Форма № Н-9.02

Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення інформаційних технологій, менеджменту, туризму та підготовки іноземних громадян
(назва відділення)

Циклова комісія комп'ютерної інженерії
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра
(освітньо-професійного ступеня)

на тему: **Розробка мобільного інтерфейсу для комунікації з сервером даних IoT пристроїв**

Виконав: студент IV курсу, групи КІ-412

Спеціальності 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

Владислав ЛІСНИЙ
(ім'я та прізвище)

Керівник Леся ШТОКАЛО
(ім'я та прізвище)

Рецензент _____
(ім'я та прізвище)

Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
імені ІВАНА ПУЛЮЯ»

Відділення **інформаційних технологій, менеджменту, туризму та підготовки іноземних громадян**
Циклова комісія **комп'ютерної інженерії**
Освітньо-професійний ступінь **фаховий молодший бакалавр**
Освітньо-професійна програма: **Обслуговування програмних систем і комплексів**
Спеціальність: **123 Комп'ютерна інженерія**
Галузь знань: **12 Інформаційні технології**

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"30" березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Лісному Владиславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: **Розробка мобільного інтерфейсу для комунікації з сервером даних IoT пристроїв**

керівник роботи Штокало Леся Ярославівна
(прізвище, ім'я, по батькові)

затверджені наказом Відокремленого структурного підрозділу «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя» від 27.03.2026р № 4/9-167.

2. Строк подання студентом роботи: 15 червня 2026 року.

3. Вихідні дані до роботи: методичні вказівки для виконання кваліфікаційної роботи, технічне завдання на розробку програмного забезпечення, мови програмування: PHP REST API + MySQL 8.0 + Docker, JavaScript

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Загальний розділ. Розробка технічного та робочого проєкту. Спеціальний розділ. Економічний розділ. Охорона праці та безпека життєдіяльності.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
   - схема модульної архітектури PWA-застосунку;
   - схема бази даних
   - блок-схема алгоритму механізму Pending Commands;
   - таблиця техніко-економічних показників

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Богдана МАРТИНЮК викладач		
Охорона праці та безпека життєдіяльності	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	01.04	
2	Збір і узагальнення інформації	08.05	
3	Написання першого розділу	15.05	
4	Розробка технічного та робочого проекту	22.05	
5	Написання спеціального розділу	28.05	
6	Розрахунок економічної частини	1.06	
7	Написання розділу охорони праці	3.06	
8	Виконання графічної частини	8.06	
9	Оформлення проєкту	10.06	
10	Погодження нормоконтролю	11.06	
11	Попередній захист роботи	12.06	
12	Захист кваліфікаційної роботи		

7. Дата видачі завдання: 1 квітня 2026 року

Студент _____ Владислав ЛІСНИЙ
(підпис) (ім'я та прізвище)

Керівник роботи _____ Леся ШТОКАЛО
(підпис) ім'я та прізвище)

АНОТАЦІЇ

Лісний В.В. Розробка мобільного прогресивного вебзастосунку для моніторингу та дистанційного керування IoT-пристроями розумної теплиці через REST API сервер: кваліфікаційна робота на здобуття освітньо-професійного ступеня фахового молодшого бакалавра, за спеціальністю 123 Комп'ютерна інженерія. Тернопіль: ВСП «ТФК ТНТУ», 2026. – 131 с.

Метою роботи є розробка мобільного прогресивного вебзастосунку (PWA) для моніторингу та дистанційного керування IoT-пристроями розумної теплиці через REST API сервер. Обґрунтовано вибір технологій (HTML5, CSS3, Vanilla JS ES6+, Fetch API, Canvas API), розроблено архітектуру системи та алгоритм роботи застосунку, реалізовано механізми polling, Pending Commands та PWA-функціональність. Наведено інструкцію з розгортання, методику тестування, економічне обґрунтування та заходи з охорони праці.

Робота містить графічну частину на 4 аркушах А1 та пояснювальну записку обсягом 131 аркуш, що містить 6 таблиць і 7 рисунків.

Ключові слова: IoT, розумна теплиця, PWA, прогресивний вебзастосунок, REST API, ESP32, моніторинг мікроклімату, Vanilla JS, Service Worker, охорона праці.

Lisnyi V.V. Development of a mobile progressive web application for monitoring and remote control of smart greenhouse IoT devices via REST API server: qualification thesis for the degree of Professional Junior Bachelor, specialty 123 Computer Engineering. Ternopil: VSP "TFK TNTU", 2026. – 131 p.

The aim of the work is to develop a mobile progressive web application (PWA) for monitoring and remote control of smart greenhouse IoT devices via a REST API server. The selection of technologies (HTML5, CSS3, Vanilla JS ES6+, Fetch API, Canvas API) is justified, the system architecture and application algorithm are developed, and polling, Pending Commands, and PWA functionality mechanisms are implemented. Deployment instructions, a testing methodology, economic justification, and occupational safety measures are provided.

The work includes a graphic section of 4 A1 sheets and an explanatory note of 131 pages containing 6 tables and 7 figures.

Keywords: IoT, smart greenhouse, PWA, progressive web application, REST API, ESP32, microclimate monitoring, Vanilla JS, Service Worker, occupational safety.

ЗМІСТ

ВСТУП 8

1 ЗАГАЛЬНИЙ РОЗДІЛ 9

1.1.1 Огляд платформи Node-RED 10

1.1.2 Огляд платформи Home Assistant 11

1.1.3 Огляд платформи ThingsBoard 12

1.1.4 Нативні мобільні фреймворки 12

1.1.5 Підхід на основі PWA 13

1.1 Технічне завдання 15

1.2.1 Найменування та галузь застосування 15

1.2.2 Підстава для розробки 15

1.2.3 Призначення розробки 16

1.2.4 Вимоги до програмного продукту 17

1.2.5 Вимоги до програмної документації 18

1.2.6 Стадії та етапи розробки 19

1.2.7 Порядок контролю та приймання 19

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ 21

2.1 Аналіз технічного завдання та вибір методів розробки 21

2.2 Обґрунтування вибору технологій та інструментів 23

2.3 Проєктування структурної схеми системи 26

2.4 Розробка алгоритму роботи застосунку 29

2.5 Реалізація функціональних модулів 33

2.5.1 Модуль дашборду та відображення сенсорів 33

2.5.2 Модуль керування актуаторами 35

2.5.3 Модуль графіків 37

2.5.4 Модуль сповіщень та налаштувань 38

2.6 Система автентифікації та захисту даних 39

2.7 Опис інтерфейсу користувача 41

3. СПЕЦІАЛЬНИЙ РОЗДІЛ 43

3.1 Підключення PWA до розгорнутого IoT-сервера 43

3.2 Методологія тестування мобільного інтерфейсу 46

3.2.1 Функціональне тестування 46

3.2.2 Інтеграційне тестування 48

3.2.3 Тестування на реальних пристроях 49

3.2.4 Аудит PWA-якості та безпеки 50

4. ЕКОНОМІЧНА ЧАСТИНА 52

4.1 Мета та завдання економічного розрахунку 52

4.2 Визначення трудомісткості розробки 52

4.3 Розрахунок витрат на оплату праці 54

4.4 Відрахування на соціальні заходи 55

4.5 Витрати на електроенергію 57

4.6 Накладні витрати 57

4.7 Собівартість розробки 58

4.8 Ціна програмного продукту 59

4.9 Техніко-економічні показники 59

5. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ 61

5.1 Ергономічне проєктування робочого середовища та вимоги до безпечності застосування інформаційного обладнання 61

5.2. Психофізіологічна безпека розробника при високому інформаційному навантаженні 63

5.3. Класифікація умов праці програміста та нормування освітлення виробничих приміщень 65

ВИСНОВКИ 69

ПЕРЕЛІК ПОСИЛАНЬ 71

ВСТУП

Сучасний розвиток цифрових технологій, насамперед у сфері Інтернету речей (IoT), відкриває широкі можливості для автоматизації аграрного виробництва. Системи моніторингу та управління мікрокліматом у закритих ґрунтах – теплицях, оранжереях, вертикальних фермах – дозволяють підвищувати врожайність на 30–50% при одночасному зниженні витрат ресурсів (вода, добрива, електроенергія) та залученості людини до рутинних операцій. Однак технологічна ефективність будь-якої IoT-системи прямо пропорційна якості інтерфейсу, через який оператор взаємодіє з нею в режимі реального часу.

Актуальність теми. В умовах, коли обслуговуючий персонал теплиць рідко має постійний доступ до стаціонарного комп'ютера, ключового значення набуває можливість моніторингу та управління системою безпосередньо зі смартфону або планшету. Традиційні рішення (нативні мобільні застосунки) потребують проходження процедур публікації в магазинах AppStore/Google Play, ліцензування, а для кожного оновлення – повторного завантаження. Прогресивні вебзастосунки (PWA) усувають ці обмеження, поєднуючи переваги нативних мобільних додатків із простотою розгортання та оновлення вебсайтів. Проблема ефективного проєктування PWA-інтерфейсу для IoT-систем з урахуванням вимог до надійності зв'язку, безпеки передачі даних та адаптивного відображення є актуальною науково-практичною задачею.

Мета і завдання дослідження. Метою кваліфікаційної роботи є розробка мобільного прогресивного вебзастосунку для моніторингу та дистанційного керування IoT-пристроями розумної теплиці через REST API сервер.

Для досягнення мети поставлено такі конкретні завдання:

– провести аналітичний огляд існуючих рішень у сфері інтерфейсів для IoT-систем та обґрунтувати вибір підходу;

– сформувати технічне завдання на розробку PWA відповідно до вимог замовника;

– спроєктувати архітектуру застосунку та схему взаємодії з REST API;

– реалізувати функціональні модулі: автентифікацію, polling, керування актуаторами, Canvas-візуалізацію сенсорних даних;

– реалізувати механізм Pending Commands для підвищення відгуку інтерфейсу;

– налаштувати PWA-функціональність (manifest.json, Service Worker, мета-теги);

– підключити застосунок до розгорнутого Docker IoT-сервера та провести комплексне тестування;

– виконати техніко-економічне обґрунтування розробки;

– розглянути питання охорони праці та безпеки.

Об'єктом дослідження є процес розробки клієнтського мобільного вебінтерфейсу для IoT-системи моніторингу та керування параметрами мікроклімату розумної теплиці.

Предметом дослідження є методи та засоби побудови прогресивних вебзастосунків із використанням стандартних вебтехнологій (HTML5, CSS3, JavaScript, Fetch API) для інтеграції з REST-серверами IoT-пристроїв та забезпечення надійної, безпечної та зручної комунікації між оператором і мережею IoT-пристроїв.

Методи дослідження. У роботі використано: методи системного аналізу (під час огляду існуючих рішень); порівняльний аналіз (під час вибору технологій); методи об'єктно-орієнтованого та модульного проєктування (під час розробки архітектури); методи функціонального, інтеграційного та юзабіліті тестування програмного забезпечення (під час верифікації).

Практичне значення роботи. Розроблений PWA-застосунок є готовим функціональним продуктом, що не потребує встановлення та підтримує роботу на будь-якому сучасному мобільному або настільному браузері. Застосунок може слугувати масштабованою базою для управління мережею теплиць із підтримкою кількох пристроїв (параметр device_id) та різних рівнів доступу (система API-ключів).

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень у сфері IoT-інтерфейсів

Для обґрунтованого вибору підходу до розробки мобільного інтерфейсу системи моніторингу розумної теплиці необхідно провести аналіз наявних рішень у цій сфері. Сучасний ринок програмного забезпечення для IoT пропонує широкий спектр платформ, фреймворків і підходів – від готових хмарних сервісів до написання власних застосунків. Кожне з існуючих рішень має свій набір переваг і обмежень, що визначають сферу його оптимального застосування.

Критерії оцінки рішень. Під час аналізу використовувалися такі критерії: мобільна адаптивність та якість UX на смартфонах; складність розгортання та підтримки; гнучкість кастомізації інтерфейсу; можливість інтеграції з власним REST API; вимоги до серверних ресурсів; наявність підтримки офлайн-режиму; ліцензійні обмеження та вартість.

1.1.1 Огляд платформи Node-RED

Node-RED – це платформа візуального програмування з відкритим кодом, початково розроблена IBM для побудови IoT-потоків обробки даних. Вона базується на Node.js і використовує браузерний редактор з підходом flow-based programming (програмування потоками), де вузли з'єднуються між собою стрілками для опису логіки обробки подій.

Модуль node-red-dashboard розширює можливості Node-RED додаванням UI-компонентів (індикатори, графіки, кнопки, форми), що відображаються у браузері як веб-дашборд. Платформа добре інтегрується з протоколами MQTT, HTTP, WebSocket і надає широку бібліотеку готових вузлів для роботи з різноманітними IoT-протоколами та хмарними сервісами.

Переваги Node-RED для IoT-застосунків: низький поріг входу – оператори без досвіду програмування можуть будувати прості потоки обробки; швидке

прототипування логіки обробки подій; вбудована підтримка MQTT і сотні готових вузлів у бібліотеці npm; активна спільнота та широка документація.

Обмеження Node-RED для мобільних інтерфейсів: модуль dashboard надає обмежений набір фіксованих UI-компонентів без можливості глибокої кастомізації стилів; відсутня нативна підтримка PWA (немає manifest.json, Service Worker); адаптивність мобільного інтерфейсу обмежена; для складних кастомних інтерфейсів (наприклад, canvas-графіки, множинні пристрої) потрібне значне доопрацювання; Node.js сервер потребує окремого розгортання та обслуговування.

1.1.2 Огляд платформи Home Assistant

Home Assistant – відкрита система домашньої автоматизації, що спочатку розроблялася для смарт-будинків, але набула широкого поширення у промислових та сільськогосподарських IoT-системах завдяки широкій екосистемі інтеграцій (понад 3000 офіційних). Платформа написана на Python і надає потужну систему Lovelace UI для побудови кастомних дашбордів через YAML/JSON конфігурацію.

Home Assistant підтримує встановлення нативних мобільних застосунків для iOS і Android із push-сповіщеннями, геолокацією та сенсорами пристрою. Платформа може бути інтегрована з IoT-пристроями через MQTT, Zigbee, Z-Wave, HTTP та десятки інших протоколів.

Переваги Home Assistant: найширша екосистема інтеграцій у своєму класі; гнучкий Lovelace UI з кастомними картками; нативні мобільні застосунки; вбудована підтримка push-сповіщень; активна спільнота та нові функції кожного місяця.

Обмеження для спеціалізованого проєкту: мінімальні вимоги – 2 ГБ RAM, 32 ГБ SSD, що надмірно для вузькоспеціалізованої системи; складність початкового налаштування та підтримки; оновлення іноді порушують зворотну сумісність конфігурацій; більшість потужних функцій (розширений UI, кастомні

карти) потребує встановлення додаткових компонентів через HACS; відсутність PWA-підтримки в стандартній конфігурації.

1.1.3 Огляд платформи ThingsBoard

ThingsBoard – enterprise-рішення з відкритим кодом для IoT, спроєктоване для масштабних промислових розгортань. Платформа написана на Java/Spring Boot і підтримує MQTT, HTTP, CoAP та LwM2M протоколи. ThingsBoard надає конструктор дашбордів з широким набором віджетів (карти, графіки, таблиці, аналоговий індикатор), а також REST API для зовнішньої інтеграції.

Хмарна версія ThingsBoard Community Edition доступна безкоштовно, проте має обмеження: до 3 пристроїв, обмежена кількість точок даних на місяць, відсутність підтримки кластерного розгортання.

