Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматизації, комп'ютерно-інтегрованих технологій та систем
(повна назва)
Кафедра Комп’ютерно-інтегрованих технологій, автоматизації, робототехніки та безпекової інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)
Розроблення програмного забезпечення для симуляції цифрового двійника розумного будинку
(тема)

Виконав:
здобувач 4 року навчання,
групи АКТСІ-22-1
Євгеній БАРАНОВ
(власне ім’я, прізвище)

Спеціальність 151 Автоматизація та комп’ютерно-інтегровані технології
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Системна інженерія
(повна назва освітньої програми)

Керівник проф. Олександр ЦИМБАЛ
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри КІТАРБІ _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім’я, прізвище)

2026 р.

Я, Баранов Євгеній Дмитрович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу з академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

" 30 " червня 2026 р.  Євгеній БАРАНОВ

Харківський національний університет радіоелектроніки

Факультет Автоматизації, комп'ютерно-інтегрованих технологій та систем
Кафедра Комп’ютерно-інтегрованих технологій, автоматизації, робототехніки та безпекової інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 151 Автоматизація та комп’ютерно-інтегровані технології
(код і повна назва)
Тип програми освітньо-професійна
Освітня програма Системна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«27» квітня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
здобувачеві Баранову Євгенію Дмитровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розроблення програмного забезпечення для симуляції цифрового двійника розумного будинку
затверджена наказом університету від 04 червня 2026 р. № 473 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 16 липня 2026 р.
3. Вихідні дані до роботи 3.1 Клієнт-серверна архітектура;
3.2 База даних SQLite;
3.3 Персональний комп'ютер з ОС Windows

4. Перелік питань, що потрібно опрацювати в роботі 4.1 Вступ;
4.2 Аналіз предметної області та існуючих рішень;
4.3 Розробка моделей, алгоритмів та візуальних схем для цифрового двійника розумного будинку;
4.4 Програмна реалізація цифрового двійника розумного будинку;
4.5 Розробка програмного забезпечення цифрового двійника;
4.6 Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Графічний матеріал у вигляді презентації – 14 арк. ф. А4.

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної області та існуючих рішень	27.04.26-01.05.26	виконано
2	Розробка моделей, алгоритмів та візуальних схем для цифрового двійника розумного будинку	02.05.26-07.05.26	виконано
3	Програмна реалізація цифрового двійника розумного будинку	08.05.26-15.05.26	виконано
4	Розробка програмного забезпечення цифрового двійника	186.05.26-23.06.26	виконано
5	Оформлення пояснювальної записки	24.06.26-05.07.26	виконано
6	Подання роботи на нормоконтроль	06.07.26-08.07.26	виконано
7	Подання роботи на перевірку Інтернет-сервісом Unicheck	09.07.26-10.07.26	виконано
8	Подання роботи на рецензію	11.07.26-13.07.26	виконано
9	Подання роботи на підпис зав. кафедри	14.07.26-15.07.26	виконано
10	Подання кваліфікаційної роботи в ЕК	16.07.26	виконано

Дата видачі завдання 27.04.2026 р.

Здобувач _____ Євгеній БАРАНОВ
(підпис) (власне ім'я, прізвище)

Керівник роботи _____ проф. Олександр ЦИМБАЛ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 114 с., 3 табл., 46 рис., 3 дод., 25 джерел.

ЦИФРОВИЙ ДВІЙНИК, РОЗУМНИЙ БУДИНОК, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МОДЕЛЮВАННЯ, СИМУЛЯЦІЯ, КЛІЄНТ-СЕРВЕРНА ВЗАЄМОДІЯ, БАЗА ДАНИХ SQLITE, ІНФОРМАЦІЙНА СИСТЕМА, СЕНСОРИ, АВТОМАТИЗАЦІЯ, МОНІТОРИНГ, ВІЗУАЛІЗАЦІЯ, INTERNET OF THINGS (IOT), КЕРУВАННЯ ПРИСТРОЯМИ, ЦИФРОВА МОДЕЛЬ.

Метою кваліфікаційної роботи є підвищення ефективності програмного забезпечення цифрового двійника розумного будинку за рахунок забезпечення моніторингу та візуалізації стану системи в реальному часі.

Об’єктом розробки є процес моніторингу стану розумного будинку.

Предметом розробки є програмні засоби створення цифрового двійника розумного будинку.

У роботі проведено аналіз предметної області та існуючих рішень, розроблено моделі, алгоритми та візуальні схеми для цифрового двійника розумного будинку. Програмно реалізовано цифрового двійника розумного будинку та розроблено програмне забезпечення цифрового двійника.

Отримані результати можна віднести до Цілі сталого розвитку 9 «Промисловість, інновації та інфраструктура», а саме 9.5.

ABSTRACT

Explanatory note: 114 p., 3 tables, 46 figures, 3 appendices, 25 sources.

DIGITAL TWIN, SMART HOME, SOFTWARE, MODELING, SIMULATION, CLIENT-SERVER INTERACTION, SQLITE DATABASE, INFORMATION SYSTEM, SENSORS, AUTOMATION, MONITORING, VISUALIZATION, INTERNET OF THINGS (IOT), DEVICE CONTROL, DIGITAL MODEL.

The purpose of the qualification work is to increase the efficiency of the smart home digital twin software by ensuring real-time monitoring and visualization of the system status.

The object of development is the process of monitoring the status of a smart home.

The subject of development is software for creating a digital twin of a smart home.

The work analyzes the subject area and existing solutions, develops models, algorithms and visual schemes for a digital twin of a smart home. The digital twin of a smart home is implemented in software and the software of the digital twin is developed.

The results obtained can be attributed to Sustainable Development Goal 9 "Industry, Innovation and Infrastructure", namely 9.5.

ЗМІСТ

Перелік скорочень 9
Вступ 10
1 Аналіз предметної області та існуючих рішень 12
1.1 Аналіз систем управління розумним будинком 13
1.2 Аналіз платформ для створення цифрових двійників 16
1.3 Аналіз технологічних особливостей реалізації програмного забезпечення та формулювання вимог до системи 19
1.4 Постановка задач 23
2 Розробка моделей, алгоритмів та візуальних схем для цифрового двійника розумного будинку 26
2.1 Розробка клієнт-серверної архітектури цифрового двійника розумного будинку 26
2.2 Аналіз і вибір технологій комунікації клієнт-серверної взаємодії 30
2.3 Розробка серверної частини системи 33
2.4 Проєктування та реалізація клієнтської частини системи 35
2.5 Розробка прототипу цифрового двійника житлового приміщення 37
2.6 Проєктування та розробка алгоритму роботи з базою даних SQLite 38
3 Програмна реалізація цифрового двійника розумного будинку 41
3.1 Розробка серверної частини системи 41
3.2 Розробка клієнтської частини системи 48
3.3 Реалізація службових структур і параметрів системи 59
4 Розробка програмного забезпечення цифрового двійника 61
4.1 Демонстрація роботи розробленої програми 61
4.2 Розробка загального інтерфейсу програми 62
4.3 Розробка інтерфейсу сенсорів 64
4.4 Огляд інтерфейсів сенсорів 65
4.5 Демонстрація роботи програми 71

4.6 Охорона праці 75
Висновки 78
Перелік джерел посилання 80
Додаток А Код програми. Серверна частина 84
Додаток Б Код програми. Клієнтська частина 105
Додаток В Демонстраційний матеріал 114

ПЕРЕЛІК СКОРОЧЕНЬ

IoT – Internet of Things (Інтернет речей);

API – Application Programming Interface (інтерфейс прикладного програмування).

AWS – Amazon Web Services (хмарна платформа вебсервісів компанії Amazon).

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту).

JSON-RPC – JavaScript Object Notation Remote Procedure Call (протокол віддаленого виклику процедур із використанням формату JSON).

MQTT – Message Queuing Telemetry Transport (легковаговий мережевий протокол обміну повідомленнями за моделлю «видавець – підписник»).

RAM – Random Access Memory (оперативна пам'ять).

SQL – Structured Query Language (структурована мова запитів до баз даних).

UML – Unified Modeling Language (уніфікована мова моделювання).

ВСТУП

У сучасних умовах розвитку інформаційних технологій все більшого поширення набувають системи “розумного будинку”, які дозволяють автоматизувати побутові процеси, підвищити рівень комфорту проживання та забезпечити ефективне використання ресурсів. Такі системи об’єднують різноманітні пристрої та сенсори, що здійснюють контроль температури, вологості, освітлення та інших параметрів середовища.

Разом із цим, зростає потреба у зручних та ефективних засобах моніторингу і керування такими системами. Існуючі рішення часто є розрізненими або обмеженими за функціональністю, що ускладнює отримання повної інформації про стан будинку та оперативне реагування на зміни.

Одним із сучасних підходів до вирішення цієї проблеми є використання концепції цифрового двійника. Цифровий двійник дозволяє створити віртуальну модель розумного будинку, яка відображає його поточний стан на основі даних із сенсорів та дає можливість здійснювати моніторинг і керування в режимі реального часу.

Актуальність даної теми обумовлена необхідністю створення доступних, гнучких та зручних у використанні систем для контролю стану розумного будинку, а також широкими можливостями застосування цифрових двійників у цій сфері.

Метою кваліфікаційної роботи є підвищення ефективності програмного забезпечення цифрового двійника розумного будинку за рахунок забезпечення моніторингу та візуалізації стану системи в реальному часі.

Об’єктом розробки є процес моніторингу стану розумного будинку.

Предметом розробки є програмні засоби створення цифрового двійника розумного будинку.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз існуючих систем розумного будинку;

– визначити основні вимоги до програмного забезпечення;

– розробити структуру системи та бази даних;

– реалізувати взаємодію між клієнтською та серверною частинами;

– забезпечити відображення даних сенсорів у зручному вигляді.

Практичне значення роботи полягає у створенні програмного рішення, яке може бути використане для побудови систем моніторингу розумного будинку та подальшого їх розвитку.

Кваліфікаційну роботу виконано згідно ДСТУ 3008–15 [1], методичних вказівок [2], а її результати можна віднести до Цілі сталого розвитку 9 «Промисловість, інновації та інфраструктура», а саме 9.5.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

У сучасних умовах стрімкого розвитку інформаційних технологій та поширення концепції Інтернету речей (IoT) особливого значення набуває автоматизація житлового середовища. Одним із ключових напрямів такої автоматизації є створення систем розумного будинку, які забезпечують підвищення рівня комфорту, безпеки та енергоефективності житлових приміщень. Концепція розумного будинку передбачає інтеграцію різноманітних технічних засобів і програмного забезпечення в єдину систему, здатну здійснювати моніторинг, аналіз і керування різними параметрами будівлі в автоматичному або напівавтоматичному режимі.

Розумний будинок представляє собою складну кіберфізичну систему, яка об’єднує сенсорні пристрої, виконавчі механізми, обчислювальні модулі та програмні компоненти. Сенсори виконують функцію збору даних про стан навколишнього середовища (температура, вологість, освітленість, рух, якість повітря тощо), тоді як виконавчі пристрої (освітлення, системи опалення, кондиціонування, електроприлади) реалізують керуючі впливи відповідно до заданих алгоритмів або команд користувача. Центральний контролер або сервер забезпечує обробку даних, прийняття рішень та координацію роботи всієї системи [3].

Однією з ключових особливостей розумного будинку є можливість реалізації сценарного управління, тобто система може автоматично виконувати певні дії у відповідь на визначені події або умови. Наприклад, при зниженні температури нижче заданого рівня система активує опалення, а при відсутності мешканців – вимикає освітлення та електроприлади з метою економії енергії, тим самим, забезпечується не лише зручність користування, але й оптимізація витрат ресурсів.

Ще одним важливим аспектом функціонування розумного будинку є можливість віддаленого доступу та управління. Завдяки використанню веб-

інтерфейсів або мобільних додатків користувач може контролювати стан системи та керувати нею з будь-якої точки світу, що досягається за рахунок використання мережевих технологій та хмарних сервісів, що забезпечують обмін даними між пристроями та користувачем у режимі реального часу. Сучасні системи розумного будинку також активно використовують методи обробки даних та елементи штучного інтелекту, що дозволяє системі адаптуватися до поведінки користувача, прогнозувати його потреби та автоматично оптимізувати режими роботи, наприклад, система може запам’ятовувати звички мешканців і налаштовувати освітлення або температуру відповідно до їхнього розкладу [4].

З технологічної точки зору, реалізація розумного будинку базується на використанні різноманітних протоколів зв’язку та стандартів, таких як Wi-Fi, Bluetooth, ZigBee, Z-Wave та MQTT, вибір конкретних технологій залежить від вимог до системи, її масштабованості, енергоефективності та вартості реалізації, а важливою характеристикою є також забезпечення сумісності між різними пристроями та виробниками, що досягається за рахунок використання відкритих стандартів або універсальних платформ.

Концепція розумного будинку являє собою комплексний підхід до автоматизації житлового простору, що поєднує апаратні та програмні компоненти, сучасні мережеві технології та інтелектуальні алгоритми управління, вона створює основу для подальшого розвитку більш складних систем, зокрема цифрових двійників, які дозволяють не лише керувати об’єктом у реальному часі, але й моделювати його поведінку, аналізувати можливі сценарії та підвищувати ефективність функціонування всієї системи [5].

1.1 Аналіз систем управління розумним будинком

На сучасному етапі розвитку технологій системи управління розумним будинком представлені широким спектром рішень, що відрізняються за архітектурою, функціональністю, рівнем відкритості та складністю впровадження. Проведення порівняльного аналізу таких систем дозволяє

визначити їх сильні та слабкі сторони, а також сформувати обґрунтовані вимоги до розроблюваного програмного забезпечення цифрового двійника.

Однією з найбільш поширених систем є Home Assistant, яка належить до відкритих платформ з можливістю локального розгортання. Її ключовою перевагою є висока гнучкість та підтримка великої кількості пристроїв різних виробників. Система дозволяє користувачеві створювати складні сценарії автоматизації та інтегрувати різні протоколи зв'язку. Водночас значним недоліком є складність початкового налаштування та необхідність технічної підготовки користувача, що обмежує її використання серед непідготовлених користувачів [6].

Іншою популярною системою є OpenHAB, яка також є відкритою платформою та орієнтована на гнучкість і масштабованість. Вона підтримує модульну архітектуру та дозволяє інтегрувати різні типи пристроїв через розширення. У порівнянні з Home Assistant, OpenHAB має більш формалізовану структуру конфігурації, що забезпечує стабільність, але водночас ускладнює процес налаштування. Основним недоліком є менш інтуїтивний інтерфейс та складність освоєння [7].

Серед комерційних рішень варто виділити Apple HomeKit, який характеризується високим рівнем безпеки та інтеграцією в екосистему пристроїв Apple. Система забезпечує зручний користувацький інтерфейс та стабільну роботу, що робить її привабливою для широкого кола користувачів. Однак її використання обмежене пристроями, сертифікованими Apple, що знижує гнучкість і підвищує вартість впровадження [8].

Аналогічно, Google Home пропонує зручний інтерфейс і потужну інтеграцію з хмарними сервісами, включаючи голосове управління та використання алгоритмів машинного навчання. Основною перевагою є простота налаштування та доступність, проте залежність від хмарної інфраструктури створює ризики, пов'язані з безпекою та стабільністю роботи при втраті з'єднання з мережею [9].

Централізовані професійні системи, такі як KNX, відрізняються високою надійністю та стандартизованим підходом до побудови автоматизованих систем. Вони широко використовуються у промислових та комерційних об’єктах, але також можуть застосовуватись у житлових будинках. Основними перевагами KNX є стабільність, довговічність та незалежність від конкретного виробника. Водночас такі системи характеризуються високою вартістю впровадження та потребують професійного проєктування і монтажу [10]. Для узагальнення та візуалізації результатів аналізу, представимо порівняльну характеристику розглянутих систем у вигляді таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих

Система	Тип системи	Гнучкість	Простота використання	Інтеграція пристроїв	Залежність від хмари
1	2	3	4	5	6
Home Assistant	Відкрита, локальна	Висока	Середня	Дуже висока	Низька
OpenHAB	Відкрита, локальна	Висока	Низька	Висока	Низька
Apple HomeKit	Комерційна	Середня	Висока	Обмежена	Середня
Google Home	Хмарна	Середня	Висока	Висока	Висока
KNX	Професійна	Середня	Низька	Висока	Низька
Система	Тип системи	Гнучкість	Простота використання	Інтеграція пристроїв	Залежність від хмари

Відкриті системи, такі як Home Assistant та OpenHAB, забезпечують максимальну гнучкість і можливості інтеграції, однак потребують значних зусиль для налаштування та обслуговування, комерційні рішення, зокрема Apple HomeKit і Google Home, орієнтовані на зручність користування, проте

обмежують користувача у виборі пристроїв та часто залежать від хмарних сервісів, а професійні системи типу KNX забезпечують найвищу надійність, але є економічно доцільними переважно для великих об’єктів. Таким чином, жодна з існуючих систем не забезпечує повною мірою поєднання гнучкості, простоти використання та можливості моделювання станів об’єкта у вигляді цифрового двійника, що обґрунтовує необхідність розроблення спеціалізованого програмного забезпечення, яке поєднуватиме функції моніторингу, управління та симуляції, що і визначає напрям подальших досліджень у даній роботі [6-10].

1.2 Аналіз платформ для створення цифрових двійників

Концепція цифрового двійника набула широкого поширення у зв’язку з розвитком технологій Інтернету речей, хмарних обчислень та аналітики даних. Цифровий двійник являє собою віртуальну модель фізичного об’єкта або системи, яка відображає його поточний стан, поведінку та взаємодію з навколишнім середовищем у режимі реального часу. У контексті розумного будинку цифровий двійник дозволяє не лише здійснювати моніторинг параметрів, але й моделювати різні сценарії функціонування системи, прогнозувати зміни та оптимізувати управління.

Серед існуючих платформ для створення цифрових двійників значне місце займають хмарні рішення, що забезпечують масштабованість і гнучкість. Однією з найбільш розвинених платформ є Microsoft Azure Digital Twins, яка надає можливість створення складних моделей середовища з використанням графової структури зв’язків між об’єктами. Платформа підтримує інтеграцію з IoT-пристроями, обробку потокових даних та використання аналітичних інструментів. Її перевагою є висока масштабованість і широкий функціонал, однак вона потребує глибоких знань хмарних технологій та має складну конфігурацію, що ускладнює її застосування для невеликих проєктів [11].

Аналогічним за призначенням рішенням є AWS IoT TwinMaker, яке орієнтоване на створення цифрових моделей із використанням сервісів Amazon

Web Services. Платформа забезпечує зручні інструменти для візуалізації даних, інтеграції з різними джерелами інформації та побудови аналітичних моделей. У порівнянні з Azure, вона має дещо простішу інтеграцію з іншими сервісами AWS, проте також характеризується залежністю від хмарної інфраструктури та складністю налаштування для початківців [12].

Серед промислових рішень варто відзначити платформи компанії Siemens, зокрема екосистему цифрових двійників, що використовується для моделювання складних технічних систем. Дані рішення забезпечують високу точність моделювання, підтримують інженерні розрахунки та дозволяють проводити комплексний аналіз поведінки об’єкта. Проте такі платформи орієнтовані переважно на великі підприємства, мають високу вартість і потребують спеціалізованої підготовки [13].

Окрему категорію становлять інструменти, що використовуються для візуалізації цифрових двійників, зокрема середовище Unity. Воно дозволяє створювати інтерактивні 3D-моделі об’єктів і відображати їх стан у реальному часі. Unity не є повноцінною платформою цифрових двійників, однак часто використовується у поєднанні з іншими сервісами для побудови візуального представлення системи. Перевагою є висока якість графіки та гнучкість, але реалізація повноцінної логіки цифрового двійника потребує додаткових інструментів [14].

Для більш об’єктивного порівняння платформ цифрових двійників доцільно використати кількісні метрики, що характеризують їх продуктивність, масштабованість та технічні можливості, до таких показників можна віднести максимальну кількість оброблюваних пристроїв, затримку обробки даних, кількість підтримуваних потоків даних, а також орієнтовні вимоги до ресурсів. Оформимо кількісне порівняння платформ цифрових двійників у вигляді таблиці 1.2.

