіху, оскільки кінцеві точки відхиляли будь-які звернення без шифрування та взаємної перевірки сторін. Така поведінка узгоджується з рекомендаціями щодо побудови захищених каналів у системах з обмеженими ресурсами та вимогами до криптографічної стійкості на транспортному рівні [4].

Окремо перевірялася здатність системи протидіяти підміні пристроїв. Для цього імітували з’єднання шкідливого вузла, який намагався надсилати команди до теми керування обміну повідомленнями. Захист реалізовано за рахунок взаємної перевірки сертифікатів на транспортному рівні. Сервер вимагав від клієнта дійсного сертифіката, підписаного довіреним кореневим центром сертифікації. Відсутність такого підпису призводила до негайного розриву сеансу передавання, що підтверджує коректність ідентифікації пристроїв і недопущення підміни на етапі встановлення каналу зв’язку [3].

Третій етап випробувань був присвячений перевірці стійкості архітектури до навмисного перевантаження каналів і сервісів, що призводить до відмови в обслуговуванні. Такі дії особливо небезпечні для серверних застосунків через потенційне переповнення купи пам’яті та блокування пулу потоків, тому метою було встановити пороги, за яких система зберігає працездатність і контрольовані характеристики відгуку [6].

Синтетичне навантаження нарощували поетапно від ста до десяти тисяч одночасних звернень, моделюючи як короткі імпульсні сплески, так і тривалі хвилі запитів. У процесі спостерігали за станом віртуальної машини та за діями

збирача сміття, оцінювали частоту колекцій, тривалість зупинок і повернення пам’яті до стабільного рівня. Окремо контролювали заповнення черг, кількість активних потоків та поведінку механізмів повернення ресурсів після завершення сесій, щоб виключити накопичення неочищених об’єктів.

Результати показали, що засоби керування пам’яттю ефективно усувають короткочасні об’єкти автентифікації та сесій і не допускають затяжного виснаження ресурсів. Використання асинхронних сервісів дало змогу уникнути ситуації, коли запити очікують безкінечно через нестачу вільних потоків, а обмеження черг зупинило подальше наростання навантаження на канали введення і виведення. За рівня понад вісім тисяч одночасних звернень фіксувалося зростання часу відповіді до двох секунд, проте система продовжувала обробляти запити та коректно повертала коди помилок для надлишкових сесій, що свідчить про стримування атаки без повного краху сервісів.

Додатково було перевірено стійкість до повільних з’єднань, коли клієнт навмисно утримує канал відкритим і надсилає дані дуже повільно, блокуючи серверні ресурси. Завдяки налаштованим годинам очікування під час встановлення та підтримання з’єднання сервер автоматично закривав підозрілі сесії, звільняючи ресурси для доброчесних користувачів. Така поведінка відповідає вимозі забезпечувати стійкість сервісів і застосовувати захищені за замовчуванням параметри для транспортного рівня та керування з’єднаннями [1].

Окремим важливим аспектом тестування була перевірка логічної безпеки додатка, зокрема захисту від ін’єкцій та некоректного доступу до об’єктів (IDOR). Використання автоматизованого сканера OWASP ZAP дозволило підтвердити, що використання Spring Data JPA повністю захищає систему від SQL-ін’єкцій, оскільки всі запити до PostgreSQL є параметризованими на рівні драйвера. При перевірці вразливостей типу Broken Object Level Authorization

(BOLA) було імітовано ситуацію, коли користувач із обмеженими правами намагався отримати доступ до ресурсів іншого домогосподарства, змінюючи числовий ідентифікатор у GET-запиті. Впроваджений механізм перевірки власності активу на сервісному рівні Java-хабу (cross-check ідентифікатора користувача з токена та власника пристрою в БД) успішно блокував такі запити з поверненням коду помилки HTTP 403 Forbidden.

Завершальний етап випробувань був спрямований на перевірку спроможності системи не лише фіксувати порушення, а й надавати повну картину подій для подальшого розслідування. Оцінювали повноту і якість журналів аудиту, коректність часових міток і узгодженість часу на всіх вузлах, наявність однозначних ідентифікаторів подій, а також безперервність ланцюжка доставлення записів у центральне сховище. Окрему увагу приділяли незмінності журналів за рахунок криптографічного підтвердження цілісності, політикам зберігання та наявності достатнього контексту для відтворення ходу інциденту відповідно до вимог спостережливості в споживчому Інтернеті речей [1].

Перевірка показала, що кожна імітована атака була зафіксована у системних журналах з точним зазначенням часу, джерела та типу порушення, супроводжувалася доречним сповіщенням і коректною ескалацією за рівнем важливості. Це підтверджує, що система забезпечує не лише протидію, а й належний нагляд за подіями для швидкого виявлення та документування інцидентів за практиками журналювання і моніторингу, рекомендованими для екосистем Інтернету речей [4].

Отримані результати дають підстави для наступного висновку: обраний пакет технологій на основі мови програмування Java та платформи Spring Boot у поєднанні з продуманою архітектурою захисту забезпечує високу стійкість розумного будинку до актуальних кіберзагроз, відповідає міжнародним вимогам і готовий до експлуатації в умовах підвищеного ризику.

Для сценаріїв з жорсткішими вимогами до безперервності та безпеки доречно доповнити рішення поглибленою сегментацією мережі та міжзонними політиками згідно з підходами для керування промисловими й критичними середовищами [8].

