БАКАЛАВРСЬКА РОБОТА

БР.ІП – 57.00.00.000 ПЗ

Група ІП-22-2

Яблінчук Володимир

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Яблінчук Володимир Михайлович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Розробка програмного забезпечення для систем розумного будинку
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Яблінчук В.М._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ Піх Володимир Ярославович, доцент, к.т.н._____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Яблінчук Володимиру Михайловичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Розробка програмного забезпечення для систем розумного будинку "

керівник проекту (роботи) доцент, к.т.н., Піх В.Я.

затверджені наказом вищого навчального закладу від “ 21_ ” _квітня 2026 р. № ___179/7____

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз вимог до програмного забезпечення
2. Аналіз вимог і постановка задачі
3. Проектування системи
4. Реалізація проекту
5. Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 Автоматизація технологій «розумного будинку» (рис. 1.1, ст. 16)

2 Три типи домашніх додатків (рис. 2.1, ст. 36)

3 Блок-схема розробленого програмного забезпечення (рис. 2.5, ст. 46)

4 Архітектура системи управління «розумним будинком» (рис. 3.1, ст. 58)

5 Головний екран інтерфейсу (рис. 3.8, ст. 68)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2026 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	17.01.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.02.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.03.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.05.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	09.06.2026	виконано

Студент _____Яблінчук В.М.
(підпис)

Керівник роботи _____Піх В.Я.
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 88 сторінок, 32 рисунків, 4 таблиці, список використаних джерел із 40 найменувань.

Метою роботи є дослідження принципів та підходів до розробки програмного забезпечення для систем розумного будинку.

Об'єкт дослідження: системи розумного будинку та процеси їх програмної реалізації.

Предмет дослідження: методи, технології та інструменти розробки програмного забезпечення для систем розумного будинку, включаючи архітектурні підходи, а також бездротові технології передачі даних.

Результати дослідження: проведено аналіз концепції розумного будинку, досліджено сучасні підходи до проєктування таких систем, проаналізовано програмні рішення та реалізовано модель системи з урахуванням вимог.

У першому розділі розглянуто теоретичні основи систем розумного будинку, їх структуру, принципи функціонування та контекстно-залежні підходи до автоматизації житлового середовища.

У другому розділі досліджено методи та інструменти реалізації систем розумного будинку, включаючи сценарії використання, типове обладнання, датчики, виконавчі механізми та бездротові технології зв'язку.

У третьому розділі розглянуто реалізацію програмного забезпечення для системи розумного будинку, описано архітектуру системи, проведено аналіз існуючих програмних рішень і виконано тестування.

Висновок: використання сучасних підходів до розробки програмного забезпечення для систем розумного будинку дозволяє підвищити рівень автоматизації, енергоефективність та безпеку житлових приміщень, а також забезпечити гнучкість і масштабованість таких систем.

КЛЮЧОВІ СЛОВА: РОЗУМНИЙ БУДИНОК, ІНТЕРНЕТ РЕЧЕЙ, АВТОМАТИЗАЦІЯ, ДАТЧИКИ, ВИКОНАВЧІ МЕХАНІЗМИ, БЕЗДРОТОВІ ТЕХНОЛОГІЇ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, IoT, АРХІТЕКТУРА СИСТЕМ.

ANNOTATION

The bachelor's thesis contains 88 pages, 32 figures, 4 tables, and a reference list with 40 sources.

The aim of the work is to study the principles and approaches to software development for smart home systems.

Research object: smart home systems and the processes of their software implementation.

Research subject: methods, technologies, and tools for developing software for smart home systems, including architectural approaches.

Research results: the concept of smart home systems was analyzed, modern design approaches were studied, hardware components were reviewed, software solutions were examined, and a system model was implemented considering modern automation requirements.

The first section describes the theoretical foundations of smart home systems, their structure, operating principles, and context-aware approaches to home automation.

The second section analyzes methods and tools for implementing smart home systems, including usage scenarios, typical equipment, sensors, actuators, and wireless communication technologies.

The third section covers the implementation of software for smart home systems, including system architecture, analysis of existing software solutions, and testing using design patterns.

Conclusion: the use of modern approaches to software development for smart home systems improves automation levels, energy efficiency, and security of residential environments, while ensuring flexibility and scalability of such systems.

KEY WORDS: SMART HOME, INTERNET OF THINGS, AUTOMATION, SENSORS, ACTUATORS, WIRELESS TECHNOLOGIES, SOFTWARE, IoT, SYSTEM ARCHITECTURE.

ЗМІСТ

ВСТУП

У сучасному світі цифрові технології стрімко проникають у всі сфери життя людини, забезпечуючи підвищення комфорту, безпеки та ефективності використання ресурсів. Одним із ключових напрямів розвитку є концепція розумного будинку, що передбачає інтеграцію інформаційних технологій, сенсорних пристроїв і систем автоматизації для управління житловим середовищем. Зростання кількості підключених пристроїв, розвиток Інтернету речей (IoT) та необхідність оптимізації енергоспоживання формують попит на сучасні програмні рішення, здатні забезпечити централізоване управління всіма компонентами будинку. У цьому контексті особливого значення набуває розробка програмного забезпечення, яке відповідає вимогам надійності, масштабованості та зручності використання.

Актуальність теми зумовлена необхідністю створення ефективних систем автоматизації житлових приміщень, які дозволяють інтегрувати різноманітні пристрої — від освітлення та клімат-контролю до систем безпеки — в єдину керовану екосистему. Існуючі рішення часто мають обмежену сумісність, складність налаштування або недостатній рівень захисту даних, що ускладнює їх широке впровадження. У зв’язку з цим виникає потреба у розробці універсальних програмних платформ, здатних забезпечити гнучке управління, адаптацію до потреб користувача та інтеграцію з сучасними бездротовими технологіями.

Метою роботи є дослідження та розробка програмного забезпечення для систем розумного будинку, яке забезпечує ефективне управління пристроями, автоматизацію процесів та підвищення рівня комфорту і безпеки користувачів.

Завданнями дослідження є аналіз предметної області та існуючих рішень у сфері розумних будинків, визначення функціональних і нефункціональних вимог до системи, дослідження апаратних компонентів (датчиків, виконавчих механізмів, контролерів) і бездротових технологій зв’язку, проектування архітектури програмного забезпечення, реалізація основних функціональних модулів, тестування системи та оцінка її ефективності.

Об’єктом дослідження є процеси розробки програмного забезпечення для систем розумного будинку, що включають аналіз вимог, проектування, реалізацію, тестування та впровадження систем автоматизації.

Предметом дослідження є методи, технології та інструменти розробки програмного забезпечення для систем розумного будинку, зокрема архітектурні підходи, принципи інтеграції апаратних і програмних компонентів, використання датчиків, виконавчих механізмів і бездротових протоколів передачі даних.

Методи дослідження включають аналіз наукових і технічних джерел для вивчення сучасних підходів до побудови систем розумного будинку, порівняльний аналіз технологій, моделювання архітектури системи, експериментальні методи для реалізації та тестування програмного забезпечення, а також оцінку ефективності запропонованих рішень.

Розроблене програмне забезпечення передбачатиме можливість централізованого управління пристроями, автоматизації типових сценаріїв (освітлення, клімат-контроль, безпека), збору та обробки даних із датчиків у реальному часі, а також інтеграції з бездротовими мережами. Особлива увага приділятиметься питанням безпеки, енергоефективності та зручності користувацького інтерфейсу, що забезпечить ефективне функціонування системи розумного будинку.

Бакалаврська робота містить 88 сторінок, 32 рисунків, 4 таблиці, 3 розділи, список використаних джерел із 40 найменувань.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ СИСТЕМ РОЗУМНОГО БУДИНКУ

1.1 Основні принципи та поняття систем розумного будинку

За оцінками, найближчим часом близько 90 мільйонів людей у всьому світі житимуть у розумних будинках, використовуючи технології для покращення безпеки, комфорту та споживання енергії вдома [1]. Нещодавнє дослідження показало, що понад кожна четверта людина у Швеції вважає, що вона має недостатні знання та контроль над споживанням енергії, і що чотири з десяти хотіли б бути більш обізнаними та краще контролювати своє споживання енергії [2]. Рішенням є надання домовласникам зворотного зв'язку щодо споживання енергії, наприклад, за допомогою системи автоматизації розумного дому [3]. Дослідження показали, що домовласники можуть скоротити споживання енергії до 20%, отримуючи такий зворотний зв'язок [4]. Автоматизація розумного дому є яскравим прикладом розумного середовища, побудованого на різних типах кіберфізичних систем, що генерують обсяги різноманітних, неоднорідних, складних та розподілених даних з безлічі програм та датчиків. Таким чином, така домашня автоматизація також є прикладом сценарію Інтернету речей (IoT), де комунікаційна мережа розширює сучасний Інтернет, включаючи предмети повсякденного вжитку та датчики, що в цьому випадку включає можливість моніторингу та управління споживанням енергії [5]. Таким чином, системи автоматизації розумного дому включають загальні пристрої, які керують функціями будинку, але вони не лише вмикають та вимикають пристрої [6]. Наприклад, системи автоматизації розумного дому можуть контролювати конфігурацію внутрішнього середовища та дії, які виконуються, коли будинок зайнятий (і вільний). Результатом цих модифікацій технології є те, що система автоматизації розумного дому може автономно керувати пристроями та таким чином керувати будинком від імені кінцевих користувачів, тобто людей.

Автоматизація розумного дому привертає дедалі більше уваги з боку комерційних учасників, таких як постачальники енергії, постачальники інфраструктури та сторонні постачальники програмного та апаратного забезпечення [7]. Серед некомерційних зацікавлених сторін є різні урядові установи та муніципалітети, а також кінцеві користувачі. Знання, інструменти та інфраструктура, пов'язані з програмним забезпеченням і даними, почали розвиватися, щоб вирішити проблеми, спричинені складністю та неоднорідністю масово взаємопов'язаних послуг і пристроїв, але наразі немає усталеної практики для цього.

За оцінками, найближчим часом близько 90 мільйонів людей у всьому світі житимуть у розумних будинках, використовуючи технології для покращення безпеки, комфорту та споживання енергії вдома [8]. Нещодавнє дослідження показало, що понад кожна четверта людина у Швеції вважає, що вона має недостатні знання та контроль над споживанням енергії, і що чотири з десяти хотіли б бути більш обізнаними та краще контролювати своє споживання енергії. Рішенням є надання домовласникам зворотного зв'язку щодо споживання енергії, наприклад, за допомогою системи автоматизації розумного дому. Дослідження показали, що домовласники можуть скоротити споживання енергії до 20%, отримуючи такий зворотний зв'язок [9]. Автоматизація розумного дому є яскравим прикладом розумного середовища, побудованого на різних типах кіберфізичних систем, що генерують обсяги різноманітних, неоднорідних, складних та розподілених даних з безлічі програм та датчиків. Таким чином, така домашня автоматизація також є прикладом сценарію Інтернету речей (IoT), де комунікаційна мережа розширює сучасний Інтернет, включаючи предмети повсякденного вжитку та датчики, що в цьому випадку включає можливість моніторингу та управління споживанням енергії [10]. Таким чином, системи автоматизації розумного дому включають загальні пристрої, які керують функціями будинку, але вони не лише вмикають та вимикають пристрої [11]. Наприклад, системи автоматизації розумного дому можуть контролювати конфігурацію внутрішнього середовища та дії, які виконуються, коли будинок

зайнятий (і вільний). Результатом цих модифікацій технології є те, що система автоматизації розумного дому може автономно керувати пристроями та таким чином керувати будинком від імені кінцевих користувачів, тобто людей.

Автоматизація розумного дому привертає дедалі більше уваги з боку комерційних учасників, таких як постачальники енергії, постачальники інфраструктури та сторонні постачальники програмного та апаратного забезпечення [12]. Серед некомерційних зацікавлених сторін є різні урядові установи та муніципалітети, а також кінцеві користувачі. Знання, інструменти та інфраструктура, пов'язані з програмним забезпеченням і даними, почали розвиватися, щоб вирішити проблеми, спричинені складністю та неоднорідністю масово взаємопов'язаних послуг і пристроїв, але наразі немає усталеної практики для цього.

Протягом багатьох років, коли розумні системи були єдиною темою для обговорення, у групи людей виникла ідея покращити спосіб життя, враховуючи сучасні технології [13]. Вони зрозуміли, що було б чудово, якби будинки мали щось подібне до людського. Наприклад, ми можемо забути вимкнути світло, тому ми заплатимо за це гроші і пошкодуємо про це в кінці місяця, тому їм якимось чином спала на думку ідея створити систему домашньої автоматизації. Системи домашньої автоматизації – це програми, які мають доступ до всіх елементів керування будинком, таких як освітлення, телевізор, кондиціонер, гараж, двері тощо [14]. Системи управління домашньою автоматизацією завжди мають підказки на майбутнє. Світло вмикатиметься при вході в кімнату, вентилятори вмикатимуться, коли температура занадто низька, а члени сім'ї дозволятимуть входити в будинок через захисні екрани, які можуть виявити всіх членів сім'ї [15]. Здебільшого люди думають, що це неможливо, вони кажуть, що це неможливо, але насправді це правда! Спосіб мислення людей відрізняється від нашого, як розробників. Вони вважають, що для керування всім будинком потрібно багато пристроїв, але це не так. Все, що нам потрібно, це один розумний пристрій для встановлення програми, запрограмованої для конкретного будинку, і тоді ви можете запускати цю програму, як забажаєте. Системи домашньої автоматизації,

побудовані таким чином, щоб бути структурованими відповідно до потреб користувача, є доступними [16].

Нещодавні досягнення в домашніх технологіях сприяли швидкому розвитку пристроїв для користувачів розумного дому. Інтернет прискорив доступність різноманітних приладів та пристроїв удома, які можуть автоматизувати та обробляти інформацію для певних послуг, необхідних у середовищі розумного дому. Зокрема, ці пристрої спрямовані на ефективну взаємодію та інтеграцію розумного дому з кіберсередовищем, особливо з Інтернетом. Розумний дім визначається як інтелектуальне середовище, яке здатне отримувати та застосовувати знання про своїх мешканців та їхнє оточення, щоб адаптуватися та досягати цілей комфорту та ефективності [17]. Розумний дім також можна розглядати як доповнене середовище з можливістю пропонувати мешканцям будинку безпрецедентний рівень доступу до інформації та допомоги за допомогою кіберфізичних систем. Нещодавнє розширення послуг поступово перетворює розумні будинки на хмару даних з багатьма пристроями та приладами, налаштованими для певних доменів. Ця зміна призводить до кількох проблем, пов'язаних з адмініструванням та експлуатацією пристроїв у такому середовищі. Перша проблема пов'язана з існуванням кількох пристроїв у домашньому середовищі, що сприяє зростанню шлюзів. У типових домашніх умовах використовуються різні протоколи та стандарти зв'язку. Ці відмінності вимагають налаштування певної кількості шлюзів для підтримки кожного пристрою в домашньому середовищі. Шлюз відіграє значну роль у перетворенні протоколу на інший шляхом зіставлення диверсифікованих точок даних [18]. З диверсифікованими протоколами шлюзам потрібні значні зусилля для ресурсомісткого зіставлення точок даних. Це призводить до додаткових витрат для розробників систем. Розгортання шлюзів для застарілих та нових пристроїв у домашньому середовищі може уповільнити продуктивність пристроїв під час перетворення протоколів. Ще однією проблемою, що зростає, є різниця в ресурсах, операційній платформі та мовах програмування, прийнятих для систем керування будинком. Існує величезна кількість нових пристроїв та приладів від

різних постачальників, яким потрібен певний ступінь сумісності для відповідного виконання своїх завдань. Різнорідні сервіси та пристрої повинні бути сумісними один з одним, щоб спільно виконувати завдання. Ці різнорідні сервіси та пристрої сприяють розширенню геометричного діапазону для систем розумного дому, а також висувають нові вимоги, які можуть відповідати середовищу. Для домашніх систем основним завданням є забезпечення сумісності між зовнішнім середовищем та будинком за допомогою спільного стандарту або рішення. Технології XML та веб-сервісів розглядаються як потенційна відповідь у забезпеченні ступеня сумісності для керування домашніми системами. Протягом останніх років XML та веб-сервіси все частіше розглядаються як рішення для великих розподілених систем. Ці критерії відповідають домашньому середовищу завдяки його диверсифікації послуг, починаючи від мультимедіа і закінчуючи домашнім керуванням. Фундаментальними сутностями веб-сервісів є *публікація, виявлення та виклик.* Ці фундаментальні сутності визначені стандартами веб-сервісів, відомими як Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) та Universal Description, Discovery and Integration (UDDI). Серед цих стандартів SOAP діє як *з'єднувач протоколів* , що дозволяє користувачам та постачальникам послуг «спілкуватися» один з одним та обмінюватися XML-даними для забезпечення зв'язку через Інтернет. SOAP визначає XML-конверт для передачі повідомлень та виклику віддалених процедур між системами стандартним способом, який є незалежним від операційних систем, мов програмування та географічно. Використовуючи технологію SOAP, веб-сервіси можна викликати, перетворюючи виклик сервісу у формат XML-повідомлення. Нещодавно веб-сервіси привернули увагу розробників, коли Open Building Information Xchange (OBiX), ініційований OASIS, створив комплексний стандарт для обміну інформацією між різними підсистемами , що може сприяти розвитку домашньої та будівельної галузей [19]. У цій роботі ми обговоримо впровадження систем управління розумним будинком з використанням технології SOAP. Системне рішення побудовано за допомогою вбудованої системи. Вбудовані системи запускають користувацькі програмні додатки та надають

загальні функції, що нагадують ПК. Ці вбудовані системи ідеально підходять для домашнього середовища завдяки своїм характеристикам та бажаним функціям, таким як:

a) Розширюваність: їхній менший модульний розмір дозволяє розширювати їх до різних домашніх застосувань, таких як точки доступу та системи спостереження, а також до широкого спектру цифрових та аналогових модулів вводу/виводу.

b) Масштабованість: Вбудовані системні блоки мають широкий спектр типів процесорів, що дозволяє легко масштабувати їх для кращої продуктивності для різних домашніх застосувань.

c) Керування живленням: Вимоги багатьох вбудованих систем є достатньо низькими та можуть забезпечити безперебійне живлення 24/7. Це важливий критерій для систем управління будинком, особливо для критично важливих для безпеки підсистем, таких як моніторинг стану, які повинні працювати безперервно.

d) Невидимі обчислення: Вбудовані системи можуть реалізувати перспективу невидимих обчислень у середовищі розумного дому, зробивши її більш реалістичною.

Керування та моніторинг систем управління розумним будинком може здійснюватися за допомогою дротової або бездротової технології. Ми також обговоримо використання альтернативного механізму керування за допомогою GSM у разі простою сервера. Впровадження SOAP забезпечує безперебійний обмін інформацією в системах управління розумним будинком, а також ефективно розподіляється за своєю природою для підтримки керування та моніторингу побутової техніки.

1.2 Концепція розумного будинку та інтелектуальних будівель

Розумний дім або будівля — це будинок або будівля, зазвичай нова, обладнана спеціальною структурованою проводкою, яка дозволяє мешканцям

дистанційно керувати або програмувати низку автоматизованих домашніх електронних пристроїв, вводячи одну команду. Наприклад, домовласник у відпустці може використовувати телефон із тоновим набором для активації системи домашньої безпеки, керування температурними датчиками, вмикання або вимикання приладів, керування освітленням, програмування домашнього кінотеатру або розважальної системи та виконання багатьох інших завдань.

Галузь домашньої автоматизації швидко розширюється завдяки конвергенції електронних технологій. Домашня мережа охоплює системи зв'язку, розваг, безпеки, зручності та інформації.

Технологія, відома як Powerline Carrier Systems (PCS), використовується для надсилання кодованих сигналів по існуючій електропроводці будинку до програмованих вимикачів або розеток. Ці сигнали передають команди, що відповідають «адресам» або розташуванню певних пристроїв, і які контролюють, як і коли ці пристрої працюють. Наприклад, передавач PCS може надсилати сигнал по електропроводці будинку, а приймач, підключений до будь-якої електричної розетки в будинку, може приймати цей сигнал і керувати приладом, до якого він підключений.

Один поширений протокол для PCS відомий як X10, це техніка передачі сигналів для дистанційного керування будь-яким пристроєм, підключеним до лінії електропередач. Сигнали X10, що містять короткі радіочастотні (РЧ) імпульси, що представляють цифрову інформацію, забезпечують зв'язок між передавачами та приймачами.

У Європі технології оснащення будинків інтелектуальними пристроями зосереджені на розробці Європейської інсталяційної шини, або Instabus. Цей вбудований протокол керування для цифрового зв'язку між інтелектуальними пристроями складається з двопровідної шини, яка встановлюється разом зі звичайною електричною проводкою. Лінія Instabus з'єднує всі прилади з децентралізованою системою зв'язку та функціонує як телефонна лінія, через яку можна керувати приладами. Європейська асоціація інсталяційних шин є частиною Konnex, асоціації, яка має на меті стандартизацію домашніх мереж та мереж

будівель у Європі.