Таблиця 1.2 – Кількісне порівняння платформ цифрових двійників

Платформа	Макс. кількість пристроїв	Затримка обробки (мс)	К-сть подій/сек	Підтримка одночасних з'єднань	Мін. вимоги до RAM (ГБ)
Azure Digital Twins	до 1000000+	50 – 200	до 100000	до 500000	від 4
AWS IoT TwinMaker	до 1 000 000+	50 – 150	до 120000	до 500000	від 4
Siemens Digital Twin	до 10000000+	20 – 100	до 200000	до 1000000	від 8
Unity (з backend)	до 10000–100000	100 – 300	до 10000	до 50000	від 8

Наведені дані демонструють, що промислові рішення, такі як Siemens Digital Twin, забезпечують найвищу продуктивність та масштабованість, що дозволяє використовувати їх у великих інфраструктурних системах. Хмарні платформи Azure Digital Twins та AWS IoT TwinMaker також демонструють високі показники, однак їх ефективність значною мірою залежить від налаштувань хмарної інфраструктури та тарифного плану. У свою чергу, використання середовища Unity як платформи цифрового двійника обмежується значно меншими показниками продуктивності, оскільки воно переважно орієнтоване на візуалізацію, а не на обробку великих обсягів даних, що робить його доцільним лише у поєднанні з іншими системами обробки даних. Таким чином, кількісний аналіз підтверджує, що існуючі платформи орієнтовані на великомасштабні системи та можуть бути надлишковими для задач розумного будинку, що підкреслює доцільність розроблення оптимізованого рішення, яке забезпечуватиме достатній рівень продуктивності при менших вимогах до ресурсів [11-14].

1.3 Аналіз технологічних особливостей реалізації програмного забезпечення та формулювання вимог до системи

Розроблення програмного забезпечення цифрового двійника розумного будинку передбачає врахування сучасних підходів до побудови високопродуктивних, гнучких та незалежних систем, здатних працювати у режимі, близькому до реального часу. З урахуванням специфіки задачі, особлива увага приділяється вибору технологій, які забезпечують швидку обробку даних, ефективне використання ресурсів та можливість масштабування системи без суттєвих змін її архітектури.

Одним із ключових рішень є використання мови програмування C++, яка дозволяє досягти високого рівня продуктивності завдяки низькорівневому доступу до ресурсів системи, це особливо важливо для задач, пов’язаних із обробкою великої кількості даних від сенсорів та моделюванням станів у реальному часі. Використання C++ забезпечує ефективне управління пам’яттю, що дозволяє оптимізувати роботу системи навіть на обмежених апаратних ресурсах, крім того, дана мова програмування є придатною для реалізації систем реального часу, що є критично важливим для синхронізації цифрового двійника з фізичним об’єктом або його моделлю.

Для реалізації графічного інтерфейсу користувача та візуалізації стану системи доцільно використовувати бібліотеки Raylib та Raygui, які надають простий та ефективний інструментарій для створення інтерактивних додатків. Вказані бібліотеки характеризуються невеликою вагою, кросплатформеністю та достатнім набором функцій для побудови графічного інтерфейсу, що відображає стан сенсорів, виконавчих пристроїв та інших елементів системи, використання таких інструментів дозволяє створити наочне представлення цифрового двійника без значних витрат обчислювальних ресурсів, що є важливим для забезпечення швидкодії системи [15].

З точки зору архітектури, доцільним є застосування клієнт-серверного підходу, який забезпечує розподіл обчислювальних процесів та підвищує

гнучкість системи. Серверна частина відповідає за обробку даних, збереження стану системи та реалізацію бізнес-логіки, тоді як клієнтська частина забезпечує відображення інформації та взаємодію з користувачем. Для організації обміну даними між компонентами системи важливо використовувати механізми передачі даних у реальному часі, зокрема технологію WebSocket, яка дозволяє забезпечити постійне двостороннє з'єднання між клієнтом і сервером, що дає змогу досягти високого рівня синхронізації між фізичною системою або її емуляцією та цифровим двійником.

Важливим аспектом розроблення є використання open-source рішень, що дозволяє уникнути залежності від комерційних платформ і забезпечує повний контроль над системою, такий підхід сприяє гнучкості розробки, оскільки дозволяє адаптувати програмне забезпечення під конкретні потреби користувача або проєкту, а використання відкритих технологій зменшує вартість впровадження системи та полегшує її подальшу модернізацію і масштабування. Особливістю реалізації цифрового двійника є створення моделі даних, яка відображає структуру розумного будинку, включаючи приміщення, сенсори, виконавчі пристрої та їх взаємозв'язки, система повинна забезпечувати постійне оновлення даних у режимі реального часу, а також можливість моделювання змін параметрів для аналізу поведінки системи за різних умов, що передбачає реалізацію механізмів генерації тестових або симульованих даних, що дозволяє досліджувати роботу системи без підключення реального обладнання.

На основі визначених технологічних підходів сформулюємо вимоги до розроблюваної системи, до функціональних вимог належить забезпечення повноцінного моніторингу параметрів розумного будинку, включаючи відображення значень температури, вологості, освітленості, стану пристроїв та інших показників, система повинна підтримувати управління виконавчими пристроями як у ручному режимі, так і автоматично відповідно до заданих сценаріїв. Важливою вимогою є можливість створення, редагування та виконання сценаріїв, які визначають поведінку системи залежно від заданих умов. Ключовою функціональною особливістю є підтримка симуляції, яка

дозволяє реалізувати цифровий двійник як інструмент моделювання. Система повинна забезпечувати генерацію віртуальних даних, відтворення різних ситуацій та аналіз реакції системи на зміну параметрів, що дає можливість тестувати алгоритми управління, оптимізувати сценарії та підвищувати ефективність функціонування системи без ризику для реального об’єкта.

До нефункціональних вимог можна віднести вимоги до продуктивності, які передбачають швидку обробку даних та мінімальні затримки при оновленні інформації. Система повинна бути здатною працювати у режимі реального часу або з мінімальними відхиленнями від нього. Надійність системи забезпечується стабільною роботою при тривалому використанні та здатністю обробляти помилки без критичних збоїв, а масштабованість передбачає можливість розширення системи шляхом додавання нових пристроїв, сенсорів або функціональних модулів без значного ускладнення архітектури. Також увагу необхідно приділити безпеці системи, яка включає реалізацію механізмів автентифікації та авторизації користувачів, захист каналів передачі даних, а також обмеження доступу до керуючих функцій. Важливою є також зручність використання, яка досягається за рахунок створення інтуїтивного графічного інтерфейсу, що дозволяє користувачу швидко отримувати необхідну інформацію та взаємодіяти з системою без спеціальної підготовки.

Після проведеного аналізу можна підсумувати, що обрані технологічні рішення, зокрема використання мови програмування C++, бібліотек Raylib та Raygui, клієнт-серверної архітектури та відкритих технологій, дозволяють створити ефективне програмне забезпечення цифрового двійника розумного будинку, сформульовані функціональні та нефункціональні вимоги визначають основні характеристики системи та створюють основу для її подальшого проєктування і реалізації [16-17].

Розроблена схема, представлена на рисунку 1.1 відображає загальну архітектуру функціонування системи цифрового двійника розумного будинку та демонструє взаємодію між її основними компонентами. В основі системи лежить

двосторонній обмін даними між фізичним середовищем і цифровою моделлю, що забезпечує актуальність та синхронізацію станів.

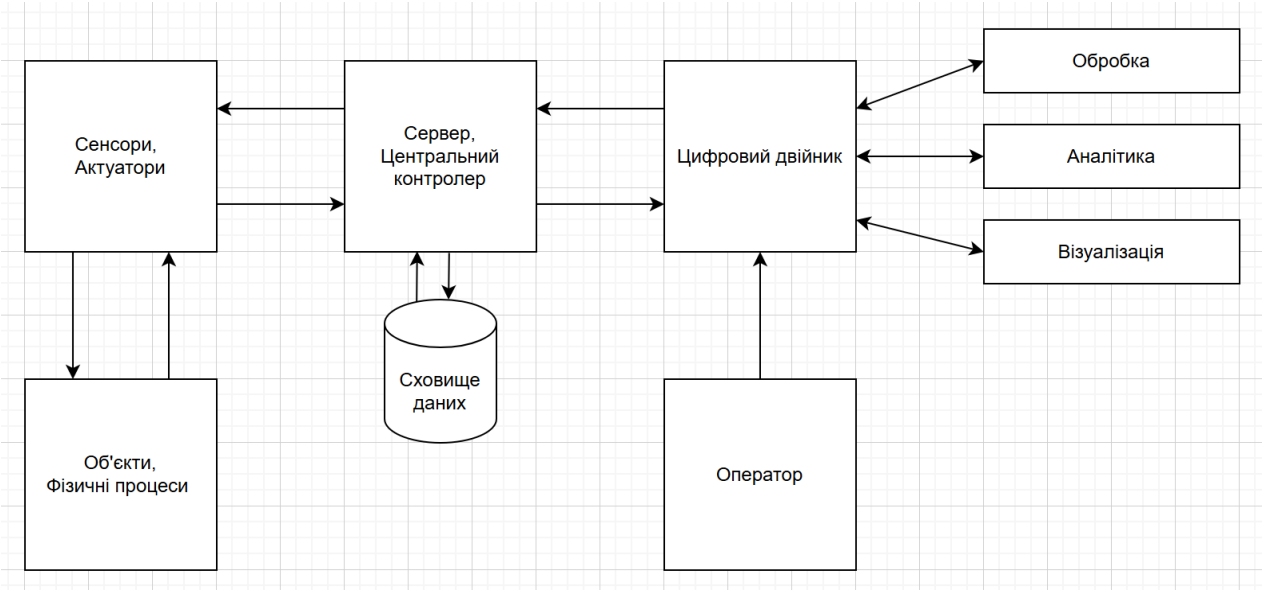


Рисунок 1.1 – Загальна архітектура функціонування системи цифрового двійника розумного будинку

Первинним рівнем системи є фізичні об’єкти та процеси, які включають інженерні системи будинку (освітлення, клімат, безпека тощо). Взаємодія з ними здійснюється через сенсори та актуатори: сенсори збирають інформацію про стан середовища (температура, вологість, рух), а актуатори виконують керуючі дії. Отримані дані передаються до центрального вузла – сервера або контролера, який виконує функції обробки, збереження та маршрутизації інформації.

Сервер виступає ядром системи, забезпечуючи накопичення даних у сховищі та їх подальшу передачу до цифрового двійника. Сховище даних дозволяє зберігати історичну інформацію, що є необхідною для аналізу, прогнозування та відтворення станів системи. Завдяки цьому забезпечується не лише поточний моніторинг, а й можливість дослідження поведінки системи у часі.

Цифровий двійник є віртуальною моделлю фізичної системи, яка відображає її стан у реальному часі. Він отримує дані від сервера, обробляє їх та надає інтерфейс для взаємодії з користувачем (оператором). Оператор, у свою

чергу, може здійснювати контроль, змінювати параметри системи або ініціювати керуючі дії, які передаються назад до фізичного рівня через сервер.

Додатково цифровий двійник забезпечує виконання функцій обробки, аналітики та візуалізації. Обробка дозволяє фільтрувати та підготовлювати дані, аналітика – виявляти закономірності та прогнозувати поведінку системи, а візуалізація – відображати інформацію у зручному для користувача вигляді. Запропонована схема реалізує замкнений цикл керування, у якому фізична система та її цифровий аналог працюють синхронно, забезпечуючи ефективне управління та підвищення функціональності розумного будинку.

1.4 Постановка задач

Проведений аналіз предметної області та існуючих систем управління розумним будинком і платформ цифрових двійників показав, що наявні рішення не повною мірою відповідають вимогам до сучасних інтелектуальних систем. Зокрема, частина систем орієнтована виключно на моніторинг і керування пристроями без реалізації концепції цифрового двійника, тоді як інші рішення є надмірно складними, ресурсоємними та орієнтованими переважно на промислове застосування, що обмежує їх ефективне використання у сфері розумного будинку, де важливими є простота, доступність та гнучкість.

Основними недоліками існуючих підходів є відсутність інтерактивної віртуальної моделі, яка б дозволяла відображати структуру та стан системи у зручному для користувача вигляді. Крім того, більшість рішень мають обмежені можливості візуалізації, що ускладнює аналіз даних та прийняття рішень. Важливим недоліком є також відсутність повноцінної підтримки симуляції, яка дозволяє моделювати поведінку системи без впливу на реальний об’єкт. Додатковими проблемами є залежність від сторонніх комерційних платформ, що обмежує гнучкість розробки, а також або недостатня продуктивність, або надлишкова складність реалізації.

У зв’язку з виявленими недоліками виникає необхідність розроблення програмного забезпечення цифрового двійника розумного будинку, яке поєднуватиме функції моніторингу, управління та моделювання. Основною задачею є створення системи, що забезпечує збір та обробку даних у режимі реального часу, відображення параметрів стану будинку у наочному та зрозумілому вигляді, а також підтримку взаємодії між компонентами системи з мінімальними затримками.

Розроблювана система повинна передбачати можливість розширення функціональності шляхом додавання нових модулів, сенсорів та сценаріїв роботи, що забезпечить її адаптивність до змінних умов використання. Важливою вимогою є використання відкритих технологій, що дозволяє уникнути залежності від сторонніх платформ, зменшити витрати на впровадження та забезпечити повний контроль над процесом розробки. При цьому система повинна характеризуватися високою продуктивністю та ефективністю, що досягається за рахунок оптимального вибору технологій і архітектурних рішень.

Постановка задачі полягає у створенні програмного забезпечення цифрового двійника розумного будинку, яке забезпечує ефективний моніторинг, управління та симуляцію стану системи, поєднуючи у собі гнучкість, масштабованість та високу продуктивність. Реалізація такого рішення дозволить підвищити якість управління розумним будинком, спростити процес взаємодії користувача з системою та створити основу для подальшого розвитку інтелектуальних систем автоматизації. Для виконання поставленної мети необхідно виконати наступні задачі:

– визначити та обґрунтувати вибір технологій реалізації системи, включаючи мову програмування, бібліотеки та архітектурні підходи;

– розробити архітектуру програмної системи та структуру взаємодії її компонентів;

– спроєктувати структуру бази даних для збереження параметрів сенсорів, пристроїв та сценаріїв роботи;

– реалізувати механізм обміну даними між клієнтською та серверною частинами системи;

– розробити модуль моніторингу та візуалізації стану розумного будинку у режимі реального часу;

– реалізувати функціонал симуляції цифрового двійника та перевірки сценаріїв роботи системи;

– провести тестування розробленого програмного забезпечення та оцінити його працездатність і продуктивність.

2 РОЗРОБКА МОДЕЛЕЙ, АЛГОРИТМІВ ТА ВІЗУАЛЬНИХ СХЕМ ДЛЯ ЦИФРОВОГО ДВІЙНИКА РОЗУМНОГО БУДИНКУ

2.1 Розробка клієнт-серверної архітектури цифрового двійника розумного будинку

В основу програмного забезпечення покладено клієнт-серверну архітектуру, яка забезпечує ефективну взаємодію між окремими компонентами системи. Такий підхід дозволяє створити продуктивний та масштабований додаток із чітким розмежуванням функціональних обов'язків між клієнтською та серверною частинами. Клієнтська частина відповідає за взаємодію з користувачем, відображення даних, візуалізацію стану системи та передачу запитів. Серверна частина, у свою чергу, виконує обробку даних, логіку керування, збереження інформації та забезпечує обмін даними між пристроями системи. Використання клієнт-серверної моделі дозволяє підвищити надійність роботи програмного забезпечення, спростити супровід системи та забезпечити можливість подальшого розширення функціоналу без значних змін у загальній структурі проєкту. Така організація сприяє ефективній роботі обох сторін програми та оптимальному використанню обчислювальних ресурсів. На рисунку 2.1 показано приклад класичної клієнт-серверної архітектури.

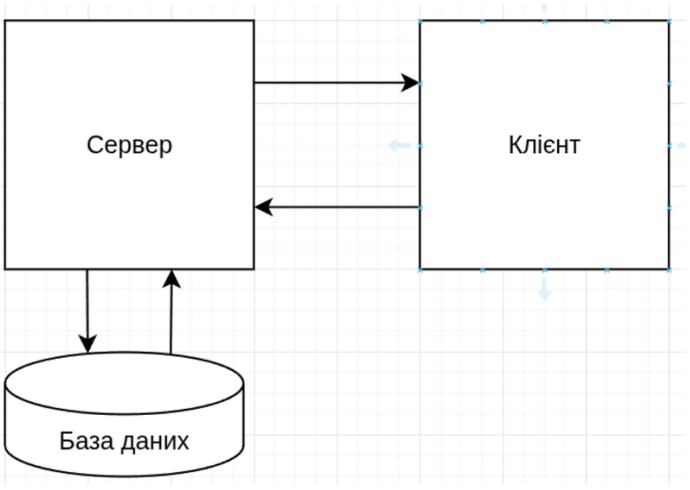


Рисунок 2.1 – Класична клієнт-серверна архітектура

Особливу увагу необхідно приділяти клієнт-серверній архітектурі в програмних системах, орієнтованих на взаємодію з користувачем-оператором. Такий підхід дозволяє забезпечити незалежність клієнтської частини від серверної, що значно підвищує зручність використання, надійність та гнучкість програмного забезпечення.

Користувач взаємодіє переважно з клієнтським застосунком, який відповідає за інтерфейс, відображення даних та візуальний контроль стану системи, тоді як серверна частина виконує обробку інформації, керування логікою процесів та зберігання даних. Подібне розділення дозволяє зменшити навантаження на клієнтську частину та забезпечити стабільну роботу всієї системи [18].

Такий принцип особливо добре застосовується у програмних комплексах цифрових двійників, зокрема цифрового двійника розумного будинку, де необхідно одночасно забезпечувати моніторинг, керування пристроями, обробку великої кількості даних та зручну взаємодію користувача із системою в режимі реального часу.

Клієнт-серверна архітектура добре піддається модифікації та подальшому розширенню, що є важливою перевагою під час розробки сучасного програмного забезпечення. Завдяки гнучкій структурі така архітектура дозволяє адаптувати систему до нових вимог без суттєвих змін у її основній логіці.

Оскільки в даній розробці основний акцент зроблено на клієнтській частині застосунку, базу даних доцільно реалізувати у вигляді вбудованого локального сховища на стороні клієнта. Такий підхід також відповідає принципам сучасної архітектури програмних систем, зберігаючи логічне розмежування компонентів, але водночас дозволяючи виконувати зберігання, обробку та швидкий доступ до даних безпосередньо на клієнтському пристрої. На рисунку 2.2 представлено клієнт-серверну архітектуру з вбудованою базою даних.

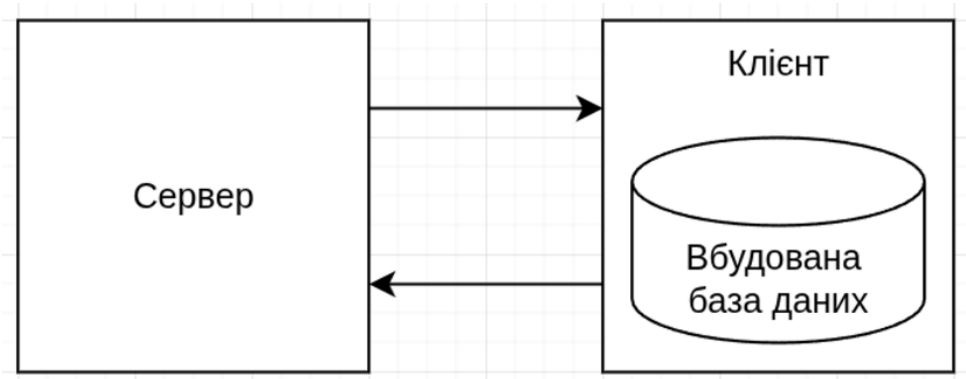


Рисунок 2.2 – Клієнт-серверна архітектура з вбудованою базою даних

Основним елементом архітектури є сервер, який виконує роль центрального обчислювального вузла системи. Його головною функцією є приймання, обробка та передача інформації між фізичними компонентами розумного будинку та клієнтською частиною цифрового двійника. Сервер забезпечує логіку функціонування всієї системи, аналізує вхідні дані від сенсорів, формує команди управління для виконавчих пристроїв, а також координує інформаційний обмін із клієнтським застосунком. Крім того, сервер відповідає за перевірку коректності отриманих даних, підтримання стабільності з’єднання та організацію обробки подій у режимі реального часу.

Функціонально серверна частина виконує кілька ключових завдань. Насамперед це збір телеметричних даних від фізичних пристроїв розумного будинку, таких як датчики температури, освітлення, вологості чи руху. Після отримання інформації сервер виконує її попередню обробку, фільтрацію та структуризацію. Далі відбувається формування пакета даних для передачі клієнтській частині, яка використовує ці відомості для оновлення стану цифрового двійника, також сервер може реалізовувати сценарії автоматизації, приймаючи рішення про зміну стану пристроїв на основі встановлених правил або команд користувача.