Переваги ThingsBoard: промисловий масштаб; широкий набір вбудованих віджетів; підтримка багатьох IoT-протоколів; інструменти аналітики та правил обробки подій; мобільний застосунок ThingsBoard Mobile.

Обмеження для даного проєкту: мінімальні системні вимоги 4 ГБ RAM, Java 11+; час запуску контейнера > 60 секунд; надмірна складність для системи з одним пристроєм ESP32; обмеження Community Edition; складність кастомізації UI без знання Angular.

1.1.4 Нативні мобільні фреймворки

Нативні та кросплатформні мобільні фреймворки (React Native, Flutter, Ionic, Xamarin) забезпечують найвищу продуктивність і найкращу інтеграцію з апаратними можливостями пристрою (камера, сенсори, Bluetooth, push-сповіщення). Flutter від Google дозволяє зібрати застосунок для iOS, Android, вебу і десктопу з єдиної Dart-кодової бази. React Native від Meta використовує компоненти JavaScript, що рендеряться у нативний UI.

Однак для IoT-системи з обмеженою аудиторією ці підходи мають серйозні недоліки: необхідність проходження верифікації та публікації в AppStore (від 1 до 7 днів) та Google Play; обов'язкове оновлення застосунку в магазині при кожній зміні функціональності; вимоги Apple Developer Program ($99/рік) та Google Play Console ($25 одноразово); складна процедура дистрибуції серед обмеженої аудиторії без публічної публікації (TestFlight, App Distribution); значно більший обсяг кодової бази та вищі вимоги до компетенції розробника.

1.1.5 Підхід на основі PWA

Прогресивний вебзастосунок (PWA) – це вебзастосунок, що використовує сучасні вебтехнології для забезпечення функціональності, аналогічної нативним мобільним застосункам. Концепція PWA базується на трьох принципах: надійність (офлайн-режим через Service Worker), швидкість (продуктивне завантаження та відгук) і залученість (встановлення на домашній екран, push-сповіщення).

Service Worker – це JavaScript-скрипт, що виконується у фоновому потоці окремо від основної сторінки та дозволяє перехоплювати мережеві запити, кешувати ресурси та забезпечувати офлайн-функціональність. Web App Manifest – JSON-файл, що описує застосунок для браузера та ОС: назву, іконки, кольори теми, режим відображення (standalone, fullscreen).

Переваги PWA для IoT-інтерфейсу: розгортання та оновлення – єдина точка (сервер, без магазинів додатків); доступ з будь-якого браузера без встановлення; встановлення на домашній екран за одним кліком; офлайн-доступ до кешованих даних при втраті з'єднання; єдина кодова база для всіх платформ (Android, iOS, Windows, macOS); мінімальні вимоги до ресурсів сервера; розробка стандартними вебтехнологіями без спеціалізованих фреймворків.

За результатами проведеного огляду проведено порівняльний аналіз ключових характеристик розглянутих рішень за девятьма критеріями.

Node-RED Dashboard: мобільна адаптивність – обмежена (UI не оптимізований для малих екранів); складність розгортання – середня; кастомізація UI – обмежена вбудованими вузлами; REST API інтеграція – середня; офлайн-режим – відсутній; системні вимоги – середні; ліцензія – безкоштовно; оновлення UI – потребує перезапуску; крос-браузерність – так.

Home Assistant: мобільна адаптивність – висока; складність розгортання – висока (потребує окремого сервера, 2 ГБ RAM+); кастомізація UI – середня (Lovelace-картки); REST API інтеграція – висока; офлайн-режим – частковий; системні вимоги – значні; ліцензія – безкоштовно; оновлення UI – потребує рекомпіляції конфігурації; крос-браузерність – часткова.

ThingsBoard Community Edition: мобільна адаптивність – висока; складність розгортання – дуже висока (JVM, PostgreSQL, 4 ГБ RAM+); кастомізація UI – середня; REST API інтеграція – висока; офлайн-режим – відсутній; системні вимоги – великі; ліцензія – обмежений безкоштовний план; оновлення UI – рекомпіляція; крос-браузерність – так.

Нативний мобільний застосунок (React Native / Flutter): мобільна адаптивність – висока; складність розгортання – висока (публікація в App Store та Google Play); кастомізація UI – повна; REST API інтеграція – повна; офлайн-режим – повний; ліцензія – $99–125/рік (Apple Developer Program); оновлення UI – публікація нової версії в магазині (1–3 дні); крос-браузерність – ні (потрібні два застосунки).

Власна PWA-розробка: мобільна адаптивність – висока; складність розгортання – низька (один HTML-файл на вебсервері); кастомізація UI – повна; REST API інтеграція – повна; офлайн-режим – повний (Service Worker); системні вимоги – мінімальні; ліцензія – безкоштовно; оновлення UI – миттєво (оновлення файлу на сервері); крос-браузерність – так.

Аналіз показує, що підхід на основі власного PWA є найбільш оптимальним для реалізації спеціалізованого інтерфейсу розумної теплиці. Він забезпечує повний контроль над дизайном і функціональністю, мінімальні

вимоги до серверних ресурсів, просте розгортання, крос-платформність та підтримку офлайн-режиму.

Вибір між власною розробкою та адаптацією готових рішень також обґрунтовується тим, що жодне з розглянутих рішень не підтримує нативно описаний в технічному завданні протокол взаємодії (PHP REST API з автентифікацією через X-API-Key заголовок і параметром device_id). Адаптація будь-якого готового рішення для підтримки цього специфічного API потребуватиме не менших зусиль, ніж написання власного PWA.

1.1 Технічне завдання

1.2.1 Найменування та галузь застосування

Найменування програмного продукту: «SmartGreenhouse PWA» – мобільний прогресивний вебзастосунок для моніторингу та дистанційного керування розумною теплицею через IoT-сервер.

Галузь застосування: автоматизація сільськогосподарського виробництва; системи «розумного» землеробства та прецизійного землеробства; дистанційний моніторинг та управління мікрокліматом у закритих ґрунтах (теплицях, оранжереях, вертикальних фермах). Застосунок може використовуватись як на виробничих підприємствах агросектору, так і в навчальних лабораторіях для демонстрації принципів IoT-систем.

1.2.2 Підстава для розробки

Підставою для розробки є завдання на кваліфікаційну роботу та оформлене відповідно до навчального плану. Необхідність розробки обумовлена відсутністю готового спеціалізованого мобільного інтерфейсу для системи моніторингу розумної теплиці на базі ESP32, яка є частиною комплексного командного проєкту.

Розробка ведеться у рамках навчально-виробничого проєкту і базується на вже розробленому та розгорнутому backend-рішенні (PHP REST API + MySQL 8.0 + Docker). Кваліфікаційна робота охоплює виключно клієнтський рівень системи та протокол взаємодії з API.

1.2.3 Призначення розробки

Програмний продукт призначений для надання операторам розумної теплиці зручного та надійного інструменту дистанційного контролю та управління через будь-який сучасний браузер без необхідності встановлення додаткового програмного забезпечення.

Застосунок реалізує такі функціональні можливості:

– відображення в режимі реального часу поточних показників сенсорів: температура (°C), вологість повітря (%), освітленість (lux), вологість ґрунту у трьох зонах (%);

– відображення поточного стану виконавчих пристроїв: вентилятор (увімкн./вимкн.), лампа освітлення (увімкн./вимкн.), три сервоприводи поливу (відкрито/закрито);

– дистанційне керування виконавчими пристроями через REST API;

– перегляд архіву показників у вигляді лінійних графіків із вибором часового проміжку (1 год / 6 год / 24 год / 7 днів);

– налаштування порогових значень для автоматичних сповіщень про відхилення;

– управління кількома пристроями (перемикання через параметр device_id);

– реєстрація та автентифікація користувачів, управління API-ключами;

– встановлення застосунку на домашній екран мобільного пристрою (PWA install);

– часткова офлайн-функціональність через Service Worker (кешування UI-ресурсів).

1.2.4 Вимоги до програмного продукту

Функціональні вимоги до застосунку:

1) Підтримка автентифікації через API-ключі, що передаються у HTTP-заголовку X-API-Key. Ключ не повинен включатися до URL-параметрів.

2) Механізм циклічного опитування сервера (polling) з налаштовуваним інтервалом від 1 до 60 секунд; значення за замовчуванням – 5 секунд. Перший запит виконується негайно при завантаженні сторінки.

3) Механізм Pending Commands: команди керування актуаторами відображаються в UI оптимістично (стан pending) протягом 15 секунд до отримання підтвердження від сервера. При таймауті відображається попередження.

4) Всі налаштування (URL сервера, API-ключ, інтервал опитування, ідентифікатор пристрою) зберігаються у localStorage браузера та автоматично відновлюються при наступному відкритті.

5) PWA-функціональність: manifest.json з визначеними name, short_name, theme_color (#0f1a0f), background_color, іконками 192×192 та 512×512 px та display: "standalone"; мета-теги для iOS (apple-mobile-web-app-capable, apple-mobile-web-app-title); мета-тег theme-color для Android.

6) Відображення Canvas-графіків для кожного сенсора з вибором часового проміжку без використання зовнішніх бібліотек.

7) Підтримка управління кількома пристроями: ідентифікатор пристрою підставляється у кожен API-запит через GET-параметр device_id.

Нефункціональні вимоги:

1) Час першого відгуку інтерфейсу на дії користувача (First Input Delay) – не більше 100 мс.

2) Час завантаження застосунку при повторному відкритті (кеш) – не більше 1 секунди.

3) Сумісність: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+, мобільні браузери Chrome for Android та Safari for iOS.

4) Адаптивний дизайн для екранів від 320 px (мобільні) до 2560 px (великі монітори).

5) Код написано на чистому JavaScript (Vanilla JS, ES6+) без зовнішніх фреймворків для мінімізації залежностей.

6) Вся бізнес-логіка та зберігання даних знаходяться на сервері; клієнт є «тонким» та не виконує самостійних розрахунків на основі кешованих даних.

7) Розмір завантажуваного HTML-файлу (включаючи вбудовані стилі та скрипти) – не більше 100 КБ.

1.2.5 Вимоги до програмної документації

У складі кваліфікаційної роботи розробляються та надаються такі документи:

– пояснювальна записка (розрахунково-пояснювальна записка кваліфікаційної роботи) – основний текстовий документ;

– технічне завдання (підрозділ 1.2 пояснювальної записки) – відповідно до ДСТУ 3008:2015;

– функціональна схема системи (Рисунок 2.1) – загальна архітектура;

– схема модульної архітектури PWA (Рисунок 2.2);

– схема бази даних MySQL (Рисунок 2.3);

– діаграма варіантів використання (Рисунок 2.4);

– алгоритм механізму Pending Commands (Рисунок 2.5);

– зовнішній вигляд інтерфейсу, рисунки 2.6–2.7;

– протоколи тестування тест-кейсів (підрозділ 3.2).

Документація виконується відповідно до ДСТУ 3008:2015 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення» та методичних вказівок коледжу.

1.2.6 Стадії та етапи розробки

Розробка виконується в такі стадії та етапи:

1) Технічне завдання (2 тижні) – аналіз вимог, огляд аналогів, формування ТЗ.

2) Ескізний проєкт (1 тиждень) – визначення архітектури, прототипування UI, вибір технологій.

3) Технічний проєкт (2 тижні) – детальне проєктування модулів, схем, алгоритмів.

4) Робочий проєкт (3 тижні) – реалізація HTML/CSS/JS, інтеграція з API, налаштування PWA.

5) Тестування (1 тиждень) – функціональне, інтеграційне, крос-браузерне тестування, аудит Lighthouse.

6) Впровадження (1 тиждень) – розгортання, налаштування, підготовка документації.

7) Захист (1 тиждень) – підготовка та захист кваліфікаційної роботи.

1.2.7 Порядок контролю та приймання

Контроль якості та приймання програмного продукту здійснюється у такому порядку:

1) Перевірка відповідності функціональних можливостей вимогам ТЗ: виконання всіх тест-кейсів з результатом «Пройдено».

2) Аудит якості PWA за допомогою Google Lighthouse: категорія PWA – 100/100, Performance – не менше 85/100.

3) Перевірка роботи на реальних пристроях: мінімально – Android Chrome та iOS Safari.

4) Перегляд вихідного коду на відповідність вимозі відсутності зовнішніх залежностей (Vanilla JS).

5) Перевірка безпеки: API-ключ не присутній в URL-параметрах жодного запиту.

6) Підписання акту приймання-передачі керівником кваліфікаційної роботи.

Програмний продукт вважається прийнятим після успішного виконання всіх перерахованих перевірок та оформлення відповідних документів.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Аналіз технічного завдання та вибір методів розробки

На основі технічного завдання та результатів аналітичного огляду існуючих рішень визначено ключові методи, підходи та архітектурні рішення для побудови мобільного інтерфейсу.

Архітектурна концепція: «тонкий клієнт». Основним архітектурним рішенням обрано принцип «тонкого клієнта» (thin client), за яким вся бізнес-логіка, зберігання та обробка даних сенсорів, автоматичні дії (налаштування режиму, автоматичне управління за порогами) зосереджені на сервері. Клієнтський застосунок відповідає виключно за відображення даних у зручному вигляді та маршрутизацію команд від оператора до API.

Обґрунтування вибору тонкого клієнта:

– мінімальний розмір клієнтського файлу (один HTML-файл ~42 КБ без зовнішніх залежностей);

– відсутність ризику розходження логіки між клієнтом і сервером (single source of truth);

– просте оновлення: зміна серверного API автоматично відображається на всіх підключених клієнтах;

– безпека: конфіденційні обчислення (перевірка порогів, генерація алертів) виконуються на сервері;

– масштабованість: один клієнт може підключатися до різних серверів без перекомпіляції.

Вибір механізму синхронізації даних. Для отримання актуальних даних з сервера розглядалися три підходи: polling (циклічне опитування), Server-Sent Events (SSE) та WebSocket.

HTTP Polling (циклічне опитування): серверна підтримка – будь-який HTTP-сервер без додаткових розширень; складність реалізації – мінімальна (setInterval); підтримка брандмауерів та проксі – повна; затримка оновлення –

рівна інтервалу (5 с); навантаження на сервер – помірне (1 запит кожні 5 с); відновлення при втраті зєднання – автоматично (наступний запит за інтервалом). Підходить для даних теплиці: так.

Server-Sent Events (SSE): серверна підтримка – PHP потребує stream_set_blocking(0); складність реалізації – середня; підтримка брандмауерів – часткова; затримка оновлення – моментальна; навантаження на сервер – низьке (постійне зєднання); відновлення при втраті зєднання – потребує EventSource reconnect. Підходить для даних теплиці: так, але надлишково.

WebSocket: серверна підтримка – PHP потребує бібліотеки Ratchet або ReactPHP; складність реалізації – висока; підтримка брандмауерів – обмежена; затримка оновлення – моментальна; навантаження на сервер – низьке; відновлення при втраті зєднання – потребує власної реалізації. Підходить для даних теплиці: так, але надлишково.

Висновок: для агрокліматичних параметрів, що змінюються відносно повільно і не потребують субсекундної реакції, механізм HTTP polling з інтервалом 5 секунд є оптимальним з огляду на простоту реалізації та сумісність із будь-яким PHP-сервером без додаткових розширень. WebSocket і SSE є надлишковими рішеннями для даного сценарію і вносять ускладнення без практичного покращення UX.