Рисунок 1.1 - Автоматизація технологій «розумного будинку»

Корпорація Echelon, творець системи LonWorks, допомагає просувати впровадження відкритого стандарту сумісності серед постачальників у галузі мереж керування. LonWorks – це відкритий стандарт для автоматизації та керування мережами для будівель, транспорту, промисловості та побуту. Американський національний інститут стандартів (ANSI) прийняв протокол, що лежить в основі мереж керування LonWorks, як галузевий стандарт. Асоціація сумісності LonMark складається з понад 200 компаній, що займаються управлінням, які працюють над стандартом для інтеграції багатопостачальникових систем на основі мереж LonWorks. [20]

Програмне забезпечення та технології для розумного дому

Технологія розумного дому була розроблена в 1975 році, коли компанія в Шотландії розробила X10. X10 дозволяє сумісним продуктам взаємодіяти один з одним через уже існуючі електричні дроти будинку. Усі прилади та пристрої є приймачами, а засоби керування системою, такі як пульти дистанційного керування або клавіатури, – передавачами. Якщо ви хочете вимкнути лампу в іншій кімнаті, передавач видасть повідомлення в числовому коді, яке включає наступне:

- Сповіщення для системи про те, що вона видає команду,
- Ідентифікаційний номер пристрою, який має отримати команду, та
- Код, що містить фактичну команду, наприклад, «вимкнути».

Все це розраховано на менш ніж секунду, але X10 має деякі обмеження. Зв'язок по електричних лініях не завжди надійний, оскільки лінії стають «шумними» через живлення інших пристроїв. Пристрій X10 може інтерпретувати електронні перешкоди як команду та реагувати, або ж взагалі не отримувати команду. Хоча пристрої X10 все ще існують, з'явилися інші технології, які конкурують за ваші витрати на домашню мережу.

Замість того, щоб проходити через лінії електропередач, деякі системи використовують для зв'язку радіохвилі, так само працюють сигнали Wi-Fi та мобільного телефону. Однак мережі домашньої автоматизації не потребують усієї потужності мережі Wi-Fi, оскільки команди автоматизації – це короткі повідомлення. Дві найпоширеніші радіомережі в домашній автоматизації – це ZigBee та Z-Wave. Обидві ці технології є mesh- мережею, тобто існує більше одного способу передачі повідомлення до місця призначення.

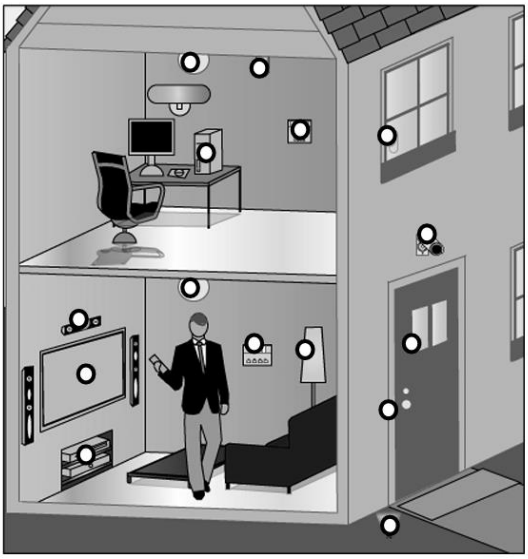

Рисунок 1.2 - Крапки позначають пристрої, які можна підключити до мережі вашого «розумного будинку».

Z-Wave використовує алгоритм маршрутизації джерела для визначення найшвидшого маршруту для повідомлень. Кожен пристрій Z-Wave має вбудований код, і коли пристрій підключено до системи, мережевий контролер розпізнає код, визначає його місцезнаходження та додає його до мережі. Коли

надходить команда, контролер використовує алгоритм, щоб визначити, як повідомлення слід надіслати. Оскільки така маршрутизація може займати багато пам'яті в мережі, Z-Wave розробив ієрархію між пристроями: деякі контролери ініціюють повідомлення, а деякі є "ведомими", що означає, що вони можуть лише передавати повідомлення та відповідати на них.

Назва ZigBee ілюструє концепцію mesh-мережі, оскільки повідомлення від передавача рухаються зигзагами, немов бджоли, шукаючи найкращий шлях до приймача. Хоча Z-Wave використовує власну технологію для роботи своєї системи, платформа ZigBee базується на стандарті, встановленому Інститутом інженерів з електротехніки та електроніки (IEEE) для бездротових персональних мереж. Це означає, що будь-яка компанія може створити продукт, сумісний із ZigBee, не сплачуючи ліцензійних зборів за технологію, що лежить в його основі, що зрештою може дати ZigBee перевагу на ринку. Як і Z-Wave, ZigBee має повністю функціональні пристрої (або ті, що маршрутизують повідомлення) та пристрої з обмеженою функціональністю (або ті, що цього не роблять).

Використання бездротової мережі забезпечує більшу гнучкість у розміщенні пристроїв, але, як і електричні лінії, вони можуть створювати перешкоди. Insteon пропонує спосіб зв'язку вашої домашньої мережі як по електричних проводах, так і по радіохвилях, що робить її подвійною мережею. Якщо повідомлення не проходить на одній платформі, воно спробує на іншій. Замість маршрутизації повідомлення, пристрій Insteon транслюватиме його, і всі пристрої підхоплюють його та транслюють, доки команда не буде виконана. Пристрої діють як вузли, на відміну від того, що один служить ініціатором, а інший — приймачем. Це означає, що чим більше пристроїв Insteon встановлено в мережі, тим потужнішим буде повідомлення.

Налаштування розумного дому

X10, Insteon, ZigBee та Z-Wave лише пропонують технологію для зв'язку в розумному будинку. Виробники уклали альянси з цими системами для створення продуктів, що використовують цю технологію. Ось кілька прикладів продуктів для розумного дому та їхніх функцій.

- Камери відстежуватимуть зовнішній вигляд вашого будинку, навіть якщо надворі зовсім темно.
- Підключіть настільну лампу до димера замість розетки, і ви зможете змінити освітлення та затемнення одним натисканням кнопки.
- Відеодомофон — це більше, ніж просто дверний дзвінок — ви отримуєте уявлення про те, хто знаходиться біля дверей.
- Датчики руху надсилатимуть сповіщення, коли навколо вашого будинку буде рух, і вони навіть може відрізнити домашніх тварин від грабіжників.
- Дверні ручки можна відкривати за допомогою сканованих відбитків пальців або чотиризначного коду, що усуває потрібність пошукати ключі від будинку.
- Аудіосистеми поширюють музику з вашої стереосистеми в будь-якій кімнаті з підключеним динаміками.
- Модулятори каналів приймають будь-який відеосигнал – від камери безпеки до вашого улюблену телевізійну станцію -- і зробити її доступною для перегляду на кожному телевізорі в будинку.
- Пульти дистанційного керування, клавіатури та настільні контролери є засобами активації програми для розумного дому. Пристрої також оснащені вбудованими веб-серверами, які дозволяють вам отримати доступ до своєї інформації онлайн.

Ця клавіатура надсилатиме повідомлення на вашу лампу. Ці продукти можна придбати в магазинах товарів для дому, магазинах електроніки, у технічних спеціалістів або онлайн. Перед покупкою перевірте, яка технологія пов'язана з продуктом. Продукти, що використовують одну й ту саму технологію, повинні працювати разом, незалежно від різних виробників, але для об'єднання продукту X10 та Z-Wave потрібен з'єднувальний пристрій.

Проектуючи розумний дім, ви можете використовувати стільки систем домашньої автоматизації, скільки забажаєте. Ви можете почати зі стартового комплекту освітлення, а потім додати пристрої безпеки. Якщо ви хочете почати з більшої системи, гарною ідеєю буде ретельно продумати, як працюватиме

будинок, особливо якщо знадобиться заміна проводки або ремонт. Крім того, вам потрібно буде стратегічно розмістити вузли бездротових мереж, щоб вони мали хороший радіус дії.

Вартість розумного будинку залежить від того, наскільки розумним є будинок. Один будівельник підрахував, що його клієнти витрачають від 10 000 до 250 000 доларів на складні системи. Якщо будувати розумний будинок поступово, починаючи з базової системи освітлення, це може бути лише кілька сотень доларів. Більш складна система коштуватиме десятки тисяч доларів, а елементи систем домашнього кінотеатру підвищують вартість системи приблизно на 50 відсотків. [21]

Переваги розумного дому

Розумні будинки, очевидно, здатні зробити життя простішим і зручнішим. Домашні мережі також можуть забезпечити душевний спокій. Незалежно від того, чи ви на роботі, чи у відпустці, розумний дім попередить вас про те, що відбувається, а системи безпеки можуть бути побудовані так, щоб забезпечити величезну допомогу в надзвичайних ситуаціях. Наприклад, розумний дім не тільки розбудить мешканця сповіщенням про пожежну тривогу, але й відімкнуть двері, зателефонують до пожежної служби та освітлять шлях до безпеки.

Розумні будинки також забезпечують певну економію енергії. Оскільки такі системи, як Z-Wave та ZigBee, знижують рівень функціональності деяких пристроїв, вони можуть переходити в «сплячий» режим і прокидатися за допомогою команд. Рахунки за електроенергію зменшуються, коли світло автоматично вимикається, коли людина виходить з кімнати, а кімнати можна опалювати або охолоджувати залежно від того, хто знаходиться в кімнаті в будь-який момент. Одна власниця розумного будинку похвалилася, що її рахунок за опалення приблизно на третину менший, ніж у звичайному будинку такого ж розміру. Деякі пристрої можуть відстежувати, скільки енергії споживає кожен прилад, і наказувати йому споживати менше.

Технологія розумного дому обіцяє величезні переваги для літньої людини, яка живе сама. Розумні будинки можуть повідомляти мешканця, коли час

приймати ліки, повідомляти лікарню, якщо мешканець падає, та відстежувати, скільки він їсть. Якщо літня людина трохи забудькувата, розумний будинок виконуватиме такі завдання, як відключення води до переповнення ванни або вимкнення духовки, якщо кухар пішов. Це також дозволяє дорослим дітям, які можуть жити в іншому місці, брати участь у догляді за своїми літніми батьками. Легкі в управлінні автоматизовані системи нададуть аналогічні переваги людям з інвалідністю або обмеженим діапазоном рухів.

1.3 Контекстно-орієнтовані підходи у технологіях розумного будинку

Усвідомлення контексту відіграє велику роль у розробці та підтримці розумного дому. Нижче наведено дослідження та розробки інструментів і технологій, що використовують усвідомлення контексту для створення розумного дому.

Контекстно-чутлива архітектура на основі правил для середовища розумного дому

Систему можна створити для контексту застосунку, який має справу з певною будівлею з певними типами датчиків, і де пропонуються певні послуги. Визначення елементів контексту (таких як місцезнаходження, значення датчиків, час, розклад користувача) та служб, які можуть бути викликані правилом, може бути виконано системним адміністратором за допомогою специфікацій XML.

Архітектура, керована подіями, де різні контексти та сервіси можуть бути зареєстровані стандартним чином, підтримує цю спільність. Фактичні правила, що визначають, які сервіси потрібні користувачеві для заданого контекстного середовища, також можна визначити за допомогою XML, але користувачеві легше їх виконати на мобільному пристрої за допомогою автоматично згенерованого інтерфейсу [22].

Свідома спільнота

«Свідома спільнота» дозволить нам перейти від парадигми усвідомленого та допоміжного дому до розвитку усвідомленої та допоміжної інфраструктури

спільноти шляхом інтеграції пристроїв та методів у невелику міську спільноту будинків, рекреаційних закладів, роздрібної торгівлі та постачальників послуг, на міських вулицях з автомобільним рухом та громадським транспортом.[23]

Громада McKIZ Aware розташована на десятиакровій ділянці в Третьому районі Маккіспорта, як показано на плані ділянки (рисунок 1.3). Розумний котедж/дослідницький центр Blueroof Smart Cottage, громадський центр YWCA, Армія Спасіння та дві діючі церкви розташовані на території McKIZ і залишаться там. Blueroof побудує низку нових споруд, включаючи 15-20 окремих будинків, невеликий продуктовий магазин та нову будівлю для Технічного центру Blueroof. Існує 22 існуючі будинки HUD (п'ять з яких доступні для людей з обмеженими можливостями), які мають інфраструктуру для розміщення датчиків та інших технологій.[

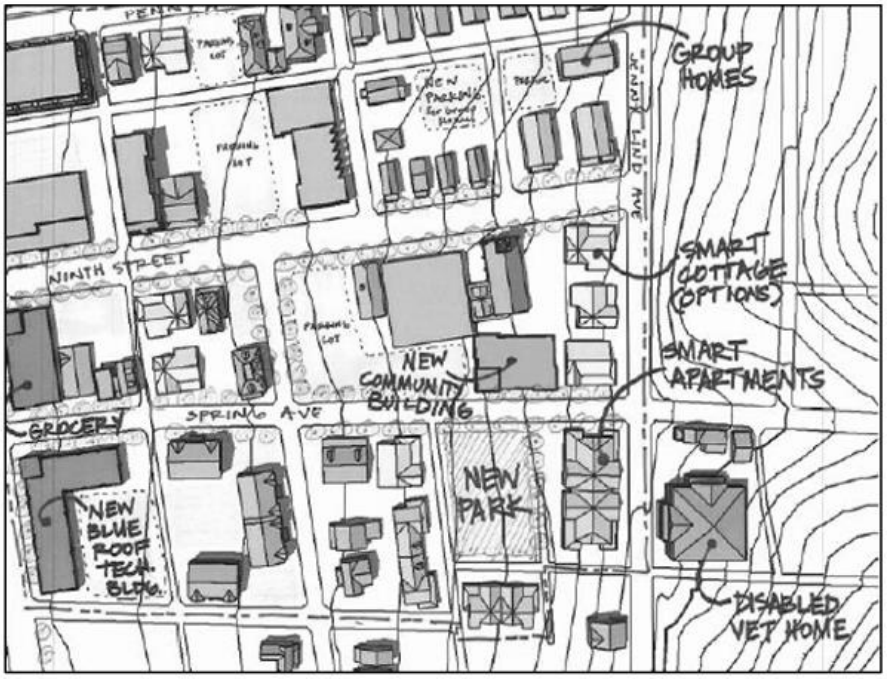

Рисунок 1.3 - План забудови громади

Контекстна усвідомленість на основі CAMUS для поширених домашніх середовищ

CAMUS має на меті забезпечити основу для розробки та виконання контекстно-залежних застосунків для мережевих роботів. URC – це нова

концепція мережевого сервісного робота. Вона дозволяє роботу розширювати свої функції та послуги, використовуючи зовнішні сенсорні мережі та віддалені обчислювальні сервери. В URC CAMUS відіграє важливу роль програмної інфраструктури, яка розширює функції та послуги робота, покращує контекстно-залежне середовище в середовищі повсюдних обчислень та підвищує інтелект робота. [24]

На рисунку 1.4 показано архітектуру системи CAMUS. CAMUS складається з чотирьох частин: CAMUS-MS (головний сервер CAMUS), SAM (менеджер сервісних агентів), SA (сервісний агент) та Planet.

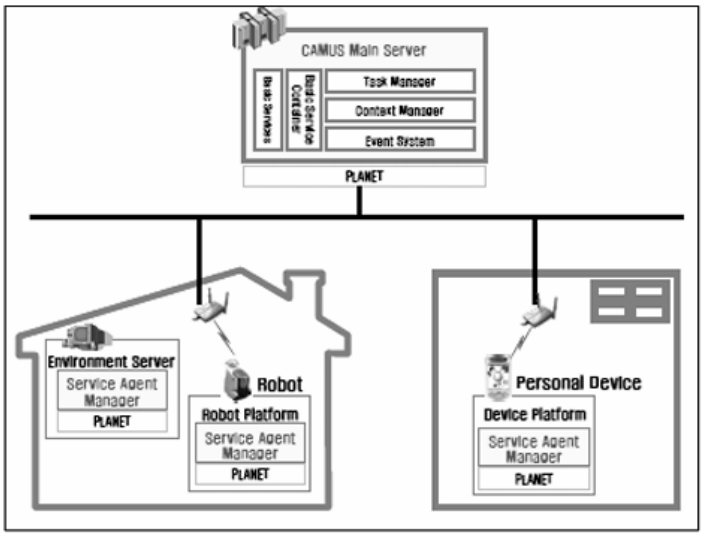

Рисунок 1.4 - Архітектура системи CAMUS

CAMUS-MS – це фреймворк, який збирає контексти з SA та використовує контекстну інформацію. Крім того, він підтримує різноманітні функції для розробки контекстно-залежних додатків. Зокрема, CAMUS-MS контролює всю інформацію про контекст користувача, включаючи вподобання користувача, що використовуються в рекомендаціях щодо контенту та контексти середовища, потім надсилає події про зміни контексту до додатків та допомагає додаткам виконувати відповідні дії для контексту. Є ще один важливий момент для CAMUS-MS: він пропонує сервісну платформу, яка може підключатися до базового сервісного агента контролера робота та різноманітного програмного

забезпечення, такого як розпізнавання голосу, розпізнавання зображень та виявлення руху. У наступному розділі ми детально опишемо компоненти CAMUS-MS. [25]

SAM – це програма, яка керує та контролює SA у середовищі. Для цього SAM встановлюється в різних середовищах у певному місці, отримує інформацію з різних датчиків у середовищі, надсилає цю інформацію до CAMUS-MS, отримує інструкції від CAMUS-MS та керує SA у середовищі. Таким чином, SAM можна встановити в будь-якому місці, наприклад, у кімнатах чи офісах, а також на робототехнічній платформі чи КПК.[26]

SA виконує функції застарілих програм та датчиків, встановлених у фізичних місцях, через зв'язок із SAM та CAMUS-MS. SA – це інтерфейси програмних модулів пристроїв та програм, які взаємодіють з CAMUS-MS. Оскільки SA надає атрибути та дії з інтерфейсами, до яких отримує доступ CAMUS-MS. Наприклад, припустимо, що

SA, яка надає інформацію про місцезнаходження користувача, має інтерфейси для надсилання ідентифікатора та фізичного місцезнаходження користувача, а RFID-датчик знаходиться в середовищі. [27]

Контекстно-залежний шлюз

Мешканець будинку може відповісти на вхідний дзвінок, вибравши бажані пристрої SIP UA, та надсилати інформацію про вихідну присутність «за потребою» (на основі попередньо визначених критеріїв контекстно-залежного обслуговування, таких як взаємозв'язки між об'єктами та поточні ситуації). Ефективним способом зробити це є розгортання інтелектуального шлюзу для вибору/керування сеансами зв'язку між домашніми мережами та Інтернетом. SIP-контекстно-залежний шлюз (SIP-CAG), який виконує роль секретаря домашніх додатків на основі SIP, забезпечує величезну гнучкість та можливості адаптації до різно (рисунок 1.5) підтримує контекстно-залежні програми на основі SIP, що базуються на мотивації, поведінці та потребах мешканців.

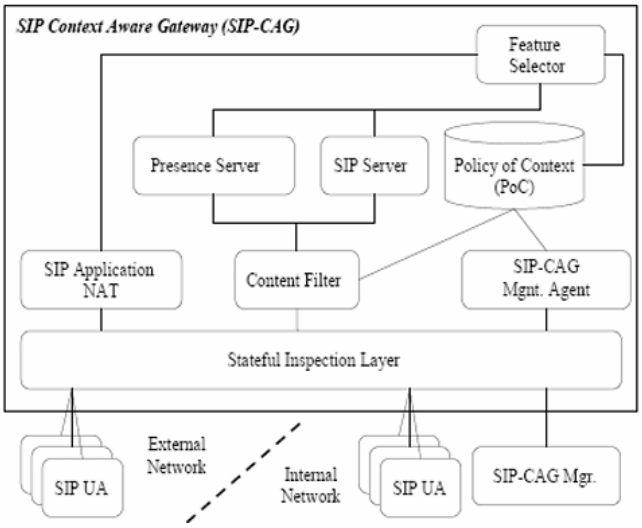

Рисунок 1.5 - Архітектура SIP-CAG

На наступному рисунку показано запропоновану архітектуру SIP-CAG, яка складається з таких основних компонентів: SIP-сервер та сервер присутності, рівень перевірки стану (SIL), фільтр контенту (CF), переадресація застосунків SIP (SAN), селектор функцій (FS), політика контексту (PoC) та агент керування SIP-CAG (SMA). Компоненти взаємодіють між собою для виконання управління контекстом. У наступних абзацах ми детальніше опишемо функцію кожного компонента.

1.4 Висновки по розділу

У першому розділі розглянуто теоретичні основи розробки систем розумного будинку. Проаналізовано основні принципи функціонування таких систем, концепцію інтелектуальних будівель та роль автоматизації в управлінні домашнім середовищем. Особливу увагу приділено контекстно-орієнтованим підходам, які дозволяють адаптувати поведінку системи до потреб користувача та поточних умов експлуатації. Встановлено, що інтеграція інформаційних технологій і засобів автоматизації є основою створення ефективних та зручних систем розумного будинку.

РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ РОЗРОБКИ СИСТЕМ РОЗУМНОГО БУДИНКУ

2.1 Сценарії використання систем розумного будинку

Перш ніж розглядати ризики, пов'язані з SHAS, шляхом застосування ISRA, ми визначимо деякі поширені сценарії для систем автоматизації розумного дому. Ці сценарії виникли в результаті обговорень з ключовими зацікавленими сторонами в галузі автоматизації розумного дому, тобто галузевими партнерами групи управління проектами SHAS.