Другим основним блоком є клієнтська частина, яка виступає засобом взаємодії користувача із системою, саме на цьому рівні реалізується графічний інтерфейс цифрового двійника, відображення поточного стану розумного будинку та механізми керування його компонентами. Клієнт забезпечує

приймання даних від сервера, їх візуальне представлення, а також формування команд управління, які передаються назад до серверної частини, що дозволяє користувачу в реальному часі контролювати роботу системи, змінювати параметри функціонування окремих пристроїв та аналізувати поточний стан будинку.

Важливою складовою клієнтського блоку є вбудована база даних, яка використовується для локального збереження інформації. Її призначення полягає у збереженні історії змін стану системи, параметрів конфігурації, сценаріїв автоматизації та іншої службової інформації. Використання локальної бази даних дозволяє забезпечити швидкий доступ до інформації без необхідності постійного звернення до сервера, що позитивно впливає на продуктивність системи. Крім того, вона дає можливість зберігати дані навіть у разі тимчасового розриву з’єднання із сервером, забезпечуючи автономність роботи клієнтської частини.

Зв’язок між сервером і клієнтом відображають механізм постійного інформаційного обміну. Передача даних від сервера до клієнта забезпечує актуалізацію цифрової моделі відповідно до стану фізичного об’єкта, тоді як зворотний канал використовується для надсилання команд керування та змін параметрів системи. Такий підхід реалізує принцип замкненого циклу керування, за якого цифровий двійник не лише відображає стан реального об’єкта, але й активно впливає на його функціонування.

Дана архітектура особливо ефективна для систем моделювання та симуляції, де важливо забезпечити оперативність обробки інформації, зменшити затримки у взаємодії та знизити залежність від постійного мережевого з’єднання, підхід також зменшує навантаження на серверну інфраструктуру та підвищує загальну стабільність роботи цифрового двійника розумного будинку [19].

2.2 Аналіз і вибір технологій комунікації клієнт-серверної взаємодії

Важливою складовою взаємодії між клієнтом і сервером є вибір типу з’єднання. Для систем цифрового двійника розумного будинку, а також інших програмних комплексів цифрових двійників, особливо важливо забезпечити постійний і стабільний обмін даними в режимі реального часу.

Оскільки система повинна безперервно отримувати інформацію про стан пристроїв, датчиків та виконавчих механізмів, необхідно підтримувати постійне з’єднання між клієнтською та серверною частинами. Саме тому в якості основного типу з’єднання було обрано технологію WebSocket.

WebSocket забезпечує двосторонній обмін даними між клієнтом і сервером без необхідності постійного повторного надсилання HTTP-запитів. Це дозволяє значно зменшити затримки, підвищити швидкість передачі даних та знизити навантаження на мережу. Такий підхід є оптимальним для систем моніторингу та керування розумним будинком, де швидка реакція та актуальність інформації мають критично важливе значення [20]. Порівняємо технології комунікації клієнт-серверної взаємодії та їх основні характеристики та представимо у вигляді таблиці 2.1.

Таблиця 2.1 – Технології комунікації клієнт-серверної взаємодії

Тип з’єднання	Опис	Переваги	Недоліки	Застосування
1	2	3	4	5
HTTP	Класична модель, де клієнт надсилає запит, а сервер відповідає	Простота реалізації, широка підтримка	Відсутність постійного з’єднання, зайві запити	Веб-сторінки, REST API
Long Polling	Клієнт тримає запит відкритим до появи нових даних	Реактивність	Високе навантаження на сервер, неефективність при великій кількості клієнтів	Чати, прості real-time системи

Продовження таблиці 2.1

1	2	3	4	5
Server-Sent Events (SSE)	Сервер сам надсилає дані клієнту по постійному каналу	Простота, стабільний односторонній потік даних	Тільки сервер надсилає. Немає двостороннього обміну	Нотифікації, моніторинг
MQTT	Легка publish/subscribe модель для IoT	Дуже низьке навантаження, ідеально для пристроїв	Потребує брокера, складніша інфраструктура	IoT, сенсори, розумний дім
WebSocket	Постійне двостороннє з'єднання між клієнтом і сервером	Мінімальні затримки, реальний час, ефективність	Складніша реалізація, потребує підтримки з'єднання	Ігри, цифрові двійники, IoT, чати

Як видно з таблиці, WebSocket є більш універсальним підходом порівняно з MQTT, особливо для реалізації постійно відкритого з'єднання та роботи в режимі реального часу. Він забезпечує двосторонній обмін даними між клієнтом і сервером із мінімальними затримками, що робить його зручним для інтерактивних систем, зокрема цифрових двійників [21].

Водночас MQTT більше орієнтований на IoT-середовища з великою кількістю легковагових пристроїв і побудований на моделі publish/subscribe, що вимагає використання окремого брокера повідомлень.

Таким чином, у межах даної розробки доцільно або використовувати WebSocket як універсальний протокол для прямого обміну даними, або, за потреби масштабування системи до великої кількості пристроїв, комбінувати його зі стандартизованими протоколами передачі даних, такими як MQTT або HTTP API. Це дозволяє забезпечити баланс між гнучкістю, продуктивністю та масштабованістю системи.

Для обміну повідомленнями між клієнтом і сервером було обрано JSON-RPC. Це легкий протокол віддаленого виклику процедур, який використовує формат JSON для передачі даних і дозволяє організувати взаємодію у вигляді викликів методів із параметрами та отриманням відповіді.

Його використання дозволяє уніфікувати структуру всіх запитів у системі та зробити взаємодію між компонентами більш передбачуваною і простою для реалізації. Завдяки цьому логіка викликів відокремлюється від транспортного рівня, що особливо зручно при використанні WebSocket як каналу зв'язку. У межах цифрового двійника розумного будинку JSON-RPC спрощує керування системою, оскільки всі операції з пристроями та отримання даних сенсорів реалізуються як окремі методи. Це робить архітектуру більш гнучкою, полегшує її розширення та підтримку, а також підвищує зрозумілість взаємодії між клієнтською і серверною частинами.

JSON-RPC у даній розробці використовується поверх WebSocket-з'єднання, що дозволяє поєднати постійний канал зв'язку з уніфікованим способом виклику процедур між клієнтом і сервером. Це забезпечує стандартизацію обміну даними та спрощує реалізацію взаємодії в системі.

Такий підхід робить повідомлення структурованими й зрозумілими для розробника, оскільки логіка викликів відокремлена від транспортного рівня. У межах прототипу цифрового двійника розумного будинку це є зручним рішенням, яке спрощує розширення функціоналу та забезпечує стабільну роботу системи в режимі реального часу [22].

Нашарування та поєднання різних протоколів дозволяє побудувати власну, але водночас стандартизовану модель обміну даними між серверною та клієнтською частинами системи. Що дозвояле розмежування компонентів системи на backend і frontend, дозволяє чітко розподілити функціональні обов'язки між частинами програмного забезпечення та забезпечити більш зручну організацію розробки. Frontend відповідає за взаємодію з користувачем, візуалізацію даних і керування інтерфейсом, тоді як backend виконує обробку інформації, логіку роботи системи та взаємодію з пристроями головними мікроконтролером. На рисунку 2.3 представлено взаємодію frontend і backend через WebSocket та JSON-RPC.

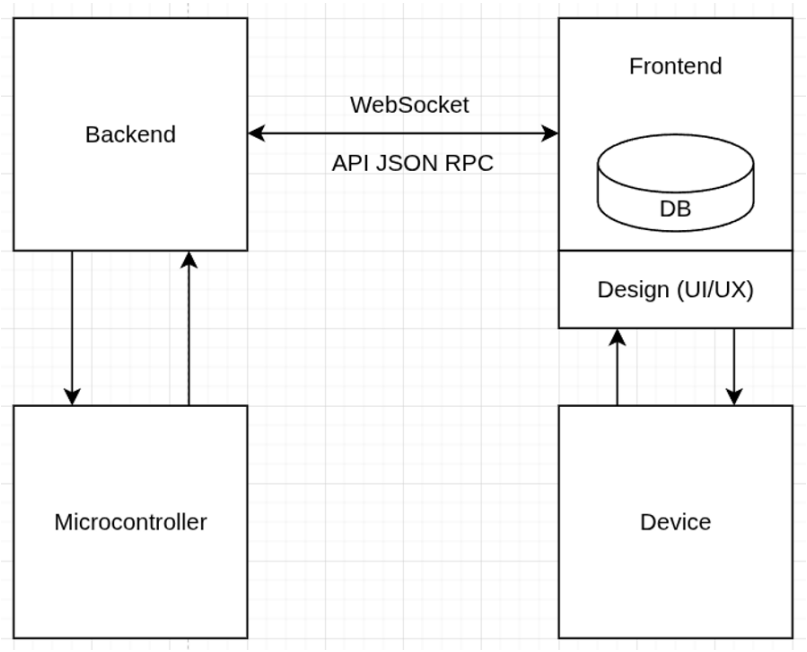


Рисунок 2.3 – Взаємодія frontend і backend через WebSocket та JSON-RPC

JSON RPC API будується на основі виклику віддалених процедур між клієнтом і сервером у стандартизованому форматі JSON. Клієнт надсилає запит, у якому вказується назва методу, параметри для його виконання та ідентифікатор запиту. Сервер приймає повідомлення, обробляє виклик відповідного методу та повертає результат у вигляді структурованої відповіді.

Основними елементами JSON RPC запиту є поле method, яке визначає назву викликаної процедури, params, що містить необхідні параметри, та id, який використовується для зв’язку запиту з відповіддю. У відповіді сервер повертає або результат виконання через поле result, або інформацію про помилку через поле error.

2.3 Розробка серверної частини системи

На наступному етапі було розпочато розробку механізму взаємодії між клієнтською та серверною частинами системи. Основна увага була приділена створенню серверної частини, яка відповідає за обробку запитів.

Серверна частина системи реалізована мовою програмування C++ стандарту C++17. Вибір саме цієї мови зумовлений її високою продуктивністю,

ефективною роботою з пам’яттю та можливістю створення швидких і надійних серверних застосунків. C++17 надає сучасні засоби розробки, покращену роботу зі структурами даних, багатопоточністю та мережевими з’єднаннями, що є важливим для побудови систем реального часу. Це дозволяє забезпечити стабільну роботу серверної частини цифрового двійника розумного будинку та спростити подальше розширення функціоналу системи. Було використано бібліотеку ixwebsocket, яка забезпечує роботу з WebSocket для з’єднанням між клієнтом і сервером. Для створення та обробки JSON-повідомлень застосовано бібліотеку nlohmann/json, яка дозволяє зручно працювати з форматом JSON та реалізувати JSON-RPC API.

Для реалізації цієї функціональності було спроєктовано та побудовано клас Server, який виконує роль центрального компонента серверної архітектури. Даний клас забезпечує прийом з’єднань, обробку повідомлень, виклик необхідних процедур та передачу відповідей клієнтській частині. UML діаграма класу Server представлено на рисунку 2.4.

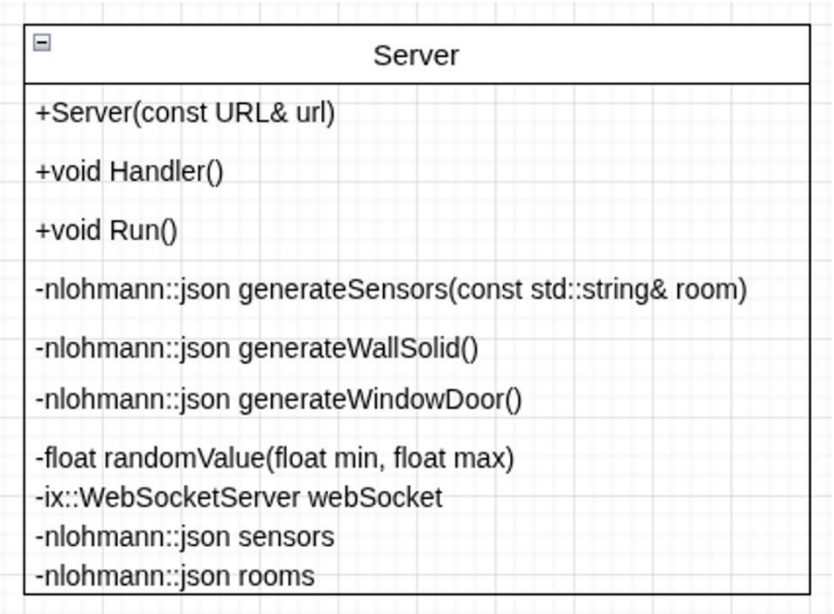


Рисунок 2.4 – UML діаграма класу Server

Клас Server є основним компонентом серверної частини системи та відповідає за організацію взаємодії між клієнтом і цифровим двійником

розумного будинку. Його основною задачею є прийом з’єднань, обробка запитів, генерація даних та передача інформації клієнтській частині через WebSocket.

Поле webSocket використовується для створення та підтримки WebSocket з’єднання між сервером і клієнтом. Поле sensors зберігає інформацію про сенсори системи, а rooms містить дані про приміщення цифрового двійника.

Метод Run() відповідає за запуск серверної частини та початок роботи WebSocket-сервера. Метод Handler() виконує обробку вхідних повідомлень і керує логікою взаємодії між компонентами системи. Метод generateSensors() використовується для генерації даних сенсорів у вибраному приміщенні. Метод generateWallSolid() формує інформацію про стіни, а generateWindowDoor() створює дані про вікна та двері.

Метод randomValue() застосовується для генерації випадкових значень, що використовуються під час моделювання станів сенсорів і симуляції роботи системи в реальному часі [23].

2.4 Проєктування та реалізація клієнтської частини системи

Розробка клієнтської частини розпочалася з проєктування класу клієнта, на який покладається основний механізм взаємодії із серверною частиною системи. Саме цей клас забезпечує прийом даних від сенсорів, станів об’єктів цифрового двійника, а також повідомлень про можливі помилки або виключні ситуації.

Після отримання та первинної обробки запиту клієнтська частина виконує структурування даних і передає їх далі по програмному пайплайну для подальшої обробки, візуалізації або збереження. Таким чином, клієнт виступає проміжною ланкою між користувачем і серверною логікою системи, забезпечуючи коректний обмін інформацією та стабільну роботу всього програмного комплексу. UML-діаграма класу Client клієнтської частини системи представлена на рисунку 2.5.

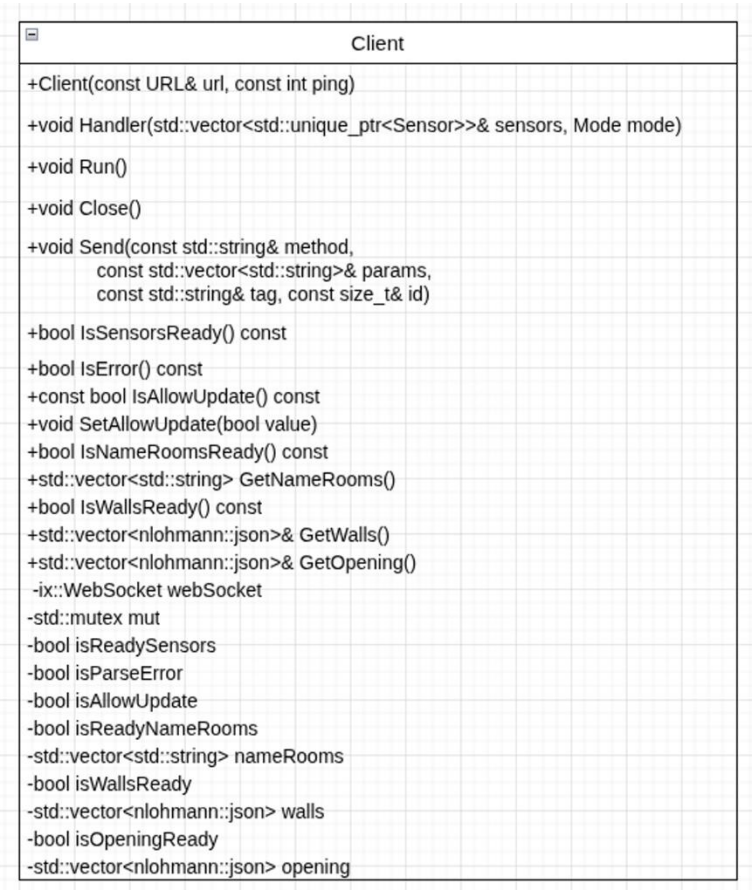


Рисунок 2.5 – UML-діаграма класу Client клієнтської частини системи

Клас Client відповідає за реалізацію клієнтської частини системи цифрового двійника та забезпечує взаємодію із сервером через WebSocket-з’єднання. Конструктор Client(const URL &url, const int ping) ініціалізує клієнт, встановлює параметри підключення до сервера та налаштовує базові параметри роботи, зокрема інтервал перевірки з’єднання.

Метод Run() запускає клієнтську частину та ініціює процес встановлення та підтримки зв’язку із сервером. Метод Close() використовується для коректного завершення роботи та закриття WebSocket-з’єднання.

Метод Handler(...) є основним обробником даних, який приймає інформацію від сервера, оновлює стан сенсорів і координує подальшу обробку даних у системі залежно від режиму роботи.

Метод Send(...) забезпечує формування та відправку запитів до сервера у стандартизованому форматі зокрема JSON RPC, включаючи метод, параметри та ідентифікатор запиту.

Методи IsSensorsReady(), IsNameRoomsReady(), IsWallsReady() та IsOpeningReady() використовуються для перевірки готовності відповідних даних, що надходять від сервера, і визначають стан синхронізації клієнта із системою.

Метод IsError() повертає інформацію про наявність помилок у процесі обробки або парсингу даних.

Методи GetNameRooms(), GetWalls() та GetOpening() надають доступ до відповідних структур даних, які містять інформацію про приміщення та елементи цифрового двійника.

Методи IsAllowUpdate() та SetAllowUpdate(bool value) керують дозволом на оновлення стану клієнтської частини, що використовується для синхронізації даних у процесі роботи системи.

2.5 Розробка прототипу цифрового двійника житлового приміщення

Розробка прототипу житлового приміщення у вигляді цифрового двійника будинку вимагала створення базової моделі середовища. З цією метою було сформовано спрощену схему житлових кімнат, яка дозволяє визначити межі функціонування сенсорів та забезпечити коректну симуляцію роботи системи.

У межах даного етапу було побудовано мінімальну модель житлового приміщення, що представляє собою абстрактний будинок із основними структурними елементами. Такий підхід дозволяє зосередитися на відпрацюванні логіки взаємодії компонентів системи, не ускладнюючи модель надлишковими деталями. Схема житлового будинку представлена на рисунку 2.6.

Схема житлового будинку виконана у мінімалістичному стилі, що дозволяє спростити процес прототипування цифрового двійника розумного будинку та зосередитися на основній функціональності системи. Такий підхід робить модель більш наочною та зрозумілою для аналізу, а також полегшує відображення роботи сенсорів і взаємодії компонентів системи.

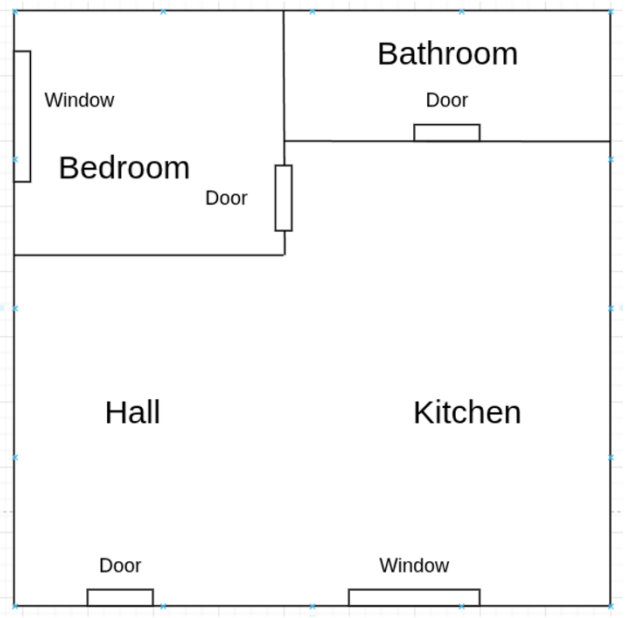


Рисунок 2.6 – Схема житлового будинку

2.6 Проєктування та розробка алгоритму роботи з базою даних SQLite

Для подальшого аналізу, збереження та обліку показників сенсорів у системі цифрового двійника розумного будинку використовується вбудована база даних SQLite. Вибір саме SQLite зумовлений її компактністю, простотою інтеграції та можливістю локального зберігання даних без необхідності розгортання окремого серверу баз даних.

Для інтеграції бази даних у систему було розроблено прототип спеціалізованого класу для роботи з SQLite. Даний клас спроєктовано із використанням шаблону проєктування Singleton, що дозволяє забезпечити єдину точку доступу до бази даних у межах усього застосунку.