3.3. Кількісна оцінка витрат на впровадження механізмів шифрування та безпеки

Впровадження комплексних механізмів кібербезпеки — шифрування трафіку, взаємної автентифікації пристроїв та перевірки цифрових підписів маркерів доступу — неминуче збільшує споживання обчислювальних ресурсів і додає затримок під час передавання даних. Для систем реального часу на кшталт «розумного будинку» кількісна оцінка такої додаткової вартості є критично важливою, адже саме вона визначає стабільність роботи сервісів і комфорт користувача в повсякденних сценаріях керування. Метою цього етапу дослідження є вимірювання зниження продуктивності центрального вузла на Java після ввімкнення різних рівнів захисту та порівняння результатів із базовою конфігурацією без засобів шифрування та перевірок доступу.

Методика оцінювання спиралася на три основні показники. По-перше, час установлення з’єднання, який відбиває затрати на криптографічні протоколи під час початку сеансу та дає уявлення про чутливість інтерфейсів керування до затримок. По-друге, обсяг мережевого трафіку, що враховує службові дані, додані захистом, - сертифікати, параметри узгодження, службові поля, - і безпосередньо впливає на пропускну здатність каналу в умовах домашніх мереж. По-третє, використання центрального процесора та оперативної пам’яті, адже криптографічні операції, обробка правил доступу і зберігання службових структур для захищених сеансів збільшують навантаження на обчислювальні ресурси. Таке вимірювання узгоджується з вимогами до захищеного зв’язку, керованої конфігурації, сформульованими у

профілях для Інтернету речей: у положеннях щодо безпечної телеметрії, захисту даних і мінімізації відкритих інтерфейсів, а також у каталогах вимог до ідентифікації компонентів, журналювання подій і контролю доступу до сервісів [1].

Перша серія експериментів була присвячена аналізу затримок при встановленні захищеного зв’язку за протоколом MQTTS порівняно зі стандартним MQTT без шифрування. Результати показали, що етап TLS 1.3 Handshake додає в середньому від 120 до 180 мілісекунд до загального часу встановлення сесії. Це зростання зумовлене необхідністю обміну сертифікатами та виконання асиметричних криптографічних операцій на базі еліптичних кривих [38]. Однак, після встановлення з’єднання, затримка при передачі окремих команд (Publish/Subscribe) зросла лише на 5-8%, що пояснюється ефективністю апаратного прискорення AES-NI у сучасних процесорах, яке використовується віртуальною машиною Java. Таким чином, основне навантаження припадає на початковий етап сесії, що обґрунтовує доцільність використання механізму «Keep-Alive» для підтримання тривалих з’єднань та уникнення частого перепідключення пристроїв [12].

Другим аспектом аналізу було оцінювання мережевого оверхеду. Використання TLS-шифрування та додавання JWT-токенів у заголовки кожного REST-запиту збільшує обсяг корисного навантаження. Вимірювання за допомогою аналізатора трафіку показали, що середній розмір заголовка запиту з JWT-токеном (підписаним алгоритмом RS256) зростає приблизно на 600-800 байт. Для стандартних Ethernet або Wi-Fi мереж такий приріст є несуттєвим, проте для енергоефективних мереж із низькою пропускною здатністю (наприклад, LoRaWAN або Zigbee) це може стати критичним фактором. На рівні MQTT оверхед TLS-пакетів склав близько 25-30 байт на кожне повідомлення за рахунок додавання ініціалізаційних векторів та тегів автентифікації GCM. Загалом, мережеві витрати визнані прийнятними для

домашньої інфраструктури, враховуючи значне підвищення рівня безпеки [9, p. 83].

Аналіз споживання системних ресурсів проводився за допомогою інструментів Java Flight Recorder (JFR) та VisualVM. Було встановлено, що активація Spring Security та механізмів перевірки JWT збільшує навантаження на CPU на 12-15% під час пікових навантажень (понад 100 запитів за секунду). Це зростання пов’язане з інтенсивними обчисленнями при перевірці криптографічних підписів. Споживання оперативної пам’яті (Heap memory) зросло на 45-60 МБ через необхідність кешування сертифікатів, публічних ключів та підтримання додаткових структур даних для TLS-сесій [10]. Важливо зазначити, що використання асинхронної обробки у Spring Boot дозволило нівелювати вплив затримок вводу-виводу на загальну пропускну здатність системи, зберігаючи стабільний час відгуку навіть при інтенсивному потоці телеметрії [8, p. 4].

Підсумовуючи результати кількісної оцінки, можна стверджувати, що впроваджені механізми захисту створюють лише помірне навантаження на систему й не перешкоджають її нормальному функціонуванню на базі сучасного апаратного забезпечення.

Експерименти на одноплатних комп’ютерах класу Raspberry Pi 4 та на промислових контролерах показали, що додаткова вартість захисту переважно проявляється як невелике зростання затримок обробки запитів та помірне використання процесорного часу під час встановлення сесій, тоді як у сталому режимі роботи overhead шифрування для потоку даних майже непомітний. На це впливають дві технічні обставини: по-перше, використання Java 21 з актуальними оптимізаціями JIT/ГС та бібліотек безпеки, по-друге, сучасні 64-бітні процесори хабів, які ефективно виконують обчислення, пов’язані з TLS 1.3 та AES-GCM.

У підсумку система стабільно утримує прийнятні значення пропускної здатності, часу відповіді та споживання пам’яті в сценаріях, що моделюють типове навантаження «розумного будинку» (телеметрія датчиків, керування виконавчими пристроями, обробка подій).

