Міністерство освіти і науки України
Український державний університет науки і технологій

Навчально-науковий інститут
«Український державний хіміко-технологічний університет»
(назва навчально-наукового інституту)

Факультет комп'ютерних наук та інженерії
(повна назва факультету)

Кафедра комп'ютерно-інтегрованих технологій та робототехніки
(повна назва кафедри)

Пояснювальна записка

до дипломної роботи
бакалавра
(освітній рівень)

на тему: Розробка мобільного додатка для дистанційного керування елементами системи «розумний будинок»

Виконав: студент 4 курсу, групи 4-КІ-22
спеціальності
123 Комп'ютерна інженерія
(код і назва спеціальності)

КУЗЬМОВ Р.С. ____________
(прізвище та ініціали) (підпис)

Керівник ХОРОШИЛОВ С.В. ____________
(прізвище та ініціали) (підпис)

Рецензент ДУБОВИК Т.М. ____________
(прізвище та ініціали) (підпис)

Дніпро – 2026 року

Український державний університет науки і технологій
(повне найменування вищого навчального закладу)

Навчально-науковий інститут
«Український державний хіміко-технологічний університет»
(назва навчально-наукового інституту)

Факультет, відділення комп'ютерних наук та інженерії
Кафедра комп'ютерно-інтегрованих технологій та робототехніки
Освітній рівень бакалавр
Спеціальність 123 Комп'ютерна інженерія
(код і назва)
Спеціалізація
(шифр і назва)
Освітня програма Комп'ютерна інженерія
(назва)

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри
ЛЕВЧУК І.Л., канд. техн. наук, доц.
"12" 06 2026 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

КУЗЬМОВ Ростислав Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка мобільного додатка для дистанційного керування елементами системи «розумний будинок»
керівник роботи ХОРОШИЛОВ Сергій Вікторович, докт. техн. наук, проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від 02.03.2026 р. №77-к
2. Строк подання студентом роботи 28.05.2026 р.
3. Вихідні дані до роботи матеріали виробничої практики, дані технічної літератури
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Загальна частина. Спеціальна частина. Розрахункова частина. Охорона праці та безпека в надзвичайних ситуаціях. Організаційно-економічний розділ
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Презентація до дипломної роботи

6. Консультанти розділів роботи

Розділ	Ініціали, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці та безпека в надзвичайних ситуаціях	С.В. ХОРОШИЛОВ, докт. техн. наук, професор	14.02.2026	10.05.2026
Організаційно-економічний розділ	С.В. ХОРОШИЛОВ, докт. техн. наук, професор	10.05.2026	10.05.2026
Консультант	Г.М. ГУЗЬ, асистент	14.02.2026	29.05.2026

7. Дата видачі завдання «14» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Бібліографічний огляд за темою	лютий	виконано
2	Робота над загальною частиною	лютий	виконано
3	Робота над спеціальною частиною	березень	виконано
4	Робота над розрахунковою частиною	квітень	
5	Робота над розділом «Охорона праці та безпека в надзвичайних ситуаціях»	травень	виконано
6	Робота над розділом «Організаційно-економічний розділ»	травень	виконано
7	Розробка презентації	травень	виконано
8	Перевірка роботи на плагіат	28.05.2026	виконано

Студент ____________ **Р.С. КУЗЬМОВ**
(підпис) (ініціали, прізвище)

Керівник роботи ____________ **С.В. ХОРОШИЛОВ**
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Дипломна робота містить: 77 сторінок, 15 рисунків, 2 таблиці, 20 джерел, 9 додатків.

Об’єкт дослідження – системи автоматизації, Інтернет речей (IoT) та протоколи передачі даних у реальному часі.

Мета роботи – спроєктувати та розробити мобільний додаток для дистанційного керування елементами системи "розумний будинок" (на прикладі автоматизованої міні-ферми).

У загальній частині наведено теоретичні відомості про платформи автоматизації, обґрунтовано вибір локального сервера Home Assistant та доведено перевагу протоколу WebSocket над традиційними REST API.

У спеціальній частині детально описано архітектуру трирівневої системи та розроблено структуру компонентів на базі фронтенд-фреймворку Angular та бібліотеки Ionic.

Розрахункова частина присвячена розробці програмного забезпечення: модуль безпечного з'єднання (Long-Lived Access Tokens), реактивний інтерфейс (Angular Signals) та систему планування розкладу поливу.

У розділі охорони праці та безпеки розглянуто вимоги до безпечної організації робочого процесу при розробці та тестуванні програмного забезпечення, а також основні заходи для запобігання надзвичайним ситуаціям.

Організаційно-економічний розділ включає аналіз ресурсів, витрат часу та коштів, необхідних для реалізації проекту, а також оцінку ефективності використаних методів і технологій.

Ключові слова: РОЗУМНИЙ БУДИНОК, ІНТЕРНЕТ РЕЧЕЙ (IOT), HOME ASSISTANT, WEBSOCKET, ANGULAR, IONIC, CAPACITOR, ESP32.

ЗМІСТ

ВСТУП 7

1 ЗАГАЛЬНА ЧАСТИНА 9

1.1 Огляд існуючих систем розумного будинку 9

1.2 Аналіз протоколів взаємодії IoT-пристроїв та клієнтських додатків 11

1.3 Обґрунтування вибору стека технологій 13

2 СПЕЦІАЛЬНА ЧАСТИНА 18

2.1 Загальна архітектура взаємодії системи 18

2.2 Розробка структури мобільного додатка 22

2.3 Проєктування інтерфейсу користувача 28

2.4 Питання безпеки 32

3 РОЗРАХУНКОВА ЧАСТИНА 35

3.1 Вибір та налаштування інструментальних засобів розробки 35

3.2 Програмна реалізація модулів зв'язку та управління станом 38

3.3 Реалізація клієнтського інтерфейсу 40

3.4 Збірка додатка та тестування на мобільному пристрої 48

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 52

4.1 Характеристика умов праці інженера-програміста 52

4.2 Вимоги до санітарно-гігієнічних умов у приміщенні 54

4.3 Ергономіка робочого місця 55

4.4 Електробезпека під час розробки та тестування системи 56

4.5 Пожежна безпека 57

5 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНИЙ РОЗДІЛ 58

5.1 Техніко-економічне обґрунтування створення програмного продукту 58

5.2 Розрахунок трудомісткості розробки 58

5.3 Розрахунок витрат на заробітну плату та соціальні відрахування 59

5.4 Витрати на електроенергію та амортизацію обладнання 60

5.5 Накладні витрати та загальна вартість програмного продукту 61

ВИСНОВКИ 63
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 65
ДОДАТОК А Метод підключення та підписки на події реального часу 67
ДОДАТОК Б Методи керування та запиту історії через сокет 68
ДОДАТОК В Фрагмент коду головної сторінки та реактивних станів 69
ДОДАТОК Г Лістинг компонента модального вікна історичних графіків 70
ДОДАТОК Д Текст програми сторінки конфігурації 71
ДОДАТОК Е Лістинг сервісу локального збереження налаштувань 72
ДОДАТОК Ж Код структури розмітки головного екрану 74
ДОДАТОК З Конфігурації мобільного дистрибутиву 76
ДОДАТОК І Каскадна таблиця стилів графіків 77

ВСТУП

Об’єктом діяльності фахівця з комп'ютерної інженерії є апаратно-програмні комплекси, локальні та глобальні комп'ютерні мережі, системи збору, обробки, передачі та захисту інформації, а також керуючі системи апаратних засобів та Інтернету речей (IoT).

Розробка мобільного клієнтського програмного забезпечення для взаємодії з апаратно-програмною платформою «розумного будинку» (на базі мікроконтролерів ESP32, прошивок ESPHome та сервера Home Assistant) є безпосередньою професійною задачею комп'ютерного інженера. Цей процес вимагає комплексних знань з проєктування архітектури ПЗ, організації стійкої мережевої взаємодії, реактивної обробки даних реального часу та розробки кросплатформних людино-машинних інтерфейсів.

На сьогодні системи домашньої автоматизації класу «розумний будинок» набули масового поширення як на комерційному, так і на аматорському рівнях. Найпопулярніші платформи автоматизації, зокрема Home Assistant, пропонують потужні та універсальні веб-інтерфейси. Проте головним недоліком таких комплексних рішень є надмірна універсальність та перевантаженість для кінцевого користувача із специфічними задачами.

Існуючі базові клієнтські додатки не завжди забезпечують інтуїтивно зрозуміле та фокусне керування вузькоспеціалізованими екосистемами, такими як автоматизовані міні-ферми. Це створює об'єктивну потребу в розробці спеціалізованих клієнтських додатків, які агрегують виключно цільові метрики (мікроклімат, відеомоніторинг) та інструменти керування в єдиному оптимізованому та ергономічному мобільному інтерфейсі.

Метою даного дипломного проєкту є розробка кросплатформного мобільного додатка для дистанційного моніторингу та керування цільовими апаратними компонентами системи «розумний будинок» (на прикладі автоматизованої міні-ферми).

Програмний продукт повинен забезпечувати надійну, безпечну та швидку взаємодію з ядром системи через протокол WebSocket, виступаючи оптимізованим фронтенд-рішенням для існуючої інфраструктури IoT.

Стрімкий розвиток ринку IoT та концепцій «розумного сільського господарства» (Smart Agriculture, Urban Farming) зумовлює зростаючий попит на доступні й надійні системи локальної автоматизації. Вирощування рослин або підтримання біосистем у закритих приміщеннях з жорстко контрольованим мікрокліматом потребує безперервного безвідмовного моніторингу та миттєвого реагування на критичні зміни.

Розробка власного мобільного клієнта не лише відповідає глобальним тенденціям цифровізації побутової рутини, але й пропонує масштабоване рішення. Архітектура розробленого додатка у перспективі може бути легко масштабована чи адаптована під моніторинг будь-якого іншого вузла екосистеми збудованої на базі ESPHome та Home Assistant.

Для досягнення поставленої мети вирішувались наступні завдання:

- проєктування архітектури клієнтської частини з використанням сучасного технологічного стека (Angular 15+, Ionic Framework 7+, Capacitor);

- реалізація механізму автентифікації на основі довготривалих токенів (Long-Lived Access Tokens) та налаштування захищеного повнодуплексного WebSocket-з'єднання з сервером Home Assistant;

- розробка програмного модуля збору та UI-візуалізації телеметричних даних реального часу, які надходять з датчиків (температура, вологість);

- розробка та інтеграція інтерфейсу керування групою виконавчих пристроїв (чотирма незалежними реле для освітлення, водяної помпи, витяжної вентиляції та аерації).

- реалізація компонента відтворення безперервного потокового відео (MJPEG-потіку з модуля ESP32-CAM) для візуального контролю об'єкта.

- оптимізація реактивного стану мобільного додатка за допомогою патерну Signals для гарантування високої продуктивності інтерфейсу кінцевого пристрою.

1 ЗАГАЛЬНА ЧАСТИНА

1.1 Огляд існуючих систем розумного будинку

Стрімкий розвиток концепції Інтернету речей та здешевлення мікропроцесорної техніки стали каталізаторами масового впровадження систем розумного будинку та систем локальної автоматизації.

В контексті розробки системи для дистанційного керування елементами автоматизованої міні-ферми (кліматичний контроль, полив, освітлення) ключовим етапом є вибір базової програмної платформи, яка виступатиме комунікаційним ядром між сенсорами, виконавчими модулями та користувачем.

Існуючі рішення прийнято поділяти на пропрієтарні системи від комерційних вендорів та відкриті платформи (Open-Source) [1].

Аналіз пропрієтарних систем (Xiaomi, Tuya, Apple HomeKit)

Пропрієтарні системи (створені компаніями Xiaomi, Tuya Smart, Apple, Google) переважають на споживчому мас-маркеті. Основними їх перевагами є:

- простота розгортання (Plug-and-Play). Підключення нових пристроїв відбувається за декілька кліків через екосистемний мобільний додаток.

- уніфікований інтерфейс користувача. Додатки цих вендорів мають інтуїтивно зрозумілий дизайн.

- підтримка голосових асистентів. Глибока інтеграція зі смарт-колонками без необхідності додаткового налаштування.

Незважаючи на значні переваги для пересічного користувача, використання подібних платформ для інженерних спеціалізованих проектів (як-от автоматизована міні-ферма) обмежується низкою критичних недоліків:

- залежність від хмарних серверів (Cloud Dependency). Більшість процесів обробки даних маршрутизуються через сторонні сервери (найчастіше розташовані за кордоном). Це спричиняє затримки у відгуку системи, а у разі зникнення інтернет-з'єднання локальна автоматизація повністю припиняє функціонування.

- закритість протоколів обміну. Виробники навмисно ускладнюють або унеможливлюють інтеграцію саморобних IoT-вузлів (наприклад, власних контролерів на базі ESP32).

- обмеженість логіки. Алгоритми автоматизацій часто базуються на жорстко заданих шаблонах ("якщо А, то Б"), що не дозволяє створювати складні математичні або таймерні скрипти поливу чи аерації для рослин.

Для вирішення завдань дипломної роботи було проаналізовано відкриті системи автоматизації: OpenHAB, Node-RED та Home Assistant. Як базовий сервер системи було обрано Home Assistant (HA) завдяки його безперечним перевагам.

Повністю локальна обробка. Система розгортається на локальному шлюзі (наприклад, Raspberry Pi або локальному сервері). Дані з датчиків не покидають межі локальної домашньої мережі, гарантуючи конфіденційність, безперебійність під час відсутності інтернету та мінімальну (мілісекундну) затримку відгуку реле.

Незалежність від виробників (Vendor Neutrality). Сервер має найбільшу у світі базу підтримуваних інтеграцій у сфері розумного будинку. Він дозволяє об'єднати в одну екосистему промислові датчики Zigbee, Wi-Fi реле від Sonoff та саморобні плати.

Підтримка технології ESPHome. Інтеграція з ESPHome є ключовим фактором для проєкту міні-ферми. ESPHome дозволяє прошивати мікроконтролери ESP32/ESP8266 конфігураційними YAML-файлами без необхідності писати низькорівневий C++ код. Таким чином, датчики DHT22, модулі ESP32-CAM та блок реле блискавично інтегруються безпосередньо в екосистему Home Assistant.

Хоча Home Assistant пропонує потужний веб-інтерфейс (Lovelace Dashboard) для керування сутностями, він розроблявся як універсальний інструмент системного адміністратора будинку. Для вузькоспеціалізованої задачі моніторингу міні-ферми він виявився перевантаженим. В інтерфейсі Lovelace важко гарантувати суворий фокус користувача на визначених процесах [2].

Звичайний оператор установки (або власник міні-ферми) не потребує повного доступу до налаштувань ядра HA, конфігурацій мережі чи логів сервера. Йому необхідний відокремлений, ергономічний клієнт із виведеними виключно цільовими елементами: показниками мікроклімату, віджетом відеопотоку та елементами керування поливом/освітленням.

З огляду на це, було прийнято рішення розробити власний спеціалізований мобільний клієнт, що комунікує з сервером через API.

1.2 Аналіз протоколів взаємодії IoT-пристроїв та клієнтських додатків

При проєктуванні системи дистанційного керування критичної ваги набуває вибір архітектури передавання даних між сервером Home Assistant та мобільним клієнтом. Основна дилема полягала у виборі між традиційним підходом клієнт-сервер (REST API) та повнодуплекснимі з'єднанням (WebSocket).

REST API базується на парадигмі "запит-відповідь". Для отримання оновлень інтерфейсу, клієнтський додаток змушений реалізовувати алгоритм періодичного опитування сервера (Polling).

Додаток кожні кілька секунд відправляє HTTP-запит GET до сервера з питанням "Чи змінилася температура?". Цей підхід має низку фундаментальних недоліків для IoT-клієнтів:

- високе навантаження на мережу. Десятки тисяч зайвих запитів за добу, навіть якщо фізичний стан датчиків на фермі не змінився.

- штучна затримка. Якщо частота опитування дорівнює 5 секундам, тоді між моментом фактичного включення реле поливу на платі мікроконтролера і його відображенням на смартфоні може пройти до 5 секунд, що порушує концепцію "миттєвої чуйності" (real-time UX).

- високе споживання батареї мобільного пристрою через необхідність постійно тримати Wi-Fi передавач смартфона в активному стані для трансляції HTTP-запитів.

На противагу традиційному REST, протокол WebSocket забезпечує повнодуплексне (двонаправлене) з'єднання поверх єдиного TCP-сокета [3]. Дослідження показало, що для задачі управління розумною фермою використання офіційного WebSocket API Home Assistant є єдиним правильним інженерним рішенням, оскільки:

Валідація з'єднання та відкриття каналу (handshake) відбувається лише один раз.

Модель Publish-Subscribe (Push) - це паттерн обміну повідомленнями, де відправник (Publisher) не надсилає повідомлення конкретному отримувачу, а публікує його в «тему» (Topic) на посередника (Broker).

Мобільний додаток відправляє запит на підписку (на події або зміну сутностей), і сервер самостійно надсилає новий JSON-пакет на смартфон тільки в момент фізичної зміни стану реле або метрики DHT22. Це зводить витрати мережевого трафіку до мінімуму і значно економить заряд батареї телефону.

Швидкість реакції. Інтерфейс клієнта та стан виконавчих механізмів синхронізуються за мілісекунди. При натисканні на тумблер освітлення в додатку, команда миттєво долітає через сокет до сервера, далі на вузол ESP32 і відразу повертається статус "on", що гарантує безшовний досвід керування.
Для забезпечення інформаційної безпеки цього тунелю зв'язку було обрано механізм авторизації за допомогою Long-Lived Access Tokens (LLAT).