Такий підхід спрощує організацію роботи з даними, запобігає створенню декількох одночасних підключень до бази та підвищує стабільність роботи системи. Крім того, централізоване керування доступом до SQLite полегшує подальше розширення функціональності та супровід програмного забезпечення. На рисунку 2.7 представлено повні інтерфейси класу Storage.

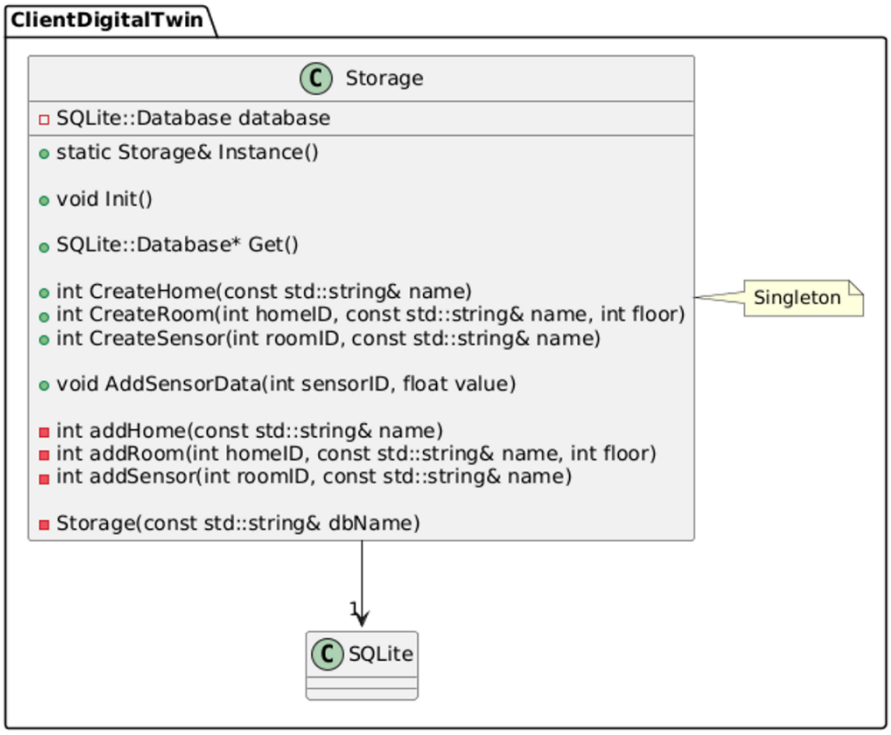


Рисунок 2.7 – Повні інтерфейси класу Storage

Для збереження та структуризації даних у системі цифрового двійника розумного будинку використовується реляційна база даних SQLite. Структура бази даних побудована за принципом логічного зв’язку між основними компонентами системи будинком, кімнатами, сенсорами та показниками сенсорів представлена на риунку 2.8.

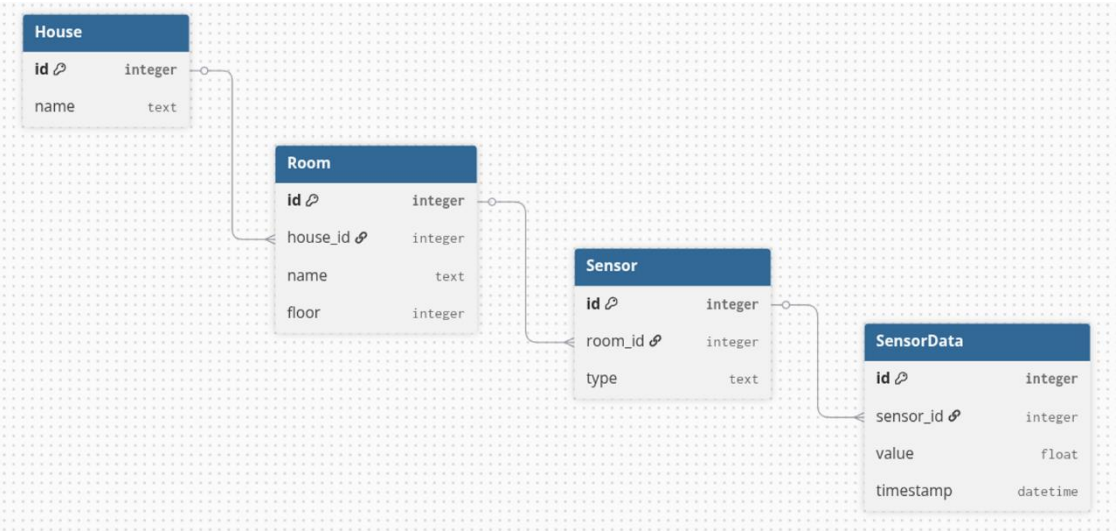


Рисунок 2.8 – Структура бази даних та зв’язки таблиць

Таблиця House використовується для збереження інформації про будинки та містить унікальний ідентифікатор і назву об’єкта. Таблиця Room призначена для збереження даних про кімнати та містить посилання на будинок через поле house_id, що дозволяє організувати зв’язок між приміщеннями та конкретним будинком.

Таблиця Sensor містить інформацію про сенсори, їх типи та прив’язку до відповідної кімнати через поле room_id. Це дозволяє визначати розташування сенсорів у структурі цифрового двійника.

Для збереження показників сенсорів використовується таблиця SensorData, яка містить значення сенсора та час отримання даних. Поле sensor_id створює зв’язок із таблицею Sensor, що дозволяє вести історію показників для кожного окремого сенсора.

Використання таких зв’язків між таблицями забезпечує цілісність даних, спрощує їх обробку та дозволяє ефективно виконувати аналіз і моніторинг стану системи цифрового двійника розумного будинку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЦИФРОВОГО ДВІЙНИКА РОЗУМНОГО БУДИНКУ

Перед реалізацією основної частини програмного забезпечення було виконано підготовку робочого середовища для розробки системи. На даному етапі було обрано необхідні інструменти розробки, редактор коду VSCodium, організовано структуру проєкту та налаштовано процес збірки застосунків.

У якості компілятора було використано clang++, який підтримує сучасний стандарт C++17 та забезпечує ефективні засоби виявлення помилок під час розробки програмного забезпечення. Структура проєкту була розділена на дві основні частини це client та server, що представляють клієнтський і серверний застосунки системи. Кожна з частин має власну файлову ієрархію та окремий Makefile для незалежної збірки й керування залежностями. Такий підхід дозволяє спростити процес розробки, зробити архітектуру проєкту більш впорядкованою та забезпечити зручність подальшого масштабування системи.

3.1 Розробка серверної частини системи

Під час розробки серверної частини системи в директорії server було організовано структуру проєкту. Для розміщення користувацького коду створено папку src, а для сторонніх бібліотек директорію external. У процесі початку розробки було інтегровано необхідні бібліотеки. Бібліотека inih використовується для роботи з конфігураційними файлами .ini. Для реалізації WebSocket з'єднання було використано бібліотеку ixwebsocket, яка забезпечує створення WebSocket серверів і клієнтів та має високу продуктивність при невеликій кількості залежностей. Для роботи з JSON повідомленнями застосовано бібліотеку nlohmann/json.

Для автоматизації процесу збірки було створено Makefile з відповідним кодом, а для контролю змін у проєкті впроваджено систему контролю версій Git.

Основними файлами серверної частини стали main.cpp, який містить точку входу в програму, server.hpp та server.cpp, що реалізують серверну логіку, а також settings.h, який містить налаштування системи. Також було створено файл settings.ini, який використовується для збереження користувацьких налаштувань програми. У даному файлі розміщуються основні параметри конфігурації системи, зокрема мережеві налаштування, параметри підключення та інші службові значення, необхідні для роботи серверної частини. На рисунку 3.1 представлено основну структуру файлової системи серверної частини застосунку.

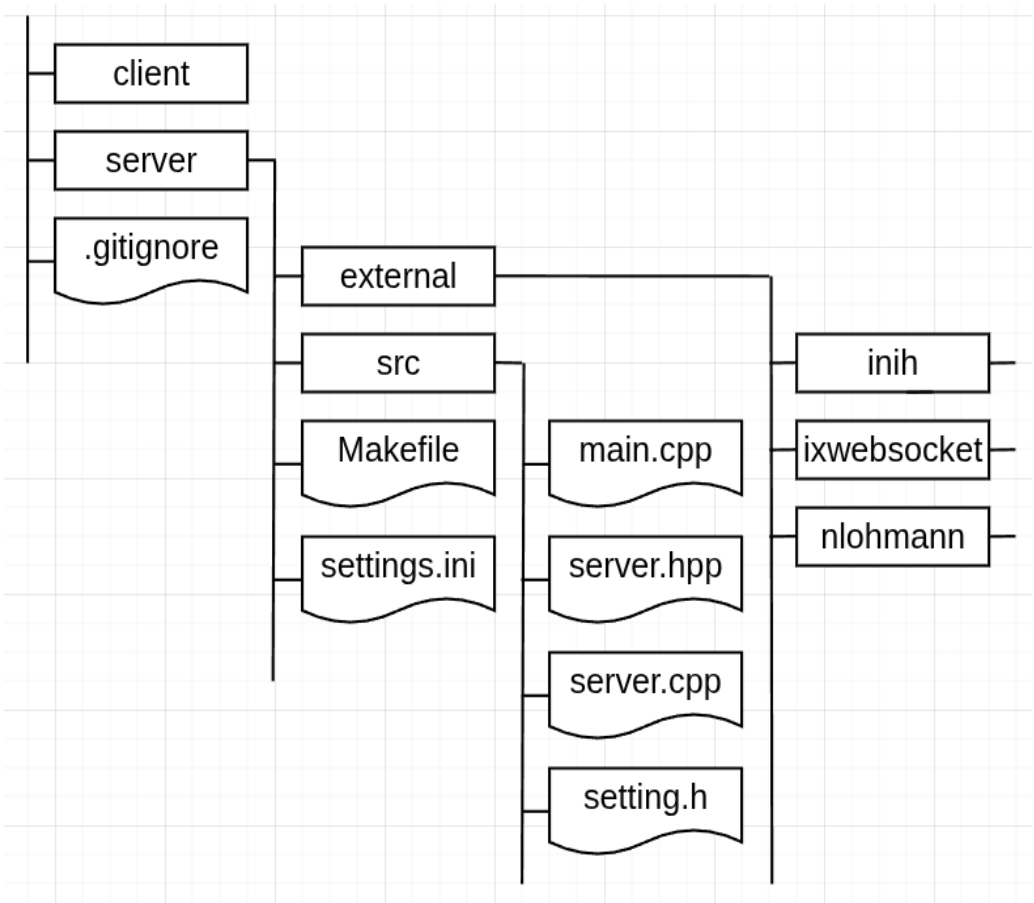


Рисунок 3.1 – Структура серверної частини проєкту

Проєктування та оголошення класу Server виконувалося на основі раніше побудованої UML-діаграми, яка відображає структуру серверної частини системи. Реалізація класу сервера представлена на рисунку 3.2.

```cpp
#pragma once
#include "settings.h"
#include <ixwebsocket/IXWebSocketServer.h>
#include <nlohmann/json.hpp>

namespace ServerDigitalTwin {
class Server {
public:
  Server(const URL &url);
  void Handler();
  void Run();

private:
  nlohmann::json generateSensors(const std::string &room);
  nlohmann::json generateWallSolid();
  nlohmann::json generateWindowDoor();
  float randomValue(float min, float max);
  ix::WebSocketServer webSocket;
  nlohmann::json sensors;
  nlohmann::json rooms;
};

} // namespace ServerDigitalTwin
```

Рисунку 3.2 – Оголошення класу Server

Далі було розпочато реалізацію класу Server, який є основним компонентом серверної частини системи цифрового двійника. На даному етапі було визначено його структуру та основні методи, що забезпечують роботу WebSocket з'єднання та обробку даних. Конструктор Server(const URL &url) відповідає за ініціалізацію сервера та налаштування параметрів підключення. Метод Run() запускає сервер і забезпечує прийом вхідних з'єднань, а Handler() використовується для обробки запитів від клієнта та керування логікою взаємодії. Методи generateSensors(), generateWallSolid() та generateWindowDoor() відповідають за формування даних цифрового двійника на кліенті, зокрема сенсорів та елементів середовища. Метод randomValue() використовується для генерації випадкових значень, що застосовуються при симуляції роботи системи.

Використовуючи параметри host і port, отримані з конфігураційного файлу settings.ini, виконується налаштування та запуск серверної частини системи. Такий підхід дозволяє гнучко змінювати мережеві параметри без необхідності редагування вихідного коду програми. Метод реалізації запуску сервера представлено на рисунку 3.3.

```cpp
void ServerDigitalTwin::Server::Run() {
  auto res: std::pair<bool, std::string> = webSocket.listen();
  if (!res.first) {
    throw std::runtime_error(
        "[!] Error: The server does not start, check the configuration");
  }

  webSocket.start();
  std::cout << "[*] WebSocket server start ws://" << webSocket.getHost() << ":"
            << webSocket.getPort() << '\n';
  webSocket.wait();
}
```

Рисунок 3.3 – Реалізація запуску серверної частини системи

Далі було реалізовано метод Handler. Який відповідає за обробку подій WebSocket з'єднання. У ньому використовується callback-функція setOnClientMessageCallback, що належить об'єкту webSocket. Метод реалізує логіку відкриття з'єднання (Open), обробку повідомлень (Message) для віддаленого виклику процедур через JSON-RPC та закриття з'єднання (Close). Це забезпечує взаємодію між клієнтом і сервером у режимі реального часу. Загальний вигляд методу Handler представлено на рисунку 3.4.

```cpp
void ServerDigitalTwin::Server::Handler() {

  webSocket.setOnClientMessageCallback(
      callback: [&](std::shared_ptr<ix::ConnectionState> connect, ix::WebSocket &ws,
          const ix::WebSocketMessagePtr &msg)-> void {
        if (msg->type == ix::WebSocketMessageType::Open) {…
        } else if (msg->type == ix::WebSocketMessageType::Message) {…
        } else if (msg->type == ix::WebSocketMessageType::Close) {…
        }
      });
}
```

Рисунок 3.4 – Обробник запитів від клієнта

Як видно, callback функція приймає необхідні параметри для обробки запитів до сервера. Параметр connect містить інформацію про поточний стан клієнтського з'єднання та дозволяє отримувати службові дані про підключення. Параметр ws є об'єктом WebSocket з'єднання, через який виконується надсилання повідомлень клієнту та керування сеансом зв'язку. Параметр msg містить отримане повідомлення від клієнта, включаючи тип події Open, Message або Close. Кожен тип події обробляється в окремій гілці callback-функції, що

дозволяє організувати структуровану та стабільну обробку запитів у серверній частині системи.

У гілці обробки подій типу Message виконується розбір JSON-RPC запиту, формування необхідних даних відповіді та подальше надсилання результату клієнту через WebSocket з'єднання. Як приклад, на рисунку 3.5 представлено реалізацію обробки процедури GetDataSensors.

```
45          if (request.contains(key: "method") &&
46              request["method"] == "GetDataSensors") {
47
48              nlohmann::json data;
49              data["sensors"] = generateSensors(room: "kitchen");
50              data["sensors"] = generateSensors(room: "bathroom");
51              data["sensors"] = generateSensors(room: "hall");
52              data["sensors"] = generateSensors(room: "bedroom");
53
54              response["id"] = request["id"];
55              response["tag"] = request["tag"];
56              response["result"] = data;
57              ws.sendText(text: nlohmann::to_string(j: response));
58
59          } else if (request.contains(key: "method") &&
60                     request["method"] == "UpdateDataSensors") {
```

Рисунок 3.5 – Обробка процедури GetDataSensors

У GetDataSensors виконується формування даних сенсорів для кімнат будинку та надсилання JSON відповіді клієнту через WebSocket з'єднання.

Як видно, в процесі обробки формується відповідь для клієнта та генеруються дані сенсорів. У реальній системі такі дані надходили б із фізичних пристроїв, однак у межах прототипу їх генерація виконується програмно. За створення показників сенсорів відповідає функція generateSensors, яка приймає назву кімнати та генерує для неї значення датчиків, заповнюючи JSON об'єкт sensors. На рисунку 3.6 для кухні виконується генерація показників сенсора температури, після чого отримані дані додаються до JSON об'єкта сенсорів для подальшого надсилання клієнту.

```
176     this->sensors.push_back(init: {[0]={[0]="name", [1]="kitchen_temp_1"},
177                                    [1]={[0]="type", [1]="temperature"},
178                                    [2]={[0]="unit", [1]="'C"},
179                                    [3]={[0]="value", [1]=randomValue(min: -50.0f, max: 50.0f)},
180                                    [4]={[0]="position", [1]=nlohmann::json(init: {
181                                                                [0]={[0]="x", [1]=-2.0f},
182                                                                [1]={[0]="y", [1]=posY},
183                                                                [2]={[0]="z", [1]=-9.5f},
184                                                            })},
185                                    [5]={[0]="angle", [1]=0.0f},
186                                    [6]={[0]="options", [1]=nlohmann::json(init: {
187                                                               [0]={[0]="heating", [1]=1.0f},
188                                                               [1]={[0]="max", [1]=35.0f},
189                                                               [2]={[0]="min", [1]=5.0f},
190                                                               [3]={[0]="recommended", [1]=21.0f},
191                                                           })}});
```

Рисунок 3.6 – Формування показників сенсора температури

Також на серверній частині генеруються параметри приміщення, зокрема характеристики кімнат, стін, вікон і дверей. На рисунку 3.7 представлено фрагмент коду, у якому виконується формування даних для створення стін приміщення, що дозволяє побудувати структуру кімнати.

```
485         // kitchen wall 1
486         [0]={[0]={[0]="type", [1]="wall"},
487          [1]={[0]="position", [1]={[0]=-0.15, [1]=2.0, [2]=-10.0}},
488          [2]={[0]="scale", [1]={[0]=20.0, [1]=4.0, [2]=0.3}}},
489
490         // kitchen wall 2
491         [1]={[0]={[0]="type", [1]="wall"},
492          [1]={[0]="position", [1]={[0]=10.0, [1]=2.0, [2]=-7.76}},
493          [2]={[0]="scale", [1]={[0]=0.3, [1]=4.0, [2]=4.8}}},
494
495         // kitchen wall 3 (mini)
496         [2]={[0]={[0]="type", [1]="wall"},
497          [1]={[0]="position", [1]={[0]=10.0, [1]=0.6, [2]=-4.06}},
498          [2]={[0]="scale", [1]={[0]=0.3, [1]=1.2, [2]=2.6}}},
499
500         // kitchen wall 4
501         [3]={[0]={[0]="type", [1]="wall"},
502          [1]={[0]="position", [1]={[0]=10.0, [1]=2.0, [2]=1.47}},
503          [2]={[0]="scale", [1]={[0]=0.3, [1]=4.0, [2]=8.5}}},
```

Рисунок 3.7 – Формування даних для створення стін кухні

Дані для розміщення вікон і дверей формуються спеціальною функцією на сервері та передаються на клієнтську частину системи для подальшого відображення, як показано на рисунку 3.8.

```
560 nlohmann::json ServerDigitalTwin::Server::generateWindowDoor() {
561   nlohmann::json data = {
562
563       // Windows
564       [0]={[0]={[0]="type", [1]="window"},
565        [1]={[0]="position", [1]={[0]=10.0f, [1]=1.2f, [2]=-4.03f}},
566        [2]={[0]="rotation", [1]=270.0f},
567        [3]={[0]="scale", [1]={[0]=4.0f, [1]=3.0f, [2]=3.0f}},
568        [4]={[0]="room", [1]="kitchen_window_1"}},
569
570       [1]={[0]={[0]="type", [1]="window"},
571        [1]={[0]="position", [1]={[0]=-5.4f, [1]=1.2f, [2]=9.99f}},
572        [2]={[0]="rotation", [1]=180.0f},
573        [3]={[0]="scale", [1]={[0]=4.0f, [1]=3.0f, [2]=3.0f}},
574        [4]={[0]="room", [1]="bedroom_window_1"}},
575
576       // Doors
577       [2]={[0]={[0]="type", [1]="door"},
578        [1]={[0]="position", [1]={[0]=10.20f, [1]=0.0f, [2]=7.46f}},
579        [2]={[0]="rotation", [1]=0.0f},
580        [3]={[0]="scale", [1]={[0]=2.0f, [1]=1.9f, [2]=2.0f}},
581        [4]={[0]="room", [1]="hall_door_1"}},
582
583       [3]={[0]={[0]="type", [1]="door"},
584        [1]={[0]="position", [1]={[0]=-2.551f, [1]=0.0f, [2]=0.0f}},
585        [2]={[0]="rotation", [1]=90.0f},
586        [3]={[0]="scale", [1]={[0]=2.0f, [1]=1.9f, [2]=2.0f}},
587        [4]={[0]="room", [1]="bedroom_door_1"}},
588
589       [4]={[0]={[0]="type", [1]="door"},
590        [1]={[0]="position", [1]={[0]=-4.60f, [1]=0.0f, [2]=-3.89f}},
591        [2]={[0]="rotation", [1]=0.0f},
592        [3]={[0]="scale", [1]={[0]=2.0f, [1]=1.9f, [2]=2.0f}},
593        [4]={[0]="room", [1]="bathroom_door_1"}}};
594
595   return data;
```

Рисунок 3.8 – Формування даних для створення вікон і дверей

Після створення необхідної інфраструктури та реалізації основних класів було розроблено файл main.cpp, який є точкою входу серверної частини системи. У ньому виконується зчитування конфігураційного файлу, ініціалізація параметрів сервера, запуск обробника запитів та подальший запуск серверної частини шляхом виклику відповідних методів, як представлено на рисунку 3.9.

```cpp
const std::string ServerDigitalTwin::CONFIG_FILE = "settings.ini";

int main() {
  INIReader reader(filename: ServerDigitalTwin::CONFIG_FILE);

  if (reader.ParseError() < 0) {
    std::cerr << "[!] Error: Can't load "
              << "'" << ServerDigitalTwin::CONFIG_FILE << "'\n";
    return 1;
  }

  ServerDigitalTwin::URL url = {
      .host=reader.Get(section: "server", name: "host", default_value: "0.0.0.0"),
      .port=(uint16_t)reader.GetInteger(section: "server", name: "port", default_value: 5000),
  };

  ServerDigitalTwin::Server server(url);
  server.Handler();
  server.Run();
```

Рисунок 3.9 – Реалізація запуску серверної частини

3.2 Розробка клієнтської частини системи

Перед початком розробки клієнтської частини було організовано файлову структуру проєкту. На рисунку 3.10 представлено кореневу структуру файлової системи клієнтської частини застосунку.