Майно, а також користувачі та інформація, яку вони генерують, є невід'ємною частиною автоматизації розумного дому, і оскільки системи автоматизації розумного дому стають дедалі більш прийнятими мешканцями, ці системні інфраструктури також викликають дедалі більший інтерес з боку інших секторів промисловості. Це значною мірою пов'язано з тим, що системи автоматизації розумного дому представляють цінну платформу для прямої участі користувачів, часто через різні підключені датчики в домашньому середовищі, де користувачі та майно взаємодіють через планшети, смартфони, комп'ютери та різні портативні пристрої. З'являться майбутні можливості розширення переваг та послуг автоматизації розумного дому, що призведе до ризику концептуального зміщення. По суті, нові постачальники отримують вигоду від контекстно-залежної інфраструктури розумного дому, наприклад, використовуючи можливість додавання нових продуктів та послуг до екосистеми. Постачальники беруть участь в екосистемах, маючи ділові зв'язки з різними іншими постачальниками, що призводить до того, що питання безпеки та конфіденційності часто нехтуються або ігноруються.

Іншим аспектом цього є те, що система (у нашому випадку SHAS) переналаштовується та оснащується новими пристроями й програмним забезпеченням, які не були включені та не враховані в початковому проекті, що призводить до того, що систему не слід вважати ні стабільною, ні статичною; вона

динамічна та постійно змінюється, до того ж, способами, які важко передбачити.

Щоб пролити світло на сценарії, пов'язані з цим розвитком, ми зараз обговоримо інтеграцію камер відеоспостереження в розумний будинок, цифрові сліди та додавання підключених пристроїв у розумні будинки.

Від енергоефективності до камер відеоспостереження

Хоча системи автоматизації розумного дому, такі як SHAS, можуть бути спочатку призначені для підтримки енергоефективності, їх можна розширити, включивши також інші типи приладів у будинках. У розумних будинках використання камери спостереження зазвичай має на меті виявлення аномалій у домашньому середовищі, тобто подій, які відрізняються від щоденного використання. Першим прикладом такої аномалії є використання камери спостереження для виявлення або перевірки пожежі з віддалених місць, наприклад, виключення хибних тривог, щоб можна було викликати служби екстреної допомоги. Такі програми вже використовуються, але їх необхідно встановлювати в критично важливих для безпеки зонах будинку, наприклад, поблизу вхідних дверей та у спальнях. Окрім очевидної проблеми автоматичного розрізнення нормального випадку від аномального, цей вид застосування також передбачає взаємодію з людиною; хтось повинен підтвердити, що відбувається. Це може спричинити серйозні проблеми з конфіденційністю, тобто хтось може спостерігати за повсякденним життям у домашньому середовищі в результаті (нецільової) передачі камери.

Камеру спостереження також можна використовувати для інших особистих цілей, наприклад, щоб побачити, хто вдома, наприклад, для догляду за дітьми, контролю сну немовлят, догляду за літніми людьми тощо. Також можна контролювати стан продуктивності дому, наприклад, чи ввімкнено або вимкнено лампи, чи зачинено або відчинено двері, чи камери показують витік води тощо. Розумні домашні камери спостереження також можна використовувати в поєднанні з іншими видами підключених пристроїв, які разом можуть забезпечити надто детальне зображення осіб, що проживають у цьому будинку.

Розширення використання автоматизації розумного дому, наприклад,

шляхом включення камер спостереження, збільшує ризик концептуального зміщення, тобто коли інформація збирається для однієї мети, та сама інформація також може бути використана для інших цілей. Таке розширення використання також зазвичай породжує потребу в посиленні обмежень безпеки, особливо щодо функцій, що дозволяють віддалений доступ до домашніх камер, можливо, за участю осіб, які перебувають у їхньому приватному стані. Основне питання полягає в тому, чи враховуються комплексні аналізи всіх можливих сценаріїв загроз під час такого постійного розширення інфраструктур автоматизації розумного дому, які зараз розвиваються в галузі, і якою мірою слід дозволяти автономне прийняття рішень системою (і постачальниками, що стоять за нею).

Виявлення цифрових слідів

Окрім об'єднання різного обладнання та функцій системи, а також даних, які вони обробляють, потенційний зловмисник може отримати небажаний рівень знань про мешканців будинку. Наприклад, вимірювання загального споживання енергії надає обмежені знання про будинок, але додавання окремих вимірювань для кожного електричного пристрою (ступінь використання, час доби тощо) генерує більш детальну метаінформацію про звички членів сім'ї. Для зменшення споживання енергії в ідеалі слід проводити детальні вимірювання потужності. Це мотивує та полегшує, наприклад, використання орієнтованих на користувача програм, які зазвичай доступні у повсякденному житті, наприклад, через використання смартфонів, для підключення до дому. У сукупності ці програми, зазвичай розроблені для інших цілей, несуть потенціал для уникнення конфіденційності користувачів та загрози безпеці дому способами, які важко передбачити.

Цифрові сліди, які користувачі (більш-менш добровільно) залишають після себе під час використання системи, можуть бути використані для створення розширених особистих профілів мешканців будинку. Грабіжник може дистанційно контролювати шаблони рутинної діяльності на основі того, коли люди в домогосподарстві знаходяться вдома, і таким чином отримати засоби для здійснення крадіжки зі зломом, коли будинок порожній. Також можливо

створювати оцінки щодо діяльності, в якій беруть участь члени домогосподарства, на основі унікального розподілу споживання електроенергії для кожного приладу [28]. Наприклад, нові методи дозволяють відфільтрувати використання окремих приладів із загальної схеми споживання з точністю близько 90%. Це дозволяє системі автоматизації розумного дому окремо контролювати час і тривалість використання, наприклад, посудомийних машин, різних ламп, чайників, телевізорів та ігрових консолей [29]. Таку діяльність з моніторингу можна розглядати як еквівалент процесу картування, наприклад, між користувачами та їхніми інтересами, який відбувається сьогодні в багатьох частинах Інтернету, таких як сайти Google або Facebook, і в майбутньому може бути частиною більшої схеми картування, де відображаються різні (фізичні) звички користувачів. Результатом може бути детальне особисте досьє з інформацією, яка містить не лише цифрову поведінку користувача, а й таку, що відбувається у звичайному фізичному світі. Це, звичайно, означає набір дуже конфіденційної інформації, яка потрапивши в чужі руки, може становити серйозну загрозу конфіденційності користувача.

Оскільки цей тип інформації є делікатним у тому сенсі, що існує ризик зловживання з серйозними наслідками, такими як небажане розголошення комерційних повідомлень, ситуації переслідування тощо, кінцевим користувачам має бути зрозуміло, яка інформація з усієї інформації, що обробляється в розумному будинку, є особистого характеру, як вона обробляється та поширюється після потрапляння до системи автоматизації розумного дому через пристрій Інтернету речей або датчик, і яку роль користувач може відігравати в цьому процесі. Звичайно, також існує конфлікт між постачальниками послуг та споживачами; перші хочуть зібрати якомога більше інформації для подальшого використання або бізнес-можливостей, тоді як споживачі в першу чергу хочуть розкривати лише невеликі фрагменти інформації, корисні в певний момент часу.

Тому важливо дослідити, який фактичний вибір мають користувачі щодо своєї участі в процесі збору даних. Для того, щоб користувачі могли прийняти обґрунтоване рішення щодо впровадження системи автоматизації розумного

дому, вкрай важливо, щоб постачальники таких систем були прозорими щодо того, як персональні дані в системі обробляються, аналізуються, взаємопов'язуються та корелюються. Без такої прозорості користувачі навряд чи зможуть надати системі інформовану згоду, на чому має ґрунтуватися будь-яка участь користувачів. Тому головним питанням є потік даних у розумних будинках; як вони виглядають, як їх можуть використовувати підключені пристрої (і, відповідно, постачальники, що стоять за ними), і як має бути розроблена підтримка забезпечення конфіденційності користувачів.

Розширення Інтернету речей у розумних будинках

Популярною сферою застосування автоматизації розумного дому є використання контролю енергії як засобу запобігання крадіжкам зі зломом, наприклад, шляхом автономного планування освітлення, щоб включати також інші типи електронних з'єднань, такі як підключене до Інтернету радіо або телебачення, у той час, коли в будинку немає людей. Це також можна зробити без взаємодії з користувачем, тобто підключені об'єкти в системах автоматизації розумного дому можуть діяти автономно, наприклад, на основі своїх можливостей навчатися на поведінці користувача. Оскільки в цьому сценарії задіяно більше пристроїв, до системи може бути додано збільшений обсяг даних як своєрідний випадковий шум, що ускладнює розгортання можливостей моніторингу автоматизації розумного дому, як згадувалося вище. Ця додаткова функція, звичайно, є протилежністю забезпеченню енергоефективності з точки зору кількості енергії, доступної лише за потреби. Однак, оскільки планування роботи електричних пристроїв може переконати потенційних грабіжників уникати злому, а також обмежити засоби моніторингу користувачів у їхніх будинках, це може натомість сприяти підвищенню конфіденційності користувачів.

Більше того, використання приладів контролю енергії таким чином може мати недолік у тому, що будинок може стати фізично вразливим до нових загроз у вигляді спроб вторгнення, які зазвичай пов'язані з традиційною онлайн-взаємодією, наприклад, впливом шкідливих програм та участю в соціальних мережах. Таким чином, аналіз безпеки повинен враховувати рівень автономії, що

забезпечується системою (тобто, які рішення може приймати система без участі користувача), а також загальну обізнаність користувача про безпеку та конфіденційність. Це розширення аналізу безпеки, яке також включає ситуації, коли немає фактичного (людського) користувача поруч, але де користувач несподівано може відігравати важливу роль у стійкості та конфігурації безпеки (або її відсутності) розумних будинків. Тим часом підключені об'єкти розумного будинку зазвичай мають обмежені засоби для заходів підвищення безпеки, таких як шифрування, а також схильні до впливу навколишнього середовища. Оскільки об'єкти можуть діяти без відома користувача, вони можуть контролювати більше, ніж цифрову сторону будинку, тобто також фізичне середовище. Тому головним занепокоєнням є значний потенціал зловживань, спричинений цим розвитком технологій розумного дому, і цікаве питання полягає в тому, як слід проектувати безпеку, щоб запобігти вторгненням і сприяти концепції приватного, але підключеного до мережі дому.

На основі вищезазначеної відповідної роботи можна зробити такі висновки:

- Існує загальна потреба в емпірично обґрунтованих методах, які підтримують оцінку ризиків у середовищах розумного дому. Без таких методів впроваджені рішення безпеки ризикують не досягти бажаних цілей безпеки та конфіденційності системи автоматизації розумного дому.
- Існує загальна потреба в інтеграції безпеки в проектування. Перспективи аналізу ризиків зазвичай враховуються в підключеному будинку ззовні, тобто аналіз ризиків не включається на етапах проектування та розробки систем і технологій автоматизації розумного дому. Безпека в проектуванні має вирішальне значення для зменшення загроз, що виникають у будинках, підключених до Інтернету речей, особливо з точки зору зменшення шкідливого програмного забезпечення, контролю доступу та розкриття конфіденційності. Таке надійне управління безпекою також має сприяти загальним системним вимогам, що забезпечується під час розробки системи.

Ризики для конфіденційності користувачів потребують подальшої специфікації. Оскільки інформація, що генерується в межах розумного дому, часто має особистий характер і тому загалом повинна вважатися конфіденційною, слід звернути увагу на можливість порушення конфіденційності, щоб проілюструвати потенційні порушення особистої сфери дому.

2.2 Апаратне забезпечення систем розумного будинку

Обладнання, яке зазвичай використовується в будинках з широким розповсюдженням, включає віконниці, освітлювальні прилади, такі прилади, як кавоварки, холодильники або пральні машини, телевізори або мультимедійні сервери, якими можна керувати дистанційно. Мікроінформатичне обладнання, таке як комп'ютери, КПК, монітори, також є частиною сфери домашньої автоматизації. Комунікаційне обладнання також є важливим у цьому типі систем: точки доступу до Інтернету, маршрутизатори та мобільні телефони дозволяють отримати доступ до інформаційних технологій, які можуть знаходитися поза домом. Крім того, кероване електрообладнання, таке як дистанційні електричні розетки, є центральним елементом цих нових середовищ.

Ці пристрої повинні мати можливість дистанційного керування за допомогою програм, які координують їхні дії. Тому пристрої повинні забезпечувати протоколи, що дозволяють здійснювати такий тип зв'язку. На сьогодні існує понад п'ятдесят протоколів зв'язку, робочих груп та стандартизацій протоколів для домашнього зв'язку. Серед найпопулярніших можна знайти X10, KNX, EIB, INSTEON, Zigbee, Bluetooth, UPnP та DPWS. Ці протоколи дозволяють здійснювати зв'язок між домашніми пристроями за допомогою різних середовищ передачі: спеціалізованих кабелів зв'язку, зв'язку по лініях електропередач або передачі по радіочастоті. Такі протоколи не забезпечують однакових функцій і, отже, не використовуються однаковими типами пристроїв. X10, KNX, EIB, INSTEON та Zigbee призначені для невеликих пристроїв, таких як димери та жалюзі. Їхня комунікаційна пропускна здатність надзвичайно обмежена, але й

їхнє енергоспоживання також обмежене. Однак ці протоколи обробляють лише зв'язок і не здатні забезпечити виявлення пристроїв.

UPnP [30], Bluetooth та DPWS [31] – це протоколи вищого рівня, які обробляють не лише зв'язок, але й виявлення пристроїв. Bluetooth – це стандарт, спочатку розроблений для забезпечення бездротового зв'язку між комп'ютерами та їхніми периферійними пристроями. Метою UPnP та DPWS є забезпечення легкого підключення периферійних пристроїв один до одного та спрощення реалізації домашніх мереж (наприклад, для обміну файлами, зв'язку та розваг) та корпоративних мереж. UPnP забезпечує такі можливості, визначаючи свої протоколи зв'язку та виявлення поверх існуючих стандартів інтернет-зв'язку, таких як HTTP, тоді як DPWS в основному використовує стандарт, визначений веб-сервісами.

Застосування

У цьому розділі представлено три програми, що є репрезентативними для галузі домашніх обчислень. Перший приклад – це програма, яка дозволяє залишатися вдома людям, що одужують, або людям похилого віку.

Ця програма вимагає використання специфічних для цієї галузі датчиків, таких як кардіомонітори, манометри артеріального тиску, сенсори та камери відеоспостереження. Ці пристрої збирають дані про пацієнта та надсилають їх до програми, яка її обробляє. У разі виникнення проблеми автоматично спрацьовує сигнал тривоги для виклику допомоги. За нормальних умов роботи програма створює звіти та регулярно надсилає їх до лікарні або лікарям.

Другий приклад – це застосунок, який керує мультимедійною розважальною системою з дому. Застосунок пропонує мешканцям набір фільмів з фільмотеки, а також пропонує оренду або придбання фільмів у інтернет-провайдерів. Потім фільм автоматично проектується на екран кімнати, в якій знаходиться користувач, і застосовується атмосфера, пов’язана з переглядом відео: закриття віконниць та зменшення яскравості світла. Дзвінок телефону автоматично вимикається. Натомість користувач бачитиме непомітне попередження на екрані про будь-які вхідні дзвінки. Він може вирішити

ігнорувати його або відповісти на дзвінки, і фільм буде автоматично поставлено на паузу.

Останній приклад застосування полягає в управлінні безпекою та мінімізації споживання енергії в будинку за відсутності користувача. Коли користувач залишає свій дім, світло вимикається, температура в кімнатах автоматично знижується, підключається сигналізація та закриваються віконниці. У разі тривалої відсутності запускається симуляція присутності, яка регулярно відкриває заслінки та освітлювальні лампи відповідно до звичок користувача. Коли користувач повертається додому, система відключається від мережі, сигналізація вимикається; освітлення та заслінки працюють відповідно до яскравості зовні.

Виходячи з нашого досвіду, ми класифікували ці програми на три категорії залежно від різних життєвих циклів цих програм (див. рис. 2.1).

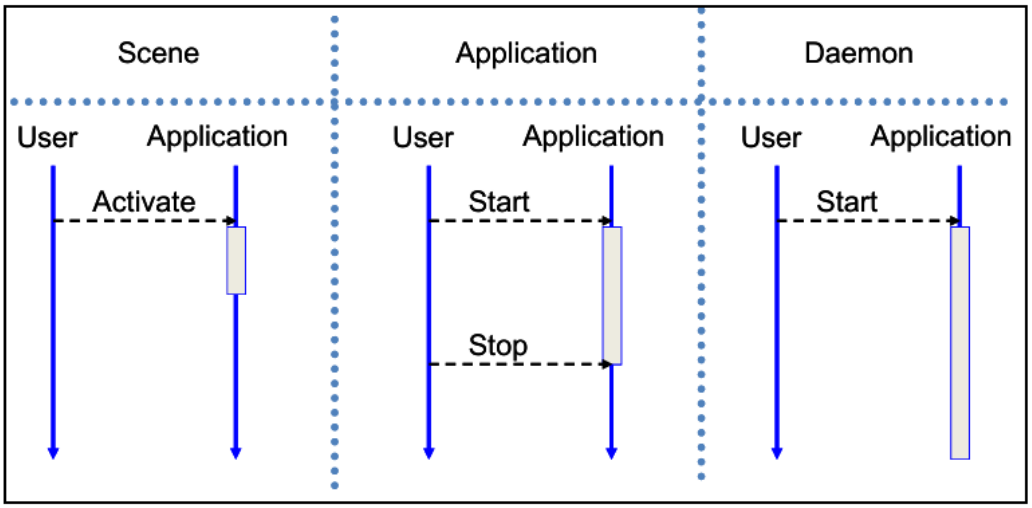

Рисунок 2.1 - Три типи домашніх додатків

Певний тип застосунку називається сценою. Ці застосунки складаються з виконання серії попередньо налаштованих дій у момент їх спрацьовування. Цей тип застосунку виконує всі ці дії та завершує свою роботу. Наприклад, коли користувач прокидається, застосунок відчиняє всі віконниці в будинку, підвищує температуру у ванній кімнаті, нагріває кавоварку та вмикає телевізор на улюбленому новинному каналі користувача.

Інший тип застосунку, який називатиметься екземплярним або просто застосунком, працює протягом обмеженого періоду часу. Ці застосунки мають особливість запускатися, а потім зупинятися. Наприклад, застосунок активується, коли користувач виходить з дому. Цей застосунок керує безпекою, мінімізуючи споживання енергії будинком. Застосунок вимикає все світло в будинку, закриває всі віконниці, знижує опалення, підключає сигналізацію. Коли користувач повертається додому, застосунок зупиняється та відключається, сигналізація вимикається, і залежно від обставин, він може знову відкрити віконниці або ввімкнути світло.

Останній тип застосунків називається демоном. Цей тип застосунків запускається та працює безперервно. Наприклад, застосунок для спостереження за пацієнтами вдома збиратиме медичні дані про пацієнта, а потім надсилатиме щоденні звіти лікарю. Такі застосунки виконуються та ніколи не зупиняються.

Виклики

Ця галузь обчислювальної техніки охоплює велику кількість застосувань, особливо корисних для допомоги людям у повсякденному житті. Головним завданням повсюдних обчислень є забезпечення цілісного повсюдного середовища, що надає корисні послуги та застосовує набір гетерогенного, розподіленого та динамічного обладнання та програмного забезпечення, що взаємодіють за різними протоколами. У цьому контексті кілька характеристик, характерних для галузі повсюдних обчислень, роблять цю галузь привабливою з точки зору промисловості та користувачів, водночас порушуючи складні наукові проблеми для розробки та управління цими системами. Розподіл. Пристрої є невід'ємною частиною середовища. Вони розосереджені у фізичному середовищі та доступні через різні протоколи, які можуть використовувати кабельні або бездротові технології. Додатки, що використовують можливості такого обладнання, не обов'язково працюють на розглянутому обладнанні і тому є розподіленими.

Гетерогенність . Наразі існує велика кількість програмних технологій та комунікаційних протоколів для галузі повсюдних обчислень. Сьогодні немає

планів щодо того, як досягти консенсусу щодо спільного та єдиного комунікаційного протоколу. Для домашніх мереж вже доступно понад п'ятдесят протоколів, робочих груп та специфікацій. Стандартизація комунікаційних протоколів неможлива, оскільки пристрої можуть мати дуже різну природу, що впливає на використовувані комунікаційні протоколи. Наприклад, лампа зв'язується за допомогою дуже простого протоколу, тоді як КПК або медіасервер можуть використовувати складніші комунікаційні протоколи, наприклад, з огляду на безпеку. Крім того, виробники, що постачають обладнання та протоколи, не мають стратегічного інтересу до цього типу єдиного протоколу, оскільки вони втратять контроль над своїм обладнанням. Динамізм . Доступність обладнання в повсюдному середовищі набагато більш мінлива, ніж в інших галузях обчислень. Ця проблема спричинена кількома факторами, включаючи: 1) користувачі вільно переміщуються та часто змінюють своє місцезнаходження, що впливає на положення обладнання, яке вони носять з собою; 2) користувачі можуть добровільно вмикати та вимикати пристрої, або ж у них може випадково розрядитися батарея; 3) користувачі та постачальники можуть періодично оновлювати розгорнуті послуги.

Кілька постачальників . Пристрої в повсюдному середовищі зазвичай надходять від різних постачальників. Крім того, програми, розгорнуті та запущені на такому обладнанні, можуть бути надані іншими постачальниками. У цьому контексті деякі програми будуть створені шляхом співпраці між різними постачальниками, що передбачає створення програм з кількома адміністративними органами. Передбачається, що постачальники обладнання та постачальники послуг захочуть зберегти певний контроль над своїми пристроями та програмним забезпеченням і таким чином обмежити доступ для зовнішніх організацій.