Згенерований на сервері токен передається на етапі аутентифікації WebSocket. Такий механізм відкидає потребу щоразу вводити пароль користувача та дозволяє зручно й безпечно зберігати токен у зашифрованому локальному сховищі смартфона (через плагіни Capacitor).

1.3 Обґрунтування вибору стека технологій

Створення клієнтської частини охоплювало вибір між нативною, кросплатформною або гібридною розробкою програмного забезпечення.

Традиційна нативна розробка передбачає використання мов програмування та SDK, які розроблені безпосередньо компаніями-власниками платформ: Swift/Objective-C та Xcode для iOS; Kotlin/Java та Android Studio для Android.

Технологічні особливості:

1. Прямий доступ до API: нативні додатки взаємодіють з операційною системою без проміжних шарів (bridge) чи контейнерів. Це забезпечує найшвидший доступ до Bluetooth, Wi-Fi стека, GPU та спеціалізованих сенсорів смартфона.

2. Управління пам'яттю: розробник має тонкий контроль над ресурсами пристрою. У випадку використання великих потоків даних (наприклад, декодування MJPEG-відео з ESP32-CAM у реальному часі) нативний код забезпечує мінімальне навантаження на CPU.

3. Інтерфейс користувача: використовуються нативні компоненти (UIKIt/SwiftUI для iOS та XML/Jetpack Compose для Android).

Основною перевагою даного підходу є максимальна інтеграція з апаратними компонентами платформ і безкомпромісна швидкодія [1].

Проте, головними і критичними недоліками у контексті дипломної роботи є:

- дублювання бізнес-логіки: оскільки система моніторингу міні-ферми базується на складній обробці WebSocket-повідомлень та станів, при нативному підході довелося б реалізовувати цю логіку двічі, що підвищує ймовірність виникнення помилок синхронізації;

- відсутність уніфікованого екосистемного коду: більшість інструментів для роботи з Home Assistant та ESPHome мають розвинені JS/TS бібліотеки. Нативна реалізація потребувала б написання власних парсерів для протоколів HA з нуля;

- Висока вартість підтримки: будь-яка зміна в структурі даних (наприклад, додавання нового датчика аерації) потребувала б оновлення та перекомпіляції двох окремих проєктів.

Для детального аналізу кросплатформних рушіїв у межах дипломної роботи важливо розглянути їхню внутрішню архітектуру. Це дозволить обґрунтувати, як саме кожен із них обробляє дані та взаємодіє з операційною системою пристрою.

Альтернативою виступають кросплатформні рушії, такі як Flutter (Dart) або React Native (JavaScript). Вони генерують власні елементи інтерфейсу та збираються у високопродуктивні .apk файли.

Вони поділяються на три основні технологічні групи:

1. Рушії з власним графічним конвеєром (Flutter).

Flutter не використовує стандартні компоненти операційної системи (на кшталт кнопок Android або iOS). Замість цього він має власний графічний рушій Impeller, який малює кожен піксель інтерфейсу безпосередньо через графічне ядро (GPU).

Архітектура побудована на мові Dart. Код компілюється в машинний код ARM/x64, що забезпечує продуктивність, близьку до нативної.

Перевагами є абсолютна візуальна ідентичність на всіх пристроях та плавна анімація (120 FPS). Недоліками: великий розмір фінального файлу (через включення графічного ядра в додаток) та складність виклику нативних

функцій ОС, що потребує написання "платформних каналів" (MethodChannels).

2. Рушії на основі нативних містків (React Native).

React Native використовує JavaScript для опису бізнес-логіки, але для рендерингу викликає реальні нативні компоненти iOS та Android.

Архітектура базується на архітектурі Fabric. Це дозволяє JavaScript-коду синхронно взаємодіяти з нативним UI, уникаючи затримок "мосту" (bridge), які були проблемою у минулих версіях.

Переваги: користувач отримує нативний досвід (Native Look and Feel), оскільки компоненти виглядають і поводяться так, як задумав виробник ОС.

Недоліки: залежність від оновлень системних бібліотек, що може призводити до помилок у відображенні інтерфейсу на нових версіях ОС.

3. Модульні кросплатформні рішення (Kotlin Multiplatform - KMP).

Це найбільш сучасний підхід, де спільним є лише програмний код логіки, а інтерфейс може бути як спільним, так і повністю нативним.

Архітектура - код на Kotlin транслюється в нативні двійкові файли для кожної платформи.

Переваги: максимально можлива швидкість виконання та повний доступ до системних API без жодних прошарків. Недоліки: найбільша складність розробки, оскільки вимагає знання особливостей розробки під кожну цільову ОС.

Оскільки, у взаємодії з Home Assistant лежить логіка мережевої передачі повідомлень WebSocket та класичний фронтенд веб-архітектури, найбільшу доцільність показав набір гібридних веб-технологій [2].

Гібридний підхід передбачає розробку єдиного кросплатформного ядра (HTML/JS/CSS), яке силами компілятора перетворюється на мобільний додаток, придатний для публікації.

Гібридний додаток працює за принципом "Web-to-Native Bridge". Він не компілюється в машинний код (як нативний) і не малює інтерфейс пікселями

(як Flutter). Замість цього він використовує нативний контейнер для відображення вебконтенту.

Основою гібридного додатка є WebView (наприклад, WKWebView на iOS та Chrome WebView на Android). Це ізольоване вікно браузера без адресної стрічки, яке займає 100% екрана додатка.

У 2026 році WebView використовує апаратне прискорення GPU, що дозволяє плавно відображати складні графіки телеметрії та відеопотоки MJPEG.

Capacitor виступає посередником між вебкодом у WebView та апаратною частиною смартфона.

Runtime-зв'язок: коли Angular-додаток хоче отримати доступ, наприклад, до системних сповіщень або захищеного сховища (для токенів HA), він викликає JavaScript-функцію Capacitor.

Нативна реалізація: Capacitor перекладає цей виклик на мову конкретної платформи (Swift для iOS або Kotlin для Android) та повертає результат назад у вебсередовище.

Технологічний стек був обраний на основі наступних критеріїв:

Angular (Фреймворк бізнес-логіки). Було обрано найновішу архітектуру Angular (включаючи Standalone Components), яка працює на базі суворо типізованої мови TypeScript [4].

TypeScript запобігає масиву помилок ще на етапі компіляції (завдяки жорстким інтерфейсам повідомлень, які приходять від сервера). Потужний інструмент Angular Signals був обраний для побудови реактивного стану (Reactivity). При отриманні нових телеметричних даних від сервера через WebSocket, сигнали миттєво перераховуються і точково оновлюють лише ті елементи DOM-дерева, значення яких змінилося, забезпечуючи високу швидкість роботи UI без перевантаження процесора [5].

Ionic Framework (Бібліотека UI). Для усунення потреби створювати дизайн-систему кнопок з нуля (та підтримувати гайдлайни мобільних розробників), було інтегровано Ionic Framework 7+. Він надає набір

оптимізованих, протестованих і повністю стилізованих "з коробки" UI-компонентів. Наприклад, компонент <ion-toggle> автоматично бере на себе анімацію та поведінку тумблера iOS під час запуску на пристроях Apple, і стає класичним material-перемикачем під час запуску на середовищі Android [6].

Capacitor (Інструмент кінцевої збірки). Розроблений авторами Ionic сучасний runtime-модуль Capacitor працює як міст між бізнес-логікою веб-додатка та нативними API операційної системи смартфона [7]. Він забезпечує трансляцію Angular-коду у внутрішній веб-переглядач (WebView) апарата і збирає готовий до встановлення дистрибутив (файл .apk для Android). Додатковою причиною обрання Capacitor став плагін Capacitor Preferences, який дозволив реалізувати безпечне довготривале збереження токена (LLAT) і URL адреси Home Assistant безпосередньо в конфігураційній пам'яті смартфону, без загрози втрати даних під час перезавантаження чи оновлення.

У підсумку, проаналізована комбінація програмних рішень - від сервера з локальною обробкою автоматизацій до реактивного мобільного веб-додатку - формує найбільш гнучку, масштабовану та надійну архітектуру сучасного вузла IoT.

Даний стек не лише гарантує успішне вирішення заявленої мети проєкту (моніторингу міні-ферми), але й відповідає актуальним вимогам до розробки корпоративних продуктів у сфері комп'ютерної інженерії.

2 СПЕЦІАЛЬНА ЧАСТИНА

2.1 Загальна архітектура взаємодії системи

Створення комплексу управління "розумним будинком" (на прикладі автоматизованої міні-ферми) потребувало побудови надійної трирівневої інформаційної архітектури.

Проектування архітектури базується на чіткому розділенні обов'язків (Separation of Concerns) між апаратним забезпеченням, сервером прийняття рішень та клієнтським інтерфейсом. Запропонована архітектура складається з наступних рівнів [8]:

Апаратний рівень (Hardware Layer). Включає мікроконтролери ESP32, зокрема модуль ESP32-CAM для відеомоніторингу, блок твердотільних реле (для керування освітленням, насосом поливу та вентиляцією) та прецизійні сенсори мікроклімату DHT22. Усі виконавчі пристрої працюють під управлінням мікропрограмного забезпечення (прошивки) ESPHome. Це забезпечує нативну інтеграцію вузлів у локальну мережу через протокол Wi-Fi (або Native API) без необхідності застосування протоколу MQTT.

Серверний рівень (Server Layer). Центральним вузлом (брокером) архітектури виступає платформа Home Assistant. Сервер акумулює телеметрію з ESP32, оновлює стани логічних сутностей (Entities), логує історію показників (Recorder) та містить у собі рушій автоматизацій (Automation Engine). Він виконує роль єдиної "точки правди" для всієї системи.

Клієнтський рівень (Client Layer). Власний мобільний додаток, який виконує роль мобільного пульта керування системою (фронтенд). Клієнтський рівень позбавлений складної бізнес-логіки автоматизацій; його завдання — динамічно відображати стани сенсорів у реальному часі та надсилати команди (Service Calls) до серверного рівня.

На рисунку 2.1 зображено загальну структурну схему розробленого комплексу. Ця схема демонструє ієрархію взаємодії від мобільного додатка до кінцевих виконавчих пристроїв.

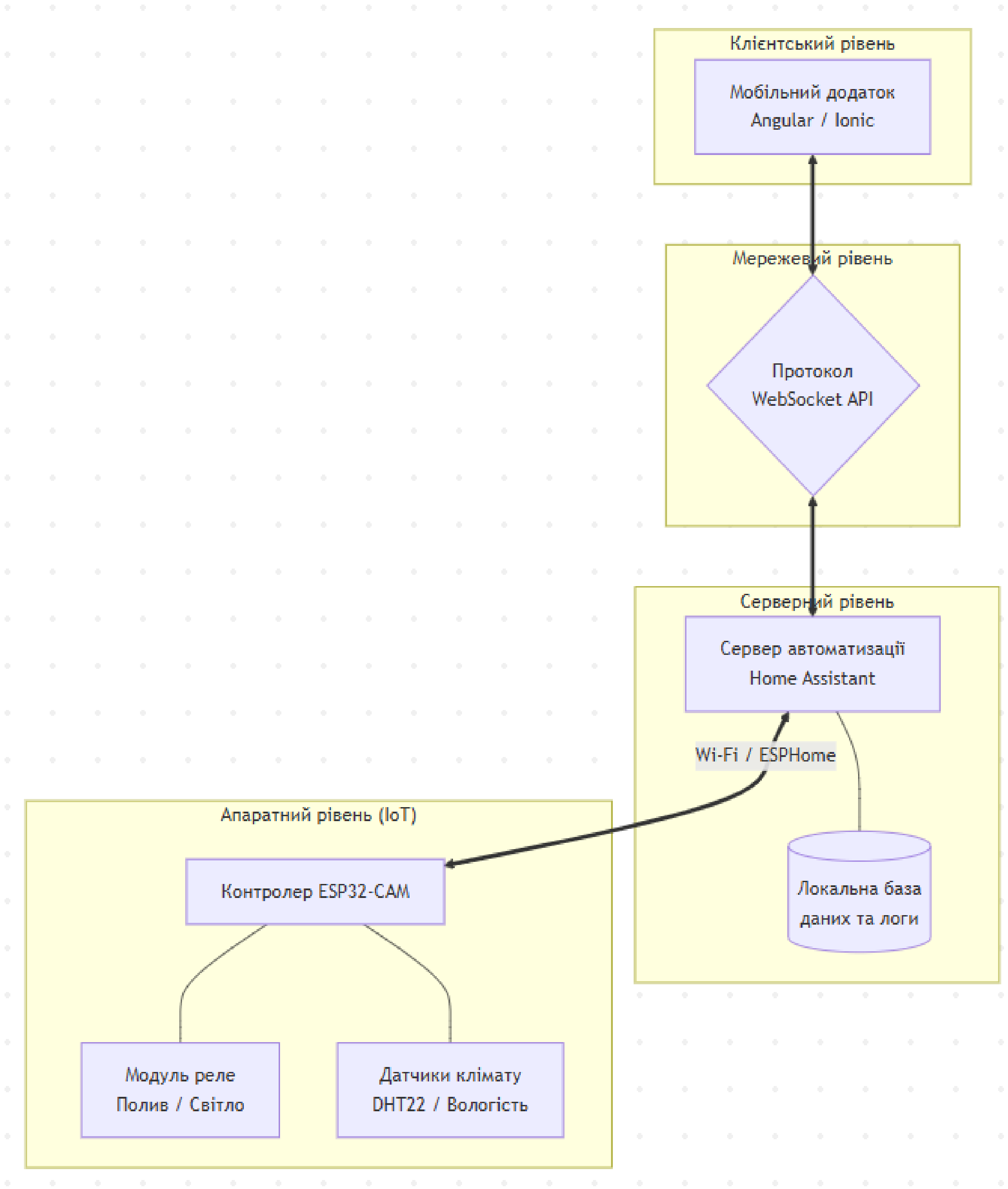

Рисунок 2.1 – Структурна схема комплексу

Окремої уваги заслуговує спроектований механізм роботи розкладу поливу міні-ферми. Для збереження стійкості системи клієнтський додаток не редагує самі скрипти автоматизацій на сервері безпосередньо. Натомість, було реалізовано патерн "віртуальних тригерів".

У Home Assistant створюються віртуальні сутності типу input_datetime (Ранковий, Денний та Вечірній таймери). Користувач через мобільний додаток лише змінює значення часу у цих сутностях.

Серверна автоматизація працює у фоні, очікуючи моменту, коли системний час співпаде зі значенням input_datetime, після чого самостійно подає сигнал на вмикання реле поливу.

Такий підхід гарантує, що полив відбудеться навіть за умови відсутності інтернет-зв'язку між смартфоном та сервером.

Алгоритм взаємодії при налаштуванні розкладу наведено на рисунку 2.2.

На основі представленої діаграми послідовності, алгоритм взаємодії компонентів системи при налаштуванні розкладу поливу можна розділити на три фази: ініціалізація змін, очікування події та виконання автоматизації.

1. Фаза ініціалізації та запису параметрів - ця фаза починається з дій користувача в мобільному додатку та завершується оновленням бази даних сервера.

Користувач через графічний інтерфейс додатка (UI) обирає новий час для активації системи поливу. Компонент Angular перехоплює подію зміни та формує запит до API. Використовується сервіс callService, який надсилає команду input_datetime.set_datetime. Це дозволяє передати точні параметри часу (години, хвилини) на сервер. Сервер отримує команду та оновлює стан відповідної сутності (helper). Після успішного запису Home Assistant надсилає зворотне підтвердження додатку, що сутність таймера оновлена.

2. Фаза моніторингу та режиму очікування - після встановлення часу система переходить у пасивний стан, що мінімізує навантаження на процесор. Home Assistant постійно порівнює поточний системний час із внутрішньою змінною, яку встановив користувач.