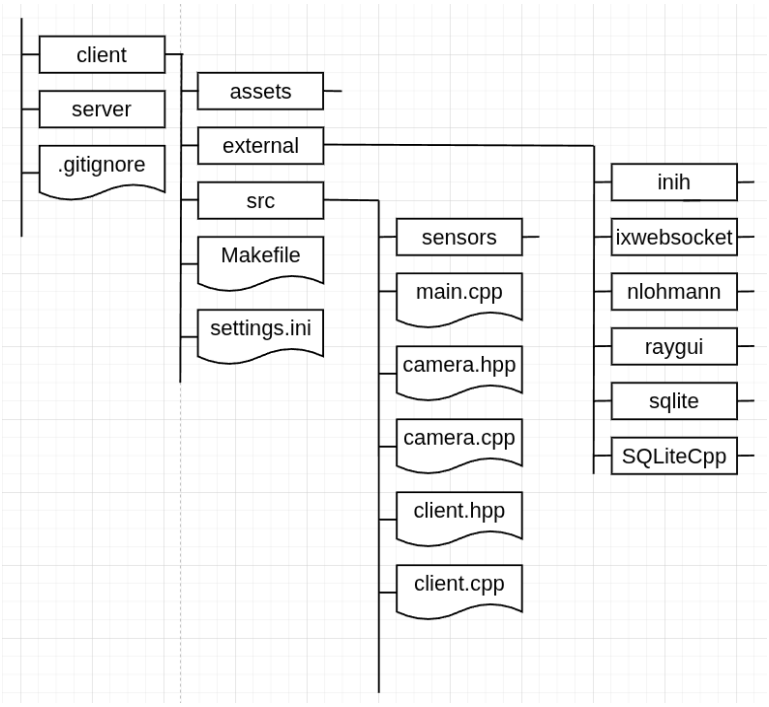


Рисунок 3.10 – Клієнтська частина файлової структури проекту

Клієнтська частина системи побудована на основі бібліотеки raylib, яка була встановлена в операційну систему глобально, тому її файли не розміщуються локально в структурі проєкту. Для побудови графічного користувацького інтерфейсу додатково використовується бібліотека raygui, яка розміщена локально в директорії external разом з іншими необхідними залежностями. У клієнтській частині також використовуються додаткові бібліотеки. Бібліотека inih застосовується для зчитування конфігураційних файлів .ini та роботи з параметрами налаштування програми. Бібліотека ixwebsocket забезпечує реалізацію WebSocket-клієнта та обмін даними із серверною частиною системи. Для роботи з JSON-повідомленнями використовується бібліотека nlohmann/json, яка забезпечує серіалізацію та парсинг даних у форматі JSON. Для локального збереження даних використовується бібліотека sqlite, а SQLiteCpp виступає C++обгорткою над SQLite та спрощує виконання SQL-запитів і роботу з базою даних у межах застосунку. Для автоматизації процесу компіляції та збірки клієнтської частини також було створено окремий Makefile, який містить необхідні параметри компіляції, підключення бібліотек і керування залежностями проєкту.

3.2.1 Побудова сцени клієнтської частини

На даному етапі було створено базову структуру для запуску вікна клієнтського застосунку. Ініціалізація графічного вікна та основних компонентів рендерингу дозволяє побудувати початкову сцену цифрового двійника та забезпечити подальше відображення елементів інтерфейсу й об’єктів системи.

Бібліотека raylib має розвинену екосистему для створення графічних застосунків, що включає підтримку 2D та 3D графіки, роботу з камерами, моделями, текстурами, аудіо та побудову користувацьких інтерфейсів через додаткові модулі на рисунку 3.11.

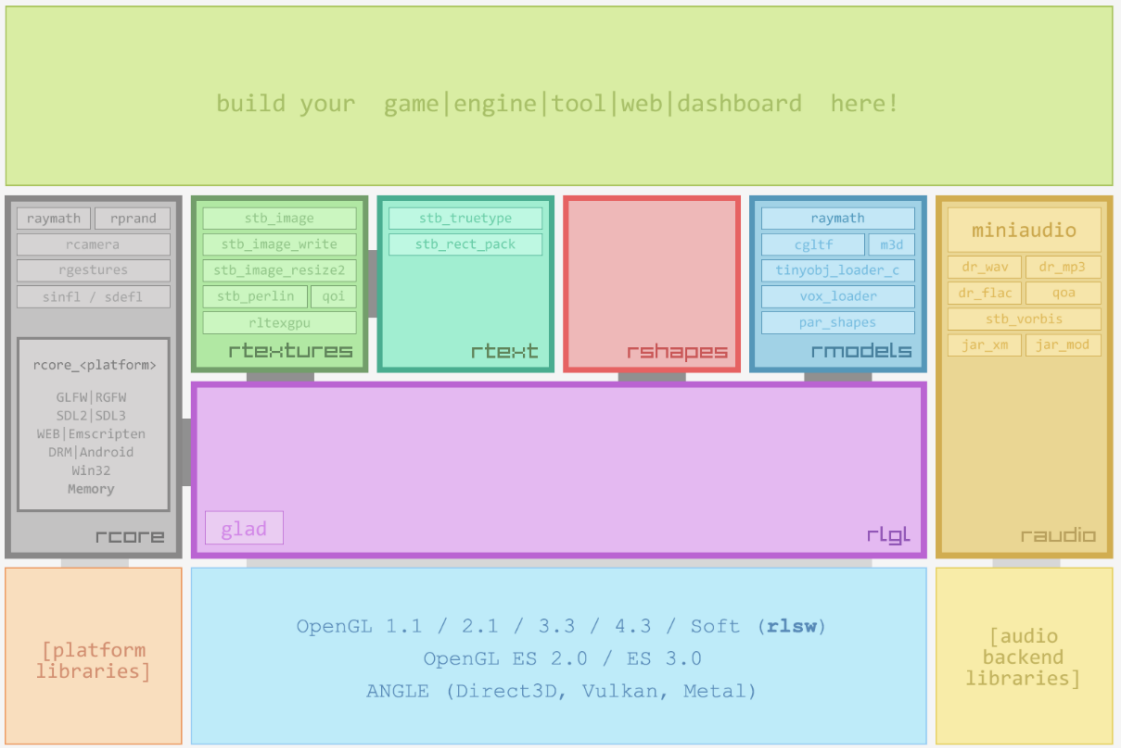


Рисунок 3.11 – Екосистема бібліотеки Raylib

Raylib працює за принципом циклічного оновлення сцени. Після створення вікна застосунку запускається основний цикл програми, у межах якого виконується обробка подій, оновлення стану об’єктів та їх подальше відображення на екрані. Побудова сцени здійснюється шляхом створення графічних об’єктів, камер, моделей та елементів інтерфейсу. Під час кожної ітерації циклу виконується рендеринг сцени, очищення кадру, відображення тривимірних об’єктів, сенсорів, приміщень та UI компонентів. Було реалізовано головний цикл подій, у якому розміщується базова логіка роботи клієнтської частини системи. У межах циклу виконується обробка подій користувача, оновлення стану об’єктів, взаємодія із сервером та рендеринг сцени цифрового двійника.

3.2.2 Розробка абстрактного класу Sensor

Було спроєктовано основний клас Sensor, від якого успадковуються класи конкретних сенсорів системи. Даний клас реалізує базову логіку роботи, взаємодії та візуалізації сенсорів у цифровому двійнику на рисунку 3.12 представлена нова UML діаграма.



Рисунок 3.12 – UML діаграма класу Sensor

Метод ShowWindow() відповідає за відображення інтерфейсу сенсора. Метод Draw() виконує рендеринг сенсора на сцені. ClickHandler() обробляє взаємодію користувача із сенсором. Метод DrawName() використовується для відображення назви сенсора у просторі сцени. Метод ModeUpdate() оновлює режим роботи сенсора, а Update() виконує оновлення значень і параметрів. Метод SwitchOpenWindow() відповідає за відкриття та закриття вікна сенсора. Допоміжні методи parsingOption(), makeOption() та makeValue() реалізують логіку обробки параметрів, формування команд і генерації значень сенсорів.

Створення сенсорів у системі реалізується через механізм успадкування від базового класу Sensor. Такий підхід дозволяє використовувати спільну логіку роботи та перевизначати лише специфічну поведінку конкретних типів сенсорів. На рисунку 3.13 представлено схему успадкування сенсорів системи.

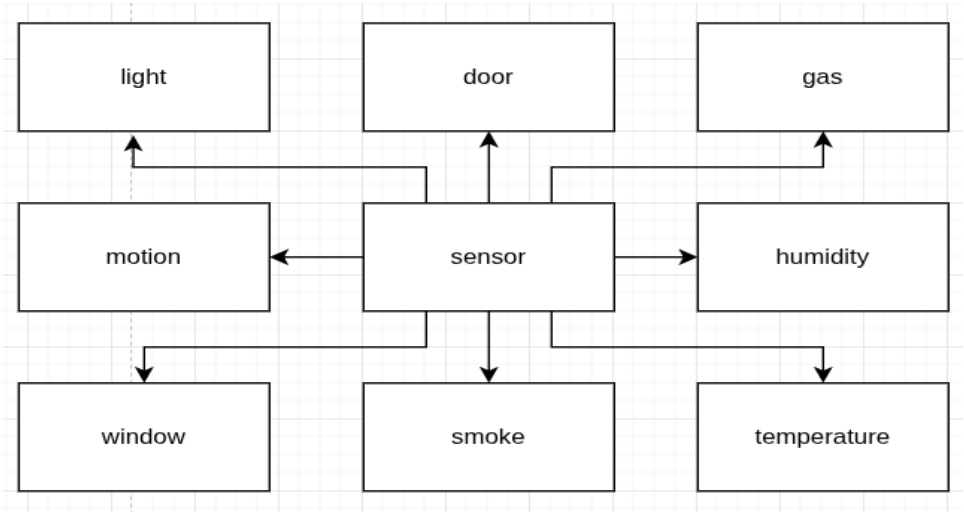


Рисунок 3.13 – Схема успадкування класів сенсорів системи

Сенсори мають універсальні риси інтерфейсу, що дозволяє забезпечити схожу поведінку під час керування системою. Саме тому в межах розробки було спроєктовано та реалізовано набір універсальних елементів інтерфейсу, які використовуються різними типами сенсорів у системі цифрового двійника, як показано на рисунку 3.14.

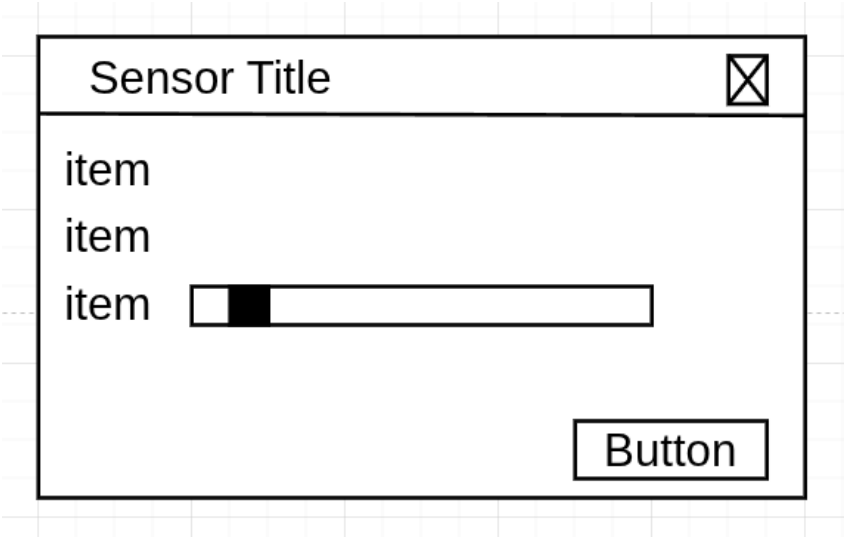


Рисунок 3.14 – Універсальний інтерфейс сенсора системи

3.2.3 Розробка класу Manager

Було розроблено клас Manager, який відповідає за керування основними рівнями роботи програми, зокрема перемиканням режимів керування та симуляції. Також клас забезпечує взаємодію з вбудованою базою даних та керує допоміжними процесами системи. Клас Manager на рисунку 3.15 виступає координуючою ланкою, що забезпечує узгоджену роботу компонентів цифрового двійника.

```
 8 namespace ClientDigitalTwin {
 9 class Manager {
10 public:
11   Manager();
12   void SelectOperatingMode();
13   const Mode GetMode() const;
14   void DrawTextMode() const;
15   void EnableTableCreation();
16   bool IsCreateTableDB() const;
17   void MakeRoomIdentification(int id, int floor, const std::string &title);
18   const int GetRoomID(const std::string &titleRoom) const;
19   void AddSensorID(Sensor *sensor, int id);
20   bool IsCreateSensorDB() const;
21   void EnableSensorCreation();
22   const std::map<Sensor *, int> &GetSensorsID() const;
23   bool IsSensorDataAdded() const;
24   void EnableSensorDataAdded();
25   const size_t &CountRequestID();
26
27 private:
28   Mode currentMode;
29   bool isCreateTableDB;
30   bool isCreateSensorDB;
31   bool isSensorDataAdded;
32   struct {
33     int id;
34     bool active;
35     std::string options;
36   } dropdownMenu;
37   struct Rooms {
38     int id;
39     int floor;
40     std::string title;
41   };
42   std::vector<Rooms> rooms;
43   std::map<Sensor *, int> sensorsID;
44   size_t requestID;
45 };
46 } // namespace ClientDigitalTwin
```

Рисунок 3.15 – Прототип класу Manager

Для керування режимами роботи системи реалізовано метод SelectOperatingMode(), який відповідає за перемикання між режимами симуляції та керування. Метод GetMode() використовується для отримання поточного

режиму, а DrawTextMode() забезпечує його відображення в інтерфейсі користувача. Для роботи з базою даних реалізовано методи EnableTableCreation() та IsCreateTableDB(), які використовуються для контролю створення таблиць бази даних. Методи EnableSensorCreation() і IsCreateSensorDB() відповідають за контроль процесу створення сенсорів у структурі бази даних. Додатково реалізовано EnableSensorDataAdded() та IsSensorDataAdded(), які дозволяють контролювати процес додавання показників сенсорів. Для побудови логічної структури будинку використовується метод MakeRoomIdentification(), який формує інформацію про приміщення, поверхи та їх ідентифікатори. Метод GetRoomID() дозволяє отримати ідентифікатор кімнати за її назвою. Для зв'язування сенсорів із базою даних реалізовано метод AddSensorID(), який встановлює відповідність між об'єктами сенсорів та їх ідентифікаторами. Метод GetSensorsID() надає доступ до сформованої структури зв'язків. Окремо реалізовано метод CountRequestID(), який використовується для формування унікальних ідентифікаторів мережевих запитів. Це необхідно для коректної роботи механізму JSON-RPC та співставлення запитів і відповідей між клієнтом та сервером. Таким чином, клас Manager об'єднує логіку керування станом системи, роботу з базою даних, організацію структури цифрового двійника та координацію взаємодії між компонентами застосунку.

3.2.4 Організація сховища даних

Для зберігання даних у клієнтській частині цифрового двійника використовується вбудована база даних SQLite. Використання локального сховища дозволяє забезпечити збереження інформації про структуру будинку, сенсори та їх показники без необхідності використання окремого сервера баз даних. База даних інтегрована безпосередньо у клієнтський застосунок, що спрощує розгортання системи та дозволяє зменшити кількість зовнішніх залежностей. Для роботи з базою даних було підключено бібліотеку SQLite, яка забезпечує механізми створення таблиць, виконання SQL-запитів та збереження інформації. Для більш зручної взаємодії з базою даних додатково

використовується бібліотека-обгортка SQLiteCpp, яка надає високорівневий інтерфейс роботи з SQLite. Це дозволяє спростити виконання запитів, зменшити кількість низькорівневого коду та зробити роботу із збереженням даних більш структурованою й ефективною. Такий підхід дозволяє організувати компактне локальне сховище даних, необхідне для роботи системи та подальшого аналізу показників сенсорів.

Було створено прототип класу Storage, представлений на рисунку 3.16, який відповідає за централізовану роботу зі сховищем даних у клієнтській частині цифрового двійника. Даний клас реалізовано за шаблоном Singleton, що дозволяє використовувати єдиний екземпляр сховища в межах усього застосунку та уникнути створення декількох підключень до бази даних.

```
#pragma once
#include <string>
#include <SQLiteCpp/Database.h>
#include "settings.hpp"

namespace ClientDigitalTwin {
class Storage final {
public:
  static Storage &Instance() {
    static Storage instance(dbName: DB_NAME);
    return instance;
  }
  void Init();
  Storage(const Storage &) = delete;
  Storage &operator=(const Storage &) = delete;
  SQLite::Database *Get();
  int CreateHome(const std::string &name);
  int CreateRoom(int homeID, const std::string &name, int floor);
  int CreateSensor(int roomID, const std::string &name);
  void AddSensorData(int sensorID, float value);

  private:
  int addHome(const std::string &name);
  int addRoom(int homeID, const std::string &name, int floor);
  int addSensor(int roomID, const std::string &name);
  Storage(const std::string &dbName);
  SQLite::Database database;
};
} // namespace ClientDigitalTwin
```

Рисунок 3.16 – Прототип класу Storage

Метод Instance() використовується для отримання глобального екземпляра класу та забезпечує єдину точку доступу до сховища даних. Метод Init()

відповідає за початкову ініціалізацію бази даних та підготовку необхідної структури для подальшої роботи. Метод Get() надає доступ до об’єкта бази даних та використовується для виконання операцій взаємодії зі сховищем. Для створення логічної структури будинку реалізовано метод CreateHome(), який відповідає за створення запису будинку, CreateRoom() використовується для створення кімнат, а CreateSensor() відповідає за додавання сенсорів до системи. Для накопичення показників сенсорів реалізовано метод AddSensorData(), який забезпечує збереження значень датчиків для подальшого аналізу та використання в системі цифрового двійника. У приватній частині класу розташовані допоміжні методи addHome(), addRoom() та addSensor(), які реалізують низькорівневу логіку додавання даних у базу. Конструктор Storage() відповідає за створення підключення до бази даних, а поле database містить екземпляр об’єкта SQLite, через який здійснюється взаємодія зі сховищем.

Таким чином, клас Storage реалізує централізований механізм керування локальним сховищем даних та забезпечує структуровану роботу з інформацією.

Після проєктування класу Storage було реалізовано його основні методи, які забезпечують створення та керування структурою бази даних цифрового двійника.