Масштабованість. Кількість пристроїв, присутніх у повсюдному середовищі, може бути дуже важливою. Це створює проблему масштабованості в додатках, що працюють у такому середовищі. Таким чином, повсюдні системи повинні бути здатні обробляти велику кількість обладнання, яке також є

динамічним.

Безпека. Безпека відіграє ключову роль у побудові повсюдних середовищ. Дійсно, такі відкриті системи дозволяють людям у середовищі мати доступ до обчислювальної системи. Однак доступ до певних пристроїв або персональних даних має бути високо захищеним. Програми, що працюють на системі такого типу, повинні гарантувати конфіденційність та цілісність даних. Доступ до приватних повсюдних систем, таких як автоматизовані будинки чи автомобілі, також повинен включати контроль доступу, щоб, наприклад, гарантувати, що злодій не зможе відключити сигналізацію, підключивши повсюдну систему.

Автоадаптація. Окрім динамічності програмного забезпечення та обладнання, повсюдні системи постійно стикаються з еволюцією у контексті їх виконання. Ці еволюції можуть включати зміни в поведінці, місці розташування, настрої та звичках користувачів, а також зміни в поведінці або доступності іншого програмного забезпечення. Програми, що працюють у такому середовищі, повинні мати можливість адаптуватися до цих змін та розробляти стратегії для реагування на різні події, які можуть виникнути під час їх виконання.

Простота використання. Зрештою, важливою характеристикою повсюдної системи є простота використання та управління. Дійсно, цей тип системи призначений для використання користувачами, які не мають знань в галузі інформатики. Як наслідок, повсюдні середовища повинні бути доступними для будь-якої людини і навіть прозорими. Мета повсюдних обчислень полягає в тому, щоб пристрої зникли з середовища. Це означає, що інтерфейс доступу до повсюдних систем має бути простим у використанні, а програми, що працюють у цих системах, повинні бути здатними адаптуватися до різних подій, які можуть втрутитися, щоб підтримувати придатність послуг до використання за будь-яких обставин.

Вимірювання відстані /[м] - середовище без будь-яких перешкод. Ідеальний стан – це коли немає твердих перешкод на шляху радіочастотного сигналу між передавачем і приймачем (Рисунок 2.2). Вимірювання показали величезні відмінності в прийомі сигналу, переважно між жалюзі та іншими

виконавчими механізмами. Вимірювання також включали тестування максимальної дальності сигналу для активного комутаційного елемента. Додаткове тестування підтвердило, що USB-інтерфейс з використанням візуалізації SmatHomeApp надійно працює на відстані до 150 метрів.

Під час цих вимірювань було встановлено оптимальну відстань 20 м для виконавчого механізму вимикача та виконавчого механізму регулювання яскравості та 10 м для виконавчого механізму жалюзі. Інтерфейс параметризації має серйозні проблеми з пошуком активних компонентів на відстанях понад 20 метрів. Це також обмежує вимірювання якості сигналу.

Вимірювання відстані /[м] - середовище з однією цегляною стіною. Цегляні матеріали є найпоширенішими матеріалами, що використовуються для будівництва будинків та квартир. Випробування цегляної стіни являє собою реальну модель та практичне застосування (Рисунок 2.2).

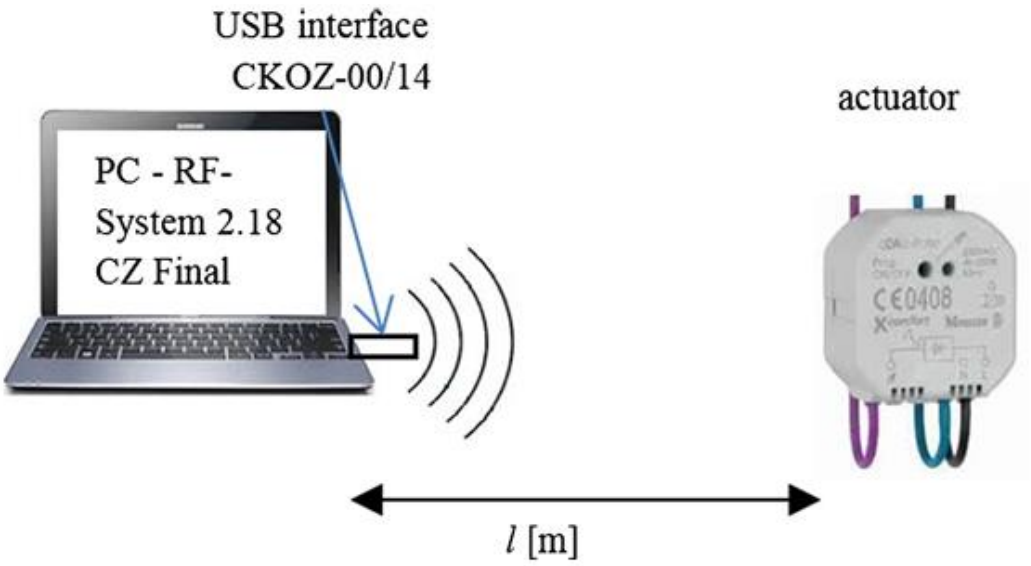

Рисунок 2.2 - Блок-схема вимірювання відстані [м] у середовищі без перешкод.

Втрата сигналу через цегляну стіну безпосередньо пов'язана з товщиною стіни, хоча цегляна глина є матеріалом з хорошими властивостями порівняно з іншими матеріалами, такими як залізобетонні конструкції або металеві матеріали. Це дозволяє спостерігати та порівнювати втрату сигналу з силою сигналу, що

поширюється через середовище без будь-яких перешкод. Максимальна відстань для керування комутаційним виконавчим механізмом за допомогою USB-інтерфейсу та візуалізації SmartHomeApp становить 96 метрів.Під час цих вимірювань було встановлено оптимальну відстань 20 м для виконавчого механізму вимикача та виконавчого механізму регулювання яскравості та 10 м для виконавчого механізму жалюзі. Інтерфейс параметризації має серйозні проблеми з пошуком активних компонентів на відстанях понад 20 метрів. Це також обмежує вимірювання якості сигналу.

Вимірювання відстані /[м] - середовище з двома цегляними стінами. Ми також змоделювали реальні умови, використовуючи середовище з двома цегляними стінами. Це відображає ситуацію, коли сигнал має проходити через одну кімнату, наприклад, через коридор між двома кімнатами.

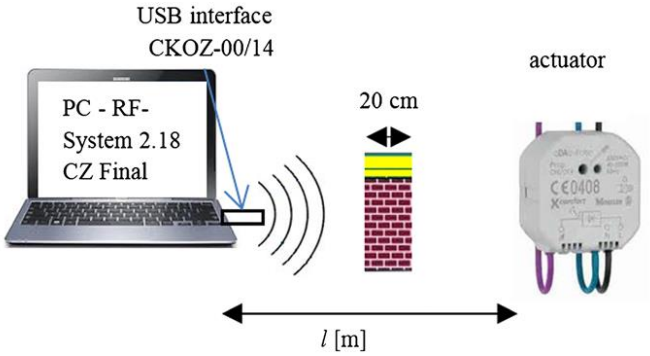

Рисунок 2.3 - Блок-схема вимірювання відстані [м] при наявності перешкоди у вигляді цегляної стіни товщиною в 1 цеглину.

Зрозуміло, що в цьому сценарії сила сигналу, що проходить крізь дві цегляні стіни, швидко погіршується.

У нашому змодельованому середовищі перша стіна була розміщена безпосередньо за USB-інтерфейсом зв'язку, а друга стіна була розміщена за два метри від першої. Під час цих вимірювань було встановлено оптимальну відстань 20 м для виконавчого механізму вимикача та виконавчого механізму регулювання яскравості та 10 м для виконавчого механізму жалюзі. Інтерфейс параметризації має серйозні проблеми з пошуком активних компонентів на відстанях понад 20

метрів. Це також обмежує вимірювання якості сигналу.

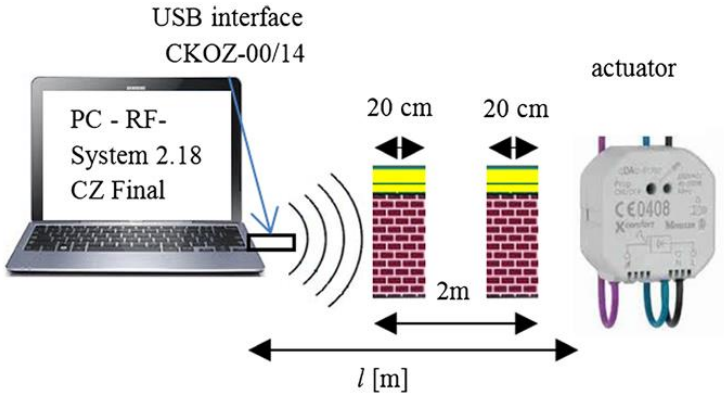

Рисунок 2.4 - Блок-схема вимірювання відстані [м] з перешкодою у вигляді двох цегляних стін

Обговорення

Майбутній розвиток візуалізації та інтелектуальних будинків. Швидкий розвиток технологій, доступних сьогодні для будинків, створив ситуацію, коли кілька електронних систем працюють одночасно в одному будинку. Тому інтеграція різних встановлених технологій стала важливим напрямком багатьох виробників, що виробляють ці системи [32]. Система автоматизованого керування, яка використовує бездротову радіочастоту, може бути інтегрована в ці системи завдяки інструментам візуалізації. Таким чином, це дозволить нам мати всі технології в одному місці та розподілити їх лише по будинку. Один зі способів досягнення цієї ситуації - створити домашній мультимедійний центр, де функції мультимедійного сервера виконуються невеликим комп'ютером з низьким енергоспоживанням. Підключення до Інтернету через велику кількість комп'ютерної периферії дозволяє нам підключити багато технологій, які в іншому випадку працюють незалежно. Як приклад можна навести інтеграцію

телебачення, радіо та керування електричними системами з Інтернетом.

Таблиця 2.1

Вимірювання відстані в середовищі без перешкод

Відстань (від джерела), м	**Перемикач %**	**Привід жалюзі %**	**Привід регулювання яскравості %**

0	100	100	100
1	100	92.2	100
2	98.1	76	100
2.5	86.6	64.7	84.2
4	82.3	48.3	73.6
5	75.3	32.4	68.2
7	72.8	42.6	63.7
9	76.5	30.3	60.3
10	60.1	28.8	60.8
15	47.6	0	50.4
20	40.5	0	38

Завдяки Wi-Fi [33] інші пристрої вдома також можуть отримувати доступ до Інтернету. Водночас ми можемо використовувати мультимедійний сервер для прослуховування інтернет-радіо або перегляду телебачення та зберігання фільмів на мультимедійному сервері, доступу до Інтернету та гри в комп'ютерні ігри. Завдяки візуалізації, Електрична проводка також може бути легко керована, оскільки мультимедійний сервер може розповсюджувати систему візуалізації на інші пристрої, такі як, наприклад, смартфони.

Таблиця 2.2

Вимірювання відстані в середовищі з однією перешкодою

Відстань (від джерела), м	**Перемикач %**	**Привід жалюзі %**	**Привід регулювання яскравості %**
0	100	100	100
1	92.4	86.2	96.4
2	95.1	73.5	90.6
2.5	85.3	63	83.1
4	78.8	47.2	65.3
5	70.6	29.9	67.8
7	65.8	34	54.8
9	67.7	27.3	53
10	56	25.2	57.3
15	39.5	0	42.6
20	35.9	0	33.1

Таблиця 2.3

Вимірювання відстані в середовищі з двома перешкодами

Відстань (від джерела), м	Перемикач %	Привід жалюзі %	Привід регулювання яскравості %
0	100	100	100
1	91.4	85.5	88.3
2	86.6	73.2	88.8
2.5	75.5	54.8	82.5
4	73.0	40.4	64.0
5	69.7	20.1	60.0
7	56.5	26.2	54.4
9	64.5	24.6	43.6
10	48.7	18.5	52.7
15	33.6	0	35.1
20	28.0	0	27.3

2.3 Датчики, виконавчі механізми та бездротові технології передачі даних

Що стосується фактичної реалізації електропроводки, бажано використовувати технології, що використовують бездротові технології (виконавчі механізми та датчики) в будинках з уже існуючою проводкою. Для керування операційними та технічними функціями в системах "Розумний дім" та "Розумний догляд за будинком" використовується бездротова технологія xComfort. Ця технологія використовує для зв'язку радіочастоту 868,3 МГц. Передача займає максимум 1% від загального часу. Дані підтверджуються під час радіочастотного зв'язку між виконавчим механізмом та датчиком. Передача даних може бути захищена паролем. Збільшення дальності радіочастотного сигналу між окремими компонентами можливе завдяки автоматичній передачі сигналу (маршрутизації) виклику.

Щодо зв'язку між базою даних, елементами візуалізації та активними елементами, було розроблено програмний драйвер, який робить цей зв'язок можливим рисунок 2.5: Розроблене програмне забезпечення для візуалізації було розроблено з урахуванням вимог до веб-інтерфейсу, можливості керування програмним забезпеченням через мобільний телефон, за допомогою голосового керування, а також з урахуванням легкої розширюваності, масштабованості та модульності.

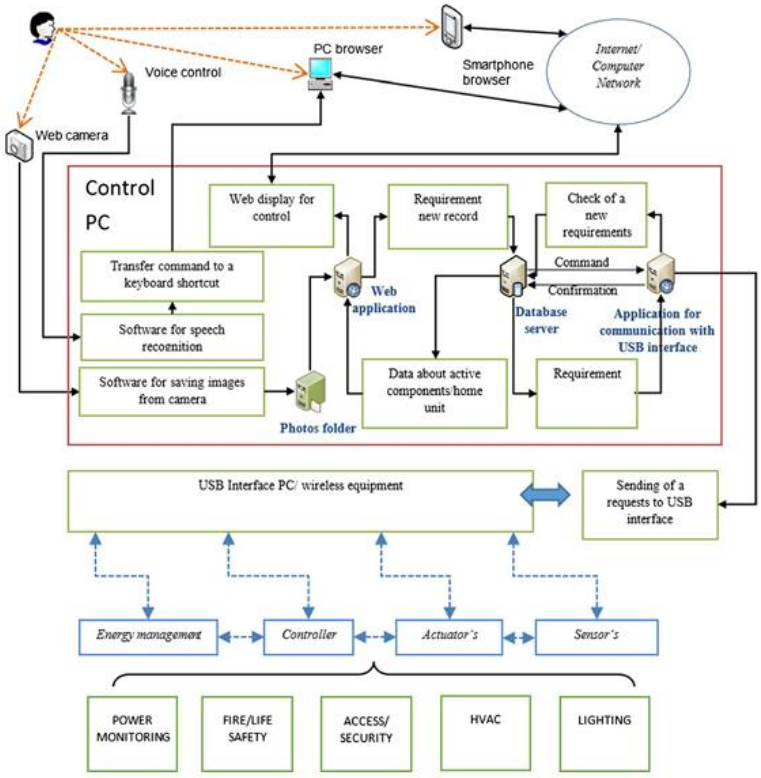

Рисунок 2.5 - Блок-схема розробленого програмного забезпечення

Для візуалізації для керування, моніторингу та візуалізації операційних і технічних функцій у системі «Розумний дім» або «Розумний догляд за будинком» з бездротовою технологією.

Інтерфейс користувача діє як проміжний шар між користувачем та керуючим комп'ютером. Він відображає інформацію про стан системи та забезпечує елементи огляду та керування, що використовуються для керування активними елементами в квартирі. Система складається з трьох логічних частин рисунок 2.6:

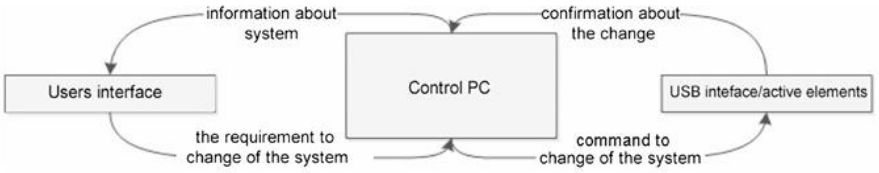

Рисунок 2.6 - Блок-схема трьох логічних частин розробленої системи.

Керуючий комп'ютер забезпечує взаємозв'язок між пристроєм за допомогою USB-інтерфейсу та інтерфейсом користувача. Він забезпечує

середовище для безперебійної роботи програмних елементів, що використовуються для процесу керування активними елементами. Він приймає та реєструє запити всієї системи. Він перевіряє та надсилає зміни, тобто запити на зміни системи. Він приймає підтвердження змін та пересилає їх до системи та до інтерфейсу користувача. Він також зберігає інформацію, яка використовується пізніше для оптимізації виконавчих механізмів системи, встановлених у житловому приміщенні. USB-інтерфейс надсилає запити до активних елементів у квартирі та отримує відповідь, яка потім пересилається до керуючого комп'ютера. З'єднавши ці рівні, можна реалізувати створену систему візуалізації для керування експлуатаційними та технічними функціями будинку за допомогою бездротової системи xComfort. Процес реалізації моделі програмного забезпечення для візуалізації:

1. Створення плану квартири.
2. Специфікації використовуваних компонентів.
3. Аналіз протоколу зв'язку та тестування пакетів.
4. Створення програмного забезпечення для візуалізації (аналіз, процес впровадження візуалізації).
5. Опис використання інших програмних застосунків.
6. Тестування застосунків.

1. Створення планування квартири

Створена візуалізація типового житлового блоку включає коридор, ванну кімнату, кухню, поєднану з вітальнею, та робочу кімнату, поєднану зі спальнею. Візуалізація реалізована у вигляді клікабельної карти (Рисунок 2.4). Основою для клікабельної карти служить планування поверхів квартири. Для потреб візуалізації планування розділене на логічні блоки відповідно до кімнат та способу їх використання. Такий поділ забезпечує користувачеві кращий огляд кімнат та простіший доступ до потрібного активного елемента.

2. Специфікації використаних компонентів

Далі необхідно було визначити окремі компоненти бездротової системи з подальшим програмуванням, так званим налаштуванням зв'язків між

виконавчими механізмами за допомогою програмного інструменту EATON RF-System. Взаємну зв'язок виконавчих механізмів та датчиків можна контролювати за лініями на екрані комп'ютера.

Аналіз протоколу зв'язку та тестування пакетів

Протокол зв'язку описує зв'язок між окремими компонентами бездротової технології xComfort. Власником цього протоколу є Eaton, а його використання та публікація регулюються договірними угодами. У документі детально описано зв'язок у базовому та комфортному режимах. Для тестування протоколу зв'язку використовувалися такі компоненти:

виконавчий механізм вимикача (CSAU-01/01), виконавчий механізм жалюзі (CSJA-01/02), виконавчий механізм дімування (CDAU-01/03), бінарний вхід (CBEU-02/02), кнопка (CTAA-02/01), інтерфейс параметризації (CRSZ-00/01), комунікаційний інтерфейс (CKOZ-00/14).

Форма пакетів, що надсилаються на USB-інтерфейс, визначається в протоколі зв'язку. Тест було проведено на активних елементах з'єднання тестового випадку за допомогою програмного забезпечення SimpleHIDWrite.

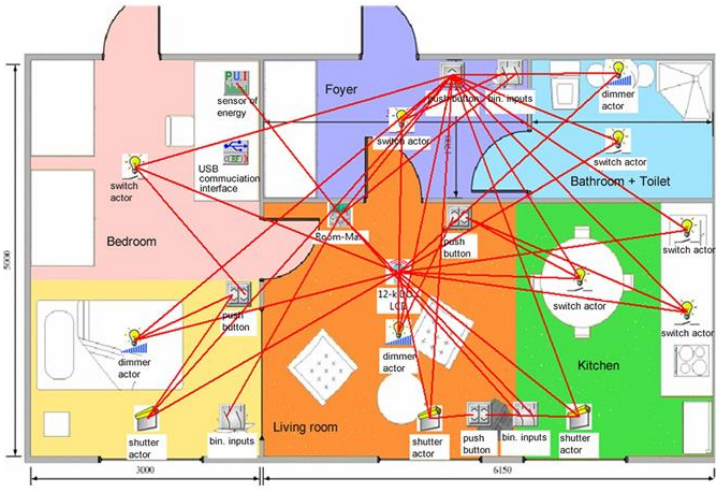

Рисунок 2.7 - Проектування плану поверху середовища візуалізації та взаємозв'язок окремих компонентів бездротової системи для системи «Розумний дім».

Для керування вищезгаданими виконавчими механізмами використовувалися такі команди: увімкнути, вимкнути, яскравіше, темніше,

зупинитися, повністю закрити, повністю відкрити, закрити на один крок, відкрити на один крок.