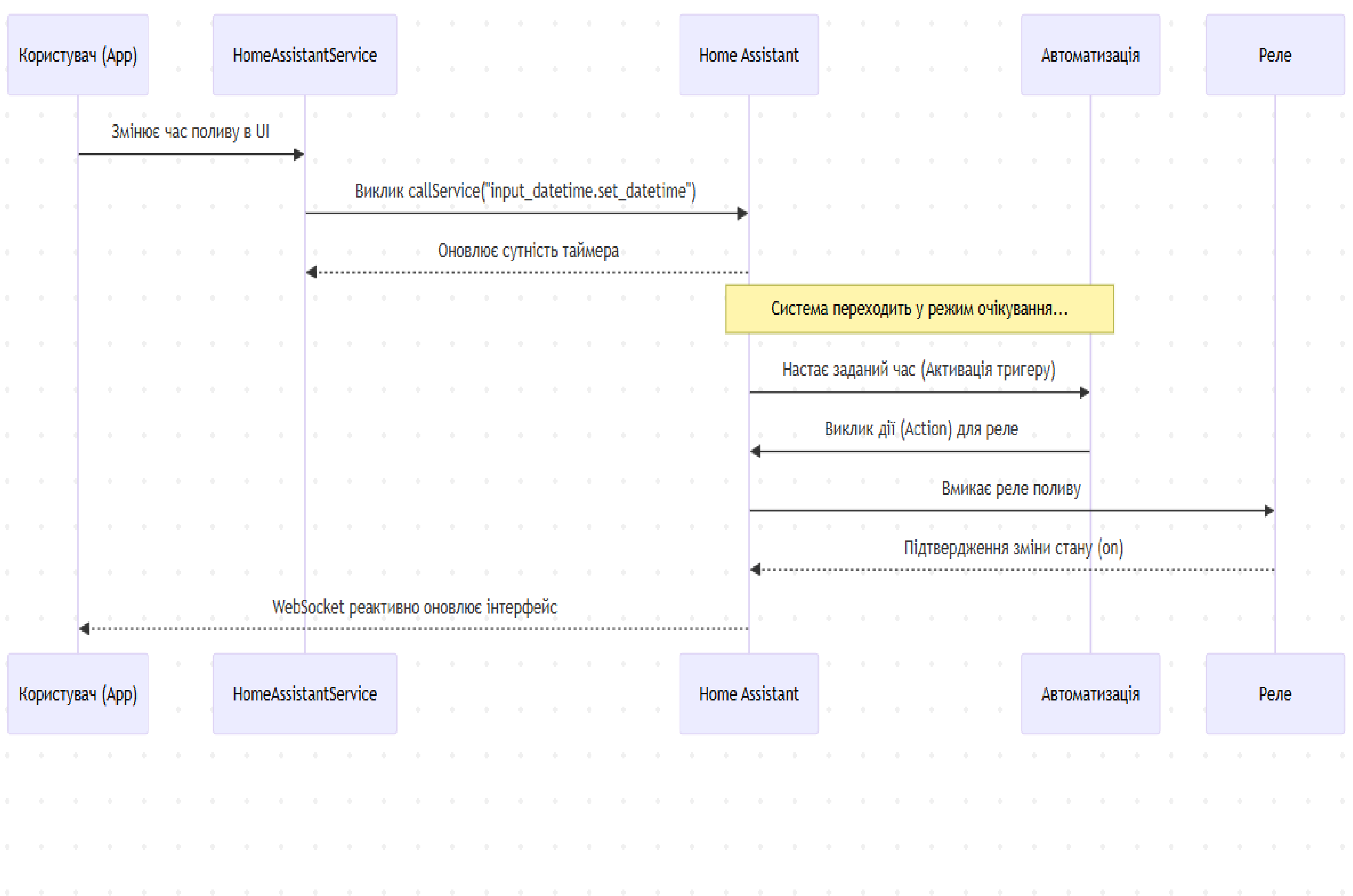

Рисунок 2.2 – Алгоритм взаємодії при налаштуванні розкладу

На цьому етапі мережева активність між додатком та сервером щодо цього завдання припиняється. Як тільки системний час збігається з установленим значенням, спрацьовує тригер, який запускає логічний блок автоматизації.

3. Фаза виконання та реактивного зворотного зв'язку - це завершальний етап, де віртуальна команда перетворюється на фізичну дію.

Внутрішній модуль автоматизації надсилає виклик дії (Action) до сутності, що відповідає за фізичне реле. Home Assistant надсилає команду на виконавчий пристрій (Реле). Реле замикає контакти, вмикаючи помпу для поливу. Виконавчий пристрій надсилає сигнал підтвердження (state: on) назад на сервер. Home Assistant через відкрите WebSocket-з'єднання миттєво транслює зміну стану реле в мобільний додаток. Завдяки використанню паттерну Signals в Angular, інтерфейс користувача оновлюється без перевантаження сторінки, відображаючи актуальний статус роботи системи в реальному часі.

Використання Home Assistant як посередника дозволяє відокремити інтерфейс керування від логіки виконання. Навіть якщо мобільний додаток буде закрито або смартфон розрядиться, автоматизація спрацює автономно, оскільки логіка "час -> дія" зберігається і виконується локально на сервері.

2.2 Розробка структури мобільного додатка

Мобільний додаток побудовано на базі сучасного фронтенд-фреймворку Angular із застосуванням патерну Standalone Components. Використання ізольованих компонентів (без прив'язки до глобальних NgModule) дозволяє зменшити зв'язність коду (loose coupling), покращує інкапсуляцію залежностей та значно прискорює "холодний" старт застосунку (Cold Start) на

мобільних пристроях. Логіка додатку була декомпонована на два ключові сервіси.

Модуль SettingsService відповідає за збереження параметрів підключення (URL, Token) та ідентифікаторів сутностей (Entity IDs), обраних користувачем. Замість зберігання даних у ненадійному сховищі браузера (LocalStorage), яке може бути очищене операційною системою при нестачі пам'яті, було використано нативний плагін Capacitor Preferences. Він забезпечує запис пар "ключ-значення" безпосередньо в системний реєстр налаштувань (SharedPreferences на Android або NSUserDefaults на iOS).

Сервіс HomeAssistantService є центральним ядром обміну даними. На етапі ініціалізації він встановлює повнодуплексне поєднання з сервером через офіційну бібліотеку home-assistant-js-websocket. Для динамічного налаштування сервіс отримує повний перелік усіх доступних пристроїв та делегує їх інтерфейсу. Для безперервного моніторингу використовується метод subscribeEntities, що формує підписку на будь-які зміни станів пристроїв [9].

Представлена діаграма класів (рис. 2.3) описує архітектуру клієнтської частини мобільного додатка на базі фреймворку Angular. Структура побудована за принципом розподілу обов'язків, де виділено рівні маршрутизації, відображення (сторінки), логіки (сервіси) та безпеки.

Розглянемо кожен компонент та їх взаємозв'язок:

1. Рівень маршрутизації та безпеки.

AppRoutes: центральний модуль навігації. Він визначає шляхи додатка:

/home: веде на головну панель моніторингу.

/settings: сторінка конфігурації підключення.

* (Redirect): автоматичне перенаправлення на головну сторінку за замовчуванням.

SettingsGuard: реалізує інтерфейс canActivate. Його завдання - перевірити наявність налаштувань (URL-адреси сервера та токена) через

SettingsService. Якщо дані відсутні, він захищає шлях /home і перенаправляє користувача на сторінку налаштувань.

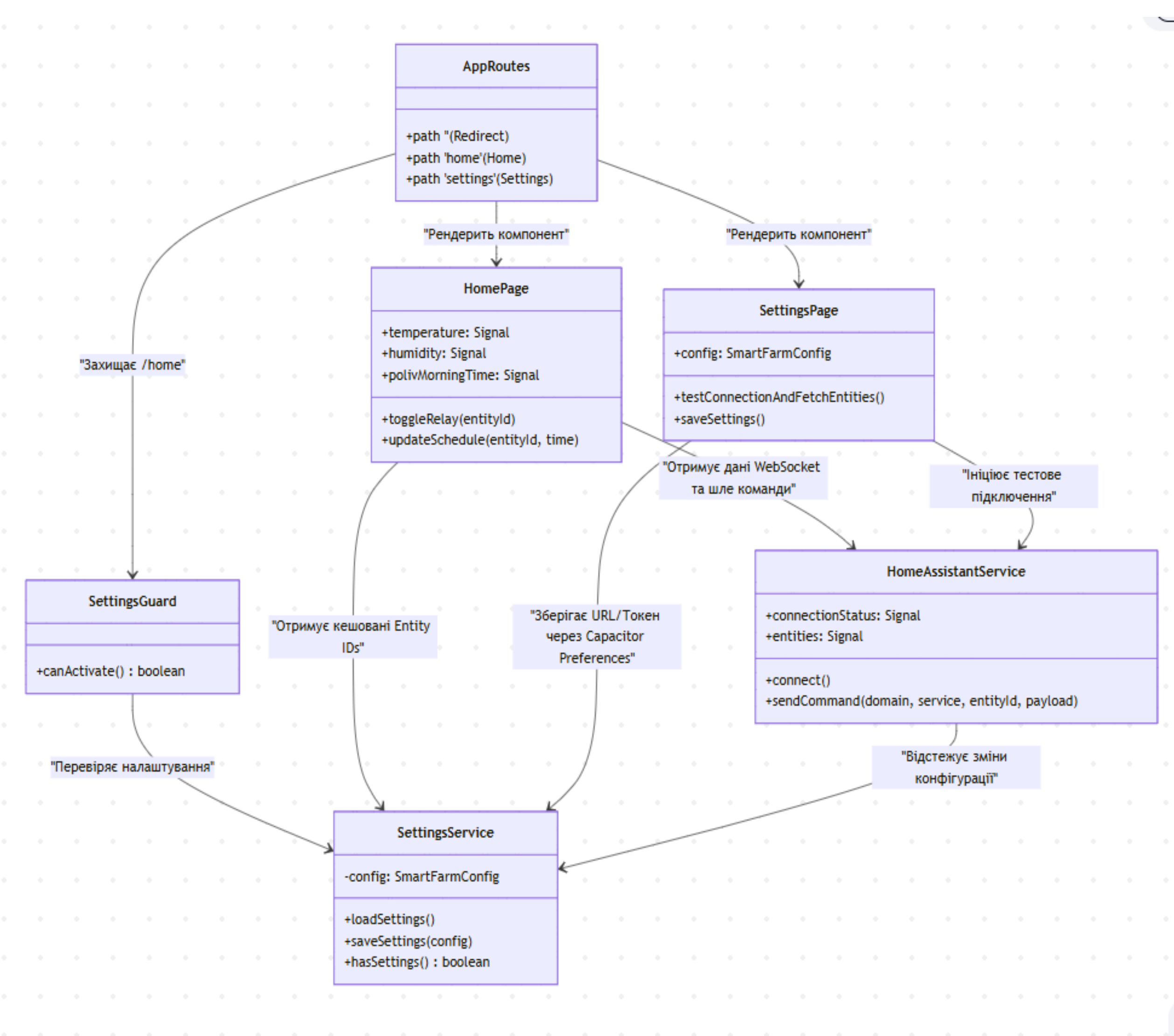

Рисунок 2.3 – Діаграма класів та сторінок мобільного додатка

2. Рівень сторінок

HomePage: Головний інтерфейс оператора. Використовує Angular Signals для збереження показників temperature, humidity та часу поливу. Це гарантує миттєве оновлення UI при зміні даних у WebSocket.

Методи: toggleRelay() для керування пристроями та updateSchedule() для зміни розкладу.

SettingsPage: Інтерфейс для керування параметрами підключення.

Дозволяє вводити дані хоста та токена, ініціює тестове підключення через HomeAssistantService та зберігає конфігурацію.

3. Сервісний рівень

Сервіси є синглтонами, що забезпечують збереження стану між зміною сторінок.

HomeAssistantService: ядро взаємодії з сервером Home Assistant.

WebSocket Connection: Керує активним з'єднанням, відстежує connectionStatus та агрегує всі отримані об'єкти (entities) у вигляді сигналів.

Методи: connect() для встановлення зв'язку та sendCommand() для надсилання команд на сервер (наприклад, перемикання реле).

SettingsService: Рівень персистентності (збереження даних).

Зберігання: використовує Capacitor Preferences для запису URL та токена в локальну пам'ять смартфона. Це гарантує, що дані не зникнуть після закриття додатка.

Методи: loadSettings(), saveSettings() та перевірка hasSettings(), яку використовує Guard.

Діаграма чітко показує використання Signal у сервісах та на сторінках. Це сучасний підхід Angular 15+, який дозволяє уникнути складних підписок RxJS і забезпечує високу продуктивність при великій кількості повідомлень з датчиків.

HomePage отримує дані від HomeAssistantService (через WebSocket) і одночасно звертається до SettingsService, щоб дізнатися ідентифікатори (Entity IDs) пристроїв, якими треба керувати.

HomeAssistantService відстежує зміни в SettingsService. Як тільки користувач зберігає нові параметри, сервіс зв'язку автоматично ініціює перепідключення з новими обліковими даними.

Використання Capacitor Preferences підкреслює гібридну природу додатка, де вебкод звертається до нативного сховища мобільного пристрою через API-міст.

Для ефективного рендерингу отриманої телеметрії було імплементовано механізм Angular Signals. На відміну від класичного механізму перевірки змін Zone.js (який глибоко сканує все дерево компонентів при будь-якій події), Signals забезпечують реактивне, точкове оновлення конкретного елемента інтерфейсу лише тоді, коли змінюється пов'язаний з ним сигнал. Завдяки цьому досягається плавність (60 FPS) інтерфейсу додатку.

Блок-схема алгоритму ініціалізації додатка та перевірки налаштувань наведена на рисунку 2.4.

1. Етап перевірки конфігурації (Автентифікація) - процес починається відразу після ініціалізації середовища Angular. Програма звертається до SettingsService, який через плагін Capacitor зчитує токен доступу та URL-адресу сервера Home Assistant із локального сховища пристрою.

Логічне розгалуження (Наявність даних):

Ні: якщо дані відсутні (перший запуск або скидання налаштувань), спрацьовує SettingsGuard, і користувач автоматично перенаправляється на сторінку налаштувань (/settings).

Так: якщо облікові дані знайдено, система переходить до встановлення зв'язку.

2. Етап встановлення мережевого з'єднання - задіюється HomeAssistantService для створення активного каналу передачі даних. Додаток ініціює повнодуплексне з'єднання. На відміну від звичайних HTTP-запитів, цей крок дозволяє створити постійний "тунель" для миттєвих повідомлень.

Перевірка успішності:

Помилка: якщо сервер недоступний або токен невалідний, інтерфейс відображає статус 'Disconnected' або 'Error', сигналізуючи користувачеві про необхідність перевірити мережу.

Успіх: при отриманні підтвердження від сервера система переходить до фінальної стадії.

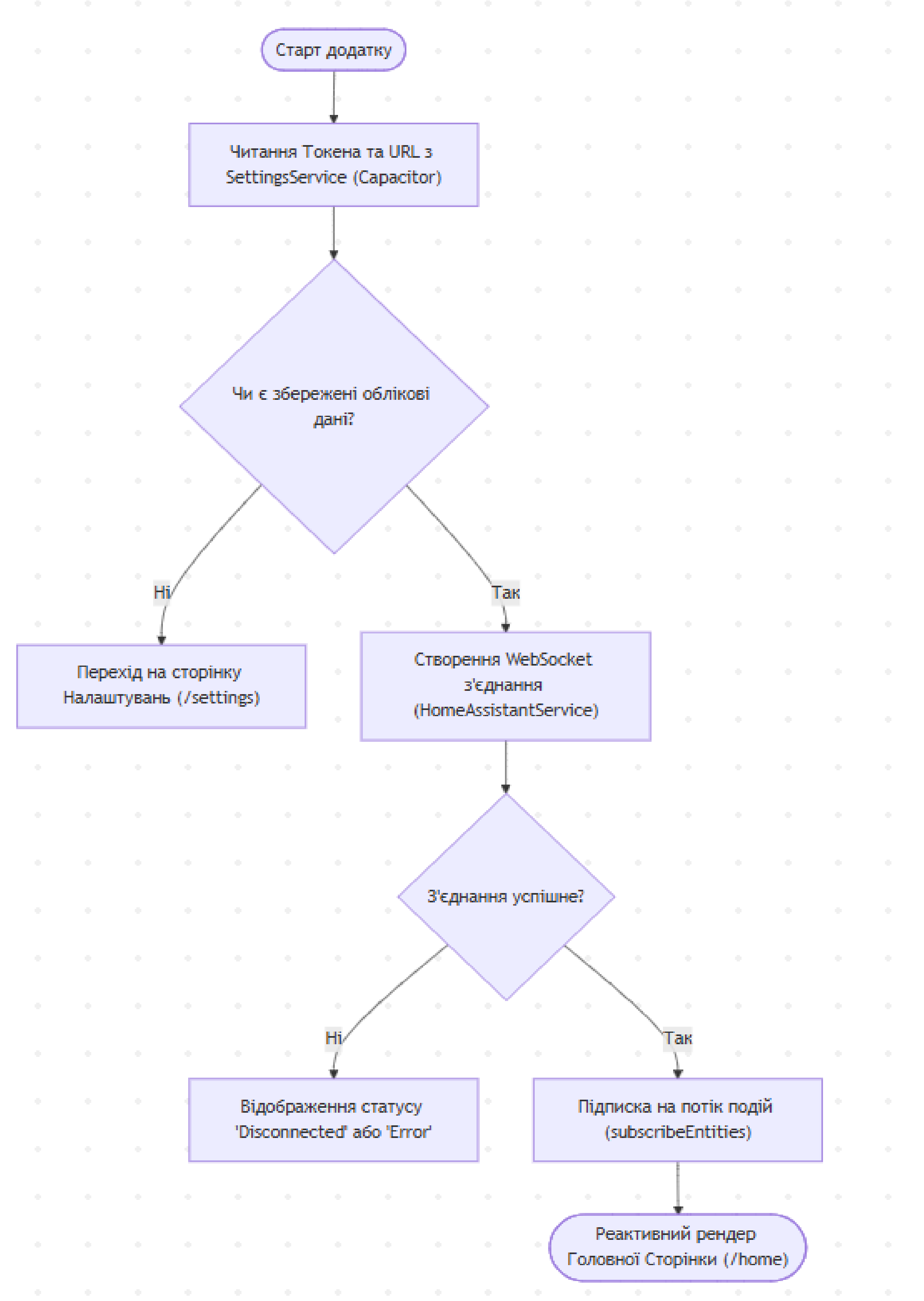

Рисунок 2.4 – Блок-схема алгоритму ініціалізації додатка та перевірки налаштувань

3. Етап синхронізації та рендерингу - це фаза переходу додатка в робочий режим моніторингу. Додаток надсилає серверу запит на отримання поточного стану всіх сутностей (датчиків, реле) та підписується на подальші зміни. Отримані дані передаються в Angular Signals. Це активує миттєве відображення показників температури, вологості та відеопотоку на головній сторінці (/home) без необхідності втручання користувача.

2.3 Проєктування інтерфейсу користувача

Проєктування візуальної складової додатку здійснювалося з дотриманням принципів мобільної ергономіки та гайдлайнів Apple Human Interface / Google Material Design.

Як бібліотеку компонентів було використано Ionic Framework, який автоматично адаптує стиль кнопок, карток та віджетів під операційну систему середовища виконання.

Сторінка конфігурації (Settings Page) (рис. 2.5).

Логіка налаштування системи спроєктована таким чином, щоб зробити неможливою помилку ручного введення з боку користувача (Human Error).