Отримані дані підтверджують робочу гіпотезу: комбінація Java 21 і оптимізованих криптографічних протоколів (TLS 1.3, AES-GCM з 256-бітним ключем) дозволяє досягти високого рівня кібербезпеки за збереження прийнятної продуктивності. Це узгоджується з вимогами до захищеного зв’язку, керування конфігурацією, спостережності та мінімізації поверхні атаки, що їх висувають норми для IoT-систем, зокрема положення ETSI EN 303 645 щодо безпечної телеметрії та обробки даних і каталоги вимог NIST SP 800-213/213A щодо журналювання, ідентифікації та контролю доступу до інтерфейсів [1].

Практичний ефект проявляється в тому, що навіть за активного обміну подіями між хмарою/шлюзом та мобільними клієнтами система зберігає стабільний часовий профіль; у режимах пікового навантаження критичні функції (автентифікація, авторизація, застосування політик доступу, шифрування трафіку) не стають «вузьким місцем», а деградація продуктивності має плавний, контрольований характер.

Таким чином, результати експериментів слугують науковим обґрунтуванням практичної придатності розробленої системи до експлуатації в реальних умовах «розумного будинку» і створюють запас міцності для подальшого масштабування. З огляду на вимоги суміжних доменів (наприклад, локальні шлюзи в IIoT/OT-середовищах), отримані показники свідчать, що обрана архітектура та стек безпеки можуть бути поширені на промислові об’єкти Інтернету речей за умови належної сегментації мереж, централізованого журналювання та керування оновленнями відповідно до профілів безпеки [1; 4–5].

Програмний код застосунка, створеного в межах виконання роботи, доступний у повному обсязі за посиланням, наведеним у додатку Д. Всі змінні параметри та конфігурація для його коректного запуску разом з інструкцією відображені у додатку Е.

ВИСНОВКИ

При виконанні кваліфікаційної роботи було проведено науково-прикладне дослідження, спрямоване на вирішення актуальної проблеми підвищення рівня безпеки в архітектурі управління розумним будинком. За результатами виконання роботи було повністю підтверджено доцільність використання програмного забезпечення з відкритим кодом та, зокрема, стеку Java-технологій, як альтернативи закритим рішенням від виробників подібних систем.

Теоретичний аналіз стану предметної області показав, що існуючі рішення часто нехтують питаннями цілісності даних на транспортному рівні, що робить їх вразливими до широкого спектру атак - від простого перехоплення трафіку до складних маніпуляцій з логікою управління. Вивчення різноманітних стандартів безпеки та аналіз можливих вразливостей дозволили сформувати основу для побудови захищеної екосистеми, в якій центральний хаб є шлюзом з функціями активної фільтрації загроз та контролю авторизації кожного інформаційного потоку.

Наукова новизна отриманих результатів полягає у розробці та впровадженні комплексної моделі захищеної взаємодії, яка поєднує безсесійну автентифікацію на рівні користувача з механізмами взаємної перевірки сертифікатів на рівні периферійних пристроїв.

Системно-аналітичний етап роботи дозволив не лише ідентифікувати потенційні загрози за методологією STRIDE, а й математично обґрунтувати пріоритетність впровадження криптографічних засобів захисту через оцінку ризиків безпеки з використанням методології DREAD.

Використання фреймворку Spring Boot в свою чергу забезпечило необхідний рівень абстракції та модульності, що дозволило інтегрувати складні

алгоритми шифрування та рольового доступу без втрати продуктивності системи.

Практична реалізація ключових модулів захищеного хабу продемонструвала високу ефективність програмного втілення теоретичних напрацювань. Розроблені сервіси автентифікації та захищеної комунікації MQTT успішно пройшли етап тестування на проникнення, де було імітовано критичні сценарії атак, включаючи спроби підвищення привілеїв, брутфорс-атаки та маніпуляції з мережевими пакетами. Результати експериментів підтвердили стійкість системи до найбільш розповсюджених вразливостей, а кількісна оцінка накладних витрат дозволила встановити оптимальні параметри конфігурації JVM та криптографічних засобів для мінімізації затримок. Було доведено, що використання апаратного прискорення та асинхронної обробки подій у Java дозволяє ефективно масштабувати систему захисту без значного навантаження на обчислювальні ресурси хабу.