Конструктор Storage() відповідає за відкриття або створення файлу бази даних SQLite. Під час створення об’єкта виконується встановлення з’єднання зі сховищем, що дозволяє надалі виконувати SQL-запити до бази даних. Одним із найважливіших методів класу є Init() на рисунку 3.17. Даний метод виконує початкову ініціалізацію бази даних, активує підтримку зовнішніх ключів foreign key та створює необхідні таблиці, якщо вони ще не існують. У результаті формується структура сховища, що складається із таблиць House, Room, Sensor та SensorData.

```
6  void ClientDigitalTwin::Storage::Init() {
7    database.exec(apQueries: "PRAGMA foreign_keys = ON;");
8
9    database.exec(apQueries: R"(
10     CREATE TABLE IF NOT EXISTS House (
11         id INTEGER PRIMARY KEY AUTOINCREMENT,
12         name TEXT
13     );
14 )");
15
16   database.exec(apQueries: R"(
17     CREATE TABLE IF NOT EXISTS Room (
18         id INTEGER PRIMARY KEY AUTOINCREMENT,
19         house_id INTEGER,
20         name TEXT,
21         floor INTEGER,
22         FOREIGN KEY(house_id) REFERENCES House(id)
23     );
24 )");
25
26   database.exec(apQueries: R"(
27     CREATE TABLE IF NOT EXISTS Sensor (
28         id INTEGER PRIMARY KEY AUTOINCREMENT,
29         room_id INTEGER,
30         type TEXT,
31         FOREIGN KEY(room_id) REFERENCES Room(id)
32     );
33 )");
34
35   database.exec(apQueries: R"(
36     CREATE TABLE IF NOT EXISTS SensorData (
37         id INTEGER PRIMARY KEY AUTOINCREMENT,
38         sensor_id INTEGER,
39         value FLOAT,
40         timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
41         FOREIGN KEY(sensor_id) REFERENCES Sensor(id)
42     );
43 )");
44 }
```

Рисунок 3.17 – Ініціалізація таблиць в базі даних

Для створення логічної структури цифрового двійника реалізовано методи CreateHome(), CreateRoom() та CreateSensor() на рисунку 3.18. Перед додаванням нового запису вони виконують перевірку наявності відповідного об'єкта в базі даних. Якщо запис уже існує, повертається його ідентифікатор, інакше виконується створення нового запису. Такий підхід дозволяє уникнути дублювання даних та забезпечує цілісність структури сховища.

```cpp
int ClientDigitalTwin::Storage::CreateHome(const std::string &name) {
  SQLite::Statement query(aDatabase: this->database,
                          apQuery: "SELECT id FROM House WHERE name = ? LIMIT 1;");
  query.bind(aIndex: 1, aValue: name);

  if (query.executeStep()) {
    auto id: int32_t = query.getColumn(aIndex: 0).getInt();
    query.reset();
    return id;
  }
  query.reset();
  return addHome(name);
}

int ClientDigitalTwin::Storage::CreateRoom(int homeID, const std::string &name,
                                           int floor) {
  SQLite::Statement query(
      aDatabase: this->database,
      apQuery: "SELECT id FROM Room WHERE house_id = ? AND name = ? LIMIT 1;");
  query.bind(aIndex: 1, aValue: homeID);
  query.bind(aIndex: 2, aValue: name);

  if (query.executeStep()) {
    auto id: int32_t = query.getColumn(aIndex: 0).getInt();
    query.reset();
    return id;
  }
  query.reset();
  return addRoom(homeID, name, floor);
}

int ClientDigitalTwin::Storage::CreateSensor(int roomID,
                                             const std::string &name) {
  SQLite::Statement query(
      aDatabase: this->database,
      apQuery: "SELECT id FROM Sensor WHERE room_id = ? AND type = ? LIMIT 1;");
  query.bind(aIndex: 1, aValue: roomID);
  query.bind(aIndex: 2, aValue: name);

  if (query.executeStep()) {
    auto id: int32_t = query.getColumn(aIndex: 0).getInt();
    query.reset();
    return id;
  }
```

Рисунок 3.18 – Реалізація створення та перевірки записів у базі даних

Безпосереднє додавання записів реалізовано у допоміжних методах addHome(), addRoom() та addSensor() на рисунку 3.19. Дані методи виконують SQL-запити INSERT та повертають ідентифікатор створеного запису для подальшого використання в системі.

```cpp
int ClientDigitalTwin::Storage::addHome(const std::string &name) {
  SQLite::Statement query(aDatabase: this->database,
                          apQuery: "INSERT INTO House (name) VALUES (?);");
  query.bind(aIndex: 1, aValue: name);
  query.exec();
  query.reset();
  return this->database.getLastInsertRowid();
}

int ClientDigitalTwin::Storage::addRoom(int homeID, const std::string &name,
                                        int floor) {
  SQLite::Statement query(
      aDatabase: this->database,
      apQuery: "INSERT INTO Room (house_id, name, floor) VALUES (?, ?, ?);");
  query.bind(aIndex: 1, aValue: homeID);
  query.bind(aIndex: 2, aValue: name);
  query.bind(aIndex: 3, aValue: floor);
  query.exec();
  query.reset();
  return this->database.getLastInsertRowid();
}

int ClientDigitalTwin::Storage::addSensor(int roomID, const std::string &name) {
  SQLite::Statement query(aDatabase: this->database,
                          apQuery: "INSERT INTO Sensor (room_id, type) VALUES (?, ?);");
  query.bind(aIndex: 1, aValue: roomID);
  query.bind(aIndex: 2, aValue: name);
  query.exec();
  query.reset();
  return this->database.getLastInsertRowid();
}

void ClientDigitalTwin::Storage::AddSensorData(int sensorID, float value) {
  SQLite::Statement query(
      aDatabase: this->database,
      apQuery: "INSERT INTO SensorData (sensor_id, value) VALUES (?, ?);");
  query.bind(aIndex: 1, aValue: sensorID);
  query.bind(aIndex: 2, aValue: value);
  query.exec();
  query.reset();
}
```

Рисунок 3.19 – Реалізація додавання записів до бази даних

Для накопичення історії показників сенсорів використовується метод AddSensorData(). Під час його виклику до таблиці SensorData додається новий запис, який містить ідентифікатор сенсора, його поточне значення та часову мітку створення запису. Це дозволяє організувати зберігання історичних даних та виконувати подальший аналіз показників цифрового двійника. Таким чином, реалізований клас Storage забезпечує повний цикл роботи зі сховищем даних: від створення структури бази даних до збереження інформації про будинок, кімнати, сенсори та результати їх вимірювань.

3.3 Реалізація службових структур і параметрів системи

Для забезпечення роботи клієнтської частини системи також було створено набір допоміжних структур, констант і перелічуваних типів, які зосереджені у файлі налаштувань рисунок 3.20. Вони використовуються для централізації

параметрів програми та забезпечують зв’язок між окремими компонентами системи.

```
#pragma once
#include <string>
#include <array>
#include <map>

namespace ClientDigitalTwin {
extern const std::string CONFIG_FILE;
extern const int PING_INTERVAL;
extern const int SIZE;
enum Tag { HOME_WALL_SOLID, SENSOR, UPDATE, SET, ROOM, HOME_OPENING };
extern const std::array<std::string, 6> TAGS;
enum class Mode { SIMULATION, CONTROL };
typedef std::map<std::string, float> Option;

struct URL {
  std::string host;
  long port;
};

extern const std::string DB_NAME;
} // namespace ClientDigitalTwin
```

Рисунку 3.20 – Опціональні налаштування клієнта

Константи CONFIG_FILE та DB_NAME визначають шляхи до конфігураційного файлу та локальної бази даних відповідно. Константи PING_INTERVAL і SIZE використовуються для налаштування параметрів роботи клієнта та мережевої взаємодії. Перелічуваний тип Tag містить набір службових ідентифікаторів повідомлень, які використовуються під час обміну даними між клієнтом та сервером. Для зручної роботи з ними реалізовано масив TAGS, що містить текстові представлення відповідних тегів. Перелічуваний тип Mode визначає режими функціонування системи та дозволяє перемикатися між режимом симуляції SIMULATION і режимом керування CONTROL.

Тип Option реалізовано на основі контейнера std::map, який використовується для зберігання параметрів та налаштувань сенсорів у вигляді пар “ключ-значення”. Для зберігання мережевих параметрів використовується структура URL, яка містить адресу сервера та номер порту, необхідні для встановлення WebSocket з’єднання. Таким чином, файл налаштувань виступає централізованим сховищем службових параметрів системи та забезпечує узгоджену взаємодію між її компонентами.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЦИФРОВОГО ДВІЙНИКА

4.1 Демонстрація роботи розробленої програми

Демонстраційна робота розробленої системи виконується на базі операційної системи Linux, а саме дистрибутива Ubuntu 22.04. Перед запуском застосунку необхідно виконати підготовку середовища та послідовно запустити серверну і клієнтську частини системи. Першим етапом запуску є старт серверної частини застосунку. Для цього через термінал виконується запуск скомпільованого виконуваного файлу server. Після успішної ініціалізації сервер встановлює WebSocket з’єднання та переходить у режим очікування запитів від клієнта.

У разі коректного запуску в терміналі відображається інформаційне повідомлення:

[*] WebSocket server start ws://127.1.1.1:5222

Даний рядок підтверджує запуск серверної частини та містить адресу і порт, за якими клієнт може встановити з’єднання. Сервер після запуску відповідає за обробку запитів, передачу даних цифрового двійника та взаємодію з клієнтською частиною через протокол WebSocket. Після запуску сервера виконується запуск клієнтської частини програми. Для цього з каталогу build запускається відповідний виконуваний файл client. Під час запуску клієнт ініціалізує графічне середовище, встановлює з’єднання із сервером та починає отримувати необхідні дані для побудови цифрового двійника. Після встановлення зв’язку між клієнтом і сервером відбувається обмін повідомленнями, передача параметрів приміщення, сенсорів та інших елементів. На основі отриманих даних клієнт формує тривимірну сцену що показано на рисунку 4.1.

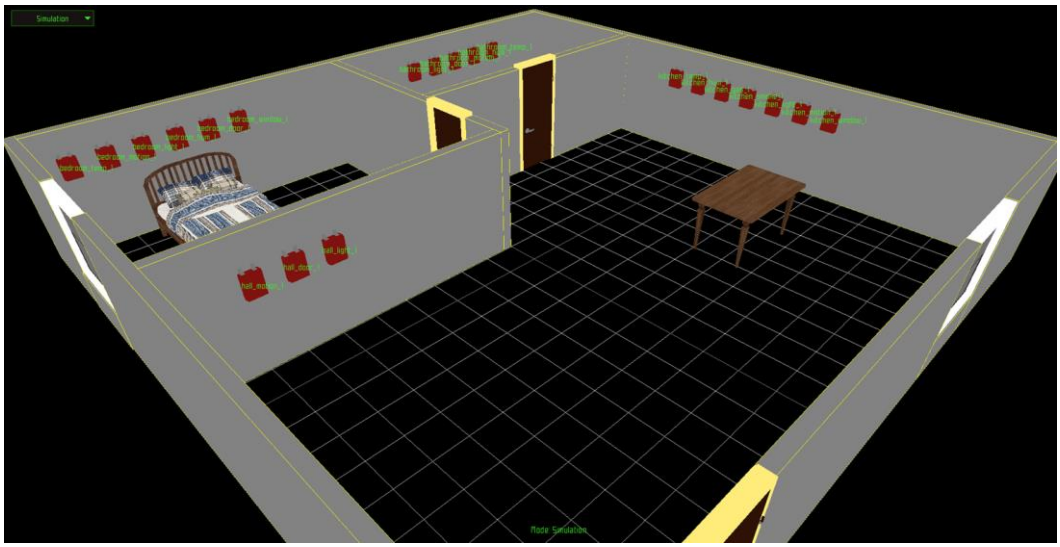
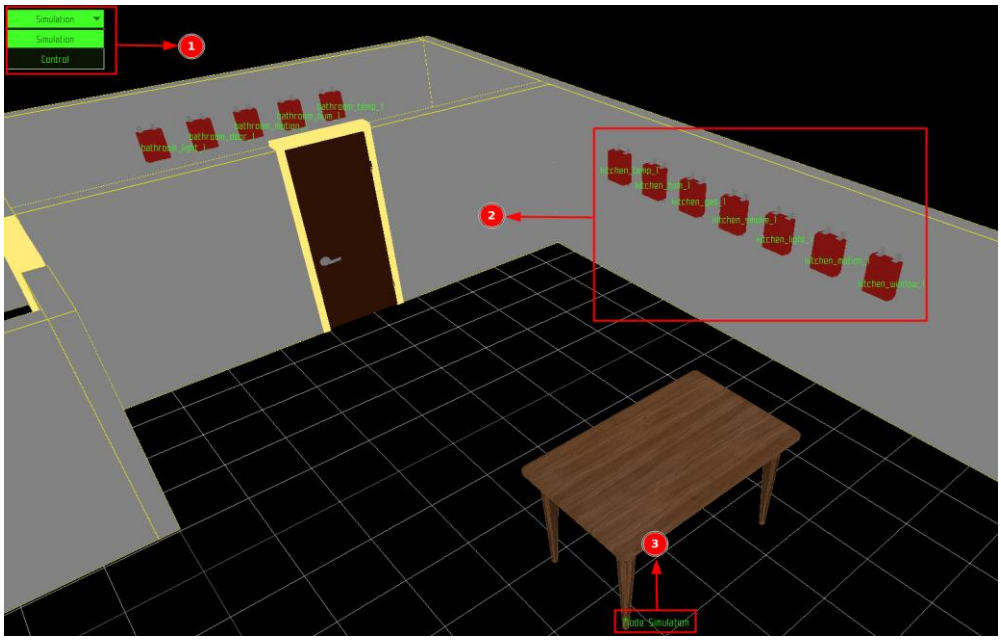


Рисунок 4.1 – Графічне представлення цифрового двійника

Таким чином, демонстрація роботи складається з послідовного запуску серверного модуля, встановлення мережевого з’єднання та подальшого запуску клієнтського застосунку, який відповідає за візуалізацію і керування цифровим двійником.

4.2 Розробка загального інтерфейсу програми

На рисунку 4.2 виділено основні елементи керування програмою, які забезпечують взаємодію оператора з цифровим двійником розумного будинку.



Рисунку 4.2 – Загальний інтерфейс програми

Меню вибору режиму роботи системи. Даний елемент дозволяє перемикатися між двома режимами функціонування програми це Simulation та Control.

Режим Simulation призначений для роботи з цифровим двійником у режимі sandbox. Він надає можливість моделювати різні сценарії роботи системи без впливу на реальні пристрої. У цьому режимі можна імітувати підвищення температури в приміщенні, виникнення задимлення, витік газу або води, а також несанкціоноване проникнення до будинку. Це дозволяє досліджувати поведінку системи за різних умов та оцінювати ефективність її роботи. Усі зміни відбуваються лише в межах цифрового двійника, що робить процес тестування повністю безпечним.

Режим Control використовується для безпосереднього керування системою розумного будинку. У реальній системі даний режим забезпечував би передачу команд на фізичні пристрої. У межах розробленого прототипу серверна частина виконує облік змін стану системи та імітує реакцію реальних пристроїв. Таким чином забезпечується робота цифрового двійника у псевдофізичному режимі. Перехід до режиму Control покладає відповідальність за керування системою на оператора, оскільки його дії можуть впливати на роботу обладнання. Наприклад, оператор може активувати охоронну сигналізацію або змінити стан окремих компонентів системи.

Інтерфейси сенсорів. Інтерфейси сенсорів виступають розширюваними елементами системи та забезпечують взаємодію користувача з окремими компонентами цифрового двійника. Після відкриття вікна сенсора користувач отримує доступ до додаткових елементів керування та засобів симуляції. Через ці інтерфейси можна змінювати параметри сенсорів, моделювати різні ситуації та спостерігати за реакцією системи. Детальний опис інтерфейсів сенсорів наведено в наступних підрозділах.

Інформаційна панель режиму роботи. Даний елемент відображає поточний режим функціонування програмного забезпечення та інформує оператора про стан системи. На поточному етапі реалізації підтримуються режими Simulation

та Control, між якими користувач може вільно перемикатися під час роботи із цифровим двійником.

4.3 Розробка інтерфейсу сенсорів

Сенсор у системі представлений у вигляді невеликого навісного модуля коричневого кольору з чотирма виводами, два з яких розташовані у верхній частині корпусу, а два у нижній. Таке візуальне виконання створює асоціацію з реальним електронним пристроєм, що може бути встановлений на стіні приміщення та входити до складу системи розумного будинку. Кожен сенсор є інтерактивним елементом цифрового двійника та підтримує взаємодію з користувачем. Для зручності роботи над кожним модулем відображається його назва та інформація про приміщення, до якого він належить. Це дозволяє оператору швидко ідентифікувати тип сенсора та його розташування в межах будинку. Зовнішній вигляд сенсора представлено на рисунку 4.3.



Рисунку 4.3 – Зовнішній вигляд сенсора

Після активації сенсора навколо нього з’являється контурна рамка, яка візуально повідомляє оператору про те, що даний елемент перебуває в активному стані та вибраний для подальшої взаємодії. Одночасно з цим відкривається вікно керування сенсором, яке надає доступ до його параметрів, режимів роботи та засобів симуляції. Такий підхід спрощує взаємодію з цифровим двійником та

забезпечує наочне відображення поточного обраного елемента. Приклад активного сенсора та його вікна керування представлено на рисунку 4.4.

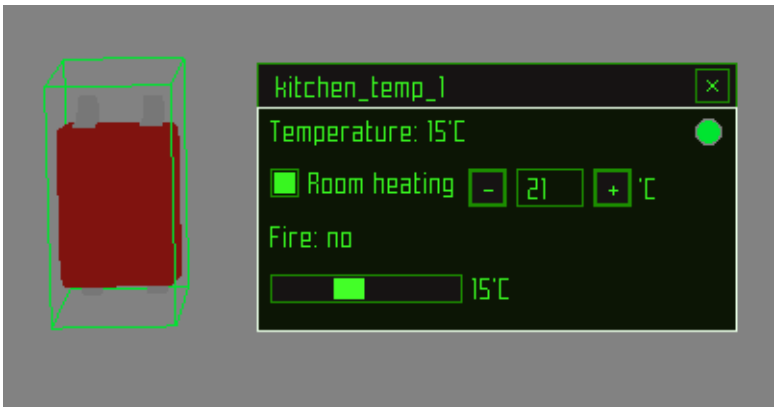


Рисунок 4.4 – Сенсор в активній роботі

4.4 Огляд інтерфейсів сенсорів

Сенсор температури має вікно керування, яке містить набір елементів для моніторингу та взаємодії з системою. У верхній частині вікна відображається назва сенсора та розташована кнопка закриття інтерфейсу. Основним інформаційним елементом є поле поточного значення температури, яке відображає температуру в приміщенні в реальному часі або в режимі симуляції. Для взаємодії із системою передбачено сигнальну кнопку та елемент керування клімат-контролем. Оператор має можливість вмикати або вимикати систему клімат-контролю, а також задавати бажане значення температури. Це дозволяє моделювати роботу автоматизованих систем підтримки мікроклімату будинку.

Також інтерфейс містить інформаційне поле попередження про можливу пожежну небезпеку. Даний елемент використовується для відображення аварійних станів у разі перевищення допустимих температурних значень. Для режиму Simulation реалізовано спеціальний елемент у вигляді повзунка, який дозволяє змінювати температуру в приміщенні вручну. Під час зміни значення оператор може спостерігати за реакцією цифрового двійника та пов’язаних компонентів системи. Вікно керування сенсором температури представлено на рисунку 4.5.

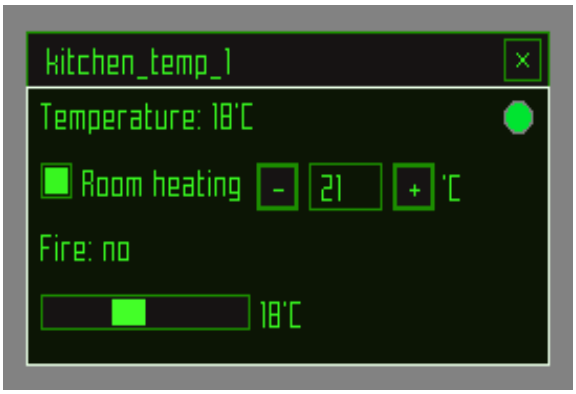


Рисунок 4.5 – Вікно керування сенсором температури

Сенсор вологості також має власне вікно керування, яке містить елементи для моніторингу та керування рівнем вологості в приміщенні. Основним інформаційним елементом є поле відображення поточного значення вологості у відсотках, що дозволяє оператору контролювати стан мікроклімату. Для підтримання необхідного рівня вологості передбачено елемент керування системою автоматичного регулювання, який дозволяє вмикати або вимикати підтримання вологості в приміщенні. Додатково інтерфейс містить інформаційне поле сповіщення про можливий витік води, що дає змогу своєчасно виявляти аварійні ситуації. У режимі Simulation користувач може змінювати рівень вологості за допомогою повзунка, імітуючи різні умови середовища та спостерігаючи за реакцією цифрового двійника на зміну параметрів. Вікно керування сенсором вологості представлено на рисунку 4.6.