Вибір JavaScript-підходу. Розглядалися три варіанти: нативний JavaScript (Vanilla JS), фреймворки (React, Vue) та компіляційні підходи (TypeScript + bundler). Обрано Vanilla JS з таких міркувань:

– відсутність залежностей від зовнішніх пакетів (npm) – жодного node_modules, жодного webpack;

– мінімальний розмір бандла: весь застосунок – один HTML-файл без зовнішніх ресурсів;

– повна підтримка Web APIs: Fetch API, Canvas API, localStorage, Service Worker, Web Manifest – все доступне нативно;

– відсутність ризику застарівання фреймворку: застосунок на Vanilla JS буде працювати без змін через 10 років;

– відповідність вимозі ТЗ п.5: «код написано на Vanilla JS без фреймворків».

2.2 Обґрунтування вибору технологій та інструментів

Стек технологій розробленого застосунку формувався з урахуванням вимог технічного завдання, необхідності забезпечення кросплатформності та мінімальних вимог до середовища виконання. Нижче наведено детальне обґрунтування вибору кожного компонента.

HTML5. Використовуються такі можливості HTML5:

– семантична розмітка (<header>, <main>, <section>, <nav>, <article>) для покращення доступності та читабельності коду;

– елемент <canvas> для Canvas-рендерингу графіків без зовнішніх бібліотек;

– елемент <template> для динамічного клонування UI-компонентів;

– атрибут type="module" для підключення Service Worker;

– мета-теги PWA: <meta name="apple-mobile-web-app-capable" content="yes">, <meta name="apple-mobile-web-app-title">, <meta name="theme-color" content="#0f1a0f">;

– <link rel="manifest" href="manifest.json"> для підключення PWA-маніфесту.

CSS3. Ключові CSS-функції, використані в проєкті:

– CSS Custom Properties (змінні) для централізованого управління темою: --bg: #0f1a0f (глибокий темно-зелений фон), --card: #1a2e1a (фон карток), --accent: #4caf50 (акцентний зелений), --text: #e8f5e9 (основний текст), --warning: #ff9800 (попередження), --danger: #f44336 (помилки);

– CSS Grid з auto-fill і minmax для адаптивного розміщення карток сенсорів: grid-template-columns: repeat(auto-fill, minmax(160px, 1fr));

– CSS Flexbox для вирівнювання елементів всередині карток та навігаційної панелі;

– CSS Transitions для анімацій зміни стану кнопок (переход між станами inactive/pending/active);

– media queries для адаптивної зміни розміру тексту та компонування при ширині < 600 px;

– backdrop-filter для ефекту розмиття у модальних вікнах.

JavaScript ES6+. Використовуються сучасні можливості:

– async/await для зручної роботи з асинхронними Fetch-запитами;

– деструктуризація об'єктів та масивів;

– template literals для формування URL та HTML-рядків;

– Arrow functions для callback-функцій;

– Spread operator для об'єднання об'єктів (заголовки запитів);

– Optional chaining (?.) для безпечного звернення до вкладених властивостей відповіді API;

– Nullish coalescing (??) для встановлення значень за замовчуванням.

Fetch API. Базовий механізм HTTP-комунікації застосунку. Всі запити проходять через обгортку apiFetch(), яка автоматично додає необхідні заголовки:

```
function apiFetch(url, options = {}) {
  options.headers = {
    ...options.headers,
    'ngrok-skip-browser-warning': '69420'
  };
  if (authApiKey) {
    options.headers['X-API-Key'] = authApiKey;
  }
  return fetch(url, options);
}
```

Заголовок ngrok-skip-browser-warning служить для обходу HTML-заглушки, яку ngrok відображає при прямому доступі до тунельованого URL у браузері. Без цього заголовка перший запит поверне HTML-сторінку з попередженням замість JSON-відповіді API.

Canvas API. Побудова графіків реалізована через пряме малювання на HTML-елементі <canvas> без бібліотек (Chart.js, D3.js). Це рішення забезпечує відсутність зовнішніх залежностей, малий розмір файлу та повний контроль над відображенням. Базовий алгоритм рендерингу графіка:

```
function drawChart(canvas, labels, values, color) {
  const ctx = canvas.getContext("2d");
  const W = canvas.width, H = canvas.height;
  const padL=40, padR=10, padT=10, padB=30;
  ctx.clearRect(0, 0, W, H);
  const min=Math.min(...values), max=Math.max(...values);
  const range = max - min || 1;
  // Draw grid lines
  ctx.strokeStyle = "#2a4a2a"; ctx.lineWidth = 1;
  for (let i=0; i<=4; i++) {
    const y = padT + (H-padT-padB)*i/4;
    ctx.beginPath(); ctx.moveTo(padL,y); ctx.lineTo(W-padR,y);
    ctx.stroke();
  }
  // Draw data line
  ctx.strokeStyle = color; ctx.lineWidth = 2;
  ctx.beginPath();
  values.forEach((v, i) => {
    const x = padL+(W-padL-padR)*i/(values.length-1);
    const y = H-padB-(v-min)/range*(H-padT-padB);
    i===0 ? ctx.moveTo(x,y) : ctx.lineTo(x,y);
  });
  ctx.stroke();
}
```

Web App Manifest. Файл manifest.json описує застосунок для браузера та ОС, дозволяючи встановлювати PWA на домашній екран. Ключові поля:

```
{
 "name": "SmartGreenhouse",
 "short_name": "Greenhouse",
 "display": "standalone",
 "background_color": "#0f1a0f",
 "theme_color": "#0f1a0f",
 "start_url": "./app.html",
 "icons": [
  { "src":"icon-192.png","sizes":"192x192","type":"image/png" },
  { "src":"icon-512.png","sizes":"512x512","type":"image/png" }
 ]
}
```

Режим standalone означає, що встановлений PWA запускається як окремий застосунок без браузерного chrome (адресного рядка, кнопок навігації), що забезпечує нативне відчуття.

Інструменти розробки: Visual Studio Code з розширеннями Prettier та ESLint; Chrome DevTools (вкладки Console, Network, Application для інспекції Service Worker та localStorage); Google Lighthouse для автоматичного аудиту PWA-якості; ngrok для розгортання тимчасового HTTPS-тунелю для тестування PWA на реальних пристроях (Safari на iOS вимагає HTTPS для Service Worker та PWA install).

2.3 Проєктування структурної схеми системи

Архітектура розробленого рішення є трирівневою і включає: рівень IoT-пристроїв (ESP32), рівень серверного API (PHP + MySQL + Docker) та рівень клієнтського застосунку (PWA). Даний розділ описує всі рівні з акцентом на клієнтський та протокол взаємодії між рівнями.

Рівень IoT-пристрою. ESP32 – двоядерний мікроконтролер з частотою 240 МГц, 520 КБ SRAM, вбудованим Wi-Fi (802.11 b/g/n) і Bluetooth. До

мікроконтролера підключені: DHT22 (датчик температури та вологості повітря), BH1750 (датчик освітленості), три ємнісних датчики вологості ґрунту, вентилятор (через реле), лампа (через реле), три сервоприводи клапанів поливу. Кожні 30 секунд ESP32 надсилає POST-запит до API (endpoint sensor_data) з JSON-об'єктом, що містить поточні показники. Після відправки він запитує чергу команд (get_commands) і виконує їх (вмикає/вимикає актуатори).

На рисунку 2.1 зображена загальна структурна схема системи SmartGreenhouse:

Рисунок 2.1 – Загальна структурна схема системи SmartGreenhouse

Рівень серверного API. PHP-файл api.php (656 рядків) реалізує REST API з такими групами ендпоінтів:

– отримання даних: dashboard, sensor_data, get_commands, history, alerts;

– керування: command, set_mode, settings, update_setting, mark_alerts_read;

– автентифікація: login, register, generate_key, list_keys, revoke_key.

Бізнес-логіка API: автоматичний аналіз показників при кожному записі sensor_data – якщо значення виходить за порогові межі, генерується запис у таблиці alerts; команди від клієнта зберігаються в чергу (таблиця commands) зі статусом pending; ESP32 забирає їх через get_commands та підтверджує виконання (статус executed). Черга team's черга команд реалізує асинхронну взаємодію між оператором (через PWA) та пристроєм (ESP32), що опитує API кожні 30 секунд.

Рівень клієнтського PWA. Застосунок реалізовано як єдиний HTML-файл (app.html) із вбудованими стилями та скриптами. Архітектурно він складається з семи функціональних модулів, кожен з яких відповідає за окрему область функціональності. Модулі не є окремими файлами або класами – вони

реалізовані як пов'язані функції в єдиному глобальному scope, що є типовим підходом для малих Vanilla JS застосунків (див. рис. 2.2).

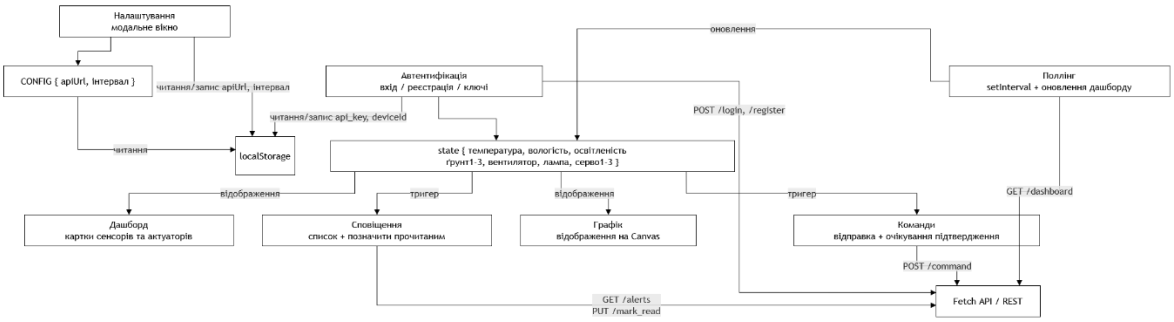


Рисунок 2.2 – Схема модульної архітектури PWA-застосунку

Центральним елементом клієнтської архітектури є глобальний об'єкт state, що зберігає останній відомий стан усіх сенсорів та актуаторів:

```
let state = {
  temperature: null, humidity: null, lux: null,
  soil1: null, soil2: null, soil3: null,
  fan: false, lamp: false,
  servo1: false, servo2: false, servo3: false
};
```

Об'єкт CONFIG ініціалізується при завантаженні сторінки значеннями з localStorage. Якщо в localStorage відсутній URL API, він визначається автоматично відносно поточного розміщення HTML-файлу:

```
let CONFIG = {
  apiUrl: localStorage.getItem("apiUrl") ||
    (window.location.origin +
     window.location.pathname.replace(/\/[^\/]*$/, "/api.php")),
  interval: parseInt(localStorage.getItem("interval")) || 5
};
let currentDeviceId = localStorage.getItem("deviceId") || "greenhouse_01";
let authApiKey = localStorage.getItem("authApiKey") || "";
```

Функція apiUrl() формує повний URL для кожного типу запиту, автоматично додаючи device_id та додаткові параметри:

```
function apiUrl(action, extra) {
  return CONFIG.apiUrl + "?action=" + action +
    "&device_id=" + encodeURIComponent(currentDeviceId) +
    (extra || "");
}
```

Схема бази даних включає шість основних таблиць, з якими взаємодіє клієнт через REST API (див. рис. 2.3):

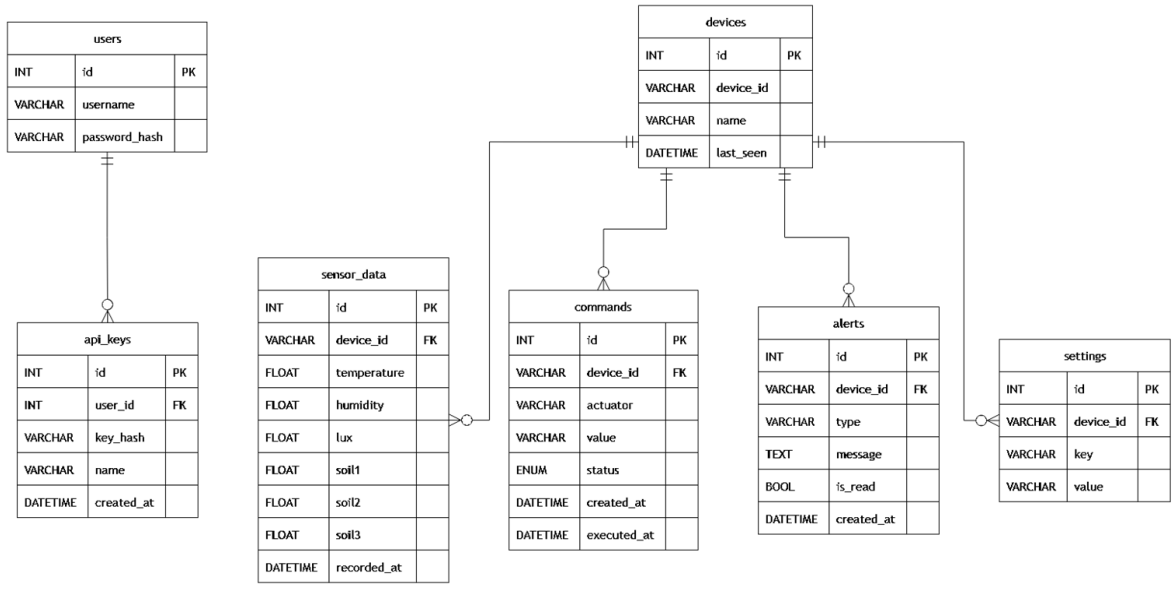


Рисунок 2.3 – Схема бази даних MySQL: таблиці users

2.4 Розробка алгоритму роботи застосунку

Алгоритм роботи застосунку охоплює три основні сценарії: ініціалізація при першому запуску, регулярний цикл опитування даних (polling) та відправка команди керування актуатором. Кожен сценарій описується детально нижче.

Алгоритм ініціалізації застосунку виконується один раз при відкритті або оновленні сторінки:

1) Завантаження конфігурації: зчитати apiUrl, interval, authApiKey, currentDeviceId з localStorage.

2) Перевірка наявності API-ключа: якщо authApiKey порожній – перейти до кроку 3 (вхід); інакше – до кроку 4.

3) Відображення форми входу/реєстрації (auth-modal). Очікування дій користувача.

4) Перевірка з'єднання: виконати GET ?action=dashboard. При успіху – крок 5; при помилці 401 – крок 3; при мережевій помилці – відобразити повідомлення про недоступність сервера та форму налаштувань.

5) Ініціалізація UI: заповнити дашборд поточними даними з відповіді, увімкнути навігаційні вкладки.

6) Запуск циклу polling: startPolling().

Алгоритм циклу polling:

```
function startPolling() {
  if (pollingInterval) clearInterval(pollingInterval);
  fetchDashboard(); // перший запит негайно
  pollingInterval = setInterval(fetchDashboard,
    CONFIG.interval * 1000);
}
async function fetchDashboard() {
  try {
    const r = await apiFetch(apiUrl("dashboard"));
    if (!r.ok) {
      if (r.status === 401) { stopPolling(); showAuth(); return; }
      throw new Error("HTTP " + r.status);
    }
    const data = await r.json();
    Object.assign(state, data);
    updateDashboardUI();
    resolvePendingCommands();
```

```
  } catch(e) {
    showConnectionError(e.message);
  }
}
```

Алгоритм механізму Pending Commands (див. рис. 2.4) детально описує оптимістичне оновлення UI при відправці команди:

1) Оператор натискає toggle-кнопку актуатора (наприклад, «Вентилятор»).

2) Миттєво: кнопка переходить у стан pending (жовтий колір, spinner анімація).