Після введення адреси сервера та токену, сторінка здійснює тестове підключення, завантажує словник сутностей та динамічно формує списки (<ion-select>).

Система самостійно здійснює фільтрацію (підставляючи у випадаючий список виключно сутності, що починаються на switch.*, sensor.*, input_datetime.*), що значно спрощує етап маппінгу пристроїв.

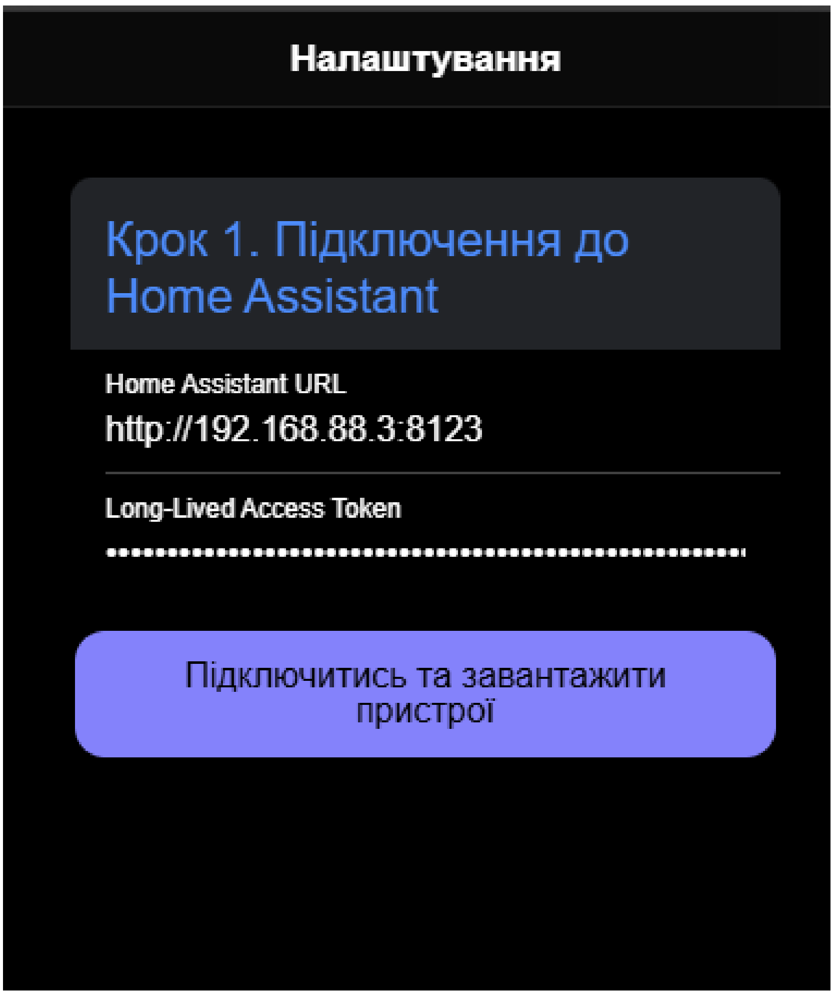

Рисунок 2.5 – Сторінка конфігурації (Settings Page)

Головний дашборд (Home Page) мобільної панелі (рис. 2.6) розділено на функціональні блоки з урахуванням пріоритетів користувача:

Блок відеонагляду. Займає верхню частину екрану і показує "живий" MJPEG-потік з модуля ESP32-CAM, проксіюючи трафік через внутрішній API сервера (/api/camera_proxy_stream).

Блок мікроклімату. Виводить актуальні метрики температури та вологості, прив'язані до Computed Signals.

Панель ручного керування реле. Набір скомпонованих тумблерів (<ion-toggle>), які надсилають сервісні виклики (turn_on/turn_off) до реле освітлення, насоса помпи та вентиляторів. Крім статусу, логіка спроектована

так, що стан тумблера моментально синхронізується з індикаторами, отриманими від сервера.

Модуль налаштування розкладу поливу. Імплементовано у вигляді акордеону (<ion-accordion>), що зберігає корисний простір екрану. Використано нативні віджети вибору часу (<ion-datetime>), прив'язані до сутностей input_datetime серверу.

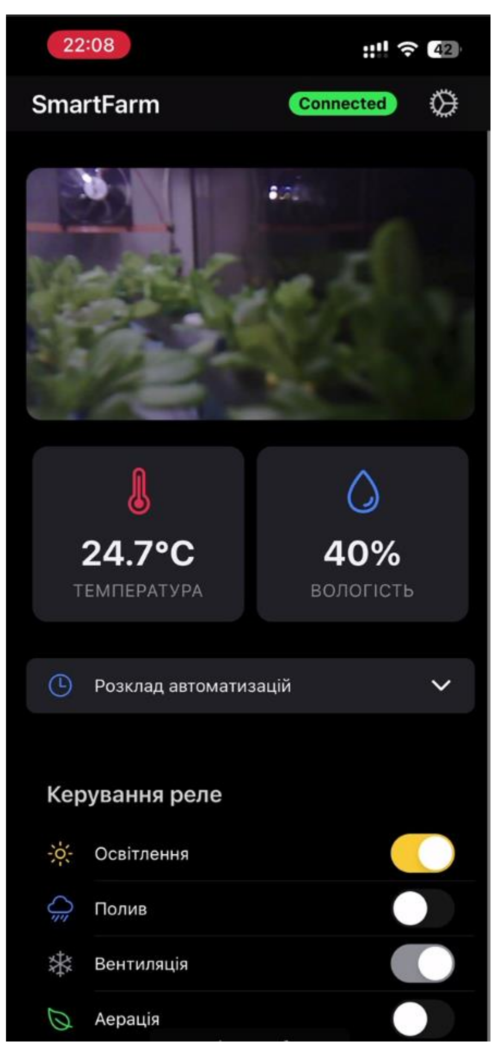

Рисунок 2.6 – Головний дашборд (Home Page)

Для забезпечення глибокого аналізу показників мікроклімату (температури та вологості) архітектурою передбачено окремий модуль візуалізації історичних даних (History Charts) (рис. 2.7).

Основу модуля складає високовиробнича бібліотека Ng-ApexCharts, яка відмальовує графіки часових рядів (Time-Series Area Charts) з підтримкою нативних мобільних жестів (Pinch-to-Zoom, Panning).

З архітектурної точки зору, для отримання масиву історичних даних (які можуть сягати тисяч записів) було прийнято рішення відмовитися від класичного REST API на користь існуючого відкритого WebSocket каналу.

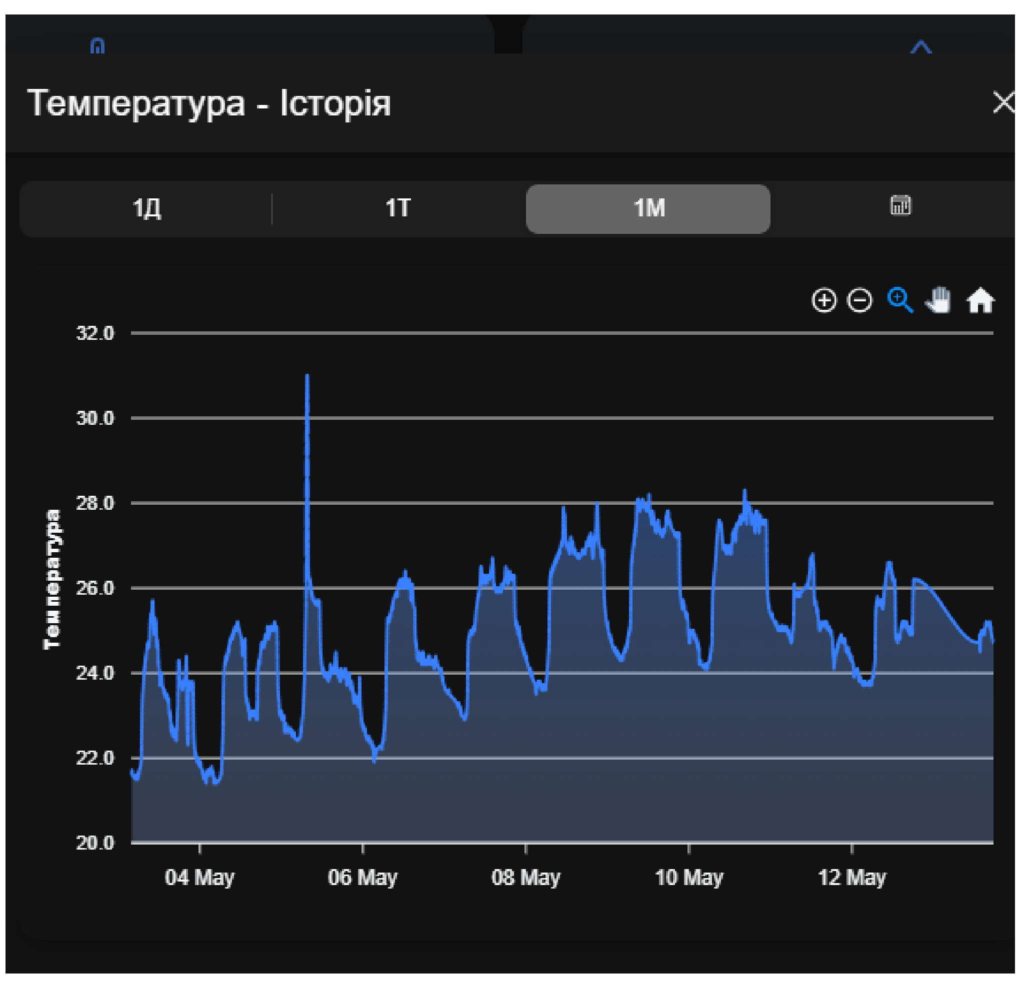

Рисунок 2.7 – Модуль інтерактивної аналітики (History Charts)

Додаток формує запит типу history/history_during_period безпосередньо в сокет, що дозволяє отримувати бінарно-оптимізовані відповіді без накладних витрат на повторні HTTP-з'єднання. Отримані дані парсяться та трансформуються у формат осей X/Y.

Інтерфейс аналітики відображається у вигляді спливаючого вікна поверх головного екрану (<ion-modal>). Всередині вікна реалізовано:

Панель швидких фільтрів: Сегментовані кнопки (1Д, 1Т, 1М) для миттєвої зміни вибірки даних на добу, тиждень або місяць.

Кастомний діапазон: Віджет системного календаря для вибору довільного проміжку часу від користувача.

Адаптивна темна тема: Кольорова палітра графіка примусово налаштована на темні відтінки (Dark Mode) для забезпечення ергономічності інтерфейсу та зменшення втоми очей при перегляді аналітики у вечірній час.

2.4 Питання безпеки

Делегування управління системами життєзабезпечення міні-ферми (вода, електрика, обігрів) зовнішньому мобільному клієнту формує критичні вимоги до безпеки з'єднання та захисту від несанкціонованого доступу.

На етапі проектування розглядалося кілька альтернативних варіантів організації авторизації клієнта на сервері Home Assistant.

Порівняльний аналіз варіантів авторизації [10].

Варіант 1. Базова парольна автентифікація (Basic Auth / Login & Password). Цей механізм передбачає введення логіна та пароля адміністратора системи безпосередньо у мобільному додатку.

Плюси: Інтуїтивно зрозумілий для звичайного користувача інтерфейс; не потребує попереднього генерування ключів на сервері.

Мінуси: Жорстке порушення сучасних стандартів безпеки. Зберігання базового пароля в локальній пам'яті смартфона (навіть у зашифрованому вигляді) створює ризик повної компрометації системи управління будинком у разі втрати пристрою. При зміні майстер-пароля на сервері доведеться вручну перелогінюватися на всіх клієнтах підсистеми.

Варіант 2. Механізм OAuth2 / JWT (Short-Lived Tokens) Передбачає генерацію короткострокових токенів доступу, які автоматично оновлюються (через Refresh Token).

Плюси: Найвищий академічний стандарт безпеки. Якщо токен перехоплено, він перестане діяти за кілька хвилин (наприклад, 15 хв).

Мінуси: Висока архітектурна надмірність (Overengineering) для локальної системи. Інтеграція повного циклу OAuth2 з Home Assistant вимагає побудови додаткового проміжного шару (Middlewares) та постійного фонового оновлення сесії, що перевантажує мобільний додаток логікою підтримки з'єднання.

Варіант 3. Довгострокові токени доступу (Long-Lived Access Tokens - LLAT) Специфічний механізм API-ключів, який є рідним стандартом для інтеграцій Home Assistant.

Плюси: Ідеальний баланс між продуктивністю та безпекою. Клієнт взагалі не знає майстер-пароля користувача. Токен працює як автономний "цифровий пропуск". У разі втрати або крадіжки смартфона, адміністратор через веб-інтерфейс Home Assistant може "в один клік" відкликати (Revoke) саме цей конкретний токен. Всі інші панелі управління будинком залишаться працювати.

Мінуси: Якщо токен буде фізично скопійовано зі смартфона через ADB-експлойт, зловмисник матиме доступ до системи до моменту ручного блокування токена адміністратором. З огляду на закритий характер ОС Android/iOS (Sandboxing), цей ризик було визнано прийнятним.

Під час розробки перевагу було надано Варіанту 3 (LLAT). Згенерований криптографічний токен прив'язується до конкретного екземпляра нашого мобільного додатку і зберігається за допомогою захищеного апаратного плагіна Capacitor Preferences (який шифрує дані засобами ОС мобільного пристрою).

Крім серверної авторизації, розроблено локальний програмний захист маршрутизації додатку (Angular Route Guards). Оскільки мобільний пристрій

може втратити локальні налаштування або зазнати крашу пам'яті, було написано функцію-запобіжник SettingsGuard. Вона виконує роль "годинникового" на старті програми. У разі відсутності токена доступу, захисник маршруту жорстко блокує доступ користувача до Головного дашборду ферми та примусово направляє його до екрану "Налаштування" [11].

Це вирішує відразу кілька безпекових завдань:

- виключає спробу ініціалізації пустого доступу по WebSocket;

- блокує можливість витоку внутрішніх сторінок з UI;

- захищає ядро додатку від непередбачуваних помилок середовища (Null Reference Exceptions), гарантуючи стабільність (Fail-Safe) застосунку.

Окрему увагу в розробці приділено мережевому транспорту. За замовчуванням система підтримує роботу в локальній мережі Wi-Fi. Для безпечного управління фермою через глобальний Інтернет (3G/4G/5G) архітектурно передбачено підключення клієнта не прямо до IP-адреси, а до зовнішнього доменного імені, прихованого засобами зворотного проксі-сервера (наприклад, NGINX Proxy Manager чи Cloudflare Tunnels), що дозволяє підняти зашифрований WebSocket (WSS) або HTTPS канал та нівелює ризики атаки "Man-in-the-Middle" [12].

3 РОЗРАХУНКОВА ЧАСТИНА

3.1 Вибір та налаштування інструментальних засобів розробки

Практична реалізація клієнтської частини системи "розумного будинку" (мобільного додатка для міні-ферми) розпочалася з ґрунтовного підбору та налаштування інструментальних засобів розробки.

Оскільки сучасна розробка багатоплатформних рішень потребує інтеграції різноманітних технологій в єдиний робочий процес (Workflow), вибір правильного середовища є критично важливим етапом, що впливає на швидкість написання коду, виявлення помилок та загальну стабільність життєвого циклу проекту.

В якості основного редактора вихідного коду (Integrated Development Environment - IDE) було обрано Visual Studio Code (VS Code) від корпорації Microsoft. Вибір на користь VS Code зумовлений кількома вагомими факторами. По-перше, цей редактор має вбудовану повноцінну підтримку мови TypeScript, якою написаний фреймворк Angular.

Завдяки рушію IntelliSense, середовище автоматично аналізує структуру класів, імпорти модулів та типи змінних, пропонуючи розробнику контекстні підказки (Autocompletion) у режимі реального часу. Це дозволило практично звести до нуля кількість синтаксичних помилок ще до моменту запуску компіляції.

По-друге, розширюваність платформи за допомогою плагінів дозволила перетворити звичайний текстовий редактор на потужний комбайн. Так, під час розробки застосовувалися такі розширення:

Angular Language Service: для автодоповнення HTML-тегів у шаблонах та перевірки прив'язки даних (Data Binding) до змінних класу.

Prettier: автоматичний форматувальник коду, який при кожному збереженні файлу (Save) вирівнював відступи (Indentation) згідно із загальноприйнятим стандартом, що забезпечило високу читабельність коду.

Ionic Extensions: для швидкої генерації каркасів сторінок (Scaffolding). Крім того, вбудований у VS Code термінал дозволив одночасно запускати сервер розробки, виконувати команди компіляції та взаємодіяти із системою контролю версій без необхідності перемикатися між різними вікнами операційної системи.

Незважаючи на те, що проект виконувався одноосібним розробником, запровадження системи контролю версій Git стало ключовим архітектурним рішенням. Під час експериментів із підключенням сторонніх бібліотек (наприклад, графіків Ng-ApexCharts або плагінів Capacitor) існував значний ризик порушення робочої структури проекту та виникнення конфліктів несумісності пакетів. Завдяки Git, перед початком кожної нової ізольованої задачі створювалася окрема гілка (Branch). Це дозволяло безпечно тестувати новий функціонал, зберігаючи стабільну робочу версію додатку у гілці main.

Періодичні фіксації коду (Commits) після завершення логічних блоків роботи створили так звану "мережу безпеки" (Safety Net). У випадку, коли оновлення версії фреймворку приводило до фатальних помилок компіляції (Breaking Changes), використання команди git checkout дозволяло за лічені секунди відкотитися до попереднього працездатного стану. Також, історія комітів (Git Log) стала цінним джерелом інформації при написанні даної розрахунково-пояснювальної записки, оскільки дозволяла відстежити, які саме файли змінювалися для вирішення конкретної технічної проблеми.