3. Опис проектування візуалізації та прикладного програмного забезпечення

Це набір технічних рішень, що охоплюють різні рівні керування, моніторингу та візуалізації станів керованих пристроїв (бездротових виконавчих механізмів системи xComfort). Однією з основних частин системи є SQL-сервер, де зберігаються всі стани системи та інструкції керування. Інструкції керування записуються. Кожен новий запис розпізнається застосунком і надсилається через інтерфейс керування до відповідних керованих елементів. Потім керований елемент надсилає інформацію, що підтверджує, чи запит був фактично виконаний. Якщо отримано інформацію, яка вказує на те, що запит виконано, стан пристрою керування змінюється, і запит позначається як виконаний. Тут блок керування представляє собою програмне забезпечення для візуалізації, відображає стани окремих пристроїв та варіанти їх конфігурації. Використовуючи сервер бази даних, можна контролювати записані інструкції та процеси, оцінювати результати окремих інструкцій і таким чином знаходити оптимальне рішення. Фактичне впровадження візуалізації було розділено на кілька етапів:

Аналіз веб-візуалізації

Аналіз даних,

Аналіз програмного забезпечення для передачі USB-інтерфейсу,

Клієнт процесу впровадження візуалізації,

Сервер процесу реалізації візуалізації.

Веб-візуалізація використовується користувачем для керування компонентами інтелектуальної електричної системи. Ключовою функцією є керування виконавчими механізмами шляхом прямих змін в електричній системі та колективної/масової модифікації електричної системи [34].

Аналіз даних описано за допомогою концептуальної діаграми бази даних та схеми бази даних зі словниками/термінами даних, що використовуються у Вимозі.

Аналіз програмного забезпечення для передачі USB-інтерфейсу

Інструмент ? USBinterfaceTransfer використовується для надсилання запитів до USB-інтерфейсу. USB-інтерфейс надсилатиме запити до активних елементів інтелектуальної електричної системи. Програма багаторазово шукає неповні/невиконані записи, а потім позначає їх як завершені/виконані рисунок 2.8 [35].

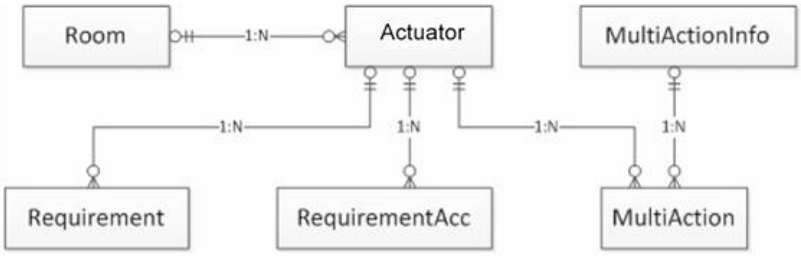

Рисунок 2.8 - Концептуальна схема бази даних.

Процес впровадження візуалізації для клієнта

Вибір технічних елементів, необхідних для відповідної візуалізації, безпосередньо залежить від наших зусиль зробити керування виконавчими механізмами доступним через веб-браузер, а також через комп'ютерну мережу або Інтернет.

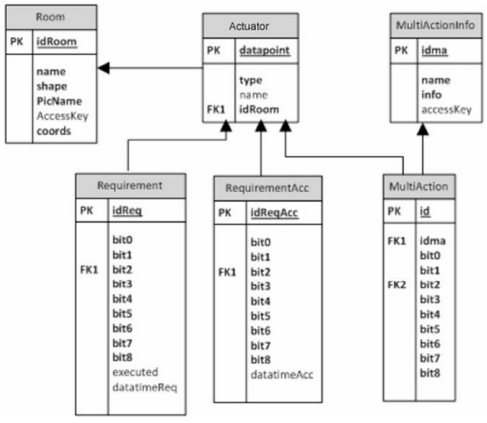

Рисунок 2.9 - Схема бази даних.

Ця частина містить описи та приклади використання технічних елементів, які були використані для створення середовища візуалізації для клієнта/на об'єкті клієнта. Клієнт являє собою веб-браузер, який використовується користувачем

для доступу до програми. Якщо використовується система голосового керування, це стосується додаткової програми, яка керує веб-браузером. Інформація, що надається користувачеві відповідного веб-браузера, динамічно завантажується з бази даних, у цьому випадку з MS SQL.

Передача інформації між керуючим комп'ютером та клієнтом здійснюється через протокол HTTP рисунок 2.10. Документ структуровано за допомогою HTML та SCC. JavaScript використовується як вища форма комфортної системи керування [37].

Рисунок 2.10 - Проектування середовища візуалізації керування операційними та технічними функціями в системі «Розумний дім».

Програмні інструменти, що використовуються в додатку, описані нижче:

Веб-сервер

Комп'ютерна програма, що відповідає за виконання запитів http-клієнтів. Для веб-сервера веб-браузер вважається клієнтом. Він є важливою частиною роботи веб-застосунків. Веб-сервер надсилає свою відповідь у вигляді HTML-документа; його можна створювати динамічно, використовуючи, наприклад, технології PHP або ASP.

Сервер бази даних

У наступному розділі описано структуру бази даних SmartHomeApp, процес її створення та використання. Предметом аналізу є вся база даних; тут будуть описані лише найважливіші схеми.

Створення бази даних:

Ця команда створила базу даних Розумного дому та налаштувала команду Зібрати для встановлення набору символів. Чеський набір символів було встановлено командою ЗІРВАТИ Чеська CI AS, яка не враховує регістр (CI) та діакритичні літери.

Створення таблиці кімнати

Містить інформацію про кімнату, життєво важливу для роботи програми SmartHomeApp.

Лістинг 2.1 - Створення таблиці кімнати

```
CREATE TABLE [dbo].[Room](
[idRoom] [int] PRIMARY KEY NOT NULL,
[name] [nvarchar](50) NULL,
[shape] [nvarchar](50) NULL,
[coords] [nvarchar](50) NULL,
[PicName] [nvarchar](50) NULL,
[accessKey] [nchar](1) NULL)
```

idRoom – служить унікальним ідентифікатором кімнати в базі даних.

Назва – назва кімнати, яка відображатиметься в системі.

Фігура – базова фігура для малювання кімнати на плоскій карті.

Координати - координати плоского розміру.

PicName – файл зображення, що показує намальоване положення кімнати.

Клавіша доступу – вибрана комбінація клавіш, необхідна для простого та безпроблемного голосового керування.

Правило PRIMARY KEY NOT NULL визначає стовпець як первинний ключ, за замовчуванням UNIQUE, але речення UNIQUE не може бути присутнім.

Створення таблиці виконавчих механізмів

Містить інформацію про всі виконавчі механізми в системі

інтелектуального дому, якими потрібно керувати.

Лістинг 2.2 - Створення таблиці виконавчих механізмів

```
CREATE TABLE [dbo]. [actuator]

CREATE TABLE [dbo].[actor](
[datapoint] [bit] PRIMARY KEY NOT NULL,
[type] [bit] NULL,
[name] [nvarchar](50) NULL,
[idRoom] [int] NULL)
```

Унікальна точка даних

ідентифікатор виконавчого механізму в системі.

Тип - тип приводу, жалюзі або інший.

Name – Назва, мітка виконавчого механізму в системі. Відображається під час процесу керування. idRoom – Ідентифікатор кімнати.

Створення таблиці вимог

Містить команди для активних елементів плоскій області. Він містить як команди, що виконуються шляхом збереження значення у виконаному стовпці, так і команди, що очікують на виконання.

idReq -Номер запиту.

bit0 bit8 – Шістнадцяткове значення, що становить пакет команди. executed – Інформація про виконання запиту.

Вищезазначене описувало ядро бази даних для візуалізації активних елементів xComfort.

За задумом, використання движка бази даних не має значення. Однак важливо зберігати інформацію про кімнати та їх атрибути, і ця мета досягається за допомогою таблиці "Кімнати".

Лістинг 2.3 - Створення таблиці вимог

```
CREATE TABLE [dbo].[requirement](
[IdReq] [int] PRIMARY KEY NOT NULL,
[bit0] [bit] NULL,
[bit1] [bit] NULL,
[bit2] [bit] NULL,
[bit3] [bit] NULL,
[bit4] [bit] NULL,
[bit5] [bit] NULL,
[bit6] [bit] NULL,
[bit7] [bit] NULL,
[bit8] [bit] NULL,
[executed] [int] NULL)
```

Важливо вести облік того, який виконавчий механізм належить до кімнати, його тип для різних елементів керування та номер, під яким він зареєстрований у системі активних елементів, і, нарешті, зберігати запити на зміни в системі. Записи команд дозволяють використовувати команди пізніше під час аналізу. Схема бази даних, описана вище, не дуже підходить для аналізу поведінки системи, але вона надає нам ядро для її запуску. Для аналізу роботи та технологічних інструментів квартири необхідно розширити систему.

2.4 Висновок по розділу

У другому розділі досліджено методи та інструменти розробки систем розумного будинку. Розглянуто основні сценарії використання таких систем, проаналізовано апаратне забезпечення, датчики, виконавчі механізми та бездротові технології передачі даних. Визначено особливості взаємодії між програмними та апаратними компонентами системи. Результати дослідження показали, що правильний вибір технологій зв’язку та обладнання забезпечує надійність, масштабованість і ефективність функціонування системи розумного будинку.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проєктування та архітектура системи розумного будинку

У цій роботі представлено інтегрований підхід з використанням протоколу SOAP для ефективних систем керування розумним будинком на основі веб-сервісів. Запропонована система керування розумним будинком здатна підтримувати кілька сервісів та пристроїв, інтегрованих за допомогою вбудованої системи. Вбудована система налаштована як житловий шлюз, а також взаємодіє з комутаційним модулем та віддаленим клієнтом. Житловий шлюз знаходиться у вбудованій системі з модулем бази даних у серверній частині. Усе підключення системи керування будинком здійснюється через конфігурацію Ethernet. Ethernet ідеально підходить для середовища розумного дому завдяки своїй продуктивності в режимі реального часу, а також враховуючи житлову кабельну мережу Cat5, яка легко доступна в будинках [38]. Ethernet знайшов своє застосування в середовищі розумного дому завдяки низькій вартості впровадження та проводки. Весь зв'язок між пристроями та системами керування розумним будинком базується на протоколі обміну повідомленнями SOAP. SOAP обраний через свою характеристику сумісності та визначає стандартний механізм обміну повідомленнями, використовуючи XML-конверт як корисне навантаження. Основною перевагою використання SOAP є те, що він забезпечує відкритий стандарт для наскрізного зв'язку, який не залежить від постачальника, а також високий ступінь гнучкості для інтеграції різних систем. Середовище розумного дому, як правило, складається з розподілених об'єктів з різнорідними сервісами та додатками. У такому середовищі вимога сумісності надає великого значення забезпеченню взаємодії між різнорідними програмами або сервісами. SOAP як технологія веб-сервісів дозволить обмін повідомленнями між двома різними підсистемами незалежно від операційної платформи чи мови, що використовується. Вся архітектура

системи показана на рисунку 3.1.

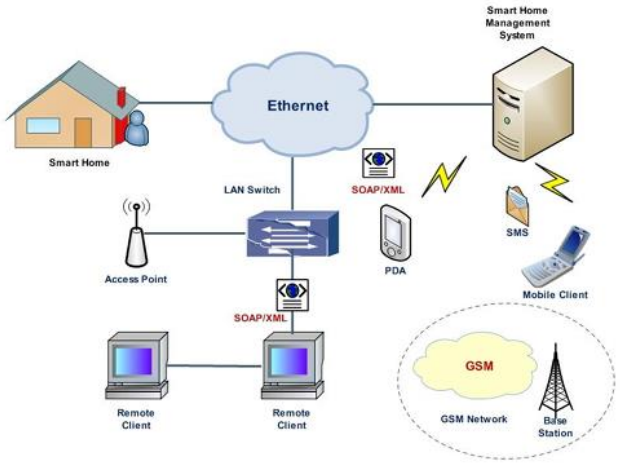

Рисунок 3.1 - Архітектура системи управління «розумним будинком»

Вбудована система

Система керування будинком розроблена з використанням вбудованих систем, що забезпечують функції зберігання даних та шлюзу. Для системи керування розумним будинком вбудовані системи утворюють центральний модуль, який містить сервер, драйвери та програмне забезпечення. Виконувані функції полягають у інтеграції пов'язаних модулів систем керування будинком та роботі в якості шлюзу. Система налаштована за допомогою Windows Server 2003 та Internet Information Services 6.0. Встановлено та налаштовано .NET Framework 2.0 [39] разом із SQL Server як база даних у серверній частині. Програмний механізм, написаний для керування розумним будинком, зберігається на 1 ГБ пам'яті Compact Flash™, доступній у системі малого розміру. Система також підтримує 2 входи та 2 виходи підключення Ethernet та системну пам'ять обсягом до 1024 МБ. Чотирьох портів USB 2.0, а також двох портів IDE достатньо для конфігурації зовнішніх пристроїв. Використовуючи з'єднання RS-232, вбудована система інтегрована з комутаційним модулем, що взаємодіє з домашніми пристроями та приладами. Однією з характеристик керування розумним будинком є цілодобова робота. Очікується, що мешканці будинку матимуть постійний доступ до інформації про прилади. Таким чином, вбудована система здається ідеальною платформою для безперервної роботи з меншим часом

простою. На рисунку 3.2 показано діаграму варіантів використання системи керування розумним будинком:

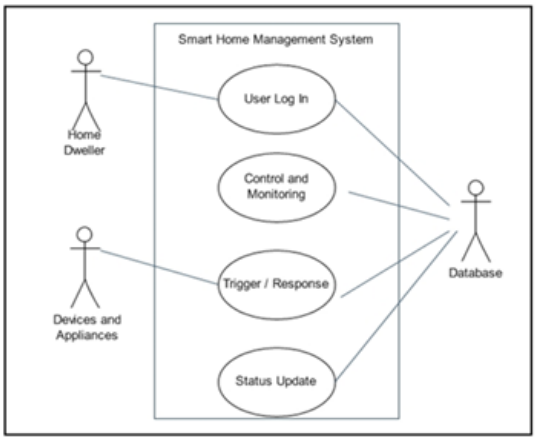

Рисунок 3.2 - Діаграма варіантів використання консолей управління

Діаграма варіантів використання на рисунку 3.1 вище зображує призначення системи керування розумним будинком для виконання чотирьох хронологічних завдань. Домашньому користувачеві потрібно буде увійти в систему та налаштувати потрібні пристрої. Після завершення налаштування система активує пристрої та оновлює їхній стан у базі даних. Оновлення стану надає мешканцям будинку зворотний зв'язок щодо працюючих пристроїв або приладів, якими керує система.

Модуль інтерфейсу

Модуль інтерфейсу складається з веб-сервера з повним стеком протоколів, встановленим на повноцінній операційній системі реального часу. Цей модуль забезпечує всю необхідну функціональність, необхідну як для Ethernet, так і для Інтернет-з'єднання. Цей модуль інтерфейсу містить попередньо налаштований функціональний стек для TCP, DHCP, HTTP та SNMP. Використовуючи API програмного інтерфейсу, весь модуль інтерфейсу розроблений для перетворення TCP у послідовний порт. Модуль повністю інтегрований з приймачем 10/100 Base T для підключення Ethernet. Модуль інтерфейсу керує та перетворює мережеві дані із системи керування будинком у послідовне з'єднання. Ці послідовні дані, отримані в результаті перетворення, будуть спрямовані на комутаційний пристрій

для виконання механізму керування домашніми системами. Весь процес відбувається навпаки та забезпечує зворотний зв'язок на клієнтський пристрій або термінал мешканця будинку.

Модуль комутації

Одним з основних модулів системи керування будинком є комутаційний модуль, який обробляє вхідний сигнал для керування та моніторингу домашніми пристроями. Комутаційний модуль – це розширений пристрій, підключений до вбудованої системи через Ethernet-з'єднання. Функціональність цього модуля полягає в прослуховуванні вхідних даних, аналізі та виконанні запуску підключених домашніх пристроїв через систему керування розумним будинком. Залежно від конфігурації реле та комутаційного модуля генерується 15 сигналів зворотного зв'язку.

Таблиця 3.1

Коди керування модулем комутації

Relay ID (ID реле)	**Hex (ON)**	**Binary (ON)**	**Hex (OFF)**	**Binary (OFF)**
R1	51	0101 0001	61	0110 0001
R2	52	0101 0010	62	0110 0010
R3	53	0101 0011	63	0110 0011

Формат сигналу зворотного зв'язку – Little-Endean; сигнали стану інтерпретуються у двійковому вигляді. Запрограмовані коди для перемикання модуля в режим УВІМК. та ВИМК. наведено в таблиці 3.1.

Комутаційний модуль розроблено з використанням 8-бітного мікроконтролера, запрограмованого для виконання механізму вибірки вхідних даних з інтерфейсного модуля та генерації відповідного вихідного комутаційного сигналу. У комутаційному модулі налаштовано 15 комутаційних каналів для сигналу дистанційної активації. Ці комутаційні канали з'єднані з групою реле, які приймають та запускають вхідний сигнал. Побутові пристрої підключені до цифрового виходу реле, що забезпечує достатню сумісність
напруги.

GSM-модуль

Розуміючи потенціал GSM-зв'язку в системах дистанційного керування, його також було включено до цієї запропонованої системи керування розумним будинком. Система керування розумним будинком повинна працювати в режимі безперервної роботи. У разі недоступності сервера або мережі, продуктивність системи не постраждає, оскільки GSM-модуль забезпечує альтернативний механізм керування за допомогою служби коротких повідомлень (SMS) для підтримки роботи домашніх пристроїв. SMS можна розглядати як передову технологію, яка підтримується мобільними та портативними пристроями. SMS також вважається найдоступнішим рішенням для дистанційного моніторингу. Мета включення GSM-модуля як альтернативного механізму керування в системи керування розумним будинком пов'язана з фактором мобільних пристроїв. Користувачі розумного будинку часто віддають перевагу мобільним пристроям як віддаленому терміналу для керування домашніми системами через низку переваг. Домашні користувачі віддають перевагу мобільним пристроям через їх добре відомий інтерфейс та простіший використання, які часто служать основним інтерфейсом користувача в середовищі розумного будинку. Ці пристрої також зазвичай носять із собою мешканці будинків, і вони, швидше за все, знаходяться в зоні дії середовища розумного будинку, коли відбувається фізична взаємодія з ними. Інші міркування пов'язані з різноманітністю бездротових мереж, мобільністю, оскільки інформація, що зберігається в системі керування будинком, залишається доступною для мешканців будинку, та персоналізацією, оскільки мобільний пристрій належить певній особі, яка використовує його виключно. Програма керування та протокол зв'язку зберігаються у вбудованій системі, тоді як GSM-модем підключений до системи через послідовний інтерфейс до комутаційного модуля. GSM-модуль діє як інтерфейс між системою керування розумним будинком та мережею GSM, забезпечує реєстрацію системи в мережі та здійснює передачу даних і зв'язок.

Запропонована в цій роботі система керування розумним будинком

використовує вбудовану систему та клієнтський пристрій, що підтримує веб-браузер, як віддалений термінал для доступу до програмного забезпечення. На рисунку 3.3 показано діаграму активності системи керування розумним будинком:

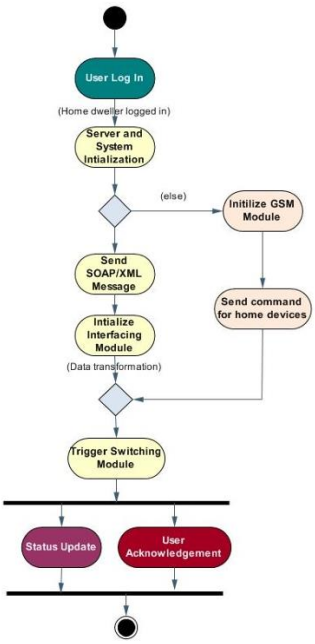

Рисунок 3.3 - Діаграма активності системи управління «розумним будинком»

Система керування розумним будинком активується шляхом встановлення Ethernet-з'єднання за допомогою попередньо визначених мережевих конфігурацій, які автоматично запускають мережеву службу в малорозмірних системах. Після ініціалізації сервера система керування розумним будинком дозволить мешканцям будинків переглядати програмний механізм SOAP, розміщений на сервері. Використовуючи віддалений клієнт, ПК або мобільний пристрій, мешканці будинків увійдуть у систему, використовуючи унікальне ім'я користувача та пароль. Після входу мешканці будинків можуть налаштувати ввімкнення або вимкнення потрібних пристроїв. Після виконання налаштування розпочнеться передача коду, а дані керування будуть передані до інтерфейсного модуля. Отримані мережеві дані через TCP-з'єднання будуть перетворені на послідовні дані та запускатимуть пристрої залежно від вхідного сигналу ВВІМК.

або ВИМК. На рисунку 3.4 нижче показано тіло повідомлення SOAP, що містить опис керування.

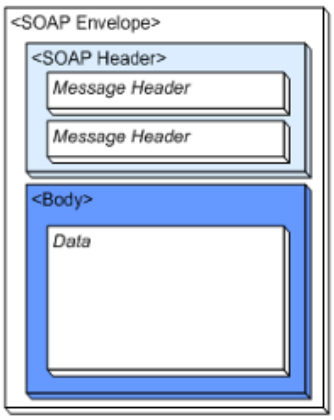

Рисунок 3.4 - Структура повідомлення SOAP

У разі недоступності сервера або з'єднання, система керування розумним будинком може продовжувати працювати та працювати за допомогою альтернативного механізму керування за допомогою GSM-модуля. Використовуючи GSM-модуль, мешканець будинку може надсилати команду у формі служби коротких повідомлень (SMS). Домашні пристрої, підключені до системи керування розумним будинком, можуть не гарантувати успішну роботу системи, оскільки можуть виникати шуми, дефекти або затримки в отриманні вхідних сигналів. Тому для вирішення таких дефектів у комутаційний модуль вбудовано блок діагностики зворотного зв'язку, який забезпечує індикацію стану пристрою після отримання керуючої команди.