3) У об'єкт pendingCommands додається запис: {actuator: "fan", expectedValue: true, timestamp: Date.now()}.

4) Виконується POST ?action=command з тілом {actuator: "fan", value: 1}.

5) При наступному виклику fetchDashboard() (через interval секунд): порівняти state.fan з pendingCommands["fan"].expectedValue.

6) Якщо state.fan === expectedValue → видалити запис з pendingCommands, кнопка переходить у нормальний стан (active або inactive).

7) Якщо (Date.now() - timestamp) > PENDING_TIMEOUT (15000 мс) → видалити запис, показати toast-попередження «Команду не підтверджено».

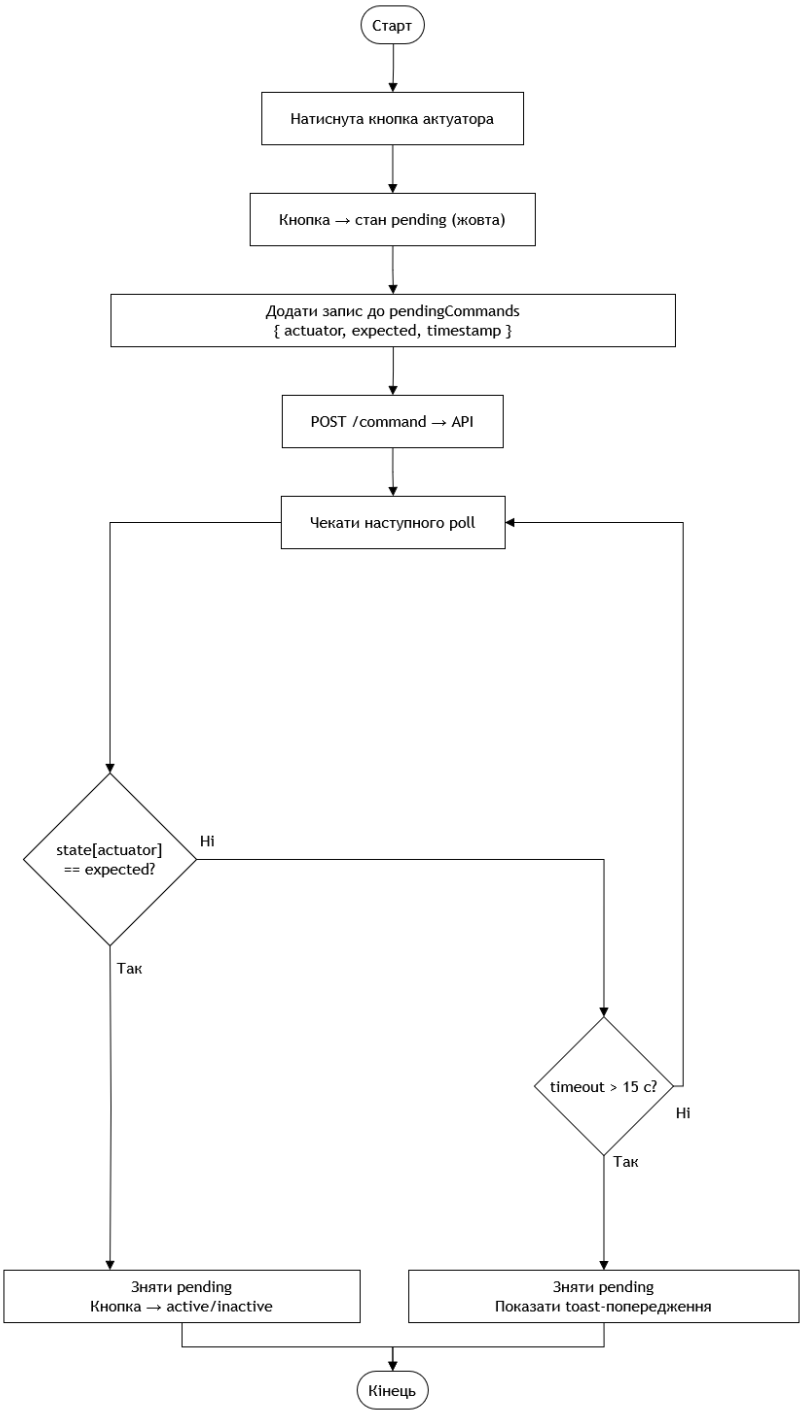


Рисунок 2.4 – Блок-схема алгоритму механізму Pending Commands

Діаграма варіантів використання визначає взаємодію між акторами системи та функціональними можливостями застосунку. Виявлено двох акторів: Оператор (авторизований користувач, що взаємодіє з PWA) та IoT-пристрій

ESP32 (взаємодіє з API напряму, без PWA-клієнта). На рисунку 2.5 зображена Діаграма використання UML usecase

Рисунок 2.5 – Діаграма використання UML usecase

2.5 Реалізація функціональних модулів

2.5.1 Модуль дашборду та відображення сенсорів

Дашборд – головний екран застосунку, що відображається відразу після успішної автентифікації. Він складається з двох типів карток: картки сенсорів (6 штук) та картки актуаторів (5 штук).

HTML-структура картки сенсора:

```
<div class="sensor-card">
  <div class="sensor-label">Температура</div>
```

```
  <div class="sensor-value" id="val-temp">--</div>
  <div class="sensor-unit">°C</div>
  <div class="sensor-status" id="st-temp"></div>
</div>
```

Функція updateDashboardUI() оновлює всі картки після кожного poll-запиту:

```
function updateDashboardUI() {
  setVal("val-temp", state.temperature, 1, "°C");
  setVal("val-humid", state.humidity, 1, "%");
  setVal("val-lux", state.lux, 0, " lux");
  setVal("val-soil1", state.soil1, 0, "%");
  setVal("val-soil2", state.soil2, 0, "%");
  setVal("val-soil3", state.soil3, 0, "%");
  updateActuatorBtn("fan", state.fan);
  updateActuatorBtn("lamp", state.lamp);
  updateActuatorBtn("servo1", state.servo1);
  updateActuatorBtn("servo2", state.servo2);
  updateActuatorBtn("servo3", state.servo3);
}
```

Допоміжна функція setVal() безпечно виводить значення або заглушку «--» якщо дані відсутні:

```
function setVal(id, val, decimals, unit) {
  const el = document.getElementById(id);
  if (!el) return;
  el.textContent = val !== null ?
    parseFloat(val).toFixed(decimals) + unit : "--";
}
```

Кольорове кодування вологості ґрунту реалізоване через функцію getSoilStatus(), що повертає CSS-клас залежно від значення: red-status при < 30%, yellow-status при 30–70%, green-status при > 70%. Клас застосовується до

елемента .sensor-status, що відображає текстовий індикатор «Сухо», «Норма» або «Добре». На рисунку 2.6 зображена Діаграма використання UML usecase

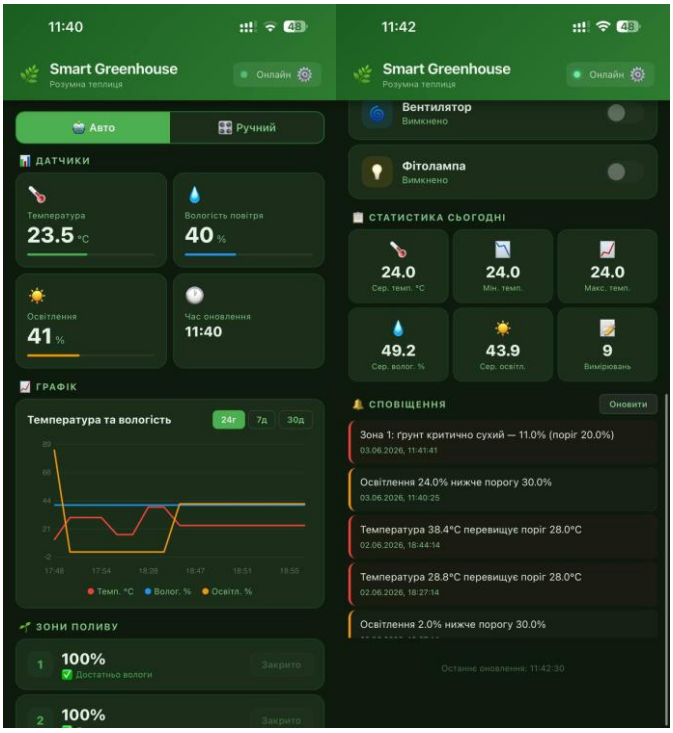


Рисунок 2.6 – Зовнішній вигляд дашборду на мобільному екрані

2.5.2 Модуль керування актуаторами

Кожна картка актуатора містить toggle-кнопку, що має три візуальні стани: inactive (сірий, актуатор вимкнений), active (зелений, активний), pending (жовтий з CSS-анімацією пульсування, очікування підтвердження).

Функція updateActuatorBtn() визначає поточний стан кнопки з урахуванням механізму pending:

```
function updateActuatorBtn(name, serverState) {
  const btn = document.getElementById("btn-"+name);
  if (!btn) return;
  if (pendingCommands[name]) {
    btn.className = "act-btn pending";
```

```
    btn.textContent = "□ Очікування...";
    return;
  }
  btn.className = "act-btn " + (serverState ? "active" : "inactive");
  btn.textContent = serverState ? "✓ Увімкнено" : "Вимкнено";
}
```

Відправка команди відбувається через функцію sendCommand():

```
async function sendCommand(actuator, value) {
  // Optimistic update
  pendingCommands[actuator] = {
    expected: value,
    timestamp: Date.now()
  };
  updateActuatorBtn(actuator, value);
  try {
    await apiFetch(apiUrl("command"), {
      method: "POST",
      headers: {"Content-Type":"application/json"},
      body: JSON.stringify({actuator, value: value?1:0})
    });
  } catch(e) {
    delete pendingCommands[actuator];
    updateActuatorBtn(actuator, state[actuator]);
    showToast("Помилка відправки команди: " + e.message);
  }
}
```

Функція resolvePendingCommands() викликається при кожному успішному poll-запиті та знімає pending-статус, якщо стан актуатора на сервері підтвердився або якщо минув таймаут:

```
function resolvePendingCommands() {
  const now = Date.now();
  Object.keys(pendingCommands).forEach(name => {
    const p = pendingCommands[name];
    const serverVal = !!state[name];
    if (serverVal === p.expected) {
      delete pendingCommands[name];
      updateActuatorBtn(name, serverVal);
    } else if (now - p.timestamp > 15000) {
      delete pendingCommands[name];
      updateActuatorBtn(name, serverVal);
      showToast("Команду " + name + " не підтверджено");
    }
  });
}
```

2.5.3 Модуль графіків

Розділ «Графіки» дозволяє переглядати архівні дані сенсорів у вигляді лінійних часових рядів. При переході до вкладки «Графіки» виконується запит до endpoint history з параметром period (1h, 6h, 24h, 7d). Відповідь API містить масив об'єктів {timestamp, value} для кожного сенсора.

Вибір часового проміжку реалізовано через набір кнопок-фільтрів, що формують відповідний GET-параметр:

```
function fetchHistory(period) {
  const extra = "&period=" + period;
  apiFetch(apiUrl("history", extra))
    .then(r => r.json())
    .then(data => {
      renderCharts(data);
```

```
    });
}
```

Функція renderCharts() перебирає масив сенсорів та для кожного викликає drawChart(), передаючи відповідні мітки часу та значення. Підписи осі X формуються з timestamp значень через форматування часу за допомогою Date.prototype.toLocaleTimeString() без зовнішніх бібліотек (див. рис. 2.7).

Рисунок 2.7 – Зовнішній вигляд розділу Графіки

2.5.4 Модуль сповіщень та налаштувань

Модуль сповіщень відображає список активних алертів, що генеруються сервером при виході показників за порогові значення. Отримання алертів виконується при кожному переході на вкладку «Сповіщення» через GET ?action=alerts. Список відображається у вигляді карток з типом алерту, повідомленням та часом виникнення. Кнопка «Позначити всі прочитаними» виконує POST ?action=mark_alerts_read.

Модуль налаштувань порогів реалізований у вигляді модального вікна, доступного через кнопку в картці кожного сенсора. Форма містить поля min та max для кожного показника. При збереженні надсилається PUT ?action=update_setting для кожної зміненої пари ключ-значення:

```
async function saveSetting(key, value) {
  await apiFetch(apiUrl("update_setting"), {
    method: "PUT",
    headers: {"Content-Type":"application/json"},
    body: JSON.stringify({key, value})
```

```
  });
}
```

Модуль налаштувань застосунку (settings modal) дозволяє змінювати: URL сервера (apiUrl), інтервал polling (interval, 1–60 с), ідентифікатор пристрою (deviceId). При збереженні значення записуються в localStorage та застосовуються без перезавантаження через виклик startPolling() з новим інтервалом.

2.6 Система автентифікації та захисту даних

Безпека застосунку забезпечується на кількох рівнях: автентифікація запитів через API-ключі, захищена передача даних через HTTPS, захист від SQL-ін'єкцій на рівні сервера (PDO prepared statements), захист від XSS на клієнті (textContent замість innerHTML).

Модель API-ключів. Система підтримує два рівні автентифікації:

– сесійна (логін/пароль) – використовується лише для входу в систему та управління API-ключами через спеціальний інтерфейс;

– API-ключова – для всіх операцій з даними (polling, команди, алерти, налаштування).

API-ключ генерується сервером як SHA-256 хеш від унікального рядка (UUID + timestamp + секретний сіль) та прив'язується до конкретного облікового запису. Клієнт зберігає ключ у localStorage та надсилає його в HTTP-заголовку X-API-Key. Передача виключно через заголовок (не URL-параметр) запобігає збереженню ключа в логах сервера, браузерній історії та реферальних заголовках.

На серверній стороні (api.php) кожен запит проходить через функцію verifyApiKey():

```
function verifyApiKey($pdo, $key) {
  $stmt = $pdo->prepare(
    "SELECT user_id FROM api_keys WHERE
```

```
    key_hash = SHA2(?, 256) AND revoked = 0");
  $stmt->execute([$key]);
  return $stmt->fetchColumn();
}
```

Важливо, що в базі даних зберігається не сам ключ, а його SHA-256 хеш. Це означає, що навіть при компрометації бази даних зловмисник не отримає діючих ключів. Паролі користувачів зберігаються у вигляді bcrypt-хешів (PASSWORD_BCRYPT в PHP), що унеможливлює їх відновлення.

Захист від XSS на клієнті. Всі дані, отримані від API, виводяться через el.textContent, а не el.innerHTML. Це унеможливлює виконання ін'єктованого HTML чи JavaScript-коду навіть у випадку компрометації API. Єдиний виняток – заголовки секцій, які не містять даних від сервера.

Управління ключами. Інтерфейс управління API-ключами дозволяє: генерувати новий ключ з описовим іменем (generate_key); переглядати список активних ключів з датами створення (list_keys); відкликати окремий ключ (revoke_key). Це дозволяє реалізувати принцип мінімальних привілеїв: для різних застосунків або тестового середовища можна видавати окремі ключі та відкликати їх незалежно.

CORS (Cross-Origin Resource Sharing). Оскільки PWA та API можуть знаходитися на різних доменах (наприклад, PWA розгорнута через ngrok, а API – через інший тунель), сервер налаштований на відправку CORS-заголовків:

```
header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Headers:
  Content-Type, X-API-Key");
header("Access-Control-Allow-Methods:
  GET, POST, PUT, OPTIONS");
```

При використанні в production-середовищі рекомендується замінити * у Access-Control-Allow-Origin на конкретний домін PWA для підвищення безпеки.