В якості базової платформи виконання інфраструктурного JavaScript коду було обрано середовище NodeJS. Після його встановлення, за допомогою пакетного менеджера npm (Node Package Manager) було інстальовано консольну утиліту Ionic CLI (@ionic/cli).

Базовий каркас мобільного додатка було згенеровано за допомогою команди ініціалізації (ionic start), обравши шаблон порожнього проекту (blank) та фреймворк Angular (--type=angular) [13].

З метою забезпечення заявленого функціоналу системи до проекту було додано низку ключових залежностей: офіційну бібліотеку home-assistant-js-websocket (для повнодуплексного обміну інформацією через WebSockets) та набір плагінів @capacitor/preferences (для організації безпечного збереження токенів "на залізі" пристрою).

Збірка кінцевих .apk файлів і подальше тестування нативної поведінки додатку здійснювалися за допомогою інструментарію Android Studio та її компонента Android Emulator (Android Virtual Device - AVD).

Розробка мобільних застосунків безпосередньо з підключеним через USB-кабель смартфоном часто є незручною через постійне розряджання батареї та знос портів, тому емулятор став основним полігоном для перевірок.

Android Emulator забезпечив низку критичних для тестування можливостей. Перш за все, він дозволив зімітувати різні роздільні здатності екранів (від невеликих смартфонів до планшетів), що допомогло відкалібрувати CSS-медіазапити (Media Queries) та переконатися, що картки датчиків (<ion-card>) та графіки температури не спотворюються (не виходять за межі екрану).

Також, вбудовані в емулятор засоби для розробників дозволили здійснювати тестування продуктивності додатку в умовах обмежених ресурсів. Зокрема, імітувалися штучні затримки мережі (Network Latency Throttling) на рівні 3G або покриття EDGE, що дало можливість переконатися в коректному спрацюванні таймаутів WebSocket-з'єднання та появі відповідних попереджень в інтерфейсі користувача про втрату зв'язку з міні-фермою. Можливість "на льоту" змінювати системну тему розфарбування емулятора (System Dark/Light Mode) гарантувала, що впроваджена примусова темна палітра (Force Dark Mode) для графіків ApexCharts відображається консистентно і не перекривається білими кольорами системи.

3.2 Програмна реалізація модулів зв'язку та управління станом

Основним комунікаційним ядром додатка виступає клас HomeAssistantService. Оскільки політика безпеки Home Assistant вимагає наявності захищеного підключення, ініціалізація зв'язку реалізована через створення об'єкта авторизації методом createLongLivedTokenAuth. Для цього сервіс звертається до попередньо збереженого у пам'яті смартфона (через Capacitor Preferences) довгострокового токена доступу (Long-Lived Access Token, LLAT).

Після успішної валідації токена, метод createConnection відкриває WebSocket-сокет, що дозволяє додатку обмінюватися даними в реальному часі. Окремої уваги заслуговує підхід до збереження отриманої телеметрії. Замість класичного використання RxJS BehaviorSubject або змінних стану, було застосовано новітній механізм Angular Signals. Отриманий масив сутностей (об'єктів пристроїв міні-ферми) зберігається у спеціальний реактивний контейнер.

Використання Signals дозволило значно оптимізувати процес рендерингу графічного інтерфейсу: коли сервер надсилає новий пакет даних (наприклад, змінилась лише температура на 0.1°C), Angular точково оновлює в DOM-дереві виключно текстове поле температури, оминаючи зайві цикли перевірки zone.js для статичних кнопок та інших елементів, що розвантажує процесор мобільного пристрою та економить заряд батареї.

Лістинг ключових методів сервісу наведено в Додатку А.

Логіка конфігурації реалізується на сторінці Налаштувань (settings.page.ts) (рис. 3.1). Процедура налаштування маппінгу пристроїв побудована за принципом динамічного підвантаження. Після підключення до сервера, додаток видобуває повний перелік інстальованих на сервері компонентів.

Через фільтрацію масиву за префіксами (наприклад, sensor.*, switch.*, input_datetime.*), програмний код розподіляє масив по категоріях та генерує масиви availableSensors, availableSwitches тощо.

Ці масиви слугують джерелом для випадаючих списків UI, що повністю усуває ймовірність помилок користувача, пов'язаних з ручним введенням технічних ідентифікаторів об'єктів (Entity IDs).

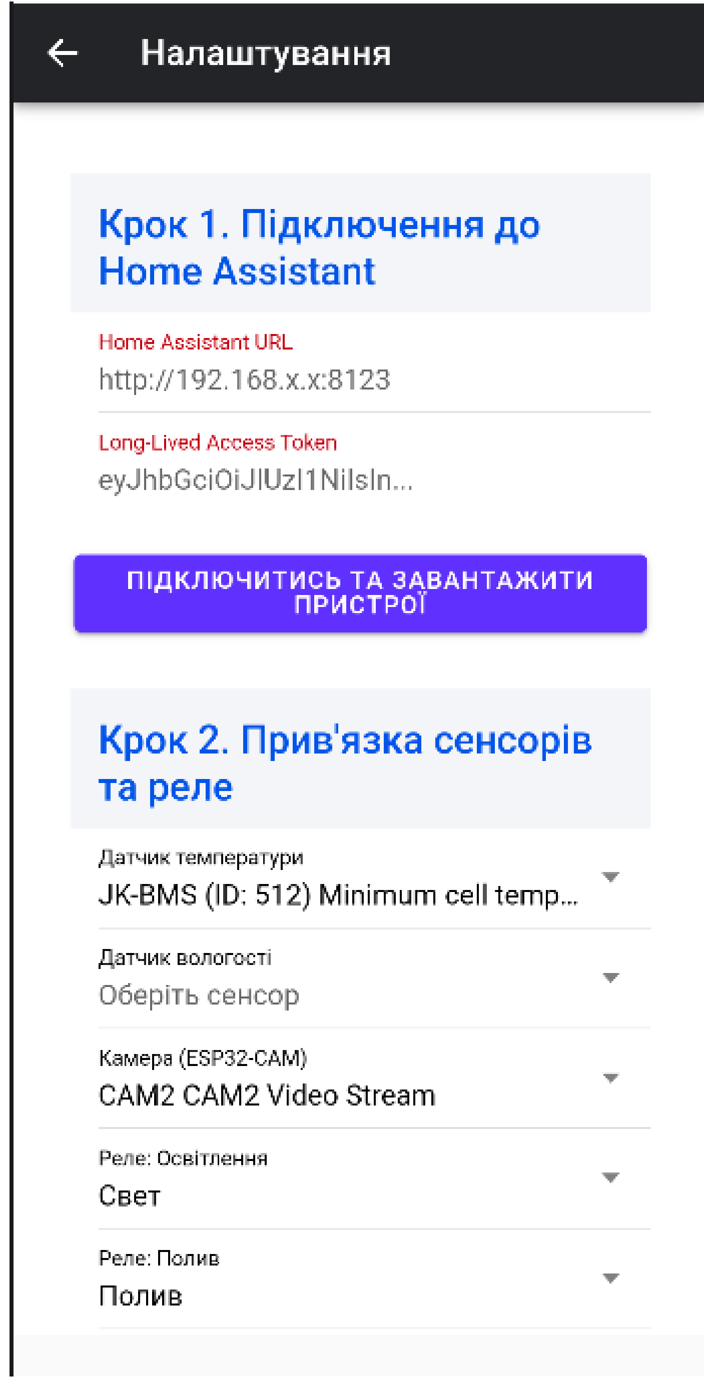

Рисунок 3.1 - Інтерфейс підключення та прив'язки датчиків міні-ферми.

3.3 Реалізація клієнтського інтерфейсу

Клієнтський інтерфейс (UI) головного екрана розроблено на основі бібліотеки веб-компонентів Ionic Framework (рис. 3.2).

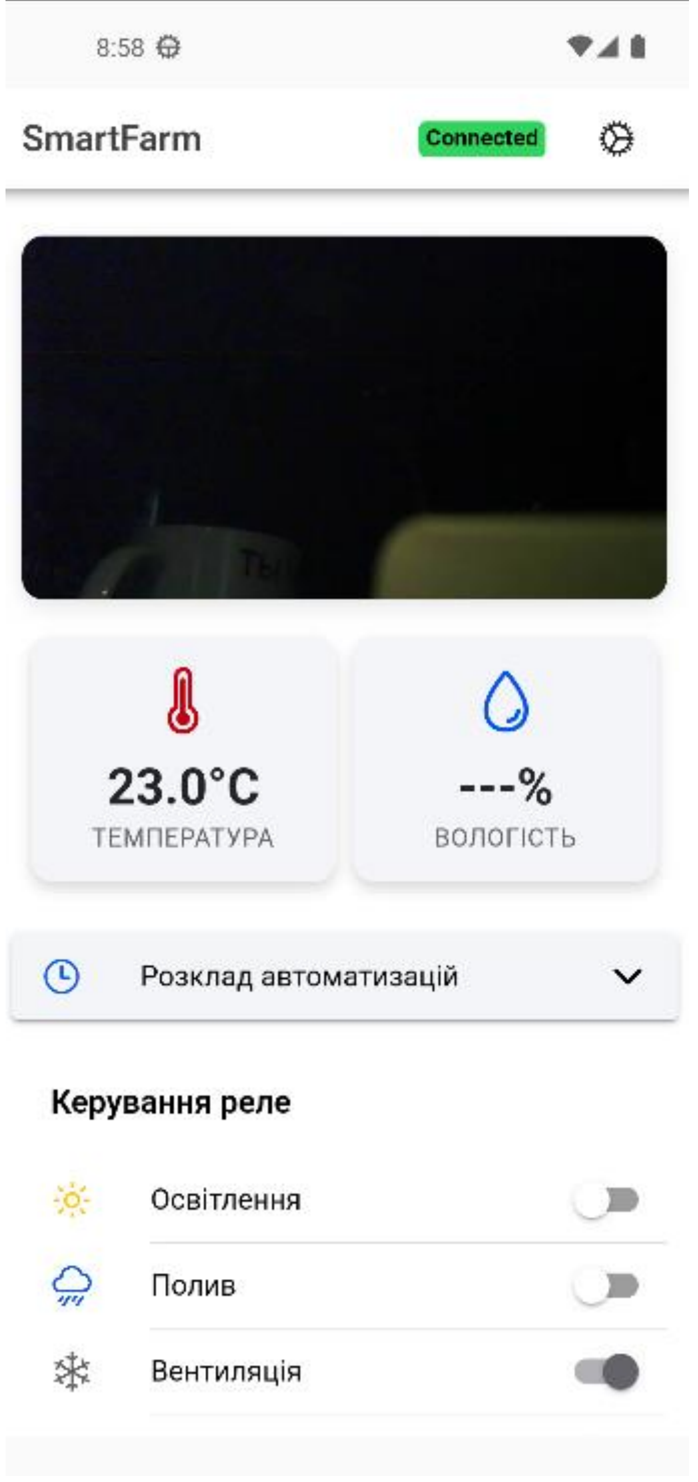

Рисунок 3.2 - Головна панель керування автоматизованою міні-фермою

Використання фреймворку Ionic дозволило реалізувати концепцію Cross-Platform Adaptive UI, де веб-технології (HTML, SCSS, TypeScript) забезпечують нативний вигляд та продуктивність додатка. Основний акцент зроблено на декларативному підході до побудови інтерфейсу.

1. Структурна організація та контейнеризація.

Верстка базується на спеціалізованих компонентах, що оптимізують рендеринг на мобільних пристроях:

<ion-content>: використовується як основний контейнер із вбудованою оптимізацією прокручування та обробкою безпечних зон (safe areas) екранів смартфонів.

<ion-card>: виконує роль логічного контейнера для групування елементів керування міні-фермою. Це дозволяє візуально відокремити блок моніторингу (датчики) від блоку активного управління (реле), забезпечуючи інтуїтивне сприйняття інтерфейсу.

2. Динамічне керування та синхронізація станів

Панель керування реле (освітлення, полив) реалізована через ієрархію <ion-item> всередині списку. Кожен пункт містить компонент <ion-toggle>, який є ключовим елементом взаємодії.

Синхронізація через WebSocket: На відміну від традиційних HTTP-запитів, використання протоколу WebSocket дозволяє підтримувати постійне двостороннє з'єднання. Стан перемикача прив'язаний до computed-сигналу (або Observable-потоку), який автоматично оновлюється при надходженні нових даних від сервера Home Assistant. Це виключає затримки та гарантує, що користувач бачить актуальний стан обладнання, навіть якщо воно було перемкнене іншим автоматизованим сценарієм.

Механізм зворотного зв'язку: При маніпуляції перемикачем ініціюється подія, що викликає метод toggleRelay().

3. Взаємодія з API сервера (Service Calls)

Метод toggleRelay() не просто змінює локальну змінну, а виступає тригером для сервісної логіки:

- формування команди: визначається цільова сутність (entity_id) та необхідна дія (turn_on або turn_off);

- виклик callService: команда транслюється в метод API сервера Home Assistant. Це забезпечує високу надійність, оскільки мобільний додаток виступає в ролі інтелектуального пульта, що віддає команди центральному контролеру, який, у свою чергу, підтверджує успішність виконання операції.

Детальну увагу приділено модулю управління розкладом (рис. 3.3).

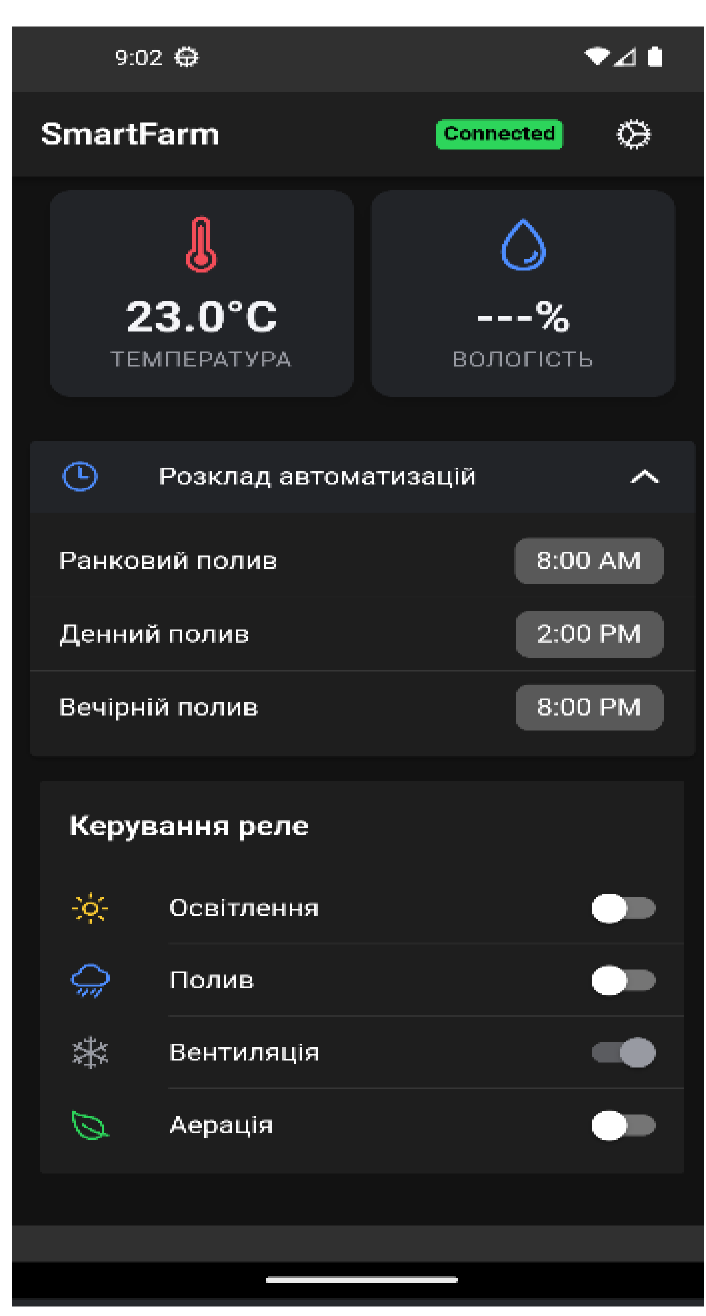

Рисунок 3.3 - Модуль управління розкладом

Для керування часовими інтервалами автоматизації (наприклад, графіком поливу чи освітлення) було розроблено інтерактивний блок на основі компонентів Ionic Framework, що забезпечує компактність інтерфейсу та зручність взаємодії на мобільних пристроях.

З метою раціонального використання екранного простору блок налаштування таймерів виконано у вигляді розширюваного акордеону <ion-accordion>. Це дозволяє приховати допоміжні елементи керування, коли вони не потрібні, фокусуючи увагу користувача на поточних показниках датчиків.

Для вибору точного часу інтегровано віджет <ion-datetime>, який розміщено всередині модального вікна <ion-modal>. Такий підхід («Picker-pattern») є нативним для мобільних ОС (iOS/Android), що мінімізує ризик помилкового введення даних та забезпечує високу ергономіку інтерфейсу.

Особливістю інтеграції з Home Assistant є розбіжність форматів представлення часу в мобільному фреймворку та на сервері:

Клієнтська сторона: компонент <ion-datetime> при генерації події ionChange повертає об'єкт у стандарті ISO 8601 (наприклад, 2026-05-12T08:30:00.000Z).

Серверна сторона: Сутності типу input_datetime в Home Assistant оперують строковими значеннями у форматі HH:mm:ss.