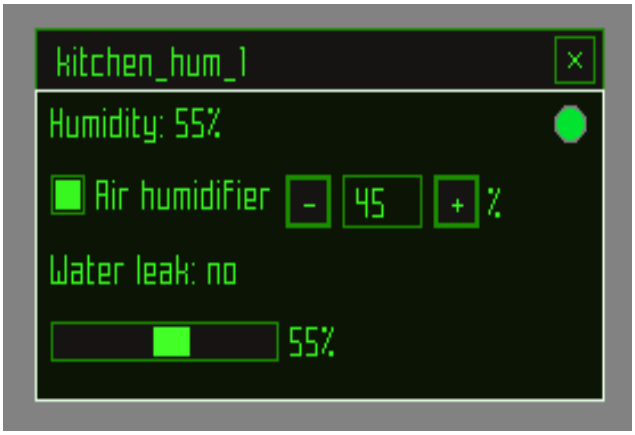


Рисунок 4.6 – Вікно керування сенсором вологості

Датчик газу також має власне вікно керування, яке призначене для відображення стану газової системи та моделювання її поведінки в межах цифрового двійника. Основним інформаційним елементом є поле стану подачі газу, яке показує, чи відкрито або перекрито подачу газу в приміщенні. Додатково в інтерфейсі відображається статус витоку газу, який може перебувати у нормальному режимі normal або аварійному стані accident. Це дозволяє оператору відстежувати потенційно небезпечні ситуації та реакцію системи на них. Для симуляції роботи датчика передбачено повзунок, який дозволяє регулювати рівень подачі газу у відсотках. Таким чином моделюється заповнення приміщення газом та можливі сценарії виникнення витоку. Є кнопка перекриття відкриття газу. Вікно керування датчиком газу представлено на рисунку 4.7.

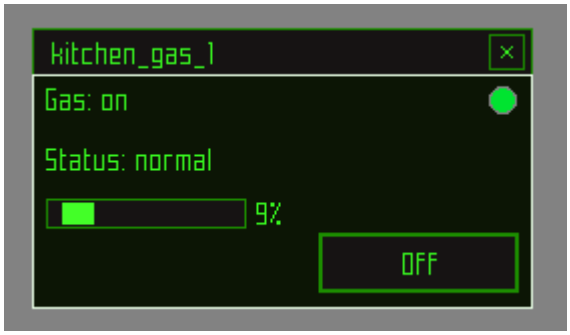


Рисунок 4.7 – Вікно керування сенсором газу

Датчик диму, як і інші сенсори системи, має власне вікно керування, яке містить назву сенсора, кнопку закриття вікна та сигнальний індикатор стану. Основним елементом є поле відображення рівня задимлення в приміщенні та статус безпеки. Система підтримує два стани роботи: normal, коли концентрація диму перебуває в допустимих межах, та accident, коли рівень задимлення перевищує безпечне значення і виникає потенційна пожежна небезпека.Для моделювання роботи сенсора в режимі Simulation реалізовано повзунок керування концентрацією диму. Значення задається у відсотках, що дозволяє імітувати поступове задимлення приміщення та спостерігати за реакцією цифрового двійника на зміну умов середовища. Вікно керування датчиком диму представлено на рисунку 4.8.

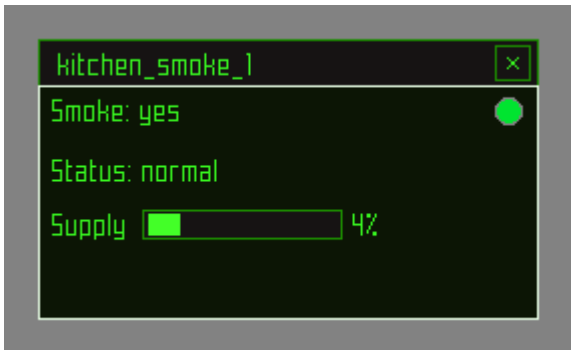


Рисунок 4.8 – Вікно керування сенсором диму

Датчик освітлення також має власне вікно керування, яке дозволяє контролювати стан системи освітлення в приміщенні. У вікні відображається поточний стан освітлення, що повідомляє оператору, чи увімкнене світло в кімнаті. Для швидкого визначення стану системи передбачено сигнальний індикатор. Інтерфейс сенсора містить елементи керування освітленням, які дозволяють вмикати та вимикати світло. Додатково реалізовано регулювання яскравості, що дає можливість змінювати інтенсивність освітлення залежно від потреб користувача. Таким чином, датчик освітлення забезпечує як моніторинг поточного стану освітлення, так і керування параметрами роботи освітлювальної системи розумного будинку. Вікно керування датчиком освітлення представлено на рисунку 4.9.

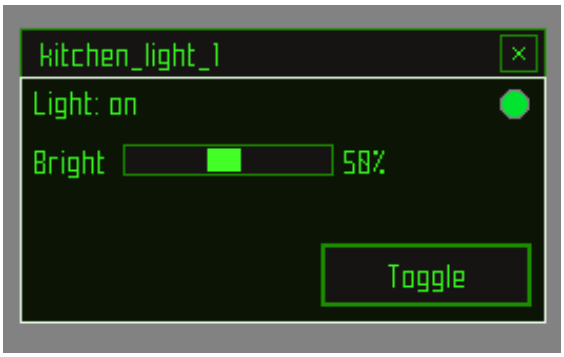


Рисунок 4.9 – Вікно керування сенсором освітлення

Датчик руху має власне вікно керування, яке дозволяє контролювати стан системи виявлення руху в приміщенні. Інтерфейс містить сигнальний індикатор та поле сповіщення, яке відображає наявність або відсутність руху в зоні

контролю сенсора. Для керування роботою датчика передбачено опцію ввімкнення або вимкнення відстеження руху. Також реалізовано можливість постановки приміщення під охорону, що дозволяє використовувати датчик як елемент системи безпеки розумного будинку. У разі виявлення руху під час активованої охорони система може формувати відповідне попередження для оператора. Для режиму Simulation передбачено окрему кнопку тестування руху, яка дозволяє імітувати появу людини в приміщенні та перевіряти реакцію цифрового двійника на подію спрацювання датчика. Вікно керування датчиком руху представлено на рисунку 4.10.

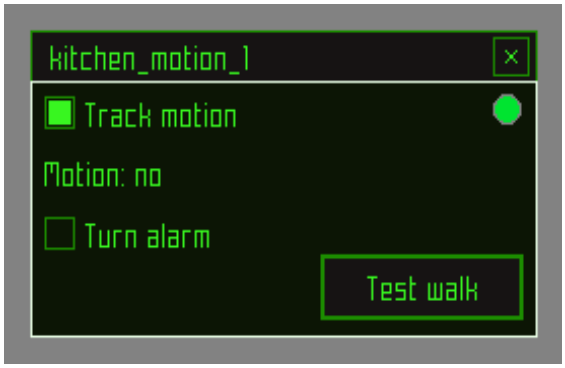


Рисунок 4.10 – Вікно керування сенсором руху

Датчик відкриття вікна також має власне вікно керування, призначене для контролю стану віконних конструкцій у приміщенні. Основним інформаційним елементом є поле стану, яке відображає, чи перебуває вікно у відкритому або закритому положенні. Інтерфейс також містить сигнальний індикатор, який використовується для відображення стану охоронної системи. У випадку спрацювання сигналізації. Для забезпечення функцій безпеки реалізовано елемент постановки вікна під охорону. Після активації даної опції система починає контролювати стан вікна та реагувати на його відкриття. Це дозволяє використовувати датчик як складову частину охоронної системи розумного будинку. Для керування станом вікна реалізовано кнопку відкриття та закриття. Вікно керування датчиком відкриття вікна представлено на рисунку 4.11.

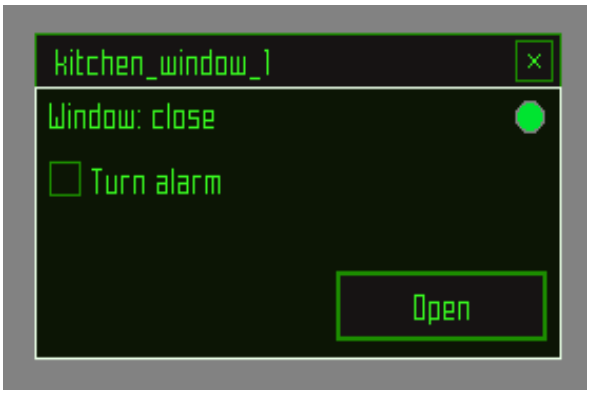


Рисунок 4.11 – Вікно керування сенсором відкриття вікна.

Датчик відкриття дверей має власне вікно керування, яке дозволяє контролювати стан дверей та забезпечувати функції безпеки розумного будинку. Основним елементом інтерфейсу є поле стану дверей, яке відображає, чи перебувають двері у відкритому або закритому положенні. Для контролю аварійних ситуацій передбачено індикатор тривоги, який сигналізує про спрацювання охоронної системи у випадку несанкціонованого відкриття дверей. Також інтерфейс містить опцію постановки дверей під охорону, , що дозволяє використовувати датчик як елемент системи безпеки будинку. Для керування станом дверей реалізовано окремі кнопку відкриття та закриття. У режимі Control вони можуть використовуватися для керування виконавчими механізмами, а в режимі Simulation дозволяють моделювати роботу дверей та перевіряти реакцію цифрового двійника на зміну їх стану. Вікно керування датчиком відкриття дверей представлено на рисунку 4.12.



Рисунок 4.12 – Вікно керування сенсором відкриття дверей

4.5 Демонстрація роботи програми

Для демонстрації можливостей розробленого цифрового двійника було проведено експериментальне дослідження роботи системи в режимі симуляції. Під час експерименту виконувалося керування станом вікна в спальні, а також моделювалося підвищення температури в приміщенні з подальшим аналізом реакції системи на зміну умов середовища. На початковому етапі експерименту було активовано панелі керування датчиками, розташованими у спальні. Це надало доступ до елементів керування сенсорами та дозволило виконувати зміну їхніх параметрів у режимі реального часу. Активовані інтерфейси датчиків спальні представлені на рисунку 4.13.

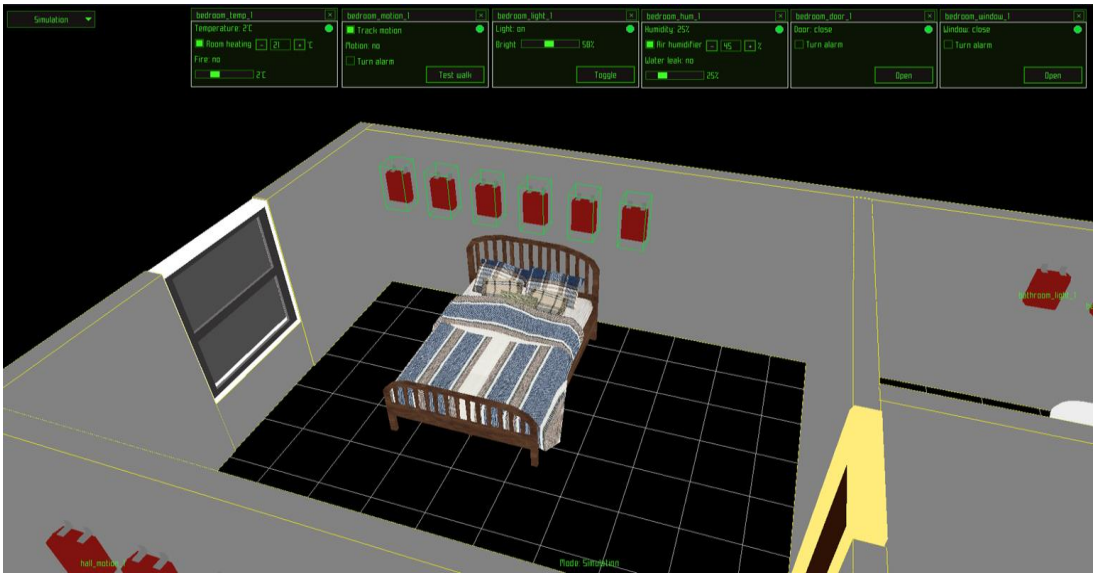


Рисунок 4.13 – Інтерфейси датчиків спальні

На даному етапі система перебуває в режимі Simulation, тому всі дії виконуються лише в межах цифрового двійника та не впливають на реальні пристрої. Для здійснення безпосереднього керування вікном необхідно перейти до режиму Control. Перехід до режиму керування виконується шляхом вибору пункту Control у меню вибору режимів роботи програми. Після перемикання система переходить у стан, який дозволяє надсилати команди керування та виконувати дії, що імітують роботу реального розумного будинку. Процес переходу до режиму Control представлено на рисунку 4.14.

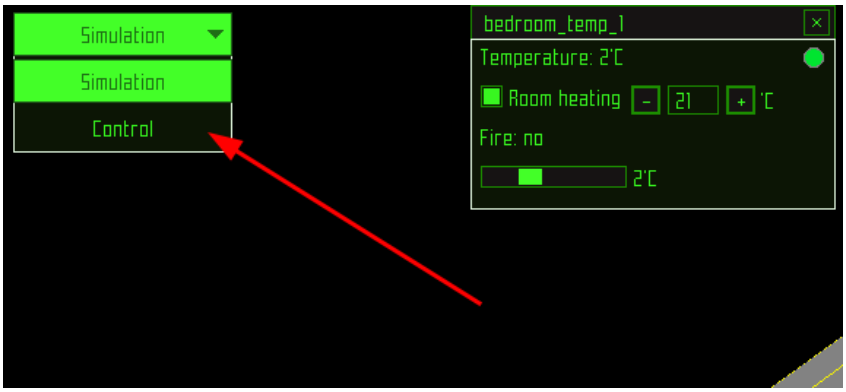


Рисунок 4.14 – Перехід до режиму Control

Після переходу до режиму Control у користувача з'являється можливість безпосередньо керувати елементами системи розумного будинку. Для зручності проведення експерименту інтерфейси сенсорів були розташовані таким чином, щоб забезпечити швидкий доступ до необхідних елементів керування. Для відкриття вікна було використано інтерфейс сенсора Window. Керуючий вплив здійснюється натисканням кнопки Open, після чого система змінює стан вікна та відображає відповідні зміни у цифровому двійнику/ Кнопка має тепер напис Close. Процес відкриття вікна представлено на рисунку 4.15.

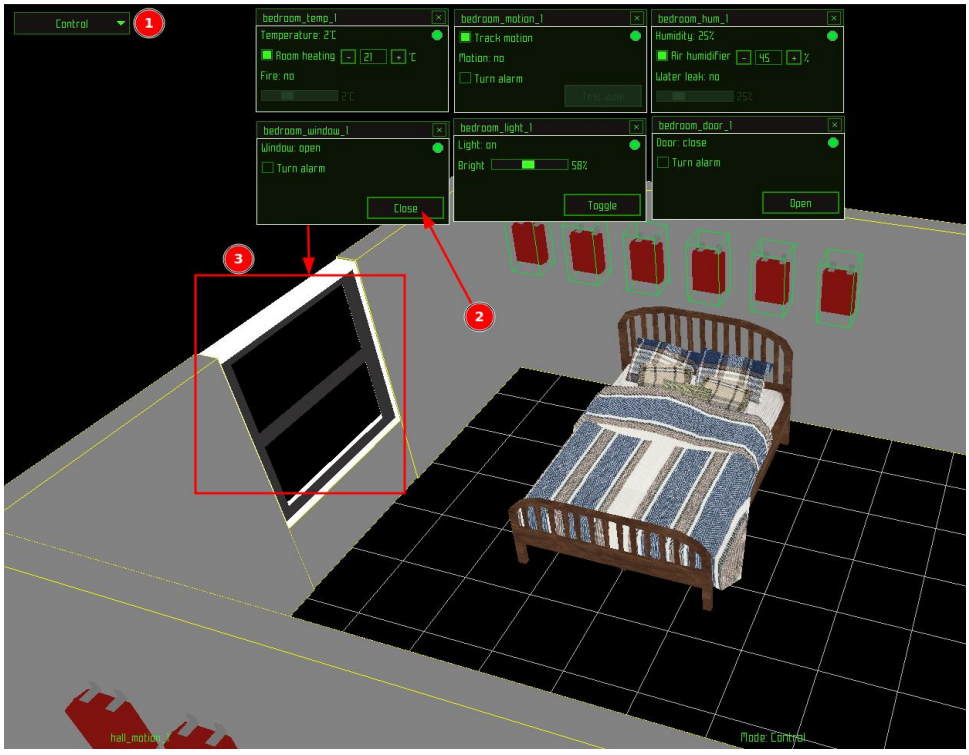


Рисунок 4.15 – Покроковий процес відкриття вікна

Далі було проведено експеримент із моделюванням аварійної ситуації, пов'язаної з різким підвищенням температури в приміщенні. Для цього система була переведена в режим Simulation, після чого було відкрито інтерфейс сенсора температури та за допомогою повзунка встановлено значення температури понад 100 °C. Після зміни параметра система автоматично відреагувала на виникнення небезпечної ситуації. Насамперед сигнальний індикатор змінив свій колір на червоний, що свідчить про вихід системи за межі нормального режиму роботи та виникнення аварійної події. Крім того, враховуючи критично високе значення температури, система сформувала припущення про можливе виникнення пожежі. У результаті поле Fire, яке відповідає за індикацію відкритого вогню, змінило свій стан на On, повідомляючи оператора про підозру на займання в приміщенні. Результат моделювання аварійної ситуації представлено на рисунку 4.16.

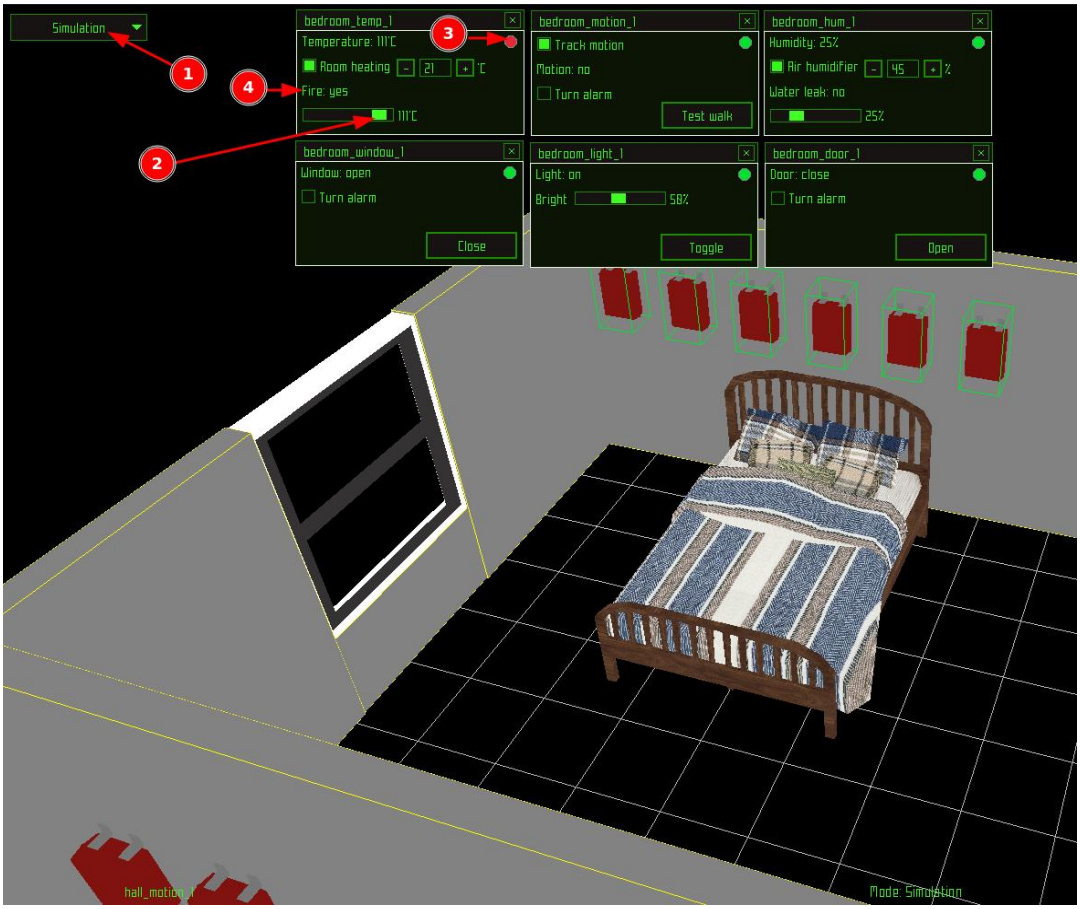


Рисунок 4.16 – Реакція системи на критичне підвищення температури

Після повернення до режиму Control система зберігає попередній стан без змін, які були внесені під час симуляції. Це означає, що всі експериментальні дії, виконані в режимі Simulation, не впливають на реальний стан керованої системи. Таким чином підтверджується, що режим симуляції є ізольованим середовищем для тестування та моделювання сценаріїв і не взаємодіє безпосередньо з режимом керування. Перехід назад до Control відновлює актуальний стан системи без врахування симульованих змін. Як видно з рисунка 4.17, система після виходу з режиму Simulation та повторного переходу до Control зберігає попередній актуальний стан.