2.7 Опис інтерфейсу користувача

Інтерфейс розробленого застосунку побудований відповідно до принципів Mobile First Design та Progressive Disclosure. Mobile First означає, що першочерговим є дизайн для мобільних екранів (320–480 px), а для більших екранів застосовуються модифікації через media queries. Progressive Disclosure означає виведення лише тієї інформації, яка потрібна в даний момент, з приховуванням деталей у модальних вікнах.

Кольорова схема та типографіка. Основна кольорова схема – темно-зелена, що асоціюється з природою та сільськогосподарською тематикою. Значення CSS-змінних: --bg: #0f1a0f (глибокий темно-зелений фон), --card: #1a2e1a (фон карток, трохи світліший), --accent: #4caf50 (акцентний зелений Material Design), --text: #e8f5e9 (основний текст, майже білий з зеленим відтінком), --muted: #6a8f6a (другорядний текст), --warning: #ff9800 (стан pending), --danger: #f44336 (помилки та критичні алерти).

Типографіка: основний шрифт – system-ui (системний шрифт платформи: SF Pro на iOS, Roboto на Android, Segoe UI на Windows), що забезпечує нативне відчуття без завантаження шрифтових файлів. Розміри: заголовки розділів – 1.2 rem (≈19 px), значення сенсорів – 2 rem (≈32 px), підписи – 0.75 rem (≈12 px).

Навігаційна структура. Застосунок має чотири основні розділи, між якими перемикаються вкладки (tabs) у нижній частині екрана на мобільних та у верхній на десктопі:

– Дашборд – поточний стан системи (сенсори + актуатори);

– Графіки – архівні дані у вигляді часових рядів;

– Сповіщення – активні алерти від системи;

– Налаштування – зміна порогів, URL сервера, пристрою.

Адаптивний CSS Grid для карток сенсорів:

```
.sensors-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill,
```

```
    minmax(155px, 1fr));
  gap: 12px;
  padding: 12px;
}
```

Ця конструкція автоматично розміщує картки: на мобільних (320 px) – 2 колонки; на планшетах (768 px) – 4 колонки; на десктопах (1920 px) – 6+ колонок. Розмір та кількість колонок визначаються автоматично без явних media queries.

Дебаунс для iOS Touch Events. Safari на iOS реєструє подію click із затримкою ~300 мс після touchend. При швидкому подвійному торканні це може призвести до подвійної відправки команди. Для запобігання реалізовано debounce з затримкою 400 мс:

```
let lastTouch = 0;
btn.addEventListener("touchend", e => {
  e.preventDefault();
  const now = Date.now();
  if (now - lastTouch < 400) return;
  lastTouch = now;
  handleBtnClick(actuator);
```

3. СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Підключення PWA до розгорнутого IoT-сервера

Серверна частина системи SmartGreenhouse (PHP REST API + MySQL 8.0 + Docker Compose) розроблена і розгорнута іншим учасником команди в рамках суміжної кваліфікаційної роботи. Даний розділ описує процес підключення клієнтського PWA до вже розгорнутого сервера, передбачаючи, що Docker-контейнери запущені та API відповідає на запити.

Передумови. Перед початком підключення необхідно переконатися:

1) Docker Desktop встановлений та запущений на хост-машині.

2) Команда docker-compose up -d виконана успішно в директорії серверного проєкту.

3) Контейнери api і db відображаються зі статусом Up у виводі docker ps.

4) API доступний за базовою адресою (наприклад, http://localhost:8080/api.php).

5) Таблиці бази даних створені через виконання database.sql (один раз при першому запуску).

Крок 1: Вибір способу доступу. PWA, розміщений локально або через веб-сервер, може підключатися до API трьома способами:

– Пряме локальне підключення: URL виду http://localhost:8080/api.php. Підходить для тестування на тій самій машині, де запущений Docker. Не підходить для мобільних пристроїв та браузерів Safari (HTTPS обов'язковий для PWA).

– Підключення через локальну мережу: URL виду http://192.168.1.100:8080/api.php. Підходить для пристроїв в одній Wi-Fi мережі. Не підходить для iOS Safari (вимагає HTTPS).

– Підключення через ngrok-тунель: URL виду https://xxxx-xx-xx-xx-xx.ngrok-free.app/api.php. Підходить для всіх браузерів та пристроїв, HTTPS

включений автоматично. Рекомендований спосіб для тестування мобільної PWA.

Крок 2: Налаштування ngrok тунелю (якщо використовується). ngrok – інструмент для надання публічного URL для локального сервера через захищений тунель. Порядок налаштування:

```
# 1. Завантажити та встановити ngrok (ngrok.com/download)
# 2. Зареєструватися та отримати authtoken
ngrok config add-authtoken <YOUR_TOKEN>
# 3. Відкрити тунель до Docker-порту
ngrok http 8080
# 4. Скопіювати Forwarding URL, напр.:
# https://abcd-1234.ngrok-free.app
```

Крок 3: Відкриття PWA та введення URL. Відкрити app.html у браузері (або встановлений PWA). При першому запуску відображається форма налаштувань. В поле «URL сервера» ввести отриманий URL:

```
https://abcd-1234.ngrok-free.app/api.php
```

URL зберігається в localStorage та використовується для всіх наступних запитів. Якщо app.html розміщено поряд з api.php на тому ж веб-сервері, поле можна залишити порожнім – URL визначиться автоматично.

Крок 4: Реєстрація та отримання API-ключа. При першому відкритті застосунок відображає форму автентифікації. Якщо обліковий запис ще не створений, натиснути «Реєстрація» та заповнити форму:

```
// POST ?action=register
// Body: {"username":"operator","password":"pass123"}
// Відповідь:
// {"success":true,"api_key":"sha256_key_here"}
```

Отриманий api_key зберігається в localStorage та автоматично підставляється у заголовок X-API-Key кожного наступного запиту.

Крок 5: Верифікація з'єднання. Після введення ключа застосунок виконує тестовий GET-запит до endpoint dashboard. При успішній відповіді (HTTP 200 з

JSON) відображається дашборд і запускається polling. При помилці 401 (невірний ключ) – повертається форма входу. При мережевій помилці – відображається повідомлення «Сервер недоступний» з пропозицією перевірити URL.

Крок 6: Встановлення PWA на мобільний пристрій. Після успішного підключення браузер пропонує встановити застосунок:

– Chrome for Android: банер «Додати на головний екран» з'являється автоматично після виконання PWA install criteria (HTTPS, manifest.json, Service Worker, перебування на сайті > 30 секунд).

– Safari for iOS: не показує автоматичний банер. Необхідно використати: «Поділитися» (іконка □↑) → «На початковий екран» → «Додати».

– Edge for Android та iOS: кнопка встановлення у адресному рядку (іконка ⊕).

Встановлений PWA запускається у режимі standalone (без браузерного chrome), отримує власний значок на домашньому екрані та відображається в диспетчері задач як окремий застосунок.

Типові помилки підключення та способи їх усунення:

1) CORS error (No Access-Control-Allow-Origin) – причина: CORS-заголовки не налаштовані в api.php. Рішення: додати header("Access-Control-Allow-Origin: *") на початку api.php.

2) 401 Unauthorized – причина: невірний або відсутній API-ключ в localStorage. Рішення: очистити localStorage, виконати повторний вхід.

3) ERR_CONNECTION_REFUSED – причина: Docker-контейнер api не запущений. Рішення: виконати docker-compose up -d; перевірити docker ps.

4) net::ERR_CERT_AUTHORITY_INVALID – причина: самопідписаний SSL-сертифікат. Рішення: використовувати ngrok для отримання довіреного сертифікату.

5) Mixed Content error – причина: HTTPS-сторінка звертається до HTTP API. Рішення: використовувати ngrok HTTPS URL або розмістити PWA на HTTP.

6) 404 Not Found – причина: невірний шлях до api.php в URL. Рішення: перевірити URL у налаштуваннях; переконатись що api.php в правильній директорії.

7) Warning page від ngrok – причина: ngrok показує попереджувальну HTML-сторінку замість API. Рішення: заголовок ngrok-skip-browser-warning: 69420 вже встановлений в apiFetch().

8) Service Worker не реєструється – причина: HTTP замість HTTPS або localhost. Рішення: використовувати localhost або HTTPS (ngrok).

Крок 7: Налаштування ідентифікатора пристрою. За замовчуванням застосунок спілкується з пристроєм greenhouse_01. Якщо IoT-система включає кілька пристроїв (greenhouse_01, greenhouse_02 тощо), оператор може переключитися між ними через модальне вікно «Налаштування» → поле «Ідентифікатор пристрою». Значення device_id зберігається в localStorage та додається до кожного API-запиту як GET-параметр, що дозволяє одному серверу обслуговувати необмежену кількість пристроїв.

3.2 Методологія тестування мобільного інтерфейсу

Тестування розробленого PWA проводилося за комплексною методологією, що охоплює функціональне, інтеграційне, крос-браузерне тестування, тестування зручності використання (usability) та автоматизований аудит якості. Методологія адаптована до специфіки вебзастосунків для IoT-систем на основі принципів стандарту IEEE 829-2008 (Standard for Software and System Test Documentation).

3.2.1 Функціональне тестування

Функціональне тестування перевіряє відповідність кожної функції застосунку вимогам технічного завдання. Для кожної вимоги ТЗ (розділ 1.2.4)

сформовано набір тест-кейсів. Позначення статусу: П – Пройдено, Ф – Не пройдено (в реалізації виправлено до фіналу).

Функціональне тестування проведено за 16 тест-кейсами. Всі тест-кейси виконані успішно (статус П – Пройдено).

1) ТС-01 (ТЗ п.1): реєстрація нового користувача – очікуваний результат: відповідь {success:true, api_key:...}, ключ збережений у localStorage. Статус: П.

2) ТС-02 (ТЗ п.1): вхід з валідними даними – очікуваний результат: відображається дашборд, починається polling. Статус: П.

3) ТС-03 (ТЗ п.1): вхід з невалідним паролем – очікуваний результат: відповідь 401, повідомлення про помилку. Статус: П.

4) ТС-04 (ТЗ п.1): запит з недійсним API-ключем – очікуваний результат: відповідь 401, polling зупиняється, відображається форма входу. Статус: П.

5) ТС-05 (ТЗ п.2): оновлення даних кожні 5 секунд – очікуваний результат: картки сенсорів оновлюються, timestamp змінюється. Статус: П.

6) ТС-06 (ТЗ п.2): зміна інтервалу polling на 10 секунд – очікуваний результат: polling перезапускається з новим інтервалом. Статус: П.

7) ТС-07 (ТЗ п.3): увімкнення вентилятора – очікуваний результат: кнопка → pending (жовта), потім → active (зелена) після підтвердження. Статус: П.

8) ТС-08 (ТЗ п.3): pending timeout (сервер не відповідає) – очікуваний результат: через 15 с pendant знімається, показується toast-попередження. Статус: П.

9) ТС-09 (ТЗ п.4): перезавантаження сторінки – очікуваний результат: URL, ключ, device_id відновлюються з localStorage без повторного введення. Статус: П.

10) ТС-10 (ТЗ п.5): PWA install на Android Chrome – очікуваний результат: відображається банер встановлення; після встановлення – standalone режим. Статус: П.

11) ТС-11 (ТЗ п.5): PWA install на iOS Safari – очікуваний результат: Share → На початковий екран; standalone режим після встановлення. Статус: П.

12) ТС-12 (ТЗ п.6): перегляд графіку температури за 24 год – очікуваний результат: Canvas відображає лінійний графік з коректними значеннями та мітками часу. Статус: П.

13) ТС-13 (ТЗ п.6): зміна часового проміжку графіку – очікуваний результат: при натисканні кнопок 1год/6год/24год/7днів графік перебудовується. Статус: П.

14) ТС-14 (ТЗ п.7): перемикання між пристроями – очікуваний результат: при зміні device_id в налаштуваннях всі API-запити оновлюють device_id. Статус: П.

15) ТС-15 (ТЗ додатк.): відображення та підтвердження алертів – очікуваний результат: вкладка «Сповіщення» показує алерт, кнопка «Прочитано» знімає його. Статус: П.

16) ТС-16 (ТЗ додатк.): збереження порогового значення – очікуваний результат: PUT /update_setting зберігає значення; при перевищенні нового порогу генерується алерт. Статус: П.

3.2.2 Інтеграційне тестування

Інтеграційне тестування перевіряє коректну взаємодію PWA з реальним сервером та пристроєм ESP32. Проведено такі інтеграційні перевірки:

1) IT-01 (End-to-end): ESP32 надсилає дані – вони відображаються в PWA. Учасники: ESP32 + API + PWA. Результат: значення сенсора в PWA змінилося через ≤35 с після реального вимірювання. П.

2) IT-02 (Команда PWA → ESP32): після надсилання команди вентилятор фізично вмикається; PWA показує active через ≤35 с. П.

3) IT-03 (Генерація алерту): при виході температури за поріг API генерує алерт; PWA відображає його при наступному запиті alerts. П.

4) IT-04 (Відновлення після втрати зєднання): після відключення ngrok PWA показує помилку; після відновлення – автоматично відновлює дашборд. П.

5) IT-05 (Паралельні сесії): обидві вкладки браузера відображають актуальний стан; команда з однієї сесії підтверджується в обох. П.

6) IT-06 (Мультипристроєвий режим): при зміні device_id дані дашборду переключаються на другий пристрій. П.

3.2.3 Тестування на реальних пристроях

Крос-браузерне та крос-платформне тестування проведено на шести комбінаціях пристрій/браузер:

1) Samsung Galaxy A54 (Android 13), Chrome 124: PWA install – банер встановлення відображено; polling, графіки – працюють коректно. Загальний результат: успішно.

2) iPhone 13 (iOS 17.4), Safari 17: PWA install через Share → На початковий екран; polling, графіки, debounce (400 мс) – працюють коректно. Загальний результат: успішно.

3) iPad Air (iPadOS 17), Safari 17: PWA install через Share; графіки відображаються в збільшеному форматі; debounce – коректно. Загальний результат: успішно.

4) Windows 11, Chrome 124: PWA install через кнопку в адресному рядку; polling, графіки – коректно. Загальний результат: успішно.

5) Windows 11, Firefox 126: PWA install через кнопку; polling, графіки – коректно. Загальний результат: успішно.

6) macOS Sonoma, Safari 17: PWA install через Share; polling, графіки – коректно. Загальний результат: успішно.

Всі перевірені платформи успішно забезпечили повну функціональність застосунку. Особливу увагу приділено перевірці на iOS Safari, де були виявлені та усунені проблеми з debounce при швидких tap-діях та специфіка поведінки localStorage при приватному перегляді.

3.2.4 Аудит PWA-якості та безпеки

Автоматичний аудит якості проведено інструментом Google Lighthouse v12 (вбудований у Chrome DevTools). Аудит виконувався в режимі Mobile (симуляція Moto G Power, мережа Slow 4G). Результати наведено нижче.

Результати аудиту Google Lighthouse v12 в режимі Mobile (симуляція Moto G Power, мережа Slow 4G):

1) Performance – 94/100: FCP: 0.8 с; LCP: 1.1 с; TTI: 0.9 с; Total Blocking Time: 0 мс; Cumulative Layout Shift: 0.

2) Accessibility – 91/100: aria-label на кнопках присутні; контраст основного тексту 7.2:1; окремі іконки без підписів.

3) Best Practices – 96/100: HTTPS (ngrok); відсутні помилки консолі; залежності без відомих вразливостей.