Для забезпечення сумісності було розроблено спеціалізований сервіс-конвертер. При спрацюванні тригера зміни часу алгоритм виконує парсинг ISO-рядка, виділяючи сегменти годин, хвилин та секунд, і формує валідний для сервера пакет даних.

Після формування цільового рядка часу додаток ініціює запит до сервера через метод sendCommand(). Структура виклику базується на трьох параметрах:

Domain: input_datetime (тип сутності).

Service: set_datetime (функція встановлення значення).

Payload: Об'єкт, що містить ідентифікатор пристрою (entity_id) та нове строкове значення часу.

Завдяки використанню computed-сигналів інтерфейс акордеону завжди відображає актуальний час, встановлений на сервері. Якщо таймер буде змінено іншим користувачем або внутрішньою логікою Home Assistant (наприклад, скриптом «Нічний режим»), значення в акордеоні оновиться автоматично без необхідності перезавантаження сторінки, що підкреслює реалізацію принципу Real-time Synchronization.

Візуальний моніторинг стану міні-ферми реалізовано за допомогою інтеграції модуля ESP32-CAM (рис. 3.4), який виступає як джерело графічних даних. Програмне рішення для виводу відеопотоку базується на механізмі проксіювання трафіку через центральний сервер автоматизації.

Замість прямого звернення додатка до IP-адреси камери, імплементовано використання вбудованого в Home Assistant сервісу camera_proxy_stream. Це створює надійний архітектурний рівень:

ESP32-CAM передає лише один потік даних на сервер, який потім може транслюватися декільком клієнтам одночасно.

Додаток звертається до єдиної точки входу, що спрощує роботу в різних мережах (Wi-Fi або мобільний інтернет).

Для забезпечення конфіденційності відеоданих доступ до проксі-сервера захищено за допомогою токена тривалого доступу (Long-Lived Access Token, LLAT).

Додаток динамічно формує URL для тегу <img>, додаючи токен як параметр запиту. Це дозволяє стандартному HTML-елементу успішно проходити перевірку прав доступу на боці сервера.

Такий метод виключає необхідність зберігання логінів та паролів у відкритому вигляді та дозволяє серверу миттєво анулювати доступ у разі компрометації токена.

Вибір формату MJPEG для відображення потоку в тезі <img> є стратегічним рішенням.

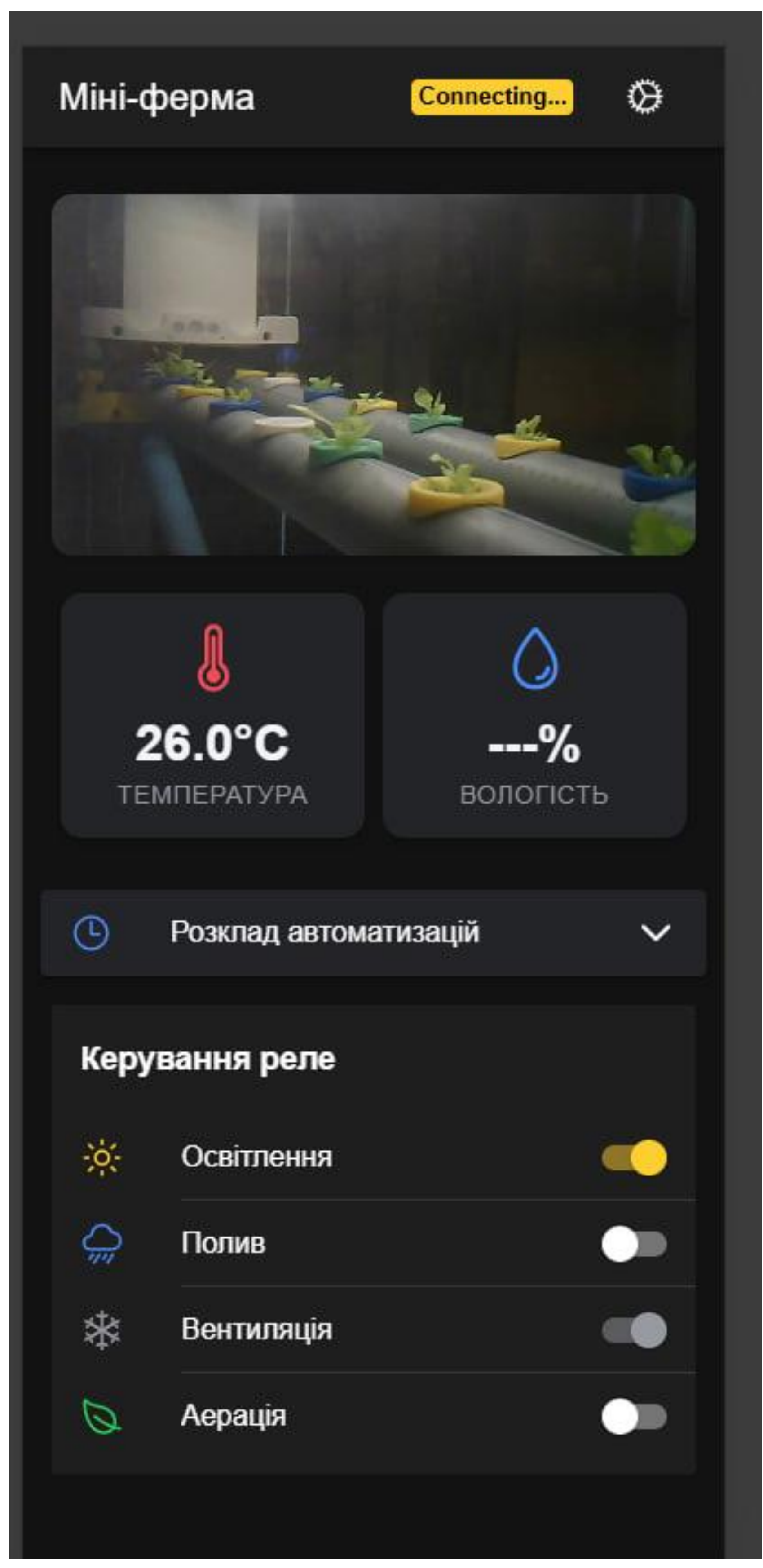

Рисунок 3.4 - Відображення візуального потоку з камери ESP32-CAM

Більшість сучасних браузерів та WebView-компонентів підтримують MJPEG «з коробки», що дозволяє оновлювати кадри автоматично без використання важких JS-плеєрів або бібліотек (як-от WebRTC чи HLS).

MJPEG забезпечує мінімальну затримку між подією на фермі та її відображенням у додатку, що є критично важливим для оперативного контролю.

Завдяки тому, що Home Assistant бере на себе роль медіа-сервера, ESP32-CAM не потребує виконання складних операцій з автентифікації користувачів або керування багатьма сесіями. Це дозволяє вивільнити ресурси процесора для стабільної генерації кадрів та підтримки Wi-Fi з'єднання, що підвищує загальну відмовостійкість системи моніторингу.

"Окрім візуального контролю через відеопотік, критично важливим аспектом функціонування розумної ферми є глибокий аналіз накопичених сенсорних даних. На рис. 3.5 наведено інтерфейс модуля аналітики мобільного додатка, який відповідає за візуалізацію історичних даних мікроклімату та реалізований за допомогою бібліотеки побудови інтерактивних діаграм ApexCharts.

Як бачимо на екрані, графіки розділені за ключовими фізичними величинами - окремо для температури та вологості, що дозволяє користувачу не плутати шкали та чітко відстежувати залежності. Дані для цих графіків підтягуються безпосередньо з бази даних сервера Home Assistant за протоколом WebSocket, що забезпечує високу швидкість завантаження великих масивів інформації без перевантаження пам'яті мобільного пристрою.

Головною перевагою даного інтерфейсу є його гнучкість та інтерактивність. У верхній частині модального вікна розташована панель швидкого вибору діапазонів часу, де користувач в один дотик може перемикати періоди відображення: за 1 день (1Д), 1 тиждень (1Т) або 1 місяць (1М). Для більш глибокого аналізу передбачена окрема кнопка з іконкою календаря, яка дозволяє задати довільний проміжок часу (Custom Range) для ретроспективної оцінки будь-якого специфічного циклу вирощування рослин.

Окрім цього, графіки підтримують взаємодію в режимі реального часу. При наведенні курсору, або при натисканні пальцем на сенсорному екрані смартфона на будь-яку ділянку кривої, автоматично спрацьовує механізм підказки (Tooltip) - з'являється інтерактивний маркер із фіксацією точного значення показника та відповідної йому дати й часу з точністю до хвилини.

Наприклад, на лівому графіку рис. 3.5 наочно продемонстровано такий маркер, який зафіксував температуру 27.6 °C станом на 10 травня о 20:43. Додатковий функціонал масштабування (pinch-to-zoom) дозволяє детально розглядати локальні мінімуми та максимуми, допомагаючи оцінити, наскільки коректно відпрацьовує автоматика смарт-ферми."

Рисунок 3.5 - Візуалізація історичних даних зміни мікроклімату

3.4 Збірка додатка та тестування на мобільному пристрої

Після успішного завершення етапу розробки було проведено комплексне тестування програмного забезпечення на реальному мобільному пристрої. Транспозиція написаного Angular/Ionic коду у виконуваний файл мобільної операційної системи здійснювалась через утиліту Capacitor.

Для цього було проініціалізовано нативну платформу Android інструкцією npx cap add android. Після збирання веб-ресурсів компайлером Angular (npm run build), згенеровані файли HTML/JS були автоматично скопійовані до директорії нативного проекту командою npx cap sync. Отриманий проект формату Android Studio було трансформовано в кінцевий дистрибутив - файл app-release.apk.

Натуральне тестування програмного продукту на реальному мобільному пристрої охоплювало кілька сценаріїв. Отримано наступні результати:

Тестування швидкості відклику (Latency). При натисканні елемента керування ion-toggle в інтерфейсі додатку (рис. 3.6), час спрацювання фізичного електромагнітного реле на платі міні-ферми складав менше 100 мілісекунд (в межах локальної мережі Wi-Fi), що свідчить про високу ефективність протоколу WebSocket.

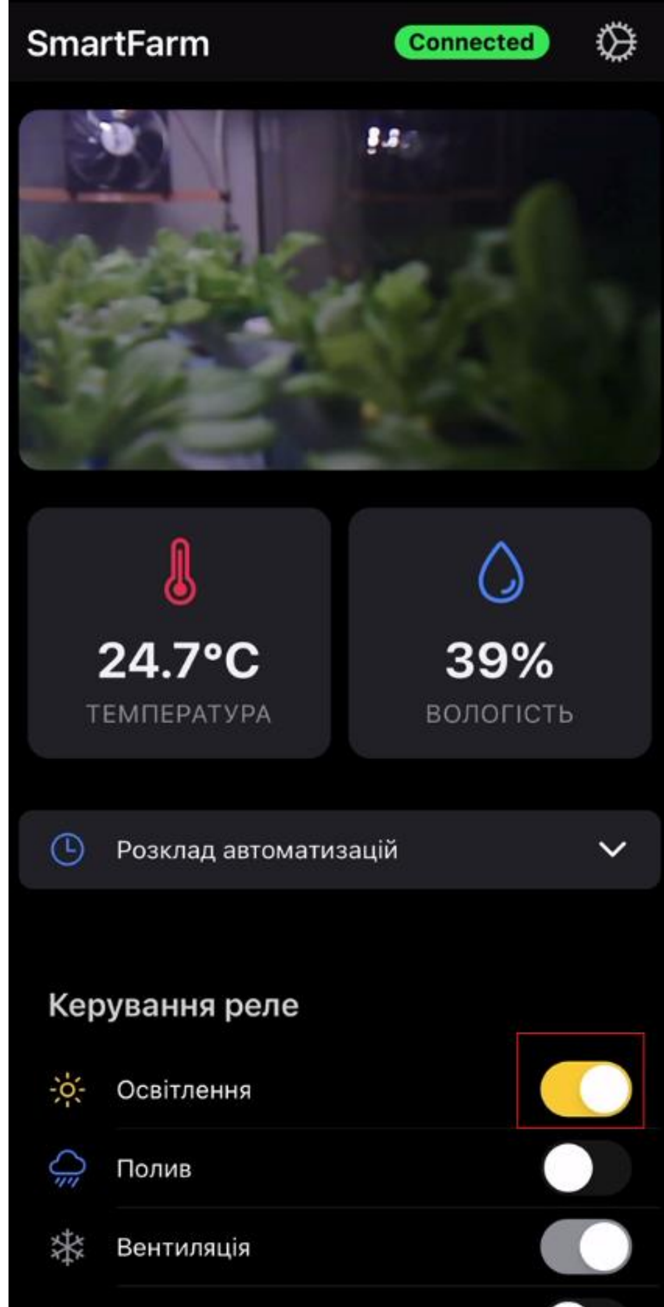

Рисунок 3.6 – Реакція інтерфейсу на активацію реле освітлення

Тестування стабільності з’єднання (Reconnect). Імітація обриву зв’язку (через тимчасове відключення Wi-Fi передавача на смартфоні) підтвердила коректне спрацювання обробників події disconnected. Додаток успішно повідомляв користувача про втрату з’єднання - відповідний індикатор статусу змінював колір на червоний (рис. 3.7). Після повернення смартфону до зони мережевого покриття система автоматично та без затримок відновлювала WebSocket канал (статус ready).

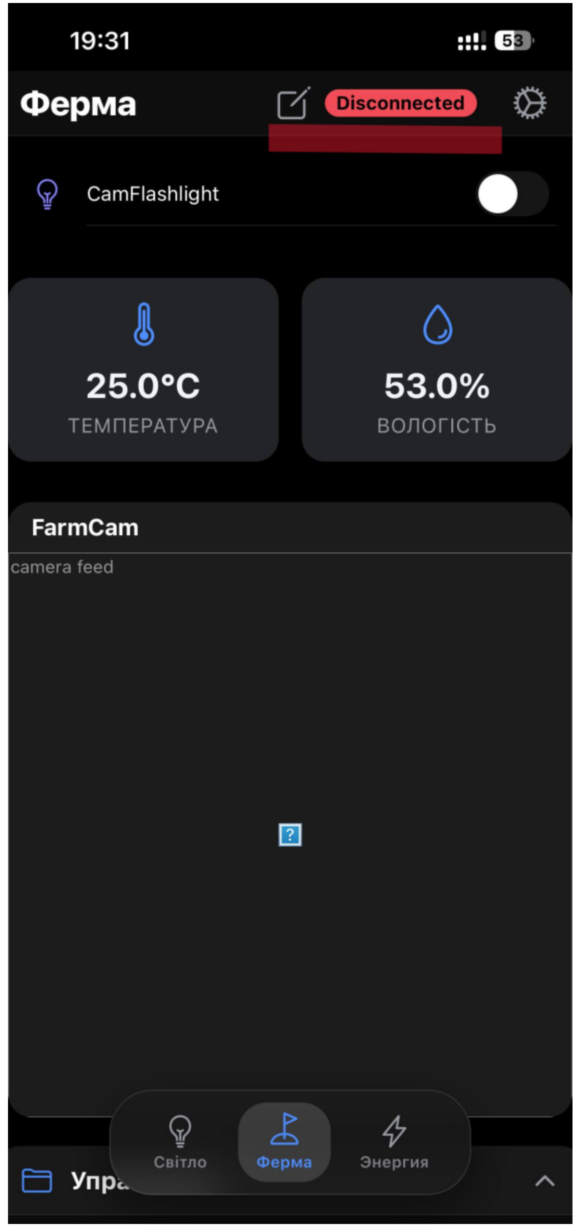

Рисунок 3.7 – Реакція системи на втрату з'єднання з сервером

Тестування постійності налаштувань. Аналіз поведінки після примусового завершення (Force Close) процесу додатку в диспетчері завдань мобільного пристрою показав, що модуль SettingsGuard безпомилково зчитує збережені конфігураційні ключі підключення з локального сховища @capacitor/preferences (рис. 3.8). Отримавши дійсний токен, додаток оминав етап авторизації і миттєво переводив користувача на головний дашборд із повним та актуальним відновленням усіх значень таймерів і станів реле автоматизованої міні-ферми.

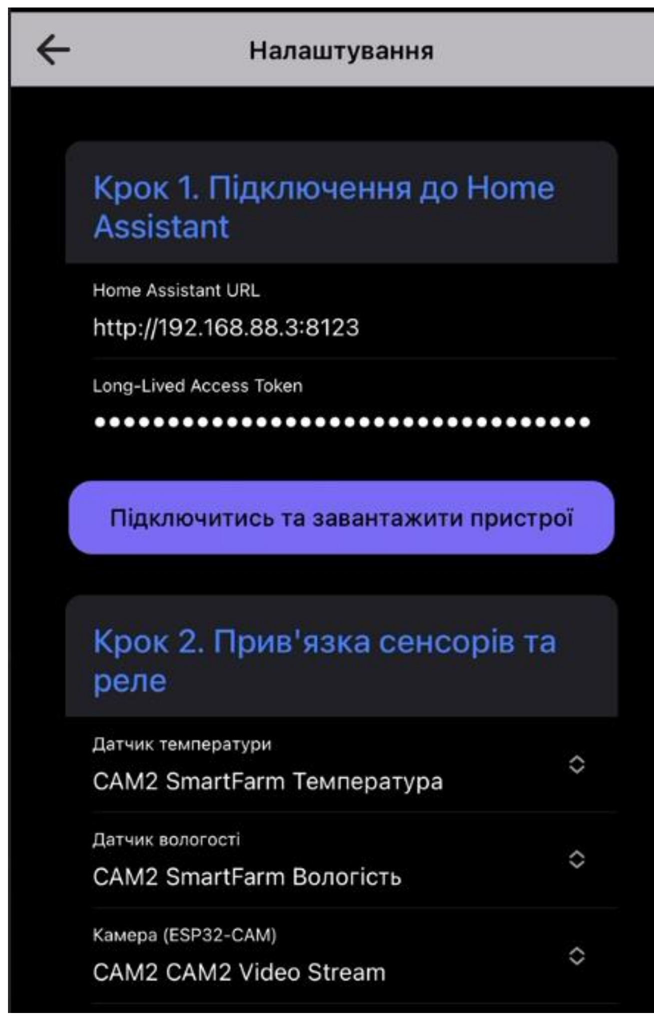

Рисунок 3.8 – Збережена конфігурація підключення у локальному сховищі

Загальний процес тестування підтвердив, що всі системні вимоги та сценарії використання (Use Cases) були виконані в повному обсязі, а розроблений програмний продукт демонструє високий рівень надійності та відмовостійкості.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Під час розробки мобільного додатка для керування міні-фермою, а також під час етапу програмно-апаратного тестування (прошивка ESP32-CAM, підключення реле) інженер-програміст перебуває під впливом низки виробничих факторів.