Аналіз продуктивності

Протокол SOAP все ще перебуває на початковій стадії розвитку, коли йдеться про міжплатформенну комунікацію для систем розумного дому [40]. SOAP є ідеальним протоколом для таких програм, як система керування розумним будинком, які працюють на одній платформі та в хмарі Ethernet (локальні мережі). Ще однією очевидною причиною використання SOAP у системі керування розумним будинком є відсутність потреби у взаємодії через брандмауери, оскільки програми розумного дому розробляються індивідуально для кожного

мешканця будинку. Для системи керування будинком оцінка часу відгуку важлива для забезпечення безперебійної роботи інтенсивності навантаження під час одночасного запиту під час роботи. На рисунку 3.5 нижче показано результат тестування часу відгуку для повідомлень SOAP.

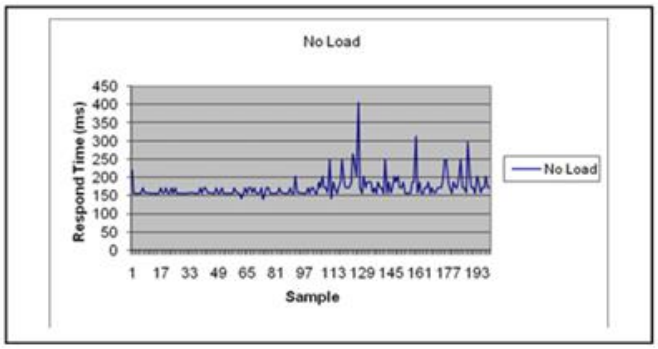

Рисунок 3.5 - Час відгуку при обробці одного повідомлення

На рисунку 3.5 видно, що результат тестування показує перший час відгуку – 219 мс. Під час ініціалізації системи програмний механізм системи керування розумним будинком ініціалізується під час виконання компілятором Just-In-Time, пов'язаним з .NET Framework. Час компіляції призводить до мінімальної затримки на етапі ініціалізації через виконання JIT-компілятора. Загалом, результат тестування показує середню продуктивність SOAP (без навантаження) 172,655 мс. Це отримане середнє значення показує, що продуктивність SOAP у системі керування розумним будинком виправдана та відповідає вимогам щодо керування гетерогенними пристроями. Результати тестування обчислюються з 200 вибірками. Стандартне відхилення становить 30,25534622 мс. Під час тестування не виникло жодних збоїв.

У цій роботі представлено систему керування розумним будинком на основі веб-сервісів, інтегровану платформу, що базується на вбудованій системі. Система розроблена таким чином, що вона забезпечує гнучкість та сумісність протоколу SOAP на основі XML, розширюючи функціональність системи керування розумним будинком на основі веб-сервісів. Запропонована архітектура забезпечує рішення для сумісності та ефективного керування домашніми

пристроями. Розроблена система повністю базується на протоколі SOAP/XML, що долає недоліки CORBA та інших власних стандартів і рішень. SOAP забезпечує масштабованість та гнучкість у забезпеченні підтримки системами розумного будинку існуючих застарілих систем, а також інтеграції нових сервісів або програм. SOAP та веб-сервіси забезпечують життєздатну основу для системи керування розумним будинком, враховуючи розподілений характер домашнього середовища з гетерогенними пристроями. Впровадження SOAP також призведе до прозорого проміжного програмного забезпечення з універсальним підключенням. Найближчим часом веб-сервіси та протокол SOAP матимуть значний вплив на забезпечення сумісності систем розумного будинку. Цікавим завданням для майбутньої роботи буде розробка загального рівня абстракції на основі визначеної схеми для керування гетерогенними системами з попередньо визначеними правилами, що поширюються на мультимедійні пристрої в домашньому середовищі

3.2 Огляд та аналіз програмного забезпечення для розумного будинку

Ми використовуємо кілька типів мікроконтролерів, але можемо розділити їх на дві категорії: одноплатні комп'ютери (SBC) та однокристальні комп'ютери (SCC). Ми використовуємо SCC з підключеними датчиками для передачі даних від датчиків до серверної частини або для керування мікроконтролерами з основного SBC. Наприклад, ESP8266 використовується для надсилання даних з Raspberry Pi Pico через Wi-Fi або для керування освітленням у кімнаті. Він має невеликий бекенд налаштування, де можна вказати мережу Wi-Fi та головний мікроконтролер для надсилання даних. Якщо ми використовуємо адаптер USB-послідовний порт, ми також можемо налаштувати його через послідовний інтерфейс.

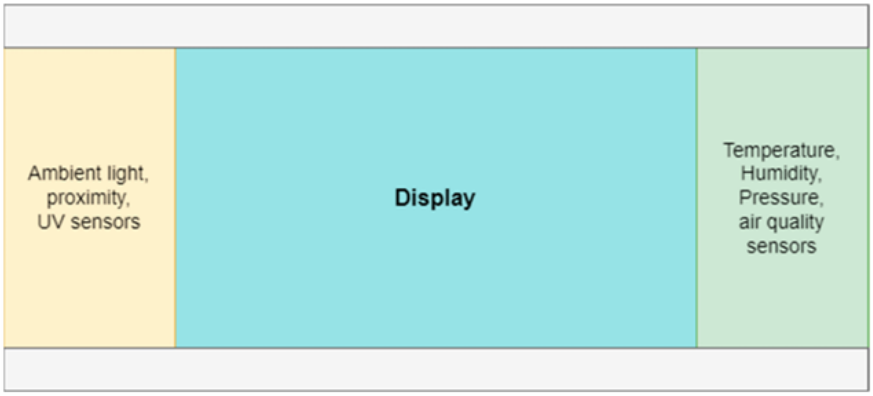

Рисунок 3.6 - Розташування дисплея та блоку датчиків

Бекенд написаний на Python і відповідає за весь зв'язок між основною платою, датчиками, іншими мікроконтролерами та клієнтськими програмами. Ми можемо основні обов'язки, наведені на рисунку 3.7.

У лівій частині діаграми, у зелених прямокутниках, ми маємо фронтенд, який складається з веб-інтерфейсу, що запускається безпосередньо при запуску SBC. Головний екран інтерфейсу представлений на рисунку. 3.7. « Ми можемо бачити налаштовані кімнати у всій системі та мати огляд даних датчиків. Ми також можемо бачити різні параметри мікроконтролера, і в майбутньому користувач зможе його налаштувати.

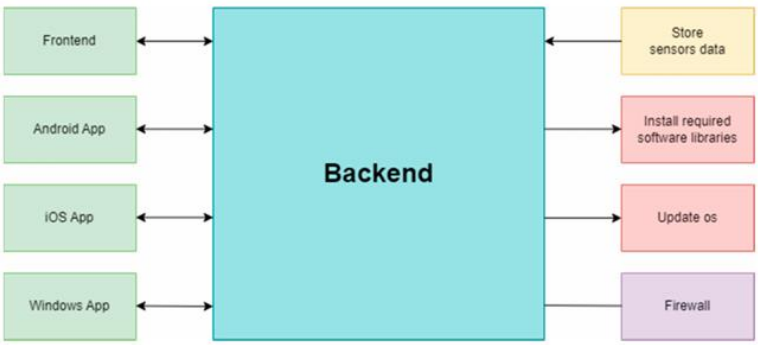

Рисунок 3.7 - Основні функції серверної частини.

На даний момент налаштування мікроконтролера можливе лише з інших клієнтів, таких як Android, iOS або Windows. Архітектуру системи можна вважати загальною та схожою на ту, яку інші дослідники вбудовують у свої системи Інтернету речей [12 , 13] .»

Додатки для Android, iOS та Windows, приклади яких показано на рисунку 3.8 , побудовані за допомогою Xamarin та безпосередньо взаємодіють з основним модулем, який ми налаштовуємо в самій програмі. Основний модуль може бути представлений будь-яким мікроконтролером SBC.

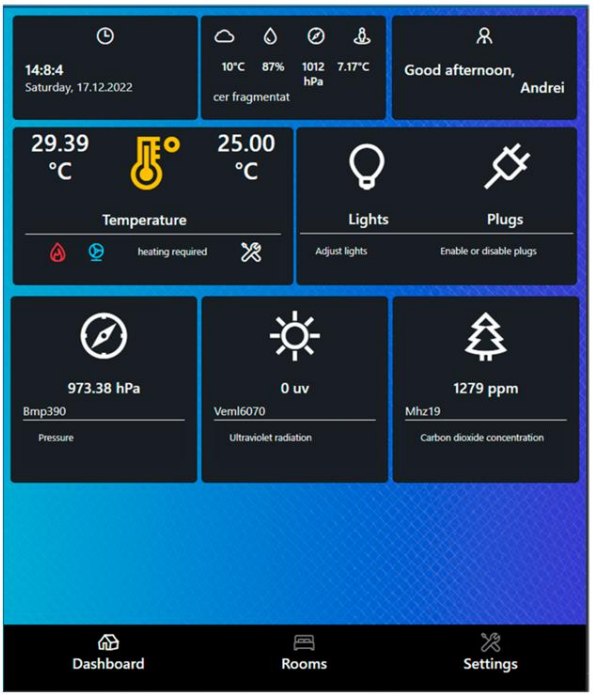

Рисунок 3.8 - Головний екран інтерфейсу.

Мікроконтролер можна використовувати для визначення кількох кімнат, і оскільки ми можемо використовувати бездротові датчики, ми можемо вибирати, які датчики присутні в кімнаті. Бекенд має кінцеву точку перенаправлення, яка відповідає за отримання даних з правильного модуля. Кожен модуль автоматично налаштовує датчик, знайдений на інтерфейсі I2C або підключений безпосередньо через USB або послідовний порт при кожному запуску. У конфігурації у нас є властивість, яка встановлює адресу I2C датчика та IP-адресу модуля, до якого підключено датчик, таким чином ми гарантуємо, що отримуємо дані для правильного датчика, навіть якщо ми використовуємо кінцеву точку перенаправлення.

Однак конфігурація кімнат зберігається лише в головному модулі. У розділі «Налаштування мережі головного пристрою» клієнтської програми ви можете визначити веб-адресу, щоб отримати доступ до серверної частини через

Інтернет. Як результат, усі невизначені кінцеві точки перенаправлятимуть вас на сторінку 404. Як функція безпеки, якщо хтось кілька разів намагається підключитися до несправних кінцевих точок, підключення до цієї машини буде відхилено, і ця особа більше не зможе отримати доступ до модуля. У майбутньому ми також плануємо шифрувати дані запитів для додаткової безпеки

Бекенд запускається під час завантаження системи через завдання «cron», яке виконується *за допомогою* скрипта. Цей скрипт запускає браузер із екраном завантаження та у фоновому режимі перевіряє наявність оновлень операційної системи та вимог до бекенда. Якщо деякі вимоги не виконуються, скрипт автоматично запитує оновлення операційної системи та встановлює необхідні оновлення. У майбутньому ми також завершимо автоматичне оновлення системи та бекенда. Після завершення цієї інструкції нам також потрібен механізм резервного копіювання. Для подальшої розробки ми хотіли б додати такі функції, як камери безпеки. Бекенд повинен мати можливість зберігати зображення з камер безпеки. На даний момент ми можемо бачити пряму трансляцію з камер безпеки за допомогою протоколу RTP.

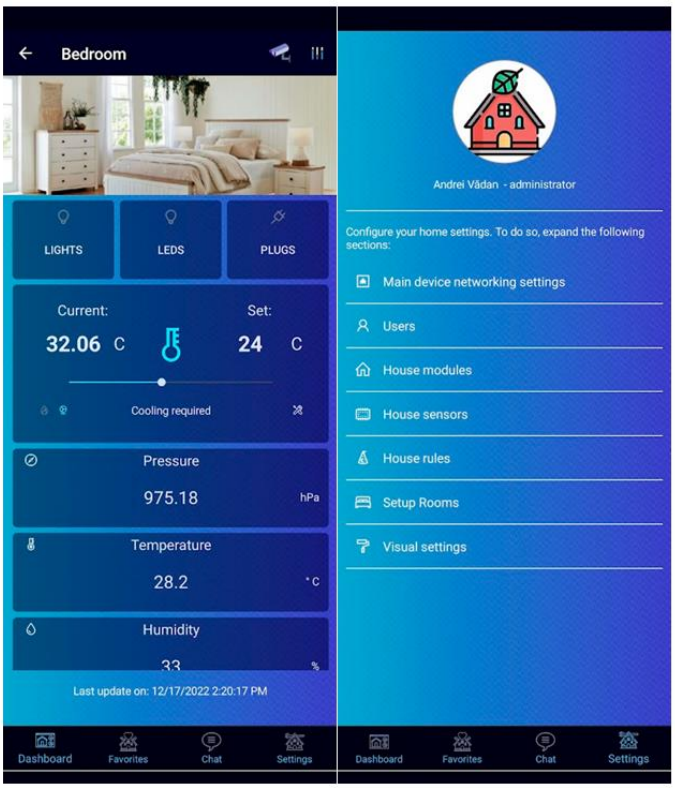

Рисунок 3.9 - Екрани головного меню та налаштувань програми для Android.

Шаблони проектування та фреймворки автоматизації

Загалом, коли ми обговорюємо тестування всієї системи, ми повинні мати на увазі автоматизоване тестування. Складні системи або програми потребують спеціальних інтерфейсів автоматизації, оскільки сучасні рішення для тестування програмного забезпечення можуть не задовольнити всі вимоги. Для мобільних програм, таких як нативні програми для Android або iOS, ми можемо використовувати вбудовані інструменти, які є потужнішими та загалом забезпечують кращі результати, і, звичайно, ми можемо скористатися перевагами тестування білої скриньки або, в найгіршому випадку, тестування сірої скриньки. Коли ми обговорюємо програми, створені для тестування або зв'язку API веб-сайту, ми зазвичай використовуємо методи тестування чорної скриньки. Далі ми опишемо, як створити спеціальний фреймворк для тестування, використовуючи прості методи, які спростять роботу, скоротять час розробки та забезпечать швидші результати під час тестування, незалежно від того, яку технологію ми використовуємо. Звичайно, існує безліч програм, створених для автоматизації, і ви, ймовірно, можете застосувати наші методи, використовуючи такі програми, але ми їх не пробували. Ми продовжимо зосереджуватися на спеціальних фреймворках для автоматизації. Ми обрали нашу систему розумного дому, оскільки вона має веб-інтерфейс, API та мобільний клієнт, а це означає, що ми покажемо вам, як ми можемо застосувати цей шаблон проектування для тестування системи такої ж складності, як наша. Оскільки ми обговорюємо шаблон проектування, ми можемо вибрати будь-яку мову програмування, яку забажаємо, тому ми покажемо вам псевдо-код, який має дати вам уявлення про те, як ви можете застосувати шаблон проектування. Коли ми говоримо про тестування користувацького інтерфейсу, ми маємо на увазі веб-, мобільні або настільні додатки. Спочатку ми представимо *короткий* вступ про шаблони проектування та модель об'єктів сторінки, а потім розглянемо Locstn, Enecute, Expect та його застосування в різних сценаріях.

Під час розробки нових програм або функцій розробники зазвичай намагаються створити архітектуру програми за допомогою шаблонів

проектування. Вперше термін «шаблони проектування» з'явився в будівництві будівель [14] , а в 1994 році чотири розробники написали *книгу* про шаблони проектування в кодуванні [15] . Відтоді, під час створення користувацьких фреймворків автоматизації, розробники автоматизації використовують шаблон проектування «Модель об'єкта сторінки». Цей шаблон проектування має кілька рівнів коду та є дуже складним. Приклад діаграми можна знайти на рисунку

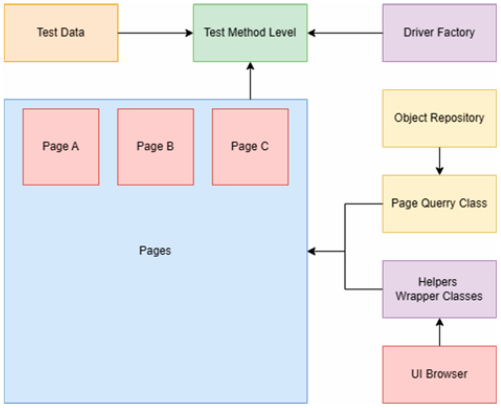

Рисунок 3.10 - Застосування шаблону проектування «Модель об’єкта сторінки» при тестуванні веб-сторінок

Основна проблема цього шаблону проектування полягає в тому, що більшість інженерів з автоматизації реалізують його неправильно, і замість переваг це зазвичай призводить до недоліків. Наприклад, у багатьох випадках ми виявляли, що люди змішують клас репозиторію об'єктів з класами сторінок та дублюють методи зі сторінки А на сторінці В. Інша проблема полягає в тому, що вони також не можуть створювати сторінки компонентів. Коли розробник створює або проектує додаток, він визначає загальні компоненти та визначає їх у коді один раз, а потім повторно використовує їх знову і знову. Проблема автоматизованого тестування полягає в тому, що багатьом інженерам із забезпечення якості автоматизації бракує навичок програмування, особливо об'єктно-орієнтованого програмування. Це означає, що вони не можуть правильно ідентифікувати компоненти, що повторно використовуються програмістами, та створити одну сторінку для компонента, один клас для кроків тощо. Щоб навести

приклад з реального життя, уявіть собі медіаплеєр, який може відтворювати музику з різних джерел. Ви можете знайти такий медіаплеєр, наприклад, в автомобілях. Тестер, як правило, створює окрему сторінку для кожної сторінки джерела медіа, навіть якщо більшість компонентів та поведінки ідентичні. Для вирішення таких задач, замість використання шаблону проектування Page Object Model, ми покажемо вам, як використовувати шаблон проектування Locate, Execute, Expect. Ми розповімо, як застосувати його для тестування системи розумного дому, як та, що представлена у вступі. Ми обговоримо тестування веб-застосунків, тестування інтерфейсу користувача та використання його для тестування API. Перед цим ми коротко обговоримо шаблони проектування та коротко опишемо як Page Object Model, так і Locate, Execute, Expect [16] .

Шаблони проектування можна класифікувати на три типи: поведінкові, креативні та структурні. Шаблони поведінки описують взаємодію між об'єктами та зосереджуються на тому, як об'єкти взаємодіють один з одним. Вони можуть звести складні блок-схеми до простих взаємозв'язків між об'єктами різних класів. Шаблони поведінки стосуються алгоритмів та розподілу обов'язків між об'єктами. Шаблони поведінки описують не лише шаблони об'єктів або класів, але й шаблони зв'язку між ними. Ці шаблони характеризують складний потік керування, який важко відстежувати під час виконання. Вони зміщують вашу увагу з потоку керування, щоб дозволити вам зосередитися лише на тому, як об'єкти взаємопов'язані.

Шаблони поведінкових класів використовують успадкування для розподілу поведінки між класами. Найпоширенішими шаблонами проектування є шаблон методу шаблону та шаблон інтерпретатора. Шаблони створення використовуються для створення об'єктів для відповідного класу, який служить рішенням проблеми. Вони підтримують створення об'єктів у системі та дозволяють створювати об'єкти в системі без необхідності ідентифікувати конкретний тип класу в коді, тому вам не потрібно писати великий, складний код для створення екземпляра об'єкта. Структурні шаблони визначають, як класи та

об'єкти формують більші структури. Структурні шаблони класів використовують успадкування для створення інтерфейсів або реалізацій.

Шаблон проектування моделі об'єкта сторінки

Існує багато концептуальних карт того, як виглядає модель об'єкта сторінки, і в наступних реченнях ми коротко опишемо її компоненти або рівні абстракції коду, які показані на рисунку. 3.9 вище. Рівень методу тестування – це шар, де описується тестовий випадок за допомогою дій та кроків перевірки. Цей файл обробляє всі вхідні дані та відповідає за прив’язку до джерела даних. Фабрика сторінок – це необов’язковий шар, який використовується, якщо ви хочете «попередньо ініціалізувати» елементи інтерфейсу перед виконанням. Це робиться за допомогою анотацій Selenium «FindBy». Для цього вам потрібно створити властивості для елементів класу сторінки та використовувати атрибут, щоб встановити ідентифікатор, який буде використовуватися для пошуку елемента. Таким чином, ви створите альтернативу функції «driver.FindElement(s)» з Selenium. На нашу думку, функція «FindElement» надає вам більше гнучкості, і легше знайти, зачекати, спробувати перевірити, чи елемент відсутній на сторінці, або перебрати список елементів. Класи об’єктів сторінки та їхні методи представляють кожен логічний розділ програми. Наприклад, кожна сторінка з програми повинна мати свій власний клас сторінки. Звичайно, якщо у нас є кілька повторно використовуваних компонентів, ми можемо розділити їх на підкомпоненти, тому ми створюємо сторінку для кожного підкомпонента сторінки. Прикладом такого використання є медіаплеєр. Основні елементи керування однакові; замість створення сторінки для музики та сторінки для відео, ми можемо повторно використовувати кнопки плеєра, і для цього ми визначаємо клас. Driver Factory ініціалізує драйвер для забезпечення підтримки тестування. У Selenium він використовується для ініціалізації браузера або для підключення до віддаленого браузера. У Appium його можна використовувати, наприклад, для підключення до пристрою. Object Repository не завжди реалізується інженерами з автоматизації, але він корисний. На цьому рівні ви абстрагуєте ідентифікатори автоматизації від класу сторінки та зберігаєте всі властивості елементів

інтерфейсу користувача у файлі, який можна запитувати з класу repository of object. Це скорочує час обслуговування у разі оновлення програм. Наявність repository of object додає ще один рівень модульності до вашого рішення та відокремлює локатори об'єктів від класів сторінок. Допоміжні класи або класи-обгортки дуже корисні для створення класу, де ви пишете обгортки або різні функції. Впровадження обгорток може допомогти користувачам керувати обробкою помилок, створювати розумніші очікування, тайм-аути, безпечні режими відмов або повторні спроби, а також реєструвати кроки або помилки. На рівні джерела дельта-тестування можна визначити джерело даних для методу тестування та використовувати тестові дані з файлу, такого як CSV, XML в Excel або навіть база даних SQL. Тестування на основі даних є життєво важливим для охоплення більшої області шляхом виконання кількох ітерацій тестування одного й того ж тестового випадку з різними сценаріями даних, що заощаджує код і час [18] .