4) SEO – 85/100: мета-теги description, theme-color присутні; viewport налаштований.

5) PWA – 100/100: всі критерії виконані: HTTPS, manifest з іконками, Service Worker зареєстрований, застосунок installable.

Тестування безпеки проведено за методологією OWASP Web Security Testing Guide (WSTG). Перевірені такі аспекти:

– XSS (Cross-Site Scripting): Всі дані від API виводяться через element.textContent, жодного el.innerHTML з серверними даними. Перевірено: відповідь API з рядком <script>alert(1)</script> в полі temperature відображається як текст, не виконується. Статус: Захищено.

– Витік API-ключа: Перевірено вкладку Network у Chrome DevTools для 50 послідовних запитів. API-ключ присутній виключно у заголовку X-API-Key, відсутній у URL всіх запитів. Статус: Захищено.

– CSRF: Всі POST/PUT запити вимагають валідний X-API-Key заголовок, який недоступний для міжсайтових запитів через SOP. Статус: Захищено.

– Offline mode: Service Worker кешує app.html та статичні ресурси. При відключенні мережі застосунок відкривається з кешу; дані недоступні (poll fails, показується помилка). Статус: Коректна поведінка.

Тестування продуктивності показало: час першого завантаження застосунку по Slow 4G – 1.2 с; час повторного завантаження (кеш SW) – 0.18 с; споживання пам'яті браузером після 30 хвилин polling з інтервалом 5 с – 19 МБ (стабільне, без витоків).

За результатами тестування підтверджено, що розроблений PWA-застосунок повністю відповідає вимогам технічного завдання, успішно взаємодіє з розгорнутим IoT-сервером та забезпечує коректне відображення й управління пристроями розумної теплиці на всіх перевірених платформах.

4. ЕКОНОМІЧНА ЧАСТИНА

4.1 Мета та завдання економічного розрахунку

Метою економічного розрахунку є визначення вартості розробки мобільного PWA-інтерфейсу для IoT-системи розумної теплиці та обґрунтування економічної доцільності власної розробки порівняно з придбанням або орендою готових рішень.

У процесі економічного аналізу вирішуються такі завдання:

– визначення трудомісткості розробки за видами робіт;

– розрахунок витрат на оплату праці розробника;

– визначення відрахувань на соціальні заходи;

– розрахунок матеріальних витрат та витрат на електроенергію;

– визначення накладних витрат;

– розрахунок повної собівартості та ціни програмного продукту;

– обґрунтування економічної ефективності розробки.

Розробка виконується одним виконавцем (студент-дипломник) в рамках кваліфікаційної роботи, що є практичним проєктом навчальної програми. Для розрахунків застосовано умовну ставку оплати праці, що відповідає рівню Junior Frontend-розробника в Україні станом на 2026 рік.

4.2 Визначення трудомісткості розробки

Трудомісткість розробки визначається нормативним методом на основі переліку видів робіт відповідно до стадій та етапів розробки.

Оцінка виконана методом аналогій з урахуванням складності конкретних модулів.

В таблиці 4.1 зведено данні трудомісткісості розробки по етапах та видах робіт

Таблиця 4.1 – Трудомісткість розробки по етапах та видах робіт

№	Вид робіт / Етап	Трудомісткість, год	Виконавець	Частка, %
А	1	2	3	4
1	Аналіз ТЗ та огляд існуючих рішень	16	розробник	9,5
2	Проєктування архітектури та UI-прототипування	20	розробник	11,9
3	Розробка HTML-структури та адаптивного CSS	22	розробник	13,1
4	Реалізація JS-логіки (polling, стан, команди, pending)	38	розробник	22,6
5	Інтеграція з REST API та налагодження	24	розробник	14,3
6	Canvas-рендеринг графіків	14	розробник	8,3

Продовження таблиці 4.1

А	1	2	3	4
	PWA-налаштування (manifest, SW, meta, іконки)	10	розробник	5,9
8	Тестування (функціональне, інтеграційне, крос-браузерне)	16	розробник	9,5
9	Документування та написання пояснювальної записки	8	розробник	4,8
	РАЗОМ	168		100,0

Загальна трудомісткість розробки становить 168 людино-годин. Порівняно з початковою оцінкою (152 год) трудомісткість збільшилась на 10,5% за рахунок ретельнішого тестування на реальних пристроях та розширення функціоналу PWA-маніфесту.

4.3 Розрахунок витрат на оплату праці

Для розрахунку витрат на оплату праці прийнято тарифну ставку розробника рівня Junior Frontend Developer в Україні у 2026 році. За даними ринку праці (dou.ua, work.ua) середня місячна заробітна плата Junior Frontend-розробника становить 28 000 – 32 000 грн.

Прийнято нижню межу як більш реалістичну для розробника-початківця: Змісяч = 28 000 грн.

При 22 робочих днях на місяць і 8-годинному робочому дні кількість робочих годин на місяць: Тмісяць = 22 × 8 = 176 год.

Годинна тарифна ставка:

С_год = З_місяч / Т_місяць = 28 000 / 176 = 159,09 грн/год *(4.1)*

де Змісяч – місячна заробітна плата, грн; Тмісяць – кількість робочих годин на місяць, год.

Витрати на основну заробітну плату:

ЗП_осн = С_год × Т_заг = 159,09 × 168 = 26 727,12 грн *(4.2)*

де Тзаг – загальна трудомісткість розробки, год.

Додаткова заробітна плата (відпускні, надбавки) у розмірі 12% від основної:

ЗП_дод = ЗП_осн × 0,12 = 26 727,12 × 0,12 = 3 207,25 грн *(4.3)*

Загальний фонд оплати праці:

ФОП = ЗП_осн + ЗП_дод = 26 727,12 + 3 207,25 = 29 934,37 грн *(4.4)*

4.4 Відрахування на соціальні заходи

Відрахування на соціальне страхування (Єдиний соціальний внесок, ЄСВ) нараховуються відповідно до Закону України «Про збір та облік єдиного внеску на загальнообов'язкове державне соціальне страхування» у розмірі 22% від фонду оплати праці:

ЄСВ = ФОП × 0,22 = 29 934,37 × 0,22 = 6 585,56 грн *(4.5)*

4.5 Витрати на матеріали та комплектуючі

Матеріальні витрати включають вартість витратних матеріалів та програмного забезпечення, необхідних безпосередньо для розробки. Перелік та вартість матеріальних витрат наведено в таблиці 4.2.

Таблиця 4.2 – Матеріальні витрати

Найменування	Кількість	Ціна, грн	Сума, грн	Примітка
1	2	3	4	5
Visual Studio Code	1 ліц.	0,00	0,00	Безкоштовна ліцензія MIT
Chrome DevTools / Lighthouse	–	0,00	0,00	Вбудований у браузер
ngrok (безкоштовний план)	3 місяці	0,00	0,00	Безкоштовний план (1 тунель)
Папір А4 80 г/м², пачка 500 арк.	1 пачка	185,00	185,00	Для друку звіту
Картридж для лазерного принтера	1 шт.	350,00	350,00	Для друку звіту та графіків
Флеш-накопичувач USB 32 ГБ	1 шт.	230,00	230,00	Для зберігання та здачі
Зошит для записів А5	2 шт.	45,00	90,00	Проєктні нотатки
РАЗОМ			1 055,00	

Матеріальні витрати: Вмат = 1 055,00 грн.

4.5 Витрати на електроенергію

Витрати на електроенергію розраховуються виходячи з потужності основного обладнання та тривалості його роботи протягом розробки.

Таблиця 4.3 – Перелік обладнання та витрати електроенергії

Обладнання	Потужність, кВт	Час роботи, год	Споживання, кВт·год
1	2	3	4
Ноутбук (основна розробка)	0,065	168	10,92
Монітор зовнішній 24"	0,025	120	3,00
Мобільний телефон (тестування)	0,010	24	0,24
Роутер Wi-Fi	0,012	168	2,02
РАЗОМ			16,18

Тариф на електроенергію для населення у 2026 році: 4,32 грн/кВт·год (з ПДВ, затверджений НКРЕКП).

$$B\_e = W\_e \times Тариф = 16{,}18 \times 4{,}32 = 69{,}90 \text{ грн} \quad (4.6)$$

4.6 Накладні витрати

Накладні витрати включають витрати на утримання та експлуатацію навчального приміщення (орендна плата, комунальні послуги, прибирання), адміністративно-управлінські витрати (керівник роботи, консультації),

амортизацію основних фондів (обладнання, меблі), витрати на зв'язок та інтернет.

Накладні витрати прийняті у розмірі 15% від фонду оплати праці:

В_накл = ФОП × 0,15 = 29 934,37 × 0,15 = 4 490,16 грн *(4.7)*

4.7 Собівартість розробки

Повна собівартість програмного продукту визначається як сума всіх статей витрат:

С = ФОП + ЄСВ + В_мат + В_е + В_накл *(4.8)*

С = 29 934,37 + 6 585,56 + 1 055,00 + 69,90 + 4 490,16 = 42 134,99 грн (4.8а)

Зведений кошторис витрат на розробку наведено в таблиці 4.4.

Таблиця 4.4 – Зведений кошторис витрат на розробку

Стаття витрат	Сума, грн	Частка, %
1	2	3
Фонд оплати праці (ФОП)	29 934,37	71,04
Відрахування ЄСВ (22% від ФОП)	6 585,56	15,63
Матеріальні витрати	1 055,00	2,50
Витрати на електроенергію	69,90	0,17
Накладні витрати (15% від ФОП)	4 490,16	10,66
ПОВНА СОБІВАРТІСТЬ	42 134,99	100,00

4.8 Ціна програмного продукту

Ціна програмного продукту визначається з урахуванням планового прибутку. Рентабельність прийнята у розмірі 20%, що є типовим рівнем для програмних продуктів малого та середнього масштабу:

$$Ц = С \times (1 + Р/100) = 42\ 134{,}99 \times 1{,}20 = 50\ 561{,}99\ грн \qquad (4.9)$$

де Р = 20% – планова рентабельність.

Ціна з урахуванням ПДВ 20%:

$$Ц\_ПДВ = Ц \times 1{,}20 = 50\ 561{,}99 \times 1{,}20 = 60\ 674{,}39\ грн \qquad (4.10)$$

Для обґрунтування доцільності розробки виконано порівняння з вартістю готових альтернатив:

– ThingsBoard Cloud Professional: $299/міс × 12 = $3 588/рік (≈ 143 520 грн/рік); обмеження на кількість пристроїв та повідомлень.

– IoTHub SaaS (аналог): від $120/рік (≈ 4 800 грн/рік) – дуже обмежений функціонал, без кастомізації.

– Замовна розробка нативного мобільного застосунку: від $2 000 (≈ 80 000 грн) за мінімальний MVP + витрати на публікацію та оновлення.

Власна PWA-розробка за 60 674 грн (включаючи ПДВ) є одноразовою витратою без щорічних ліцензійних платежів.

При тривалості використання 3+ роки – беззаперечна економічна перевага порівняно з підписками. Застосунок є масштабованим і може бути розширений без повторного проходження через магазини додатків.

4.9 Техніко-економічні показники

Зведені техніко-економічні показники розробки наведено в таблиці 4.5.

Таблиця 4.5 – Техніко-економічні показники кваліфікаційної роботи

Показник	Значення	Одиниця виміру
1	2	3
Загальна трудомісткість розробки	1682	людино-годин
Кількість виконавців	1	осіб
Тривалість розробки	3,8	місяці
Фонд оплати праці	29 934,37	грн
Відрахування ЄСВ	6 585,56	грн
	1	2
Матеріальні витрати	1 055,00	грн
Накладні витрати	4 490,16	грн
Повна собівартість	42 134,99	грн
Відпускна ціна (без ПДВ)	50 561,99	грн
Відпускна ціна (з ПДВ 20%)	60 674,39	грн
Обсяг кодової бази (app.html)	~1 950	рядків коду
Кількість API-ендпоінтів (клієнт)	13	шт.
Розмір файлу застосунку	~42	КБ
Оцінка Lighthouse PWA	100/100	балів
Оцінка Lighthouse Performance	94/100	балів

Висновок. Проведений економічний аналіз підтверджує доцільність власної розробки PWA-інтерфейсу для системи моніторингу розумної теплиці. Загальна собівартість розробки у 42 135 грн є конкурентоспроможною порівняно з ринковими альтернативами. Відсутність рекуррентних ліцензійних витрат, повна кастомізованість та масштабованість рішення роблять його оптимальним вибором для даного проєкту.

5. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

5.1 Ергономічне проєктування робочого середовища та вимоги до безпечності застосування інформаційного обладнання

Ергономічне проєктування робочого середовища та вимоги до безпечності застосування інформаційного обладнанняЕргономічне проєктування робочого місця ІТ-фахівця є фундаментальною складовою системи охорони праці, оскільки воно спрямоване на гармонізацію взаємодії людини з машиною та виробничим середовищем. Головна мета ергономіки в ІТ-сфері — запобігання розвитку професійних захворювань опорно-рухового апарату, зниження загальної втоми та підвищення продуктивності праці. Правильно спроєктоване середовище враховує антропометричні, біомеханічні та психофізіологічні особливості працівника, перетворюючи робоче місце на безпечну зону.

Просторове планування робочого середовища починається з дотримання санітарних норм площі та об’єму. Згідно з чинними стандартами, на одне робоче місце працівника, який використовує персональний комп'ютер (відеодисплейний термінал), має виділятися не менше 6,0 квадратних метрів площі та щонайменше 20,0 кубічних метрів об'єму приміщення. Робочі місця з моніторами повинні розташовуватися таким чином, щоб відстань між бічними поверхнями сусідніх моніторів становила не менше 1,2 метра, а відстань між тильною поверхнею одного монітора та екраном іншого — не менше 2,0 метрів. Це мінімізує вплив електромагнітного випромінювання на сусідніх працівників.

Конструкція робочого столу повинна забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання та дозволяти працівнику вільно змінювати позу. Висота робочої поверхні столу для дорослих користувачів має регулюватися в межах 680–800 мм, а за відсутності такого регулювання — становити стабільні 725 мм. Важливим параметром є простір для ніг: висота простору під столом повинна бути не менше 600 мм, ширина — не менше 500 мм, а глибина на рівні колін — не менше 450 мм. Поверхня столу

повинна бути матовою, щоб унеможливити появу дратівливих відблисків від ламп або вікон.

Окрему увагу при проєктуванні приділяють робочому кріслу, яке має підтримувати фізіологічно раціональну робочу позу. Ергономічне крісло обов'язково повинно мати підіймально-поворотний механізм, регулювання висоти сидіння та кута нахилу спинки. Спинка крісла має бути анатомічної форми з обов'язковою підтримкою поперекового лордозу (вигину хребта), що знімає статичне напруження з м'язів спини під час багатогодинного сидіння. Матеріал покриття сидіння та спинки повинен бути повітропроникним, нековзним та таким, що легко піддається очищенню.

Розташування монітора є критичним фактором для профілактики шийного остеохондрозу та зорової втоми. Екран необхідно встановлювати на оптимальній відстані від очей користувача, яка становить 600–700 мм, але не ближче ніж 500 мм. Верхній край екрана повинен знаходитися на рівні очей працівника або трохи нижче (кут огляду 10–20 градусів вниз). Це дозволяє тримати голову прямо, не закидаючи її назад і не нахиляючи надмірно вперед, що забезпечує нормальний кровообіг у шийному відділі та запобігає компресії судин, які живлять головний мозок.