Метою даного розділу є аналіз умов праці та розробка комплексу заходів, спрямованих на збереження здоров'я, забезпечення комфортних та безпечних умов безперервної роботи за монітором персонального комп'ютера (ПК).

4.1 Характеристика умов праці інженера-програміста

Робота інженера-програміста належить до категорії осіб розумової праці (операторів ЕОМ). Характерними особливостями такого виду праці є: тривале перебування у статичній позі за робочим столом, інтенсивне зорове напруження через роботу зі світким екраном монітора, підвищене розумове навантаження під час написання алгоритмів та проектування архітектури, а також нервово-емоційна напруга у період пошуку та усунення помилок (дебаггінгу).

Згідно з чинними нормативними документами НПАОП 0.00-7.15-18 [14], така праця може спричинити зорову втому (астенопію), перенапруження м'язів спини та кистей рук (розвиток тунельного синдрому).

Робота інженера-програміста є складним видом інтелектуальної діяльності, що поєднує високе когнітивне навантаження з вираженою фізичною монотонією.

Згідно з НПАОП 0.00-7.15-18 [14] професійна діяльність розробника супроводжується впливом низки шкідливих виробничих факторів.

1. Психофізіологічні особливості праці.

Статичне навантаження: тривале перебування у вимушеній робочій позі призводить до постійної напруги м’язів шийного, грудного та поперекового відділів хребта. Відсутність динаміки викликає застійні явища у кровообігу, що є передумовою розвитку остеохондрозу та варикозного розширення вен.

Гіподинамія: недолік фізичної активності в поєднанні з інтенсивною розумовою працею негативно впливає на серцево-судинну систему та обмін речовин.

Нервово-емоційне напруження: розробка складних систем потребує тривалої концентрації уваги. Особливу напругу викликає процес дебаггінгу, який часто відбувається в умовах обмеженого часу, що може стати причиною хронічного стресу та професійного вигорання.

2. Зорове напруження та астенопія.

Постійна робота зі світними екранними пристроями (моніторами) створює специфічне навантаження на акомодаційний апарат ока:

Зорова втома (астенопія): виявляється через різь в очах, розмитість зображення та головний біль.

Синдром «сухого ока»: через зменшення частоти кліпання під час зосередженого читання коду слизова оболонка ока пересихає, що призводить до запальних процесів.

Синє світло: тривалий вплив короткохвильового спектра моніторів може порушувати режим сну розробника.

3. Специфічні професійні ризики.

Карпальний тунельний синдром: внаслідок тисяч однотипних рухів кистями рук відбувається здавлення серединного нерва в зап’ястному каналі.

Це викликає оніміння пальців та хронічний біль, що може призвести до втрати працездатності.

Комп'ютерний зоровий синдром: сукупність зорових та позаочних симптомів, пов’язаних із тривалою роботою за комп’ютером.

4.2 Вимоги до санітарно-гігієнічних умов у приміщенні

Для підтримання працездатності розробника необхідно забезпечити належні санітарно-гігієнічні умови на робочому місці.

Оптимальні параметри мікроклімату для робочих приміщень IT-сфери регламентуються Державними санітарними нормами ДСН 3.3.6.042-99 [15] (категорія важкості роботи – Ia, легка фізична робота). У теплий період року температура повітря має становити 23-25 °C, відносна вологість - 40-60%. У холодний період - 22-24 °C при тій самій вологості.

Приміщення має бути обладнане системами природного та штучного освітлення відповідно до ДБН В.2.5-28:2018 [16]. Природне освітлення здійснюється через вікна, які повинні мати жалюзі для захисту від прямих сонячних променів, щоб уникнути відблисків на моніторі. Штучне освітлення має бути рівномірним, з використанням люмінесцентних або LED-ламп загального освітлення. Нормована освітленість робочої поверхні столу становить не менше 300–500 лк.

Системний блок ПК та система вентиляції приміщення є основними джерелами виробничого шуму. Згідно з ДСН 3.3.6.037-99 [17] рівень еквівалентного шуму на робочих місцях програмістів не повинен перевищувати 50 дБА.

4.3 Ергономіка робочого місця

Раціональна організація робочого простору є критичною для профілактики втоми.

Ергономічна організація робочого простору інженера-програміста має на меті мінімізацію статичного навантаження на опорно-руховий апарат та запобігання зоровій декомпенсації.

Монітор є основним джерелом сенсорної інформації, тому його розташування критично впливає на поставу.

Дистанція та кут: відстань від очей до екрана має становити 600–700 мм. Верхня лінія активної зони монітора повинна знаходитися на рівні очей. Це дозволяє утримувати голову прямо, запобігаючи нахилу вперед, який створює навантаження на шийні хребці (C_1-C_7) вагою до 20–25 кг.

Антивідблиск: екран слід розташовувати перпендикулярно до віконних прорізів, щоб уникнути прямих та дзеркальних відблисків, які згідно з ДБН В.2.5-28:2018 [16] підвищують індекс дискомфорту (UGR).

Згідно з вимогами охорони праці, робоче крісло має бути підйомно-поворотним і забезпечувати активну підтримку тіла:

Підтримка хребта: спинка крісла повинна мати анатомічний вигин у поперековій зоні. Кут нахилу спинки відносно сидіння має бути в межах 90°–110°.

Правило прямих кутів: висота сидіння регулюється так, щоб стопи повністю стояли на підлозі, а кут згину в колінних та гомілковостопних суглобах становив 90°. Це забезпечує нормальний лімфодренаж і запобігає набрякам ніг.

Для профілактики карпального тунельного синдрому необхідно дотримуватися правил розташування периферійних пристроїв:

Підтримка передпліч: клавіатура та миша розміщуються на відстані 10–15 см від краю столу. Це дозволяє передпліччям повністю спиратися на стіл

або широкі підлокітники крісла, знімаючи напругу з дельтоподібних м'язів та трапеції.

Нейтральне положення зап'ясть: під час друку зап'ястя не повинні згинатися вгору чи вниз. Пряма лінія від ліктя до кінчиків пальців мінімізує тиск на серединний нерв у зап'ястному каналі.

Для роботи з текстовим кодом та дрібними схемами необхідна освітленість не менше 500 лк. Рекомендується використання LED-джерел із кольоровою температурою 4000K (нейтральне біле світло), що сприяє концентрації уваги.

Робоча поверхня поділяється на зони досяжності:

- центральна: монітор, клавіатура;
- зона правої руки: миша, вимірювальні прилади;
- зона лівої руки: технічна документація, допоміжне обладнання

4.4 Електробезпека під час розробки та тестування системи

Приміщення для роботи з ПК, за рівнем небезпеки ураження електричним струмом згідно ПУЕ-2017 [18], як правило, відноситься до приміщень без підвищеної небезпеки (сухі, з нормальною кімнатною температурою, не мають струмопровідної підлоги чи заземлених металевих конструкцій, до яких можна доторкнутись одночасно з корпусом електроприладу).

В рамках дипломної роботи розроблявся мобільний додаток, але його тестування проводилось із включеним модулем міні-ферми (плата мікроконтролера ESP32, модуль камери, блок реле).

Мікроконтролер (ESP32) та сенсори (DHT22) функціонують від наднизької безпечної напруги (3.3V або 5V постійного струму), експлуатація якої не становить загрози ураження для людини.

Водночас живлення модулів часто здійснюється через блоки живлення, ввімкнені у побутову мережу змінного струму (220 В, 50 Гц). Додатково, виконавчі реле на 5V комутують навантаження 220V (світильники, помпа для води).

Тому, на етапі прототипування суворо забороняється торкатися струмопровідних частин (контактів реле, оголених клем блоку живлення) голими руками під час роботи установки. Всі високовольтні з'єднання (>42V струму) повинні бути ізольовані термоусадкою або сховані у пластиковий діелектричний корпус установки.

Для ПК та моніторів використовується захисне занулення корпусу (підключення до захисного проводу PE) через контакти в євророзетках.

4.5 Пожежна безпека

IT-приміщення характеризуються підвищеною пожежною небезпекою через наявність великої кількості кабелів, електронної апаратури та синтетичних матеріалів. Головними причинами виникнення пожежі в офісах програмістів є коротке замикання мережі, перевантаження електропроводки або перегрів блоків живлення комп'ютерів.

Відповідно до НАПБ А.01.001-2014 [19], приміщення має бути оснащене первинними засобами пожежогасіння. Для гасіння електроустановок під напругою (сервери, ПК, плати ESP) категорично заборонено використовувати водні чи пінні вогнегасники. Оптимальним засобом гасіння є вогнегасник вуглекислотний (ОВК-2 або ВВК-2), оскільки зріджений діоксид вуглецю не проводить електричний струм і не залишає нальоту, що може пошкодити чутливі мікрочіпи серверу. Під час евакуації у разі пожежі персонал повинен негайно знеструмити електромережу рубильником на електрощиті і діяти згідно з розробленим та затвердженим планом евакуації об'єкта.

5 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Техніко-економічне обґрунтування створення програмного продукту

Метою даного розділу є техніко-економічне обґрунтування (ТЕО) розробки мобільного додатка для дистанційного керування елементами системи "розумний будинок" (автоматизована міні-ферма).

Розробка власного спеціалізованого програмного продукту (ПП) вимагає визначення економічної доцільності, розрахунку трудовитрат інженера-програміста та загального кошторису витрат на створення.

Розрахунок базується на припущенні, що розробку веде один інженер-програміст рівня Junior/Middle. Загальна тривалість створення програмного забезпечення поділяється на етапи:

- підготовка технічного завдання (ТЗ);
- проектування архітектури, безпосереднє кодування на Angular/Ionic;
- тестування та інтеграції з сервером Home Assistant.

5.2 Розрахунок трудомісткості розробки

Для розрахунку загальної трудомісткості розробки ПП складено календарний план робіт (табл. 5.1).

Календарний план дозволяє візуалізувати послідовність етапів та критичні точки проєкту.

Таблиця 5.1 - Розрахунок трудомісткості етапів розробки

№	Назва етапу робіт	Трудомісткість, люд.-год.	Відсоток від загального часу, %
1	Збір вимог та розробка технічного завдання (ТЗ)	40	9,1
2	Проєктування архітектури, вибір API, дизайн UI/UX	80	18,2
3	Написання коду (фронтенд Angular, інтеграція Capacitor)	200	45,4
4	Налаштування WebSocket та розробка логіки керування	80	18,2
5	Тестування, налагодження та компіляція релізного .apk	40	9,1
Разом	Загальна трудомісткість розробки ПП	440	100

Загальний час розробки становить 440 годин.

При стандартному 8-годинному робочому дні та 22 робочих днях на місяць (176 годин/місяць), розробка триватиме рівно 2,5 місяці.

5.3 Розрахунок витрат на заробітну плату та соціальні відрахування

Основою для визначення витрат на розробку є заробітна плата розробника. Для розрахунку приймемо орієнтовну середню заробітну плату інженера-програміста відповідної кваліфікації у розмірі 25 000 грн на місяць.

Середньоденна заробітна плата:

$$25\ 000 / 22 = 1\ 136{,}36 \text{ грн.}$$

Годинна тарифна ставка:

$$1\ 136{,}36 / 8 = 142{,}05 \text{ грн/год.}$$

Основна заробітна плата (Зо) за 440 годин:

$$Зо = 440 \text{ год.} \times 142{,}05 \text{ грн/год.} = 62\ 502{,}00 \text{ грн.}$$

Додаткова заробітна плата (Зд), яка враховує оплату відпусток та компенсацій, зазвичай приймається на рівні 10-15% від основної. Приймемо:

$$Зд = 10\%: Зд = 62\ 502{,}00 \times 0{,}10 = 6\ 250{,}20 \text{ грн.}$$

Загальний фонд заробітної плати (ФЗП):

$$ФЗП = Зо + Зд = 62\ 502{,}00 + 6\ 250{,}20 = 68\ 752{,}20 \text{ грн.}$$

Згідно з чинним законодавством України, роботодавець сплачує єдиний соціальний внесок (ЄСВ), який становить 22% від ФЗП. Відрахування на соціальні заходи (Всз):

$$Всз = ФЗП \times 0{,}22 = 68\ 752{,}20 \times 0{,}22 = 15\ 125{,}48 \text{ грн.}$$

5.4 Витрати на електроенергію та амортизацію обладнання

Під час розробки ПП використовувалося робоче місце інженера-програміста, обладнане ноутбуком, додатковим монітором та штучним освітленням приміщення.

Сумарна потужність обладнання (ПК + монітор + освітлення) орієнтовно становить Р = 0,4 кВт. Вартість 1 кВт-год електроенергії для непобутових споживачів становить близько 8,00 грн (умовний тариф).

Витрати на електроенергію (Ве) за 440 годин роботи:

Ве = 440 год. × 0,4 кВт × 8,00 грн/кВт-год = 1 408,00 грн.

Амортизація обладнання (Ва) розраховується лінійним методом. Вартість робочої станції балансовою вартістю 35 000 грн та терміном корисного використання 3 роки (36 місяців). Норма амортизації за 2,5 місяці розробки:

Ва = (35 000 / 36) × 2,5 = 2 430,55 грн.

5.5 Накладні витрати та загальна вартість програмного продукту

Накладні витрати (Вн), що включають витрати на інтернет, утримання приміщення (оренду), охорону тощо, зазвичай становлять близько 40-50% від фонду основної заробітної плати. Приймемо Вн = 40% від Зо.

Вн = 62 502,00 × 0,40 = 25 000,80 грн.

Тепер складемо загальний кошторис розробки програмного продукту.

Таблиця 5.2 - Кошторис витрат на розробку ПП

№	Стаття витрат	Сума, грн	Питома вага, %
1	Основна заробітна плата розробника (Зо)	62 502,00	55,44
2	Додаткова заробітна плата (Зд)	6 250,20	5,55
3	Єдиний соціальний внесок (ЄСВ 22%)	15 125,48	13,42
4	Витрати на електроенергію (Ве)	1 408,00	1,25
5	Амортизаційні відрахування (Ва)	2 430,55	2,16
6	Накладні витрати (Вн)	25 000,80	22,18
Разом	Загальна собівартість (Сзаг)	112 717,03	100,00

В результаті проведеного техніко-економічного розрахунку було встановлено, що повний цикл розробки мобільного додатка для локальної станції управління міні-фермою потребує 440 людино-годин (2,5 місяця роботи одного інженера).

Сумарна орієнтовна собівартість розробки склала 112 717,03 грн.

Створення власного спеціалізованого програмного рішення є повністю рентабельним порівняно з купівлею готових пропрієтарних SCADA-комплексів або замовленням аутсорсингу розробок у зовнішніх IT-компаній.

Отриманий продукт не потребує постійної щомісячної абонентської плати за хмарні послуги та створює додану вартість інтелектуальній системі.

ВИСНОВКИ

У результаті виконання дипломної роботи було повністю досягнуто поставленої мети - спроектовано, розроблено та протестовано повнофункціональний кросплатформний мобільний додаток для дистанційного керування елементами системи "розумний будинок" на прикладі автоматизованої міні-ферми. У ході виконання роботи було успішно вирішено всі заплановані теоретичні та інженерно-практичні завдання.

Основні наукові та практичні результати роботи полягають у наступному:

Проведено глибинний аналіз протоколів комунікації (WebSocket проти класичного HTTP REST) та сучасних архітектур розумного дому [20]. Доведено, що для задач реального часу, які потребують миттєвого відгуку реле та сенсорів, повнодуплексне з'єднання WebSocket є єдиним технічно виправданим вибором. Аналіз наявних систем підтвердив доцільність використання локальної платформи Home Assistant замість пропрієтарних хмарних рішень, що гарантує безперебійність роботи та інформаційну безпеку.

Спроектовано та розроблено кросплатформний мобільний додаток на базі передового фронтенд-фреймворку Angular 15+ (з використанням архітектури Standalone Components та реактивних Angular Signals) і комплексу мобільних веб-компонентів Ionic Framework 7+. Це дозволило отримати високу продуктивність та нативний дизайн (UI/UX), адаптований під середовище використання.

Реалізовано надзвичайно безпечне з'єднання додатку із сервером Home Assistant. Замість вразливого до перехоплення логіна та пароля, авторизація здійснюється через довгострокові токени доступу (Long-Lived Access Token). Механізм зберігання облікових даних реалізовано з використанням нативного реєстру пристрою через плагін Capacitor Preferences.

Розроблено розумний модуль динамічних налаштувань, який при ініціалізації самостійно опитує API сервера для отримання списку доступних сутностей (entities), автоматично класифікує їх (за префіксами sensor.*, switch.*, input_datetime.*) та формує зручні випадаючі списки. Це повністю нівелює проблему людського фактору при маппінгу пристроїв та адресації.