Практична цінність магістерської роботи полягає у створенні цілісного, прототипу фреймворку для розробки захищених систем управління об'єктами інфраструктури та житлової автоматизації з використанням виключно програмного забезпечення з відкритим кодом, що дозволяє мінімізувати накладні витрати на кінцевий продукт.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лук'яненко С.В. Напрями підвищення рівня захищеності системи управління SMART- будинком /С. В. Лук'яненко, І. Д. Погорєловська //ІТ-простір сьогодення: тенденції, інновації та перспективи розвитку: збірник тез доповідей ІІ Міжнародної науково-практичної студентської конференції (15 жовтня 2025 року, м. Харків, Україна) [Електронний ресурс]. - Харків : ХНУ імені В. Н. Каразіна, 2025. - 1 ел. опт. диск (CD-ROM). - Систем. вимоги: Процесор Pentium-класса ; ОС Windows 7/10 ; дисковод CD-ROM; Acrobat Reader 10. С. 88-90.
2. Gubbi J., Buyya R., Marusic S., Palaniswami M. Internet of Things (IoT): A vision, architectural elements, and future directions // *Future Generation Computer Systems*. - 2013. - Vol. 29, No. 7. - P. 1645-1660. - DOI: 10.1016/j.future.2013.01.010.
3. Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M., Ayyash M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications // *IEEE Communications Surveys & Tutorials*. - 2015. - Vol. 17, No. 4. - P. 2347-2376. - DOI: 10.1109/COMST.2015.2444095.
4. Roman R., Zhou J., Lopez J. On the Features and Challenges of Security and Privacy in Distributed Internet of Things // *Computer Networks*. - 2013. - Vol. 57, No. 10. - P. 2266-2279. - DOI: 10.1016/j.comnet.2012.12.018.
5. Zanella A., Bui N., Castellani A., Vangelista L., Zorzi M. Internet of Things for Smart Cities // *IEEE Internet of Things Journal*. - 2014. - Vol. 1, No. 1. - P. 22-32. - DOI: 10.1109/JIOT.2014.2306328.
6. OWASP Foundation. OWASP Internet of Things Project [Електронний ресурс]. - Режим доступу: https://owasp.org/www-project-internet-of-things/
7. NIST. *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks* : NIST IR 8228 [Електронний ресурс]. - Режим доступу: https://nvlpubs.nist.gov/nistpubs/ir/2019/nist.ir.8228.pdf
8. ENISA. Baseline Security Recommendations for IoT [Електронний ресурс]. - Режим доступу: https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot
9. ENISA. Guidelines for Securing the Internet of Things [Електронний ресурс]. - Режим доступу: https://www.enisa.europa.eu/publications/guidelines-for-securing-the-internet-of-things
10. Shostack A. *Threat Modeling: Designing for Security*. - Indianapolis : Wiley, 2014. - 432 p.
11. Myagmar S., Lee A. J., Yurcik W. Threat Modeling as a Basis for Security Requirements [Електронний ресурс]. - Режим доступу: https://people.cs.pitt.edu/~adamlee/pubs/2005/sreis-05.pdf
12. Fernàndez E. B., Washizaki H., Yoshioka N. Modeling Security Patterns Using STRIDE // *IEEE Software*. - 2019. - Vol. 36, No. 3. - P. 68-76. - DOI: 10.1109/MS.2018.2883312.
13. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3 : RFC

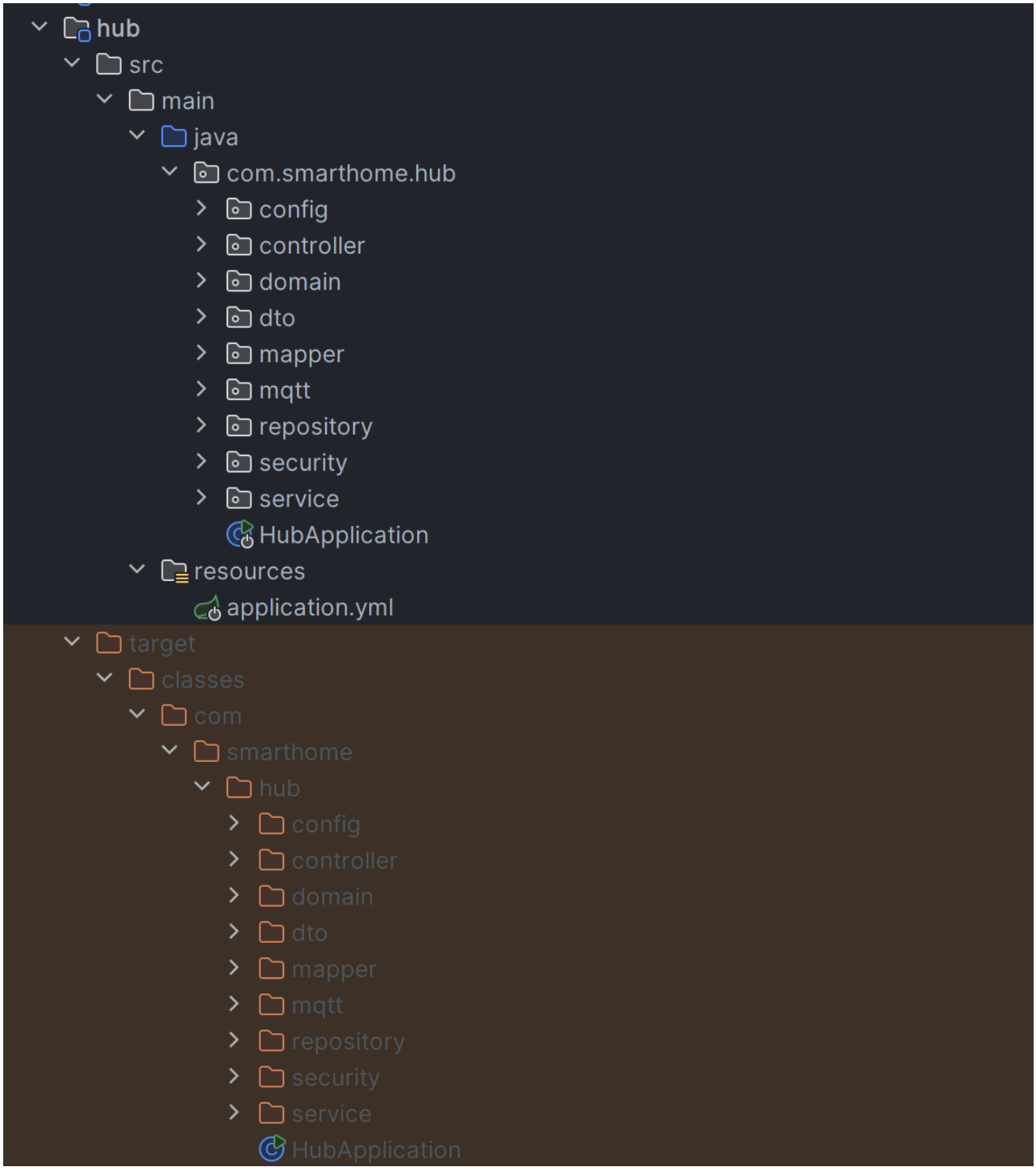
8446 [Електронний ресурс]. - Режим доступу: https://datatracker.ietf.org/doc/html/rfc8446
14. Banks A., Gupta R. MQTT Version 3.1.1 : OASIS Standard [Електронний ресурс]. - Режим доступу: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
15. OASIS Open. MQTT Version 5.0 [Електронний ресурс]. - Режим доступу: https://www.oasis-open.org/standards/mqtt
16. Light R. Mosquitto: Server and client implementation of the MQTT protocol // *Journal of Open Source Software*. - 2017. - Vol. 2, No. 13. - DOI: 10.21105/joss.00265.
17. Stallings W. *Cryptography and Network Security: Principles and Practice*. - 7th ed. - Boston : Pearson, 2017. - 760 p.
18. Anderson R. *Security Engineering*. - 3rd ed. - Wiley, 2020. - 1248 p.
19. Bishop M. *Computer Security: Art and Science*. - 2nd ed. - Addison-Wesley, 2018. - 1200 p.
20. McGraw G. *Software Security: Building Security In*. - Addison-Wesley, 2006. - 336 p.
21. Kleppmann M. *Designing Data-Intensive Applications*. - Sebastopol : O'Reilly, 2017. - 616 p.
22. Bass L., Clements P., Kazman R. *Software Architecture in Practice*. - 3rd ed. - Addison-Wesley, 2012. - 624 p.
23. Richards M. *Software Architecture Patterns*. - Sebastopol : O'Reilly, 2015. - 44 p.
24. Newman S. *Building Microservices*. - Sebastopol : O'Reilly, 2015. - 280 p.
25. Fowler M. *Patterns of Enterprise Application Architecture*. - Boston : Addison-Wesley, 2002. - 560 p.
26. Spring. Spring Boot Reference Documentation [Електронний ресурс]. - Режим доступу: https://docs.spring.io/spring-boot/docs/current/reference/html/
27. Spring. Spring Security Reference Documentation [Електронний ресурс]. - Режим доступу: https://docs.spring.io/spring-security/reference/
28. Bloch J. *Effective Java*. - 3rd ed. - Boston : Addison-Wesley, 2018. - 416 p.
29. Walls C. *Spring in Action*. - 6th ed. - Manning, 2022. - 520 p.
30. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519 [Електронний ресурс]. - Режим доступу: https://datatracker.ietf.org/doc/rfc7519/
31. Hardt D. The OAuth 2.0 Authorization Framework : RFC 6749 [Електронний ресурс]. - Режим доступу: https://datatracker.ietf.org/doc/html/rfc6749
32. IETF. OAuth 2.0 Bearer Token Usage : RFC 6750 [Електронний ресурс]. - Режим доступу: https://datatracker.ietf.org/doc/html/rfc6750
33. Sandhu R. S., Coyne E. J., Feinstein H. L., Youman C. E. Role-Based Access Control Models [Електронний ресурс]. - Режим доступу: https://csrc.nist.gov/csrc/media/projects/role-based-access-control/documents/sandhu96.pdf