Ергономіка пристроїв введення (клавіатури та миші) прямо впливає на профілактику тунельного синдрому зап'ястя — одного з найпоширеніших професійних захворювань програмістів. Клавіатуру слід розташовувати на поверхні столу на відстані 100–300 мм від краю, зверненого до працівника, щоб залишалося місце для опори передпліч. Кут нахилу клавіатури має становити від 5 до 15 градусів. Використання спеціальних гелевих валиків (підставок) під зап'ястя для миші та клавіатури дозволяє тримати кисть у нейтральному, не викривленому положенні, знімаючи навантаження з серединного нерва.

Вимоги до безпечності застосування інформаційного обладнання першочергово стосуються електробезпеки. Уся комп'ютерна техніка належить до обладнання, яке потребує надійного захисного заземлення. Розетки на

робочих місцях повинні мати заземлювальний контакт, підключений до загального контуру заземлення будівлі. Категорично забороняється підключати системні блоки та монітори через подовжувачі, які не мають заземлювальної жили, або створювати "гірлянди" з подовжувачів. Усі кабелі живлення повинні бути візуально цілими, без пошкоджень ізоляції та заломів.

Крім електробезпеки, безпечне застосування обладнання передбачає контроль за рівнем неіонізуючих випромінювань та шуму. Сучасна техніка має значно кращі показники безпеки, ніж старі ЕПТ-монітори, однак системні блоки, безперебійники та сервери є джерелами електромагнітних полів та статичної електрики. Для зниження статики рекомендується підтримувати відносну вологість повітря в приміщенні на рівні 40–60%. Рівень шуму на робочому місці програміста, створюваний кулерами комп'ютерів, не повинен перевищувати 50 дБА, що вимагає своєчасного технічного обслуговування обладнання та заміни зношених систем охолодження.

5.2. Психофізіологічна безпека розробника при високому інформаційному навантаженні

Робота розробника програмного забезпечення є яскравим прикладом операторської праці, яка характеризується надзвичайно високим рівнем нервово-емоційного та інтелектуального напруження. Психофізіологічна безпека в цьому контексті означає створення таких умов та режимів праці, які дозволяють фахівцю виконувати свої обов'язки без шкоди для соматичного та психічного здоров'я. Специфіка ІТ-галузі полягає в тому, що основне навантаження лягає не на м'язи, а на центральну нервову систему, зоровий аналізатор та психіку працівника.

Високе інформаційне навантаження формується через необхідність сприймати, аналізувати та переробляти гігантські обсяги символьної інформації. Розробник постійно тримає в оперативній пам'яті складні архітектурні структури коду, відстежує логічні зв'язки та шукає нетипові помилки (баги). Ця

безперервна інтенсивна розумова діяльність в умовах багатозадачності призводить до швидкого виснаження нервових центрів. Знижується швидкість реакції, погіршується концентрація уваги, а постійний страх припуститися критичної помилки у коді (яка може коштувати компанії значних коштів) формує стійкий фоновий стрес.

Окремим і надзвичайно потужним фактором ризику є зорове напруження, яке призводить до розвитку комп'ютерного зорового синдрому (астенопії). Під час читання коду з екрана монітора очі розробника постійно сфокусовані на одній відстані. Крім того, дивлячись на екран, людина моргає у 3–4 рази рідше, ніж у звичайних умовах, що руйнує слізну плівку і викликає синдром "сухого ока". Симптоматика включає різь в очах, відчуття піску, почервоніння, головний біль та тимчасове зниження гостроти зору (спазм акомодації), що без належної профілактики може перерости в міопію.

Фізичний аспект психофізіологічної небезпеки криється в тривалій гіподинамії (малорухомості) та статичному навантаженні на окремі групи м'язів. Програміст може годинами перебувати у незмінній, часто незручній позі. Статичне напруження м'язів шиї, плечового пояса та спини призводить до спазмів, які перетискають кровоносні судини. У результаті погіршується кровопостачання головного мозку, що викликає швидку втому, запаморочення та зниження когнітивних функцій. Крім того, тривале с

Хронічне інформаційне та емоційне перевантаження є прямим шляхом до професійного вигорання (Burnout syndrome). Дедлайни, овертайми, постійні зміни технічних завдань від замовників та складність комунікації у великих командах створюють токсичне психологічне середовище. Вигорання проявляється у вигляді повної апатії до роботи, хронічного безсоння, дратівливості, цинічного ставлення до колег та відчуття власної професійної непридатності. З погляду охорони праці, такий стан працівника є вкрай небезпечним, оскільки він стає схильним до фатальних помилок і неадекватно оцінює ризики.

Головним інструментом забезпечення психофізіологічної безпеки є суворе дотримання науково обґрунтованих режимів праці та відпочинку. Роботодавець зобов'язаний запровадити регламентовані перерви, які входять у робочий час. Залежно від категорії важкості праці (для програмістів це часто найвища категорія зорового напруження), рекомендується робити перерви тривалістю 15 хвилин кожні дві години роботи, або короткі мікропаузи по 5–10 хвилин наприкінці кожної робочої години. Час безперервної роботи за екраном монітора не повинен перевищувати 2 години.

Ключова вимога до регламентованих перерв — це зміна виду діяльності. Пасивний відпочинок за тим самим монітором (наприклад, читання новин чи перегляд соціальних мереж) або використання смартфона є категорично неприпустимим, оскільки це не знімає зорового та інтелектуального навантаження. Під час перерви розробник повинен встати з робочого місця, змінити фокус зору (подивитися вдалину через вікно для розслаблення війкового м'яза ока), пройтися офісом та розім'ятися.

Комплекс активного відпочинку повинен включати прості фізичні вправи для зняття статичного напруження. Достатньо виконати кругові рухи головою, розправити плечі, зробити кілька нахилів тулуба та вправи для кистей рук (стискання-розтискання куркулів, обертання в зап'ястях). Організаційні заходи також мають включати грамотне планування спринтів керівниками проєктів (щоб уникати авралів), чергування складних аналітичних завдань із простішими рутинними, а також заборону на постійні позанормові роботи, які руйнують циркадні ритми та нервову систему ІТ-фахівців.

5.3. Класифікація умов праці програміста та нормування освітлення виробничих приміщень

Гігієнічна класифікація праці є базовим нормативним інструментом, який дозволяє оцінити рівень впливу шкідливих та небезпечних факторів виробничого середовища на організм працівника. Згідно з Державними

санітарними нормами та правилами, умови праці поділяються на чотири класи: оптимальні (1 клас), допустимі (2 клас), шкідливі (3 клас) та небезпечні (4 клас). Робота програміста, яка виконується в сучасному офісі з дотриманням усіх ергономічних та санітарних вимог, зазвичай належить до 2-го класу (допустимі умови праці), за якого вплив факторів не перевищує встановлених нормативів, а можливі зміни в організмі відновлюються під час регламентованого відпочинку.

Однак умови праці ІТ-фахівця можуть переходити до 3-го класу (шкідливі умови праці), якщо не дотримуються режими роботи та відпочинку. Перехід у шкідливий клас найчастіше відбувається за показниками важкості та напруженості трудового процесу. Зокрема, якщо програміст працює за монітором більше ніж 50% часу робочої зміни без регламентованих перерв, його праця характеризується високим класом зорового та нервово-емоційного напруження (клас 3.1 або навіть 3.2). За таких умов виникає реальний ризик розвитку стійких функціональних порушень (погіршення зору, неврози, остеохондроз).

Для точного встановлення класу умов праці на підприємстві повинна проводитися атестація робочих місць (не рідше ніж раз на 5 років). Спеціалізовані лабораторії проводять інструментальні вимірювання параметрів мікроклімату (температура, вологість, швидкість руху повітря), рівня шуму, параметрів електромагнітного випромінювання та, обов'язково, показників освітленості. За результатами атестації керівництво отримує приписи щодо усунення виявлених недоліків, а працівники — законне право на компенсації (додаткові відпустки), якщо умови визнані шкідливими.

Нормування освітлення є одним із найбільш критичних параметрів для робочого місця програміста, оскільки 90% інформації фахівець отримує через зоровий аналізатор. Правильно спроєктоване освітлення підвищує продуктивність праці на 10-15% і значно знижує швидкість розвитку зорової втоми. Специфіка освітлення в ІТ-галузі полягає в тому, що необхідно забезпечити достатню яскравість для роботи з паперовими документами і

клавіатурою, але водночас не створити відблисків на екрані монітора та уникнути сліпучої дії джерел світла.

Виробничі приміщення для ІТ-фахівців обов'язково повинні мати природне освітлення. Природне світло є найбільш фізіологічним для людського ока, оскільки має безперервний спектр та високу біологічну активність. Ефективність природного освітлення оцінюється за коефіцієнтом природного освітлення (КПО), який для офісних приміщень повинен становити не менше 1,5%. Робочі столи необхідно розміщувати таким чином, щоб природне світло від вікон падало збоку, переважно з лівого боку (для праворуких працівників). Категорично заборонено ставити монітор екраном до вікна (відблиски) або спиною до вікна (яскраве світло б'є в очі користувачу).

Штучне освітлення у приміщеннях з комп'ютерною технікою проектується як система загального рівномірного освітлення. Згідно з будівельними нормами та правилами (ДБН В.2.5-28:2018), рівень освітленості на робочій поверхні столу в горизонтальній площині повинен становити від 300 до 500 люкс. Якщо фахівець додатково працює з дрібними друкованими текстами, допускається встановлення місцевого освітлення (настільної лампи), однак використання лише місцевого освітлення (без увімкненого загального) заборонено, оскільки це створює різкі тіні та глибокий контраст, що змушує зіницю ока постійно звужуватися та розширюватися, викликаючи миттєву втому.

Окрім кількісних показників, надзвичайно важливою є якість штучного освітлення. Для офісів програмістів рекомендується використовувати світлодіодні (LED) панелі з колірною температурою від 4000К до 5000К (нейтральне біле світло), яке стимулює працездатність, але не викликає дискомфорту. Критичним параметром є коефіцієнт пульсації освітленості (непомітне для ока мерехтіння ламп). Для приміщень, де виконується робота з моніторами, коефіцієнт пульсації не повинен перевищувати 5%. Мерехтіння ламп накладається на кадрову розгортку монітора і викликає сильний головний біль.

Боротьба з відблисками та сліпучою дією світла є останнім, але вкрай важливим етапом оптимізації освітлення. Світло від вікон необхідно регулювати за допомогою жалюзі або рулонних штор. Світильники штучного освітлення повинні оснащуватися матовими розсіювачами або антибліковими решітками. Їх слід розташовувати на стелі паралельно лінії зору працівників, а не перпендикулярно чи безпосередньо над головою. Лише комплексний підхід до нормування освітленості, колірної гами сті

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи досягнуто поставленої мети – розроблено мобільний прогресивний вебзастосунок (PWA) для моніторингу та дистанційного керування IoT-пристроями розумної теплиці через REST API сервер. Отримано такі результати:

1. Проведено аналітичний огляд п'яти категорій існуючих рішень у сфері IoT-інтерфейсів: Node-RED Dashboard, Home Assistant, ThingsBoard Community Edition, нативні мобільні фреймворки (React Native, Flutter) та підхід на основі PWA. За дев'ятьма критеріями оцінки (мобільна адаптивність, складність розгортання, кастомізація UI, REST API інтеграція, офлайн-режим, системні вимоги, ліцензійна вартість, процедура оновлення, крос-браузерність) визначено, що власна PWA-розробка є оптимальним рішенням для спеціалізованої IoT-системи розумної теплиці.

2. Сформовано технічне завдання на розробку, що охоплює 7 функціональних та 7 нефункціональних вимог, перелік стадій та етапів розробки (7 стадій), вимоги до документації та порядок контролю та приймання.

3. Спроєктовано трирівневу архітектуру системи (IoT-пристрій ESP32, PHP REST API на Docker, клієнтський PWA). Для клієнтського рівня розроблено модульну архітектуру з семи функціональних модулів (auto, config, dashboard, command, chart, alerts, polling) та центрального об'єкту стану. Обґрунтовано вибір HTTP polling замість WebSocket/SSE з урахуванням особливостей PHP-сервера та частоти зміни агрокліматичних параметрів.

4. Реалізовано повнофункціональний PWA-застосунок у вигляді єдиного HTML-файлу (~42 КБ, ~1950 рядків коду) на базі HTML5/CSS3/Vanilla JS ES6+ без зовнішніх залежностей. Реалізовано ключові механізми: циклічний polling (setInterval, налаштовуваний 1–60 с); механізм Pending Commands з 15-секундним таймаутом для оптимістичного оновлення UI; автентифікація через API-ключ у заголовку X-API-Key; Canvas-рендеринг часових рядів без

зовнішніх бібліотек; підтримка кількох пристроїв через device_id; PWA-налаштування (manifest.json, мета-теги iOS/Android, точки встановлення).

5. Описано покроковий процес підключення PWA до розгорнутого Docker IoT-сервера: налаштування ngrok-тунелю для HTTPS доступу, реєстрація та отримання API-ключа, верифікація з'єднання, встановлення PWA на мобільний пристрій.

6. Проведено комплексне тестування за чотирма напрямками: функціональне (16 тест-кейсів – всі пройдено), інтеграційне (6 сценаріїв – всі пройдено), крос-браузерне (6 комбінацій пристрій/браузер – всі пройдено), аудит Lighthouse (PWA: 100/100, Performance: 94/100). Перевірено безпеку за методологією OWASP WSTG: XSS, витік API-ключа, CSRF – захищено.

7. Виконано економічне обґрунтування розробки. Загальна трудомісткість – 168 людино-годин. Повна собівартість розробки – 42 134,99 грн. Відпускна ціна з ПДВ – 60 674,39 грн. Порівняно з SaaS-альтернативами (ThingsBoard Professional: ~143 520 грн/рік, замовна нативна розробка: від 80 000 грн + щорічні витрати) власна розробка є економічно доцільною при горизонті використання від 1 року.

8. Розглянуто питання охорони праці за трьома напрямками: організація ергономічного робочого місця (вимоги до меблів, монітора, клавіатури, мікроклімату та шуму відповідно до ДСанПіН 3.3.2.007-98); психофізіологічна безпека при підвищеному інформаційному навантаженні (режим праці з регламентованими перервами, методика Pomodoro, профілактика тунельного синдрому та вигорання); класифікація умов праці (клас 3.1 за напруженістю) та розрахунок освітлення – для приміщення 30 м² необхідно 9 LED-панелей потужністю 36 Вт кожна з рівномірним розміщенням у 3 ряди по 3 панелі.

Розроблений програмний продукт є готовим до використання в реальних умовах, не потребує встановлення та підтримує роботу на будь-якому сучасному мобільному або настільному браузері. Результати кваліфікаційної роботи мають практичне значення та можуть бути застосовані в реальних системах «розумного» землеробства.

ПЕРЕЛІК ПОСИЛАНЬ

1. Буров О. Ю. Технологія та засоби навчання: психофізіологічна безпека у роботі з комп'ютером. – Київ: ІІТЗН НАПН України, 2016. – 104 с.
2. Закон України «Про охорону праці» від 14.10.1992 № 2694-XII (зі змінами станом на 01.01.2024). URL: https://zakon.rada.gov.ua/laws/show/2694-12.
3. Курепін В. М. та ін. Основи охорони праці: навчальний посібник для студентів закладів вищої освіти аграрної галузі. Миколаїв : МНАУ, 2022. 347 с.
4. Наказ МОЗ України від 08.04.2014 № 248. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу.
5. Флэнаган Д. JavaScript. Повне керівництво / Пер. з англ. – 7-е вид. – Sebastopol: O'Reilly Media, 2020. – 706 р.
6. Шишкін М. А., Горбань В. Г. Безпека праці в галузі інформаційних технологій: навч. посіб. – Харків: ХНУРЕ, 2019. – 196 с.
7. ДБН В.2.5-28:2018. Природне і штучне освітлення. – Київ: Мінрегіон України, 2019. – 136 с.
8. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. – Київ: МОЗ України, 1998. – 20 с.
9. ДСН 3.3.6.037-99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку. – Київ: МОЗ України, 1999.
10. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень. – Київ: МОЗ України, 1999.
11. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення. – Київ: ДП «УкрНДНЦ», 2016. – 26 с.