Знайти, виконати, очікувати шаблон проектування

підпису Locate, Exscuts, Expect L має менше шарів , ніж POM. У нас є шар під назвою L Locate, де ми визначаємо методи для ментифікації елементів інтерфейсу користувача та автоматизації елементів Ш. Рекомендується створювати файли для кожної сторінки та встановлювати ідентифікатори на сторінці або створювати вкладені довідники для елементів інтерфейсу користувача . Щоб навести конкретний приклад , якщо ви повернетеся до рисунка 3.8, ви побачите деякі екрани мобільних застосунків . Програма «Шини» має чотири основні екрани: «Панель інструментів», «Улюблені», «Чад» та «Налаштування». Коли ми натискаємо на кімнату на сторінці «Панель інструментів», ми відкриваємо сторінку, яка відображає інформацію, зібрану з датчиків, і ми можемо виконувати різні дії та відкривати інші сторінки. Для кожної сторінки в програмі ми створюємо клас сторінки в коді та описуємо ідентифікатор автоматизації. Для меню ми можемо створити окрему сторінку, як ми зазвичай робимо за допомогою POM. У класі «Locate» ми створимо вкладений словник, де основний словник може мати кілька підсловників. Шар «Execute»

містить клас, який повинен містити основні методи, що виконують дії або обробляють очікування. Якщо ми тестуємо системи та маємо велику кількість методів, ми можемо розділити код на кілька класів на основі сторінок, на яких ми використовуємо програми. Шар «Expert» схожий на «Виконати», але тут ми розміщуємо лише кроки верифікації. Важливо зазначити, що рівень верифікації не включає методи з рівня дій, але рівень «Виконати» може включати методи з рівня «Експерт». Таким чином, ми уникаємо циклічних залежностей. На рисунку са ми можемо побачити загальну діаграму шаблону проектування LEE.

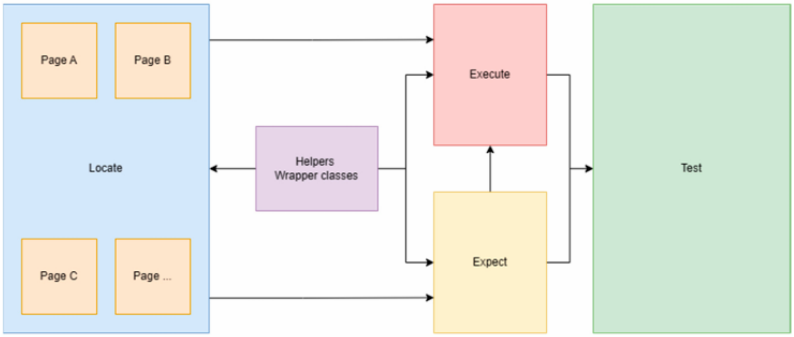

Рисунок 3.11 - Схема шаблону проектування LEE.

Якщо це необхідно , ми можемо включити допоміжні класи, але вони не вважаються частиною LEE. Якщо ми хочемо поєднати цей шаблон проектування, ми також можемо використовувати його разом із POM без жодних проблем. Це пояснюється тим, що шар «Locate» в основному представлений репозиторієм об'єктів та класами сторінок, нам потрібно повторно використовувати методи, визначені в класі дій та верифікації. Таким чином, методи з класів об'єктів сторінок використовують узагальнені методи, і обслуговування коду набагато простіше. Однак комбінація цих шаблонів проектування не є ідеальною, оскільки при використанні з комбінацією обгорток вони можуть призвести до багатьох проблем, особливо з інтерпретованими мовами програмування. Крім того, коли ми намагаємося налагоджувати, ми переходимо від методу до методу, і для початківців це може бути клопітно.

3.3 Тестування системи та оцінка її ефективності

У цьому абзаці ми продемонструємо, як застосувати шаблон проектування Locate, Execute, Expect у загальному фреймворку Єог. Тестування інтерфейсу користувача загалом. Для цього прикладу нам слід уявити собі просту сторінку входу в програму (ERF). Що має бути видно на сторінці входу? Wr повинен мати r ЄєхЄ поле для введення імені користувача та пароля, а також кнопка для входу, як мінімум. Ми спробували навести базовий приклад на рисунку 3.12. Коли ми натискаємо на кнопку «Вхід», якщо дані, які ми надали в тесті, працюють, ми повинні перенаправити на іншу сторінку, а якщо наші облікові дані неправильні, ми повинні побачити спливаюче вікно з повідомленням про помилку

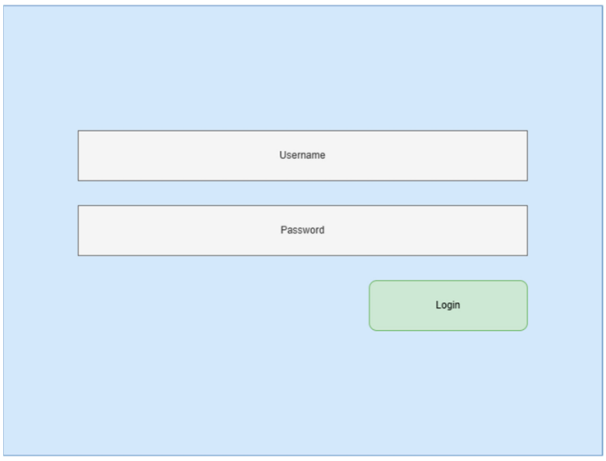

Рисунок 3.12 - Приклад сторінки входу.

Ми застосуємо наступний тестовий випадок для тестування цієї сторінки: користувач введе 'my_email@email.com' у поле 'Ім'я користувача', 'mypassword' у поле 'Пароль' та натисне кнопку 'Увійти'. Оскільки облікові дані не працюють, ми повинні побачити спливаюче вікно з повідомленням: 'Вибачте, ваші облікові дані не працюють!'. У перекладі на Codo це має призвести до класу констант 'Test variables', де ми повинні побачити ім'я користувача, пароль та повідомлення про помилку. Шар Locate повинен містити два класи : один пов'язаний зі сторінкою входу, а інший - зі спливаючою сторінкою помилки. На цих сторінках ми визначаємо ідентифікатори автоматизації або локатори. Шар Execute повинен

містити клас Actions, де ми повинні визначити два методи: 'click_on_element' та 'type_text_into_textfield'. На шарі Expect ми можемо створити клас Verifications, де нам також знадобляться два методи: 'compare_text' та 'check_if_element_is_displayed'. У лістингу 3.1, наведено псевдокод реалізації з використанням підходу Selenium [16] .

Лістинг 3.1 - Псевдокод прикладу тесту входу в систему.

```
----------Test variables class----------
email_text = "my_email@email.com"
password_text = "mypassword"
login_error_message = "Sorry, your credentials did not work!"
----------Login identifiers class----------
email_textbox = webdriver.find_element(By.ID, "email_textbox")
password_textbox = webdriver.find_element(By.ID, "pass_textbox")
login_button = webdriver.find_element(By.ID, "login_button")
----------Pop-up identifiers class----------
popup_error_root = webdriver.find_element(By.ID, "popup_root")
popup_message = webdriver.find_element(By.ID, "error_message")
----------Execute layer / Actions class----------
def click_on_element(element: UiElement):
        element.click()
def type_text_into_textfield(element: UiElement, text: String):
        element.text = text
----------Expect layer / Verifications class ----------
def check_if_element_is_displayed(element: UiElement):
        assert element.is_displayed() , "Element {'element'} is not displayed"
def compare_text(element: UiElement, required_text):
        assert element.text != required_text ,
                "Text on {'element'} does not match! \n
                Current text found: {'element.text'}, \n
                Expected text: {'requiredtext'}"
----------Login test suite----------
#Assuming we start directly from login page
def test_login_with_bad_credentials():
        execute.type_text_into_textfield(LoginLocators.email_textbox, Constants.email_text)
        execute.type_text_into_textfield(LoginLocators.password_textbox, Constants.password_text)
        expect.compare_text(LoginLocators.email_textbox, Constants.email_text)
        expect.compare_text(LoginLocators.password_textbox, Constants.password_text)
        execute.click_on_element(LoginLocators.login_button)
        expect.check_if_element_is_displayed(PopupLocators.popup_error_root)
```

У нашому псевдокоді, в зоні визначення тесту, ви можете помітити послідовність ключових слів «execute» та «expect», за якими йдуть методи. Якщо ми використовуємо правильні ключові слова в іменуванні методів, сам тест стає легким для читання навіть для нетехнічних фахівців. Ще один важливий момент, який слід зазначити, це те, що тест повинен завершуватися перевіркою, оскільки у нас є перелік кроків, і кінцева перевірка має бути метою тестового випадку. Використовуючи цей підхід, ви створюєте методи перевірки та переконуєтесь у методах, що перевірка виконується. Після того, як метод було перевірено та використано, ми знаємо, що метод працює правильно, і ми можемо

використовувати його знову і знову. Наприклад, коли ми використовуємо POM, однією з поширених помилок є те, що інженер з автоматизації копіює метод з іншого класу та дублює код, і зазвичай вони вносять зміни лише в одному місці. Це можна виправити, інтегрувавши LEE з POM.

Застосування шаблону проектування «Визначити, виконати, очікувати» за допомогою Gherkin , як виглядає простий тест входу за допомогою псевдокоду. З LEE у нас немає обмежень щодо мов програмування , і ми можемо за бажанням використовувати поточний : міжнародний підхід BDD , німецький . Це швидко використовується в більшості користувацьких автоматизованих фреймворків , і одна з причин цього полягає в тому , що це значно полегшує читання тестового випадку. Якщо ми хочемо застосувати це до нашого тесту з лістинга 3.1 , нам потрібно створити додаткові файли для покрокової реалізації та змінити спосіб визначення тестового випадку. Ми можемо використовувати всі інші методи, оскільки вони можуть залишатися незмінними . Приклад такого підходу показано на рисунку лістингу 3.1.

Лістинг 3.2 - Приклад тесту входу в систему з використанням Gherkin

```
----------Login steps----------
@step ('User inputs the credentials')
def input_credentials():
        execute.type_text_into_textfield(LoginLocators.email_textbox, Constants.email)
        execute.type_text_into_textfield(LoginLocators.password_textbox, Constants.password)
...
@step ("Check text: {required_text}")
def compare_popup_text(required_text):
        expect.compare_text(PopupLocators.popup_message,  required_text)
----------Login test suite feature file----------
#Assuming we start directly from login page
Feature: Login functionality
Scenario: Test login with bad credentials
        Given User inputs the credentials
        Then The user checks the credentials
        When The user clicks on login button
        And The user checks the error popup is displayed
        Then Check text: "Sorry, your credentials did not work!"
```

Коли ми тестуємо систему або дуже складну програму, кроки слід розділити на два файли, один з яких містить дії, а інший – перевірки. Крім того, нам слід створювати файли для кроків залежно від того, яку функціональність ми тестуємо. Це допоможе зменшити кількість рядків коду у файлах, а також покращити якість коду. Замість того, щоб визначати методи в класах, ми можемо створювати узагальнені елементи. Одна з проблем із узагальненими кроками полягає в тому, що ми використовуємо їх на різних сторінках, і нам потрібно надавати елементи, з якими ми взаємодіємо. У цьому випадку у нас буде кілька умов if-else. З нашої точки зору, поєднання прикладу з лістинга 3.1 з корнішоном, як показано на лістингу 3.2– це найкращий підхід, оскільки він забезпечує легкість читання коду, мінімальну кількість рядків коду, і ми можемо визначити конкретні кроки. Це буде краще для молодших інженерів з автоматизації тестування.

Застосування шаблону проектування «Locate, Execute, Expect» для тестування API

Ми пояснили, як використовувати шаблон проектування LEE для тестування інтерфейсу користувача. Ми можемо застосувати цей шаблон проектування для тестування API. Коли ми дивимося на адресу веб-сайту в нашому браузері, ми помічаємо, що більшість веб-сайтів з кількома сторінками мають URL-адресу, а за нею йде URI або параметри. Якщо ми подивимося на те, як працюють запити, то коли ми виконуємо запит, ми очікуємо результат, незалежно від того, який тип запиту ми виконуємо. Якщо ми уважно подивимося, ми можемо помітити кілька змінних для наших тестів. Ми можемо визначити URL-адресу, URI або параметри на рівні Locate, методи, які відповідають за виконання запиту, на рівні Execute, та перевірку відповіді на рівні Expect. На лістингу 3.3 ми продемонстрували приклад тестування запиту наш логін test.

Лістинг 3.3 - Приклад тесту API для перевірки входу з неправильними обліковими даними.

```
----------Locate class----------
url = "http://127.0.0.1"
uri = "/login"
----------Execute class----------
def perform_request(url, uri="", headers="", params="", data="", method=GET):
        return request.execute(method, url+uri, params, headers, data)
----------Expect class----------
def check_response_code_and_message(request_result, expected_response_code, expected_message):
        assert request_result.response_code = expected_response_code,
                "Response code is incorrect! \n" + Expected: { expected_response_code } \n" +
                "Result: { request_result.response_code }"
                …
----------Login API test----------
def test_login_with_bad_credentials():
        response = execute.perform_request (url, uri, headers)
        expect. check_response_code_and_message (response, 200, "OK")
```

У нашому прикладі з лістинга 3.3, ми створюємо метод для перевірки коду відповіді та повернутого повідомлення. У реальному сценарії бажано створити метод для кожної речі, яку ми хочемо перевірити. У нашому тесті у нас є змінна «відповідь», яка містить відповідь, надану виконанням запиту. Якщо ми хочемо мати послідовність Execute та Expect, нам потрібно створити механізм для позбавлення від цієї змінної. Один із способів, наприклад, полягає у створенні глобальної змінної або словника на рівні Execute та збереженні даних у цьому словнику, а потім наданні даних на рівень Expect. Якщо ми використовуватимемо Python у поєднанні з Gherkin для виконання тестування API за шаблоном проектування LEE, у нас не буде цієї проблеми, оскільки ми можемо використовувати змінну контексту, яка доступна глобально.

Застосування шаблону проектування «Locate, Execute, Expect» для Ul-тестування кросплатформних мобільних застосунків

Ми можемо успішно використовувати цей шаблон проектування для тестування мобільних додатків, незалежно від того, яку мову програмування ми використовуємо. Ми не обмежуємося тестуванням білої скриньки, ми також можемо проводити тестування сірої або чорної скриньки, залежно від технології, яку ми використовуємо для розробки додатка. Наприклад, ми створили рішення для тестування додатків для Android за допомогою Kotlin та Java, де у нас був

підхід сірої скриньки. Для тестування iOS це більше схоже на підхід чорної скриньки, оскільки саме так працює XCUITest. Однак у цьому випадку ми використовували Swift. На рисунку 3.13 у нас є Solution Explorer з Visual Studio, де ви можете побачити основні класи, що використовуються для реалізації LEE. У правій частині рисунка 3.13 наведено просту реалізацію тесту [16] .

Більш детальну інформацію можна знайти в розділі «Знайти, виконати, очікувати шаблон проектування» [16] у розділі IV, розділі C. Ми можемо піти далі та використовувати цей шаблон проектування складним чином, тестуючи вбудовані системи, використовуючи одне рішення, але ми пояснимо, як це зробити, у наступному розділі.

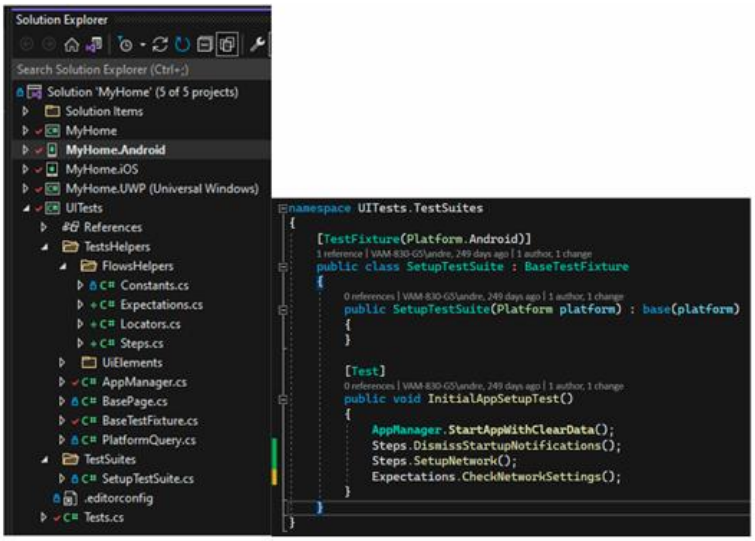

Рисунок 3.13 - Приклад застосування LEE в Xamarin

Застосування шаблону проектування «Locate, Execute, Expect» у тестуванні вбудованих систем

Ми розробили рішення на Python для тестування системи HMI для автомобільної інформаційно-розважальної системи. Щоб успішно виконати це завдання, ми розробили невеликі модулі, які відповідають за різні завдання. Наприклад, ми розробили модуль для UiAutomator, який використовується для взаємодії з пристроями Android, один для системного бекенду, один для користувацького інтерфейсу, який є веб-орієнтованим, та обгортку для Appium

для взаємодії з пристроями iOS. Ми можемо застосувати той самий принцип до систем Інтернету речей, незалежно від того, яку технологію ми використовуємо для розробки користувацького інтерфейсу.

Ми створили структуру дуже простим способом; ми створили спеціальні папки для Gherkin та Behave для підтримки запуску тестів, що використовують ці технології, тому у нас є папки «features» (функції), де знаходяться тестові випадки у файлах feature. Крім того, для цього потрібен файл «environment» (середовище), який використовується виконавцем тестів для запуску різних подій, таких як перегляд середовища перед виконанням тесту або після виконання тесту. Файли feature містять кроки, і в основній папці нам потрібно створити папку для кроків. На діаграмі з рисунка 3.14, ми назвали їх «кроки Gherkin». Ці елементи можна називати як завгодно, але ми рекомендуємо використовувати домовленість про іменування, засновану на функціональності, і якщо у вас є якісь функції, ви можете розділити їх на два типи: один файл повинен містити кроки, пов'язані з діями, один — кроки перевірки; інший — кроки перевірки. У елементах SEP ви можете безпосередньо використовувати загальні методи, визначені для шарів «Execute» або «Expect», і реалізовувати лише базову логіку, якщо це необхідно. Поза папкою «features» ви можете містити структуру так, як вам подобається. У нашому випадку ми це дуже чітко вказали. У нас є папка для констант, де ми визначаємо різні константи, що використовуються в проєкті. Окрім цієї папки, у нас є одна з різними допоміжними класами, одна з базовими рівнями для візуальної перевірки та власне «Репозиторій об'єктів» або шар «Locate». У цій папці у нас є невелика структура; Для вбудованої системи ми створюємо окрему підпапку, і ми зробили те саме для мобільних платформ. Наприклад, все змінюється від однієї версії Android до іншої, і це стосується також iOS. Оскільки користувачі iOS зазвичай отримують кілька оновлень з часом, ми підтримуємо лише останню версію iOS і постійно оновлюємо ідентифікатори автоматизації. Основна проблема полягає в Android, де ми підтримуємо всі системи від Android 10 до останньої, Android 13. У цьому випадку замість того, щоб створювати структуру для кожної версії операційної системи, ми створили загальний клас,

який використовується загалом, і якщо ідентифікатор автоматизації відрізняється, ми знаємо, яка версія Android встановлена на нашому пристрої, і запитуємо

змінна з класу, розробленого для цієї конкретної версії Android. У нашій кореневій папці є різні утиліти-скрипти для синхронізації результатів тестів або налаштування проекту, наприклад, та для взаємодії з backdnd.

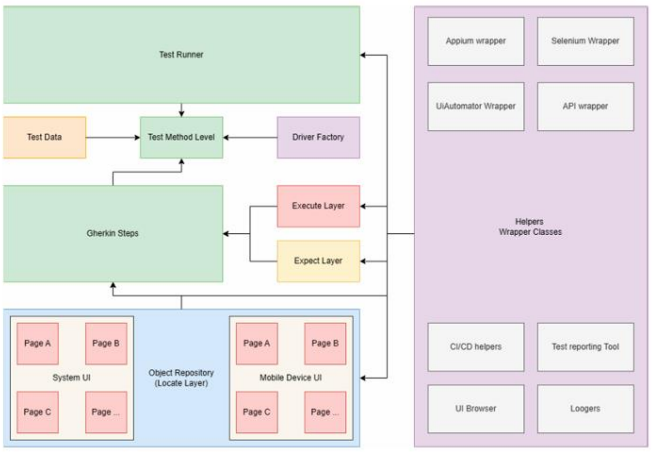

Рисунок 3.14 - Приклад схеми користувацької платформи автоматизації.