34. Ferraiolo D., Kuhn D. Role-Based Access Controls // *15th NIST-NCSC National Computer Security Conference*. - 1992.
35. Hu V. C., Ferraiolo D., Kuhn R. et al. Guide to Attribute Based Access Control (ABAC) Definition and Considerations : NIST SP 800-162 [Електронний ресурс]. - Режим доступу: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-162.pdf
36. OWASP Foundation. OWASP Web Security Testing Guide (WSTG) [Електронний ресурс]. - Режим доступу: https://owasp.org/www-project-web-security-testing-guide/
37. OWASP Foundation. OWASP Top 10 Web Application Security Risks [Електронний ресурс]. - Режим доступу: https://owasp.org/www-project-top-ten/
38. NIST. Guide to Intrusion Detection and Prevention Systems (IDPS) : NIST SP 800-94 [Електронний ресурс]. - Режим доступу: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-94.pdf
39. Microsoft Learn. IoT security (architecture & guidance) [Електронний ресурс]. - Режим доступу: https://learn.microsoft.com/
40. Google Cloud Architecture Center. IoT security guidance [Електронний ресурс]. - Режим доступу: https://cloud.google.com/architecture
41. Стандартизація. Інформаційні технології. Кібербезпека. Основні терміни та визначення понять [Електронний ресурс]. - Режим доступу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=83590
42. ETSI EN 303 645 V2.1.1 (2020-06). Cyber Security for Consumer Internet of Things: Baseline Requirements [Електронний ресурс]. - Sophia Antipolis : European Telecommunications Standards Institute, 2020. - Режим доступу: https://www.etsi.org/deliver/etsi_en/303600_303699/303645/02.01.01_60/en_303645v020101p.pdf
43. IEC/ISA 62443**.** Industrial communication networks - Network and system security [Електронний ресурс]. - International Electrotechnical Commission ; International Society of Automation. - Режим доступу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=107733