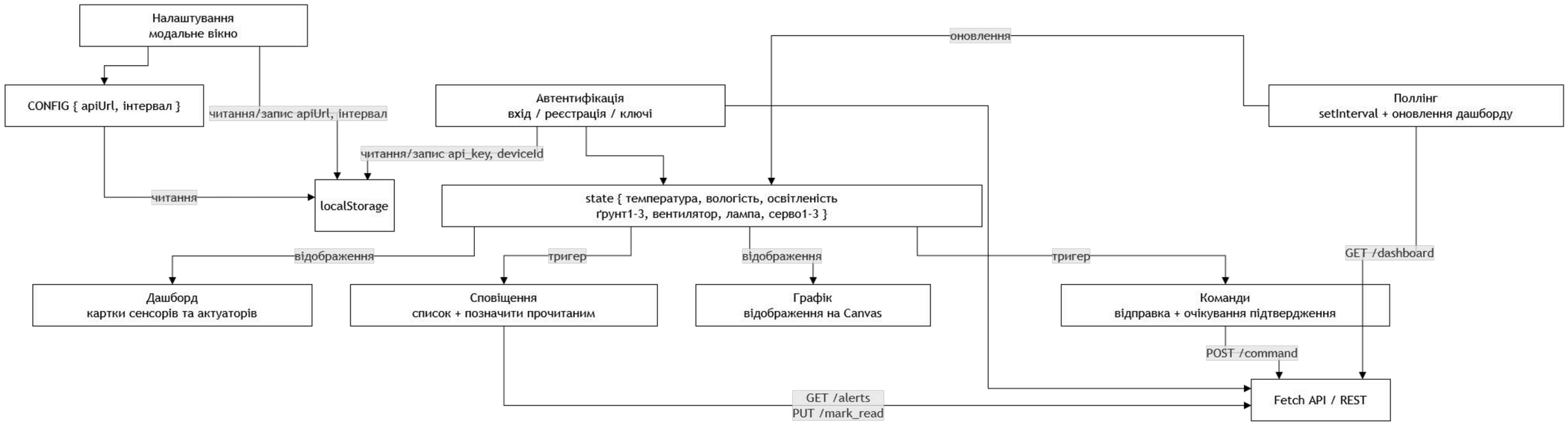
12. НПАОП 0.00-7.15-18. Вимоги безпеки та захисту здоров'я під час роботи з екранними пристроями. – Київ, 2018.
13. Docker Documentation. Overview of Docker Compose. URL: https://docs.docker.com/compose/.
14. Espressif Systems. ESP32 Technical Reference Manual. Version 5.3. – Shanghai, 2024. – 1212 p. URL: https://www.espressif.com/documentation/esp32_technical_reference_manual_en.pdf.
15. Firtman M. Progressive Web Apps on iOS are here. 2018. URL: https://medium.com/@firt/progressive-web-apps-on-ios-are-here-d00430dee3a7.
16. Google Developers. Lighthouse documentation. URL: https://developer.chrome.com/docs/lighthouse.
17. Google Developers. Service Workers: an Introduction. URL: https://developers.google.com/web/fundamentals/primers/service-workers.
18. Google Developers. What makes a good Progressive Web App? URL: https://web.dev/articles/pwa-checklist.
19. Home Assistant Documentation. URL: https://www.home-assistant.io/docs/.
20. IEEE 829-2008. Standard for Software and System Test Documentation. – New York: IEEE, 2008. – 150 p.
21. Mozilla Developer Network. Canvas API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API.
22. Mozilla Developer Network. Progressive Web Apps (PWA) – Introduction. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.
23. Mozilla Developer Network. Using the Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch.
24. Mozilla Developer Network. Web App Manifest. URL: https://developer.mozilla.org/en-US/docs/Web/Manifest.
25. Mozilla Developer Network. Window.localStorage. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage.

26. MySQL 8.0 Reference Manual. URL: https://dev.mysql.com/doc/refman/8.0/en/.
27. ngrok Documentation. URL: https://ngrok.com/docs.
28. Node-RED Documentation. Dashboard. URL: https://nodered.org/docs/.
29. OWASP. REST Security Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html.
30. PHP Manual. PHP Data Objects (PDO). URL: https://www.php.net/manual/en/book.pdo.php.
31. ThingsBoard Documentation. URL: https://thingsboard.io/docs/.

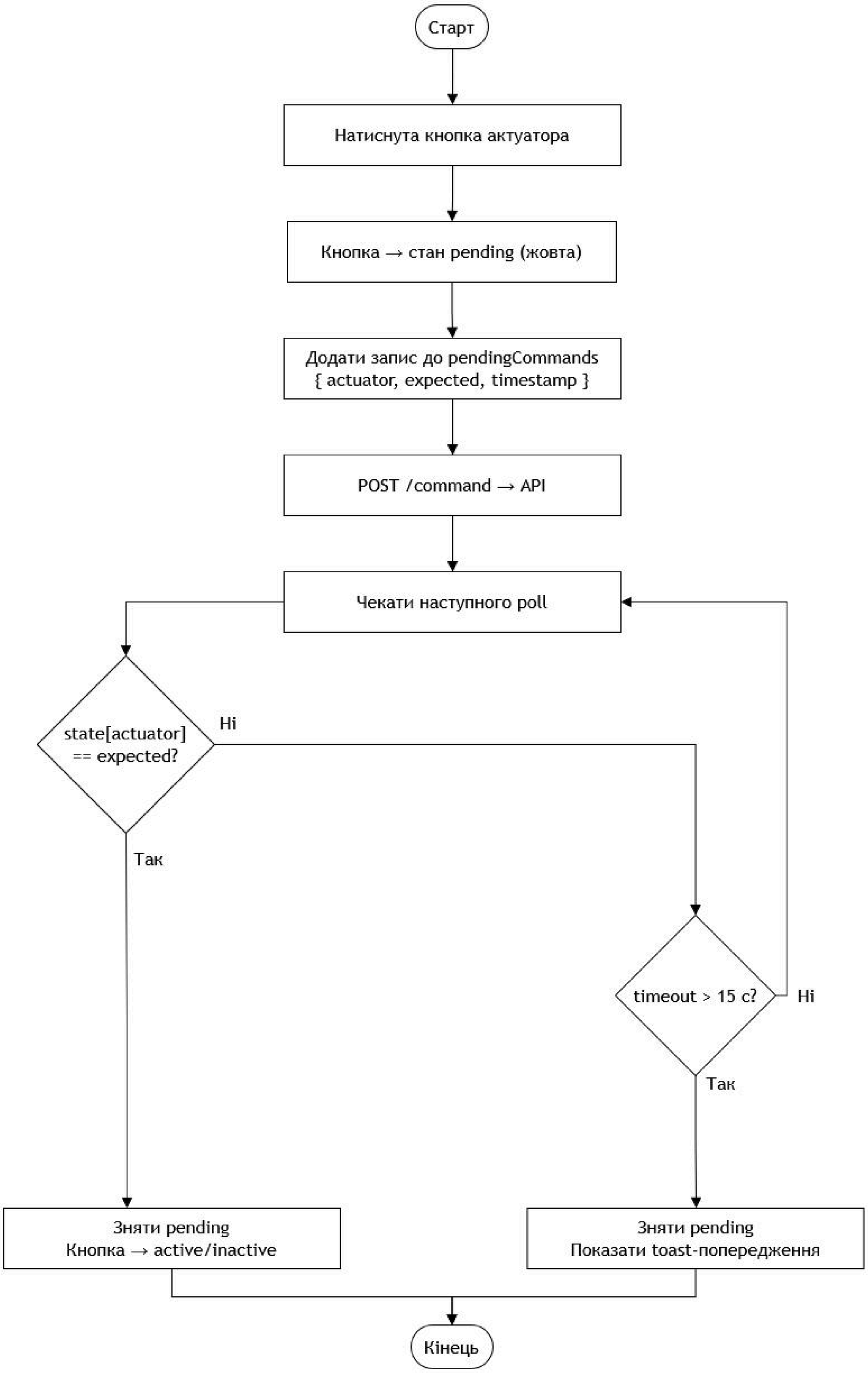
Таблиця техніко-економічних показників

Технічні показники			Економічні показники			
№ п/п	Показник	Значення	№ п/п	Показник	Одиниці вимірю-вання	Значення
1	Мова розробки застосунку	HTML5/CSS3/ Vanilla JS ES6+	1	Собівартість	грн	42 134,99
2	Тип застосунку	PWA (Progressive Web App)	2	Плановий прибуток	грн	8 426,99
3	Контрольовані параметри IoT	темп., вологість повітря/ґрунту, освітленість	3	Ціна (без ПДВ)	грн	50 561,99
4	Оцінка Lighthouse PWA / Performance	100/100 94/100	4	Ціна (з ПДВ 20%)	грн	60 674,39
5	Трудомісткість розробки	168 люд.-год	5	Термін окупності	рік	1,0

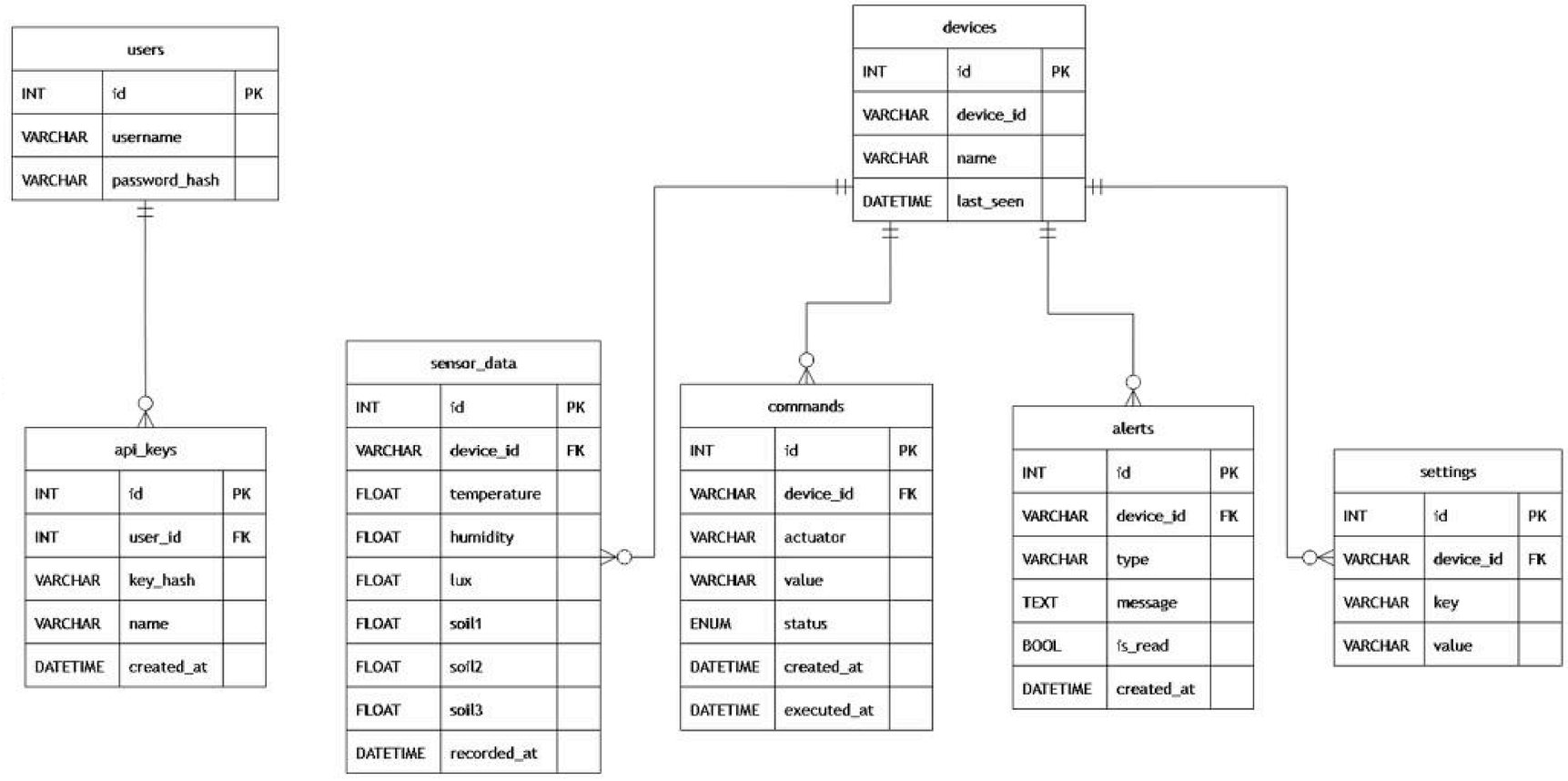
					2026.КВР.123.412.06.00.00 ТБ	
Зм.	Арк.	№ документа	Підпис	Дата	Розробка мобільного інтерфейсу для комунікації з сервером даних IoT пристроїв Таблиця техніко-економічних показників	Літ. / Маса / Масштаб
Розробив		В.ЛІСНИЙ				
Перевір.		Л.ШТОКАЛО				Аркуш / Аркушів
Т. контр.						
Реценз.						ВСП ТФК ТНТУ КІ-412 м. Тернопіль
Н. контр.		А.ЮЗЬКІВ				
Затверд.						



					2026.КВР.123.412.06.00.00 СС	
Зм.	Арк.	№ документа	Підпис	Дата	Розробка мобільного інтерфейсу для комунікації з сервером даних IoT пристроїв Структурна схема	Літ. / Маса / Масштаб
Розробив		В.ЛІСНИЙ				
Перевір.		Л.ШТОКАЛО				Аркуш / Аркушів
Т. контр.						
Реценз.						ВСП ТФК ТНТУ КІ-412 м. Тернопіль
Н. контр.		А.ЮЗЬКІВ				
Затверд.						



					2026.КВР.123.412.06.00.00 БС	
Зм.	Арк.	№ документа	Підпис	Дата	Розробка мобільного інтерфейсу для комунікації з сервером даних IoT пристроїв Блок схема	Літ. / Маса / Масштаб
Розробив		В.ЛІСНИЙ				
Перевір.		Л.ШТОКАЛО				Аркуш / Аркушів
Т. контр.						
Реценз.						ВСП ТФК ТНТУ КІ-412 м. Тернопіль
Н. контр.		А.ЮЗЬКІВ				
Затверд.						



						2026.КВР.123.412.06.00.00 ДБ
Зм.	Арк.	№ документа	Підпис	Дата	Розробка мобільного інтерфейсу для комунікації з сервером даних IoT пристроїв	Літ. / Маса / Масштаб
Розробив		В.ЛІСНИЙ				
Перевір.		Л.ШТОКАЛО			Схема бази даних MySQL: таблиці users	Аркуш / Аркушів
Т. контр.						
Реценз.						ВСП ТФК ТНТУ КІ-412 м. Тернопіль
Н. контр.		А.ЮЗЬКІВ				
Затверд.						