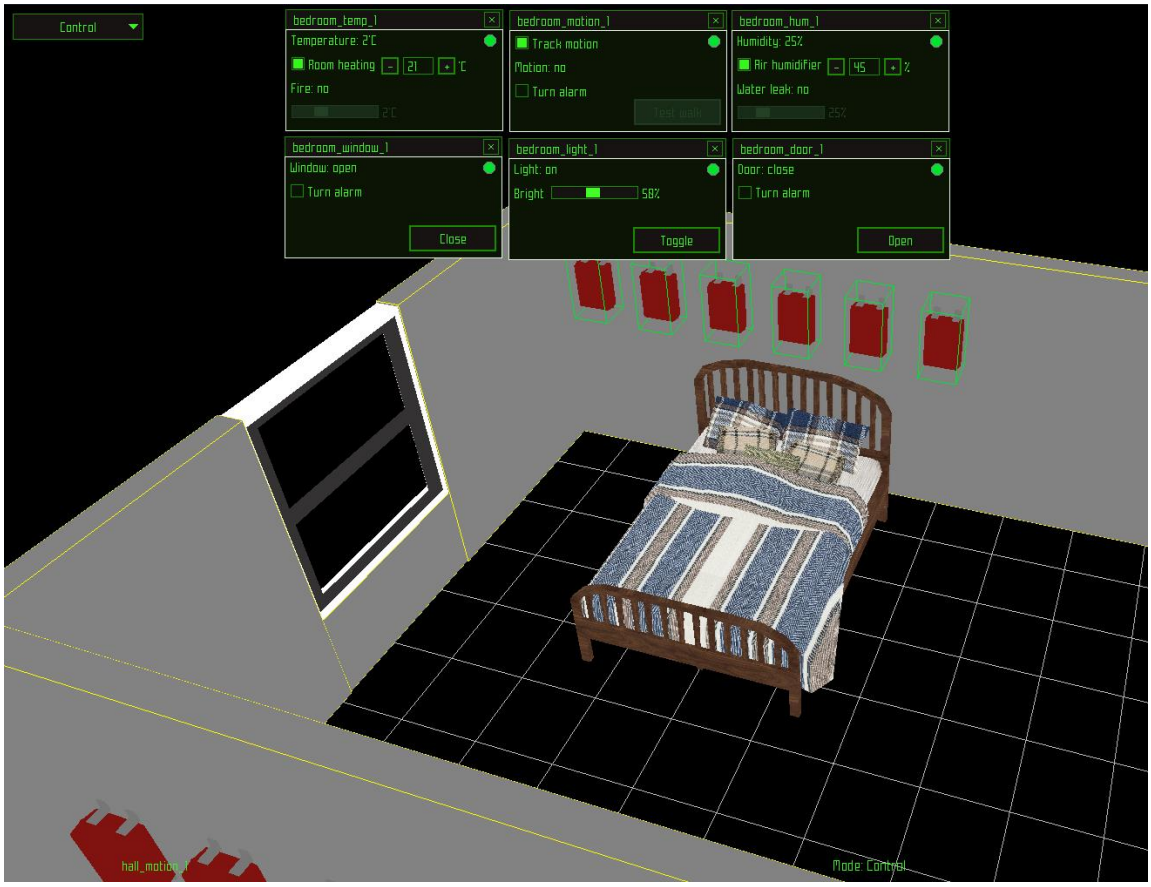


Рисунок 4.17 – Повернення системи до режиму керування після симуляції

Також було проведено перевірку роботи бази даних, у ході якої підтверджено коректний запис показників із сенсорів без помилок та втрати точності. Було знайдено сенсор температури у спальні та перевірено його значення у базі даних. Отримані дані повністю збігаються з фактичними значеннями сенсора без округлення та спотворень. Таким чином

експериментально підтверджено, що система коректно записує дані до бази даних, а облік показників сенсорів виконується стабільно та без порушень цілісності інформації. На рисунку 4.18 представлено запис значення сенсора температури спальні в базі даних.

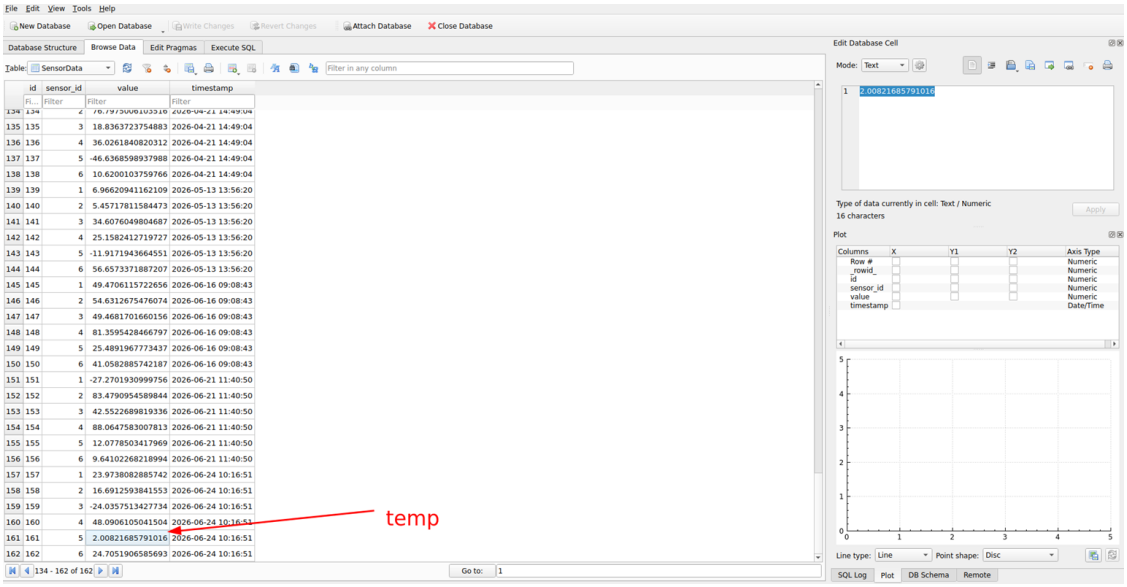


Рисунок 4.18 – Збереження даних сенсора температури спальні у SQLite

Таким чином, у ході експериментальних досліджень підтверджено коректну роботу системи як у режимі керування, так і в режимі симуляції. Система адекватно реагує на зміну параметрів та виникнення інцидентів, забезпечуючи відповідне відображення станів у цифровому двійнику. Також встановлено, що значення, отримані в режимі Control, коректно записуються до бази даних, що підтверджує правильну роботу механізму збереження даних. Експериментально доведено стабільність, узгодженість та працездатність усіх основних компонентів системи.

4.6 Охорона праці

Під час розроблення програмного забезпечення цифрового двійника розумного будинку робочим місцем розробника є приміщення, обладнане персональним комп'ютером, засобами відображення інформації, периферійними

пристроями та офісними меблями. Робота належить до категорії розумової праці та характеризується тривалим використанням персонального комп'ютера. Робоче місце обладнується регульованими столом і кріслом. Відстань від очей користувача до екрана монітора повинна становити 600–700 мм, верхня межа екрана має знаходитися на рівні очей або трохи нижче. Клавіатура розташовується на відстані 100–300 мм від краю столу.

Температура повітря у приміщенні повинна підтримуватися в межах 22–24 °С, відносна вологість – 40–60 %, швидкість руху повітря – не більше 0,1 м/с. Особливу увагу необхідно приділити організації режиму праці та відпочинку. При роботі за персональним комп'ютером рекомендується через кожні 60 хвилин роботи робити перерву тривалістю 10–15 хвилин для зменшення втоми очей та опорно-рухового апарату.

Якісне освітлення є одним із найважливіших факторів під час виконання робіт із програмування, оскільки недостатня освітленість призводить до швидкої втоми, погіршення концентрації уваги та збільшення кількості помилок. Проведемо розрахунок загального рівномірного освітлення методом коефіцієнта використання світлового потоку. Приймемо такі вихідні дані:

– довжина приміщення A=6 м;

– ширина B = 4 м;

– площа S = 24 м²;

– висота приміщення H = 3 м;

– висота робочої поверхні hp = 0,8 м;

– висота підвісу світильників h = 2,2 м;

– нормована освітленість для роботи за ПК En = 400 лк;

– коефіцієнт запасу Kz = 1,5;

– коефіцієнт нерівномірності Z = 1,1;

– коефіцієнт використання світлового потоку η = 0,6.

Необхідний сумарний світловий потік можна визначити за формулою:

$$Ф = Eн \cdot S \cdot Kz \cdot Z / \eta, \quad (4.1)$$

Підставимо числові значення до формули:

$$F\Sigma = 400 \cdot 24 \cdot 1{,}5 \cdot 1{,}1 / 0{,}6 = 26400 \text{ лм.} \quad (4.2)$$

Використаємо світлодіодні світильники зі світловим потоком Ф1=4400 лм. Тоді необхідна кількість світильників дорівнюватиме:

$$N = \Phi / \Phi 1 = 26400 / 4400 = 6. \quad (4.3)$$

Отже, необхідно встановити шість LED-світильників. Перевіримо фактичну освітленість приміщення, яке становитиме:

$$E = N \cdot \Phi 1 \cdot \eta / S \cdot Kz \cdot Z, \quad (3.4)$$

$$E = 6 \cdot 4400 \cdot 0{,}6 / 24 \cdot 1{,}5 \cdot 1{,}1 = 400 \text{ лк.}$$

У результаті аналізу умов праці розробника програмного забезпечення цифрового двійника розумного будинку визначено основні небезпечні та шкідливі виробничі фактори й запропоновано заходи щодо їх усунення. Виконано розрахунок штучного освітлення методом коефіцієнта використання світлового потоку, який показав, що для приміщення площею 24 м² достатньо встановити шість світлодіодних світильників зі світловим потоком 4400 лм кожний. Отримана освітленість 400 лк відповідає вимогам чинних нормативних документів для робіт, пов'язаних із використанням персонального комп'ютера. Запропоновані заходи з електро- та пожежної безпеки забезпечують безпечні умови праці під час розроблення програмного забезпечення. [24].

ВИСНОВКИ

При виконанні кваліфікаційної роботи на тему «Розроблення програмного забезпечення для симуляції цифрового двійника розумного будинку» було вирішено низку теоретичних і практичних завдань, пов'язаних із дослідженням та реалізацією технології цифрових двійників для систем розумного будинку.

На початковому етапі роботи проведено аналіз сучасних систем керування розумним будинком та існуючих платформ для побудови цифрових двійників. На основі отриманих результатів сформовано функціональні та нефункціональні вимоги до програмного забезпечення, а також визначено основні принципи побудови майбутньої системи.

У процесі проєктування було розроблено клієнт-серверну архітектуру застосунку та визначено розподіл функціональних обов'язків між клієнтською та серверною частинами. Також було спроєктовано модель бази даних, розроблено алгоритми взаємодії зі сховищем даних та реалізовано механізми обліку показників сенсорів.

У межах практичної реалізації створено програмне забезпечення цифрового двійника розумного будинку, яке забезпечує візуалізацію тривимірної моделі будинку, моніторинг стану сенсорів та підтримує два режими роботи таки як режим керування та режим симуляції. Реалізовано систему взаємодії з датчиками, що дозволяє як керувати окремими компонентами розумного будинку, так і моделювати різні аварійні та штатні ситуації.

Під час експериментальних досліджень було перевірено працездатність розробленої системи. Проведено експерименти з керуванням станом вікна, моделюванням підвищення температури та аналізом реакції системи на виникнення інцидентів. Також підтверджено коректність роботи підсистеми збереження даних, оскільки всі показники сенсорів без помилок записувалися до бази даних та відповідали фактичним значенням.

Отримані результати показали, що розроблене програмне забезпечення може ефективно працювати навіть на малопотужних апаратних платформах завдяки використанню мови програмування C++ та низькорівневої бібліотеки Raylib для побудови двовимірної та тривимірної графіки. Використання бібліотеки Raygui дозволило реалізувати легкий та ефективний графічний інтерфейс користувача без залучення складних зовнішніх фреймворків.

Перевагою розробленої системи є її модульність, незалежність від пропрієтарного програмного забезпечення та можливість подальшого розширення функціональності. Архітектура системи не залежить від конкретної серверної реалізації та дозволяє інтегрувати клієнтську частину з різними серверними компонентами й фізичними пристроями Інтернету речей.

Таким чином, поставлені в кваліфікаційній роботі завдання були повністю виконані, а розроблена система підтвердила свою працездатність і може бути використана як основа для подальшого розвитку цифрових двійників розумних будинків та інтеграції з реальними системами автоматизації, а отримані результати можна віднести до Цілі сталого розвитку 9 «Промисловість, інновації та інфраструктура», а саме 9.5 [25].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. – Введ. 2015-06-22. – К. Держстандарт України, 2017 – 29 с.

2. Методичні вказівки з підготовки кваліфікаційної роботи для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології освітньої програми «Системна інженерія» / Упоряд.: І.Ш. Невлюдов, О.М. Цимбал, О.В. Токарєва, А.І. Бронніков. Харків: ХНУРЕ, 2023. 65 с.

3. Huda, N. U., Ahmed, I., Adnan, M., Ali, M., & Naeem, F. (2024). Experts and intelligent systems for smart homes' Transformation to Sustainable Smart Cities: A comprehensive review. Expert Systems with Applications, 238, https://doi.org/10.1016/j.eswa.2023.122380.

4. Douha, N. G. Y. R., Bhuyan, M., Kashihara, S., Fall, D., Taenaka, Y., & Kadobayashi, Y. (2022). A survey on blockchain, SDN and NFV for the smart-home security. Internet of Things, 20, https://doi.org/10.1016/j.iot.2022.100588.

5. Hammi, B., Zeadally, S., Khatoun, R., & Nebhen, J. (2022). Survey on smart homes: Vulnerabilities, risks, and countermeasures. Computers & Security, 117, https://doi.org/10.1016/j.cose.2022.102677.

6. Kluth, O. (2026). Ihr Smart Home mit Home Assistant: Der Quickstart-Guide-Vom Einstieg bis zur individuellen Heimautomation. MITP-Verlags GmbH & Co. KG.

7. Borissova, D. I., Danev, V. K., Rashevski, M. B., Garvanov, I. G., Yoshinov, R. D., & Garvanova, M. Z. (2022). Using IoT for automated heating of a smart home by means of OpenHAB software platform. IFAC-PapersOnLine, 55(11), 90-95, https://doi.org/10.1016/j.ifacol.2022.08.054.

8. Molnár, G., Koprda, Š., & Balogh, Z. (2025, September). Design and Implementation of a Modular Smart Home System Using ESP32 and Apple HomeKit Integration. In 2025 IEEE 23rd Jubilee International Symposium on Intelligent

Systems and Informatics (SISY) (pp. 000125-000130). IEEE, DOI: 10.1109/SISY67000.2025.11205419.

9. Nugraha, G., Juliansyah, F., Firdaus, F., & Aribowo, D. (2023). Analisis Perbandingan Google Home Dan Amazon Alexa Pada Rumah Pintar (Smart Home). Jurnal Sains dan Teknologi, 2(2), 75-80, https://doi.org/10.58169/saintek.v2i2.250.

10. Aljaffan, N., Alhazzaa, L., & Albahli, A. (2023, December). Utilizing Reusable Test-Ready Models of Smart Home Systems for Testing KNX Devices. In 2023 International Conference on Computational Science and Computational Intelligence (CSCI) (pp. 1558-1564). IEEE, DOI: 10.1109/CSCI62032.2023.00257.

11. Issa, R., Hamad, M. S., & Abdel-Geliel, M. (2023, February). Digital twin of wind turbine based on microsoft® azure iot platform. In 2023 IEEE Conference on Power Electronics and Renewable Energy (CPERE) (pp. 1-8). IEEE, DOI: 10.1109/CPERE56564.2023.10119576.

12. Rashmi, R., Prahalad, H., Kumar, H. S., & Hoslok, A. (2023, November). Exploring Digital Twins for Plant Growth Monitoring. In 2023 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS) (pp. 1-6). IEEE, DOI: 10.1109/CSITSS60515.2023.10334087.

13. Singh, P. D., & Tripathi, V. (2024, February). Digital Twins: A comprehensive study on models, platforms, applications and challenges. In 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 1072-1077). IEEE, DOI: 10.23919/INDIACom61295.2024.10498975.

14. Paduraru, C., Stefanescu, A., & Jianu, A. (2024, July). Unit test generation using large language models for unity game development. In Proceedings of the 1st ACM International Workshop on Foundations of Applied Software Engineering for Games (pp. 7-13), https://doi.org/10.1145/3663532.3664466.

15. Мороз, Є. О. Модель системи захисту розумного будинку : випускна кваліфікаційна робота : 125 "Кібербезпека" / Є. О. Мороз ; керівник роботи Т. А. Петренко ; НУ "Чернігівська політехніка", Кафедра кібербезпеки та

математичного моделювання. – Чернігів, 2022. – 65 с, http://ir.stu.cn.ua/handle/123456789/26673.

16. Rock, L. Y., Tajudeen, F. P., & Chung, Y. W. (2024). Usage and impact of the internet-of-things-based smart home technology: a quality-of-life perspective. Universal access in the information society, 23(1), 345-364, DOI https://doi.org/10.1007/s10209-022-00937-0.

17. Слободенюк, І. В. Модернізація та дослідження автоматизованої системи оптимізації споживання енергії для розумного будинку : кваліфікаційна магістерська робота : спец. 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» / наук. кер. І. А. Березюк ; Центральноукраїн. нац. техн. ун-т. – Кропивницький : ЦНТУ, 2025. – 86 с, https://dspace.kntu.kr.ua/handle/123456789/19123.

18. Sousa, B., & Cruz, T. (2026). Digital twins in energy-related home devices. In Home Digital Twins (pp. 295-311). Elsevier, DOI 10.1016/B978-0-443-44084-7.00007-X.

19. Ayyalusamy, V., Sivaneasan, B., Kandasamy, N. K., Xiao, J. F., & Chandra, A. (2022, November). Hybrid digital twin architecture for power system cyber security analysis. In 2022 IEEE PES Innovative Smart Grid Technologies-Asia (ISGT Asia) (pp. 270-274). IEEE, DOI: 10.1109/ISGTAsia54193.2022.10003563.

20. Perepelitsyn, A., Vdovichenko, O., & Mikhalevskyi, V. (2023). Service for communication of devices with internet access: analysis of technologies and method of creation. Radioelectronic and Computer Systems, (4), 196-208, DOI: https://doi.org/10.32620/reks.2023.4.14.

21. Khalid, I., Maglogiannis, V., Naudts, D., Shahid, A., & Moerman, I. (2024). Optimizing hybrid v2x communication: An intelligent technology selection algorithm using 5g, c-v2x PC5 and dsrc. Future Internet, 16(4), 107, https://doi.org/10.3390/fi16040107.

22. Reyna, J. M., Graffigna, J. P., Holeywell, P. M., & Garcia, A. R. (2022, September). Comparative analysis in medical image communication services. In

Congreso Argentino de Bioingeniería (pp. 372-378). Cham: Springer Nature Switzerland, https://doi.org/10.1007/978-3-032-06401-1_129.

23. van der Heide, G. (2024). Fixed-Point (Value) Recursion with Algebraic Effects and Handlers in Haskell (Doctoral dissertation, Delft University of Technology).

24. ДСТУ 3400:2015. Охорона праці. Безпека при роботі з електричними установками. – К.: Національний стандарт України, 2015. – 33 с.

25. Ціль 9. Промисловість, інновації та інфраструктура // Дія, 2025. URL: https://business.diia.gov.ua/entrepreneurhandbook/item/cil_9_promislovist_innovaciyi_ta_infrastruktura (дата звернення: 15.06.2026).