ДОДАТКИ

Додаток А

Структура проекту



Додаток Б

Зміст Docker файлу

```
FROM maven:3.9-eclipse-temurin-21 AS build
WORKDIR /app

COPY pom.xml .
COPY hub/pom.xml ./hub/
COPY device-simulator/pom.xml ./device-simulator/

RUN mvn dependency:go-offline -f pom.xml -pl hub -am

COPY hub/src ./hub/src

RUN mvn clean package -f pom.xml -pl hub -am -DskipTests

FROM eclipse-temurin:21-jre-alpine
WORKDIR /app


RUN apk add --no-cache wget


COPY --from=build /app/hub/target/hub-*.jar ./app.jar


RUN [ -f app-original.jar ] && rm app-original.jar || true

# Налаштування порту
EXPOSE 8080

HEALTHCHECK --interval=30s --timeout=3s --start-period=40s --retries=3 \
  CMD wget --no-verbose --tries=1 --spider http://localhost:8080/actuator/health || exit 1


ENTRYPOINT ["java", "-XX:+UseContainerSupport", "-Djava.security.egd=file:/dev/./urandom", "-jar", "app.jar"]
```

Додаток В

Конфігурація безпеки хабу

```java
package com.smarthome.hub.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.convert.converter.Converter;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AbstractAuthenticationToken;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.oauth2.jwt.Jwt;
import org.springframework.security.oauth2.server.resource.authentication.JwtAuthenticationConverter;
import org.springframework.security.oauth2.server.resource.authentication.JwtGrantedAuthoritiesConverter;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.Stream;

@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    private final RestAuthenticationEntryPoint authenticationEntryPoint;
    private final RestAccessDeniedHandler accessDeniedHandler;

    public SecurityConfig(RestAuthenticationEntryPoint authenticationEntryPoint,
                          RestAccessDeniedHandler accessDeniedHandler) {
        this.authenticationEntryPoint = authenticationEntryPoint;
        this.accessDeniedHandler = accessDeniedHandler;
    }

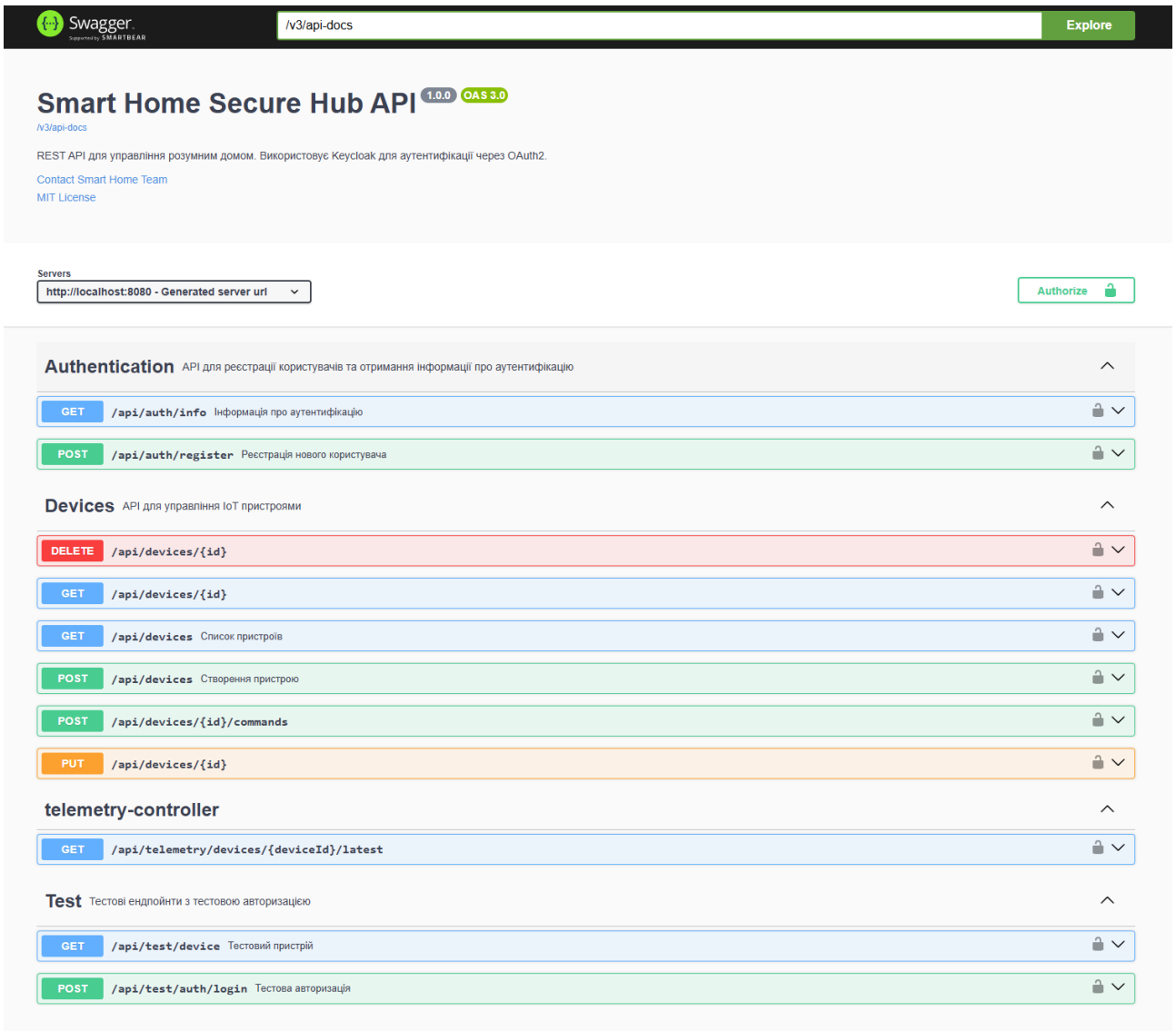
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .csrf(csrf -> csrf.disable())
                .sessionManagement(sm -> sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .exceptionHandling(ex -> ex
                        .authenticationEntryPoint(authenticationEntryPoint)
                        .accessDeniedHandler(accessDeniedHandler))
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/actuator/health").permitAll()
                        .requestMatchers("/api/auth/**").permitAll() // Keycloak handles auth
                        .anyRequest().authenticated()
                )
                .oauth2ResourceServer(oauth2 -> oauth2
                        .jwt(jwt -> jwt
                                .jwtAuthenticationConverter(jwtAuthenticationConverter())
                        )
                );
        return http.build();
    }
```