На малюнку 3.14 ми навели приклад схеми нашого фреймворку автоматизації. Оскільки ми використовуємо Python та віртуальні середовища, обгортки фактично встановлені в папці 'venv', але ми можемо розглядати їх як частину проекту, оскільки ці модулі були розроблені спеціально для такого виду діяльності, і без них ми не можемо виконувати тестування.

На рисунку 3.15 ми проілюстрували код класу poge з рівня абстракції коду «Locate», а на рисунку 3.16 те саме; закриття за допомогою POM. Знімки екрана зроблені на різних етапах розробки нашої системи автоматизації та повинні чітко показувати, як шаблон проектування LEE зменшує злиття речовин щодо правильного використання POM. Зображення коду після використання LEE показано на рисунку. 3.17. Ми застосували подібний підхід до тестування нашої системи розумного будинку, і ефективність використання LEE зростає зі зростанням складності проекту.

Основна відмінність полягає у визначенні методу всередині класу сторінки. Використання LEE дозволяє уникнути цього, а код набагато чистіший

та легший для читання. На рисунку 3.16 , у нас є чотири методи, які по суті роблять одне й те саме – виконують клік на певному елементі. Уявіть, що ми тестуємо систему з більш ніж 100 переглядами або сторінками, і ми використовуємо підхід, показаний на рисунку 17. Це чудовий приклад того, як ми можемо збільшити можливість повторного використання методів за допомогою LEE.

```
class GlobalSearchResult(Component):
    _root_locator = Locator(By.XPATH, '//*[@id="GlobalSearchResultsScreen"]')
    loading_spinner_locator = Locator(By.XPATH, '//*[@id="HeadLine"]//pag3-loading-spinner')
    cancel_in_speller_locator = Locator(By.XPATH, '//*[contains(@class,"icon-delete_entry")]')
    result_list_locator = Locator(By.XPATH, '//*[@id="DetachedHeadline"]')
    ok_button_locator = Locator(By.XPATH, '//*[contains(@class,"key--ok")]')
    result_list_page_locator = Locator(By.XPATH, '//*[contains(@class, "list__content-container")]')
    first_result_in_speller = Locator(By.XPATH,
                                      '//*[@id="SplitScreenContainerInteraction"]/div/pag3-speller/pag3-i18n-keyboard/div[1]/p[1]')
    results_in_speller = Locator(By.XPATH,
                                 '//*[@id="SplitScreenContainerInteraction"]/div/pag3-speller/pag3-i18n-keyboard/div[1]/p')

    @property
    def cancel_in_speller(self) -> ElementProxy:...

    @property
    def result_in_speller(self) -> ElementProxy:...

    @property
    def result_list_headline(self) -> ElementProxy:...

    @property
    def ok_button(self) -> ElementProxy:...

    @property
    def result_page(self) -> ElementProxy:...

    @property
    def speller_result_list(self) -> Iterator[ElementProxy]:...
```

Рисунок 3.15 - Приклад визначення класу сторінки з використанням LEE.

На рисунку 3.17 , ви можете помітити, як ми застосували LEE в одному з наших фреймворків автоматизації. Однак, щоб переконатися в хорошій стандартизації коду, наприклад, назву першої функції слід перейменувати. Функції або методи, що описують дії, повинні мати у своїй назві, загалом, тип дії, елемент та місце виконання. Методи верифікації повинні містити, загалом, лише верифікації, а другий метод є прикладом неправильного використання. 'drag_element_by_text' та 'take_screenshot' можна визначити у два різних кроки, а не включати їх у крок верифікації. Ми можемо однозначно погодитися, що набагато легше виявити помилки у визначеннях кроків, коли ми застосовуємо LEE в проекті.

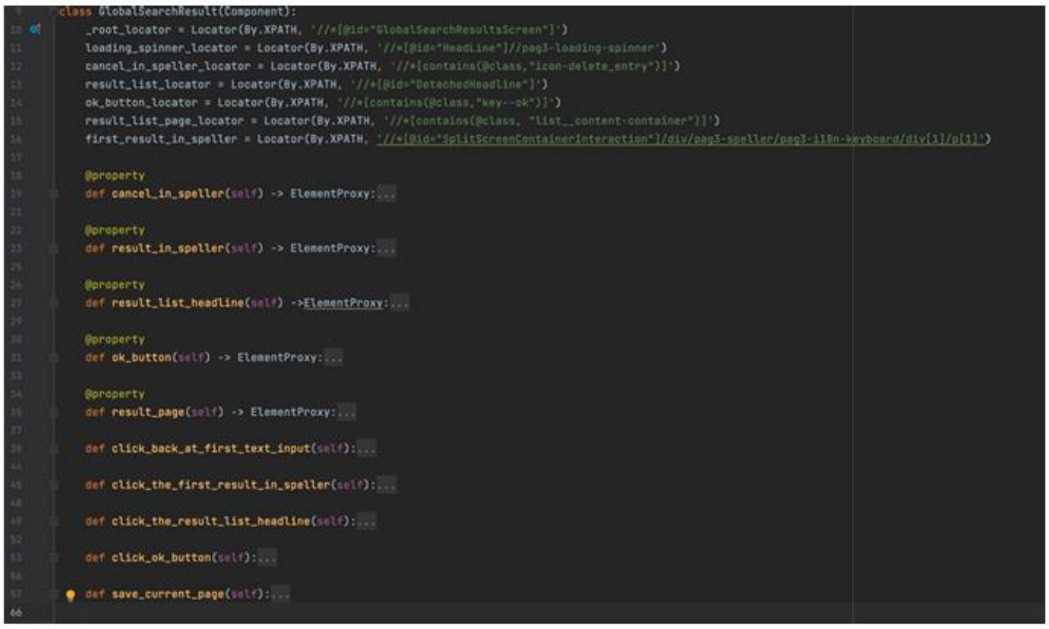

Рисунок 3.16 - Приклад визначення класу сторінки за допомогою POM. F

З 2018 року ми застосовували шаблон проектування LEE для тестування кількох систем та додатків. Початкова робота проводилася з тестування мобільних додатків для Android та iOS з використанням нативних рішень, реалізованих відповідно на Kotlin та Java, з використанням Swift. Над початковим проектом працювала команда з п'яти-десяти інженерів з контролю якості для ручного тестування та двох інженерів з автоматизації. Застосовуючи шаблон проектування у фреймворку автоматизації, на той момент ми створили фреймворк автоматизації, який можуть легко використовувати тестувальники, що виконують ручну роботу, і який допомагає менеджерам розуміти, що ми тестуємо та які результати. Деякі тестувальники, що виконують ручну роботу, почали писати автоматизовані тести, і їм це дуже легко вдавалося. Деякі з них почали програмувати вперше та стали самостійними у написанні тестів через 3 місяці. Це стало можливим завдяки тому, що ми розробили більшість необхідних методів, а тестувальники, що виконують ручну роботу, навчилися їх використовувати.

Потім ми почали працювати над проектом Інтернету речей, коротко представленим у цій роботі, щоб дослідити, як ми можемо застосувати цей шаблон проектування до інших сценаріїв, таких як тестування бекенду та веб-інтерфейсу користувача. Проект також включав клієнтський застосунок, який є

кросплатформним, і ми успішно протестували шаблон проектування, використовуючи технології Xamarin та C#, а не лише веб-версію та API.

```python
@step('the user taps {option} on right bar')
def tap_right_bar(context, option):
    right_launcher_bar = RightLauncherBar(context.locate)
    if option == "notification":
        context.execute.click_on_element(right_launcher_bar.notification_element)
        context.logger.info(f'Click BTN_notification.')
    elif option == "devices":
        context.execute.click_on_element(right_launcher_bar.devices_element)
        context.logger.info(f'Click BTN_devices.')
    else:
        context.logger.error(f'{option} was not found!')
        raise Exception(f'{option.capitalize()} was not found!')


@step("the user will verify that the {element} app is displayed on navigation bar")
def verify_element_displayed_navi_bar(context, element):
    context.execute.drag_element_by_text(element)
    context.execute.take_screenshot(name=f"{element}_mtb")
    context.expect.verify_shape(element=f"{element}_mtb")


@step("the user will verify that the amount of unseen notification are displayed on notification
def verify_unseen_notification(context):
    launcher = LauncherPage(context.locate)
    counter = launcher.notification_count_element.get_text()
    notification_launcher_tile = int(counter)
    assert notification_launcher_tile >= 0, f"Notification with title: {notification_launcher_tile
```

Рисунок 3.17 - Приклад правильного визначення кроків за допомогою LEE.

Під час роботи над проєктом Інтернету речей ми застосували шаблон проектування для тестування вбудованих автомобільних систем, а точніше, інформаційно-розважальної системи виробника автомобілів. У цьому проєкті попередні інженери неправильно застосували шаблон проектування POM, що призвело до великої кількості дублікатів коду, який було важко підтримувати. Оскільки проєкт тестує всю систему, існує понад 100 сторінок інтерфейсу користувача, які іноді містять більше 20 елементів та 10 методів. Виробник встановлює одну й ту саму інформаційно-розважальну систему в кілька типів автомобілів, і тести виконувалися для кожного типу автомобіля. Використовуючи цей підхід, нам вдалося об'єднати весь код в один фрагмент і значно скоротити час обслуговування. Важливим аспектом є те, що при автоматизації нового тесту час економиться, і тест у більшості випадків можна застосовувати до всіх типів

автомобілів, тому ми зменшили витрати на розробку більш ніж на 5%. Застосовуючи LEE, нам вдалося зробити код повторно використовуваним та очищеним, легше інтегрувати нові команди та отримати плавніші переходи від однієї версії програмного забезпечення до іншої. Однак у цьому проєкті у нас виникли деякі труднощі через обгортки, створені над універсальними функціями Selenium та Appium, а також той факт, що Python є інтерпретованою мовою, і нам потрібно було гарантувати, що ми використовуємо правильний тип об'єктів. Загалом, застосування його до нових або існуючих проєктів може принести переваги в довгостроковій перспективі. Ми використовували обгортки, створені спочатку на другому проєкті, і результат полягав у тому, що ми отримали зрілий фреймворк, побудований за кілька днів, та повторно використали універсальні методи для реалізації понад 30 тестів за місяць у режимі всього лише одним інженером з автоматизації. Це означає, що після створення основної структури за допомогою LEE ви можете повторно застосувати її до будь-якого проєкту, використовуючи ту саму технологію, незалежно від її складності.

(Перевагою колег було те, що ми можемо застосувати цей принцип до розробки програмного забезпечення загалом, але ми не змогли визначити сценарій, де він перевершує інших і може бути кращим за поточні шаблони проектування, які ми вже маємо в своєму розпорядженні. Ви можете зіткнутися з проблемами, якщо маєте схильність створювати кілька рівнів абстракції та використовуєте інтерпретовані мови програмування, такі як Python, але ці проблеми виникають рідко і їх можна легко вирішити, визначивши тип параметрів або правильно застосувавши принципи об'єктно-орієнтованого програмування.)

Шаблон проєктування LEE дуже добре застосовується в тестуванні, особливо на користувацьких фреймворках автоматизації, оскільки ми можемо використовувати його в найпоширеніших сценаріях, таких як тестування веб-сайту, мобільного додатку або тестування API з високою ефективністю. Він допомагає інженерам з автоматизації підтримувати чистоту коду та спрощувати інтеграцію новачків. Він може легко замінити POM або інші шаблони проєктування, що використовуються в поточних проектах, і підвищує можливість

повторного використання поширених методів. Інтеграція з системами безперервної інтеграції або безперервної доставки дуже проста, оскільки не потребує додаткових плагінів.

У майбутньому планується розширити основне використання шаблону проектування та дослідити, як його можна застосувати, наприклад, у модульному тестуванні. Крім того, щоб довести ефективність фреймворку, що використовує цей шаблон проектування, хотілося б контролювати деякі параметри в команді інженерів з автоматизації тестування. Для цього повинні враховувати найважливіші аспекти, такі як час, необхідний для реалізації тестового випадку, час, витрачений на обслуговування, та можливість повторного використання методів, наприклад. Також хочеться дослідити, чи є вплив на тип мережі, який ми використовуємо в нашій автоматизації розумного будинку, подібно до того, що виявили автори «Порівняльної оцінки продуктивності бездротових мереж ZigBee та LoRa в середовищі будівлі» [20], а також вплив мови програмування, яку ми використовували для тестування тієї ж системи, намагаючись застосувати алгоритми та методи, представлені в «Оцінці продуктивності мов програмування C/C++, MicroPython, Rust та TinyGo на мікроконтролері ESP32» [21]. На ринку існує кілька програм для автоматизованого тестування, і тому хочеться дослідити, чи можна застосувати цей шаблон проектування у вже створеній програмі для автоматизованого тестування.

3.4 Висновок по розділу

У третьому розділі розглянуто практичні аспекти проєктування, реалізації та оцінювання програмного забезпечення для систем розумного будинку. Проаналізовано архітектуру системи, виконано огляд програмних рішень та досліджено особливості їх інтеграції з апаратними компонентами. Проведене тестування дозволило оцінити ефективність роботи системи, її продуктивність та

надійність. Отримані результати підтвердили доцільність використання сучасних програмних засобів для автоматизації процесів управління розумним будинком і підвищення комфорту користувачів.

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено та розроблено програмне забезпечення для систем розумного будинку, що є комплексним процесом, спрямованим на створення ефективного, надійного та зручного рішення для автоматизації житлового середовища. У роботі проаналізовано сучасні підходи до побудови систем розумного будинку, розглянуто їх

архітектурні особливості, а також досліджено переваги й недоліки різних технологій і засобів взаємодії програмного забезпечення з апаратними компонентами. У результаті реалізовано функціональні можливості системи, які відповідають сучасним вимогам до якості, безпеки та масштабованості. Процес охопив аналіз вимог, моделювання архітектури, розробку програмних компонентів, тестування та оцінку ефективності системи.

На початковому етапі було проведено аналіз предметної області, визначено функціональні та нефункціональні вимоги до системи розумного будинку. До функціональних можливостей віднесено управління освітленням, клімат-контролем, системами безпеки та іншими підсистемами, а також підтримку автоматизованих сценаріїв роботи. Нефункціональні вимоги включали забезпечення надійності, масштабованості, енергоефективності та захисту даних користувача. Моделювання системи дозволило структурувати процеси взаємодії між користувачем, програмним забезпеченням і апаратними компонентами (датчиками та виконавчими механізмами), а також визначити ключові аспекти оптимізації роботи системи.

Реалізація програмного забезпечення охопила розробку архітектури системи розумного будинку, що забезпечує інтеграцію різноманітних пристроїв через бездротові технології. Було створено програмні модулі для збору та обробки даних із датчиків, управління виконавчими пристроями та взаємодії з користувачем через інтерфейс. Особливу увагу приділено забезпеченню стабільної роботи системи, обробці помилок і захисту даних, що є критично важливими для систем автоматизації житлового середовища.

У процесі роботи проведено тестування програмного забезпечення, яке включало перевірку функціональності, продуктивності та надійності системи. Було виявлено потенційні проблеми, пов’язані із затримками передачі даних та стабільністю взаємодії пристроїв, що дозволило оптимізувати алгоритми обробки даних і покращити загальну ефективність системи. Застосування шаблонів проєктування сприяло підвищенню гнучкості та підтримуваності програмного коду.

Загалом розроблена система відповідає поставленим цілям і дозволяє ефективно керувати компонентами розумного будинку, забезпечуючи автоматизацію процесів і підвищення комфорту користувача. Перевагами запропонованого підходу є модульність архітектури, можливість масштабування, інтеграція з різними типами пристроїв та зручний інтерфейс управління. Водночас існують певні обмеження, зокрема необхідність подальшого вдосконалення механізмів безпеки та оптимізації роботи системи при великій кількості підключених пристроїв.

У перспективі розроблена система може бути розширена шляхом інтеграції з хмарними сервісами, впровадження аналітики даних, використання технологій штучного інтелекту для адаптивного управління та підтримки мобільних платформ. Це дозволить підвищити рівень автономності системи, розширити її функціональні можливості та забезпечити більш ефективне використання ресурсів у системах розумного будинку.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Angular Documentation. (2025). angular.io. https://angular.io/docs
2. Azure IoT Hub Documentation. (2025). learn.microsoft.com. https://learn.microsoft.com/en-us/azure/iot-hub/
3. Azure IoT Central. (2025). learn.microsoft.com. https://learn.microsoft.com/en-us/azure/iot-central/
4. Azure DevOps CI/CD Pipelines. (2025). learn.microsoft.com. https://learn.microsoft.com/en-us/azure/devops/pipelines/
5. Microsoft .NET IoT Libraries. (2024). learn.microsoft.com. https://learn.microsoft.com/en-us/dotnet/iot/
6. Bootstrap Documentation. (2025). getbootstrap.com. https://getbootstrap.com/docs/
7. Clean Code Principles in .NET. (2023). devblogs.microsoft.com. https://devblogs.microsoft.com/dotnet/clean-code-principles/
8. Cypress End-to-End Testing. (2025). docs.cypress.io. https://docs.cypress.io/guides/overview/why-cypress
9. Deloitte. (2025). Smart Home and IoT Trends. deloitte.com. https://www2.deloitte.com/
10. Fowler, M. (2002). Patterns of Enterprise Application Architecture. https://martinfowler.com/eaaCatalog/
11. Freeman, E., & Robson, E. (2021). Head First Design Patterns. O'Reilly Media. https://www.oreilly.com/
12. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns. Pearson. https://www.pearson.com/
13. Google Home Developer Documentation. (2025). developers.google.com. https://developers.google.com/home
14. Amazon Alexa Smart Home Skills. (2025). developer.amazon.com. https://developer.amazon.com/en-US/alexa/smarthome

15. Apple HomeKit Documentation. (2025). developer.apple.com. https://developer.apple.com/homekit/

16. MQTT Protocol Specification. (2024). mqtt.org. https://mqtt.org/documentation

17. Zigbee Alliance. (2024). Zigbee Specification. https://zigbeealliance.org/solution/zigbee/

18. Z-Wave Alliance. (2024). Z-Wave Technology Overview. https://www.z-wavealliance.org/

19. Hassan, M., & Khan, A. (2023). Smart home architectures and IoT integration. IEEE Access, 11, 14567–14580. https://doi.org/10.1109/ACCESS.2023.3245678

20. Gartner. (2024). Smart Home Technology Trends. gartner.com. https://www.gartner.com/en/information-technology

21. MongoDB Documentation. (2025). mongodb.com. https://www.mongodb.com/docs/

22. .NET Core Best Practices. (2023). learn.microsoft.com. https://learn.microsoft.com/en-us/dotnet/core/extensions/best-practices

23. Osherove, R. (2019). The Art of Unit Testing. Manning. https://www.manning.com/books/the-art-of-unit-testing-third-edition

24. OWASP Top Ten Security Risks. (2024). owasp.org. https://owasp.org/www-project-top-ten/

25. Postman API Testing Guide. (2025). learning.postman.com. https://learning.postman.com/docs/

26. Resnick, M. (2023). Continuous deployment with Azure. learn.microsoft.com. https://learn.microsoft.com/en-us/azure/architecture/devops/

27. SASS Documentation. (2024). sass-lang.com. https://sass-lang.com/guide

28. Selenium Documentation. (2025). selenium.dev. https://www.selenium.dev/documentation/

29. Siemens Smart Infrastructure. (2024). siemens.com. https://www.siemens.com/smart-infrastructure

30. Schneider Electric Smart Home Solutions. (2024). se.com. https://www.se.com/

31. Wang, L., & Zhang, Y. (2025). IoT-based smart home systems. ACM Transactions on IoT, 19(1), 1–20. https://doi.org/10.1145/3647890

32. Chen, J., & Li, X. (2024). Wireless sensor networks for smart homes. Sensors, 24(3), 1193. https://doi.org/10.3390/s24031193

33. Kumar, S., & Sharma, R. (2023). Microservices architecture for IoT systems. Journal of Software Engineering, 11(2), 67–82. https://doi.org/10.1186/s40411-023-00189-9

34. Green Software Foundation. (2024). Energy-efficient software systems. https://greensoftware.foundation/

35. ISO/IEC 30141:2018. Internet of Things (IoT) Reference Architecture. https://www.iso.org/standard/65695.html

36. ISO/IEC 27001:2022. Information Security Management Systems. https://www.iso.org/isoiec-27001-information-security.html

37. NIST. (2024). IoT Security Guidelines. nist.gov. https://www.nist.gov/iot

38. Cisco. (2024). IoT and Smart Home Solutions. cisco.com. https://www.cisco.com/

39. IBM. (2024). Internet of Things Platform. ibm.com. https://www.ibm.com/internet-of-things

40. Microsoft Azure Digital Twins. (2025). learn.microsoft.com. https://learn.microsoft.com/en-us/azure/digital-twins/

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: "Розробка програмного забезпечення для систем розумного будинку"

Обсяг пояснювальної записки: 89 аркушів

Дата закінчення бакалаврської роботи 10 червня 2026р.

Підпис студента _____