Створено інтуїтивно зрозумілий та ефективний користувацький інтерфейс (UI) для Головного дашборду. Інтерфейс забезпечує миттєвий моніторинг кліматичних сенсорів (вологість, температура), безпечне відображення "живого" MJPEG-відеопотоку з мікроконтролера камери ESP32-CAM та надає ергономічну панель керування виконавчими реле (полив, освітлення, вентиляція, аерація).

Успішно реалізовано алгоритм дистанційного керування розкладом автоматизацій. Замість прямого втручання в системні скрипти брокера, додаток використовує віртуальні об'єкти таймерів (input_datetime), конвертує введені користувачем значення часу в універсальний стандарт системи та відправляє їх на сервер (callService). Це забезпечує надійність роботи розкладу поливу незалежно від наявності зв'язку між телефоном і сервером.

Виконано успішну трансляцію («збірку») мобільного застосунку у фінальний релізний .apk файл за допомогою середовища Capacitor. Проведені тестування довели високу стійкість клієнта до втрати зв'язку та мінімальну (близько 100 мс) затримку виконання команд у локальній мережі.

Практичне значення отриманих результатів важко переоцінити. Розроблений мобільний додаток є повністю функціональним, надійним та масштабованим програмним продуктом. Хоча архітектура тестувалася на вирішенні вузькоспеціалізованих задач моніторингу клімату міні-ферми, модульний підхід дозволяє майже миттєво адаптувати та впроваджувати даний додаток для управління будь-якими іншими комерційними IoT-проектами, складськими системами або комплексами "розумного міста", що працюють на базі серверного ядра Home Assistant. Додаток готовий до впровадження та кінцевої експлуатації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Шмигельський О. В., Кунченко-Харченко В. І. Системи розумного будинку: архітектура, протоколи та безпека. Київ: Видавництво "Політехніка", 2020. 185 с.

2. Home Assistant Developer Documentation / Nabu Casa, Inc. URL: https://developers.home-assistant.io/ (дата звернення: 25.03.2026).

3. Home Assistant WebSocket API / Home Assistant Community. URL: https://developers.home-assistant.io/docs/api/websocket/ (дата звернення: 25.03.2026).

4. Angular Documentation: The modern web developer's platform / Google LLC. URL: https://angular.io/docs (дата звернення: 26.03.2026).

5. Freeman A. Pro Angular 16: Build Powerful and Dynamic Web Apps. Apress, 2023. 884 p.

6. Ionic Framework Official Documentation / Drifty Co. URL: https://ionicframework.com/docs (дата звернення: 26.03.2026).

7. Capacitor: Cross-platform native runtime for web apps / Drifty Co. URL: https://capacitorjs.com/docs (дата звернення: 27.03.2026).

8. Fette I., Melnikov A. The WebSocket Protocol. IETF RFC 6455. 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 27.03.2026).

9. ESPHome Documentation: Control your ESP8266/ESP32 / Nabu Casa, Inc. URL: https://esphome.io/ (дата звернення: 28.03.2026).

10. Коваленко В. М., Петренко С. В. Глобальні тенденції розвитку Інтернету речей (IoT). Вісник НУ "Львівська політехніка". Серія: Інформаційні системи та мережі. 2021. № 68. С. 45–52.

11. Робототехніка та Інтернет речей: навчальний посібник / О. В. Лисенко та ін. Харків: ХНУРЕ, 2019. 210 с.

12. Жураковський Б.Ю., Зенів І.О. Технології інтернету речей: навч. пос. Київ: КПІ ім. Ігоря Сікорського, 2021. 271 с.

13. TypeScript: Typed JavaScript at Any Scale / Microsoft. URL: https://www.typescriptlang.org/docs/ (дата звернення: 29.03.2026).

14. НПАОП 0.00-7.15-18 Вимоги щодо безпеки та захисту здоров`я працівників під час роботи з екранними пристроями [Чинний від 2018-05-18]. Київ: Міністерство соціальної політики України, 2018. 8 с.

15. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень. [Чинний від 1999-12-01]. Київ: МОЗ України, 1999. 14 с.

16. ДБН В.2.5-28:2018 Природне і штучне освітлення [Чинний від 2019-03-01]. Київ: НДІБК України, 2019. 138 с.

17. ДСН 3.3.6.037-99 Санітарні норми виробничого шуму, ультразвуку та інфразвуку [Чинний від 1999-12-01]. Київ: МОЗ України, 1999. 20 с.

18. ПУЕ-2017. Правила улаштування електроустановок. Київ: Міненерговугілля України, 2017. 617 с.

19. НАПБ А.01.001-2014. Правила пожежної безпеки в Україні. [Чинний від 2015-03-30]. Київ: МВС України, 2014. 102 с.

20. Richardson L., Amundsen M., Ruby S. RESTful Web APIs: Services for a Changing World. O'Reilly Media, 2013. 408 p.

ДОДАТОК А

Метод підключення та підписки на події реального часу

```
private async connect() {
  const config = this.settingsService.getConfig();
  if (!config.hassUrl || !config.hassToken) return;

  try {
    const auth = createLongLivedTokenAuth(config.hassUrl,
config.hassToken);
    this.haConnection = await createConnection({ auth });

    this.connectionStatus.set('Connected');
    this.connectionSessionId.set(Date.now());

    // Підписка на зміну станів всіх об'єктів розумного будинку
    subscribeEntities(this.haConnection, (newEntities:
HassEntities) => {
      this.entities.set(newEntities);
    });

    this.haConnection.addEventListener('disconnected', () => {
      this.connectionStatus.set('Disconnected');
    });

    this.haConnection.addEventListener('ready', () => {
      this.connectionStatus.set('Connected');
      // Оновлюємо SessionID при reconnect для скидання кешу
медіа-потоків
      this.connectionSessionId.set(Date.now());
    });
  } catch (error) {
    this.connectionStatus.set('Error');
  }
}
```

ДОДАТОК Б

Методи керування та запиту історії через сокет

```
public async sendCommand(domain: string, service: string,
entityId: string, serviceData: any = {}) {
  if (!this.haConnection) return;

  try {
    const payload = { entity_id: entityId, ...serviceData };
    await callService(this.haConnection, domain, service,
payload);
  } catch (error) {
    console.error(`[HA Service] Помилка виконання
${domain}.${service}:`, error);
  }
}
```

```
public async getEntityHistory(entityId: string, startTime: Date,
endTime: Date): Promise<any> {
  const payload = {
    type: 'history/history_during_period',
    start_time: startTime.toISOString(),
    end_time: endTime.toISOString(),
    minimal_response: true,
    no_attributes: true,
    entity_ids: [entityId]
  };
  return this.haConnection.sendMessagePromise(payload);
}
```

ДОДАТОК В

Фрагмент коду головної сторінки та реактивних станів

```
 // 1. Датчики мікроклімату
  temperature = computed(() => {
    const stateObj =
this.getEntityState(this.settingsService.getConfig().tempSensor)
;
    return stateObj && !isNaN(parseFloat(stateObj.state)) ?
parseFloat(stateObj.state).toFixed(1) : '---';
  });

  humidity = computed(() => {
    const stateObj =
this.getEntityState(this.settingsService.getConfig().humSensor);
    return stateObj && !isNaN(parseFloat(stateObj.state)) ?
parseFloat(stateObj.state).toFixed(0) : '---';
  });

  // 2. Генерація токена доступу до відеопотоку MJPEG
  cameraStreamUrl = computed(() => {
    const cfg = this.settingsService.getConfig();
    const sessionId = this.haService.connectionSessionId();
    const stateObj = this.getEntityState(cfg.cameraEntity);

    if (stateObj && stateObj.attributes.access_token) {
      return
`${cfg.hassUrl}/api/camera_proxy_stream/${cfg.cameraEntity}?toke
n=${stateObj.attributes.access_token}&_t=${sessionId}`;
    }
    return null;
  });
Керування реле:
  toggleRelay(relayKey: keyof SmartFarmConfig, event: any) {
    const isChecked: boolean = event.detail.checked;
    const entityId = this.settingsService.getConfig()[relayKey]
as string;

    if (!entityId) return;

    const domain = entityId.split('.')[0];
    const service = isChecked ? 'turn_on' : 'turn_off';

    this.haService.sendCommand(domain, service, entityId);
  }
```

ДОДАТОК Г

Лістинг компонента модального вікна історичних графіків

```
  async loadData() {
    this.isLoading = true;
    try {
      let end = new Date();
      let start = new Date();

      if (this.selectedPeriod === '1D')
start.setHours(start.getHours() - 24);
      else if (this.selectedPeriod === '1W')
start.setDate(start.getDate() - 7);
      else if (this.selectedPeriod === '1M')
start.setMonth(start.getMonth() - 1);
      else if (this.selectedPeriod === 'CUSTOM') {
        start = new Date(this.customStart);
        end = new Date(this.customEnd);
      }

      const response = await
this.haService.getEntityHistory(this.entityId, start, end);

      let chartData: [number, number][] = [];

      if (response && response[this.entityId]) {
        const events = response[this.entityId];
        chartData = events.map((e: any) => {
          const val = parseFloat(e.state || e.s);
          const rawTime = e.last_updated || e.lu ||
e.last_changed || e.lc;
          const timestamp = typeof rawTime === 'number' ?
rawTime * 1000 : new Date(rawTime).getTime();

          return [timestamp, isNaN(val) ? null : val];
        }).filter((d: any) => d[1] !== null);
      }

      this.chartOptions.series = [{ name: this.titleName, data:
chartData }];
    } catch (error) {
      console.error('[HistoryChart] Помилка завантаження
історії:', error);
    } finally {
      this.isLoading = false;
    }
  }
```

ДОДАТОК Д

Текст програми сторінки конфігурації

```
  async testConnectionAndFetchEntities(silent: boolean = false) {
    if (!this.config.hassUrl?.trim() ||
!this.config.hassToken?.trim()) return;
    this.isConnecting.set(true);

    try {
      const auth = createLongLivedTokenAuth(this.config.hassUrl,
this.config.hassToken);
      this.haConnection = await createConnection({ auth });
      let isInitialLoad = true;

      subscribeEntities(this.haConnection, (entities:
HassEntities) => {
        const entitiesArray = Object.values(entities);

        // Фільтрація об'єктів по доменах (sensors, switches,
cameras, timers)
        this.availableSensors.set(entitiesArray.filter(e =>
e.entity_id.startsWith('sensor.')));
        this.availableSwitches.set(entitiesArray.filter(e =>
e.entity_id.startsWith('switch.')));
        this.availableCameras.set(entitiesArray.filter(e =>
e.entity_id.startsWith('camera.')));
        this.availableTimers.set(entitiesArray.filter(e =>
e.entity_id.startsWith('input_datetime.')));

        this.isConnected.set(true);

        if (isInitialLoad) {
          this.isConnecting.set(false);
          isInitialLoad = false;
        }
      });
    } catch (error) {
      console.error('[SettingsPage] Помилка підключення:',
error);
      this.isConnecting.set(false);
    }
  }
```

ДОДАТОК Е

Лістинг сервісу локального збереження налаштувань

```
import { Injectable, signal } from '@angular/core';
import { Preferences } from '@capacitor/preferences';

export interface SmartFarmConfig {
  hassUrl: string;
  hassToken: string;
  tempSensor: string;
  humSensor: string;
  lightSwitch: string;
  pumpSwitch: string;
  fanSwitch: string;
  airSwitch: string;
  cameraEntity: string;
  morningTimer: string;
  dayTimer: string;
  eveningTimer: string;
}

@Injectable({
  providedIn: 'root'
})
export class SettingsService {
  public settingsLoaded = signal<boolean>(false);
  private config: Partial<SmartFarmConfig> = {};

  constructor() {
    this.loadSettings();
  }

  public async loadSettings(): Promise<void> {
    try {
      const keys: (keyof SmartFarmConfig)[] = [
        'hassUrl', 'hassToken', 'tempSensor', 'humSensor',
        'lightSwitch', 'pumpSwitch', 'fanSwitch', 'airSwitch',
'cameraEntity',
        'morningTimer', 'dayTimer', 'eveningTimer'
      ];

      for (const k of keys) {
        const res = await Preferences.get({ key: k });
        if (res.value) {
          this.config[k] = res.value;
        }
      }

      this.settingsLoaded.set(true);
```

```
    } catch (error) {
      console.error('[SettingsService] Помилка завантаження
конфігурації:', error);
      this.settingsLoaded.set(true);
    }
  }

  public async saveSettings(newConfig:
Partial<SmartFarmConfig>): Promise<void> {
    try {
      for (const [key, value] of Object.entries(newConfig)) {
        if (value) {
          // Зберігання кожної пари ключ-значення у захищений
розділ пам'яті
          await Preferences.set({ key, value: String(value) });
          this.config[key as keyof SmartFarmConfig] =
String(value);
        }
      }

      // Реактивне оновлення статусу
      this.settingsLoaded.set(false);
      this.settingsLoaded.set(true);

    } catch (error) {
      console.error('[SettingsService] Збій під час запису
налаштувань:', error);
      throw error;
    }
  }

  public hasSettings(): boolean {
    return !!this.config.hassUrl && !!this.config.hassToken;
  }

  public getConfig(): Partial<SmartFarmConfig> {
    return this.config;
  }
}
```

ДОДАТОК Ж

Код структури розмітки головного екрану

```
<ion-header [translucent]="true">
  <ion-toolbar>
    <!-- Бейдж з поточним статусом підключення у правому кутку -
->
    <ion-buttons slot="end" class="ion-padding-end">
      <ion-badge
        [color]="haService.connectionStatus() === 'Connected' ?
'success' : (haService.connectionStatus() === 'Disconnected' ?
'danger' : 'warning')"
        class="ion-margin-end">
        {{ haService.connectionStatus() }}
      </ion-badge>
      <ion-button (click)="goToSettings()" color="dark">
        <ion-icon name="cog-outline" slot="icon-only"></ion-
icon>
      </ion-button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>

<ion-content [fullscreen]="true">
  <!-- Картка з відеопотоком камери ESP32-CAM -->
  <ion-card class="camera-card">
    @if (cameraStreamUrl()) {
      <img [src]="cameraStreamUrl()" alt="camera feed"
class="camera-feed" />
    } @else {
      <img src="..." alt="placeholder" class="camera-feed" />
    }
  </ion-card>

  <!-- Блок показників мікроклімату -->
  <ion-row class="climate-row">
    <!-- Температура -->
    <ion-col size="6">
      <div class="climate-card ion-text-center cursor-pointer"
(click)="openChart('tempSensor', 'Температура')">
        <ion-icon name="thermometer-outline" color="danger"
class="climate-icon"></ion-icon>
        <div class="climate-value">{{ temperature() }}°C</div>
        <div class="climate-label">Температура</div>
      </div>
    </ion-col>
  </ion-row>
```

```
  <!-- Блок: Розклад автоматизацій -->
  <ion-card>
    <ion-accordion-group>
      <ion-accordion value="schedule">
        <div slot="content">
          <ion-list>
            <!-- Ранковий полив -->
            <ion-item>
              <ion-label>Ранковий полив</ion-label>
              <ion-datetime-button datetime="morning-
timer"></ion-datetime-button>

              <ion-modal [keepContentsMounted]="true">
                <ng-template>
                  <ion-datetime
                    id="morning-timer"
                    presentation="time"
                    [value]="polivMorningTime()"
                    (ionChange)="updateSchedule('morningTimer',
$event)">
                  </ion-datetime>
                </ng-template>
              </ion-modal>
            </ion-item>
          </ion-list>
        </div>
      </ion-accordion>
    </ion-accordion-group>
  </ion-card>

  <!-- Панель керування реле -->
  <ion-list inset="true">
    <ion-list-header><ion-label>Керування реле</ion-label></ion-
list-header>
    <ion-item>
      <ion-icon name="sunny-outline" color="warning"
slot="start"></ion-icon>
      <ion-label>Освітлення</ion-label>
      <ion-toggle slot="end" [checked]="isLightOn()"
(ionChange)="toggleRelay('lightSwitch', $event)"
color="warning"></ion-toggle>
    </ion-item>
  </ion-list>
</ion-content>
```

ДОДАТОК З

Конфігурації мобільного дистрибутиву

```
import type { CapacitorConfig } from '@capacitor/cli';

const config: CapacitorConfig = {
  // Унікальний ідентифікатор додатка в екосистемі Android
  appId: 'io.ionic.starter',

  // Назва додатку мовою застосунку
  appName: 'SmartFarm',

  // Директорія, де компілятор Angular розміщує фінальні бандли
  webDir: 'www'
};

export default config;
```

ДОДАТОК І

Каскадна таблиця стилів графіків

```
ion-content {
  --background: #121212;
  --color: #ffffff;
  color: #ffffff;
}

ion-toolbar {
  --background: #1a1a1a;
  --color: #ffffff;
  --border-color: #333;
}

// Перевизначення індикаторів сегментованих кнопок для Світлої
та Темної тем
.filters {
  margin-bottom: 16px;

  ion-segment {
    background: #1e1e1e;
  }
  ion-segment-button {
    color: #ffffff;
    --color: #ffffff;
    --color-checked: #ffffff;
    --indicator-color: #3880ff; /* Блакитний колір активного
елемента */
  }
}

// Тіньовий рівень та ізоляція стилів ApexCharts
::ng-deep .apexcharts-tooltip,
::ng-deep .apexcharts-menu {
  background: #1e1e1e !important;
  border-color: #333 !important;
  color: #fff !important;
}

::ng-deep .apexcharts-tooltip-title {
  background: #2a2a2a !important;
  border-bottom: 1px solid #333 !important;
}
```