Продовження додатку В

```
58
59      @Bean
60      public Converter<Jwt, AbstractAuthenticationToken> jwtAuthenticationConverter() {
61          JwtAuthenticationConverter converter = new JwtAuthenticationConverter();
62          converter.setJwtGrantedAuthoritiesConverter(jwtGrantedAuthoritiesConverter());
63          return converter;
64      }
65
66      @Bean
67      public Converter<Jwt, Collection<GrantedAuthority>> jwtGrantedAuthoritiesConverter() {
68          JwtGrantedAuthoritiesConverter defaultConverter = new JwtGrantedAuthoritiesConverter();
69          defaultConverter.setAuthorityPrefix("ROLE_");
70          defaultConverter.setAuthoritiesClaimName("realm_access.roles");
71
72          return jwt -> {
73              Collection<GrantedAuthority> authorities = defaultConverter.convert(jwt);
74
75              // Also extract roles from resource_access if needed
76              Map<String, Object> resourceAccess = jwt.getClaim("resource_access");
77              if (resourceAccess != null) {
78                  @SuppressWarnings("unchecked")
79                  Map<String, Object> clientAccess = (Map<String, Object>) resourceAccess.get("smart-home-hub");
80                  if (clientAccess != null) {
81                      @SuppressWarnings("unchecked")
82                      List<String> roles = (List<String>) clientAccess.get("roles");
83                      if (roles != null) {
84                          Collection<GrantedAuthority> clientRoles = roles.stream()
85                                  .map(role -> new SimpleGrantedAuthority("ROLE_" + role.toUpperCase()))
86                                  .collect(Collectors.toList());
87                          return Stream.concat(authorities.stream(), clientRoles.stream())
88                                  .collect(Collectors.toList());
89                      }
90                  }
91              }
92
93              return authorities;
94          };
95      }
96  }
```

Додаток Г

Специфікація хабу



Додаток Д

Посилання на репозиторій

https://github.com/deolexx/smart-home-secure

Додаток Е

Інструкція з налаштування проекту

Запуск через Docker (рекомендовано)

```bash
# Запуск всього стеку
docker-compose up -d

# Перегляд логів
docker-compose logs -f

# Зупинка
docker-compose down
```

Це запустить:

- **Keycloak** на порту 8090 (IAM сервер)
- **PostgreSQL Database** на порту 5432
- **Mosquitto MQTT Broker** на портах 1883 (без TLS) та 8883 (з TLS)
- **Hub** (Spring Boot API) на порту 8080

- **Device Simulator** - автоматично підключається до MQTT

Запуск локально (без Docker)

Збірка:

```bash
mvn clean package -DskipTests
```

Запуск Hub:

```bash
cd hub
mvn spring-boot:run
```

Запуск Device Simulator:

```bash
cd device-simulator
java -jar target/device-simulator-1.0.0-jar-with-dependencies.jar
```

Запуск Mosquitto:

```bash

cd deploy/mosquitto

docker compose up -d

```

Keycloak Setup

Швидкий старт

Keycloak автоматично запускається з усім стеком. Після запуску:

1. **Відкрийте Admin Console**: http://localhost:8090

2. **Увійдіть**: Username: `admin`, Password: `admin123!`

3. **Створіть Realm**: `smarthome`

4. **Створіть Client**: `smart-home-hub`

5. **Створіть Roles**: `ADMIN`, `USER`, `DEVICE`

Продовження додатку Е

6. **Створіть користувачів** та призначте ролі

Детальні інструкції див. KEYCLOAK-SETUP.md

Отримання токену

```bash
curl -X POST http://localhost:8090/realms/smarthome/protocol/openid-connect/token \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "username=admin" \
  -d "password=admin123!" \
  -d "grant_type=password" \
  -d "client_id=smart-home-hub" \
  -d "client_secret=YOUR_CLIENT_SECRET"
```

Використання токену

```bash
curl -X GET http://localhost:8080/api/devices \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"
```

Docker Setup

Швидкий старт

```bash
docker-compose up -d
```

Перегляд логів

```bash
# Всі сервіси
```

docker-compose logs -f

Конкретний сервіс

docker-compose logs -f hub

docker-compose logs -f device-simulator

docker-compose logs -f mosquitto

docker-compose logs -f postgres

```

### Зупинка

```bash

docker-compose down

```

Перебудова після змін коду

```bash

docker-compose up -d --build

### Окремі сервіси

**Запуск тільки Hub:**

```bash
docker-compose up -d postgres mosquitto hub
```

Запуск тільки Device Simulator:

```bash
docker-compose up -d mosquitto device-simulator
```

Доступ до сервісів

- **Hub API**: http://localhost:8080
- **Hub Health Check**: http://localhost:8080/actuator/health
- **Keycloak Admin Console**: http://localhost:8090 (admin/admin123!)

- **PostgreSQL**: localhost:5432 (user: smarthome, password: smarthome123, database: smarthome)

- **Mosquitto MQTT**: tcp://localhost:1883 (dev) або ssl://localhost:8883 (prod)

Змінні середовища

Hub:

```yaml
hub:
  environment:
    - SPRING_PROFILES_ACTIVE=prod
    - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/smarthome
    - SPRING_DATASOURCE_USERNAME=smarthome
    - SPRING_DATASOURCE_PASSWORD=smarthome123
    - MQTT_BROKER_URL=ssl://mosquitto:8883
    - MQTT_TLS_ENABLED=true
```

Device Simulator:

```yaml
device-simulator:
  environment:
    - SIM_BROKER_URL=tcp://mosquitto:1883
    - SIM_CLIENT_ID=my-device-id
```

Розробка

Перебудова одного сервісу:

```bash
# Тільки hub
docker-compose build hub
docker-compose up -d hub

# Тільки device-simulator
docker-compose build device-simulator
docker-compose up -d device-simulator
```

Запуск без кешування:

```bash

docker-compose build --no-cache

docker-compose up -d

```

Структура Docker файлів

- `docker-compose.yml` - головний файл для запуску всього стеку

- `hub/Dockerfile` - збірка Hub модуля

- `device-simulator/Dockerfile` - збірка Device Simulator модуля

- `.dockerignore` - файли, які ігноруються при збірці

Примітки

- Hub чекає на готовність PostgreSQL та Mosquitto перед запуском

- Device Simulator чекає на готовність Hub перед запуском

- Всі сервіси автоматично перезапускаються при падінні (`restart: unless-stopped`)

- Дані PostgreSQL зберігаються в Docker volume `postgres-data` і не втрачаються при перезапуску контейнерів

- Для видалення всіх даних PostgreSQL: `docker-compose down -v`

Device Simulator

Швидкий старт

Запуск через Docker (рекомендовано):

```bash
docker-compose up -d device-simulator
```

Запуск локально:

```bash
cd device-simulator
mvn clean package
```

Продовження додатку Е

java -jar target/device-simulator-1.0.0-jar-with-dependencies.jar

```

### Конфігурація

Device Simulator налаштовується через змінні середовища:

**Обов'язкові параметри:**

- **`SIM_BROKER_URL`** - URL MQTT брокера

  - Для розробки (без TLS): `tcp://localhost:1883` або `tcp://mosquitto:1883` (в Docker)

  - Для production (з TLS): `ssl://localhost:8883` або `ssl://mosquitto:8883` (в Docker)

**Опціональні параметри:**

- **`SIM_CLIENT_ID`** - Унікальний ID пристрою (за замовчуванням: `device-{UUID}`)

- **`SIM_KEYSTORE_PATH`** - Шлях до keystore для mutual TLS (PKCS12)

- **`SIM_KEYSTORE_PASSWORD`** - Пароль keystore
```

- **`SIM_KEY_PASSWORD`** - Пароль приватного ключа (за замовчуванням = keystore password)

- **`SIM_TRUSTSTORE_PATH`** - Шлях до truststore з CA сертифікатом (JKS)

- **`SIM_TRUSTSTORE_PASSWORD`** - Пароль truststore

Приклади використання

1. Запуск локально без TLS:

```bash

export SIM_BROKER_URL=tcp://localhost:1883

export SIM_CLIENT_ID=my-device-001

java -jar target/device-simulator-1.0.0-jar-with-dependencies.jar

```

2. Запуск з TLS сертифікатами:

```bash

export SIM_BROKER_URL=ssl://localhost:8883

export SIM_CLIENT_ID=my-device-001

export SIM_KEYSTORE_PATH=/path/to/device-keystore.p12
```

export SIM_KEYSTORE_PASSWORD=changeit

export SIM_KEY_PASSWORD=changeit

export SIM_TRUSTSTORE_PATH=/path/to/ca-truststore.jks

export SIM_TRUSTSTORE_PASSWORD=changeit

java -jar target/device-simulator-1.0.0-jar-with-dependencies.jar

```

**3. Запуск кількох пристроїв одночасно:**

```bash

# Пристрій 1

SIM_BROKER_URL=tcp://localhost:1883 SIM_CLIENT_ID=device-001 java -jar app.jar

# Пристрій 2

SIM_BROKER_URL=tcp://localhost:1883 SIM_CLIENT_ID=device-002 java -jar app.jar

```

Перевірка роботи
```

Продовження додатку Е

Перевірка логів:

```bash
docker-compose logs -f device-simulator
```

Перевірка через Hub API:

```bash
# Отримати токен від Keycloak
curl -X POST http://localhost:8090/realms/smarthome/protocol/openid-connect/token \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "username=admin" \
  -d "password=admin123!" \
  -d "grant_type=password" \
  -d "client_id=smart-home-hub" \
  -d "client_secret=YOUR_CLIENT_SECRET"

# Список пристроїв (використайте access_token з попереднього запиту)
```

```
curl -X GET http://localhost:8080/api/devices \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"
```

Перевірка через MQTT клієнт:

```bash
# Підписка на телеметрію всіх пристроїв
mosquitto_sub -h localhost -p 1883 -t "devices/+/telemetry"
```

Надсилання команд до пристрою

Через Hub API:

```bash
curl -X POST http://localhost:8080/api/devices/1/commands \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"action":"turn_on","value":true}'
```